



FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Timo Mattila

**INTEGRATION OF ARCTIC NODE
THREAT INTELLIGENCE SHARING
PLATFORM WITH SURICATA**

Master's Thesis
Degree Programme in Computer Science and Engineering
June 2020

Mattila T. (2020) **Integration of Arctic Node threat intelligence sharing platform with Suricata**. University of Oulu, Degree Programme in Computer Science and Engineering, 59 p.

ABSTRACT

The Internet has connected the modern world. Nowadays anyone can access anything straight from their home computer. Everything and everyone has a presence in the cyber domain, including those who seek to conduct criminal activities. In response to this, the field of cyber security has been born.

Cyber threat intelligence refers to information about cyber threats such as computers with open vulnerabilities and malicious Internet sites. The amount of available information is vast and the only way to handle such a large amount of data is automation. Threat intelligence sharing platforms have been developed for this task. They are used to fetch threat intelligence data from multiple sources, harmonize and analyze the data, and share it further.

One part of the automation process is the integration of threat intelligence sharing platforms with other cyber security applications. The goal of this thesis was to integrate a new intrusion detection and prevention system into the Arctic Node threat intelligence sharing platform. Suricata was chosen as the integrated system. A new integration submodule was created for Arctic Node to convert threat intelligence collected by the platform into Suricata rules and send it automatically to Suricata. One of the prominent features of this new module was the capability to deduplicate the output data.

The new integration submodule was compared to a similar functionality in another threat intelligence sharing platform, MISP. Testing was conducted in a custom virtual environment using real threat intelligence from seven different feeds. The results of these tests indicated that the new submodule was able to notice a greater number of possible threats, and it generated a more diverse set of different types of Suricata rules from the same input data. Deduplication was found to only have a small impact in reducing the size of the generated rule set as the overlap between different threat intelligence feeds was minimal.

Keywords: Cyber security, Cyber threat intelligence, threat intelligence management, intelligence feeds

Mattila T. (2020) Arctic Node -alustan yhdistäminen Suricata-ohjelmaan. Oulun yliopisto, Tietotekniikan tutkinto-ohjelma, 59 s.

TIIVISTELMÄ

Internet on yhdistänyt nykyajan maailman. Kaikilla on pääsy kaikkialle suoraan kotikoneelta. Myös rikolliset ovat havainneet tämän, ja hakkerointi on noussut modernin maailman suureksi riesaksi. Internetissä tapahtuvan rikollisuuden estämiseksi on syntynyt kyberturvallisuuden ala.

Tietoturvatieto käsittää tietoa eri uhkatekijöistä internetissä. Tieto voi koskea esimerkiksi vaarallisia sivustoja tai haavoittuvaisia tietokoneita. Tietoturvatietoa on olemassa valtavasti, eikä kaikkea tätä tietoa voida hallita manuaalisesti, vaan sen hallinta täytyy automatisoida. Tietoturvatiedon hallintaa varten on kehitetty erilaisia tietoturvatiedon hallinta-alustoja, joilla tietoa voidaan kerätä eri lähteistä, rikastaa, analysoida ja jakaa eteenpäin.

Yksi tietoturvatiedon hallinta-alustojen toiminnallisuuden osa-alueista on niiden integraatio toisten tietoturvaohjelmien kanssa. Tämän diplomityön tavoitteena oli yhdistää uusi tietoturvaohjelma Arctic Node -alustaan. Yhdistettäväksi ohjelmaksi valittiin Suricata. Diplomityössä tehtiin toiminnallisuus, joka muuttaa Arctic Node -alustan keräämää tietoturvatietoa Suricata-säännöiksi ja joka kykenee jakamaan säännöt automaattisesti Suricata-ohjelmalle. Yksi tämän toiminnallisuuden tavoitteista oli poistaa toisto luoduista Suricata-säännöistä, jotta jokainen sääntö olisi uniikki.

Arctic Node -alustan uutta toiminnallisuutta testattiin MISP-alustan samanlaista toiminnallisuutta vastaan. Testin tuloksena oli, että uusi toiminnallisuus tuotti jonkin verran enemmän Suricata-sääntöjä kuin MISP-alusta samoilla tietoturvalähteillä, ja säännöt olivat myös monipuolisempia kuin MISP-alustan luomat säännöt. Testeissä Arctic Node -alustan tuottamat säännöt havaitsivat haitallista liikennettä paremmin kuin MISP-alustan tuottamat säännöt. Testeissä ei havaittu Suricata-sääntöjen toiston poistamisen olevan kovinkaan merkittävää, koska tietoturvalähteiden välillä on melko vähän päällekkäisyyksiä.

Avainsanat: tietoturva, kyberturvallisuus, tietoturvatieto, tietoturvatiedon jakaminen

TABLE OF CONTENTS

ABSTRACT	
TIIVISTELMÄ	
TABLE OF CONTENTS	
FOREWORD	
LIST OF ABBREVIATIONS AND SYMBOLS	
1. INTRODUCTION	7
1.1. Cyber threat intelligence	8
1.1.1. Categories	9
1.1.2. Indicators of Compromise	11
1.2. Cyber threat intelligence feed	11
1.2.1. Event	12
1.2.2. Sharing	12
1.3. Cyber threat intelligence sharing platform	13
1.3.1. Challenges	14
1.4. Intrusion Detection and Prevention Systems	16
2. STATE OF THE ART	19
2.1. History of intrusion detection and prevention systems	19
2.2. Open source intrusion detection and prevention systems	19
2.3. Suricata	21
2.3.1. Suricata rule system	21
2.4. Threat Intelligence Sharing Platforms	23
2.5. MISP	24
2.6. Arctic Hub and Node	25
2.6.1. Structure and Features	25
3. METHODOLOGY	30
3.1. Suricata rule generation	30
3.1.1. The structure of the generator	31
3.1.2. IP rules	34
3.1.3. URL rules	34
3.1.4. User interface of the generator	35
4. CASE STUDIES	37
4.1. The test setup	37
4.2. The metrics for comparing different rule sets	41
4.3. MISP virtual machine	42
4.4. Feeds	42
4.5. Input Data	45
4.6. Test Cases	46
4.7. Test result	47
4.8. Future Work	51
5. CONCLUSION	53
6. REFERENCES	54
7. APPENDICES	57
7.1. MISP feeds	57

FOREWORD

The goal of this thesis was to integrate the Arctic Node threat intelligence sharing platform with a new intrusion detection and prevention system. Suricata was chosen as the integrated system. This thesis was done at the University of Oulu between 2019 and 2020 for the Arctic Security company. I work at the Arctic Security company, and have been part of the development of Arctic Hub and Arctic Node for several years. Due to my prior experience in the field it was a natural option to write my thesis on the subject of threat intelligence sharing platforms.

I thank everybody who has helped me with my studies, thesis and work. There have been moments when it felt like I was never going to get this done, but your help has kept me on the right path. Special thanks also go to those who have helped me proofread this paper.

Many thanks to my family and friends, who have always been there for me. I want to also thank a young puppy named Vyöti who came into this world and my life during the year and a half I've spent working on this thesis. Vyöti has reminded me that there is more to life than graduation and has helped me to take breaks from work.

Oulu, 16th June, 2020

Timo Mattila

LIST OF ABBREVIATIONS AND SYMBOLS

ASN	Autonomous System Number
C2	Command and Control
CTI	Cyber Threat Intelligence
CTIF	Cyber Threat Intelligence Feed
DGA	Domain Generation Algorithm
HIDS	Host based Intrusion Detection System
HIPS	Host based Intrusion Prevention System
IDS	Intrusion Detection System
IoC	Indicator of Compromise
IPS	Intrusion Prevention System
NIDS	Network based Intrusion Detection System
NIPS	Host based Intrusion Prevention System
IDPS	Intrusion Detection and Prevention System
SIEM	Security Information and Event Management
TI	Threat Intelligence
TTI	Technical Threat Intelligence
TISP	Threat Intelligence Sharing Platform
QA	Quality Assurance
VM	Virtual Machine
WIDS	Wireless Intrusion Detection System
WIPS	Wireless Intrusion Prevention System

1. INTRODUCTION

Over the years the world has grown dependent on the Internet. It is used by both individuals and organizations for various purposes such as business, sharing information, shopping, socializing, recreation, and controlling devices. Everything and everyone has a presence in the cyber domain, including those who seek to conduct criminal activities. The possible consequences of these criminal activities range from the simple defacement of websites to the destabilization of the power grid and espionage. With everything connected to the Internet, all facets of modern society can be targeted and hit by malicious actors.

The field of cyber security has risen in response to the cyber criminals. Just like in the security field in the physical world, there are various forms of cyber security. Instead of locks, guards, and walls, the tools of cyber security include encryption, antivirus, and firewalls. These forms of security are all centered around the individual target and its protection. However, fighting off all the threats alone is not the only option; there is also security in numbers. The malicious actors do not hide in the shadows and strike without warning. Their assets can be identified and information about their attacks can be shared with others so the targets can prepare in advance for the possible strike.

Sharing of cyber threat intelligence, or threat information, has become an important part of cyber security. In the past this was done via improvised channels such as emails and phone calls, but nowadays information is spread via specialized cyber threat intelligence sharing platforms and other advanced channels. Various nations and organisations such as NATO are investing into cyber threat intelligence sharing [1]. This intelligence can be used in multiple different ways including both passive and active applications. Communicating threat intelligence to other parties creates a net benefit for the whole community. For example, if a network of infected computers conduct a cyber attack against a company, the victim of the attack can inform other companies and organisations in the cyber threat intelligence sharing community of this new threat. This gives the potential next targets time to patch their systems so that the same exploits cannot be used against them and to block the IP addresses used in the previous attack. Information about the cyber attack could also be forwarded to the Internet providers used by these bot computers in order to cut off their access and inform the owner of the computers that their assets have been infected.

Large quantities of low-level threat intelligence are readily available online, and various cyber threat intelligence feeds generate new information constantly. By using this information, cyber attacks can be detected and stopped before they have time to cause more harm. Forms of active protection against cyber threats include intrusion detection and prevention systems. Some of these systems require rules which describe to the system what types of items are malicious. These rules can be automatically generated from the information of different threat intelligence feeds.

Threat intelligence sharing platforms can be used as a bridge to combine threat intelligence feeds with intrusion detection and prevention systems. These sharing platforms collect information from multiple sources and store it in one central location. They are thus a gathering point for threat intelligence similar to how

books are collected together to form libraries. Where libraries make it possible to search for information from a great number of books at once, threat intelligence sharing platforms make it possible to access information from multiple threat intelligence sources and to act on the information from all of those sources at the same time. Therefore, adding a way to manipulate the data collected by these platforms makes it possible to act on information gained from hundreds of different feeds at once instead of having to go through them one by one.

Using multiple feeds as sources for the automatic rule generation comes with several problems. One of these is overlap between feeds. If two or more feeds happen to report the same source, the generator might produce multiple copies of the same rule. This can be prevented by including some form of deduplication, which ensures that multiple copies of the same rule are not created. Another problem is presented by the fact that the intrusion detection and prevention systems only have limited storage space for rules. While deduplication can help alleviate this problem, there is a need for some form of logic in generating the rules that serves to drop old and inactive entries which are no longer relevant.

The goal of this project was to create a new integration for the Arctic Node threat intelligence sharing platform providing a new intrusion detection and prevention system. The aim of the integration is to resolve the problems associated with using multiple sources of information with large quantities of data.

1.1. Cyber threat intelligence

A threat is the potential for violation of security when there is an entity or event which could cause harm [2]. Cyber threats refer to threats related to computer systems and information technology. The realization of a threat is called a threat action, and the entity that performs this action is called a threat agent or actor [2]. Both intentional and accidental threats exist. Intentional threats require an intelligent threat actor, e.g. a hacker, who can intentionally cause harm, or in other words attack a system. Accidental threats refer to cases where harm can be caused unintentionally by human error, equipment failure, or an act of God.

Threats can cause harm by exploiting vulnerabilities of the target system. Vulnerabilities are flaws in the design, operation, implementation and management of a system [2]. As seen in Figure 1, risk is the probability of a threat exploiting a vulnerability to cause a particular harmful event[2]. As threats require vulnerabilities to cause harm, there is little or no risk for a system to be harmed if it has no vulnerabilities, or alternatively if it has no potential threats.

A threat can be seen as an integral whole that consists of three elements: intent, capability, and opportunity [3]. Integral whole means that all three elements are required for a threat to be viable. A threat with less than three elements is called a potential threat. Intent is the goal or motivation behind the threat. Capability is access to the resources and knowledge required to exploit the vulnerability. Opportunity refers to access to the system or target that makes it possible for the threat to carry out an attack.

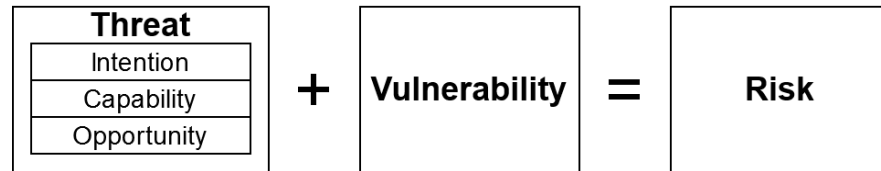


Figure 1. Threat and vulnerability together form the risk to a particular system. Threat consists of three integral parts, intention, capability and opportunity.

Cyber threat intelligence is data on cyber threats which has been processed or refined through some logical and analytical process to be relevant, actionable, and valuable to the end user [4]. The processing of the data can be done by either human or automated software.

Being relevant means that the intelligence has to relate to the one consuming it; for example, the attacks could potentially target them. For the intelligence to be actionable, there has to be a way to react to the data. One possible action for the end user is the informed decision to not act on the data. Finally, the intelligence must be valuable and have the capability to protect the business interests of the company in some way.

1.1.1. Categories

Threat intelligence can be divided into different categories. The UK Centre for the Protection of National Infrastructure divides intelligence into four categories based on the consumer of the data: strategic, operational, tactical, and technical [5]. These four categories can be split along two different axis, time and complexity level, as seen in Table 1.

The time axis describes the timescale on which the information is useful and should be consumed. Long-term threat intelligence yields benefit over time. Short-term threat intelligence gives information on immediate threats, but it also becomes outdated faster than long-term threat intelligence. For example, if a company gets operational threat intelligence that they might get attacked by hackers tomorrow, they must act today before the attack happens. After the attack the next day the threat intelligence is no longer useful. On the other hand, long-term tactical information on specific attack methodologies and tools can prepare a cyber security team to be ready for attacks which might happen within a year, and timing is no longer as crucial. However, the benefit of this preparation is not as immediately apparent as with shorter term information.

The complexity level axis indicates how complex the intelligence is as well as on what level and by whom it should be consumed. More complex intelligence should be consumed by those higher in the organizational hierarchy, as the information is more abstract and gives guidance on how to act on a bigger scale. Less complex intelligence is to be consumed by those lower in the hierarchy as it gives them greater detail on how to act. For example, tactical intelligence on the type of malware being used at the moment is not useful for the CEO of a company to help make large scale decisions on how to adjust the cyber defence budget. However,

the incident response team can use the same intelligence to build defenses for their systems against this particular malware.

Strategic threat intelligence is high-level information intended for the board or other senior decision-makers. It can help them allocate resources for cyber defence and make other high-level decisions which affect the company in long-term. Strategic threat intelligence comes in the form of reports, briefings, and conversations.

Operational threat intelligence is high-level information on near-future cyber attacks against the organization. It is intended to be consumed by senior cyber security staff. This can be information on what groups are planning to attack the organization and details on their capabilities and infrastructure.

Tactical threat intelligence is low-level intelligence on attacker tactics, techniques and procedures. It is intended for the defenders and incident response teams. Tactical intelligence can be information on how new attacks are conducted and how other organizations defend against those exploits. This information is meant for long-term use and can be gathered from studying the new vulnerabilities found by the cyber security community or by studying what malicious actors are doing in the wild.

Technical threat intelligence (TTI) is low-level intelligence on threats such as malicious IP addresses, URLs, and malware hashes, which are also called *indicators of compromise* (IOC). Technical threat intelligence is intended to be consumed by mechanical means, and the intelligence is only useful in the short term. Technical threat intelligence is spread by various means such as cyber threat intelligence feeds, and it is distributed in forms which are easy for machines to read, such as JSON or CSV.

Table 1. Four threat intelligence categories defined by the UK Centre for the Protection of National Infrastructure.

Name	Description	Level	Time
Strategic	High-level Information on changing risk.	High	Long-term
Operational	Details of a specific incoming attack.	High	Short-term
Tactical	Attacker methodologies, tools and tactics.	Low	Long-term
Technical	Indicators of specific malware.	Low	Short-term

Technical threat intelligence is on the edge between intelligence and data. While the UK Centre for the Protection of National Infrastructure does include indicators of compromise as one category of threat intelligence, it can be argued that they are not intelligence but rather just information on threats. It can be argued that lists of indicators of compromise, malware hashes and such lack the processing and analysis required to elevate them from information to intelligence. This does not mean that the data is useless, merely that the definition of intelligence in this context is somewhat strict. The ambiguity in the categorization can be resolved by splitting technical threat intelligence into the category of **threat information**, which is different from **threat intelligence** [6].

1.1.2. Indicators of Compromise

The two most common types of indicators of compromise are **malware indicators** and **network indicators** [5]. Malware indicators are information related to the malware used by attackers. Hashes of malware binaries are one common type of shared indicators. These can be used to detect malicious files, as all files with the matching hash are the same. Typical hash forms used by the industry are MD5, SHA-1 and SHA-256. The names of malware files and registry keys used by malware are another example of types of malware indicators. Network indicators are information on different internet resources used to spread and control malware. Common types of network indicators are IP addresses, domain names, URLs, URIs, and network artifacts such as user agent.

Threat intelligence is used to combat the efforts of threat agents such as hackers. In response to this, threat agents try to counter the efforts of the defenders by changing the resources or procedures used. Attackers can make changes on their end to render indicators of compromise obsolete [5]. Some indicators are harder to change than others, as seen in Figure 2. Changing malware hash values is trivial, as a change to any byte is enough to generate a new hash. Switching IP addresses takes more effort, but an attacker with a botnet of thousands of computers has a wide selection of addresses to use. On the other end of the spectrum, it would be difficult for attackers to craft entirely new malware capable of deceiving detailed intrusion detection system rules such as Yara. Similarly, if tactical threat intelligence is used to thwart the attacker's tactics, techniques, and procedures, it would require more effort for the attacker to come up with ways to attack the target system.

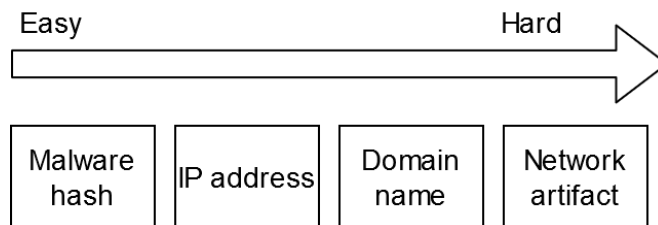


Figure 2. How difficult it is for attackers to change the indicator of compromise.

1.2. Cyber threat intelligence feed

A cyber threat intelligence feed (CTIF) is a source of threat intelligence or information. These feeds are provided by various organizations and individuals, and can be for profit or non-profit. Feeds can provide information of all different threat intelligence categories. However, technical threat intelligence is the most common, as it is relatively easy to collect compared to other categories. As technical threat intelligence is best suited for automatic consumption, and it has to be consumed within a short period of time, the steady stream of new information from threat intelligence feeds is very suitable for this purpose. This paper will mostly focus on feeds which provide technical threat intelligence.

Technical threat intelligence can be collected in various ways. Some feed providers utilize honeypots to collect information on threat actors [7]. Honeypots are systems which pose as a vulnerable service, e.g. a database which is accessible via the Internet and vulnerable to an SQL injection. Another possible source of information is malware analysis, where malware is analyzed in a sandbox environment in order to find out how it works and what command-and-control (C2) servers it tries to connect to [8]. A host of feeds work on a voluntary basis, collecting reports submitted by individuals whenever they come across a malicious site [9].

1.2.1. Event

The singular item of threat intelligence provided by a feed is called *an event*. An event is a discrete time-bound activity in which an adversary uses their capabilities and resources against a victim [10]. Several events are required for adversaries to achieve their goal.

Events can contain different kinds of information. Events provided by technical threat intelligence feeds typically contain different indicators of compromise such as IP addresses, domain names, and malware hashes. They may contain additional information about these indicators, such as the owner of the domain or what malware families the IP address is connected to. Events may also contain information on the victim or the adversary behind the event. All events must contain information on the time domain, when the event was observed, or, in the case of persistent threats, when it was last observed online. This is because technical threat intelligence is highly time sensitive.

1.2.2. Sharing

Threat intelligence feeds provide their events over the internet by various means. These methods can range from simple text files uploaded to a server or spread via email to elaborate web APIs. Open feeds which are free for anyone to use tend to use relatively simple approaches, while commercial feeds might need to use more complex approaches in order to make the information only accessible to paying customers.

Multiple different standards, ontologies, and taxonomies exist for sharing threat intelligence [11]. As feeds often report their findings using their own notation instead of any of the established ones, they expand the set of different notations used to describe threat intelligence. The variety of different forms of notation pose a challenge in the field of cyber threat intelligence. They make the automation of feed consumption more difficult, as the mapping between different notations has to be done manually and there may be confusion about the exact content of the events. Threat intelligence harmonization, normalization, and consolidation are some of the challenges that cyber threat intelligence sharing platforms aim to solve.

European Union Agency for Network and Information Security, ENISA, conducted a study in 2014 to find out the different standards and tools used to exchange and process threat intelligence [12]. In this study a total of 53 different information sharing standards were found. These were broken down into seven different categories as shown in Table 2. As the study was conducted in 2014, it is reasonable to assume even more standards have been developed since then.

Table 2. The seven ENISA categories of information sharing and transportation standards.

Name	Found in study	Description	Example
Formats for low-level data	5	Unprocessed data collected by security monitoring systems.	PCAP
Actionable observables	6	Indicators of compromise.	Snort rules
Enumerations	9	Standard which defines the meaning of vocabulary used by security community.	CVE
Scoring and measurement frameworks	4	Frameworks for quantitative description of threats.	CCSS
Reporting formats	10	Report structure for high-level details on threats.	STIX
High level frameworks	2	Process frameworks for exchanging security information.	CYBEX
Transportation and serialization	17	Different methods to transport and serialize threat data.	email, CSV

1.3. Cyber threat intelligence sharing platform

A cyber threat intelligence sharing platform (TISP) is a service that organizations can use to share and receive information on current cyber threats. These platforms can gather threat intelligence from multiple different threat intelligence feeds and unify the data into one common form. They may also process the data further and add new information, such as automatically adding geolocation data based on the IP address. In other words, threat intelligence sharing platforms help organizations consume existing threat intelligence. Another term used for cyber threat intelligence sharing platforms is threat intelligence management platforms, which underlines the fact that the platforms are used not only to share, but also control, direct, and organize threat intelligence.

NATO's concept of Cyber Security Data Exchange and Collaboration Infrastructure (CDXI) describes their view on cyber threat intelligence sharing [1]. According to the concept, cyber threat intelligence sharing platforms have three main objectives:

1. facilitation of information sharing

2. enabling automation
3. facilitation of the generation, refinement, and vetting of data

1.3.1. Challenges

Brown has identified eight main challenges cyber threat intelligence sharing platforms need to overcome in order to support current cyber security practices [6]. These high level requirements further break down the functionalities expected from threat intelligence sharing platforms. This is not an exhaustive list of all possible challenges which threat intelligence sharing platforms might need to address. Conversely, a platform does not have to address all eight to be considered a true threat intelligence platform. The challenges are:

1. diversity of intelligence sources
2. normalization and consolidation
3. enrichment
4. relevancy determination
5. complex analysis
6. production and delivery
7. integration
8. analytic management and collaboration

The diversity of intelligence sources is a problem when the various sources have to be combined to serve as the input for a threat intelligence sharing platform. Different sources use different standards and methods of sharing their information, which means no single solution can work for every source. The information quality differs from source to source, and metadata might get lost when information is transported to different systems. One problem related to diversity is simply the challenge of finding all the different sources available on the market.

The challenge of information normalization and consolidation arises from the diversity between sources. Sources might use different names for the same values (e.g. IP address, IP, target IP), which then have to be normalized into one format for automatic consumption. If threat intelligence is written by hand, it might contain spelling errors or different forms of spelling between authors. Sources might lack a specific description of the sort of information they provide or change the naming of the data, which can break automatic normalization. Consolidation is the merging of data, which can then be deduplicated so that only a single copy of each record is left. Consolidating data makes the information more concise, which makes it easier to consume large quantities of information by making them more dense.

Information gathered by threat intelligence sharing platforms can be further enriched with external sources. This could mean fetching additional network data related to different network indicators, adding geolocation data, or providing external links for detailed information on various malware. Enrichment can provide additional context to aid the consumption of the intelligence. Integrating these additional enrichment sources is another challenge faced by threat intelligence sharing platforms.

Users of threat intelligence sharing platforms may want to have a way to determine the relevancy of the data. Determining relevancy can be based on a multitude of different criteria such as the credibility of the information, the severity of the potential risk, and the depth of collection.

Cyber threats are complex problems that require tools for complex analysis. Threat intelligence management platforms are expected to provide tools for the user to search through, filter, and visualize the collected information. These complex tools can also help to facilitate a target-centric approach to threat intelligence [13], where the intelligence collectors, analysts and customers work together and communicate more freely across a network between all involved parties. This approach is a step forward from the more traditional process, where the intelligence life cycle starts from decision makers issuing a request for intelligence to be collected, processed, and analyzed, after which they receive the finished intelligence product. Traditionally, each phase of this cycle is isolated from the others. In the modern day environment this sort of rigid analysis process might not be enough to handle the complex cyber threats organizations face every day.

Different threat intelligence categories have different consumers, from the strategic intelligence consumed by high-level decision makers to the less complex technical threat intelligence meant for incident response teams. Threat intelligence sharing platforms need to be able to facilitate the production and delivery of threat intelligence for all levels of consumers. Fulfilling this need might require the ability for the platforms to automatically produce and deliver time-sensitive intelligence to those who need it. This can be done via emails or other communication channels.

Integration refers to the ability to integrate the threat intelligence sharing platforms into other cyber security technologies, such as various intrusion detection and prevention systems. This makes it possible to get the most benefit out of the technical threat intelligence. The variety of interfaces between different cyber security technologies requires flexibility from the platforms, which need to be able to output various types of information in different formats.

Analytic management and collaboration refers to the threat intelligence process of managing stakeholder needs, resources, tasks, work assignments and other factors. Threat intelligence sharing platforms should facilitate the process of governing threat intelligence analysis. Brown notes that these are often overlooked parts of cyber intelligence technologies, and that they are key components of a successful threat intelligence sharing platform.

1.4. Intrusion Detection and Prevention Systems

Intrusion is defined as deliberate attempts of causing harm to a system. The goal of intrusion may be to gain access to data with no authorization, trying to manipulate data with no authorization, or to disable the system [14]. The term intrusion includes both attempted and successful cases of intrusions against the target system.

Intrusion detection systems (IDS) are software designed to monitor computer activity and detect possible cases of intrusion [14]. The monitored computer activity can be anything, and there is a wide range of different intrusion detection systems which attempt to find evidence of intrusions in various ways. Some potential targets for monitoring are network traffic, files on the computer, and system log files. Intrusion detection systems log their findings and raise an alert if a positive case of intrusion is detected.

Intrusion prevention systems (IPS) are similar to intrusion detection systems, but in addition to detecting intrusions, they try to actively prevent intrusions [14]. Intrusion detection system can be seen as the next step from intrusion detection systems, but they are not a perfect replacement. Intrusion prevention can only work in real time, so these systems are limited to a very immediate timescale, while intrusion detection systems can be used to find evidence of past intrusions. False positives of intrusion prevention systems can be more harmful than false positives of intrusion detection systems, as they can disturb the normal usage of the computer system.

Intrusion detection systems can be categorised in different ways. One way of categorization is dividing them according to where they are deployed. Systems that are deployed on the host computer to directly monitor activity are called **host based IDS** (HIDS). Systems deployed on the network to monitor the traffic between multiple computers are called **network based IDS** (NIDS). The wide spread adoption of wireless internet connections has lead to the development of **wireless IDS** (WIDS) [15], which specifically monitors intrusion attempts over various wireless technologies and protocols, such as IEEE 802.11 b. These categories are also applied to the intrusion prevention systems, i.e. **host based IPS** (HIPS), **network based IPS** (NIPS), and **wireless IPS** (WIPS).

Examples of how network based systems and host based systems are deployed are shown in Figure 3. Host based systems are installed on each host machine and monitor the activity of that host. Network based systems, on the other hand, monitor a network of multiple machines. Network based systems are relatively easy to deploy and are independent of the platform of the host machine [14]. A single network based system can monitor the traffic of multiple host machines in a network. The effort to deploy network based intrusion detection systems decreases as the number of host machines in the monitored network increases.

Host based systems are capable of securing the host machines and have access to the clear text traffic as it is deciphered on the host machine [14]. This is not possible with network based system, as traffic over the network is usually encrypted. One of the drawbacks of host based systems is the additional strain on the host machine's resources.

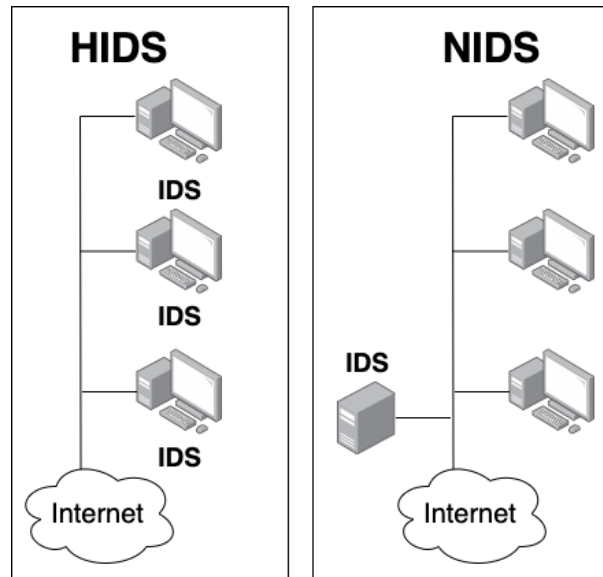


Figure 3. Example of the different deployment locations between HID and NID. Host based detection systems monitor each host in the network separately, while network based systems monitor the traffic between the network of connected computers and the internet.

Wireless intrusion detection systems focus on protecting wireless networks from intrusions. Wireless networks can be attacked in various ways, such as masquerading as another wireless access point or jamming the radio frequency [15]. To be able to detect these intrusion attempts, the detection system must be deployed in such a way that it can monitor wireless network activity. One example would be to deploy the system on a device with a wireless network sensor so that it can monitor wireless network traffic [15].

Intrusion detection and prevention systems can also be divided into different categories based on the method which they use to detect the intrusion. There are three categories: anomaly recognition, misuse recognition, and stateful protocol analysis [16]. Intrusion detection systems may also combine different detection methods to form hybrid methods.

Misuse recognition systems try to detect intrusions by trying to match traffic against signatures of known malicious activity [14]. Some examples of these signatures are patterns of malicious software or addresses of known malicious internet sites. Misuse recognition systems are also known as rule-based detection systems, signature detection systems, or pattern matching systems.

Systems based on **anomaly recognition** create a profile based on the normal traffic or system behaviour. These systems find out intrusions by first building a baseline of what is normal activity, and then analysing ongoing activity in order to find cases that differ from normal behaviour [16]. Aspects that may be monitored include the number of connections or traffic to new sites, processor usage, and whether a user has accessed files which they normally do not. The strength of anomaly recognition systems is that they can react to new, previously unknown threats. On the other hand, anomaly recognition systems have a higher rate of false alarms than other detection methods.

Stateful protocol analysis, also known as specification-based detection, tracks the state of protocols such as TCP. Intrusions are detected by monitoring for cases where the state differs from the specification. This can be a useful way to detect cases where a protocol is being exploited by using it in an unintended way, but the method is incapable of detecting attacks which do not differ from the normal use [16].

2. STATE OF THE ART

This section will look briefly into different intrusion detection and prevention systems, as well as the two threat intelligence sharing platforms, MISP and Arctic Node, which were used in the test. Five intrusion detection and prevention systems were considered as candidates for the target system for the generator. They were selected as candidates because all of them were open source and were considered to be of interest at Arctic Security.

After evaluating all options Suricata was chosen as the system to be integrated by the Arctic Security company. This is because Suricata is widely used, open source, and easier to work with than the other candidates. It is described on a more technical level than the rest of the intrusion detection and prevention systems to clarify the capabilities and structure of Suricata rules.

Arctic Node is also described on a more technical level in order to understand how Suricata can be integrated with the threat intelligence sharing platform. The section describing Arctic Node and Hub has been vetted by Arctic Security in order to make sure the level of detail in which the work is described is acceptable.

2.1. History of intrusion detection and prevention systems

First intrusion detection systems were developed in the 1980s by the U.S. Air Force. The basis for the idea was a paper written in 1980 called Computer Security Threat Monitoring and Surveillance [17], which described how audit records could be used to identify computer misuse, and how statistical analysis could be used to detect anomalies in user activity.

This theory was put into practice by a research team led by Dr. Dorothy Denning, who developed a prototype intrusion detection system called the intrusion detection expert system (IDES). Dr. Denning published a paper on the real-time intrusion model [18] that described the system in more detail.

Intrusion detection systems continued to develop through the 1980s and 1990s. U.S. Air Force continued to develop different systems, and commercial systems such as the Net Stalker and Network Flight Recorder began to emerge in the 1990s [14]. The first few intrusion detection systems focused on analyzing audit records for signs of intrusions. Subsequent systems soon shifted into monitoring internet traffic as the internet grew more prominent. As the technology matured towards the late 1990s, the first intrusion prevention systems were created.

2.2. Open source intrusion detection and prevention systems

Various commercial and open systems have been developed during the 30 year history of intrusion detection systems and the 20 year history of intrusion prevention systems. This section gives a brief review of the intrusion detection and prevention systems which were considered to be used in this thesis.

Zeek is an open-source intrusion detection system designed to monitor network traffic. Its development began in 1995 by Vern Paxson [19], and is nowadays

handled by a team of researchers at the International Computer Science Institute in Berkeley [20]. Zeek was developed with seven goals: high speed, no packet drops, real-time notification, mechanism separate from policy, extensibility, avoiding simple mistakes, and the assumption that the monitor will be attacked [19]. Zeek uses a versatile event-driven scripting language. Zeek is licensed under a BSD licence. It was previously known as Bro, but the name was changed to Zeek in 2018 [21].

Yara is a host based intrusion detection system which is designed to be used by malware researchers to identify malware samples. Yara uses a misuse recognition system as the detection method. It has its own rule language that can be used to write patterns, which are then searched for in the scanned files [22]. The rules consist of two logical parts, strings and conditions. Strings can be text, binary, or regular expressions, and the condition part dictates how they should be matched. For example, the string part could define three variables, \$A, \$B and \$C, and if matching one of them was enough to determine the file a match, the condition part would read \$A, \$B or \$C. The conditions can also refer to other rules, enabling the creation of complex rules. A third, non-logical part of the rule is called the meta, where it is possible to give additional information about the rule, such as the author of said rule.

Sigma is a host based intrusion detection system for log files and security information and event management systems (SIEM). The first version of Sigma was released in 2017, and the full version is not yet finished. Like Yara, Sigma uses a misuse recognition system. The rule language used by Sigma has been developed to be generic for all log events. For example, it can be used to create rules for both Linux and Windows based system logs. The rule language is supposed to be supplemented by a Sigma Converter, which could be used to generate SIEM queries from the Sigma rules. As of 2019, this converter is still under development. [23] Despite being a relatively new addition to the field, Sigma has already caught some outside interest. For example, the MISP project has support for sharing Sigma rules.

Snort is an intrusion detection system created by Martin Roesch in 1998 [24]. Snort was developed as a free, lightweight alternative to the commercial intrusion detection systems of the day. Since then, Snort has grown in popularity, and is now in wide use. Snort can be used to monitor, log, and drop traffic based on rules. The language of these rules is complex, and offers a flexible tool to monitor network traffic in various ways.

In 2001, Martin Roesch founded the Sourcefire company to develop Snort for the commercial market [25]. Sourcefire was bought by Cisco Systems in 2013, and is currently being developed by the Cisco Talos Intelligence group. Snort is developed as an open-source project under the GPLv2 licence. Recently, the developers of Snort have begun to develop a new version of Snort from the ground up, called the Snort3 project [26]. It will boast new features such as multi-threading, better cross platform support, and automatic generation of reference documentation.

2.3. Suricata

Suricata is an intrusion detection and prevention system developed by Open Information Security Foundation (OISF) as a open-source project licensed under the GPLv2 license. Suricata has been in development since 2009 [27]. OISF is a non-profit organization funded by the US department of Homeland Security and various private companies in the OISF Consortium.

Suricata shares some similarities with Snort, as its rule system is an extended version of the one used by Snort. However, unlike Snort, Suricata has been developed from the beginning to support multiple threads. Over its development Suricata has managed to surpass Snort in both single-threaded and multi-threaded environments [28]. Suricata drops fewer packets than Snort in five out of six different test cases, with Snort outperforming Suricata only when testing the detection of fragmented package attacks. Suricata has higher utilization rates of system memory and CPU power than Snort. Both Snort and Suricata work better on Linux based machines than on Windows platforms.

Suricata splits the task of detecting alerts into four different thread-modules:

- Packet acquisition,
- Decode and stream application layer,
- Detection and
- Output.

The *packet acquisition* module reads packets from the network stream. The *decode and stream application layer* module decodes the packets, constructs them into a stream, and reads the application layer data from the packet. The *detection* module is responsible for running the detection logic and matching the packets to the rules given to Suricata. There are often multiple instances of the detection module running to speed up the handling of packets. Finally, the detection modules feed their results into the *output* module, which is responsible for writing the results on disk. Figure 4 shows one possible way to configure a Suricata instance. Three detection modules are running in parallel to speed up the processing of the live network traffic. When monitoring faster networks it is possible to add even more modules to keep up with the increased traffic.

Suricata comes with a rule download and management tool called **Suricata-update**. This tool can be given new sources from which to download Suricata rules. A new source can be added to Suricata-update by supplying the URL, possible HTTP headers, and user agent. Suricata-update has been written in Python.

2.3.1. Suricata rule system

The structure of the rule system used by Snort and Suricata consists out of the following parts:

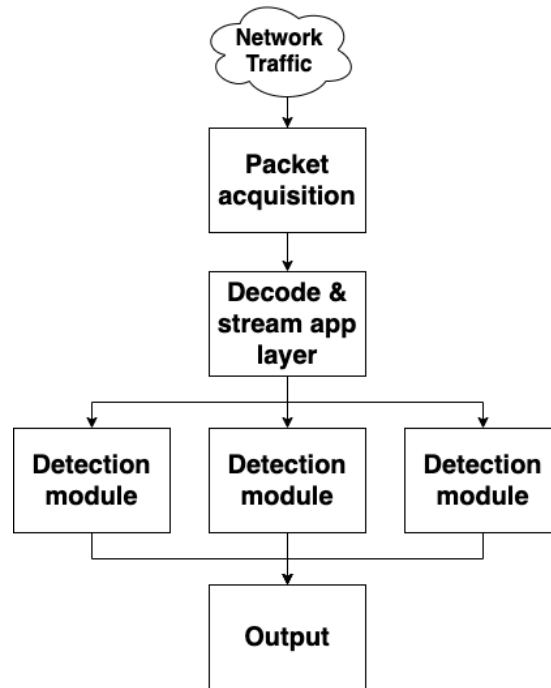


Figure 4. Example of a possible Suricata thread-module configuration with three detection thread-modules.

1. action
2. protocol
3. source
4. source port
5. direction
6. destination
7. destination port
8. options

Action dictates what to do when a match to the rule is found. Possible actions include pass, drop, reject, and alert. Passing means no action is performed when a match is found. Drop and reject actively block the package from getting through, and in addition generate an alert message. Alert lets the package pass through, but logs an alert for the system administrator to see.

Protocol dictates which types of packages are scanned by the rule. The possible values include, for example, *tcp*, *udp*, *ip*, *dns*, *http*, *ftp*, and *ssh*. Protocols may also have their protocol-specific options, which can help to create better rules.

Source and **destination** are used to configure the source and destination of traffic. Their values can include either IP addresses or ranges, along with variables. Some of the typical variables are *\$HOME_NET* and

\$EXTERNAL_NET. These variables can be configured in the Suricata configuration file for better accuracy. Multiple IP addresses can be grouped together, e.g. [1.2.3.4/16, 5.6.7.8].

Direction is used to configure the direction of traffic between the source and destination. The possible symbols are `->` and `<>`. The former (`->`) is used to denote traffic from one end to the other, but not the other way around. The latter (`<>`) is used to denote traffic passing in any direction between the two ends.

Source port and **destination port** are used to set monitoring for traffic from specific ports. More than one port can be monitored by using ranges (`[80:90]`, `[80:]`), grouping (`[80, 85, 90]`), or negation (`!80`).

Options are listed at the end of the rule, enclosed by parentheses and separated by semicolons. Options can be either keywords, e.g. `http_header`, or key-value pairs, e.g. `content:"google"`. Some of the keywords are protocol specific, for example the `http_header` keyword can be used in rules where the protocol is http to guide the content search to the content of http header. The `msg` keyword can be used to log additional information when the rule is triggered, e.g. `msg:"this connection originated from a known malicious host"`. Rules also typically contain a signature ID, which can be used to identify each rule. This is usually added as the last or second-to-last option in a rule. The keyword of a signature ID is `sid`, which is the unique Suricata ID value of the rule.

For example, let us assume that the IP address 1.2.3.4 is known to be a C2 server, and we wish to block any access to it from our home network. The following rule could be crafted to achieve this:

- `drop ip $HOME_NET any -> 1.2.3.4 any (msg:"Block access to a known C2 server"; sid: 1000000)`

This rule would drop all packages originating from the home network to the IP address 1.2.3.4, as well as raise an alarm with the message *"Block access to a known C2 server"*, which would alert the system administrator. Note that traffic flowing in the opposite direction, from 1.2.3.4 to the home network, is permitted by this rule.

2.4. Threat Intelligence Sharing Platforms

Cyber security professionals agree that threat intelligence sharing is beneficial for organizations [29]. It can help organizations to be more aware of their security breaches and help them to increase their resilience against cyber threats. Setting up the infrastructure required for threat intelligence sharing is expensive upfront but is generally seen to bring down the security costs in the long term. On the other hand, inconsistent terminology, varying quality of data, and redundancy are seen as the main problems of threat intelligence sharing. The lack of training on how to utilize threat intelligence is also seen as a problem in the field by cyber security professionals.

A study was conducted in 2016 [30] where a workshop of cyber security experts evaluated the at the time cutting edge of threat intelligence sharing platforms.

They identified and analyzed 22 different threat intelligence sharing platforms on the current market. These platforms were split into four types:

- sharing of threat intelligence,
- sharing aggregated data,
- security information repository and
- other.

The study had eight key findings. First of all, no common definition was found for threat intelligence sharing platforms. While some platforms focused on sharing threat intelligence between different organizations, others focused on collecting and sharing automated information from various threat intelligence feeds. The majority of platforms seemed to focus on collecting and sharing indicators of compromise, while analysis was given less importance. STIX was the de-facto standard used to describe threat intelligence. From the results of this study it appears only some of the challenges described by Brown [6] are answered by current threat intelligence sharing platforms.

The different threat intelligence sharing platforms answer different needs in the market. There is a need for sharing indicators of compromise and new rules for intrusion detection and prevention systems in the market, and the information is readily available in the Internet. This might explain why several threat intelligence sharing platforms focus mostly on sharing this type of low-level information rather than well analyzed threat intelligence. Such intelligence is simply more difficult to generate. In the strict sense, these sort of platforms are not sharing threat intelligence, yet they are identified as threat intelligence sharing platforms. From this it is evident that the term *threat intelligence sharing platform* is being used in a rather loose sense by the community.

2.5. MISP

MISP (Malware Information Sharing Project) is an open source threat intelligence platform. It is used for storing, sharing, and working with indicators of compromise. Christophe Vandeplass started MISP (first called CyDefSIG - Cyber Defence Signatures, later changed to MISP) as his personal project back in 2011, and it soon came to be used by his employer, the Belgian Defence. In 2012, the project caught the attention of NATO, who hired more developers to help Vandeplass with MISP [31]. As of 2019, the MISP project is being funded by CIRCL, Computer Incident Response Center Luxembourg, and the EU [32].

MISP has support for 57 open feeds (full list in appendices) and provides rudimentary tools for integrating additional feeds which can be fetched with a simple HTTP request. MISP is capable of consuming feeds in either CSV or MISP format. Feeds that consist of plain text can be consumed by utilizing regular expressions. These custom feeds can be given headers, and they have support for basic access authentication. Events can be exported from MISP

in various different formats. These formats include Snort, Suricata, Bro, CSV, OpenIOC, JSON, MISP XML, STIX and plain text.

MISP has automatic correlation to detect overlapping data between different events. These events can be visualized with an event graph, which shows the relationships in graphical form. MISP also facilitates the collaborative sharing and analysis of threat intelligence. Intelligence can be shared via different channels such as emails or the MISP API, and different MISP instances can communicate with each other.

2.6. Arctic Hub and Node

Arctic Hub is a commercial closed source threat intelligence sharing platform developed by Arctic Security. It was launched in 2018, and is based on the earlier AbuseSA product by the same company [33]. Hub is complemented by another product by Arctic Security, Arctic Node [34]. The products share parts of their code base and functionalities, and are intended to work together as a cyber defence cell [35]. These cells consist of organizations from the same interest group, including both threat intelligence providers and consumers. Each cell consists of multiple consumers centered around one threat intelligence producer. Each cell works together to share threat intelligence bidirectionally and strengthen their cyber defence. Cells can also form a larger network of multiple cyber defence cells.

Arctic Hub has been developed as a platform to be used by the cyber defence cell's threat intelligence provider. It has the capability to collect threat intelligence from multiple sources, enrich it with new external data, analyze the results, and share it onwards. Arctic Node has been developed to serve as the main platform for the consumers in the cyber defence cell. With Node, they can receive threat intelligence from the central Hub and integrate the information into their attached cyber security solutions such as intrusion detection and prevention systems. Both Arctic Hub and Node are threat intelligence sharing platforms, but they do not share all the same responsibilities and capabilities. Using both platforms together in a cyber defence cell brings together all the different functionalities that are expected from threat intelligence sharing platforms. An example of a cyber defence cell formed out of one Arctic Hub and two Arctic Nodes is depicted in Figure 5.

2.6.1. Structure and Features

Arctic Hub and Node consist of two parts, backend and frontend. Both platforms work as a typical web service. They can be deployed on an on-premise or cloud server that meets the hardware requirements. A deployed system can be accessed via a graphical user interface (GUI) on a web browser. The frontend provides the GUI and has the capability of visually depicting the stored data in various different ways. It is also used to configure the system. The backend manages

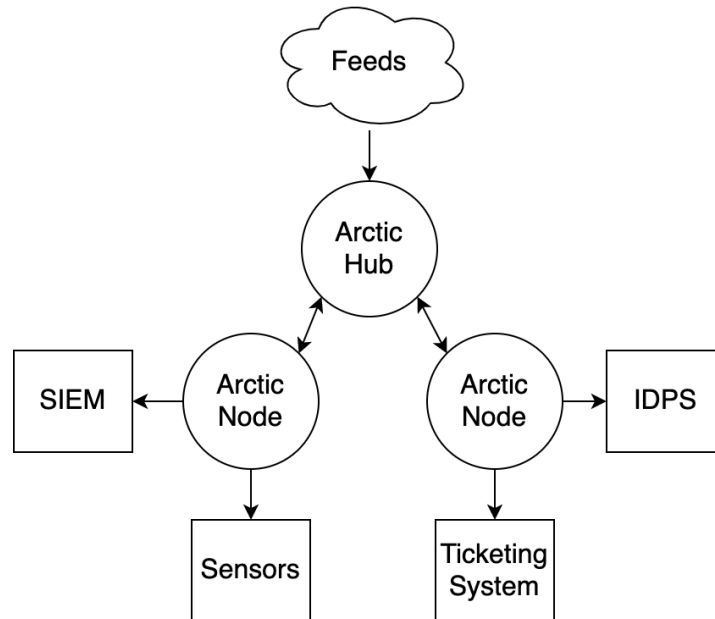


Figure 5. A cyber defence cell which consists out of a single Hub and two Nodes. The Hub fetches new threat intelligence from various feeds, which it shares with the two Nodes. The Nodes move the threat intelligence over to the various security systems, as well as give feedback back to the Hub.

the rest of the systems running on the server. The main parts, i.e. modules, of backend are:

- configurations,
- authentication,
- feeds,
- labeling,
- storage,
- sharing,
- analytics and
- integrations.

The frontend interacts mainly with the backend via configurations, authentication and storage. **Configurations** stores the configurations of other parts of the backend, such as feed and integrations configurations. Configurations can be changed via the graphical user interface. The frontend also interacts with the **Authentication** module, as it authenticates all connections which originate from the outside.

Feeds handles the different feed integrations of Arctic Hub and Node. They both have support for over a hundred different public, restricted, and commercial

feeds [33]. These feed integrations are actively maintained to keep up with possible changes to the data or delivery methods. Data gathered from various feeds is harmonized according to data harmonization ontology, which has been developed in collaboration between CERT-EU, CERT-AT, CERT-FI, CERT.PT and Arctic Security [36]. The goal of data harmonization is consistency across all sources. The data harmonization ontology allows using keywords outside of the core language. It does not define the limited set of keywords allowed to be used in harmonization.

Information gathered from the various feeds is enriched with additional data such as the geographical location of the IP address and DNS records of the domain. Events are also given additional metadata such as information that helps to track the source, when the event was created, and a human readable description. Finally, the events can be enriched with user specified information. Users of Arctic Hub and Node can use the **labeling** to add additional data to the events if the event matches a filter written by the user. Once events have been fetched, parsed, harmonized, augmented (the term for enrichment used by Arctic Security), and labeled, they are saved to **storage**, which is a MongoDB database. Other parts of the system can then retrieve the events from this database. Events are fetched from storage using Arctic Node rule language. The rule language can be used, for example, to fetch events from the same country.

The following public feeds and feeders are supported by Arctic Hub and Node:

- Abuse.ch,
- Bambenek Consulting feeds,
- Brute Force Blocker,
- Malware Domain List,
- VxVault,
- Zone-h and
- Phishtank.

The rest of the feeds supported by Arctic Hub and Node are either commercial or restricted to certain customers. These feeds are not listed here.

As the central point of a cyber defence cell, Arctic Hub is intended to be used by organizations which monitor the cyber security of their stakeholders. One of the main features of Arctic Hub is improving the situational awareness by automatically mapping indicators of compromise against the assets of the stakeholders. The Hub user can add the different IP ranges and domains used by their stakeholders to the Hub. When Arctic Hub receives new events with IP addresses of domain names belonging to the stakeholder, it can automatically notify them via email, API, or Arctic Node if they are using it. This way stakeholders receive actionable threat intelligence which is relevant to their usage, i.e. indicators of compromise which are directly related to their assets. The mapping of assets is done by **labeling**, and **sharing** takes care of delivering the information to the customers.

Analytics calculates various different metrics from the stored data, such as how many events have been recorded per day. Users can define filters for what events should be included in the analysis. For example, users may wish to show events from certain countries or only C2 infrastructure. These metrics can be visualized in various forms such as lists or bar charts. Metrics can be grouped together on one page to form a dashboard. An example of the dashboard view on Arctic Node can be seen in Figure 6.

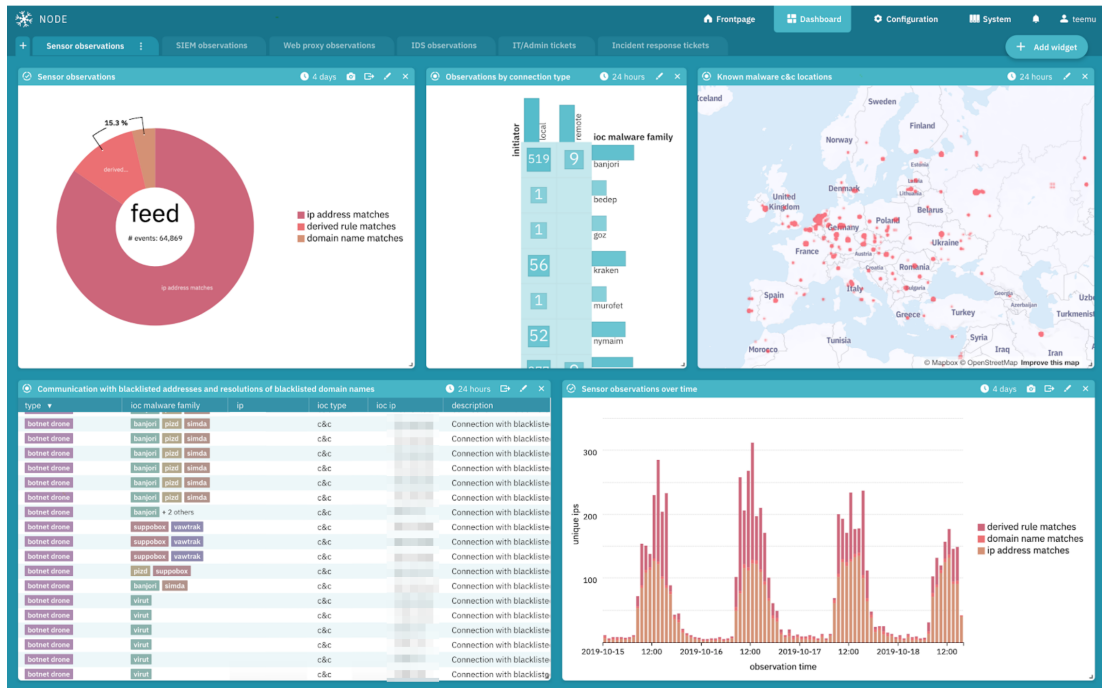


Figure 6. Arctic Node dashboard. IP addresses have been blurred due to privacy reasons.

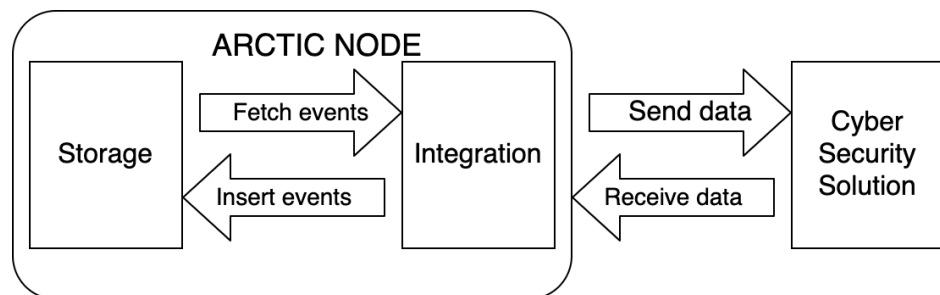


Figure 7. Arctic Node's integration module works as the interface between Node and different cyber security solutions. Information can flow both ways, with the module working as the translator between different data forms.

Arctic Node can be integrated with various cyber security solutions. These solutions include intrusion detection and prevention systems, SIEMs, ticketing systems and network sensors. These integrations are handled by the **integrations** module. Integrations are configured through the Arctic Node GUI. Arctic Node

can communicate with integrated systems over the internet in both directions, either by automatically sending new information to the integrated system, or fetching information from the integrated system. The exact details can differ between integrations. Figure 7 depicts how the integration module acts as the interface between Arctic Node and different cyber security solutions. The integration module turns the events stored in the database into a form that is suitable for the cyber security solutions and vice versa. Each Arctic Node integration runs as a separate process. This means that one Arctic Node instance can have multiple copies of the same integration running at the same time.

3. METHODOLOGY

A key functionality of threat intelligence sharing platforms is their integration with other existing cyber threat security applications. This allows threat intelligence to be automatically consumed by various means without the need for human interaction.

In this project, a new integration between the Arctic Node threat intelligence sharing platform and the Suricata intrusion detection and prevention system was created. This integration is called the **Suricata rule generator**. The goals of this integration were to turn threat intelligence stored in Arctic Node into Suricata rules and to serve them in such a way that they can be downloaded automatically by Suricata. It should also be possible to control which feeds are turned into Suricata rules, and the size of the generated rule set should be kept small enough for Suricata to handle.

As different feeds collect threat intelligence individually, their findings might contain overlapping data. This overlap might be useful for a researcher to notice relatively active pieces of criminal infrastructure, but with automatic generation of intrusion detection or prevention rules, this might lead to multiple copies of the same rule. These redundant rules take up space and computational power, as the system has to check the same rule multiple times during runtime. The Suricata rule generator was developed with this issue in mind, and will not produce redundant copies of rules.

The performance of the submodule is measured by testing it against a similar feature in MISP. Both MISP and the submodule are run through a similar test setup, after which their results are compared. First, both a MISP instance and a Hub instance are used to fetch same feeds. This threat intelligence is exported from both of the instances as Suricata rules. These Suricata rule sets are tested in a test setup with two virtual machines, one of which runs an instance of Suricata. The other machine is used to create internet traffic between these two machines that the Suricata instance monitors. This setup is used to measure the performance of the different rule sets.

3.1. Suricata rule generation

Any traffic or contact attempt between the host machine and the C2 server can be deemed as malicious, and should raise an alert. They might be a sign that the host machine has been compromised, or that the user of that machine has accessed a malicious site. The events collected by Arctic Security Node have three network indicators which can be used to determine network traffic: IP address, URL, and domain name. These indicators are used by the Suricata rule generator to create the Suricata rules.

All Suricata rules have to be numbered with a sid, Suricata identity number. The number range starts from 1, with the range from 1000000 to 1999999 reserved for custom sids. Other sid ranges before the custom sid range have been reserved for different vendors and threat intelligence feeds. The rule generator starts the numbering of rules from sid 1000000. This means that two sets of rules created

by the module start from the same number. To avoid errors with identical sids, new rulesets should be used to replace the old ruleset rather than using them simultaneously.

The rule generator can be given a list of threat intelligence feeds it uses to download events and turn them into Suricata rules. The Suricata rule generator removes any duplicate indicators of compromise it sees in the input data. Only unique URLs, domains, and IP addresses are used to generate rules. This saves space and memory, as there are no redundant Suricata rules in the output.

The Suricata rule generator can be integrated with Suricata by supplying the URL of the web API to the Suricata-update tool. This way Suricata will automatically download the new rules from Arctic Node. The rules are served as a simple text file, with one rule per line.

3.1.1. The structure of the generator

The Suricata rule generator is a new integration added to the integration module of Arctic Node. The generator consists of two parts: a worker thread which generates the Suricata rule list, and an API which serves the rule list. The worker thread creates the rules periodically, and saves them into a text file. The API reads and returns this file whenever it is called with a GET request.

The reason for creating rules periodically was improving the performance and speed of the submodule. The alternative for the periodic generation of rules would be to create the rules on demand. This would lower the constant computational demand at the cost of increased time required to fetch the new rules, as the process of creating rules can take a few minutes. In addition, generating the rules on demand could open a possibility to abuse the submodule by requesting rule creation multiple times.

The time period between generating new rules was chosen as 15 minutes. This is because the feed module of Arctic Hub and Node fetches most feeds on an hourly basis, and due to the short lived nature of technical threat intelligence it should be utilized in a timely manner. The processing of new events can take varying lengths of time depending on the exact size and nature of the feed. The process may also be delayed due to network or other external errors. Therefore, it is not possible to pinpoint a set time each hour when the new events are ready. As new events are coming in roughly on a hourly basis, it is sensible to utilize them within the hour. Creating rules from the most recent data several times an hour ensures that when the new events get processed, they are available for use as soon as possible. As a result, 15 minutes was chosen as the most efficient time period. This choice is not a final one, and the period can be adjusted to be longer or shorter.

The rule generation consists of the following steps:

1. Fetch the events of a feed from Arctic Node.
2. Create a bucket for the events.
3. For each event, extract indicators from it, and add them to the bucket.

4. Repeat this process for each feed.
5. When all feeds are done, create a new master bucket.
6. Add all the indicators from the feed buckets to the master bucket.
7. Create Suricata rules from the indicators in the master bucket.
8. Write the Suricata rules into a file on the disk.

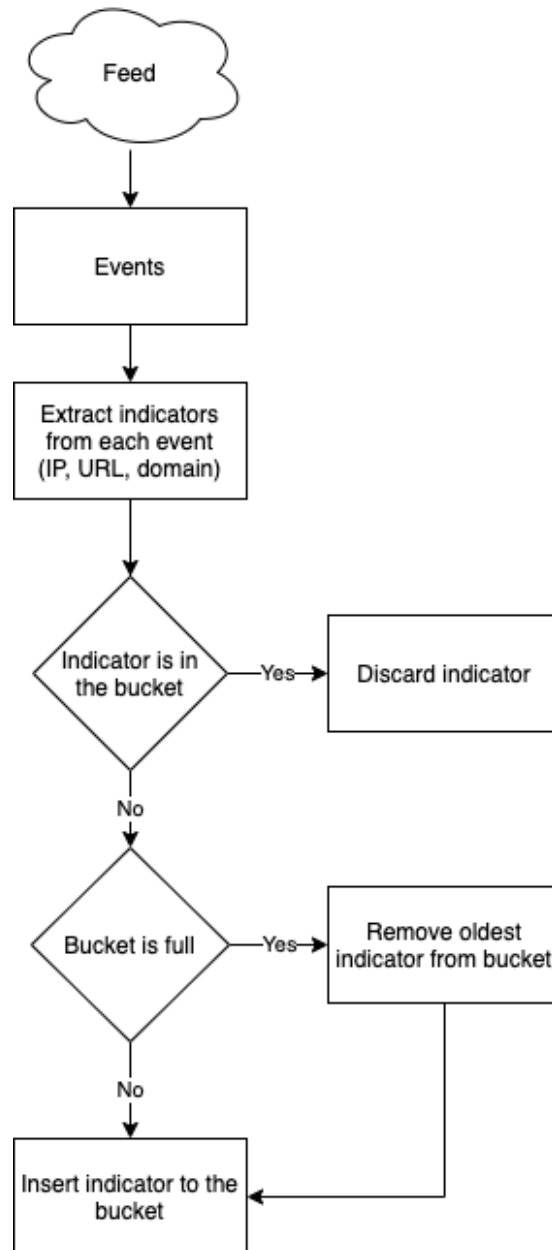


Figure 8. Steps 1-4 of rule generation. The submodule extracts indicators from each feed, and places them into feed specific buckets.

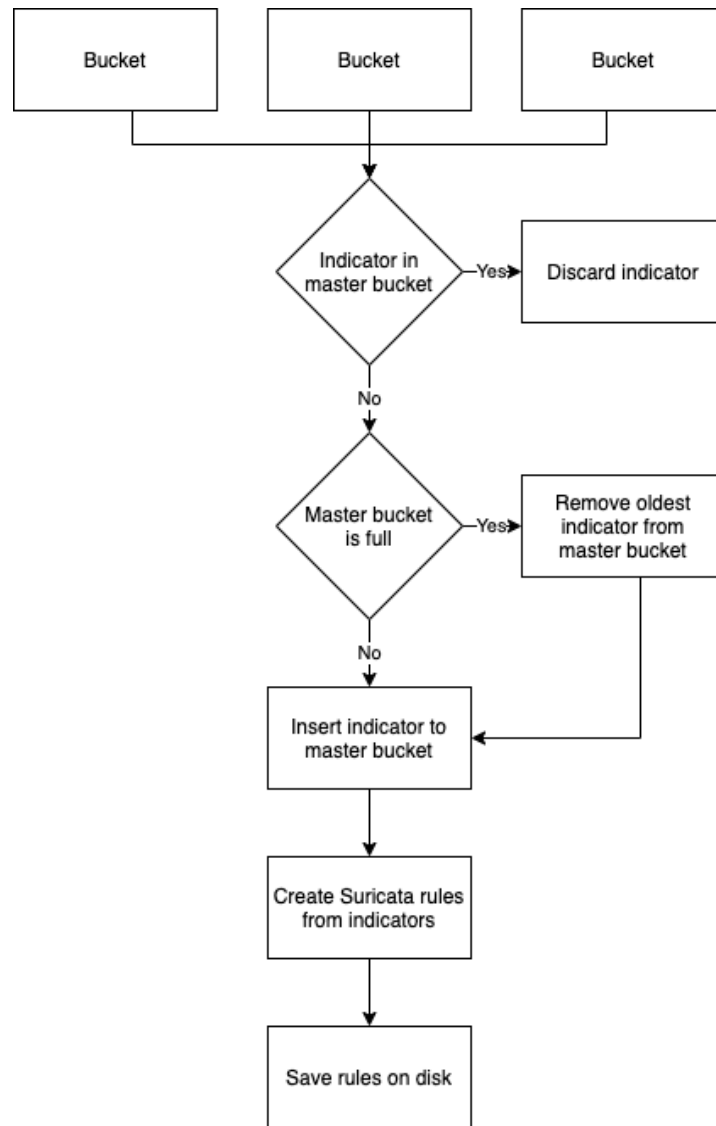


Figure 9. Steps 5-8 of rule generation. Buckets are combined into a master bucket, and the indicators are deduplicated. Finally, Suricata rules are generated and saved on disk.

A bucket is a set of unique values with a maximum size. They work on a first in, first out principle. This means that when a new value is added to a full bucket, the oldest value is discarded. The master bucket is set to have maximum size equal to the number of feeds times the maximum size of buckets. Therefore, it can hold all the values from the other buckets. The maximum size for buckets was chosen to be 100000, but the size is not final. If and when the Suricata generator is tested and used more extensively, the bucket size can be adjusted as needed.

The reason for using buckets with limited size is the fact that feeds constantly produce new events, and Arctic Node stores vast amounts of threat intelligence. There has to be a limiting factor for how many events are being used to create the rules. Otherwise, the rule generator would create more rules than Suricata can handle. One possible solution would be to limit the events to a specific time range, but even that would not be enough to ensure that the number of events

does not exceed Suricata's limits. This could be the case if the number of events in the given time range were to be larger than expected. The only way to ensure that the size of the ruleset does not exceed the capabilities of Suricata is to put a hard limit on the number of rules created.

Figure 8 depicts the first three steps of the process the submodule uses to turn feed data into Suricata rules. These three steps are performed separately for each feed. Once all events from each feed have been processed into their own buckets, a master bucket is created. The indicators from all of the other buckets are combined in it, as depicted in Figure 9.

3.1.2. IP rules

The creation of rules for detecting traffic to malicious IP addresses is quite straightforward. Alert should be raised whenever there is any traffic between the home network and any known malicious IP addresses. The rule works both ways, and will raise an alert for connections which originate either from the home network or from the IP address. Events often also include the ports used in malicious activity. These are opted out from the IP rules, as traffic from any port to the suspected IP address should be a reason for caution.

The rule generator submodule uses the following template to craft rules to monitor traffic based on the IP address:

- alert ip \$HOME_NET any <> **IP** any ('msg: "Malicious IP (**IP**)"; sid:1000000;)

Note that **IP** is a placeholder for the actual IP address.

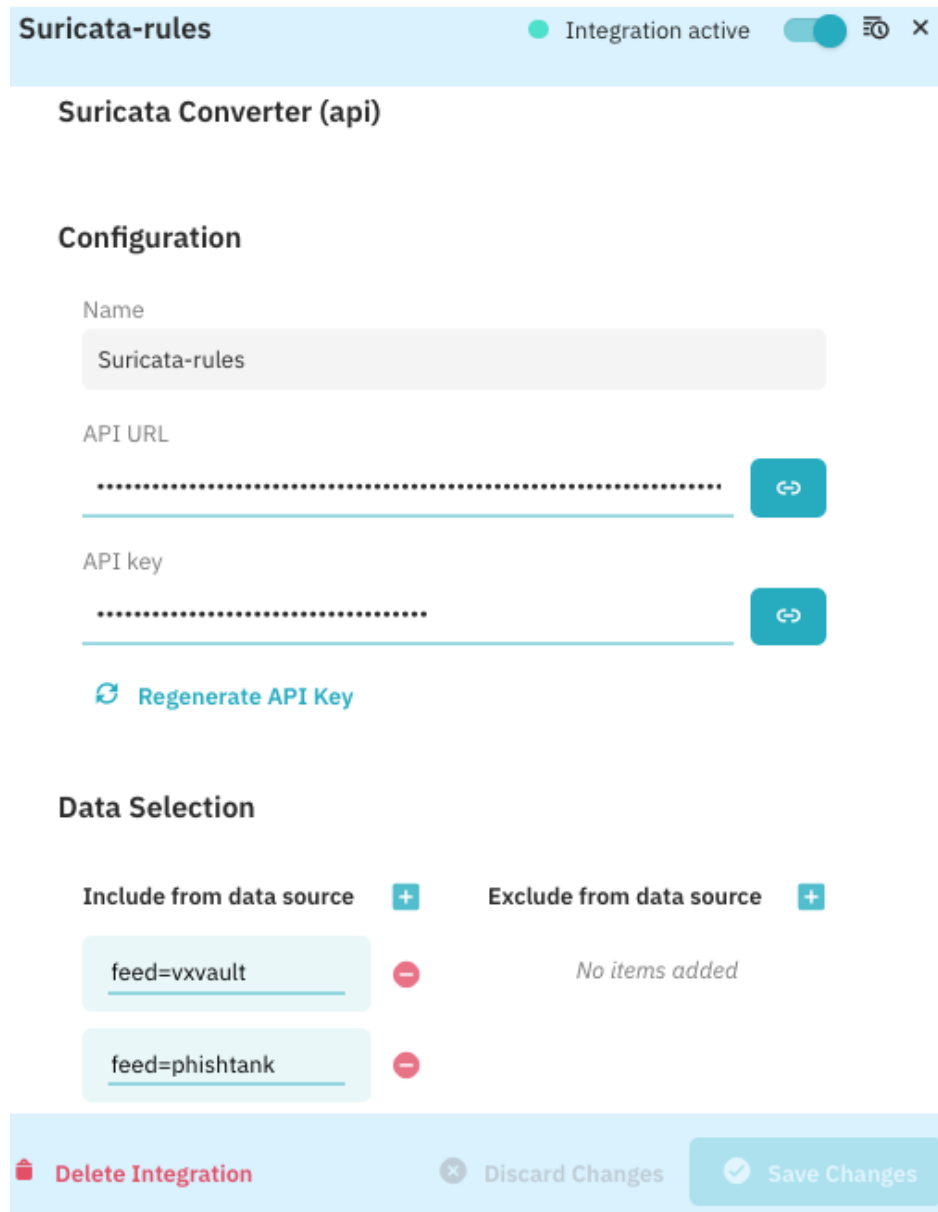
3.1.3. URL rules

URLs can be used to create two types of rules: HTTP rules and DNS rules. HTTP rules monitor HTTP traffic, while DNS rules monitor DNS requests. Both of them should be monitored separately, as DNS lookups can be done without HTTP requests. The generator extracts domains from URLs to create the DNS rules. If the URL contains a specific port, that port is left out from the rule creation. Any connections from any port to that URL are considered to be malicious, and the rules should catch them regardless of the port used.

The following templates are used by the rule generator submodule to craft the suitable rules to monitor DNS and HTTP traffic. Note that `example.net` is used as a placeholder for the real URL.

- alert http \$HOME_NET any -> \$EXTERNAL_NET any (msg: "Malicious URL"; content: **www.example.net**; http_header; content: "/evil/malware.exe"; http_uri; sid:1000000;)
- 'alert dns \$HOME_NET any -> \$EXTERNAL_NET any (msg: "Malicious DNS query"; dns_query; content: **www.example.net**; sid:1000000;)'

The HTTP rule matches both the domain and the uri of the URL. This is needed because malicious software might be uploaded and distributed from legitimate domains.



Suricata-rules Integration active

Suricata Converter (api)

Configuration

Name
Suricata-rules

API URL
.....

API key
.....

Regenerate API Key

Data Selection

Include from data source + Exclude from data source +

feed=vxvault - No items added

feed=phishtank -

Delete Integration Discard Changes Save Changes

Figure 10. The graphical user interface of Suricata rule generator. API URL and API key can be used to retrieve the data. Used data sources can be added via data selection.

3.1.4. User interface of the generator

Suricata generator is a new integration on the integration module of Arctic Node. Like any Arctic Node integration, it has a user interface which can be accessed via the frontend. The interface is shown in Figure 10. The frontend automatically

generates the graphical interface based on the code written for the interface in the backend. This is an existing feature of the frontend, and was not developed as part of this work. Multiple instances of the Suricata rule generator can be active at once.

The interface consists out of four parts:

- name,
- API URL,
- API key and
- sources.

Name is used to give the interface an identifiable name on the frontend. All interfaces are shown in a list, and the text written in the name field is displayed in that list. This can be used to distinguish multiple Suricata rule converters from each other.

API URL and **API key** are used to retrieve data from the integration. The text contained in these fields can be copied to clipboard by pressing the buttons next to the fields. The rules can be accessed via the API URL. API key must be added to the URL as a URL variable. The *regenerate API key* button underneath can be used to generate a new API key, which makes the old API key obsolete. Arctic Node and Suricata can be integrated together by giving the API URL to Suricata-update, which then downloads the rules from Node. Note that authentication and web access were not developed as part of this project.

The left column under *Data Selection* is used to add new feeds as **sources** for the integration. Feeds are selected by writing the correct name of the feed in the text field. Arctic Node users will be able to find the feeds available to them from other parts of the frontend. The text field accepts Arctic Security rule language filters. Events are fetched from the storage based on the filter. Sources can be removed from the selection by pressing the red button next to them.

Other parts of the interface include the slider in the upper left corner, which can be used to quickly disable and activate the interface. The status next to the slider indicates whether the integration is active or not. The cross next to the slider can be used to close the interface window. The right column under *Data Selection*, *Exclude from data sources*, is part of the default interface and is not utilized by this integration. It could be used later when the interface is developed further to add specialized rules to exclude parts of the input data used by the generator. All changes to the integration must be confirmed with the button in the lower right corner. The button next to it can be used to discard changes. The interface can be deleted with the button in the lower left corner.

4. CASE STUDIES

Tests were conducted to see how well the new Arctic Node submodule works compared with a similar functionality in another modern threat intelligence sharing platform, in this case the MISP. The tests measured how well the deduplication feature of Arctic Node works with real data, as well as how successfully the two different threat intelligence sharing platforms turn indicators of compromise from the same threat intelligence feeds into working Suricata rules. For the test setup a virtual environment was created in order to simulate real malicious traffic between two computers, and a Suricata instance was set up to monitor the traffic between these machines.

Intrusion detection and prevention systems are only as useful as their capability to detect and prevent intrusions. One goal of the test setup is to observe how successfully the rule sets generated by MISP and Arctic Node detect possible intrusion attempts. Testing the detection capabilities of intrusion detection and prevention systems against real threats is complicated. These tests try to approach reality from the base assumption that indicators from threat intelligence feeds are dangerous, and traffic to and from those indicators should raise an alert. The detection capability is measured by how much of the malicious traffic from the original input data is stopped or detected, and how much of it goes by undetected.

One of the features of the Suricata rule generator is the capability to deduplicate the indicators of compromise. In theory, it is easy to prove how the deduplication function works by creating a set of repeated indicators, and using that set to create the Suricata rules. However, while such a test would prove that the deduplication works, it would not provide information on how useful its presence is. By testing the deduplication function against real indicators of compromise produced by real threat intelligence feeds it is possible to see how useful deduplication could be in actual use.

4.1. The test setup

The test setup is as follows: Arctic Node and MISP are used to fetch the events from same threat intelligence feeds, and the data is output as Suricata rules. This generates two sets of rules: MISP Suricata rules and Arctic Node Suricata rules. In addition, indicators of compromise from the same feeds are manually downloaded and used to craft a test data set which can be used to simulate malicious traffic. The manually downloaded data set is used to create fake traffic in a virtual network between two machines. One of the machines is running an instance of Suricata loaded with different Suricata rulesets generated with the Arctic Node and MISP. The performance of these rulesets is then measured and compared.

The test was conducted using real technical threat intelligence provided by seven different threat intelligence feeds. Real sources were used because the goal of the test was to see how well the new submodule works with real data. Accessing real, possibly malicious sites is dangerous. The test machine could get infected with malware, the malicious actors could be alerted to the fact someone has tried

to access the site, or someone monitoring access to that site could take a note of the attempted connection. Extreme caution was exercised to make sure that no such connection was ever made over the internet. For this reason, the test traffic was contained in a virtual network between two virtual machines. The virtual machines and network were set up in such a way that all test traffic would seem real for the Suricata instance used in the tests. Therefore, Suricata behaved the same way during these tests with the test traffic as it would behave in real life with real traffic.

The test setup is presented in Figure 12. Two Ubuntu virtual machines were used in the test, virtual machine 1 and virtual machine 2. These machines were connected via a virtual network, and both of the machines had another separate virtual connection to the host machine. The host machine was MacBook Pro 2015 with 3,1 GHz Intel core i7 with 16GB DDR3 memory. Both Ubuntu virtual machines were given base memory of 8062MB and one processor core. The test machines were first connected to the internet to download all the necessary libraries and programs needed for the tests, after which they were separated from the internet. The rest of the files were transferred to the virtual machines over their connection to the host machine. For an additional layer of safety, the host machine was also cut off from internet during the tests to make it impossible for any connections to be established over the internet to any malicious sites by accident.

Virtual machine 1 is the simulation of the victim’s machine, which tries to establish a connection to a known malicious host. The machine also has a Suricata instance running on it to monitor potentially malicious traffic. The main configurations of the Suricata are listed in Table 3. During the tests the instance is loaded with the two different sets of rules, the set of rules generated with MISP and the set generated by the Node submodule. These two sets will be referred to as the MISP rule set and Node rule set.

Table 3. Suricata configurations on machine 1. The IP addresses are the IPs of machine 1

Configuration	Value
\$HOME_NET	[192.168.1.1, 2001:db8:bad:a55::1]
\$EXTERNAL_NET	!\$HOME_NET

Virtual machine 1 was configured to send IP packages via virtual machine 2. A sequence diagram of sending a GET request to IP address 192.0.2.10 can be seen in Figure 11. NGINX on virtual machine 2 was configured to send all IP packages to a dummy server running on the same machine, and a netfilter rule on virtual machine 2 redirected all traffic back to the virtual network between the machines. The end result of this was that it was possible to connect to any IP address from virtual machine 1, but all connections went to the dummy server on virtual machine 2. From the point of view of virtual machine 1 the packages were sent to the original IP address, and responses were sent from the same IP address. This way the Suricata instance running on virtual machine 1 could work with real data and rules generated from that data.

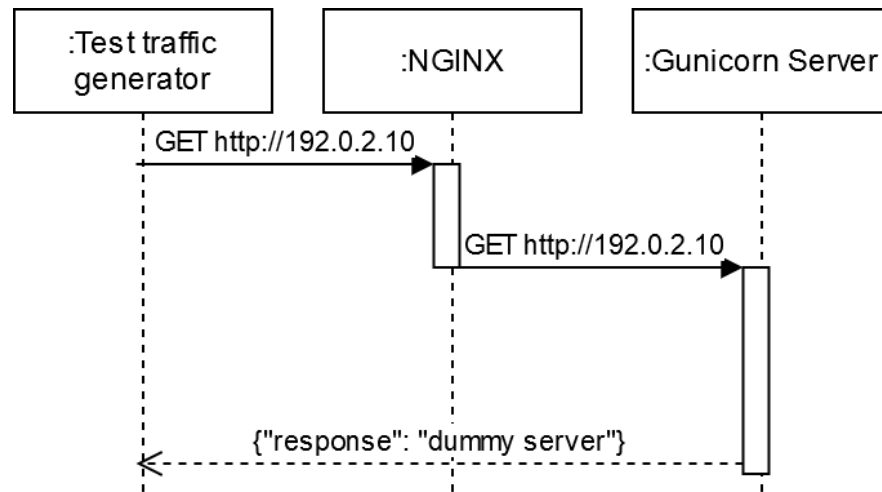


Figure 11. A sequence diagram of sending a GET request from virtual machine 1 to a IP address during test. The IP package is rerouted to virtual machine 2. NGINX reroutes the package to the dummy server.

Malicious traffic was simulated by trying to connect to known malicious URLs and IP addresses with machine 1. The connection was redirected to machine 2 with a dummy server running on it. Malicious connections were simulated by sending a simple GET request to the URL or IP address. This simulates cases where a malware on a host machine tries to connect back to the command and control server or when a user opens a known phishing link. This test does not simulate every possible way of creating malicious traffic, but the traffic generated by these tests should be enough to test the different sets of rules and to see how they perform in use.

The test connections were done by a **test traffic generator program**. The program was written in Python 3 and could be given commands as command line inputs. It could be instructed to run test connections, where it tried to connect to the dummy server using all the different connection methods used during the test. To generate the test traffic the program was given a file which listed all the URLs and IP addresses. It then tried to connect to each of the indicators of compromise one by one. The program expected to get the default response from the dummy server on virtual machine 2. If the connection failed or the response was different from the one expected, the connection was marked as a failure. During the test the generator printed the current progress so the length of the test could be estimated. If connection could not be established, the generator wrote the error into output. The full output during the tests was saved on disk.

As the input data from feeds used by the test traffic generator was somewhat lacking, it tried several different fixes during testing if connection could not be established. The program tried to add missing schemas and top-level domains to URLs. The missing schema was fixed to HTTP, and the missing top-level domain was fixed to use .net. For example, *example* would become *http://example.net*. For IP addresses, top level domain was not added as it is not needed. Some of the indicators had port 443 as part of their URL, e.g. *example.net:443*. In these cases, the the schema was changed to HTTPS, e.g. *https://example.net:443*.

The test traffic generator program had two modes: normal mode, which runs the test and generates the traffic, and test mode, which was used to make sure that the connection between the virtual machines worked before conducting the actual tests. The test mode tested the connection with six different URLs, which cover the range of possible indicators of compromise: connection to an IPv4 address over HTTP and HTTPS, connection to an IPv6 address over HTTP and HTTPS, and connection to a domain over HTTP and HTTPS. These tests were required, as often one of these test cases did not work, which required restarting and resetting different parts of the test setup.

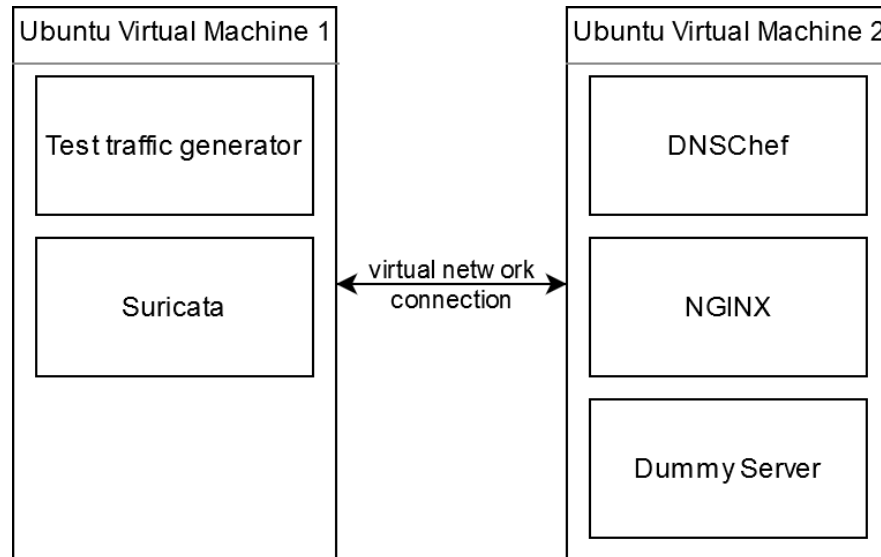


Figure 12. Test machine setup. Two virtual machines were used in the test. Machine 1 generated the malicious traffic, which was to be detected by the Suricata instance. Machine 2 worked as the DNS server and endpoint for the malicious traffic.

Virtual machine 2 had two different tasks: to work as the DNS server and to host the dummy server to respond to the requests of virtual machine 1. The DNS server was set up with DNSChef v 0.4 [37]. The server was set up to always respond with the IPv4 or IPv6 address of virtual machine 2. This way, all requests to malicious URLs would be directed to the dummy server running on virtual machine 2. A sequence diagram for connecting to a malicious file can be seen in Figure 13. First, virtual machine 1 asks the DNS server for the IP address of *example.net*. The DNSChef responds with the IP address of virtual machine 2. Virtual machine 1 then sends the GET request to the same IP address, which is rerouted to the dummy server via the NGINX.

The dummy server was written in python 3 using the Falcon and Gunicorn libraries. It was configured to respond to all GET requests to any URI located on the domain with a static JSON message *'response': 'dummy server'*. The test traffic generator program checked the responses for this message to make sure the connection worked properly. This is enough to fulfill the minimum requirements for establishing a connection to a server, which will trigger the Suricata rules monitoring for traffic between two machines. The server was able to respond to

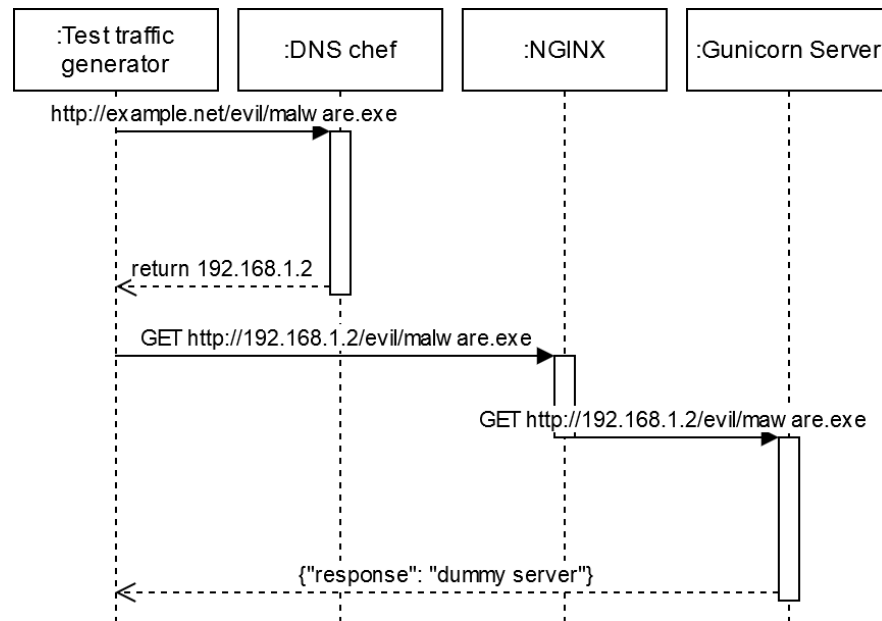


Figure 13. A sequence diagram of sending a GET request to malicious URL during test. First, the virtual machine 1 asks the DNS server on virtual machine 2 for the IP address of the domain. The DNS server responds with the IP address of virtual machine 2. The virtual machine 1 then send a GET request to this address.

both IPv4 and IPv6 traffic. NGINX was configured to listen to port 80 for HTTP traffic, and to port 443 for HTTPS traffic. Some of the data in the test used other ports than these two. There was an attempt to configure NGINX to also listen to all the other ports, but this did not work. There was a possibility to listen to a subset of these extra ports, but as the amount of indicators which tried to connect to these extra ports was relatively low in the test data, it was decided to leave their support out of the test.

Separate virtual machines for the DNS server and the dummy server were considered, but limiting the number of virtual machines was seen as more valuable. Configuring each machine took time, and the complexity of the test setup increased with the number of machines involved. In addition, there was no reason to expect problems in combining the DNS server machine with the dummy server machine, as only IP packages needed to be rerouted, and DNS requests could be sent alongside with the IP packages.

4.2. The metrics for comparing different rule sets

Two types of metrics were used to compare the rules. The first is the size of the ruleset. Simply put, this measures the size of the ruleset generated from the same input data. Size can be used to evaluate how well the deduplication functionality of the Arctic Node rule generator works by comparing the difference in size with the ruleset generated by MISP. The distribution of different Suricata rule types, or protocols, is also measured and analyzed.

The second metric used is the amount of unique true positive Suricata alerts generated during the test. A positive Suricata alert is an alert which was triggered by a connection to a malicious site during the test. Due to the anatomy of a connection, it is possible for a connection to trigger multiple alerts. The first type of alert is querying a DNS server for the IP of a domain. This can trigger rules which monitor DNS traffic. After that, it is possible to connect to the server. If the indicator was an IP address, the DNS query is skipped, and the server is connected to immediately. In the test setup, sending the single HTTP GET request can trigger two IP alerts, one for sending the request and another for receiving the answer. This GET request can also trigger the HTTP request.

The second metric counts each type of alert only once, which is why the metric is called unique. The uniqueness of the rules is needed to prevent abuse of the metric. For example, a ruleset of a thousand identical rules would be triggered a thousand times by a single matching connection. These thousand identical alerts would contain the same information as a single alert. Thus, only unique rules should be used.

4.3. MISP virtual machine

The Arctic Node Suricata rule generator was tested against a similar functionality in MISP. The official MISP virtual machine image v.2.4.108 was used in these tests. It was used to fetch the feeds and to export the fetched data as Suricata rules. The virtual machine was deployed on the same computer used in testing.

It was noted that fetching feeds with large quantities of data, e.g. Ransomware tracker, was slow with the MISP virtual machine. Generating Suricata rules from a large number of indicators was also slow. Both of these processes could take approximately an hour to complete. The performance issues are likely explained by the limited resources given to the MISP virtual machine.

4.4. Feeds

The seven feeds listed in Table 4 were chosen for the test. All of the feeds are supported by both MISP and Arctic Node. All of the feeds are also free to use and in open distribution. The feeds can be divided into three groups based on the type of information they provide: C2 servers, phishing sites, and malware links. All of the feeds provide network indicators, which can be used by both Arctic Node and MISP to create Suricata rules.

Feodo Tracker is another project by abuse.ch. It was launched in 2010 to track the C2 servers and hashes of a trojan called Feodo [38]. The project has continued to track the new generations of the malware family such as Cridex, Dridex, Geodo, Heodo and Emotet. Out of these iterations of Feodo malware family, as of 2019, Heodo and Dridex are the only members of Feodo malware family which are still in active use. Feodo Tracker also tracks a trojan called TrickBot, which Emotet deploys on the target machine. The feed reports the IP address of the C2 server, destination port, timestamp, ASN, malware, and

Table 4. Feeds chosen for the test and what indicators of compromise they provide. Type indicates what the information is related to.

Feed name	Type	Network indicator	Malware indicator
Bambenek C2 master list	C2	Domain name, IP, nameserver	Malware description
Bambenek C2 Domain master list	C2	domain name	Malware description
Bambenek C2 IP master list	C2	IP	Malware description
Feodo Tracker	C2	IP	Malware name
Phishtank	Phishing	URL, description, target	-
Ransomware tracker	C2, infra	URL, domain, IP	-
VxVault	Malware	URL, domain	Not provided on the text version.

the activity of the server. The feed can also be downloaded as Snort or Suricata rules. Feodo tracker provides different types of lists. MISP has support for the Feodo Tracker ip blocklist, which has been chosen as the Feodo Tracker feed for this test.

Ransomware tracker is another project by abuse.ch, launched in 2016 to track various ransomware families [39]. The project tracks the C2 servers, distribution sites and payment sites of the following malware families: TeslaCrypt, CryptoWall, TorrentLocker, PadCrypt, Locky, CTB-Locker, FAKBEN and PayCrypt. Ransomware tracker provides network indicators like the IP address, domain name, and URL of C2 servers as well as other infrastructure such as distribution sites of the ransomware.

Bambenek Consulting is a company which deals with cybersecurity investigations [8]. They provide various threat intelligence feeds for free, one for each malware family they track, alongside collective lists of all malware families. They also provide high-confidence lists, which are a subset of the collective lists. These high-confidence lists contain only the indicators that they are confident are not false positives. Bambenek Consulting generates most of their IPs and domain names by using the known domain generation algorithms (DGA) of various malware families. They provide both a full list of DGAs and a much shorter list of IPs and URLs of active C2 servers. These malware families use generators to create a list of domain names they try to access every day. When a hacker wants to send commands to machines with this malware, they register one of these domains for a day and use it to send commands to their botnet.

Arctic Node has support for the C2 all indicators feed. MISP has support for the high-confidence C2 IP feed, the high-confidence domain feed and the high-confidence DGA domain feed. Therefore, it was not possible to choose the exact same feed from Bambenek Consulting for both MISP and Arctic Node. However, the high-confidence C2 IP and domain feeds are subsets of the C2 all

indicators feed. These two feeds can be combined together to approximate the C2 all indicator feed. Therefore, the high-confidence C2 IP feed and domain feed were chosen from MISP as sources for Bambenek Consulting. A full list of the Bambenek feeds and their support by MISP and Node can be seen in Table 5. Note that the size refers to the size of a feed at one point in time. The exact numbers fluctuate over time, but they can be used as an estimate of the scale. Size refers to the lines of text in each feed, and not to the amount of indicators of compromise. Feeds which have been separated by the indicator of compromise, such as C2 IP and C2 domain, contain one indicator of compromise per line. The C2 all indicator feed and High-Confidence C2 All Indicator feed list both the IP address and URL or domain of the C2 server, but not for all lines. For better comparison with other feeds, their estimated sizes should be doubled.

Table 5. Feeds provided by the Bambenek Consulting, their sizes and support by MISP and Hub

Feed name	size, lines of text	Node	MISP
C2 IP	585	-	-
C2 Domain	718	-	-
C2 All Indicator	719	Yes	-
DGA Domain	835190	-	-
High-Confidence C2 IP	185	-	-
High-Confidence C2 Domain	709	-	Yes
High-Confidence C2 All Indicator	339	-	Yes
High-Confidence DGA Domain	356454	-	Yes

VxVault is a public feed for malware URLs and hashes. The site has been up from at least 2010, and the oldest samples have been uploaded in 2006. The site is hosted by an active member of the computer security domain, but beyond this not much information is available on the feed provider. Each event on the site contains a malware URL, IP address of the host, and detailed information on the malware. Each event contains the malware name, filename, MD5 hash, SHA-1 hash, SHA-256 hash, and size. The feed can be accessed by two different ways. The first is a PHP based website where all events ever posted by VxVault are listed. From this site, it is possible to access the full information on each event. The second, more machine friendly feed is a plain text file of the one hundred most recent URLs posted by VxVault. This list contains only the malware URLs and no other information. Both MISP and Arctic Node use the URL list, likely because it is easier to fetch than the more complex full information feed which seems to be designed for humans rather than machines.

Phishtank is a project launched in 2006 by OpenDNS with the aim of tracking of phishing sites and emails. The collection of phishing data is crowdsourced. Users can submit phishing attempts to Phishtank and verify submissions by other users [9]. Phishtank collects information about the phishing site and takes a screenshot of the site if possible. Phishtank provides their data for free, but without free registration downloading is limited to a few times per day. Events can be downloaded in four different formats: XML, JSON, PHP and CSV. The

test data was downloaded as CSV as it is the format fetched by MISP. Each event contains the phishing URL, link for additional details, timestamp, a value which tells whether the site is still online, and the target which the phishing site imitates.

Another abuse.ch feed called **URLhaus** was planned to be the eighth feed in the test, as it is supported by both MISP and Arctic Node. However, the large size of the feed turned out to be a problem. Fetching and processing the feed with the MISP virtual machine was slow, and it consequently slowed down tests and analysis considerably. The feed provided over 200,000 events, which is an order of magnitude more than the rest of the feeds combined. Therefore, it was removed from the list of feeds. All tests and analysis have been done without URLhaus. The size of URLhaus was enough to exceed the size of a bucket in the Arctic Node rule generator, which caused roughly half of the indicators from URLhaus to be dropped.

4.5. Input Data

The test used real events from seven open source feeds which are supported as input data by both the MISP and Arctic Hub. Feeds provide events as a continuous stream. Data used in the test was a snapshot of the different feeds collected at the same time. This snapshot of events was downloaded three times by different methods: once by hand, once with the MISP virtual machine and once with Arctic Node. To ensure that all of the different methods captured the same snapshot of data, all of them were collected within a time period of a couple of hours. The slight variance in the timing of data collection might have caused some differences between these three sets. However, this variance should be negligible. Any events in the MISP and Arctic Node data sets which are not in the manually collected set do not increase the performance metrics of either, as the traffic is only generated from the manually collected set. For example, if one of the data sets were to contain events from two years ago, these extra events would not be likely to increase performance in these tests, as the indicators of compromise used to create the test traffic are more recent.

Feeds used in the test and the amount of indicators obtained from them with the three different collection methods are listed in Table 6. The difference between manual collection and downloading the events with MISP seems to be the smallest across the different feeds. Event count collected by Arctic Node differed more from the other two methods.

Arctic Node downloads new events from feeds every hour. The events for Node were collected by setting the Node instance to download the feed chosen for the test. A filter was applied to select events at the same time as MISP and manual collection. This filter had to be written temporarily into the code of the Suricata rule generator, as the version of the generator which was used for the test did not have support for such filters. This filter was used to ensure the Arctic Node's event set matched the other two sets as closely as possible.

The events downloaded by hand were used to create the **test traffic data set**. This data set was used as the input for the test traffic generator to create the

simulated traffic used in testing. The test traffic data set was further cleaned by extracting two different indicators from each event: IP addresses and URLs. Duplicate IPs and URLs were cleared from the data set. This cleaning and indicator extraction was done by writing a short parse script in Python for each of the feeds, as the data format was different for each feed. The final size of this cleaned data set turned out to be 31353. Table 6 lists the breakdown on how many unique indicators were produced from each of the feeds. Some of the indicators did not work as expected during the tests. The test traffic generator program had a limited capability to fix these cases, as described in an earlier chapter.

Table 6. Feeds chosen for the data set and the amount of events in each of them, separated by different collection methods.

Feed name	Fetches manually	MISP	Node
Bambenek C2 master list	1269	-	4592
Bambenek C2 Domain master list	757	755	-
Feodo Tracker	333	329	692
Bambenek C2 IP master list	182	181	-
Phishtank	11193	11364	12257
Ransomware tracker	18517	27935	14822
VxVault	94	94	492

The test traffic data set was analyzed to determine how much overlap there was between the feeds. This overlap analysis was done on the discarded duplicate IP addresses and URLs. Results of this analysis are shown in Table 7. Feeds with no overlap have been left out of the results. Most overlap was between the different Bambenek lists. This was to be expected, as the Bambenek C2 Domain master list and the C2 IP master list are subsets of the Bambenek C2 master list. Otherwise, the overlap seems quite small between the feeds. MISP reports similar overlap between their default feeds [40], with most feeds having no overlap between them.

Table 7. The number of overlapping indicators between the different feeds.

Feed name	Feed name	Overlap
Bambenek C2 master list	Bambenek C2 Domain master list	755
Bambenek C2 master list	Bambenek C2 IP master list	181
Bambenek C2 master list	Ransomware Tracker	56
Bambenek C2 IP master list	Ransomware Tracker	13

4.6. Test Cases

A total of four tests were conducted to evaluate the performance of MISP and Arctic Node. Both of them were tested twice. On the first round of tests (i.e. cases 1 and 2), each of them were used to generate a rule set for each feed separately. These rule sets were then tested one by one against the test data set. For the

second round of tests (i.e. cases 3 and 4), all of the feeds were given together as the input to create one set of rules. These rules were then tested against the test traffic data set.

Each rule set was tested under the same conditions. Test traffic data set was used to create the traffic, and the rule set was given to the Suricata instance as input. The results of these tests were also analysed under the same conditions, by measuring what indicators the rule set detected from the test traffic set. The tests are numbered as follows:

1. Rules from a single feed by Arctic Node.
2. Rules from a single feed by MISP.
3. Rules from multiple feeds by Node.
4. Rules from multiple feeds by MISP.

In each test case, the test traffic data set was given as input to the test traffic generator on virtual machine 1. The traffic over the virtual network was the same in each test case. As the test traffic data set contained the indicators used to generate the tested Suricata rules, each of the Suricata rules in the tests should be triggered at least once.

4.7. Test result

In each test, Suricata generated a log file of the alerts raised by the tested rules. These logs were analysed to find out how well the rules detected the test traffic. Any alert caused by an indicator was considered a positive hit. The total number of hits was calculated by going through all the test indicators and checking for matching log lines. If a match was found, the indicator was marked as a *hit*, as this indicated some part of the connection was detected by Suricata and caused an alert. Each log line which detected a hit was then discarded, as each line could only refer to one indicator. The logic for matching log lines to indicators was written separately for both MISP and Node, as both of them had different forms of Suricata logs.

The test results for case 1 are listed in Table 8. In this case, Arctic Node was used to create Suricata rules for each of the feeds used in testing, after which the rulesets were tested against the test traffic data set. The hits column has been calculated as the fraction of indicators detected from the total set of possible indicators of the test traffic set. Summing up these values in cases 1 and 2 might result in total value of over 100%. This is due to overlap between feeds, which means that in some cases rules from one feed might match indicators from another feed.

The results for test case 2 are listed in Table 9. In this case, MISP was used to create rules from the events of each feed, after which the rules were tested against the test traffic data set. MISP created rules on a 1:1 ratio from events of Bambenek C2 IPs, VXvault and Feodo C&C feeds. This comes as no surprise, as

Table 8. The results of test case 1, rules from single feeds by Arctic Node.

Rule set	Events	Rule count	Hit count	Hits
Bambenek C2 master list	4592	1292	1268	4.15%
Feodo C&C	692	341	333	1.09%
Phishtank	12257	18217	11037	36.10%
Ransomware Tracker	14822	28499	18507	60.54%
VXVault	492	239	97	0.32%

each of these feeds list only IP addresses in their events. Bambenek C2 domains created rules on a 1:2 ratio. Each of the events for this feed list a domain name of a C2 server. These are turned into two rules by MISP, one to monitor DNS requests for said domains, and one to monitor HTTP traffic to that domain. Unlike other feeds, MISP generated fewer rules per event for Phishtank. Phishtank events contain phishing links, which MISP turns into HTTP rules. It does not create DNS rules from these links, and does not create any rules from phishing links which use HTTPS instead of HTTP. This explains why MISP created less rules from Phishtank events. Ransomware Tracker events contained both URLs and IP addresses as the indicators, and thus MISP created DNS, HTTP and IP alerts from these events. As not all of the events contain all these indicators, this ultimately created rules on a 1:1.35 ratio.

Table 9. The results of test case 2, rules from single feeds by MISP.

Rule set	Events	Rule count	Hit count	Hits
Bambenek C2 IPs	181	181	177	0.59%
Bambenek C2 Domains	755	1510	751	2.46%
Bambenek, combined	936	1691	928	3.05%
Feodo C&C	329	329	341	1.08%
Phishtank	11364	7397	6648	21.75%
Ransomware Tracker	27935	37618	18136	59.32%
VXVault	94	94	85	0.28%

When comparing the results of cases 1 and 2, it can be seen that in each case the Arctic Node Suricata rule generator either reached the same performance level as the Suricata rule generation of MISP or outperformed it.

Table 10. Distribution of different Suricata rule types in test case 1.

Rule set	IP rules	DNS rules	HTTP rules
Bambenek C2 master list	524	768	0
Feodo C&C	341	0	0
Phishtank	4573	6278	7366
Ransomware Tracker	4946	9735	13818
VXVault	79	56	104
Total	21.5%	34.7%	43.8%

Table 11. Distribution of different Suricata rules in test case 2.

Rule set	IP rules	DNS rules	HTTP rules	TCP rules
Bambenek C2 IPs	181	0	0	0
Bambenek C2 Domains	0	755	755	0
Bambenek, combined	181	755	755	0
Feodo C&C	329	0	0	0
Phishtank	0	0	7395	2
Ransomware Tracker	4344	9728	23546	0
VXVault	0	0	94	0
Total	10.3%	22.2%	67.5%	<0.1%

The distribution of different protocols per feed for case 1 is shown in Table 10, and the distribution for case 2 is shown in Table 11. Three main types of protocols were present in the rule sets, IP, HTTP, and DNS. MISP also generated two rules with TCP protocol from the Phishtank feed. The rules generated by Arctic Node have a more even distribution of different protocols than rules generated by MISP. For both rulesets, HTTP protocol is used the most.

In case 1, Arctic Node had collected 390% more Bambenek events than MISP in case 2. This size difference was switched the other way around in the number of rules generated. Arctic Node compressed the events down to 1292 rules, 399 rules fewer than the combined MISP Bambenek feeds. The protocol distribution reveals the difference between these two rule sets. Arctic Node had created 190% more IP rules than MISP, but did not create HTTP rules from the domains listed in Bambenek C2 master list. MISP, however, created both DNS and HTTP rules from the C2 domains, which likely explains the difference in the size of the rulesets. The events collected by Arctic Node likely contained a large quantity of duplicate indicators, which were then compressed away by the deduplication process of the Suricata rule generator. The Arctic Node ruleset performed better than the MISP ruleset. The Bambenek feed in case 1 had 340 more hits, or 37.0%, than the combined Bambenek feeds of case 2.

MISP and Arctic Node created approximately the same number of rules for the Feodo C&C feed, and had similar performance. While Arctic node created 12 rules more than MISP, it had 8 hits fewer than MISP. Arctic Node did have more than twice the number of events, but it appears that around half of them were duplicates that were dropped by the deduplication process. In both cases, the protocol for all rules was IP. This is the expected result from this particular feed, as it only reports IP addresses.

Comparing the results of for Phishtank feed, Arctic Node had 7.9% more events than MISP, but created 146.3% more Suricata rules. The Arctic Node Phishtank ruleset had 66.0% more hits in the test than the MISP Phishtank ruleset. It appears that an increase in the size of the ruleset does not reflect a similar increase in the relative test performance. This difference might relate to the fact that MISP could not handle phishing links which used HTTPS, while Arctic Node does handle these links and has the ability to create rules from them. Furthermore,

MISP only created rules with the HTTP protocol, while Arctic Node created an even distribution of different kinds of rules.

The results for Ransomware Tracker did not differ dramatically between cases 1 and 2. The difference in hit count was minimal, with case 2 having only 371 fewer hits than case 1. However, a large difference can be seen in the number of events and rules. Case 2 had 88% more events and 32% more rules than case 1.

Arctic Node collected more than five times the number of events for VXVault than MISP. This difference was due to the fact that Arctic Node had been collecting VXVault events constantly, and the events used for this study were taken from Node by querying a slice of them on the date of test data collection. Only half as many rules were created from the events collected by Arctic Node. The ruleset in case 1 was 154% larger than in case 2, or 145 rules. This size difference can be attributed to the lack of IP and DNS rules in case 2. Case 1 had only 10 more HTTP rules than case 2, or 10.6%. The performance of VXVault was similar in both cases. Case 1 had 12 more hits than case 2, or 14.1%.

In test case 3, Arctic Node was used to create Suricata rules from the combination of all test feeds. Results of this are shown in Table 12 on the Node combined line. The total amount of events in this case was 32792, which is 63 events more than the combined total of all of the events in case 1. These events were turned into 48359 rules, which is 229 rules fewer than all the rules combined in case 1. This means that the compression rate was minimal, around 0.47%, without taking into account the difference in total number of events. This shows how little overlap there is between the tested feeds, resulting in the low compression rate.

In test case 4, MISP was used to create Suricata rules from all the test feeds together. Results of this are shown in Table 12 on the second line. There is no difference between the total number of rules and events of test case 2 and the number of rules and events of test case 4. From this result it is evident that MISP does not conduct any sort of deduplication between feeds, as was expected.

Table 12. Results of test cases 3 and 4, rules from multiple feeds by Node and MISP.

Rule set	Events	Rule count	Hit count	Hits
Node combined	32792	48359	30515	99.8%
MISP combined	40658	47129	26111	85.4%

Table 13. Rule distribution for cases 3 and 4.

Rule set	IP rules	DNS rules	HTTP rules	TCP Rules
Node combined	10240	16831	21288	0
MISP combined	4854	10483	31790	2
Node Total	21.2%	34.8%	44.0%	0%
MISP Total	10.3%	22.2%	67.5%	<0.1%

When comparing the results of cases 3 and 4, it can be seen that Node had 7866 events less than MISP, or 19.3% fewer events. Despite this, Node created a total of 1230 rules more than MISP. This is 2.4% more rules with 80.3% of the events, or 1:1.47 ratio for Node and 1:1.59 ratio for MISP. When tested, the Node ruleset had 4404, or 16.9% more hits than MISP ruleset. This would indicate that the ruleset created by Arctic Node performed better than the one created by MISP.

Table 13 shows the distribution of different protocols in cases 3 and 4. The total distribution of different protocols stayed the same between cases 2 and 4. This is to be expected, as the rule set of case 4 is just the sum of rule sets of case 2. There is a slight difference in the distribution of different protocols between cases 1 and 3, but the difference is not great. All the changes are less than one percentage unit.

From these tests it appears the Arctic Node Suricata rule generator generates more rules per event than MISP does. Rules created by Arctic Node seemed to work on the same level or better than the rules created by Node. The deduplication feature of Arctic Node did not seem to offer much benefit between the different feeds, but it did work well in regards of compressing the large number of events into a smaller number of rules. The deduplication process appears to be especially good for Arctic Node, as the way Arctic Node generates events seems to result in a large number of events having duplicate indicators between them.

In test case 4, the ruleset raised 8 errors when loaded into Suricata. These errors were narrowed down to originate from the Phishtank rules. These same errors were present in test case 1 when testing the Phishtank rules. No errors were raised in any other test case.

The test traffic data set contained 3262 indicators with set ports which were different from those listened to by the NGINX. There was an attempt to configure the NGINX to listen to these ports, but the large number of different ports turned out to be difficult to handle and the configuration did not work. As these connections could not be resolved, they were dropped from the final results.

One challenge faced during this analysis was the fact that the order of the logged alerts did not always match the order in which the indicators were tested. This is due to the multithreaded nature of Suricata. However, it can be assumed that all of the alerts have been produced by the test, and the connections used to run it. Therefore, if an indicator does match one of the lines, it can be counted to have been detected by Suricata.

4.8. Future Work

The current version of the Suricata rule generator is still in need of improvement. After the tests had been run, an error was found in the logic that creates Suricata rules from the events. Some events that only had domain names did not list them under the *url* key but rather under the *domain name* key. The old logic used values under the URL key, but not under the domain name key. As a result, a couple hundred events were not used to create either domain or URL rules even though they contained the necessary information. This error does not invalidate the test, as it only effected the rule set produced by the generator and not the

test setup. Tests were not repeated after the discovery and fixing of this error, as the results would have only changed slightly and there was no time for running additional tests. If the tests had been repeated, the Suricata rule generator would have created a couple hundred more rules, which could have resulted in slightly more hits. In other words, the rule generator was slightly improved after the tests, and the results of the tests are applicable to the previous iteration of the generator. Repeating the tests with the more recent version would produce results which differ slightly from the results presented in this work.

It was observed that the feed data set used in the tests contains holes, inaccuracies, and mistakes. This is often true for any data set. Intelligence from feeds will always require cleaning before it can be used. Some of the errors in the test were produced by unusual protocols and ports used by malicious actors. They appear to utilize the whole spectrum of possible URLs, likely to deceive unsuspecting victims and security solutions. Testing could be improved to make sure it covers all the possible ways intruders try to mask their presence and surpass defenses. Creating such tests requires deep technical knowledge on the way URLs work. The developers of security solutions need to be as smart as the opposition, because the intruders are ready to research deep into the technology they are abusing in order to get in.

One possible path for creating better Suricata rules is creating custom tailored generators for each feed. Currently, in both the Node and MISP versions, the same type of generator is used for all feeds. For the more information rich feeds, customized generators could yield more accurate rules. However, custom integration for each feed and the upkeep required to keep the integrations up to date requires a large amount of manual work.

Currently, the alerts only mark the indicator as malicious, but not the reason for this. Metadata could be added to these alerts. Using the data in the events it could be possible to add more detailed meta data such as what type of malware the rule is connected to, what possible vulnerabilities it is exploiting, or what sort of malicious activity the link is connected to. Metadata could also describe how often the indicator has been detected. When conducting the deduplication, it could be possible to note down the timeframe when the events happened, and add this information to the event. This would make it possible to communicate how active the rule is, which might help with understanding the alert at the receiving end. MISP has taken some steps forward on this front by adding a link to the event the rule was generated from. The types of metadata that should be included is going to vary from case to case, depending on who is going to use the rules and who is going to react to the alerts. The metadata could perhaps be configurable from the user interface of the integration.

5. CONCLUSION

The goal of this work was to integrate the Arctic Node threat intelligence sharing platform with a new intrusion detection and prevention system. Suricata was chosen as the integrated system. The new integration was tested against a similar functionality in the MISP threat intelligence sharing platform, which was also capable of exporting the collected threat intelligence as Suricata rules. One major distinction between these two was the fact that the new Suricata integration of Arctic Node featured deduplication for the rules so that no two logically identical rules were created. These two platforms also differ in the ways they gather data from threat intelligence feeds and create the Suricata rules.

From the tests it can be seen that the Arctic Node Suricata generation submodule outperformed the similar functionality in MISP. Node had 19.3% less events than MISP, but created 2.4% more rules than MISP. Arctic Node also had 16.9% more hits during the evaluation. The difference in volume can be explained by the difference in the way Arctic Node collects and creates events, as well as the way the generator works.

Suricata rules created by Arctic Node contained more diversity between different types than rules created by MISP. Finally, it can be observed from the results that there is relatively little overlap between the different open feeds.

6. REFERENCES

- [1] Dandurand L. & Serrano O.S. (2013) Towards improved cyber security information sharing. In: 2013 5th International Conference on Cyber Conflict (CYCON 2013), IEEE, pp. 1–16.
- [2] Shirey R. (2007) Internet security glossary, version 2. RFC 4949, RFC Editor. URL: <https://tools.ietf.org/html/rfc4949>.
- [3] Little E.G. & Rogova G.L. (2006) An ontological analysis of threat and vulnerability. In: 2006 9th International Conference on Information Fusion, pp. 1–8.
- [4] Dalziel H., Olson E. & Carnall J. (2014) How to Define and Build an Effective Cyber Threat Intelligence Capability. Elsevier Science.
- [5] Chismon D. & Ruks M. (2015) Threat intelligence: Collecting, analysing, evaluating. MWR InfoSecurity Ltd .
- [6] Brown S., Gommers J. & Serrano O. (2015) From cyber security information sharing to threat management. In: Proceedings of the 2nd ACM Workshop on Information Sharing and Collaborative Security, WISCS '15, Association for Computing Machinery, New York, NY, USA, p. 43–49.
- [7] (accessed: 27.5.2020), About project honey pot. URL: https://www.projecthoneypot.org/about_us.php.
- [8] (Accessed 27.6.2019), Bambenek consulting homepage. URL: <http://www.bambenekconsulting.com>.
- [9] (Accessed 27.6.2019), Phishtank homepage. URL: <https://www.phishtank.com>.
- [10] Caltagirone S., Pendergast A. & Betz C. (2013) The diamond model of intrusion analysis. DTIC Document URL: http://www.threatconnect.com/files/uploaded_files/The_Diamond_Model_of_Intrusion_Analysis.pdf.
- [11] Mavroeidis V. & Bromander S. (2017) Cyber threat intelligence model: An evaluation of taxonomies, sharing standards, and ontologies within cyber threat intelligence. In: 2017 European Intelligence and Security Informatics Conference (EISIC), pp. 91–98.
- [12] Pawlinski P., Jaroszewski P., Urbanowicz J., Jacewicz P., Zielony P., Kijewski P. & Gorzelak K. (2014) Standards and tools for exchange and processing of actionable information. European Union Agency for Network and Information Security, Heraklion, Greece .
- [13] Clark R. (2019) Intelligence Analysis: A Target-Centric Approach. SAGE Publications. URL: <https://books.google.fi/books?id=6ER\DwAAQBAJ>.

- [14] Endorf C. (2004) *Intrusion Detection & Prevention*. McGraw-Hill/Osborne.
- [15] Lim Y., Yer T.S., Levine J. & Owen H.L. (2003) Wireless intrusion detection and response. In: *IEEE Systems, Man and Cybernetics Society Information Assurance Workshop, 2003.*, pp. 68–75.
- [16] Liao H.J., Lin] C.H.R., Lin Y.C. & Tung K.Y. (2013) Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications* 36, pp. 16 – 24. URL: <http://www.sciencedirect.com/science/article/pii/S1084804512001944>.
- [17] Anderson J.P. (1980) *Computer security threat monitoring and surveillance*. Fort Washington .
- [18] Denning D.E. (1987) An intrusion-detection model. *IEEE Transactions on Software Engineering* SE-13, pp. 222–232.
- [19] Paxson V. (1999) Bro: a system for detecting network intruders in real-time. *Computer Networks* 31, pp. 2435 – 2463. URL: <http://www.sciencedirect.com/science/article/pii/S1389128699001127>.
- [20] (Accessed 4.7.2019), Zeek homepage. URL: <https://www.zeek.org>.
- [21] (accessed: 22.1.2020), Renaming bro project. URL: https://blog.zeek.org//2018/10/renaming-bro-project_11.html.
- [22] (Accessed 5.7.2019), Yara documentation. URL: <https://yara.readthedocs.io/en/v3.5.0/index.html>.
- [23] (Accessed 10.7.2019), Sigma project. URL: <https://github.com/Neo23x0/sigma>.
- [24] Roesch M. (1999) Snort - lightweight intrusion detection for networks. In: *Proceedings of LISA 1999: 13th Systems Administration Conference*, Seattle, Washington, USA, vol. 99, pp. 229–238.
- [25] (accessed: 25.10.2019), Snort homepage. URL: <https://snort.org>.
- [26] (accessed: 27.10.2019), Snort3 homepage. URL: <https://www.snort.org/snort3>.
- [27] (accessed: 25.10.2019), Suricata homepage. URL: <https://suricata-ids.org>.
- [28] Brumen B. & Legvart J. (2016) Performance analysis of two open source intrusion detection systems. In: *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, IEEE, pp. 1387–1392.
- [29] Zibak A. & Simpson A. (2019) Cyber threat information sharing: Perceived benefits and barriers. In: *Proceedings of the 14th International Conference on Availability, Reliability and Security, ARES '19*, Association for Computing Machinery, New York, NY, USA.

- [30] Sauerwein C., Sillaber C., Mussmann A. & Breu R. (2017) Threat intelligence sharing platforms: An exploratory study of software vendors and research perspectives. In: Proceedings of the 2017 Wirtschaftsinformatik, St. Gallen, Switzerland.
- [31] (Accessed 23.5.2019), Who is behind the misp project? URL: <https://www.misp-project.org/who/>.
- [32] (Accessed 5.8.2019), Misp model of governance. URL: <https://www.misp-project.org/governance/>.
- [33] (Accessed: 20.8.2019), Arctic security hub launch announcement. URL: <https://arcticsecurity.com/news/2018/06/20/arctic-security-set-to-launch-a-next-generation-cyber-intelligence-product-arctic-hub-this-summer/>.
- [34] (Accessed: 20.8.2019), Arctic security node launch announcement. URL: <https://arcticsecurity.com/news/2018/12/13/arctic-security-launches-arctic-node-to-help-enterprises-increase-their-cyber-security/>.
- [35] (Accessed: 4.6.2020), Benefits of an organized cyber defense. URL: <https://arcticsecurity.com/guides/2018/12/13/benefits-of-an-organized-cyber-defense/>.
- [36] (accessed: 4.6.2020), Data harmonization ontology (tlp white). URL: <https://github.com/arcticsecurity/public/blob/master/docs/Harmonization.md>.
- [37] (Accessed: 16.8.2019), Dnschef v 0.4. URL: <https://github.com/iphelix/dnschef>.
- [38] (Accessed 27.6.2019), Feodo tracker homepage. URL: <https://feodotracker.abuse.ch>.
- [39] (Accessed 28.6.2019), Ransomware tracker homepage. URL: <https://ransomwaretracker.abuse.ch>.
- [40] (Accessed 22.5.2020), Feed overlap analysis matrix. URL: <https://www.misp-project.org/feeds/#feed-overlap-analysis-matrix>.

7. APPENDICES

7.1. MISP feeds

MISP supports the following feeds.

1. CIRCL OSINT Feed - CIRCL
2. The Botvrij.eu Data - Botvrij.eu
3. ZeuS IP blocklist (Standard) abuse.ch
4. ZeuS compromised URL blocklist - abuse.ch
5. blockrules of rules.emergingthreats.net - emergingthreats
6. malwaredomainlist - malwaredomainlist
7. Tor exit nodes - TOR Node List dan.me.uk
8. Tor ALL nodes - TOR Node List dan.me.uk
9. cybercrime-tracker.net - all - cybercrime-tracker.net
10. Phishtank online valid phishing - Phishtank
11. listdynamic dns providers - <http://dns-bh.sagadc.org>
12. ip-filter.blf - labs.snort.org - <http://labs.snort.org>
13. longtail.it.marist.edu - longtail.it.marist.edu
14. longtail.it.marist.edu 7 days - longtail.it.marist.edu
15. diamondfox_panels - pan-unit42
16. booterblacklist.com Latest - booterblacklist.com
17. pop3gropers - home.nuug.no
18. Ransomware Tracker CSV Feed - Ransomware Tracker abuse.ch
19. Feodo IP Blocklist - abuse.ch
20. hosts-file.net - hphost - malwarebytes - hosts-file.net
21. hosts-file.net - hphost - malwarebytes - EMD classification ONLY - hosts-file.net
22. OpenPhish url list - openphish.com
23. firehol_level1 - iplists.firehol.org
24. IPs from High-Confidence DGA-Based C&Cs Actively Resolving - osint.bambenekconsulting.com

25. Domains from High-Confidence DGA-based C&C Domains Actively Resolving - osint.bambenekconsulting.com
26. ci-badguys.txt - cinsscore.com
27. [alienvault reputation generic](https://alienvault.com) - alienvault.com
28. blocklist.de/lists/all.txt - blocklist.de
29. [VNC RFB](https://dataplane.org) - dataplane.org
30. [sshpwauth.txt](https://dataplane.org) - dataplane.org
31. [sipregistration](https://dataplane.org) - dataplane.org
32. [sipquery](https://dataplane.org) - dataplane.org
33. [sipinvitation](https://dataplane.org) - dataplane.org
34. All current domains belonging to known malicious DGAs - osint.bambenekconsulting.com
35. [VXvault - URL List](https://vxvault.com) - [VXvault](https://vxvault.com)
36. [abuse.ch SSL IPBL](https://abuse.ch) - abuse.ch
37. [abuse.ch Dyre SSL IPBL](https://abuse.ch) - abuse.ch
38. <http://cybercrime-tracker.net> - <http://cybercrime-tracker.net> hashlist
39. <http://cybercrime-tracker.net> - <http://cybercrime-tracker.net> gatelist
40. [hpHosts - GRM only](https://hpHosts.com) - [hpHosts](https://hpHosts.com)
41. blocklist.greensnow.co - greensnow.co
42. [conficker all domains generated](https://cert.at) - cert.at
43. [CoinBlockerLists domains](https://CoinBlockerLists.com) - A list for administrators to prevent mining in networks - [CoinBlockerLists](https://CoinBlockerLists.com)
44. [CoinBlockerLists optional domains](https://CoinBlockerLists.com) - An additional list for administrators - [CoinBlockerLists](https://CoinBlockerLists.com)
45. [CoinBlockerLists browser mining domains](https://CoinBlockerLists.com) - A list to prevent browser mining only - [CoinBlockerLists](https://CoinBlockerLists.com)
46. [CoinBlockerLists IPs](https://CoinBlockerLists.com) - A additional list for administrators to prevent mining in networks - [CoinBlockerLists](https://CoinBlockerLists.com)
47. [URLHaus Malware URLs](https://abuse.ch) - [Abuse.ch](https://abuse.ch)
48. [CyberCure - IP Feed](https://www.cybercure.ai) - www.cybercure.ai
49. [CyberCure - Blocked URL Feed](https://www.cybercure.ai) - www.cybercure.ai

50. CyberCure - Hash Feed - www.cybercure.ai
51. ipspamlist - ipspamlist
52. mirai.security.gives - security.gives
53. malsilo.url - MalSilo
54. malsilo.ipv4 - MalSilo
55. malshare.com - current all - malshare.com
56. Benkow.cc RAT - benkow.cc
57. Panels Tracker - [Benkow.cc](http://benkow.cc)