Automatic Generation of MAEC and STIX Standards for Android Malware Threat Intelligence

Jungsoo Park[1], Long Nguyen Vu[2], George Bencivengo[3] and Souhwan Jung[2*]
[1] School of Software Convergence, Soongsil University, Seoul, South Korea
[e-mail: ddukki86@ssu.ac.kr]
[2] Department of Electronic Engineering, Soongil University, Seoul, South Korea
[e-mail: longnv@ssu.ac.kr, souhwanj@ssu.ac.kr]
[3] GINIA Inc, Washington DC, US
[e-mail: George.Bencivengo@ginia.com,]
*Corresponding author: Souhwan Jung

Received February 14, 2020; revised April 29, 2020; revised June 19, 2020; revised July 24, 2020; accepted August 3, 2020; published August 31, 2020

Abstract

Due to the increasing number of malicious software (also known as malware), methods for sharing threat information are being studied by various organizations. The Malware Attribute Enumeration and Characterization (MAEC) format of malware is created by analysts, converted to Structured Threat Information Expression (STIX), and distributed by using Trusted Automated eXchange of Indicator Information (TAXII) protocol. Currently, when sharing malware analysis results, analysts have to manually input them into MAEC. Not many analysis results are shared publicly. In this paper, we propose an automated MAEC conversion technique for sharing analysis results of malicious Android applications. Upon continuous research and study of various static and dynamic analysis techniques of Android Applications, we developed a conversion tool by classifying parts that can be converted automatically through MAEC standard analysis, and parts that can be entered manually by analysts. Also using MAEC-to-STIX conversion, we have discovered that the MAEC file can be converted into STIX. Although other researches have been conducted on automatic conversion techniques of MAEC, they were limited to Windows and Linux only. In further verification of the conversion rate, we confirmed that analysts could improve the efficiency of analysis and establish a faster sharing system to cope with various Android malware using our proposed technique.

Keywords: MAEC, STIX, Android Malware, Cyber Threat Intelligence

A preliminary version of this paper was presented at ICONI 2019, and was selected as an outstanding paper.
This work was supported in part by the Institute for Information and communications Technology Promotion (IITP) Grant funded by the Korean Government (MSIT) (Machine Learning based Intelligent Malware Analysis Platform) under Grant 2017-0-01853 and This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2020-0-00952, Development of 5G Edge Security Technology for Ensuring 5G+ Service Stability and Availability)

1. Introduction

Recently, cyber threat has become one of the most serious threats to the economy and national security. It threatens the security of social infrastructures which are directly connected with the lives of people. To cope with such the situation, the United States has made efforts to share cyber threat information between public and private institutions by establishing standards for cyber threat information sharing and enacting related laws based on DHS [1]. We believe that cyber threat information sharing would upgrade the cyber countermeasure and analysis of malware, and hacking information could be utilized as big data for defending intelligent attacks. By sharing such cyber threat information, existing attacks can be mitigated more effectively due to security firms and other organizations that can identify attack trends as well as the evolving new techniques of these attacks so that countermeasures can be built appropriately. We believe that cyber threat information sharing can lead to the following effects. First, effective defense against existing attacks. Second, quick detection of attack trends and the emergence of new techniques. Third, the capability to swiftly build a proper countermeasure security system from information gathered in advance.

MITRE, a nonprofit research and development center under DHS, announced MAEC in 2010 and CybOX standards in 2012 for sharing malware analysis. In 2010, MITRE started development to share automate cyber threat information and announced TAXII and STIX in 2013[2]. As stated in the introduction MAEC is used by malware analysts and reverse engineers to share information about malware. STIX uses the MAEC schema to describe attack techniques related to malware in detail. In 2018, MITRE released MAEC 5.0, which provides features for malware sharing system in Android and iOS to cope with malware in mobile environments[3]. However, in the case of MAEC, analysts have to write their data based on the analyzed results. Therefore, research is underway on technologies that can automatically identify and input components[4][5]. Although MAEC 5.0 was released recently, additional research and development are required on automatic conversion techniques through the analysis of Android malware. In this paper, for the establishment of an efficient and accurate malware sharing system for Android, we first proposed and implement a technique of creating MAEC through APK file analysis and automate conversion into STIX.

Our proposed contributions are as follows:

- Analyze the malware analysis method defined in MAEC 5.0 and performed the task of matching with the analyzed result through our developed analysis system.
- Develop technology to automatically convert Android malware analysis results to the MAEC 5.0 standard.
- Calculate the rate of automation to see how well it performs.
- Attach the implementation results of the technology to provide Android malware sharing by changing to STIX based on MAEC 5.0.

In Section 2, we explain the cyber threat information sharing system such as MAEC and STIX. We describe Android malware definition, dynamic analysis technique in Section 3. In Section 4, we introduce the techniques for the automatic conversion of static and dynamic analysis results to MAEC. In Section 5, we present the experimental results and conclude our study in Section 6.

2. Background

In this section, STIX and MAEC, the standards for sharing cyber threat information, are discussed in detail. NIST issued the "Guide to Cyber Threat Information Sharing" document in October 2016[6]. This document describes the current problems, benefits of sharing cyber threat information with organizations, and summarizes the features that should be provided for sharing. It is limited at the time, due to the lack of sharing information. Threats were not described in detail and were developed in various forms (xls, PDF, RSS, xml, e-mail, etc.) In order to solve it, there was another proposal of MAEC and STIX versions in 2017, which are currently being used.

2.1 MAEC

MAEC is a language for expressing structured information on malware based on attributes such as malware behavior, artifacts, and attack patterns[7]. By eliminating ambiguity and inaccuracies that may exist in the description of malware and reducing reliance on signatures, it is possible to improve communication between person-to-person, person-to-tool, and tool-to-tool. In addition, it is possible to develop a faster countermeasure method by using a function within a case of malware[8][9]. It also enables association, integration, and automation, which can reduce the possibility of duplication of researchers' analysis of malware. In June 2014, MAEC 4.1 was introduced, and MAEC 5.0 was released in October 2017. In the previous MAEC 4 version, MAEC Bundle, Package, and Container are classified according to usage. Each model has its own XML schema. MAEC 5.0 represents a significant refactoring and simplification of previous MAEC data models. Broadly, the MAEC 5.0 development goals were to simplify and refactor any components that were overly complicated or nested, to deprecate any unused components, and to align with the design principles of STIX 2[10].

2.2 STIX

STIX (Structured Threat Information eXpression) is a structured language for describing cyber threat information. Users can share, store, and analyze cyber threat information in a consistent way using STIX [11] The leading language for threat information within the United States is STIX. It is a standard language being developed in collaboration with any and all interested parties for the specification, capture, characterization, and communication of standardized cyber threat information. It does so in a structured fashion to support more effective cyber threat management processes and the application of automation. STIX provides a common mechanism for addressing structured cyber threat information across and among this full range of use cases improving consistency, efficiency, interoperability, and overall situational awareness. It consists of eight components. Users can express threat information using each component depending on different factors. The implementation has been succinctly changed from existing XML to JSON[12].

3. Android Malware Analysis

In this section, we introduce methods to acquire the necessary content for automatic conversion to MAEC through Android malware analysis. Using existing Android malware and dynamic analysis tools, it is possible to check whether the output is available for automatic conversion.

3.1 Android malware static analysis

Different static analysis tools can be used for analyzing Android applications. Apktool is a reverse engineering tool that can decompile APK files into smali code[13]. It is also possible to decode an APK, modify the extracted resource, rebuild it, and repackage it into a new APK. The result of decompiling an APK file with apktool contains an AndroidManifest.xml, images, XML files for UI layout, values, and a source file decoded in the form of smali code. Parsed AndroidManifest.xml files can be used for MAEC conversion. Certificates, signatures, etc. can also be retrieved in the Static features in MAEC 5.0. Dex2jar is a reverse engineering tool that can convert classes.dex files extracted from APK file into JAR files. Using this tool, Java code can be obtained using JAR extracted from classes.dex files[14]. Java decompiling tools like jdx and jd-gui can be used for checking the Java source code from the extracted JAR file. The extracted Java code appears similar to the original, so source level analysis is possible. JEB is a commercial tool, but it has the advantage of inserting APK files or DEX files as input data and seeing results of dex2jar, Java code, and apktool as a result of smali code[15]. It also provides a cross-reference function to track the call relationship of a function or variable, allowing faster analysis. If it can be restored in the original source form, it can be applied to MAEC by confirming the contents of the API call.

3.2 Android malware dynamic analysis

3.2.1 Existing dynamic analysis method

The main executable code can sometimes be hidden from decompiling an APK file using static analysis tools due to various solutions to protect the app. For example, the actual loading function is hidden using Java's reflection API, or the executable code encryption technique causes the main executable code part to dynamically decompress (or decrypt) into memory during runtime. In this case, it is limited when running static analysis alone, therefore the app needs to be analyzed dynamically for further investigation. In the case of Android malware dynamic analysis, there are various methods such as analyzing logs by connecting ADB to a rooted phone and using an emulator[16]. One can also take advantage of the dynamic analysis tools available from various companies.

3.2.2 Our dynamic analysis method

In this study, we used our own dynamic analysis platform and constructed it to monitor malware, from sensitive APIs to system calls in SDK to NDK. Extract the events received from the main activities, services, and broadcast receivers from the app's AndroidManifest.xml file. It creates a process that records the analysis log in real-time before running the app. At this time, record the analysis log using the modified logcat so that the log cannot be deleted. Then run the main activity extracted from the app. When the main activity is executed, the process of the app is created. If the main activity of the app does not exist, the app process is created when the service is first executed. After executing the main activity, all services included in the app are executed to improve code coverage, and all intent events received from the broadcast receiver are executed. After that, the UI event is randomly generated with the Monkey tool to test the app.

When the dynamic analysis process is completed, the recorded dynamic analysis log is collected, the Android is stopped, and the analysis is terminated. A total of 153 behaviors are defined in the MAEC specification. There are a total of 131 behaviors related to mobile malware and can be declared using Android dynamic analysis data such as capture camera input, capture GPS data, encrypt data, encrypt files, etc. These traces persist after system reboot and steal SMS database is included. In addition, the JSON file classified as malware

action is declared for actions such as connect to IP address, call library function, create files, and create threads. In this case, further analysis of system calls should be conducted. Section 4 shows an example of identifying these behaviors.

3.3 Verification by many AV vendors using VirusTotal

VirusTotal is a website that provides free file checking [21] by using up to 70 different virus scanning software products. The files to be scanned can be uploaded via a website or e-mail; up to 128MB Analysis results can be retrieved by public or private APIs by importing HTTP Post using the issued token. Scanned results of each vendor display detected (true or false), version, and update. Based on this information, it is possible to output the result of MAEC 5.0 x-maec-avclass.json file. The date of scan, the date of submission, the result of detection, the name of the malware classified from the vendor, and AV tool name, and engine version etc. of each vendor can be retrieved [23].

4. Definition for automatic generation of Android malware

This section identifies requirements to automatically convert Android malware to MAEC. In the case of MAEC, it is divided into 20 files. First, some automatic conversion files, the files that need to be created, and the explanation of automatic conversion function for each part. Among the automatically convertible files, the parts that need to be created in detail are specified separately. **Table 1**. below shows the files that can be converted automatically in MAEC 5.0 and the files that require to be created.

Table 1. Automatically generated file classification

Generated by automatic tool	Generated by human being
analysis-metadata.json	field-data.json
collection.json	malware-development-environment.json
api-call.json	external-reference.json
behavior.json	relationship.json
binary-obfuscation.json	process-tree-node.json
capability.json	
malware-action.json	
dynamic-features.json	
malware-family.json	
malware-instance.json	
name.json	
package.json	
static-features.json	
x-maec-avclass.json	
signature-metadata.json	

There are cases where it is possible to automatically convert each of these JSON files, and some areas may not be automatically converted even in a JSON file. For example, in the case of the comment property in analysis-metadata.json, this is the area that the analyst should add to the comment. These parts cannot be converted automatically and are the parts that the analyst should enter later. The following is an explanation of the contents of automatically converting each JSON file from the analysis result.

In the case of Analysis-metadata.json, it shows the analyzed version, start time, operation time, etc. This information is obtained by storing the information of the definite and dynamic analyzer that the initial analyst constructed in the database, separating the fixed value and the fluctuation value parsing possibility through the query.

In the case of Collection.json, the grouping is performed based on the static and dynamic analysis results by displaying the association between the objects defined in MAEC. For example, if a process that compresses a file and sends a socket is regarded as a data leak, these operations are grouped and defined as a rule in advance, and the relationship is displayed when such an action is sequentially logged.

For the Api-call.json file, we use the static and dynamic analysis to analyze the system calls that are called from the Java API and the source code that the Android app calls. We then record the function names, parameters, and return values. In particular, when the API is called with a return value, we can create value through the dynamic analysis hooking log for the returned value.

In the case of Behavior-json, we can define and use behaviors that malware performs based on the Android API and system call. The following **Table 2**. shows examples analyzed based on action and API call for behavior definition.

Table 2. Example of Action based classification

Behavior	Action
capture system memory	read from process memory
erase data	Delete file
identify	OS Get process environment variable
install legitimate software	load library
send system information	Enumerate libraries Get process environment variable
	Get system hostname Connect to IP address
	Connect to socket address
	Connect to socket
	Send data on socket
	Send http connect request
	Send http get request
	Send http post request
	Send http put request

For the Binary-obfuscation.json file, we use the open-source project APKiD to identify the obfuscation and packer tools used by malware (e.g. DexGuard) and record obfuscation

techniques applied to malware and techniques used in obfuscation tools (e.g. code encryption). However, since obfuscation and packer techniques were developed recently, additional obfuscation and packer detection techniques should be applied.

In the case of the Capability.json file, it is possible to output the system call confirmation that detects the environment using gdb, confirmation of the rebooting process, etc. as logs related to environment detection, system access, reboot, etc.

For the Malware-action.json file, we use dynamic analysis to define the malware activity that the malware instance performs based on the Android Java API and system calls actually made by the application. An instance of a malware refers to a Java API or system call that is called for the execution of an API call type and an action and specifies an operation for accessing a directory and a file. Examples of malware-actions are shown in the **Table 3**. below. In the case of the SDK, it refers to the behavior occurring in the Java domain. In the case of the NDK, it refers to an action using the SO file.

Table 3. Example of API call-based classification

Behavior	Action
create directory	(SDK)mkdir,(NDK)mkdir
create file	(SDK)File Construct
create service	(SDK)startService
create socket	(SDK)Socket
create thread	(SDK)Load Thread using Class.forName
delete directory	(SDK)(NDK)rmdir
delete file	(SDK)File.delete(NDK)unlink
enumerate libraries	(SDK)(NDK)Access /system/lib or /system-1/lib
enumerate processes	(SDK)ActivityManager.getRunningAppProcesses
find file	(SDK) vlistFiles, (NDK) stat,lstat
load library	(SDK)System.load, System.loadLibrary, DexClassLoader
open directory	(SDK)Access directory using FileConstruct, (NDK)open, access
open file	(SDK)File.Constructor, (NDK)open
read from process memory	(SDK)(NDK)Access /proc/(pid_or_self)/maps

In addition, existing research on dynamic analysis defines various behaviors for Android analysis results. According to TRAPDROID, they defined the behavior characteristics for system call and Binder transaction in 7 categories including Network, File system, Process, and Injection[18]. According to Y Zhang et al. they have proposed a system called VetDroid to identify all calls to the Android API and define behavior by checking how resources are used[19]. However, VetDroid has been monitoring through rooting and it has been difficult to cope with recent intelligent malware. According to O Somarriba et al. defines a behavior detection rule to distinguish malware family using SMS[20]. Also, it shows how each API can act on API usage. For example, the use of an API such as getApplicationInfo() was associated with Search Service behavior, and in the case of getIMEI, Utils behavior. According to Fan Yuhui et al., they classify the classifiable behaviors based on the permissions used in static

analysis and dynamic analysis[21]. In static analysis, 17 behaviors were classified and in dynamic analysis, 5 behavioral groups were classified. These results are difficult to classify and judge the malware as a whole, although it is difficult to express them in the MAEC because they use their own analyzers to define and express the behavior for the analyzable parts. Therefore, in this paper, we declared all 131 dynamic analysis activities and defined the related action. **Table 4**. below is a comparing the number of behaviors defined in various studies and the number of dynamic analysis behaviors classified in this study.

Table 4. Various Malware behavior classification in previous work

Approaches	Feature	Characteristic
TRAPDROID	System call and binder transaction	7 behaviors are represent
VetDroid	API Call	Need monitoring tool to check all API calls, but do not have to represent behaviors.
Fan Yuhui et al.	33 static feature, 24 permission feature, 7 file permission	Classify only five behavior based on various features
Gagandeep Singh et al[22].	Permission, System resource CPU, Memory, Traffic)	Define malware behavior in 20 categories
Our Scheme	Permission, API Call, System Call, Activity, Service, etc.	131 behaviors in dynamic analysis. 95 features automatically converted to MAEC

In the dynamic-feature.json file, depending on the type of STIX network traffic, the malware performs the configuration for the actual operation and network traffic in the instance. In the case of a network, a tool such as TCPdump is used to monitor the actual network traffic and to check the specific ports.

The Malware-family.json file, the name.json file, and the x-maec-avclass.json file are based on the results obtained using the VirusTotal API. The malware family obtained from VirusTotal is fully identified, and the malware-family.json file can be created through most family classifications. In the case of x-maec-avclass, the VirusTotal result is shown as described in Section 3.

For the Malware-instance.json file, we use static and dynamic analysis to refer to the dynamic features, static functionality, and metadata of the malware instance. We then, record the analytical environment in which the family and malware run.

In the case of the Package.json file, it converts the malware instance and family into the standard output format. It then expresses the reference values for malware-family.json, name.json, and the like from the past. Also, the version, architecture, etc. of the analyzing system are applied as fixed values.

In the Static-feature.json file, it records the string contained in the obfuscation tool, packer tool, and header information used, extracts the certificate in the APK file from the static analysis stage, and parses the information. In addition, extraction results for other files such as image files included in the APK file would be created. The list of files extracted in the APK file static analysis step is automatically inputted into the database, which can be called through the query.

In the case of Process-tree-node.json, the dynamic analysis should be performed to obtain information such as parent_action_ref by checking the generation relationship between the parent process and the child process through gdb. To achieve this, we output the log using a command like pod of program_name for process monitoring and fetch it later.

In the case of Signature-metadata.json, it is not applicable in this study, but it can be applied automatically to various AV vendors. Signature information of various malware and, reference material for malware maker is required. It can be applied to AV vendors having a large amount of data by mapping. However, in this study, it is out of scope to collect a database of AV vendors.

In the case of a relationship.json that cannot be analyzed automatically, it requires analysis of using the correlation between one application and another at the same time. For example, when a vaccine app is executed before the execution of a banking application, which should extract information about your app. In cases like this, automation is difficult because the analysis information of other apps is required and the reference model to be compared has to be mapped.

Filed-data.json lists information about where malware started when it was first discovered, and so on. The malware starting point cannot be included in the static or dynamic analysis and should be based on the information held by the analyst.

In the case of malware-development-environment, it is difficult to identify information about the tools used, which should be written in the environment that the malware was produced. Of course, it is possible to check information about obfuscation and packing tools, but it is not possible to accurately grasp the tools (e.g. android studio, eclips, etc.) in which actual source code is developed.

Finally, relationship.json is the part that includes the relationship with other malicious apps. It reflects the results of other malicious apps and the analyst's opinion. After the analyst verifies the characteristics and analysis results of the app through its family name where an app contains the same source code, it is difficult to automatically, statically and dynamically analyze one app to express its relationship with other malicious apps. Therefore, this part should be accompanied by an additional process for mapping based on various databases in AV Vendors.

5. Evaluation

This section shows a number of features that can be automatically changed when automatic conversion to MAEC occurs. We calculated the automatic conversion rate through our Android malware and dynamic analysis system. MAEC.py created a rule-based on static analysis and dynamic analysis results. It performs the role of automatically converting input values based on the result of static analysis. It includes basic contents based on Package name, time, analysis information, Permission, etc. taken from static analysis. API calls from dynamic analysis, and actions defined based on system call information. As described above, the rule was created based on call information in accordance with the MAEC standard and was automatically converted according to the rule matching result

Table 5. MAEC 5.0 Format automatic generation rate

Format	Feature	Automated	Rate (%)
Anlaysis-metadata.json	12	11	91.7
collection.json	6	5	83.3
api-call.json	4	3	75
behavior.json	8	6	75
binary-obfuscation.json	7	4	57.1
capability.json	6	5	83.3
malware-action.json	9	9	100
dynamic-features.json	4	3	75
malware-family.json	12	12	100
Malware-instance.json	16	12	75
name.json	2	1	50
package.json	6	6	100
static-features.json	6	6	100
x-maec-avclass.json	8	8	100
signature-metadata.json	7	4	57.1
process-tree-node.json	4	0	0
field-data.json	3	0	0
malware-development-environment.json	2	0	0
external-reference.json	4	0	0
relationship.json	6	0	0
Total	132	95	72

5.1 Behavior automatic generation rate

It is not possible to automate 17 behaviors that are not performed in mobiles such as test-for-firewall, persist-after-os-changes, and persist-after-hardware-changes. As our platform targets application testing, we do not automate the content that requires external connection actions, such as prevent-concurrent-execution.

Table 6. Behavior automatic generation rate

Format	Feature	Automated	Rate %	Rate for Exclude other OS attributes %
Behavior	153	131	85.6	96.3

5.2 Action automatic generation rate

It is impossible to create and automate 68 items that are not made in mobile, such as Add-windows-hook, create-window, and enumerate-windows. It is difficult to automate the

actions for content that has not been defined in the past in rules such as create-critical-section, create-semaphore, etc. In the case of the action part, it represents a lot of the internal operation and the network behavior of the system, so tracing of the system call and analysis of the network packet should be performed simultaneously.

Table 7. Action automatic generation rate

Format	Feature	Automated	Rate %	Rate for Exclude other OS attributes%
Behavior	194	117	60.3	92.8

5.3 Capability name automatic generation rate

It is difficult to confirm persistence through the analysis of malicious apps and dynamic analysis or confirm the propagation and persistence through device infection due to the characteristics of the analysis tool. Also, the following three areas such as persistence, infection-propagation, and availability-violation are impossible to perform.

Table 8. Capability name automatic generation rate

Format	Feature	Automated	Rate %
Capability	19	16	84.2

5.4 Entity Associations automatic generation rate

File system entity, network entity, and process entity may be automatically inputted into Collection.json files. 100% auto-input is possible.

Table 9. Entity Associations automatic generation rate

Format	Feature	Automated	Rate %
Entity Associations	13	13	100

5.5 Common Attribute automatic generation rate

Capabilities.json and behavior.json are attributes that can be expressed mostly but are not linked with Common Vulnerabilities and Exposures (CVE) or Open Source Vulnerability Database (OSVDB). Also, in this analysis, it is impossible to express the timing of malicious action triggers because the logic bomb does respond properly.

Table 10. Common Attribute automatic generation rate

Format	Feature	Automated	Rate %
Common Attribute	28	25	89.2

5.6 Compare with existing method

To demonstrate the feasibility of the automation process, we measured the automation rate and time. Existing MAEC 5.0 does not provide the automatic conversion. However, for comparison purposes, we have identified components that do not change, such as analyst name and analytic OS. We determined that this was an automation area in the existing MAEC 5.0. The results of the automation rate are as follows.

Table 11. Automatic generation rate

Format	Feature	Automated	Rate %
Existing MAEC	132	13	1.36
Proposed Scheme	132	95	71.9

And we measured time. Conventional MAEC 5.0 and STIX creations cannot be compared for a time as the analysts enter them directly. Therefore, we measured the time varying according to the amount of log extracted from the analyzed APK. Logs extracted from static analysis results include information such as Permission, Activity, and Service. Dynamic analysis results include information obtained through hooking. We have measured the time taken from input to finally storing it in the database. The result is shown in the figure below. Based on this result, it can be seen that even if a lot of logs occur, it can be automatically converted in a short time compared to the time input by the analyst.

Table 12. Common Attribute automatic generation rate

Static Analysis Log	Dynamic Analysis Log	Time (s)
143	3780	4.32
172	851	2.76
84	212	2.1
64	151	1.84
97	254	2.34
121	81	1.65
118	1397	3.87
169	557	1.98
131	648	2.13

6. Implementation Result

The flow for the automatic conversion of malware for Android is shown in **Fig. 1**. below. When APK is received and dynamic analysis is performed and the result of VirusTotal is received, each received JSON file is subjected to primary transformation in MAEC form. Thereafter, the MAEC file is automatically transformed into STIX and applied to the secondary transformation.

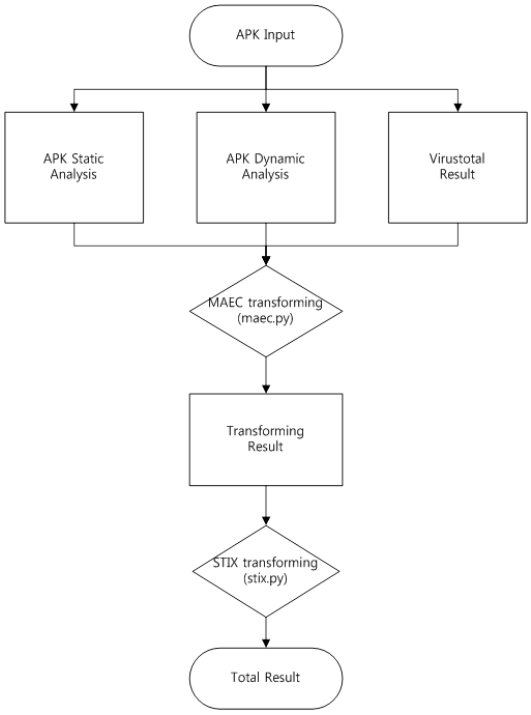
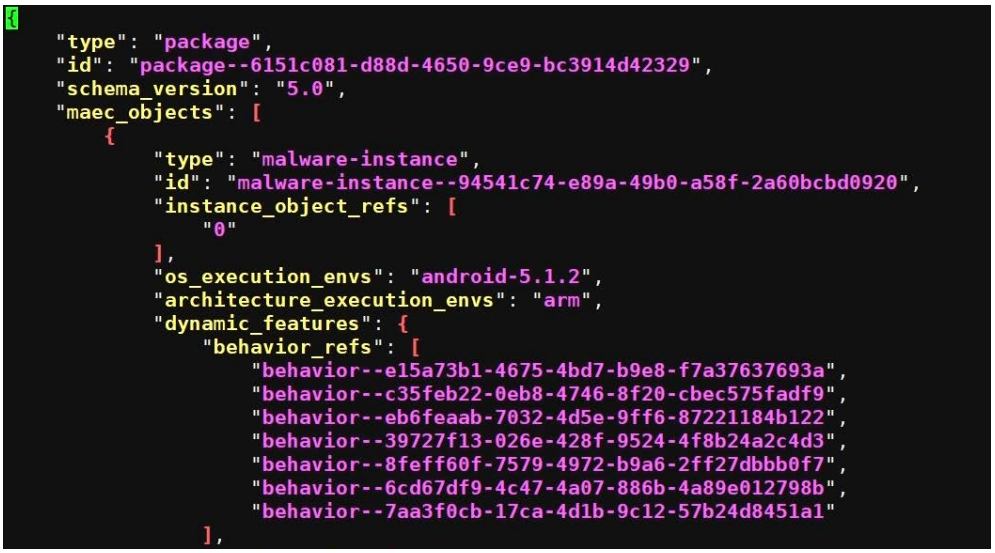


Fig. 1. Android malware to MAEC automatic generation flow

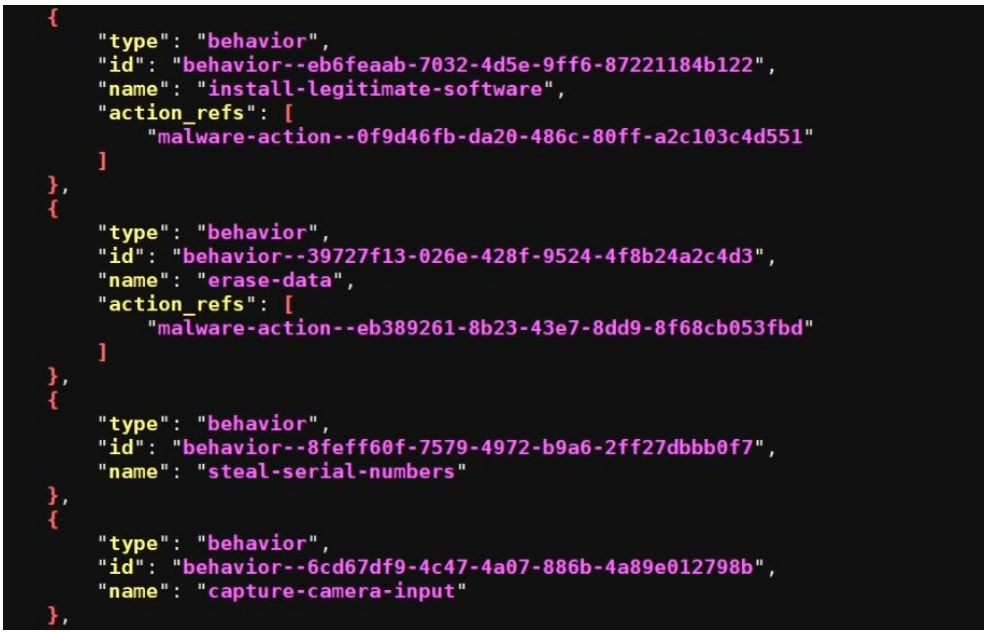
The result of this transformation is shown in the figure below. **Fig. 2**. shows the MAEC type, version, etc. when it is converted to MAEC. Also, the OS version and architecture that performed the operation will be described as well as the dynamic feature. The reference value of the behavior is automatically generated, and the value of the reference value is additionally provided as shown in **Fig. 3**. below. The behavior is mapped to the action and all of these are included in the report.

This is converted to the MAEC standard and finally converted to the STIX format. When converting to STIX format, the result parsed from MAEC is expressed as STIX. We found a MAEC-to-STIX tool, which automatically converts MAEC to STIX, but it has not yet been finalized for MAEC 5.0. For MAEC-to-STIX, it provides APIs and scripts for wrapping MAEC packages in STIX and extracting STIX indicators from dynamic analysis data captured at MAEC[17]. The following figure is a screen converted to STIX. Unlike MAEC, the initial type is changed, but the id value is the same. Generated time, malware label, etc., and show the contents expressed in MAEC as being useful for sharing.



```
{
    "type": "package",
    "id": "package--6151c081-d88d-4650-9ce9-bc3914d42329",
    "schema_version": "5.0",
    "maec_objects": [
        {
            "type": "malware-instance",
            "id": "malware-instance--94541c74-e89a-49b0-a58f-2a60bcbd0920",
            "instance_object_refs": [
                "0"
            ],
            "os_execution_envs": "android-5.1.2",
            "architecture_execution_envs": "arm",
            "dynamic_features": {
                "behavior_refs": [
                    "behavior--e15a73b1-4675-4bd7-b9e8-f7a37637693a",
                    "behavior--c35feb22-0eb8-4746-8f20-cbec575fadf9",
                    "behavior--eb6feaab-7032-4d5e-9ff6-87221184b122",
                    "behavior--39727f13-026e-428f-9524-4f8b24a2c4d3",
                    "behavior--8feff60f-7579-4972-b9a6-2ff27dbbb0f7",
                    "behavior--6cd67df9-4c47-4a07-886b-4a89e012798b",
                    "behavior--7aa3f0cb-17ca-4d1b-9c12-57b24d8451a1"
                ],
```


Fig. 2. MAEC transformation result



```
{
    "type": "behavior",
    "id": "behavior--eb6feaab-7032-4d5e-9ff6-87221184b122",
    "name": "install-legitimate-software",
    "action_refs": [
        "malware-action--0f9d46fb-da20-486c-80ff-a2c103c4d551"
    ]
},
{
    "type": "behavior",
    "id": "behavior--39727f13-026e-428f-9524-4f8b24a2c4d3",
    "name": "erase-data",
    "action_refs": [
        "malware-action--eb389261-8b23-43e7-8dd9-8f68cb053fbd"
    ]
},
{
    "type": "behavior",
    "id": "behavior--8feff60f-7579-4972-b9a6-2ff27dbbb0f7",
    "name": "steal-serial-numbers"
},
{
    "type": "behavior",
    "id": "behavior--6cd67df9-4c47-4a07-886b-4a89e012798b",
    "name": "capture-camera-input"
},
```


Fig. 3. Explain for behavior

7. Conclusion

Recently, various versions of Android malware has been increasing in number. Various types of malware are introduced by different countries. malware increasingly becoming intelligent enough to prove difficult to analyze. In order to respond and prepare for these increasing malware, countries such as the US as well as nations in Europe have established a cyber threat sharing system. The DHS created a standard to express cyber threat information and it can be used in a wide range to include international standards because they use

automatic sharing and automatic search queries. In addition, various global corporations, US government agencies, and the European Union(EU) are currently promoting cyber threat information sharing systems using STIX. MAEC has been offered as an analytical standard for Android and iOS since 2017, as malware has become more common in mobile environments. In the past, mobile environments had to be linked to PC environments for malware analysis and sharing, which made. It was difficult to automate and organize the malware due to different parts. However, recently these problems have been solved, and the malware analysis result can be sufficiently provided. In summary, we first performed the automatic conversion of Android malware analysis to MAEC format. Since MAEC is not fully automated, we automated many parts using static analysis, dynamic analysis, and VirusTotal results analysis as well as working with analysts' opinions and existing analytical results. One can then establish an efficient and fast sharing system. If the analysis result is properly parsed into MAEC, and then implemented using open source tools such as MAEC-to-STIX provided by MITRE, it is possible to express it in an accurate sharing system. However, intelligent applications with technologies that interfere with static analysis such as obfuscation, packing, etc., along with interference with dynamic analysis such as routing detection, emulator detection, etc., have emerged, and, as result, further research should be conducted.

References

[1] Asgarli, Elchin, and Eric Burger, "Semantic ontologies for cyber threat sharing standards," in *Proc. of Technologies for Homeland Security (HST), 2016 IEEE Symposium on. IEEE*, 2016. Article (CrossRef Link)

[2] Barnum, Sean, "Standardizing cyber threat intelligence information with the Structured Threat Information eXpression (STIX)," *MITRE Corporation*, 11, 2012

[3] MAEC 5.0 Core Specificaiton, https://maecproject.github.io/releases/5.0/MAEC_Core_ Specification, 2017

[4] Kampanakis, Panos, "Security automation and threat information-sharing options," *IEEE Security & Privacy*, 12(5), 42-51, 2014. Article (CrossRef Link)

[5] Kim, Eunsoo, et al., "CyTIME: Cyber Threat Intelligence ManagEment framework for automatically generating security rules," in *Proc. of the 13th International Conference on Future Internet Technologies. ACM*, pp.1-5, 2018. Article (CrossRef Link)

[6] Johnson, Chris, et al., "Guide to cyber threat information sharing," *NIST special publication 800-150*, 2016. Article (CrossRef Link)

[7] Yavvari, Chaitanya, et al., "Malware characterization using behavioral components," in *Proc. of International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security. Springer, Berlin, Heidelberg*, pp. 226-239, 2012. Article (CrossRef Link)

[8] Lee, Alan, Vijay Varadharajan, and Udaya Tupakula, "On malware characterization and attack classification," in *Proc. of the First Australasian Web Conference-Volume 144. Australian Computer Society, Inc*., 2013

[9] Pektaş, Abdurrahman, and Tankut Acarman, "Classification of malware families based on runtime behaviors," *Journal of Information Security and Applications*, 37, 97-100, 2017. Article (CrossRef Link)

[10] MAEC 5.0 specification Vocabularies, https://maecproject.github.io/ releases/5.0/MAEC_Vocab ularies_Specification, 2017

[11] Apoorva, M., et al., "A latest comprehensive study on structured threat information expression (STIX) and trusted automated exchange of indicator information (TAXII)," in *Proc. of the 5th international conference on frontiers in intelligent computing: theory and applications. Springer, Singapore*, 477-482, 2017. Article (CrossRef Link)

[12] Tosh, Deepak K., et al., "Cyber-investment and cyber-information exchange decision modeling," in *Proc. of High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on. IEEE*, 2015. Article (CrossRef Link)
[13] Android-apktool: A tool for reengineering Android apk files, https://ibotpeaches.github.io/Apktool, 2018
[14] Yen, Yao-Saint, and Hung-Min Sun, "An Android mutation malware detection based on deep learning using visualization of importance from codes," *Microelectronics Reliability*, 93, 109-114, 2019. Article (CrossRef Link)
[15] JEB, https://www.pnfsoftware.com/, 2019
[16] Verma, Neha, Sarita Kansal, and Huned Malvi, "Development of Native Mobile Application Using Android Studio for Cabs and Some Glimpse of Cross Platform Apps," *International Journal of Applied Engineering Research*,13(16), 2018
[17] MAEC-to-STIX, https://github.com/MAECProject/maec-to-stix, 2016
[18] Alptekin, Halit, et al., "TRAPDROID: Bare-Metal Android Malware Behavior Analysis Framework," in *Proc. of 2019 21st International Conference on Advanced Communication Technology (ICACT). IEEE*, 2019. Article (CrossRef Link)
[19] Zhang, Yuan, et al., "Vetting undesirable behaviors in android apps with permission use analysis," in *Proc. of the 2013 ACM SIGSAC conference on Computer&communications security. ACM*, 611-622, 2013. Article (CrossRef Link)
[20] Somarriba, Oscar, et al., "Detection and visualization of android malware behavior," *Journal of Electrical and Computer Engineering*, vol.2016, 2016. Article (CrossRef Link)
[21] Yuhui, Fan, and Xu Ning, "The Analysis of Android Malware Behaviors," *International Journal of Security and Its Applications*, 9(3), 335-346, 2015. Article (CrossRef Link)
[22] Jaafar, Fehmi, Gagandeep Singh, and Pavol Zavarsky, "An Analysis of Android Malware Behavior," in *Proc. of 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C). IEEE*, 2018. Article (CrossRef Link)
[23] Virustotal, https://www.virustotal.com/gui/home/upload, 2019

Jungsoo Park received the B.S and M.S. degree in Electronics Engineering from Soongsil University in 2013 and 2015, respectively. He is currently a PhD student in Software Convergence at Soongsil University. His research interests include cloud security, mobile security, and Machine Learning.

Long Nguyen-Vu received B.S in Vietnam National University of Information Technology in 2012 and received his M.S. degree in Electronics Engineering from Soongsil University in 2016. He is currently a PhD student in Electronics Engineering at Soongsil University. His research interests include cloud security, mobile security, and Machine Learning.

George Bencivengo received his B.S degree in Computer Science from George Mason University in 2016. He is currently a Cyber-security and Information Assurance consultant at GINIA and has supported contracts with the USDA and DHS. His research interests include cloud security, mobile security, and machine learning.

Souhwan Jung received the B.S and M.S. degree in Electronics Engineering from Seoul National University in 1985 and 1987, respectively, and the PhD degree from the University of Washington, Seattle, USA in 1996. From 1996 to 1997 he was a senior software engineer at Stellar One Corporation, Bellevue, USA. In 1997, he joined the School of Electronic Engineering at Soongsil University, Seoul, Korea, and currently serves as a professor. He is an executive director of the Korea Institute of Information Security and Cryptology. He was also a R\&D program director of Ministry of Knowledge Economy in Korea for information security area from 2009 to 2011. His research area includes wireless network security, cloud security, mobile security, Machine Learning, and IoT security.