UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

Privacy Aware Sharing of IOCs in MISP

Dissertation presented by
Charles JACQUET

for obtaining the Master's degree in
Computer Science and Engineering

Supervisor(s)
Ramin SADRE

Reader(s)
Antoine CAILLIAU, Alexandre DULAUNOY, William ROBINET , François-Xavier STANDAERT

Academic year 2016-2017

Abstract

Malwares are plaguing this computer age. An actual solution to protect computer systems against them is threat information sharing: Organisations share lists of Indicators Of Compromise with each other. For that, an interesting open source platform called MISP allows organisations to share these IOCs but also malware analyses and attack correlations in an online fashion. On the other hand, offline lookups would also be an important feature but is stopped by the need of data privacy and confidentiality. This master thesis looks for sharing the MISP dataset of IOCs allowing later offline lookups while still protecting privacy and confidentiality by not directly disclosing the information. After a review of the state of the art, a possible found solution has been implemented with some improvements before finally being analysed.

Acknowledgements

First of all, I would like to really thank my master thesis promoter, Professor Ramin SADRE for his advice, his follow-up during the year as well as for his final reading. Equally, I want to thank Conostix which welcomed me during the last Summer holidays and followed my work all year long. I have learned a lot via the master thesis but mostly thanks to the people I met. I would also like to give a special thanks to William ROBINET and Xavier CLAUDE from Conostix who have monitored my work and gave me advice as well as ideas. I would also like to thank CIRCL and especially Alexandre DULAUNOY who followed closely the development of the project. Finally, of course, a big thank to my family for all their support and help they gave me throughout my studies.

Contents

Chapter 1

Introduction

In their last reports [31, 30], the Ponemon Institute conducted analyses over 385 companies in 12 countries[1]. They have estimated the average total cost of a data breach to be around $4 millions in 2016. A data breach meaning that records containing private data linked to a name have been leaked, they also have been more precise by saying that the average cost is about $158 by leaked record. This originates from system glitches, human errors but mostly from malicious and criminal attacks. In France, the percentages are respectively 23%, 27% and 50%. Besides that, they have estimated the likelihood for a company to have a data breach involving 10 000 records in the next 24 months to be around 26%. It means that French companies have a 13% chance of loosing $1 580 000 in the next two years due to a malicious attack. It is also important to know that the cost mostly comes from business losses due to the lack of confidence resulting from a data breach. Unfortunately, the cost of malicious breaches is still growing from year to year and it will probably continue to increase as the global annual cybercrime costs predicted by Cybersecurity Ventures will grow from $3 trillion in 2015 to $6 trillion by 2021.

Companies therefore need to protect themselves against business losses and so against cybersecurity threats by increasing their computer system defences and being able to react quicker to new attack vectors. An important characteristic about attack vectors to notice is that they are generally used against the same type of organisations as they use the same kinds of systems containing similar vulnerabilities. From that, the idea of making correlations between trusted organisations arose and gave birth to threat sharing: as soon as an organisation discovers a threat, they can prevent others to be compromised just by informing them about it.

For helping information sharing, a lot of tools appeared like standards (National Institute of Standards and Technology (NIST), European Union Agency for Network and Information Security (ENISA), ...), guidelines to involve an organisation in information sharing group but also platforms. One of them is called Malware Information Sharing Platform (MISP). It allows organisations to share information like Indicator Of Compromise (IOC), analyses of the compromises and correlations between events. This platform is mostly used in an online fashion with a direct access to the web interface or available Application Programming Interfaces (APIs). But on the other side, this work tries to go one step further to allow directly sharing the dataset using privacy-aware data-structures. It could have a lot of uses like for sharing information between MISP instances (comparable to an IOCs database), for incident response team, in forensics, to allow later offline lookups and so on. A real impact could be seen, for example, with the hunt for traces in an already or potentially compromised system. In that case, computer analysts do not want to connect directly their MISP instance to the system while on the other hand, using privacy-aware data-structure allows getting all indicators from a MISP instance to an

[1] United States, United Kingdom, Germany, Australia, France, Brazil, Japan, Italy, India, the Arabian region (United Arab Emirates and Saudi Arabia), Canada and, South Africa

incident response place. Besides that, with some privacy-aware structure, we can have other properties like fast lookup useful for Digital Forensic and Incident Response (DFIR) teams or like been a compact data-structure which could be useful as some DFIR tools are integrated into embedded systems. Unfortunately, a large wall stands between the idea and its realisation: the combination of privacy and confidentiality needs. This is easily understandable because if companies trust enough an organisation to share private and confidential data about attacks they were facing, they absolutely don't want these information to be shared publicly or to be exposed to situations where an attacker could manage to retrieve them. But on the other side, if another company, not especially as trusted, is fighting against a known attack, there is no problem sharing them information to help them to overcome this situation. A technique is therefore required for **sharing** information without **disclosing** damageable information to other parties. Consequently, this master thesis works on finding a way of sharing an encrypted database of these IOCs contained in MISP to allow offline lookups while protecting the data by slowing down bruteforce attacks in respect to the user specification while still making this database useful for the targeted users. The user specification will thus be a trade off choice between data protection and lookup time.

The organisation of this work is done in two phases, the first one is composed of the four first chapters and is the theoretical part. It starts with a background explanation, a review of the state of the art and ends with an explanation of the concepts used in the implementation. Then chapter 5 will be on going through the different steps realised during the implementation. The following chapter will be on the results to analyse the system in depth before concluding on all this work in the last chapter. The whole project is available on two repositories, my personal repository[2] contains the code, the benchmarks and the report while the MISP official repository[3] contains only the code.

1.1 Organisation of the development

Before going deep into the subject, I want to point out some aspects of this project development. Firstly, last year, I was searching for a security subject but I was not really excited about the proposed ones. On the other side, I had no idea either on what I could work in cybersecurity. Therefore, I have looked for ideas where there are plenty of them, directly in security companies. I have found a company called Conostix which provides security and system services in Luxembourg and they weren't short of ideas. Later on, William Robinet (from Conostix) contacted me again to propose another idea and it was to work on MISP. I have thus met Alexandre Dulaunoy from CIRCL with William and they gave me the starting point of the project which was malware sharing in MISP.

As I had zero knowledge in this domain, I have done an internship of four weeks in Conostix (not credited) where I have learned the basics of MISP. I have also worked on network log parsing. But what I mostly learned was that four weeks for trying to specialise myself in a completely new domain was just enough to give me a rough idea of what it is. During these four weeks, I mostly worked on logs, I have created nothing new, actually, it was quite the opposite but it was still interesting to understand how things were working and why they could be useful. In short, I have started to understand how the proxy squid3 was working and I have installed it on my computer in order to get logs to work on. Once done, I tried to get information from them by manually parsing the logs (which was a mistake). For that purpose, I have started to implement a parser that was using regular expressions to transform each log lines into a set of objects. But even if it was funny to implement, I spent a lot of time on it and each time I finished, I found

[2] `https://github.com/charly077/MISP-privacy-aware-sharing-master-thesis`
[3] `https://github.com/MISP/misp-privacy-aware-exchange`

it to be not good enough, not modular enough, losing locality and ordering information, and so on. I have thus realised that I was loosing my time as of course, they are plenty of tools for this kind of job like the later used Logstash. Then I have started to work with MISP to see how it was working. I have reimplemented a similar hash store like the misp-workbench project. I have then used it with my log system by parsing the data from MISP before adding them to the hash database on Redis. I have then also created a way to feed the parsed logs in my system and briefly analysed them. Besides that, I also had the opportunity to benefit from a one-day MISP training organised by CIRCL. I have then continued to work on my master thesis the whole year long while keeping in touch with William and Xavier from Conostix and Alexandre from CIRCL.

Chapter 2

Background

This small chapter is intended to give a quick overview of the tools and concepts on which this master thesis is based.

2.1 Indicator of Compromise (IOC)

When there is a crime, detectives come and look for clues. They do that to answer important questions as why and how it happened. This is really important, not only for solving a specific crime but also for gathering information that could help solving other correlated crimes and perhaps even to avoid other similar ones.

This is exactly the same idea here: even if the attacker tries to minimise its traces, there are always some. It can be email addresses, IP addresses, malware, URLs and so on. The idea is then to use these traces to trigger alarms on other computer systems as soon as they are seen to protect them as well as gathering new information. This is the reason why we call these type of clues Indicator Of Compromise.

2.2 Confidentiality and Privacy

These two terms are used a lot and nearly in an interchangeable way in this master thesis. But it is still important to look more closely at the difference in the meaning between those two terms at first.

Confidentiality is related to confidence, we entrust others with our information even if they could affect us by revealing it to non-authorised people. Confidentiality is a right in some domain like medicine and so on but is also requested by companies when they accept sharing information.

Privacy, on the other hand, is that we don't want to be investigated more on what we share. This is not because we give the finger that the arm could be taken.

2.3 Redis

Redis[1] is an open source in memory key-value store that can be used as a database, as a cache or even as a message broker.

Redis is highly efficient and allows persistence and replication of data. It handles strings, lists of strings, sorted sets of strings, hash tables, hyperLogLogs and geospatial data. This system

[1] `https://redis.io`

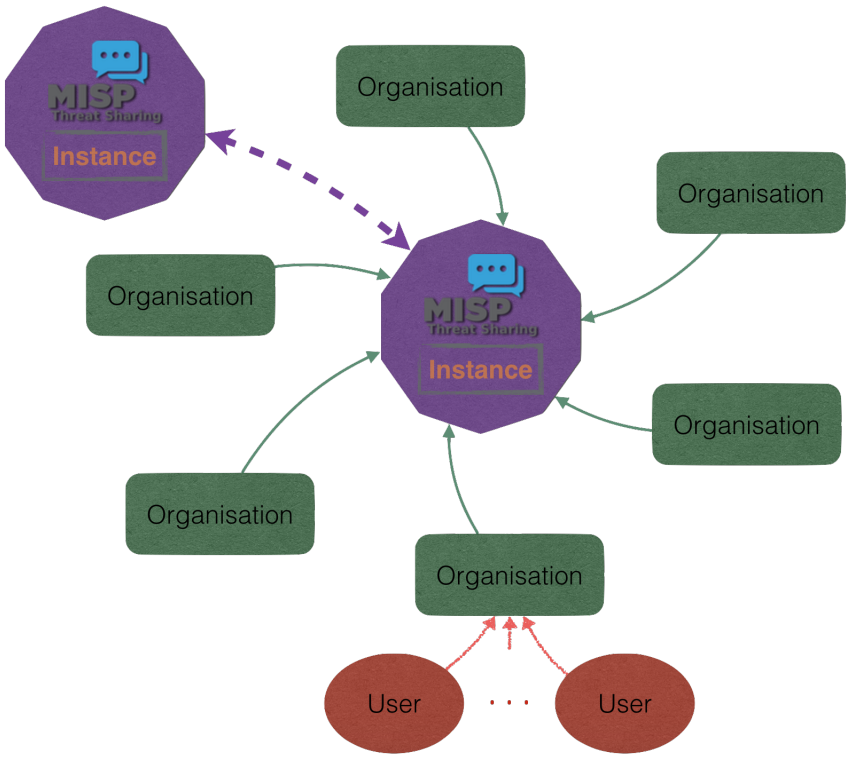


Figure 2.1: Relation between instances, organisations and users

works in a client-server model which is a real advantage as it allows to start one server and to request data from any applications or any process of the same application. Moreover, no needs worrying of complex data synchronisation as transaction operations are directly implemented in it.

2.4 MISP and Threat Sharing

I have introduced the idea of malware sharing to defend computer systems against digital threats and attacks. But MISP is not the only platform to do so, current ones are DShield, Critical Stack's Intel, Microsoft's Interflow, AlienVault's Open Threat Exchange, ThreatExchange (Facebook). But here I focused only on the MISP project which is, as said on their website, an open source software solution for collecting, storing, distributing and sharing cybersecurity indicators and threat about cybersecurity incidents analysis and malware analysis. MISP is designed by and for incident analysts, security and IT professionals or malware reverser to support their day-to-day operations to share structured information efficiently. MISP relies on a hub and spoke sharing model (Trusted Automated eXchange of Indicator of Information (TAXII) [1] explained in section 3.1.3). There is a central point called the hub which is the server responsible for a MISP instance, where all data are kept, and the spokes are the organisations that interact with the hub (see fig. 2.1).

The main advantages compared to the proprietary solutions are:

- Transparency in term of code and of the contributor.
- Quantity and diversity of connected entities.
- The price and the non-existence of entrance barriers.

- The opportunity to extend the MISP core format (financial indicators and mobile phone number are examples of already implemented extensions) with an open source solution where a standardisation will take too much time to be integrated

A recent paper on MISP [40] has been published this year and could be interesting if a better understanding of the underlying system is needed. Additionally, there is also a recent Internet Engineering Task Force (IETF) draft on the MISP core format[5]. I want also to point out that MISP is spreading, and I have the feeling that communities will continue to grow. I have discussed in Belgium with companies like banks, insurances, consultancy companies that are using it, I have even met people knowing it when I was in Singapore for the Black Hat Asia conference (2017). All this is, at least to my point of view, a good sign on the utilisation of the system.

2.4.1 History

As said on the MISP project website, Christophe Vandeplas started the project as he was tired of the way IOCs were shared (email, pdf, ...). Hence he created a cakePHP website as a proof of concept and managed to convince the Belgian Defence (where he was working) that the project was worth it. They even allowed him to work on it during his work-hours. At that time, the project was named CyDefSIG: Cyber Defence Signatures.

Then North Atlantic Treaty Organization (NATO) (OTAN in French) heard of it and got the first presentation in January 2012 to introduce it in more depth and they started to invest themselves into the project. Andrzej Dereszowski was the first part-time developer from the NATO side. One thing led to another and some months later NATO hired a full-time developer to improve the code and add more features. A collaborative development started from that date. It was then collaboratively decided that the project would be released publicly under the Affero General Public License (GPL) license. This to share the code with as many people as possible and to protect it from any harm. The project was then renamed to MISP: Malware Information Sharing Project, a name invented by Alex Vandurme from NATO. In January 2013 Andras Iklody (initially hired by NATO) became the main full-time developer of MISP.

Meanwhile other organisations started to adopt the software and promoted it around the CERT world (CERT-EU, CIRCL, and many others . . .). Nowadays, Andras Iklody is the lead developer of the MISP project and works for CIRCL.

2.4.2 Basics of MISP

The idea of MISP was first to create an IOC database. That is, if there is an attack where IOCs have been seen: "IOC@malware.mail" and "192.168.16.2", we need to keep two types of information, the two IOCs by themselves but also the relation between them. For expressing the later one, IOCs are grouped by event, a more general term than "attack" previously used in this work. Then, the IOCs are considered as the attributes of the specific event representing that particular attack. That representation is quite efficient to analyse as it can immediately correlate events thanks to their attributes.

For clarity concerns, attributes are divided into 13 categories again divided into types. As MISP is not anymore limited to standard IOCs, it is interesting to notice that we can also see categories like Financial Fraud but more standard ones are Network Activities, Antivirus detection, Artifacts dropped, and so on. All the categories and types can be found on their

website[2].

One other really interesting feature is sightings. When an IOC is referenced, we know that it has been seen by someone, but it does not confirm that the IOC is really related to that particular event and is not an error. Moreover, what says that if a dynamic IP address that is an IOC today is still going to be one in 3 months? Sightings, which is the number of times that the IOC sighting has been reported, is thus the solution to this problem as we can monitor an IOC, knowing if it has been seen at other places and if they are still relevant.

There are also two important things on the sharing strategy that I want to point out: MISP instances and communities. The first one is easy to understand, MISP is an open source project which means that everybody can decide to run a completely isolated MISP instance for its own needs. For example, a company that want to store its own confidential data. But that does not mean that the instance must be isolated, they have implemented a way to share information between these instances. While the second one, communities, is a characteristic of each instance. When connected to MISP, a user is connected thanks to the organisation he belongs to (fig. 2.1). Then, the organisation can choose to share or not to share their events/attributes with other communities or even with other instances.

MISP became a really good IOC database that additionally automatically looks for correlations between events. These correlations can be represented under the form of a graph that helps the analyst to visually see connections between events.

Besides all that, a lot of different tools have been implemented like a Python client library and are available in their repository: `https://github.com/MISP/`. They have also added a lot of plugins to the web interface like the ability to export the attributes in different formats, standard ones and some others for specific Intrusion Detection System (IDS).

2.4.3 MISP in a few Pictures and the Traffic Light Protocol

There is nothing better to explain all that than real screenshots of the web application. But, it raised one really important question: what can be shared from MISP and with whom? For example, am I allowed to show screenshots of data from MISP without any risk? For answering that question, I need to introduce the Traffic Light Protocol (TLP) that was created in order to facilitate information sharing by defining the authorised level of disclosure. Therefore, it can be used to know what can be shared with a specific audience. This protocol is defined by FIRST [17] but also explained in this NIST cyber threat information sharing guide [21].

With that in mind I know that I can share the data from the MISP virtual machine as the default Open Source Intelligence (OSINT) data feed used by this virtual instance is tagged as TLP:WHITE. Which means, according to FIRST, that the disclosure is not limited:

```
Sources may use TLP:WHITE when information carries minimal or no foreseeable
risk of misuse, in accordance with applicable rules and procedures for public
release. Subject to standard copyright rules, TLP:WHITE information may be
distributed without restriction.
```

The main page of the web application is a list of all latest events and can be seen in figure 2.2 (notice the "tlp:while" appearing in the tags).

Then, by clicking on an event, we can get information on it (fig. 2.3).

As well as its attribute list (fig. 2.4).

[2] `https://www.circl.lu/doc/MISP/categories-and-types/index.html`

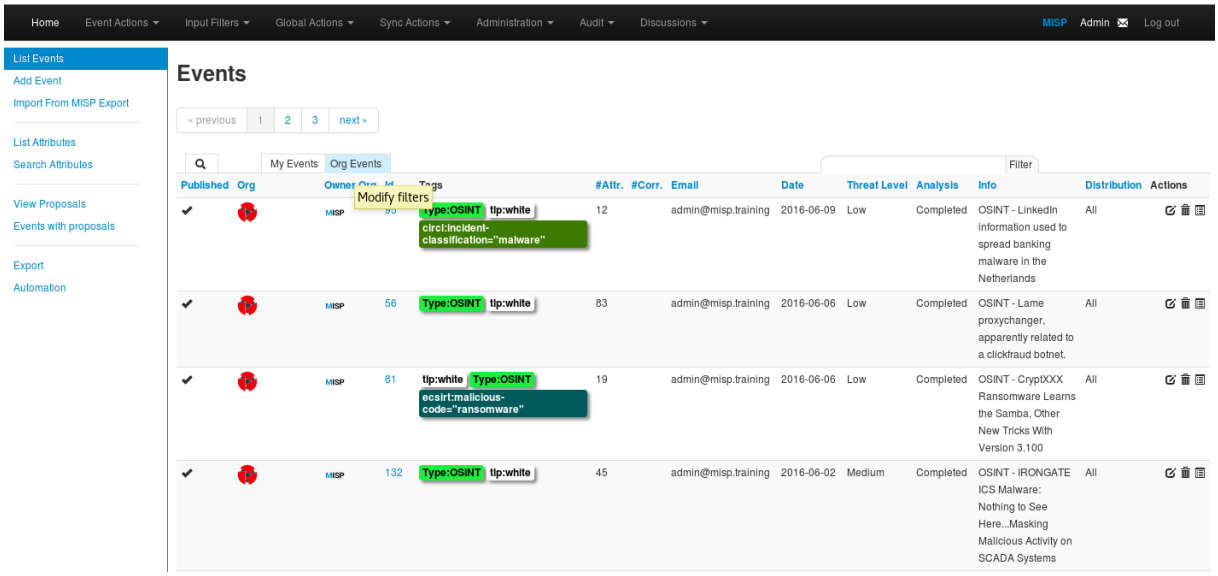


Figure 2.2: MISP: List of events

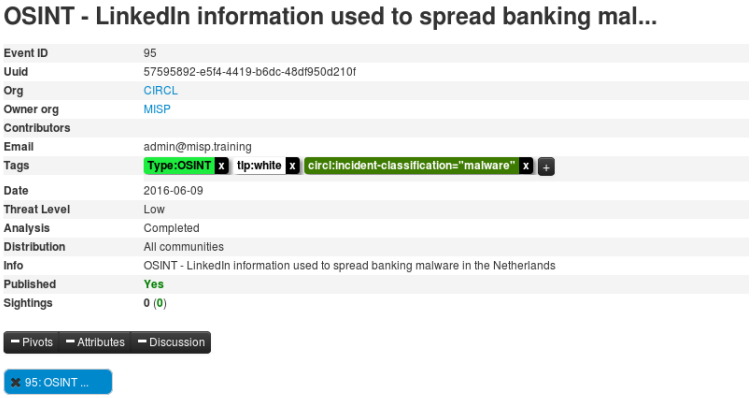


Figure 2.3: MISP: Specific information on the event

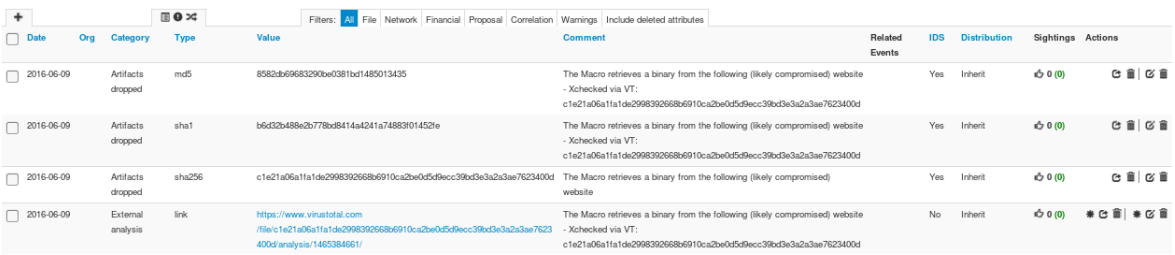


Figure 2.4: MISP: Attributes of the event

And the last thing that I want to show is one of the ways of showing the correlations with other events and is called the correlation graph (fig. 2.5).

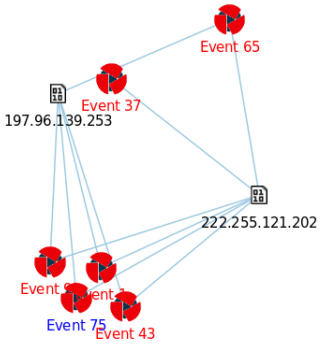


Figure 2.5: MISP: Correlation graph for an event

2.4.4 Use cases

Now that we see more concretely what is information sharing, a good question is to understand how it is used in nowadays systems. In MISP, it has mainly two utilities. The first one is to be able to gather the information of more teams that could work on similar incidents and make them collaborate on their researches. The second one is about directly protecting computer systems by using the information. The user can create rules for an Intrusion Detection System (IDS) like Snort, Suricata or Bro. These IDS rules can be directly generated inside MISP. The contribution of this master thesis would be to be able to create rules for attributes that the user could not directly get back information from the rule but only when needed. These rules could also work in Intrusion Prevention System (IPS) but it is still important to consider the IDS version with the analyse of network logs in the system. This is useful as sometimes the information of an attack is known after the attack and if it went undetected perhaps the attacker is still on the system and we need to know it as soon as possible. Today, the estimation is that the mean time an attacker stays on a system before being detected is of about 200 days which can have dramatic financial consequences mostly due to the loss of clients' trust (Considered as the biggest cost by the Ponemon Institute).

2.4.5 Misp-Worbench - hashstore

MISP workbench[3] is a set of tools to export data out of the MISP MySQL database and use and abuse them outside of this platform.

CIRCL has already implemented a privacy aware tool called hashstore and implemented in Redis. This is working by creating a dataset of all hashed IOCs. And then by the use of the Redis server, we can use a Redis client to request data. Like that, if you want an information on a particular IOC, you need to already know it to be able to take its value's fingerprint and make the request.

On the other hand, this implementation is facing some weaknesses against bruteforce attacks on small data. This problem is well defined in the following chapters and analysed in section 6.4. But briefly, taking the example of Internet Protocol version 4 (IPv4), a bruteforce attack would be composed of only 4 228 250 625 different lookups which is clearly feasible with a tool as fast as Redis.

[3]https://github.com/MISP/misp-workbench

Chapter 3

Information Sharing State of the Art

This chapter is aimed to give an overview of existing works in the two main domains considered in this master thesis. It starts with everything related to information sharing and then it goes through techniques used for privacy concerns.

3.1 Information Sharing

This section is aimed to give a brief state of the art concerning information sharing. It will start from its origins, and then it discusses a little about its different vendors, guidelines and standards.

3.1.1 Beginning of Information Sharing

Gregory White and Keith Harison have done a state of the art in [41] where they explain the evolution of the information sharing by comparing it with the evolutions of the USA laws and their impacts. Even if the analysis is about the USA, I believe it is important to understand the origin of the concept. For an example, to know that the first attempt of sharing standardisation was done just after the worm released by Robert Morris in 1988. By analysing the impact of this worm, they rapidly realised that information sharing could have been a better and faster way to protect critical infrastructures. They also explain, in correlation with the two principal laws called the Presidential Decision Directive-63 in 1998 and the Executive Order 13636 in February 2013, how organisations like Information Sharing and Analysis Organisations (ISAOs) were created with their four foundational objectives that can be summarised as:

- Each organisation should be able to participate.
- Development should be kept public.
- Involvement should always be voluntary and should never be a requirement.
- Take into account the need for confidentiality and privacy.

They also introduce the first sharing program developed by the Department of Homeland Security (DHS) which is the Cyber Information Sharing and Collaboration Program (CISCP). After that, they go through challenges that are facing the ISAOs. The major one is a privacy and confidentially concern, any information that an organisation agrees to share with others, no matter who those others are, need to be kept private and confidential and only released to individuals or organisations that have a right to have access based on the agreements that are signed by members of an ISAO. This concern is really interesting as this master thesis is really on trying to ensure that property without the need of trust. They also explained that privacy is

that personal information about individuals within a member organisation should remain private (in the context of information sharing). While confidentiality refers to information about an organisation that could lead giving others a competitive advantage. In this article, they focused a lot on trust which is one of the most important property to ensure when dealing with a sharing group as we cannot share confidential information if we do not trust the other party to keep it confidential and to respect privacy!

3.1.2 Vendors

After that, a recent survey trying to compare the different vendors for threat information sharing was published in 2017 [33]. The authors have conducted a systematic study of 22 threat intelligence sharing platforms and this article was a short sum up of their key findings. I believe there is a lack of example to support their says, but it is still an interesting survey even if a detailed analysis of the 22 different tools would be really welcome. On the other hand, their analysis of the different types, focuses, standards and licenses used by the different platforms brings a lot of information on what should be used in nowadays sharing platforms.

3.1.3 Standards

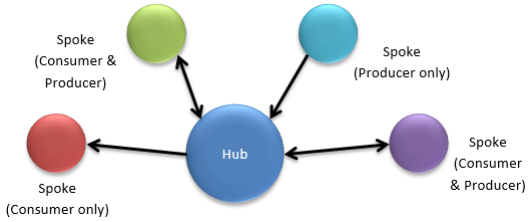
As we want organisations to share threat information, we need them to "speak the same language" or more formally, to use the same standards (lists of standards can be found in [2, 28]). Actually, standards can be categorised into four categories, **Enumerations**, **Scoring Systems**, **Languages** and finally **Transport**. Following the most promising standards according to [18, 33], I will only present the following ones Cyber Observable eXpression (CybOX), Structured Threat Information eXpression (STIX) and MISP-core format for the languages and then TAXII for the transport standard. CybOX, STIX and TAXII have been developed under the coordination of the MITRE Corporation and have very strong momentum in the adoption by industry leaders and threat intelligence communities such as the Financial Services - Information Sharing and Analysis Center (FS-ISAC). The Structured Threat Information eXpression (STIX) [7] provides a language to represent cyber threat information in a structured manner and it provides a structure to express a wide set of contextual information regarding threats in addition of the IOCs. The complete list is:

- Cyber Observables
- Indicators
- Incidents
- Adversary Tactics, Techniques, and Procedures (including attack patterns, malware, exploits, kill chains, tools, infrastructure, victim targeting, etc.)
- Exploit Targets (e.g., vulnerabilities, weaknesses or configurations)
- Courses of Action (e.g., incident response or vulnerability/weakness remedies or mitigations)
- Cyber Attack Campaigns
- Cyber Threat Actors

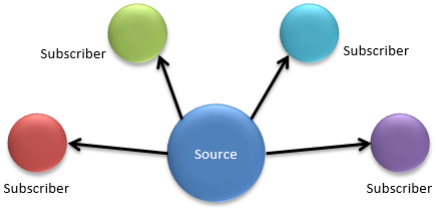
One of its real advantages is that it tries to stay as "human-readable as possible". The complete STIX structure is deeply explained in the ninth chapter of [7]. For expressing the observable, STIX is using the Cyber Observable eXpression (CybOX)[8] that allows to state the

specification of events or stateful properties. At first, CybOX was developed independently but is now integrated into version 2.0 of STIX. It is a structured language for cyber observables.
Then, we need a standardised transport mechanism and for that, we have the Trusted Automated eXchange of Indicator of Information (TAXII) [11, 3] which is a community effort to standardise the trusted, automated exchange of cyber threat information. It defines a set of services and message exchanges that, when implemented, enable sharing of actionable cyber threat information across organisation and product/service boundaries for the detection, prevention, and mitigation of cyber threats.
TAXII is defined around sharing models. There exist three of them:

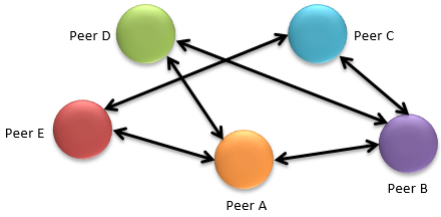
- **Hub and Spoke** is a sharing model where one organisation functions as the central clearinghouse for information, or hub, coordinating information exchange between partner organisations, or spokes. Spokes can produce and/or consume information from the Hub.



- **Source/Subscribers** is a sharing model where one organisation functions as the single source of information and sends that information to subscribers.



- **Peer to Peer** is a sharing model where two or more organisations share information directly with one another. A Peer to Peer sharing model may be ad-hoc, where information exchange is not coordinated ahead of time and is done on an as-needed basis, may be well defined with legal agreements and established procedures, or somewhere in the middle.



On the other hand, in the MISP project, they have decided to support more attribute types and to have an easier model to use. Therefore, A. Dulaunoy and A. Iklody describe an alternative to STIX, the MISP core format, in the second version of the draft[5] while still allowing to directly export data into the STIX format from the web interface. On the other hand, they are using the Hub and Spoke TAXII model for transportation as explained in the introduction with additionally the possibility to have more than one hub (fig. 2.1).

3.1.4 Guidelines

These standards just explained make possible threat sharing but also allow to make it automated thanks to the structure but besides that, when an organisation decides to embark in threat sharing. There is too much at stake than to allow themselves to make mistakes that have

already been made. That is why some organisations like the National Institute of Standards and Technology (NIST) and the European Union Agency for Network and Information Security (ENISA) took care of publishing guidelines to help organisations to involve themselves in threat sharing.

In Europe, the Commission recognised that they have a role to play in making the information more secure. In order to do so, they wanted to create the first pan European Information Sharing and Alerting System (EISAS) on which we can find a report on the implementation in [15]. They consequently have asked to ENISA to define the requirements. Afterwards, member states that weren't using information sharing yet were really interested in and asked ENISA to develop a good practice guide based on the observation of existing exchange. This gave this guide [14] aimed to assist member states but also relevant stakeholders like organisations using communication network and information systems in setting up and running network security information exchange. In this guide, they go through setting up information sharing but they also took care of explaining the advantages as well as identifying laws that could get in the way. This is really important for the organisations to ensure themselves not being recognised by the laws as a cartel or sharing as a commercial advantage. They also focused on the trust that is needed, for them, it must be created thanks to regular face-to-face meetings and TLP must be used as well during these meetings. Then, in 2016, NIST also published its guide [20] that seems to me more complete and more mature. This publication is also more focused on organisation than on states (unlike ENISA) and provides guidelines to improve cybersecurity operations and risk management activities through safe and effective information sharing practices. For them, cyber threat information consists of IOCs, TTP (tactic, techniques and procedures), suggested actions to detect, contain or prevent attacks and finally the findings from the analyses of incidents. They go through different topics like the benefits and challenges encountered, how to establish and participate in sharing relations. Once again, they draw attention to the importance of trust and to be compliant with legal and organisational requirements among other challenges.

3.1.5 Risk of Sharing

On another side, the paper written in February 2017 by Aziz Mohaisen et al. [28] focus themselves on rethinking threat intelligence and to assess the risks. They argue by [27] that intelligence sharing is the only way to combat our growing skills gap in term of security. But they mostly claim that there are a lot of issues that needs to be explored in order to realise efficient and effective information sharing paradigms for actionable intelligence. They even go further by saying that understanding the risk of sharing as well as not sharing is absolutely needed for every organisation. Briefly, not sharing is a risk as we don't receive information that could avoid us to be compromised. While on the other hand, sharing information without proper restrictions may leak a significant amount of information about participants and their operation context and all that can be used by an attacker to learn their vulnerabilities. One of the commonly proposed solutions to that is to have a very limited sharing community only with highly trusted participants. But if the participants are not enough, we could also not succeed in gathering enough information to protect the organisation. On the contrary, if we have less trusted participants, we also need to understand the risk of leakage that could lead to both monetary and reputation losses. Taking all that into account, they propose a new way of thinking for threat intelligence by defining models, communities and adversaries with their threat model. They finally propose an architectural solution as a way of assessing the quality of sharing.

3.2 Privacy-Preserving Techniques

Then, It is interesting to see existing techniques used in information sharing to ensure privacy and confidentiality in a case where we do not want to rely on trust. These following sections are used to discover what is used and what was used before in order to get ideas on what could be implemented.

The first idea is the simplest one and is about removing and hiding parts of the data. This technique is called data sanitization. Then there will be articles on secure two-parties (S2P) Computations and finally on PPRL.

3.2.1 Sanitization

One of the first articles that present some concerns about privacy in sharing security alert is [25]. More precisely, they were concerned about protecting site-private topology, proprietary content, client relationship and site defensive capabilities or vulnerabilities. This was done in two steps, the first one is data sanitization, which consists in removing confidential data and useless information: don't take the risk of revealing information to an attacker if this information is not needed by the user. The second one is the correlation/aggregation work were alerts are linked together for analysis purpose. They have used three different categories of data coming from DShield and Symantec's DeepSight:

- Firewalls: They consider all "deny" as a possible attack
- IDS: They remember logs of attacks that the IDS has found
- Antivirus information

After removing all useless information, we can hash all confidential data. As the companies want to be ensured of the confidentiality, they assume this job of directly hashing the data before sending them to the server. This technique is quite well working if, the data has a certain size. But on the other hand, it is not useful for IP addresses, if an attacker is targeting a company, it has to precompute only 256 or perhaps 65536 IP address hashes. Thus this is not brute force resistant. For each alert, we have two different IPs, the source IP (ip_src) and the destination IP (ip_dest). We can classify all these IPs in two categories:

- Internal IPs: IPs that belong to the company
- External IPs: IPs external to the company

The first category is, of course, the one that we need to protect and in order to do so, these IPs are hashed with a keyed hash function like keyed-Hash Message Authentication Code (HMAC). While the second type is only hashed by a simple hash algorithm like SHA-1. The result is that we can compare all SHA-1 hashed IPs together while only companies can decrypt their own internal IP addresses. An attacker cannot make the difference between an IP hashed by HMAC or SHA-1. Thus, for attacking the dataset, he would need to try to crack all IPs as if they were SHA-1 fingerprints (For example with a TMTO attack). This, without forgetting that they receive millions of IPs, makes it unfeasible for an attacker to brute-force the whole dataset. They are also using another set of protections like the randomised threshold for publication of an alert but it goes out of the scope of this work. In sanitization, they also round all timestamps to the nearest minute in order to add some uncertainty. The second step is the correlation, they spoke about historical trend analyses, source/target-based analyses and event-driven analyses. These analyses worth spending time on them but are not related to the specific subject of this work.

Then, [42] was also working on confidential data sharing starting from the first article, but they came up with a new interesting idea, instead of hashing confidential data, why not generalising it and doing probabilistic correlations. The first idea was a guided alert sanitization with concept hierarchies. For example, if we have an IP 192.168.1.123/32, we can generalise it to 192.168.0.0/16. The depth of the generalisation is chosen thanks to the entropy or by the differential entropy technique explained in [12].
Then, for alert correlation, they focused on defining similarity functions between sanitised attributes and building attack scenarios from sanitised attributes. They also used a technique to create probabilistic attack scenario which is a set of alerts put together to create a bigger attack.

These articles were looking at data obfuscation while still being able to create correlation analyses but, it is difficult to apply it in order to create a database of still usable data for prevention or detection as it looses information.

Sanitization is used to protect information by keeping privacy, but, as [28] is referencing, there are other ways for sanitising data. Some of them are k-anonymity [35], l-diversity [26] and the key privacy guarantee that has emerged is the differential privacy [13]. First, k-anonymity is an attempt to solve the problem of anonymizing a person-specific field-structured data with formal guaranteed while still producing useful data. A dataset is said to have the k-anonymity property if the information for each person in the dataset cannot be distinguished from at least k-1 individuals. This is done by removing attributes or by generalisation as seen earlier. This technique was, unfortunately, performing poorly for certain applications. Therefore l-diversity was an extension of it in order to handle some of the weakness of k-anonymity. It increases groups diversity for sensitive attributes in the anonymization mechanism.

3.2.2 Secure Two-Party Computation (S2PC)

On the other side, we first focus on hiding data while sharing, another point of view would be to be ready for sharing only if a benefit is guaranteed. For that, we make the hypothesis that the benefits that can bring such operations are directly linked to the amount of similar IOCs on each database. Thus we need to find a way to get the intersection or the cardinality of the intersection of these two databases without leaking any other information. Fortunately for us, there exist algorithms for that called Secure Two-Party Computation. With these metrics, we can therefore make a thoughtful decision of sharing or not. The paper written by Freudiger et al.[19] is focused on this problem by working on a DShield dataset. They have experimented some strategies to know if it could be useful to share or not to another organisation. And then, they also experimented to share the whole dataset of the company, only the data set linked to the intersection other dataset or only the intersection (just to get a rough idea of what they have in common ...). Their conclusions were intuitively expected but still interesting:

- More information we get on an attacker, the better the prediction are.
- The chosen collaboration strategy has a really big impact (some of the strategies are really useless).
- Collaboration with companies improves not only the true positive rate (called predictions accuracy in the article) but also removes a lot of false positive.
- Sharing only about common attackers is almost as useful as sharing everything.

This kind of secure two-party computations are used a lot and is also used in some articles for multi-party private matching.

3.2.3 Privacy-Preserving Record Linkage (PPRL)

After all these techniques, Privacy-Preserving Record Linkage (PPRL) gathers a set of techniques (including all previously explained) that are even closer to this work goal. As defined by [39], a recent state of the art, PPRL aims to address this problem [concerning impede sharing or exchanging data for linkage across different organisations] by identifying and linking records that correspond to the same real-world entity across several data sources held by different parties without revealing any sensitive information about these entities. Even if it is more used in public health surveillance, it is also used for crime and fraud detection. The article is a long survey of 36 pages that first explains the background knowledge required to understand the applications, processes, and challenges of PPRL before presenting all existing PPRL approaches and researches. This article presents nearly all what would be needed to achieve this work goal in different manners and could bring a lot of different ideas to implement MISP sharing in other privacy aware manners. On the other hand, this master thesis focused on sharing directly the information to gather a protected dataset while PPRL is more focus on direct record linkage by linking online databases of different organisations. Some of these approaches are really important as well and should be considered as further works.

3.3 Discussion

Back on this sharing problem that is being tackled in this work, there is a dataset of malicious data, IOCs, events coming from MISP and we want to share them in a privacy-preserving way and moreover, we want to limit the need of trust in sharing. Computer specialists would be able to check on computers to discover infections, problems and moreover learn how to fix them thanks to previous analyses contained in MISP. The problem is that some information are confidential and we need to have some confidentiality concerns while sharing.

We also know that, as we want the database to be given, an attacker, who have succeeded in getting the encrypted database, could retrieve the information from it by bruteforcing it. Meaning testing every possible element up to find the ones contained. We therefore need to make it so difficult that it could be considered as unfeasible. Unfortunately, small data like IPv4 are a problem by the number of possible values which is of 4 billion. Bruteforce is therefore feasible for the actual computer power. This means that we need to make it really harder but making the process harder makes it harder for a normal user to request the database as well. We thus already knows that we are going to need to find a trade off. And moreover, the goal would be that the user could choose the tradeoff he wants in function of the confidentiality of the data.

Then, it was really useful to see these existing techniques but it unfortunately does not mean that it could solve all of our problems. They had other purposes when designing their solutions and it can lead to some disadvantages in what we are attempting to do. Sanitization is a good example as it is really useful for the existing implementation but in this case, it would modify data up to make them unusable for everyone. Other sharing strategy needs server always up which is not interesting either so we still need to find a way of sharing only if the user has knowledge of the event and can share information on it as well, or is really infected by and need help that could be given by the information.

Chapter 4

Implementation Concepts

The general theoretical part ends for the real concepts used in the master thesis. First, some concepts used are explained in order to understand their weaknesses but mostly their strong points that could be useful. Afterwards, a small discussion leads to the real implemented solution before explaining the system.

4.1 Bloom Filters

Bloom filter [9] is a space efficient probabilistic data structure used to efficiently test the membership of specific values. It has an enormous advantage which is to be able to test very quickly for membership without requiring to store the complete set of data. On the other hand, even if this data structure never forgets a member, it can falsely believe in the membership of a value even not related to it. The rate to which a data is wrongly recognised is called the False Positive rate and will be explored later on. This small example of a small dataset of small data already shows the memory advantage: the data set of all the students' name and surname at Universite Catholique de Louvain (UCL). Taking the hypothesis that each name and surname has approximately 6 characters and knowing that there are about 29 933 students in 2015-2016. The dataset should use $29\,933 \cdot 6 \cdot 2 \cdot 8$ (=2873568) bits while a 1% false positive bloom filter could be only around $29\,933 \cdot 10$ bits which is already about 9 times smaller.

4.1.1 Data Structure

A bloom filter is a m bits array (fig. 4.1) with all bits set to 0 at the starting point. Besides that, there are also k independent hash functions.

To add the value v1, each hash function gives a fingerprint of the value which is considered as an index in the array where the value needs to be set to 1. In the opposite, if we want to test a value for membership, we have to check all indexes given by the hash functions. As we can see on figure 4.1, v1 is in the set while v2 is not.

4.1.2 False Positive Rate

Bloom filter is a probabilistic data structure as when we are checking for a value, either the value is **not** in the set or is **perhaps** in the set. In the latest case, this is due to false positive where all the k hash functions give indexes already set to 1 while the element should not be in the set. An example can be seen in figure 4.2 where v3 do not belong to the set.

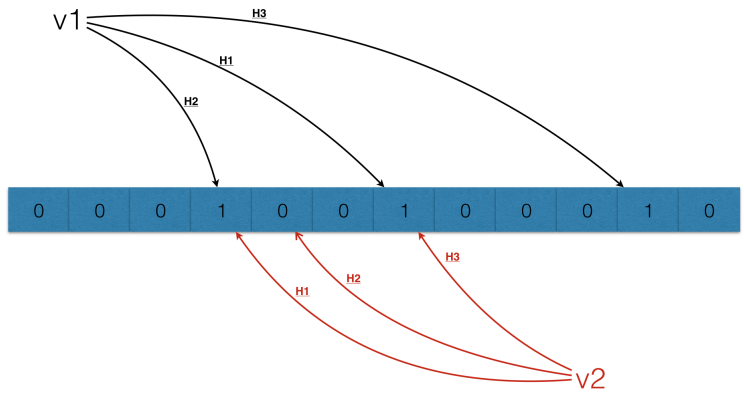


Figure 4.1: Bloom Filter with m=12 and k=3

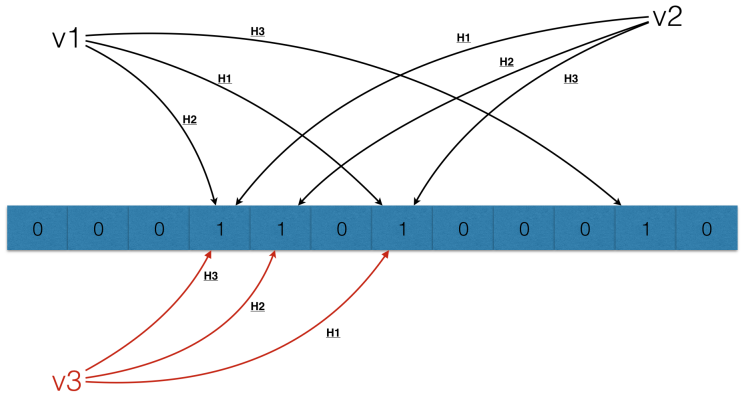


Figure 4.2: False positive v3 in a bloom filter with m=12 and k=3

4.1.3 Control the False Positive Rate

If we check for a value in a bloom filter, at the end, we only know if it is not in the set or a probability to be in the set. Playing with this probability could be interesting for hiding some information and can be a really important feature for privacy and confidentiality. This is why it could be useful to get more information on the false positive rate and how we could use it.

If n is the number of elements inside the data structure, m the number of bits in the array and k, the number of hash functions, we can approximate[1] the false positive rate (p) by $(1 - e^{\frac{-kn}{m}})^k$ but a more useful way is to define k in function of the willing rate for the false positive. In this case, we can notice that k is not dependent on the number of elements inside the set while it is m who needs to be modified in function of the number of values n:

$$m = -\frac{m \ \ln(p)}{\ln(2)^2}$$

$$k = -\frac{\ln(p)}{\ln(2)}$$

(These are approximations)

[1] https://en.wikipedia.org/wiki/Bloom_filter

4.1.4 Information Leaked by Bloom Filters

It may seem useless but nevertheless, it can have huge impacts. If we create bloom filter and we ensure a false positive rate, with the size of the array (m), the number of hash functions (k) and the number of bits set to 1 (X). An attacker can approximate the number of values in a bloom filter with the formula shown by Swamidass et al. [34]:

$$n* = -\frac{m}{k} \ln \left[1 - \frac{X}{m} \right]$$

This does not bring a lot of information and the number of elements in the set is leaked as well in the final solution but it is important to have a knowledge of that while implementing a system based on those solutions.

4.1.5 Bloom Filter alternatives

There are also other alternatives to bloom filters like the ones explained in [32] but also one particularly interesting called Cuckoo Filters [16] that could be considered as a further work. A Cuckoo Filter is a compact variant of Cuckoo Hash Table [29] that stores only fingerprints for each item inserted. This data structure has four big advantages, the first one could be helpful for the system updates (section 5.13) as it is that values can be dynamically added and removed. After that, this data structure is also more efficient in space as well as it provides better lookup performance (in most of the cases). Finally, it is easier to implement.

4.2 Secure Multi-Party Computation

Each organisation possesses its IOC database and is not ready to share all their information. But, once again, if it could help another organisation to solve its problem, they agree on sharing only these information. That is where Secure Multi-Party Computation could play a role as there are techniques to find the intersections of 2 datasets. This is also called private matching [6, 24].

This technique has only a big disadvantage which is heavy computations on both sides. But still, it could be very useful to share information between MISP instances.

4.3 Proof of Work Database

This idea is to slow down brute force attack thanks to proof of work. Even if this idea is one step closer to the final implementation, it is still different by the need of a server to communicate with for each request.

Here, we want to keep a database, we don't want any false positive nor false negative. But by doing that, we make our database available to brute force attacks. We just need to add a new wall of time-consuming computations between the user and the dataset for the requests.

A possible implementation could be with a random key chosen by the database server. This key could be regenerated with a small period and also every thousand[2] requests. On the other hand, the client should bruteforce this small key each time it has changed. This slows down brute-force attacks while still be useful for a user that is doing less than one thousand requests to test its systems. For example, a possible request message for an IP could be: hash(IP)||hash(IP||key)

The idea is interesting even if it is still not bullet proof against brute force attack. It also needs a server to respond to each request.

[2]This number can be modified with the user will

4.4 Discussion

I have explained some basic ideas that could lead to different implementations, a lot of work are done especially on private multi-party matching that could lead to the best options if requests are rare or, for example, if it is used to periodically compare and synchronise data related to the same events between MISP instances. On another hand, nearly every solution still needs a server that we would like to get rid of (after the data have been generated) and for that, bloom filters are the right track but there is still a lack as we do not get back information. That is why the final implemented solution will find something similar to the proof of work to slow down the attack and the bloom filter to increase the system speed.

For that, a nice article [38] was published in October 2016 where they have found a technique that could also work here as their goal was similar: allow parties to share information without the need to immediately reveal private information. Here, I will make an implementation working with MISP and improve its matching properties and the modularity of the code.

4.5 Private Sharing of IOCs and Sightings [38]

This paper considers a cryptographic approach to hide the details of an indicator of compromise. The authors consider two different phases, sharing these IOCs and privately reporting the sightings of IOCs.

Reporting the sightings is important for monitoring the events and attributes in MISP. Moreover, MISP is now able to remember a sightings number per attribute but their technique seems difficult to implement in MISP for a standard subscriber as we should go through each attribute for each instance and each reporter while being online periodically (not practical at all). I have got some other ideas but I finally only focused myself on the sharing part as I would have not enough time. But still, it could be the most important and really useful further work to do.

They also started by warning the reader that using these cryptographic function is better than not using them at all but it is not a miracle technique as it is theoretically impossible to hide the IOC's content in the used context (subscribers receive the data to check on theirs systems). Additionally, it has a performance cost but in my beliefs, it is well worth it. Their implementation is in the source-subscriber TAXII sharing model but it can nevertheless fit with our model as we are here not considering the fact that users can report new events and attributes. First, they define an IOC as propositional formula where the propositional variables are defined over features or observables like Internet Protocol (IP) or fingerprints of malicious programs (but it could be any IOCs contained in MISP). They also claim that every IOC can be expressed in the Disjunction Normal Form (DNF) without any negation (e.g. destIP= 198.51.100.43 $\wedge$ destPort = 80). Hereinafter the values of the rules generated are considered as the concatenation of all values of the IOCs that are linked together: IPv4||port_number (fig. 4.3). It has two advantages, the first is to represent all linked values directly together and then, increasing the size and the number of elements hashed increases the difficulty for an attacker to perform a bruteforce attack or to precompute all possible hashes.

Rules, as created, can be directly checked on computers but they do not fit with our model yet as every value is still in clear. To solve this problem, they decided to hash the values concatenation while still keeping the value types (e.g. ip-dst||port) in clear. The value types are what make the rules still usable as it says the user which are the values they need to try against the rule. Therefore there are only additional computations when the user needs to hash his values each time he tries to match the rule. In this way, we are still really close to the hashstore previously

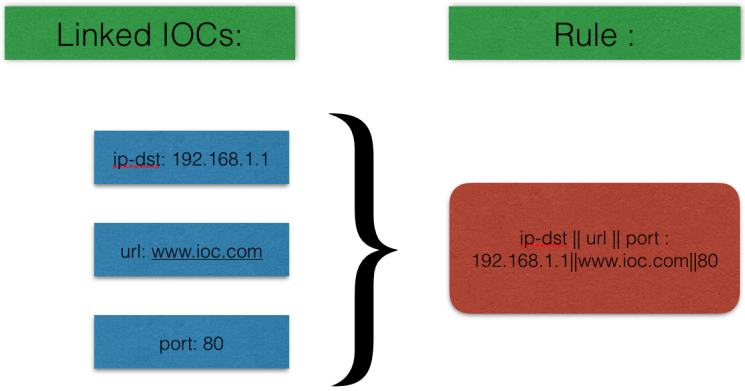


Figure 4.3: IOCs transformed into a rule

explained but with the advantage of being able to individually store and share rules. We still lack some protections since every individual variable can only take a limited set and we still do not get back information while there is a match. For solving that, they are using a non-secret salt value chosen at random for **each** rule (IOC). Besides that, the attack time can be increased as well thanks to the cryptographic hash function used. The second part is really affecting the honest subscriber performance as well but is needed to slow down enough an attack for small data.

They didn't stop there, considering all these ideas, they have added the possibility to include a Course Of Action (COA) message with each rule. This is the real advantage compared to other techniques as the COA gives the user the actions he needs to apply in order to defend himself. The output is precise, there are neither false positive nor false negative and additionally, a match can give information to the honest user. In this work, the COA will be replaced by a message composed of identifiers allowing the user to get more information via the web interface. To do so, instead of hashing the combined values, they use it as the input of a KDF that generates a key used for encrypting a message. Thus this new type of rule, instead of just obfuscating the data, contains the type of value (as before), the salt used for the rule and the encrypted message (fig. 4.4).

Trust is one of the major problems in information sharing and is discussed in all standards, guides and articles. Here they have an attempt of solution that could really work. As a user cannot normally decrypt the complete rules data set, if for each generated rule, we add a source identifier in the KDF. This results in the ability to track the origin of a leak. Even if it is a small addition, I'm intimately convinced that it could help organisations to decide to share more confidential data as they could turn themselves against the leakage source in case of problems which could limit the cost as well as the reputation damages. Moreover, in MISP there is also a user identifier that could be used and which is called a Token.

This pseudo code shows how to generate a rule from the IOCs:

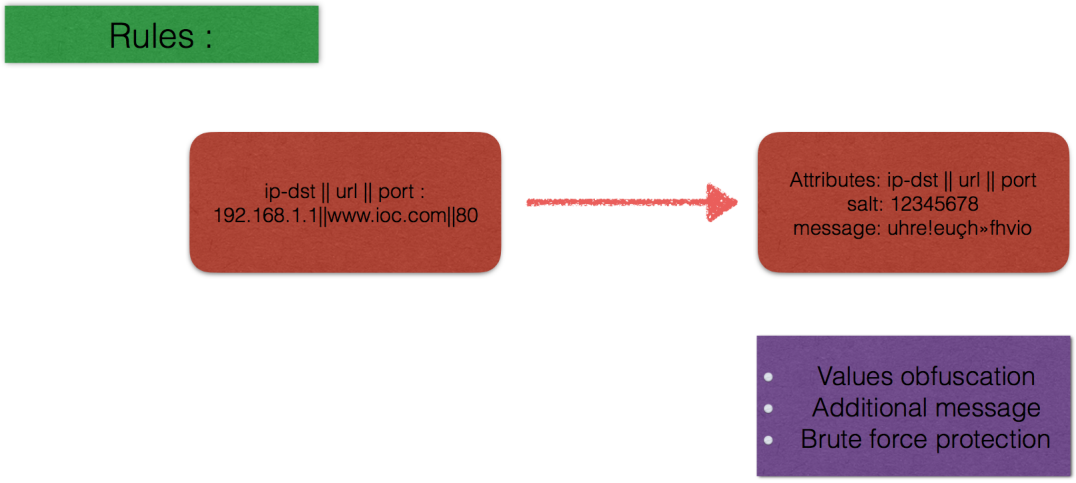


Figure 4.4: New format for the rules

```
Function createRule(AttributeTypes, values, message, Token):
        pass = concatenate(values, separator='||')
        salt = GenSalt()
        key = kdf(token + salt + pass)
        message = repeatNullByte(16) + message
        encr_message = encrypt(message, key)
        return dictionary(attributes=AttributeTypes,
                          salt=salt,
                          ciphertext=encr_message)
```

Then, this pseudo code explains how to check values against one rule:

```
Function matchRule(dictionaryValuesToCheck(type -> value), rule, token):
values = dictionaryValuesToCheck.get(rule.attributeType)
        pass = concatenate(values, separator='||')
        key = kdf(token + rule.salt + pass)
        if (decipher_16first_bytes(rule.message, key) = repeatNullByte(16)):
                return decipher(rule.message, key)
        else
                return notMatched
```

In their implementation, they are using both HMAC and PBKDF2 for the key derivation function. I had the opportunity to exchange emails with the main author of the paper that prefers the HMAC algorithm as it is designed to give a pseudo random key in just a single iteration and that it has been deeply analysed [22]. While PBKDF2 is generating random keys that become more secure as the number of iterations increases. Moreover, the behaviour has not been described (to the best of our knowledge) in academic paper even if there is a NIST recommendation on its usage. They also used an Advanced Encryption Standard (AES) encryption scheme to encrypt the CAO message.

Their proof of concept code is available on their repository, `https://github.com/CRIPTIM/private-IOC-sharing`.

Chapter 5

Implementation

The article of Tim van de Kamp et al. gave me the starting point I needed as the code was available and under the MIT license. I have thus created two basic scripts starting from theirs where I have written them back to fit with MISP. The first step was to retrieve data from MISP, either from the web API, by a direct connection towards the MySQL database or directly via CSV files. Then I have improved the code with different new functionalities and realised that the project often needed to be completely refactored to allow additional features and to keep it simple for an external user to understand how it works. I will thus go through theses different implementation parts in the following few sections.

To help the understanding of this chapter, figure 5.1 shows the different interactions between the different existing main scripts (dark blue), helper scripts (green), folders (red) and MISP (blue). The complete explanation of the data flow will follow in the next chapter as well as two sequence diagrams explained with PBKDF2.

Then, figure 5.2 shows the general way of using the system. First, the rules are generated. Then a user that has data to check can use these rules. The output of the system is the data that have matched with their corresponding message. This message allows the user to retrieve all information he needs from the MISP instance.

5.1 Get data from MISP

To create the rules with the encrypted messages, we first need to get the subset of IOCs contained in MISP destined to IDS. For doing so, there are tree possibilities, the first one is working with the MISP web API, the second one is to be directly connected to the MySQL database while the third one directly reads from a CSV file located in the res folder and allowing more external regulations. The recommended mode will be with the web API, but as rules will be generated directly on the server, it could be more efficient to use directly the MySQL connection.

For making automated request on MISP, CIRCL has implemented a Python library called PyMISP. The first idea was to use it but it was finally not interesting to import the whole library only to make one request. That is why, I have changed the system by making myself the requests to the automation API by using the 'requests' Python library and the MISP token referenced in the configuration file.

The MISP virtual machine has been of a great utility to understand how the MISP system is working, and especially how the database is structured. I have also added an additional check like verifying that the email address specified is the one related to the token when the MySQL connection is used.

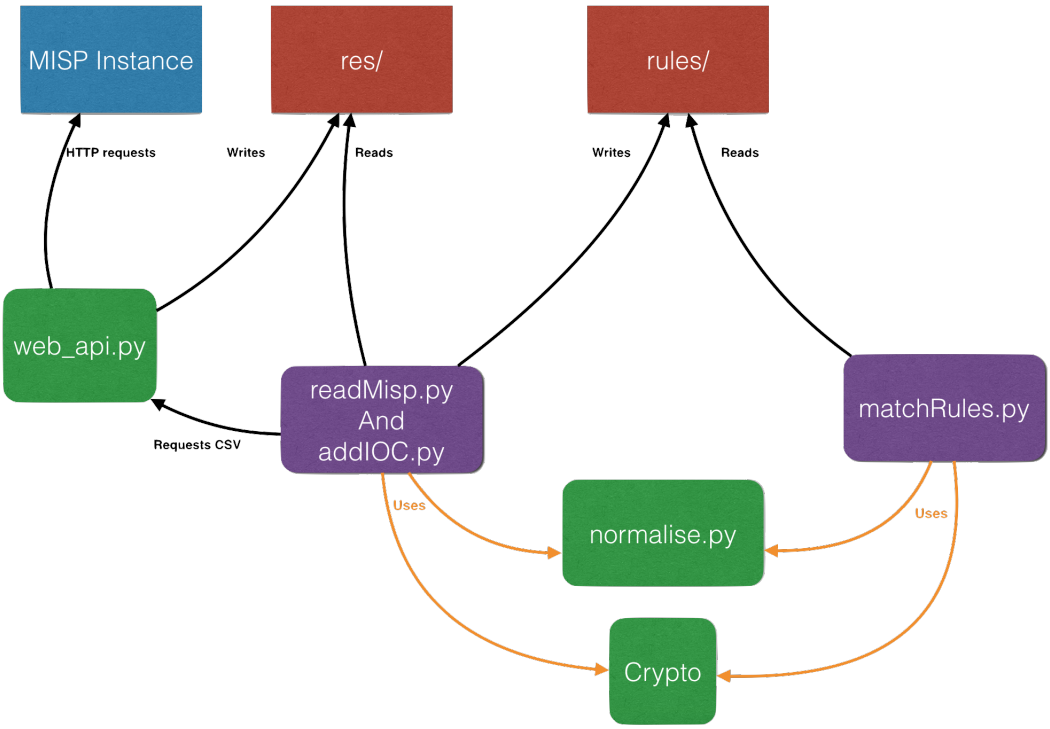


Figure 5.1: Interactions between scripts, folders and MISP

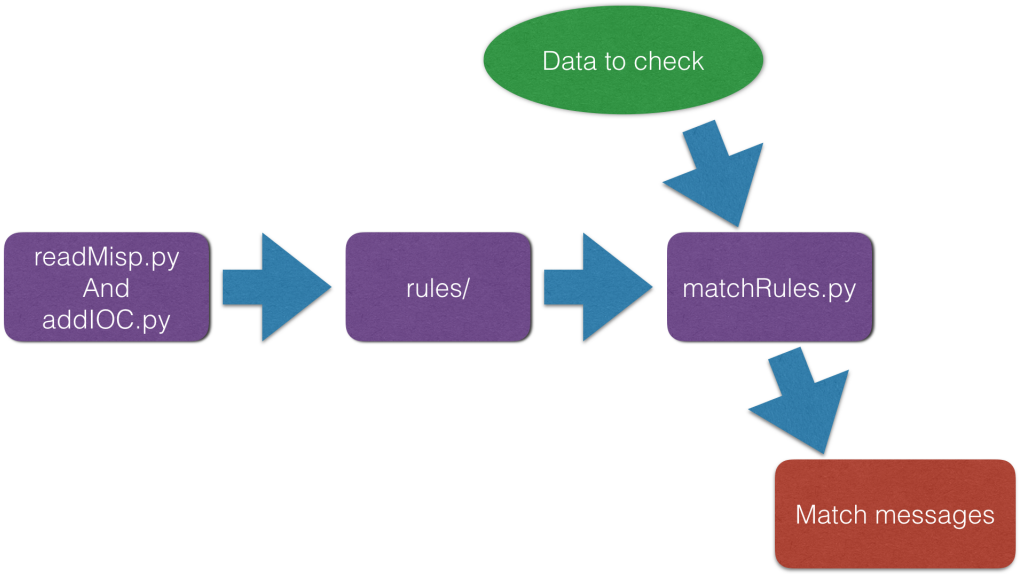


Figure 5.2: General flow

5.2 Parsing logs

The idea was to test the system with log files as I did with the hashstore. First I tried to reuse the parser I have created but I tried to go too deep in the analyse and it has bad performances as well as a lot of particular cases that were not working. I therefore changed my mind and used Logstash and Redis to store the parsed log. It also has the big advantage to be modular and only a new configuration file is needed to parse any other type of log files. Besides that, I'm storing the data inside a Redis queue and polling on it in the script matchRules for trying to match a rule.

5.3 Multiprocessing

When there are a lot of logs, looking at them one by one is taking too much time, but we can test simultaneously more than one log line at the same time by creating processes. Also it is important to point out that I have started this by trying to use threads but, as I'm using the cpython interpreter, it is impossible to have two programs running simultaneously due to the Global Interpreter Lock (GIL). That is why, I finally chose to create a pool of processes instead.

5.4 I/O and Rule Size Optimisation

In the implementation of Tim van de Kamp et al., for each rule, they create a file to store the rule. It works for a proof of concept but it is a really bad idea in UNIX systems. Each file has a minimum size of 4k. Already with 6300 events each one possessing 153 attributes in average, it is easy to see the problem coming. Moreover, the I/O functions were really slowing down the system.

These problems were easy to solve by creating a complete CSV file for all rules. But was still not a complete solution because it would mean that we should load everything in memory even if we don't use everything. That is why I finally chose to have one CSV file per type of IOCs. For example, the system creates files url.csv for URL rules, ip-dst.csv for destination IPs and so on. This allows having a better management of what needs to be load in cache memory by the script.

5.5 Comma-Separated Values (CSV) or Tabulation-Separated Values (TSV)?

This is a really small modification but makes files more readable and avoid existing coma in the data to make the parsing not working. Therefore, I have chosen the tabulation instead of the coma as an element separator in the rule files.

5.6 URL Normalisation

The goal of the system is looking for matches in MISP data. But off course, if an event has for Uniform Resource Locator (URL) www.ioc.com I would like http://www.ioc.com, http://www.ioc.com/, http://www.ioc.com:80/, ... to match as well. I first dug a little into URL normalisation like the standards and techniques already used in practice. I have realised that normalisations are difficult as, in the standards, they want URL to always work after, and really be the same. For example http://www.ioc.com is not https://www.ioc.com while in matching, I absolutely don't care and I would like to both trigger an alert. Besides that, the standards on URL normalisation [Request For Comments (RFC) 3986] contains either normalisations that preserve or usually preserve the semantic of the URL but none that does not preserve it. Standards thus focus on keeping the number of false positive near to zero but it has for impact to miss a lot of

real positive and we cannot afford that. I have thus found this paper [23] which was focus on URL normalisation and that goes beyond standards normalisation steps like case sensitivity in the path, the last slash symbol at the end of the path and the designation of the default page (index.html, ...). Besides that, a great summary of all these standardisations steps is available on Wikipedia [4].

In the implementation, I have used the Python library url_normalize:

- Take care of IDN domains.
- Always provide the URI scheme in lowercase characters.
- Always provide the host, if any, in lowercase characters.
- Only perform percent-encoding where it is essential.
- Always use uppercase A-through-F characters when percent-encoding.
- Prevent dot-segments appearing in non-relative URI paths.
- For schemes that define a default authority, use an empty authority if the default is desired.
- For schemes that define an empty path to be equivalent to a path of "/", use "/".
- For schemes that define a port, use an empty port if the default is desired
- All portions of the URI must be utf-8 encoded NFC from Unicode strings

I have manually added some additional steps like:

- Removing directory index: default.asp, index.html, index.php, index.shtml, index.jsp, default.asp.
- Remove the fragment (#...) which is never seen by the server.
- Removing the protocol http://, https://.
- Removing “www” as the first domain label.
- Removing the "?" when the query is empty.

5.7 IP Normalisation

Like for the URLs some IPs can be expressed differently due to shortcuts. For example, the Internet Protocol version 6 (IPv6) address 2001:0db8:0000:85a3:0000:0000:ac1f:8001 is equivalent to this IPv6 address 2001:db8:0:85a3:0:0:ac1f:8001 as well as 2001:db8:0:85a3::ac1f:8001. Moreover, some MISP IP attributes specify an IPv4 range instead of an IPv4. This also needs to be handled in a proper way.

To solve these problems, I’m using a simple library called ipaddress with two specific functions, ip_network for handling the network ranges and ip_address to handle standard IPs. The last function creates an object which is either an IPv4 or IPv6 instance. For the IPv6 address 2001:0db8:0000:85a3:0000:0000:ac1f:8001, it would return IPv6Address(’2001:db8:0:85a3::ac1f:8001’). Then a simple str function allows getting back the normalised value. Also, this function raises an error when there is a problem with one IP, in this case, I simply add them without any changes and print the problematic value with the generated error explanation. This brings a lot of information on malformed IPs, on IPs that should not be used for IDS as the host bit is set. This is even thanks to these errors that I realised that there were IP ranges specified as IP and

that I needed to take care of them.

For the IP ranges, there are three possibilities to handle them, either we create ranges and it matches only if the user tries the range (meaning that when trying to match an IP we also need to try to match the different possible ranges) or, we can accept only ranges '/x' where x >23 and for these cases, create the 256 (or less) possible rules. This second idea could be really inefficient due to the high number of generated rules. A third one could be to normalise the different type of ranges and to try matching the different standardised ranges as well when matching. This is unfortunately not feasible with the current implementation and could be a nice further work to find a simple way to implement it.

5.8 PBKDF2

I have first started by limiting the key derivation function (KDF) choice to PBKDF2 which is not the author advised one but allow to parametrize the difficulty for an attacker to brute force some small data. It is also important to know that a high number of iteration is needed if we want the key to look random. But this fits with our aims as we also need a lot of iterations to slow down the brute force attacks. Then, I have implemented an other KDF following the same scheme as with PBKDF2 called Bcrypt which is explained in 5.15.2. But HMAC was not added as it does not allows the user to parametrize the time cost of the system and is too fast to slow down an attack on small entities.

5.9 Library: Pycrypto towards Cryptography

While discussing with Tim van de Kamp by email, he said that if he had to start over, he would choose the cryptography library [1] instead of pycrypto because it seems to be a more professional implementation. I thus have made some research on it. There are some discussions about it but `libhunt.com` gives this table 5.1 where we can see that even if pycrypto seems better written, the cryptography library is more up to date and still with a lot of activities. Besides that, I found the library well defined and really easy to use that is why, I have chosen to refactor the code with this particular library instead.

Table 5.1: Cryptography Libraries Comparison

	PyCrypto	**Cryptography**
Popularity	7.2	**7.3**
Activity	0.0	**9.0**
Stars	1 345	**1 431**
Watchers	**103**	86
Last Commit	about 3 years ago	**7 hours before checking on the 13 April 17**
Code Quality (L1 to L5)	**L4**	L2

5.10 Structure improvement

The code was becoming less and less clear to understand and as I wanted it to be modular for the cryptographic system used. I have decided to completely re factor the code.

[1] https://cryptography.io/en/latest/

The used structure is now as following:

- src: Source code.
- conf: Configuration files.
- res: All downloaded resources like the CSV downloaded from MISP.
- rules: All Generated rules.

Inside src, there is also a crypto subdirectory in which are located all cryptographic functions in the different possible modules implemented.

5.11 Top Configuration File

Instead of having a lot of configuration files as when I started, I have gathered them back in one general file where we only have to fill in what will be used. The configuration file is divided into categories and is read thanks to the configparser Python library.

5.12 The Encrypted Message

In the original paper, the authors were using the Course Of Action (COA) for the message. This is really useful to automatically react as soon as possible, but in our case we are mostly interested in protecting data as well as analysing the incident. Therefore, my choice was to use a message to allow the user to retrieve the attributes and events on MISP and fetch the needed information. This has two advantages, the first one is to only allows to know that an element is known by **someone** using MISP and nothing more. This is really important for the privacy. Then, in order to get back information the user needs to connect to MISP where all authorizations are checked by the standard system.

Another advantage of this system is to be able to mix rules from different MISP instances. But for that, we need to be able to identify the different instances thanks to a universal unique identifier. After discussing it with CIRCL, it will normally be soon implemented.

Besides that, not every user needs the same message, so the message content can be modified in the top configuration file but in general at least the Unique Universal Indentifier (UUID) is needed to reference uniquely the element. The argument 'rules->message' is therefore associated with the attributes (separated by a space) contained by MISP attributes like: "uuid event_id date".

5.13 Add a Rule

After having created the MISP rules, if we want to add new ones, we can do it thanks to the addIOC script. Then we can choose to specify the rule directly inside the terminal or via a CSV file located in the 'res' folder. This needs to be used carefully as no verifications are made on the added rules. Checking every new IOC if it already exists would have been really too time consuming. On the other hand, unfortunately, if a bloom filter is used, it is completely regenerated for adding new rules as we cannot add/remove rules dynamically while keeping the same False Positive (FP) rate.

5.14 Update Rules

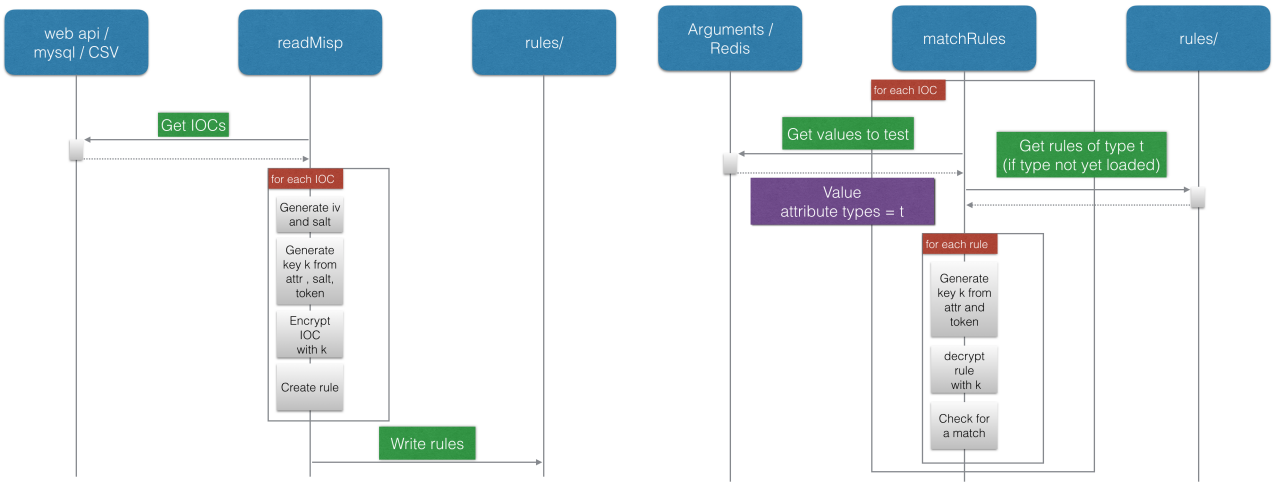
After all that, making rules harder to bruteforce, makes it harder to generate as well. This is why we need the capability of updating the rules with the recent IOCs. This kind of implementation is only useful when the rules are generated via the MISP web API. Therefore, the update implementation works only in that case. First, a new metadata file is generated in the 'res' folder and keeps track of the last IOCs date. Then via the addIOC script with a '–update' argument, it gets the new MISP attributes thanks to a specific web request, saves it in an CSV file called update<number of the update>.csv. The CSV events are then transformed into rules and added to existing ones before finally updating the metadata file with the new latest date.

5.15 Chosen Cryptographic Systems

The idea of the generalisation was to add modules for handling the way data are stored/encrypted, we can use completely different cryptographic systems later called modules. In this section, I will discuss my implementation choices for each implemented schemes.

5.15.1 PBKDF2

This is the basic system for which all the structure has been optimised and has been thought for. In this section, I will thus briefly introduce these two sequence diagrams to help the understanding of the system:



For the cryptographic system, there is also a small difference in the rules compared to the article. Instead of directly encrypting the whole message, I have decided to divide it into two parts (easy to do with the cryptography library). The first is the ciphertext-check and the other one is the ciphertext. The ciphertext-check is simply the first block of the encrypted message. It is used only for matching, decrypting it must results in only zeros if there is a match. (cfr. 4.5)

For this algorithm, the user must choose an important parameter that will determine the "security level". This parameter is the number of iterations for the key derivation function. In 2000, the [RFC2898] advised 1000 iterations for password storing. In 2015, OWASP password storage cheat sheet [37] advised about 10 000 iterations while NIST recommends in the Appendix A.2.2 of its guide [36] is to use as many iterations as it is possible with the needed performances.

Some additional analyses can be made on the ability to sustain an attack, the article made

by the CryptoSense[2] website is really well documented. They have analysed previous work and have done some estimation for 2015. Their approximation is more a guess than anything else and I wouldn't dare to try that for 2017 but I will nevertheless show their results to get a rough idea of what could be the time to crack a password from a key generated with PBKDF2 (AES + SHA-1):

Table 5.2: Cryptosense table

Password Complexity	Estimated Entropy (bits)	1 000 iterations	10 000 iterations
Comprehensive8	33	4h46	47h
8 random lowercase letters	37	12h	5 days
8 random letters	45	123 days	3 years 5 months
8 letters + numbers + punctuation OR 4 random Diceware words	52	325 years	3250 years

Finally, here is a simple example of a rule generated with this system. This IOC is ioc.com of type test 5.3. As we can see, all attributes but "attributes" are encoded in base64.

5.15.2 Bcrypt

This implementation is similar to my implementation with PBKDF2. Bcrypt is a password hash function based on the Blowfish cipher, as PBKDF2, it also incorporates salts to be protected against rainbow table attacks (TMTO). It has also the advantage of being more configurable and is more efficient against GPU attacks. It also seems better to use but the NIST recommendations are still about PBKDF2.

One problem with this algorithm is that it only considers the 72 first characters of the password for the key generation. It can seem a lot but even this URL `https://translate.google.com/#fr/en/72%20caract%C3%A8re%2C%20ce%20n'est%20pas%20beaucoup` is 88 characters long. This means that if we add the token at the end (like in PBKDF2) it will not be included. On the other side, if we first put the token before the data, an attacker would know that the 40 firsts characters will always be the same and moreover, there would be only 32 characters left for the data which is not enough. As the goal was that the whole string has an impact to the key generated as well as the token to have an impact on all bits. The solution found was thus to hash the concatenation of the value with the token and then to feed this fingerprint to the key generation function.

[2] `https://cryptosense.com/parameter-choice-for-pbkdf2/`

Table 5.3: Example of a rule

	rule
salt	jiGxB1CdO43YiPk5NUIbcQY0tYlQltKME1YVLgKKvRI=
attributes	test
nonce	IzizTKP6fVsWnidzEx4Pkg==
ciphertext-check	SxzWSDR3nA4h5HwdTg3WJA==
ciphertext	RQMQ0a7dxJuFxhZDdLuCp72CWaT0wzn4+w==

5.15.3 Bloom Filter

In this section, I will argue the choice for the python-bloom library. I needed to find a fast implementation where I was also able to store the Bloom Filter structure in a file without any additional information. So the first I have found was to use bloomd server with a Python client. But, it would mean that we would have to install all these things which seems not really interesting. But why not using Redis to store data? It is fast and there was a really fast implementation of bloom filters in Python for Redis (actually in C interfaced with Python) called pyrebloom. But the code was really complicated and not easy to read. Moreover, I discovered that they were keeping additional information on keys to be able to remove elements. That, without forgetting the fact that it doesn't seem savable in a file make me drop this implementation. I have finally found the python-bloom library, it seems less efficient but well implemented, without any additional state and there is a way to save the bloom filter.

This was then difficult to implement it in the system as I needed the bloom filter to behave as a rule in a file while being loaded every time it exists. Therefore, I created a 'joker' file that is loaded every time and that could contain the bloom filter. In contrary of standards rules, I'm using only one bloom filter for the whole set of rules.

For this library, I could not install it in the standard way as I did a small modification on the code to make it works with Python3 correctly. This is why the code of the library is directly included in the repository.

5.15.4 Bloom filter used to improve the performance of Key Derivation Functions

Key derivation functions are working to slow down a brute force attack. This is working really well as the time complexity of a brute force attack increases exponentially with the length of the string to "crack". On the other hand, if we know exactly the data we are looking for and if there is a small set of different values. We cannot do anything against it but slowing it down as explained later in the results.

So if we came back on a longer sized data like URLs, brute forcing them (if we are not looking especially for a value) is close to impossible and even if making the task easier for a user makes the task easier as well for the attacker, it can still be nearly unfeasible for an attacker to brute force the whole dataset while making the system a way faster for the user.

With this idea in mind, I have used both types of implementations to create key derivation function improved with a bloom filter. The bloom filter avoids decrypting rules where the Bloom Filter refutes the membership of the data we are looking for. And, on the other side, when there is a match, rules can be used to get back the information needed to retrieve the event information from MISP.

This is a serious advantage for matching. In function of the bloom filter FP rate the system could be made a lot faster. So much faster that we could use it directly as an IDS that could be updated time to time and that could work on computer logs and stand-alone to generate reports on events that could affect a system. Then, the analyst would just have to go on MISP to get more specific information. The previous system was first aimed to be used like that but with a recommended number of iterations, the system was too slow to be used in that way as data to check accumulates at a much bigger rate than the system needs to look through all rules.

Thus increasing the speed can be done by decreasing the rate of false positive of the bloom

filter but what if an attacker is only interested in knowing the element belonging to the data set? The bloom filter could give him all the information he needs. This is the reason why we can not take a too small rate for the bloom filter.

A complete analyse of this solution will follow in the chapter about the results.

5.15.5 An other KDF

An open competition called Password Hashing Competition (PHC) started on 2013 to meet the needs of a standard password hashing algorithm. It ended in 2015 by selecting the winner as Argon2 and also with a special recognition to some other algorithms like Catena, Lyra2, Yescrypt and Makwa.

Argon2 was designed at the University of Luxembourg and there are three versions:

- Argon2d maximises resistance to GPU cracking attacks.
- Argon2i is optimised to resist side-channel attacks.
- Argon2id is a hybrid version out of Argon2d and Argon2i.

There are two available libraries that implement Argon2 which are passlib and argon2_cffi. This could be interesting to use it instead of bcrypt or PBKDF2 but unfortunately, the two existing libraries have directly integrated the salt generation which makes it impossible to use with the current implementation.

Chapter 6

Results

Finally, is it really working? Is the additional computation worth it? Can we trust this system with our data? Could we improve the code for the computation time? This chapter will try to answer those questions. This first part will be on understanding the used dataset, then I will focus on code profiling to understand which parts could be improved. After that I will explain a set of benchmark to try to show the interest of this system.

6.1 My Settings

MISP is really easy to use, but if we want to go deeper in it, there are a lot of things to understand. That's why they give training and make available a MISP training virtual machine in order to train ourselves and test new developments.

At the beginning, I have started to work on a private instance of MISP hosted by CIRCL but, every time that I was working, I needed to have a really good web connection to download all the data. That's why I finally downloaded the virtual machine. The MySQL database was not accessible from outside the virtual machine but, as I was going to work on my computer and I didn't want to be stuck programming inside the virtual machine, I have done some modifications to make available the database to my local network. The first step was to check that my virtual machine runs using the virtual vboxnet0 interface (Standard interface used for a network only between hosts and the hypervisor for Virtualbox) and was using the IPv4 address 192.168.56.50. Normally it works without any problems but it didn't work on my computer, I thus added an IP in the right range by using the command: **ip add add 192.169.56.2/24 dev vboxnet0** . I'm also sometimes using a second virtual machine to work with and on this virtual machine, I have configured two network interfaces, the first one was to have an internet connection while the second one was the vboxnet0 interface. For simplifying the use of the vbox interface, I have configured my network to automatically add an IP in the right range (/etc/network/interfaces). This is the addition to the basic configuration:

```
auto eth1
iface eth1 inet static
address 192.168.56.1
netmask 255.255.255.0
```

After that, I was able to contact the virtual machine via the network but it wasn't enough to have a MySQL access. Therefore, I modified the configuration file (/etc/mysql/my.conf or /etc/mysql/mysql.conf.d/mysql.cnf depending on the VM version) where I have just commented the "bind-address 127.0.0.1" line. The next step was to create a user with the rights from the outside. For that, I have connected myself to the database as the MISP user and I have added the user:

- mysql -uroot -pPassword1234
- CREATE USER 'username'@'%' IDENTIFIED BY 'Password1234';
- GRANT ALL ON *.* TO 'username'@'%';

Once done, I could continue to program more easily and create some tests as I could control the whole available dataset.

6.2 Dataset

The objective was to make all data available if additional analyses are wanted. For that, following the same reflection as with the starting screen-shots, the data directly inside the virtual machine are TLP white and are available for everyone as we can directly download them on the CIRCL website: `https://circl.lu/services/misp-training-materials/`. Thus I have used these data instead of the one directly from an instance. Even if there are a lot less data here, the computations and analyses give the necessary idea.

In order to get a rough idea of the IOCs content of the virtual machine, it contains a set of 161 710 attributes divided into 70 different types including 6 033 URLs and 11 257 destination IPs.

Besides that, the system has also been tested with a real MISP instance hosted by CIRCL. The behaviour of the system was similar, the only difference was the time needed as there are more rules to generate and to test in the system.

6.3 Code Profiling

This section is divided into three parts, a deeper understanding of the code workflow is needed to understand where the time is lost during the execution. Then, the code will be profiled for two different executions in the PBKDF2 mode, the first is aimed to compare the time of the key generation with only one iteration. That gives a right comparison point between the different functions while the second will be with 1000 iterations to look at the behaviour of the system in a possible configuration. Finally, I will focus on the bloomy implementation.

6.3.1 Code Flow

As previously explained, a deeper understanding of the different workflows of the code is important to understand the later profile. This explanation will be separated into the read, match and crypto part as crypto is used by both.

ReadMisp

ReadMisp (fig. 6.1) will first retrieve the data, either via the web API, the MySQL connection or directly from CSV (located in the res folder). The advantage of the MySQL is the speed for two reasons, first, there is no data transformation as we request the data as it is stored and secondly, in general, the MySQL server is accessible directly from the server that generates the rules which mean that there is less network delay. On the flow chart, we can also see the function printv which is a call to print only if the program runs in verbose mode. Once the IOCs are available, the attributes are parsed and the message is created. The message is the important information as it identifies the threat and allows the user to connect himself to MISP in order to get back all information related to the event.
For the parsing is the real creation of the rule, which is done in several steps:

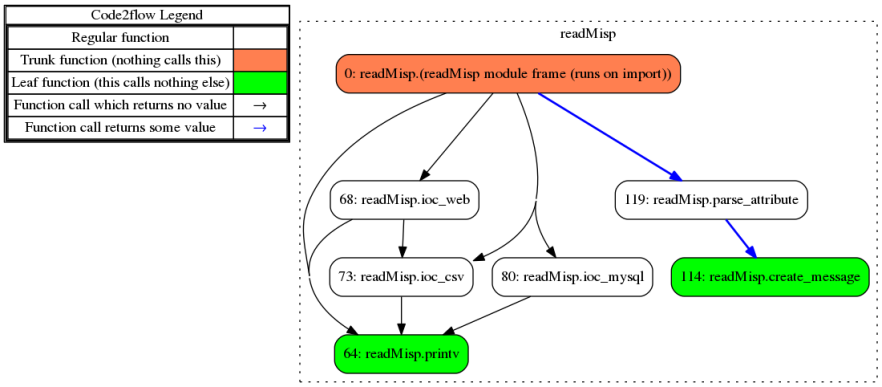


Figure 6.1: ReadMisp WorkFlow

1. Values from MISP are split and associated with their type in a dictionary.
2. URL/IP are normalised.
3. The message is created.
4. The Crypto package is called to create the rules.

In order to be modular, all the code for rule creation is separated. Each time that the program is started, it goes to the configuration file to see which Crypto module to load from the crypto package. Every module needs to implement the interface (interface.py) and then must be added in the choose_crypto.py script. For an example, the workflow for the PBKDF module is available in figure 6.3 and will be explained afterwards.

MatchRules

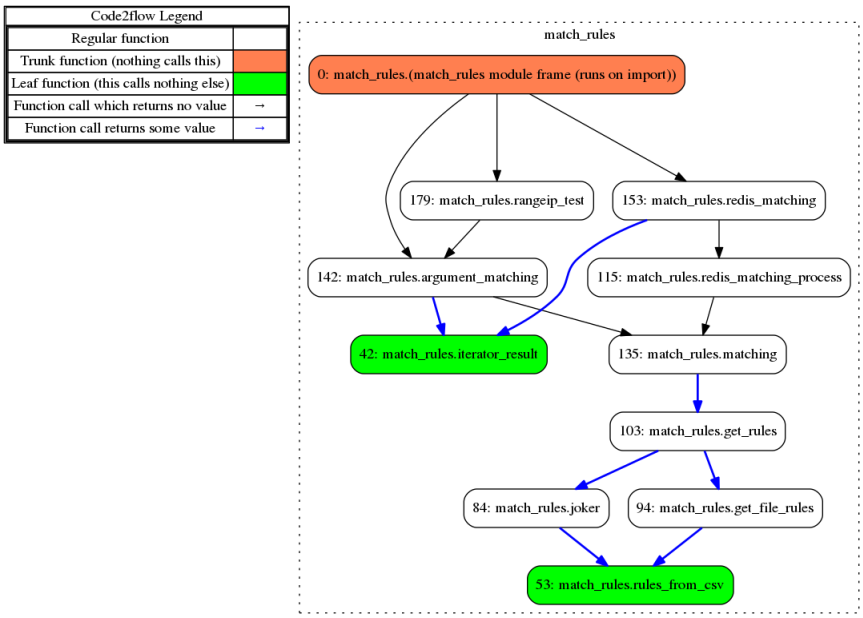


Figure 6.2: MatchRules WorkFlow

MatchRules has two really different ways of working, try to match an element from the arguments (argument matching) or from Redis if the system with the logs is used. The additional rangeip test is just for a testing purpose and will be further explained in the benchmark sections.

For Redis matching, the only difference is that a pool of processes running a Redis matching process is used. Each process is actually polling the Redis queue for new logs and is then feeding them to the matching system. In the other system, the arguments are directly fed to the matching system.

The matching system is working in different steps, first, in function of the argument types, it will load and cache the needed rules (get_rules). There are two different sorts of rules, the standards one are divided into files while the 'joker' file is used for the bloom filter.

Once the rules are loaded, for each attributes to check, the crypto module will try to decrypt each rule up to find or not a match.

Crypto Module PBKDF2

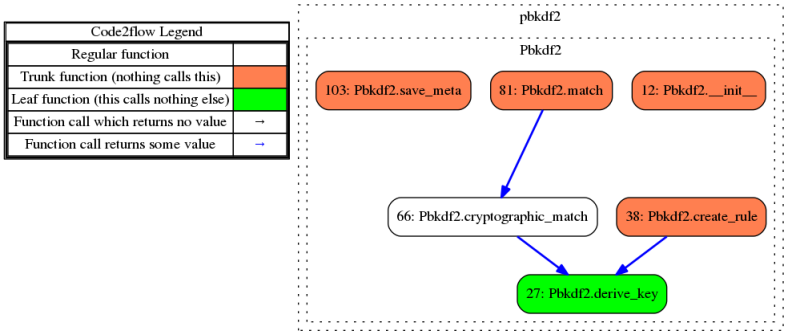


Figure 6.3: PBKDF2 Crypto Module WorkFlow

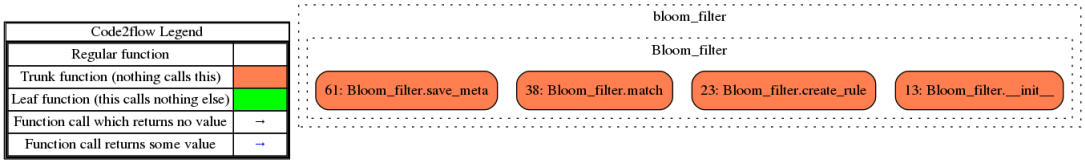


Figure 6.4: Bloom Filter Crypto Module WorkFlow

The crypto modules are responsible for creating and matching rules. In the particular case of the PBKDF2 module, it generates the keys and encrypt the message or try to decipher it. I have additionally added the bloom filter chart as well to see that the difference is small as they inherit the same interface.

As the types of file can differ from modules to modules, this is also the crypto module which is responsible for saving data and metadata files. This part can take a lot of time as for saving the elements (as well as getting them) we cannot afford problems inside the files with special characters. So for that, the idea is to use base64 encoding for the encrypted message, the salt/nonce and also the cipher-check argument (used for faster match checking).

6.3.2 Profile of ReadMisp for 1 iteration

We thus need to verify that generating the keys and encrypting/decrypting data is much more time consuming than everything else even with one iteration. We should also do especially close attention to the normalisation of URLs. Normalisations could be inefficient as I don't know how the library is implemented and moreover, I'm additionally iterating for searching regex after that which can be inefficient.

For profiling the code, I have had some difficulties, first, I tried to use the vprof profiler as it seems to be the best and it was said that it supports Python3. But there were a lot of errors and I first didn't succeed in using it. I have thus tried to use a line_profiler (only working with Python2) then profiling which was really complicated to use. I finally managed to use Vprof with a little modification (to force it using Python3). We can have graphs of the function utilisations, a line by line profile as well as a memory analyse which can be really useful. On the following Vprof graphs, the horizontal axis represents the time. The vertical axis represents function calls meaning that if a function A is represented on top of another function B, it means that this is function B that has called function A.

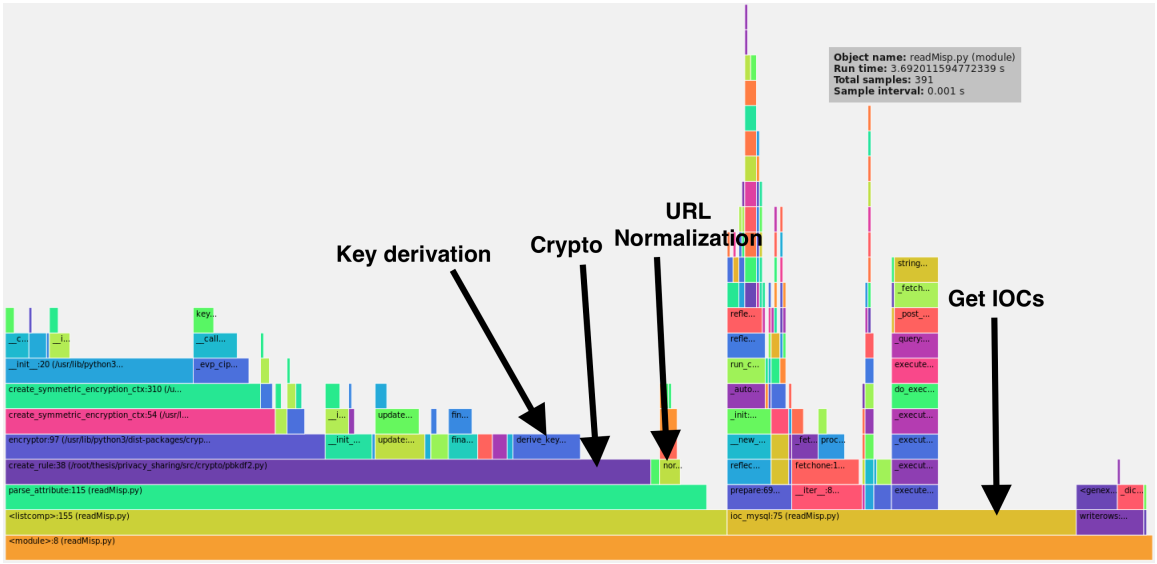


Figure 6.5: Vprof - Flame Chart Profile of readMisp PBKDF2 - 1 iteration (–misp mysql)

This graph 6.5 shows that getting the IOCs from MySQL takes approximately one-third of the time. And there, we also see that two-thirds of this section are used directly for MySQL but the one-third left is used for ordering and small modifications of the format of some values. This could be slightly improved but this would not increase the average performance as we can see with 1000 iterations. Then writing data to files (Rightmost dark blue) takes time but cannot be improved significantly. The first question about the normalisation time is answered and we see that it shows that the normalisation process of URLs is already a small amount of time compared to one iteration for the key derivation. We also see that in this particular case, the major part of the time is taken by the encryption of the messages.

With the memory graph, we can see that the highest level (on the right) is approximately at 200MB used for 27663 rules. This does not vary with the number of iterations.

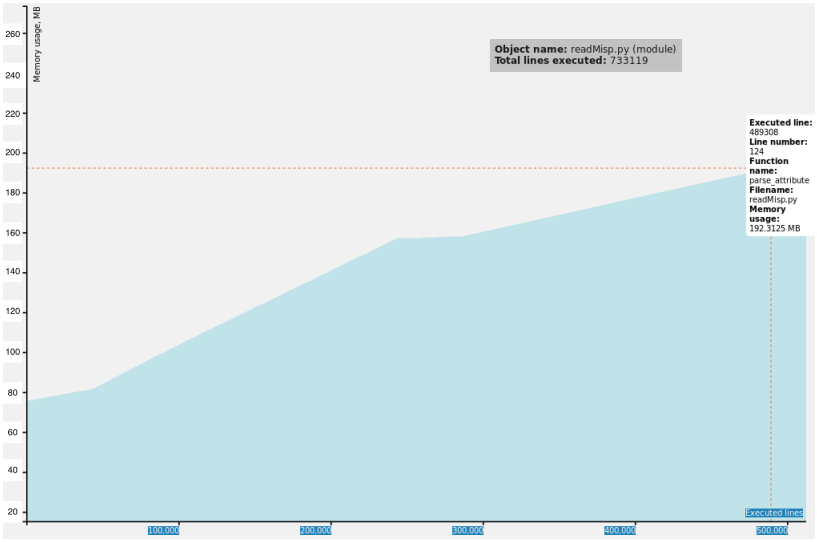


Figure 6.6: Vprof - Memory Chart Profile of readMisp PBKDF2 - 1 iteration (–misp mysql)

6.3.3 Profile of ReadMisp for 1000 iterations

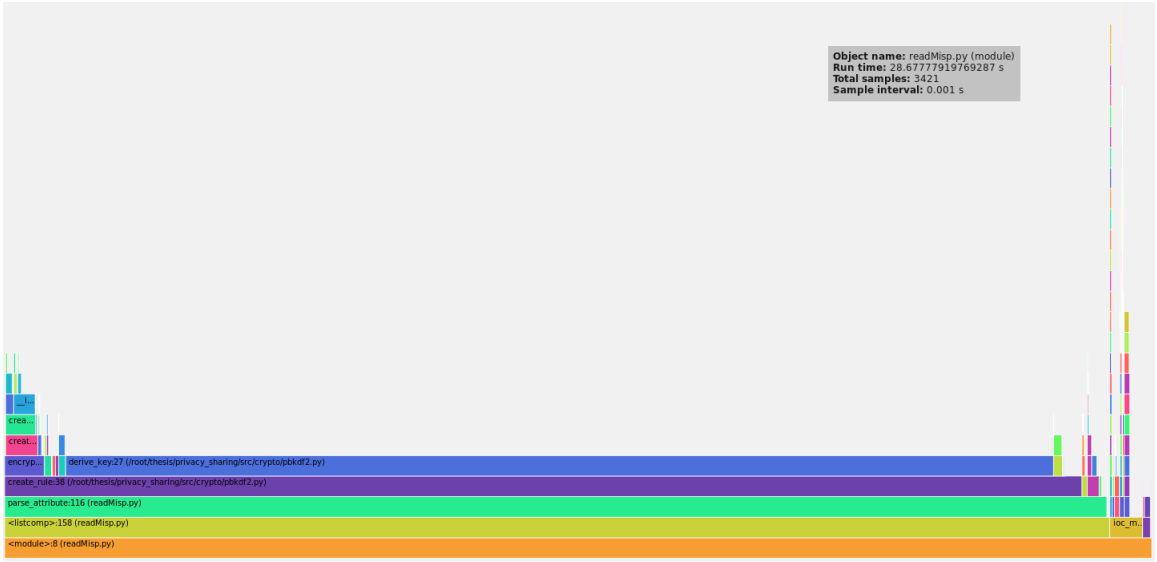


Figure 6.7: Vprof - Flame Chart Profile of readMisp PBKDF2 - 1000 iterations (–misp mysql)

The goal of this work is to slow an attack and consequently to generate random looking keys, which means that a lot of iterations are going to be used. This is the reason why this is closer to a real execution of the system. Of course we can see that the major part of the time is used by the key derivation function (fig. 6.7).

6.3.4 Profile of the bloomy_pbkdf2 module

The first step is to analyse the computational overhead for creating the rules with the bloomy module. Figure 6.8 shows that the key derivation function is still more computationally intensive than the creation of the bloom filter, to be more precise, we can even see that the bloom filter

generation takes only 1.9% of the time.

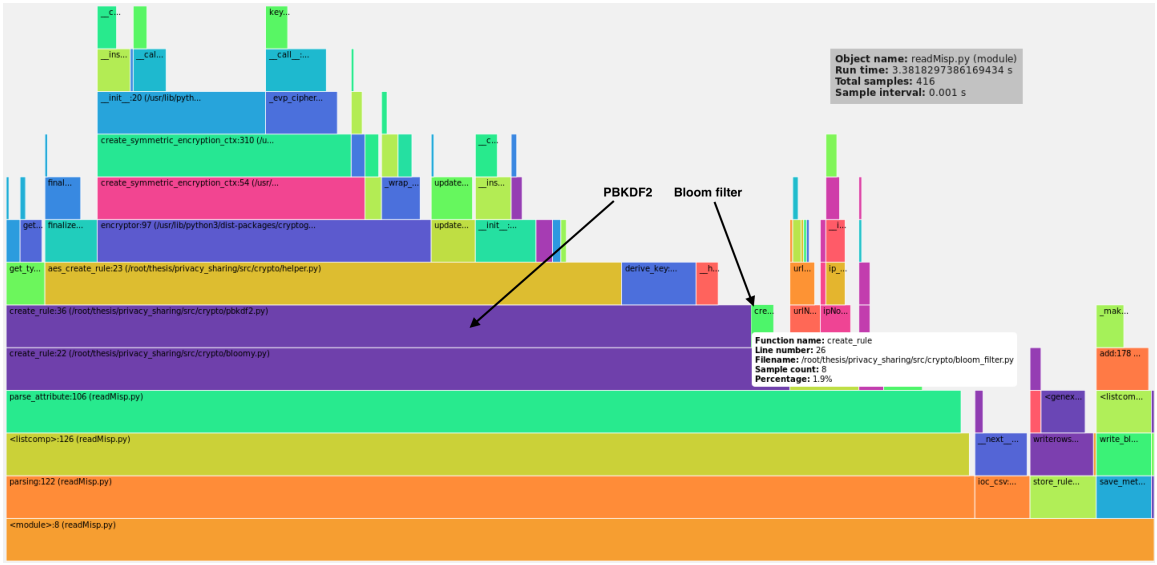


Figure 6.8: Vprof - Flame Chart Profile of readMisp bloomy_pbkdf2 -1 iteration (–misp res)

Then, when the code is running, there are three existing paths for checking a value. Firstly, if there is a rule match, due to the null rate of false negative, we know that the bloom filter will match. Then all rules will be checked and the match(es) will be found. On the other side, if there is no match, there are still two possibilities. Either the bloom filter matches due to its false positive rate and therefore all rules will be checked as well or, the bloom filter will reject the membership and it is the special case that improves the overall performance of the matching system.

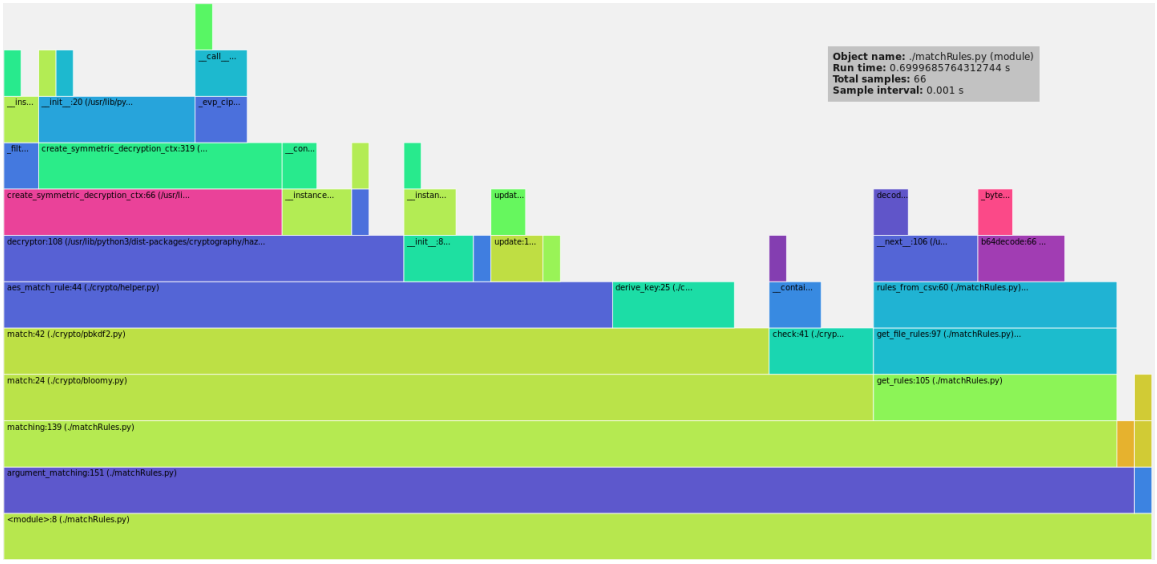


Figure 6.9: Vprof - Flame Chart Profile of matchRules bloomy_pbkdf2 (1 iteration) with a match

On figure 6.9 we can see that it takes more time than the simple PBKDF2 module as we have the overhead of the bloom filter. This is exactly the same behaviour when there is no match

but a false positive bloom filter match.

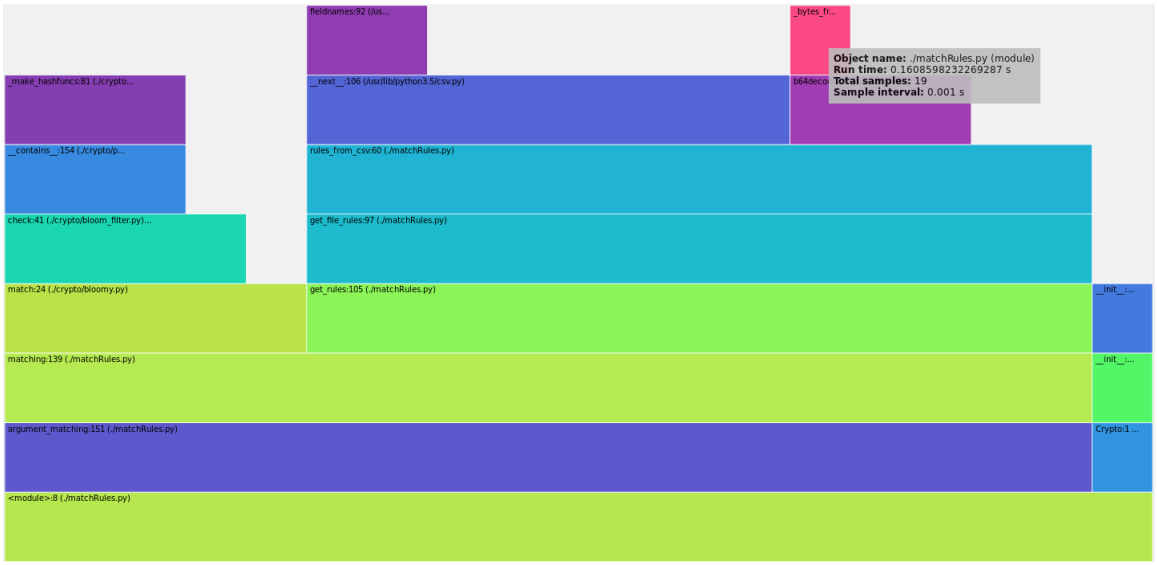


Figure 6.10: Vprof - Flame Chart Profile of matchRules bloomy_pbkdf2 (1 iteration) with no match in the bloom filter

While we can clearly see the advantage of the system in figure 6.10 as the bloom filter immediately realise this value does not belong to the set.

In this case, we can realise that there is a problem with the implementation, as I tried to have a modular implementation and doing everything inside the modules, it does not always fit well with the system. Here, for one attribute, whatever the result of the bloom filter for one rule it will test the bloom filter for every different rule. This is really bad as the bloom filter is not related to the rule but only to the attribute tested. To solve this problem a cache must be implemented. The first cache implementation already improved the system but was really bad as I was using a dictionary of matched values. There are two problems with that, first, to put the orderedDictionary used for representing an attribute as a key, we need to transform that into a string. And then add it to the cache. But for a matching, we each time need to transform the value to a string and while checking the dictionary it actually hashes the string attribute. All that was taking a way too much time and I finally opted for a simple but nevertheless efficient implementation with only two cache variables, the last OrderedDictionary representing the attribute to check with the last bloom filter result. This cache is so fast that the bloomy overhead does not even appear on the profiles anymore, the script complete run when bloom filter discover the attribute is not in the set is about 0.16s.

6.4 Benchmarking

The ideas have been explained, in the first few sections, then the implementation has been explained followed by profiles that help to understand the behaviour of the system and how the computation power is used inside the system. Now it is time to see how it is working with real data, what would be the right parameters to use and why these choices. Moreover, this is in this section that we could analyse if the goal had been totally reached, reached for some part and decide if it can or cannot be used in real implementations. This is important as we are aware of the benefits that it could have but we must stay aware of the risk of that exposure on which a

system like this one could lead.

So in this case, what is important is to benchmark the time used to generate a rule in function of the number of iterations and of the algorithm used. The space memory needed for the rule and the time used for testing an element against a rule. And this, with or without bloom filters. Then it is also important to assess the advantage of the bloomy implementation in function of the number of available rules, the number of iterations and especially the false positive rate used.

6.4.1 Key Derivation Functions

In rule generation and matching algorithms, the KDF is the most costly function. For this reason, I have decided to analyse their behaviour in function of the length of the initial value as well as the time complexity used (iterations or rounds).

It was first very surprising but the length of the key has no impact on the time used to generate the values and that for the two algorithms. Thus, in this section, I will only present the graphs about the impact of the number of iterations and rounds.

Also, for both implementations, there are some variations due to system interrupts. I tried to stabilise them by doing a mean of 100 tests for each point. Variations are still there but doing it more time would have taken too much time whereas the needed information is already clearly visible.

PBKDF2

As we can see on figure 6.11 the time increases linearly in function of the number of iterations. As it is linear, it means that each iteration takes the same amount of time to be computed and

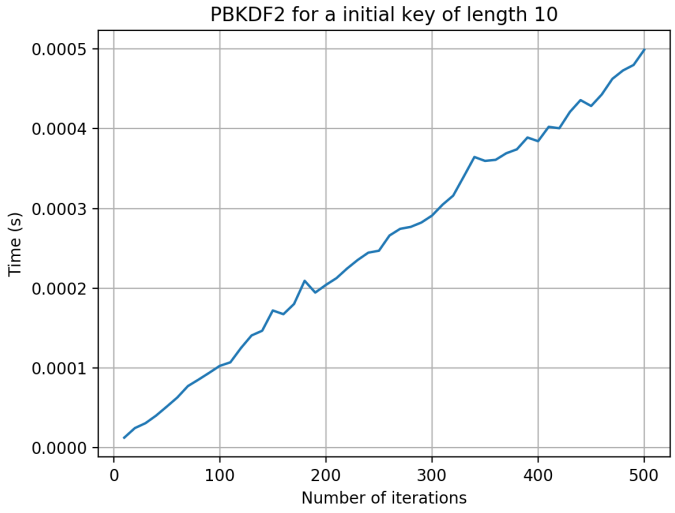


Figure 6.11: Benchmarking: Impact of the number of iterations on the key derivation algorithm PBKDF2

therefore, we can approximate that the time needed for generating a password with one iteration is about 10^{-6}s.

Bcrypt

As we can see on figure 6.12 the time also increases linearly in function of the number of rounds. The same kind of approximation can be done and we get a 0.005s by iteration.

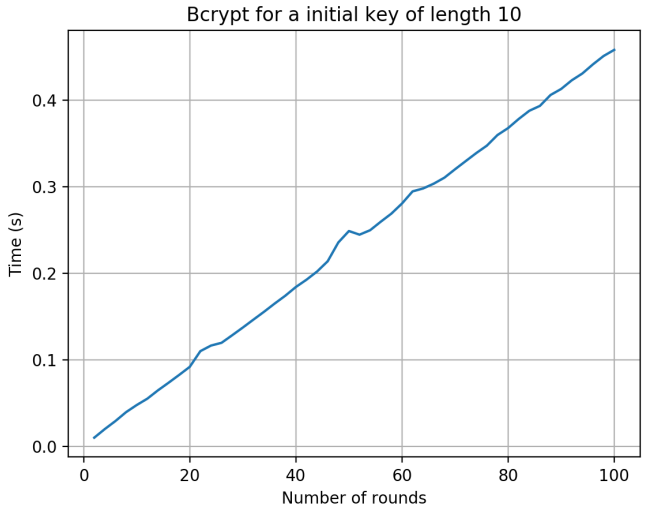


Figure 6.12: Benchmarking: Impact of the number of rounds on the key derivation algorithm Bcrypt

Conclusion on KDF

Both algorithms used behaves similarly, the key derivation cost does not vary with the length as well as it varies linearly with the cost parameter. The only observed difference in these graphs is that for the same cost parameter, Bcrypt takes much more time which is better to slow down a bruteforce attack. As the behaviour is similar, the conclusion would be similar as well for both KDF. Therefore the following benchmarks will be done only with PBKDF2 as it is the recommended KDF by NIST.

6.4.2 Bloomy efficiency

Bloom filters are really efficient and used in this context, they allow to make the matching system a way faster. To characterise this improvement, I need to find data similar in the type of content and on the size as well as easily forgeable. It could have been strings of a fixed size n with a smaller alphabet like {a,b,c}. But I have chosen IPv4 to use this benchmark as well to analyse the feasibility of bruteforcing them. After that, I needed to decide a number of IPs to test at each step. It needs to be not too small to have precise results but not too much as well to avoid waiting too long for the results. Therefore I have decided to take ten "/24" network ranges which mean 2560 IPs. I have chosen to use 192.168.0.0/24 up to 192.168.9.0/24. Thus for each configuration, the matchRules script ran through all these IPs like a bruteforcing algorithm would have done it.

The final complete benchmark was run on an old computer with 4GB of RAM and an Intel i7 processor for a complete week. I have tested with it the PBKDF2 module and its bloomy mode for different FP Rate with an increasing number of Destination IP (ip-dst) rules and an increasing number of iterations.

As we need to try to decrypt the whole set of rules in the PBKDF2 module, it is normal that the time increases linearly with the number of IPs in the system. But it is interesting to understand and see the behaviour of the time in function of the number of IPs in the system for the bloomy implementation (fig. 6.13):

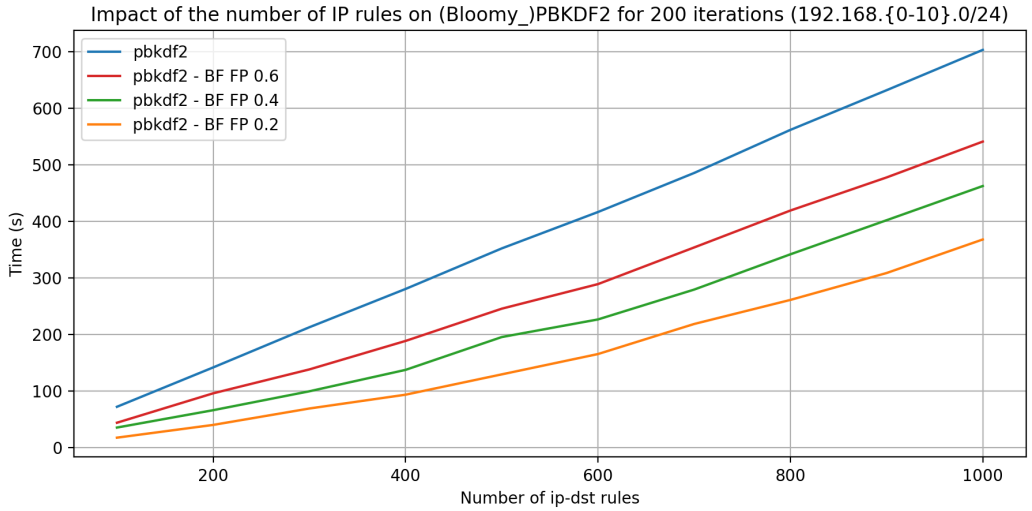


Figure 6.13: Benchmarking of the evolution of the number of IPs in the system

The figure shows clearly a linear behaviour for PBKDF2 without bloom filter (which one can be considered as a False Positive rate of 1). Then, the behaviour is linear as well with the different rates even if they are less straight. This behaviour is due to the false positive rate which is just a probability and not the exact proportion. We can notice that less is the False Positive rate less the slope is. We can also see that the version without bloom filter is approximately twice longer than the version including a bloom filter with a 0.2 false positive rate.

This resulted benchmark seems corresponding to what should be expected as, knowing that for the simple PBKDF2 it took 702.89s for 1000 ip-dst rules (nIP), 200 iterations (nIt) and that we test 2560 IPs (nIPTested):

- Time needed for testing a an IP on a rule is $\frac{Time}{nIPtested \cdot nIP \cdot nIt} = \frac{702.89}{2560 \cdot 1000 \cdot 200} = Time1IP1It = 1.37283203125^{-6}s$
- The number of IP to test when a bloom filter is used can be approximate with $nIP + (nIPTested - nIP) \cdot FPrate$

This is really interesting to see that Time1IP1It is really close to the time needed for one iteration with PBKDF2.

With that, we can estimate other points, for example, for 200 IPs, 200 iterations and a FP rate of 0.2, we should have $Time1IP1It \cdot [200 + (2650 - 200) \cdot 0.2] \cdot 200$ (nb of rules) $\cdot$ 200 (nb of iterations) which is equal to 36.9s while the real result is about 40s. We can notice a slight difference but the approximation tends to follow the right behaviour.

After that, we need to check that the system behaves similarly in function of the number of iterations as we took the hypothesis in the formula derived.

As we can see (fig. 6.14), the behaviour is effectively similar and the formula could be

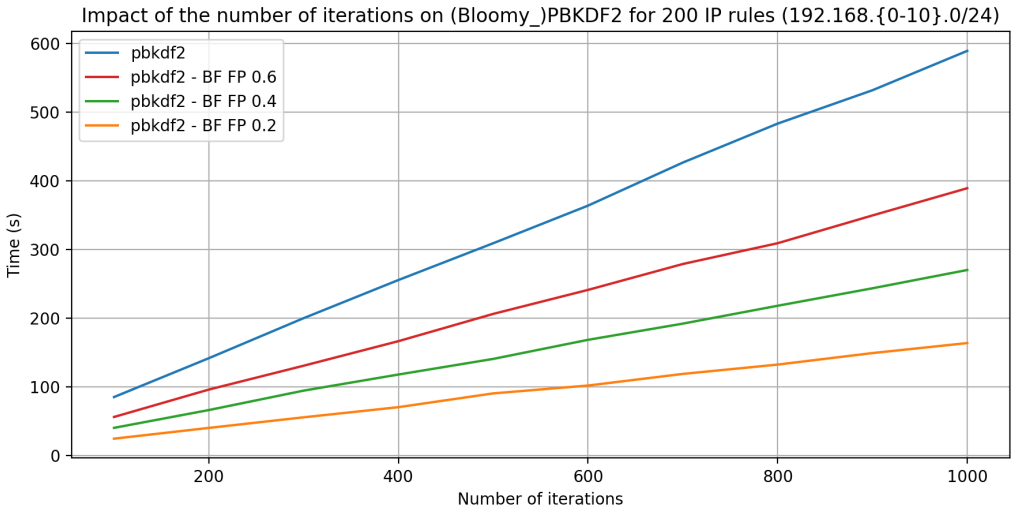


Figure 6.14: Benchmarking of the evolution of the number of iterations used in the system

generalised like:

$$time(nIP, FPrate, nIt, nIPTested) = [nIP + (nIPTested - nIP) \cdot FPrate] \cdot nIP \cdot nIt \cdot (Time1IP1It)$$

Finally, the attentive reader would notice that there is a difference between the slope in figure 6.13 and 6.14. This difference is only due to the additional time needed to decrypt the entire rule when there is a match.

6.5 Rules

The section is intended to give some more technical information about the generated rules.

6.5.1 Time for creating a rule

The major part for generating the rule is the time used by the key derivation function (cfr fig.[6.11, 6.12]). After that, there is a little overhead due to the normalisation and the encryption of the message but they are quickly negligible when the KDF cost is increased.

6.5.2 Space memory consumed by a rule

Bloom Filters are used to decrease the space memory consumption while memorising the membership of big dataset. Here instead of focusing on that, the goal was to keep every element and transform them into another format that needs information and cryptographic computations to read them.

This lead me to add salts, nonces (IV) and a message on top of that. This has a big impact on memory as the elements are not remembered anymore, but the value is only used to encrypt the message.

With that, the size of a rule is nearly a constant as it only depends on the size of the message. For example, the size of a rule for an ip-dst is:

- Salt (base64): 44 bytes
- Attribute type ("ip-dst"): 6 bytes
- Nonce (base64): 24 bytes
- Ciphertext Check (base64): 24 bytes
- Ciphertext (base64 with message uuid event_id date): varies with the id size but around 24 bytes

This means that we are around 122 bytes by elements plus the size consumed by the Bloom Filter which depends on the FPrate used as well as the number of elements. This is huge compared to the size of an IPv4 address but is already less than some URLs.

6.5.3 Lookup

The major part of the lookup time is correlated to the number of rules related to the attribute type searched as well as the algorithm and cost used. And finally, the last cost is for decrypting the ciphertext check element (when it does not match).

Then a simple approximation could be made with the section 6.4.1:

PBKDF2

$$Nb\ rules \cdot Nb\ Iterations \cdot 10^{-6}s + Nb\ rules \cdot Time\ AES$$

Bcrypt

$$Nb\ rules \cdot Nb\ Iterations \cdot 5 \cdot 10^{-3}s + Nb\ rules \cdot Time\ AES$$

6.5.4 Bruteforce

Bruteforcing is the original problem that this work is trying to face. For IPv4 the bruteforce is feasible as the number of different IPs is limited. Therefore we can only use this system to make it harder for an attacker to bruteforce the whole dataset.

The following approximation should be taken with care as it follows from a lot of different really strong hypotheses:

- The attacker tests all non-reserved addresses incrementally.
- The number of IPv4 in MISP is about 50 000 (realistic number).
- The number of non-reserved IPv4 addresses is about 592 708 864 (≈13.8% of the 4 billion possible IPv4s) which gives 3 702 258 432 IPv4s to test.
- Every non-reserved IPv4 has the same probability to be in a rule.
- The attacker stops searching when he found the whole set of values related to the rules.
- We want the database to sustain an attack for at least 10 years.

The last hypothesis is just to get an idea of the time the database needs to resists to a bruteforce attack and can be modified as all other hypotheses in function of what is needed. Moreover an IP could have a time limitation due to the fact that most of the attacker are using dynamic IPs. Thus after a while, even if the attacker does not disappear, the information of its IPs could not

be right anymore.

Now, the goal is to get an estimation of the cost to use for the algorithm if we want the user having a probability of 80% to have fully discovered the complete set of IPv4 in the database after 10 years by applying a bruteforce attack. First, I need to find a number of IP that the attacker needs to test to have a probability of 0.8 of having discovered the whole set.

We therefore need to find x so that $[P(maliciousIP > x)]^{50000} = 0.8$ and thus :

$$x = e^{\frac{\ln(0.8)}{50000} + \ln(3702258432)} = 3702241909$$

This result means that the attacker needs to go nearly over the whole set of possibilities even with that 0.8 probability. Then, I want this number of IP x, to take 10 years to be checked. We thus need to find the cost for PBKDF2 so that cost $\cdot$ x $\cdot$ Time1IP1It $\cdot$ number of rules $\geq$ 10 $\cdot$12 $\cdot$ 31 $\cdot$ 24 $\cdot$ 60 $\cdot$ 60. Where Time1IP1It was found in section 6.4.2. This means that only approximately 1.26 iterations are needed for PBKDF2 with these hypotheses.
Considering 2 iterations, it means that on the other side, a user looking for a specific element would take only 0.13s. But with PBKDF2, the number of iterations needs to be greater to generate keys that look random. This is the reason why, when there are enough rules, we can reach a level of security bigger than what is needed and therefore the bloomy system can be used to parametrize the system to be faster. Therefore, knowing the content, the user needs to tune these two parameters in order to reach his needs.

In order to get a rough idea, if we would have used a simple Redis implementation. The official benchmarks[1] claims being able to do 508388 get requests by second. This would mean that a similar bruteforce attack would have taken only 2 hours.

Finally, we know that we don't have to worried about other kinds of data as the complexity of bruteforcing is exponential in function of the length. Therefore it is nearly unfeasible to bruteforce them. For example, if we take a seven-character alphanumeric password, for each character, there are 26 (alphabet)+26 (ALPHABET) + 10 (0-9) possibilities that gives approximately $(62^7) \cdot Time1IP1It/(60 \cdot 60 \cdot 24 \cdot 31 \cdot 12)$ years which means that it is already 547.8 years for Bcrypt. This gives an idea why bruteforce is very difficult against IOCs like URLs.

[1] `https://redis.io/topics/benchmarks`

Chapter 7

Conclusion

This small chapter is aimed to go through what have been done and what still need to be done.

7.1 System Conclusion

The goal of this work is to share information while respecting privacy and confidentiality meaning that we need to hide the elements of the set but mostly who has reported each element. This is what have been achieved and now, if someone wants to test if a specific value is known by the system. First, he needs to get the generated rules. Secondly, to be sure he is the only one to use these rules, his secret token is included in the rules creation what makes them unusable for someone that does not know it while also allowing to trace back the origin of a data leak. Once he has these rules and the right token. he can check for a value. So, in the worst case scenario, if an attacker manages to get those, he can succeed in knowing if some specific values are known (via the rules or via the boom filter) and to get information allowing him to fetch the information on the MISP web interface. Which means that, at that moment, there is again a barrier to know the identity of the reporter of the event and any additional information as he needs the right MISP credentials to get more information.

Therefore we know that we could possibly (but already difficulty) imagine that this system can leak two important information to an attacker. The number of elements known by the system as well as the fact that someone belonging to an organisation linked to the instance or to some data feed used by the instance has knowledge of a value that the attacker has chosen.

On the other side, if someone wants to get directly the information from one rule. It is nearly impossible. If the attacker wants to bruteforce the ciphertext, he first needs to be able to break an AES encryption scheme with a 128-bit key. And then, he could get the message but still not the IOC related to the information. To get the IOC, he would still need to be able to reverse the hash function used.

Besides that, the bloom filter addition is a new feature that allows the user to have more power in parametrizing the behaviour of the system as explained in chapter 6.

The last problem was about bruteforcing the dataset and it has been shown in the benchmarking chapter that with this system, the user can parametrise the security he needs in function of the time that the dataset needs to withstand.

7.2 Conclusion

This master thesis was working on creating a shareable dataset of threat information while keeping a special focus on the need of the combination of trust, privacy and confidentiality. Unfortunately, I had a lot of difficulties starting this project as I had neither the knowledge nor a concrete goal. But now, I can say that it is really amazing to look at all existing ideas and implementations but also that I badly oriented my search at the beginning and even if I found articles, they were not linked to the right subject but just to subject related to it. By luck, putting all them together bring me to succeed in doing what I wanted to do but now that I know better the subject, that I understand the challenges, if I had to start over, I would completely reorient the research thanks to the findings of these the last few months. For searching information on information sharing, the starting point must be to look at existing tools as it shows exactly the challenges as well as how implementations and researches are directed. This is why this awesome list [2] could be a really nice starting point. While on the other side, for the privacy and confidentiality concerns, all needed information and researches are directly linked to PPRL and the previously cited state of the art [39] gives all needed information.

This master thesis is thus related to those two broad domains by taking a different approach lead by the paper [38]. I brought with this work an improvement of this proof of concept technique as well as a real implementation on a flourishing open source project called Malware Information Sharing Platform and Threat Sharing (MISP). This implementation succeeded in creating datasets of Indicator Of Compromise (IOC) that can be generated on demand for a specific user. It respects most privacy and confidentiality concerns by obfuscating the data while still allowing it to be decrypted only if the targeted user has the need of the information or information that could enrich analyses. Matching a specific IOC then gives the user the identifier of the information in MISP and, he could then ask permission to access the instance (if the information belongs to another instance) as well as perhaps the permission to get to know the organisation that had seen the indicator. One other major advantage of this implementation is also this relaxation in the needed trust for sharing due to the idea of Kim van de Kamp et al. to add an identifier of the user inside the key generation. Thanks to that, the risk of leakage is decreased as the source of the leakage could be identified.

7.3 Further Work

This section is aimed to give ideas to continue this work. To be honest, I believe this domain so big and interesting that there will always exist possible improvements and new techniques to test. Although, new papers are flowing as well as current researches. On this implementation, the major improvement would be to consider the sightings reporting. This would not be too complicated as information needed to report that an element has been seen already exists in the MISP API but it brings some additional challenges as, how could we ensure not reporting twice the same sighting while still being anonymous in reporting? An idea could be to report the sightings by sending the rule seen. By doing that, the MISP instance could check the identity thanks to the token in the request, then he could check that the rule corresponds to the IOC and finally only saving the timestamp and the ciphertext as the identifier. With that, while adding a new sighting, MISP can avoid adding twice the same, checking that the user is authorised to report a sighting while still preserving its identity from being revealed to other organisations.

After that, it could be a good idea to implement the IP network ranges normalisations to allow matches like explained in section 5.7.

With the actual implementation additional modules can be easily created to test other techniques of rules generation. The technique that could be the most interesting to implement and to analyse would be a way to avoid parallelization in the matching process. Now, in the current implementation, if we would like to divide the number of rules into two sets and then to use two "matching process" function on these separate processes, we would get the same result in a nearly twice faster way. To avoid that, an idea could be to make the rules generation dependent on the previously generated rules. This could be easily done like by simply using the decryption of the previous rule in the generation of a new rule. This could make bruteforce even less efficient but it would make really difficult to create the rules as well as for each new rule, we would need to go through all previously generated ones. But in this case, the bloomy implementation could be even more efficient.

Besides that, this implementation leaks the number of IOCs contained in the system. It is even worst than with bloom filter as in that case, statistic analyses could be done to approximate the number of elements but here we only need to count the number of rules. This leaked information could be minimised with the addition of 'false' rules. These false rules could be generated for example by choosing a random element and creating the rule each time with new salts but instead of encrypting 16 zero bytes before the message, we could use 16 one bytes. Doing that for padding files with rules to always have a multiple of a certain number of rules. Unfortunately, this does not totally hide this information as it would still leak a range on the number of possible elements.

A last implementation could be to create a plug-in using this system directly inside an existing IDS. As well as using other data-structures like cuckoo filters (section 4.1.5) or Roaring bitmaps [10].

Glossary

AES Advanced Encryption Standard. 28, 36, 53

API Application Programming Interface. 5, 29, 35, 40, 54

COA Course Of Action. 27, 34

CSV Comma-Separated Values. 4, 29, 31, 34, 35, 40

CybOX Cyber Observable eXpression. 16, 17

DFIR Digital Forensic and Incident Response. 6

DHS Department of Homeland Security. 15

EISAS European Information Sharing and Alerting System. 18

ENISA European Union Agency for Network and Information Security. 5, 18

FP False Positive. 34, 37, 48, 49, 51

GPL General Public License. 11

HMAC keyed-Hash Message Authentication Code. 19, 28

IDS Intrusion Detection System. 12, 14, 19, 29, 32, 37, 55

IETF Internet Engineering Task Force. 11

IOC Indicator Of Compromise. 5, 6, 11, 12, 14, 16, 18, 20, 21, 25–27, 29, 31, 34–36, 40, 43, 53–55

IP Internet Protocol. 4, 9, 12, 19, 20, 25, 26, 31–33, 39–41, 48, 49, 51, 52, 54

ip-dst Destination IP. 48–51

IPS Intrusion Prevention System. 14

IPv4 Internet Protocol version 4. 14, 26, 32, 39, 48, 51, 52

IPv6 Internet Protocol version 6. 32

ISAO Information Sharing and Analysis Organisation. 15

KDF Key Derivation Function. 4, 27, 33, 38, 47, 48, 50

MISP Malware Information Sharing Platform. 5–7, 10–12, 14, 17, 21, 25–27, 29, 31, 32, 34, 35, 37, 39–41, 53, 54

NATO North Atlantic Treaty Organization. 11

NIST National Institute of Standards and Technology. 5, 12, 18, 28, 35, 36, 48

OSINT Open Source Intelligence. 12

PBKDF2 Password-Based Key Derivation Function 2. 4, 28, 29, 33, 35, 36, 38, 40, 42, 45, 47–49, 51, 52

PPRL Privacy-Preserving Record Linkage. 3, 19, 21, 54

RFC Request For Comments. 31, 35

STIX Structured Threat Information eXpression. 16, 17

TAXII Trusted Automated eXchange of Indicator of Information. 10, 16, 17, 26

TSV Tabulation-Separated Values. 4, 31

UCL Universite Catholique de Louvain. 23

URL Uniform Resource Locator. 4, 9, 31, 32, 36, 37, 40, 41, 43, 51, 52

UUID Unique Universal Indentifier. 34

Bibliography

[1] About taxii. https://taxiiproject.github.io/about/.

[2] Awesome threat intelligence. https://github.com/hslatman/awesome-threat-intelligence.

[3] Taxii project. taxiiproject. http://taxiiproject.github.io/about/.

[4] Url normalization. wikipedia. https://en.wikipedia.org/wiki/URL_normalization.

[5] A. Dulaunoy, A. I. Misp core format. https://tools.ietf.org/html/draft-dulaunoy-misp-core-format-01.

[6] Agrawal, R., Evfimievski, A., and Srikant, R. Information sharing across private databases. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data* (2003), ACM, pp. 86–97.

[7] Barnum, S. Standardizing cyber threat intelligence information with the structured threat information expression (stix^{TM}). *MITRE Corporation 11* (2012).

[8] Barnum, S., Martin, R., Worrell, B., and Kirillov, I. The cybox language specification. *draft, The MITRE Corporation* (2012).

[9] Bloom, B. H. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM 13*, 7 (1970), 422–426.

[10] Chambi, S., Lemire, D., Kaser, O., and Godin, R. Better bitmap performance with roaring bitmaps. *Software: practice and experience* (2015).

[11] Connolly, J., Davidson, M., and Schmidt, C. The trusted automated exchange of indicator information (taxii). *The MITRE Corporation* (2014).

[12] Cover, T. M., and Thomas, J. A. Elements of information theory, 1991.

[13] Dwork, C. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation* (2008), Springer, pp. 1–19.

[14] ENISA. Good practice guide network security information exchange.

[15] ENISA. Eisas – european information sharing and alert system for citizens and smes.

[16] Fan, B., Andersen, D. G., Kaminsky, M., and Mitzenmacher, M. D. Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies* (2014), ACM, pp. 75–88.

[17] FIRST. Traffic light protocol (tlp) first standards definitions and usage guidance — version 1.0. https://www.first.org/tlp, June 2016.

[18] Fransen, F., Smulders, A., and Kerkdijk, R. Cyber security information exchange to gain insight into the effects of cyber threats and incidents. *e & i Elektrotechnik und Informationstechnik 132*, 2 (2015), 106–112.

[19] Freudiger, J., De Cristofaro, E., and Brito, A. E. Controlled data sharing for collaborative predictive blacklisting. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (2015), Springer, pp. 327–349.

[20] Johnson, C., Badger, L., and Waltermire, D. *Guide to cyber threat information sharing (draft)*. 2014.

[21] Johnson, C., Badger, L., Waltermire, D., Snyder, J., and Skorupka, C. Guide to cyber threat information sharing. *NIST Special Publication 800* (2016), 150.

[22] Krawczyk, H. Cryptographic extraction and key derivation: The hkdf scheme. Cryptology ePrint Archive, Report 2010/264, 2010. `http://eprint.iacr.org/2010/264`.

[23] Lee, S. H., Kim, S. J., and Hong, S. H. On url normalization. In *International Conference on Computational Science and Its Applications* (2005), Springer, pp. 1076–1085.

[24] Li, Y., Tygar, J., and Hellerstein, J. Private matching. *Computer Security in the 21st Century* (2005), 25–50.

[25] Lincoln, P., Porras, P. A., and Shmatikov, V. Privacy-preserving sharing and correlation of security alerts. In *USENIX Security Symposium* (2004), pp. 239–254.

[26] Machanavajjhala, A., Kifer, D., Gehrke, J., and Venkitasubramaniam, M. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD) 1*, 1 (2007), 3.

[27] Malik, J. Threat intelligence sharing: The only way to combat our growing skills gap. *Information Security Magazine* (jun 2016).

[28] Mohaisen, A., Al-Ibrahim, O., Kamhoua, C., Kwiat, K., and Njilla, L. Rethinking information sharing for actionable threat intelligence. *arXiv preprint arXiv:1702.00548* (2017).

[29] Pagh, R., and Rodler, F. F. Cuckoo hashing. In *European Symposium on Algorithms* (2001), Springer, pp. 121–133.

[30] Ponemon, L. Cost of data breach study: France. *Poneomon Institute sponsored by IBM* (2016).

[31] Ponemon, L. Cost of data breach study: Global analysis. *Poneomon Institute sponsored by IBM* (2016).

[32] Putze, F., Sanders, P., and Singler, J. Cache-, hash-and space-efficient bloom filters. In *International Workshop on Experimental and Efficient Algorithms* (2007), Springer, pp. 108–121.

[33] Sauerwein, C., Sillaber, C., Mussmann, A., and Breu, R. Threat intelligence sharing platforms: An exploratory study of software vendors and research perspectives. WIRTSCHAFTSINFORMATIK 2017.

[34] Swamidass, S. J., and Baldi, P. Mathematical correction for fingerprint similarity measures to improve chemical retrieval. *Journal of chemical information and modeling 47*, 3 (2007), 952–964.

[35] Sweeney, L. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 10*, 05 (2002), 557–570.

[36] Turan, M. S., Barker, E., Burr, W., and Chen, L. Nist: Special publication 800-132, recommendation for password-based key derivation. *Computer Security Division Information Technology Laboratory. http://csrc. nist. gov/publications/nistpubs/800-132/nist-sp800-132. pdf* (2010).

[37] Van Acker, S., Hausknecht, D., Joosen, W., and Sabelfeld, A. Password meters and generators on the web: From large-scale empirical study to getting it right. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy* (2015), ACM, pp. 253–262.

[38] van de Kamp, T., Peter, A., Everts, M. H., and Jonker, W. Private sharing of iocs and sightings. In *Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security* (2016), ACM, pp. 35–38.

[39] Vatsalan, D., Sehili, Z., Christen, P., and Rahm, E. Privacy-preserving record linkage for big data: Current approaches and research challenges.

[40] Wagner, C., Dulaunoy, A., Wagener, G., and Iklody, A. Misp: The design and implementation of a collaborative threat intelligence sharing platform. In *Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security* (2016), ACM, pp. 49–56.

[41] White, G., and Harrison, K. State and community information sharing and analysis organizations. In *Proceedings of the 50th Hawaii International Conference on System Sciences* (2017).

[42] Xu, D., and Ning, P. Privacy-preserving alert correlation: a concept hierarchy based approach. In *21st Annual Computer Security Applications Conference (ACSAC'05)* (2005), IEEE, pp. 10–pp.

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**