UNIVERSIDADE D
COIMBRA

João António Rodrigues Ferreira

INTEGRATION OF AGENTS AND COMPONENTS ON A DISTRIBUTED SECURITY ARCHITECTURE

Dissertation within the Integrated Master's Degree in Electrical and Computer Engineering, Specialization in Computers, supervised by Ph.D's Nuno Gonçalves, Tiago Cruz and Paulo Simões and presented to the Department of Electrical and Computer Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

September 2019

UNIVERSIDADE D
COIMBRA

Integração de agentes e componentes numa arquitetura distribuída de segurança

João António Rodrigues Ferreira

Dissertação para obtenção do Grau de Mestre em
Engenharia Electrotécnica e de Computadores

Orientador: Prof. Doutor Nuno Miguel Mendonça da Silva Gonçalves
Co-Orientador: Prof. Doutor Tiago José dos Santos Martins da Cruz

Júri

Presidente: Prof. Doutor Jorge Miguel Sá Silva
Orientador: Prof. Doutor Nuno Miguel Mendonça da Silva Gonçalves
Vogal: Prof. Doutor António Paulo Mendes Breda Dias Coimbra

Setembro de 2019

Success is 1*% inspiration,* 98*% perspiration and* 2*% attention to detail.*

- Phil Dunphy

Agradecimentos

Gostaria de começar por agradecer aos meus familiares, em destaque para os meus pais, irmã e avós, pela constante ajuda, presença e aconselhamento nos bons e nos maus momentos.

Quero Agradecer também a toda a equipa do Laboratório de Telemática e Comunicação da sala G5.4, por me terem acompanhado e guiado ao longo deste ano de tese, ajudando com qualquer dúvida que tivesse. Ao Vítor, por todo o insight que deu em Go, bem como a ”formação” por este colega dada.
Um especial obrigado ao Miguel Freitas pela paciência que teve em me guiar e ajudar com qualquer dúvida que eu lhe pedisse para me tirar pessoalmente em âmbito deste trabalho.

Aos Professores orientadores Tiago Cruz, Paulo Simões e Nuno Gonçalves pela possibilidade de ter trabalhado num ambiente diferente ao habitual, bem como a disposição de ajuda que se submeteram.

Aos meus colegas de curso, mais propriemente à SQUAD, pelos momentos de convívio, diversão e pelo espírito de companheirismo e apoio mútuo ao longo destes 6 anos.
Um especial obrigado e um enorme abraço aos meus grandes amigos do ”FQI”.
O mesmo digo a todos colegas da residência Polo2-II, obrigado por todas as vivências ao longo deste curso.

Aos meus amigos que me acompanharam e apoiaram durante toda esta etapa, sejam de onde forem. E a todos os que contribuíram direta ou indiretamente para o meu processo de formação académica e pessoal.

A todos,

Muito Obrigado.

Contents

List of Figures

List of Tables

Abstract

Over the past recent years there has been a massive industry growth, and something we call the Internet of Things has emerged. This phenomenon has set the path for a paradigm shift in the Industrial Sector, which has allowed for a wider range of digital interconnectivity when it comes to various industry components. This however has also opened a way for different types of malicious activity agaisnt the industry. Since the systems went from being air gapped to being remotely connected to a vast range of networks and other third-party systems, various exploits have been made possible in order to maliciously affect those industry components. ATENA, an European Project, aims to provide a range of modern tools to detect and prevent such activity.

This thesis' work is part of the lowest level in the ATENA architecture, the detection of anomalies and events taking place at any point in a host or network, done by multiple probes. These events should then be taken to a platform where the administrator can monitor and analyze them. In order to transfer such data between multiple components, a format (IADS Data Model) is followed throughout the entirety of the IDS platform, the IADS. In order to read, encode and send to the other components all events from all existing, and yet under development, probes, a wrapper or adapter had to be created, giving the administrator the tools required in order to configure how the probe's output is translated into an IADS Data Model format event. This wrapper, as well as the mechanisms of differentiating probe outputs, encoding and validating them, and sending the encoded events to the rest of the IADS Platform, is part of this thesis' work.

Keywords

Industrial Automation and Control systems, Intrusion and Detection Systems, Security information and event management

Resumo

Nos últimos anos, e com um crescimento massivo da indústria, surgiu algo que chamamos de Internet das Coisas. Esse fenómeno estabeleceu o caminho para uma mudança de paradigma no setor industrial, que permitiu uma interconectividade digital dos vários componentes da indústria. No entanto, isso também abriu caminho para diferentes tipos de atividades maliciosas contra a indústria. Desde que os sistemas deixaram de separados isoladamente uns dos outros e passaram a ser conectados remotamente a uma vasta gama de redes e outros sistemas de terceiros, vários exploits foram feitos possíveis para afetar maliciosamente esses componentes do setor. O ATENA, um projeto europeu, visa fornecer uma gama de ferramentas modernas para detectar e impedir essa atividade.

O trabalho desta tese faz parte do nível mais baixo da arquitetura ATENA, a detecção de anomalias e eventos que ocorrem em qualquer ponto de um host ou rede, realizado por várias sondas. Esses eventos devem ser enviados para uma plataforma na qual o administrador possa monitorizar e analisá-los. Para transferir esses dados entre vários componentes, um formato (Modelo de Dados IADS) é seguido em toda a plataforma IDS, o IADS. Para ler, codificar e enviar para os outros componentes todos os eventos de todas as sondas existentes e ainda em desenvolvimento, um wrapper ou adaptador teve que ser criado, dando ao administrador as ferramentas necessárias para configurar como a saída da sonda é traduzida num evento no formato IADS Data Model. Esse wrapper, bem como os mecanismos de diferenciação das saídas da sonda, codificação e validação, e envio dos eventos codificados para o restante da plataforma IADS, fazem parte do trabalho desta tese.

Palavras-Chave

Industrial Automation and Control systems, Intrusion and Detection Systems, Security information and event management

Acronyms

ATENA **A**dvanced **T**ools to ass**E**ss and mitigate criticality of ICT compo**N**ents and their dependencies over Critical Infr**A**structures. 2, 12

CI **C**ritical **I**nfrastructure. 6

CIA **C**onfidentiality, **I**ntegrity and **A**vailability. 7

DDoS **D**istributed **D**enial-of-**S**ervice. 15

DEEC **D**epartment of **E**lectrical and **C**omputer **E**ngineering. 2

DEI **D**epartment of **I**nformatics **E**ngineering. 2

HIDS **H**ost-based **I**ntrusion **D**etection **S**ystems. 9

IACS **I**ndustrial **A**utomation and **C**ontrol **S**ystems. 6

IADS **I**ntrusion and **A**nomaly **D**etection **S**ystem. 2, 10–12, 14, 38

ICT **I**nformation and **C**ommunication **T**echnology. 6

IDMEF **I**ntrusion **D**etection **M**essage **E**xchange **F**ormat. 14

IDS **I**ntrusion **D**etection **S**ystem. 7–9

IoT **I**nternet **o**f **T**hings. 6

IPS **I**ntrusion **P**revention **S**ystem. 8

LCT **L**aboratory of **C**ommunications and **T**elematics. 14

MITM **M**an **I**n **T**he **M**iddle. 38

MQTT **M**essage **Q**ueuing **T**elemetry **T**ransport. 12

NIDS **N**etwork **I**ntrusion **D**etection **S**ystems. 9

NIST **N**ational **I**nstitute of **S**tandards and **T**echnology. 7

SCADA **S**upervisory **C**ontrol and **D**ata **A**cquisition. 10, 11

SIEM **S**ecurity **I**nformation and **E**vent **M**anagement. 9

YAML **Y**AML **A**in't **M**arkup **L**anguage. 25

Chapter 1

Introduction

The purpose of this chapter is to introduce the work carried out in this thesis. Some motivation aspects and goals are defined and a brief explanation of the developed software is given, along with the main contributions where such software was used and validated.

1.1 Motivation

Despite being a student of the **D**epartment of **E**lectrical and **C**omputer **E**ngineering (DEEC), this thesis was developed in a different environment, in the **D**epartment of **I**nformatics **E**ngineering (DEI), due to an interest in aspects such as cyber-security and Software Engineering and the correlation of such aspects with the thesis general theme.
This thesis' work was undertaken in context of ATENA's WorkPackage 4 - **I**ntrusion and **A**nomaly **D**etection **S**ystem (IADS), coordinated by a team of researchers from the **L**aboratory of **C**ommunications and **T**elematics (LCT), which is integrated within the Centre for Informatics and Systems of the **U**niversity of **C**oimbra(UC). This WorkPackage is better contextualized in Chapter 2, which contains the whole overview of this thesis' context. Specifically, the team needed the development of a working agent that acts as a wrapper to each and every probe deployed in a security architecture.

1.2 Goals

The goal of the thesis was to research and develop a generic and platform-neutral piece of software that can be wrapped to any detection component, in order to consolidate the whole detection layer with the rest of the architecture, which already respected the IADS Data Model, explained later in chapter 2. This software, which we call the agent, had to be easily configurable and easily deployable in an agile fashion. The former requirement was one of the key factors to take in account, since probe administrators would be required to normalize the output of their specific detection component. Therefore, the agent needed to be configurable to the point of giving full control of sources and data-model field assignments.

1.3 Implementation and Key Contributions

During the thesis' work plan, some demonstrations were scheduled as part of the ATENA consortium schedule, and while the first ones were mostly used as a way to gather feedback from the rest of the WP participants on how to better adapt the agent, the last one was a project-wide review that required a fully functional agent as part of the efforts to

deliver the expected results for Work Package 4 —(WP4).
Two main implementations of this agent were developed, in line with the work plan. The first one was a prototype to gather some feedback on a ”real-case scenario” tested for the first demo, also documented for Deliverable 6.4[5] - Design and development report of the 2nd release of components, taking place in 17th of October 2018. This prototype had run in two instances alongside two probes, namely a Snort[6] and a ConPot[7] probe, which were used on a use case scenario to demonstrate the IADS detection capabilities for a series of network attacks against the reference testbed. This test was successful, providing enough feedback and inputs to further refine the IADS, as well as the agent being developed.
From here, a new branch was created to further develop the application and, in the following months, multiple versions of the agent were released until it satisfied the requirements needed for the Final validation Demo, also documented in the deliverable D7.5[8] - Final ATENA prototype validation and evaluation of validation results (Final DEM0). This final agent was tested and validated by the team, being able to successfully send multiple events from different types of probes, to the domain processors, using the established data model. All the mentioned and contributed-to deliverables are referenced in the bibliography section of this document.

1.4 Document Structure

The remainder of this dissertation is organized in the following structure: in chapter 2 the theoretical foundations framed within this work are presented, as long as an overview of the related work and research done that can be of use for this thesis. Chapter 3 displays the work made in this dissertation, and finally, in chapter 4, the final conclusions and proposals for future works are specified.

Chapter 2

Context and State of the Art

This chapter lays out the theorical foundations of the thesis' work, focusing on introducing its context in terms of the general overview of the ATENA project and the Work Package 4. It is then explained where the whole IADS architecture fit within the ATENA architecture and then how this thesis' work would be situated in the scope of the IADS architecture. We also go into detail on existing technologies related to this work and elicit some researched and developed work that can be of use in ambit of the Lab's Work Package.

2.1 Industrial Automation and Control Systems

The recent massive industry growth and, by association, the sheer rising number of interconnected devices and data transferred between components, has led to the dawn of the "**I**nternet **o**f **T**hings (IoT)" age. This has of course led to a paradigm shift in **I**ndustrial **A**utomation and **C**ontrol **S**ystems (IACS). Whereas some years ago Industrial Systems boundaries only went as far as monitoring single, non-interconnected and far less complex components, the new environment has raised the bar on these Control systems when it comes to **C**ritical **I**nfrastructure (CI), meaning they now have to take in account the rapidly expanding network of operators and dependencies that are involved in a complex system (e.g power-grid Infrastructures, water-supply infrastructures, ...)[9].
With the rise of the IoT age, available technology is also under pressure to keep up, and ICT/IT (**I**nformation and **C**ommunication **T**echnology) can benefit from modern technologies like virtualization and innovative algorithms to provide better and more efficient monitoring and management of a set of components, devices, and data traffic.
This new paradigm allows for the development of modern solutions to keep up with the exponentional Industry growth, and the correlated need for the introduction of security and monitoring mechanisms.

2.2 The ATENA Project

Following this paradigm shift explained in the previous section, as a follow-up to two previous European Research activities, the CockpitCI and MICIE-EU projects, and in the context of an European Commission Framework Program for Research and Technological Development, Horizon 2020, the ATENA, or AdvancedTools to assEss and mitigate the criticality of ICT compoNents and their dependencies over Critical InfrAstructures project was created and aimed for innovation in the context of ICT's and, inherently, CIs and their security and resilience.[10] This is made possible through the use of modern anomaly detection algorithms and risk assessment methodologies[1] in the development of a group of tools and methods which not only enhance the mentioned aspects of security and resilience in CIs, but also preserves the efficient and flexible monitoring and management inherent to them.[11]

ATENA had most of the its partners already involved the the previous two mentioned projects, CockpitCI [12] and MICIE-EU [13], so each and every one of the thirteen partners was an expert in one or more security fields. Every partner had a specific task or set of tasks. These tasks, or modules that ought to be developed in ambit of the project, are displayed in figure 2.1.

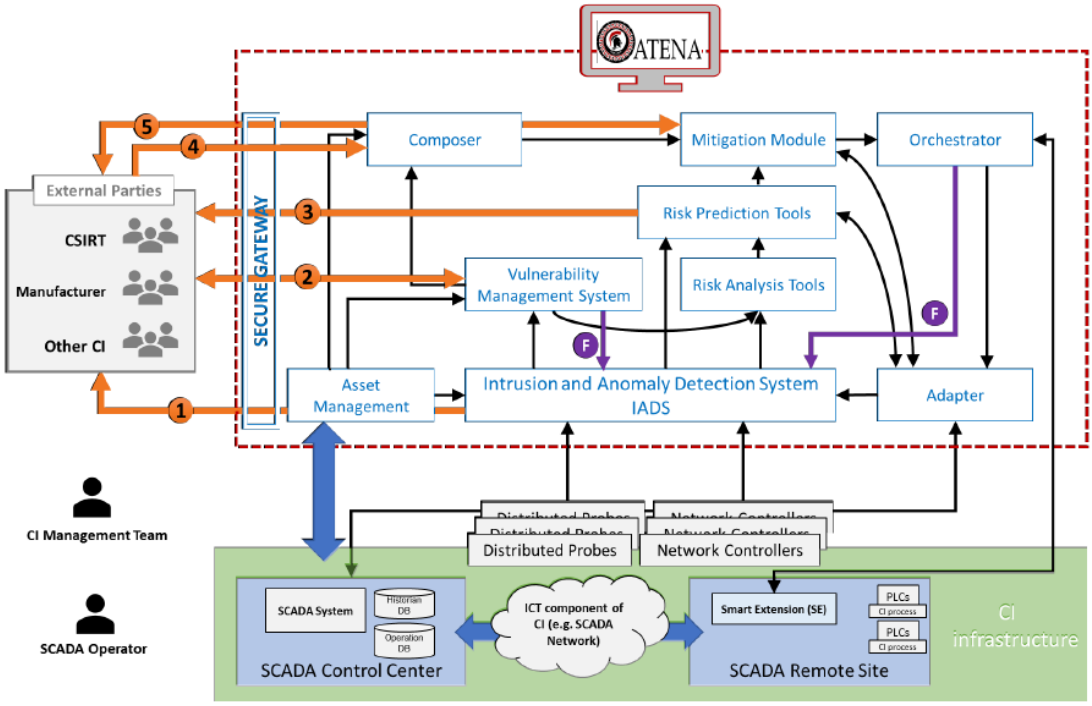


Figure 2.1: ATENA Architecture and it's modules, from [1]

2.3 Intrusion, Intrusion detection and IDS

To better understand the concept of a **I**ntrusion **D**etection **S**ystem (IDS), we need to explain and make the distinction between the concepts of Intrusion, Intrusion detection and IDS. We can start by describing what an intrusion is, which, in accordance with **N**ational **I**nstitute of **S**tandards and **T**echnology (NIST)'s description is an attempt to disturb or compromise with ill-intent the **C**onfidentiality, **I**ntegrity and **A**vailability of a single computer system or network[14]. These can be, for example, Denial of Service attacks, system information manipulation, denial of a system action and retransmission of valid messages under invalid circumstances to produce unauthorized effects.[15] Intrusion Detection is simply

the process of monitoring and analyzing events that take place in a network or system that might suggest an intrusion, while an IDS is a system that allows for the automation and configuration of the Intrusion Detection process. There is also the existing concept of **I**ntrusion **P**revention **S**ystems, but since the emphasis of the work is on the detection rather than the prevention of Intrusions, we won't go into detail about them. IDS detection mechanisms can be of two different types: Signature-based and Anomaly-based.

2.3.1 Signature-based and Anomaly-based IDS

Signature-based IDS detect intrusions by checking for specific patterns in network traffic and instruction sequences, taken from known malicious activity samples, interpreted as their 'signature'. Since these systems rely on a database of some sort that contains multiple malicious signatures, they are often not reliable for new or unknown attacks or events, and sometimes fail to detect or give out an alarm for an intrusion if one occurs.[16] Anomaly-based IDS on the other hand can be used to detect intrusions through the comparison of normal behaviour model created with machine-learning and the new model of unknown new behaviour[17]. If the models show significant differences between them, an detection alert might be given.
Below is a list of advantages and disadvantages of each type of IDS, based on [15].

- **Signature-based IDS**
  - **Advantages**
    - Accurate, low false-positives rate
    - Easier to track alarm cause, detailed logs
  - **Disadvantages**
    - Dependant on existing signature databases;
    - Administrator needed to update databases;
    - Can cause systems to slow down with the huge amount of traffic packets analyzed, sometimes making them be dropped;

- **Anomaly-based IDS**
  - **Advantages**
    - No need for constant manual database updates;
    - Little maintenance required;
    - Gets more accurate the longer the system is used.
  - **Disadvantages**
    - If the system creates a model while the system is under malicious activity, might consider it normal behaviour, thus failing to send out an alarm in that case;
    - Higher rate of false-positives;

2.3.2 SIEM

IDS systems are usually dependant of a **S**ecurity **I**nformation and **E**vent **M**anagement (SIEM) to capture events occuring in the system. These systems gather data from a broad multitude of sources and channels of different types [18]. This thesis' work is centered on this part of the IDS platform that was developed (the IADS), since each tool or probe has a specific manner of outputting detected events.

2.3.3 NIDS and HIDS

Other than classifying IDS based on their detection mechanism, we can also classify them through their point of implementation in a system (whether the IDS component is actively functioning on a host or on a network point) and therefore range of data scanned. To the system that analyzes a given single host or device, meaning all its inbound and outbound data as well as the integrity of critical system files, we give the name of **H**ost-based **I**ntrusion **D**etection **S**ystems (HIDS). These systems are usually strategically deployed in hosts where configurations and files are not likely to change.
On the other hand, **N**etwork **I**ntrusion **D**etection **S**ystems (NIDS) are placed in strategic points of a network in order to monitor all traffic within that subnet, generating events for

any abnormal activities in the network. IADS contains both of these types of mechanisms in different points of the system.

NIDS	HIDS
Host Independant	Host Dependant
High false positivity rate	Low false positivity rate
Requires acceptable bandwidth	No need for Bandwidth
Detects Network attacks	Helps detect complex attacks on certain hosts and machines
May slow down the network where it is implemented	May slowdown it's IDS host
Examples:	
Snort[6]	OSSEC[19]
Suricata[20]	Samhain[21]
IBM QRadar[22]	Fail2Ban[23]

Table 2.1: Table showing some properties and examples of each type of IDS

2.4 SCADA Systems

Supervisory **C**ontrol and **D**ata **A**cquisition (SCADA) systems are industrial control systems that through the use of supervisory computers and human-machine interfaces, allow the user to monitor a wide range of components in a given project, like **P**rogrammable **L**ogic **C**ontrollers (PLC) or PIDs (**p**roportional–**i**ntegral–**d**erivative controller), through a **G**raphical **U**ser **I**nterface (GUI).

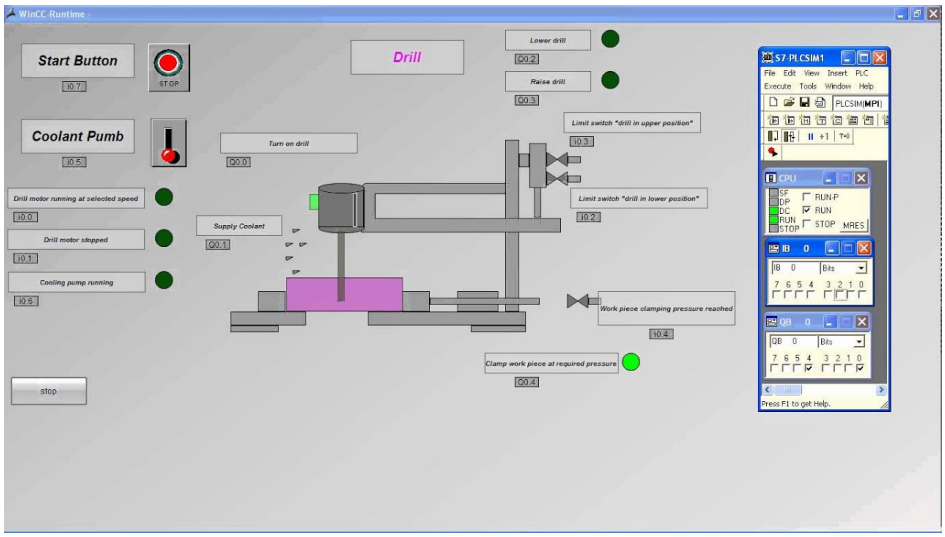


Figure 2.2: Example of a SCADA system. Still taken from [2]

2.5 ATENA Work Package 4 - IADS

In the context of the ATENA project, already introduced in section 2.2 and displayed in figure 2.1, the LCT/CISUC team was in charge of leading the development of the **I**ntrusion and **A**nomaly **D**etection **S**ystem, or IADS[1], a platform consisting of a set of tools and components that allow for the detection, aggregation, filtering and analysis of security data that might be of critical importance to CI's. Since this system uses both physical and virtual tools, enabled by technologies such as SDNs, it can be considered a Cyber-Physical IDS. This IDS platform, along with the all the algorithms and components required for its integration and efficiency, was developed within the context of WP4. Essentialy, the project aimed to provide a solution that enables a higher standard of protection in a **S**upervisory **C**ontrol and **D**ata **A**cquisition (SCADA) environment, monitoring the data transfered between its components and detecting anomalies that might reveal a threat to the Critical Infrastructure. In order for this to be possible, anomaly detection probes had to be developed along with a system that allows for the data to be aggregated and analyzed.

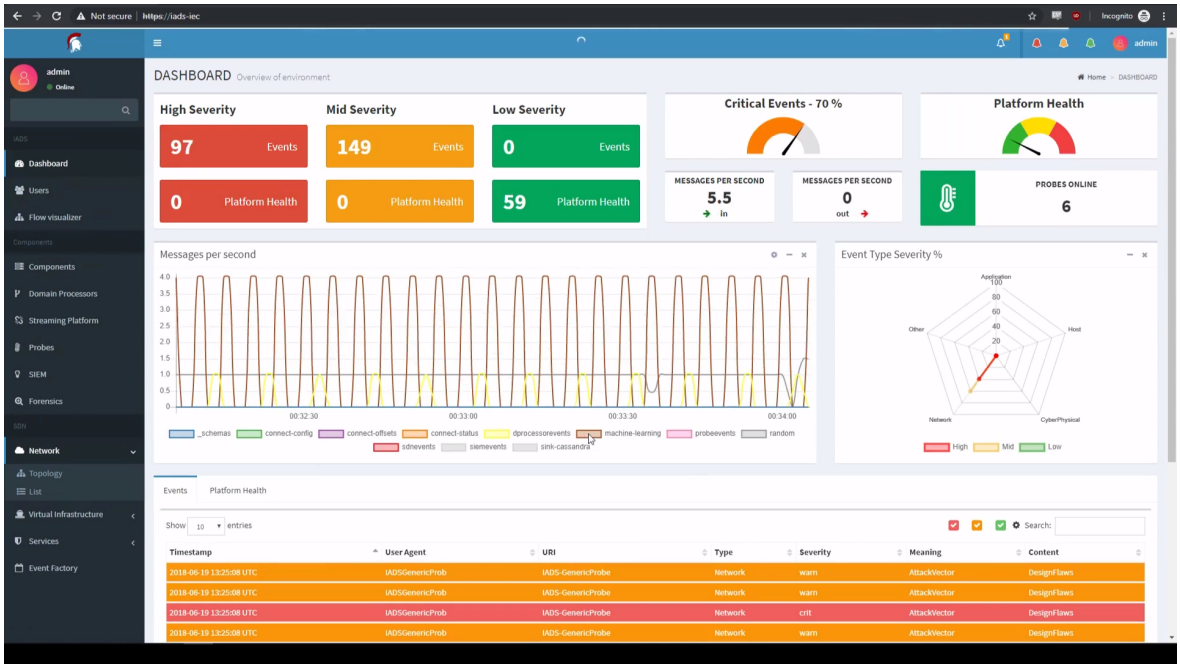


Figure 2.3: The IADS Platform. Detected events are shown in the lower table. Still taken from [3]

2.5.1 Architecture of the platform

The IADS, while being composed of multiple tools and mechanisms, has an architecture of its own, composed by three main sub components:

- The management component;
- The Distributed Intrusion Detection System component;
- The Forensics and Compliance Auditing component.

These three work in parallel, with the first one being in charge of configuration and settings of the other two components; the forensics and Compliance component acts as the front-end of the platform, allowing the administrator to monitor and filter the detected events in it; The main one, the Distributed Intrusion detection system does the main work of the architecture, from the point where a probe is applied to a host or node in network and sends an event to the domain processor, to the point where this second component sends an event via **M**essage **Q**ueuing **T**elemetry **T**ransport (MQTT) to an external point in the ATENA architecture.

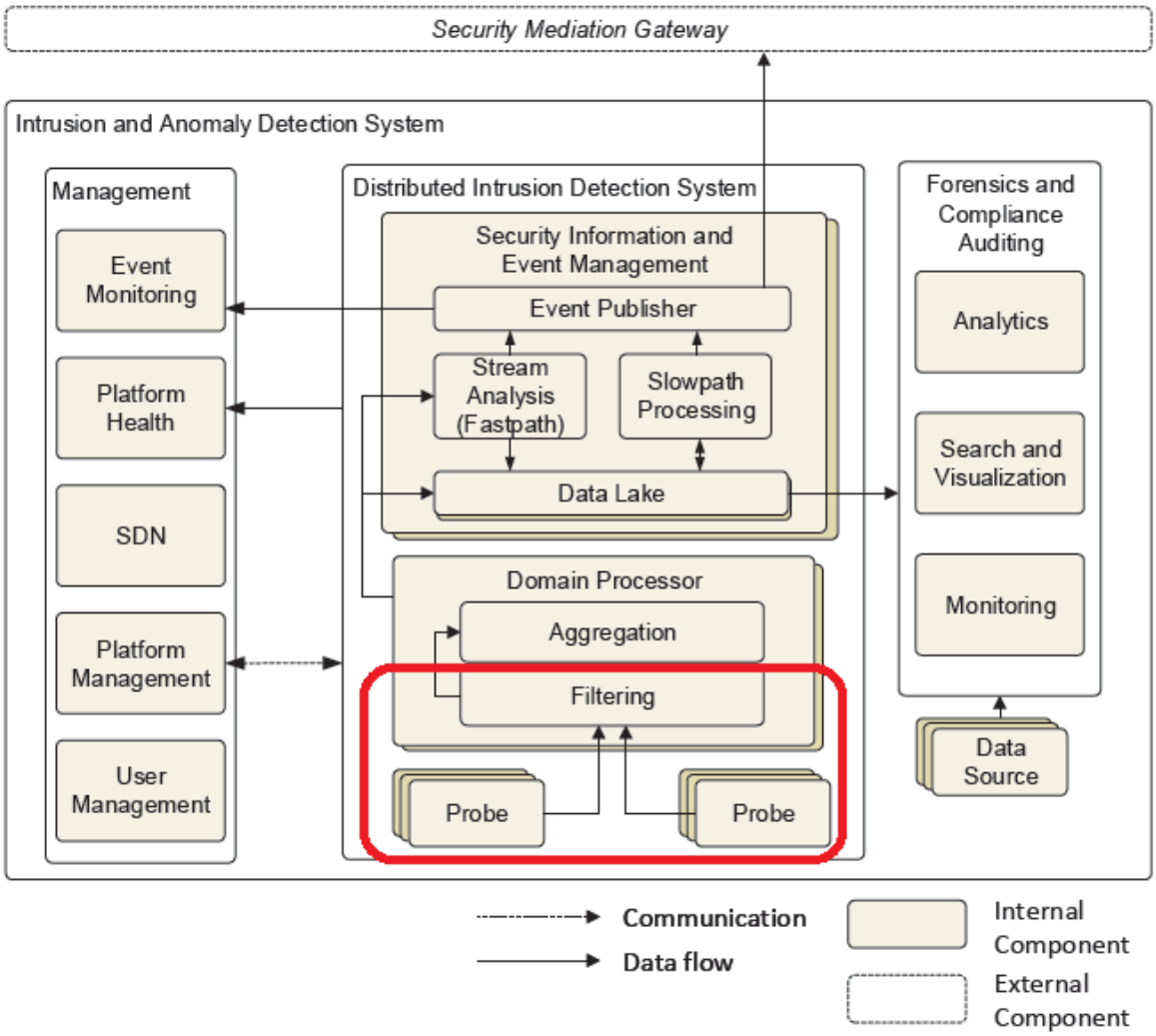


Figure 2.4: IADS Architecture and it's components.[4]

The figure 2.4 displays the IADS architecture and its components and data flow channels. In red is the part where this thesis work is inserted, having an emphasis in the probes and their communication and data flow channels. The Data from the probes is being to a given Kafka[24] cluster, Domain Processor in the architecture, which the various components in the platform are subscribed to, and therefore can access the messages sent to it.

2.5.2 The IADS-Events Datamodel

As we can observe in figure 2.4, throughout the entire IADS data is sent and received through multiple data flow channels, so a uniform and user-friendly format of data

interchangeability between IADS components was proposed and implemented[25] as part of the whole platform. This format was based on the **I**ntrusion **D**etection **M**essage **E**xchange **F**ormat (IDMEF), and the LCT denominated it as the ”IADS-Events Datamodel”. Every component developed in ambit of this project must respect it and every message transferred between them must follow it.[26]

The Data-Model is composed of a header field with metadata information, a payload field, where all the actual event information is stored and detailed and an **U**niversally **u**nique **i**dentifier (uuid).

```
uuid
Metadata:
  Origins: (1..*)
    URI
    UserAgent
    Timestamp
    PreviousOrigins (0..1)
  Checksum
Payload:
  Events: (1..*)
    Type
    URIofType
    Severity
    Data:
            ...
```

Figure 2.5: General overview of the IADS-Events Datamodel.

The payload field ’Data’ changes depending on the Event, which can be of five different types:

- Application;
- CyberPhysical;
- Host;

- Network;
- Other;

In Figure 2.6, the details on a specific type (Application) can be seen, with the remaining types' Payloads examples being annexed in 5.

```
Metadata:
    Check the Metadata structure here: https://github.com/lmrosa/cpids-events
Payload:
    Events:
      URIofType
      Type (Network)
      Severity (emerg, alert, crit, err, warn, notice, info, debug  - rfc5424)
      Data:
          SubType
          Entities:
              Sources (0..*)
                  ProtoName
                  Value (0..*)
              Destinations (0..*)
                  ProtoName
                  Value (0..*)
          Items: (0..*)
              Meaning
              Content (Binary/ASCII)
```

Figure 2.6: Example of the Application Event type Payload field.

2.6 Security Event-Generating Probes

To firstly detect any kind of activity, we needed a tool or set of Tools, either physical or virtual, that act as an examination or sifting mechanism. To these tools we call Probes, and in this context they are the lowest layer in the architecture. It is from their processes that we are able to extrapolate what is happening in a given host and/or network, and that information is transferred up for processing in the IADS.

Multiple Probes can be running at the same time in the same machine, since each one can be configured to detect a specific type of activity. For example, a snort probe can be used to detect a **D**istributed **D**enial-of-**S**ervice (DDoS) attack, while a ConPot probe will most likely be used to detect new, strange, connections to the system. These are the two probes

we mainly used and validated in the testing scenarios.

While using multiple probes has it upsides, it also has its disadvantages. Since every probe works differently from each other, they can and mostly do have different ways of outputting activity data. They can send it through the syslog [27] , output it to a file in their system or even to a Network and/or UnixSocket.

Examples of these differences in the probes' outputs can be seen below, in this case, when there are two probes outputting to a logfile.

```
Snort Line:
 2018-10-09T19:25:15.354460-04:00 nids2 snort: (spp_arpspoof)
  ↪ Attempted ARP cacheoverwrite attack,( Mismatch mapping
  ↪ 00:1c:7f:32:18:9d <-> 172.27.248.1, sha b8:27:eb:d9:bc:e0, spa
  ↪ 172.27.248.1, tha 00:0d:22:09:bd:dd, tpa 172.27.248.213
Conpot Line:
 2018-10-10T02:39:21.356365+00:00 a0e47df0f785 New Modbus connection
  ↪ from 192.168.1.15:59272. (2b60648e-f337-404a-b8b0-be6a9a0cd670)
```

Table 2.2: Two example lines, the first from Snort and the second from ConPot.

Since all the IADS platform mechanisms must follow the IADS-Events Model, this creates a challenge to the administrator, that now needs to configure each probe properly in order to output it in an encoded model format. This can be time-consuming as the number of probes increases and each probe changes. Therefore, an agent needed to be developed in order to read any probe's output, encode it to the Data-Model format, and lastly send it to the Kafka Cluster.

2.7 Previous Implementation

In ambit of a previous work[10], an agent wrapper was developed called Syslog-Probe [28], which was on version 0.4 at the time I joined the project. Unfortunately, that version only worked in very specific scenario, where the probe was sending Network events to the syslog, and no other scenario. Between updating the SyslogProbe to the desired standards and building a new wrapper with a more general-approach to every type of probe possible from scratch, the team chose the later option.

The OSSEC HIDS[19] also has a component capable of collection data from various

sources [29], but to implement it in this case would be a case of 'using a sledgehammer to crack a nut', since it is a somewhat heavy tool to use in the machines, and containers, the probes would run in.

2.8 Chapter Overview

Having gone through the context of the project and some of the work that has already been done in ambit of it up until the date I joined the project, the foundations were laid in order for the thesis' main work to be done. There was already a tool in use for similar use cases, LogStash[30], but since it was developed and configured by default for another third-party data-flow, we opted to develop a generic adapter instead. Some advantages were also taken in account from the capability syslog-ng[27] has in received a feed of data from multiple sources, but ultimately they were not used in the adapter since they didn't quite directly serve the purpose. Some knowledge of technologies such as Regular Expressions [31] also facilitated the development of a solution.

Chapter 3

Implementation

In this chapter we present the all the requirements, and their rationale, that needed to be fulfilled in order for the agent to be validated. We also give a brief overview of some useful libraries used in the project, as well as an explanation of why we chose the GO Language and YAML formats for the development of the agent.

Afterwards, we present the main architecture of the program and then an explanation of every sub-component in the following sections. We finish up this chapter by mentioning some notes about the development process itself, such as obstacles found, and reasoning for multiple branches. The development process of the agent can be seen in detail in its Git Repository[32], although it is inherently private. This thesis' work tasks, alongside their order and goals followed a timing plan, annexed as well in this thesis.

3.1 Requirements elicitation

The development of an agent in the Project's environment has certain functional requirements, some taking a higher level of priority than others.

These requirements are taken from a set of needed functionalities in the probe/developer's point of view, the User Stories. The priorities in table 3.1 are rated in an ascending order, starting in the highest degree of priority, 1.

The software also needed to run on small hosts, often within Docker[33] containers, so a pre-built agent would be preferred for the deployment. For this reason, it was decided that the development language had to be based based on a compiler and not on an interpreter, meaning a compiled programming language had to be used. A large number of dependencies was also to be avoided for the same reason, so Languages such as Python were excluded.

Since a Java[34] Implementation would require the whole Java Virtual Machine to be deployed along with the agent, it was also taken out of equation. This left languages such as C, C++ and GO on the table. Compared to the former two languages, GO has advantages such as the use of interfaces, easier pointer manipulation, easier multi-platform deployment, better package management and functioning optimal garbage collection. For those reasons, I opted to develop the agent in this modern language.[35]

User Stories	Requirements	Priority
The agent must be able to read from any of the probe's output methods	Development of a generalized method of reading output from the following sources: syslog, files, sockets and command-line prompted values	1
The user must be able to map any value he wants to any field in the event	Capability to set a data source as static, and therefore gathering information from a string given by the user.	1
The agent must be able to map any type of Data-Model* Event.	Develop a parsing&mapping method for any kind of data-model event type	1
The agent must be as general as possible, supporting a broad range of probes with different output methods between eachother	Support any type of probe, multi-platform capabilities	1
The agent needs to map any data from the probe's output to the correct field in the Data-Model Event	Map any given information into a Data-model* event field	1
The agent needs a certain level of concurrency when mapping multiple events	Capability to parse multiple events from the same probe, and a setup file which allows to do so.	1
For debug purposes, the agent's user may be able to read a log of the agent's functions	Log each and every main agent's process, as well as the line-by-line input and output of the agent.	1
The user must be able to set the destination of the parsed & mapped event	Kafka cluster message transfering integration and configuration	1
The user may have the ability to validate the output's data	Development of a way to validate each mapped data through use of regex's	2
The configuration of the agent should be as intuitive as possible	Use of a setup file that has a format/language similar to the Data-Model Event	2
The agent must be able to set a backup source in case the data isn't validated	Have the possibility of a backup/fallback route** configured in the mapping configuration file	3

Table 3.1: Requirements elicitation with User-stories.

3.2 Useful Tools & Libraries

As explained in the previous section, the software was developed in Go, so any tools and libraries had to be compatible with it. Due to the recent nature of the language, those

weren't as easy to find as packages are found in Python implementations. Nonetheless, after some research, packages that helped with the configuration of the agent and the last process the agent needed to do, producing an message with the event information to a kafka cluster, were found:

Package	Description
Viper[36]	Configuration solution for Go applications. It is designed to work within an application, and can handle all types of configuration needs and formats.
pKafka[37]	Producer for a Kafka topic, from an Avro Message.

Table 3.2: Main packages directly used throughout the development.

3.3 IADS-Events Agent Architecture

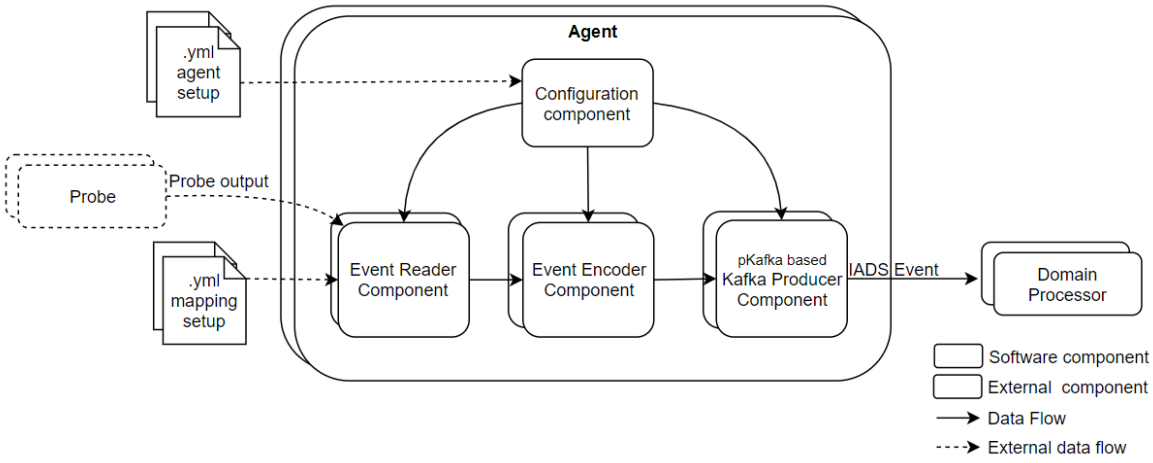


Figure 3.1: Depiction of the Agent's Architecture.

This agent was be deployed as a 'middleware', placed between any given probe and a kafka broker, acting as an adapter or wrapper for all types of events and probes.
The diagram on image 3.1 depicts where the agent stands in the general IADS architecture, as well as its four main process components.
The agent makes the data communication between its multiple parts through the use of channels, a handy mechanism that Go allows us to use in order to efficiently process and if desired, multiplex, each sub-component.[35]

3.3.1 Configuration Component

The first component that's actually ran is the Configuration one. Basically what it does is read all the variables set in the *config.yml* file and store them in a Public Structure which every other part of the agent can access. An example of a variable stored is *Verbose*, a *boolean* variable that the every processes check every time a function is called to print any information to the log. With these settings assigned and set, the software's main components are ready to run.

3.3.2 Event Reader Component

Having set the required configurations, the first component in the event mapping process is the event reader. From a succinct point of view, it does the following actions, in order:

1. Checks the source of a given field to be passed on to the Data-Model Event. This is described in the Mapping file;
2. Reads any source text outputted through the probe, line by line;
3. Applies the given regex to the text found, extracting the useful information out of it and stores it into a struct;
4. Sends the created organized structure through a *DataForMessage*-type channel, to the Encoder component.

3.3.3 Event Encoder Component

After receiving the event from the Reader Component, and through the use of the *DataForMessage*-type channel, this component goes through the information and assigns the gathered data to their specific fields in a Go interface.

```
AvroMessage := map[string]interface{}{
        "uuid": uuid.New().String(),
        "Metadata": map[string]interface{}{
                "Origins":  OriginsArray,
                "Checksum": CheckSumString,
        },
        "Payload": Payload,
}
```

Figure 3.2: Structure of the event in an Go Interface Format.

After evaluating the type of event and having all the information correctly assigned, we use the method *BinaryFromNative*, which is based on the GoAvro.v2[38] package, to encode the Go interface into an Apache Avro [39] Binary Message Format, a format our Kafka Producer can use to send a message to our Domain Processor, which is en essence a Kafka Cluster. It then sends the message through a *Binary* channel to the next component in the architecture, the Kafka Producer.

3.3.4 Kafka Producer Component

While integrated in the agent, this component was based on prior work, developed by an UC team member,, the pKafka [37] Package. It takes the configuration properties from the *config.yml* file and produces a message in the given topic, in the Kafka Cluster.

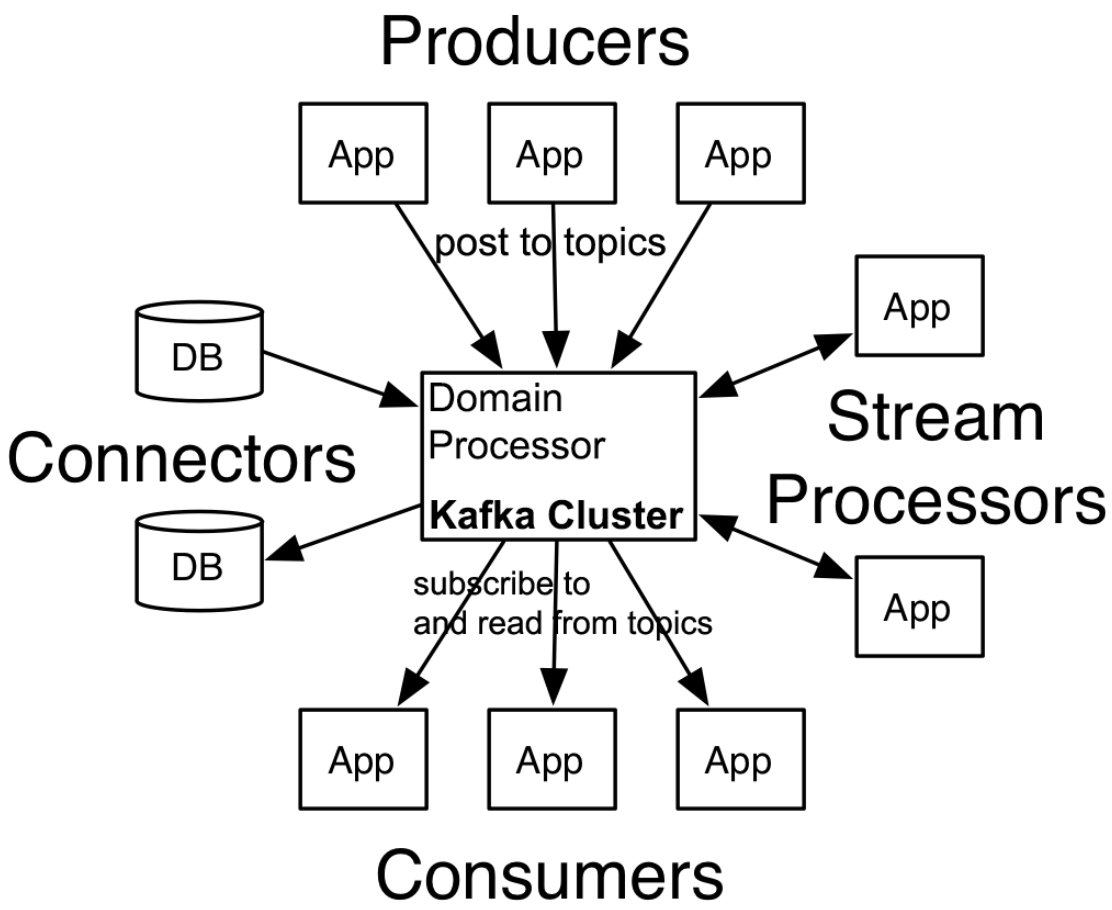


Figure 3.3: Depiction of how Kafka Cluster message transfers work.

3.4 Configuration and Mapping Files

Two main files are used to configure the agent and to specify what information from a source goes to a certain field in the data Model[26]. These are the *config.yml* and, by default, *EventMap.yml* files respectively. These were determined to be formatted in YAML (**Y**AML **A**in't **M**arkup **L**anguage), a variant of JSON, for its simplicity to read, since no delimiters are used, whereas JSON has them and thus makes it less human-readable. The use of this Format also allowed us to make the file itself really similar to the data model schema, which is also an advantage for the administrator who will end up mapping the information in the file.

```
{
  "person": {
    "firstname": "Tom",
    "lastname": "Smith",
    "year": 1982,
    "favorites": ["tennis",
     ↪  "golf"]
  }
}
```

```
<person>
  <firstname>Tom<firstname>
  <lastname>Smith<lastname>
  <year>1982<year>
  <favorites>
    <value>tennis</value>
    <value>golf</value>
  </favorites>
</person>
```

```
person:
  firstname: "Tom"
  lastname: "Smith"
  year: 1982
  favorites:
    - "tennis"
    - "golf"
```

Table 3.3: Differences between JSON(1), XML(2) and YAML(3).

3.4.1 Configuration file

The first file, *config.yml*, is a file with simple fields to fill out, in regards to some debug settings, and, most importantly, the name/path of the second YAML file or pipe and the properties of the kafka cluster, to where we want the encoded data to be sent, along with its topic. An example of a fully filled out *config.yml* file can be seen in table 3.4.

```
global:
  verbose: true
  printJSON: true
  benchmarks: false
source:
  MappingFile: "EventMap"  #path of file, with the extension excluded
kafka:
  nbrokers: 1
  addrs: "123.45.67.890:1337"
  topic: "probeevents"
  # magicBytes should contain an even set of hexadecimal bytes that
   ↪  are added to begin of kafka messages
  magicBytes: "0000000001"
```

Table 3.4: Example of a filled out *config.yml* file.

3.4.2 Mapping file

The Mapping file was structured with the possibility of allowing the user/developer to:

- Define the method in which the probe's & event's information is captured. Static values are supported.
- Define fields as arrays. This means that a field can have multiple entries, as required per the Data Model.
- Parsing and transmission of multiple & different events from the same probe, using only one agent per probe.

This file is configured by the administrator to work with any given probe that outputs non-normalized data to any feed of information, effectively working as a dictionary to the agent, so it correctly assigns each word or set of words in the data feed to a field in the message sent in the format of the data model. Since the file is in YAML format, careful indentation is required in order to take advantage of the format's inherent hierarchy.

3.4.2.A Source Field

The first field to be filled in the file is the *Source* field. It defines the input's main source, and it may be overridden by each and every declared field, in case some specific data is gathered from a different source than the one declared in the Source field.

3.4.2.B Field Structure

Every Data Model field is defined in the following manner, represented in table 3.5

```
Field:
  GetFrom:
  Value:
  Regex:
  ValidateRegex:
  Fallbacks:
    -FallBack1:
      GetFrom:
      Value:
      Regex:
    ...
```

Table 3.5: Layout of every field in this type of file.

Field is the name of the field to be filled in the Data Model message (URI, User-Agent, URIofType,...). Depending on the method used to obtain the field's information, various values can be attributed to the *GetFrom* property: STATIC, FILE, STREAM, UNIX-SOCKET and EXEC. *Value*, an optional value in the STREAM case, can be a static string, the file path or the command used to extract such information.
Regex is the regular expression, in RE2 Syntax[40] (in order to be compatible with golang's *regexp* package), interpreted by the agent to find specific information contained in strings whether in streams, files or commands.
ValidateRegex is the validation field, also written in RE2's Regex syntax. This field allows the user to set a validation method using a regex that targets the extracted information. If the check fails, the agent tries to obtain the information from the list of FallBacks. $-Fallback$ is an optional field that acts as backup in case the regular method of obtaining information fails. More than one fallback may be defined, in case the fallback also fails the validation check. It is recommended that the last fallback be of static type.

3.4.2.C Item Field

```
Item1:
  Format:
  ID:
    GetFrom:
    Value:
    Regex:
    ValidateRegex:
  Meaning:
    GetFrom:
    Value:
    Regex:
    ValidateRegex:
  Content:
    GetFrom:
    Value:
    Regex:
    ValidateRegex:
```

Table 3.6: Layout of every Item Field in this type of file.

The *Item* field, contained in multiple event types, can have its meaning in two formats: ASCII(string) and Binary(Byte Array). Therefore an additional field is required to define which one of the two is used: the Format field. This field can take on two values, ”string” or byte.

3.4.2.D Complete Layout of the file

The Metadata part of the file, as well as the type and severity is exactly the same for all event types. Only the payload’s *Data* field changes per type. It is assumed that each agent runs in parallel with one, and one only probe. This probe may or may not send various types of events.

```
Metadata:
  Origins:
    URI:
      GetFrom:
      Value:
      Regex:
      ValidateRegex:
    UserAgent:
      GetFrom:
      Value:
      Regex:
      ValidateRegex:
    Timestamp:
      GetFrom:
      Value:
      Regex:
      ValidateRegex:
Payload:
  Events:
    - Event1:
      Type:
        Value:
      URIofType:
        GetFrom:
        Value:
        Regex:
        ValidateRegex:
      Severity:
        GetFrom:
        Value:
        Regex:
        ValidateRegex:
      Data:
```

Table 3.7: EventMap file for a general event type.

The *Type* field is required to have one of the following 5 values attributed: Application, Network, Host, CyberPhysical, Other. The *Data* fields for each type are defined in table 3.8, and the field key should be renamed to APPDATA, HOSTDATA, CPDATA, OTHERDATA or NETWORKDATA.

```
Application Event:
Data:
   Assets:
     - Asset1:
       AssetType:
         GetFrom:
         Value:
         Regex:
         ValidateRegex:
       Value:
         GetFrom:
         Value:
         Regex:
         ValidateRegex:
       - ItemID1:
         GetFrom:
           Value:
           Regex:
           ValidateRegex:
   Items:
     - Item1:
       Format:
       Meaning:
         GetFrom:
         Value:
         Regex:
         ValidateRegex:
         FallBacks:
           - FallBack1:
             GetFrom:
             Value:
             Regex:
       Content:
         GetFrom:
         Value:
         Regex:
         ValidateRegex:
         FallBacks:
           - FallBack1:
             GetFrom:
             Value:
             Regex:
.
```

```
Network Event:
   Data:
       Subtype:
         GetFrom:
         Value:
         Regex:
         ValidateRegex:
       Entities:
         Sources:
           - Source1:
             Protoname:
               GetFrom:
               Value:
               Regex:
               ValidateRegex:
             Value:
               GetFrom:
               Value:
               Regex:
               ValidateRegex:
         Destinations:
           - Destination1:
             Protoname:
               GetFrom:
               Value:
               Regex:
               ValidateRegex:
             Value:
               GetFrom:
               Value:
               Regex:
               ValidateRegex:
       Items:
         - Item1:
           Format:
           Meaning:
             GetFrom:
             Value:
             Regex:
           Content:
             GetFrom:
             Value:
             Regex:
             ValidateRegex:
             FallBacks:
               - FallBack1:
                 GetFrom:
                 Value:
                 Regex:
```

```
Host Event:
    Data:
       Items:
         - Item1:
           Format:
           Meaning:
             GetFrom:
             Value:
             Regex:
             ValidateRegex:
             FallBacks:
               - FallBack1:
                 GetFrom:
                 Value:
                 Regex:
           Content:
             GetFrom:
             Value:
             Regex:
             ValidateRegex:
             FallBacks:
               - FallBack1:
                 GetFrom:
                 Value:
                 Regex:
         - Item2:
           ...
.
```

```
CyberPhysical Event:
     Data:
       NodeID:
         GetFrom:
         Value:
         Regex:
         ValidateRegex:
       Location:
         Values:
           - Value1:
             GetFrom:
             Value:
             Regex:
             ValidateRegex:
       Items:
         - Item1:
           Format:
           Meaning:
             GetFrom:
             Value:
             Regex:
             ValidateRegex:
             FallBacks:
               - FallBack1:
                 GetFrom:
                 Value:
                 Regex:
           Content:
             GetFrom:
             Value:
             Regex:
             ValidateRegex:
             FallBacks:
               - FallBack1:
                 GetFrom:
                 Value:
                 Regex:
```

```
    Data:
      Code: #optional
        GetFrom:
        Value:
        Regex:
        ValidateRegex:
      Items:
        - Item1:
          Format:
          Meaning:
            GetFrom:
            Value:
            Regex:
            ValidateRegex:
            FallBacks:
              - FallBack1:
                GetFrom:
                Value:
                Regex:
          Content:
            GetFrom:
            Value:
            Regex:
            ValidateRegex:
            FallBacks:
              - FallBack1:
                GetFrom:
                Value:
                Regex:
        - Item2:
          ...
  - Event2:
  ...
```

Table 3.8: All the Data fields for all types of events possible.

Example files for all types of events are appended to this thesis.

3.5 Data Structures

With the proposed configuration and mapping file formats, the agent needed an appropriate data structure to store the information from them. A total of 34 *Structs* were implemented in GO in an almost identical way the fields in the previous files were formated,

one of them being the struct to store the configuration (debug variables and such).
With the use of the Viper [36] package, it was extremely easy to immediately map the information in the files to the structs, as long as it was defined what variable in the list would store what data. The full list of structures implemented is appended aswell. The Main struct field is , like in the files, a ”FieldStruct” struct.
Three other structs were created to store all the information, now organized, that will be encoded to create the final Event in the Data-Model Format, displayed in 3.5.

```
//Basic Structures for event-fetch mapping (from yaml)
type FallBack struct {
        GetFrom string `yaml:"GetFrom"`
        Value   string `yaml:"Value"`
        Regex   string `yaml:"Regex"`
}

type FieldStruct struct {
        GetFrom       string     `yaml:"GetFrom"`
        Value         string     `yaml:"Value"`
        Regex         string     `yaml:"Regex"`
        ValidateRegex string     `yaml:"ValidateRegex"`
        FallBacks     []FallBack `yaml:"FallBacks"`
}
type ItemStruct struct {
        Format  string      `yaml:"Format"`
        Meaning FieldStruct `yaml:"Meaning"`
        Content FieldStruct `yaml:"Content"`
}
```

Figure 3.4: The three main structs used, from which the remaining 30 are based on.

```
type SendOrigin struct {
        URI       string
        UserAgent string
        Timestamp string
}
type SendEvent struct {
        Type        string
        URIofType   string
        Severity    string
        AppData     SendApplicationData
        NetworkData SendNetworkData
        HostData    SendHostData
        CPData      SendCyberPhysicalData
        OtherData   SendOtherData
}

type DataForMessage struct {
        Origins SendOrigin
        Events  []SendEvent
}
```

Figure 3.5: The three main structs created for information ready to be encoded.

3.6 Usage

The user should configure both the *config* and *EventMap*(this one can be renamed) .yaml or .yml files in a per case manner. The config file is pretty self-explanatory due to the verbosity of values stored in it. If the configuration file is in any directory other than the one the agent is located in, the user should specify its path using the flag *c* when running the agent.

Example: If the agent's location is $/home/$ and the *config.yml* is located in $/home/config/$, the user should start the agent with the command: $./agent - c = "/home/config/config.yml"$ or $./agent - c = "/home/config/config"$

Chapter 4

Tests and Validation

This chapter serves as a form of validation (functional and non-functional) to the work developed by presenting the results, feedback and validation from the ATENA team, and representations of the agent in action. Some benchmarking data was also gathered and is represented after the former results.

4.1 Agent in action

In the following images, we can see the results of the agent's successful work on June 12th, reading from a Snort 2.9.13 probe's output, which was running on a docker container on an Intel NUC, with 6 CPUs, 12 GB RAM and 40GB of disk space. The attack the probe was detecting was an ARP-Based **M**an **I**n **T**he **M**iddle (MITM) attack. The probe administrator configured the agent correctly and the agent began reading, encoding and sending the information to the the specified cluster and topic, which multiple components in the **I**ntrusion and **A**nomaly **D**etection **S**ystem (IADS) platform were subscribed to, allowing them to be fed with information from the security probe integrated by means of the agent that was developed.

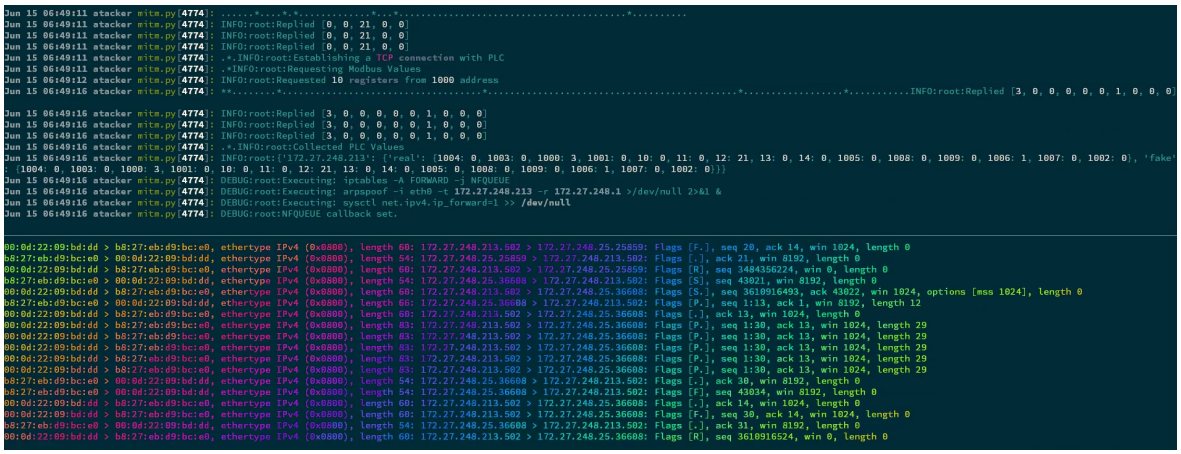


Figure 4.1: Depiction of the attack, ran by a script 'mitm.py'.

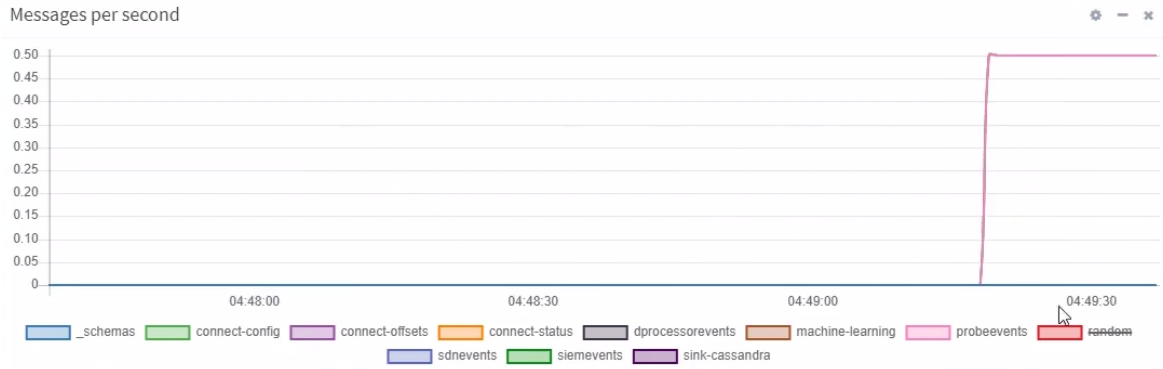


Figure 4.2: Depiction of the graphic of rapidly growing number of events being detected in the platform, from the *probeevents* topic in the Domain Processor, the topic the agent was configured to produce messages to.

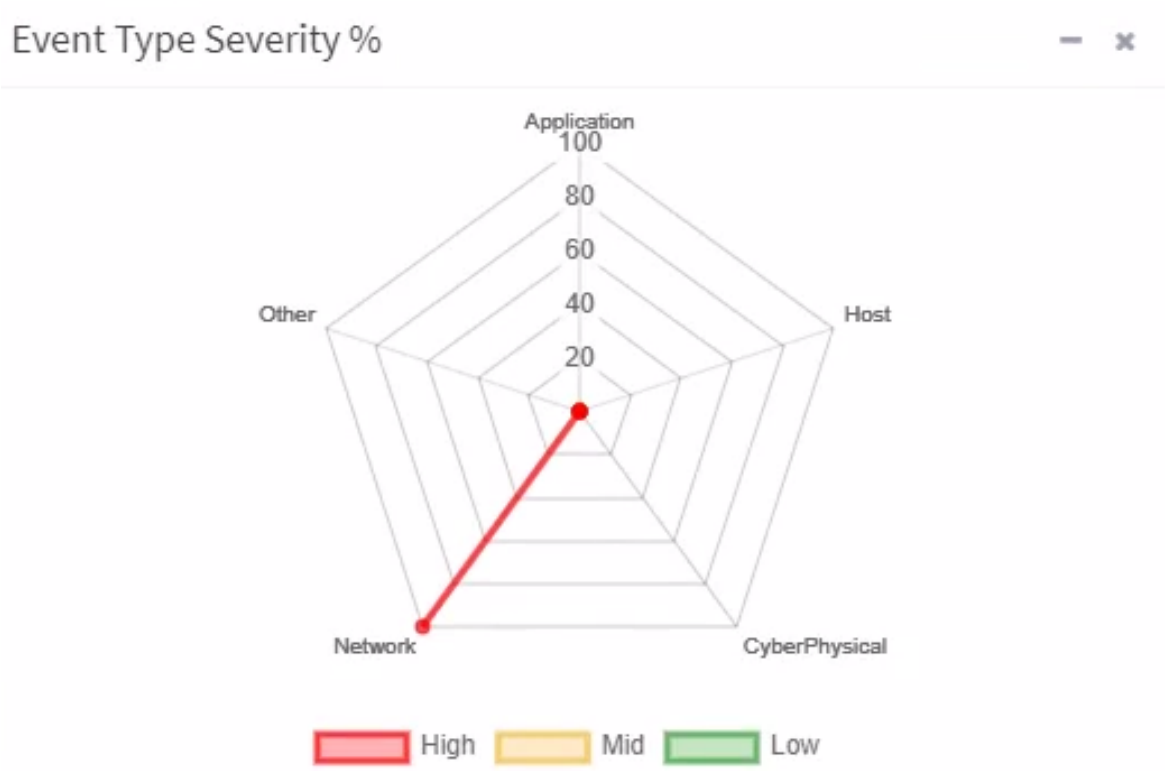


Figure 4.3: Depiction of the number of event severities and types detected, in this case, all of *emerg* severity and of Network types.

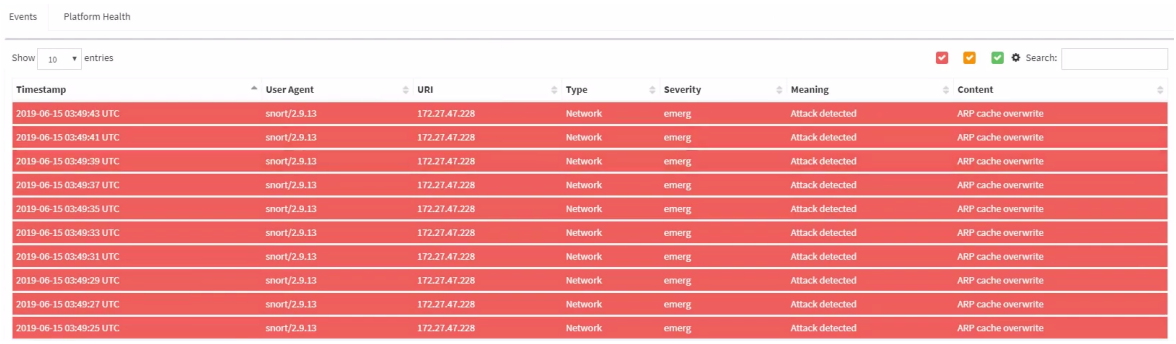


Figure 4.4: The list of events detected.

When the administrator clicks on an event on this list, a full JSON message can be observed in the data model format, and more detailed properties can be read by doing so. Example of an event of this type in image 4.5.

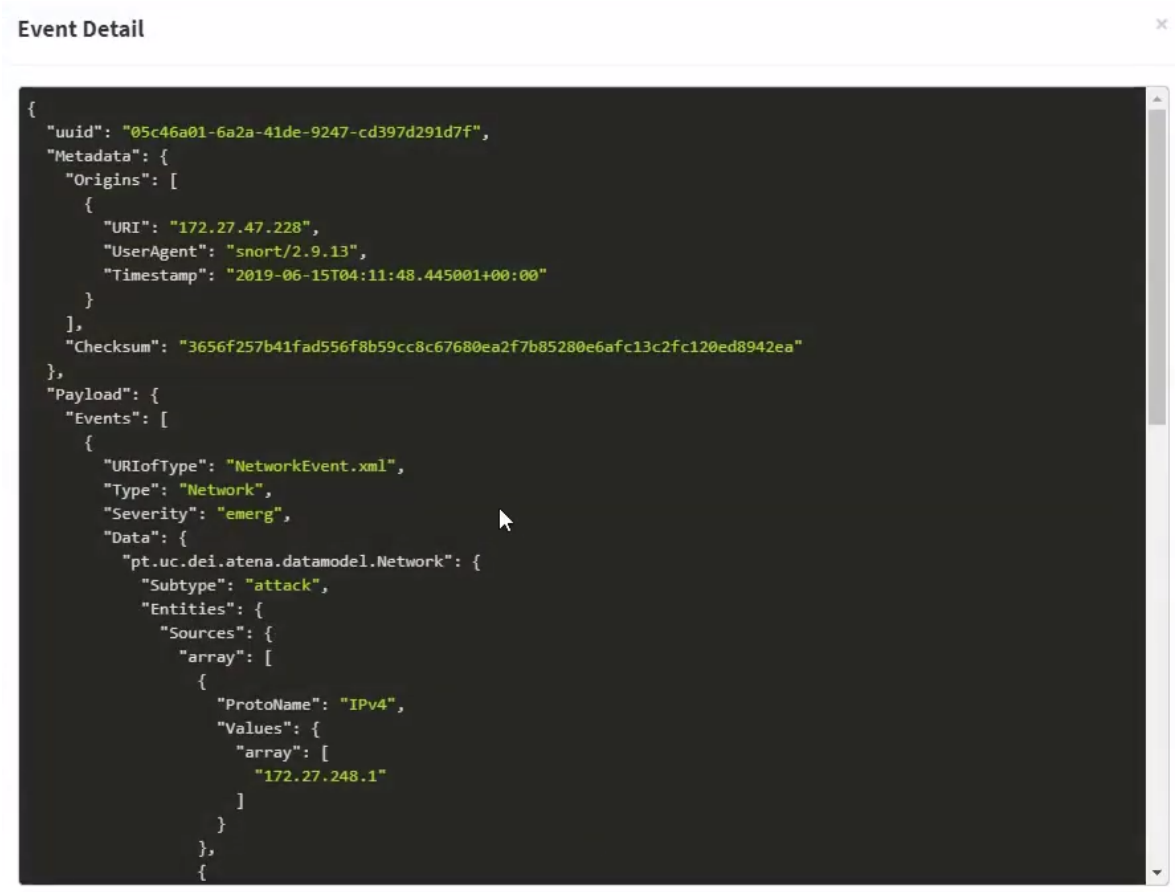


Figure 4.5: JSON Message with the Probe's output, converted into the Data Model.

4.2 Runtime performance evaluation

A diagram of the Agent flow is displayed in figure 4.6, in order to contextualize the reader about the different steps where the timing was measured in this section.

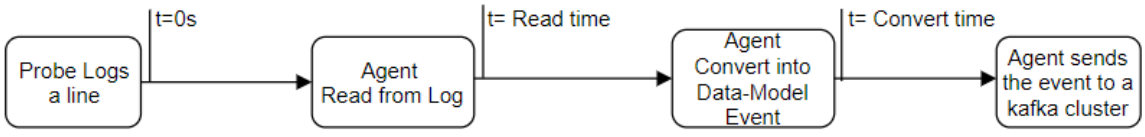


Figure 4.6: Agent Flow diagram.

From a Log File where the Agent was getting its input from, only a small subset of the entries (around 7%) corresponded to actual valid events. The agent would therefore need to separate the valid ones from the non-valid ones in order to prevent the cluttering of trash

in the Kafka cluster, and therefore the IADS platform. It succesfully did so by detecting the invalid lines as garbage and not even trying to encode them, with the right configurations of data to be actually detected and read. A benchmark procedure was undertaken, in order to evaluate the performance of the agent, depicted in figures 4.7 to 4.9. Since *pKafka*, and therefore the sending component, are dependant on network states and inherent bottlenecks, as well as the package being developed by a different developer, the sending times are not measured.

	Input file (log) with 29 lines, 2 valid events. amongst them.		
	Probe	Read from Log	Convert into Data-Model Event
	0	0.581	
	0	0.558	
	0	3.648	
	0	3.735	
	0	0.623	
	0	2.799	
	0	0.747	
	0	1.407	
	0	0.751	
	0	1.528	
	0	0.545	
	0	1.336	
(ms)	0	4.192	
	0	0.478	
	0	1.136	
	0	0.627	
	0	0.915	
	0	0.332	
	0	0.346	
	0	0.837	
	0	1.006	
	0	0.372	0.212
	0	0.749	
	0	1.384	
	0	0.858	
	0	0.341	
	0	1.029	0.165
	0	0.706	
	0	0.597	
Total Avg		**34.163**	**0.377**
Avg		**0.751**	**0.1885**
STD.Dev		**1.034715869**	**0.033234019**

Figure 4.7: Benchmark table of the running time of the agent's multiple components.

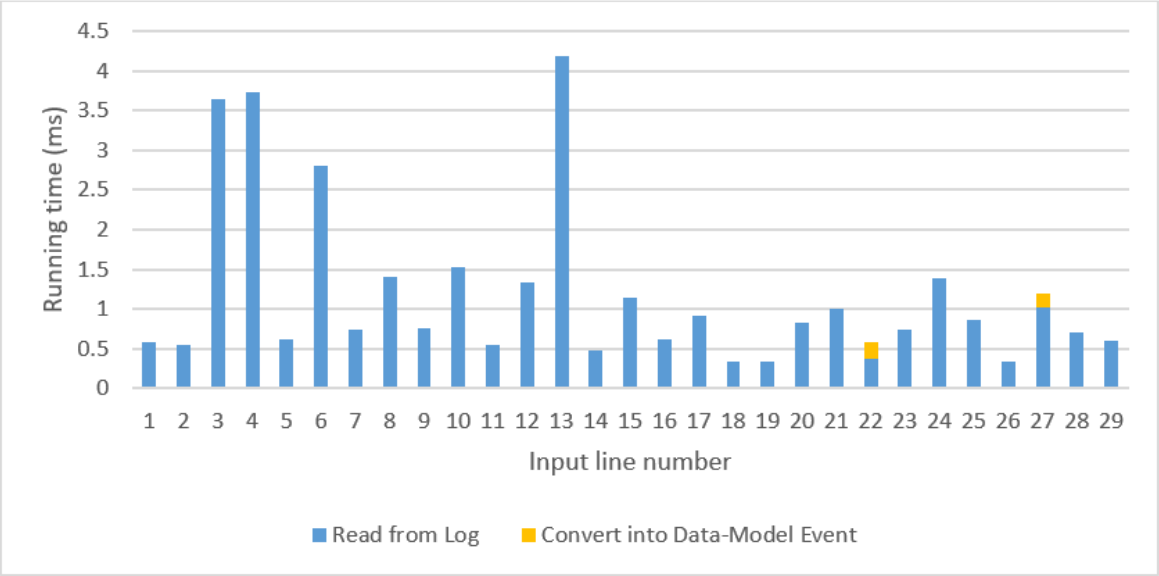


Figure 4.8: Running time (ms) of each agent's main component, per source line.

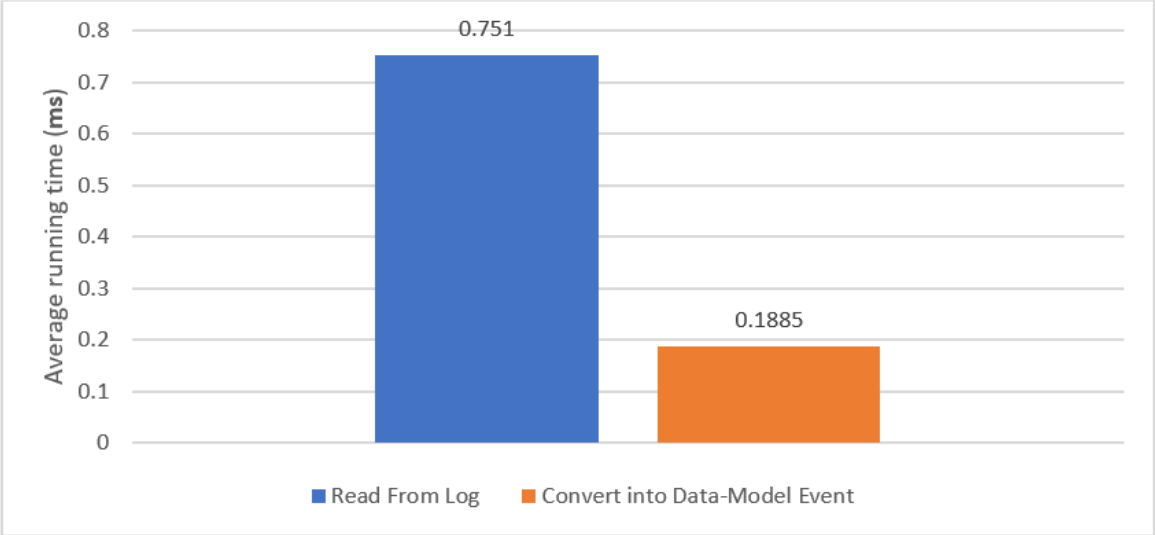


Figure 4.9: Average time(ms) of each main agent's component (excluding kafka sender), on a given valid event line table. Standard deviation was of 1.0347ms on the Read component and of 0.0332ms on the encoding component.

All this data was also sent on the D7.5 contribution, which served to describe the final results of ATENA prototype validation and demonstration of the ATENA prototype. The D7.5 deliverable finalizes the deliverable D7.3 ”Demonstration of the ATENA prototype (Interim DEM)”. The performance of the agent was concluded to be satisfactory, with the added advantage of being multiplexable if required. The possible bottleneck happening on the sender component is something the user or administrator might have problems mitigating, since it is extremely dependant on network conditions and KafKa Cluster configurations, which might be of a third-party’s ownership.

Chapter 5

Conclusion and Future Work

This brief chapter serves not only as the conclusion of this thesis but also as a way to give some insight on future work that might be done if so required. In essence, the agent fulfilled its purpose and, along with the rest of the IADS components, was submitted for review and was validated successfully in the final Prototype validation Delivery. All the work and its code was passed on to the team and further maintenance or stability releases can be made.

If interest arises, the agent developed may be applied to multiple use cases, not only from a security point-of-view, given its inherent generalism. Its components may be used in any case where a user might need to retrieve information from any source specified in this document, such as a *syslog* or a *unixsocket*, and map it to any structure the user configures previously, which might be usefull in another environment. Further work in this agent can also be done, such as the development of an interface that allows for easier configuration of the two files required for the successfull functioning of the software.
This concludes my work as part of the LCT's team, and on this project.

Bibliography

[1] ATENA Consortium. {ATENA:} {A}dvanced {T}ools to ass{E}ss and mitigate the criticality of {ICT} compo{N}ents and their dependencies over {C}ritical {I}nfr{A}structures, {H}orizon 2020 {S}ecure {S}ocieties - {DS}-3-2015 {G.A.} 700581, https://www.atena-h2020.eu, 2016.

[2] Scada example drill by using wincc. URL `https://www.youtube.com/watch?v=mqW7edwe75A`.

[3] IADS dashboard walkthrough. URL `https://www.youtube.com/watch?v=oVOgus5-FLw&list=PLyl2zhykZrH_jltWOC2NKUjChOJ-5o4R1&index=3&t=0s`.

[4] Miguel Rosado Borges de Freitas, Doutor Tiago Cruz Co-Supervisor, and Doutor Paulo Simões. Network Softwarization for IACS Security Applications. Technical report, 2018.

[5] ATENA Consortium. ATENA - Deliverable 6.4 - Design and development report of the 2nd release of components, 2019.

[6] Martin Roesch. Snort-Lightweight Intrusion Detection for Networks. Technical report.

[7] Arthur Jicha, Mark Patton, and Hsinchun Chen. SCADA honeypots: An in-depth analysis of Conpot. In *IEEE International Conference on Intelligence and Security Informatics: Cybersecurity and Big Data, ISI 2016*, pages 196–198. Institute of Electrical and Electronics Engineers Inc., 11 2016. ISBN 9781509038657. doi: 10.1109/ISI.2016.7745468.

[8] ATENA Consortium. ATENA - Deliverable 7.5 - Final ATENA prototype validation and evaluation of validation results (Final DEM0), 2019.

[9] Abdulmalik Humayed, Jingqiang Lin, Fengjun Li, and Bo Luo. Cyber-Physical Systems Security—A Survey. *IEEE INTERNET OF THINGS JOURNAL*, 4(6), 2017. doi: 10.1109/JIOT.2017.2703172.

[10] Filipe Sequeira. *Development and Integration of Specialized Probes for IACS Applications*. PhD thesis, 2018.

[11] L. P. Dias, J. J.F. Cerqueira, K. D.R. Assis, and R. C. Almeida. Using artificial neural network in intrusion detection systems to computer networks. In *2017 9th Computer Science and Electronic Engineering Conference, CEEC 2017 - Proceedings*, pages 145–150. Institute of Electrical and Electronics Engineers Inc., 11 2017. ISBN 9781538630075. doi: 10.1109/CEEC.2017.8101615.

[12] Executive Summary. CockpitCI Cybersecurity on SCADA : risk prediction , analysis and. (285647):1–20.

[13] Marco Castrucci, Alessandro Neri, Filipe Caldeira, Jocelyn Aubert, Djamel Khadraoui, Matthieu Aubigny, Carlo Harpes, Paulo Simoes, Vincenzo Suraci, and Paolo Capodieci. Design and implementation of a mediation system enabling secure communication among Critical Infrastructures. *International Journal of Critical Infrastructure Protection*, 5(2):86–97, 7 2012. ISSN 18745482. doi: 10.1016/j.ijcip.2012.04.001.

[14] Rebecca Bace and Peter Mell. NIST Special Publication on Intrusion Detection Systems Intrusion Detection Systems. Technical report.

[15] Brandon Lokesak. A Comparison Between Signature Based and Anomaly Based Intrusion Detection Systems. 2008.

[16] Christos Douligeris. Network Security: Current Status and Future Directions - Christos Douligeris, Dimitrios N. Serpanos - Google Livros. URL `https://books.google.pt/books?id=dHys9OXMFMIC&lpg=PA86&dq=signature+IDS+disadvantage&pg=PA86&redir_esc=y#v=onepage&q=signatureIDSdisadvantage&f=false`.

[17] Rowayda A Sadek, M Sami Soliman, and Hagar S Elsayed. Effective Anomaly Intrusion Detection System based on Neural Network with Indicator Variable and Rough set Reduction. 2013. URL `www.IJCSI.org`.

[18] Maurizio Martellini. *Cyber and Chemical, Biological, Radiological, Nuclear, Explosives Challenges*. 2017. URL https://books.google.pt/books?id=klE8DwAAQBAJ&lpg=PA31&dq=siem+alarm+filtering&pg=PA31&redir_esc=y#v=onepage&q=siemalarmfiltering&f=false.

[19] Daniel B Cid. Log Analysis using OSSEC. Technical report, 2007.

[20] David Jonathan Day, David J Day, and Benjamin M Burns. *A Performance Analysis of Snort and Suricata Network Intrusion Detection and Prevention Engines Games Physiotherapy for Children with Cystic Fibrosis View project A Performance Analysis of Snort and Suricata Network Intrusion Detection and Prevention Engi*. 2011. ISBN 9781612081168. URL https://www.researchgate.net/publication/241701294.

[21] Host Integrity Monitoring Using Osiris and Samhain - Brian Wotring - Google Livros. URL https://books.google.pt/books?hl=pt-PT&lr=&id=CGE2synNNSEC&oi=fnd&pg=PP1&dq=Samhain&ots=BuujZ7np4p&sig=wHzDVE3PcmlE_O4z7wEJb5IBGjY&redir_esc=y#v=onepage&q=Samhain&f=false.

[22] Filip Holik, Josef Horalek, Sona Neradova, Stanislav Zitta, and Ondrej Marik. The deployment of Security Information and Event Management in cloud infrastructure. In *Proceedings of 25th International Conference Radioelektronika, RADIOELEKTRONIKA 2015*, pages 399–404. Institute of Electrical and Electronics Engineers Inc., 6 2015. ISBN 9781479981175. doi: 10.1109/RADIOELEK.2015.7128982.

[23] T Amick, L Soles, and D Snider. Moving Towards an Adaptive Enterprise Intrusion Detection and Prevention System. Technical report.

[24] Apache Kafka - Documentation. doi: 10.1145/1570000/1563874. URL https://kafka.apache.org/documentation/.

[25] Atena- Deliverable 4 . 8 - Design of the Distributed IDS for IACS (final version). (700581):1–61, 2018.

[26] Luis Rosa. IADS-Events Data Model, Private Repository, 2018. URL https://github.com/lmrosa/iads-events.

[27] Balabit - Syslog-ng, . URL https://github.com/balabit/syslog-ng/releases.

[28] syslogProbe: Probe that consumes events from Syslog and sends them to Kafka, . URL https://github.com/vgraveto/syslogProbe.

[29] Centro Nacional de CiberSegurança. C-Days 2019 - 26JUN - Sala/Room B - Sessões da Tarde / Afternoon Sessions. 2019. URL https://www.youtube.com/watch?v=XHnYrYQcPk0&feature=youtu.be&t=3528.

[30] Sushma Sanjappa and Muzameel Ahmed. Analysis of logs by using logstash. In *Advances in Intelligent Systems and Computing*, volume 516, pages 579–585. Springer Verlag, 2017. ISBN 9789811031557. doi: 10.1007/978-981-10-3156-4{_}61.

[31] Jeffrey E F Friedl. *Mastering regular expressions*. " O'Reilly Media, Inc.", 2006.

[32] João Ferreira. IADS events agent, 2019. URL https://github.com/vgraveto/iads-events-agent.

[33] Charles Anderson. Docker, 5 2015. ISSN 07407459.

[34] Ken Arnold, James Gosling, and David Holmes. *The Java programing language, Fourth Edition*. 2005. ISBN 0321349806.

[35] Version May. The Go Programming Language Specification. 2013. URL http://golang.org.

[36] Spf13. spf13/viper: Go configuration with fangs. URL https://github.com/spf13/viper.

[37] vgraveto/pKafka: Producer for Kafka. URL https://github.com/vgraveto/pKafka.

[38] linkedin/goavro. URL https://github.com/linkedin/goavro.

[39] Apache Avro^{TM} 1.9.0 Documentation. URL https://avro.apache.org/docs/current/.

[40] Syntax · google/re2 Wiki. URL https://github.com/google/re2/wiki/Syntax.

Appendices

A Thesis Planning

Thesis Planning

João Ferreira

Main Goals

This thesis has a main implementation goal:

- **Goal 1 -** The development of a generic adapter agent that should be ran in parallel with any given probe that may or may not exist at the time of writing.

Thesis Milestones

Along with the goals, some set milestones are:

- **Demo 1 - 17/10/18 -** A functional prototype of an agent described in **goal 1** had to be developed until this date. This prototype had to run in two instances alongside two probes, namely a Snort and a ConPot probe. It served it's purpose and some validation input was taken from the team members.
- **Demo 2 - Final demo and Validation - 12/06/19 -** A functional prototype of an agent described in **goal 1** had to be developed until this date. This prototype had to run in two instances alongside two probes, namely a Snort and a ConPot probe. It served it's purpose and some validation input was taken from the team members.
- **Final Thesis Delivery - 31/07/19 or 09/09/19 -** The final written thesis delivery milestone. This is the date of the Normal season for dissertations. If needed, access to the special season can be requested, on the second date mentioned.

Despite this thesis' background in the Department of Informatics Engineering, it is still a dissertation from the Department of Electrical and Computer Engineering and, as such, has no inherent intermediate delivery for evaluation, altough some form of continuous development of the dissertation is encouraged.

Tasks

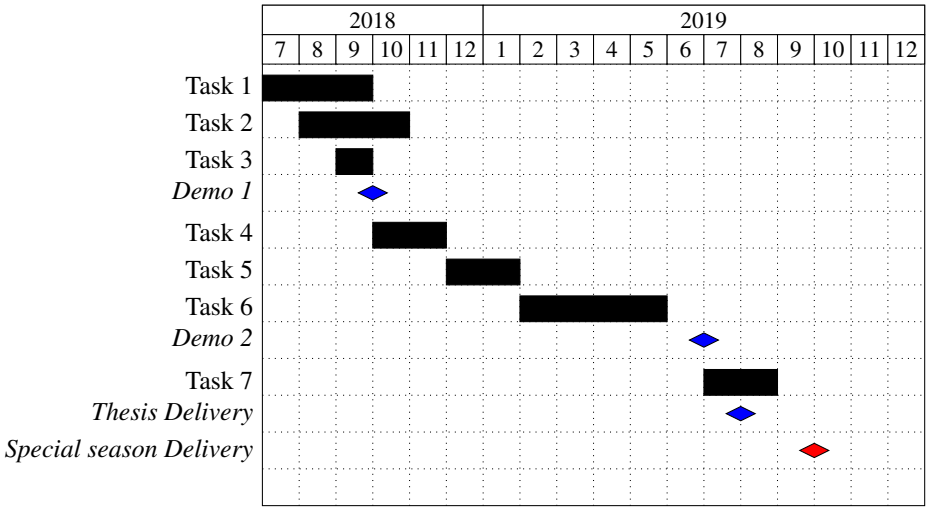
Some defined tasks were/are:

- **Task 1 -** Introduction to the project, team involved, coordinators and past work.
- **Task 2 -** Research on multiple options and paths to fulfill requirements for **Goal 1**.
- **Task 3 -** Development of a prototype for September's demo.
- **Task 4 -** Research of further options according to demo's validation results; start of Thesis' writing.
- **Task 5 -** Further Development of **Goal 1**'s Agent.
- **Task 6 -** Validation of different incremental releases of agent mentioned in above task.
- **Task 7 -** Final Thesis' writing.

B Thesis Planning

Planned schedule

Below is a Gantt chart of the planned and scheduled work, as of 06-11-18. This schedule can suffer changes, and a final schedule will be made in the final writing of the thesis for comparisons.



Overlapping courses

Along with the thesis, I am also enrolled in the following courses:

- **Controlo Digital** - 1st Semester.
  - 1st Theorical Evaluation: 13/11/18
  - 2nd Practical Evaluation: 12/12/18
  - 2nd Theorical Evaluation: 17/01/19

Some time before the dates mentioned above is needed for preparation purposes, which can overlap with the thesis development schedule.

C CyberPhysical Data-Model Payload

```
Metadata:
    Check the Metadata structure here: https://github.com/lmrosa/cpids-events
Payload:
    Events:
        URIofType
        Type (CyberPhysical)
        Severity
        Data:
            NodeID
            Location: (0..1)
                Values (1..*)
            Items: (1..*)
                Meaning
                Content (Binary/ASCII)
```

D Host Data-Model Payload

```
Metadata:
    Check the Metadata structure here: https://github.com/lmrosa/cpids-events
Payload:
    Events:
        URIofType
        Type (Host)
        Severity
        Data:
            Items: (1..*)
                Meaning (syslog, ...)
                Content (Binary/ASCII)
```

E Network Data-Model Payload

```
Metadata:
    Check the Metadata structure here: https://github.com/lmrosa/cpids-events
Payload:
    Events:
      URIofType
      Type (Network)
      Severity (emerg, alert, crit, err, warn, notice, info, debug  - rfc5424)
      Data:
          SubType
          Entities:
              Sources (0..*)
                  ProtoName
                  Value (0..*)
              Destinations (0..*)
                  ProtoName
                  Value (0..*)
          Items: (0..*)
              Meaning
              Content (Binary/ASCII)
```

F Other Data-Model Payload

```
Metadata:
    Check the Metadata structure here: https://github.com/lmrosa/cpids-events
Payload:
    Events:
        URIofType
        Type (Other)
        Severity
        Data:
            Code (Types of HTTP errors to be reading by machines)
```

```
            Items: (1..*)
                Meaning
                Content (Binary/ASCII)
```

G Application Configuration Example .yaml file

```
Source: "FicheiroLogTeste.txt"  #name of file/pipe
Metadata:
  Origins:
    URI:
      GetFrom: "STATIC"
      Value:  "/home/file.txt"
      Regex:
      ValidateRegex: '*'
      FallBacks:
        - FallBack1:
          GetFrom: 'STATIC'
          Value: 'not found'
          Regex:
    UserAgent:
      GetFrom: 'STATIC'
      Value: '/home/file2.txt'
      Regex:
      ValidateRegex: '*'
      FallBacks:
        - FallBack1:
          GetFrom: 'STATIC'
          Value: 'not found'
          Regex:
    Timestamp:
      GetFrom: 'EXEC'
      Value: 'date'
```

```
      Regex:
      ValidateRegex: '*'
      FallBacks:
        - FallBack1:
          GetFrom: 'STATIC'
          Value: 'not found'
          Regex:
Payload:
  Events:
    - Event1:
      Type:
        Value: 'Application'
      URIofType:
        GetFrom: "STATIC"
        Value: 'ApplicationEvent.xml'
        Regex:
        ValidateRegex: '*'
        FallBacks:
          - FallBack1:
            GetFrom: 'STATIC'
            Value: 'not found'
            Regex:
      Severity:
        GetFrom: "STATIC"
        Value: 'alert'
        Regex:
        ValidateRegex: '*'
        FallBacks:
          - FallBack1:
            GetFrom: 'STATIC'
            Value: 'not found'
            Regex:
```

```
AppData:
  Assets:
    - Asset1:
      AssetType:
        GetFrom: STATIC
        Value: "Assetname"
        Regex:
        ValidateRegex: '*'
        FallBacks:
          - FallBack1:
            GetFrom: 'STATIC'
            Value: 'not found'
            Regex:
      Value:
        GetFrom: STATIC
        Value: "plc1.disney.dei.uc.pt"
        Regex:
        ValidateRegex: '*'
        FallBacks:
          - FallBack1:
            GetFrom: 'STATIC'
            Value: 'not found'
            Regex:
      ItemIDs:
        - ItemID1:
          GetFrom: STATIC
          Value: "A1"
          Regex:
          ValidateRegex: '*'
          FallBacks:
            - FallBack1:
              GetFrom: 'STATIC'
```

```
          Value: 'not found'
          Regex:
- Asset2:
  AssetType:
    GetFrom: STATIC
    Value: "IP"
    Regex:
    ValidateRegex: '*'
    FallBacks:
      - FallBack1:
        GetFrom: 'STATIC'
        Value: 'not found'
        Regex:
  Value:
    GetFrom: STREAM
    Value:
    Regex: "`(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)(.(25[0-5]|2[0-4][0-9]
    ValidateRegex: '*'
    FallBacks:
      - FallBack1:
        GetFrom: 'STATIC'
        Value: 'not found'
        Regex:
    ItemIDs:
      - ItemID1:
        GetFrom: STATIC
        Value: "A2"
        Regex:
        ValidateRegex: '*'
        FallBacks:
          - FallBack1:
            GetFrom: 'STATIC'
```

```
              Value: 'not found'
              Regex:
Items:
  - Item1:
    Format: "string"
    ID:
      GetFrom: STATIC
      Value: "A1"
      Regex:
      ValidateRegex: '*'
      FallBacks:
        - FallBack1:
          GetFrom: 'STATIC'
          Value: 'not found'
          Regex:
    Meaning:
      GetFrom: STATIC
      Value: "PID"
      Regex:
    Content:
      GetFrom: STREAM
      Value:
      Regex: '\b Controller ID:\b.*\b'
  - Item2:
    Format: "byte"
    ID:
      GetFrom: STATIC
      Value: "A2"
      Regex:
    Meaning:
      GetFrom: STATIC
      Value: "PID"
```

```
            Regex:
          Content:
            GetFrom: STREAM
            Value:
            Regex: '\b Controller ID:\b.*\b'
```

H CyberPhysical Configuration Example .yaml file

```
Source: "FicheiroLogTeste.txt"  #name of file/pipe
Metadata:
  Origins:
    URI:
      GetFrom: "FILE"
      Value:  "/home/file.txt"
      Regex:
      ValidateRegex: '*'
      FallBacks:
        - FallBack1:
          GetFrom: 'STATIC'
          Value: 'not found'
          Regex:
    UserAgent:
      GetFrom: 'FILE'
      Value: '/home/file2.txt'
      Regex: '\bUserAgent:\b.*\b'
      ValidateRegex: '*'
      FallBacks:
        - FallBack1:
          GetFrom: 'STATIC'
          Value: 'not found'
          Regex:
    Timestamp:
```

```
      GetFrom: 'EXEC'
      Value: 'date'
      Regex:
      ValidateRegex: '*'
      FallBacks:
        - FallBack1:
          GetFrom: 'STATIC'
          Value: 'not found'
          Regex:
Payload:
  Events:
    - Event1:
      Type:
        Value: 'CyberPhysical'
      URIofType:
        GetFrom: "STATIC"
        Value: 'CyberPhysicalEvent.xml'
        Regex:
      Severity:
        GetFrom: "STATIC"
        Value: 'info'
        Regex:
      CPData:
        NodeID:
          GetFrom: 'STATIC'
          Value: "PLC_1"
          Regex:
        Location:
          Values:
            - Value1:
              GetFrom: 'STREAM'
              Value:
```

```
            Regex: '\bCoordinates:\b.\b*\b'
        Items:
          - Item1:
            Format: "string"
            Meaning:
              GetFrom: 'STATIC'
              Value: 'syslog'
              Regex:
              ValidateRegex: '*'
              FallBacks:
                - FallBack1:
                  GetFrom: 'STATIC'
                  Value: 'not found'
                  Regex:
            Content:
              GetFrom: 'STREAM'
              Value:
              Regex: '\bPID:\b\b.*\b'
              ValidateRegex: '*'
              FallBacks:
                - FallBack1:
                  GetFrom: 'STATIC'
                  Value: 'not found'
                  Regex:
```

I Host Configuration Example .yaml file

```
Source: "FicheiroLogTeste.txt"  #name of file/pipe
Metadata:
  Origins:
    URI:
      GetFrom: "FILE"
```

```
      Value:  "/home/file.txt"
      Regex:
      ValidateRegex: '*'
      FallBacks:
        - FallBack1:
          GetFrom: 'STATIC'
          Value: 'not found'
          Regex:
    UserAgent:
      GetFrom: 'FILE'
      Value: '/home/file2.txt'
      Regex: '\bUserAgent:\b.*\b'
      ValidateRegex: '*'
      FallBacks:
        - FallBack1:
          GetFrom: 'STATIC'
          Value: 'not found'
          Regex:
    Timestamp:
      GetFrom: 'EXEC'
      Value: 'date'
      Regex:
      ValidateRegex: '*'
      FallBacks:
        - FallBack1:
          GetFrom: 'STATIC'
          Value: 'not found'
          Regex:
Payload:
  Events:
    - Event1:
      Type:
```

```
  Value: 'Host'
URIofType:
  GetFrom: "STATIC"
  Value: 'HostEvent.xml'
  Regex:
  ValidateRegex: '*'
  FallBacks:
    - FallBack1:
      GetFrom: 'STATIC'
      Value: 'not found'
      Regex:
Severity:
  GetFrom: "STATIC"
  Value: 'info'
  Regex:
  ValidateRegex: '*'
  FallBacks:
    - FallBack1:
      GetFrom: 'STATIC'
      Value: 'not found'
      Regex:
HostData:
  Items:
    - Item1:
      Format: "string"
      Meaning:
        GetFrom: 'STATIC'
        Value: 'syslog'
        Regex:
        ValidateRegex: '*'
        FallBacks:
          - FallBack1:
```

```
                GetFrom: 'STATIC'
                Value: 'not found'
                Regex:
          Content:
            GetFrom: 'STREAM'
            Value:
            Regex: '\bPID:\b\b.*\b'
            ValidateRegex: '*'
            FallBacks:
              - FallBack1:
                GetFrom: 'STATIC'
                Value: 'not found'
                Regex:
```

J Network Configuration Example .yaml file

K Other Configuration Example .yaml file

```
Source: "FicheiroLogTeste.txt"  #name of file/pipe
Metadata:
  Origins:
    URI:
      GetFrom: "FILE"
      Value:
      Regex:
      ValidateRegex: '*'
      FallBacks:
        - FallBack1:
          GetFrom: 'STATIC'
          Value: 'not found'
          Regex:
    UserAgent:
```

```
      GetFrom: 'FILE'
      Value: '/home/file2.txt'
      Regex: '\bUserAgent:\b.*\b'
      ValidateRegex: '*'
      FallBacks:
        - FallBack1:
          GetFrom: 'STATIC'
          Value: 'not found'
          Regex:
    Timestamp:
      GetFrom: 'EXEC'
      Value: 'date'
      Regex:
      ValidateRegex: '*'
      FallBacks:
        - FallBack1:
          GetFrom: 'STATIC'
          Value: 'not found'
          Regex:
Payload:
  Events:
    - Event1:
      Type:
        Value: 'Other'
      URIofType:
        GetFrom: "STATIC"
        Value: 'OtherEvent.xml'
        Regex:
      Severity:
        GetFrom: "STATIC"
        Value: 'info'
        Regex:
```

```
OtherData:
  Code:
    GetFrom: "STATIC"
    Value: "103"
    Regex:
  Items:
    - Item1:
      Format: "string"
      Meaning:
        GetFrom: 'STATIC'
        Value: 'syslog'
        Regex:
        ValidateRegex: '*'
        FallBacks:
          - FallBack1:
            GetFrom: 'STATIC'
            Value: 'not found'
            Regex:
      Content:
        GetFrom: 'STATIC'
        Value: "AAA"
        Regex:
        ValidateRegex:
        FallBacks:
          - FallBack1:
            GetFrom: 'STATIC'
            Value: 'not found'
            Regex:
```