Pushing the Limits of Cyber Threat Intelligence: Extending STIX to Support Complex Patterns

Martin Ussath, David Jaeger, Feng Cheng and Christoph Meinel

Abstract Nowadays, attacks against single computer systems or whole infrastructures pose a significant risk. Although deployed security systems are often able to prevent and detect standard attacks in a reliable way, it is not uncommon that more sophisticated attackers are capable to bypass these systems and stay undetected. To support the prevention and detection of attacks, the sharing of cyber threat intelligence information becomes increasingly important. Unfortunately, the currently available threat intelligence formats, such as YARA or STIX (Structured Threat Information eXpression), cannot be used to describe complex patterns that are needed to share relevant attack details about more sophisticated attacks.

In this paper, we propose an extension for the standardized STIX format that allows the description of complex patterns. With this extension it is possible to tag attributes of an object and use these attributes to describe precise relations between different objects. To evaluate the proposed STIX extension we analyzed the API calls of the credential dumping tool Mimikatz and created a pattern based on these calls. This pattern precisely describes the performed API calls of Mimikatz to access the LSASS (Local Security Authority Subsystem Service) process, which is responsible for authentication procedures in Windows. Due to the specified relations, it is possible to detect the execution of Mimikatz in a reliable way.

Keywords Cyber threat intelligence · STIX · CybOX · Complex pattern · Attribute relation

1 Introduction

Nearly every day, attackers try to compromise computer systems and network infrastructures to steal, for example, bank account information or to exfiltrate intellectual

M. Ussath(✉) · D. Jaeger · F. Cheng · C. Meinel
Hasso Plattner Institute (HPI), University of Potsdam, 14482 Potsdam, Germany
e-mail: {martin.ussath,david.jaeger,feng.cheng,christoph.meinel}@hpi.de

S. Latifi (ed.), *Information Technology New Generations*,
Advances in Intelligent Systems and Computing 448,
DOI: 10.1007/978-3-319-32467-8_20

property. Different signature-based [12] and anomaly-based approaches were developed to prevent and detect such attacks. These approaches are implemented in different security systems, such as firewalls, Intrusion Detection Systems (IDSs) and anti-virus products, which are widely used in current environments. In general, the deployed security systems provide a decent protection level against standard attacks, which are often highly automated and do not have specific targets. Beside standard attacks, there are also more sophisticated attacks that target specific companies, public authorities and organizations. The main characteristics of such sophisticated attacks, which are sometimes also called Advanced Persistent Threats (APTs), are that they are able to bypass existing security systems and they are also capable to largely infiltrate the target network without triggering alerts. To achieve this, the adversaries behind these attacks use different approaches, such as encryption, polymorphic malware or the imitation of legitimate activities to hinder the prevention and detection. Due to the fact that current security systems are not able to reliably prevent and detect sophisticated attacks, the sharing of cyber threat intelligence information becomes increasingly important.

The sharing of attack details, such as used command-and-control servers, malware or attack tools, is not a new approach. Different organizations and companies provide such information for quite some time, for example through investigation reports, blacklists or web services. Due to the fact that the different information feeds most often use a custom format, the automated processing and utilization of this information is complex. To simplify the usage of cyber threat intelligence information different formats, such as YARA, Open Indicators of Compromise (OpenIOC) or Structured Threat Information eXpression (STIX), were proposed. With the help of these standardized formats not only the efficiency of threat intelligence sharing should be improved, but also the integration of this information into prevention and detection systems should be much simpler [15]. Through the sharing and the automated processing of threat information, potential targets are capable to detect attacks in a faster manner. Also the sharing of classified threat intelligence information within a group of trusted partners is possible. By directly sharing such information, it is no longer necessary to wait until the vendor of a utilized security system releases for example a signature or pattern update.

Most publicly available cyber threat intelligence feeds parse other information feeds that use a custom format and provide the same information in a standardized format. Through the conversion of the information, various threat information details can be easily used to support prevention and detection activities. Although the new formats allow a very detailed description of attacks, most feeds only share basic information, such as IP addresses, hash sums, static patterns and domain names. This type of information might be helpful for standard attacks, but for complex and sophisticated attacks more details are needed for a reliable and precise detection. Especially for complex multi-step attack activities it is relevant to identify all related steps, otherwise it might not be possible to detect them, as single steps might look benign. The existing cyber threat intelligence formats do not support the description of complex patterns that are needed to detect more sophisticated attack activities.

In this paper, we propose an extension of the STIX format that allows the description of complex patterns. These pattern are based on object attributes and their relations. Through the sharing of such patterns it is possible to detect relevant attack activities that are performed during sophisticated attacks. Furthermore, we will evaluate the proposed extension by describing a pattern of the well-known credential dumping tool Mimikatz.

The remainder of this paper is structured as follows: Section 2 briefly describes three commonly used formats to share cyber threat intelligence information. Furthermore, in this section we also present three public services that provide threat intelligence information. Afterwards, in Section 3 we propose a STIX extension that allows the description of complex patterns. In the following section we evaluate the extension by describing the characteristic API calls of the credential dumping tool Mimikatz within a complex pattern. Finally, we conclude our paper and propose future work in Section 5.

2 Related Work

Through the sharing of threat intelligence information it is possible to create and provide new patterns and signatures, which should allow an effective prevention and detection of attacks in a timely manner. The usage of standardized formats [9] is one relevant requirement for the sharing of threat intelligence among different parties [14], [7]. Only if all involved parties have a common understanding about the meaning of the shared information [10], [4], an efficient and automated processing of this information can be implemented. In the following we will describe three common formats for the sharing of threat intelligence information.

Different public services, such as Open Threat Exchange from AlienVault [1] and X-Force Exchange from IBM [8], already use some of these formats to provide attack details. To show what kind of threat intelligence is currently shared, we will briefly describe the provided information of three different public services.

2.1 YARA

YARA [2] is an application that uses rules to identify potentially malicious files or suspicious processes in memory. Each YARA rule contains different characteristics that can be used to detect malicious files or processes. A rule usually has a *strings* and a *condition* section to describe relevant characteristics. The most common method is to use text strings, hexadecimal strings and regular expressions to describe relevant parts of a file or process. In the condition section a Boolean expression describes the relations of the strings and other properties that need to be matched to correctly identify the described file or process. With various properties it is possible to specify for example the file size, number of sections or exported functions. However, if various operators are used, the specified condition of a YARA rule can be relatively complex.

Listing 1.1 shows a YARA rule that describes characteristics of the TinyZBot malware. This rule was published as part of an investigation report [5] to allow potential targets to search for this type of malware. The rule defines nine different strings that are characteristic for this malware. The condition shows that the rule will return true if a file or process contains one of the four possible combinations of the specified strings.

```
rule TinyZBot
{
 strings:
    $s1 = "NetScp" wide
    $s2 = "TinyZBot.Properties.Resources.resources"
    $s3 = "Aoao WaterMark"
    $s4 = "Run_a_exe"
    $s5 = "netscp.exe"
    $s6 = "get_MainModule_WebReference_DefaultWS"
    $s7 = "remove_CheckFileMD5Completed"
    $s8 = "http://tempuri.org/"
    $s9 = "Zhoupin_Cleaver"

 condition:
    ($s1 and $s2) or ($s3 and $s4 and $s5) or ($s6 and $s7 and $s8) or $s9
}
```

Listing 1.1 TinyZBot YARA Rule [5]

YARA and the corresponding rules are focused on the detection of potentially malicious files and processes. This is also the reason why the format does not support the description of other characteristics that are for example relevant to detect malicious network traffic. Furthermore, the format does not provide easy to use extension capabilities. Although the detection of suspicious files and processes is crucial for the detection of attacks, there are various other attack details that are relevant and should be shared. Especially in cases where the attackers use legitimate credentials and administrative tools, it is not possible to use YARA to detect such type of malicious activities.

2.2 *Open Indicators of Compromise (OpenIOC)*

OpenIOC [11] is an XML-based format that can be used to describe forensic artifacts of attacks. The single indicators, which contain details of an intrusion, are called IOCs (Indicators of Compromise). The format was proposed by the company Mandiant. Due to the standardization and public availability of the OpenIOC format, also other vendors started supporting the usage of IOCs as threat intelligence format.

OpenIOC supports, by default, more than 500 different IOC terms[1], which can be used to describe different artifacts, such as file and process attributes or network indicators. By sharing and using such IOCs, the identification of forensic artifacts, which belong to an attack, should be much simpler. The design of the format also allows to add extensions for additional artifacts. A single IOC can describe various artifacts of an attack and these characteristics can be linked through operators, such as *OR* as well as *AND*.

[1] http://openioc.org/terms/Current.iocterms

Listing 1.2 shows an IOC that describes the artifacts of the CHOPSTICK backdoor that was used during the APT28 campaign [6]. To ensure clarity of the example, some parts with meta-information were removed. This IOC contains two single artifacts and one combined artifact that are linked with *OR*. The two single artifacts describe a file name (*edg6EF885E2.tmp*) and a name of a handle (*\Device\Mailslot\check_mes_v5555*). The combined artifact is linked with AND and consists of a registry path (*Microsoft\MediaPlayer\E6696105-E63E-4EF1-939E-15DDD83B669A*) and a name of a regstiry value (*chnnl*). If one of the three artifacts, one of the two single artifacts or the combined artifact can be found on a system, then this strongly indicates that this system is or was infected with the CHOPSTICK malware.

```
<ioc id="[...]" last-modified="2014-10-20T18:50:53Z">
 <definition>
  <Indicator id="[...]" operator="OR">
   <IndicatorItem id="[...]" condition="is">
    <Context document="FileItem" search="FileItem/FileName" type="mir"/>
    <Content type="string">edg6EF885E2.tmp</Content>
   </IndicatorItem>
   <IndicatorItem id="[...]" condition="is">
    <Context document="ProcessItem" search="ProcessItem/HandleList/Handle/Name" type="mir"/>
    <Content type="string">\Device\Mailslot\check_mes_v5555</Content>
   </IndicatorItem>
   <Indicator id="[...]" operator="AND">
    <IndicatorItem id="[...]" condition="contains">
     <Context document="RegistryItem" search="RegistryItem/Path" type="mir"/>
     <Content type="string">Microsoft\MediaPlayer\{E6696105-E63E-4EF1-939E-15DDD83B669A}</Content>
    </IndicatorItem>
    <IndicatorItem id="[...]" condition="contains">
     <Context document="RegistryItem" search="RegistryItem/ValueName" type="mir"/>
     <Content type="string">chnnl</Content>
    </IndicatorItem>
   </Indicator>
  </Indicator>
 </definition>
</ioc>
```

Listing 1.2 CHOPSTICK IOC[2]

The OpenIOC standard is capable to describe various forensic artifacts of an attack, which can be used to identify potentially compromised systems. By default, the format only supports the combination of artifacts with an OR or AND operator. In most cases these two operators are enough to describe the relations between artifacts, but for sophisticated attacks it is often necessary to specify the relation in a more detailed way. Especially for multi-step attacks, which sometimes use legitimate credentials and built-in administrative programs, it is essential to identify artifacts that are related for example through a session identifier to ensure that these steps belong to a malicious activity. Furthermore, the standardized OpenIOC format does not support the enrichment of artifacts with relevant context information.

2.3 STIX

STIX [3] is another XML-based format that is still under active development. The corresponding project is led by OASIS organization. Nevertheless, the development

[2] https://raw.githubusercontent.com/fireeye/iocs/master/APT28/bdf7929c-3f0b-4fdd-bcc5-b4a82554ad92.ioc

of the format is open and community-driven. This might be also one reason why numerous vendors and products[3] started to support the STIX format.

The objective of STIX is to describe attacks and relevant context information in a comprehensive way to enable an efficient and effective threat information sharing. The format can make use of eight different core constructs, which are *Campaign*, *CourseOfAction*, *ExploitTarget*, *Incident*, *Indicator*, *Observable*, *ThreatActor* and *TTP* (tactics, techniques and procedures). The CybOX format is used within STIX to describe Observables, but it is an additional project that is developed in conjunction with STIX. To allow a comprehensive description of attacks and context information, the objects of the different core constructs can be linked with other objects. Due to the increasing complexity of sophisticated attacks, STIX was designed to be flexible and extensible to support most cases. Therefore, it is also possible to embed other formats and patterns, such as YARA rules or IOCs, into STIX. Another standard that is developed in conjunction with the STIX project, is the so-called Trusted Automated eXchange of Indicator Information (TAXII) standard that enables a secure and automated sharing of cyber threat intelligence information.

Listing 1.3 shows a simple STIX document[4] that describes an IP address of a command-and-control infrastructure of an attacker. To ensure clarity, the sample was shortened to the relevant parts of the STIX document. First of all, an IPv4 address object (*198.51.100.2*) is defined as an observable. In the second part of the sample the defined observable is referenced to describe the used infrastructure of the attacker as part of a TTP object.

```
<stix:STIX_Package id="[...]" version="1.2">
 <stix:Observables cybox_major_version="1" cybox_minor_version="1">
  <cybox:Observable id="example:observable-c8c32b6e-2ea8-51c4-6446-7f5218072f27">
   <cybox:Object id="[...]">
    <cybox:Properties xsi:type="AddressObject:AddressObjectType" category="ipv4-addr">
     <AddressObject:Address_Value>198.51.100.2</AddressObject:Address_Value>
    </cybox:Properties>
   </cybox:Object>
  </cybox:Observable>
 </stix:Observables>
 <stix:TTPs>
  <stix:TTP xsi:type="ttp:TTPType" id="[...]">
   <ttp:Title>Malware C2 Channel</ttp:Title>
   <ttp:Resources>
    <ttp:Infrastructure>
     <ttp:Type>Malware C2</ttp:Type>
     <ttp:Observable_Characterization cybox_major_version="2" cybox_minor_version="1">
      <cybox:Observable idref="example:observable-c8c32b6e-2ea8-51c4-6446-7f5218072f27"/>
     </ttp:Observable_Characterization>
    </ttp:Infrastructure>
   </ttp:Resources>
  </stix:TTP>
 </stix:TTPs>
</stix:STIX_Package>
```

Listing 1.3 STIX Sample[4]

Due to the design of the STIX format, various issues of previous threat intelligence formats could be addressed. The approach of STIX is to allow a holistic description of attack details and relevant context information, such as confidence levels, which are crucial for sharing threat intelligence information. Although the format allows to describe various relations between different objects, it is not possible to describe more complex attack steps that are connected through certain characteristics.

[3] http://stixproject.github.io/supporters/

[4] https://stixproject.github.io/documentation/idioms/c2-ip-list/

2.4 Standardized Cyber Threat Intelligence Usage

According to a survey of the SANS Institute [15], only 38 % of the organizations are using cyber threat intelligence information in standardized formats. About 46 % of these companies use the STIX format for cyber threat intelligence information. The related standards CybOX (26 %) and TAXII (33 %) are also relatively often utilized. The OpenIOC format is used in 33 % of the organizations that employ standardized formats for their threat intelligence information. This survey shows that the overall adoption of standardized threat intelligence formats is not very high and this probably also has a negative impact on the efficient sharing of such information. One reason for this might be that currently most public services only provide already available information in a standardized format. Therefore, no additional threat intelligence is offered, but further overhead is introduced through the usage of relatively complex formats.

2.5 Existing Services

Hail a Taxii[5] and X-Force Exchange[6] from IBM are two public services that provide threat intelligence information in the STIX format. For transferring the information, both services use the TAXII protocols and data formats. Hail a Taxii converts already existing threat intelligence information, for example from the different malware trackers of abuse.ch or the CyberCrime tracker, into a STIX compliant format. In general this service offers mainly so-called STIX watchlists for domains, URLs, file hashes and IPs. These lists consist of the corresponding indicators without providing any further details or insights. The service of IBM provides mainly investigation reports in the STIX format. Unfortunately, the indicators of the different reports are most often not correctly converted into the corresponding STIX or CybOX objects. In these cases the indicators are only mentioned in the description field of the STIX document, which makes the automated processing of the threat intelligence information very complex. Nevertheless, the X-Force Exchange service also offers context information about IPs, URLs and vulnerabilities.

Open Threat Exchange (OTX)[7] from AlienVault is another service that provides threat intelligence information. This service should provide a platform for the community to share threat intelligence information. Users can create so-called pulses to share information with other community members. To create a pulse it is possible to upload PDF reports, plain text files or STIX documents. The parser of OTX tries to extract relevant indicators like IP addresses, domains, file hashes, file paths and email addresses. The shared information can be downloaded in different formats, such as CSV, OpenIOC 1.0 or 1.1 and STIX. A brief evaluation of the OTX service

[5] http://hailataxii.com/

[6] https://exchange.xforce.ibmcloud.com/

[7] https://otx.alienvault.com/

showed that different indicators were only extracted, because of the fact that the parser identified a product version number as an IP address. Furthermore, most often the shared information was taken from investigation reports or articles and the threat intelligence information was not enriched with relevant context information.

3 STIX Extension

The mentioned public services only provide basic threat intelligence information, such as IP addresses and domain names. These services do not offer any complex patterns that are needed to detect the increasing number of complex malicious activities. Furthermore, also the standardized cyber threat intelligence formats do not support the description of complex patterns and therefore it is not possible to use these formats to describe sophisticated attack activities.

Therefore we extended the existing STIX format in a way that it supports the description of complex patterns with object attribute relations.

3.1 Current Capabilities

The current version of STIX (1.2), including CybOX (2.1), supports mainly two different ways to describe relations or compositions of elements. For very simplistic relations it is possible to use *Composite_Indicator_Expression* (*indicator* schema) or *Oberservable_Composition* (*cybox* schema) elements together with the corresponding *operator* attribute. The value of this attribute can be *OR* or *AND*. Due to the limited values of the *operator* attribute and the fact that it is not possible to describe relations between the different object attributes, the defined relations are very imprecise. Another possible method to describe relations in STIX is to use a *Related_Object* (*cybox* schema) element and one term of the predefined relationship vocabulary [13] (*ObjectRelationshipVocab-1.1*), such as *Created*, *Deleted_by* or *Uploaded_By*. By using a specific relationship vocabulary term, it is also implicitly defined which source and related objects can be used. For example, to describe that a specific process locks a certain file, the *Locked* relationship term can be used in conjunction with a *Process* and *File* object. Although, the second method allows to describe relations in a more fine-granular way, it is also not possible to describe connections between different object attributes.

3.2 Complex Patterns

For the description of complex patterns, it is necessary to specify how single elements are related with each other. These relations are often based on characteristics

or attributes that are identical or similar for different connected elements. Without describing such relations, the identification of complex patterns might be imprecise, due to the fact that elements from different activities are wrongly connected. For example, if different events recorded the malicious activity of an attacker, these events are often connected through a session identifier that ensures that all these events belong to the session of the attacker and not to an unrelated benign activity. Therefore, complex patterns are not only relevant for describing more sophisticated multi-step attacks, but also to define very precise patterns to reduce the false positive rate. Furthermore, complex attack activities often consist of multiple steps and it is usually not possible to identify one characteristic step that can be used to detect such malicious actions in a reliable way. Thus, all relevant steps of an activity and their relations have to be identified. These characteristics can then be used to describe precise patterns.

3.3 Extension

Our proposed STIX extension allows the description of complex patterns by using object attribute relations. Furthermore, this extension uses the flexible design of STIX for a seamless integration into the format. To achieve the latter it is necessary to have a thorough understanding of the format to correctly extend the format schemas and minimize the required changes.

The main idea is to tag certain attributes of an object that are later used to describe object relations. The different relations between the object attributes are specified in a separate part of the STIX document to ensure flexibility.

To allow a seamless integration and the usage of existing CybOX objects, the extension adds an attribute group to the *BaseObjectPropertyType* of the *cybox_common* schema. Due to the fact that all CybOX elements are indirectly based on the *BaseObjectPropertyType*, the added attribute group *ReferenceFieldGroup* will be available within all object properties. Therefore, it is easily possible to extend already described patterns with detailed information about the relations of the different objects. The *ReferenceFieldGroup* only contains the attribute *reference_name* that is used to address the object property and allow the description of related properties of different objects.

For describing complex object property relations, the extension makes use of the *CompositeIndicatorExpressionType* of the *indicator* schema to provide the necessary details. As already explained the *Composite_Indicator_Expression* element, which is based on the *CompositeIndicatorExpressionType*, allows by default the description of relatively abstract object relations. To reuse this existing feature of STIX, the extension adds *COMPLEX* to the list of valid *operator* values. Furthermore, an additional *Relation* element is specified within the *CompositeIndicatorExpressionType*. Each *Relation* element contains a mandatory attribute that describes the type of the object attribute relation, such as *Equals*. Within the *Releation* element, the related objects with their corresponding *id* are referenced, including the *ReferenceName* of

the element property that is used to describe the relation. Due to the fact that every relation consists of at least two elements, the minimal number of related elements within the *Relation* element is two.

4 Evaluation

For the evaluation of the proposed STIX extension, we will describe a pattern for Mimikatz. Mimikatz is a tool that is often used within more sophisticated attacks, because it is capable to dump credentials. The attackers can misuse the gathered credentials for example to compromise further systems or to move laterally in the environment. It is very difficult to distinguish between benign activities and malicious activities that make use of legitimate credentials. Therefore, it is very crucial to detect the execution of Mimikatz. Due to the fact that this tool is open source, potential attackers can easily modify the source code and re-compile it. Thus, it is not possible to use hash sums to detect the Mimikatz executables in a reliable way. However, it is very unlikely that an attacker will change the general approach of the tool.

Therefore, it is necessary to identify the main characteristics and afterwards use these characteristics to create a meaningful pattern. In the case of Mimikatz we analyzed the performed API calls and identified six different calls that are characteristic for this tool and also relevant to get access to the memory of the LSASS (Local Security Authority Subsystem Service) process and dump credentials. The mentioned LSASS process is responsible for authentication procedures in Windows. Based on the identified API calls it is possible to reliably identify Mimikatz processes.

Due to the fact that the API parameters are dynamic and related, it is not possible to define a static pattern with the different functions and the corresponding parameters to precisely detect the execution of Mimikatz. We rather need to describe a pattern that consists of the different API calls and their relations. These relations are based on API parameters as well as values and ensure that the calls are related. This also improves the precision of the described pattern, because it will not match with processes that call the identified APIs in an unrelated way.

For the identification of the characteristic API calls of Mimikatz (version 2.0 alpha), we executed the *privilege::debug* and the *sekurlsa::logonpasswords* command from the Mimikatz command-line interface. Through these commands the process requests additional privileges and afterwards the logon passwords are dumped, including hashes and clear text passwords. Figure 1 visualizes the different API calls and the relations between these calls. All parameters that are statically define within the pattern are printed italic and the colored parameters are used within object relations. First of all the *RtlAdjustPrivilege* API is called to request further privileges that are needed to access the process memory of the LSASS process. Afterwards, Mimikatz needs to identify the process id of the LSASS process. Therefore, the Unicode string "*lsass.exe*" is loaded into the memory with the help of the *RtlInitUnicodeString* API and then this memory address is used within the *RtlEqualUnicodeString* API call to identify the correct process information of the LSASS process

and get access to the process id. To get all information about the running processes, Mimikatz uses the *NtQuerySystemInformation* API call. Thereafter, the identified id is used for the *OpenProcess* API call, which returns a handle to the LSASS process. This handle is then used to read the memory of the process with the help of the *ReadProcessMemory* API call.

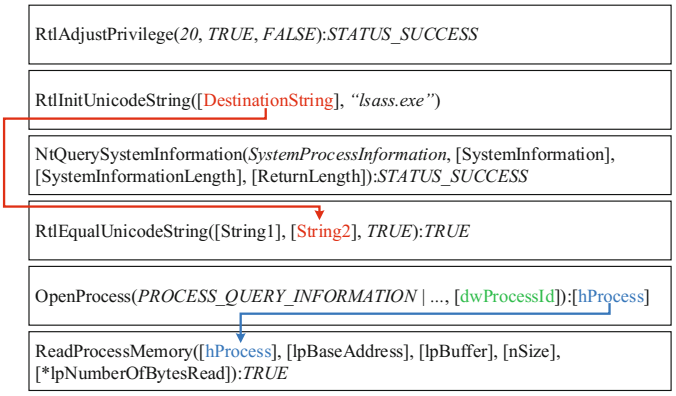

Fig. 1 Characteristic API Calls of Mimikatz

To describe a complex STIX pattern that consists of the identified API calls and the corresponding relations between them, it was necessary to create a new object type that allows the precise description of Windows API calls. This new object type is called *WindowsAPIObjectType* and is based on the default *APIObjectType*. It allows the description of the parameters that are used for the API call and also the return value.

Listing 1.4 shows the complex STIX pattern that describes the different API calls of Mimikatz. To ensure readability, the shown STIX document only contains the description of one relation and one API call. In the first part of the document the different element relations are described. For example the handle that is returned from the *OpenProcess* API call has to be identical to the handle that is used for the *ReadProcessMemory* call. Therefore, within the *Relation* element the corresponding API calls are referenced via the *idref* attribute and the *ReferenceName* of the related element attributes. The single API calls, with their attributes, are defined as *Indicators*. The element attributes that are used for describing relations, have a *reference_name* attribute so that they can be easily referenced within the relations. The values of the different parameters of the API calls are only defined statically, if they are necessary for the successful execution of Mimikatz. Next to the already mentioned API calls, the complex pattern also defines a process object for the LSASS process to obtain the process id of it. The reason for this is that the *NtQuerySystemInformation* returns a complex structure that contains various information about all processes that are

running on the system. This structure cannot be easily used within a pattern and therefore it is necessary to obtain the process id of LSASS from somewhere else. Nevertheless, the Mimikatz process calls *NtQuerySystemInformation* and this is also a relevant characteristic for the execution and cannot be omitted for a precise pattern.

```
<stix:STIX_Package>
 <stix:Indicators>
  <stix:Indicator id="[...]" xsi:type='indicator:IndicatorType'>
   <indicator:Composite_Indicator_Expression operator="COMPLEX">
    [...]
    <indicator:Relation relation_type="Equals">
     <indicator:RelationElement idref="1">
      <indicator:ReferenceName>hProcess</indicator:ReferenceName>
     </indicator:RelationElement>
     <indicator:RelationElement idref="2">
      <indicator:ReferenceName>hProcess</indicator:ReferenceName>
     </indicator:RelationElement>
    </indicator:Relation>
    [...]
    <indicator:Indicator id="[...]" xsi:type='indicator:IndicatorType'>
     <indicator:Observable id="[...]">
      <cybox:Object id="2">
       <cybox:Properties xsi:type="WinAPIObj:WindowsAPIObjectType">
        <Function_Name xmlns="[...]#APIObject-2">ReadProcessMemory</Function_Name>
        <WinAPIObj:Parameter reference_name="hProcess" position="1"></WinAPIObj:Parameter>
        <WinAPIObj:Return_Value>TRUE<WinAPIObj:Return_Value>
       </cybox:Properties>
      </cybox:Object>
     </indicator:Observable>
    </indicator:Indicator>
   </indicator:Composite_Indicator_Expression>
  </stix:Indicator>
 </stix:Indicators>
</stix:STIX_Package>
```

Listing 1.4 Complex STIX Pattern of Mimikatz API Calls (shortened)

5 Conclusion and Future Work

In this paper, we proposed a generic STIX extension that allows the description of complex patterns. These patterns are based on attribute relations that can be described through the tagging of relevant attributes and the specification of object attribute relations. Due to the fact that the extension changes the *cybox_common* schema and the *indicator* schema, all default objects can be used to define complex patterns.

For the evaluation of our STIX extension we analyzed the API calls of the credential dumping tool Mimikatz and described the characteristic calls within a STIX document that uses complex relations. To support the description of Windows API calls, we also defined a new *WindowsAPIObjectType*. With the specified API calls and their relations, it is possible to detect the execution of Mimikatz in a reliable way.

The evaluation showed that it is possible to use the STIX extension to describe all relevant characteristics of more sophisticated attack activities. Therefore, it is possible to share relevant attack details about complex attacks and use this information to support prevent and detection systems.

For future work it might be interesting to improve existing patterns with additional relations and evaluate the change of the precision. This evaluation could also reveal, if it is always reasonable to invest additional efforts and describe complex patterns with relations. Another task could be to propose and implement a system that is capable to automatically enhance existing patterns, for example with the help of a knowledge base.

References

1. AlienVault: AlienVault Open Threat Exchange (OTX)™ User Guide, October 2015. https://www.alienvault.com/doc-repo/OTX/user-guides/AlienVault-OTX-User-Guide.pdf
2. Alvarez, V.M.: Yara User's Manual (2011). https://yara-project.googlecode.com/files/YARA%20User's%20Manual%201.6.pdf
3. Barnum, S.: Standardizing Cyber Threat Intelligence Information with the Structured Threat Information eXpression (STIX™). MITRE Corporation, February 2014. https://stixproject.github.io/getting-started/whitepaper/
4. Costa, D.L., Collins, M.L., Perl, S.J., Albrethsen, M.J., Silowash, G.J., Spooner, D.L.: An ontology for insider threat indicators: development and application. In: Proceedings of the 9th Conference on Semantic Technology for Intelligence, Defense, and Security (2014)
5. Cylance: Operation Cleaver, December 2014. http://www0.cylance.com/assets/Cleaver/Cylance_Operation_Cleaver_Report.pdf
6. FireEye Labs: APT28: A Window Into Russia's Cyber Espionage Operations? October 2014. https://www.fireeye.com/content/dam/fireeye-www/global/en/current-threats/pdfs/rpt-apt28.pdf
7. Haass, J.C., Ahn, G.J., Grimmelmann, F.: Actra: a case study for threat information sharing. In: Proceedings of the 2nd ACM Workshop on Information Sharing and Collaborative Security, pp. 23–26. ACM (2015)
8. IBM: IBM X-Force Exchange Data Sheet, April 2015. http://public.dhe.ibm.com/common/ssi/ecm/wg/en/wgd03055usen/WGD03055USEN.PDF
9. Kampanakis, P.: Security Automation and Threat Information-Sharing Options. Security Privacy, 42–51. IEEE, September 2014
10. Kul, G., Upadhyaya, S.: A preliminary cyber ontology for insider threats in the financial sector. In: Proceedings of the 7th ACM CCS International Workshop on Managing Insider Security Threats, pp. 75–78. ACM (2015)
11. Mandiant: An Introduction to OpenIOC (2011). http://openioc.org/resources/An_Introduction_to_OpenIOC.pdf
12. Meier, M.: A model for the semantics of attack signatures in misuse detection systems. In: Information Security. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg (2004)
13. MITRE Corporation: Object Relationships. http://cyboxproject.github.io/documentation/object-relationships/
14. Serrano, O., Dandurand, L., Brown, S.: On the design of a cyber security data sharing system. In: Proceedings of the 2014 ACM Workshop on Information Sharing and Collaborative Security, pp. 61–69. ACM (2014)
15. Shackleford, D.: Who's Using Cyberthreat Intelligence and How? SANS Institute, February 2015. http://www.sans.org/reading-room/whitepapers/analyst/cyberthreat-intelligence-how-35767