# Integration of outputs from tools for static and dynamic code analysis into an ontological model

Štefan Balogh, [1] and Peter Švec [1]

[1] Faculty of Electrical Engineering, Slovak Technical University, Bratislava

**Abstract. I**n this paper we discuss the integration the outputs of analysis tools into an ontological model.

**Keywords:** dynamic analysis, ontological model.

## 1 Tools for static and dynamic code analysis

### 1.1 Holmes processing

The designed system for static and dynamic analysis allows you to enter samples via a web interface or script, automatically or manually run static and dynamic analysis, save samples, display results, and add results to the ontology. The Holmes Processing project was used as the basis of the system [1]. Fife modules were selected from the Holmes Processing project, namely Gateway, Storage, Totem, Totem-Dynamic, Interrogation and Frontend. They have been combined into one project to facilitate installation.

### 1.2 Static analysis

The system uses services running using the Docker tool for static analysis. For the demonstration of the solution was created one static service that obtains information about PE files. For analysis, the Python module called pefile is used, with which it is possible to extract data from the header from executable files such as: imported libraries, strings, sections and other data. It also allows you to determine whether packer or obfuscation was used when creating the file based on the entropy of each data section. The result of the analysis is transformed into MAEC format so that it can be added to the ontology and sent directly to the REST service for conversion to OWL. A pefile-to-maec library [2] from Mitre was used for data extraction and conversion. It was necessary to transform the output to MAEC 5.0 because the library only supports conversion to MAEC 4 in XML format. The service uses the pefile Python library to obtain all values from the PE header and maps the key names to MAEC format. Then the resulting python dictionary is converted to JSON in MAEC format according to the STIX definition. Data such as size, data sections, their entropy and optional headers such as linker version, operating system version and the like are selected from the pefile to the MAEC.

### 1.3 Dynamic analysis

The system uses two tools for dynamic analysis. Cuckoo [3], which is one of the most popular tools, and Drakvuf [4], which is difficult for malware to detect. The existing Cuckoo Sandbox service in the Holmes system did not support obtaining results in MAEC format [5]. It was therefore necessary to add this functionality to the service. The Cuckoo tool itself also had to be modified. Cuckoo only supports MAEC results with a Mitre plug-in.

The Drakvuf Sandbox is used to work with the Drakvuf analyzer. For connecting the Holmes system to the Drakvuf Sandbox a separate service was created. Its function is similar to the service for the Cuckoo tool, which means that it is used as a Docker container, which offers a REST API for communication with Holmes and queries the Drakvuf Sandbox tool where it creates new tasks, finds out the status and obtains results. In addition, it also processes the obtained results and creates a valid JSON document from them in a format similar to Cuckoo. Thanks to the use of the same output format as Cuckoo, it is possible to use a modified tool from the Mitre company to convert it to MAEC format. which gives almost the same results as the conversion from Cuckoo. The same results in the conversion are necessary to combine the outputs when saving to the ontology. To use the same conversion tool, the service had to be implemented in Python, unlike the Cuckoo service, which is in Go. The service creates HTTP queries on the Drakvuf Sandbox API which allows it to create a new task, find out the status of the task and get the results of the analysis.

## 2 Ontology model

To store MAEC data obtained during analysis in ontology, it is first necessary to convert them to OWL format. The own custom Java application was used to do it. Supported objects from the MAEC standard are malware-instances and malware-actions and they share one MAECObject class. One ObservableObject data class was created from the STIX standard for all objects, but not all properties but only those used in the ontology. After saving STIX objects, MAEC objects are converted. The MAEC objects are stored in a worksheet through which it is iterated, and the conversion method is called according to the object type. Currently, only malware-action and malware-instance types are supported. The malware instance is identified by the md5 hash of the STIX object that is its instance. Although there may be multiple instances, each must have in MAEC specification the same hash. The individual is therefore created from the hash of the first instance. In a malware action, the individual identifier in the ontology is created first. The identifier consists of hashes of input STIX objects (input_object_refs), hashes of output STIX objects (output_object_refs) and the action name. These values are combined into a single string to form an md5 hash. In a malware action, the name of the action, input STIX objects in the input relation, and output STIX objects in the output relation are stored in the ontology. The counter is in a "do" relationship with the process, such as “process to counter” and in relation "count" with

class Malware_Action, such as “counter count action.”. The counter identifier is created from the md5 hashes of the malware action and instance. These are connected into one chain and are of them created an md5 hash and added the word Counter at the end, for example “a00e5 ... Counter”. A role “initiated_malware_action” has also been added to the ontology to directly link process and action. The identification of individual elements of the ontology is as follows:
• Malware_Instance - md5 hash of the STIX object.
• Malware_Action - md5 hash from md5 hashes of all input and output objects combined into one string and action name.
• ObservableObject - Either an existing md5 hash file or an md5 hash from a string created by combining a type, name, path, command line, key, and mime type.
• Counter - an md5 hash created from a string joined from an md5 hash action and an md5 hash instance.
After saving instances and actions, the processes obtained from the dynamic analysis are saved. It goes through the list of all stored instances and its processes are connected to each via OWLObjectProperty instance-process. It is handled that the method for saving processes is skipped if the sheet is empty, so if it has no processes. Each process connects only once, even if it is called multiple times, because the identifier consists of a hash of the detected data.
The conversion speed was tested on two files, the first with 28729 malware actions and 109 STIX objects and the second with 15668 actions and 1117 objects. Conversion of the first file in about 2 seconds. The conversion time of the second file was approximately 5 seconds. Application optimization brought the required conversion acceleration.

## References

1. WEBSTER, G., HANIF, Z., LUDWIG, A., LENGYEL, T., ZARRAS, A., and ECKERT, C. Skald: A scalable architecture for feature extraction, multi-user analysis, and real-time information sharing. vol. 9866, pp. 231–.
2. The MITRE Corporation. pefile-to-maec [online]. [cit. 24.04.2020]. dostupné z: https://github.com/MAECProject/pefile-to-maec, 2017.
3. Oktavianto, D., & Muhardianto, I. (2013). *Cuckoo malware analysis*. Packt Publishing Ltd.
4. LENGYEL, T., MARESCA, S., PAYNE, B., WEBSTER, G., VOGL, S., and KIAYIAS, A. Scalability, fidelity and stealth in the drakvuf dynamic malware analysis system.
5. The MITRE Corporation. Maec - malware attribute enumeration and charac-terization [online]. [cit. 29.03.2020]. dostupné z: https://maecproject.github.io/, 2020.