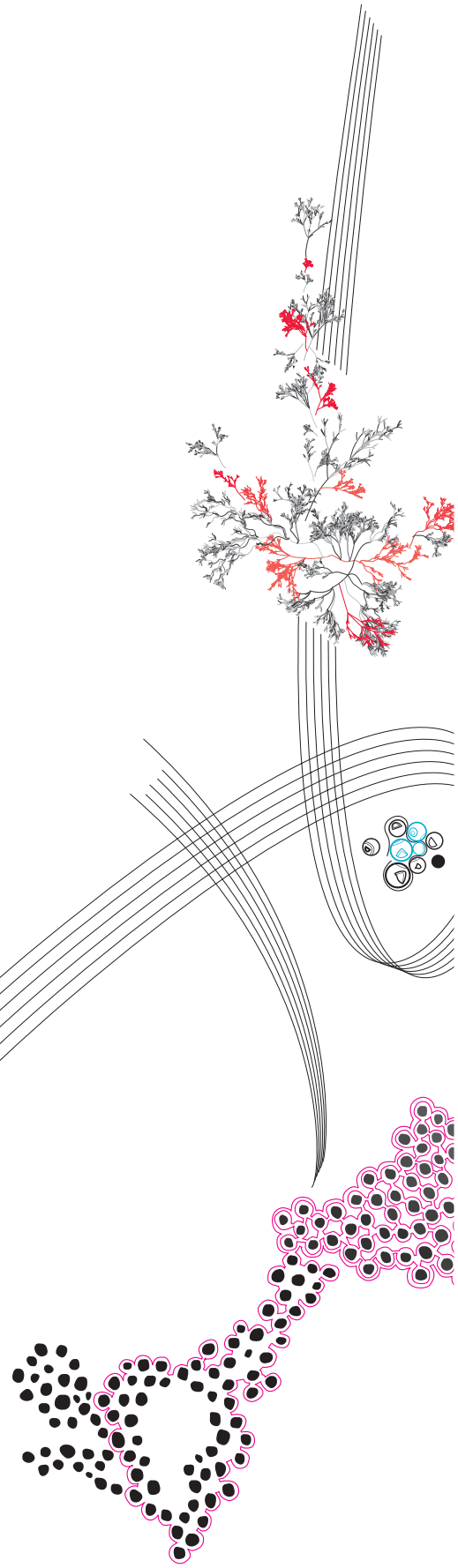


Master Thesis

Enabling Cyber Threat Intelligence Sharing for Resource Constrained IoT

August Karlsson

Supervisor: Shahid Raza
Jeroen van der Ham-de Vos

October, 2023

Department of Computer Science
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Abstract

The number of cyber attacks and the costs because of them is increasing, all the while the means with which to conduct them is decreasing. It follows a stark trend over the past couple of years which seems to continue for the ones to come. To better defend against this increasing threat cyber threat intelligence has developed. This shared intelligence has allowed participants to improve their awareness of threats, resolve vulnerabilities and have a better understanding of the security landscape. To improve the state of cyber threat intelligence sharing the standards STIX and TAXII were introduced. STIX addresses the need for a data format that can be automatized while still being human-readable and TAXII the ways in which it is shared. Threat intelligence platforms, of which MISP is one, are an important part in sharing cyber threat intelligence as well as they collect, aggregate and share data. The development of cyber threat intelligence has still missed much of what happens in the sphere of the Internet of Things, IoT. IoT, which are network connected devices, such as sensors, have also increased in numbers and are now commonplace. The heterogeneity of IoT coupled with the constraints that they commonly have on memory, processing power and energy capacity, as well as a history of poor security have made them targets in many cyber attacks. As such it is of interest to improve cyber threat intelligence on IoT. Currently cyber threat intelligence for IoT is gathered by honeypots disguised as IoT devices or by finding commonalities with cyber threat intelligence for standard computing systems. This work complements this by adding a compressed data format, tinySTIX, which can be shared over CoAP with OSCORE using observable resources. Two implementations for this have been created and evaluated. One in Python which is intended to be integrated to the MISP threat intelligence platform and one in C that can be run with the Contiki operating system on a resource constrained IoT device. It was shown that tinySTIX reduces the message size by 35% on average and CoAP with OSCORE reduces both the number of packets and size of a session by 85% compared to the reference OpenTAXII implementation. Furthermore it was validated that tinySTIX messages could be sent from a resource constrained IoT device, although OSCORE could not be combined with observable resources as intended. It was shown that sending a typical message would increase the energy consumption by 5% over the baseline consumption for a duration of 640-2560 milliseconds. While more work has to be done to provide such a system that is of production quality, the proposed model and implementation can be considered feasible for cyber threat intelligence sharing for resource constrained IoT.

Keywords: Cyber Threat Intelligence, CTI, STIX, TAXII, Internet of Things, IoT, Indicator of Compromise, IoC, MISP

Acknowledgements

This thesis work has been carried out at the Research Institutes of Sweden, RISE, in 2023. RISE is a research organization that works closely with academic institutions as well as industry. They conduct research on their own as well as with other research partners nationally and internationally. This thesis work follows the work conducted by researchers at RISE as well as the work from international projects. Without this or their help, this work would not have been possible to conduct. Because this work ties into what has previously been created and worked on as well as the research that is done at RISE, choices have been made during so that this work has been aligned with their goals and intentions for this work. Choices have also, if deemed reasonable, also taken their expertise into account, such as if there is more experience with a platform over another.

I would like to express my gratitude towards the Cybersecurity Unit at RISE for taking me on for this project and for supporting me along the way. I would also like to especially thank Shahid Raza, Alfonso Iacovazzi and Rikard Höglund with whom I could not have managed without.

At the University of Twente, I would like to thank Jeroen van der Ham-de Vos. I truly appreciate your patience, honesty, feedback and helpfulness. My gratitude also goes to Sabine at BOZ, for your help with administration throughout.

To my family and friends for the support during and preceding this work, thank you!

Contents

I. Introduction

In 2022 cyber attacks increased by 38% over the previous year [1]. It follows a longer trend of increasing number of cyber attacks. It is estimated that cyber attacks globally could cost $10,5 trillion by 2025, up from $3 trillion in 2005 [2]. The 2023 Data Breach Investigations Report from Verizon found that a majority of attacks are conducted by an actor outside the organization, 83% in 2022, and of those, a majority was part of organized crime, at about 70%. Nation-state or state-affiliated attacks are much less common, being slightly less than 10% of the attacks. Furthermore financial motives still drive nearly all breaches, following a trend of previous years, at 94,6% of the analyzed breaches. Ransomware still is a major threat against organizations of all sizes as part of what is classified as system intrusion incidents. So are denial of service attacks and basic web application attacks, although the latter is now less prevalent than both lost and stolen assets, and social engineering [3].

The simplicity of conducting such attacks and the potential financial gains from conducting them have created a plethora of services enabling cyber crime. Ransomware is offered as a service, credentials and exploits too, are being offered on the dark web for a price, as are botnets for performing attacks like denial of service.

The means for launching a cyber attack therefore, are very small. Similar attacks using the same vulnerability or tool can therefore be used for different targets. An attacker does not need to find a new vulnerability or tool but can extensively reuse existing ones just by finding a new target. This may, at least in part, explain the increase in cyber attacks against all regions and sectors. Selling, or sharing, credentials, tools, vulnerabilities and more on the dark web and between cyber criminals has allowed for performing more attacks with less resources. Network effects in cyber crime has led to more attacks, larger costs for victims and lower entry barriers to cyber crime. It is therefore argued by McKinsey, a consultancy [4], that defenders too must take a collaborative approach. Companies cannot bear the cost or evolve at the necessary speed themselves to efficiently protect their infrastructure and data, they argue. Collaborative efforts and tapping into cyber threat intelligence (CTI) are low cost alternatives that paired with data analysis can make a reactive defense model preventative [4]. To improve cyber ones cyber security, one of the main given advice is and has long been regularly updating systems to have the latest patches. While this advice still is important, keeping all systems on par with the latest update is often unfeasible and other defences too need to be used. It is not only a challenge because of system dependencies, it is also the time and the amount of systems that may need to be updated. While not taking away from the value of system updates, a multi-pronged approach to cyber security is the most likely to give good results at a bearable cost. Here data, data analytics and shared cyber threat intelligence plays a part.

Intrusion detection systems (IDSs) are systems that can detect anomalies, malicious behaviors, or policy violations in networks and systems. They are key to knowing what happens in an organization's digital infrastructure. These IDSs generate alerts that are handled by analytics software and key findings are commonly sent to a Security Operations Center (SOC). Critical and serious findings are immediately acted upon and lesser ones are scheduled. It is a data driven approach and a larger number of alerts are kept manageable by better pre-processing with data analytics. These IDSs are themselves data driven. Signature detection in IDSs rely on those signatures once being found and added to the systems, so does anomaly detection. A data sharing approach therefore, has a lot to bring.

Between cyber security professionals this has been through emails, meetings, forums and such. An automated data sharing, and more specifically cyber threat intelligence sharing, has been missing. Threat Intelligence Platforms (TIPs) have as such been introduced, where data is collected, aggregated and analyzed. While one challenge has been increasing data sharing from organizations, another has been collecting such intelligence in a way that is comparable, understandable and actionable. The latter has been addressed with the development of STIX, Structured Threat Intelligence eXpression, a data format that can be automatized while still being human-readable.

In parallel, a greater digitization has brought about the Internet of Things, commonly referred to as IoT. These small "things" are often low power and resource constrained computers, that are wirelessly connected. Low cost and low power usage is what makes them useful for continuous measurements, for example. They are part of what is described as the Industry 4.0 shift in manufacturing, one where IoT devices can send data about temperature, pressure or other information about the production in real-time. They are seen as part of a shift to a manufacturing process where real-time data optimizes production flow, minimises errors and improves automation. Naturally this also means that these devices also can be vulnerable and attacked.

Fortinet found 93% of companies using IoT to have had one or more cybersecurity intrusions in the past year and 78% had had more than three. Increasingly these attacks are also targeting IoT operations. The reasons for this are multiple. As IoT adoption increases, so does the interest in attacking them. According to Statista there will be 29 billion such devices in 2030, up from the current 9.7 billion. Advances with 5G networks and remote work has increased this adoption rate. Securing these devices and their environment poses a challenge because of the increased attack surface, new attack vectors and them often publicly visible. IoT devices can be targeted for remote execution, such as with the Mirai botnet, for getting access to sensitive data on the device, as in the case with connected medical devices, or as an entry point to launch further attacks. Low adherence to security standards worsen the situation and security standards made for desktop devices are not fit for resource constrained IoT [5]. This leaves a security situation for IoT and the environment in which it is used in a difficult spot. IoT devices are used increasingly for more purposes while security of those devices still is a low priority for many manufacturers and users. Izzat Alsmadi, a Computer science professor at Texas A&M University in San Antonio explains the situation with IoT standards as the following [5]:

> "Today's IoT standards are relevant, but not enough and in some cases not up to date or not up to security challenges. That's because some of today's existing security mechanisms were initially designed for desktop computers and are difficult to implement on resource-constrained IoT devices."

Addressing security for IoT therefore has to cater to the resource constraints, to consider the heterogeneity of IoT devices, and take into account that IoT devices are networked. For cyber security in general cyber threat intelligence in different forms have been instrumental for knowing about attacks, developing preventative measures and protecting against them [6]. In order to know the kinds of attacks that are being performed there has been work in creating IDSs for IoT devices, but there is little work done on cyber threat intelligence for the same. As such this master thesis addresses the lack of cyber threat intelligence for IoT devices by adapting the STIX data format and accompanying TAXII communication

protocol, to the resource and power constraints of resource constrained IoT devices. The different design decisions and choices are evaluated, it will be implemented in software and following that evaluated on hardware.

II. Technology Background

i. Threat Intelligence Sharing

Cyber Threat Intelligence (CTI), hopes to bring situation awareness among sharing stakeholders. It is seen as a way to work proactively against attacks instead of just reactively. By sharing threat intelligence, the stakeholders can better keep up to date with the latest vulnerabilities and remedies. Some methods of intelligence sharing between stakeholders are, emails, phone calls, meetings, shared databases, data feeds, and web portals. These have the drawbacks of being slow in sharing new threats, having a human error rate during processing, and subjective relevance filtering. CTI is information that has been analyzed and that is actionable.

Automation is needed as the number of experts that can analyze it are scarce and there is an increase in data to analyze. For this to be possible threat intelligence needs to be structured to be efficiently analyzed. These are needs like analyzing cyber threats, specifying indicator patterns, managing response activities and sharing of cyber threat information. This can be achieved with STIX and then sent over TAXII. The STIX standard defines the data format in which threat intelligence is written, which addresses the lack of a common language which hinders global CTI sharing. Something that is essential for its effectiveness [7].

One common reason why stakeholders hesitate to share their CTI is the belief that they possess nothing worth sharing, and competitors could potentially exploit the information against them. Nevertheless, according to the European Union Agency for Network and Information Security (ENISA), there are approximately 80 initiatives, organizations, and more than 50 national and governmental Computer Security Incident Response Teams (CSIRTs) involved in CTI sharing within the European Union (EU) and European Economic Area (EEA). Many organizations have recognized the necessity of CTI exchange to survive future attacks. Efforts to improve information sharing have been made by governments worldwide, including the United States, Japan, South Korea, and several EU member states. However, collaborative CTI sharing can introduce privacy risks, particularly when data is shared on an application layer, potentially leading to the sale of sensitive information on the dark web [7].

Non-participation in threat intelligence sharing reduces the possibility of mitigating attacks, which can have significant impacts not only on assets but also on reputation and brand. CTI is a collection of various attributes that collectively form actionable intelligence. While IP addresses are not CTI themselves, they can be part of the intelligence. Other attributes include threat actors, campaigns, motivation, and Indicators of Compromise (IoC). However, it is important to note that CTI indicators primarily focus on enterprise IT and often neglect fields such as the Internet of Things (IoT), industrial IoT (IIoT), and the automotive sector. Nevertheless, devices in these areas could still benefit from CTI indicators originally intended for enterprise IT [7].

Governments and organizations have established industry-specific sharing groups to address sector-specific vulnerabilities. These sectors encompass finance, retail, academia, automotive, and more. Effective CTI exchange involves secure exchange, environmental

sustainability, rapid customization, correct labeling, anonymity, relevance, trust, and confidentiality. Tactical intelligence exchange within CTI also includes Techniques, Tactics, and Procedures (TTP) and IoCs [7].

The lack of incentives to share sensitive security information often results in organizations exhibiting free-riding behavior. Stakeholder behavior regarding CTI sharing can be categorized as either obedient, following regulations and policies, or malicious, disregarding regulations and policies. Malicious actors may exploit CTI collected from obedient stakeholders to launch attacks [7].

Trust is a crucial component of CTI sharing, typically established over time and through face-to-face meetings. It is considered the most challenging attribute in the threat intelligence sharing ecosystem. Stakeholder trustworthiness is evaluated based on trust and reputation. ENISA identified three trust relationships: organizations trust the platform to ensure confidential data is not exposed to unauthorized stakeholders, the correct handling of information (e.g., Traffic Light Protocol (TLP) labeling), and shared information is credible and reliable [7].

Filtering plays a vital role in managing the large quantities of data involved in CTI. Filtering allows stakeholders to sift through the data load and identify relevant information. This filtering can be done based on industry and area of responsibility, such as networks, software, and hardware. Privacy and anonymity are also relevant considerations. The disclosure of sensitive information can be used against stakeholders if obtained and understood by malicious actors, potentially discouraging their participation in threat intelligence sharing [7].

ii. STIX

Data interoperability remains a challenge due to the lack of a globally common format for CTI exchange. The Structured Threat Information Expression (STIX) format is currently the most widely accepted standard for sharing threat intelligence. Additionally, local laws and regulations may shape what is permissible to share when it comes to data, such as IP addresses being considered personal information in Germany but not in the UK [7].

STIX, or Structured Threat Intelligence eXpression, is a standard language for describing Cyber Threat Intelligence. It is done in a way that it can be read easily by humans while also being understood by machines so that it can be acted upon and automatized. STIX combines properties that sign malicious activity with those that provide contextual information. The properties are extensive and can thereby give detailed information. STIX uses JSON as a data format and can be sent with any transport mechanism. It was however designed together with the TAXII protocol that is specifically designed to transport STIX messages [8]. Both STIX and TAXII are now managed by OASIS Open [9][10].

STIX has three main components, STIX Domain Objects (SDOs), STIX Cyber-observable Objects (SCOs), and STIX Relationship Objects (SROs). SDOs are descriptions of core concepts such as campaign, indicator, course of action, etc., while SCOs describe facts that are part of a cyber-attack such as files, IP addresses, registry keys, etc.. SROs describes the relationships between the former two. [11][10].

Because of the relationships between SDOs, SCOs as described with the relationship objects (SROs), STIX messages can describe complex attacks and relationships between for example multiple attacks and threat actors. As these messages can become large they are commonly showed in a graph with nodes and links.

iii. TAXII

TAXII, an abbreviation of Trusted Automated Exchange of Intelligence Information, is an application layer protocol made specifically to communicate STIX messages, although it can also deliver payloads of other formats as well. It defines a RESTful API together with requirements for client and server implementations to share and discover CTI[9].

TAXII allows for sharing logical groupings of CTI together with metadata that allows for search. A TAXII client can request desired CTI from a TAXII server based on a set of metadata filters in the request. A manifest of available CTI can also be requested, so can information about the structure of a CTI collection[9].

TAXII builds on top of HTTP for negotiation and authentication. TAXII servers can be discovered within a network via DNS service records. TAXII communication is done over HTTPS[9].

TAXII has a API Root which can be learned about through the Discovery information. The API Root then can host Collections, the request-response sharing paradigm, or Channels, the publish-subscribe sharing paradigm, as models to share CTI. Collections include getting a manifest of CTI contained in the Collection, adding new CTI and retrieving CTI from it. Channels have yet to be specified in an upcoming version of TAXII. The API Root also hosts Status which stores information such as if a CTI submission was accepted and added [12][9].

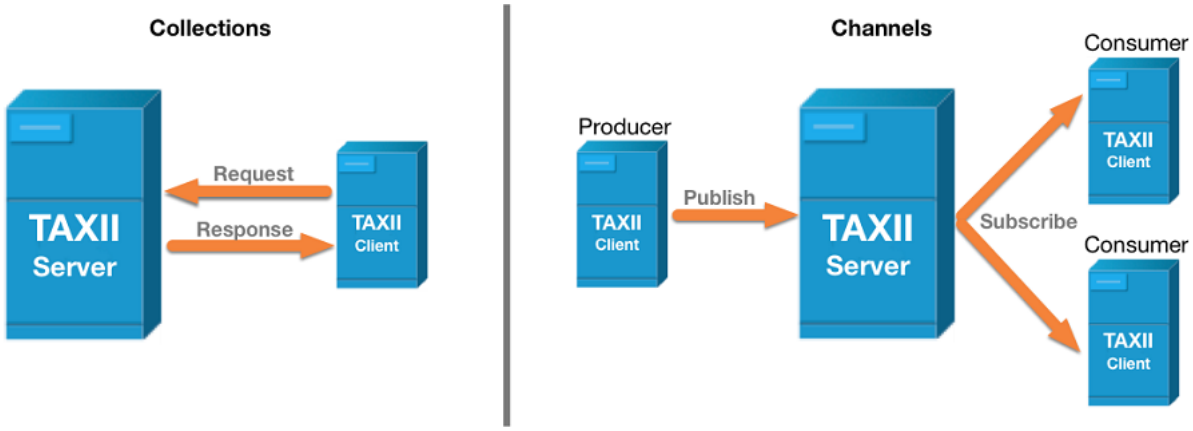


FIGURE 1: TAXII sharing models [12]

iv. Communication Protocols

For resource constrained devices TCP (and QUIC), that are the transport protocols for HTTP, are too demanding and the more lightweight protocol UDP is instead used. It does

not provide the reliability, error checking and order that TCP does but provides a communication method with less overhead and lesser resource demands. Starting from UDP as the transport layer protocol for this IoT version of TAXII and there are several application layer protocols to choose from. MQTT and CoAP are notable ones. MQTT supports a publish-subscribe communication model whereas CoAP mainly mimics the REST capabilities of HTTP. CoAP does however also provide observable resources which are similar to how publish subscribe works. MQTT uses TLS for message confidentiality while CoAP offers a choice between DTLS and OSCORE [13]. Because TAXII has a RESTful API it is of interest to do the same in the implementation for resource constrained IoT. As such CoAP is the preferred choice for this implementation.

DTLS, Datagram Transport Layer Security, is the common way of providing message security in CoAP and what is commonly referred to when talking about CoAPs. It is similar to TLS which runs on TCP but is adapted to instead run over UDP, which is what CoAP for example, uses. As it largely is based on TLS it follows the development and consequently trails behind. DTLS 1.3 became a proposed standard in April 2022 following TLS 1.3 which was released in 2018. Vulnerabilities found in TLS could therefore remain in DTLS for longer before being patched. DTLS provides equivalent security guarantees as TLS 1.3 with the exception of order protection/non-replayability. Consequently if an attacker would obtain a message, it could be resent in order to provide wrong or intentionally harmful information [13].

OSCORE, Object Security for RESTful Environments, takes a different approach to message security and does not encrypt the packet headers. As such it can provide end-to-end security even when using a proxy since the proxy does not need to decrypt and then encrypt the message again in order to pass it on. This does however mean that some information about the packet is leaked as the header is in plain text. Since CoAP has methods like the ones in HTTP, translation between the two is simple. With OSCORE this means the message does not have to be decrypted in the translation between the two protocols. Furthermore OSCORE does not specify a key-sharing algorithm as part of its specification and is left as an implementation choice [13].

III. Formal Problem Definition

i. Settings Definition

IoT is an umbrella term for a wide variety of connected devices, from medical equipment in a hospital to connected cars to humidity sensors in a field, they can all be considered IoT. In this work the focus is on the resource constrained IoT, small devices like the humidity sensor. They have smaller memory, processing power and energy efficiency is important as they often are battery powered. These constraints are stricter than for other IoT and the work in this thesis could therefore, also run on other IoT that are not resource constrained.

Because of these resource constraints the devices in this thesis work are not considered consumers of CTI. This is because most CTI currently is for enterprise systems and because CTI most commonly is contextualized, objects are related to each other or to other data, CTI can be large and too large for resource constrained IoT. The IoT devices in this work are therefore only considered as generators of alerts, small pieces of information such as a log file, which then, on an enterprise system, can be put together to become CTI that can be shared more widely.

RISE, the organization at which this master thesis is conducted, is in parallel working on the threat intelligence platform MISP. It is an open source threat intelligence platform that aggregates data, correlates and links data, visualizes it and allows for the data to be exported in a variety of data formats, including STIX. There are default data feeds that are used for MISP and more data can either be imported manually or automatically in different formats. The core software is mainly written in PHP but there is a Python library that uses the MISP Rest API. There is also a converter from the MISP data format to STIX.

The work in this thesis is made with the intent to be integrated into the MISP project. Because of that the implementation has to be written in Python, which is the language used for MISP. That also means that the communication can be directly between the IoT device and the MISP platform, through the provided Python implementation. Other communication means such as multicast or broadcast are not considered. This master thesis does not cover the integration of the provided solution to the MISP threat intelligence platform however.

Another part of this is considering the role of the IoT device. A communication protocol and accompanying data format are naturally possible to use in either direction of communication, but the data that is sent and the restrictions under which the communicating parties communicate can make this unfeasible, especially considering one party being a resource constrained device. For the purpose of design, implementation and evaluation a reference hardware has therefore been chosen.

As a reference for a resource constrained device a breakout board like the Zolertia Firefly can be used. It has an ARM Cortex-M3 that has 512kB flash memory and 32kB RAM and can be powered by two AA batteries. It is suitable for most IoT applications. It can run Contiki as the operating system and it has a hardware encryption engine [14].

Given this reference hardware and its constraints, the role of the IoT device can be revisited.

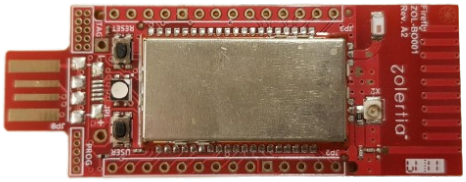


FIGURE 2: Zolertia Firefly Revision A

The MISP platform can aggregate data, correlate it and link data together, all to make it more understandable and actionable, ultimately creating cyber threat intelligence from what initially could have been unorganized data. By doing this the data naturally becomes larger and more complex. Sending this data to a resource constrained device is not feasible simply because it can be larger than the entire storage capacity of the device. Even the input data can be too large for a resource constrained device to handle. Analyzing the data sizes from the default data feeds that are used for MISP and the following can be seen.

	CIRCL	URLHaus Malware URLs	Threatfox	Botvrij
Number of items	1500	890	918	343
Smallest data item	461	1,5M	9,4K	288
Largest data item	418M	178M	40M	9,5M
Median item size	26K	21M	225,5K	6,4K
Number of items $<$ 2KB	32	0	0	33

TABLE 1: Data from a sample of MISP default data feeds in MISP data format. Sizes in bytes. Taken 21/09-2023

Since the reference hardware only has 512 KB of flash memory only a small subset of the data from these feeds would be possible to send to the resource constrained device, and that data might not be relevant for IoT. Furthermore data of concern for IoT might be filtered out simply because it would be too large to send to the device. Consequently, resource constrained IoT devices are not suitable to be consumers of indicators of compromise or cyber threat intelligence but can instead be generators of alerts and other smaller pieces of information. This can then be put together in MISP where small contributions from resource constrained IoT paint the picture of the current threat landscape. There is ongoing research on host-based intrusion detection systems for IoT, and it is from such a system that these alerts are expected to come. Testing such a system however would add to much variability and complexity to the work and for the purpose of implementation and evaluation such a system is assumed.

Because of the resource constraints, C will also be used as the programming language for the implementation for the IoT device. A lightweight operating system such as Contiki or Zephyr is expected to be installed on the device. It is also assumed that in the system in which this resource constrained device is added it is done so by an administrator. It is therefore known to the administrator and does not need to be discovered and potential

security parameters can be set up before the start of the communication with the device.

IV. Related Work

The road to cyber threat intelligence for IoT devices has multiple paths. Looking at data generation, and at a later stage cyber threat intelligence is one. Approaches for data gathering in the realm of IoT may be by generating data locally, such as incident alerts, or by scouring through existing data to find what is applicable to IoT devices. In the former, approaches such as honeypots serve as one way to gather information from an attacker. Here Conpot [15] is one implementation for Industrial Control Systems (ICS), another non-enterprise system. Honeypots can also act as IoT devices without actually running on one, as in the work of [16] Dowling et al., where ZigBee ports were emulated data about attackers was captured. Such data could also be used to create intelligence. Another approach for data generation is through an IDS (Intrusion Detection System). This has been tried and tested in the work of Raza et al. with the development of SVELTE [17].

The state of cyber security for IoT has received increased attention as IoT devices have proliferated and their inclusion in more networks and services has risen. IoT devices can, among other places, be found in homes, factories, agriculture, infrastructure. When addressing security for IoT devices, it is to preserve privacy, confidentiality, ensure the security of the users, data, infrastructure, the devices themselves and the availability of the services they provide. Most current approaches involve security mechanisms based on traditional network infrastructures. It is however more challenging to address IoT security than that. The heterogeneity of IoT devices, from self-driving vehicles to resource constrained sensors, mean that devices and protocols alike are varied. Another challenge is the number of nodes in such a network of IoT devices. Much research has been conducted in how to address key management, confidentiality, integrity, privacy and policy enforcement for IoT systems. As such it has been suggested to use traditional cryptography, Software Defined Networking (SDN) and Blockchain to solve current IoT security issues [18].

Incident alerts or data alone is of relative insignificance; it can rarely be acted upon and lacks relations to other kind of data. STIX is one of the data formats developed to bind together data and relationships to build intelligence. It is human-readable and can be automatized.

Legislation is both pushing disclosure of incident information and sharing of it in the form of incident information as is as well as in the form of reports and statistics. An example of this are the NIS-directives from the European Union with NIS in 2016, Directive (EU) 2016/1148, and the updated NIS2 in 2022, Directive (EU) 2022/2555. The NIS directives aim to mitigate threats to network and information systems that provide essential services in key sectors. They require actors responsible for such services to disclose incidents and provide reports about them. It also requires each Member State to have at least one computer security incident response team (CSIRT). The NIS2 Directive adds among other things more responsibilities to these CSIRTs and more key sectors that this concerns [19].

Iacovazzi et al. worked from the STIX data format to model an implementation for IoT devices. This model uses CBOR, Concise Binary Object Representation, for data encoding instead of JSON, JavaScript Object Notation, in STIX and CoAP for message communication instead of TAXII. They calculate a message size reduction of between 24 and 52 percent depending on the data. The paper motivates the need for this so called tinySTIX and proposes a method for mapping STIX to tinySTIX messages and how to map TAXII

to CoAP. It however does not go through certain aspects of these models, nor does it provide an implementation for it. This is where this work can continue and build upon the previously proposed model, evaluate the proposed design decisions, implement such system and evaluate it [20].

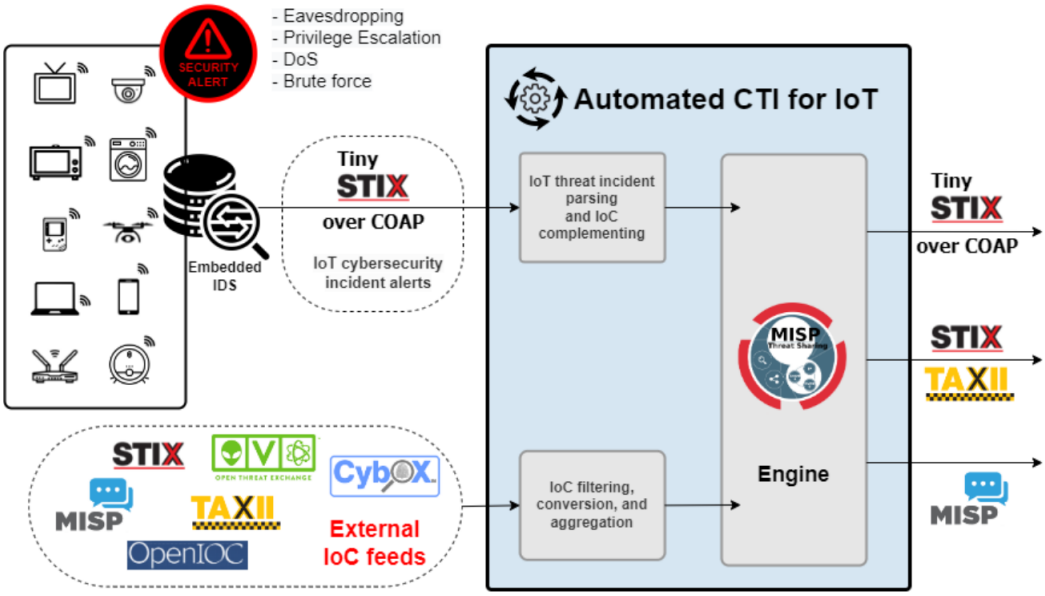


FIGURE 3: Proposed model by Iacovazzi et al. for automated CTI for IoT devices [20]

V. Research Questions

There are two protocols, DTLS and OSCORE, that can be used to provide alternatives to HTTPS used in TAXII.

Question 1. - Comparing DTLS and OSCORE: which is the best suited protocol for message confidentiality in the context of resource constrained IoT?

TAXII supports two sharing models, collections and channels. The former being a request-response model with a client and server where the latter is a publish subscribe model with publishers and subscribers.

Question 2. - For the purpose of cyber threat intelligence sharing from resource constrained IoT devices; which of the models collections and channels is the most suitable?

Question 2.1. - Could both collections and channels be mapped in CoAP with the same implementation?

Question 2.2. - Given the choice of modelling collections, channels, or both, which role is the IoT devices considered to take, client or server?

Prior to converting STIX messages, for which JSON is used as the data format, to CBOR, strings can be converted to integers in order to reduce the message size.

Question 3. - Could both key and value string values be mapped to integers before being formatted to CBOR?

Question 3.1. - How can a mapping from strings to integers be made so that the resulting message size is as small as possible?

TAXII requires HTTP Basic Authentication to provide authentication and additional authentication methods can be supported as well.

Question 4. - Is authentication desirable for communication over CoAP?

Question 4.1. - Which party, client or server, is it desirable to have authenticated?

Question 4.2. - How could authentication be added?

The implementation will be tested and evaluated on a chosen hardware. Different systems can have different libraries for implementing CoAP and OSCORE.

Question 5. - Can the implementation be separated so that different libraries for CoAP and OSCORE easily be changed in order to tailor to different hardware?

Following our implementation of tinySTIX over CoAP this will be tested. The Python implementation will be tested on an enterprise device and the implementation for a resource

constrained device will be tested on such a system. The Python implementation will also be compared on message size reduction using tinySTIX. CoAP with the protocol chosen for message confidentiality will be compared with TAXII and other relevant communication protocols.

Question 6. - Can tinySTIX be implemented in practice?

Question 6.1. - How does the tinySTIX implementation compare to the results of Iacovazzi et al. [20] when it comes to reduction of message size?

Question 7. - What is the message overhead for data sent over CoAP with the protocol chosen for message confidentiality?

Question 7.1. - What is the conversation overhead for multiple messages sent over CoAP with the protocol chosen for message confidentiality?

Question 8. - Is it practically feasible to send CTI in tinySTIX format to a resource constrained device?

For the implementation to be tested it will be run on a hardware like Contiki NG or Zephyr. Because of the nature of these devices, not having an operating system for example, and less debugging options, implementations on these devices can add time and complexity. Additional questions therefore relate to the hardware on which this can be implemented and what that means for evaluation since measuring power consumption is difficult on an enterprise system where there may be multiple processes running in the background.

Question 9. - Can CoAP with the protocol chosen for message confidentiality be tested on a resource constrained IoT device?

Question 9.1. - What is the energy consumption for sending a message over CoAP with the protocol chosen for message confidentiality?

VI. Modelling STIX and TAXII

Extending STIX and TAXII to work on resource constrained IoT devices means changing the underlying protocols. Functions, protocols and original design choices have to be mapped to the new implementation. While this means that a one-to-one mapping is not possible, a mapping that is as close as possible is desirable not to deviate unnecessarily from the original design. It can however, in some cases, be motivated to differ from the original design choices if there are better choices to be made for the IoT use cases. This chapter therefore goes through the original implementation of STIX and TAXII, and motivates the design choices for an IoT implementation, both those that keep close to the original implementation and ones that purposefully differ.

i. TAXII

TAXII is an application layer protocol that builds on top of HTTP, HyperText Transport Protocol. It is in turn built on top of TCP (HTTP/3 is built on top of QUIC), a transport layer protocol, which provides reliable, ordered and error checked communication. Secure communication for TAXII is made with HTTPS, ensuring confidentiality between communicating parties. With this as the basis of TAXII two sharing models have been created, collections and channels. In TAXII a collection is a sharing model in which items held on a server are requested by a client. It is a request-response model that is RESTful, meaning that it is stateless, has a uniform interface and supports a set of defined methods.

Key features

- Request-response communication model with intended support for publish-subscribe
- HTTP Basic Authentication enforced
- TAXII server discovery through DNS Service records and/or by Discovery endpoint
- Mandatory support for STIX 2.1

Choice of communication model In the laid out use-case and model for including IoT devices in cyber threat intelligence sharing they have been imagined as generators of incident alerts, a notification about an anomaly in the device or communication to it. These incident alerts are sent to an enterprise device or threat intelligence platform for data aggregation, analysis and potential action. As the data is generated by the IoT device and given its resource constraints, it is desired to send it off as soon as possible. As such the IoT device should initiate the communication.

- Initialized communication from IoT device
- Low resource requirements
- Close mapping to original TAXII standard
- Support for message confidentiality

Given the desire to keep the new implementation close to the original, a request-response model is best suited. CoAP is therefore a good choice as it has low resource requirements, provides a mapping to the HTTP request methods and has support for both DTLS and OSCORE for message security. It also has observable resources through a publish-subscribe feature can be added.

Message confidentiality To secure CoAP communication there are two options: DTLS, Datagram Transport Layer Security, and OSCORE, Object Security for RESTful Environments. Both provide message confidentiality but do so by encrypting different amounts of the message. DTLS also includes a handshake for key exchange which OSCORE does not. This could instead be added separately for OSCORE [13].

Authentication Authentication in TAXII is provided through the use of HTTP headers, namely the WWW-Authenticate and Authorization headers. HTTP basic authentication is required but additional authentication methods can be used. One commonly used is JSON Web Tokens (JWT) which after the first authentication is used instead of the user credentials.

With OSCORE a security context is set-up on both communicating devices. Every security context is unique. A security context could for example be set up by the network administrator. Given this as a scenario and the existence of a security context could be taken as an authentication to the other party, commonly the server. Furthermore different permissions could be given to any of these entities as it simply would require the server to know which entity it communicates with, which it does because of the security context, and then treat it accordingly. While this method of authentication and access control does not require additional protocols. It is crude in the access control since it either allows for access control over all entities at once or at a device-by-device basis. It also does not specify in which way the security context is set up and is left to be completed at an earlier stage.

While excluded from the scope of this project additional parts could be added to provide more control to the access control measures and to take control of security context set up so that this can be done dynamically. To create a security context EDHOC, Ephemeral Diffie-Hellman over COSE, can be used. With EDHOC a handshake is performed between the two parties and using asymmetric encryption a OSCORE security context is generated. It is a lightweight protocol relying on CBOR, COSE (CBOR Object Signing and Encryption) and CoAP. As part of the generation of the security context three (optionally four) messages are sent. [21]

A less crude implementation of access control would be through the use of ACE-OAuth, a framework for authentication and authorization in IoT environments [22].

ii. STIX

STIX, Structured Threat Intelligence eXpression, is a language and serialization format used to exchange CTI. It is open source and is part of the OASIS Open. While the first version used XML as the data format, the current and second version relies on JSON. It is meant to be human readable and easily automated. As such it defines three different

kinds of objects: SCO, STIX Cyber-observable Object, SDO, STIX Domain Object, and SRO, STIX Relationship object.

Listing 1: CDO

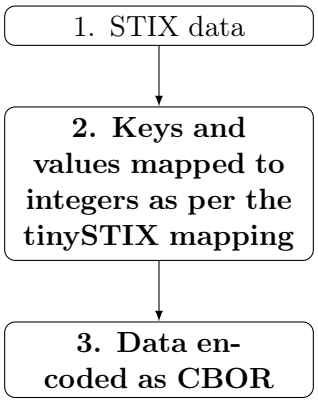
```
{
    "type": "campaign",
    "id": "campaign--8e2e2d2b-17d4-4cbf-938f-98ee46b3cd3f",
    "spec_version": "2.1",
    "created": "2016-04-06T20:03:00.000Z",
    "modified": "2016-04-06T20:03:23.000Z",
    "name": "Green Group Attacks Against Finance",
    "description": "Campaign by Green Group against targets
    in the financial services sector."
}
```

Key features

- Human readable and easily automated
- JSON data format
- Extensive type set
- Extensive property set
- Open vocabulary and enumerated sets

1) Data format

STIX has since version 2 used JSON, as the data format. It centers around objects in which each JSON object has a set of keys to which values are mapped. Keys are unique while values can be the same. Keys can only be strings in JSON. JSON is easy to work with and read but has not been designed to be compact. In the use case of resource constrained IoT devices it is therefore motivated to compromise human readability in favor of size reduction. A smaller version of the data can be created with CBOR, a binary object representation that easily can be converted to and from JSON. Because it is a binary format it is smaller in size but also loses the human-readability of it. Considering the use case in which an IoT device would generate an incident alert in tinySTIX format that then is sent to an enterprise device where it is converted to STIX, losing the feature of human readability is a reasonable compromise for a smaller message size.



2) Keywords

CBOR, unlike JSON, is not limited to strings as the data type for keys and in CBOR keys can be integers. Because of this a mapping can be made so that keywords as strings can be uniquely converted to integers and later reversed. By doing this the message size can be reduced further.

To take advantage of this a mapping from keywords to integers was created. There was no available list

Using a python script all keywords that are listed in the STIX 2.1 specification were extracted and mapped to a unique integer value.

Listing 2: Keyword to integer mapping

```
{
  "extensions": 0,
  "type": 1,
  "spec_version": 2,
  "revoked": 3,
  "object_marking_refs": 4,
  "lang": 5,
  "labels": 6,
  "id": 7,
  "granular_markings": 8,
  "external_references": 9,
  ...
}
```

3) Value set

The values in CBOR could naturally also be integers, as can they in JSON, but given that STIX has enumerated value sets and open vocabularies, listed values that can be extended, these too could have an integer mapping. As such known string values can first be mapped to integers before the data is converted to CBOR format.

Listing 3: Value to integer mapping

```
{
  "type": {
    "binary": 1,
    "boolean": 2,
    "dictionary": 3,
    "enum": 4,
    "external-reference": 5,
    "float": 6,
    "hashes": 7,
    "hex": 8,
    "identifier": 9,
    "integer": 10,
    ...
  },
  "account_type": {
    "facebook": 1,
    "ldap": 2,
    "nis": 3,
    ...
  },
  "primary_motivation": {
    "accidental": 1,
    "coercion": 2,
    "dominance": 3,
    ...
  }
}
```

4) Data size in mappings

In the generation of these mappings consideration can also be given to the frequency of which the values are used, giving favor to smaller integer values for the values that are used most often. In CBOR values in the range [0,23] are one byte, and values in the range [24, 255] are two bytes. This is because the three most significant bits are used to represent the major version. No value is mapped to an integer that would map to more than two bytes, and most would have value mappings within the range that results in one byte. Still, in order to minimize the data size the common properties for STIX Objects are placed first in the mapping so that they have lower integer representations.

The mappings are based on the properties, enumerations and open vocabularies that are listed in the STIX 2.1 standard. The open vocabularies can be extended and in that case will not have an integer mapping. This means the mapping will be able to convert without data loss.

VII. Methodology

i. Libraries

The goal of this research does is to provide a model for cyber threat intelligence sharing for resource constrained IoT. As such, existing libraries will be used as much as possible. This means finding libraries that provide the desired functionality such as CoAP, observable resources, and OSCORE. There also needs to be support for CBOR which is used as the data encoding in tinySTIX.

Setting up the OSCORE security context that is required for secure communication is assumed to be set by an administrator or other with access to the device.

Both the Zolertia and Contiki operating systems have support for CoAP and OSCORE, although Contiki has it in a forked repository from the main one. As such both of them could be considered.

- C :
  - CoAP and OSCORE for Contiki NG: https://github.com/Gunzter/contiki-ng
  - OSCORE for Zephyr: https://github.com/zephyrproject-rtos/zephyr/pull/46983
- Python :
  - https://github.com/chrysn/aiocoap

Support for CBOR exists natively in both Zephyr and Contiki and can be used in Python with the cbor2 library, https://pypi.org/project/cbor2/.

ii. Testing

Because the Python implementation can be run on a standard computing system it can also be compared to other implementations. The primary one for comparison is the OpenTAXII implementation from EclecticIQ which too is written in Python. An HTTPS server also serves as a reference as it provides the same basic functionality as TAXII but without the added TAXII functions.

On the hardware meanwhile it is not possible to do any reference testing. Instead the testing lies in implementing the proposed model to see if it can be done within the constraints of the device. Additionally it is possible to conduct energy consumption testing on the hardware as it is possible to better control what processes are run on it so that other processes on the device interfere.

VIII. Implementation

i. TAXII

The TAXII implementation for resource constrained devices is made with the aiocoap library. It supports both CoAP, OSCORE and observable resources.

The implementation is based on the plugtest files for the aiocoap library. It has functionality for a server, client and shared functionality in their own files. Resources can be accessed with or without OSCORE and can be observed as an observable resource.

In the server a resource tree is created, as commonly done, by having a .well-known/core in which all resources are listed.

CoAP is based on URI and as such each resource can be given on a unique URI.

ii. Verifying OSCORE

To test whether or not communication with OSCORE is functioning Wireshark was used and the same resource was accessed with and without providing an OSCORE security context. It was also verified that the data was encrypted when sent using OSCORE.

1) OSCORE Context

Each OSCORE security context consists of three files: secret.json, sequence.json, and settings.json. The files contain information needed to establish secure communication.

iii. STIX

The tinySTIX translator is written in Python and relies on the cbor2 library for conversion to and from CBOR. It consists of functions that can translate from STIX to tinySTIX and vice versa. In order to do this two JSON files containing a mapping between values and an integer representation of them has been created.

1) Value mapping

The mappings were also created using Python since no mappings or keyword lists could be found. The keywords and values were extracted from the HTML version of the STIX 2.1 version. In order to do this the BeautifulSoup library for data scraping was used. One file contains the properties for each object, such as type, name, and description. In the second file each of these properties has a mapping from the values it could have, either from an enumerated set or open vocabulary, and an integer. There are several different type objects, but they have in this case been put together. Since the values are unique there is no information loss because of it.

iv. Hardware

The chosen hardware representative of a resource constrained IoT device is the Zolertia Firefly Revision A. It is a breakout board with 512kB of flash memory and 32kB of RAM. It has a hardware encryption engine and supports 802.15.4 radio transmission. For the implementation Contiki-NG was chosen as the operating system as it has native support for CoAP and support for OSCORE can be found in a forked repository from the Contiki-NG main repository on GitHub. Support for OSCORE and CoAP could also have been found in the Zephyr OS but the use of Contiki at RISE, where this work was conducted, favored the choice of Contiki as more support could be provided.

Contiki has support for the Zoul module which is the core of the Zolertia Firefly (other boards than the Firefly from Zolertia use the same core module). Contiki is run in a docker container from which a program can be uploaded to the device. When the program is uploaded to the device the operating system is compiled with the program and they are written to the device. Each upload therefore overwrites the memory of the device. After uploading a program to a device it is run when the device is powered on and can consequently be given any power source and run the program. To communicate with the device however two choices are given. Communication can either be done with a link-local address over 802.15.4 radio transmission or by using a second device that serves as a border router.

In the option of using a border router, Contiki provides one that can be uploaded to a second device. This is then attached through USB to the computer and through it provides an IPv6 address that is reachable for other services on the main computer, although unless otherwise configured, only in the same docker image.

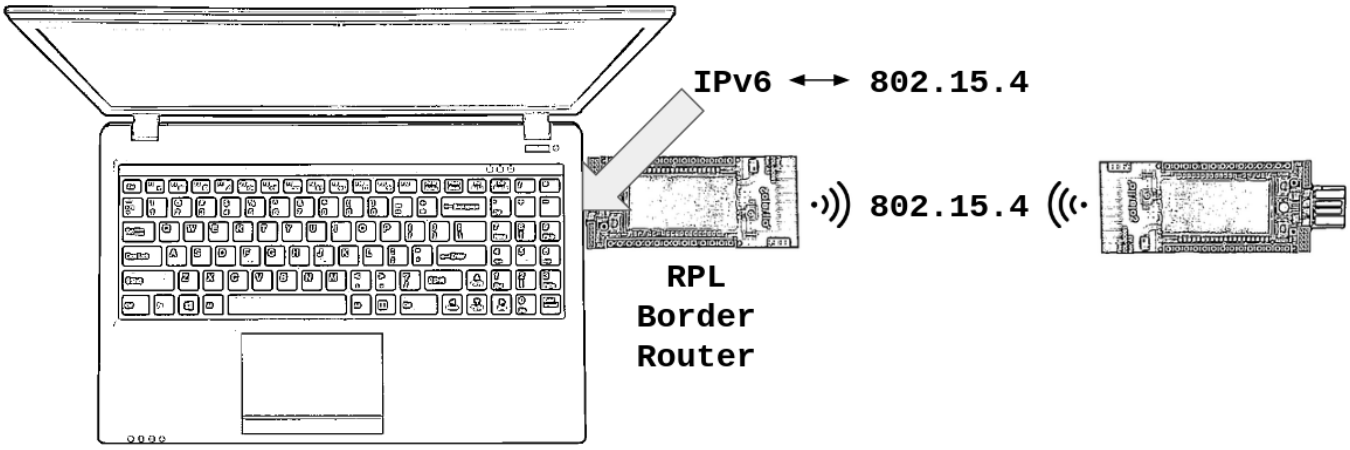


FIGURE 4: Communication with border router

Programs for Contiki are written in C and with the forked repository from the main Contiki-NG repository OSCORE is provided. Based on the example files in it a server has been made. The server has the values for the OSCORE security context, which have to be matched by the communicating client. It imports the resources it uses, sets the paths to them and enables oscore support. The resources are provided in .c files. They contain the data to send but also the functionality with which it can be handled. In this case the

main functionality is provided through a resource get handler as get it is the REST request used.

Because of memory constraints and memory allocation, macros are used to avoid dynamic memory allocation. Through the use of macros in the code, which are part of the pre-compilation of the code, values can be set and imports can be handled prior to compilation. Part of the macros are also header-guards that limit the repeated inclusion of the same files as well as what code is included in compilation depending on what resources have been provided, such as the devices that have been attached and inclusion of the necessary files for OSCORE.

One of the constraints set on communication over CoAP is a block size of 64 bytes. Any resource larger than that therefore has to be sent in chunks. In the functionality of an observable resource the get handler is accompanied by a resource periodic handler that provides the functionality for sending notifications triggering the get request at the client side.

The implementation for this was made in stages. Firstly testing the files in their original format and then as they were modified. Just like the Python implementation a .well-known/core resource is provided to provide the resource tree.

The primary configuration was having the server, with the added resources, on one IoT device (Zolertia Firefly), with the other running the rpl-border-router program to provide a reachable IPv6 address to the server. The client was run in the same docker Contiki docker image as the border router. At first that client was the one part of the aiocoap library used for the Python implementation. It could be used for sending CoAP requests but when simultaneously analyzing the network traffic data it was seen that it also sent CoAP requests when the OSCORE option was chosen, this did not happen when using it outside the Contiki docker image. As a second approach the second IoT device ran a client program instead serving as a border router, instead relying on the link-local address. That approach made it so that the network traffic could not be analyzed and as such it could not be verified whether or not packets were OSCORE protected or not. Functionality on the server side that should have rejected CoAP requests to an OSCORE protected resource additionally did not do that and an OSCORE protected resource was sent regardless. Instead the best solution was changing the client in the same set-up as the first but instead of using the one provided with the aiocoap library, taking a Java client based on the Californium library. With this approach the network traffic could be analyzed and requests could be verified to have been made using OSCORE.

When implementing a resource for IoCs, on `test/ioc_obs`, however several issues became apparent. If a resource was larger than 64 bytes it had to be sent blockwise. If OSCORE was used with it as well however then that limit was lowered to 62 because of two bytes being used for OSCORE.

To continually test this as the implementation progressed different methods were used. In the implementation of the resource

IX. Results

i. Testing

There are several aspects to take into account for testing. One is the objectives which are set. The testing should be conducted in such a way that it tests against them. Second is what possibilities for testing that exists. Third is ensuring that the tests are conducted in such a way that they are replicable and that the testing is as fair as possible.

The main objective is to have a system that can enable cyber threat intelligence sharing for resource constrained IoT, but in an effort to give perspective on the system and the design choices, other tests can highlight features, choices and compare to other solutions.

Because of the proposed integration into MISP, both a Python as well as a C implementation of a server using CoAP and OSCORE exists. From this it naturally follows to base the testing on this. The Python implementation can be run on an enterprise computer and can therefore be compared to other alternatives. There are no alternative implementations that can be run on the hardware and as such that implementation will be used to test the main objective of the system, sharing CTI from a resource constrained device.

For the Python implementation both tinySTIX and CoAP with OSCORE are evaluated. For the testing on hardware only CoAP with OSCORE is evaluated as the data is expected to already be generated in tinySTIX format and as such only has to be sent.

1) Choosing data

Many data fields in STIX have open vocabularies, a set of values that can be extended, or enumerated sets, value sets which cannot be extended. Most of the fields however, do not have values.

Of the 197 properties, 37 have an open vocabulary or enumerated set associated with it. Some of these properties still have values that will have set format such as UUID and time but most have free text values. That means the size of the value can vary drastically. Fields such as name can have one word while fields such as description can have texts in pages.

Free text values represent a challenge for tinySTIX conversion as these cannot be converted to integer values and even with CBOR encoding message sizes remain large. This is especially the case of values that account for a large part of the overall message size. As such the worst case for tinySTIX conversion reduction rates is the one in which all fields have free-text values, in which case the reduction would be the same as if only keys would have been converted to integers prior to CBOR encoding. These values approach the limit which is only CBOR encoding when values increase in proportion of size to the keys. This is tested later on for reference.

The tested data can therefore be split into two parts. One being regular CTI from a threat intelligence feed such as MISP, Circl.lu, botvrij.eu or other. The data there is general and not specific for IoT devices, nor is it consistent as some messages can have small message

sizes counting in kB where other is MB. The latter most often being because of large free text fields. The message size reduction after tinySTIX conversion therefore varies greatly.

The second data set can be generated by modifying current CTI to better represent what could be generated by a resource constrained IoT device. While there is research into CTI for IoT devices there are currently no open CTI feeds for such data. There does exist data feeds that include CTI from IoT devices, or honeypots posing as IoT devices. No such data had been possible to access and as such, data from common open CTI feeds has been tailored to better represent what could be generated from an IoT device.

Since the data will automatically be generated, for example by an IDS residing on the IoT device or a log file on it, free text fields such as description can largely be ignored. The IoT device itself will only generate small pieces of information and the connections that make it intelligence happens only on the enterprise side. As such the data can be kept small as relations to other data is avoided. It is however, worth considering this, in the case that resource constrained IoT devices would be considered as consumers of CTI. Challenges for this would include information loss in the case that fields such as description are removed and the challenge of having large message sizes due to the relations between different STIX objects, making it so not just one object is sent but potentially many of them.

2) Representative data

The data on which the tests are performed is taken from Botvrij, circl.lu (the Computer Incident Response Center Luxembourg), Urlhaus, and Threatfox, all open threat intelligence feeds.

- https://botvrij.eu/data/feed-osint/
- https://circl.lu/doc/misp/feed-osint/
- https://urlhaus.abuse.ch/downloads/misp/
- https://threatfox.abuse.ch/downloads/misp/

The feeds are chosen because they are open access and the data is in MISP core format, a data format from the MISP Project that is based on JSON and that is used to exchange attributes and events. Furthermore they are some of the default data feeds that feed into the MISP platform. As such the data is part of the platform in which this work hopefully can be integrated.

To better represent the variation in data that exists in these feeds, four of them where chosen, and from each of them three CTIs where used. Because the content of the CTI can influence the results the most recent three CTIs from each feed where chosen, in order to ensure no bias was part of the selection of data. These CTI vary drastically in size and different feeds had CTI in orders of magnitude different from each other. All CTIs did however have the same structure of having three fields: type, id, and objects.

```
  {
"type": "bundle",
```

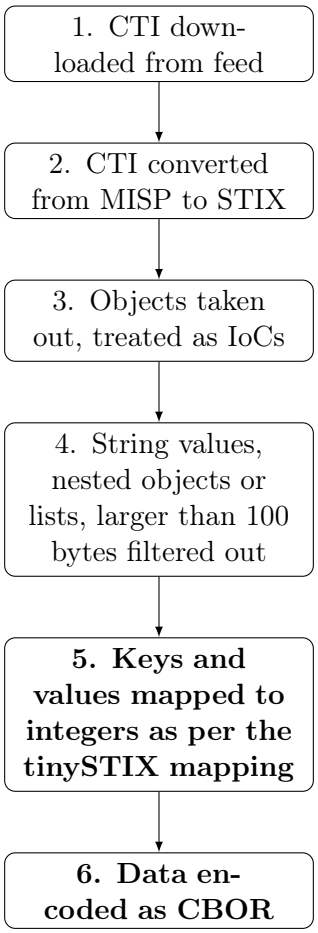
```
"id": "bundle--02a470d8-493e-41d9-8367-622460dddbe8",
"objects": [
  {
    "type": "identity",
    "spec_version": "2.1",
  ...},
  ...]}
```

The data from the threat intelligence feed is then converted to STIX using the MISP-STIX-Converter.

```
misp_stix_converter export --version 2.1 -f original_data/*
--output_dir stix_data/
```

The circl.lu feed posts OSINT, Open Source Intelligence. Hence data varies largely in size and content. Generally the data provided through a feed such as circl.lu is cyber threat intelligence, data which is contextualized, correlated and actionable. Because of that it is larger and commonly other objects are listed as part of one cyber threat intelligence unit in order to understand relationships between data. This is not the kind of data that is expected to be generated from an IoT device. That data is supposed to be incidents or alerts from a host-based IDS. It is only in the threat intelligence platform that it is put together with other data to form cyber threat intelligence. Consequently, in order to take data from an OSINT feed, cyber threat intelligence objects have to be taken apart to form pieces more representative of its constituents. The first part of this process is taking apart nested and related objects. These are split into separate objects instead of being listed as part of one. The second step is removing data that would not have been added had this data been generated by an IDS. This includes values such as description, value and other free text fields. OSINT can have many origins and some of it is written by cyber security professionals. Trying to remove such data and other fields that might have been mapped to keys simply because another data standard has a different key set, is another part of the process to make the data more realistic to the set of automated data sharing from IoT. While some fields, such as description, could be immediately omitted, other fields are harder to filter out.

To simplify the process of adapting the data, all values that were larger than 100 bytes were filtered out. Admittedly this means some fields which could have been generated by an IoT could be filtered out it also keeps fields that are less than 100 bytes that would not have been generated by it. It has also been verified that filtering out values larger than 100 bytes does not remove fields that most likely will be part of the data generated, such as UUID and timestamps. By taking multiple objects from circl.lu, which in turn contains several objects, these differences between data contents make the individual data differences less prominent.



3) Message size calculation

The size of the STIX message is calculated with Python code. The size of the STIX message is calculated by taking the message as a string and converting it to the UTF-8 charset after which the length of the message is taken with the len function. Similarly the size of the tinySTIX data which is in CBOR is calculated by applying the len function of it. The data size was confirmed with the CBOR online calculator at cbor.me and the word count tool in Linux (wc).

4) Evaluation tinySTIX conversion

TABLE 2: Data size reduction in conversion from STIX to tinySTIX

File Name	# Obj.	Min. Red. (%)	Max. Red. (%)	Avg. Red. (%)
botvrij/...be8.txt	16	29.11	46.96	35.77
botvrij/...a8d.txt	442	23.35	46.96	35.87
botvrij/...b24.txt	26	29.49	46.96	36.20
circl.lu/...c79.txt	140	23.59	47.26	33.41
circl.lu/...57b.txt	24	27.34	47.26	32.60

Continued on next page

Table 2 – continued from previous page

File Name	# Obj.	Min. Red. (%)	Max. Red. (%)	Avg. Red. (%)
circl.lu/...e45.txt	275	27.93	47.26	36.21
urlhaus/...2bc.txt	15.6k	33.04	46.55	38.69
urlhaus/...843.txt	6.8k	33.43	46.55	39.80
urlhaus/...f4a.txt	10.5k	33.96	46.55	38.27
threatfox/...d0d.txt	3.8k	31.51	46.55	32.67
threatfox/...07c.txt	263	31.51	46.55	35.51
threatfox/...d26.txt	207	31.91	46.55	34.75

By the time of conversion from STIX to tinySTIX the CTI taken from the four data feeds has been processed in the four first steps. The conversion to tinySTIX is therefore the remaining steps, 5 and 6. The first being changing keywords and known values, either enums or open vocabulary in the STIX 2.1 standard, to integers, as per the tinySTIX mapping that has been created. The second step is changing the encoding of the data to CBOR. CBOR is a more compact data format than JSON that is used for STIX.

Each file (one CTI) consists of many objects; for these files, between 16 and 15679. Each of these objects correspond in size to an IoC, indicator of compromise. When converting from STIX to tinySTIX, each of the IoCs is converted individually. For each IoC of each file the difference in size between the original data, in STIX format, and the resulting data, in tinySTIX format, is recorded in a list. The list is then sorted and the smallest, largest, and average reduction in size is taken out. When presented in the table above, 2, each row is the data based on each file. It can be noted that while the percentages for many of the files are the same, the data is in fact different. Instead they have the same structure but with differing values which is why the conversion rate is the same.

In total, 38233 IoCs are converted and analyzed. Of these the smallest size reduction is 23,35% and the greatest 47,26%. The average size reduction is around 35%. The tinySTIX conversion performs reliably over the different data sets, with the smallest reductions still being significant. Over all IoCs the average reduction is 35.13%.

Mapping examples

To illustrate the steps in which the conversion is made, the best and worst performing IoCs in the tinySTIX conversion have been taken from the file [...]c79.txt in circl.lu.

```
1. Original IoC
{
   "type": "identity",
   "spec_version": "2.1",
   "id": "identity--55f6ea5e-2c60-
   40e5-964f-47a8950d210f",
   "created": "2020-12-07T06:54:35.000Z",
   "modified": "2020-12-07T06:54:35.000Z",
   "name": "CIRCL",
   "identity_class": "organization"
}

Mapped with tinySTIX
{
   2: 21,
   3: "2.1",
   8: "identity--55f6ea5e-2c60-
   40e5-964f-47a8950d210f",
   16: "2022-01-28T11:13:31.000Z",
   15: "2022-01-28T11:13:31.000Z",
   14: "CIRCL",
   146: 4
}

Converted to CBOR
0b10100111 0b10 0b10101 0b11 ...

Reduction: 23,59%
```

```
2. Original IoC
{
   "type": "marking-definition",
   "spec_version": "2.1",
   "id": "marking-definition--613f2e26
   -407d-48c7-9eca-b8e91df99dc9",
   "created": "2017-01-20T00:00:00.000Z",
   "definition_type": "tlp",
   "name": "TLP:WHITE",
   "definition": {
      "tlp": "white"
   }
}

Mapped with tinySTIX
{
   2: 73,
   3: "2.1",
   8: "marking-definition--613f2e26
   -407d-48c7-9eca-b8e91df99dc9",
   16: "2017-01-20T00:00:00.000Z",
   196: "tlp",
   14: "TLP:WHITE",
   195: {
      "tlp": "white"
   }
}

Converted to CBOR
0b10100111 0b10 0b11000 0b1001001 ...

Reduction: 47,26%
```

Because the tinySTIX mapping to integers is dependent on the data that is converted it is worth considering the worst case for data size reduction in the conversion from STIX to tinySTIX. The worst case is when the value/values is a large part of the total data size, and the value is not in the tinySTIX mapping. Such is the case when the value is a free-text value. This could be fields such as description or tags. In this case the mapping of the keyword to an integer has negligible impact and because the value is free-text it cannot be mapped to an integer. In this case the difference in size between the mapped and original data will approach 0 as the size of the value increases.

$\lim_{x\to\infty}(x - f(x)) = 0$, where f is the mapping of keys and values to integers and x is the input data

The performance in the worst case is therefore the same as only changing the data encoding to CBOR. This is presented in table below. From this we can see that with some variation the lower end reduction in the worst case is around 12%.

File Name	# Obj.	Min. Red. (%)	Max. Red. (%)	Avg. Red. (%)
botvrij/...be8.txt	16	11.54	20.00	14.45
botvrij/...b24.txt	26	12.50	18.14	14.47
botvrij/...a8d.txt	442	10.66	20.37	14.55
circl.lu/...c79.txt	140	10.77	18.14	13.52
circl.lu/...e45.txt	275	11.62	18.14	14.60
circl.lu/...57b.txt	24	10.21	18.14	13.95
urlhaus/...2bc.txt	15.6k	12.45	18.14	15.56
urlhaus/...843.txt	6.8k	12.45	18.14	15.36
urlhaus/...f4a.txt	10.5k	12.45	18.14	15.23
threatfox/...d0d.txt	3.8k	12.40	18.14	13.12
threatfox/...07c.txt	263	12.40	18.14	13.79
threatfox/...d26.txt	207	12.40	18.14	13.68

TABLE 3: Data size reduction, only CBOR encoding

The average over all objects is 15,15% when using only CBOR, and as stated before 35.13% when using the full tinySTIX conversion. Since it is the same data as before, the tinySTIX mapping can be calculated to add on average 20%-units in reduction.

ii. Message size in transit

The message, payload, is not only what is sent on the wire, additional headers to package the data and route it correctly are sent too. The data headers are based on the used network stack, and for each layer additional data is added in order to describe destination, options, data format et cetera. By changing the protocols used, the header lengths can be reduced.

To see the actual communicated data size Wireshark has been used. Data was captured on loopback to capture the data that was sent on localhost (IPv6 address [::1]) using capture filter `src host ::1`.

	CoAP	HTTP†	TAXII‡
Payload	259B	315B	955B
Application layer	CoAP: 8B	HTTP: 148B	HTTP: 320B
Transport layer	UDP: 8B	TCP: 96B (32B)	TCP: 64B (32B)
Internet layer	IPv6: 40B	IPv6: 120B (40B)	IPv6: 80B (40B)
Link layer	Ethernet: 14B	Ethernet: 42B (14B)	Ethernet: 28B (14B)
Total	**329B**	**721B**	**1447B**

TABLE 4: Data from a sample of STIX default data feeds in MISP data format, data size 259B. Sizes in bytes. Size per frame in parentheses.
†Three frames sent ‡Two frames sent

iii. Benchmarking

A reduction in message size does not only impact the content size of the message, but if the message size is large enough it also impacts over how many messages the content has to be sent. A reduction in message size can therefore reduce the number of packets being sent and therefore also sending less content headers.

When evaluating the performance of CoAP with OSCORE and comparing it to OpenTAXII and HTTPS, two values are of difference. Total size of the session and the number of packets in it. The size of the session is of interest as it represents the size of the data that the IoT device would have to send. It takes into account all the messages to set up and close down a session as well as messages to send the data as well as acknowledgements and confirmations. It is desired to keep the session size as small as possible while retaining the desired functionality, and as such the over all session size is an indication of it. The second feature, namely number of packets in a session, is of interest as it is the number of messages that is sent. With that conclusions can be drawn on whether the best approach for reducing the session size is through reducing the number of packets or the size of each packet. Knowing the number of packets also gives the possibility to calculate possible packet size reductions through further optimization, such as by changing from IPv6 to 6LowPAN.

File Name	# Obj.	OpenTAXII	HTTPS	OSCORE	-% OT†	-% H‡
botvrij/...be8.txt	16	238	255	34	85.71	86.27
botvrij/...a8d.txt	442	6202	6633	886	85.72	86.63
botvrij/...b24.txt	26	378	405	54	85.71	86.67
circl.lu/...c79.txt	140	1974	2114	282	85.71	86.66
circl.lu/...57b.txt	24	350	375	50	85.71	86.67
circl.lu/...e45.txt	275	3864	4138	552	85.71	86.66
urlhaus/...2bc.txt	15.6k	219521	234870	31360	85.71	86.65
urlhaus/...843.txt	6.8k	95284	101756	13612	85.72	86.62
urlhaus/...f4a.txt	10.5k	147602	157927	21086	85.71	86.65
threatfox/...d0d.txt	3.8k	53410	57173	7630	85.72	86.65
threatfox/...07c.txt	263	3696	3957	528	85.71	86.67
threatfox/...d26.txt	207	2929	3116	416	85.79	86.65

TABLE 5: Number of packets sent over different implementations
†OSCORE smaller than OpenTAXII [%]
‡OSCORE smaller than HTTPS [%]

File Name	# Obj.	OpenTAXII	HTTPS	OSCORE	-% H‡	-% H‡
botvrij/...be8.txt	16	57838	90216	7496	87.03	91.69
botvrij/...a8d.txt	442	1543111	2373162	218599	85.83	90.80
botvrij/...b24.txt	26	92906	144010	12628	86.41	91.23

Continued on next page

Table 6 – continued from previous page

File Name	# Obj.	OpenTAXII	HTTPS	OSCORE	-% OT†	-% H‡
circl.lu/...c79.txt	140	485714	752047	66028	86.41	91.23
circl.lu/...57b.txt	24	85917	133411	11761	86.31	91.19
circl.lu/...e45.txt	275	967254	1484305	141461	85.37	90.47
urlhaus/...2bc.txt	15.6k	53874412	83301867	7032480	86.94	91.56
urlhaus/...843.txt	6.8k	23223010	35998482	2907278	87.47	91.92
urlhaus/...f4a.txt	10.5k	35971003	55807667	4525687	87.42	91.89
threatfox/...d0d.txt	3.8k	13777021	20904787	2345469	82.99	88.77
threatfox/...07c.txt	263	931282	1426794	142429	84.71	90.01
threatfox/...d26.txt	207	739516	1128130	116346	84.27	89.68

TABLE 6: Size of sessions sent over different implementations
†OSCORE smaller than OpenTAXII [%]
‡OSCORE smaller than HTTPS [%]

HTTP/HTTPS Sending a single IoC over different implementations.

	HTTP	HTTPS	OSCORE	TAXII
Packets	28	30	4	28
Total size	3237	10205	473	6135

TABLE 7: Data from sending a single IoC over different implementations. Size in bytes.

Because the TAXII server could not be tested with HTTPS, HTTP and HTTPS have been tested separately so that the overhead of adding HTTPS could be estimated.

For comparison to the implemented communication method a regular TAXII client has been used, in this case OpenTAXII from EclecticIQ for the TAXII server and Cabby as the TAXII client.

iv. Hardware testing

For the IoT device, in this case a Zolertia Firefly, the objective of the testing is to validate the proposed model for CTI sharing and its implementation. The tests are therefore ones that answer what could not been answered in the testing of the Python implementation.

The first objective is to see if the proposed model is possible to implement. The second objective is to analyze the memory capacity and to what extent it limits the application and possibly the tests. The final objective is to perform energy measurements on the device to see how sending IoCs affect the energy consumption of the device.

The validation of the implementation already happened in the implementation phase where it was concluded that not all objectives for sending observable IoCs over OSCORE could be met. OSCORE was not possible to make compatible with neither observable resources or resources that had to be sent blockwise, as in the case of sending IoCs. Consequently the

testing is adapted to get as much information about such an implementation even though it could not be made as intended.

It is here worth reminding of the use case for which these IoT will be used. The main purpose of the IoT device is to carry out the task for which it was acquired. That purpose could be to send temperature data, for example. As a side task it will be monitoring the activity to or on the device as part of a logging system, intrusion detection system or other. It is this system that will generate the IoC that is then sent over CoAP with OSCORE. It can be expected that the system generating the IoC will do so in tinySTIX format. The testing therefore commences in the stage where an IoC has been generated and is ready to be sent off.

1) Results from implementation

To test this IoCs in tinySTIX format where hard-coded to the resource for the server and then sent from it. Because block-wise communication was not possible, each IoC was split into blocks of a maximum of 62 bytes each. This is because the block size is set to 64 bytes and 2 bytes are used for OSCORE, leaving 62 bytes of each packet to be used for payload. When using only one (observable) resource on the server, the IoCs split in blocks could use a maximum of 765 bytes before the stack overflowed the FRSRAM on the device. While all IoCs that have previously been tested are less than that, it is not enough to fit more than one at a time of the larger ones. Of the IoCs in tinySTIX format, the largest being 489 bytes, with many being above 400 (8672 IoCs). Importantly, if the IoCs would not have been converted to tinySTIX format the largest IoC would have been 714 bytes and a total of 87 IoCs would have been more than 700 bytes.

While the implementation itself gave insights to the constraints and limitations of the device and support for it, energy testing is required to also determine whether or not it is feasible to send IoCs from such a device.

2) Energy testing

Measuring energy consumption is only possible on a device on which other processes can be controlled. This is because background processes otherwise impact the power consumption and what part is caused by the service of interest cannot be extrapolated. It is therefore only possible to measure on the Zolertia Firefly and not with the Python implementation that too has been created.

Joulescope To measure the energy expenditure the Joulescope DC energy analyzer was used. It has an USB-A port to which the Firefly device can be attached and the energy consumption from it is displayed in real-time to the Joulescope GUI, as shown in 6. It is very precise because of the high sampling rate, two million times per second, and it can measure in 10 orders of magnitude, from nanoamps to amps.

Experimental set up To test the energy consumption the Joulescope energy analyzer was attached to a wall outlet and by USB to the computer to which the data was fed to the Joulescope interface. From the Joulescope USB outlet the OSCORE server, which was to be monitored, was attached. Directly to the computer was also the second Zolertia Firefly

device acting as a border router and providing a IPv6 address from which the service could be accessed. On the computer the Contiki docker image was running from which the address could be accessed. The client run in the docker image was one based on the Java library Californium which provides CoAP and OSCORE functionality. In the same docker image tshark (a command-line interface of Wireshark) was also run on the tun0 interface that the border router supplies.

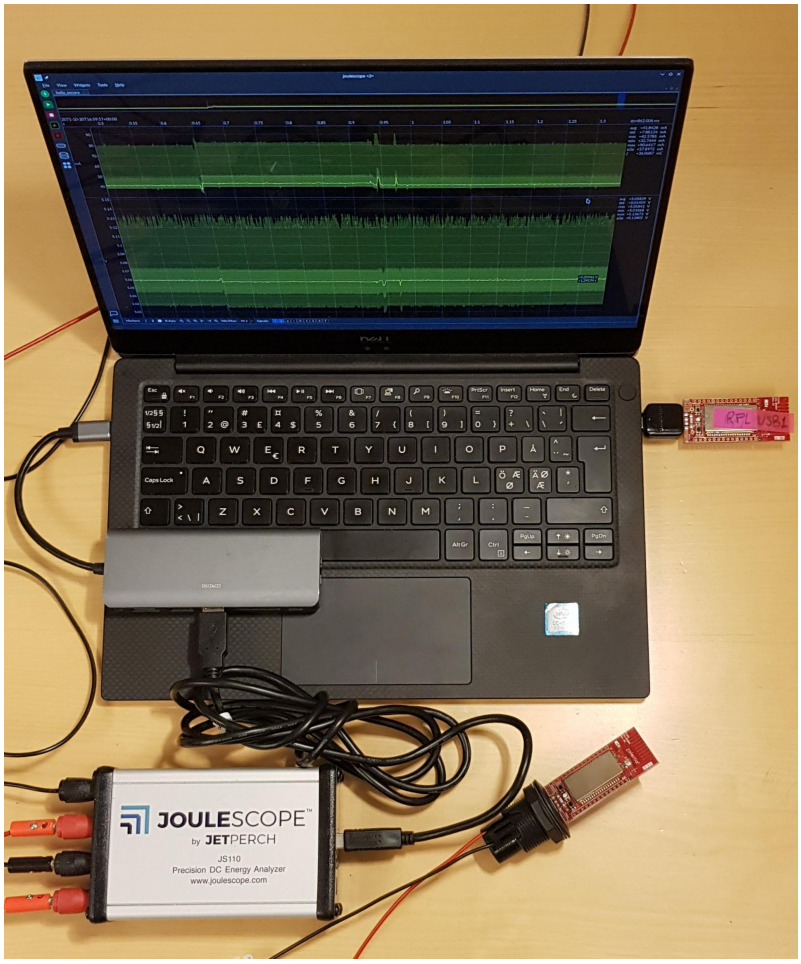


FIGURE 5: Experimental set-up.
Border router is the Zolertia Firefly on the right.
Joulescope with attached Zolertia Firefly running the server on the bottom.
Joulescope graphical user interface on the computer.

Testing Energy consumption tests were done for two resources on the server `test/hello` and `ioc_obs`. The first one is an OSCORE protected resource without observe or blockwise transfer, it is part of the example resources provided and it sends 48 bytes of data. The second is the added resource sending IoCs. It is an observable resource but does not send the data through the CoAP blockwise transfer functionality. Instead the data has manually been split into blocks of a maximum of 62 bytes prior, due to OSCORE not being compatible with blockwise transfer. OSCORE also was not compatible with observable resources and as such the tests the following:

- `test/hello`: CoAP single request, 42 byte payload

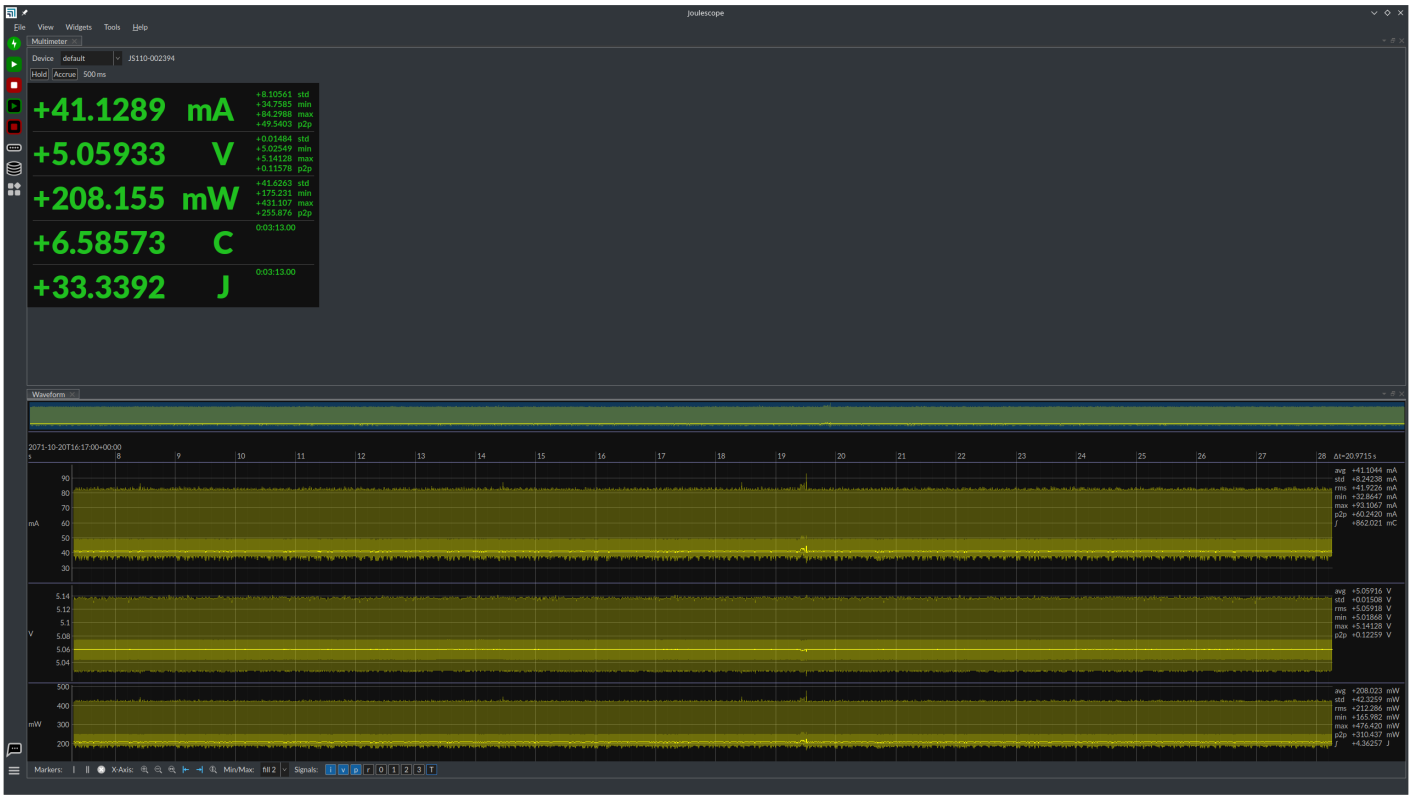


FIGURE 6: Joulescope GUI

- `test/hello`: OSCORE single request, 42 byte payload
- `test/ioc_obs`: CoAP observable request, 7 responses, $\leq$ 62 byte payload

The two tests on `test/hello` are to compare CoAP with OSCORE on the same resource and the test on `test/ioc_obs` is to see the energy consumption for observable resources. Furthermore an important observation for all is the baseline energy consumption from running the server and functions needed for it such as radio. `.well-known/core`, which is a standard resource for CoAP that has the URI tree, also exists as a resource but will not be used often and is therefore not tested.

The baseline energy consumption in all tests was, as expected, the same, around 41 milliampere (mA).

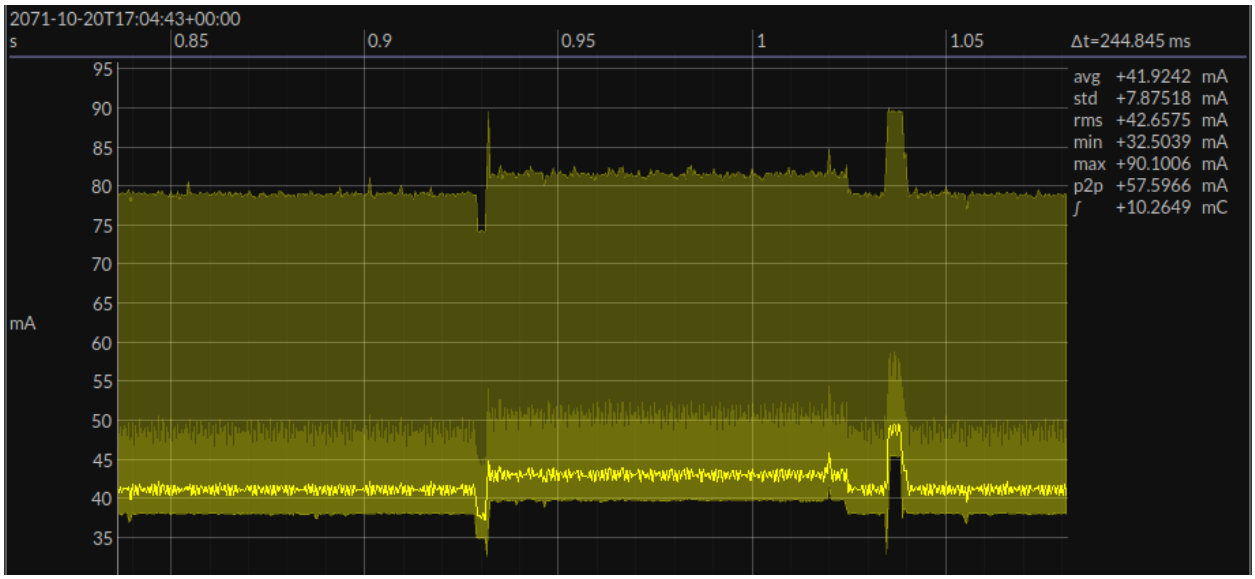


FIGURE 7: test/hello with CoAP

When comparing the two tests on the same resource with and without OSCORE and several similarities can be seen. One is that the instantaneous energy consumption is about the same, around 43 mA before spiking to 49 mA in the end. The duration differs between the two however. When analyzing the traffic in Wireshark the packet size is 8 bytes larger for OSCORE (128 compared to 110). That is a small difference in comparison to the almost three times as long time difference between the two, 107 ms for CoAP and 298 ms for OSCORE measured from the first dip to the end of the first spike. When looking at it more in detail however it can be seen that for the first, and long, part in sending a message there is a large difference in time between CoAP and OSCORE, 93 ms for the former and 287 ms for the latter, a threefold increase in time. For the second part, a much shorter increase in energy consumption, less of a difference can be seen, 3,45 ms and 3,9 ms for CoAP and OSCORE respectively. When also looking at the size of the two packets, the one sent over CoAP is 110 bytes and the one over OSCORE 128. The difference due to the added OSCORE headers. It is reasonable to assume that the second part consequently is the actual data transmission. Dividing the number of bytes by the time to send them and then comparing to the same for OSCORE and it is approximately the same.

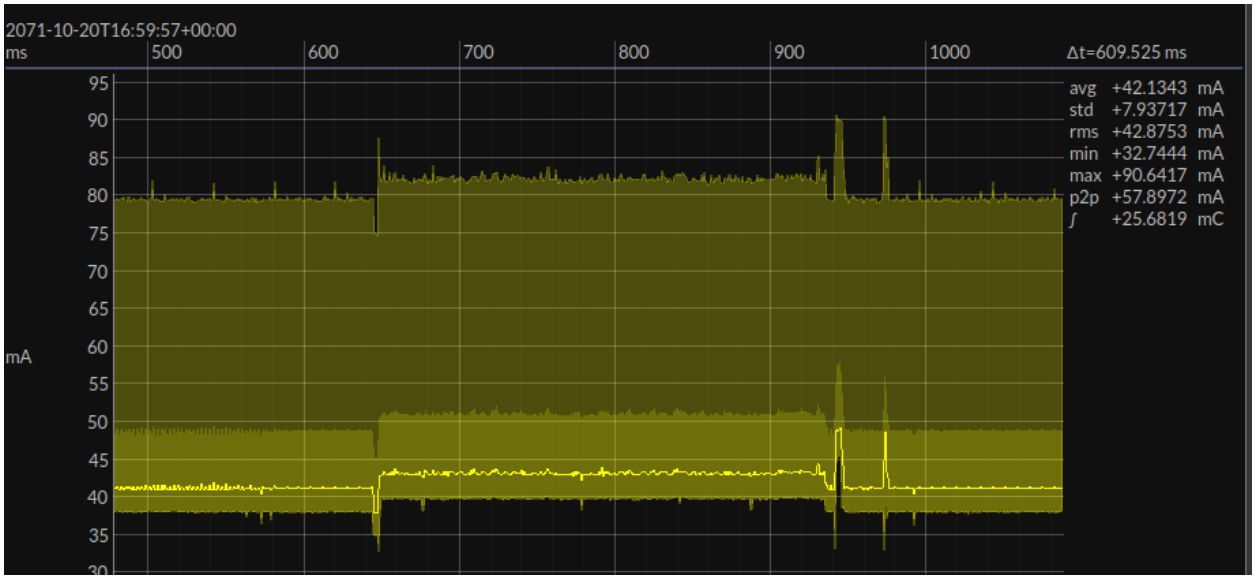


FIGURE 8: test/hello with OSCORE

$$110[bytes] \div 3.45[ms] \approx 31,9[bytes/ms]$$
$$31,9[bytes/ms] \times 3.9[ms] \approx 124,4[bytes] \approx 128[bytes]$$

Because of the small byte overhead of sending packets with OSCORE instead of CoAP the main penalty occurs in the prior stage, presumably where the encryption happens.

Based on these numbers it is possible to do calculations to what sending an IoC would cost, had it been sent over OSCORE.

A packet with 62 bytes payload would be 142 bytes with the OSCORE payload, take 320 ms for the initial phase and 4,4 ms for the transmitting phase.

With IoCs ranging from ∼120 bytes to 480 bytes that means sending 2 to 8 packets of 62 bytes each. Assuming the time between the initial and transmission phase is in the range of the 5-10 ms, sending one packet of 62 bytes over OSCORE requires ∼ 14 mC. Sending a full IoC would therefore require an energy consumption in the range 28-112 mC. In doing these calculations it is important to highlight that this is the cost given no other process would be running otherwise. In comparison to the baseline energy consumption and the initial phase increases energy consumption by 2 mA, a 5% increase, for 640-2560 ms, and the transmitting phase increases energy consumption by 8 mA, a 20% increase, for 9-35 ms. Combining them and sending one packet of 62 bytes gives a +5% penalty to the baseline energy consumption for a duration of 325 ms, for an IoC this would instead be for 650-2600 ms.

Sending one or multiple packets from one resource with observe should not affect the energy consumption, something that is confirmed in the testing. All sent packets follow the same pattern in energy consumption as the `test/hello` resource as when requested without OSCORE.

FIGURE 9: test/hello with OSCORE

X. Discussion

In the case of implementing the proposed systems several challenges appeared. That IoT security has been of low importance, or at least of less importance than enterprise security, is apparent, and many repositories are maintained by individuals or small groups of people. DTLS trails TLS and it is far from the only case in which IoT security is an adapted version of an enterprise protocol or system. This work, too, is of that kind. That is not necessarily bad, many of the innovations and lessons learned from enterprise security can be adapted for IoT. There are also several parts that cannot. One of the challenges with the development has been trying not to just miniaturize STIX and the TAXII protocol but to make versions that are STIX and TAXII compatible but still adapted to the IoT use case. This work is a continuation of that but there is still much work needed for this system to be fully useful. It is therefore of interest to revisit the use case in which this system is expected to be.

i. On implementation

As previously mentioned several of the repositories that have been used in this project are maintained by one individual or a few contributors. This means that some parts that were of interest to use were untested or incomplete. This especially turned out to be the case with OSCORE. An example of this is the aiocoap library.

The aiocoap library is a Python library that provides communication over CoAP and optionally OSCORE protected communication. It also provides observable resources, that

too can be OSCORE protected. OSCORE, is however in implementation made as a wrapper around a regular site, meaning that resources that are public as part of a server are available both through regular communication over CoAP and over CoAP with OSCORE. It is not possible to only have resources available with the right OSCORE security context as the resources will also be available without it. In a real environment this is of course not desirable. There is no need for secure communication if the same resources can be accessed without it. The implementation therefore works in the purpose of testing but cannot be used in cases where message confidentiality is a necessity. For testing purposes this suffices but for an implementation intended for real world use, modifications would have to be made. Workarounds should be possible however and a message request that does not come with an OSCORE security context could be ignored or appropriately handled.

The library has plugtest resources, from which this implementation has been created. The plugtest tests however are not available and the testing that it is supposed to provide has not been possible to perform. From the comments in the code alone there seems to be unfinished functionality when it comes to large resources but this has not been an issue during the testing for this work. It does however add to the work that has to be performed in order to take this into a real environment, as these functionalities have to be thoroughly tested.

The libraries used for the tinySTIX to STIX translator however, did not seem to have these issues. Most likely because more basic functionality was used and the libraries themselves have more usage and support. The STIX standard however did not have lists that could be found of the keywords that are used, nor the open-vocabularies or enumerations. Because of that they instead had to be extracted from the STIX 2.1 HTML code with a Python script using the web scraping library beautifulsoup. The risk with this approach is of course that keys or values could have been missed, but because any key or value that does not appear in these two generated lists are kept in original form, no data is lost because of it.

While this was a design decision in choosing to keep the tinySTIX mapping close to the original, it could also have been a design choice in making the tinySTIX to STIX mappings specifically adapted to resource constrained IoT, simply by removing keys and values that are not useful in the case for IoT. Similarly keys and values that are used often that are not in the STIX 2.1 standard could be added and in the translation to STIX be converted to free text values. By doing this some values could be omitted from the mapping, thereby reducing the size of the mapping file itself, and possibly some values could be one byte in CBOR instead of two.

In order to test CoAP with OSCORE on a resource constrained device an operating system compatible with such a device is needed. Contiki is one such option. It provides CoAP functionality along with much more. It does, however, not provide OSCORE. That can instead be found in a fork from the Contiki-NG repository. Contiki is preferrably run in a docker container and using makefiles the program which should be run can be compiled to different targets. For the Zolertia Firefly board used in this project the target is Zoul but the programs should also be possible to compiled to run natively. This is not possible as it gives errors on missing a file, an endian.h. It is currently 1130 commits behind the development branch of the main contiki-ng repository, something that could be an issue if later additions or bug fixes are needed.

It did however work to compile to target Zoul and as such it could still be tested. Additional problems arose however when OSCORE was not compliant with either blockwise transfer nor observable resources. It is not the case of incompatibility but simply that the functionality for it was missing in the library. The tests could therefore not be run with the intended functionality but by testing them separately a combination of them could be reasoned about.

ii. On results

From the testing on the Python implementation results were given on tinySTIX conversion and how CoAP with OSCORE compares to the reference implementations. In testing conversion of IoCs in STIX to tinySTIX minimum reductions of 23,35% and maximum of 47,26% were recorded. The average reduction over all tested IoCs was 35,13%. It is a significant reduction in size that is not just desired to lower the energy consumption in sending it, it is also needed to save space as the implementation on the reference hardware revealed. Memory is a limiting factor on resource constrained IoT and for the reference hardware the maximum size for a resource was 765 bytes. Given that sharing IoCs should not be the main task of the IoT device it is crucial that the implementation and data takes as little space as possible. This also confirms one of the requirements that was set when developing an alternative for TAXII for resource constrained IoT, namely quick to immediate sending of the data once it has been generated, so that it is not kept on the device. A model in which the IoT device is in charge of when to send the data is important, as in the case of observable resources as proposed in this work.

An initial idea was for new connecting clients to the server (IoT device), could request the last IoC that had been sent out. A limit that could have been set to 5, 10 or something else. That is not possible to implement currently as two large IoCs would overflow the FRSRAM of the device.

Here it is important to note that it was a choice not to modify the underlying implementation, and that only other resources were removed. The intention of sending IoCs is that this would be a secondary function of the device with the main one being the one which it was installed for, sending measurements of temperature for example. Modifying the device in such a way that it would optimize for sending IoCs would therefore be misleading as that is not the main function of the device. An example of this is is the FRSRAM. When trying to write more than two IoCs the stack would overflow the buffer for the FRSRAM and it would not be possible to write the program to the device. The FRSRAM as it turns out, while being 32 kB, is split in two. One part is the one to which the program with the added IoCs is written and the second part is for low power and sleep modes. This is because one part is full retention and one is not. It is possible to deactivate low power mode and consequently get access to the full RAM. Because low power mode is desired and might also be so for the main function of the device, this has not been altered, even though that would have allowed for many more IoCs to fit and be tested. With the same reasoning other features that might have saved energy, increased performance or memory have not been activated or disabled.

iii. Reevaluating OSCORE

The issues explained above lends the question to whether or not OSCORE is mature enough to be used in a production setting, especially considering DTLS exists as an alternative that too provides message confidentiality and has more adoption. Providing message confidentiality with DTLS comes with slightly higher costs in terms of data size and the encryption of data headers remove the possibility of end-to-end security over proxy. Given the maturity of OSCORE in implementations for both Python and Contiki-NG, DTLS seems a better choice to provide message confidentiality.

iv. Adding header compression

Because of the large number of IoT devices, IPv6 with its larger address space is predominantly used. This, however, means that instead of a header size of 20 bytes as with IPv4, 40 bytes are instead used. Because of the need of a larger address space while still keeping packet sizes small, a reduced way of handling IPv6 was created, 6LoWPAN. With the results from the OSCORE sessions with both the number of packets and the total size it is possible to calculate the percentage that each part of the data headers take. IPv6 headers can be calculated to be around 18% of the total data of a session, and are therefore a significant part of all data that is sent. Adding header compression through the use of 6LoWPAN can therefore bring additional data savings to the implementation. It is here worth noting that 6LoWPAN headers are depending on the use case and are not a fixed size. Instead it ranges from 2-20 bytes depending on the use case, the lesser being in the case of two communicating devices in the same 6LoWPAN network, and the greater communication to a device outside the 6LoWPAN network and the prefix is not known. [23]

Even in the worst case 6LoWPAN provides a 50% reduction in header size, an approximately 9% reduction in overall session size for CoAP with OSCORE on the tested data. For larger payload sizes this percentage will decrease and for smaller it will increase.

v. The results in a broader context

While the comparisons and tests have been made to illustrate the improvements in cyber threat intelligence sharing for resource constrained IoT, it is also worth considering how these results would do in the a context more encompassing than the one limited to cyber threat intelligence sharing.

It could be seen in the results that sending an IoC requires an energy consumption of 28-112 mC, or 1,3-5,1 mC more than the baseline consumption, depending on the size of the message (124-496 bytes). These marginal costs of sending an IoC highlight the importance of radio duty cycling, sleep modes and other energy saving measures that limit the general energy consumption as sending IoCs, as shown, contributes little to the over all energy consumption.

Another factor to consider is the reachability of the IoT device. When Avast [24] deployed

500 fake IoT-like devices with common ports such as 8088 (Chromecast and Google Smart Home Speakers), 23 (Telnet), and 22 (SSH), open over four days they recorded 23,2 million attempts to connect with them. That is 11 588 attempts per device per day. Assuming this would be the case and that each attempt is sent as an IoC of 248 bytes, that would drain a 3000 mAh (such as one AA) battery in just over 16 days, if sending these IoCs were its sole function. Securing the device and closing unnecessary ports is therefore essential for a system like this to be practical, as is a well-functioning system that selects what is sent as an IoC.

XI. Conclusion

This thesis work has extended the state of the art in Cyber Threat Intelligence (CTI) sharing by extending a previously suggested model, implementing it and evaluating the results. CTI is receiving more attention as regulations are mandating incident disclosure and users seek to improve their cyber security state through collaborative efforts and shared information. More data has been added as more organizations and users have joined in sharing data and improved data analytic methods have improved the actionability of that data. One remaining challenge has been data structure and contextualizing it. Addressing this while keeping data both human readable and possible to automate has been the motivation for the STIX standard, and further the TAXII protocol specifically designed for sending STIX messages. While this addresses the need for enterprise computers, it does extend to resource constrained devices for which these standards are not possible to run. In order to improve data sharing, especially with resource constrained IoT devices, which long have had security risks, a new standard is needed. Currently there is no standard for cyber threat intelligence sharing for resource constrained IoT devices, but previous work by Iacovazzi et al. [20], modelled a version of STIX for resource constrained IoT devices, tinySTIX, and CoAP as an alternative to TAXII, on which this work could be made.

Adding to this OSCORE was chosen as the protocol for ensuring message confidentiality. Its slightly smaller size made it preferred for resource constrained IoT. Better knowledge of OSCORE within the organization at which this thesis was conducted, too, contributed to choosing OSCORE **[RQ1]**. Furthermore, when choosing CoAP, resources could be accessed through both request-response and with observable resources. Observable resources are resources that are accessed in a similar way to how publish-subscribe works. In this scenario the IoT device is considered the server, since it has the resources, and the threat intelligence platform, with which it communicates, the client. While observable resources were chosen as the main way of communicating alerts from resource constrained IoT, both sharing models publish-subscribe (implemented with observable resources) and request response could be mapped in the same implementation **[RQ2, RQ2.1, RQ2.2]**.

Opting for other protocols than HTTP and HTTPS meant relaxing requirements in TAXII but were considered reasonable trade offs when considering the lesser resource requirements. Because of this CoAP with OSCORE is not compliant with the TAXII standard, and not just because of not using HTTP, but because of omitting features such as discovery methods which are considered unnecessary in the realm of resource constrained IoT. An implementation with CoAP and OSCORE, such as the one in this thesis work or any other communication channel, can in a larger system be considered inter operable. With this implementation the message overhead can be significantly reduced while also adding the feature of observable resources, something that in the TAXII standard is defined but not implemented or specified in detail.

It was found desirable to have authentication, especially authentication of the IoT device, in this model considered the server, from which the alerts are sent. The possession of a OSCORE security context was considered sufficient for authentication as such a security context is set up by a system administrator. With this solution an IoT device can be considered authenticated and multiple IoT devices can either be handled individually or all together by using the unique identifiers of the OSCORE security context to provide access control **[RQ4, RQ4.1, RQ4.2]**. While it is a crude way of providing access control,

alternative ways add complexity in implementation and additional resource demands for the IoT device.

To reduce the sizes of STIX messages, both keys and values can be mapped to integers. All keys can be mapped to integers, as they are a closed set, whereas values can be enumerations, open vocabularies or free text, and it is only the former two that can be mapped to integer values. Data of the latter type, which cannot be mapped, is data like UUID, unique identifiers, time, and descriptions. This is done as part of the process to convert STIX messages to tinySTIX format. Following the conversion from string to integer values, it is converted to CBOR **[RQ3]**. The mappings are based on the STIX 2.1 standard and all listed keys, enumerations and open vocabularies are mapped to integer representations. To keep the data sizes small the keys are in one mapping and the values in another. Because of that the highest values mapped to have been possible to keep low, and thereby reducing data size. The mandatory values have been placed first in the lists and as such they will primarily map to one byte values where values above 23 map to two bytes in size **[RQ3.1]**.

With sample STIX data, and although not all fields can be mapped, tests showed an average size reduction of 35% in tinySTIX format compared to the original STIX message. Importantly the conversion from STIX to tinySTIX and back results in no data loss. Still, most OSINT data is too large to be practically useful for resource constrained IoT, even after tinySTIX conversion, and tinySTIX over CoAP is not a significant enough size reduction to warrant CTI consumption for IoT devices. It would also require having systems on the IoT device itself that is able to work with alerts and not full CTI as that would be too big **[RQ8]**.

CoAP and OSCORE are commonly part of the same library and as such there is no reason for separation. For resource constrained IoT such as Contiki, CoAP is part of the Contiki NG operating system. The main version of which does not support OSCORE, a branched version of it has support for that instead. In that version both CoAP and OSCORE exist and as such there is no reason for separation in the use of that either **[RQ5]**.

The testing conducted, using the Python implementation on an enterprise device, shows that STIX messages can be reduced in size by approximately 35% by using tinySTIX. Furthermore communication over CoAP reduces the header size by 82% and 85%, 70 to 406 and 492 bytes respectively, compared to HTTP and TAXII with the OpenTAXII implementation **[RQ6.1, RQ7]**.

Testing done on entire sessions show that the implementation reduces the number of packets sent by more than 85% compared to both OpenTAXII and HTTPS. It also lowers the over all session size by 85-91% when compared to the same **[RQ7.1]**.

In the process of implementing a server with observable resources sent over OSCORE on the resource constrained device, several issues were found. Neither observable resources nor blockwise transfer was compatible with OSCORE in Contiki and as such they were tested separately. That testing could however show that sending an IoC requires 28-112 mC depending on the size of the IoC. In comparison to the baseline consumption that is 5% above for a duration of 640-2560 ms **[RQ9, RQ9.1]**.

As tinySTIX conversion only is expected to occur on enterprise devices, the Python implementation is enough to show that tinySTIX can be implemented in practice. Apart from the limitations in how a resource is sent, blockwise and with OSCORE, it can still be con-

cluded that it also is possible to send such tinySTIX messages from resource constrained IoT **[RQ6]**.

In order to take this work into industry the requirements should be reconsidered to closely cater to the needs. OSCORE is still quite new and there are still issues when it comes to the maturity of packages that implement it. In the case that a proxy or end-to-end encryption is not needed DTLS is more mature and has better support in packages. For small messages it could also be considered to use 6LowPAN in order to reduce the header size which for small messages make up for a large enough part of the packet to warrant the use of it.

XII. Future work

This work has proposed a model for sharing cyber threat intelligence from resource constrained IoT devices by leveraging the MISP threat intelligence platform for parsing indicators of compromise sent from these devices. A Python implementation has been developed for the integration in MISP that has the functionality to convert from tinySTIX, a compressed version of STIX, and receive and send OSCORE encrypted messsages over CoAP. The work as part of this thesis has shown the potential in the model with testing on the implementation to back it up. It has not been in the scope of the project however to ensure production quality. Instead these implementations can be seen as a proof of concept. In the integration to MISP it is therefore of high importance that the code is tested to ensure production quality and functionality. Part of this is also verifying that the libraries used also have this. The Aiocoap library for example, has tests for sending data blockwise but it is written in them that the functionality only covers the minimum in order to pass the test. Furthermore, an OSCORE protected resource is also possible to get with a regular CoAP request. This too has to be addressed. Either by contributing to the Aiocoap library, by using the message data to drop any request without OSCORE to a resource protected with it, or by using another library.

As part of this work a server for resource constrained devices has also been added and tested. Similar issues with the CoAP and OSCORE library were encountered with it. Resources were available both with and without an OSCORE security context despite requests without it should have been discarded. It was also not possible to combine observable resources or blockwise transfer with OSCORE. All of this has to be resolved in order to have a production quality implementation. Another option is to use Zephyr, another operating system with support for CoAP and OSCORE.

Given the limitations in OSCORE functionality of both the Aiocoap library and Contiki it is worth considering DTLS instead of OSCORE as it has more mature implementations. The use of pre-shared keys could then be one option in order to omit the otherwise costly handshake at the start of a session.

In the case that the client, MISP, is expected to connect to, for it, unknown devices, it could instead be of interest to keep the handshake of DTLS or add one if OSCORE still is preferred. For OSCORE this could be added with EDHOC [21]. In that case it might also be worth extending authentication and improving access control. For OSCORE this could be added with Ace-OAuth [22].

In order to have a production ready system more testing also has to be conducted that emulates real-world scenarios. This means testing on a device that also is running another service, such as providing temperature data, and seeing how they behave side by side. It also means testing with an IDS on the device to see how the entire process from incident, discovery, tinySTIX generation and sharing performs. Testing should also be done with lossy networks and poor connection to understand data loss and whether or not CoAP confirmable messages (RFC 7252 [25], section 4.5 in RFC 7641 [26]) should be added to address this. It is worth considering in this that confirmable messages also would require the client to keep track of the message-exchange state [27].

With the testing that has been conducted it has been shown that IoCs can be sent from

a resource constrained device in such a way that the memory constraints and low energy requirements are respected. It is also shown in the discussion that despite this, the system can be unfeasible to use if too many IoCs are sent. While further improvements to tinySTIX and CoAP are possible they are insignificant if the system as a whole is unprotected or the IDS is generating too many IoCs to be sent.

It is therefore also of interest to test such an IDS and see if it is possible to have it generate IoCs in tinySTIX. Otherwise a parser could be added to the resource in the server so that it takes the IoC as input and sends it in tinySTIX format. This would however require tinySTIX mappings to be placed on the device to convert from the original IoC format to tinySTIX.

Radio duty cycling should also be tested to confirm that it minimally impacts the sending of IoCs while at the same time lowering energy consumption to the greatest possible extent.

Naturally there are also improvements to be made solely within the same setting as used in this work. Using 6LoWPAN instead of IPv6 would reduce the header size by at least 20 bytes per packet, potentially even more [23]. Conversion from STIX to tinySTIX could convert nested values, which currently is not done. The tested data does not seem to have nested values but in the case that larger IoCs are used, that could be the case. Further separating the tinySTIX mapping so that all values are 23 or smaller would save one byte for each such value that currently is mapped to an integer value of 24 or greater. Taking the second example in the tinySTIX mappings and the size of the message would reduce by another 3 bytes, from 125 to 122, a 2,4% reduction in size. For the first IoC in the same example and the improvement would only be one byte, a 0,83% improvement. This would however most likely also increase the size of the tinySTIX mappings. If the tinySTIX mappings only are used in the Python implementation, as they are now, that could be a worthwhile compromise. Additional message compactness could probably also be achieved with the use of packed CBOR [28]. With the tested data the header compression in the worst case with 6LoWPAN would still reduce the header size by 50% and reduce the size of a session by 9%. All together it is reasonable to assume that these improvements should reduce the over all session size by at least another 10%.

References

[1] Check Point Research Team. *Check Point Research Reports a 38% Increase in 2022 Global Cyberattacks*. en-US. Jan. 2023. URL: https://blog.checkpoint.com/2023/01/05/38-increase-in-2022-global-cyberattacks/.

[2] Mike Mclean. *2023 Must-Know Cyber Attack Statistics and Trends | Embroker*. en-US. June 2023. URL: https://www.embroker.com/blog/cyber-attack-statistics/.

[3] Verizon. *Verizon 2023 Data Breach Investigations Report*. URL: verizon.com/dbir.

[4] McKinsey Company. *Perspectives on transforming cybersecurity 2019*. en. 2019, p. 128. URL: https://www.mckinsey.com/~/media/McKinsey/McKinsey%20Solutions/Cyber%20Solutions/Perspectives%20on%20transforming%20cybersecurity/Transforming%20cybersecurity_March2019.ashx.

[5] Logan Kugler. "Standards to Secure the Sensors That Power IoT". In: *Commun. ACM* 66.6 (May 2023), pp. 14–16. ISSN: 0001-0782. DOI: 10.1145/3591215. URL: https://doi.org/10.1145/3591215.

[6] Chris Johnson, Larry Feldman, and Greg Witte. *Cyber-Threat Intelligence and Information Sharing*. ITL Bulletin. Computer Security Division, Information Technology Laboratory. National Institute of Standards and Technology, U.S. Department of Commerce, May 2017.

[7] Thomas D. Wagner et al. "Cyber threat intelligence sharing: Survey and research directions". In: *Computers Security* 87 (2019), p. 101589. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2019.101589. URL: https://www.sciencedirect.com/science/article/pii/S016740481830467X.

[8] The MITRE Corporation. *Standardizing Cyber Threat Intelligence Information with the Structured Threat Information eXpression (STIX™)*. The MITRE Corporation. 2012. URL: https://www.mitre.org/sites/default/files/publications/stix.pdf.

[9] Bret Jordan and Drew Varner. *TAXII Version 2.1*. OASIS Standard. June 2021. URL: https://docs.oasis-open.org/cti/taxii/v2.1/os/taxii-v2.1-os.html.

[10] Bret Jordan, Rich Piazza, and Trey Darley. *STIX Version 2.1*. OASIS Standard. June 2021. URL: https://docs.oasis-open.org/cti/stix/v2.1/os/stix-v2.1-os.html.

[11] *Introduction to Structured Threat Information Expression (STIX)*. en. URL: https://oasis-open.github.io/cti-documentation/docs/Introduction_to_Structured_Threat_Information_Expression.pdf.

[12] *Introduction to Trusted Automated eXchange of Intelligence Information (TAXII)*. en. URL: https://oasis-open.github.io/cti-documentation/docs/Introduction_to_Trusted_Automated_eXchange_of_Intelligence_Information.pdf.

[13] Kristofer Nedergaard, Bhupjit Singh, and Birger Andersen. "Evaluating CoAP, OSCORE, DTLS and HTTPS for Secure Device Communication". In: *Internet of Everything*. Ed. by Teresa Pereira, John Impagliazzo, and Henrique Santos. Cham: Springer Nature Switzerland, 2023, pp. 117–132. ISBN: 978-3-031-25222-8.

[14] *Zolertia Firefly — Contiki-NG documentation*. URL: https://docs.contiki-ng.org/en/develop/doc/platforms/zolertia/Zolertia-Firefly.html.

[15] ICS/SCADA honeypot. URL: github.com/mushorg/conpot.

[16] Seamus Dowling, Michael Schukat, and Hugh Melvin. "A ZigBee honeypot to assess IoT cyberattack behaviour". In: *2017 28th Irish Signals and Systems Conference (ISSC)*. June 2017, pp. 1–6. DOI: 10.1109/ISSC.2017.7983603.

[17] Shahid Raza, Linus Wallgren, and Thiemo Voigt. "SVELTE: Real-time intrusion detection in the Internet of Things". en. In: *Ad Hoc Networks* 11.8 (Nov. 2013), pp. 2661–2674. DOI: 10.1016/j.adhoc.2013.04.014. URL: https://www.sciencedirect.com/science/article/pii/S1570870513001005.

[18] Mardiana binti Mohamad Noor and Wan Haslina Hassan. "Current research on Internet of Things (IoT) security: A survey". en. In: *Computer Networks* 148 (Jan. 2019), pp. 283–294. DOI: 10.1016/j.comnet.2018.11.025. URL: https://www.sciencedirect.com/science/article/pii/S1389128618307035.

[19] "Directive (EU) 2022/2555 of the European Parliament and of the Council of 14 December 2022 on measures for a high common level of cybersecurity across the Union, amending Regulation (EU) No 910/2014 and Directive (EU) 2018/1972, and repealing Directive (EU) 2016/1148 (NIS 2 Directive) 2022". en. In: *Official Journal of the European Union* 333 (Dec. 2022).

[20] Alfonso Iacovazzi et al. "Towards Cyber Threat Intelligence for the IoT". en. In: *The 4th International Workshop on Security and Reliability of IoT Systems (SecRIoT 2023)* (June 2023).

[21] Göran Selander, John Preuß Mattsson, and Francesca Palombini. *Ephemeral Diffie-Hellman Over COSE (EDHOC)*. Internet-Draft draft-ietf-lake-edhoc-20. Work in Progress. Internet Engineering Task Force, July 2023. 110 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-lake-edhoc/20/.

[22] Ludwig Seitz et al. *Authentication and Authorization for Constrained Environments Using the OAuth 2.0 Framework (ACE-OAuth)*. RFC 9200. Aug. 2022. DOI: 10.17487/RFC9200. URL: https://www.rfc-editor.org/info/rfc9200.

[23] Jonas Olsson. "Demystifying 6LoWPAN". English. In: (Oct. 2014). URL: https://www.ti.com/lit/wp/swry013/swry013.pdf?ts=1697386185822.

[24] *When big fish get caught with big bait*. URL: https://blog.avast.com/millions-of-attacks-on-fake-iot-devices (visited on 10/23/2023).

[25] Zach Shelby, Klaus Hartke, and Carsten Bormann. *The Constrained Application Protocol (CoAP)*. RFC 7252. June 2014. DOI: 10.17487/RFC7252. URL: https://www.rfc-editor.org/info/rfc7252.

[26] Klaus Hartke. *Observing Resources in the Constrained Application Protocol (CoAP)*. RFC 7641. Sept. 2015. DOI: 10.17487/RFC7641. URL: https://www.rfc-editor.org/info/rfc7641.

[27] Klaus Hartke and Michael Richardson. *Extended Tokens and Stateless Clients in the Constrained Application Protocol (CoAP)*. RFC 8974. Jan. 2021. DOI: 10.17487/RFC8974. URL: https://www.rfc-editor.org/info/rfc8974.

[28] Carsten Bormann. *Packed CBOR*. Internet-Draft draft-ietf-cbor-packed-09. Work in Progress. Internet Engineering Task Force, July 2023. 25 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-cbor-packed/09/.

A General terminology

Actionable Intelligence — ENISA defines actionable intelligence as CTI that meets criteria such as relevance, timeliness, accuracy, completeness, and ingestibility. The Ponemon Institute adds additional criteria, including timeliness, priority, implementation, trustworthiness of the source, relevance to the industry, clear guidance to resolve the threat, and sufficient context.

Contiki — Contiki is an operating system for low power and resource constrained devices, like IoT. Contiki-NG is a forked version of the main repository where newer features are added. NG stands for next generation.

End-to-end encryption — Data is encrypted by the sender and can only be decrypted by the intended recipient.

HTTP (Hypertext Transfer Protocol) — HTTP is the protocol used for communication between clients and servers over the internet. It defines a set of rules and methods for clients and servers to exchange resources over the internet. HTTP allows clients to request resources from servers and receive responses, enabling the retrieval of web pages, images, documents, and other content.

HTTPS (Hypertext Transfer Protocol Secure) — HTTPS is a secure version of HTTP that provides encrypted communication between clients and servers. It incorporates SSL/TLS protocols to establish a secure and authenticated connection, protecting sensitive data from eavesdropping and tampering. HTTPS is commonly used for secure transactions, such as online banking and e-commerce.

IoC (Indicator of Compromise) — Indicators of compromise are clues or evidence that suggest that a network or an endpoint has been breached. It can be various kinds of data but IP addresses, network traffic patterns, filenames, paths and hashes are some.

IoT (Internet of Things) — IoT is an acronym for the Internet of Things, a loosely defined concept for connected devices. IoT devices could be anything from a drone, an autonomous car, a machine in a factory to a humidity sensor in a field. The common divider being that they are connected. Naturally the constraints on these devices vary considerably and as such this work focuses on resource constrained IoT devices, ones with limited computational power, memory and most often energy supply.

JSON (JavaScript Object Notation) — JSON is a lightweight data interchange format that uses a simple syntax to represent structured data. It is widely used for transmitting data between a server and a client in a readable and easily parseable format.

REST (Representational State Transfer) — REST is an architectural style for designing networked applications. It emphasizes the use of standard HTTP methods (GET, POST, PUT, DELETE) and URIs to interact with resources. RESTful APIs provide a uniform

interface for clients to access and manipulate resources on a server, promoting scalability, simplicity, and statelessness in web services.

SOC (Security Operations Center) — A Security Operations Center is responsible for protecting an organization against cyber attacks. The analysts in it perform monitoring of systems and networks in the organization to investigate any potential incidents.

TCP (Transmission Control Protocol) — TCP is a reliable, connection-oriented protocol that ensures error-checked and ordered data transmission between applications. It establishes a connection before data exchange, guaranteeing data integrity but with increased overhead and latency. TCP is commonly used for applications such as web browsing, email, and file transfers, where reliability is crucial.

Threat Intelligence Platform — A threat intelligence platform (TIP) is a solution that collects, aggregates and organizes threat intelligence data from multiple data sources.

UDP (User Datagram Protocol) — UDP is a lightweight, connectionless protocol that offers low-latency and fast communication between applications. It does not provide reliability mechanisms like TCP, resulting in faster transmission but with the possibility of data loss or disorder. UDP is ideal for real-time streaming, online gaming, and VoIP applications, where speed and responsiveness are prioritized over reliability.

B Used data for testing

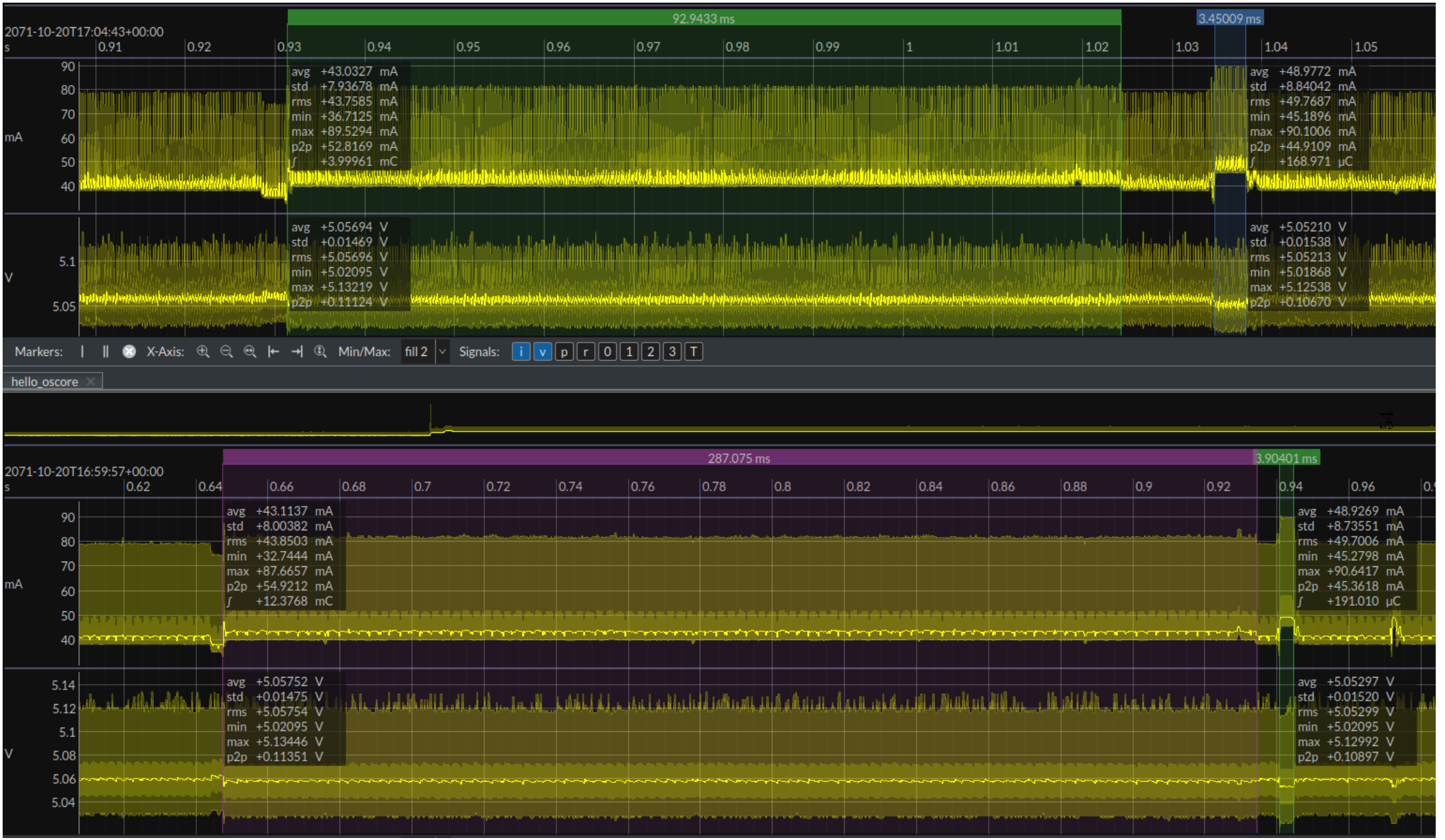
```
Number of IoCs (lines in file), file
    16 botvrij/tinystix_data/02a470d8-493e-41d9-8367-622460dddbe8.txt
   442 botvrij/tinystix_data/033e3332-eb5b-45d3-9a5c-68ca17aa9a8d.txt
    26 botvrij/tinystix_data/0319b483-5973-4932-91ea-5a44c2975b24.txt
   484 total

   140 circl.lu/tinystix_data/0b988513-9535-42f0-9ebc-5d6aec2e1c79.txt
    24 circl.lu/tinystix_data/0e887f03-5aa2-4a7b-b0f7-66208c6c657b.txt
   275 circl.lu/tinystix_data/0ebe51c2-31f1-4ba4-b7ab-1f5e62531e45.txt
   439 total

 15679 urlhaus/tinystix_data/0a31d356-9328-4ec5-9852-7e63290182bc.txt
  6805 urlhaus/tinystix_data/0a67ca83-d7ac-4c8c-9312-f73b50921843.txt
 10542 urlhaus/tinystix_data/0a135e36-5287-40fb-bdd7-b4ca2f2eaf4a.txt
 33026 total

  3814 threatfox/tinystix_data/0a47b872-e20e-4f88-b87e-5152c079ed0d.txt
   263 threatfox/tinystix_data/0a5460c5-e2a0-4a91-8911-fe79cd25e07c.txt
   207 threatfox/tinystix_data/0ac62ccd-8465-4a3f-8df1-c0d89162dd26.txt
  4284 total
```

C Full-size energy consumption


92.9433 ms
3.45009 ms
2071-10-20T17:04:43+00:00
s
0.91
0.92
0.93
0.94
0.95
0.96
0.97
0.98
0.99
1
1.01
1.02
1.03
1.04
1.05
mA
90
80
70
60
50
40
avg +43.0327 mA
std +7.93678 mA
rms +43.7585 mA
min +36.7125 mA
max +89.5294 mA
p2p +52.8169 mA
∫ +3.99961 mC
avg +48.9772 mA
std +8.84042 mA
rms +49.7687 mA
min +45.1896 mA
max +90.1006 mA
p2p +44.9109 mA
∫ +168.971 µC
V
5.1
5.05
avg +5.05694 V
std +0.01469 V
rms +5.05696 V
min +5.02095 V
max +5.13219 V
p2p +0.11124 V
avg +5.05210 V
std +0.01538 V
rms +5.05213 V
min +5.01868 V
max +5.12538 V
p2p +0.10670 V
Markers:
X-Axis:
Min/Max:
fill 2
Signals:
i
v
p
r
0
1
2
3
T
hello_oscore
287.075 ms
3.90401 ms
2071-10-20T16:59:57+00:00
s
0.62
0.64
0.66
0.68
0.7
0.72
0.74
0.76
0.78
0.8
0.82
0.84
0.86
0.88
0.9
0.92
0.94
0.96
mA
90
80
70
60
50
40
avg +43.1137 mA
std +8.00382 mA
rms +43.8503 mA
min +32.7444 mA
max +87.6657 mA
p2p +54.9212 mA
∫ +12.3768 mC
avg +48.9269 mA
std +8.73551 mA
rms +49.7006 mA
min +45.2798 mA
max +90.6417 mA
p2p +45.3618 mA
∫ +191.010 µC
V
5.14
5.12
5.1
5.08
5.06
5.04
avg +5.05752 V
std +0.01475 V
rms +5.05754 V
min +5.02095 V
max +5.13446 V
p2p +0.11351 V
avg +5.05297 V
std +0.01520 V
rms +5.05299 V
min +5.02095 V
max +5.12992 V
p2p +0.10897 V