TÉCNICO LISBOA

Profiling Agent and Rule-set Optimization in Misuse Based Intrusion Detection Systems

André Alves Gomes Rei

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Doutor Carlos Nuno da Cruz Ribeiro

Examination Committee

Chairperson:	Prof. Doutor António Manuel Ferreira Rito da Silva
Supervisor:	Prof. Doutor Carlos Nuno da Cruz Ribeiro
Member of the Committee:	Prof. Doutor André Ventura da Cruz Marnoto Zúquete

June 2015

Dedicated to my family.

Resumo

As Infraestruturas Críticas são recursos muito importantes, e com o crescimento massivo da tecnologia, têm-se tornado alvos de ataques cibernéticos. Sistemas de Detecção de Intrusos são usados para proteger estas infraestruturas, mas estes não são perfeitos. Os sistemas de detecção baseados em mau uso produzem falsos alarmes se as regras não estiverem bem configuradas. A nossa solução pretende resolver esse problema, optimizando o conjunto de regras usado pelo sistema de detecção.

A ferramenta desenvolvida recolhe informação acerca das regras usadas pelo sistema de detecção e relaciona-a com as aplicações instaladas na máquina onde o sistema de detecção se encontra a actuar, com o objectivo the manter apenas as regras relacionadas com as aplicações instaladas. Desenvolvemos também um módulo capaz de detectar quais as aplicações instaladas na máquina, que será usado como input do módulo de optimização de regras. A nossa solução utiliza algumas técnicas, tais como comparação de strings aproximadas e web scraping.

Através da avaliação da solução, foi possível concluir que o principal objectivo foi conseguido, visto que a ferramenta gerou uma optimização de 80% no conjunto de regras.

Palavras-chave: Sistema de detecção de intrusos, assinatura, regra, mau uso, optimização, caracterização.

Abstract

Critical Infrastructures are very important assets and with the massive technology growth, these infrastructures have become cyber-attack targets. Intrusion Detection Systems are used to protect the infrastructures but they are not perfect. Misuse Based IDSs generate false alarms if the rules are not configured or used properly. Our solution tries to solve this problem by optimizing the rule-set used by the misuse based IDS.

Our tool collects information about the rules used by the IDS and matches this information with the applications installed on system where the IDS is running, so that only rules related to those applications are used by the IDS. We also developed a module capable of detecting what applications are installed on the host, which is used as input by the rule-set optimization module. Different techniques are used in our tool, such as approximate string matching and web scrapping.

From the evaluation, we were able to conclude that the main objective was accomplished, as the tool produced an optimization of 80%.

Keywords: IDS, signature, rule, misuse, optimization, profiling.

Contents

List of Tables

List of Figures

List of Algorithms

Acronyms

API Application Programming Interface.

CSS Cascading Style Sheets.

DOM Document Object Model.

DoS Denial of Service.

HIDS Host Intrusion Detection System.

HMI Human-Machine Interface.

HTML Hypertext Markup Language.

HTQL Hypertext Query Language.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

IDS Intrusion Detection System.

IP Internet Protocol.

IT Information Technology.

NIDS Network Intrusion Detection System.

OS Operating System.

PLC Programmable Logic Controller.

RTU Remote Terminal Unit.

SCADA Supervisory Control and Data Acquisition.

SECUR-ED The Secured Urban Transportation – European Demonstration.

SSH Secure Shell.

TCCA Tehama-Colusa Canal Authority.

TCP Transmission Control Protocol.

URL Uniform Resource Locator.

VPN Virtual Private Network.

XSS Cross-site scripting.

Chapter 1

Introduction

Critical Infrastructures play a big role in today's world. Without it there would be no society and economy. So, what are these Critical Infrastructures? As defined in the Council Directive 2008/114/EC [4], it is "an asset, system or part thereof located in Member States which is essential for the maintenance of vital society functions, health, safety, security, economic or social well-being of people, and the disruption or destruction of which would have a significant impact in a Member State as a result of the failure to maintain those functions". Infrastructures that manage Energy, Transportation and Water are examples of infrastructures that are considered critical. In their operation, these infrastructures execute a set of physic industrial processes (manufacturing, production, power generation, fabrication, refining, etc.) and IT processes to help it accomplish its goal. These processes are monitored and controlled by Supervisory Control and Data Acquisition (SCADA) systems.

1.1 Problem

SCADA systems are computer-controlled systems that are considered privileged targets for cyber-attacks, because when they were developed, the goal was to create a control system that provided good performance and with features that made it easy to control. Security was not a concern then [27]. Also, to keep up with the technology growth, these system must also grow and connect to other systems to increase the scope, making it open to threats. So, to safeguard these infrastructures, there is a need for measures to deal with these threats. Intrusion Detection Systems are a strong mechanism to help detect attacks in the systems, but they have a common problem of generating many false warnings. So, to increase their effectiveness, they need to adapt to the environment where they are running.

1.2 Motivation

In 2007, an electrical supervisor of a small California canal system TCCA installed unauthorized software on the Infrastructure's SCADA system, damaging the computer used to divert the water from the river [13]. TCCA provides water for agriculture in central California, so it is of extremely importance. The insider now faces 10 years in prison and his actions cost the TCCA more than $5,000 in damages,

showing that the threats don't necessarily have to come from the outside of the system. Also, in 2011, hackers managed to access servers hosting confidential information of companies involved in the research, development and manufacture of chemicals and advanced materials [13]. The Trojan called "Poison Ivy" was planted on Windows PCs via message attachment on emails and was installed unknowingly when users opened that attachment. After that, the attackers could issue instructions to the compromised computers and fish for higher level passwords which could grant them access to the desired servers. Twenty-nine of the forty-eight firms that were attacked were in the chemical and advanced materials trade (some with connections to military vehicles) while the others were in a variety of fields, including the defense sector. This shows how easy it was for the attackers to inject the Trojan in the system and that this type of attacks are becoming common in systems that deal with sensitive data. As we can see, threats are everywhere, either on the outside or on the inside, and they can cause a big impact in any Critical Infrastructure that doesn't have the proper security mechanisms in practice. The The Secured Urban Transportation – European Demonstration (SECUR-ED) project was created with the goal of developing a set of technologies and processes that will help improve urban transport security [2]. To take advantage of this effort that is being made on the Mass Transportation Infrastructure, it would be great if some of those techniques could be applied or adapted to work in different Critical Infrastructures and even cooperate amongst each other. In [19], it is described a hybrid Intrusion Detection System (IDS) that uses concepts of both specification-based and misuse detection techniques. This IDS was designed for the mass transportation infrastructure (which extensively uses SCADA systems), but it might be adaptable to other critical infrastructures. We expect that our solution will contribute to this system, by providing a tool to optimize its misuse detection rule-set.

1.3 Solution

The main goal of an IDS is to protect the Infrastructure's IT systems from internal and external attacks. So, the perfect IDS would be one that only produced the right alerts, i.e. no false negatives and no false positives. Recent experience shows that achieving that is pretty much impossible [29], so we must focus on finding a balance between those alerts while reducing them as much as possible, so that we can make the IDS as effective as possible. For the misuse detection component, intrusion detection systems simply deploy all the existing signatures on the repository, which leads to unnecessary false alarms, because the IDS is applying the rules defined for all the applications, even those that do not exist on the system where the IDS is running. Our goal will be to analyze all the existing rules in the repository, and determine which ones are related with the applications running on the system, so that only those will be applied on the sensor.

1.4 Objectives

The main objectives of this work are:

- Designing and implementing a module capable of:
  - Detecting which applications are installed on the system.
  - Detecting which rules are related to the applications installed on the system where the IDSs are running.
- Evaluate the results.

1.5 Structure of the document

The next Chapter describes the state-of-the-art, where we describe the Intrusion Detection Systems, and give some examples of their application. We also describe Critical Infrastructures to give a main idea of it is composed and what is it used for, and finally, we present some approximate string matching techniques In Chapter 3, the solution description will be described and in Chapter 4, the solution evaluation will be presented. Finally, Chapter 5 contains the conclusions about the project and to finish the paper we will have the references in last section.

Chapter 2

State Of The Art

2.1 Intrusion Detection Systems

With the technology continuous growth, everything depends on the Internet nowadays, and because of that, new ways of bypassing the network security are created every moment, and if we are not prepared, we could be the next target.

When we use the Internet, we want to prevent all the unauthorized accesses to our network. It is the same principle that we use when building a house, by putting locks in all the places that give access to the house, like doors and windows. Unfortunately, intruders manage to find ways of exploiting our security mechanisms, like destroying the locks, breaking the windows, etc., and from that moment there's nothing we can do.

With our network, we must have the same concern. When we use a firewall to close all the ports and accesses, it is not always the most effective solution to protect our network. It closes the doors but we still need a watchdog, monitoring everything that is happening on the network, checking if the accesses are trustworthy or if there are unknown connections, and when it happens, he can do something about it, like "biting" the intruders, or alerting his owner. Well then, in the IT world, that watchdog is known as Intrusion Detection System.

An Intrusion in IT is defined as an attempt to compromise the confidentiality, integrity, availability, or to bypass the security mechanisms of a computer network. This act of gaining access to a system normally leaves traces that can be detected by Intrusion Detection Systems. So, an Intrusion Detection System is an entity that scans all inbound and outbound network activity and identifies suspicious patterns that may indicate an attack (or intrusion) from someone attempting to access or compromise the system[29].

Intrusion Detection Systems contributes to any security infrastructure by providing a set of features:

- Prevents bad behaviors by increasing the risk of discovery and punishment for potential attackers;
- Detects attacks and other violations that are not prevented by other security measures;
- Deals with pre-attack activities;
- Documents the threats to an organization;

- Provides quality control for security design and administration;
- Provides useful information about intrusions, allowing improved diagnosis, recovery, and correction of vulnerabilities.

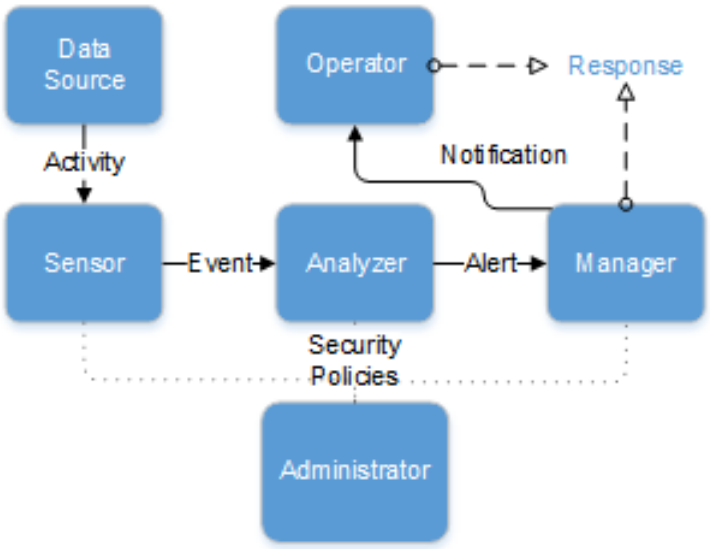


Figure 2.1: IDS Basic Model

We can see in Figure 2.1 the Basic Model of an Intrusion Detection, which contains the following components[36]:

- Data Source - The information that the IDS uses to detect attacks. It includes network packets, audit logs and checksum data. This data is captured by the sensor.
- Sensor - Collects the data from the Data Source and forwards events to the Analyzer.
- Analyzer - Component that detects the intrusions. It receives the events from the sensor and analyzes it looking for anomalies or patterns that represent a possible attack.
- Manager - Component used to configure all the components of the IDS, such as sensor configuration, analyzer configuration, event notification management, data consolidation, and reporting.
- Administrator - The human responsible for setting the security policies of the organization.
- Operator - The human that monitors the IDS and produce a response if needed.

The alarms generated by the IDS is an important topic. All the generated output is its responsibility, such as automatic responses, alerts of suspicious activity and user notifications. We must take into consideration that those alerts might not be conclusive or that the IDS can contain both configuration or analysis errors, which can generate false positives (alerts generated by the IDS in response to evidence of trustworthy activity). Those same errors can also generate false negatives, which is when the IDS doesn't generate an alert when a real attack happens. The perfect IDS wouldn't have false negatives and positives.

2.1.1 Architecture

The two main architectural components of the IDS are the Host, the system where the IDS software runs, and the Target, the system being monitored.

When the IDSs first appeared, they used a Host-Target Co-location architecture. It was too expensive to put the IDS running on a different machine, so it was running in the same system that it was monitoring. This presented an obvious security problem, as the attacker could just disable the IDS if he successfully hacked into the system.

A few years later, technology made pretty much everything cheap, so the IDSs started employing a Host-target Separation architecture, where the Host and Target were two different systems. This approach made it much easier to hide the existence of the IDS from the attackers.

2.1.2 Control Strategy

An IDS that monitors in the standalone mode is limited to its environment. On the other hand, an IDS that monitors in a distributed way is able to detect specific types of attacks that would be more difficult in a standalone mode.

2.1.2.1 Centralized

In this approach, the data gathered by the distributed sensors, is sent to the same centralized location, to be analyzed. The main advantage is having the data and the event analysis centralized. On the other hand, it creates a scalability problem, as the main server can get overloaded, fail or be attacked, compromising the IDS functionality.

2.1.2.2 Distributed

In the distributed strategy, both the data and the analysis are done in a distributed way. It solves the scalability problem, as well server failures and attacks. There is a few different distributed architectures:

- Hierarchical Analysis: This environment is composed by several IDS agents, installed in different network locations. When there is an alert, the agent will report to a higher level analyzer. The shortcoming of this approach is if the used topology converges to a single root node, it can still be overloaded or be the target of DoS attacks.
- Autonomous Agents: This is the most distributed approach. An autonomous agent is a program with learning ability that commonly uses artificial intelligence techniques or biological analogy. The advantage is that the agents can act as independent IDSs.
- Event Correlation: The goal of this type of IDS is to collect data from different sources and even other IDSs, and then correlate the data to detect malicious activity. This is an efficient mechanism to provide a wider scope of attack detection, that couldn't be done with isolated IDSs. The disadvantage is that it can be slow and inefficient if not optimised to deal with huge quantity of data.

2.1.3 Analysis Strategy

The way events are analysed in order to detect attacks can be classified into three main categories: Signature Detection or Misuse Detection, Anomaly Detection, and Specification-Based Detection.

2.1.3.1 Misuse Detection

In Misuse Based IDSs, also known as Signature Based, the information gathered is analyzed and compared to a large database of attack patterns, known as rules or signatures, looking for a match. Commonly, each signature corresponds to an attack. A simple example is when an attacker tries to connect to the system through a remote access, such as Secure Shell (SSH), and he enters the wrong password more than three times. That action will generate an authentication error message (a signature) on the logs, which will generate an alert by the IDS, informing the administrator and blocking the attacker's access to the system.

As opposing to Anomaly Based, in Misuse Based IDSs, it knows what the abnormal behavior of the system is, and accepts everything that is not considered abnormal. This of course, makes the system only as good as the database it uses to compare packets against. So, it fails to detect new attacks until they are added to the database.

Main Advantages:

- The detection is very efficient when comparing to anomaly detection, generating few false alarms;
- It can detect the use of a specific attack tool or technique.

Main Disadvantages:

- It only detects known attacks, i.e. those that have a corresponding signature on the database, so it needs to be updated continuously;
- Most of these detectors have very specific signatures, failing to detect variations of the same attack.

The network based IDS Snort[3] is one of the most widely deployed and used system that provides misuse based detection. It is focused on collecting packets as quickly as possible and processing them in its detection engine. It can detect many types of malicious activity in the packet payload that can be matched to a unique detection signature. By allowing the user to manipulate the rules database, Snort is a very useful tool as it can adapt to different environments.

USTAT[11] uses a different approach called state transition analysis. It considers that the computer initially exists in a secure state, but as a result of penetrations (modelled state transitions) it ends up in a compromised state. The audit trail produced by the computer is used as the source of information to compare the occurred state transitions with the intrusion specifications.

IDIOT[15] is another example. Its basic principle is to employ Petri-nets (a mathematical modelling language) for signature based intrusion detection. These signatures (or patterns) are written in textual language and parsed, resulting in a new pattern matching engine.

2.1.3.2 Anomaly Detection

In Anomaly Detection, the normal state and functionality of the system and the network is defined using heuristics or rules, and tries to detect any type of behavior that falls out of the normal system operation previously defined.

A classic example of this approach is when a specific user uses the Internet during a certain period of the day. The IDS collects this information during the week and creates a profile for that user, defining the use of the Internet during the say as normal behavior. If some day after this behavior is active, this user decides to use the Internet at night, the IDS will detect that this behavior is abnormal and block his Internet access. In this case, the IDS detected a different pattern in the behavior properly. However, we know that is actually a false positive, as the user should be allowed to use it when he wants.

As we can see, this type of detection, normally generates a lot of false alarms, as the system and user's behavior varies broadly. Despite this advantage, researchers claim that this IDSs can identify new attacks, while Misuse Detection Systems can't. Besides, the information gathered by the anomaly detection systems can be used as information source for the signature based detectors.

Although some commercial IDSs include some limited anomaly detection features, it is very rare to see one relying exclusively on this technology. The anomaly detection is still being extensively researched, so it might become very important in the IDSs development in the future.

The biggest challenge in this approach is teaching the system to recognize normal system activity, which can be accomplished in several ways, such as Neural Networks and mathematical models. The biggest shortcoming of this technique is the high rate of false positives, which is when legit traffic raise an alert.

Main Advantages:

- It detects unusual behaviors, so it has the ability to detect attacks without a previous knowledge of it;
- It produces data that can be used to create signatures for the IDSs based on signatures.

Main Disadvantages:

- It produces a large number of false alarms, due to the unpredictable behavior of the users;
- It requires a huge amount of data collection sessions, in order to create a normal behavior pattern.

IDES[20] is one of the oldest and best documented intrusion detection system. It uses as motivation the fact that users behave in a consistent way from time to time, when they perform activities on the system, and it uses that to calculate a set of statistics for the user behaviour. It can then compare the current activity happening on the system with the calculated profile, and flag any deviation as an possible intrusive behaviour.

PAYL[35] is a payload-based anomaly detector that tries to address this issue, by using a technique that computes a profile byte frequency distribution and the standard deviation of the application payload during a training phase, and then uses an operation called Mahalanobis distance during the detection phase to calculate the similarity of new data against the stored profile.

Haystack[30] was developed for the detection of intrusions in the US Air Force computer system. It employs both anomaly detection and signature based detection. The anomaly detection has two main concepts; models of how users have behaved in the past, and predefined generic user group models specifying the acceptable behavior of each group. Many of the problems of each approach is solved by combining the two.

2.1.3.3 Specification based

In order to combine the strengths of anomaly and misuse based detection, specification based techniques have been proposed. In this approach, legitimate program behavior is characterized by manually developing a specification. So, when unusual behavior is encountered, an alarm will only be generated if the program behavior is not legitimate, meaning that a low rate of false alarms will be obtained. Because of this ability to detect deviations from legitimate behavior, it is capable of detecting previously unknown attacks.

Main Advantages:

- If designed by an expert, it presents a low number of false positives;
- Attacks can be contained, even if the administrator doesn't know the source code of the application being attacked.

Main Disadvantages:

- The amount of specifications checks can create bottlenecks and make the system slow;
- Some types of attacks cannot be specified with the existing specification languages.

A model (or specification) based IDS proposed in [6] is an example of a system using specification based techniques. It was designed to protect SCADA networks that exist in critical infrastructures, by detecting attacks that cause violations of the models characterizing the expected behavior of the system.

2.1.4 Information Sources

2.1.4.1 Network Based

In the Network Based IDSs, also known as NIDS, the network packets are analyzed looking for signs of attacks, network misuse and anomalies, which could indicate that an intrusion is in progress. It can be installed in two modes: inline and passive. In the inline mode, it is installed in a location that intercept the network flow, acting as a bridge, so it can capture the packets and analyze them to detect intrusions. In the passive mode, the NIDS is connected to a switch that forwards copies of the network packets to be analyzed.

Main Advantages:

- When installed in the passive mode, it become invisible to the attackers;
- It is platform independent and it doesn't interfere with the hosts' performance.

Main Disadvantages:

- It struggles when dealing with high-speed networks;
- It is not able to analyze encrypted data.

One of the most used Network Based IDSs is Snort[3], a free and open source software that has the ability to perform real-time traffic analysis and packet logging on IP networks. A different NIDS developed by the University of Ohio, is INBOUNDS[33], which uses anomaly based analysis. The main task is analyzing TCP connection's timers and its size.

2.1.4.2 Host Based

Also know as HIDS, the Host based IDS is a system that monitors a single host to detect suspicious activity. It gathers information from two sources: Operating System's audit trails and system logs. The audit trails are normally produced by the kernel, so they are more detailed and protected than the system logs. On the other hand, the system logs are smaller and easier to understand.

Main Advantages:

- It can detect attacks through local events, which cannot be detected by the network;
- It can analyze encrypted data;
- It is not affected by network switches.

Main Disadvantages:

- It is difficult to install and maintain;
- The HIDS itself can be an attack target;
- Detecting network attacks is a very difficult task;
- It can interfere with the host's performance.

Two widely used HIDS are OSSEC[10] and Tripwire[14], both free and open source. OSSEC is capable of performing a set of functionalities, such as log analysis, rootkit detection and time-based alerting, amongst others, and it provides intrusion detection for most operating systems. Tripwire focuses on monitoring and alerting on file changes on a range of systems. This is done by comparing scanned files with older stored files contained in a database.

2.1.5 Snort

Like introduced before, Snort is an open source Network Intrusion Detection system, very popular for its flexibility in the configuration of the rules. It has the biggest repository of rules, it is lightweight, small, and can detect anomalies in the system network.

There are three primary subsystems that make up Snort: the packet decoder, the detection engine, and the logging and alerting subsystem.

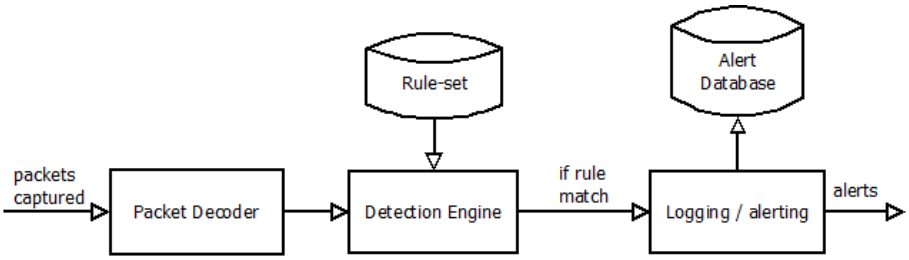


Figure 2.2: Snort Overview

Packet decoder. In this phase, the captured packets are decoded so that they can be used in the next phase. The decoding routines are called in order through the protocol stack up to the application layer. Most of the functionality of the decoder consists of setting pointers into the packet data.

Detection Engine. This is the phase where the rules will be matched with the data decoded in the previous phase. The rules are maintained in a two dimension linked list, and they are recursively searched for each packet in both directions in order to find a match.

Logging / alerting. When there is a match in the detection phase, the alerting and logging subsystem is selected at run-time with command line switches. It can be con-figured to log packets in a decoded, human readable format, allowing for a fast analysis of data collected by the system. It can also send the alert to the system administrator using syslog.

2.2 Critical Infrastructures

As already described, “Critical Infrastructures” is a term used to describe assets that are so important, that if they were destroyed or incapacitated, it would have a devastating effect on security, economy, public health or safety.

To assist in their operation, these Critical Infrastructures use control systems, such as the Supervisory Control and Data Acquisition (SCADA). This system provides auto-mated control and remote human monitoring of real world processes. It also offers near real time monitoring, with time delays than can go from seconds to minutes and it may cost thousands or even millions of dollars, depending on their size and sophistication [12].

2.2.1 SCADA

The common scenario of a SCADA system, is collecting the information and transferring it to the central site. When this information is obtained, it is time to analyze and control it, followed by displaying it in the operator screens. After that, the control actions need to be passed back to the process so it can continue working properly.

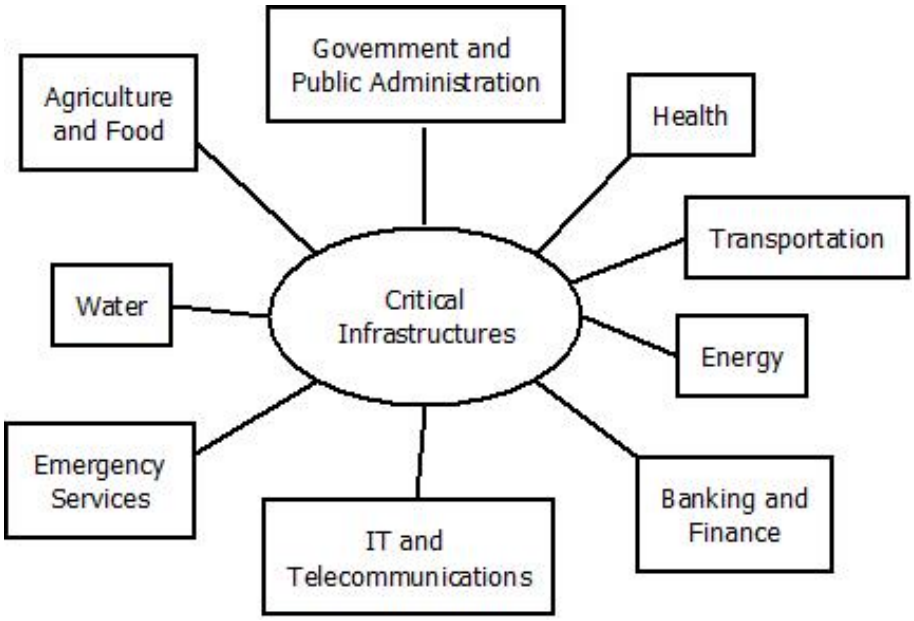


Figure 2.3: Main Critical Infrastructures.

2.2.1.1 SCADA Components

Human-Machine Interface (HMI). Where the collected data is processed and presented to be analyzed by a human operator. This information is presented graphically, in the form of a mimic diagram so that the operator can see a schematic representation of what's going on. Another important aspect is alarm handling. The system monitors whether certain conditions are met, and activates the alarm so that the operator can be aware of it and take proper action.

Software. Links the HMI and SCADA's databases, so it can provide trending, diagnostic data and management information such as scheduled procedures, logistic information and troubleshooting guides.

Hardware. SCADA's systems normally include Distributed Control Systems components, but the use of Remote Terminal Units (RTUs) and Programmable Logic Controller (PLCs) have also become common, as they are capable of executing simple logic processes autonomously.

- Remote Terminal Units connect to sensors and converts the electric signals to digital values. It sends this signals to the supervisory system and also received commands from it, by using telemetry hardware that are built into them.

- Programmable Logic Controllers have the same function as RTUs but they more economical, versatile, flexible and configurable.

  Supervisory Station. This component is the link between the HMI software running on workstations and the hardware (PLCs, RTUs, Sensors, etc.). It can be a single machine, or it may include multiple servers, distributed software applications, and disaster recovery sites.

2.2.1.2 SCADA Threats

The existing threats in SCADA systems can be divided in two groups:

- **Non-Intentional**
  - Accidents
  - Natural disasters
  - Equipment failure
- **Intentional**
  - Malware: Viruses, worms, Trojan horses, malicious mobile code, blended at-tacks, spyware tracking cookies and attacker tools such as backdoors and root-kits. A program that is inserted into a system with the intent of compromising the confidentiality, integrity, or availability of the data, applications, or operating system[21].
  - Insider: A worker that might have motivation to damage or disrupt the system. They may also attempt to illicitly gain higher privileges for their convenience. Engineers may also make errors that cause problems to the system.
  - Hacker: An outsider that gains access to the system illicitly with the goal of probing, intruding or controlling it just for the challenge, or for getting confidential information.
  - Terrorist: Someone that desires to do disable or take control of the SCADA systems, either for disrupting it or taking advantage of it to execute a malicious action that will harm the functioning of the critical structure and affect the society.

2.2.1.3 SCADA Vulnerabilities

The IT world has seen an exponential growth in the number of vulnerabilities existing in common distributed systems. This growth poses a key challenge, as everyone must be aware of their existence and find a way to prevent it.

A traditional way of dealing with vulnerabilities is creating patches, updates or fix-es. However, this way cannot be applied in SCADA systems because these systems are bundled packages, preventing the end-user from knowing what's really inside.

Like traditional IT systems, SCADA systems also have a set of vulnerabilities that exist in their environment[24][12]:

- **Public Information.** When security was not a priority, some of SCADA's system owners published papers on the design of the system, allowing everyone to find vulnerabilities. Same happened with consultants that end up revealing information about their work on SCADA systems.
- **Wireless.** SCADA systems use forms of communication that can be vulnerable to some kind of attacks, such as microwave, data radios and cellular packet services.

- **Authentication.** For convenience purposes, SCADA system commonly have shared passwords. This of course removes all sense of authentication and accountability. A two-factor authentication could help solve this problem by adding an inherence factor but it is limited in some cases because the workers might have dirty hands or might be using equipment that covers the zone to be scanned.

- **Physical Security.** SCADA systems are distributed over long distances, making it impossible to watch over every square meter. These unstaffed locations have locks but it is obviously not enough to provide proper security.

- **Remote Processors.** The computation power and memory resources of the processors are weak, making it unsuitable for upgrades. Because of that, it stays in place for many years, resulting in vulnerable equipment for a long time.

- **Monitoring and Defenses.** Intrusion Detection System are not common in SCADA systems. Also, the firewalls and anti-virus software is not universal, together with the lack of time by the staff to review logs, makes the system vulnerable to zero-day worms.

- **Staff Experience.** Reliability and Availability have been the most important concerns of the staff regarding the system operation, making security policies a foreign concept for them. Because of that, SCADA staff might also be receptive to IT staff recommendations.

- **SCADA Software.** The software that SCADA systems uses have weak security features and has a lot of design vulnerabilities.

- **Operating System Vulnerabilities.** Like normal IT systems, SCADA system also have the same operating system problems present in this kind of systems. The difference is that, unlike normal IT systems, SCADA systems cannot be patched as easily because they are expected to run without interruptions.

- **Interconnections.** Economic and enterprise pressures result in internal connections between SCADA network and business network resulting in an increase of exposure and vulnerabilities present in SCADA systems.

- **Remote Access.** It is not uncommon for SCADA system to be configured remotely, either by dial-up access or VPN access over the Internet, which poses obvious security vulnerabilities.

2.3 Approximate String Matching

We use regular String Matching when we want to check if two string are exactly equal, but sometimes we want to compare two string that are really similar but not the same, and we still want to get a match, as it represents the same real-word entity. This is important because strings referring to the same real-word entity are often different, due to typing errors, different formatting conventions, abbreviations, etc.

Approximate String Matching solves that problem using a similarity measure, by calculating the similarity between the strings and return a match if that similarity is over a pre-specified threshold.

2.3.1 Techniques

Different Similarity Measure techniques have been created to calculate the similarity between strings:

- Sequence-based: Views the strings as sequence of characters, and computes the cost of transforming one string into the other.
- Set-based: Views the strings as sets or multi-sets of tokens, and uses set-related properties to compute similarity scores.
- Hybrid: Combines the benefits of sequence-based and set-based methods.
- Phonetic: Matches the strings based on their sound.

Levenshtein Distance[17] (also known as Edit Distance) was one of the first string matching algorithms and is still one of the most used. It is defined by the minimum number of operations (insertions, deletions or replacements of characters) needed to transform one string into another. Given two string x and y and their edit distance, denoted by d(x,y), the similarity function is given by s(x,y) = 1 - d(x,y) / max(length(x),length(y)). Regarding the performance, the time complexity is O(|x||y|), while the space complexity is only O(min(|x||y|)), where |x|and |y|are the length of string x and y. Despite not being very efficient, it is one of the most flexible, as it can adapt to several distance functions.

Smith Waterman[31] is another algorithm very similar to Levenshtein's, with only a few differences. It aligns substrings instead of strings. It introduces some changes in the scoring system, allowing the end and beginning of the two strings to be misaligned. The performance is similar to Levenshtein's, and one advantage is allowing two strings with similar prefixes to have a good similarity score.

A similar approach is used in the Hamming Distance[23] algorithm. To compute the similarity, it only considers substitutions, meaning that the score will be the number of characters that must be substituted in a string to become equal to another string with the same length. This algorithm is very efficient, but because it doesn't allow insertions or deletions, it only works well in some situations.

Soundex[9] is different from all the previous algorithms. It translates the string into a Soundex Code, which is formed by a letter and three digits. If two strings are translated into the same code, they are considered similar. The idea is to match strings that are often spelled in different ways but sound the same when we say them. It has a good performance, as it is executed in linear time, but it should only be used to check phonetic errors, because it will produce bad results in different applications. Another problem is the fact that it was developed for the English language, meaning that it would need to be adapted to work with different languages.

2.4 Web Scraping

Web Scraping is a technique used to extract data from websites. The idea is to analyze the structure of the page to identify data patterns and then we apply scraping to get it, automatically. Web Scraping scripts and applications simulate a person viewing the website with a browser. It starts by sending an HTTP request to the website, so it can obtain the source code of the required page, and then extracts specific data based on tags, class or ids patterns and some other elements present in the page.

The most common tasks of a Web Scraper are the following:

- Connect to the data source: To establish communication with the target website, it uses the HTTP protocol, through the GET and POST methods.
- Parsing the HTML content: In this step, the scraper extracts the contents of interest. A common technique used to accomplish that, is regular expression matching, which should be made as general as possible, to better adapt to changes in the HTML document. An alternative is the use of HTML parsing libraries.
- Output building: Once the scraper has extracted the data, it then transforms it into a structured representation, to be analysed and stored.

2.4.1 Web Scraping Techniques

Despite being a recent field, there has been created some different techniques to execute Web Scraping. The current solutions range from ad-hoc, to fully automated systems able to produce structured documents from the websites.

- Regular expression and text grepping: This is one of the most simple, yet very powerful techniques. It can be applied using the UNIX grep command or by using regular expressions resources from programming languages, such as Perl and Python.
- HTTP Programming: This techniques uses socket programming to post HTTP requests and retrieve information, allowing clients to get very accurate data.
- HTML Parsers: Includes semi-structured data query languages, like XQuery and HTQL, that allow retrieving and transformation of web content.
- DOM Parsing: This is a hierarchy-based parser that creates an object model of the entire document. It breaks down the document into elements, attributes and data.

2.4.2 Web Scraping Tools

2.4.2.1 JSoup HTML Parser

JSoup [21] is an open source library that allow us to work with the HTML code of web pages. It parses HTML and provides an API for extracting and manipulating data.

JSoup allows us to:

- Scrape and parse HTML from a URL, file, or string;
- Find and extract data, using DOM or CSS;
- Manipulate the HTML elements, attributes, and text;
- Clean content to prevent XSS attacks;
- Output tidy HTML

For our solution, we are interested in the data extraction feature. For example, to extract the first paragraph of a Wikipedia article, we can use the following:

```
String url = "http://en. wikipedia.org/wiki/Critical_infrastructure";
Document doc = Jsoup.connect(url).get();
Elements paragraphs = doc.select(".mw-content-ltr p");
String firstParagraph = paragraphs.first().text();
```

The JSoup library can be easily integrated with JAVA, allowing us to create a java program to extract the data we need, with low development effort.

Chapter 3

Solution Description

In this section we will present the description of the developed solution. Out motivation was integrating it with IDSs running on Critical Infrastructures systems, but it can be applied to any IDS.

3.1 Architecture Overview

As stated in the introduction, our goal was to develop a solution capable of detecting which applications are installed on a given host and then matching those applications with the rules being used in the misuse detection component of an intrusion detection system. This will allow us to detect what rules affect the host, so that we can keep only those, and exclude the others. It is expected that the false alarm rate will be reduced, as the intrusion detection sensors will be restricted to the applications being used.

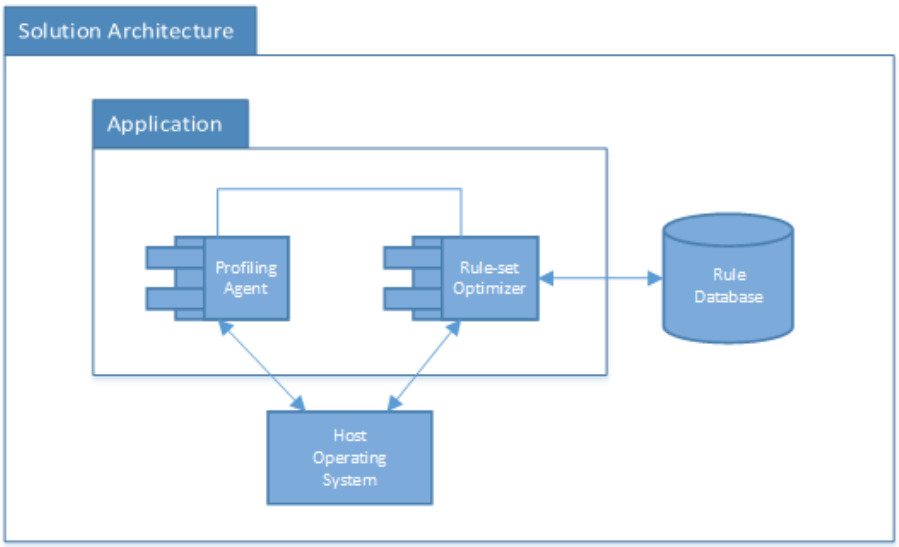


Figure 3.1: Solution Architecture Overview

Our solution contains two main components, as shown in figure 3.1:

- Profiling Agent;
- Rule-set Optimizer.

Our modules communicate with the host operating system where the application is running, and with a database, that can be local or external.

3.2 Profiling Agent

The Profiling Agent is the first component of our application. It is responsible for producing a list with all the applications installed on the host where it is being executed.

The following assumptions were taken when designing this component:

1. All the applications used on the host, were installed properly through the operating system installation services.
2. Only applications that are properly installed, are executed on the host. No application is executed from external media.

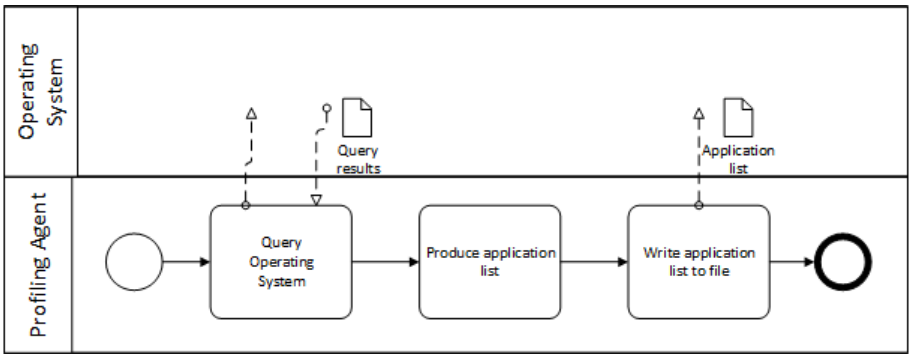


Figure 3.2: Profiling Agent Activity Diagram

As we can see in Figure 3.2, the Profiling Agent has three main tasks.

As already stated before, we want to know which applications exist in the host. If those applications are installed, it means the operating system is aware of that, so there must be a place somewhere in the OS where that information is registered. Our first step will be to retrieve that information, by communicating directly with the OS.

Our component was designed to work with Windows and Ubuntu platforms, meaning we have different ways of communicating with the host OS, depending on its platform:

- On Windows platforms, we get this information from the registry, which is a database that contains information about installed programs and settings, system hardware and user profiles. The communication with the registry is through the command-line utility reg.exe, which allows us to perform operations on registry subkey information and values. When an application is installed

properly, through the Windows Installer, the OS creates a registry entry in the Uninstall Registry Key, meaning that all we have to do is query that registry key to get the information we need.

The applications can either be installed for just one user, or for all users, and it can either be a 32-bit or a 64-bit application. Therefore, we use the command "reg query <KeyName>" to get all the subkeys of the Uninstall Registry Key, in four different paths:

- HKCU\Software\Microsoft\Windows\CurrentVersion\Uninstall
- HKCU\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall"
- HKLM\Software\Microsoft\Windows\CurrentVersion\Uninstall
- HKLM\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall"

  For each application, it returns all the entries registered, such as the name, version, publisher, location, etc.

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\Currentversion\Uninstall\Office
15.PROPLUS
   Publisher    REG_SZ    Microsoft Corporation
   DisplayIcon    REG_SZ    C:\Program Files\Common Files\Microsoft
Shared\OFFICE15\Office Setup Controller\OSETUP.DLL,1
   DisplayName    REG_SZ    Microsoft Office Professional Plus 2013
   DisplayVersion    REG_SZ    15.0.4569.1506
   InstallLocation    REG_SZ    C:\Program Files\Microsoft Office
   ModifyPath    REG_SZ    "C:\Program Files\Common Files\Microsoft
Shared\OFFICE15\Office Setup Controller\setup.exe" /modify PROPLUS /dll
OSETUP.DLL
   UninstallString    REG_SZ    "C:\Program Files\Common Files\Microsoft
Shared\OFFICE15\Office Setup Controller\setup.exe" /uninstall PROPLUS /dll
OSETUP.DLL
   ProductID    REG_SZ    00216-40000-00000-AA185
```

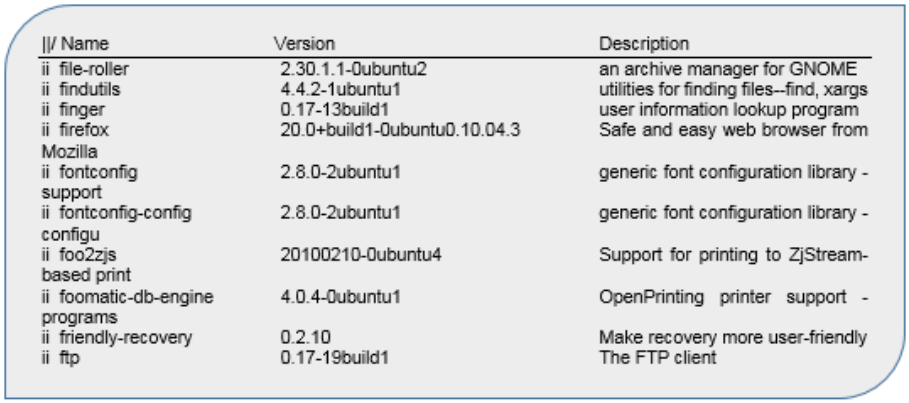
Figure 3.3: Sample Windows Registry Entry

In Figure 3.3 we can see a sample registry entry of the Microsoft Office Professional Plus 2013 application installed on the machine. The reg query commands returns one of this entries for each application installed.

- On Ubuntu, the procedure is similar. Instead of the registry, we have the Package Management System, which is a collection of tools that provides an automatic way of installing, updating, configuring or removing system packages. To communicate with the Package Manager, we use the tool dpkg with the option "-l", which lists information about all the installed packages, including name, version and description. We can see an example in Figure 3.4 displayed below.

At the end of the first step, we have a list with several information regarding all the installed applications in the host.

In the second step we will retrieve only the relevant information, which in this case is the name of the application. In both platforms we receive a list with information from our query, which contains the name of the application, and then we extract it with the help of regular expressions.



|/ Name	Version	Description
ii file-roller	2.30.1.1-0ubuntu2	an archive manager for GNOME
ii findutils	4.4.2-1ubuntu1	utilities for finding files--find, xargs
ii finger	0.17-13build1	user information lookup program
ii firefox	20.0+build1-0ubuntu0.10.04.3	Safe and easy web browser from Mozilla
ii fontconfig	2.8.0-2ubuntu1	generic font configuration library - support
ii fontconfig-config	2.8.0-2ubuntu1	generic font configuration library - configu
ii foo2zjs	20100210-0ubuntu4	Support for printing to ZjStream-based print
ii foomatic-db-engine	4.0.4-0ubuntu1	OpenPrinting printer support - programs
ii friendly-recovery	0.2.10	Make recovery more user-friendly
ii ftp	0.17-19build1	The FTP client

Figure 3.4: Section from dpkg -l output

Once we have all the names of the installed applications, we move to the last step, which is writing all this information to a file, concluding the Profiling Agent phase. For compatibility reasons, the use the .txt file format which is considered universal or platform independent.

The file generated by the Profiling Agent is going to be used by the Rule-set Optimizer, so it is crucial that it contains all the applications.

3.3 Rule-set Optimizer

Now that we know which applications are installed on the host, we will move to the Rule-set optimizing phase. In this phase, we will transform the Original Rule-set into an Optimized Rule-set, containing only the rules that affect applications installed in our host.

Although we developed a module to scan for all the applications installed on the host, our Rule-set Optimizer Module will also accept a manually inserted Host Image, if the user so intends to.

The following artifacts will be part of our module's input and output:

- Original Rule-set: List with all the rules being used on the misuse detection component;
- Optimized Rule-set: Optimized version of the Rule-set that contains only rules related with the applications running on the system;
- Host Image: List with all the applications running on the host.

As for our solution, it will be composed of 3 main phases, as shown in Figure 3.5

- Rule Analyzing Phase;
- Rule Matching Phase;
- Rule-set Generation Phase;

The following assumptions were taken when designing this component:

1. The rule information database is always available.
2. The database has information about all the rules.

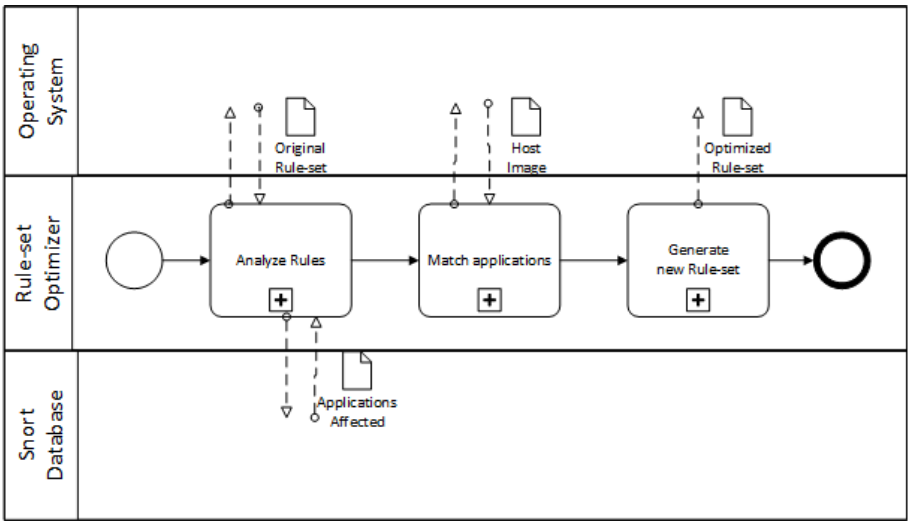


Figure 3.5: Rule-set Optimizer Activity Diagram

3.3.1 Rule Analyzer

This is the first phase of our Rule-set Optimizer module. In this phase, we will collect relevant information about each rule present in the original Rule-set, used by the misuse component of the IDS. The original Rule-set is obtained from one or more text files in the .rules format, present in the host's file system. The path of the .rules file or the directory containing the files, must be supplied by the user.

Rules are meant to detect a vulnerability or exploit, and we are interested in knowing the applications that it affects.

The module was designed to work with different rule formats, but for demonstrating purposes, we will only consider the Snort[3] Rules' format.

```
alert tcp $EXTERNAL_NET any -> $SQL_SERVERS 1433 (msg:"SQL
raiserror possible buffer overflow"; flow:to_server,established;
content:"r|00|a|00|i|00|s|00|e|00|r|00|r|00|o|00|r|00|"; fast_pattern:only;
metadata:ruleset community; reference:bugtraq,3733;
reference:cve,2001-0542; reference:nessus,11217; classtype:attempted-
user; sid:1387; rev:13;)
```

Figure 3.6: Snort Rule

A Snort Rule has several keywords, but we are only interested in the sid keyword, which is the keyword used to uniquely identify Snort rules. In the example shown in Figure 3.6, the sid is 1387. This keyword exist in all the Snort rules, so that is what we will use to get information about the rule. To accomplish this, we will look at the rule documentation available on the Snort's Database, which can be accessed through the following URL:

https://snort.org/rule_docs/1387

This will return a page with all the documented information about the rule, such as a summary, impact, affected systems, etc., but as we mentioned before, we are only interested in the affected systems, which is the only thing that we will extract.

Figure 3.7: Section of the page returned by the url https://snort.org/rule_docs/1387

In the example shown in Figure 3.7, we are only interested in extracting the “Microsoft SQL Server 7.0” and “Microsoft SQL Server 2000” strings, so we can compare with the applications listed in our Host Image produced by the Profiling Agent.

To communicate with Snort’s Database, we use the web scraping technique HTML Parsing, provided by the library JSoup [1]. We use JSoup to parse the page into a DOM (Document Object Model), so that we can easily access each node in the DOM tree. For security purposes, the communication with the Snort’s Database is done under the HTTPS protocol.

```
<h3>Affected Systems</h3>
<ul >
<li>Microsoft SQL Server 7.0</li>
<li>Microsoft SQL Server 2000</li>
</ul>
```

Figure 3.8: Source code of the page section present in Figure 3.7

In the example shown in Figure 3.8, we simply look for the textual content in the node “li” under the node "h3" containing the text “Affected Systems”.

As mentioned before, this is the procedure executed to extract information about snort rules, which was done using an external data source. When dealing with different rule formats, this data source can be also be local. We also need to add new methods to communicate with this new data source and deal with this format.

At the end of this phase, we have a list of all the applications affected by the analyzed rule.

3.3.2 Rule Matching

Now that we have both the list of installed applications on the host, and the list with applications affected by each rule, we will compare them so see if they match.

This is a very important step in our component, because if we don’t match applications that are supposed to be matched, we will be deleting rules that are needed by the IDS. We need to make sure

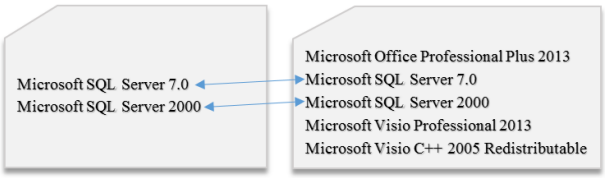


Figure 3.9: Matching of the affected applications (left) with the applications present the Host Image (right).

that we are only deleting rules that don't affect absolutely anything in the host.

The name used by Snort in the affected systems section of the page is not standardized, meaning that it could have syntax errors or could be written in a different way, like having the words in a different order. Because of that, a simply string match is not enough, so we decided to use a fuzzy string matching, which is used to detect similar strings.

To match the applications, we will consider that:

- An application contains one or more words;
- A word contains one or more characters.

So, for every entry <affected> in the Affected Systems section of the Snort's Database, and every application <app> in the Host Image, we will consider it a match if <affected> contains at least 60% of <app>'s words. We chose 60% as our threshold because as explained before, the data on the Snort's Database is not standardized, meaning that an application composed of several words, could be referred to as only one or two. The Host Image is represented as a file that our module reads from the OS's file system. This file can either be produced from the Profiling Agent, or a file chosen by the user,with manually inserted data, for example.

```
foreach String affected in affectedList do
    foreach String app in appList do
        if (matches(affected,app) / app.lenght) > 0.6 then
            return true;
        end
    end
end
return false;
```

Algorithm 1: Application Matching algorithm

To check if two words match we use the Levenshtein Distance Algorithm, represented as matches() in Algorithm 1 and defined in Algorithm 2

```
foreach String aff in affected do
    foreach String a in app do
        if (1 - LevenshteinDistance(affected,app) / aff.lenght) > 0.8 then
            totalMatches+=1;
        end
    end
end
return totalMatches
```

Algorithm 2: Word Matching Algorithm

If there is a match, it means that the rule is needed, so it cannot be deleted, and will therefore be marked as “matched”.

When there is not enough information to conclude if the rule is related, we will consider it a match, because it is obviously better to keep a rule that is not needed than removing a rule that could be needed.

At the end of this phase, we will have a list with all the rules that had a match, which will be used to generate the Optimized Rule-set in the next phase.

3.3.3 Rule-set Generation

Finally, in the 3rd phase of the module, we will generate the new Rule-set containing only the rules that affect the services running in the system where the IDS is running. We will look at all the rules analyzed in the previous phases and create a new list containing only the rules that have been marked as “matched” on the matching phase.

Figure 3.10: New Rule-set generation.

This Optimized Rule-set is then written in the file system, ready to be used by the IDS.

Chapter 4

Solution Evaluation

4.1 Assessment Methodology

In order to provide a good assessment and to get a clear picture of the quality of the developed work, we propose a Quantitative Assessment method.

We believe this is a good method to evaluate our solution as produces a statistical evaluation, which can be expressed in numbers or percentages, allowing us to evaluate the effectiveness of the solution.

As the solution has one clear single goal, which is optimizing a rule-set, we will calculate the efficiency and accuracy of that optimization. We will test our module with a sample rule-set and check if it detects properly what rules should be kept for the optimized rule-set. This will be dome by executing the module and then manually analyzing the results.

We will consider efficiency, the ability to delete all the rules that are not related with the host. It is the most important assessment as it indicates the percentage of optimization that is applied to the rule-set. It can be obtained from the following expression:

$$Efficiency = \frac{totalrulesdeleted}{totalrules} \tag{4.1}$$

where <total rules deleted>is the number of rules that did not have a match, and therefore not used to create the optimized rule-set, and <total rules>is the total number of rules existing in the original rule-set.

As for the accuracy, it will be the ability to detect the rules that are actually related to the host, from those that were matched. The accuracy and the efficiency are related. The higher the accuracy, the higher efficiency. We will use the following expression:

$$Accuracy = \frac{totalrulesrelated}{totalrulesmatched} \tag{4.2}$$

where <total rules related>is the number of rules that are actually related, and <total rules matched>is the number of rules that were matched.

In the next section, we will present the results of the solution, as well as the process followed to obtain it.

4.2 Quantitative Evaluation

Our tests were executed in a host with the Windows 7 operating system installed. We used as host image, the application list produced by the Profiling Agent, which contained 320 entries, and the original rule-set contained 5615 rules.

The following results were obtained:

total rules	total rules matched	total rules related	total rules unknown	total rules not related
5615	1096	656	438	2

Table 4.1: Tests results

- Total rules: The total number of rules existing in the original rule-set;
- Total rules matched: The total number of rules that were matched;
- Total related: The total number of rules that were matched and that are actually related with the host;
- Total rules unknown: The total number of rules that were matched, but there's not enough information to conclude if they are related;
- Total rules not related: The total number of rules that were matched, but are not related with the host.

4.2.1 Efficiency

From our results shown in Table 4.1, we can see that from our total number of 5615 rules, only 1096 were matched, meaning that 4519 were removed. We can then calculate the Efficiency of our solution:

$$Efficiency = \frac{4519}{5615} = 0,80 \tag{4.3}$$

This result shows that only 20% of the rules were kept to the new rule-set, corresponding to an 80% improvement from the original rule-set.

4.2.2 Accuracy

From the Table 4.1, we can also see that the total number of rules related is 656 and the total number of rules matched is 1096, so we can calculate the Accuracy:

$$Accuracy = \frac{656}{1096} = 0,60 \tag{4.4}$$

This result means that from the matched rules, we are only sure that 60% of them are related with the host. This value is not higher because we are playing safe in the matching phase, for security reasons. When we are not sure if a rule is really related or not, we consider it a match because we believe it is best to keep a unrelated rule than removing a related one.

4.3 Evaluation Discussion

The results have shown that we were able to obtain an optimization of 80%. All the unrelated rules were successfully removed, meaning that the IDS will stop throwing false alarms regarding applications that are outside of the host's scope. It will also help the IDS administrator managing the rule-set as he will have less rules to maintain.

The results are influenced by the target IDS considered to optimize the rules. As mentioned before, our results were obtained using Snort rules. Since the information about the rules, existing in the Snort's database is not standardized, we were not able to obtain a better efficiency, without removing related rules. We believe that, if we test our solution with a standardized information source about the rules, we will be able to increase the Accuracy to values close to 100%, and also increase the global efficiency of the solution.

Chapter 5

Conclusions

5.1 Achievements

The objective of this thesis was to develop and evaluate a tool capable of optimizing a given rule-set from an Intrusion Detection System.

We have presented a solution containing profiling and rule-set optimization modules and we believe that, although simple, our proposed solution was able to accomplish the proposed objectives. We couldn't find any research work proposing a similar system so it was a big challenge designing everything from scratch. Different techniques were used to produce our tool, like approximate string matching and screen scrapping techniques, amongst others.

To validate the effectiveness of our solution, we applied our tool to a rule-set containing Snort rules. As the information about the rules contained in the Snort's database is not standardized, we expected that an optimization close to 100% was not going to be very likely to happen.

From our tests, we were able to conclude that the main objective was accomplished. We obtained an optimization of 80%, showing that there is still room for improvement. The results depends on how the information source displays the information. We believe that we will obtain a higher efficiency when optimizing a rule-set from a different IDS containing more standardized information about the applications affected by each rule.

We also conclude that the number of false positives produced by the IDS will be reduced when using our optimized rule-set, as the unrelated rules will no longer generate alarms.

5.2 Future Work

Despite being able to successfully accomplish the proposed goal, there are still a few things that could be further studied.

The Profiling Agent only captures properly installed applications. It would be interesting to have a dynamic Profiling Agent that looks at executed applications instead of installed. It could be done by scanning the running processes in real time and updating the list of applications when a new one is

found.

The Snort rule-set used in our solution was a good source to evaluate the effectiveness of our solution, but it would help us getting a better idea, if we test it with different rule formats and information sources.

Bibliography

[1] jsoup - Overview. http://jsoup.org/apidocs/.

[2] SECUR-ED - About us.

[3] SNORT Users Manual - 2.9.6. http://manual.snort.org/.

[4] The Council of the European Union: Council Directive 2008/114/EC of 8 December 2008 on the identification and designation of European critical infrastructures and the assessment of the need to improve their protection, 2008.

[5] R. Bace and P. Mell. NIST Special Publication on Intrusion Detection Systems, 2001.

[6] S. Cheung, B. Dutertre, F. Martin, U. Lindqvist, K. Skinner, and A. Valdes. Using Model-based Intrusion Detection for SCADA Networks, 2006.

[7] M. Drahansky. A Study on The Security Vulnerabilities in Control Systems, Critical Infrastructure Systems and SCADA, 2008.

[8] D. Glez-Peña, A. Lourenço, H. López-Fernández, M. Reboiro-Jato, and F. Fdez-Riverola. Web scraping technologies in an api world. *Briefings in bioinformatics*, 15(5):788–797, 2014.

[9] P. A. Hall and G. R. Dowling. Approximate string matching. *ACM computing surveys (CSUR)*, 12(4):381–402, 1980.

[10] A. Hay, D. Cid, and R. Bray. OSSEC host-based intrusion detection guide. Syngress, 2008.

[11] K. Ilgun, R. Kemmerer, and P. Porras. State transition analysis: a rule-based intrusion detection approach. *IEEE Transactions on Software Engineering*, 21(3):181–199, Mar. 1995.

[12] S. Institute. Security for Critical Infrastructure SCADA Systems, 2005.

[13] M. Keefe. Timeline: Critical infrastructure attacks increase steadily in past decade. 2012.

[14] G. H. Kim and E. H. Spafford. The design and implementation of Tripwire: A file system integrity checker. pages 18–29. ACM Press, 1994.

[15] S. Kumar and E. H. Spafford. A pattern matching model for misuse intrusion detection. In *In Proceedings of the 17th National Computer Security Conference*, pages 11–21, 1994.

[16] S. Lance. *Honeypots: Tracking Hackers*. Addison Wesley, 2002.

[17] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[18] Z. Li, M. Sanghi, Y. Chen, M.-Y. Kao, and B. Chavez. Hamsa: fast signature generation for zero-day polymorphic worms with provable attack resilience. pages 15 pp.–47, May 2006.

[19] J. Lima. *Specification-based intrusion detection*. PhD thesis, Instituto Superior Técnico, 2012.

[20] T. F. Lunt, R. Jagannathan, R. Lee, S. Listgarten, D. L. Edwards, P. G. Neumann, H. S. Javitz, A. Valdes, T. F. Lunt, R. Jagannathan, R. Lee, S. Listgarten, D. L. Edwards, P. G. Neumann, H. S. Javitz, and A. Valdes. Ides: The enhanced prototype - a real-time intrusion-detection expert system. Technical report, SRI International, 333 Ravenswood Avenue, Menlo Park, 1988.

[21] P. Mell, K. Kent, and J. Nusbaum. Guide to Malware Incident Prevention and Handling - Recommendations of the National Institute of Standards and Technology, 2005.

[22] M. Mohammed, H. Chan, N. Ventura, M. Hashim, I. Amin, and E. Bashier. Accurate signature generation for polymorphic worms using principal component analysis. pages 1555–1560, Dec. 2010.

[23] G. Navarro. *Approximate text searching*. PhD thesis, University of Chile Santiago-Chile, 1998.

[24] U. S. G. A. Office. CRITICAL INFRASTRUCTURE PROTECTION - Challenges in Securing Control Systems, 2003.

[25] G. Portokalidis, A. Slowinska, and H. Bos. Argos: An Emulator for Fingerprinting Zero-day Attacks for Advertised Honeypots with Automatic Signature Generation. EuroSys '06, pages 15–27, New York, NY, USA, 2006. ACM.

[26] R. J. Robles and M.-k. Choi. Assessment of the Vulnerabilities of SCADA, Control Systems and Critical Infrastructure Systems, 2009.

[27] R. J. Robles, M.-k. Choi, E.-s. Cho, S.-s. Kim, G.-c. Park, and S.-S. Yeo. Vulnerabilities in SCADA and Critical Infrastructure Systems.

[28] M. Roesch and S. Telecommunications. Snort - Lightweight Intrusion Detection for Networks. pages 229–238, 1999.

[29] K. Scarfone and P. Mell. Guide to Intrusion Detection and Prevention Systems (IDPS), 2007.

[30] S. Smaha. Haystack: an intrusion detection system. In *Aerospace Computer Security Applications Conference, 1988., Fourth*, pages 37–44, Dec. 1988.

[31] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[32] Symantec. Symantec 2010 Critical Infrastructure Protection Study - Global Results, 2010.

[33] B. Tjaden, L. Welch, S. Ostermann, D. Chelberg, R. Balupari, M. Bykova, A. Mitchell, D. Lissitsyn, L. Tong, M. Masters, P. Werme, D. Marlow, B. Chapell, and P. I. Iv. Inbounds: The integrated network-based ohio university network detective, 2000.

[34] P. Uppuluri and R. Sekar. Experiences with Specication-based Intrusion Detection, 2001.

[35] K. Wang and S. J. Stolfo. Anomalous Payload-Based Network Intrusion Detection. In E. Jonsson, A. Valdes, and M. Almgren, editors, *Recent Advances in Intrusion Detection*, number 3224 in Lecture Notes in Computer Science, pages 203–222. Springer Berlin Heidelberg, Jan. 2004.

[36] M. Wood and M. Erlinger. Intrusion Detection Message Exchange Requirements. RFC 4766 (Informational), March 2007.

[37] D. Zhang and C. Leckie. An Evaluation Technique for Network Intrusion Detection Systems. InfoScale '06, New York, NY, USA, 2006. ACM.

[38] S. Zheng, R. Song, J.-R. Wen, and D. Wu. Joint optimization of wrapper generation and template detection. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 894–902. ACM, 2007.