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

Andr Alves Gomes Rei
andre.rei@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

June 2015

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.

1. Introduction

Critical Infrastructures play a big role in todays world. As defined in the Council Directive 2008/114/EC [3], 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. In their operation, these infrastructures execute a set of physic industrial processes which are monitored and controlled by SCADA (Supervisory Control and Data Acquisition) systems.

SCADA (Supervisory Control and Data Acquisition) 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 [11].

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.

In [9], it is described a hybrid IDS (Intrusion Detection System) 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.

The main goal of an IDS is to protect the Infrastructures 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 [12], 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.

1.1. 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.

The document is structured as follows: Introduction; State-of-the-Art; Solution Description; Solution Evaluation; Conclusions.

2. State-of-the-art

2.1. Intrusion Detection Systems

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[12].

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 Analysis Strategy

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.

The network based IDS Snort[2] 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.

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.

IDES[10] 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.

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.

A model (or specification) based IDS proposed in[4] 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.

2.1.2 Information Sources

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.

One of the most used Network Based IDSs is Snort[2], 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[13], which uses anomaly based analysis. The main task is analyzing TCP connection's timers and its size.

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.

Two widely used HIDS are OSSEC[6] and Tripwire[7], 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 compar-

ing scanned files with older stored files contained in a database.

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.

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. 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.

2.2.1 Techniques

Levenshtein Distance[8] (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.

Soundex[5] is different approach. 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.

2.3. 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.

2.3.1 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.

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

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.

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.

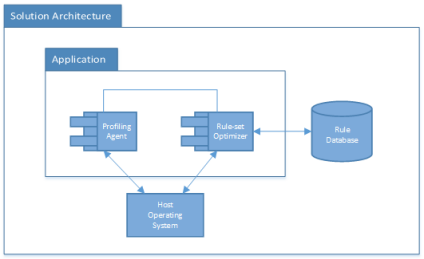

Figure 1: Solution Architecture Overview

Our solution contains two main components, as shown in figure 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.

   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.

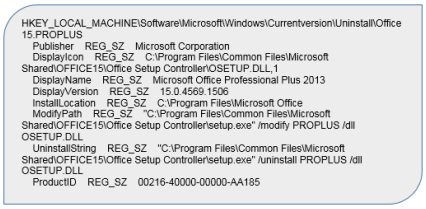

Figure 2: Sample Windows Registry Entry

In Figure 2 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.

3. 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 displayed below.

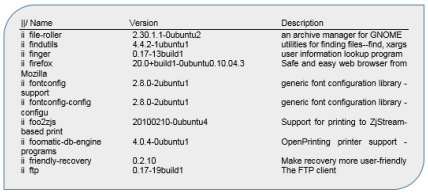

Figure 3: Section from dpkg -l output

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.

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.

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 modules 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.

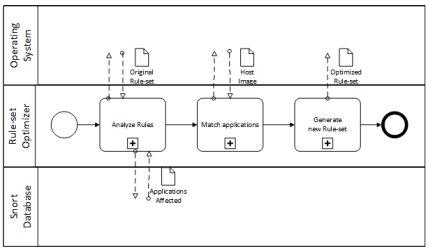

Figure 4: Rule-set Optimizer Activity Diagram

As for our solution, it will be composed of 3 main phases, as shown in Figure 4

- 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.

3.4. 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[2] Rules' format.

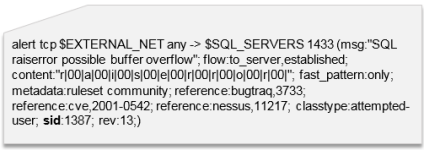

Figure 5: Snort Rule

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 5, 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 6: Section of the page returned by the url https://snort.org/rule_docs/1387

In the example shown in Figure 6, 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 7: Source code of the page section present in Figure 6

In the example shown in Figure 7, 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.

3.4.1 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.

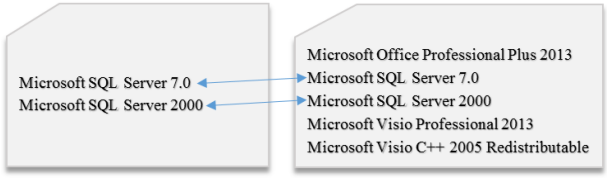

Figure 8: Matching of the affected applications (left) with the applications present the Host Image (right).

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 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.

3.4.2 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 9: New Rule-set generation.

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

4. Results

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:

$$Efficiency = \frac{totalrulesdeleted}{totalrules} \quad (1)$$

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. We will use the following expression:

$$Accuracy = \frac{totalrulesrelated}{totalrulesmatched} \quad (2)$$

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 5612 rules.

The following results were obtained:

TR	TRM	TRR	TRU	TRNR
5615	1096	656	438	2

Table 1: Tests results

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

$$Efficiency = \frac{4519}{5615} = 0,80 \quad (3)$$

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 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:

$$Accuracy = \frac{656}{1096} = 0,60 \quad (4)$$

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.

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.

References

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

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

[3] 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.

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

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

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

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

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

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

[10] 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.

[11] 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.

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

[13] 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.