Deep Learning and Isolation Based Security for Intrusion Detection and Prevention in Grid Computing

Dissertation
for attaining the doctoral degree
of Natural Sciences

submitted to the Faculty of Computer Science and
Mathematics
of the Johann Wolfgang Goethe University
in Frankfurt am Main, Germany

by
Andrés Gómez Ramírez
born in Marinilla, Colombia

Frankfurt am Main 2018
(D 30)

accepted by the Faculty of Computer Science and Mathematics of the Johann Wolfgang Goethe University as a dissertation.

Dean:	Prof. Dr. Andreas Bernig
Expert assessors:	Prof. Dr. Udo Kebschull
	Prof. Dott. Ing. Roberto V. Zicari
Date of disputation:	

Abstract

The use of distributed computational resources for the solution of scientific problems, which require highly intensive data processing is a fundamental mechanism for modern scientific collaborations. The Worldwide Large Hadron Collider Computing Grid (WLCG) is one of the most important examples of a distributed infrastructure for scientific projects and is one of the pioneering examples of grid computing. The WLCG is the global grid that analyzes data from the Large Hadron Collider (LHC) at the European Organization for Nuclear Research (CERN), with 170 sites in 40 countries and more than 600,000 processing cores. The grid service providers grant users access to resources that they can utilize on demand for the execution of custom software applications used for the analysis of data. The code that the users can execute is completely flexible, and commonly there are no significant restrictions. This flexibility and the availability of immense computing power increases the security challenges of these environments. Attackers are a concern for grid administrators. These attackers may request the execution of software with a malicious code that gives them the possibility of compromising the underlying institutions' infrastructure. Grid systems need security countermeasures to keep the user code running, without allowing access to critical components but whilst still retaining flexibility. The administrators of grid systems also need to be continuously monitoring the activities that the applications are carrying out. An analysis of these activities is necessary to detect possible security issues, to identify ongoing incidents and to perform autonomous responses. The size and complexity of grid systems make manual security monitoring and response expensive and complicated for human analysts. Legacy intrusion detection and prevention systems (IDPS) such as Snort and OSSEC are traditionally used for security incident monitoring in the grid, cloud, clusters and standalone systems. However, IDPS are limited due to the use of hardcoded fixed rules that need to be updated continuously to cope with different threats.

This thesis introduces an architecture for improving security in grid computing. The architecture integrates the use of security by isolation, behavior monitoring and deep learning (DL) for the classification of real-time traces of the running user payloads also known as grid jobs. The first component of the proposal, the Linux containers (LCs), are used to provide isolation between grid jobs and to gather specific traceable information about the behavior of individual jobs. LCs offer a safe environment for the execution of arbitrary user scripts or binaries, protecting the sensitive components of the grid member organizations. The containers consist of a software sandboxing technique and form a lightweight alternative to other technologies such as virtual machines (VMs) that usually implement a full machine-level emulation and can, therefore, significantly affect the performance. This performance loss is commonly unacceptable in high-throughput computing scenarios. Containers enable the collection of monitoring information from the processes running inside them. The data collected via the LCs monitoring is employed to feed a DL-based IDPS.

DL methods can acquire knowledge from experience, which eliminates the need for operators to formally specify all the knowledge that a system requires.

These methods can improve IDPS by building models that are utilized to detect security incidents automatically, having the ability to generalize to new classes of issues. DL can produce lower false positive rates for intrusion detection, but also provides a measure of false negatives, which can be improved with new training data. Convolutional neural networks (CNNs) are utilized for the distinction between regular and malicious job classes. A set of samples is collected from regular production grid jobs from the grid infrastructure of "A Large Ion Collider Experiment" (ALICE) and malicious Linux binaries from a malware research website. The features extracted from these samples are utilized for the training and validation of the machine learning (ML) models. The utilization of a generative approach to enhance the required training data is also proposed. Recurrent neural networks (RNN) are used as generative models for the simulation of training data that complements and improves the real collected dataset. This data augmentation strategy is useful to supplement the lack of training data in ML processes.

The design characteristics, implementation details and testing environment of a proof-of-concept realization of the researched architecture called Arhuaco are described. Arhuaco combines the isolation and behavior monitoring ideas with deep learning, using a hybrid supervised classification approach with natural language processing for the feature selection and the preprocessing of text-like input data from the traces of the job's system calls and network activity. The proof-of-concept was evaluated in the context of the grid of the ALICE collaboration, a member of the WLCG. Via empirical evaluations, it is described how recently proposed DL methods could outperform traditional ML methods in the task of intrusion detection in grid computing. CNNs applied to the classification of the grid job behavior are compared to support vector machines (SVMs). SVMs are one of the most popular algorithms in IDPS research. A long short-term memory (LSTM) has been tested to validate the idea that RNNs are helpful to improve and increase the training dataset coverage for intrusion detection in grid computing. An average runtime increase of 6.11% was observed when testing a set of regular ALICE grid jobs that was run with Arhuaco using LC isolation, behavior monitoring and classification with DL. An accuracy of 99.52% was obtained when validating CNNs in the classification of previously unseen system call traces as usual or malicious. For the validation of network traces, an accuracy value of 98.75% was achieved. The SVM was trained with simulated data to evaluate the LSTM method. Once the model was built, the SVM was applied to the classification of novel unseen network data traces from the original dataset. There was a 0.72% improvement in the accuracy.

The results demonstrate that LCs utilized for isolation produce a moderate performance impact that can be reduced with several configuration options. CNNs applied to the classification of behavior trace data could distinguish between normal and malicious jobs with close to 100% accuracy. The generative method via LSTM improved and increased a training dataset for intrusion detection in grid computing. The proposed approaches solve the problems of analyzing the activity of grid jobs, identifying malicious activity and keeping traceability of the user-generated events. Therefore, better and stronger evidence to detect attacks and to find their source can be collected.

Zusammenfassung / German Summary

Das Grid Computing beschreibt ein Paradigma, welches vor allem im wissenschaftlichen Umfeld leistungsstarke Werkzeuge hervorgebracht hat, so zum Beispiel in der Hochenergiephysik, Genomforschung oder im Pharmaziebereich. Viele tausend Institute mit Computereinrichtungen weltweit schließen sich hierfür in Kollaborationen ambitionierter Wissenschaftsprojekte zusammen, die alle eine gemeinsame Anforderung verbindet: Sie erfordern massive Rechenleistung. Teilnehmer des Grid Computings unterstützen sich dabei gegenseitig durch die Bereitstellung verteilter Rechenkapazitäten und der dazugehörigen Infrastruktur an interessierte Forscher. Hierbei wird die Erlaubnis erteilt, Programmcode auf den Computern der teilnehmenden Institute auszuführen. Die sich daraus ergebenden Sicherheitsrisiken stellen eine große Herausforderung dar. So wäre es Angreifern zum Beispiel möglich, durch die missbräuchliche Ausführung von Schadcode die zugrundeliegende wissenschaftliche Infrastruktur eines Standortes zu kompromittieren. Die dabei erbeuteten Ressourcen können wiederum dazu verwendet werden, Schadcode zu verbreiten oder Angriffe gegen andere Organisationen zu führen. Grid Systeme erfordern daher Sicherheitsvorkehrungen um laufende Anwendungen voneinander abzuschirmen und den Zugriff auf kritische Komponenten einzuschränken. Die Administratoren solcher Systeme benötigen zudem eine Möglichkeit zur Beobachtung der Aktivitäten aller laufenden Anwendungen. Die Analyse dieses Verhaltens ist notwendig, um mögliche Sicherheitslücken zu identifizieren, laufende Vorfälle automatisch zu erkennen und Reaktionen darauf anzustoßen. Die Größe und Komplexität von Grid Systemen gestaltet eine manuelle Überwachung und Reaktion auf Sicherheitsvorfälle durch Analytiker jedoch kompliziert und kostspielig.

Abschirmung und Verhaltensüberwachung in Kombination mit „Intrusion Detection and Prevention Systems“ (IDPS) zählen daher zu den grundlegenden Anforderungen um die Sicherheit in einem Grid gewährleisten zu können. Standardmäßig eingesetzte IDPS basieren auf einer Menge fest codierter Regeln, die zur Identifikation von Ereignissen führen [1]. Sie kommen üblicherweise in Grids für die Erkennung bekannter Signaturen vergangener Ereignisse zum Einsatz, können jedoch keine Aussagen über das Auftreten neuer Bedrohungen treffen. Angreifer können jedoch bekannte Methoden modifizieren, Binärcode von Schadsoftware umschreiben oder eigene Schadprogramme entwickeln um die Erkennungsregeln zu umgehen. Fortgeschrittene, durch künstliche Intelligenz (KI) unterstützte, autonom agierende Methoden bieten die Möglichkeit, die Fähigkeiten von IDPS zu verbessern, um auch auf neue Bedrohungen im Grid reagieren zu können. Ihre Verwendung in Grid Umgebungen ist bedingt durch deren dynamische Natur, Größe, Komplexität und durch deren Möglichkeiten zur Übermittlung und Ausführung von beliebigem Programmcode höchst relevant. Das dahingehend entwickelte Konzept, genannt Arhuaco, wurde als Machbarkeitsstudie realisiert. Seine Designmerkmale, Implementierungsdetails sowie eine Testumgebung werden im Folgenden vorgestellt. Arhuaco kombiniert dabei die Abgrenzungs- und Überwachungsgedanken mit „Deep Learning“ (DL). Hierfür nutzt es einen hybriden, überwachten Klassifizierungsansatz mit natürlicher Sprachverarbeitung

für die Merkmalsauswahl und die Vorverarbeitung von text-ähnlichen Eingabedaten aus der zu einem Job gehörigen Verhaltensüberwachung einschließlich Systemaufrufen und Netzwerkaktivitäten.

Linux Container zur sicheren Abschirmung

Die von Nutzern in der Grid Umgebung ausgeführten Anwendungen sollten innerhalb eines sicheren Sandkasten-Modus laufen, in dem sowohl lokale Grid Ressourcen wie auch sensitive Netzwerkbereiche ausreichend geschützt sind. Hierfür wurden in der Vergangenheit vielfach „Virtuelle Maschinen“ (VM) vorgeschlagen. VM beeinflussen jedoch die System-Performance in einer Weise, die in vielen Fällen den Einsatz im Bereich des „High Performance Computing“ (HPC) verbietet. „Linux Container“ (LC) haben sich daraufhin als die bestmögliche Alternative zur sicheren Ausführung von Programmcode in Grid Systemen herausgebildet. Durch ihre hervorragende Balance zwischen Sicherheit und Performance [2] wurden sie auch in dieser Arbeit für die Realisierung von Sicherheit durch Abschirmung ausgewählt. Darüber hinaus stellen LC eine Abschirmungstechnologie dar, welche sich in geeigneter Weise auch auf Linux-basiertes Grid Computing in der Hochenergiephysik (HEP) anpassen lässt.

Neben der Programmausführung im Sandkasten-Modus ermöglichen LC auch eine Netzwerk-Isolation. Hierfür kann ein verschlüsseltes, virtuelles Netzwerk innerhalb eines anderen physikalischen oder virtuellen Netzwerkes generiert werden, um die darin laufenden Prozesse vom Zugriff auf sensitive Bereiche abzuhalten. Diese Funktionalität wird im Grid Computing benötigt, da einige Standorte ihre Ressourcen mit anderen Projekten oder mit experimenteller Infrastruktur teilen. Abbildung 1 zeigt schematisch die Eigenschaften bei Sicherheitsabschirmung. Im linken Teil laufen Grid Jobs ohne Abschirmung, was es ihnen ermöglicht, andere Jobs oder das zugrundeliegende System und Netzwerk zu beeinflussen. Der rechte Teil zeigt Grid Jobs, die durch LC voneinander abgegrenzt wurden und bei denen jeder in einer reduzierten Version des Gesamtsystems läuft und damit keinen Zugriff auf andere Jobs oder sensitive Ressourcen der Verarbeitungsknoten hat. Die Kommunikation verläuft über ein virtuelles Netzwerk, ausgegrenzt von anderen beschränkten Netzwerkbereichen.

Konventionelle Grid Systeme nutzen „Batch Engines“ wie zum Beispiel Condor [3] um Jobs innerhalb eines Computing Clusters zu planen. Die Verwendung von Containern anstelle standardmäßiger Batch Jobs erfordert jedoch moderne Orchestrierungs-Werkzeuge, die eine Container Ausführung auch über ein verteiltes Cluster hinweg planen. „Docker Swarm“ ist die für diese Arbeit übernommene Engine zum Betrieb verteilter Container.

Abschirmung genügt jedoch noch nicht um die Sicherheit eines Grids zu schützen. Ein Job könnte weiterhin Angriffe gegen die Infrastruktur des Institutes oder gegen Dritte führen. Jobs könnten die Computerressourcen auch für unerlaubte Aktivitäten wie zum Beispiel die Verbreitung von Schadsoftware, die Teilnahme an sogenannten „Bot“ Netzwerken oder das Schürfen von Krypto-Währung missbrauchen. Eine Lösung für dieses Problem stellt die Verhaltensbeobachtung aller in einer Grid Infrastruktur laufenden Grid Jobs dar. Im Kontext dieser Arbeit war

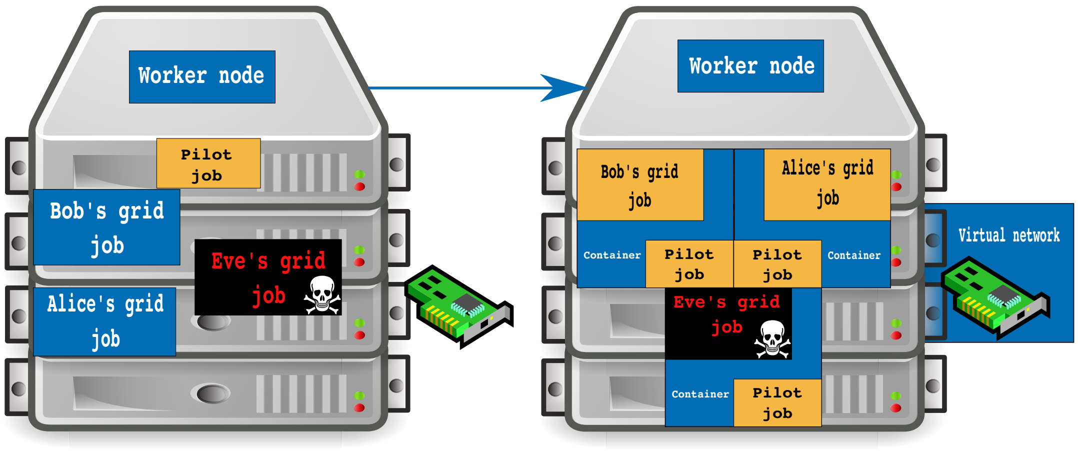


Abbildung 1: In konventionellen, unsicheren Grid Umgebungen (links) können Grid Jobs von Eve auf die Jobs von Bob und Alice zugreifen, um sie für Infrastruktur-Angriffe zu missbrauchen, ohne dabei eine Spur zu hinterlassen. Bösartige Grid Jobs können die physikalischen Systeme und Netzwerke direkt kompromittieren. Abschirmung mittels Linux Containern (rechts) wird eingeführt; dadurch sind Grid Jobs beschränkt auf eine Sandkasten-Umgebung in der ihr Verhalten nachverfolgt and analysiert werden kann um Einbruchsversuche automatisch zu erkennen.

daher die angebotene LC-Funktionalität in Bezug auf Überwachungsmöglichkeiten nützlich. Hierbei können unter anderem der Ressourcenverbrauch von CPU, Speicher und Festplatte, der Netzwerkverkehr oder Systemaufrufe eines spezifischen Containers aufgezeichnet werden. Dies impliziert, dass die Gewinnung von Daten für jeden Job umsetzbar ist. The Quelle eines Sicherheitsvorfalls lässt sich so mit höherer Genauigkeit ermitteln oder sogar das Sammeln forensischer Daten zur weiteren Analyse.

Jede Sandkasten-Technik kostet Leistung. Ein wichtiges Ziel ist daher die optimale Balance zwischen Leistung und Sicherheit zu finden. Um Aussagen über Leistungsveränderungen treffen zu können und um den durch LC bedingten Overhead zu messen wurden daher zwei Messmetriken definiert. Die erste Metrik basiert auf dem in [2] beschriebenen Ausführungs-Durchsatz mittels der Linpack [4] Benchmark Bibliothek. Linpack ist eine Bibliothek, die ein dichtes System linearer Gleichungen mit einem auf LU-Zerlegung [5] mit Teilpivotisierung basierenden Algorithmus verwendet, um die Verarbeitungsfähigkeiten von Rechnersystemen mit hoher Leistung zu ermitteln. Der gemessene Durchsatz ist dabei generell definiert als die maximale Rate bei der ein System Berechnungen durchführt. Die zweite Metrik berücksichtigt die Laufzeit, die ein typischer Grid Job des „A Large Ion Collider Experiment„ (ALICE) für seine vollständige Abarbeitung benötigt. Hierbei wurde die in den ALICE Bibliotheken bereits verfügbare Benchmark Anwendung *PbPbbench* [6] gewählt, um den durch Abschirmungs- und Überwachungsebenen bedingten Laufzeit-Overhead zu bestimmen. Weitere

Härtungsmaßnahmen, die Sicherheit durch Abschirmung noch zusätzlich verbessern, ohne dabei maßgeblichen Einfluss auf die Leistung zu haben, wurden untersucht.

Deep Learning für die Klassifikation von Grid Jobs

IDPS sind üblicherweise Werkzeuge um das Verhalten laufender Grid Jobs zu analysieren und darin Angriffsmuster zu finden [1]. Verbreitete kommerzielle IDPS wie zum Beispiel Bro [7], Snort [8] und OSSEC [9] verwenden statische Regeln und suchen damit nach bekannten Angriffssignaturen um mögliche Angriffe zu identifizieren. Sie stoßen jedoch an ihre Grenzen sobald unbekannte oder leicht abweichende Angriffsmethoden Verwendung finden, daher müssen sie regelmäßig aktualisiert werden [10]. Dies ist für den Einsatz in hoch-dynamischen Grid Umgebungen jedoch ungeeignet. Jobs und Container in Grid Umgebungen starten und enden kontinuierlich, daher ist die Verwendung statischer Regeln zur Erkennung von Vorfällen durch diese Jobs ineffektiv.

Im Anwendungsfeld der „Intrusion Detection“ wurde „Machine Learning“ (ML) zur Modellierung und Auswertung von Protokollen und Netzwerkdaten mit anschließender automatisierter Klassifikation von Sicherheitsvorfällen allgemein empfohlen [11]. ML-Methoden erlauben IDPS Systemen sich an wechselnde Einsatzszenarien ohne statische Anwendungen anzupassen. Abbildung 2 zeigt, wie diese Forschungsarbeit die Auswertung von Grid Job Überwachungsdaten für die „Intrusion Detection“ unter Echtzeitbedingungen anwendet. Dies bedeutet, dass eine Auswertung von Betriebssystemprozessen (Linux) ausgeführt wird. Diese Arbeit stellt die Verwendung einer Kombination aus Techniken des DL, „Convolutional Neural Networks“ (CNN) [12] sowie word2vec [13] im „Intrusion Detection“ Anwendungsfeld des Grid Computings vor.

Systemaufrufe und Netzwerkverbindungsverläufe dienen als Eingabedaten. Diese Daten sind in einem menschenlesbaren Format kodiert und müssen daher zunächst mittels „Natural Language Processing“ (NLP) Methoden in ein geeignetes Sprachmodell umgewandelt werden. Kürzlich vorgestellte DL Methoden für NLP repräsentieren gelernte Wortvektor-Darstellungen mittels neuronaler Sprachmodelle [14]. Dabei werden Wörter unter Verwendung neuronaler Netze mit verschiedenen Zwischenschichten in einen Vektorraum geringerer Dimensionalität projiziert [15]. Semantisch nahe beieinander liegende Wörter in den Trainingsdaten sind auch in Vektorräumen mit geringer Dimensionalität mathematisch nahe beieinander liegend. Der word2vec Algorithmus [13] wurde für Arhuaco gewählt, um die Eingabemerkmale zu generieren. Es handelt sich dabei um ein Vorhersagemodell für das Lernen von Worteinschlüssen. Word2vec Vektoren entwickeln dabei geeignete Eingaben für CNNs, da sie es erlauben, Eingabedaten als Matrizen zu behandeln, ähnlich eines Feldes von Pixeln in einem Bild.

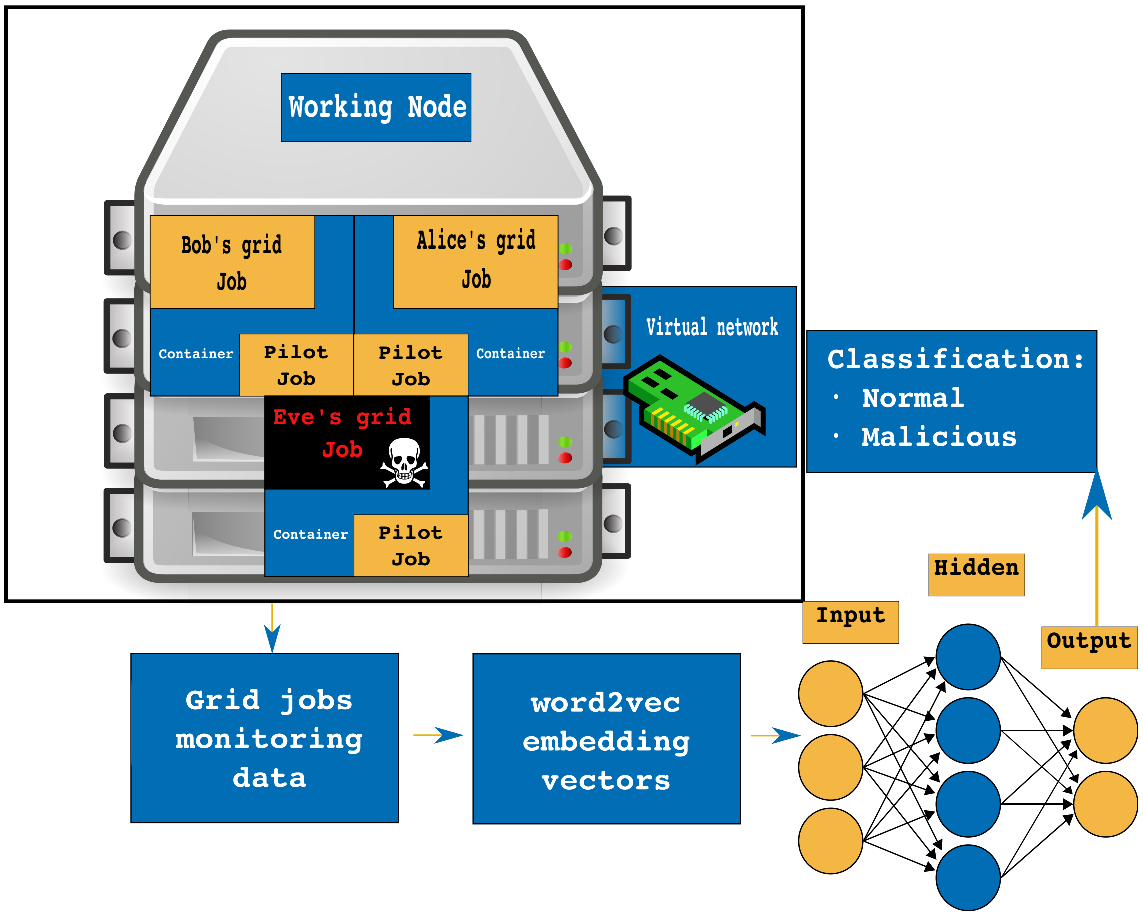


Abbildung 2: Arhuaco extrahiert Daten aus der Verhaltensüberwachung von Grid Jobs. Diese Daten werden anschließend vorverarbeitet und mittels eines DL-Algorithmus klassifiziert. Abschließend werden die Grid Job Spuren als regulär oder bösartig gekennzeichnet.

Trainingsdatensatz und generatives Verfahren

ML-basierte IDPS benötigen einen Trainingsdatensatz. Es konnte jedoch kein verfügbarer Datensatz für die Klassifikation von Grid Jobs gefunden werden, daher wurden eigene Daten gesammelt. Für die Generierung von Standard-Daten kamen dabei Jobs aus ALICE Produktiv-Grids zum Einsatz, während Schad-Informationen unter Verwendung von Linux Schadsoftware-Mustern entstanden. Die Ergebnisse in Abschnitt 6.4.3 zeigen, wie es dem CNN möglich war, gänzlich neue Datenmuster mit nahezu optimalen Messmetrikwerten korrekt zu klassifizieren. Diese Tatsache bewies, dass der gesammelte Datenbestand, obwohl nicht gänzlich vollständig, gut genug dafür geeignet ist, eine Trainings-Quelle für das Problem innerhalb dieses Kontextes darzustellen - „Intrusion Detection“ in Grid Computing. Diese Daten können neben der Hochenergiephysik auch bei anderen Arten von Grids Anwendung finden. Die bösartigen Netzwerkdaten hatten den geringsten Stichprobenanteil der Sammlung. Es kam daher eine generative Modellierungsmethode zu Einsatz, um diese Daten zunächst zu vervollständigen.

Um den Trainingsdatensatz der Netzwerkverläufe zu vervollständigen wurde für Arhuaco ein generatives Verfahren mit Sprachenmodell auf Zeichenebene gewählt. Das Ziel des genutzten Modells ist es, das nachfolgende Zeichen innerhalb einer gegebenen Sequenz vorherzusagen. Hierfür wurde ein LSTM gewählt, das die Wahrscheinlichkeitsverteilung über Sequenzen lernt. Mit der daraus resultierenden bedingten Verteilung ist es möglich, Daten in einer Weise zu prozessieren, bei der das jeweils nachfolgend erhaltene Zeichen einer generierten Zeichenkette wiederum als Eingabedatum an die LSTM übergeben werden kann [16]. Nach Abschluss dieses Trainingsprozesses können neue Daten erzeugt werden. Diese generierten Daten wurde als zusätzliche Trainingsdaten genutzt, um die Generalisierungsfähigkeiten eines „Support Vector Machines„ (SVM) Klassifikators zu erweitern.

Auswertung

Diese Studie wurde im Rahmen des Grid Systems der ALICE Kooperation als Mitglied der „Worldwide Large Hadron Collider Computing Grid“ (WLCG) durchgeführt. Das WLCG als globales Grid analysiert Daten des „Large Hadron Collider“ (LHC) Experiments des „Conseil Européen pour la Recherche Nucléaire„(CERN). Es setzt sich aus 170 Standorten in 40 Ländern zusammen und bietet dabei die Rechenkapazität von mehr als 600.000 Prozessorkernen. Im Rahmen eines Tests mit einer Menge an ALICE Grid Jobs unter Verwendung von LC Abschirmung, Verhaltensüberwachung und ML-Klassifikation konnte ein durchschnittlicher Laufzeitoverhead von 6,11% beobachtet werden. Bei Durchsatz Tests mit dem Linpack Benchmark zeigte sich eine Leistungsreduktion von 0,4% im Falle der Nutzung von LC im Vergleich zu nativen Linux-Aufrufen. Die Überwachung mit Arhuaco fügte 0,5% Mehraufwand hinzu. Verschiedene Studien [2] im HPC vergleichen die Verwendung von LC und VM bezüglich der Nutzung von Anwendungen im Sandkastenverfahren. Diese Studien zeigen den signifikanteren Leistungs-Overhead bei der Nutzung von VM im Vergleich zu LC.

Ein weiterer Vergleich wurde zwischen den vorgeschlagenen CNN und der SVM-Klassifikationsmethode aufgestellt, einem der bekanntesten Algorithmen auf dem Gebiet der IDPS [17]. Das bewährte bag-of-words Modell [18] wurde dabei für die Generierung der Merkmalsvektoren für die SVM Eingabe verwendet. Bei der Validierung von CNNs hinsichtlich der Klassifikation zuvor unbekannter Verläufe an Systemaufrufen in die Gruppen regulär oder bösartig zeigte eine Genauigkeit von 99,52%. Bei der Validierung von Netzwerkverläufen wurde eine Genauigkeit von 98,75% erreicht. Die Genauigkeit der durch Training mit generierten Daten erhaltenen SVM nach Anwendung auf zuvor unbekannte Netzwerkdaten-Verläufe aus dem ursprünglichen Datensatz ergab eine Verbesserung um 0,72%.

Diskussion

Die als Konzeptnachweis mittels Arhuaco implementierten Verfahren zeigten, dass sich die Sicherheitsanforderungen im Grid Computing erfolgreich umsetzen lassen. Die Methoden der Sicherheit durch Abschirmung ermöglichen eine Sandkastenumgebung, in der sich Nutzerjobs ohne Zugriff auf sensitive Ressourcen wie Serverkonfigurationsdaten oder verbotene Netzwerkbereiche ausführen lassen. Die Abschirmung erlaubte weiterhin das Sammeln von Überwachungsinformationen bezüglich des Verhaltens einzelner Jobs.

Die mit word2vec vorverarbeiteten Eingaben generierten CNNs stellten eine bessere Alternative im Vergleich zu den SVM mit bag-of-words Eigenschaften dar. Die im Rahmen von Arhuacos Konzeptnachweis gewählten CNNs erhöhten die Genauigkeit und reduzierten die Falschpositiv-Rate bei der Klassifikation von Grid Jobs. Dem CNN war es dabei möglich in der Validierungsphase völlig unbekannte Stichproben zu generalisieren. Dadurch verbesserte es die Erkennung bösartiger Aktivitäten die aus den User Jobs eines Grids hervorgehen. Weitere Ergebnisse bieten Nachweise darüber, dass der word2vec Algorithmus die auf semantischer Kontexterhaltung basierenden Eingabemerkmale in einer genaueren Art und Weise auszudrücken vermochte. Dieser zur Analyse von Eingaben in Textform dienende Ansatz der natürlichen Sprachverarbeitung lässt sich dabei komfortabel von Systemaufrufen und Netzwerkverlaufsdaten auf Informationen anderer „Intrusion Detection“ Systeme, Überwachungsdaten anderer Quellen oder auf andere Systemprotokolle ausdehnen.

Diese Studie zeigte, dass die Nutzung eines generativen Modells zur Verbesserung der Abdeckung des Trainingsdatensatzes in Grid „Intrusion Detection“ Systemen eine erhöhte Genauigkeit sowie eine reduzierte Falschpositiv-Rate bewirkt. Diese Auswertungen ergaben im Vergleich zu den Anfangsergebnissen eine Erhöhung der Messmetriken, welche sich in den Trainings- und Validierungsphasen ergaben. Weiterhin werden die praktischen Vorteile der Nutzung von LSTM für die Modellierung und Datengenerierung im Umfeld von „Intrusion Detection“ Systemen gezeigt, im vorliegenden Fall für Grid Computing Jobs.

Acknowledgements

I would like to thank all who contributed to this thesis.

First of all to my supervisor, Prof. Dr. Udo Kebschull for the opportunity to work in his research group, the meaningful discussions we have had, and his support that allowed me to focus on my research. Many thanks to Prof. Dott. Ing. Roberto V. Zicari for his willingness to review my thesis. I want to give special thanks to Dr. Camilo Lara who supported me to access the PhD program, advised me during the doctoral path, and reviewed this thesis and the related papers. I would also like to thank HGS-Hire for the provided scholarship. Without their support, this work would not have been possible.

I want to thank my colleagues and friends Cruz Garcia, Antonio Lucio, Frederik Grüll, Heiko Engel, Andrei Oancea, Thomas Janson, Alexander Adler, Jano Gebelein and Stefan Böttger for their help in many aspects related to my research, as well as for proof-reading the thesis. Daniel Bilanovic made the additional test with isolation alternatives. I appreciated his contribution very much. Special thanks to Jano Gebelein and Heiko Engel who reviewed my dissertation, helped me with the German abstracts and in general supported me in all aspects of my doctoral program.

My thanks to the teams of the ALICE HLT, ALICE offline and the CERN security department, for their advise and support with the finalization of this research projects, and the review of the papers. Special thanks to Miguel Martinez Pedreira, Costin Grigoras, Latchezar Betev, Romain Wartel, and Stefan Lueders.

I would like to thank all my friends who were always supporting me. Gracias a mi familia, especialmente a Martha, Gerardo, Francis y Jeisson, por el cariño y soporte dado. No hubiera logrado nada en mi vida sin ellos. Muchas gracias a Melissa, que ha sido mi soporte y alegría, en los momentos felices y en los difíciles.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

1.1 Motivation

Grid computing has emerged as a powerful tool for scientific projects. It has been utilized in areas such as high energy physics (HEP), genome research and pharmaceutical production. Institutions all around the world collaborate on ambitious scientific or commercial projects that require massive computing power. The grid concept has similarities with the cloud computing infrastructure. In the grid and the cloud, users are granted access to resources for the execution of arbitrary software applications. The flexibility that this generates, combined with the high availability of processing power, creates security risks such as the submission of malicious code to compromise the underlying computational infrastructure or even the scientific instruments. Adversaries interested in taking advantage of such resources become a concern. Some examples of security issues are users mining cryptocurrencies, attackers hosting malware or even sensitive scientific results being stolen.

Grid systems need security methods to protect critical internal components that could be reachable by malicious adversaries. The continuous monitoring of the activities that custom applications are carrying out is a requirement for administrators in order to detect security incidents generated from those applications. For instance, malicious users could attempt to escalate their authorized privileges or interact with forbidden systems. The complexity of grid systems renders a manual security monitoring, and response incrementally expensive as the systems grow in size. Therefore, performing autonomous detection, as well as reactions without the need for administrator intervention, is the desired solution.

Isolation of grid jobs in combination with security monitoring powered by intrusion detection and prevention systems (IDPS) increases the level of grid security. Traditionally utilized IDPS are based on sets of hard-coded rules to detect incidents. They are capable of detecting known signatures of past events and make it possible to reduce the chances of false positives by tuning the rules. A drawback of these systems is that they cannot identify new threats, even if such threats are similar to past events. Adversaries can modify existing attack methods, rewrite malware binaries or create custom exploits to circumvent detection techniques. Advanced autonomous methods supported by artificial intelligence (AI) are an exciting solution to improve the IDPS capabilities to cope with new incidents in the grid. AI-based IDPS are relevant in environments such as the grid and the cloud, given the highly dynamic nature, the size and complexity of such environments.

Deep learning (DL), a way of implementing machine learning (ML), has shown outstanding success in many areas, such as autonomous driving, computer vision, financial forecasting and speech recognition among others. DL provides advanced methods to preprocess and model input data automatically. Heterogeneous sources of data can be collected from the grid and the running jobs, that providing information about the current safety status. This data can be utilized for the training of DL algorithms that learn the desired state of the grid systems, thus, DL enables the creation of autonomous systems that can monitor and react to potential attacks in such systems.

1.2 Overview

This research project focuses on solving security issues commonly found in the grid job execution environments built around site worker nodes (WNs), i.e., the machines in the institutes' computing clusters where the jobs are executed. A brief description of the security issues are listed as follows:

- For common organizations, one of the main security tasks is to keep the computing resources protected from external attackers. If an attacker can execute code inside the organizations' computers, it is already considered as a security breach. In the grid, the users can run arbitrary code inside the grid collaborations' computational infrastructure by design. A malicious application could take advantage of this design to damage or misuse sensitive components such as the servers, restricted organizations' networks or experimental data, hence, isolation between the physical infrastructure and the user jobs is a critical requirement.

- Different users' jobs running in the same system without proper isolation from each other is another issue in grid computing. A malicious job could tamper and inject code inside other users' jobs. This situation makes the traceability of incidents difficult since an attacker could blame other participants for its actions without being noticed by the system administrators.

- An enforced and isolated execution environment is essential to protect the grid, but it is not enough to completely avoid security incidents. Attackers may still take advantage of the grid resources to perform, for instance, denial of service (DoS) attacks, cryptocurrency mining [19] or using the WNs for compromising third-party organizations.

- IDPS can be used to monitor the behavior of grid jobs. However, the users' ability to run arbitrary code makes intrusion detection a difficult task since traditional, rule-based IDPS have been designed for static environments. Due to the static nature of their set of rules for the detection task, rule-based

IDPS cannot be adapted to a dynamic environment and learn to detect novel intrusions in the grid.

The ALICE grid is the production environment utilized to evaluate the proposed solutions. The Worldwide Large Hadron Collider Computing Grid (WLCG) is the global grid, including the ALICE grid, that analyzes data from the Large Hadron Collider (LHC) at the European Organization for Nuclear Research (CERN). The WLCG is exposed to internal and external attackers, just as are other grids. The computing power, the organization's reputation and the accessibility from anywhere on the Internet are reasons that may motivate an adversary to try to breach one of these systems. An illustrative example of how an adversary may attempt to vulnerate the WLCG would be as follows: the attacker searches for public information about potential users of the grid. These users, frequently physicists, may not have proper training in cybersecurity topics. By sending phishing emails, the attacker could get access to their authentication certificates. The physics experiment information is typically encoded in ROOT data files [6]. Since it is common for grid users to interact with these files, a backdoor may be planted inside them, which grants the attacker access to the physicists' computers. Once there, if the grid certificates are stolen, an adversary could submit malicious jobs to the grid, that could execute or inject code into other users' jobs while leaving no proof in a system log. Since authenticated users can freely run arbitrary code or transfer any data on the grid infrastructure, attackers would have the same access availability. A more formal and detailed description of the vulnerabilities and possible attacks in the grid is given in section 4.1.

1.2.1 Aim of the Thesis

An architecture that increases the security enforcement in grid computing is researched in this thesis. The architecture utilizes an integrated approach that combines security by isolation, behavior monitoring and DL for the detection and prevention of intrusions that come from the grid jobs. These methods can solve the problems of running untrusted software inside the grid infrastructure, analyzing the activity of grid jobs, identifying malicious activity and keeping traceability of the user-generated events.

Several sandboxing alternatives, including Linux containers (LCs), virtual machines and kernel hardening, are explored as a way to provide security by isolation in the grid. Isolation enforcement in the execution environment, in such a way that jobs cannot access sensitive server or network resources, is a requirement for the grid. This isolation solves the security grid problem of having a trusted environment where users have the freedom to run any software while the participating institutes' sensitive systems remain out of reach.

Deep learning methods for the detection and analysis of intrusions and training data augmentation are studied. Running grid jobs in sandboxed environments enables the collection of behavior monitoring data to be forwarded for analysis with DL methods in near real-time. The objective is to detect and prevent intrusions coming from malicious grid jobs. In addition, a dataset for IDPS training and data model validation in grid computing is proposed. The dataset made of inputs extracted from production jobs and Linux malware samples is utilized to train and optimize the classification algorithms. Generative methods with neural networks are explored to improve the dataset and enhance the training coverage and effectiveness. An evaluation with a prototype implementation of the ideas is carried out in the ALICE collaboration grid, a member of the WLCG.

Monitoring and protecting the grid infrastructure from unauthorized and malicious code is a focus of the solution. By collecting and processing data generated by jobs, such as system calls and network connections, the required input for DL algorithms that enable the automation of data analysis to find security-related incidents is available. Legacy intrusion detection and prevention systems provide attack detection with fixed rules based on signatures that have to be continuously updated by human operators. DL methods grant the ability to detect generalized attack variants.

1.2.2 Defended Statements

This thesis describes research involving security isolation methods for grid jobs sandboxing and deep learning to support the detection of intrusions generated from jobs. The objective is to defend the following statements and provide clear evidence to demonstrate the effectiveness of the proposed architecture.

1. **Effective isolation and monitoring of grid jobs in Linux Containers:** LCs are used to provide isolation for grid jobs and to obtain traceability information about individual job activities. System call and network trace monitoring data are collected. LCs contribute to improving the security of the grid by separating user-controlled components from sensitive system resources and keeping traceability of the activities of the jobs in case of intrusion attempts. As a result, better evidence to find the source of an attack can be collected.

2. **Accurate classification of grid jobs for intrusion detection, based on deep learning:** near real-time classification of the grid job's traces by using deep neural networks (DNN) is proposed. The data collected via the LCs is used as the source to extract text-based vector embeddings as input features. Convolutional neural networks can be applied as a successful method for the classification of grid jobs into malicious and normal.

3. **Improved classification results via generative models:** a recurrent neural network (RNN) is proposed as a generative model for enhancing the training dataset coverage in grid IDPS. A data generation process can increase the resulting accuracy of the chosen classification algorithm.

4. **Collection of a relevant benchmark dataset for model validation:** a benchmark dataset for the validation of an intrusion model and malware classification is necessary for grid computing. A custom dataset enables the validation of the training and testing steps of machine learning models to automate security monitoring in the grid.

1.3 Outline

Chapter 2 introduces the essential background information on security by isolation, security monitoring, classification using deep learning and generative models for data augmentation. Chapter 3 presents the state of the art for the relevant topics. The previous approaches towards ML for security and isolation-based methods applied to distributed environments such as grid computing are explored.

The design principles and choices for the isolation, security monitoring, and deep learning methods are described in full detail in the "The Design of Arhuaco" section in chapter 4. The arguments for choosing such techniques and how they contribute to improving the security of the grid are stated. The implemented technologies and the chosen algorithms are also explained in this chapter. Chapter 5 shows how the ideas are implemented in Arhuaco, the proof-of-concept implementation.

The evaluation metrics and tests carried out for the approaches are shown in chapter 6. A list of the collected evidence that demonstrates the benefits of using the proposed methods over other compared alternatives is described in detail. Finally, chapter 7 makes a series of conclusions about this study and summarizes the findings, including a judgment on the defended statements. This chapter also indicates the possible future paths of research that may be pursued.

1.4 Publications

1. **A. Gomez Ramirez**, C. Lara, U. Kebschull for the ALICE Collaboration. *"Intrusion Prevention and Detection in Grid Computing - The ALICE Case"*. Journal of Physics: Conference Series, 664(6):062017, 2015. [20]

2. **A. Gomez Ramirez**, M. Martinez Pedreira, C. Grigoras, L. Betev, C. Lara and U. Kebschull for the ALICE Collaboration. *"A Security Monitoring Framework For Virtualization-Based HEP Infrastructures"*. Journal of Physics: Conference Series, 898(10):102004, 2017. [21]

3. **A. Gomez Ramirez**, C. Lara, L. Betev, D. Bilanovic, U. Kebschull for the ALICE Collaboration. *"Arhuaco: Deep Learning and Isolation Based Security for Distributed High-Throughput Computing"*. Manuscript submitted to the Journal of Grid Computing (2018). [22]

4. H. Engel, T. Alt, T. Breitner, **A. Gomez Ramirez**, T. Kollegger, M. Krzewicki, J. Lehrbach, D. Rohr, and U. Kebschull. *"The ALICE High-level Trigger read-out upgrade for LHC Run 2"*. Journal of Instrumentation, 11(01):C01041, 2016. [23]

5. J. Lehrbach, M. Krzewicki, D. Rohr, H. Engel, **A. Gomez Ramirez**, V. Lindenstruth, D. Berzano, and ALICE Collaboration. *"ALICE HLT Cluster operation during ALICE Run 2"*. Journal of Physics: Conference Series, 898(8):082027, 2017. [24]

6. Ananya, A Alarcon Do Passo Suaide, C Alves Garcia Prado, T Alt, L Aphecetche, N Agrawal, A Avasthi, M Bach, R Bala, G Barnafoldi, A Bhasin, J Belikov, F Bellini, L Betev, T Breitner, P Buncic, F Carena, W Carena, S Chapeland, V Chibante Barroso, F Cliff, F Costa, L Cunqueiro Mendez, S Dash, C Delort, E Denes, R Divia, B Doenigus, H Engel, D Eschweiler, U Fuchs, A Gheata, M Gheata, **A Gomez Ramirez** et.al. for the Alice collaboration. collaboration. *"O2 : A novel combined online and offline computing system for the alice experiment after 2018"*. Journal of Physics: Conference Series, 513(1):012037, 2014. [25]

Chapter 2

Basic Principles

This chapter covers the concepts that form a knowledge base required to understand the scientific contributions of the present thesis. First, an introduction to the components of e-science grid computing related to the Worldwide LHC Computing Grid (WLCG) and the ALICE experiment is presented. A generalization to the case of grid computing is further provided. Then, the security by isolation (SbI) topic with a focus on Linux containers (LCs) technology is explored. Finally, the machine learning (ML) and deep learning (DL) principles that we apply for classification and data generation used for training dataset improvement are described. We start with the discussion of e-science computing.

2.1 E-Science Computing

Our work is placed in the context of e-science computing security research. E-science is usually described as the usage of distributed computational resources in the research of scientific problems that require highly intensive data processing [26]. It is seen as the collaboration among independent research organizations and possibly single researchers within the organizational framework of a virtual organization (VO). The collaboration may include international members and distributed facilities. Unified data acquisition and distribution of scientific data is typically used. Data replication throughout a geographic and organizational distribution is another important characteristic. A dedicated service layer is provided for researchers that can conduct the simulation, preprocessing and analysis of scientific data by the submission of predefined task descriptions, which are executed within the infrastructure without their required presence [27]. These researchers are technically enabled to supply arbitrary program code and data, and to use them in line with their research requirements. We restrict our view to e-science grid computing. In general, the computing grid has been envisioned as an analogy to the electrical grid, but with computing resources on demand mainly for scientific purposes.

2.1.1 E-Science in High Energy Physics

High energy physics (HEP) has been one of the most successful areas of application of e-science grid computing. This thesis is carried out in the context of the ALICE (A Large Ion Collider Experiment) experiment's computing infrastructure. ALICE is a member of the Large Hadron Collider (LHC) at the European Organization for Nuclear Research (CERN). The LHC accelerates protons and lead-ions to 99.9999991% of the speed of light and collides them in four specific sections. A collision section is surrounded by one of four major experiments: ALICE [28], a toroidal LHC apparatus (ATLAS) [29], the compact muon spectrometer (CMS) [30] and the Large Hadron Collider Beauty (LHCb) [31]. A map of the CERN accelerator complex that includes the four mentioned experiments can be seen in Figure 2.1.

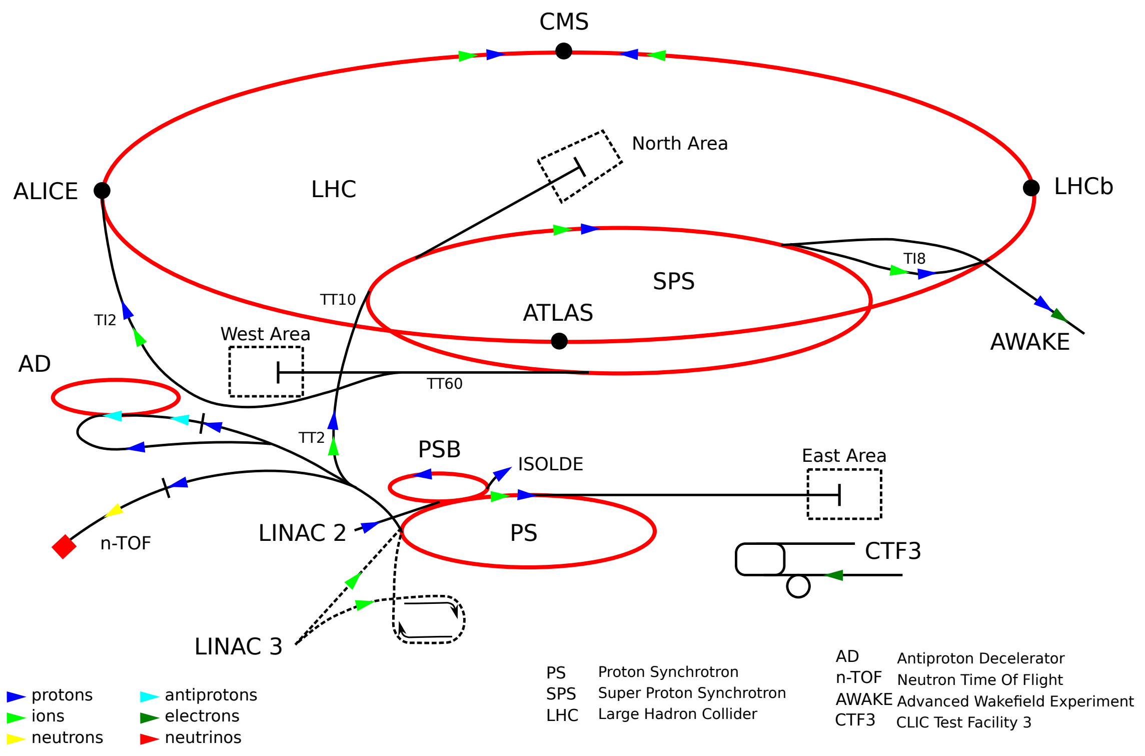


Figure 2.1: The CERN accelerator complex map. The LHC is the largest ring with four main experiments including ALICE [32].

The ALICE collaboration [28] has built a detector to research on to the unique physics potential of nucleus-nucleus collisions at LHC energies. ALICE aims to study the physics of strongly interacting matter at the highest energy densities reached so far in the laboratory. ALICE is a general purpose, heavy-ion detector at the CERN LHC which focuses on quantum chromodynamics (QCD), the strong interaction sector of the standard model of quantum physics. It is designed to address the physics of the quark-gluon plasma at extreme values of energy density and temperature in nucleus-nucleus collisions. It runs with Pb ions, but the physics programme also includes collisions with lighter ions, lower energy running and dedicated proton-nucleus runs. The experiment has 18 sub-detectors, each with its technology selection and design constraints, driven by the physics requirements and the experimental conditions expected at LHC. The data acquisition system of the detector generates raw data sets describing selected particle interactions, within the detector's measurement range and within a certain time frame, at an annual data rate of more than one petabyte per year during LHC operation. These raw data sets are reconstructed to identify for instance the concerned particles, their tracks, energy and lifetime. The reconstructed data is then analyzed in matters of confirmation of existing or formulation of new physics theories and models. Figure 2.2 shows the structure of the ALICE experiment and the required detectors and facilities for collecting collision data.

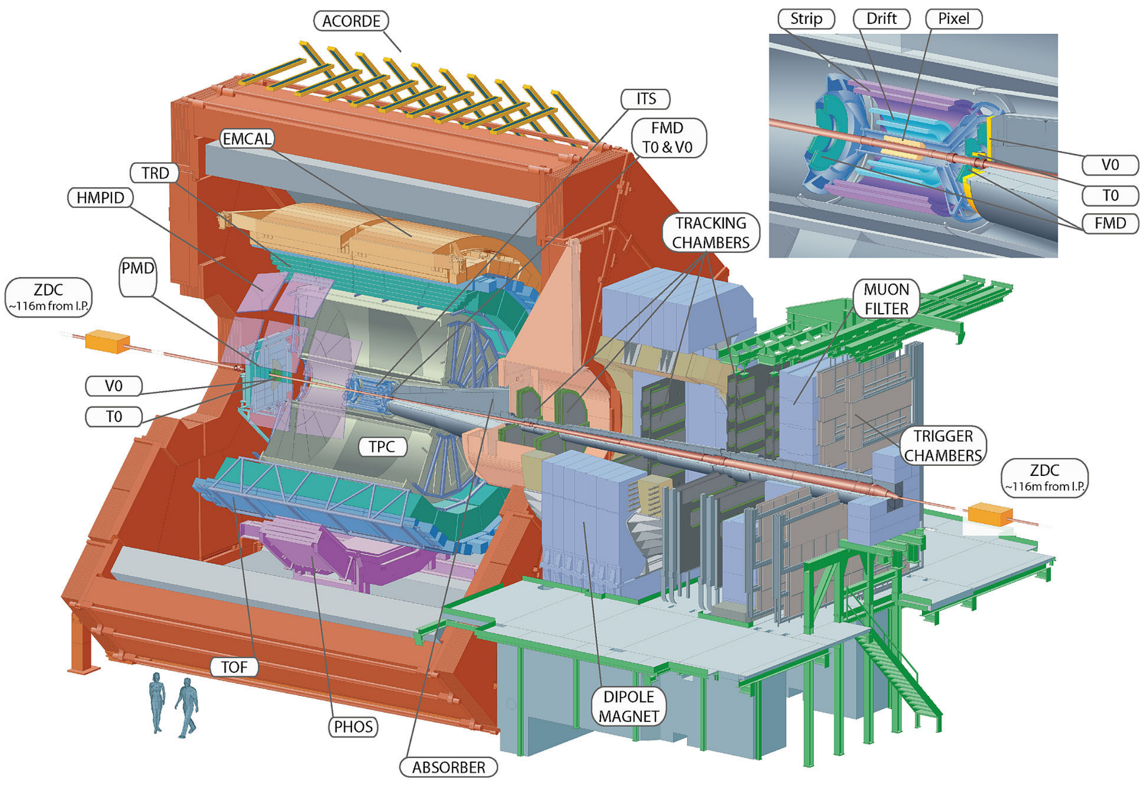


Figure 2.2: Structure of the ALICE experiment with its detectors [32].

The WLCG and the ALICE Grid

The WLCG members allow scientists to analyze massive amounts of physics data produced in the LHC collisions. One of the members is the ALICE collaboration. The WLCG was fundamental to support the experimental validation of the existence of the Higgs boson [33]. It is made of computer centers worldwide that provide computing and storage resources into a single infrastructure accessible by every LHC member physicists. Currently, it combines the power of nearly 170 sites in 40 countries, connected with 10-100 Gb links, with more than 600,000 processing cores and 700 PB of storage capacity. It is capable of processing more than 2 million jobs per day. Figure 2.3 represents the flow of data in the WLCG throughout worldwide Internet connections.

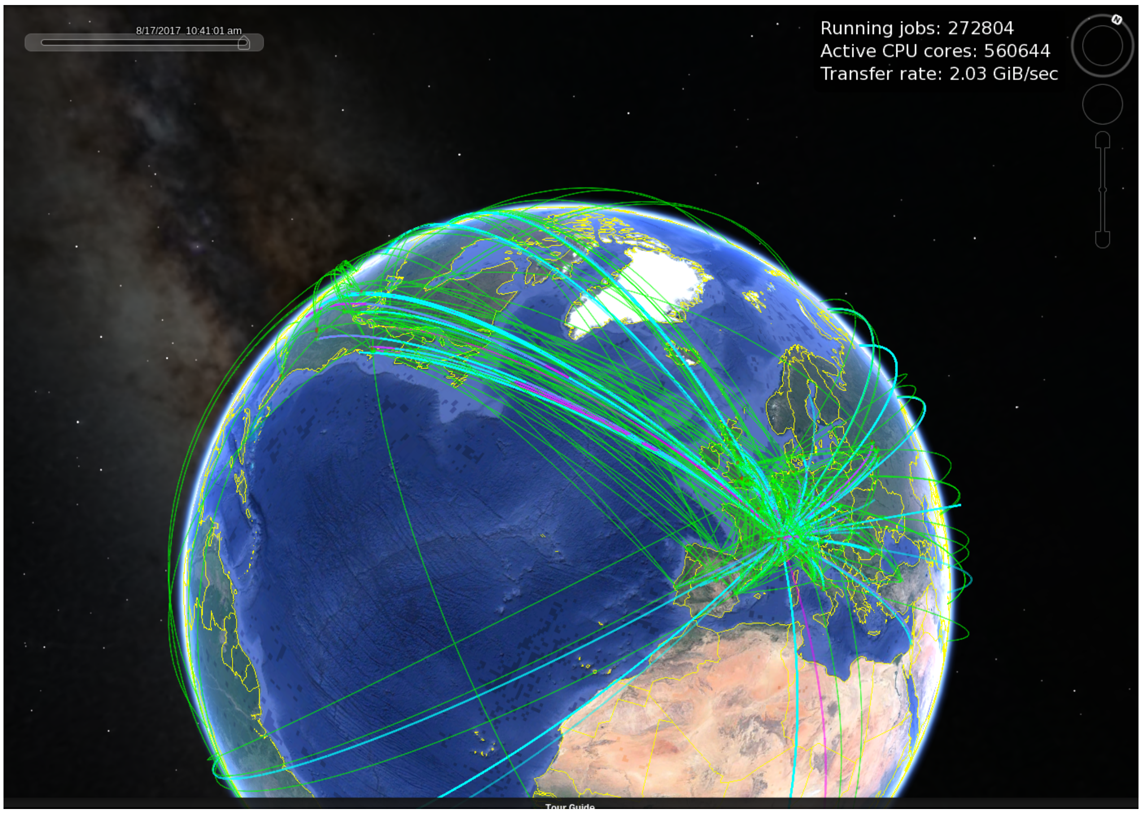


Figure 2.3: Representation of the real-time WLCG data flow around the world. In the upper right corner, information about the significant amount of processing capacity of the LHC grid is shown.

The ALICE grid services [34] are a globally distributed open e-science research cyberinfrastructure. These services are the main computing facility of the ALICE experiment. At the time of the current research, the ALICE grid has more than 70 computing centers, provided by collaborating institutes located in over 30 countries, combining up to 50,000 CPU cores and 30 petabytes of data storage and it provides access to approximately 1,000 active users within the ALICE collaboration. This collaboration as the operator of the ALICE detector as well as the ALICE grid services is an example of an e-science virtual organization [35]. The collaborating institutes should provide storage and computational resources inside their computing centers to the overall grid services. The VO includes both the computing resource providers and the users. The ALICE grid services are used for the main tasks of simulation, reconstruction and analysis of the data collected in the particle detector [36], [37], [38].

The raw data generated by the data acquisition system of the ALICE detector is directly stored in subsystems in the grid services, replicated and processed within the grid. The systems usage and executed tasks are classified into scheduled processing organized centrally for the collaboration and individual user-driven processing. The ALICE experiment has developed the ALICE production environment (AliEn) [39], implementing many components of the grid computing technologies that are needed to store, process and analyze the collected data. Through AliEn, the computing centers that provide CPU and storage resources can be seen and used as a single entity. Any available node executes jobs and file access is transparent to the users; those nodes might be located in any place in the worldwide infrastructure.

The grid services implement a globally distributed file system by unifying a distributed storage infrastructure and a centralized job submission portal. Users of ALICE are allowed to submit arbitrary jobs and data inside a predefined storage capacity quota. Users have to declare though to use the infrastructure only in line with their research within the experiment according to the WLCG security policy [40]. There is a usage policy [41] that states that all data produced by the detector or by data processing in the infrastructure, can be accessed by every member of the collaboration. Access is granted to every user account inside the ALICE grid. The four large LHC experiments ALICE, ATLAS, CMS, and LHCb focus on different research aspects of particle physics and are based on different detectors and grid environments. However, they share the same basic scenarios [36, 42, 43, 44]. Similarly to the ALICE grid, the ATLAS, CMS, and LHCb experiments have grid-based e-science infrastructures for simulation, reconstruction, and analysis of detector data and they are based on worldwide collaborations. These collaborations represent both the grid users and institutes providing computing resources. All of those infrastructures provide data layers and frameworks for submission of predefined jobs for scheduled and user-driven data processing.

2.1.2 Grid Computing Beyond HEP

The usage of distributed high-throughput computing (HTC) farms for data processing tasks has been very successful in other areas beyond HEP such as weather forecasting, brain, and astronomy research, to mention a few examples of e-science projects. For instance, the LIGO collaboration uses a data grid that combines the aLIGO DCS computers with other clusters around the world to handle the considerable computing load for the gravitational-wave analysis from the LIGO and Virgo detectors [45]. In [46], a study of biomedical research workflows and patterns concerning a potential utilization of high-performance and grid computing infrastructures is presented. The use cases include the simulation, preprocessing and analysis of scientific data, based on distributed computational infrastructure and unified cross-organizational access to distributed and independent data sources.

The GEO (global earth observation) grid [47] created a unified data storage of satellite imagery, geological archives and sensors of different organizations and collaborations. It provides facilities for distributed simulation and analysis in geological and ecological research and disaster mitigation [48]. The e-BioGrid [49] is a dutch e-science infrastructure for life science research. It is based on a gateway for biomedical data analysis [50] connected to the national e-science grid infrastructure SURFSara [51]. The gateway provides unified access to computing facilities and software appliances for biomedical research. The e-science infrastructure is utilized for simulation and analysis of biological samples, genomics and nanoscopy, and medical image data. Another instance of an e-science infrastructure based on a gateway to an independent grid infrastructure is the Charité grid portal [52], which has a similar usage scenario in medical research.

In the next section, a description of the ALICE grid services is provided, that helps to get an idea of the general structure of grid computing.

2.1.3 ALICE Grid Services Implementation

ALICE uses a central grid middleware called AliEn. It is written in the Perl programming language [53] and it is based on a distributed service communicating via the simple object access protocol (SOAP) [54]. It uses MySQL [55] databases and the lightweight directory access protocol (LDAP) [56] for persistent data. AliEn provides a globally unified grid layer built by a grid data layer based on a central file catalog as a logical data structure and a distributed set of storage systems given by the collaborating computing centers. It also includes a job layer that serves a central task queue with a workload management system. Beyond the ALICE collaboration as the originator and main user, the ALICE grid service architecture and its middleware AliEn are adopted by the particle physics experiments CBM [57] and PANDA [58] at FAIR in the GSI Helmholtzzentrum für Schwerionenforschung GmbH [59], Darmstadt, Germany.

The grid fabric of the grid services, according to the definition in [60], is built upon core computing systems running in central services located at CERN. A multitude of distributed computing centers is part of it. Each computing center (site) provides hundreds to thousands of worker nodes (WNs) as computing facilities and one or several storage systems, called storage elements (SEs). WNs on a site are aggregated within the site resource management system. They are accessible to the grid services via one or more batch system interfaces, called computing elements. These computing elements are connected to the ALICE grid services via ALICE specific site services, running on a dedicated site based computing system, called VOBox. The portal service to a site, the AliEn ClusterMonitor, is running on a site's VOBox and acts as an access gateway to a computing element and as a communication proxy for grid jobs on WNs. The Linux operating system powers all central services within the ALICE grid as well as VOBoxes and WNs on the site level.

The access to the grid resources for users is granted by using X.509 [61] proxy certificates derived from the actual user's certificate and the verification of the corresponding record in an LDAP directory. Grid users are identified by the username listed within their LDAP record. All authorization and authorship of object entities is based on these usernames. Four user interfaces can reach the ALICE grid services. Two command-line interfaces, a library in the analysis framework and a graphical web interface. The AliEn Perl shell [62] is a command-line shell used by both grid users and administrators. It allows users to connect to central or site-based services. Another command-line interface, the AliEn shell (aliensh) [63] gives an extensive user functionality. It is the primary interface for scientific users. The initial authentication and authorization is based on the same mechanism as within the AliEn Perl shell using the X.509 proxy certificate, while subsequently a session token is negotiated and used by client interface and service.

The AliEn ROOT interface [63] provides access to the grid from within the experiment's physics analysis software framework [64], utilized both by users and within grid jobs as an interface for grid file access. The Alimonitor [65] graphical interface grants user access to monitoring and administration controls. This interface gives access to statistical data.

Several layers can describe the grid structure. Within the grid layer, the ALICE detector is represented by a dedicated user. This user has prioritized access to the virtual file system to allow users the upload and registration of the detector's physics data. The data is physically distributed over sites according to a tiered structure [66], based on performance characteristics, capacity and service level assurances.

The computational layer enables the submission of grid jobs as textual task specifications to a central task queue. A task queue is accessible via central services and managed by a central workload management system [67]. The management system processes and schedules submitted jobs into the queue. It also initiates their propagation to sites and the execution on WNs. The grid job layer can be described as a platform as a service (PaaS) as well as a software as a service layer (SaaS), it has characteristics of both paradigms. The SaaS is set up by a package management and distribution mechanism that gives automatic on-demand installation of grid applications on WNs. Jobs are specified by a job description language (JDL) as a formatted textual specification derived from Condor ClassAds [68]. A job is submitted to the central task queue either by reference to a JDL file in the file catalog or directly as a textual JDL input. A submitted JDL entry specifies at least a logical file name (LFN) as the executable of the job, which is later downloaded to the WN and executed as an operating system process. Consistently with the WLCG scenario, the portal connects to an underlying grid infrastructure providing SEs and WNs accessibility via computing elements. Within the grid data layer, one central logical grid file catalog with a file-based data management delivers a unified grid file system for its users. Users can further submit jobs to a centralized queue, which are consequently propagated to WNs utilizing a pilot job model.

Following we proceed to define a general view of grid computing, based on the description of the ALICE grid architecture.

2.1.4 E-Science Grid Generalization

Given the described characteristics of the ALICE grid services and its AliEn grid middleware, as well as different example architectures, a more general architecture can be defined according to [27]. This general definition provides a reference terminology for the current study. A distributed grid layer is implemented based on central components and services, known as central grid services. Computing centers are resource providers; they will be named sites. A virtual organization or e-science VO provides and maintains the central grid services. Sites accommodate storage and computing infrastructure as storage elements and worker nodes.

The services and entities are connected through private or public networks. The connection between clients and central grid services and between central grid services and site-based grid services is Internet-based by default.

VO users can submit the grid jobs as text specifications to a task queue, to be validated and processed. That processing could include transformations of initial job submission, such as splitting into multiple job requests. The task queue provides workload management functionality and maintains the scheduling and matchmaking of grid jobs and WNs. Site WNs are integrated into the grid job layer by VO services known as grid job agents. These agents use a small part of WN computational resources to establish a connection to its grid infrastructure and advertise the available capabilities. Grid job agents receive job requests from a task queue and run them as operating system processes on the WNs. The grid job requests sent to grid job agents has to define the executable as well as files and libraries, preconditions and target locations for their output, that are required. The grid job agent is designed to execute a grid job on behalf of a grid job submitter or grid user. The grid site agent is executed on a dedicated site based computing system, the VOBoxes. The agent also maintains the submission of grid job agent requests to resource management system of a site.

Grid job requests can specify software library requirements and presume their availability on a WN during execution. This feature is implemented by using a predefined scope of available software applications called grid applications. These grid applications can be available on a site or WN in a static way. They can be installed as well on a WN on demand before a grid job execution. The on-demand installation has to be triggered and processed by a grid job agent. The grid's users can connect to the grid layer via grid client interfaces. They provide access to distributed file systems and allow users to submit and manage grid jobs. Grid users do not have technical restrictions to upload any data, including program code as data, to the grid file system or to register data provided by external sources. The users are free to specify the utilization or execution of any grid file system entries within grid jobs if the access to these entities is authorized. These features can cause security issues in several situations as discussed in this thesis.

After this introduction to the grid computing topic, the background information required to understand our proposed contributions to the grid security area is explored in this research.

2.2 Security by Isolation

Security by isolation is a technique that enforces hardware or software components separation in a computing system. The primary aim of this separation concept is that, when an attacker compromises one of the components, the others should remain safe [69]. The separation is usually implemented by assigning users and applications a specific space on the system that keeps away other users or application. There are several implementation technologies, with several levels of isolation, for instance, virtual machines (VMs), Linux containers (LCs) or the Unix multiuser scheme. There are even security-focused operating systems [70] that advertise SbI as one of their core features [71, 72, 73]. We will give some more details about the most popular approaches, virtual machines, and Linux containers.

2.2.1 Virtual Machines

A virtual machine is defined as an emulation of a computing system [74]. VMs add an intermediate layer between a virtualized operating system (guest) and the actual hardware. This intermediate layer is commonly called a hypervisor. VMs are typically classified into two main categories. The first is bare metal hypervisors that run directly on the native hardware. They carry out their basic routines to manage the hardware resource access. The second are VMs that run on top of a conventional, non-virtualized operating system called the host operating system, integrated into the kernel of the host OS or in userspace. This second category uses the routines provided by the host operating system to access hardware. VMs are known to provide a strong level of isolation among base hosts and OS running in the guest. However, they involve an important performance loss for the guest systems [2].

2.2.2 Linux Containers

The LCs are grouped as a set of processes, that may belong to the same users, running on top of a shared kernel [75], [76]. These processes are isolated from other user's processes in the OS, and they should not affect the host or other containers. LCs take advantage of namespaces to provide a private view of the system (network interfaces, PID tree, mount points). Cgroups are applied to have a limited assignment of resources. Containers can be seen as an extension of the virtual memory space concept to a wide system scope. They provide a set of advantages over other virtualization technologies. They are lightweight and fast on booting. Linux containers have a small memory footprint and are close to the bare metal performance [77]. Figure 2.4 describes a set of containers working together, isolated and sharing the same kernel. In the next sections, we describe some of the main components of the LCs.

Cgroups

Linux cgroups constitute a fundamental piece of Linux containers. Cgroups are a kernel feature that limits the resource usage of the running processes. It was merged into the Linux kernel version 2.6.24. Similar to the processes, cgroups are hierarchical, child cgroups inherit specific attributes from their parent cgroup [75]. The main difference is that many hierarchies of cgroups can exist at the same time on a system. The Linux process model is a single tree of processes. The cgroup model is composed of one or several individual and unconnected trees of tasks. Multiple separate hierarchies of cgroups are necessary because each hierarchy is attached to one or more subsystems. A subsystem is a single system resource, such as CPU time or memory. Linux provides ten cgroup subsystems as listed below:

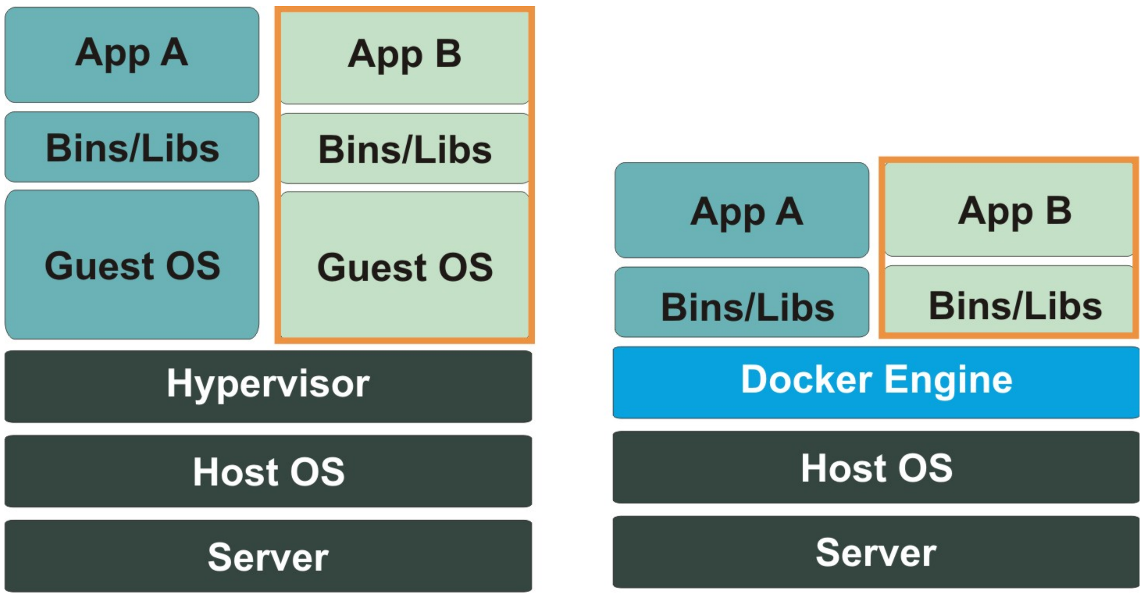


Figure 2.4: The Linux containers have an isolated environment while running over a shared kernel. In opposition, VMs may have several kernels on top of a common hypervisor [78].

- **blkio**: sets limits on input-output access to and from block devices such as drives (disk, solid state, or USB).
- **cpu**: utilizes the scheduler to provide cgroup tasks access to the CPU.
- **cpuacct**: this subsystem generates reports on CPU resources used by the tasks in a cgroup.
- **cpuset**: assigns individual CPU cores and memory nodes to tasks in a cgroup.
- **devices**: allows or denies users to access the devices by tasks in a cgroup.
- **freezer**: suspends or resumes tasks.
- **memory**: this subsystem sets limits on memory usage in a cgroup and generates reports on memory resources used by these tasks.
- **net_cls**: assigns an identifier to network packets with a class tag that allows the Linux traffic controller to detect packets that come from a particular task.
- **net_prio**: provides a way to dynamically set the priority of network traffic per network interface.
- **ns**: is the namespace subsystem, for which furhter details are given in the next section.
- **perf_event**: carries out performance analysis based on cgroup membership of tasks.

User Namespaces

Namespaces are another essential feature of the Linux kernel that makes containers possible. Namespaces isolate and virtualize system resources of a collection of processes. Some of such resources can include process IDs, hostnames, user IDs, network access, interprocess communication, and filesystems. User namespaces allow the system to map permissions per user and group IDs. The LC users and groups may have privileges for certain operations inside the container without having those privileges outside the container. Therefore, the set of capabilities of a process for operations inside the user namespace may differ from its set of privilege capabilities in the host system. For instance, one of the goals of user namespaces is to allow a process to have root privileges for operations inside the container, while it keeps being a normal unprivileged process on the system that hosts the container [79].

Each user ID of a process has two values: one inside the container and another outside. This property is similar to the group ID. Here each user namespace has a table that maps user IDs on the host system to corresponding user IDs in the namespace. In this case, for instance, the user ID 1000 on the host system could be mapped to user ID 0 inside a namespace. Hence a process with a user ID 1000 would be a typical user on the host system while having root privileges inside the namespace. After a short introduction to isolation technologies, we take a look of the intrusion detection and prevention system topics.

2.3 Intrusion Detection and Prevention Systems

Computational systems are not perfect in the real world. They have errors and fail eventually. Given the physical and logical limitations of these systems, computing errors cannot be avoided. Some of these errors could cause computing machines to get broken while others could allow adversaries to get access to sensitive resources. This problem creates a need for continuously monitoring computing systems to find possible sources of failure before errors occur and to find evidence of security breaches. Intrusion detection systems (IPS) are tools designed to detect intrusions by monitoring the current and past events in a computer system or network and search for security incident evidence. The intrusions they analyze are defined as attempts to compromise the confidentiality, integrity, and availability, or to bypass the security mechanisms of a host or network [80]. Modern IDS can also stop current intrusions by real-time analysis of incidents. These systems are called intrusion detection and prevention systems (IDPS) since they can prevent further intrusions that compromise the monitored infrastructure. Instead of passively monitoring the activity on systems or networks, IDPS can dynamically block unauthorized activity before it is completed.

2.3.1 Basic Components of an IDPS

A common IDPS has several fundamental components: sensors (or sources), analyzers, a database and a response engine [80]. The sensor component is responsible for collecting data about the state of the monitored system. This data can belong to a multitude of sources such as network connection traces, system log files, and system call traces, to name a few examples. The sensors can be hardware or software components. The collected information is forwarded to the analyzers for the search of intrusion patterns.

An IDPS is commonly categorized into three main groups depending on where the information is collected from [81]: network, host or application level. A network-based IDS (NIDS) examines network traffic by placing sensors on several points of an organization's network. They monitor the connections and perform local analysis, reporting possible attacks to the network administrators. A host-based IDS (HIDS) collects event information from host nodes. Usually, the collected data belongs to two categories, operating system API call traces and system logs. Operating system API calls are more detailed and better protected than the system logs since they are generated at the kernel. The system logs are more straightforward and shorter than API calls, and hence they can be more easily analyzed. While NIDS may have problems with encrypted connection environments, HIDS provide a higher degree of visibility at the hosts. HIDS can detect attacks that are invisible to NIDS due to encryption for example and can monitor events locally produced by malicious software or other software integrity breaches. However, HIDS are vulnerable if an attacker targets a host and the sensors are compromised. This situation makes their information sources untrusted. An application based IDS (APIDS) analyzes the behavior of applications by reviewing the events stored in their log files or function call traces. The primary objective of APIDS is to detect suspicious behavior of insider attackers exceeding their authorization or allowed valid activities.

The analysis component in an IDPS receives the source data to search for events of potential security breaches. It gets its inputs from one or more sensors or other analysis modules. This component should decide whether an intrusion is currently taking place or has already occurred in the past and provide evidence to validate the produced alerts. The results of the analysis are sent back to the system as additional events, typically representing alarms. There are two main strategies for the analysis of attack occurrence: misuse detection and anomaly detection [82]. Misuse based detection also known as signature-based detection, usually identifies abnormal behavior by matching events against predefined patterns of events which describe known attacks. Since this model looks for patterns known to cause security problems, it is called a misuse or attack signature detection model. The patterns of known attacks are called signatures. A misuse detection model can detect predefined known attacks with high accuracy and with a small false positive rate. However, a significant disadvantage is that this model is vulnerable to novel attacks and it requires to be regularly updated with signatures of such new attacks. Anomaly-based or behavior-based intrusion detection tries to identify anomaly patterns of activities that deviate from a defined standard profile. This approach is based on average usage profiles of users, systems or network. Significant deviations from these profiles are searched to detect security-related problems. Several methods have been proposed to decide if a system is running according to normal behavior. Several of the most common anomaly detection methods employ classification or clustering, statistical methods or information theory [82].

The storage o database component in an IDPS stores data produced by the sensors and the analysis modules. It enables the persistence of event information and grants security researchers the possibility to analyze forensic data. The response module executes predefined actions in case an intrusion is detected, so further damage to the protected systems is avoided. A response is then a set of actions taken after the detection of a security-related incident. There may be as well several levels of confidence in the presence of an attack and an increased level of risk and predefined actions for such different levels. There are two main approaches to intrusion response, passive and active responses. Passive responses involve notifications and alerts to administrators. They have traditionally been used since the conception of IDPS, and they are still present in every intrusion detection product. Active responses are the actions taken to stop the detected security breaches actively. An active response mechanism may have a wide range of options. Some examples may include increasing the alarm level, logging particular events generated for forensic analysis or even kill malicious applications.

2.4 Machine Learning

Finally, the general ideas about the machine and deep learning topics are described. Machine learning is a research area of applied mathematics [83], inspired by human-like intelligence behavior. ML objective is to find methods to replicate the learning abilities of human beings with machines. One of its widespread application is the statistic estimation of highly non-linear functions. ML is frequently applied in automatic classification of data that belongs to unknown distributions, a task that traditionally has been carried out by human operators. A core objective of a learning algorithm is to generalize from the collected experience. Generalization in this context is the ability to perform accurately process new, unseen examples or tasks after having experienced with a training dataset. The training examples come from some unknown probability distribution. The learning algorithms have to build a general model about the data space that produces accurate predictions over new data.

In the context of this thesis, the act of assigning a set of data samples or objects to a set of classes to which the objects belong is called a classification problem. For example, a classification problem is to determine the correct brand of a set of cars automatically based on visual features. This task that traditionally would require human abilities and visual analysis is expected to be the kind of problems that the aid of machine learning technologies for image analysis would solve. More generally, a classifier in ML receives as input data $\vec{x}$, which is a vector of d elements:

$$\vec{x} = (x_1, ..., x_d) \in \mathbb{R}^d, \tag{2.1}$$

called a feature vector. To classify the input $\vec{x}$ means to evaluate a classification function

$$C_W : \mathbb{R}_d \mapsto c_1, ..., c_k, \tag{2.2}$$

on $\vec{x}$. The output of the function is defined as

$$c_{k^*} = C_W(\vec{x}), \tag{2.3}$$

where $k^* \in \{1...k\}$; c_{k^*} is the class to which $\vec{x}$ corresponds, based on the model W [84]. ML algorithms use several methods to find the model W, which is highly constrained by the training dataset. The optimization objective for the model parameters is normally defined as a loss function $\mathcal{L}$ that represents the penalty given to the calculated error in the classification of the training data. The loss $\mathcal{L}(W)$ based on parameters of W is the average of the loss over the training examples $\vec{x}_1, \ldots, \vec{x}_N$, where N is the total number of samples, as follows:

$$\mathcal{L}(W) = \frac{1}{N}\sum_i \mathcal{L}(W, \vec{x}_i). \tag{2.4}$$

The training process consists of finding the parameters of W that result in an acceptably small error, in the best case the smallest one (global minimum).

2.4.1 Deep Learning

Deep learning is a sub-area of machine learning, which is also a component of the artificial intelligence (AI) area. Figure 2.5 shows the relation between deep learning, ML, and AI.

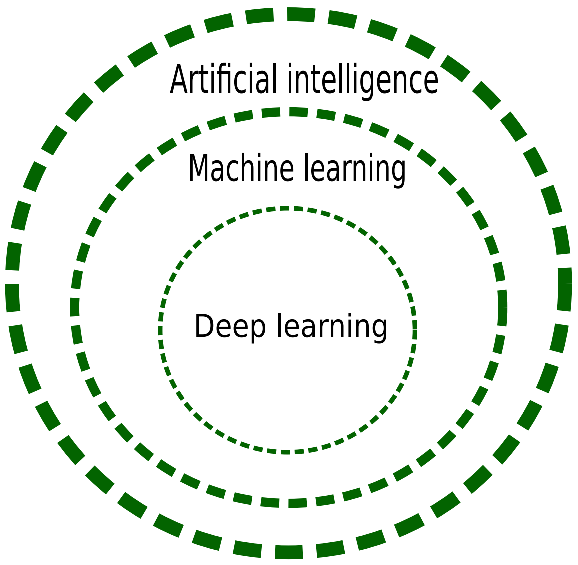


Figure 2.5: Diagram of the relation among AI, ML, and DL knowledge areas. AI is the top general knowledge area, which includes ML and finally, DL is a way of implementing machine learning.

DL allows researchers to solve increasingly complicated applications with increasing accuracy [85]. One of the most popular learning models within DL are the deep feedforward networks. They are also known as feedforward neural networks or multi-layer perceptrons. One of their primary applications is the approximation of functions f^* given some known data points that belong to that functions. A feedforward network defines a mapping:

$$\vec{y} = f(\vec{x}; \theta), \tag{2.5}$$

and it requires finding the value of the parameters θ that enable the best approximation of the desired function. These models are called feedforward because information flows throughout the evaluated function from the input $\vec{x}$, the intermediate computations used to define f and finally to the output $\vec{y}$. If the feedforward neural networks are extended to include feedback connections, then they are called recurrent neural networks (RNNs). We will have a closer look at RNNs in section 2.4.5. Feedforward networks form the foundation of many popular commercial applications. Convolutional neural networks (CNNs) are a specialized kind of feedforward network. CNNs have been used for object recognition on pictures and natural language text classification [15].

Feedforward neural networks are called networks because normally they can be represented by many different functions evaluated in a chain. They can also be described as a directed acyclic graph of consecutive functions. For instance there might be three functions $f^{(1)}$, $f^{(2)}$, and $f^{(3)}$ connected in a chain, to form:

$$f(\vec{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\vec{x}))). \tag{2.6}$$

These chained functions are very frequently used structures of neural networks. Here $f^{(1)}$ is known as the first layer of the network, $f^{(2)}$ is the second layer, and so on. The overall length of the chain gives the depth of the model [85]. The final layer is called the output layer. Therefore, this representation adopts the name of deep learning, given the deep hidden layers in it. During the neural network training, the objective is achieving a close as possible output value between $f(\vec{x})$ and $f^*(\vec{x})$.

The training data provides approximated examples of $f^*(\vec{x})$ with noise, evaluated at different training points. Each example $\vec{x}_i$ is accompanied by a label $\vec{y} \approx f^*(\vec{x}_i)$. The training samples dictate what the output layer should present at each point $\vec{x}_i$. It must generate a value that is close to $\vec{y}$. The training data do not directly specify the behavior of the hidden layers. The learning algorithm should decide how to utilize these layers to produce the desired output. The employed learning algorithm should decide how to update these layers for implementing an approximation of f^*. They are called hidden layers since the training data does not show the desired output for each of them. There are two main categories of learning algorithms that are described in the next sections.

2.4.2 Supervised Training

A learning algorithm used for classification is said to be supervised when the training data is tagged with the desired output for every available sample. It can be described as a training method with a "teacher" who has background information on a specific problem. This "teacher" knows the corresponding output for each entry in the dataset [86]. For artificial neural networks (ANN), the algorithm parameters (e.g., number of layers, number of neurons per layer or the network weights) are iteratively adjusted while the network processes the samples inputs, and the obtained outputs are compared with the expected ones. The optimization goal here is to minimize the error generated by the actual algorithm's output in comparison to the desired output. The final goal is to teach the network to correctly classify new data samples that the network has not seen previously. This goal implies that the network will be able to generalize and it will become useful for practical applications. Some examples of algorithms that can be trained in a supervised manner are feedforward neural networks, support vector machines and radial basis networks.

2.4.3 Unsupervised Training

Unsupervised training is said to lack of a "teacher" with information about the specific application area [83]. Only the training dataset inputs are available, and the desired outputs are unknown. The classification algorithm has to automatically assign a tag or class to each data sample based on the similarities that can be found in the data input features. The algorithm parameters are automatically modified too, based on predefined researcher criteria and the statistical analysis of the available information. Some examples of algorithms that can be trained via unsupervised learning are K-means clustering and self-organizing maps.

2.4.4 Discriminative and Generative Models

There is another distinction between Ml models that we introduce in this section. Discriminative models are those that generate outputs based on predefined specific inputs. Supervised classification algorithms are a good example. Models that can generate both inputs and outputs given some hidden parameters are called generative models (GMs). GMs are used in machine learning for modeling data and observations drawn from a probability density function. Many ML models used for classification employ a discriminative approach, they process input data and give a probabilistic membership value to a certain class. ML methods based on generative models try to learn the probability distribution function that generates the training data (input data space) [85]. Those methods are useful in practical applications, for instance, to create or simulate training samples.

GMs can also be applied as an intermediate step to form a conditional probability density function. A conditional distribution can be made from a generative model using the Bayes rule. Discriminative models do not need to model the distribution of the observed variables. However, they can not generally express complex relationships between the observed and target variables. They do not necessarily perform better than generative models at classification and regression tasks [83]. Both classes are complementary and commonly seen as different views of the same procedure.

2.4.5 Recurrent Neural Networks

Recurrent neural networks [87] are a type of neural network that has frequently been utilized for processing sequential data such as time series. Similarly to convolutional networks that are specialized for processing a grid of values X represented as matrices or tensors, for example, images or videos, RNNs are specialized for processing a sequence of values that are a function of time:

$$x^{(1)}, \dots, x^{(T)}. \tag{2.7}$$

RNNs can scale to long sequences that would not be practical for networks without sequence-based specialization. Most recurrent networks can also process sequences of variable length. They have been used as generative methods with outstanding success in applications. One of those models is especially of interest for this research. The long short-term memory model (LSTM) [88] uses a gating mechanism to ensure proper propagation of information through many time steps. LSTM networks have a specific memory cell and can capture long-term dependences in sequential data. LSTM are valuable tools for language modeling problems [89]. More details about this type of network will be given in the section 4.5.8.

2.5 Summary

An overview of the fundamental principles that form our scientific contributions has been made. First, the components of e-science and grid computing were introduced. A particular focus on the Worldwide LHC Computing Grid and the ALICE collaboration was given. We described the main characteristics of the security by isolation topic. A detailed exploration of Linux containers technology was provided. Several essential concepts of machine learning including deep learning were introduced. We have explained the principles needed in the classification and generation of input data used for training and dataset improvement. In the next chapters, we will introduce the previous related work in our contribution topics. We explain how the basics concepts in this chapter are used to contribute to the improvement of the security for grid computing.

Chapter 3

State of the Art

A survey of the studies that are related to the proposed contributions is presented in this chapter. Grid computing security requirements and constraints related research are explored. The background of the proposed architecture for grid security improvement is deeply analyzed. The topics of security by isolation, applied machine learning and generative methods for intrusion detection systems in the grid are analyzed based on previous approaches.

3.1 Security of E-Science Grid Computing

Grid computing has particular security requirements that make it different from other computational and distributed systems. Several of these requirements shall be mentioned, for instance, in [90], the areas of authentication, access control, integrity, privacy, and non-repudiation are listed as general grid security requirements. A grid security policy is defined based on trust domains as logical and administrative structures governed by a local security policy. A grid environment is based on multiple trust domains with heterogeneous sets of users, resources and local security policies. The interactions between domains are required to be consistent with all affected local security policies. Subjects within a grid environment have global and local representation. They require a mapping between the global and each local trust domain. While the authentication of a global subject and its defined mapping to a local subject is required to be equivalent to the local authentication of the respective local subject, access control occurs only at the local level based on local subjects. The mutual authentication of interacting entities in different trust domains is required. The processes may have delegated subsets of user permissions to act on their behalf.

In [91], the security issues related to grid environments are defined as grid user and account management, user behavior accountability and system usage given the user provided code. In [92], the security topics concerning on-demand grids with large numbers of unknown users are specified as secure application deployment, worker node sandboxing, middleware separation, secure service deployment, and secure workflow execution.

The authors of [93, 94] describe a security analysis concerning on-demand grid and cluster computing. They identify authentication, authorization, delegation, confidentiality, secure communications, data availability and auditing as main security challenges. The authors discuss a scenario with three types of players: resource providers, solution producers, and users. Solution producers supply software and data to their users and utilize the infrastructure of the resource provider for user processing on demand. The use of the third-party code is pointed out as a particular security issue in grid and cluster computing. The authors differentiate between internal and external attackers, with emphasis on the potential masquerading of external attackers, the use of a third-party code, as well as the possible complicity of insiders and external attackers. Three levels of security and trust requirements are defined for on-demand grid and cluster computing environments. Users are always required to trust their solution producer, while lateral or side-by-side operating users and solution producers require no mutual trust, sharing infrastructure of the resource provider. On the first trust level, a resource provider is required to trust the solution producers, concerning the non-malicious use of their environment. Virtualization is presented as an approach to solving this problem on the resource-provider-side.

The study published in [1] creates a taxonomy of grid computing security issues. It establishes three fundamental categories: host level, architecture level, and credential level. The first category is divide into data protection and job starvation, the second one into information security, policy mapping and denial of service. The authors assign different issues relating to the exposure of host systems to data and program code from unknown or untrusted entities and individuals. Virtualization and sandboxing mechanisms are presented as examples of potential approaches to these issues.

A security analysis of an e-science grid infrastructure has been presented in [95], [96], [97], and [98]. The authors define a generic e-science scenario based on the ALICE distributed e-science grid and other e-science infrastructures. The security characteristics and potential attack motives and targets are specified for this given scenario. They have identified several vulnerabilities of the ALICE grid services, especially relating to the deficiencies of unrestricted delegation based on X.509 proxy certificates. Alternative approaches to delegations and their capabilities are proposed in their work. A new delegation framework, the mediated definite delegation was presented. This framework is based on textual statements for both task delegations and attestation of data authorship which are required to be signed using public-key signatures. The framework and its mechanisms are integrated into the defined e-science grid architecture.

The listed requirements, extracted from the previous studies, have a common trust problem. Users of the grid must be authenticated and have the right permissions to access restricted resources, besides, the executed code in the computing farms (sites) should be monitored to detect security incidents. In addition, the monitoring information should be used for forensic analysis if a problem is detected. This project aims to be a robust solution for the code execution trust problem. The goal is to execute the users' payloads in sandboxed environments while monitoring the behavior of such payloads.

3.2 Security by Isolation in Grid Computing

Security by isolation (SbI) is a critical requirement for the grid. The multi-user environment and the external user's freedom to run arbitrary code make SbI a priority. Several related proposals have been made for the grid. The Java virtual machine is employed in [99] as the runtime environment since, the Java's sandboxing mechanisms fit with security isolation requirements and enable fine-grained access, security checks for all Java programs and configurable security policy. This approach is useful only when a JVM supported language is utilized as a base for the grid environment. Virtual machine (VM) hypervisors have frequently been proposed to solve the isolation requirements in grid computing. According to [100], virtualization tools are prevalent in the context of grid computing, given their ability to run several operating systems on a single host and to provide a confined execution environment.

In the context of ALICE, in [101] the authors show how cloud computing resources can be used within the AliEn framework for performing simulation, reconstruction, and analysis of physics data. They deployed a virtual software appliance for the LHC experiments that were developed by the CernVM [102] project. Virtualization tools are envisioned to execute as many VMs per host as possible, in order to run as many grid jobs. This fact raises the concern of how scalable such systems are, considering the performance issues of VMs.

Several researchers have proposed the use of Linux containers (LC) to provide isolation between grid jobs and the underlying system and networks. Docker has been the de facto Linux container solution since its release in March 2013 [103]. However, several containerization approaches have been proposed previously; lightweight operating system virtualization techniques, such as Solaris Zones and OpenVZ [104] are examples. CoreOS [105], an operating system focused on the cloud, introduced rkt [106] as its competing solution against Docker, arguing that Docker does not concentrate on building the best possible container runtime and does not engage in the security problems as much as is required. LXD [107], from Ubuntu, is another alternative for running containers; this provides a lightweight approach for virtualizing a whole operating system. In addition, there is an ongoing effort, called the open container initiative, that is attempting to bring all solutions together by working on a container format specification and a runtime implementation [108].

A noticeable issue to consider is that computing farms used in e-science need a distributed management solution or orchestration system. Container technology requires a management tool for executing applications in distributed environments. Several tools have emerged to provide such a solution. Some of the most popular are Kubernetes [109], Apache Mesos [110] and Docker swarm [77]. In ALICE, Mesos has been deployed as a management system, described in [111]. OpenStack [112], a traditional cloud computing solution, mainly focused on virtual machines, has also been proposed to manage containers instead of common VMs. In [104], a summary of the usage of Docker [77] as a container engine for distributed SaaS environments, such as the cloud and the grid, is presented. The authors state that Docker allows administrators to run composite applications in the cloud.

They claim that their integration components and tool enhancements are the most comprehensive management suite available.

In [76] a performance comparison of several virtualization technologies, including VMs and LCs, is presented. Container-based systems showed a near-native performance of CPU, memory, disk and network utilization. In high-performance computing (HPC) and high-throughput computing (HTC), optimizing the available system performance is a critical requirement. Linux containers help to reduce the overall performance impact while providing the execution of sandboxed applications. The study in [113] analyzes LCs and VMs and finds similar results regarding performance and scalability.

The authors of [114] describe a development of LCs that provide isolation in a grid site at the ALICE High Level Trigger (HLT), a cluster that has physical access to the LHC experiment and, therefore, has critical security constraints. The authors create a set of requirements for distributed container factories. They explore the usage of Apache Mesos [115] as a container orchestration tool. In [111], the same authors present the experience of the ALICE experiment in the deployment and use in production of the Apache Mesos ecosystem for several grid-related tasks, using hybrid OpenStack over bare-metal resources.

This research is the first to explore on isolation mechanisms provided by Linux containers in the grid, that can be combined with a security monitoring system powered by deep learning (DL) algorithms. The information gathered via the LC system monitoring is used to feed a DL-based intrusion prevention and detection system. Linux containers provide isolation for grid jobs and, at the same time, can obtain specific traceability information about individual job activities and their measured behavior. These characteristics make this project different from those previously explored in the grid security topic.

3.3 Intrusion Detection and Prevention

Relevant studies related to intrusion detection and prevention systems (IDPS) are summarized in this section, including the machine learning methods used to improve them and some of the features commonly utilized as input. How these topics have been applied to e-science grid computing research will be reviewed.

In [82], an extensive review of intrusion detection systems is presented. Intrusion is defined in that review as the attempt to defeat the confidentiality, integrity, and availability of valuable information. Intrusion detection is the act of monitoring the events occurring in a computer system or network and analyzing them in the search for signs of intrusions. The cited study presents several open source technologies as the most used solutions for IDPS, for instance, SNORT [8] and OSSEC [9]. False positive and false negative are frequently found metrics, utilized to assess the degree of accuracy of such tools. Relevant features are the set of audit trails (e.g., system logs or system commands) on a host, network packets or connection traces, wireless network traffic and application logs.

The survey [17] makes a summary of the contributions of IDPS in the area of cloud computing. It also describes various types of intrusions affecting the availability, confidentiality, and the integrity of the information in cloud resources and services. They are similarly relevant to grid computing. The authors of that survey claim that the most common attacks present in those environments are:

- Insider attack: authorized users may attempt to misuse acquired privileges.
- Denial of service: an attacker sends a massive amount of network data that overpasses the hosts' ability to respond to legitimate users.
- User to root attacks: the adversary steals valid user credentials or certificates and then escalates privileges via system vulnerabilities.
- Port scanning: an intruder, previous to an attack acquires information about network services on specific hosts.
- Attacks on virtual machines or hypervisor: this means to exploit vulnerabilities in the virtualization software, which allows an attacker to overcome the cloud isolation protection and access sensitive resources.
- Backdoor channel attacks: once a system is compromised, an intruder creates a persistent means to access the affected system in multiple opportunities.

The authors of the mentioned survey also define the hypervisor-based intrusion detection system. This IDPS runs at the hypervisor layer, allowing users to monitor and analyze the communications among different VMs, between hypervisors and VM and within the hypervisor-based virtual network. Therefore, incorporating an IDPS on the VM allows monitoring the activity of the VM itself. The cloud user should be held responsible for deploying, managing and monitoring IDPS on VMs. Placing IDPS on the underlying hypervisor provides the ability to detect intrusion activity, including communication between VMs on that hypervisor. However, a significant amount of data collection reduces the performance of IDPS or causes packet dropping. According to the authors, deploying, managing and monitoring IDPS should be undertaken by the cloud provider.

Several machine learning (ML) approaches have been studied to improve IDPS accuracy and to reduce false positives. ML methods have been traditionally proposed for IDPS to automate the creation of attack signatures and patterns. Common IDPS rules need to be manually created and updated by experts, based on the new attack characteristics while they are discovered; ML offers the possibility of automating this process and adapting to new threats. In [116], a review of the state of the art of ML techniques applied to intrusion detection and prevention is shown. The authors claim that the most commonly used techniques in this area have been the k-nearest neighbor (k-NN), support vector machines (SVM), artificial neural networks (ANN), self-organizing maps, decision trees, naïve Bayes networks, genetic algorithms, and fuzzy logic. These methods are used as single classifier approaches. For hybrid classifiers, i.e., by using several layers of classifiers, neuro-fuzzy techniques and clustering-based approaches have been used primarily for parameter tuning and classification. The single classifiers k-NN and SVMs are very popular; the SVMs can be seen very often in this research area. For hybrid approaches, it is common to implement an integrated framework, where one method is used for feature selection, while another is employed for classification. The KDD99 training dataset [117] is presented as the standard for validation of ML models for IDPS.

In [118], an overview of the use of computational intelligence research on IDPS is described. According to that review, the misuse detection approach is widely adopted by the majority of commercial systems, because it is simple and effective, however, it does have problems to detect new or targeted attacks. Anomaly detection, on the other hand, extracts patterns from the behavioral habits of users or the usage history of networks and hosts. In the intrusion detection field, supervised learning usually produces classifiers for misuse detection from class-labeled training datasets. Unsupervised learning is usually focused on anomaly detection requirements. The authors of the survey present two standard validation benchmarks, the DARPA-Lincoln datasets [119], and the KDD99 datasets [117] as the most popular. According to their work, the most commonly used algorithms are neural networks such as feed-forward neural networks, radial basis function neural networks, recurrent neural networks, support vector machines, self-organizing maps, and adaptive resonance theory.

The authors of [17] describe several artificial intelligence related contributions in IDPS for cloud computing. According to the authors, the objective of employing ANNs for intrusion detection is to be able to generalize from incomplete data and to classify as normal or intrusive according to the analyzed data. Common ANNs used in IDPS are multi-layer feed-forward neural nets and multi-layer perceptrons, with the backpropagation algorithm. Fuzzy logic is utilized for a fuzzy description of intrusions; it allows researchers the flexibility to overcome the uncertain problem of intrusion detection. SVMs are utilized to detect intrusions based on limited sample data. They are useful where the dimensions of data will not affect the accuracy. To select network features, or to determine optimal parameters, genetic algorithms (GAs) are essential tools. GAs can be used in other techniques for achieving optimization and improving the accuracy of IDPS. Finally, a combination of these techniques can be utilized as the so-called hybrid methods.

3.3.1 Intrusion Detection and Prevention Systems for the Grid

Several studies describe the research made on intrusion detection systems that cope with the requirements of grid computing. For instance, in [120], the authors introduce the concept of having a grid-based IDPS, known as GIDS. They claim to design the first intrusion detection system specific to the grid environment. Their system main tasks are: auditing the Globus system and operating system files, checking the log file for breaches, intrusion discovery by searching for anomalies, signature matching, secure communication and monitoring all GIDS servers.

The study made in [121] proposes an architecture using homogeneously distributed intrusion detection servers by employing learning vector quantization neural networks. In [122], the authors state that the existing grid IDPS architectures lack protection against exploits and the typical computer host and network attacks. They describe a distributed GIDS architecture that uses the grid computing resources and covers some predefined intrusions. They re-utilize available IDPS software via standard inter-IDS communication. The work in [123] describes a grid-enabled system area networks trace analysis (SANTA-G), an instrument monitoring framework. This utilizes a relational grid monitoring architecture, helping to generate instrumented monitoring for grid-wide intrusion detection systems.

The authors in [124] study how traditional host-based intrusion detection systems (HIDS) are suitable for the grid environment. They have found that grid attacks are more complicated to catch via traditional HIDS since they cannot identify grid users and they have a performance overhead. Therefore, they present a bottleneck verification approach for building a grid host-based intrusion detection system (GHIDS). In [125], the authors argue that current network IDPS lack the necessary flexibility needed by the grid environment and it can not dynamically adjust themselves to the dynamic grid applications. They propose a multi-agent approach for intrusion detection applied to the grid (MAIDG). In [126], the implementation of a customizable IDPS by workflows is introduced. According to the authors of that work, this approach allowed them to overcome the limitations of IDPS that use “ad hoc” infrastructures without extensibility, adaptability, and scalability. The research in [127] explains a model of intrusion detection, entitled GIDIA; this is based on immunity and multi-agents methods. The authors make a theoretical analysis and show results that their system has an improved self-adaptability and detection rate.

In [128], a streaming database approach as an alternative for traditional log-files or single host databases in IDPS for the grid, is introduced. This architecture allowed the authors to process attack data in multiple sites, giving them performance benefits in large-scale systems. They show two example attacks in a grid environment and the resulting streaming detection logic. In [129], the authors suggest that enforcing security in a distributed system requires more than user authentication and confidentiality in data transmission, stating that the grid and cloud computing IDPS should integrate knowledge and behavior analysis to detect intrusions. They introduce a grid and cloud intrusion detection system called GCCIDS; this, they claim, has an audit system designed to cover attacks that NIDS and HIDS cannot detect. It also leverages knowledge and conducts an analysis. The work in [130] introduces fault tolerance for grid IDPS. It adopts gossip algorithms to correlate the information gathered from traditional high-level IDPS such as Snort. Their IDPS functionality is mainly based on simple pattern matching. In [131], a correlation mechanism for security alerts is proposed; this reduces the number of alerts that administrators have to deal with while continuing the detection of attacks in grid computing networks.

A comprehensive taxonomy of the latest IDPS and alarm techniques to detect and prevent intrusions in cloud computing systems is made in [132]. The authors define a set of challenges of IDPS development in distributed systems such as the cloud computing environment. In contrast to traditional static environments, the cloud requires the monitoring of virtual machines that are dynamically added and removed. The cloud has several system security administrators, however, this creates an adverse effect on intrusion response time. The authors of [132] also state that most of the potential attacks come from insiders. Similar to the case of grid computing, insider attackers already have the freedom to execute code in the protected systems. The shared infrastructure and virtualization technology make these systems more sensitive to security bugs. Any flaw in hypervisors or containers exposes the platform to an adversary being able to overcome the access and control measures. Virtual machine traffic on a virtual host platform creates difficulties on the event visibility since the network interfaces may also be virtualized. Finally, cloud service providers are not willing to provide the security log, audit data and security practice details needed to increase the transparency on security management practices such as auditing, security policies, logging, vulnerability and incident response, thus, dismissing customer awareness. In addition, tracking data across different platforms and access policies of different service providers is a challenging task.

Some studies focus on machine learning theory for improving IDPS in grid computing, for instance, [121] adopts learning vector quantization neural networks (LVQ). The authors describe how LVQ can learn the normal user behavior via interactions with system resources and then it detects any deviation from the normal conduct. These ideas are extended in [133], but adding distributed processing and exploiting the grid resources.

In [134], a grid IDPS based on distributed intelligent agents and soft computing (SCGIDS) is introduced. The authors use a hybrid approach, of a soft computing-based self-organized map dimension reduction technique, a fuzzy neural network, and a genetic algorithm. Detection of host and network attacks with grid specific attacks and customer anomalies are integrated into [135]. For this goal, a feedforward neural network is implemented. In [129], the authors test a user behavior-based technique via artificial intelligence by a feed-forward neural network. In a simulation environment, they set up a test with five intruders and five legitimate users in order the measure the effectiveness of the network. Autoimmune systems and multi-agents with a defined hierarchical architecture are applied in [127]. MINDS (Minnesota intrusion detection system), a tool that uses data mining to identify both known and unknown network intrusions, is described in [136]. MINDS was initially designed to be centralized; the authors developed a distributed model of MINDS via grid technologies, achieving the distribution of services. In this approach, the researchers are not protecting the grid but are using its features to increase the capabilities of their developed system.

A real-world scenario on Microsoft Azure is described in [137]. The authors list a set of challenges they had to face on setting up a machine learning-based security detection in a continuously evolving cloud environment. They argue that ML algorithms and models present results that are difficult to analyze when investigating security incidents in the cloud. They suggest using knowledge-based rules to help to interpret the results and to reduce the total amount of false positives which waste the time of the security analysts. Those rules, similar to the ones used in traditional IDPS, can be located before, during or after the actual detection process by the ML models.

In [138], the concept of a security operation center (SOC) is defined to overcome the limitations of simple IDPS. According to the authors, traditional IDPS are unable to give a global view of the security of a network. Furthermore, a distributed security operation center (DSOC) [139] is proposed. DSOC can detect simultaneous attacks on several sites in a network, providing a global view of the security state. It overcomes the limitation of the SOC and shows better stability in multi-site networks by detecting distributed denial of service attacks. However, according to [131], the DSOC does not give reliable results when deployed in multi-administrative and grid networks.

The WLCG security operations centre (WLGC SOC) working group has recently been stablished [140], defining the security monitoring architecture for the LHC grid. Security monitoring is an essential topic for sites in the Worldwide LHC Computing Grid. This group is analyzing the set of tools available for monitoring the security status of the virtual organizations (VOs) related to CERN. The WLCG SOC is mainly interested in the type of analytics that big data methods provide; their goal is the integration of these tools into an SOC. The open source project Apache Metron [141] is described as an increasingly popular example of one integration tool that implements the SOC requirements. The use of these platforms could be critical for security in modern grid and cloud sites across every scientific discipline. In addition, the WLCG SOC describes threat intelligence sharing platforms as a fundamental component to build trust among the grid community. The group's [140] long-term goal is to develop a scalable SOC reference design. They have started by working on the deployment of MISP [142] (threat intelligence sharing) and the Bro intrusion detection system [7] in several WLCG sites as SOC components.

No previous approach to intrusion detection and prevention in grid computing utilizes deep learning techniques. DL allows researchers to process and extract insights from the vast and dynamic real-time streams of data produced by a grid's payload. There is no study employing generative methods to improve training datasets in IDPS studies. Recurrent neural networks are used in this research for augmenting the available training samples.

3.3.2 Feature Selection for Intrusion Detection

Selecting relevant inputs (information sources) for intrusion detection has a significant impact on the ability to find security incidents. These inputs are the set of measurements taken from the monitored system that is utilized to determine the current security status. Based on such features, a validation benchmark dataset has to be collected. It allows researchers to validate the effectiveness of the developed detection methods. A training and validation dataset for ML algorithms is required as well.

Many metrics and datasets have been suggested and studied in the security community for IDPS research. For instance, a survey on the data mining methods used in IDPS is made in [143]. The authors show that 42% of their analyzed papers use the KDD99 cup dataset [117]. This dataset was introduced in the third international knowledge discovery and data mining tools competition. The competition goal was to build a network intrusion detector based on a predictive model for classification between bad connections, called intrusions or attacks, and good or regular connections. This data comprises a standard set of samples to be audited, which includes a wide variety of intrusions simulated in a military network environment. According to the authors of [143], 20% of the studies analyzed by them use the DARPA dataset, and 38% of the studies utilize other datasets. They compare the effectiveness of their proposed methods for misuse detection, anomaly detection or both by employing such datasets. The DARPA dataset benchmark for intrusion detection [144] is composed of network traffic and audit logs collected on a simulation network made by the defense advanced research projects agency in an artificially created network by MIT Lincoln Laboratory.

In [145], a description of a systematic approach to create IDPS datasets is shown. The authors use this approach to develop their own database [146]. In that research the authors describe other important databases, like CAIDA [147], PREDICT [148], The Internet traffic archive [149], LBNL traces [150] and DEFCON [151].

Grid IDPS Related Data Sets

According to [137], the last relevant dataset for intrusion detection is the KDD99 [117] of network intrusions derived from the DARPA's one in the MIT Lincoln Laboratory. For the cloud, there are no standard benchmarks. The same can be stated for grid computing. The authors of the mentioned study claim that such benchmarks may not exist given the following issues:

- Emulating a realistic test environment with artificially created attacks is difficult. The resulting data set would represent only a static scenario. This characteristic is problematic for a cloud-based or grid system, where virtualized payloads are created and deleted all the time dynamically.
- Real world attacks leave few traces of information in the relevant logs, that can be used to create a training dataset.
- Benchmark datasets for ML such as MNIST [152] for handwritten character recognition can represent an essential part of the instance space. In opposition to this, for IDPS the obtained data for training is assumed to be incomplete since it is difficult to know certainly all possible forms of security incidents.
- Transferring log data from one organization to another is less effective than in other domain problems because security incidents are usually context specific.
- There is no incentive for a compromised cloud organization to share their raw logs to the public since it could generate further negative exposition.

As a conclusion of these restrictions, cloud security data scientists normally should create their baseline datasets. This same argument is valid for the grid. A review in the literature shows that there are not available benchmark data sets for ML-based IDPS training in grid computing. However, several studies describe their custom utilized data and metrics. For instance, [120] audit Globus logs and operating system files. It also checks application log files. In [121], the researchers employed data of one or more log files from system users. The work in [124] uses an operating system kernel module. It explains a method called bottleneck verification by integer comparison by gathering system calls.

In [135], the feature measurements are extracted from audit data of low-level IDPS. To identify misuse committed by grid users, GIDS must analyze their behavior, and that is done with resource usage data like CPU time and memory consumption. The authors gather audit data from HIDS, also extracting operating system (OS) data through the grid middleware and the syslog protocol. In the implementation level, they use OSSEC-HIDS and Snort. The authors of [128] use log-files, packet header information such as IP, port, and packets per time window. The study in [130] uses Snort alerts as the input metrics. The intrusion detection exchange protocol (IDXP) is employed for such a task.

The usage of user-level data is proposed in [127]: user ID, role, type and quantity of resources being consumed. They also use system-level data: CPU usage rate, state of main and the secondary memory and attributes of system files. The identification, type, priority, and status of processes and the states of CPU when they are running are organized into process-level data. IP address and port number of source and destination, type of protocol, flags are grouped into network-level data. The authors of [134] state that many intrusion detection schemes based on system calls have been developed in recent years. In that paper, the extracted features are system calls (ID, return value, return status), process data (ID, IPC ID, IPC permission, exit value, exit status) and file access (mode, path, file system, file name, argument length). The extracted information is normalized between 0 and 1 for the input of a SOM. The work in [133] describes a method using generated log files as host-based intrusion detection.

No standard or recommended dataset for machine learning model evaluation in grid computing for the training of IDPS was found. Therefore, a testing grid site was set up, and a series of tools have been used and implemented to collect a custom set of samples to form a training dataset. This set was collected from the ALICE production grid and a big group of Linux malware. More details about this dataset will be given in the following chapters.

3.4 ML Based Malware Detection

This study focuses on the detection of intrusions that come from the grid jobs. Therefore, a look at the methods used to detect malicious software, such as the detection of malware, is taken here. Some previous approaches such as [153, 154] explore machine learning for malware classification, using system calls as main features for their classifier. In [155], the usage of deep learning for static analysis of malware samples for classification is introduced. However, similar approaches to the grid's specific security requirements were not found. We have found inspiration in the works of [15] for English sentences classification, which was utilized in [155] for static binary file classification according to their x86 machine code instructions.

The recent efforts on the research of malware detection have been put on mobile platforms such as Android. An ML-based method that utilizes 200 features extracted from both static analysis and dynamic analysis of Android apps for malware detection is proposed in [156]. The authors of that research employ a deep belief network (DBN) for the first unsupervised pre-training step. Then in a back-propagation step, the pre-trained neural networks are fine-tuned with labeled values in a supervised method. A cloud-based Android malware analysis service is described in [157]. The authors have created an artificial neural network based malware detection module. It analyzes the application permissions to detect unknown malware. The study in [158] states that the usage of deep neural networks for malware detection has shown promising results, however, deep neural networks are vulnerable to adversarial samples. They propose an adversary resistant method by blocking an attacker's ability to build relevant adversarial samples by randomly nullifying features within samples.

3.4.1 Intrusion Data Generation

Finally, the topic of automatic training data generation is explored. The main motivation is that collecting realistic and relevant benchmark dataset for IDPS is a difficult task. A set of reasons for it have been listed in section 3.3.2. This problem means that the available data may be insufficient in some cases. However, there are deep learning methods that could provide a solution to the lack of good quality data. A particular class of DL algorithms allows researchers to create simulated training data based on the probability distribution of the actual available data. This new data can be used as extra training data to extend the generalization capabilities of a classification system [155].

Recent developments in generative adversarial networks (GANs) suggest they are an exciting approach for increasing attack datasets. [159] and [160] show that generative adversarial networks can produce adversarial malware examples that can deceive an IDPS. In [137], the authors suggest that these same techniques can be used to synthesize attack data for improving evaluation of ML models. There are however two described drawbacks: the initial dataset must be sufficiently large, and researchers have found that GANs are particularly challenging to train. The first requirement is more accessible to overcome in the case of malware analysis. It is more problematic for generating examples of insider attacks for intrusion detection. However, no work going beyond this speculation was found, actually utilizing neural networks to improve the training set for an IDPS ML-based system. There are no previous studies that propose the usage of generative methods to synthesize attack data for improving evaluation of ML models.

3.5 Summary

In this chapter, the related work that is relevant to comprehend the context of the proposed contributions was explored. An analysis of the security requirements of the grid environment has been made. The distributed, multi-federated nature of this kind of system creates special requirements that need custom security solutions. Even though traditional intrusion detection and prevention systems have been applied to grid computing, the previous analyzed studies do not comply with all the security requirements.

Trusting arbitrary third-party software is one of those issues. Users can run code and upload any data. This situation needs a sandboxing mechanism to protect the participant computer centers. Another challenge is the forensic and monitoring traceability of the jobs. Traceability means in this scenario, the activities that a job performs that allow the administrators to detect undesired behavior but also the forensic data collected when a malicious process is observed. Virtual machines have frequently been suggested to overcome the code trust problem. VMs provide a reliable processes separation by machine level emulation. However, Linux containers provide a set of features, mainly related to application performance and resource usage, that make them more suitable to be used in the grid to provide a trusted execution environment as sandboxing tool.

Regarding the issue of job traceability and security monitoring, several machine learning based methods have been proposed to improve the accuracy of grid IDPS. The advantage of using ML methods instead of the standard rule-based IDPS is that ML can be dynamically and automatically adapted to novel threats. Support vector machines and artificial neural networks are two very commonly used algorithms for ML-based intrusion detection. According to the state of the art review, the most used metrics on IDPS research are the feature space in the KDD99 dataset and the OS system calls. In the case of the grid, some features such as network trace data have frequently been employed.

Chapter 4

The Design of Arhuaco

The grid systems have dynamical environments where user-controlled applications cannot be seen and analyzed statically. The application execution life cycle depends on the users, and therefore the behavior is unpredictable. Users can run arbitrary processes inside the collaborating computing infrastructure. Executing user processes in a sandboxed space is a requirement to keep the systems safe. Monitoring the jobs behavior is required to detect malicious activities. In this chapter, proposed architecture to improve the grid security is described. The usage of security by isolation (SbI) for process sandboxing and enabling security monitoring properties via Linux containers is introduced. Deep learning (DL) is employed to analyze the collected real-time monitoring data from running grid jobs. An approach of hybrid supervised classification is defined, using word2vec [13] for feature selection and preprocessing of text like input data extracted from the system call and network traces from the jobs. The model proposed for classification of jobs is the convolutional neural network (CNN). The utilization of a generative method to enhance the training data is also provided. Recurrent neural networks (RNNs) are suggested for the generation and simulation of unseen training data that complements a collected dataset. A benchmark dataset for intrusion and malware detection in the grid was created.

Arhuaco, the proof-of-concept implementation of the described ideas, is introduced. Arhuaco is designed with a focus on the security of e-science grid computing, but it can be useful to a broader range of application such as microservices in cloud computing. The discussion starts with the definition of the threat model for e-science computing grid.

4.1 Grid Threat Model

A threat model is used to specify a set of security requirements and their scope. The goal of threat modeling is the optimization of system security efforts by identifying objectives and vulnerabilities of a specific system. Hence, a set of countermeasures to prevent or mitigate the effects of threats can be defined [161]. A threat is a potential situation arising from a malicious agent that inflicts harm or loss to a defined system asset. Threats are often combined, and the exploitation of one of them might increase the probability or impact of others [27]. The assets and security risks associated with an e-science virtual organization (VO) are described in the following section. The list of assets defined in a Worldwide LHC Computing Grid (WLCG) risk assessment document [162] is used as a starting point.

4.1.1 Security Risks Associated to E-Science Infrastructure

Risk management helps a project or organization to evaluate the threats that could affect them and to prioritize efforts to manage these risks. An asset is a valuable object, quality or resource of an organization, in this case of a grid collaboration [27]. Assets define what the organization should protect, the security policies, procedures, and the required controls. Any measure should be evaluated according to the level of protection it provides for the assets. The following is the list of assets that are important for grid systems such as the WLCG [162].

- **Collaborating participants trust**: the trust between the VO participants, collaborating infrastructures, external partners and funding agencies that should be encouraged and maintained.
- **Reputation**: the opinion of the general public, funding agencies and participants about the VO collaboration safety.
- **Intellectual property**: copyright material and the results of scientific work developed on the VO resources.
- **Data protection**: the sensitive data collected, stored and processed by the VO resource, for instance, personal information of the users.
- **Digital identities**: credentials and attributes that allow the VO to authenticate and authorize the users and services.
- **System resources**: the physical or virtual components that enable services to perform calculations, for instance, the worker nodes (WNs).
- **Storage resources**: the physical or virtual components that are utilized by services to store experimental data.
- **Network resources**: network components that allow the VO participants to cooperate and users to access VO resources.
- **Services**: the computing or software systems, that give access to information or control tangible assets. Some examples are the services necessary to the usage, support, operation and monitoring of the infrastructure.
- **Data integrity**: the scientific data stored on VO resources that should be protected against on-purpose modification or corruption.

In the following section, a set of general security characteristics to be considered when modeling grid safety requirements and the solutions for such conditions is defined.

4.1.2 Security Characteristics

Participant entities of the grid have to communicate via an insecure public environment. The Internet as a public network infrastructure or other shared and uncontrolled networks determine the connection of distributed systems and services within a grid. The control and administration of site infrastructure and the scheduling decisions are made at the site level, and a site is free to share its facilities with other parties and collaborations. Sites are resource providers that control their infrastructure given their physical and administrative access. The provision and sharing of resources depend on site decisions as a provider, and it is only regulated with the use of service level agreements. Disruption with other consumers must be taken into consideration. A grid supplies users with access to sites and WNs with a software as a service (SaaS) or platform as a service (PaaS) layer. Users are allowed to submit and execute arbitrary program code and data. The payload sent to WNs is assumed to be free to connect to arbitrary locations via private, shared or public networks, such as the Internet, for instance, to download data or program code during the execution sequence. Users are assumed to be enforced only by a policy to utilize the infrastructure for the approved and valid research.

A user can utilize the grid client interfaces to connect to the grid layer from any personal computing device, even though the integrity and security of such devices cannot be assured. A VO does not have control over grid client environments. The control of the users over their computing systems is granted. Given that VOs are based on an international collaboration of different organizations and institutions, the relations between entities must be respected. A grid represents a conjunction or intersection of independent organizations or institutions as legal entities in different countries or states. Therefore, legal concerns, like liability or due diligence, apply not only within the grid as such but also mutually between each of the interacting entities.

Frequently, jobs running in the WNs have more access rights than is required, sometimes they are restricted only by a local user account [63]. If all the jobs are executed with the same local account, an attacker can tamper with another users' jobs. Usually, the processes that spawn from the execution of a job have access to sensitive server data, for instance, to the list of users in /etc/passwd, that can give the adversary vital information to extend an attack and make it persistent for further accesses. Figure 4.1 shows a diagram of a conventional grid execution environment with no isolation among several users. Sometimes all the grid users are mapped to local machine user accounts. This mapping would allow an adversary to compromise the system and attribute the intrusion to other users. Even the pilot job, the process that executes other user jobs, is exposed in this scenario. This kind of setup does not permit to comply with the traceability and secure isolation requirements of the grid.

A threat model should consider not only the requirements of a system but also the attackers that may be interested in that system and their motivations.

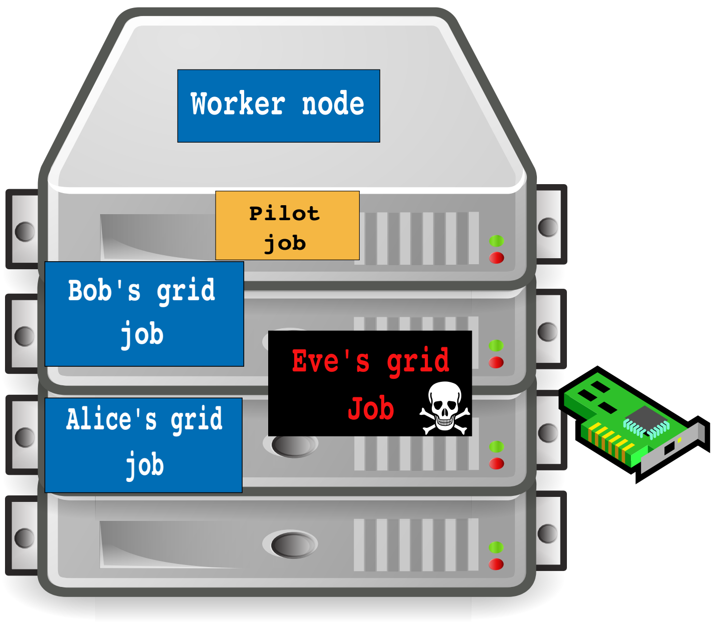


Figure 4.1: In typical and unsafe grid environments, Eve's (an adversary) grid jobs can access Bob's and Alice's jobs to make them attack the infrastructure without leaving any trace. Malicious grid jobs may directly compromise the physical systems and networks.

4.1.3 Attacker Motivation

An adversary is defined as the individual that executes non-authorized actions to increase his assigned authorization and privileges, with the objective of abusing system resources or information [163, 164]. In the grid, an adversary might have goals such as the following:

- Steal critical information like private encryption keys, users' certificates or access cryptographic tokens.
- Compromise users' machines to distribute rogue software, viruses or steal private user information.
- Carry out a denial of service (DoS) attacks with other organizations as targets or even the current running computing center.
- Abuse the grid resources for criminal or not allowed activities, for instance, to deploy botnets or mine crypto-coins.
- Induce damage to an organization's reputation by using resources to attack other entities or merely exposing the weakness of its protection measures.

In general, the potential motives of attackers to compromise targets related to the grid should consider the relationships among multiple entities, the shared infrastructure and the interconnections with other parties. This consideration has, for instance, an impact on the distinction of insider and outsider attackers, due to the overlap of system layers in the grid. Next, an approach for discerning the possible view of adversaries, attack motives and targets in grid computing is given:

- An attacker may target the primary purpose of a VO. The threat might be intended as an interference with the daily operations. This motivation has the VO as the principal objective. Some examples are attempts to delete, alter or produce false data or obtain read access, with the goal of damaging, influencing services or manipulating a data taking process.
- An attack that alters a VO but does not directly points at the primary purpose of it can be defined as an attack with a secondary target. It can be the attempt to hinder VO members from doing their tasks or potential utilization of the system as an interface or intermediary entity to access infrastructure or operations of sites or other VOs via a WN.
- Any other motivation, for instance, demanding for ransom, vandalism or exploitation of the grid infrastructure for any different objective is an attack with the system as a target. Some examples of this case are the incorporation of WNs as tools for attacks on any third parties via the Internet. Another may be just the desire for destruction.

Finally, to achieve the mentioned potential objectives, an adversary may perform several actions. Generally, an adversary may exploit the vulnerabilities in the security design, implementation or in the social interactions of the organization that is targeting. Following some of the methods an attacker may carry out are mentioned:

- Steal valid user certificates to access the grid and then execute malicious code inside the grid infrastructure.
- Exploit unknown or known but unfixed vulnerabilities in the software or hardware of the grid services.
- Listen to users' network traffic to gather sensitive clear text information.
- Perform a man-in-the-middle attack.
- Tamper with other user jobs.
- Escalate the authorized privileges to gain further access to sensitive and forbidden components.
- Access sensitive server configuration data as a source of information for advanced threats.

The defined threat model gives a set of guidelines to design and build a solution fulfilling the grid security requirements. In the following section, the design decisions and contributions that the architecture grants are described.

4.2 The Arhuaco Architecture

Given the described conditions that define the grid security constraints, there is a fundamental property that needs to be introduced. Measuring the impact of security incidents on grid infrastructures like the WLCG involves finding the degree of compromise that an attack has generated. The level of traceability implemented on the affected services and resources is critical for this to be successful. Traceability is defined as the ability to correlate the identifiable entities in a security-related episode, chronologically. It enables the security teams to identify the cause, the timeline, and the consequences of a safety event. Most occurrences whose origin could not be determined are probably occurring again in the future. Traceability is key to incident response, and a sufficient level of traceability is essential to protect systems such as the WLCG against cyber-attacks. An example of a traceability measure are the system logs, where relevant events with the specific timestamp are registered. Another fundamental property required in the context of grid computing is security isolation. The isolation refers to the separation of untrusted user software from sensitive system resources and other users' applications.

According to the defined threat model, an architecture for the isolation and security monitoring of the grid job payload activities is proposed and designed in this project. This architecture covers the sandboxing of untrusted third-party code and data, the safety monitoring and the traceability of users' jobs. It includes the analysis and classification of the monitoring data by deep learning. Arhuaco, the proof-of-concept implementation of the ideas proposed in this thesis, was created as a host-based intrusion detection and prevention system (IDPS) for grid computing. In this section, the high-level decisions for the proposed architecture are described.

To give an initial context of how Arhuaco could be integrated with the WLCG security context, the WLCG security operation center (SOC) is described. Inspired by the CERN SOC architecture [140], the WLCG is in the process of defining a common distributed SOC for the participating institutes in the LHC grid operations. Figure 4.2 shows the current schema of the CERN SOC. It extends the concept of IDPS by the inclusion of a big data infrastructure and the collection of information from multiple and heterogeneous sources. The collected data is stored and analyzed by libraries such as Spark [165], that provide distributed big data analytics. The SOC gathers and shares threat information throughout a malware information sharing platform (MISP). This platform allows researchers to exchange information between organizations related to the threats they have to face and investigate. The CERN SOC itself can share its analyses. This architecture is integrated with a security information and event management system (SIEM), which is a module that manages information (such as alerts) that comes out of the SOC and presents it to the final user, e.g., the system administrator.

The sources of data of the described SOC are inputs that give information about the state of the system. Some examples are IDPS, system logs, weblogs, network connections. The sources of information are those that give a context about the sources of data such as DNS and DHCP servers, that give identification information of different nodes in the network. A variety of options can manage the

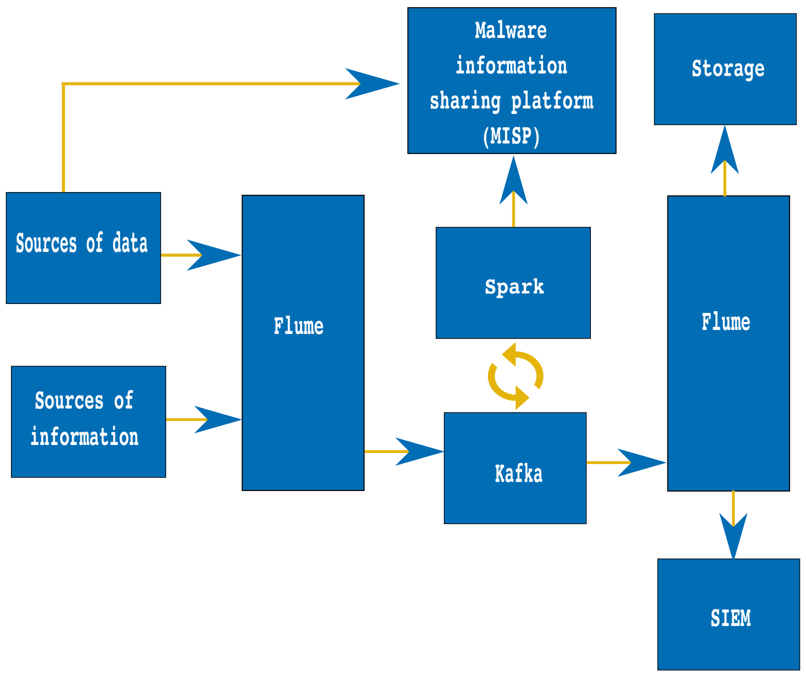


Figure 4.2: The architecture of the CERN security operation center. Based on [140].

storage of detected events, historical information, and forensic data in centralized and persistent repositories for further analysis. Some examples are HDFS [166] long-term storage, Elasticsearch [167] and Kibana [168].

Arhuaco can be integrated into the described SOC architecture. It can be utilized as an additional source of information, i.e., an IDPS. Another option is that it can be a security module for big data processing and analysis. Alternatively, it can be seen as a response engine, performing preconfigured activities on detected security incidents. Initially, the proposed approach has been designed to be deployed locally in the WNs as a standalone endpoint detection and prevention tool. Figure 4.3 illustrates the intended Arhuaco distributed deployment architecture in a grid site.

The proposed architecture is divided into several components with specialized functionality. The execution component provides an interface with container scheduling engines such as Docker. The grid jobs are executed inside Linux containers on the WNs. The necessary communication between jobs with external networks is made via virtual networks. The executed containers are monitored, extracting real-time data for analysis. The data is transformed via a feature extraction mechanism inspired by natural language processing (NLP). The obtained preprocessed input feature vectors are forwarded to the classification and generative components that form the analysis module. The response engine executes actions on suspicious events. These actions can be configured with a set of predefined alternatives, such as sending alerts to administrators, stopping offending jobs, or collecting information for off-line forensic analysis.

Figure 4.4 shows a diagram of the architectured use cases. The users that will

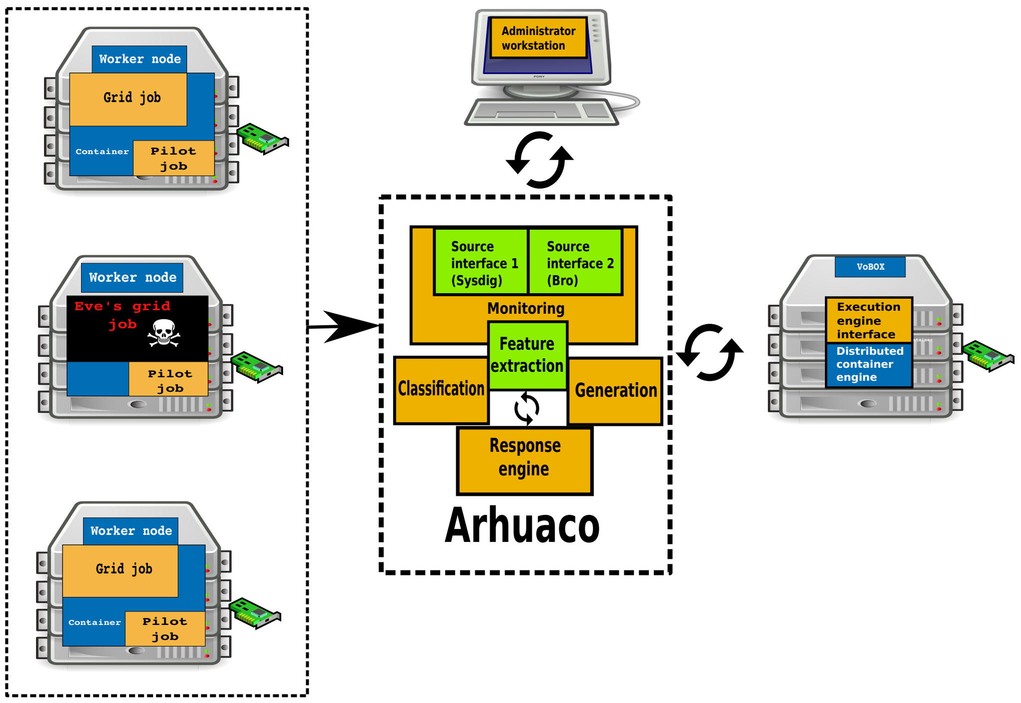


Figure 4.3: A schema of the proposed architecture implemented in Arhuaco: it collects information from the sandboxed grid jobs running in a grid site. The collection and analysis of the data can be done locally in the worker nodes or in a centralized node, such as the VoBox. The system administrator can check the safety state of the system, receive alerts and, configure parameters and response options.

mostly interact with the system are system administrators and security professionals. Administrators will be able to provide configuration parameters, be informed about the current system security health and receive alerts when suspicious incidents happen. Security researchers will be able to review, update and improve the machine learning models. They will also analyze the collected information from the grid jobs to deduce if the system is missing relevant events. The worker nodes are the source of information about the jobs activities and the target for predefined responses. Grid users indirectly interact with the system when they submit jobs to be scheduled for execution in a monitored farm.

In the Arhuaco architecture, the proposed feature extraction mechanism is the word2vec algorithm [13]. These features are the input for the analysis component. The analysis component is composed of a CNN as the classifier and an RNN for language modeling and training data simulation.

Two main approaches are available to collect data from grid jobs for the task of classification. The first option is to extract static information from the files that are part of the job such as scripts and binaries. Many IDPS and antivirus systems look for attack signatures statically via file information, such as binary headers or executable sections. This functionality is often ineffective since the content of the files can be encrypted, obfuscated, encoded by polymorphic code methods

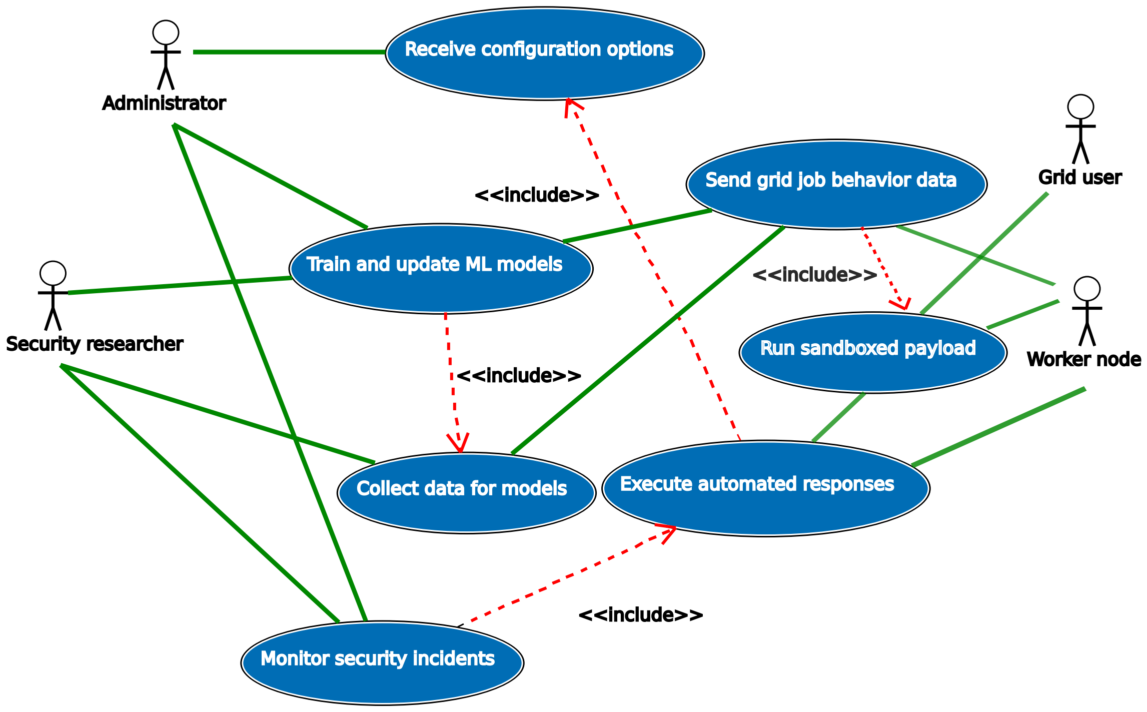


Figure 4.4: Diagram of the Arhuaco use cases: it shows the interaction of possible actors with the proposed architecture.

or a combination of these techniques. Hence, the processes coming from these binaries might only be thoroughly analyzed when they are running, the relevant sections of the executable are available, and their behavior can be understood. The second approach is collecting information from the jobs on execution, which means, measuring specific metrics while they perform their intended activities. This option is called dynamic analysis. Applying dynamic analysis enables the administrators to be alerted about incidents as soon as they are detected in near real-time. It increases the ability of the security crew to defend their systems in an agile manner, something fundamental in a distributed and sophisticated environment with a hundred thousand nodes to be controlled. Dynamic job payload analysis was chosen for the detection of malicious activities in the grid. Arhuaco monitors the processes that are spawned from the job execution, collects relevant information and searches for intrusion patterns in the received data.

In the next sections, a detailed description of the components of the proposed architecture for improved security in grid computing is given. This description is started with the isolation approach.

4.3 Security by Isolation using Linux Containers

The first component of the proposed architecture enables grid jobs to be executed in a sandboxed execution environment by a SbI approach. Linux containers (LCs) are selected in this thesis as SbI provider in the grid environment. They permit equilibrated security vs. performance balance. LCs are suitable to be utilized in high-performance computing (HPC) and high-throughput computing (HTC) scenarios. Besides, a technology easily adaptable in the Linux powered grid computing for high energy physics (HEP) environment is convenient.

Figure 4.5 is a schema of the desired isolation characteristics in the proposed architecture. The advantage of LCs to provide secure isolation for grid jobs is utilized. This isolation protects the system from the jobs running over it, but also the other user's jobs that are simultaneously being executed in the same hosts are sandboxed. The architecture provides a safeguard for the pilot job as well.

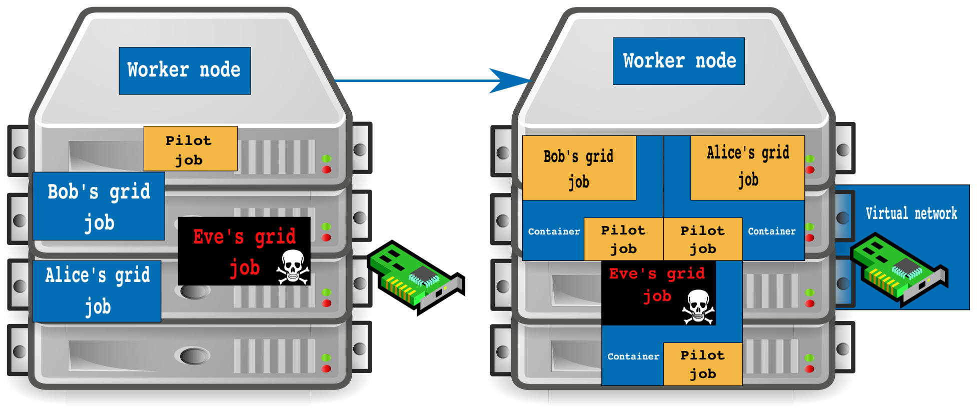


Figure 4.5: In the left, the previously described insecure scenario with adversary Eve having the freedom to compromise the components of the system. In the right, isolation via Linux containers is introduced. In this new scenario, grid jobs are constrained to a sandboxed environment where their behavior can be traced and analyzed to detect intrusion attempts automatically.

An LC is a Linux kernel feature that grants developers the ability to group a set of processes running independently from other processes in the same operating system (OS) while having a unique vision of the entire system [76]. This group of processes coexists isolated from the rest of the machine and by design cannot affect the physical host or other containers. LCs utilize namespaces to have a private view of the shared system. Subsystems such as network interfaces, PID tree or mount points appear to be unique for a container. The security level that this isolation can provide depends on the Linux kernel and the container engine security. Exploitable vulnerabilities allowing a malicious process to overcome the security controls are still a possibility which also affect other technologies such as virtual machines (VMs).

Linux resources can be shared and limited between groups of processes via the usage of cgroups. Containers have emerged as an extension to the virtual memory space concept. Not only the processes appear to have a full space of memory for them, but also they see other subsystems such as the file system and the network interfaces in isolation. LCs have a set of characteristics that provides several significant advantages over other virtualization technologies such as VMs [77]:

- LCs are lightweight in memory consumption. There is no need to load a full hypervisor or machine emulator to execute an entire OS.
- Containers are fast on booting; there is no need for executing full system boot steps. Containers run on top of the already working kernel.
- They have a small memory footprint since no other kernel has to be loaded, only the memory of the group of isolated processes is necessary.
- The processing performance using LCs is close to the bare-metal performance since no instruction, nor machine emulation is needed. The container instructions are executed as userland processes on top of the current kernel.

In Figure 4.6, a set of containers running together, isolated and sharing the same kernel are shown. The opposite case can be found in the VMs that have several kernels on top of a single hypervisor. LCs have seen a considerable increase in their usage in microservice and cloud computing providers because of their performance-related advantages [169].

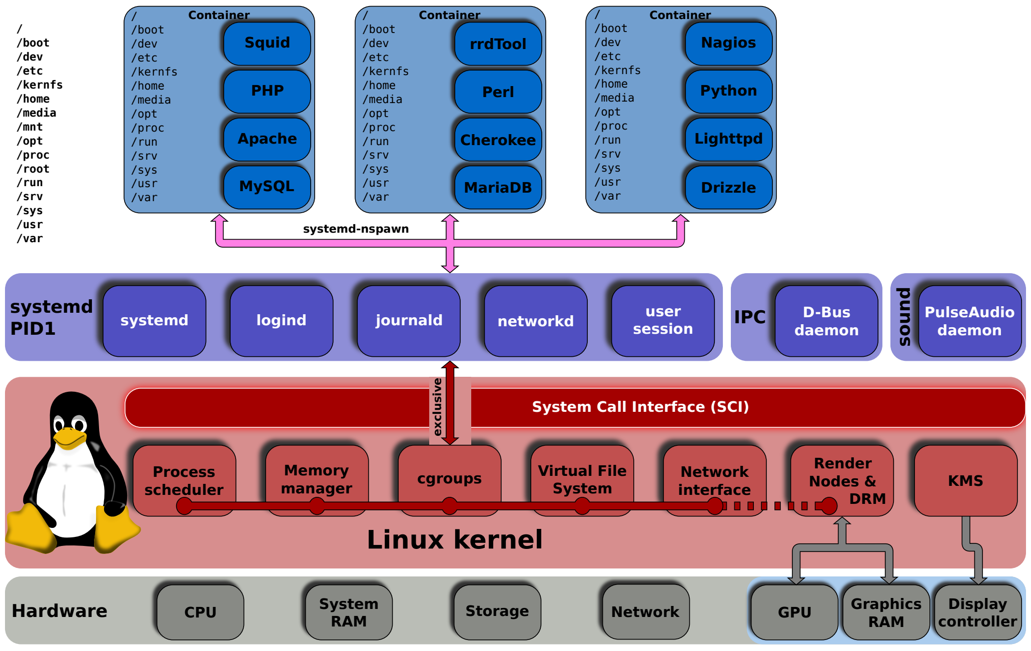


Figure 4.6: The traditional architecture of Linux containers: a group of processes runs confined, with a different self-view of the system, while they share the same kernel. [78].

Some container engines, for instance, Docker [77], provide not only in-host process isolation but they implement additional network isolation. This network isolation makes it possible to create virtual networks on top of a physical or another virtual networks. Docker also contributes with a readily applicable encryption setup for these environments. This network isolation is useful to restrict processes running inside computer farms from accessing sensitive assets in the current network or others. This feature is crucial for protecting grid computing sites, which may share resources and physical infrastructure with other projects or experiments. A real-world example of the need for network isolation is in the ALICE High Level Trigger (HLT) cluster [24] which currently hosts an ALICE grid site. The cluster's resources are shared with third party grid jobs that may have access to the sensitive LHC experiment network [170]. Virtual network isolation is utilized in this case to avoid threats over such sensitive network.

Docker enables developers to assemble several types of virtual networks easily. One of them, the overlay network, provides out-of-the-box traffic encryption and network isolation. Figure 4.7 offers a visualization of the Docker overlay network. Docker Engine version 1.12 (the utilized version) introduced several features including encryption and service load balancing. Besides, other container engine solutions and kernel hardening methods have been investigated to explore further isolation improvements.

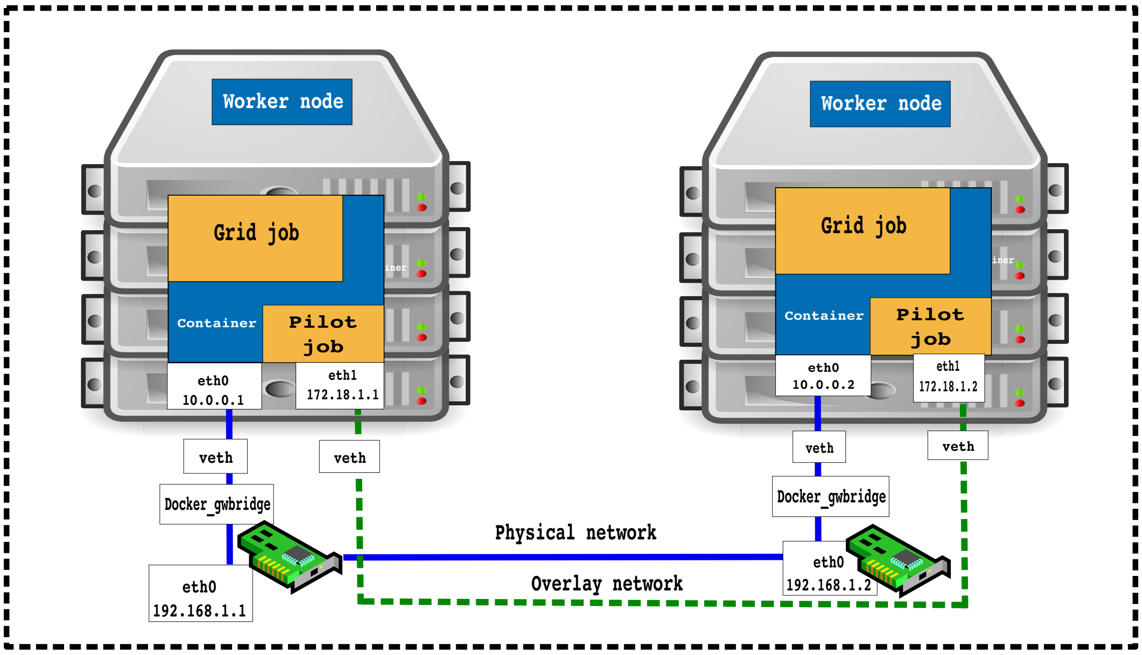


Figure 4.7: A sample of the Docker swarm overlay network on top of a physical network.

The Docker engine handles the native virtual extensible LAN (VXLAN) features from the Linux kernel to create overlay networks. VXLAN is documented by the IETF in RFC 7348 [171]. VXLAN is a network virtualization technology created to address the scalability problems associated with large cloud computing deployments, and it fits naturally in the grid environment. This network type is entirely operated at kernel space. It has been a part of the Linux kernel since version 3.7.

The Docker binary distributions are called images. These images are OS structures that can be run over a different host OS, although normally with a compatible running kernel. They are built from a source file called a `Dockerfile`. The compilation of such files is made by the `docker build` command as shown in Listing 4.1.

```
#!/bin/bash

docker build --tag tests:alien .
```

Listing 4.1: Command line utilized to create a Docker image from a Dockerfile.

Docker utilizes file system layers that are stacked on top of each other to generate container images. When a container is started, a writable container layer is added on top of the other layers. Any changes the container makes to the filesystem are stored in these layers. All changes made to a running container such as creating new files, modifying existing files and deleting files, are written to this writable container layer. The files that the container does not change do not get copied to this writable layer. A storage driver handles how these layers interact with each other. Several drivers are available. The AUFS [172] storage driver has been the default for managing images and layers in Linux. On kernel version 4.0 or higher, OverlayFS [173] is also available. It has potential performance advantages over the AUFS driver. These Docker features create a layer of isolation for grid jobs, the read and write access requests to the file system are not handled by the physical WNs. Therefore, sensitive server files are protected from malicious code.

Beyond Docker, there are several alternative container engines. For instance rkt [106], developed by CoreOS [105], is claimed to be security minded. Its main aim is to be utilized in cloud computing environments. The core execution unit of rkt is the pod, a collection of one or more applications executing in a shared context. rkt allows users to set up different configurations to pod-level or more granular application level. In rkt, each pod executes directly in the classic Unix process model hence being different from Docker which has a central daemon. rkt created an open standard container format, the App Container (appc) spec. However, it can also execute other container images, like those created with Docker. Among the security features that are worth to mention are the following:

- The ability to use KVM for VM-based isolation when extra isolation is needed.
- Integration with SELinux.
- Seccomp filtering enforcement on containers inside so-called pods. rkt leverages systemd seccomp features to enforce isolation by blocking unsafe system calls and hence, privilege escalation.

- Containers are signed and verified by default.
- It is possible to control granular trust permissions.
- It allows users to leverage the Trusted Platform Module (TPM) for container security. It ensures that only trusted containers are executed.

Singularity is an interesting option [174]. It was developed to be mainly applied inside HPC and HTC environments. Singularity containers can be used to package entire scientific workflows, software and libraries, and even data. The Singularity software can import Docker images; there is no need for a Docker installation or superuser credentials. It can be installed inside another container image and shared among collaborators, so there is no need to install missing dependencies. A Singularity's important problem is that it can not set up different OS images on top of the current host OS. For example, it can not execute CentOS guest images on top of an Ubuntu host. However, Singularity can bootstrap from a Docker image. Similarly to Docker, it also has a recipe file containing every command needed to set up and create a container. Figure 4.8 represents a comparison between several virtualization technologies, including VMs, Docker and Singularity.

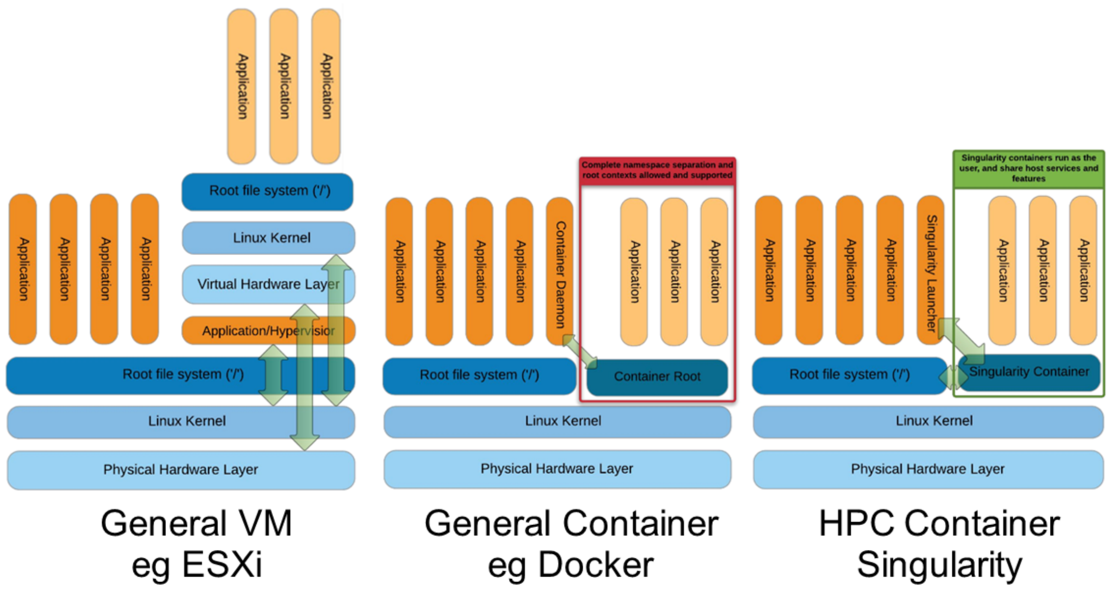


Figure 4.8: Comparison of several virtualization technologies: the Singularity authors claim to have better performance than virtual machines and Docker. However, its focus is not on providing secure isolation [174].

Docker was selected for the initial Arhuaco's proof-of-concept implementation. It is the first container engine supported in the proposed architecture given its broad adoption, it is a pioneer in offering usable containerization technologies, and it is currently the most mature container engine [169]. The network isolation features with out-the-box encryption make it an ideal choice for the needs of this research. Besides, as shown in section 4.3.2, Docker already integrates distributed container scheduling, which is necessary to execute payloads in environments such as Linux clusters. The listed alternative LC engines can still be integrated into the proposed architecture. Arhuaco could support other container technologies beyond Docker as the ones described above. These alternatives were explored in a master thesis that complements this project, supervised by the author of this research, as described in [175].

Access to the CERN high energy physics libraries is needed to run ALICE jobs. The CernVM File System (CernVM-FS) [176] was used in this research project for that purpose. The CernVM-FS is a scalable framework the distribution application libraries in HEP collaborations. It enables administrators to deploy software in a distributed computing infrastructure used to run data processing applications. It allows the WLCG sites to access physics libraries without the need to install dependencies nor worrying about specific versions. CernVM-FS belongs to a suite of CERN-made solutions to facilitate the deployment of virtual infrastructures among their scientist communities. CernVM-FS may replace package managers and shared software areas in cluster file systems to distribute the software utilized to process experiment data. CernVM-FS was deployed in a testing grid site to run jobs to evaluate Arhuaco's features and performance. It enabled the access to HEP libraries for the ALICE payloads. Figure 4.9 represents the different components of CernVM-FS.

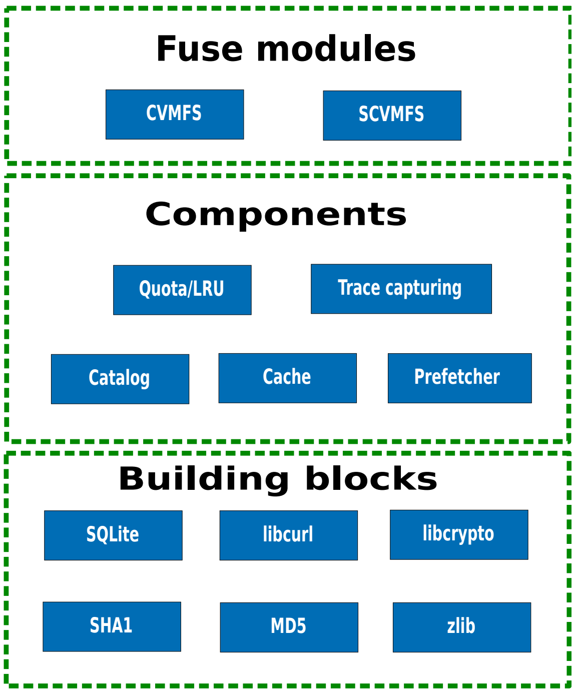


Figure 4.9: A diagram of the main components of the CernVM-FS.

CernVM-FS was implemented as a POSIX like a read-only file system in userspace. There, files and directories are hosted on standard web servers and mounted in the namespace "/cvmfs". In this case, all the libraries can be found in the mounted location "/cvmfs/alice.cern.ch/". CernVM-FS uses content address-able storage and Merkle trees to maintain file data and meta-data. It transfers data and meta-data on demand and verifies data integrity by cryptographic hashes. A custom Puppet module does the installation of this tool on the site for access to HEP libraries. In the next section, a technology that can complement LCs is reviewed, the hardening of a Linux kernel.

4.3.1 Linux Kernel Hardening

The kernel of an operating system is the most privileged and sensitive process running on a modern computer. It might have only fewer privileges than the hardware itself, the firmware and the hardware management engines [177]. In this thesis, a level of trust in the Linux kernel and the underline components such as the CPU itself is assumed. However, critical vulnerabilities do take place even in hardware components such as the meltdown and spectre bugs [178, 179], which make any kernel and userland security measures less effective.

The OS kernel manages hardware resources, interprocess communication, processes scheduling, among other tasks. It should also control the level of privilege of the processes running on top of it. To accomplish this, it needs to have the highest possible privilege in the system. Every user application or data relies on the kernel to remain safe. An attacker that compromises the kernel has almost full access and control over the affected system. This complete control makes the kernel an attractive target for attackers. Hence, it should be protected with strong safety measures.

The security of the Linux kernel determines the safety of LCs. An exploited vulnerability in the kernel would allow an attacker to bypass any security measure provided by a container. However, it is possible to set up software measures that make it much harder to exploit vulnerabilities in the kernel effectively. Those measures involve a certain performance overhead. The best known and most adopted Linux kernel security enhancement methods are the ones provided by the Grsecurity & PaX patches [180]. Grsecurity and PaX [180] are open source software patches for the Linux kernel. Grsecurity creates several defensive and hardening protections. It prevents the kernel to execute code that resides in a memory region owned by a user process, provides greater separation of memory stacks belonging to different processes, grants boundary checking when copying data from a userland process to the kernel, among other features. The patches in PaX add memory safety features. Executable memory is made read-only to avoid buffer overflow exploitable bugs. Address-space layout randomization (ASLR) is implemented to prevent an attacker from being able to find specific addresses in the kernel memory that could be used as the pivot for exploiting memory bugs. These patches also increment the event details in the system logs. Memory access violations, execution filtered system calls, and other events are logged. These improved logs could be used as input for the intrusion detection and prevention system to detect anomalies in running processes. Several tests to measure the impact of these kernel patches to the performance of grid jobs are described. A conclusion is made about the usefulness of such measures.

Linux kernel hardening can be integrated into the Arhuaco architecture to improve the security of running containers even further. This possibility is explored in the master thesis in [175]. In the following sections, how LC containers are executed in a multiple WN cluster is shown.

4.3.2 Orchestration Systems for Linux Containers

Typical grid systems use batch engines such as Condor [3] for scheduling job scripts and binaries in distributed computing clusters. Executing containers instead of standard batch jobs requires modern orchestration tools that concede users the ability to schedule virtualized payloads. Several popular alternatives exist, including Google Kubernetes, Apache Mesos, and Docker Swarm. They are open source applications that provide deployment, scaling, and management of containerized applications.

Google Kubernetes [109] has been developed by Google. kubernetes is claimed to be the result of years of experience running production workloads. It was designed to run billions of containers per week. It provides horizontal scaling, automated rollouts and rollbacks, storage orchestration, self-healing, service discovery, load balancing, secret and configuration management and batch execution.

Apache Mesos [115] was built with a similar design than the Linux kernel. The Mesos kernel is set up on every machine in a cluster and provides applications such as Hadoop, Spark, Kafka and Elasticsearch with an application programming interface (API) for the resource management and job scheduling in data centers. Mesos employes Linux cgroups to provide isolation for CPU, memory, I/O and file system isolation, so it presents itself as an alternative LC engine for Docker but with a distributed orchestration of containers.

Docker Swarm [77] is one of the functionalities of the Docker engine for distributed deployments. It offers cluster management integrated with the engine via the command line interface (CLI). It can be seen as a Docker native clustering system. Docker swarm has been selected as the first supported container management system for the Arhuaco architecture. It extends the LC concept by providing a full distribution of entire OS to be executed on top of a host operating system. For instance, it is possible to run a containerized version of Ubuntu on top of Debian. Since Docker Swarm is integrated with Docker, the container engine selected for this research, it was the sensible collection. It allows the researchers to rapidly prototype and to do experiments in the testing setup.

After exploring the isolation components of the proposed architecture, a description of how containers provide monitoring information that can be utilized to determine the behavior of an application is given.

4.4 Behavior Monitoring via Linux Containers

The ability to encapsulate a set of processes with their view of the entire system brings the possibility to capture specific per-process metrics for their isolated behavior analysis. For instance, resource consumption data such as CPU, memory and disk, network connection data, and system calls for a specific container can be obtained and, therefore, for each running grid job. Then, it is possible to detect with increased precision the source of a security incident or even collect forensics data for further analysis. Traceability is needed to carry out an investigation of security incidents and determine their initial source. It fulfills technical and administrative requirements of the collaborations sites. Isolation and traceability are related. Enforcing one of them improves the another [27].

A usual supposition for normal grid operations is that a small portion of identities and credentials is always compromised. The WLCG security traceability and logging policy remarks the importance of the isolation and security monitoring for grid systems in the document in [181]. According to that document, the aim towards incident detection in the grid is to be able to answer the basic questions who, what, where, and when related to a security breach. For this to be achieved, data related to the behavior of applications running inside the WNs has to be collected.

The Docker engine provides functionalities for container monitoring. Getting basic statistics about locally running containers via the command line interface is simple. An example of the kind of data that can be collected by using the Docker `stats` in the command line interface is shown in Listing 4.2. In that listing, two of the implemented Docker containers are monitored, and their system usage information is extracted.

```
$ docker stats alien1 alien2

CONTAINER  CPU %  MEM USAGE / LIMIT  MEM %  NET I/O
alien1     0.07%  796 KB / 64 MB     1.21%  788 B / 648B
alien2     0.07%  2.746 MB / 64 MB   4.29%  1.266 KB / 648B

BLOCK I/O
3.568 MB / 512 KB
12.4 MB / 0 B
```

Listing 4.2: A sample output of the Docker stats command.

In this thesis, specialized features are used to collect metrics that give information about the security status of the grid. This information is not offered directly by the Docker `stats` command. The specific security monitoring data we initially are gathering for this study is:

- The Linux system calls that are invoked by the processes spawned in the grid jobs, with the full parameter information. The tool sysdig [182] is used for this task. It employs the isolation properties of LCs to monitor the processes inside a specific container.

- Network connection information. A set of features from the connections that the jobs make from the site working nodes are extracted. The Bro network security monitor [7] is the tool utilized for this task.

In the next chapter 5, more details about the utilized tools are provided. The Linux system call set is the primary interface between a userland application and the Linux kernel. System calls are generally invoked via wrapper functions in

Input data type	**Features**
System call	Type, name, parameters
Network connection summary	Source IP, destination IP, original DNS request, source port, destination port

Table 4.1: The format of the utilized input data that is later preprocessed and forwarded to the Arhuaco analysis module.

Glibc [183] or other libraries. The list of those calls made by a process is a rich source of information about its behavior. A standard application cannot make any relevant activity in an OS without calling the OS API, for instance, to access file systems, to allocate memory or to connect to some network. The set of calls and the corresponding parameters enable the extraction of hints about the activities a process is carrying out. System calls are a popular input for intrusion detection systems because a sequence of them can give information about possible security incidents.

Concerning the network input features, Bro allows the users to collect summarized information about the connections made via specific network interfaces including virtual ones. Since virtual interfaces can be created via the Docker overlay network, data can be directly collected from specific processes that belong to particular jobs. Therefore, the behavior of connections made from each job can be analyzed. Even though network information may be extracted directly from the system calls, higher level information provided via `Bro` is much more readable and useful to analyze.

In Table 4.1 the components of the input data that the proposed security architecture utilizes are described. For the system calls, the log lines that sysdig provides are taken, with the type of system calls (e.g., file, network, IPC), the name (e.g., open, read, write) and the specific parameters for each system call. For the network data, a summary of each connection made by the job is collected, including the source and destination IP addresses, source and destination ports and the full DNS request that originated the connection. In the next sections, the transformation of such data into numeric vectors suitable to be used for machine learning algorithms will be explained.

In Listing 4.3, some examples of the data used as input are presented. The utilized features for the deep learning algorithms are extracted from that kind of data. The IP addresses are hidden for privacy reasons. Both the system call and network data logs have variable length strings. The listed data have been already cleaned from characters that do not contribute to increasing the available information, for instance, the parenthesis, coma, and others. I.e., deleting such characters does change the information that can be extracted from the data.

The described input data is transformed into numeric vectors that form suitable inputs for the classification methods in the analysis component of the architecture. In the next section, these methods are described in details.

```
Malware log line samples:
* IP.src IP.dst irc.qeast.net 1 C_INTERNET 1 A
* IP.src IP.dst x.secureshellz.net 1 C_INTERNET 1 A
* IP.src IP.dst irc.server 1 C_INTERNET 1 A
* IP.src IP.dst haxmedown.cz.cc 1 C_INTERNET 1 A
* IP.src IP.dst forum.hy575.tk 1 C_INTERNET 1 A
* file open fd 6  f /etc/passwd  name /etc/passwd
  flags 4097 O RDONLY O CLOEXEC mode 0
* file lseek fd 6  f /etc/passwd  offset 2081 whence 0 SEEK SET
* file read res 38 data gdm session worker pam/gdm password
* file write res 54 data Traceroute v1.4a5 exploit by sorbo
  sorbox yahoo.com .
* file write res 21 data .telnetd

Common grid job log line samples:
* IP.src IP.dst alice-disk-se.gridka.de 1 C_INTERNET 1 A
* IP.src IP.dst p05798818e80213.cern.ch 1 C_INTERNET 1 A
* IP.src IP.dst eosalice.cern.ch 1 C_INTERNET 1 A
* IP.src IP.dst alice-authen.cern.ch 1 C_INTERNET 1 A
* IP.src IP.dst aliendb4.cern.ch 1 C_INTERNET 1 A
* file stat res 2 ENOENT path /cvmfs/alice.cern.ch/x86 64 2.6 gnu
  4.8.3/Modules/modulefiles/vAN 20140629/default
* net connect res 2 ENOENT tuple 0
  ffff8801b64b7bc0 /var/run/nscd/socket
* file write fd 10 f /var/lib/aliprod/.alien/tmp/alien job
  752099366/OCDBrec.root  size 8388608
```

Listing 4.3: A sample of the type of data from which the inputs are extracted.

4.5 Machine Learning for Grid Computing Security

Machine learning (ML) methods are proposed in this thesis for the classification and modeling of data collected from grid jobs. The goal is to learn, based on accumulated experience, to detect malicious activities in the jobs' behavior. The analysis module in Arhuaco includes the implementation of these methods. ML has been created as a tool to automatically assimilate knowledge from raw data as an alternative to expert systems where experience has to be hard-coded by humans. The same principle applies to intrusion detection and prevention systems (IDPS), where commonly the attack detection information is encoded in a set of rules that need to be updated continuously by operators to deal with new threats.

Deep learning (DL) is a technique to implement ML methods via the usage of several layers of functions that analyze and assimilate raw collected data with the goal of getting insights on that data. DL models, mainly represented by artificial neural networks (ANN), have seen increasing success in applications such as autonomous driving, computer vision and financial forecasting. This success can be explained by the availability of more computation power which allows researchers to have bigger models. Other reasons are the availability of huge datasets with examples to train the models and the discovery of techniques to train deeper models with better accuracy.

The usage of ML algorithms for intrusion detection and prevention systems has been described in the literature (As seen in chapter 3) as a step for improving the accuracy and generalization ability of such systems. Traditional industrial IDPS such as Snort [8] and OSSEC [9] employ fixed rules that are hard-coded based on previous known incidents. These rule-based IDPS search for known attack signatures to find possible attacks. They exhibit a low false positive rate (FPR), i.e., a reduced number of false alarms. However, they have problems to detect unknown or slightly different intrusion vectors, hence the rule set needs constant updates [10]. These traditional methods are time-consuming and prone to errors, and machine learning is presented as an automation solution to this problem.

The analysis and extraction of pieces of information representing an object are necessary to feed ML algorithms, for instance, visual representations for the classification of objects in an image. These pieces are called features of the object. In this case, relevant features are needed from the behavior data generated by grid jobs running in a distributed computing infrastructure. As described before, in this research the proposed features use data extracted from the system calls and network traces of the jobs. The reasons for this selection and the feature processing will be explained in a subsequent section.

DL architectures such as convolutional neural networks (CNN), deep belief networks and recurrent neural networks (RNN) have been utilized in computer vision, speech recognition, and emulation, natural language processing, to name a few areas of interesting applications. They have even produced some results comparable or even superior to human experts [184]. This thesis explores for the first time the usage of DL to improving intrusion detection and prevention in grid systems. DL is employed here not only for the classification task but additionally to model the probability distribution of the input data. This modeling allows to simulate new training data to enhance the current dataset and, therefore, to improve the resulting overall accuracy in the classification. In the next sections, a detailed explanation of the classification algorithms used in the proposed architecture is given.

4.5.1 Grid Job Trace Classification

In section 2.4, a definition of classification in ML was provided. Here, the application of that definition to the analysis of grid job data with artificial neural networks is described. First, the input data must be preprocessed and transformed into vectors of numbers so that they can be used as input for the ANNs. The transformed input data is a vector $\vec{x}$ with k elements:

$$\vec{x} = (x_1, ..., x_k) \in \mathbb{R}^k. \tag{4.1}$$

This representation is called a feature vector. The vector is obtained, after a feature extraction step, from the jobs' system calls and network traces. The utilized preprocessing and transformation methods are introduced in 4.5.3. The classification in this context for an input $\vec{x}$ consists of the evaluation of a function that maps a set of vectors to a set of membership classes o groups, such as:

$$C_W(\vec{x}) : \mathbb{R}^k \mapsto y_1, ..., y_H. \tag{4.2}$$

This mapping provides as a result an output:

$$y_h = C_W(\vec{x}), \tag{4.3}$$

where $h \in \{1, 2, ..., H\}$; y_h is the class to which $\vec{x}$ belongs, based on the model W and its parameters. The current classification problem is defined as a two classes decision problem:

$$C(\vec{x}) = W^T \phi(\vec{x}) + \vec{b}, \tag{4.4}$$

where $\vec{b}$ is a vector of bias parameters and $\phi(\vec{x})$ is a feature-space transformation. The training dataset, used to optimized the parameters W, has N samples:

$$\vec{x}_1, \ldots, \vec{x}_N, \tag{4.5}$$

with target values $y_h \in \{normal, malicious\}$.

Selecting the classification model that gives accurate results for a specific problem is a challenging task. In this context, a classifier that can assign the correct category among normal and malicious depending on the input data with the lowest error rate is needed. The usage of ANNs for this task is proposed. ANNs were selected because they have shown big success in automated and complex classification tasks, becoming increasingly relevant with applications of deep neural networks in the last years. They are the most prominent methods in DL [85]. DL is utilized here with three goals: the automated extraction of features from trace data, the classification of normal and malicious jobs and the enhancement of the data with generative methods. It can automate intrusion detection and prevention tasks in very dynamic and data-intensive environments such as the grid. A comparison to a traditional machine learning input extraction and classifier in the task of intrusion detection is further provided.

4.5.2 Deep Neural Networks for Grid Job Classification

The scope of the analysis component of the architecture is to improve the detection of attack attempts in the grid infrastructure that come from grid jobs. This problem is addressed for the dynamic discovery of malicious software, i.e., the detection of malware. Dynamic in this context means that running processes that are spawned from one or more jobs and might try to attack the grid are inspected.

As seen in section 2.1.1, the massive amount of jobs that may run simultaneously in systems such as the WLCG. A big data analysis approach is required to be able to detect intrusions in that flow of data generated by the jobs submitted to the grid. A training dataset with normal and malicious software samples has been collected to accomplish that approach. Instead of creating custom programs to cover a comprehensive set of malicious and normal traces, DL advantages are used to analyze data from already existing examples. This way, a wider variety of training data from the real world is included. Common production ALICE grid jobs were the source for the average data. Linux malware samples were run in a controlled environment to collect malicious data. There are several websites from which it is possible to obtain malware samples. They make them available for the research community. Some of those sites are virushare [185] or via a paid subscription with virustotal [186].

As described in the state of the art chapter 3, machine learning has been suggested in intrusion detection as a tool to model and analyze log files and network traces for the autonomous detection and classification of security incidents. Here, the dynamic analysis of grid jobs powered by deep learning, for near real-time intrusion detection, is examined. Figure 4.10 shows a representation of the idea. The usage of convolutional neural networks as the classification model is proposed. CNNs are is utilized in this research since they have shown a strong ability to model time series data and grid-like data topologies and have been very successful in practical applications such as image classification. CNNs were recently proposed for text classification [15] with promising results. Hence, their ability to classify log traces extracted from grid jobs is investigated. In the next sections, the basic steps to train and test a generic neural network are described, and later the specific details for CNNs are explored.

Neural Network Training

The results that an ANN can achieve depend heavily on the training dataset and the learning algorithm. A training dataset (X, Y) is made of input instances $\vec{x}_i \in X$ and the corresponding outputs $\vec{y}_j \in Y$, which are the labels necessary for the supervised classification. The weights of the ANN are optimized so $C_W(\vec{x}) = \vec{y}_*$, i.e., the class assigned to any input vector is as accurate as possible. A very popular and effective method for the optimization of weights W and bias vector $\vec{b}$, given the training set (X, Y) is the back-propagation algorithm [187].

Back-propagation in ANNs is a method to calculate the error contribution of each neuron after a batch of training data is processed. It is commonly used with the gradient descent optimization (GD) algorithm [188] to adapt the weight of

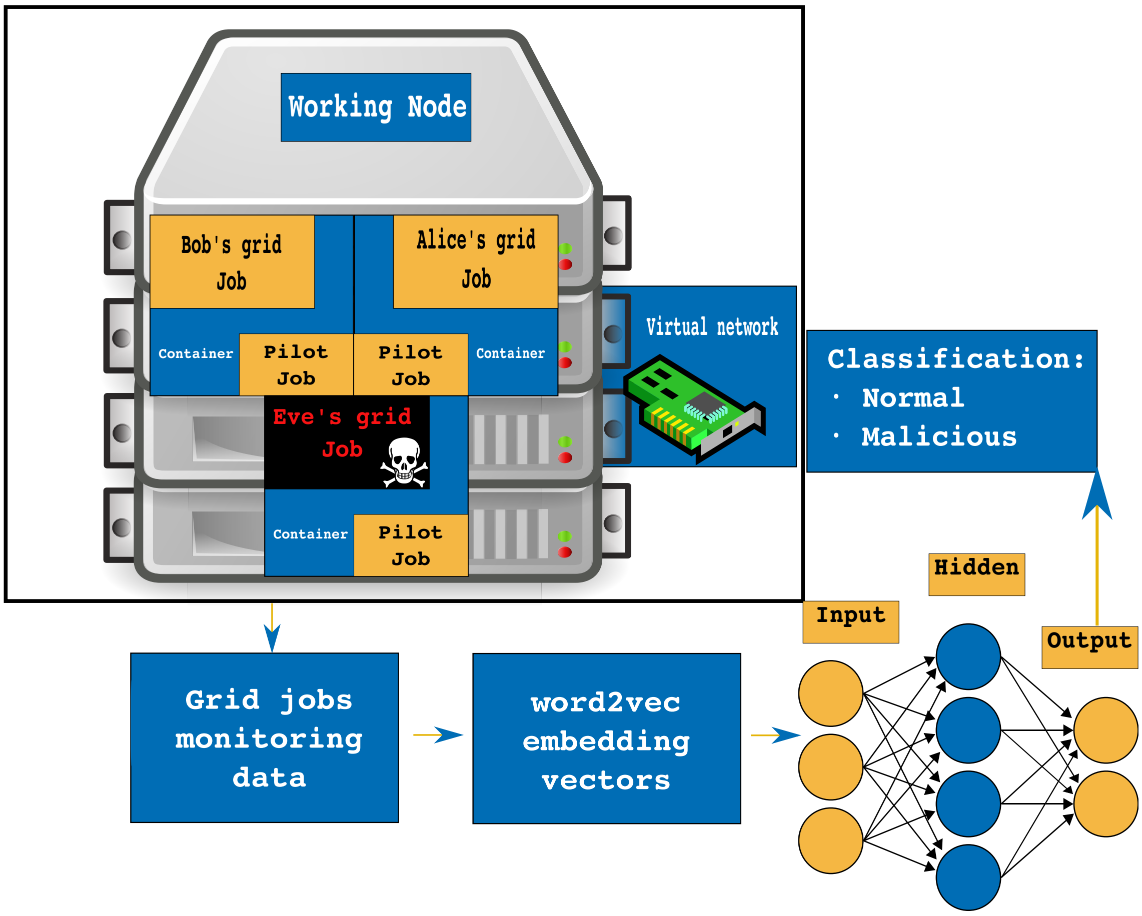


Figure 4.10: Arhuaco extracts behavior monitoring measurements from the confined grid jobs. The data is preprocessed and classified with deep learning algorithms. Finally, grid job traces are tagged as normal or malicious.

neurons by calculating the gradient of the loss function. It is also known as the backward-propagation of errors, given that the error is calculated at the output and distributed back through the network layers. The back-propagation algorithm consists of a step where the error between the predicted $\vec{y}_*$ and the expected $\vec{y}$ is propagated from the output to each neuron. Then, the errors between the actual and expected activations of a neuron are calculated and stored. The backwards-propagation step can be defined as a function $g_W(Y, Y_*)$.

For each iteration of C_W, β is defined as the matrix of all neuron activation values and Δ as the errors in the activation of all neurons. The current β and Δ and a learning rate $\ell \in (0, 1]$ are taken, so W and $\vec{b}$ can be incrementally optimized by calculating the gradient descent. This procedure is also known as batch training, since for each iteration, GD updates the weights according to the collective errors of all instances in X. The algorithm 1 is the pseudo-code details of the gradient descent.

An alternative for GD is the stochastic gradient descent (SGD), which is employed in this thesis. SGD is a stochastic approximation of the GD optimization. Its

Algorithm 1 The back-propagation algorithm for ANNs training with GD batch-training.

```
1: procedure TRAINING_WITH_GD($W, X, Y$, iteration_max)
2:     $W \leftarrow \mu(-\frac{1}{\|\ell_{(0)}\|}, \frac{1}{\|\ell_{(0)}\|})$        # Random initialization
3:     $iteration \leftarrow 0$
4:     while $iteration < iteration\_max$ do
5:         $\beta, Y_* \leftarrow C_W(X)$        # Forward propagation
6:         $\Delta \leftarrow g_W(Y, Y_*)$        # Backward propagation
7:         $W \leftarrow GD_\ell(\beta, \Delta)$        # Weight update
8:         $iteration++$
9:     return $W$
```

goal is to minimize an objective function that is written as a sum of differentiable functions. By using the SGD in the back-propagation algorithm, the weights are updated according to the resulting errors of each sample. GD converges to optimum values generally better than SGD; however, SGD initially converges faster [187]. A crucial advantage of SGD is that it enables adding more training steps of an ANN even after the first training phase is over. New individual samples can be used to update the network weights. The algorithm 2 shows the SGC pseudo-code.

Algorithm 2 The back-propagation algorithm for ANNs. The stochastic-training with SGD is applied sample-by-sample to update the weights.

```
1: procedure TRAINING_WITH_SGD($W, X, Y$, iteration_max)
2:     $W \leftarrow \mu(-\frac{1}{\|\ell_{(0)}\|}, \frac{1}{\|\ell_{(0)}\|})$        # Random initialization
3:     $iteration \leftarrow 0$
4:     while $iteration < iteration\_max$ do
5:         for $x_i$ in $\|X\|$ do
6:             $\beta, y_* \leftarrow C_W(x_i)$        # Forward propagation
7:             $\Delta \leftarrow g_W(\vec{y}, \vec{y_*})$        # backward propagation
8:             $W \leftarrow GD_\ell(\beta, \Delta)$        # Weight update
9:         $iteration++$
10:    return $W$
```

SGC is the optimizer used for the proposed ANN. It has become a very frequently utilized algorithm for ANNs since it enables faster training that can be further upgraded if more data is available. Before giving details about the selected CNN, the process of extracting the features and the input vector creation are described.

4.5.3 Feature Extraction

System calls and network traces give meaningful information about the activities that a process is performing. A different behavior should be expected from regular jobs in comparison to the operation of malicious applications aiming to take advantage of the grid infrastructure. Regular jobs tend to execute specific operations over big pieces of data and to connect to specific Internet services to gather such data. Something that diverges from the normal behavior can be marked as suspicious. Scientists with access to the computing centers that form the grid usually have the freedom to develop arbitrary scripts or binaries. A fixed set of security rules cannot cover all the possible operations they might want to perform. Automated processing via deep learning offers a more convenient approach. It brings the ability to analyze and learn from raw samples without the need for an expert to code these rules. Hence, DL provides adaptation to a dynamic environment in an automated way. It can also extract the most relevant features from the data without human intervention. Given the described advantages, the usage of DL algorithms is proposed, not only for classification but also for raw input data transformation into feature vectors suitable for ANNs algorithms.

A technique to extract the most relevant features from the collected input data should be selected. The raw input data is encoded in a human-readable format. It is generated on-line for real-time analysis and stored as log files so that human operators may read and analyze them. Many host-based intrusion detection systems (HIDS) analyze the system logs to detect attack patterns. Those log files are usually humanly readable, too. Therefore, natural language processing (NLP) techniques become suitable to build a convenient language model [14] that is useful in this research. NLP can easily extend the scope of inputs that can be added to the analysis in the system, just by adding new system logs. The problem can be defined as the classification task with text data as input and two classes as output.

Recent DL proposals for NLP implement learning word vector representations by neural language models [14]. In those vectors, words are projected from a 1-of-T encoding, where T is the number of different words available or the vocabulary size, in a lower dimensional vector space using neural network hidden layers [15]. Semantically nearby words in the training corpus are mathematically close in the lower dimensional vector space. The word2vec algorithm [13] was chosen for Arhuaco to transform the raw input data into relevant feature vectors. It is a predictive model for learning word embeddings. Word2vec vectors create suitable inputs for convolutional neural networks, the proposed classifier, because they enable treating input values as data matrices, similar to the array of pixels in an image. The "tokens" term is utilized instead of "words" since the dataset can also contain numbers, paths, IP addresses, among other types of information.

Before applying word2vec, other preprocessing steps are needed. The characters and tokens in the raw input data that do not increase the amount of available information should be removed, e.g., the parenthesis characters. As described in section 4.4, each input logline is composed of system call code and all the parameters for the system calls and connection information such as DNS, IP addresses and ports for the network data.

Since the goal is to classify grid jobs as regular or malicious, an important decision is to define the level of granularity of the input features. Should every system call or network connection a process makes from start to end be taken as the object to classify? Should single trace lines be classified as unique objects? Should a middle ground be choosen? The first option would not allow the system to analyze data in real-time, because it would have to wait until the program ends to be able to examine all the data. The second option is not useful, because a single system call or network connection is likely not informative enough about the overall job behavior. Therefore, a balanced solution is required, that enables real-time detection while still having enough data to find a trend.

The solution is that the object of classification is defined as l consecutive log lines, a parameter that can be optimized depending on the input data characteristics, e.g., a different number of lines may be selected for system calls and network traces. The input log lines have a variable size. Hence, only the first m tokens per line are taken, adding a padding token when a line is shorter. Then, l consecutive lines are taken, which gives us a final sequence of $n = m \times l$ tokens in total. From these tokens, the input feature vectors are extracted.

To compare the proposed method for classification, with word2vec based input vectors and the CNN, the traditional bag-of-words (BoW) model for feature extraction is utilized as input for a support vector machine (SVM).

The Bag-of-Words Model

The BoW model is a representation used in NLP and information retrieval. The raw input data is represented as the bag of its words (tokens in this case). It omits grammar and word order but keeps quantities. In its simplest form, BoW takes all the different words in a set of texts (e.g., documents or phrases) and creates a dictionary. Then it counts the numbers of occurrences of each word on each text and creates a vector with as many components as the dictionary length. Each component is formed with each different word and its corresponding number of occurrences.

Formally, the objective with BoW is to calculate a score between a token t and a set of phrases in a document d, based on a feature of t in d. A straightforward method is to make the feature equal to the number of occurrences of the token t in d. This number is known as term, token or word frequency and is represented by $\mathrm{tf}_{t,d}$. For a document d, the set of features extracted by tf may be viewed as a quantitative digest of that document. The BoW is obtained from that representation. Here the exact ordering of the words or tokens in a document is ignored, but the number of occurrences of each term is fundamental. Only the information about the number of occurrences of each word or token is retained. It is also common to use the document frequency df_t, defined as the number of documents in a data corpus (e.g., a training dataset) that contain a token t. N is defined as the total number of documents in a dataset, then the inverse document frequency of a token t is defined as:

$$\mathrm{idf}_t = \log \frac{N}{\mathrm{df}_t}. \tag{4.6}$$

The term frequency and inverse document frequency are taken to create a composite weight for each term in each document. The tf-idf weighting scheme assigns to each token t a weight in a document d:

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t. \tag{4.7}$$

Then, each document or set of phrases can be represented by a vector with one feature corresponding to each token or word in the vocabulary, i.e., the set of all different tokens in all the documents, together with a weight for each component that is given by equation (4.7). For instance, vocabulary tokens that do not occur in a document have zero weight. This tf-idf$_{t,d}$ schema is the most common BoW representation found in the machine learning library implementations [18] and hence it is the one used here. The BoW model only accounts for the number of appearances of each token in a document. It does not store semantic information nor order relations of tokens. In the next section, the word2vec method is introduced as a more robust representation model for text-like data.

Word2vec for Feature Selection of Grid Job Data

Word2vec [13, 189] is a set of models utilized to create word embeddings. These models are made from two-layer neural networks trained to reconstruct linguistic contexts of words. It has as input a large corpus of text and produces a vector space. Each unique word in the corpus corresponds to a vector in such space. Word vectors are located in the vector space in a way that makes words that share familiar contexts in the corpus to be located proximately to one another in the space. It has been applied with success in other security areas for extracting features from text-like data [190]. For system calls and network traces the order in which they are made is essential. A specific sequence of system calls, for example, gives different information depending on the order of the calls. Even more, the order of the parameters is critical. So in this project, word2vec is considered to be an appropriate method for the transformation of the input data in word vectors.

The main components of the word2vec model are described, starting with the skip-gram method where a corpus of tokens w and contexts s are defined. The conditional probabilities $P(s|w)$ is considered given a corpus T. Then, the goal is to find the parameters θ of $P(s|w;\theta)$, so the corpus probability is maximized as follows:

$$\arg\max_\theta \prod_{w\in T} \left[\prod_{w\in S(w)} P(s|w;\theta) \right] \tag{4.8}$$

where $S(w)$ is the set of contexts for each token w. Another simpler way to express this problem is:

$$\arg\max_\theta \prod_{(w,s)\in d} P(s|w;\theta), \tag{4.9}$$

d is defined as a document, and it is composed of token and context pairs from the full set of tokens T. The skip-gram model parameters are found via a neural network language model. The conditional probability $P(s|w;\theta)$ model can be then defined with the softmax [191] normalized exponential function as:

$$P(s|w;\theta) = \frac{e^{v_s \cdot v_w}}{\sum_{s'\in S} e^{v_{s'} \cdot v_w}} \tag{4.10}$$

Here $v_s, v_w \in R^d$ are vector representations for s and w respectively, and S is the set of all available contexts. The parameters θ are v_{s_i}, v_{w_i} for $w \in V$ the vocabulary, $s \in C$ the available contexts, $i \in 1, \cdots, D$ the list of all documents. The parameters are assigned such that the product is maximized (see equation (4.9)). An important premise here is that maximizing that objective should result in good embeddings, in such a way that similar words will have similar or "close" numerical vectors.

In [189], a negative sampling method is proposed as a more efficient way of deriving word embeddings. It employs a different objective function than the skip-gram model. A pair (w, s) of token and context are taken from the training data. Then $P(d = 1|w, s)$ is defined as the probability that (w, s) exists in the corpus data. Here, $P(d = 0|w, s) = 1 - P(d = 1|w, s)$ is the probability that (w, s) is not extracted from the corpus data. Again, θ are the parameters that control the distribution $P(d = 1|w, s; \theta)$. The new goal is to find the parameters that maximize the probabilities that all of the observations are extracted from the training data, as follows:

$$\arg\max_{\theta} \sum_{(w,s)\in d} \log P(d = 1|w, s; \theta) \tag{4.11}$$

Furthermore, if the softmax function is utilized in the probability function $P(d = 1|s, w; \theta)$ then:

$$P(d = 1|w, s; \theta) = \frac{1}{1 + e^{-v_s \cdot v_w}}, \tag{4.12}$$

which now enables the definition of a different objective:

$$\arg\max_{\theta} \sum_{(w,s)\in d} \log \frac{1}{1 + e^{-v_s \cdot v_w}} \tag{4.13}$$

A problem with this formulation is the probability that all the vectors could have the same component values. A solution is to disable some (w, s) combinations. This solution is achieved by generating the set d' of random (w, s) pairs that are expected to be incorrect. They should not belong to the training corpus. With this addition the optimization function becomes:

$$arg \max_{\theta} \sum_{(w,s)\in d} \log \frac{1}{1 + e^{-v_s \cdot v_w}} + \sum_{(w,s)\in d'} \log(\frac{1}{1 + e^{v_s \cdot v_w}}) \tag{4.14}$$

If we let $\sigma(x) = \frac{1}{1+e^{-x}}$ then:

$$\arg\max_{\theta} \sum_{(w,s)\in d} \log \sigma(v_s \cdot v_w) + \sum_{(w,s)\in d'} \log \sigma(-v_s \cdot v_w) \tag{4.15}$$

After describing the word2vec model, it is shown how that model is utilized to extract meaningful features from the grid job data. In the defined problem, the word vectors $(v_w)^i \in \mathbb{R}^k$ are k-dimensional token embedding vectors, where k is the embedding dimension, corresponding to the i-th token in the input sequence. n was defined as the total number of tokens that form the "document" d, such as $n = m \times l$, the number of lines multiplied by the number of tokens per line extracted from the system call and network trace logs. A set of token vectors is created by:

$$(v_w)^{1:n} = (v_w)^1 \oplus (v_w)^2 \oplus \ldots \oplus (v_w)^n \tag{4.16}$$

In the equation (4.16), $\oplus$ is the concatenation operator, and $(v_w)^{i:i+j}$ is the concatenation of token vectors $(v_w)^i$, $(v_w)^{i+1}$, ..., $(v_w)^{i+j}$ that belong to the input sequence or "document" d. These v_w embedding vectors are the result of applying the word2vec method on the grid job training corpus. Finally, $(v_w)^{1:n}$ is the input for the proposed CNN classifier, introduced in the next section.

The word2vec models are commonly initialized and trained with substantial natural language corpus with millions of words. For instance, some robust English books are utilized. The word2vec training corpus is composed of the system call and network trace dataset that is used for optimizing the classification model of the convolutional neural network. This model creates word-to-vector embeddings suitable for the context of this problem, where not only words are involved but also many different tokens such as numbers and IP addresses.

4.5.4 CNNs for Grid Job Classification

Convolutional neural networks were first proposed in [192]. They are based on traditional artificial neural networks. The difference is that a convolution operation is performed in one of the CNN layers replacing an ANN matrix operation. CNNs are useful for time series, sequential and grid-like data topologies. They have been very successful in practical applications such as image classification [193]. CNNs were recently proposed for text classification [15] with promising results. Since the input data has a sequential structure, where the order is important, the usage of CNN to classify real-time grid job behavior information is proposed in this thesis. As shown previously, the objective function of the classification problem is to find the parameters that provide a solution to:

$$C(\vec{x}) = W^T\phi(\vec{x}) + b, \tag{4.17}$$

CNNs deliver a solution for equation (4.17) by processing the input $\vec{x}$ throughout a series of convolutional filters and simple non-linear functions [12]. A CNN has a hierarchical architecture. Starting from the input data $\vec{x}$, each following layer output $\vec{x}_j$ is computed as:

$$\vec{x}_j = \phi W_j \vec{x}_{j-1} \tag{4.18}$$

In equation (4.18), W_j is a linear operator and ϕ is a non-linear function. CNNs normally have a stack of convolutional filters W_j and ϕ is commonly a rectifier $\max(x, 0)$ or $Sigmoid$ $1/(1+\exp(-x))$. For W_j the layers are filter maps and each layer can be written as a sum of convolutions of the previous layer:

$$\vec{x}_j(a, h_j) = \phi\Big(\sum_h (\vec{x}_{j-1}(., h) * W_{j,h_j}(., h))(a)\Big), \tag{4.19}$$

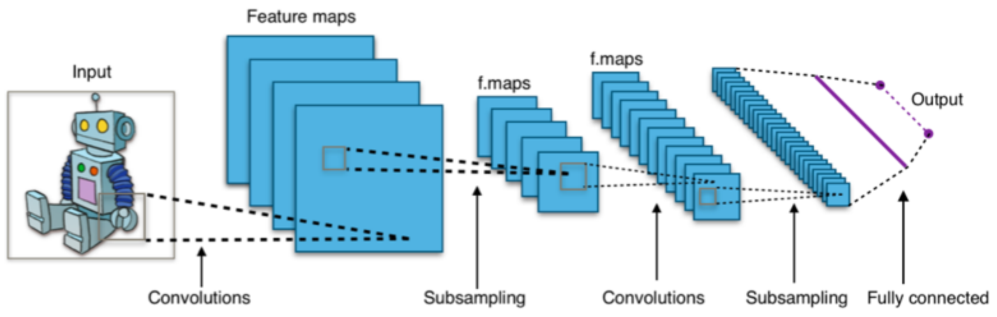


Figure 4.11: A typical convolutional neural network architecture for image classification [194].

where $*$ is the discrete convolution operator, a is the age of measurement and h is the number of filters. A discrete convolution operation in deep learning can be defined as:

$$s(t) = (x * g)(t) = \sum_{a=-\infty}^{\infty} x(a)g(t-a), \tag{4.20}$$

here x is an input measurement on time or index t, and g is a weighting function (also known as the kernel) that depends on the age of measurement a. The optimization problem defined by a convolutional neural network is highly non-convex. Therefore, the parameters or weights W_j are learned by stochastic gradient descent, using the back-propagation algorithm to compute the required weights updates. Figure 4.11 shows a typical CNN architecture utilized for object detection in an image.

In this thesis, the interest is focused on the ability of CNN to classify text data [15]. Figure 4.12 shows an example of a CNN using sliding filters to analyze system calls log lines, where the convolution is applied in the first layer. The subsequent layers in a CNN are normally composed of classical fully connected neurons such as deep neural networks. $Sigmoid$ and rectified linear units (ReLUs) are common activation functions $\phi(x)$ used by CNN.

The classification of the input data, after it has been transformed into embedding word vectors via the word2vec model starts with a convolution operation involving a filter $G \in \mathbb{R}^{hk}$. A window of h tokens with k embedding dimensions, is taken from the input object, in this case l log lines, to produce a new feature. If we represent a token as v_w then a feature z_i is generated from a window of tokens $v_{wi:i+h-1}$ by:

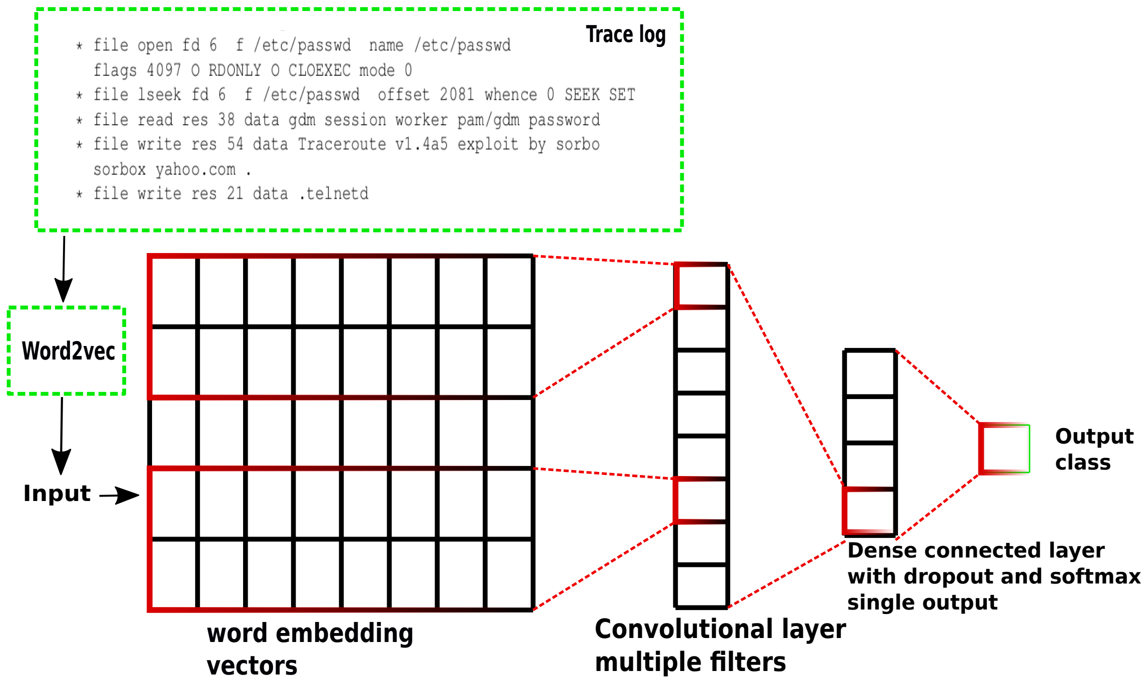
$$z_i = f(G \cdot v_{wi:i+h-1} + b) \tag{4.21}$$

Here $b \in \mathbb{R}$ is a bias term and f is a non-linear function. This filter is applied to each possible window of tokens in the input sequence made of a total of n tokens:

$$\{v_{w1:h}, v_{w2:h+1}, ..., v_{wn-h+1:n}\}, \tag{4.22}$$

to produce a new feature vector:

$$z = [z_1, z_2, ..., z_{n-h+1}], \tag{4.23}$$


Trace log
Word2vec
Input
word embedding vectors
Convolutional layer multiple filters
Dense connected layer with dropout and softmax single output
Output class

```
* file open fd 6  f /etc/passwd  name /etc/passwd
  flags 4097 O RDONLY O CLOEXEC mode 0
* file lseek fd 6  f /etc/passwd  offset 2081 whence 0 SEEK SET
* file read res 38 data gdm session worker pam/gdm password
* file write res 54 data Traceroute v1.4a5 exploit by sorbo
  sorbox yahoo.com .
* file write res 21 data .telnetd
```

Figure 4.12: A representation of the utilized convolutional network architecture: trace data from system calls and network connection summaries are transformed in embedding vectors with word2vec. The vectors are the input for several convolutional filters that extract relevant information. A conventional fully connected neural network layer is then employed to calculate the label of the trace.

with $z \in \mathbb{R}_{n-h+1}$. A max-over-time pooling operation is then calculated over the feature vector, taking the maximum value $z^* = \max z$ as the only feature resulting from this filter. The most important feature, one with the highest value, is kept for each feature map.

The process is repeated with multiple filters of different window sizes, to obtain several features. These features are the result of applying the convolutional calculation in the first layer of the CNN. They are forwarded to a fully connected $ReLu$ layer whose output goes to the last dense layer with $Sigmoid$ activation. Finally, the outputs are the probability distribution over the classification labels, "normal" and "malicious" in the problem setup.

4.5.5 SVM for Grid Job Trace Analysis

The state of the art section about intrusion detection and prevention systems has shown that the support vector machines are one of the most used classifiers in the IDPS area [116]. For feature extraction, the BoW model is traditionally utilized for natural language data or similar [18]. Therefore, SVMs with BoW are applied as the baseline comparison for the proposed classification method with deep neural networks.

The training steps for an SVM classifier are similar to the described for ANNs. The objective function is different though. The SVM training phase builds a model that assigns new samples to one out of two categories via a non-probabilistic binary linear classifier. This process makes SVMs suitable for solving the current intrusion detection problem. SVMs employ hyperplane decision classifiers in a similar way to neural networks. However, the optimization objective is to maximize the margin, defined as the distance between the decision boundary and the training data that are closest to that hyperplane [18]. The classification model parameters W are made of M vectors in $\mathbb{R}^k$ where $W = \{\vec{w}_i\}_{i=1}^{M}$. The objective is to optimize the parameters of W and b such as:

$$C_n(W^T \phi(\vec{x}) + b) \geq 1, n = 1, \ldots, N. \tag{4.24}$$

The optimization problem can be expressed in a more straightforward way as:

$$\underset{W,b}{\arg\min} \frac{1}{2} ||W||^2. \tag{4.25}$$

For BoW, a vocabulary is created with the list of all possible tokens in the training dataset. Then this vocabulary reduced by using only the most frequently appearing tokens. A vector is created with its dimensionality equal to the vocabulary size. Each component of this vector is the number of times a given token appears in the classification object. As introduced in section 4.5.3, tf-idf$_{t,d}$ is a commonly utilized frequency-based calculation for extracting features from a document d. In this project d is built from the set of l trace lines extracted from the system calls and network connection logs.

4.5.6 ANN Optimization

The utilized procedures to train CNNs and SVMs, and how to extract the input features have been described. Another important topic that needs discussion is the structure of the employed networks and their hyper-parameters. The hyper-parameters are values that determine the architecture of an ML model. Values such as the number of layers, the number of neurons per layer or the learning rate are known as hyper-parameters and are related to the specific type of network or training algorithm used. These parameters are relevant in the final performance of an ANN, and they are different according to the context of the specific problem or function to be approximated, and the training data.

The grid search is a straightforward optimization technique for setting optimized hyper-parameters. It is an exhaustive search throughout a manually specified subset of possible best values. Various performance metrics guide this search, typically measured by cross-validation on the training set or evaluation over a validation subset. Since the parameter space of a machine learning algorithm may include real-valued or infinite value spaces for certain parameters, manually setting bounds and discretization may be necessary before applying a grid search. The grid search algorithm finds the settings that achieved the highest score in the validation procedure. It suffers from the curse of dimensionality, but it can be made in parallel given that hyper-parameter settings are evaluated independently of each other. Other more advanced optimization methods such evolutionary algorithms are not used in this thesis but are interesting to explore in the future.

A set of discrete hyper-parameter values has to be defined. With this set, the training process is done with the same subset of the training data. Then, the combination of parameters that provides the best cross-validation accuracy result is selected. The relevant hyper-parameters are defined for the CNN as follows:

- **Learning rate**: this parameter determines the size of the steps taken to update the network weights in every iteration of the learning process. Values close to 1 may lead to a fast convergence but suboptimal values, while values close to 0 could lead to optimal values but at prohibitive speed.

- **Momentum**: it accelerates SGD in the direction that maximizes the weights optimization and moderates possible oscillations.

- **Decay**: the learning rate does not have to be constant. Reducing it in each iteration is a typical network tuning technique that helps to reach an optimal convergence. Decay is the amount the learning rate is reduced in each step.

- **Nesterov**: this is a modification to the momentum that only takes true or false values. It grants a dynamic adjustment of the momentum so an optimal value can be found.

- **Regularizer parameter**: regularizers bring the ability to penalize certain layers in a network to avoid over-fitting and improve generalization.

- **Embedding dimension**: this is the number of components of the embedding vector that represent the corpus tokens. A bigger value of this parameter allows to represent more complex mappings.

- **Filter sizes and the total number of filters**: these parameters define the number of processing steps in the convolutional layer.

- **Hidden neurons**: the number of neurons in the hidden layer that influences the complexity and robustness of the neural model.

- **Dropout rate**: dropout is utilized to prevent over-fitting by randomly setting several inputs to zero at each iteration during training.

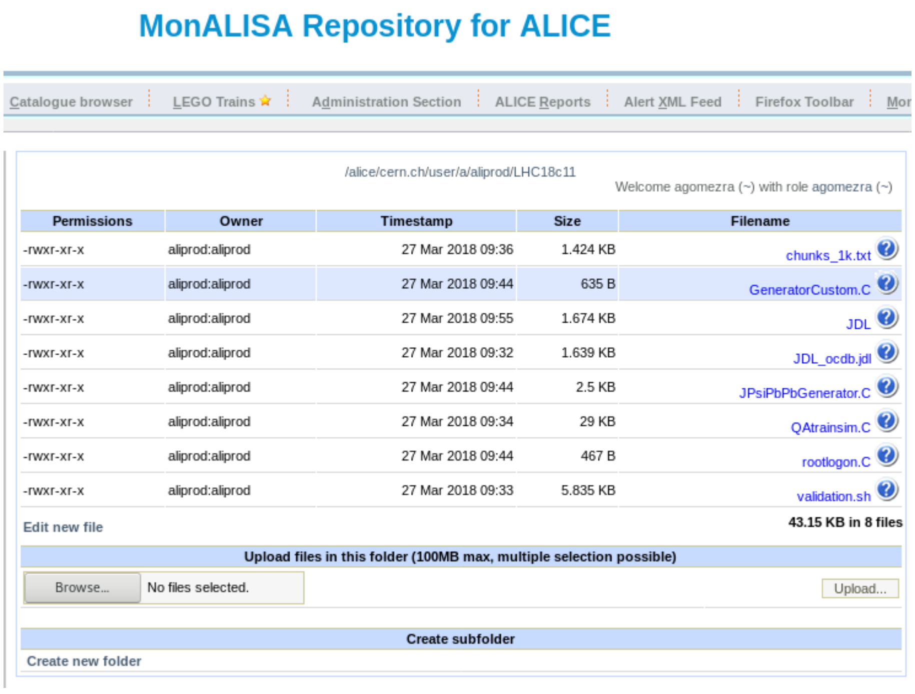
- **m, l, n**: they were described in section 4.5.3. They define the configuration of the object for classification, the set of tokens seem like a snapshot of a set of grid jobs actions regarding system calls and network connections information.

DL methods need to be fed with data to learn from the experience that data can provide. In the next section, the utilized dataset that is used for the CNN and SVM training is described.

4.5.7 Model Validation Dataset

No standard database for the training of IDPS in grid computing was found. A custom one was created for this thesis. The training and validation of the proposed classification and generative algorithms are fed with a dataset composed of regular and malicious system call and network connection logs. To have more realistic samples and to simplify the training process, already existing applications are used to produce the training data.

Regular grid jobs were gathered from the ALICE grid production environment, using the testing grid site, located at the University of Frankfurt am Main. Figures 4.13 and 4.13 show a production job, which can be found in the Monalisa website `http://alimonitor.cern.ch`. That website is the monitoring interface for the ALICE grid. This kind of jobs mainly simulate, reconstruct and analyze the data collected in the particle detectors. They have been manually analyzed and confirmed to be clean of malicious code.



/alice/cern.ch/user/a/aliprod/LHC18c11

Welcome agomezra (~) with role agomezra (~)

Permissions	Owner	Timestamp	Size	Filename
-rwxr-xr-x	aliprod:aliprod	27 Mar 2018 09:36	1.424 KB	chunks_1k.txt
-rwxr-xr-x	aliprod:aliprod	27 Mar 2018 09:44	635 B	GeneratorCustom.C
-rwxr-xr-x	aliprod:aliprod	27 Mar 2018 09:55	1.674 KB	JDL
-rwxr-xr-x	aliprod:aliprod	27 Mar 2018 09:32	1.639 KB	JDL_ocdb.jdl
-rwxr-xr-x	aliprod:aliprod	27 Mar 2018 09:44	2.5 KB	JPsiPbPbGenerator.C
-rwxr-xr-x	aliprod:aliprod	27 Mar 2018 09:34	29 KB	QAtrainsim.C
-rwxr-xr-x	aliprod:aliprod	27 Mar 2018 09:44	467 B	rootlogon.C
-rwxr-xr-x	aliprod:aliprod	27 Mar 2018 09:33	5.835 KB	validation.sh

Edit new file

43.15 KB in 8 files

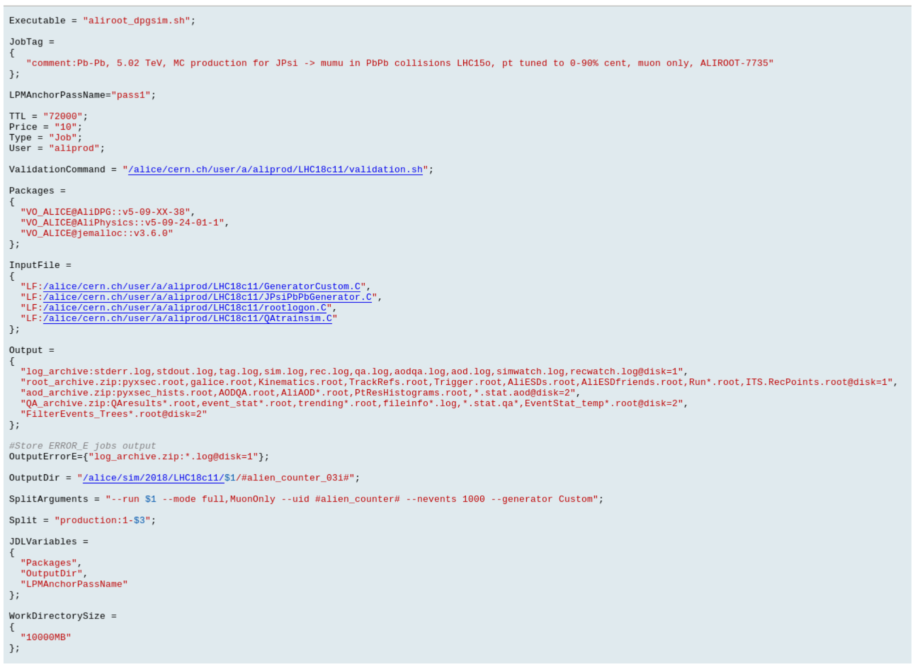
Upload files in this folder (100MB max, multiple selection possible)

Create subfolder

Create new folder

Figure 4.13: A sample of a folder containing all the required files for the execution of a typical ALICE job.

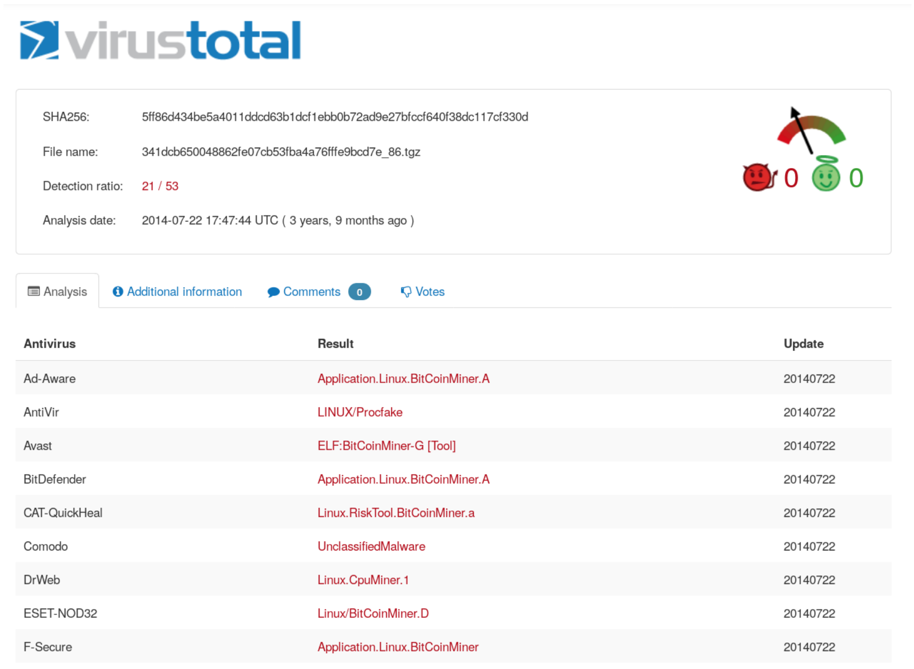
A set of 10,000 Linux malware samples were downloaded from a security research website [185]. This set gives the ability to cover a more significant range of malicious activities that would be very time consuming and error prone to do manually. The Virustotal service [186], a Google-owned tool that grants security researchers the possibility to extract useful information about potentially harmful files was utilized. Such a service was used to have approximated information about the activities that the downloaded samples could perform. For instance, it was possible to find several crypto-coin mining malware binaries there. Samples



```
Executable = "aliroot_dpgsim.sh";

JobTag =
{
   "comment:Pb-Pb, 5.02 TeV, MC production for JPsi -> mumu in PbPb collisions LHC15o, pt tuned to 0-90% cent, muon only, ALIROOT-7735"
};

LPMAnchorPassName="pass1";

TTL = "72000";
Price = "10";
Type = "Job";
User = "aliprod";

ValidationCommand = "/alice/cern.ch/user/a/aliprod/LHC18c11/validation.sh";

Packages =
{
  "VO_ALICE@AliDPG::v5-09-XX-38",
  "VO_ALICE@AliPhysics::v5-09-24-01-1",
  "VO_ALICE@jemalloc::v3.6.0"
};

InputFile =
{
  "LF:/alice/cern.ch/user/a/aliprod/LHC18c11/GeneratorCustom.C",
  "LF:/alice/cern.ch/user/a/aliprod/LHC18c11/JPsiPbPbGenerator.C",
  "LF:/alice/cern.ch/user/a/aliprod/LHC18c11/rootlogon.C",
  "LF:/alice/cern.ch/user/a/aliprod/LHC18c11/QAtrainsim.C"
};

Output =
{
  "log_archive:stderr.log,stdout.log,tag.log,sim.log,rec.log,qa.log,aodqa.log,aod.log,simwatch.log,recwatch.log@disk=1",
  "root_archive.zip:pyxsec.root,galice.root,Kinematics.root,TrackRefs.root,Trigger.root,AliESDs.root,AliESDfriends.root,Run*.root,ITS.RecPoints.root@disk=1",
  "aod_archive.zip:pyxsec_hists.root,AODQA.root,AliAOD*.root,PtResHistograms.root,*.stat.aod@disk=2",
  "QA_archive.zip:QAresults*.root,event_stat*.root,trending*.root,fileinfo*.log,*.stat.qa*,EventStat_temp*.root@disk=2",
  "FilterEvents_Trees*.root@disk=2"
};

#Store ERROR_E jobs output
OutputErrorE={"log_archive.zip:*.log@disk=1"};

OutputDir = "/alice/sim/2018/LHC18c11/$1/#alien_counter_03i#";

SplitArguments = "--run $1 --mode full,MuonOnly --uid #alien_counter# --nevents 1000 --generator Custom";

Split = "production:1-$3";

JDLVariables =
{
  "Packages",
  "OutputDir",
  "LPMAnchorPassName"
};

WorkDirectorySize =
{
  "10000MB"
};
```

Figure 4.14: A sample of an ALICE grid application used in this research.

of the kind of utilized malware are shown in Figures 4.15, 4.16 and 4.17.



SHA256: 5ff86d434be5a4011ddcd63b1dcf1ebb0b72ad9e27bfccf640f38dc117cf330d

File name: 341dcb650048862fe07cb53fba4a76fffe9bcd7e_86.tgz

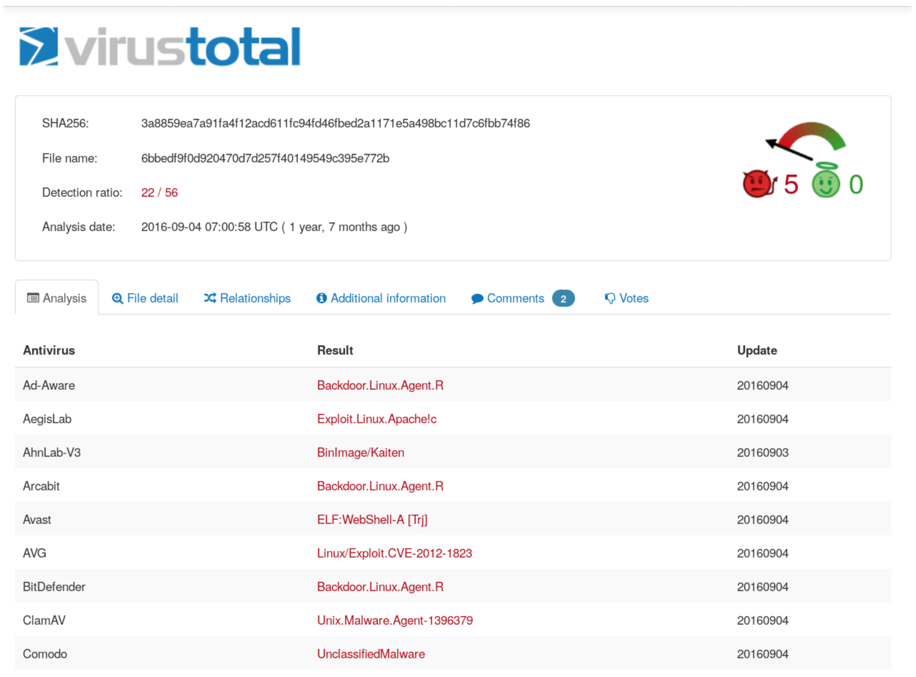
Detection ratio: 21 / 53

Analysis date: 2014-07-22 17:47:44 UTC (3 years, 9 months ago)

Antivirus	Result	Update
Ad-Aware	Application.Linux.BitCoinMiner.A	20140722
AntiVir	LINUX/Procfake	20140722
Avast	ELF:BitCoinMiner-G [Tool]	20140722
BitDefender	Application.Linux.BitCoinMiner.A	20140722
CAT-QuickHeal	Linux.RiskTool.BitCoinMiner.a	20140722
Comodo	UnclassifiedMalware	20140722
DrWeb	Linux.CpuMiner.1	20140722
ESET-NOD32	Linux/BitCoinMiner.D	20140722
F-Secure	Application.Linux.BitCoinMiner	20140722

Figure 4.15: A sample of a Malware binary identified as a Linux BitCoin Miner according to VirusTotal.

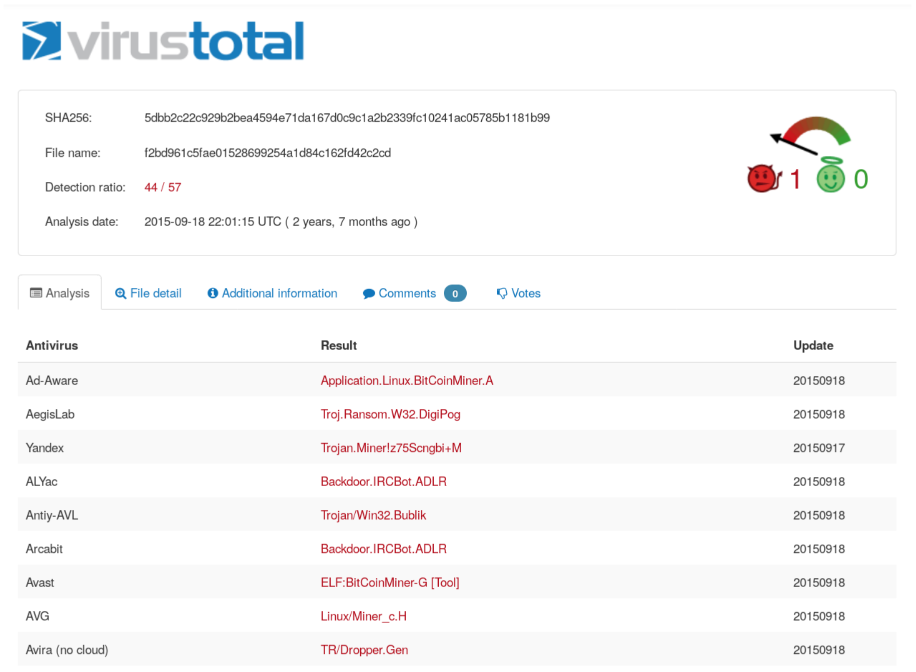
A total of 325 normal and 10,000 malicious payloads were executed inside Linux containers, and the isolation and monitoring features were used to collect



SHA256: 3a8859ea7a91fa4f12acd611fc94fd46fbed2a1171e5a498bc11d7c6fbb74f86
File name: 6bbedf9f0d920470d7d257f40149549c395e772b
Detection ratio: 22 / 56
Analysis date: 2016-09-04 07:00:58 UTC (1 year, 7 months ago)

Antivirus	Result	Update
Ad-Aware	Backdoor.Linux.Agent.R	20160904
AegisLab	Exploit.Linux.Apache!c	20160904
AhnLab-V3	BinImage/Kaiten	20160903
Arcabit	Backdoor.Linux.Agent.R	20160904
Avast	ELF:WebShell-A [Trj]	20160904
AVG	Linux/Exploit.CVE-2012-1823	20160904
BitDefender	Backdoor.Linux.Agent.R	20160904
ClamAV	Unix.Malware.Agent-1396379	20160904
Comodo	UnclassifiedMalware	20160904

Figure 4.16: A Malware sample identified as a Linux-based backdoor by VirusTotal.



SHA256: 5dbb2c22c929b2bea4594e71da167d0c9c1a2b2339fc10241ac05785b1181b99
File name: f2bd961c5fae01528699254a1d84c162fd42c2cd
Detection ratio: 44 / 57
Analysis date: 2015-09-18 22:01:15 UTC (2 years, 7 months ago)

Antivirus	Result	Update
Ad-Aware	Application.Linux.BitCoinMiner.A	20150918
AegisLab	Troj.Ransom.W32.DigiPog	20150918
Yandex	Trojan.Miner!z75Scngbi+M	20150917
ALYac	Backdoor.IRCBot.ADLR	20150918
Antiy-AVL	Trojan/Win32.Bublik	20150918
Arcabit	Backdoor.IRCBot.ADLR	20150918
Avast	ELF:BitCoinMiner-G [Tool]	20150918
AVG	Linux/Miner_c.H	20150918
Avira (no cloud)	TR/Dropper.Gen	20150918

Figure 4.17: A sample of a Malware binary identified as a Ransomware.

every system call and network connection made. Although the number of normal payloads is small compared to the malicious ones, the first generate bigger traces that produce many more data input samples than the later, as shown in section 6.2.

A tool called `sysdig` [182] for collecting system calls and the `Bro` network security monitor [7] for network connection data were utilized. A grid job can be composed of several processes, and each process can have several threads. The collected system calls and network traces were grouped on a thread level. Hence, the individual behavior of threads can be followed, and the correct source of a possible attack can be determined.

A testing environment with limited Internet access was deployed for the malware samples, using `Inetsim` [195] for network connection emulation. The limited connection has two explanations: it is dangerous to execute tens of thousands of malware samples without restrictions, and since the samples are already known, most of their server should not be accessible. The Cuckoo sandbox [196] have also been employed to isolate and monitor these runs. The collected data was then transformed with word2vec to embedding vectors suitable to be processed by the classification algorithms. In section 6.2, more details about the created training dataset are given, with the specific numbers of samples for validation and testing. Following the generative methods that were used to improve the dataset are explored.

4.5.8 Recurrent Neural Networks for Training Data Generation

In section 2.4.5 the recurrent neural networks have been described to show outstanding success for processing sequential data. They are different from other ANNs because their neurons have one or several feedback loops [86]. The feedback loops are recurrent cycles over time or along sequence indexes. The optimization objective, similar to other ANNs, is to minimize the difference between the output and target samples by tuning the weights of the network.

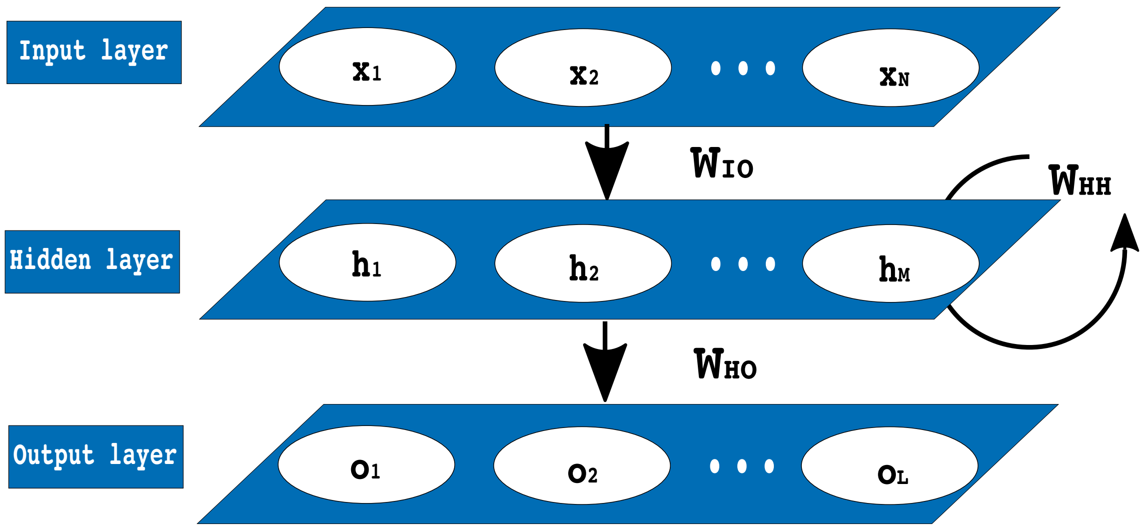


Figure 4.18: A typical structure of a recurrent neural network: it shares several characteristics with the common feedforward neural network. However, RNNs add a feedback loop in the hidden layers.

A simple RNN is composed of three layers: the input, the recurrent hidden layer, and the output. These layers can be visualized in Figure 4.18. The input layer has N input units. The network receives as input a sequence of vectors $\vec{x}_t = (x_1, x_2, ..., x_N)$ that depend on an index t. This index can also be the time of the mesurement of $\vec{x}$. The input neurons are connected to the hidden units in the hidden layer. These connections are represented by a weight matrix W_f. The hidden layer has M hidden units $\vec{h}_t = (h_1, h_2, ..., h_M)$, that are connected to each other with feedback connections. The hidden layer defines the state memory of the system as:

$$\vec{h}_t = f_h(\vec{o}_t), \tag{4.26}$$

in this equation, $\vec{o}_t$ is calculated as follows:

$$\vec{o}_t = W_f\vec{x}_t + W_h\vec{h}_{t-1} + \vec{b}_h, \tag{4.27}$$

where $f_h(\cdot)$ is the activation function for the hidden layer, and $\vec{b}_h$ is the bias vector of the hidden neurons. The hidden units are connected to the output layer with a matrix W_h of weighted connections. The output layer has L units $\vec{o}_t = (o_1, o_2, ..., o_L)$. An RNN uses a simple nonlinear activation function in every unit, and it is capable of modeling rich dynamics. It is assumed that the input vectors are sequential, so the steps described are repeated n times for $t = (1, ..., n)$.

Long short-term memory (LSTM) networks are a recently proposed version of recurrent neural networks useful for long interrelated sequences of data [88]. LSTM was chosen in this research for the generation of textual-like data. This model was employed for the improvement of the training data in the classification of grid jobs. LSTM networks have a specific memory cell and can capture long-term dependences in sequential data. They can be defined with the following set of equations:

$$\begin{aligned}
\vec{f}_t &= \sigma_g(W_f\vec{x}_t + U_f\vec{h}_{t-1} + \vec{b}_f),\\
\vec{i}_t &= \sigma_g(W_i\vec{x}_t + U_i\vec{h}_{t-1} + \vec{b}_i),\\
\vec{o}_t &= \sigma_g(W_o\vec{x}_t + U_o\vec{h}_{t-1} + \vec{b}_o),\\
\vec{c}_t &= \vec{f}_t \circ \vec{c}_{t-1} + \vec{i}_t \circ \sigma_c(W_c\vec{x}_t + U_c\vec{h}_{t-1} + \vec{b}_c),\\
\vec{h}_t &= o_t \circ \sigma_h(c_t).
\end{aligned} \tag{4.28}$$

Similarly to the common RNN, $\vec{x}_t$ is the input vector at a given iteration t, $\vec{h}_t$ is an output vector of the hidden layer and c_t is a cell state. In this case, W and U are parameter matrices and $\vec{b}$ are bias vectors. $\vec{f}_t$ is a forget gate vector, $\vec{i}_t$ is the input gate vector and $\vec{o}_t$ is the output gate vector. The operator $\circ$ is the entrywise product of matrices.

LSTM can be utilized to learn the probability distribution of some input data, and then be used to generate similar data extracted from the learned distribution. That functionality is leveraged to produce data that complements the training dataset. In the proposed architecture, this approach permits to obtain better results in the training of the classification algorithms.

A character level language model powered by LSTM has been utilized as a generative method in Arhuaco. The objective of this model is to predict the next character in a sequence based on the training data. A training corpus $(char_1, ..., char_T)$

is defined, where $char_i$ is a single character and T is the total number of characters in the training dataset, hence all the characters that form the collected job's system calls and network traces.

In the proposed model, the LSTM is utilized to calculate the sequence of output vectors $(\vec{o}_1, ..., \vec{o}_T)$ by a sequence of distributions $P(char_{t+1}|char \leq t) = \sigma(\vec{o}_t)$. This fact means the probability of the next character is modeled in a sequence given all the previous characters. In the end, new sequences of characters can be simulated to create new classification objects (the sequence of a system call or network connection tokens) which enhance the coverage of the training process. Here σ is the $softmax$ distribution defined by:

$$P(\sigma(o_t) = j) = \frac{\exp(o_t^j)}{\sum_k \exp(o_t^k)}, \tag{4.29}$$

where j and k are the dimensions of the output vector $\vec{o}_t$.

The objective function is to maximize the total $\log$ probability of the training sequence $\sum_{t=0}^{T-1} \log P(char_{t+1}|char \leq t)$. From the conditional distribution $P(char_{t+1}|char \leq t)$ it is possible to sample and get the next character in a generated string, and provide it again as the next input to the LSTM [16] for a predefined number of steps or iterations. After the training process has finished, new data can be generated that can be used as extra training data to extend the generalization capabilities of the classification system.

4.6 Summary

In this chapter, an architecture to increase the level of security in grid computing has been described. Linux containers for grid job isolation, integrated with behavior monitoring, were introduced. A threat model was defined to support the design choices for a solution to improve the accuracy of intrusion detection and prevention in the grid. Arhuaco is the proof-of-concept implementation of the proposed architecture to enhance grid security. It manages the execution of payloads in grid sites via Docker Swarm. It can be naturally integrated into the security operation center model proposed for WLCG.

Deep learning methods are utilized to analyze real-time monitoring data of running grid payloads. The job's system call and network traces are utilized as the source of information. The selected deep learning methods form a hybrid supervised classifier. The word2vec model is utilized for feature selection. Convolutional neural networks are utilized for tagging of jobs between the regular and malicious classes. A novel dataset for the training of ML-based IDPS in grid computing was collected. It is composed of regular jobs and Linux malware samples. A recurrent neural network for simulation and generation of unseen training data that complements the collected dataset has been created. It permits to generate simulated system call and network traces. An SVM classifier with the bag-of-words inputs is utilized as the baseline method to compare the CNN effectiveness.

Chapter 5

Prototype Implementation

The proof-of-concept implementation of an architecture for grid security improvement is described in this chapter. An explanation about the testing environment for performing evaluations on the implementation is also given.

5.1 Arhuaco Modules

The architecture proposed in this research was implemented in several modules. They were developed using the Python language, with its version 3.4 [197]. The primary reason for the selection of such language was the wide availability of open source data science libraries and the bindings with other programming languages. This implementation provides interfaces to grid middlewares, container engines and data collection tools developed in other languages. Further, the Arhuaco building blocks are described in detail. As shown in section 2.3, the modules are inspired by the general architecture of intrusion detection and prevention systems, but they also include a job execution interface.

Figure 5.1 shows the class diagram used for the implementation of Arhuaco. It explores the relationships between the modules. The input data comes from the source tools `sysdig` and `Bro`, that collect data from the jobs running over the Docker Swarm cluster setup. The collected information is forwarded to the analysis module on each WN or the VoBox for detection of anomalies in the jobs activity. In the training phase, the convolutional neural network interacts with the recurrent neural network (RNN) to enhance the dataset coverage. In online mode, the analysis module sends data to the storage or communicates with the response module in case of a potential intrusion. Finally, the response module executes predefined actions such as alerting the administrators by e-mail or killing suspicious jobs.

5.1.1 Execution Engine

Submission and execution of jobs in a distributed environment such as a grid site cluster is the goal of the execution module. Arhuaco provides an initial proof-of-concept implementation with an interface for Docker which receives jobs from a

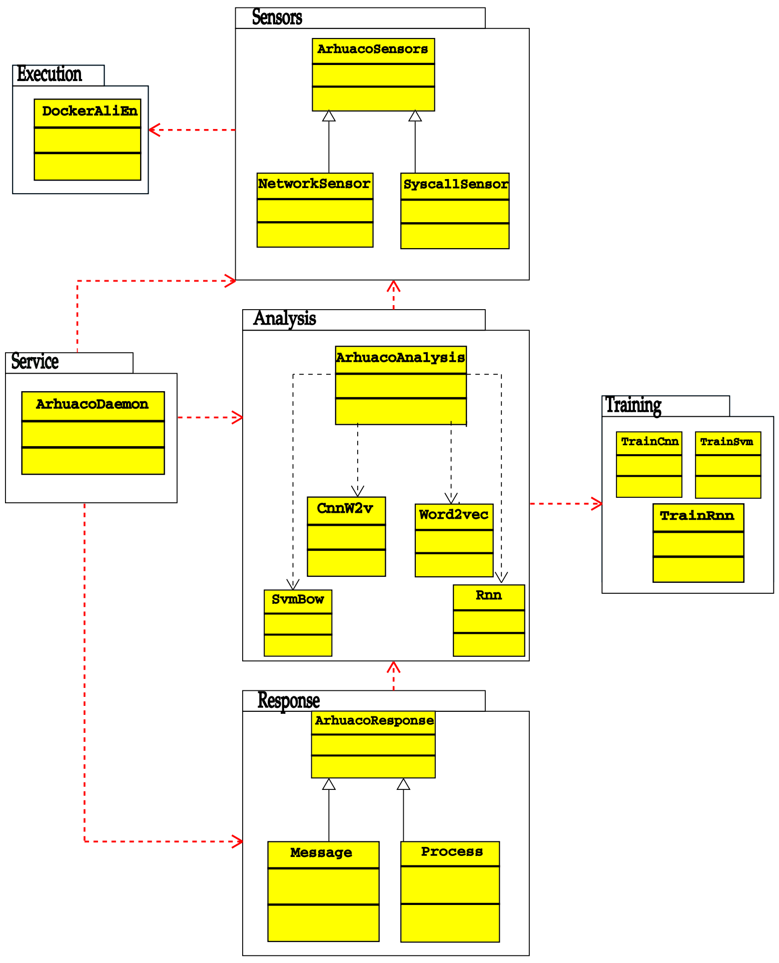


Figure 5.1: The modules and classes in Arhuaco are distributed in independent components that can be reusable.

central grid service and schedules them in a cluster. Docker Swarm was chosen as the first supported distributed container engine for the execution of jobs. Its simplicity and fast deployment in testing environments were the main criteria for the selection.

The proof-of-concept implementation of the proposed architecture was integrated into a testing site of the ALICE grid. An interface between a Docker Swarm installation in the test site and the AliEn [39] middleware was developed. This interface enabled Arhuaco to receive job execution requests from the ALICE central services. The interface made it possible to process the requests and to run the payloads inside Linux containers in one of the worker nodes (WNs). Figure 5.2 shows the flow of information starting from the grid central services, throughout the execution engine, up to the actual job execution. The jobs' operation can be started as described in section 2.1.3 via the AliEn grid user interfaces or by the central service scheduling predefined jobs. The information collected about the payloads behavior was monitored and processed by the machine learning algorithms. Finally, the administrator can verify the security state of the jobs executed in the site, and be informed with an alert if a security incident is detected.

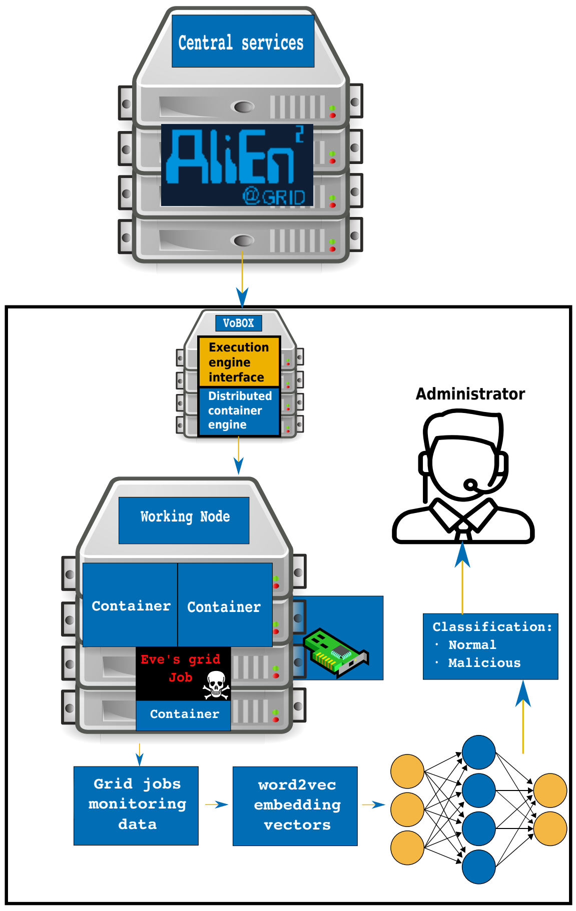


Figure 5.2: A representation of the flow of jobs and the collected information, from the ALICE central services to the Arhuaco modules where the job behavior is classified.

The Arhuaco-AliEn interface was created with a Perl [53] script. Perl is the language utilized for AliEn development and its API. In Listing 5.1, there are some of the most relevant code sections of such created interface.

Isolation Implementation

One component of the execution module is the security by isolation (SbI) confinement of the jobs. One important aspect to consider the execution of grid jobs on a site is how many of them should be scheduled inside. There are several alternatives:

- Running several jobs from different users inside the same container would make it lose the traceability and isolation properties since that would be a similar scenario to one without any isolation. Malicious jobs could still tamper with another user's jobs to hide the source of an attack.

```
# subroutine to define the specific batch system commands
sub initialize() {
    my $self = shift;

    $self->{PATH} = $self->{CONFIG}->{LOG_DIR};
    $self->{X509}=AliEn::X509->new();

    $self->debug(1,"In DOCKER.pm initialize");

    # https://docs.docker.com/engine/reference/commandline/service_create/
    $self->{SUBMIT_CMD} =  "docker service create";

    # https://docs.docker.com/engine/reference/commandline/service_rm/
    $self->{KILL_CMD} = "docker service rm";

    # https://docs.docker.com/engine/reference/commandline/service_ls/
    $self->{STATUS_CMD} = "docker service ls";

    $self->{GET_QUEUE_STATUS}="$self->{STATUS_CMD}";
    if ( $self->{CONFIG}->{CE_STATUSARG} ) {
        $self->{GET_QUEUE_STATUS}.=" @{$self->{CONFIG}->{CE_STATUSARG_LIST}}"
    }

    $self->debug(1,"DOCKER intialize finished");
    return 1;
}

my $docker_submit = " --restart-condition none --restart-max-attempts 0 ".
" --name alien-$containerID ".
" --mount
type=bind,source='/var/lib/aliprod/.alien/tmp/agent.startup.$jobAgentID'".
",target='/var/lib/aliprod/.alien/tmp/agent.startup.$jobAgentID' ".
" --mount type=bind,source='/cvmfs/alice.cern.ch'".
",target='/cvmfs/alice.cern.ch'".
" --workdir '/var/lib/aliprod/.alien/tmp' ".
" --env ALIEN_CM_AS_LDAP_PROXY=$cm ".
" --env ALIEN_JOBAGENT_ID=$$.$self->{COUNTER} ".
" --env ALIEN_ALICE_CM_AS_LDAP_PROXY=$cm ".
" --network ufnet ".
" test:alien $command ";
```

Listing 5.1: Source of the Perl based AliEn interface with Docker Swarm.

- Executing several jobs from the same user may be another possibility. However, it makes it harder to know what each of the different processes is doing on each WN. The traceability of individual jobs would be more difficult under this scenario.

- A single job per container is the best option to increase traceability and isolation properties. Therefore, this is the selected option. Grid payloads are wrapped inside Linux containers in such a way that one job per container is executed. It is the natural microservice model for LCs, where a single application runs in a unique container.

The sites in the ALICE grid have a head node or VoBox where several services are hosted (See section 2.1.3). These AliEn-middleware-based services grant access to a central file catalog and a distributed set of storage systems provided by the collaborating computing centers. The site services permit the connection to the job layer that serves a central task queue with a workload management system. One of those services in the VoBox, the job agent, pulls newly available jobs from the task queue. When there are new jobs, a pilot job is created and executed in one of the available working nodes. This pilot job further runs the actual grid job payloads. This model is initially supported for the proof-of-concept implementation for the ALICE grid. However, instead of directly executing the pilot job inside a WN, it is first wrapped in a CentOS 6 [198] Docker container image. The corresponding grid job is launched inside the container. As decided, in this research only one job is executed per container which means that every pilot job executes only one grid job.

The Docker engine limits privilege capabilities of a container before its execution. Therefore, processes inside of containers have reduced privileges. A daemon called `dockerd` takes care of the administration. It listens for commands over the network or via the regular command line interface provided by the Docker binaries on the local machine. The daemon has root privileges on the host machine.

A custom container image was developed for Arhuaco, according to the ALICE collaboration software guidelines [198]. A Dockerfile specification was created, which inherits from a standard CentOS 6 base. This Linux distribution is the one that is usually utilized in the ALICE-related computing systems. The Dockerfile has commands to install all HEP dependencies required to execute ALICE jobs. It also creates all the configuration files, working directories, and copies all the external dependencies to the compiled binary image. In Listing 5.2 the Dockerfile is shown.

When software is executed inside an isolated environment, a question is how to give that software access to the libraries it needs to carry out its tasks. The Docker engine has a mechanism to share directories between the host machine and guest containers. This mechanism is useful, for example, to grant containers with read-only access to folders with data or libraries. This property was leveraged, so the utilized containers could access HEP libraries via CERN-VM-fs (See section 4.3). Listing 5.3 shows the command to start an LC with the directory `/cvmfs/` on the host mapped to the same directory in the guest.

The network communication inside the grid sites is another issue to consider. Even if jobs are executed isolated from the underline host system, they still may have access to sensitive network segments. They could compromise critical components in the organization where the code is executed. Even external organizations that are reachable via the Internet could be affected. As described in section 2.2, network virtualization is used via native virtual extensible LAN (VXLAN) which are Linux kernel features to create overlay networks. Docker supports this feature. The Arhuaco implementation of this network architecture is shown in Figure 4.7.

Other Tested Container Engines and Security Measures

In the master thesis in [175], other container engines and a kernel hardening measure were tested that could improve Arhuaco for future developments. Here, some significant details of that related study are described.

The first tested alternative container engine was rkt [106]. A tool called `acbuild` gives users the ability to create and manipulate container images. One of the options is to specify a Docker container that rkt can import to its native image format. The application container image (ACI) and the open container initiative (OCI) are the currently supported image formats. The ACI format was used in the described study. The rkt-image was created from the AliEn Docker image implemented in this research with the command `docker2aci`. This command receives a Docker image path as the input parameter and transforms such an image to an ACI, as illustrated in Listing 5.4.

Similarly to Docker, rkt enables the users to configure privilege capabilities to a pod before it is started. Instances of such privilege protections are blocking the access to TCP/IP ports below 1,000, avoiding that a process can change its capabilities and call chroot functions, among others. A detailed list of the default capabilities can be seen in [199]. Other required capabilities can be set or revoked by adjusting the parameters or by setting different values at the configuration files `capabilities-retain-set` and the `capabilities-remove-set`.

Singularity [174] is the other LC engine that was evaluated. Just like Docker, Singularity delivers tools to create, modify, run and manage container images. An LC can be created via the compilation of a script, similar to the Dockerfile. Another option is to import and transform Docker images to the Singularity-specific format. It can be integrated into existing HPC batch systems such as SLURM [200], SGE [201] or Condor. Singularity allows developers, just as Docker and rkt, to have shared directories with the host. Listing 5.5 shows an example command to start a container with five concurrent instances.

Docker's container safety can additionally be enforced with SELinux, AppArmor, or other Linux policy tools. In [175], the related research, an isolation test with Grsecurity patches was carried out. The setup was done in the same grid test site utilized for this study. The implementation of these measures involved placing the Linux kernel source code and the Grsecurity patch files in the same directory for compilation purposes. Grsecurity and PaX were applied following the command instructions in Listing 5.6. After the kernel compilation ends, the resulting Debian packages are written inside the parent directory. They can be installed through the usual Linux distribution package manager.

The purpose of the explored isolation measures was to make a comparison with Docker, the selected container engine of this research. A performance overhead evaluation is prepared as described in the evaluation chapter 6. The results allowed the compararison of the performance impact and the convenience of the different lightweight isolation options. In the next section, the architecture components that enable gathering information about the behavior of running sandboxed applications is described.

5.1.2 Sensors

The sensors module of Arhuaco collects security monitoring data from different sources related to the state of the grid jobs that are running in the WNs. Two separate information sources are considered, the payload system call trace and the network activity summary. The sensors module forwards the information to the analysis module that searches for intrusion patterns.

As said in section 4.4, the tool `sysdig` [182] was applied for collecting system calls form the jobs inside the LCs. `Sysdig` is an open source system monitoring and debugging tool that captures kernel events via call hooks. It has a command line interface to configure a set of filters that give great flexibility regarding what kind of information is observed in the system. This tool is locally installed in each of the WNs. It is a more powerful and robust alternative to the traditional `strace` command in the Linux OS distributions. Listing 5.7 shows the actual CLI command used to capture the containers system calls. In the training phase, Arhuaco collects testing grid job information and store it in log files. This functionality allows users to gather custom datasets and carry out forensic analysis to improve the ML models. In online mode, i.e., when Arhuaco is detecting intrusions in production, the system call flow is captured from the `sysdig` standard output. More optimal ways of doing this task will be explored in the future.

The Bro network security monitor [7] is another open source tool, useful for network traffic analysis. The same developers of tcpdump [202] created it. Its goal is to give visibility of the real-time flow of events happening in a system network. It also brings the ability to analyze offline information. It has traditional network intrusion detection features. However, only its analysis characteristics have been utilized to collect monitoring data. For testing and research purposes all the traffic produced by the grid jobs is captured with `tcpdump`. Then the Bro offline features are used to extract the relevant input data. A sample script used to retrieve this relevant information is written in Listing 5.8.

In Arhuaco's online mode, Bro captures the relevant real-time traffic from the containers. It can be installed on a single machine with access to the virtual overlay network the grid jobs are using to communicate. Thanks to Docker, it is possible to monitor the virtual overlay network deployed on top of the physical network that allows the communication among the running containers, and with the Internet or other authorized networks.

5.1.3 Analysis Engine

The analysis module is the implementation of the proposed deep learning algorithms: the convolutional neural network (CNN) for classification, the support vector machine (SVM) for baseline comparison, word2vec and bag-of-words (BoW) for feature extraction and the recurrent neural networks for data modeling. This module receives input data from the sensors. It searches throughout the collected information to expose suspicious security incidents.

Arhuaco's analysis module has two workflows. The first is the training phase. In this phase, the RNN is applied to enhance the input dataset that improves the resulting accuracy of the ML models such as the CNN or the compared SVM. The stochastic gradient descent learning algorithm is utilized to optimize the weights of the models. The word2vec model is built based on the tokens available in the training corpus. The second workflow is the online detection phase, where the word2vec and the CNN classifier use the learned parameters to detect malicious activities coming from the grid jobs in real-time. In the next sections, a description of the implementation details of the deep learning (DL) algorithms is given.

Deep Learning Methods Implementation

The CNN, SVM and long short-term memory (LSTM) networks have been implemented via the Python library Keras version 2.0 [203] with Theano [204] and TensorFlow [205] as backends, both are supported. Keras is a free and open source library that simplifies the development of DL algorithms and enables parallel training and testing of models supported by the backend libraries. Keras is convenient for high-throughput computing (HTC) applications since it can be used in shared resource environments in parallel with other software. It is sharply focused on GPU usage to increase the parallel processing performance. A small section of the CNN implementation code is shown in Listing 5.9; there, the Keras API is used to create the utilized network structure. Several filters are applied in the first convolutional layer ($Conv1D$). The token vectors are the input for this first layer. The $MaxPooling1D$ function is employed to extract the resulting features for the following layers.

The extraction of features via the word2vec algorithm was done with another open source Python library, called Gensim [206]. Gensim gives a convenient API for the retrieval of embedding vectors as well as many functions for the processing of natural language information. It enables big data processing methods to deal with information that does not fit in RAM. These methods are necessary for the big size of the training corpus. The bag-of-words model was built upon the sklearn [207] library.

Hyper-Parameters Optimization

The implemented algorithms require their hyper-parameters to be tuned, so the obtained result in the classification is optimized. An empiric grid search was utilized for this tuning task (See section 4.5.6). Some examples of parameters that need to be optimized are momentum and decay. They are configured to ensure the models to converge. Another example is the dropout, a method to randomly block specific input features in each step of the training to prevent over-fitting. The Keras implementation was combined with sklearn [207] that provides a convenient interface for the grid search, to apply a systematic search of those parameters. Keras has bindings to call models from the sklearn code. Through this, the models can be evaluated under different values of the parameters until the ones that produce the best measurements of accuracy are found. Table 5.1 lists the set of candidate parameters that are explored for the CNN via the grid search, they were empirically preselected. The goal of the grid search is to examine the obtained training results for different models that have the listed parameters until all the possible combinations are evaluated.

The support vector machine was configured with standard default values provided by the Keras deep learning library. The $Hinge$ loss function and the $Adadelta$ optimizer [208] were utilized. The loss is the distance between the expected output (samples label) and the actual calculated output. The $Hinge$ loss is defined as:

$$\ell(f(\vec{x}), \vec{y}) = \max(0, 1 - \vec{y} \cdot f(\vec{x})), \tag{5.1}$$

Parameter	Candidate values
Learning rate	0.001, 0.01, 0.1
Momentum	0.0, 0.2, 0.4, 0.6, 0.8, 0.9
Decay	0.0, 1e-5, 1e-6, 1e-7
Nesterov	True, False
Regularizer parameter	0.1, 0.01, 0.001
Embedding dimension	5, 10, 20, 30
Filter sizes	(1, 2, 3, 4), (3, 4, 5), (5, 6)
Total number filters	5, 10, 20, 30
Hidden neurons	5, 10, 20, 30
Dropout rate	0.0, 0.5, 0.1, 0.01
m	4, 5, 7, 10
l	1, 4, 5, 6, 10

Table 5.1: The list of candidate values used in the grid search, which is the utilized optimization method for finding good hyperparameters.

with $f(\vec{x})$ as the desired output and $\vec{y}$ as the actual obtained output. In the context of this problem, the SVM output $\vec{y}$ is defined as $\vec{y} = \pm 1$, with -1 representing the "normal" label and 1 representing the "malicious" label. The parameters m, l and n, introduced in section 4.5.6 are fundamental for extracting relevant input feature vectors. They are the same for both the CNN and SVM since they represent the same input features extracted independently of the used classification algorithm. These parameters were selected to keep a good balance between the classification accuracy of normal and malicious classes and the ability to detect intrusions in real-time.

The default parameters are also assigned to the LSTM network that are given by the Keras library. The root mean square (RMS) propagation optimizer was chosen. A learning rate of 0.01 and a categorical cross-entropy loss function were applied. After describing the implementation of the DL algorithms, the storage and response modules of the architecture are introduced.

5.1.4 Storage

At the proof-of-concept stage, Arhuaco stores the information about jobs behavior and security incidents in plain text log files. A portion of the data gathered via the sensors are kept in storage in such logs mainly for training purposes as shown in section 6.2. Other parts of the data are stored for an offline forensic analysis, which makes it easier for executing quick tests and a more straightforward interpretation of the available information.

5.1.5 The Response Module

The response module carries out predefined actions following the detection of a suspicious security-related incident coming from the grid jobs running on the site WNs. It corresponds to the prevention concept of the IDPS based architecture. It must avoid further damage to the grid system and support in the traceability of any attack. Initially, the preconfigured actions are composed of two tasks:

- Sending alert messages (via e-mail) in case that a security incident has been detected. Here, one crucial need is to minimize the amount of spam that is sent to the system administrators, hence reducing the number of false positives is critical.
- Depending on the selected user configuration, Arhuaco can kill those jobs that represent a confidently high level of threat.

5.1.6 Distributed Installation

The installation of all the Arhuaco components has been automated via DevOps scripts for Linux-based servers. It includes the modules, the interfaces with other systems and the evaluation setup. The configuration management tool Puppet [209] was used for this task. Puppet is based on a subset of the Ruby programming language. Several custom modules have been developed to automate the installation of all the component needed for Arhuaco and other pieces of the evaluation environment such as a distributed files system for accessing the required libraries (described in the next section) and for the unification of analysis results. Listing 5.10 shows a section of the Puppet code employed for the Arhuaco Python module installation.

Distributed Filesystem

The ALICE grid jobs need read access to a shared directory, where high energy physics libraries can be found in the Arhuaco testing setup. Hence, the CernVM File System (CernVM-FS) [176] was installed on the worker nodes. CernVM-FS enables library access by mounting a local read-only directory. This directory is shared as a volume inside the AliEn containers to grant access to the required libraries.

An important point is that the jobs running in the WNs generate output files derived from the analysis they make of the scientific data. These files should be transferred to the AliEn central services. The transfer can happen directly via an Internet connection or the VoBox if the nodes are offline. In any case, a way to transfer files from the WNs to the VoBox is required. A network file system (NFS) solution [210] was configured, which allows processes to have a shared directory where they can store their files. Furthermore, this shared location was protected with AppArmor; this granted that if a job managed to escape from the container isolation, the AppArmor rules kept such malicious process away from forbidden directories.

5.2 Evaluation Environment Setup

A testing environment was built to make a set of evaluations on the proposed approaches and provide evidence to support the advertised improvements for grid security. An ALICE grid site was deployed in a Linux cluster at the Goethe University in Frankfurt am Main, Germany. The goal of this setup was to collect real grid job data and to measure the performance and accuracy of the Arhuaco methods.

The testing grid site has five working nodes and a head node or VoBox. The nodes possess the Linux distribution Ubuntu 14.04. All nodes consist of a Supermicro X8DAH mainboard with two Intel Xeon E5520 processors. Each WN processor has in total 8 physical cores realized as 16 virtual cores using hyper threading, dynamically clocked to up to 2.27 GHz. The machines have 12 GB of RAM and a Western Digital WD5002ABYS-0 HDD with a capacity of 500 GB, 7200 RPM and 16 MB cache. The swap area was set up as a file on the root file system, not as a separate partition. The network interface is an Intel 82576 Gigabit network card, capable of transferring 1 Gbps over Ethernet.

Regarding the software, AliEn [39], the ALICE grid middleware was configured and installed in the cluster located in Frankfurt. The required infrastructure to be a member of the ALICE grid was available, and the test site was allowed to join the grid. The administrators have supported the project with access to production jobs, advisory on the infrastructure setup and general scientific feedback. In the following sections, the characteristics of the implemented test are shown.

5.3 Summary

The implementation details of the proposed architecture, the isolation, and security monitoring approach for distributed grid computing were presented in this chapter. A proof-of-concept setup for the ALICE grid has been described. Automatically installable modules for the testing environment were developed. Puppet modules install all the components and the testing setup. In that environment, the performance and accuracy characteristics of the approach to improving grid computing security can be evaluated.

Arhuaco is implemented with the Python language using Keras as the deep learning library and Theano and TensorFlow as the backends for automated parallelism management. The system architecture is similar to the intrusion detection and prevention systems. The building blocks and components were listed. An interface with the AliEn middleware was created with Perl. Currently, the supported container orchestration engine is Docker Swarm. However, further research with optional container engines such as rkt and singularity is also described.

```
# Execute ALICE jobs inside Linux containers
# This script creates all the needed AliEn environtment.

FROM centos:centos6
MAINTAINER Andres Gomez andres.gomez@cern.ch

LABEL container.label=worker-node

# Add cern repos
RUN curl http://linuxsoft.cern.ch/wlcg/wlcg-sl6.repo -o /etc/yum.repos.d/wlcg-sl6.repo && \
        curl http://linuxsoft.cern.ch/wlcg/RPM-GPG-KEY-wlcg -o /tmp/RPM-GPG-KEY-wlcg && \
        rpm --import /tmp/RPM-GPG-KEY-wlcg && \
        /usr/bin/yum --enablerepo=*-testing clean all && \
        rm /tmp/RPM-GPG-KEY-wlcg && \
        rm -rf /var/cache/yum

# Install prerequisites.
RUN yum update -y
    groupadd -g 355 aliprod
    useradd -g 355 -d /var/lib/aliprod -m aliprod
    yum install -y HEP_OSlibs_SL6
    yum install -y which
    yum install -y gcc-gfortran
    yum install -y redhat-lsb-core-4.0-7.el6.centos.x86_64
    sed -i '$ a\export PATH="/cvmfs/alice.cern.ch/bin:$PATH"' /var/lib/aliprod/.bashrc
    sed -i '$ a\export LANG=C' /var/lib/aliprod/.bashrc

# Create the read only configuration files for
# AliEn services and Jobs.
COPY opt/.alien /var/lib/aliprod/.alien

# Create writable directories for the AliEn jobs
RUN mkdir /var/lib/aliprod/.alien/cache
    mkdir /var/lib/aliprod/.alien/logs
    mkdir /var/lib/aliprod/.alien/tmp

# Make aliprod the owner of these directories and files
RUN chown -R aliprod:aliprod /var/lib/aliprod/.alien

ENV HOME="/var/lib/aliprod"
ENV PATH="/cvmfs/alice.cern.ch/bin:$PATH"
ENV LANG=C

USER aliprod
WORKDIR /var/lib/aliprod/.alien/tmp
```

Listing 5.2: Source of the implemented AliEn Dockerfile. The image that results from compiling this source file is utilized to execute and monitor jobs with Arhuaco.

```
#!/bin/bash

docker run --mount type=bind,source=/home/alien/logs \
,target=/home/alien/logs --name alien tests:alien \
/var/lib/aliprod/.alien/tmp/PbPbbench/runtest.sh
```

Listing 5.3: Example of a command to share a host volume with Docker containers.

```
#!/bin/bash

docker export tests:alien > dockerimg.tar
docker2aci ./dockerimg.tar
mkdir workdir
tar zxf dockerimg.aci
mv rootfs manifest workdir
vim workdir /rootfs/var/lib/aliprod/.alien/tmp/PbPbbench/runtest.sh
tar -pczf rkt1.aci workdit/manifest workdir/rootfs/
```

Listing 5.4: Samples of commands utilized to create and modify an ACI container from a Docker image.

```
#!/bin/bash

singularity run -w -B /home/alien/singularity:/runtimes -B \
/cvmfs/alice.cern.ch singularity_1.img 5 2
```

Listing 5.5: A sample of a command to execute a container based on Singularity image.

```
#!/bin/bash

cd linux-4.9.23/
patch - p1 < ../grsecurity-3.1-4.9.23-201704181901.patch
make menuconfig
fakeroot make deb-pkg
```

Listing 5.6: A sample of the commands utilized to apply and compile the Grsecurity and PaX patches to the kernel [175].

```
#!/bin/bash

sysdig -p'%container.id %evt.category %evt.type %evt.args' \
evt.category!= sleep and evt.category!=wait and evt.category!=IPC \
and evt.category!=ipc and evt.category!=scheduler \
and container.name contains alien
```

Listing 5.7: Source of one of the scripts utilized for system call data capture via sysdig.

```
#!/bin/bash

if [ "$1" == "dns" ]; then
    # normal
    cat $(find /home/data/normal/ -name "dns.log" \
    | sed ':a;N;$!ba;s/\n/ /g') \
    |   /opt/bro/bin/bro-cut query duration \
    qclass qclass_name qtype qtype_name \
    |   grep -v "ubuntu\|192\.\|\-\s" > /home/data/network_normal.log
    # malicious
    cat /home/data/malicious/network-sandbox/dns.log \
    |   /opt/bro/bin/bro-cut query duration qclass \
    qclass_name qtype qtype_name
    \ |   grep -v "ubuntu\|192\.\|\-\s" > /home/data/network_malicious.log
fi
```

Listing 5.8: Source of one of the scripts utilized for network information extraction using Bro.

```
# Build the model
# Graph subnet with one input and one output,
# convolutional layers concateneted in parallel
graph_in = Input(shape=(sequence_length,embedding_dim))
convs = []
for fsz in filter_sizes:
    # Conv1D: keras convolutional layer
    # Embedding: it allows to use word vectors as inputs
    conv = Conv1D(activity_regularizer=l2(
                           regularizer_param),
                  padding="valid",
                  strides=1,
                  kernel_regularizer=l2(
                           regularizer_param),
                  filters=num_filters,
                  activation="relu",
                  kernel_size=fsz)(graph_in)
    pool = MaxPooling1D(pool_size=pool_size)(conv)
    flatten = Flatten()(pool)
    convs.append(flatten)
out = None
if len(filter_sizes) > 1:
    out = Concatenate()(convs)
else:
    out = convs[0]
graph = Model(outputs=out, inputs=graph_in)
self.model = Sequential()
self.model.add(Embedding(len(self.vocabulary)+1,
               embedding_dim,
               input_length=sequence_length,
               weights=self.embedding_weights))
```

Listing 5.9: A sample of a relevant section of the CNN implementation with Python. In this part of the code, the relevant input structure of the neural network is created.

```
class profile::arhuaco inherits profile::params {

    # Install arhuaco library
    package { "docker-py":
        ensure  => present,
        provider => pip,
    }
    ~>
    package { "prometheus_client":
        ensure  => present,
        provider => pip,
    }
    ~>
    file {'/var/log/arhuaco':
        ensure => 'directory',
        owner  => 'root',
        group  => 'root',
        mode   => '0644',
    }
    ~>
    exec { 'arhuaco-library':
        command => "cd /tmp && wget
        http://iri03.iri.uni-frankfurt.de/arhuaco-0.5.tar.gz && tar -xvzf
        arhuaco-0.5.tar.gz && cd arhuaco-0.5 && python setup.py install &&
        touch /var/log/arhuaco/.install.done",
        path   => "/bin:/usr/bin:/usr/local/sbin:/usr/local/bin",
        unless => 'test -f /var/log/arhuaco/.install.done',
        provider => 'shell',
        } ~> file { '/etc/init/arhuaco.conf':
        path    => '/etc/init/arhuaco.conf',
        ensure  => present,
        owner   => root,
        group   => root,
        mode    => 0755,
        notify  => Service['arhuaco'],
        content => template("profile/arhuaco.conf.erb"),
    }
    ~>
    service { "arhuaco":
        ensure    => stopped,
        enable    => true,
        pattern   => "arhuaco",
        hasstatus => false,
        provider => 'upstart',
    }
    ~>
    file {'/home/data':
        ensure => 'directory',
        owner  => 'root',
        group  => 'root',
        mode   => '0644',
    }
}
```

Listing 5.10: A section of the Puppet code utilized for the installation of the Arhuaco Python components.

Chapter 6

Evaluation and Results

An empiric evaluation of the approach and the Arhuaco's proof-of-concept implementation is presented in this chapter. Quantitative information about the benefits of the research proposals was gathered. The performance impact caused by sandboxed jobs with security by isolation (SbI) measures and security monitoring features was tested. The classification accuracy and data generation effectiveness of the deep learning algorithms implemented in Arhuaco are compared to validate their contribution. The goal of this chapter is to evaluate the following research questions:

- Linux containers (LCs) are proposed for the isolation of grid jobs and to get specific traceability information about individual job activity. Does the isolation raise the overall security of the grid environment? How do LCs affect the performance of the grid sandboxed applications? Is the measured overhead acceptable if contrasted with the obtained level of security?

- Classification of grid job traces based on deep learning (DL) methods is proposed. Convolutional neural networks (CNN) were the selected DL classifiers. Word2vec model is utilized to extract the input features from the monitoring data collected via LCs. It is stated that this approach produces augmented results over the support vector machines (SVM) with the bag-of-words features, two of the most popular methods used in machine learning (ML) related studies of grid-based IDS. How can the most optimal network structures be built to have the best training results? What are the optimal hyper-parameters for the selected networks? Is the CNN classifier effective enough to distinguish among normal and malicious grid jobs? Does the CNN provide higher classification accuracy and lower false positives rates than the compared SVM?

- A long short-term memory (LSTM) network was created as a generative model for improving the training dataset coverage and enhance the results of the network connection logs classification. Does this data generation improve the obtained training results?

- A dataset for the training of deep learning based grid intrusion detection was collected. It is composed of inputs and labels extracted from production

ALICE grid jobs and Linux malware binaries. Is the collected dataset an appropriate benchmark to validate ML-based models for grid intrusion detection?

In the following sections, verifiable answers to these questions are delivered. A set of evaluations carried out in the testing setup at the Frankfurt grid site is presented.

6.1 Performance Evaluation Setup

As described in the previous chapters, this project aims to increase the safety of grid jobs by isolation and to monitor their behavior. However, just as virtual machines, Linux containers may cause a performance overhead in the execution of the scientific payloads. This overhead takes place because isolation measures commonly affect the execution of the OS processes at different levels. For instance, VMs frequently analyze and interpret machine-level binary code instructions while containers intercept and validate system calls and kernel functions. These isolation features cause the execution of additional portions of code or more context switching which produces an overall increased process runtime if compared with the non-isolation mode. A setup to empirically measure the performance impact of the isolation and monitoring enhancements was created. The results of such measurements grant useful information to determine if the extra security is worth to consider or that the impact is prohibitive.

A standard ALICE application, `PbPbbench` [211] version v5-05-Rev-21, extracted from the CernVM-FS volume, was utilized in the evaluations to measure the runtime overhead caused over regular grid jobs; it was selected in order to have a "realistic" example of the type of applications running in the grid. The library is located in the same mounted directory as the other HEP libraries, exposed in a path on the WNs: "/cvmfs/alice.cern.ch/". The benchmark application executes a Monte Carlo simulation of particle collisions. The simulation environment and boundary conditions are stored in an OCDB directory located in the PbPbbench folder. This simulation calculates the generation of particles that are formed during a collision, their energy deposition, and path throughout the detector [6]. This application was employed since it has all the typical characteristics of the WLCG grid jobs. An example of the kind of script utilized to start the PbPbbench evaluation inside Docker containers is shown in Listing 6.1. In addition, the Linpack [4] benchmark library was utilized to measure the execution throughput shortcoming of isolated jobs. It leverages a dense system of linear equations for testing the processing abilities of high-performance computing systems.

Docker has been tested as the main container engine and isolation provider, but the results of tests made with other isolation mechanisms are also described: rkt, singularity and kernel hardening with Grsecurity patches. A combination of them was also analyzed.

```bash
#!/bin/bash

# Set up the dockerfile
DIR=`pwd`

docker build -t alien_perf $DIR

# run the test
mkdir -p results
for  (( j = 0; j < $1; j++ )); do
    for (( i = 0; i < 16; i++ )); do
        log="results/docker-$j-$i.log"
        docker run --cpuset-cpus="$i" --rm \
        --name docker-perf-${i} \
        -h docker-perf-${i} \
        -v /cvmfs/alice.cern.ch:/cvmfs/alice.cern.ch \
        alien_perf /bin/sh ./runtest.sh &>> $log &
    done
    wait
done
```

Listing 6.1: One of the scrips utilized to start a performance test inside Docker with a PbPbbench-based payload.

6.2 Machine Learning Evaluation Setup

A dataset of regular and malicious system call and network connection logs was built for this project. The goal was to have a baseline benchmark for the training and validation of the proposed classification and generative algorithms. Instead of creating a custom set of job samples, a more convenient approach is to utilize already available binaries.

In this project, 325 production ALICE grid jobs were selected from the total of jobs executed in the Frankfurt's site with a success termination state (Without errors that make them stop in the middle of their tasks). Figure 6.1 shows the cumulative number of jobs executed in the grid site called "UF" during this research. Some jobs were not considered because they were repeated or they belong to the predefined analyses created in the ALICE central services instead of being created by the grid users. The runtime of many of the selected payloads can reach several hours. The chosen jobs were monitored and the information collected to build the "normal" tagged portion of the training dataset. The rest was used for development and testing purposes, but they contributed to the production ALICE data processing. The selected ALICE jobs were run inside Docker containers, and the isolation and monitoring features were used to collect every system call and network connection. This data was gathered with the current Arhuaco source module tools, `sysdig` [182] for system calls and `Bro` [7] for network connection summary traces.

A set of 10,000 Linux malware samples downloaded from a security research website [185] was used for this study. This set enables the creation of the malicious part of the training dataset. The samples were executed and monitored in a protected environment. They had no contact with the UF grid site. An open source tool called Inetsim [195] for network connection emulation it is possible to run the malicious binaries just like they were connected to the Internet. This tool is

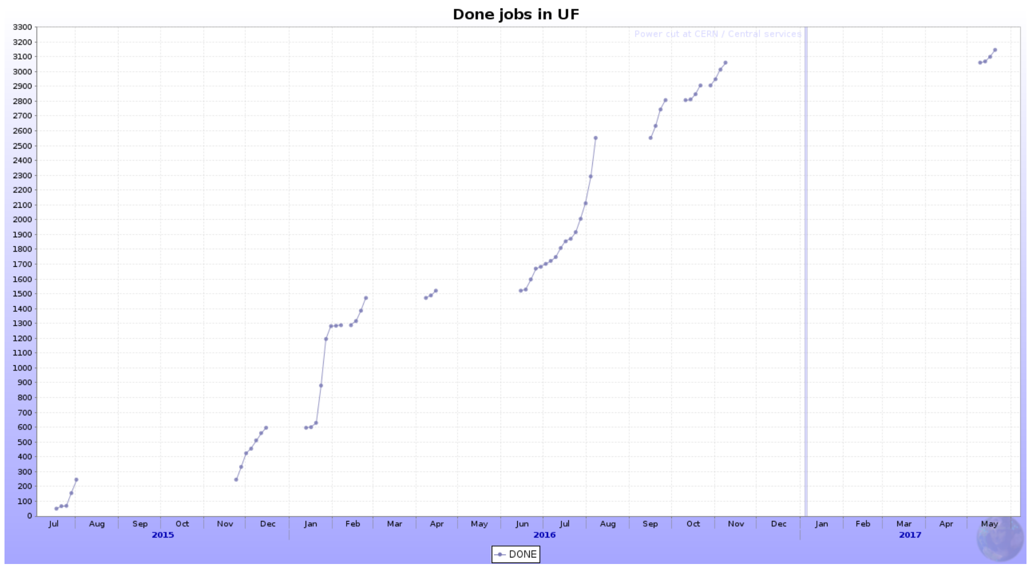


Figure 6.1: A cumulative number of jobs run without an error state on the grid site in the period 2015-2017.

Dataset	Normal	Malware
System call	12 GB - 127’100,000 lines	8.2 GB - 127’054,763 lines
Network	868 KB - 20,733 lines	108 KB - 2,937 lines

Table 6.1: The complete set of information describing the analyzed grid jobs and malware behavior, as collected log-lines.

beneficial to collect more information than in an off-line environment, taking away the risk of affecting other organization networks. The Cuckoo sandbox [196] was applied for isolation and monitoring of the malware runs. This tool written in Python gives a set of library functions that simplify the sandboxing and analysis of malicious applications; it simplifies the management of virtual machines, where the samples were kept for increased safety.

The collected raw input dataset is summarized in Table 6.1. The training dataset is constituted by the log lines that `sysdig` and `Bro` captured from the ALICE jobs and Linux malware in the Frankfurt test environment. As can be seen in the listed raw dataset, more data was collected from the grid jobs than from the malware, although fewer grid samples were selected. This can be explained by the fact that grid jobs can run for hours while the malware binaries do not run normally more than few minutes. In addition, the network data is unbalanced between the normal and the malware dataset, the number of network connection traces utilized in the training phase had to be restricted according to the reduced available malware network data.

Table 6.2 lists the tagged samples resulting from the feature extraction step with word2vec when processing the input corpus. The number of samples depends

Dataset	**Training**	**Validation**
System calls traces	10’000,000	100,000
Network traces	20,000	2,000

Table 6.2: Training and validation samples obtained after the feature extraction method.

on the selected number of lines l and the numberof tokens m chosen to build the input object, i.e., a set of lines from the network and system calls logs. This dataset has been split as follows: 80% of the data was used for training, 10% for validation and a remaining 10% of entirely unseen data was available to test the final ability of the deep learning algorithms to generalize.

Some lines in the training dataset may be found in both categories, “normal” and “malicious”. For instance, there might be system calls sequences that are inside both types of sets. An example of this kind of sequences is the one composed by multiple “signal rt sigaction” that have been frequently found in the runs. These repeated sequences are filtered to further improve the learning ability of the classifiers. Since that data is available in both classes, they do not increase the available information. Hence, their removal is a way to improve the training phase and mainly to reduce false positives.

6.3 Isolation and Monitoring Performance Impact

The performance test scope was confined to the measurement of the throughput and average execution time of grid jobs under several configurations. This evaluation made it possible to determine the impact of the SbI methods and to answer the research questions related to the performance impact in the presence of increased security measures.

6.3.1 Evaluation Measurement Metrics

Two measurement metrics were defined to enable the performance comparison. The average runtime that an ALICE grid job takes to finish its execution, is the first metric, expressed in seconds. The benchmark application *PbPbbench* [211], from the ALICE software analysis framework was chosen to simulate the normal execution of a HEP job. The execution throughput, as described in [2] with the Linpack [4] benchmark library, is the next considered metric. Linpack is a library that employs a dense system of linear equations with an algorithm implementing LU (lower–upper) factorization [5] with partial pivoting for testing the processing abilities of high-performance computing (HPC) systems. The measurement of the evaluation metrics considers three scenarios for each metric:

- *Native*: jobs running over physical working nodes (WN) with a Linux distribution, under common grid configurations. This condition is the baseline,

Setup	ALICE job average runtime (Seconds)	Standard deviation
Native	110.77	10.03
Docker	114.22 (3.12%)	12.58
Arhuaco	117.54 (6.11%)	11.83

Table 6.3: Results of the performance overhead related to the runtime of the ALICE-based jobs.

normal scenario without any isolation.

- *Docker*: jobs running inside Docker containers over the same Linux OS machines.
- *Arhuaco*: In addition to the Docker containers, system call and network trace interception and processing by Arhuaco on the same described machines was made.

In the performance evaluation, only values related to local processing metrics were measured: throughput and runtime. The performance impact of Arhuaco over the network data transmission was not measured. Arhuaco collects network traces from jobs locally and passively, both by the system calls and by the Bro network security monitor. Therefore, the execution overhead of monitoring and isolation of jobs is the focus and not the data transmission rates.

6.3.2 Performance Results

Physical machines with an Ubuntu 14.04 OS were utilized for the performance test. The setup of the machines is described in section 5.2. A set of jobs based on the ALICE's PbPbbench scripts [211] were executed for the first type of performance test. A total of 1600 jobs were executed. The runtime, from the start to the end, was collected for each job. The total average runtime and standard deviation were calculated in the three described scenarios. Each PbPbbench-based job implemented a simulation, reconstruction, and analysis of the same ALICE HEP data, which results in the same expected deterministic processing. Table 6.3 shows the obtained results. The measurement unit is expressed in seconds. The second column lists the average runtime and the percentage of overhead regarding each scenario. The third column lists the standard deviation. The native label means that the evaluation was implemented over the machines without isolation. The second group of tests involved the usage of Docker over the same machines for the isolation of the job. Finally, a Docker plus Arhuaco monitoring group of tests were made.

Benchmark jobs based on the Linpack library were tested for each of the three desired scenarios, in the second type of performance test. The throughput achieved by these jobs was measured, and the overhead was calculated. The measurement unit is floating operations per second. Table 6.4 lists the results. The average

Setup	**Linpack average (GFLOPS)**	**Standard deviation**
Native	3.78443	0.00771472
Docker	3.77343 (-0.29%)	0.0101847
Arhuaco	3.76172 (-0.6%)	0.00485492

Table 6.4: Results of the performance impact measured concerning throughput of the Linpack-based jobs.

throughput for each scenario is in the second column. The percentage of overhead regarding the native scenario is also shown in the same column. The standard deviation is in the third column. 2 jobs per CPU core were executed, with 16 jobs in parallel and 100 repetitions for a total of 1600 runs. All the used jobs had the same constant inputs and parameters, so the expected execution was deterministic. Both the Linpack test and the ALICE job tests used the CentOS 6 container image described in section 5.1.1. However, for the ALICE job case the base operating system utilized was CentOS 6 as well. The reason for the difference is that the CERN libraries showed a different performance for Ubuntu 14.04 and CentOS 6, which was not the case for the throughput test. A discussion about the results is presented in section 6.3.4.

6.3.3 Results of the Alternative Isolation Methods

Some of the results of the master thesis [175] supervised by the author, related to several alternative isolation mechanisms in addition to Docker are mentioned here. That research utilizes the same experimental setup with the ALICE PbPbbench based jobs, the Ubuntu 14.04 machines, and the CentOS 6 container image. The average runtime of an incremental number of parallel jobs was measured. The number of jobs goes from 1 to 10. Four scenarios were tested: The first with a native Ubuntu OS, the second was the execution of jobs inside Docker, the third utilized rkt and the fourth used singularity. All the scenarios utilized a regular stock Linux kernel version 4.9. In Figure 6.2 the results are listed. For each x axis value, the increased number of parallel is shown. The y axis lists the average runtime calculated from those parallel jobs, in each of the four scenarios. From the figure, a runtime increment trend can be observed, when the number of parallel jobs gets higher. The Docker scenario showed the lowest runtime overhead in most of the measurements, making it the best option when kernel hardening is not utilized.

In another experiment, kernel hardening was added to the isolation scenarios. A Linux kernel version 4.9 was utilized. As described in section 6.1, the Grsecurity patches were added to the kernel and then were compiled. The same setup and the same kind of ALICE jobs were used as in the above-described experiment. Figure 6.3 shows the measured average runtime of an increased number of parallel jobs, for 1 to 10. The graphic illustrates the impact of job sandboxing in addition to security enforcement of the Linux kernel with anti-vulnerability exploiting

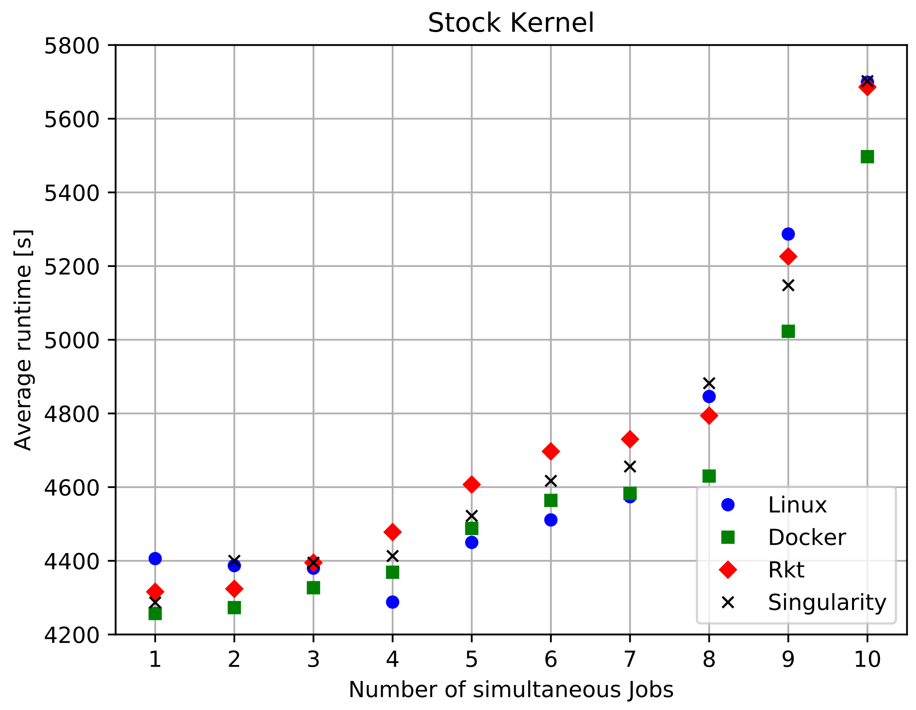


Figure 6.2: A comparison of the average runtime of sandboxed ALICE jobs within several Linux container engines [175].

measures. In this case, we have three scenarios instead of four, because it was not possible to make Docker work with the hardened kernel. For this experiment, rkt showed the best results regarding runtime in most of the measurements.

6.3.4 Discussion

The first research question was defined as: do the isolation measures raise the overall security level of the grid environment? Containerization has been proven as a mature and robust technology for application isolation in high-performance computing [2]. LCs have been tested in other areas such as microservices [169] with significant success. Hence, as far as there are no Kernel or container engine vulnerabilities, Linux containers contribute to increasing the grid security by making it harder for an attacker to compromise the computational infrastructure or sensitive network sections. Although bugs are always a possibility in any computational system, including virtual machines, the security hardening provided by isolation technologies have been utilized in many computing domains, including desktop operating systems [73].

Now, how does the implemented SbI affect the performance of the sandboxed applications? The results in Table 6.4, related to the throughput test with Linpack jobs, show a reduction of 0.29% of the average number of executed floating point operations in the Docker scenario and 0.6% in the Arhuaco scenario. Linpack jobs

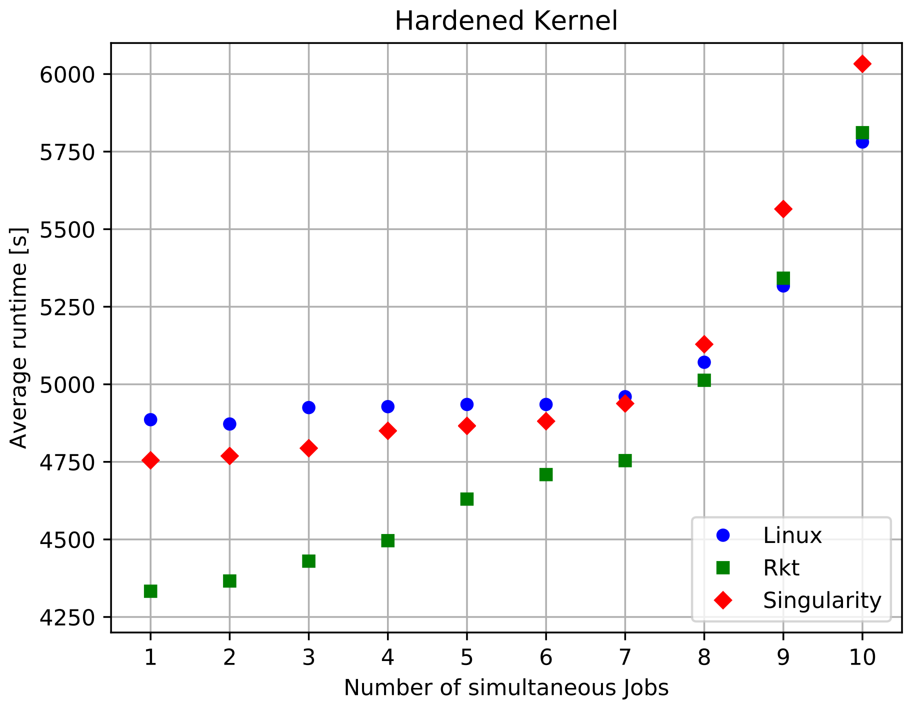


Figure 6.3: A comparison of the average runtime of sandboxed jobs within several Linux container engines on top of a security-enhanced Linux kernel [175].

are mostly composed of floating point operations. Therefore, these jobs do not need a frequent context switching since most of the instructions are arithmetic, so no extensive involvement from the kernel API is expected. The same behavior may be found in many e-science applications, in opposition to, for example, a database solution. This scenario should be even more common for high-throughput computing, i.e., where high-throughput applications are the primary jobs running in an infrastructure. The other type of performance test with the PbPbbench based applications shows an increase in the average runtime in seconds or equivalently a reduction in the processing speed. Table 6.3 lists an average runtime increment of 3.1163% in the Docker scenario and 6.1125% in the Arhuaco scenario. According to these results, when the tested ALICE jobs are isolated with Docker, the execution becomes in average around 3% slower. When isolation and behavior monitoring is provided, the job execution is around 6% slower.

The following research question is, how critical is the measured impact created by the isolation methods given the security provided? The described reductions in the throughput caused by isolation measures are acceptable in grid computing, given the critical requirement of keeping the computing infrastructure safe against malicious jobs. The impact overhead is higher when evaluating the average execution time of PbPbbench jobs. These results can be considered as a very affordable compromise if compared with the consequences of a successful attack in the grid. LCs are increasingly utilized in grid computing collaborations and as the results have shown, adding extra monitoring and analysis by Arhuaco creates an impact that is not severe. Another point to consider is that there are studies such as [2] that compare the usage of LCs against virtual machines (VMs) for scientific applications sandboxing. The studies demonstrate a more significant overhead in the performance VMs generate compared to LCs. Several measures have been implemented in the proof-of-concept to reduce the isolation overhead further. Arhuaco was developed with several configurable options:

- The first possibility is to analyze only a reduced set of random grid jobs. The full set of available production jobs are not considered. This option would be better for preserving performance, but it has the least robust security.
- The second option is to analyze only network summary data in the first place. Then, on any suspicious behavior detected, Arhuaco activates the system call collection.
- The third possibility is the safest mode in which all the data is gathered and analyzed. This last configuration is the default mode that brings the most enhanced safety level.

Regarding the other isolation methods studied in [175], the results can be seen in Figures 6.2 and 6.3. With the regular (stock) Linux kernel, the best average runtimes of sandboxed jobs were obtained with Docker (the green squares). This fact shows that Docker is an appropriate selection as a good performance isolation solution. The evaluations with the hardened kernel showed the best average runtimes with rkt (again the green squares). These results suggest that the usage of rkt and kernel hardening are a promising direction to explore in future developments of Arhuaco. Therefore, for increased security requirements, this looks like a convenient possibility for extra protection against kernel vulnerability exploitation. In some of the tests, the runtime of the jobs without isolation was worse than the jobs with container isolation. An explanation for this fact was not found in the related study.

6.4 Supervised Classification Results

The evaluations made to the classification metrics of the machine learning algorithms for grid job are explained in this section. Results of the improvement obtained when using convolutional neural networks instead of support vector machines is given. The goal here was to show how recently proposed deep learning methods as CNN with word2vec features outperform traditional machine learning methods such as SVMs with the bag-of-words inputs for the task of intrusion detection in grid computing.

Parameters	**System calls**	**Network connections**
Learning rate	0.001	0.01
Momentum	0.8	0.9
Decay	1e-5	1e-5
Nesterov	False	True
Regularizer parameter	0.001	0.001
Embedding dimension	10	10
Filter sizes	(1, 2, 3, 4)	(2, 3)
Total number filters	30	10
Hidden neurons	30	20
Dropout rate	0.0	0.0
m	7	5
l	6	1
n	42	5

Table 6.5: Convolutional neural network parameters selected by a grid search method.

6.4.1 Grid Search Optimization

First, the tuning of the structure of the utilized CNN was explored. The question, defined in the research objectives, is how an optimized network structure can be built to have improved training and evaluation results? As described in section 4.5.6, deep neural network hyper-parameters are the values that determine the architecture of a network and hence the decision of which parameters to use affects the final result. Grid search, a method where several preselected parameters and network structures are tried, generally in parallel, was employed in this study. Therefore, the grid search is the used method to find an optimized network structure.

The past answer takes to the next question: which are the best parameters for the selected network based on the grid search? Doing an exhaustive search with all the possible parameters would be computationally prohibitive. The grid search made it possible to find the best set of hyper-parameters for a reduced subset of the search space. The full list of finally optimized hyperparameters is shown in Table 6.5 with the optimal values. Since the system call and the network traces are two different input context, the resulting parameters differ in some cases.

For the CNN, momentum and parameter decay were used to ensure the model convergence, and dropout to prevent overfitting. However, it was found that the dropout parameter was not needed, with an obtained optimal value of 0.0 for it. The $Hinge$ loss function, the $Adadelta$ optimizer and a set of standard hyper-parameters provided by the Keras library for the support vector machine were chosen. The parameters m, l and n that define the input format for both types of models were described in section 4.5.3. They were also found in the grid search. We proceed to the next section with the description of the relevant evaluation measurement metrics.

6.4.2 Classification Evaluation Metrics

In machine learning, popular performance metrics are those that measure the number of input samples with a correctly calculated output. Accuracy (ACC), sensitivity or true positive rate (TPR), specificity (SPC) and false positive rate (FPR) are common statistical measures of the performance of binary classifiers. That is the type of classifiers utilized here, so those metrics are relevant to this project. In the context of this study, sensitivity indicates how precise a classifier is to predict the "positive" category (it is also known as true positive rate) and specificity describes how the negative class is predicted (it is also known as true negative rate).

Accuracy measures the algorithm's ability to predict the correct output categories or classes. Therefore, the overall performance of the trained classifier is often evaluated by this metric. Accuracy measures the number of instances that were correctly classified - which are both the true positives (TP) and true negatives (TN) - divided by the entire size of a training subset - which is the sum of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN)-. The TPR, SPC and ACC values are calculated as follows:

$$Sensitivity(TPR) = \frac{TP}{TP + FN}, \tag{6.1}$$

$$Specificity(SPC) = \frac{TN}{TN + FP}, \tag{6.2}$$

$$Accuracy(ACC) = \frac{TP + TN}{TP + TN + FP + FN}, \tag{6.3}$$

False positive rate is defined as the number of instances of the "negative" category classified in the "positiveöne, divided by the number of all instances that are classified into first category. FPR is calculated by the equation (6.4). The TPR, SPC, ACC and FPR values are defined in the range $[0, 1]$. They can also be interpreted as percentage values in the range $[0\%, 100\%]$. For TPR, SPC and ACC, the best possible value is 1, while for FPR the best value is 0.

$$FalsePositiveRate(FPR) = \frac{FP}{FP + TN}, \tag{6.4}$$

Another favorite evaluation metric in ML is the k-fold cross-validation [212]. It consists on splitting the training dataset into k subsets, from which $k - 1$ of those sets are used for training and 1 of them for validation. This procedure is repeated k times, choosing a different subset in every step. In the end, the average accuracy of the k validations are calculated. However, this method is not commonly applied in deep learning, given the significant size of the involved datasets and the cost of optimizing ordinarily huge models. Hence, this metric was provided. Besides, enough data was available to run multiple epochs with different samples on each one of them, which makes the cross-validation unnecessary.

Testing dataset	**TPR**	**SPC**	**FPR**	**ACC**
System call	0.9972	0.9932	0.0068	0.9952
Network traces	0.9765	0.9997	0.0006	0.9875

Table 6.6: Results of the classification test of the convolutional neural network, using new input samples extracted from the system calls and network traces.

Testing dataset	**TPR**	**SPC**	**FPR**	**ACC**
System call	0.9965	0.9313	0.0687	0.9639
Network traces	0.9507	0.6219	0.3781	0.7871

Table 6.7: Classification test results of the support vector machine, using new input samples extracted from the system calls and network traces.

Additionally, Keras enabled to make a validation step on each epoch, which enabled the verification that the networks exhibited similar behavior with different data and it can generalize.

6.4.3 Classification Results

The dataset described in section 6.2 was utilized for the optimization of the CNN and SVM network weights via stochastic gradient descent and Adadelta training algorithms respectively. Ten epochs were run for each training process with 100,000 feature inputs extracted from the system calls and 10,000 from the network samples in each epoch. Once the training phase ended, an evaluation of the CNN classifier with entirely new testing data was made. The obtained measurement metrics of that test are shown Table 6.6. The test data was composed of 10,000 new samples extracted from system calls and 1,000 extracted from network traces. Both sets were randomly selected from the full available data and are different from the training sets.

The same procedure was followed for the SVM with the same data subsets for training, validation, and testing. The testing results were collected after the training process; they are listed in Table 6.7.

A comparison of the results of both CNN and SVM algorithms is given in Table 6.8. The table shows in the second and third columns the accuracy metric values. In the fourth and fifth columns, the false positive rate values are listed. As shown in the second and fourth columns inside the parenthesis, the CNN exhibited a higher ACC value (3.24%) and a lower FPR (−90.10%) than the SVM for the system call data. The same trend was observed with the network trace data (25.46% and −99.84%). When we take a look of Tables 6.6 and 6.7, we can see that the TPR and SPC values are also higher in the CNN than in the SVM. These measurements indicate the increased effectivenesses of using the proposed convolutional neural networks with word2vec features for classification

Testing dataset	CNN ACC	SVM ACC	CNN FPR	SVM FPR
System call	0.9952 (3.24%)	0.9639	0.0068 (−90.10%)	0.0687
Network traces	0.9875 (25.46%)	0.7871	0.0006 (−99.84%)	0.3781

Table 6.8: Comparison of the evaluation metrics between CNN vs. SVM for new testing samples extracted from the system calls and network traces.

of grid jobs. The results promote the system as a more powerful approach for the detection of intrusions in the grid environment. Better measurement metric values were obtained than with the compared support vector machine and the bag-of-words features, which is one of the most popular methods in ML-based intrusion detection systems.

Further information to support these results is presented. The training and validation curves that both alternative classifiers have produced are shown. The resulting curves composed by values from the convolutional neural network accuracy measurements in 10 epochs with training and validation input features of system calls can be seen in Figure 6.4. From the graphic we can remark that the CNN validation curve is close to the training curve, approaching 100%, which suggest that proper training was conducted.

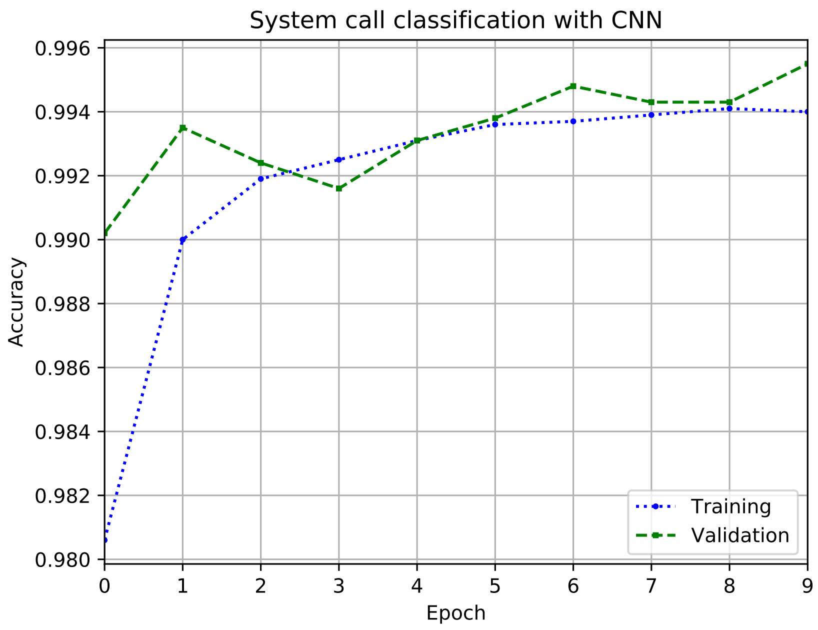


Figure 6.4: A plot of the curves traced from the CNN training and validation Accuracy, in the classification of input features from the system calls.

Figure 6.5 describes the accuracy behavior of the compared SVM concerning the embedding vectors of system calls for training and validation. A trend of over-fitting can be seen in the learning validation curve compared to the training curve. This behavior can be deduced from the fact that the validation curve is higher than the training curve in the first epochs, but it decreases in the following epochs. The SVM model is far smaller than the CNN which may explain the observed over-fitting.

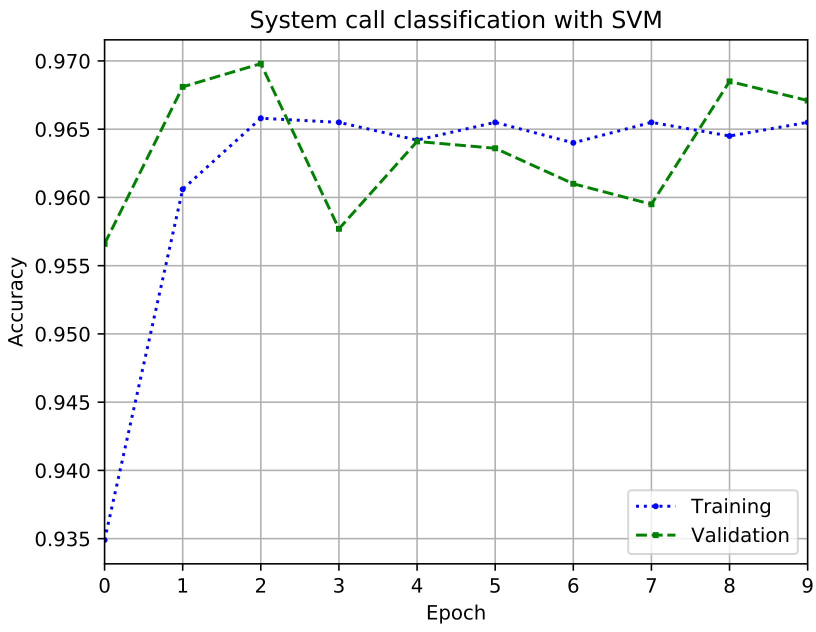


Figure 6.5: A plot of the Accuracy obtained in the several epochs of training and validation of the SVM, applied to input features extracted from the system calls.

To show a comparison among the proposed and baseline method with ACC and FPR measurement metrics, the graphics in Figure 6.6 and Figure 6.7 are provided. They show the curves of learning with feature vectors extracted from the system call data. The CNN exhibited a higher accuracy and decreased false positive rate than the SVM during every epoch of the training phase. The minimum ACC value for the CNN was 0.99 in the first epoch while the maximum ACC value for the SVM was 0.97 in the second epoch. These results indicate a better general behavior of the proposed technique.

Similar results were found for the network connection data. The measured metrics of the network trace classification with CNN and SVM methods are shown in Figure 6.8 and Figure 6.9. The convolutional neural network exhibited results with the training and validation data that are close to the system call case, approaching 99%. The SVM suffered over-fitting here too, and the accuracy did not even reach 80%.

The comparison curves for the two classifiers trained on the network summaries is also given. The accuracy in validation curve displays a better result for the CNN (Figure 6.10). An identical conclusion can be observed for the false positive rate measurement for validation data in Figure 6.11 where also CNN beats

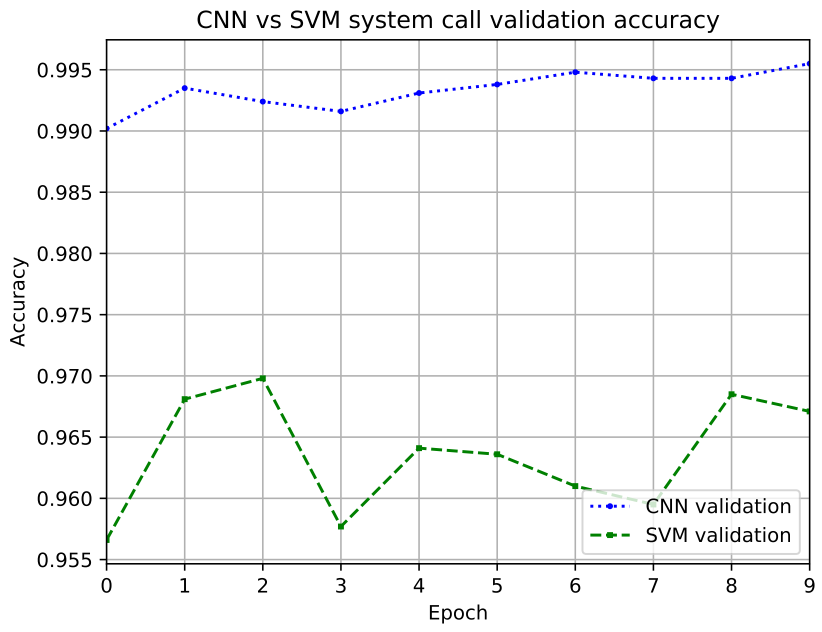


Figure 6.6: A plot comparing the curves of the accuracy of CNN vs SVM, with validation data, applied to system call embedding vectors.

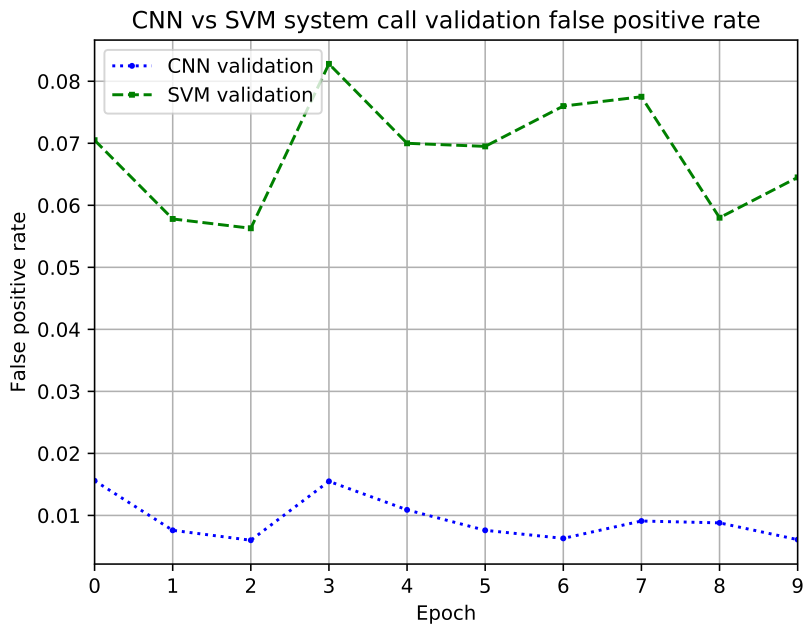


Figure 6.7: A comparison of the traces of false positive rate, of CNN vs. SVM, with validation data of system call embedding vectors.

the SVM.

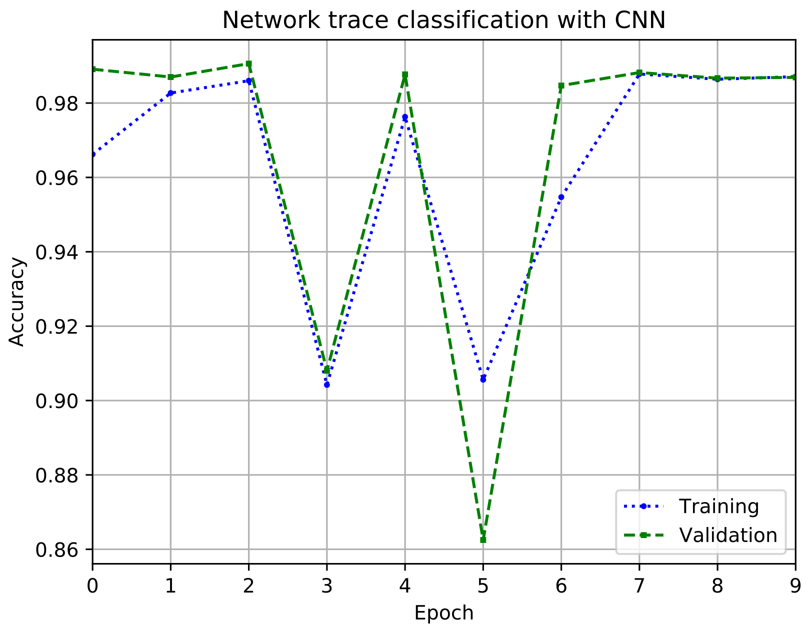


Figure 6.8: Plotting of the accuracy on training and validation during several epochs, of a CNN using network data.

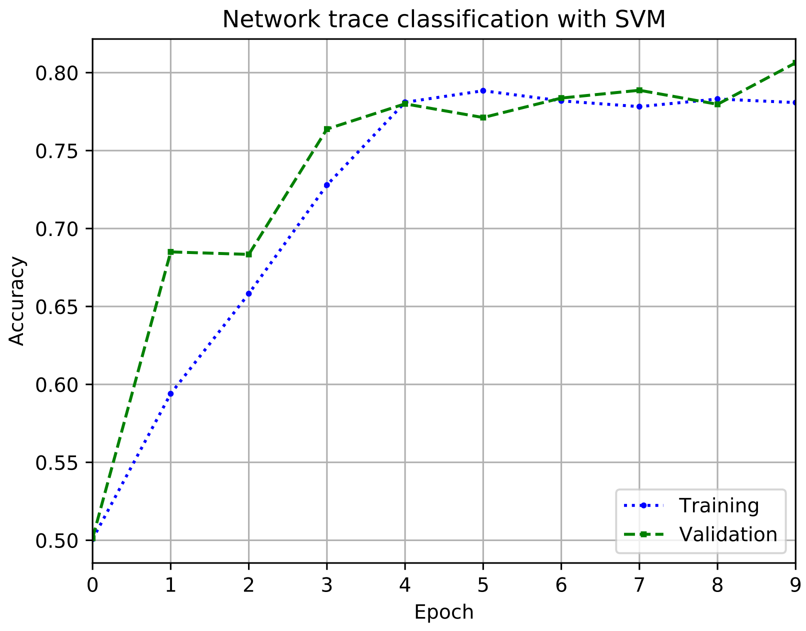


Figure 6.9: A plot of the accuracy obtained in training and validation, for an SVM using network data.

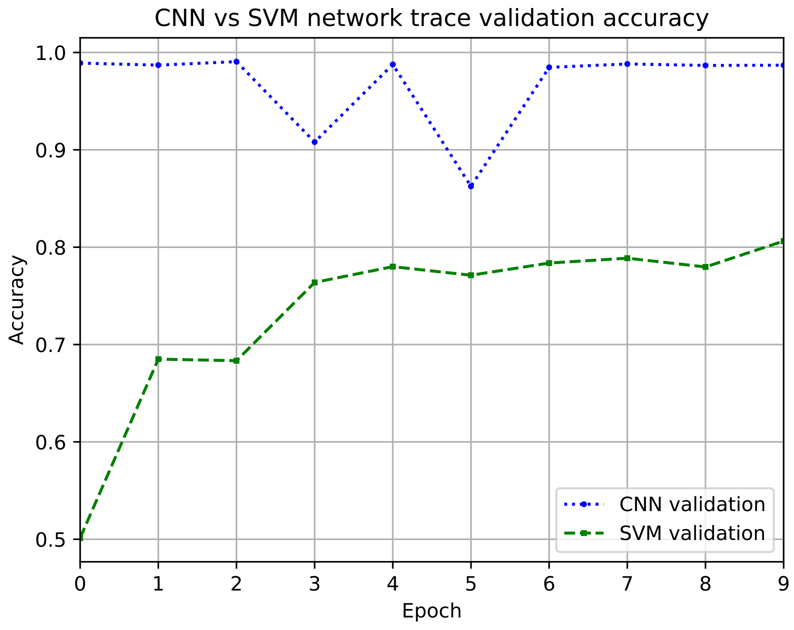


Figure 6.10: A comparison of the ACC plots, with validation data applied to CNN vs. SVM, using network trace embedding vectors.

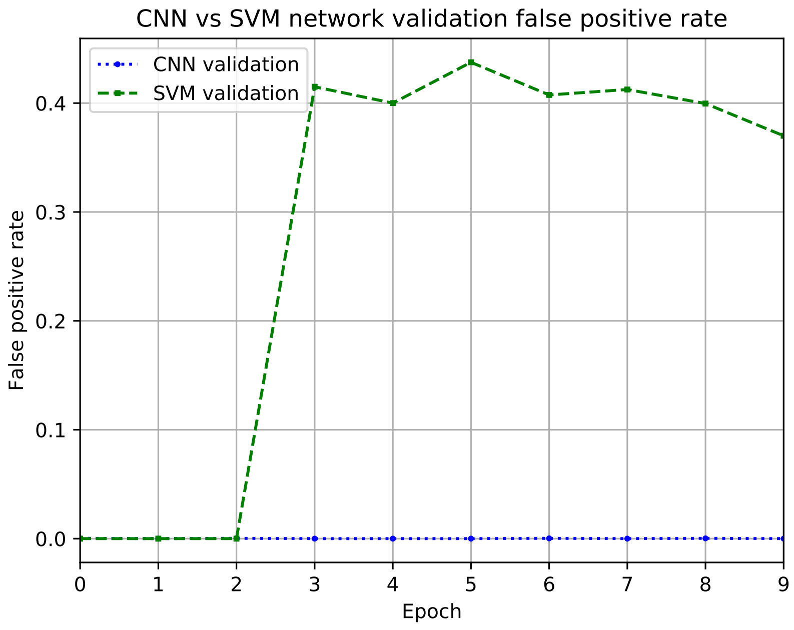


Figure 6.11: A plot comparing the FPR curves produced by the CNN vs. SVM, with validation data from the network trace embedding vectors.

6.4.4 Discussion

Based on the collected facts the next research questions can be answered: Is the CNN classifier useful to distinguish among normal and malicious grid jobs?

Does the CNN provide higher classification accuracy and lower false positives rates than the compared SVM? The results enable to answer positively to both questions. The CNN chosen for the Arhuaco's proof-of-concept increased the accuracy and reduced the false positive rate in classification of grid jobs and was able to generalize to entirely unseen samples in the validation phase. The classification accuracy with both system calls and network traces was close to 100%. Hence, it improves the detection of undesired activity inside the e-science grid that comes from within the jobs. When compared to a traditionally employed method, convolutional neural networks demonstrated a better classification ability, as proven for one of the most popular classifiers, the support vector machines with the bag-of-words as input vectors.

Information was also offered about how the proposed word2vec model for the creation of embedding vectors was able to express the input features accurately based on semantic context preservation. It was a more appropriate preprocessing method for the input data than the bag-of-words model. Another topic to remark is that the natural language processing approach for analyzing input as text-like data can be conveniently extended from system calls and network trace to inputs from other intrusion detection systems, sources of monitoring data or system logs. The Arhuaco's approach could also be adapted beyond high-throughput computing, to monitor cloud services running in containers over orchestration engines such as Kubernetes and Mesos, to mention some examples.

6.5 Generative Model Results

The results of the SVM classification of network connection information had the lowest accuracy among all carried out tests (see Table 6.8). If we observe the accuracy results of the CNN with network data, it approaches 100% right from the first epoch. The reason is the small availability of data with regards to the big size of the model.

The lack of malicious network data presented an opportunity to explore ways to improve the results by artificially augmenting the training dataset. A generative model was utilized via a long short-term memory network. The LSTM was trained with the character corpus of available network traces and then modified to generate 20% of new network trace log lines autonomously. The new data was added to the previously non-generated data to create a new training dataset. Then, the SVM was trained again to measure the further resulting accuracy. The data utilized for validation was the same as the first non-augmented evaluation. Figure 6.12 shows the newly obtained curves of the SVM accuracy in training and validation with the additionally generated data. If we take a look of Figure 6.9, that shows the training and validation accuracy of the SVM for the original network data, we can see how the accuracy increases with the generated data from the first epochs.

Figure 6.13 explores the comparison of the results obtained with the same SVM, evaluated with the non-generated dataset compared to the new enhanced data. Higher accuracy can be seen from the first epochs. Table 6.8 describes the same

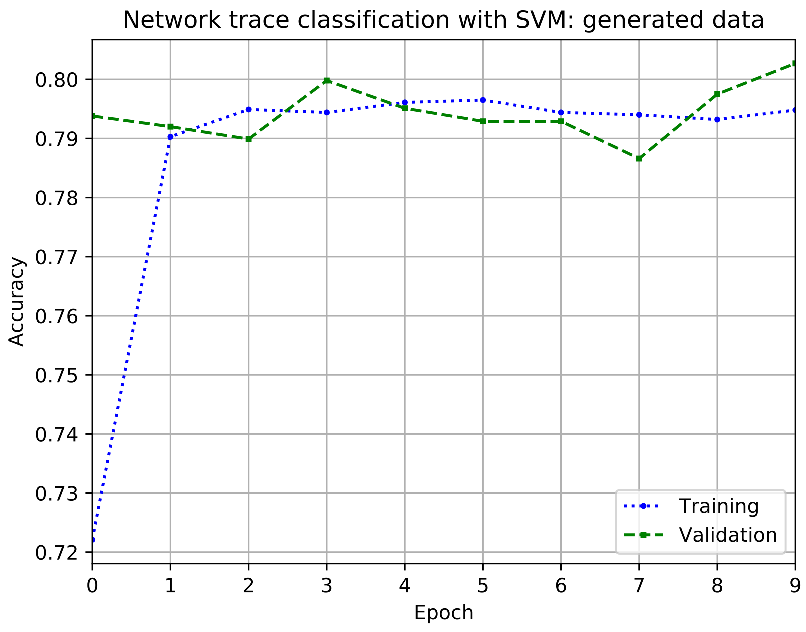


Figure 6.12: A plot showing the curves of the training and validation of classification accuracy, using an SVM with newly generated network samples.

Testing dataset	**TPR**	**SPC**	**FPR**	**ACC**
Network traces normal	0.9507	0.6219	0.3781	0.7871
Network traces generated	0.9712 (2.16%)	0.6206	0.3839	0.7928 (0.72%)

Table 6.9: Resulting accuracy of the SVM tested with previously unseen data. These results compare the training made with the original network samples vs. the new dataset with generated data.

comparison but testing new samples that were not used in the training process. The table shows an increment of 0.72% of the ACC with the generated data in comparison with the original testing data. The TPR was also higher with a 2.16% change rate. Finally, the SPC had instead a reduced, but close value (−0.20%) and the FPR had an increased amount (1.53%).

6.5.1 Discussion

The aim in this section is to answer the LSTM related research question: Does the data generation approach by LSTM improve the obtained training and validation results? As seen in Table 6.9, the obtained ACC of the SVM trained with the addition of generated data, applied to novel unseen network data traces from the original dataset was 0.7928. From this result, the conclusion is that a 0.72% improvement rate was observed if compared to the original value of ACC that

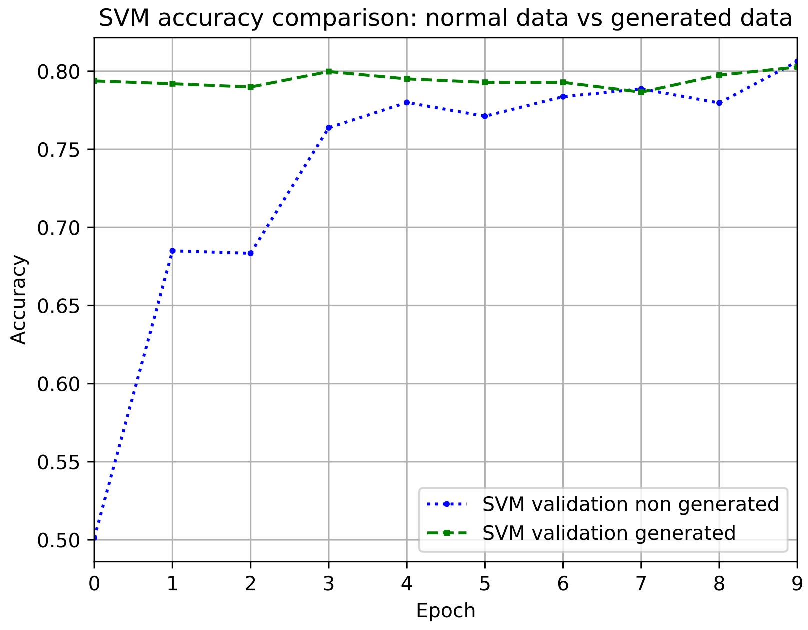


Figure 6.13: A plot comparing the obtained validation accuracy for the collected network dataset vs. the same dataset with additional generated samples.

was 0.7871. The FPR had an increment of 1.53%, which brings the possibility for improvement in future developments. The training and validation accuracy curves in Figure 6.9 and Figure 6.12 allow us to see how the generated dataset makes the SVM to raise the measurements from the first epochs. Higher ACC values are observed in the following epochs. These measures validate the practical benefits of using the LSTM model for modeling and generating data in the context of intrusion detection systems, in this case for grid computing jobs.

Here the last research question we can be answered: Is the collected dataset, formed by regular ALICE grid jobs and Linux malware samples an appropriate source of training of IDS for grid systems? As summarized in Table 6.8, the proposed CNN was able to classify correctly new data samples, with very optimal measurement metric values. This fact demonstrates that the collected dataset, although not entirely complete, is sufficient to afford a good source of training for the context problem, intrusion detection in grid computing, which could be applied to other kinds of grids.

This chapter contributed with results to support the convenience of using the proposed approaches. It was shown how the isolation methods with Linux containers generate moderated performance impact, that can be decreased with several configuration options. Other hardening measures were explored, that could further enforce the isolation security without causing a significant performance pitfall.

The performance tests adopted standard HEP benchmark and throughput applications. These jobs simulate, process and analyze data like real ALICE grid jobs. They help to verify the performance of the proposed isolation measures and the potential impact on other kinds of payloads. The accuracy tests utilize a collected dataset. This dataset is composed of standard ALICE grid job data and Linux malware information from a security research website. This real-world data makes the evaluation simpler and more comprehensive than manually creating custom binaries.

Convolutional neural networks applied to the classification of grid job security behavior had a close to 100% accuracy measurement in training, validation and testing. CNNs offered a higher accuracy, sensitivity, specificity and a lower false positive rate than the compared support vector machines. The proposed word2vec input preprocessing algorithm was shown as an expressive way to represent system log like data, with better results than the alternative bag-of-words. The generative method, the long short-term memory network was tested. The idea that LSTM networks improve and increase simulated information for a training dataset in intrusion detection for grid computing was validated.

Chapter 7

Conclusions and Outlook

Four statements were formulated in the introduction of this thesis (see section 1.2) about intrusion detection, intrusion prevention, and isolation based security for grid computing. The results in chapter 6 provide facts to support these statements. The conclusions drawn about them will be discussed in more detail in the following sections. Below, the statements are listed:

1. The selected sandboxing technique for grid computing, Linux containers (LCs), enable the isolation of user jobs and provides the traceability of individual job activities. The usage of LCs and behavior monitoring does not critically affect the jobs performance.

2. Convolutional neural networks (CNNs) are effective for malware detection in the grid and provide higher classification accuracy and lower false positives rates than support vector machines (SVMs).

3. Long short-term memory (LSTM) networks utilized as generative models are effective for improving the training dataset coverage in grid intrusion detection and prevention systems (IDPS).

4. A benchmark dataset of intrusion and malware classification is relevant in grid computing for model validation and to get more accurate training results.

A software-based proof-of-concept of the proposed ideas, called Arhuaco, was implemented. An empirical set of tests were done to validate the mentioned statements. The tests were performed in a testing ALICE grid site, part of the Worldwide LHC Computing Grid (WLCG).

The selected security by isolation (SbI) method enables the execution of user payloads inside restricted environments. LCs provide a convenient solution to having an increased security level on the grid computational infrastructure, since they create an isolated version of the full system for each grid job. LCs produce a reduced performance overhead compared to virtual machines (VMs) [2], a popular virtualization approach.

Several tests were carried out to measure the impact that the sandboxing mechanism induces in the execution of grid jobs. The effect that the grid job behavior monitoring produces was also evaluated. A test with typical ALICE-based grid jobs was carried out. The goal was to measure the increase in the total average execution time of the payloads. A maximum average runtime impact of 6.1125% was observed when utilizing isolation and behavior monitoring. However, the decrement was not considerable given the level of security improvement provided. The measured reduction in throughput of Linpack-base applications given the additional layers of isolation and extra monitoring was less than 0.6%. The measurements showed that the jobs which had few input-output (IO) interactions had almost no throughput decrement with the usage of containers. The ALICE-based grid jobs had more IO iterations than the Linpack-based jobs, for instance, the former read and write gigabytes of physics experiment data to the disks while the later mostly execute floating point operations with few disk access.

Several configuration measures have been implemented in the proof-of-concept to reduce the harm caused by the overhead when it is not acceptable. The realization of the proposals provides several configuration options to minimize this impact, most importantly in the case of system calls collected for security monitoring purposes.

According to the results shown in Figure 6.2, the use of Docker containers with a stock Linux kernel generated lower performance overheads than with the other Linux container based isolation alternatives, Singularity and rkt. These results give credence to the selection of Docker as the first supported container engine in the architecture. The results in the Figure 6.3, as discussed in section 6.3.4, suggest that the use of rkt and kernel hardening features with Grsecurity are a promising direction to explore in future developments. Thus, for increased security requirements, this looks like a convenient possibility for extra protection against kernel vulnerability exploitation.

CNNs have been proposed in this thesis as an effective method to classify grid jobs and detect malware within them. With word2vec for embedding vector extraction, the CNNs demonstrated a 3.24% higher classification accuracy for system calls and 25.46% for the network trace data than SVMs with the bag-of-words as input vectors, as shown in Table 6.8. SVM is one of the most popular classifiers for ML-based IDPS [17]. The chosen CNN increased the accuracy, whilst reducing the false positive rate, on the classification of the tested grid jobs, and was able to generalize entirely unseen samples in the validation phase. This approach enhances the detection of malicious activity that originates from within the jobs inside the e-science grid. These results revealed that the word2vec model was able to express the input features accurately based on the semantic context preservation. The utilized natural language processing approach for analyzing logs input as text data can be conveniently extended from system calls and network traces to inputs from IDPS, sources of monitoring data or system logs. The architecture could also be adapted, beyond high throughput computing (HTC), to monitor microservices running in containers over orchestration engines such as Kubernetes [109] and Mesos [115].

Benchmark datasets are fundamental to validate machine learning models. No available standard, or even non-standard dataset, was found for intrusion detection model validation in grid computing. Hence, a custom set was built via the collection of job traces, both normal and malicious. With this dataset, the CNN and SVM algorithms were able to be trained and to classify correctly wholly new data samples. This data has been a good source of training for the context problem, intrusion detection in grid computing, however, it may also be useful to other e-science grids.

This research project demonstrated that a generative model utilized for improving the training dataset coverage in IDPS brings increased accuracy results. The collected network data from malware samples was limited in comparison with the availability of system call data. An LSTM network was trained with the available network data and utilized to generate utterly new training samples. The accuracy obtained by the SVM trained via adding the generated data, when applied to novel unseen network data traces from the original dataset was 79.28%, i.e., 0.72% more than with the initial training samples. These results revealed an improvement in the measurement metrics found in the training and validation steps, if compared with the original results. Therefore, these results certified the practical benefits of using LSTM for modeling and generating data in the context of intrusion detection systems, in this case for grid computing jobs.

The proposed architecture and implementation contribute to advance the status of the security in grid computing. This proposal solves the problems created by the arbitrary software execution, the need for bad behavior monitoring and the possibility of attacks generated from the jobs. This architecture allows grid administrators to classify the activity of the jobs, to detect incidents, to keep traceability of the user-generated events, and to configure predefined actions on detected security situations. In addition, more specific and concrete evidence to detect attacks and find their source is collected.

7.1 Outlook

The continuous development of distributed and decentralized systems for scientific projects and industrial applications is increasingly seen every day in society. The security requirements will increase, in parallel with the threats and their sophistication. Attackers using artificial intelligence as a tool for breaching third-party systems will be a growing trend scenario in the near future. Autonomous systems will be required to protect computational systems, for which human administrators could not cope with their size and complexity. From grid computing to blockchain based technologies, solutions similar to Arhuaco will be required to grant a desired level of security.

Several points of this thesis require further development. False positives are always a concern for grid administrators since they consume precious time that could be expended on more critical tasks. Reducing the rates of false positives to a value close to zero is an objective for any IDPS, therefore, more work towards achieving this goal will be continued. Research into better methods to inform about the detected security incidents will also be carried out. More interfaces for other container orchestration engines may be created.

The training process in the presence of adversarial samples will be explored.

The research field known as adversarial machine learning is dedicated for improving ML techniques in adversarial settings, where an attacker could introduce manipulated inputs with the aim of exploiting vulnerabilities of learning algorithms and further compromise the system security. For instance, these adversary samples may affect the training process making the IDPS miss security incidents. Hence, approaches to mitigate such exposure in Arhuaco are required. Another exciting area to review is the usage of generative adversarial networks (GANs) for modeling and create intrusion detection samples that allow researchers to complement training data as an alternative method for RNNs.

Techniques to protect the privacy of the data sets and the ML models such differential privacy and homomorphic encryption will also be studied. In addition, the employment of the distributed nature of the grid to improve the decentralized detection of intrusions will be investigated. This topic can be useful, for instance, to autonomously inform other grid site IDPS about security incidents that could spread in the network. A further enhancing of the isolation in the containers by using kernel hardening, such as Grsecurity, is ongoing. Furthermore, the integration with other container solutions and grid engines beyond ALICE is currently being tested. Finally, optimizations that can be introduced into Arhuaco for reducing the overhead caused by the isolation measures will be examined; this approach seems feasible, given the performance results for the Linpack benchmark.

Bibliography

[1] Anish Damodaran, Anirban Chakrabarti, and Shubhashis Sengupta. Grid Computing Security: A Taxonomy. *IEEE Security & Privacy*, 6:44–51, 2008.

[2] Wes Felter, Re Ferreira, Ram Rajamony, Juan Rubio, Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An Updated Performance Comparison of Virtual Machines and Linux Containers, 2014.

[3] Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the grid. In *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., December 2002.

[4] Piotr Luszczek, Jack J. Dongarra, David Koester, Rolf Rabenseifner, Bob Lucas, Jeremy Kepner, John Mccalpin, David Bailey, and Daisuke Takahashi. Introduction to the HPC Challenge Benchmark Suite. Technical report, 2005.

[5] Ernie Chan, Robert van de Geijn, and Andrew Chapman. Managing the complexity of lookahead for lu factorization with pivoting. In *Proceedings of the Twenty-second Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '10, pages 200–208, New York, NY, USA, 2010. ACM.

[6] ALICE Offline Project. The ALICE Offline Bible, 2017. `http://aliweb.cern.ch/secure/Offline/sites/aliweb.cern.ch.Offline/files/uploads/OfflineBible.pdf`.

[7] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-time. In *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7*, SSYM'98, pages 3–3, Berkeley, CA, USA, 1998. USENIX Association.

[8] Martin Roesch. SNORT - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on System Administration*, LISA '99, pages 229–238, Berkeley, CA, USA, 1999. USENIX Association.

[9] Andrew Hay, Daniel Cid, and Rory Bray. *OSSEC Host-Based Intrusion Detection Guide*. Syngress Publishing, 2008.

[10] Ar Lazarevic, Aysel Ozgur, Levent Ertoz, Jaideep Srivastava, and Vipin Kumar. A comparative study of anomaly detection schemes in network intrusion detection. In *In Proceedings of the Third SIAM International Conference on Data Mining*, 2003.

[11] Richard Zuech, Taghi M Khoshgoftaar, and Randall Wald. Intrusion detection and Big Heterogeneous Data: a Survey. *Journal of Big Data*, 2(1), December 2015.

[12] Jayanth Koushik. Understanding Convolutional Neural Networks. *CoRR*, abs/1605.09081, 2016.

[13] Yoav Goldberg and Omer Levy. Word2Vec Explained: deriving mikolov et al.'s negative-sampling word-embedding method, 2014. cite arxiv:1402.3722.

[14] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A Neural Probabilistic Language Model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.

[15] Yoon Kim. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1746–1751, 2014.

[16] Ilya Sutskever, James Martens, and Geoffrey E. Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, 2011.

[17] Chirag Modi, Dhiren Patel, Bhavesh Borisaniya, Hiren Patel, Avi Patel, and Muttukrishnan Rajarajan. A survey of intrusion detection techniques in Cloud. *Journal of Network and Computer Applications*, 36(1):42–57, January 2013.

[18] Sebastian Raschka. *Python Machine Learning*. Packt Publishing, 2015.

[19] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*, pages 104–121, May 2015.

[20] A. Gomez Ramirez, C. Lara, Udo Kebschull, and ALICE Collaboration. Intrusion Prevention and Detection in Grid Computing - The ALICE Case. *Journal of Physics: Conference Series*, 664(6):062017, 2015.

[21] A. Gomez Ramirez, M. Martinez Pedreira, C. Grigoras, L. Betev, C. Lara, U. Kebschull, and ALICE Collaboration. A Security Monitoring Framework For Virtualization Based HEP Infrastructures. *Journal of Physics: Conference Series*, 898(10):102004, 2017.

[22] A. Gomez Ramirez, C. Lara, L. Betev, D. Bilanovic, U. Kebschull, and the ALICE Collaboration. Arhuaco: Deep Learning and Isolation Based Security for Distributed High-Throughput Computing. *Journal of Physics: Conference Series*, 898(10):102004, 2017.

[23] H. Engel, T. Alt, T. Breitner, A. Gomez Ramirez, T. Kollegger, M. Krzewicki, J. Lehrbach, D. Rohr, and U. Kebschull. The ALICE high-level trigger readout upgrade for LHC Run 2. *Journal of Instrumentation*, 11(01):C01041, 2016.

[24] J. Lehrbach, M. Krzewicki, D. Rohr, H. Engel, A. Gomez Ramirez, V. Lindenstruth, D. Berzano, and ALICE Collaboration. ALICE HLT Cluster operation during ALICE Run 2. *Journal of Physics: Conference Series*, 898(8):082027, 2017.

[25] Ananya et al. and the Alice collaboration. O 2 : A novel combined online and offline computing system for the ALICE Experiment after 2018. *Journal of Physics: Conference Series*, 513(1):012037, 2014.

[26] National e-Science Centre. National e-Science Centre definition of e-Science, 2017. `http://www.nesc.ac.uk/nesc/define.html`.

[27] Steffen Schreiner. *A Security Architecture for e-Science Grid Computing*. PhD thesis, Technische Universität, Darmstadt, 2015.

[28] The ALICE Collaboration. The ALICE experiment at the CERN LHC. *Journal of Instrumentation*, 3(08):S08002, 2008.

[29] G. Aad et al. The ATLAS Experiment at the CERN Large Hadron Collider. *JINST*, 3:S08003, 2008.

[30] M. Bontenackels. The CMS muon spectrometer. *Nucl. Phys. Proc. Suppl.*, 156:124–128, 2006.

[31] The LHCb Collaboration. The LHCb Detector at the LHC. *Journal of Instrumentation*, 3(08):S08005, 2008.

[32] Christiane Lefèvre. The CERN accelerator complex. Complexe des accélérateurs du CERN. Dec 2008.

[33] G. Brumfiel. High-energy physics: Down the petabyte highway. *Nature*, i7330:282–283, 2015.

[34] ALICE Grid. ALICE Grid Monitoring with MonALISA, 2017. `http://pcalimonitor.cern.ch/`.

[35] Ian Foster, Carl Kesselman, and Steven Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *The International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[36] P Cortese, Federico Carminati, Christian Wolfgang Fabjan, Lodovico Riccati, and Hans de Groot. *ALICE computing: Technical Design Report*. Technical Design Report ALICE. CERN, Geneva, 2005. Submitted on 15 Jun 2005.

[37] I. Bird, K. Bos, N. Brook, D. Duellmann, C. Eck, I. Fisk, D. Foster, B. Gibbard, C. Grandi, F. Grey, et al. LHC computing Grid. Technical design report. 2005.

[38] S Jézéquel and G Stewart. ATLAS Distributed Computing Operations: Experience and improvements after 2 full years of data-taking. *Journal of Physics: Conference Series*, 396(3):032058, 2012.

[39] S Bagnasco, L Betev, P Buncic, F Carminati, C Cirstoiu, C Grigoras, A Hayrapetyan, A Harutyunyan, A J Peters, and P Saiz. AliEn: ALICE environment on the GRID. *Journal of Physics: Conference Series*, 119(6):062012, July 2008.

[40] Joint Security Policy Group. Grid Acceptable Use Policy, 2017. `http://cern.ch/proj-lcg-security`.

[41] ALICE Offline Group. Computing rules, 2017. `http://aliweb.cern.ch/Offline/General-Information/ComputingRules.html`.

[42] G. Duckeck, D. Barberis, R. Hawkings, R. Jones, N. McCubbin, G. Poulard, D. Quarrie, T. Wenaus, and E. Obreshkov. ATLAS computing: Technical design report. 2005.

[43] G L Bayatyan, Michel Della Negra, Foà, A Hervé, and Achille Petrilli. *CMS computing: Technical Design Report*. Technical Design Report CMS. CERN, Geneva, 2005. Submitted on 31 May 2005.

[44] The LHCb Collaboration. *LHCb computing: Technical Design Report*. Technical Design Report LHCb. CERN, Geneva, 2005. Submitted on 11 May 2005.

[45] LIGO Scientific Collaboration. Subsystem: Data and Computing Systems, 2017. `https://www.advancedligo.mit.edu/dcs.html`.

[46] Tahsin Kurc, Shannon Hastings, Vijay Kumar, Stephen Langella, Ashish Sharma, Tony Pan, Scott Oster, David Ervin, Justin Permar, Sivaramakrishnan Narayanan, Yolanda Gil, Ewa Deelman, Mary Hall, and Joel Saltz. HPC and Grid Computing for Integrative Biomedical Research. *The International Journal of High Performance Computing Applications*, 23(3):252–264, 2009. PMID: 20107625.

[47] Geo Grid. Global Earth Observation Grid, 2017. `http://www.geogrid.org/en/index.html`.

[48] Satoshi Sekiguchi et al. Design Principles and IT Overviews of the GEO Grid. *IEEE Systems Journal*, 2(3):374–389, 2008.

[49] e-BioGrid. http://www.e-biogrid.nl, 2017. `http://www.e-biogrid.nl`.

[50] Shayan Shahand, Mark Santcroos, Antoine H. C. van Kampen, and Sílvia Delgado Olabarriaga. A grid-enabled gateway for biomedical data analysis. *Journal of Grid Computing*, 10(4):725–742, Dec 2012.

[51] SURFsara. http://www.surfsara.nl, 2017.

[52] Jie Wu, René Siewert, Andreas Hoheisel, Jürgen Falkner, Oliver Strauß, Dinko Berberovic, and Dagmar Krefting. The charité grid portal: User-friendly and secure access to grid-based resources and services. *Journal of Grid Computing*, 10(4), Dec 2012.

[53] Larry Wall. *Programming Perl*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 3rd edition, 2000.

[54] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/soap, 2000.

[55] Michael Widenius and Davis Axmark. *Mysql Reference Manual*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1st edition, 2002.

[56] Brian Arkills. *LDAP Directories Explained: An Introduction and Analysis*. Addison-Wesley Professional, 2003.

[57] Ilya Selyuzhenkov and Alberica Toia, editors. *CBM Progress Report 2016*. GSI, Darmstadt, 2017. Literaturangaben.

[58] PANDA collaboration and Klaus Peters. Technical Design Report for the Panda Forward Spectrometer Calorimeter. 2017.

[59] Hans H. Gutbrod. International Facility for Antiproton and Ion Research (FAIR) at GSI, Darmstadt. *Nuclear Physics A*, 752(Supplement C):457 – 469, 2005. Proceedings of the 22nd International Nuclear Physics Conference (Part 2).

[60] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[61] S. Chokhani, W. Ford, R. Sabett, C. Merrill, and S. Wu. Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework, 2003.

[62] ALICE Offline Project. AliEn User Howto, 2017. `http://alien2.cern.ch/index.php?option=com_content&view=article&id=55&Itemid=95`.

[63] A. J. Peters. AliEn Grid User Reference Guide, 2017. `http://project-arda-dev.web.cern.ch/project-arda-dev/alice/apiservice/`.

[64] Abhineet Agarwal. Improving performance of AliRoot using C++11. Aug 2014.

[65] Harvey B. Newman, Iosif C. Legrand, Philippe Galvez, Ramiro Voicu, and Catalin Cirstoiu. Monalisa: A distributed monitoring service architecture. *arXiv preprint cs/0306096*, 2003.

[66] E. Auge et al. M. Aderholz, K. Amako. Models of networked analysis at regional centres for lhc experiments (monarc). phase 2 report. Technical report, 2000.

[67] Pablo Saiz, Predrag Buncic, and Andreas J. Peters. AliEn Resource Brokers. *CoRR*, cs.DC/0306068, 2003.

[68] Nicholas Coleman, Rajesh Raman, Miron Livny, and Marvin Solomon. Distributed policy management and comprehension with classified advertisements. Technical Report UW-CS-TR-1481, University of Wisconsin - Madison Computer Sciences Department, April 2003.

[69] Steve Mansfield-Devine. Security through isolation. *Computer Fraud & Security*, 2010(5):8–11, 2010.

[70] Secure-OS. `https://secure-os.org/`, 2017. `https://secure-os.org/`.

[71] Tails. `https://tails.boum.org/`, 2017.

[72] Qubes. `https://www.qubes-os.org/`, 2017.

[73] Subgraph-OS. `https://subgraph.com/sgos/`, 2017.

[74] Stefan Boettger. Virtual Machine Scheduling in Dedicated Computing Clusters, Jan 2014. Presented 14 Nov 2013.

[75] Redhat. What's a Linux container?, 2017. `https://www.redhat.com/en/topics/containers/whats-a-linux-container`.

[76] Miguel G. Xavier, Marcelo V. Neves, Fabio D. Rossi, Tiago C. Ferreto, Timoteo Lange, and Cesar A. F. De Rose. Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments. In *Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, PDP '13, pages 233–240, Washington, DC, USA, 2013. IEEE Computer Society.

[77] Dirk Merkel. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux J.*, 2014(239), March 2014.

[78] from Wikimedia Commons Shmuel Csaba Otto Traian [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0) or GFDL (http://www.gnu.org/copyleft/fdl.html)]. Linux kernel unified hierarchy cgroups and systemd, 2018.

[79] Linux Weekly Edition. User namespaces progress, 2017. `https://lwn.net/Articles/528078/`.

[80] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based Network Intrusion Detection: Techniques, Systems and Challenges. *Comput. Secur.*, 28(1-2):18–28, February 2009.

[81] Dimitrios Damopoulos. *Anomaly-Based Intrusion Detection and Prevention Systems for Mobile Devices: Design and Development*. PhD thesis, Laboratory of Information and Communication Systems Security ,University of the Aegean, 2013.

[82] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, January 2013.

[83] Christopher M. Bishop. *Pattern recognition and machine learning*. Information science and statistics. Springer, New York, 2006.

[84] Raphaël Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine Learning Classification over Encrypted Data.

[85] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[86] S.S. Haykin. *Neural Networks and Learning Machines*. Number Bd. 10 in Neural networks and learning machines. Prentice Hall, 2009.

[87] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[88] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[89] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling, 2016.

[90] Ian Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A Security Architecture for Computational Grids. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, CCS '98, pages 83–92, New York, NY, USA, 1998. ACM.

[91] Ali Raza Butt, Sumalatha Adabala, Nirav H. Kapadia, Renato J. Figueiredo, and José A. B. Fortes. Grid-computing Portals and Security Issues. *J. Parallel Distrib. Comput.*, 63(10):1006–1014, October 2003.

[92] M. Smith, M. Schmidt, N. Fallenbeck, T. Dörnemann, C. Schridde, and B. Freisleben. Secure on-demand grid computing. *Future Generation Computer Systems*, 25(3):315 – 325, 2009.

[93] Matthew Smith, Michael Engel, Thomas Friese, Bernd Freisleben, Gregory A. Koenig, and William Yurcik. Security Issues in On-Demand Grid and Cluster Computing. In *CCGRID*, page 24. IEEE Computer Society, 2006.

[94] Matthew Smith, Thomas Friese, Michael Engel, and Bernd Freisleben. Countering Security Threats in Service-oriented On-demand Grid Computing Using Sandboxing and Trusted Computing Techniques. *J. Parallel Distrib. Comput.*, 66(9):1189–1204, September 2006.

[95] Steffen Schreiner, Latchezar Betev, Costin Grigoras, and Maarten Litmaath. A Mediated Definite Delegation Model allowing for Certified Grid Job Submission. *CoRR*, abs/1112.2444, 2011.

[96] Worldwide LHC Computing Grid Collaboration. WLCG Grid job and Worker Node security assessment, 2012. `https://twiki.cern.ch/twiki/pub/LCG/AAIOnTheWorkerNodes/WLCG_WN_Security-06.pdf`.

[97] Steffen Schreiner, Costin Grigorasb, Alina Grigorasb, Latchezar Betevb, and Johannes Buchmannac. A security architecture for the ALICE Grid Services. In *The International Symposium on Grids and Clouds (ISGC)*, volume 2012. Citeseer, 2012.

[98] Steffen Schreiner, Costin Grigoras, Maarten Litmaath, Latchezar Betev, and Johannes Buchmann. Mediated definite delegation - Certified Grid jobs in ALICE and beyond. *Journal of Physics: Conference Series*, 396(3):032096, 2012.

[99] Shuo Yang, Ali Raza Butt, Xing Fang, Y. Charlie Hu, and Samuel P. Midkiff. A Fair, Secure and Trustworthy Peer-to-Peer Based Cycle-Sharing System. *J. Grid Comput.*, 4(3):265–286, 2006.

[100] Benjamin Quétier, Vincent Neri, and Franck Cappello. Scalability comparison of four host virtualization tools. *Journal of Grid Computing*, 5(1):83–98, Mar 2007.

[101] A Harutyunyan, P Buncic, T Freeman, and K Keahey. Dynamic virtual AliEn Grid sites on Nimbus with CernVM. *Journal of Physics: Conference Series*, 219(7):072036, 2010.

[102] A Harutyunyan, P Buncic, T Freeman, and K Keahey. Dynamic virtual AliEn Grid sites on Nimbus with CernVM. *Journal of Physics: Conference Series*, 219(7):072036, April 2010.

[103] Nane Kratzke. Lightweight Virtualization Cluster - Howto overcome Cloud Vendor Lock-in. *Journal of Computer and Communication (JCC)*, 2(12), October 2014.

[104] René Peinl, Florian Holzschuher, and Florian Pfitzer. Docker cluster management for the cloud - survey results and own solution. *Journal of Grid Computing*, 14(2):265–282, Jun 2016.

[105] Sreenivas Makam. *Mastering CoreOS*. Packt Publishing, 2016.

[106] Dirk Merkel. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux J.*, 2014(239), March 2014.

[107] J. Ma, H. Kim, and Y. Kim. The Virtualization and Performance Comparison with LXC-LXD in ARM64bit Server. In *2016 6th International Conference on IT Convergence and Security (ICITCS)*, volume 00, pages 1–4, Sept. 2016.

[108] Open Container Initiative. Creating open standards around container formats and runtimes, 2017. `https://github.com/opencontainers`.

[109] David K. Rensin. *Kubernetes - Scheduling the Future at Cloud Scale*. 1005 Gravenstein Highway North Sebastopol, CA 95472, 2015.

[110] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy H. Katz, Scott Shenker, and Ion Stoica. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In *NSDI*, volume 11, pages 22–22, 2011.

[111] D Berzano, G Eulisse, C Grigoras, and K Napoli. Experiences with the ALICE Mesos infrastructure. *Journal of Physics: Conference Series*, 898(8):082043, 2017.

[112] Tiago Rosado and Jorge Bernardino. An Overview of Openstack Architecture. In *Proceedings of the 18th International Database Engineering &#38; Applications Symposium*, IDEAS '14, pages 366–367, New York, NY, USA, 2014. ACM.

[113] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating system concepts*. Wiley, Hoboken, NJ, ninth edition edition, 2013.

[114] D Berzano, J Blomer, P Buncic, I Charalampidis, G Ganis, and R Meusel. Lightweight scheduling of elastic analysis containers in a competitive cloud environment: a docked analysis facility for ALICE. *Journal of Physics: Conference Series*, 664(2):022005, 2015.

[115] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: A Platform for Fine-grained Resource Sharing in the Data Center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 295–308, Berkeley, CA, USA, 2011. USENIX Association.

[116] Chih-Fong Tsai, Yu-Feng Hsu, Chia-Ying Lin, and Wei-Yang Lin. Intrusion detection by machine learning: A review. *Expert Systems with Applications*, 36(10):11994–12000, December 2009.

[117] Saharon Rosset and Aron Inger. KDD-cup 99: Knowledge discovery in a charitable organization's donor database. *SIGKDD Explor. Newsl.*, 1(2):85–90, January 2000.

[118] Shelly Xiaonan Wu and Wolfgang Banzhaf. The use of computational intelligence in intrusion detection systems: A review. *Applied Soft Computing*, 10(1):1–35, January 2010.

[119] John McHugh. Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations As Performed by Lincoln Laboratory. *ACM Trans. Inf. Syst. Secur.*, 3(4):262–294, November 2000.

[120] Ong Tian Choon and A. Samsudin. Grid-based intrusion detection system. In *9th Asia-Pacific Conference on Communications (IEEE Cat. No.03EX732)*, volume 3, pages 1028–1032 Vol.3, Sept 2003.

[121] M. Tolba, M. Abdel-Wahab, I. Taha, and A. Al-Shishtawy. GIDA: Toward enabling grid intrusion detection systems. In *Proc. of the Second International Intelligent Computing and Information Systems Conference*, 2005.

[122] A. Schulter, J. A. Reis, F. Koch, and C. B. Westphall. A Grid-based Intrusion Detection System. In *International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL'06)*, pages 187–187, April 2006.

[123] Stuart Kenny and Brian Coghlan. Towards a grid-wide intrusion detection system. In *Advances in Grid Computing-EGC 2005*, pages 275–284. Springer, 2005.

[124] Guofu Feng, Xiaoshe Dong, Weizhe Liu, Ying Chu, and Junyang Li. GHIDS: Defending computational grids against misusing of shared resources. In *Services Computing, 2006. APSCC'06. IEEE Asia-Pacific Conference on*, pages 526–533. IEEE, 2006.

[125] Pei-You Zhu, Ji Gao, Bo-Ou Jiang, and Hui Song. A new flexible multi-agent approach to intrusion detection for Grid. In *Machine Learning and Cybernetics, 2006 International Conference on*, pages 7–12. IEEE, 2006.

[126] A. Bosin, N. Dessì, and B. Pes. A Service Based Approach to a New Generation of Intrusion Detection Systems. In *2008 Sixth European Conference on Web Services*, pages 215–224, Nov 2008.

[127] Ni Jiancheng, Li Zhishu, Sun Jirong, and Xing Jianchuan. Self-adaptive intrusion detection system for computational grid. In *Theoretical Aspects of Software Engineering, 2007. TASE'07. First Joint IEEE/IFIP Symposium on*, pages 97–106. IEEE, 2007.

[128] M. Smith, F. Schwarzer, M. Harbach, T. Noll, and B. Freisleben. A Streaming Intrusion Detection System for Grid Computing Environments. In *2009 11th IEEE International Conference on High Performance Computing and Communications*, pages 44–51, June 2009.

[129] Alexandre Schulter, Kleber Vieira, Carla Westphall, and Carlos Westphall. Intrusion Detection for Grid and Cloud Computing. *IT Professional*, 12:38–43, 2009.

[130] I. Ungureanu, C. Leordeanu, and V. Cristea. Grid-Aware Intrusion Detection System Using Gossip Algorithms. In *2010 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 286–292. IEEE, September 2010.

[131] Raheel Hassan Syed, Jasmina Pazardzievska, and Julien Bourgeois. Fast attack detection using correlation and summarizing of security alerts in grid computing networks. *The Journal of Supercomputing*, 62(2):804–827, Nov 2012.

[132] Ahmed Patel, Mona Taghavi, Kaveh Bakhtiyari, and Joaquim Celestino Júnior. An intrusion detection and prevention system in cloud computing: A systematic review. *Journal of Network and Computer Applications*, 36(1):25–41, January 2013.

[133] M. Tolba, M. Abdel-Wahab, I. Taha, and A. Al-Shishtawy. Distributed Intrusion Detection System for Computational Grids. In *International Conference On Intelligent Computing And Information Systems*, volume 2, page 2005, 2005.

[134] Guiling Zhang and Jizhou Sun. Grid intrusion detection based on soft computing by modeling real-user's normal behaviors. In *Granular Computing, 2006 IEEE International Conference on*, pages 558–561. IEEE, 2006.

[135] Alexandre Schulter, Kleber Vieira, C. Westphall, and S. Abderrahim. Intrusion detection for computational grids. In *New Technologies, Mobility and Security, 2008. NTMS'08.*, pages 1–5. IEEE, 2008.

[136] Seonho Kim, Jinoh Kim, and Jon B. Weissman. A security-enabled grid system for minds distributed data mining. *Journal of Grid Computing*, 12(3):521–542, Sep 2014.

[137] Ram Shankar Siva Kumar, Andrew Wicker, and Matt Swann. Practical Machine Learning for Cloud Intrusion Detection: Challenges and the Way Forward. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, AISec '17, pages 81–90, New York, NY, USA, 2017. ACM.

[138] Renaud Bidou, Julien Bourgeois, and Francois Spies. *Towards a Global Security Architecture for Intrusion Detection and Reaction Management*, pages 111–123. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[139] Abdoul Karim Ganame, Julien Bourgeois, Renaud Bidou, and Francois Spies. A global security architecture for intrusion detection on computer networks. *Computers & Security*, 27(1):30–47, 2008.

[140] David Crooks and Liviu Vâlsan. Wlcg security operations centres working group. In *International Symposium on Grids & Clouds 2017 (ISGC 2017)*, December 2017. Authors ... for the WLCG SOC Working Group.

[141] Apache Software Foundation. Apache Metron: real-time big data security, 2017. `http://metron.apache.org/about/`.

[142] Cynthia Wagner, Alexandre Dulaunoy, Gérard Wagener, and Andras Iklody. MISP: The Design and Implementation of a Collaborative Threat Intelligence Sharing Platform. In *Proceedings of the 2016 ACM on Workshop on Information*

Sharing and Collaborative Security, WISCS '16, pages 49–56, New York, NY, USA, 2016. ACM.

[143] Chandrashekhar Azad and Vijay Kumar Jha. Data Mining in Intrusion Detection: A Comparative Study of Methods, Types and Data Sets. *International Journal of Information Technology and Computer Science*, 5(8):75–90, July 2013.

[144] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. The 1999 darpa off-line intrusion detection evaluation. *Comput. Netw.*, 34(4):579–595, October 2000.

[145] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*, 31(3):357–374, May 2012.

[146] Canadian Institute for Cybersecurity. Intrusion detection evaluation dataset (ISCXIDS2012), 2017. `http://www.unb.ca/cic/datasets/ids.html`.

[147] The cooperative association for internet data analysis. CAIDA, 2017. `http://www.caida.org/`.

[148] RTI International. PREDICT: Protected repository for the defense of infrastructure against cyber threats, 2017. `http://www.predict.org/`.

[149] Lawrence Berkeley National Laboratory. The internet traffic archive, 2017. `http://ita.ee.lbl.gov/index.html`.

[150] Lawrence Berkeley National Laboratory and ICSI. LBNL/ICSI enterprise tracing project, 2017. `www.icir.org/enterprise-tracing/`.

[151] The Shmoo Group. LBNL/ICSI enterprise tracing project, 2017. `http://cctf.shmoo.com/`.

[152] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[153] Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Düssel, and Pavel Laskov. Learning and classification of malware behavior. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 108–125. Springer, 2008.

[154] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, 19(4):639–668, 2011.

[155] Daniel Gibert Llauradó. *Convolutional neural networks for malware classification*. PhD thesis, Universitat Politecnica de Catalunya, 2016.

[156] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. Droid-Sec: Deep Learning in Android Malware Detection. *SIGCOMM Comput. Commun. Rev.*, 44(4):371–372, August 2014.

[157] Hanlin Zhang, Yevgeniy Cole, Linqiang Ge, Sixiao Wei, Wei Yu, Chao Lu, Genshe Chen, Dan Shen, Erik Blasch, and Khanh D. Pham. ScanMe Mobile: A Cloud-based Android Malware Analysis Service. *SIGAPP Appl. Comput. Rev.*, 16(1):36–49, April 2016.

[158] Qinglong Wang, Wenbo Guo, Kaixuan Zhang, Alexander G. Ororbia, II, Xinyu Xing, Xue Liu, and C. Lee Giles. Adversary Resistant Deep Neural Networks with an Application to Malware Detection. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 1145–1153, New York, NY, USA, 2017. ACM.

[159] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick D. McDaniel. Adversarial Perturbations Against Deep Neural Networks for Malware Classification. *CoRR*, abs/1606.04435, 2016.

[160] Weiwei Hu and Ying Tan. Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. *CoRR*, abs/1702.05983, 2017.

[161] Linzhang Wang, Eric Wong, and Dianxiang Xu. A Threat Model Driven Approach for Security Testing. In *Proceedings of the Third International Workshop on Software Engineering for Secure Systems*, SESS '07, pages 10–, Washington, DC, USA, 2007. IEEE Computer Society.

[162] Dave Kelsey, Maarten Litmaath, Steffen Schreiner, Von Welch, Romain Wartel, John White, Christoph Witzig, and the members of the WLCG Security Technology Evolution Group. WLCG Computer Security Risks Analysis, 2017. `http://rwartel.web.cern.ch/rwartel/security_teg/WLCG%20Risk%20Assessment.pdf`.

[163] Chirag Modi, Dhiren Patel, Bhavesh Borisaniya, Hiren Patel, Avi Patel, and Muttukrishnan Rajarajan. Review: A Survey of Intrusion Detection Techniques in Cloud. *J. Netw. Comput. Appl.*, 36(1):42–57, January 2013.

[164] Bruce Schneier. *Applied Cryptography (2Nd Ed.): Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[165] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

[166] Apache Hadoop. Transparent Encryption in HDFS, 2018. `https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/TransparentEncryption.html`.

[167] Clinton Gormley and Zachary Tong. *Elasticsearch: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2015.

[168] Vishal Sharma. *Getting Started with Kibana*, pages 29–44. Apress, Berkeley, CA, 2016.

[169] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. *Microservices: Yesterday, Today, and Tomorrow*, pages 195–216. Springer International Publishing, Cham, 2017.

[170] Dario Berzano. A ground-up approach to High-Throughput Cloud Computing in High-Energy Physics.

[171] Mallik Mahalingam, Dinesh Dutt, Kenneth Duda, Puneet Agarwal, Larry Kreeger, T. Sridhar, Mike Bursell, and Chris Wright. Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. RFC 7348, August 2014.

[172] Gentoo. Aufs (Another Union File System), 2018. `https://wiki.gentoo.org/wiki/Aufs`.

[173] Docker. The OverlayFS storage driver, 2018. `https://docs.docker.com/storage/storagedriver/overlayfs-driver/`.

[174] Emily Le and David Paz. Performance Analysis of Applications Using Singularity Container on SDSC Comet. In *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*, PEARC17, pages 66:1–66:4, New York, NY, USA, 2017. ACM.

[175] Daniel Bilanovic. Analysis of security isolation technologies for HEP-Computing. Master's thesis, Johann Wolfgang Goethe-Universität, 2018. `http://publikationen.ub.uni-frankfurt.de/frontdoor/index/index/docId/47931`.

[176] María Arsuaga-Ríos, Seppo S Heikkilä, Dirk Duellmann, René Meusel, Jakob Blomer, and Ben Couturier. Using S3 cloud storage with ROOT and CvmFS. *Journal of Physics: Conference Series*, 664(2):022001, 2015.

[177] Intel. Getting Started with Intel Active Management Technology (AMT), 2018. `https://software.intel.com/en-us/articles/getting-started-with-intel-active-management-technology-amt`.

[178] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg. Meltdown. *ArXiv e-prints*, January 2018.

[179] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. Spectre Attacks: Exploiting Speculative Execution. *ArXiv e-prints*, January 2018.

[180] Grsecurity. Grsecurity patches for the Linux kernel, 2018. `https://grsecurity.net/`.

[181] Joint Security Policy Group. Grid Security Traceability and Logging Policy, 2018. `https://edms.cern.ch/ui/file/428037/3/Traceability-Logging-v2.0.pdf`.

[182] Sysdig. `http://www.sysdig.org`, 2017.

[183] Jeff Garzik. *Glibc: a comprehensive reference to GNU/Linux libC*. 2000. Edited by Laurie Petrycki and others.

[184] Jurgen Schmidhuber. Multi-column Deep Neural Networks for Image Classification. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pages 3642–3649, Washington, DC, USA, 2012. IEEE Computer Society.

[185] VirusShare. `http://virusshare.com/`, 2017. `http://virusshare.com/`.

[186] Virustotal. `https://www.virustotal.com/`, 2017. `https://www.virustotal.com/`.

[187] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. *The Network and Distributed System Security Symposium (NDSS)*, January 2018.

[188] Suvrit Sra, Sebastian Nowozin, and Stephen J. Wright. *Optimization for Machine Learning*. The MIT Press, 2011.

[189] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. *CoRR*, abs/1310.4546, 2013.

[190] Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong. Vuldeepecker: A deep learning-based system for vulnerability detection. *CoRR*, abs/1801.01681, 2018.

[191] Steven Gold and Anand Rangarajan. Softmax to softassign: Neural network algorithms for combinatorial optimization. *Journal of Artificial Neural Networks*, 2:2–4, 1995.

[192] Marc'Aurelio Ranzato, Y-Lan Boureau, and Yann LeCun. Sparse Feature Learning for Deep Belief Networks. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 1185–1192, 2007.

[193] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[194] By Aphex34 [CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0)] from Wikimedia Commons. Typical CNN, 2018.

[195] Inetsim. `http://www.inetsim.org/`, 2017.

[196] Digit Oktavianto and Iqbal Muhardianto. *Cuckoo Malware Analysis*. Packt Publishing, 2013.

[197] Python Core Team (2018). Python: A dynamic, open source programming language, Python Software Foundation, 2018. `https://www.python.org/`.

[198] Dario Berzano. CentOS 6 Cloud Image, 2018. `https://dberzano.github.io/cloud/centos/`.

[199] CoreOS. rkt container capabilities. [Online, accessed November 12th 2017].

[200] Morris A. Jette, Andy B. Yoo, and Mark Grondona. Slurm: Simple linux utility for resource management. In *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*, pages 44–60. Springer-Verlag, 2002.

[201] Alastair P. Droop. qsubsec: a lightweight template system for defining sun grid engine workflows. *Bioinformatics*, 32(8):1267–1268, 2016.

[202] Felix Fuentes and Dulal C. Kar. Ethereal vs. Tcpdump: A Comparative Study on Packet Sniffing Tools for Educational Purpose. *J. Comput. Sci. Coll.*, 20(4):169–176, April 2005.

[203] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[204] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[205] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[206] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[207] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[208] Matthew D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. *CoRR*, abs/1212.5701, 2012.

[209] Thomas Uphill. *Mastering Puppet - Second Edition*. Packt Publishing, 2nd edition, 2016.

[210] Thomas Haynes and David Noveck. Network File System (NFS) Version 4 Protocol. RFC 7530, March 2015.

[211] ALICE Collaboration. AliRoot Test PbPbbench, 2018.

[212] J. D. Rodriguez, A. Perez, and J. A. Lozano. Sensitivity analysis of k-fold cross validation in prediction error estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(3):569–575, March 2010.

Appendix A

Abbreviations

ACC	Accuracy
API	Application programming interface
AI	Artificial intelligence
ALICE	A large Ion Collider Experiment
AliEn	ALICE production environment
APIDS	Application based IDS
ANN	Artificial neural networks
ATLAS	A Toroidal LHC Apparatus
ASLR	Address-space layout randomization
BoW	Bag-of-words
CERN	European Organization for Nuclear Research
CernVM-FS	CernVM File System
CMS	Compact Muon Spectrometer
CNN	Convolutional neural network
CLI	Command line interface
DBN	Deep belief networks
DNN	Deep neural network
DSOC	Distributed security operation center
DL	Deep learning
DoS	Denial of service
DDoS	Distributed denial of service
FPR	False positive rate
FP	False positives
FN	False negatives
GAN	Generative adversarial network
GN	Genetic algorithms
GM	Generative model
GSI	Helmholtzzentrum für Schwerionenforschung
HEP	High energy physics
HIDS	Host-based IDS
HLT	High level trigger
HPC	High performance computing

HTC	High throughput computing
IO	Input-output
IDPS	Intrusion detection and prevention system
IDS	Intrusion detection system
IDXP	Intrusion detection exchange protocol
JDL	Job description language
K-NN	K-nearest neighbor
LDAP	Lightweight directory access protocol
LC	Linux container
LFN	Logical file name
LHC	Large Hadron Collider
LHCb	Large Hadron Collider Beauty
LSTM	Long short-term memory
ML	Machine learning
NLP	Natural language processing
NIDS	Network-based IDS
OS	Operating system
PaaS	Platform as a service
QCD	Quantum chromodynamics
RNN	Recurrent neural networks
RMS	Root mean square propagation
SaaS	Software as a service layer
SbI	Security by isolation
SE	Storage element
SIEM	Security information and event management system
SOAP	Simple object access protocol
SOC	Security operation center
SPC	Specificity
SVM	Support vector machine
TPM	Trusted platform module
TPR	True positive rate
TN	True negatives
TP	True positives
VO	Virtual organization
VM	Virtual machine
WLCG	Worldwide LHC Computing Grid
WN	Worker node

Appendix B

Curriculum Vitae

B.1 Personal Details

Name	Andrés Gómez Ramírez
Date of Birth	September 24, 1985
Place of Birth	Marinilla, Colombia
Address	IRI group, Goethe-Universität Frankfurt am Main Hauspostfach 23 Senckenberganlage 31 60325, Frankfurt am Main
E-mail	andres.gomez@iri.uni-frankfurt.de
Citizenship	Colombian

B.2 Education

2003-2011	Universidad de Antioquia, Bachelor on Computer Sciences. Supervisor: Prof. Dr. Claudia Isaza Narvaez.
2013-2018	Goethe University Frankfurt am Main, Ph. D. on Computer Sciences. Supervisor: Prof. Dr. Udo Kebschull

B.3 Publications

1. **A. Gomez Ramirez**, C. Lara, U. Kebschull for the ALICE Collaboration. *"Intrusion Prevention and Detection in Grid Computing - The ALICE Case"*. Journal of Physics: Conference Series, 664(6):062017, 2015. [20]

2. **A. Gomez Ramirez**, M. Martinez Pedreira, C. Grigoras, L. Betev, C. Lara and U. Kebschull for the ALICE Collaboration. *"A Security Monitoring Framework For Virtualization Based HEP Infrastructures"*. Journal of Physics: Conference Series, 898(10):102004, 2017. [21]

3. **A. Gomez Ramirez**, C. Lara, L. Betev, D. Bilanovic, U. Kebschull for the ALICE Collaboration. *"Arhuaco: Deep Learning and Isolation Based Security for Distributed High-Throughput Computing"*. Manuscript submitted to J. Grid Computing (2018). [22]

4. H. Engel, T. Alt, T. Breitner, **A. Gomez Ramirez**, T. Kollegger, M. Krzewicki, J. Lehrbach, D. Rohr, and U. Kebschull. *"The ALICE High-level Trigger read-out upgrade for LHC Run 2"*. Journal of Instrumentation, 11(01):C01041, 2016. [23]

5. J. Lehrbach, M. Krzewicki, D. Rohr, H. Engel, **A. Gomez Ramirez**, V. Lindenstruth, D. Berzano, and ALICE Collaboration. *"ALICE HLT Cluster operation during ALICE Run 2"*. Journal of Physics: Conference Series, 898(8):082027, 2017. [24]

6. Ananya, A Alarcon Do Passo Suaide, C Alves Garcia Prado, T Alt, L Aphecetche, N Agrawal, A Avasthi, M Bach, R Bala, G Barnafoldi, A Bhasin, J Belikov, F Bellini, L Betev, T Breitner, P Buncic, F Carena, W Carena, S Chapeland, V Chibante Barroso, F Cliff, F Costa, L Cunqueiro Mendez, S Dash, C Delort, E Denes, R Divia, B Doenigus, H Engel, D Eschweiler, U Fuchs, A Gheata, M Gheata, **A. Gomez Ramirez** et al. for the Alice collaboration. collaboration. *"O2 : A novel combined online and offline computing system for the alice experiment after 2018"*. Journal of Physics: Conference Series, 513(1):012037, 2014. [25]

7. **A. Gomez Ramirez**, C. Lara, U. Kebschull. *"Machine Learning Based Monitoring of HEP Computing Infrastructures"*. GSI Scientific Report 2016, doi:10. 15120/ GR-2017-1, page 364, Darmstadt, Germany, 2017.

8. **A. Gomez Ramirez**, C. Lara, U. Kebschull. *"Machine learning based monitoring of HEP computing infrastructures"*. CBM Progress Report 2016 - Computing, page 163, Darmstadt, Germany, 2017.

9. **A. Gomez Ramirez** for the ALICE collaboration. *"Deep Learning and Isolation Based Security for Grid Computing"*. DPG Conference, HK 52.67, Bochum, Germany, March 2018.

10. **A. Gomez Ramirez**, C. Lara, U. Kebschull. *"Intrusion prevention and detection in Grid computing"*. ALICE Tier-1/2 Workshop. Torino, Italy, 2015.

11. **A. Gomez Ramirez**, C. Lara, U. Kebschull. *"Alice computing monitoring: O2, security and debugging"*. FSP ALICE Meeting 2017 – Buchenau, Eiterfeld.

12. **A. Gomez Ramirez**, C. Lara, U. Kebschull. *"HLT Cluster Monitoring"*. ALICE Germany Meeting, 2016.

13. **A. Gomez Ramirez**, C. Lara, U. Kebschull. *"Improving Grid Security for HEP – an AliEn implementation"*. FSP ALICE meeting 2015.

14. **A. Gomez Ramirez**, C. Lara, U. Kebschull. *"Computer Security in the ALICE experiment: detecting and reacting to cyber-attacks"*. FSP ALICE meeting. 2014.

15. **A. Gomez Ramirez**, C. Lara, U. Kebschull. *"Improving Grid Security for HEP"*. 26th CBM Collaboration Meeting 2015.

B.4 Publications as Collaborator of ALICE

The full list of publications is available in the URL:
`http://inspirehep.net/author/profile/A.Gomez.ramirez.2`