Malware Attribute Enumeration and Characterization

4 authors, including:

Penny Chase
MITRE
5 PUBLICATIONS **102** CITATIONS
SEE PROFILE

Robert A. Martin
MITRE
37 PUBLICATIONS **427** CITATIONS
SEE PROFILE

Malware Attribute Enumeration and Characterization

Ivan Kirillov
MITRE Corporation
202 Burlington Road
Bedford, MA 01730
1-781-271-5364
ikirillov@mitre.org

Penny Chase
MITRE Corporation
202 Burlington Road
Bedford, MA 01730
1-781-271-2113
pc@mitre.org

Desiree Beck
MITRE Corporation
202 Burlington Road
Bedford, MA 01730
1-714-915-0310
dbeck@mitre.org

Robert Martin
MITRE Corporation
202 Burlington Road
Bedford, MA 01730
1-781-271-3001
ramartin@mitre.org

MITRE

This page is intentionally blank.

Abstract

Malware represents one of the most prevalent threats to cyber security and is increasingly able to circumvent previously standardized detection, mitigation, and characterization techniques. Although new methods for combating malware have been developed, it is still difficult to communicate and share useful information garnered through these techniques without ambiguity and corresponding data loss. To close this significant gap in malware-oriented communication, this paper introduces and defines a language for characterizing malware based on its behaviors, artifacts, and attack patterns.

Table of Contents

1. Introduction

Malicious software, or malware, has been around in one form or another since the advent of the first PC virus in 1971. In its various forms, from spyware to rootkits, it is presently responsible for a host of illicit activities, ranging from the vast majority of spam email distribution through botnets [1], to the theft of sensitive information via targeted social engineering attacks [2]. Effectively an autonomous agent operating on behalf of the attacker, malware has the ability to perform any action capable of being expressed in code and represents a prodigious threat to cyber security.

The protection of computer systems from malware is therefore currently one of the most important information security concerns for organizations and individuals, since even a single malware infection can result in damaged systems and compromised data. Being disconnected from a computer network does not completely mitigate this risk of infection, as exemplified by the recent wave of malware that utilizes USB as a vector [3]. As such, the main focus of a large number of anti-malware efforts to data has been on preventing damaging effects through early detection.

There are currently several common methods utilized for malware detection, based mainly on physical signatures and heuristics. These methods are effective in terms of their narrow scope, although they have their own individual drawbacks, such as the fact that signatures do not scale and are therefore unsuitable for dealing with zero-day, targeted, polymorphic, and other forms of emerging malware. Similarly, heuristic detection may be able to generically detect certain types of malware while missing those that it does not have patterns defined for, such as kernel-level rootkits. These methods, while still useful, cannot be exclusively relied upon to deal with the current influx of malware.

More modern methods for detecting and combating malware often rely on the characterization of malware *attributes* and *behaviors* [6-9]. Such behaviors and attributes are commonly discovered through the use of static [10] and dynamic [11] analysis techniques. The combination of the two allows for an encompassing profile of malware to be constructed based upon its disassembly and observed run-time behavior. Yet, such techniques are hampered by the non-existence of a widely accepted standard for unambiguously characterizing malware.

The lack of such a standard means that there is no clear method for communicating the specific malware attributes detected in malware by the aforementioned analyses, nor for enumerating its fundamental composition. Several major problems result from this, including non-interoperable and disparate malware reporting between organizations, disjointed or inaccurate malware attribution, the duplication of malware analysis efforts, increased difficulty in determining the severity of a malware threat, and a greater period of time between malware infection and detection/response, among others.

On this basis, it is clear that a standard for describing malware in terms of its *attack patterns* [12], *artifacts*, and actions is needed to address such issues and allow for the clear communication of the information gained using static and dynamic analysis. This is the approach being taken by the Malware Attribute Enumeration and Characterization (MAEC) effort. The characterization of malware using such abstract patterns offers a wide range of benefits over the usage of physical signatures. Namely, it allows for the accurate encoding of how malware operates and the specific actions that it performs. Such information can not only be used for malware detection but also for assessing the end-goal the malware is pursuing and the corresponding threat that it represents.

The rest of the paper is structured as follows. In Section 2, we lay out the history and major factors leading up to this work. Section 3 describes our objectives for this project. Section 4 briefly covers the scope of our proposed language. In Section 5, we discuss the high-level details of the MAEC framework. Section 6 describes a MAEC test case involving the characterization of a real malware instance. In Section 7, we cover MAEC's relationship with other MITRE standards. Section 8 details a number of MAEC use cases. Finally, section 9 discusses a number of issues and challenges that we have foreseen relating to the development of MAEC. A glossary of commonly used terms can be found at the end of the document in Appendix A (with terms being used for the first time highlighted in *italics*), while the full MAEC characterization described in Section 6 can be found in Appendix B.

2. Background

2.1 Malware Identification & Naming

Viruses and other malware are commonly identified by AV product vendors and others using the Computer Anti-virus Researcher Organization (CARO) naming scheme [14], first adopted in 1991. Although the value of knowing the specific malware threat that one is dealing with is significant, the fact remains that the CARO naming scheme is not an official standard and is not applied consistently among its adopters. As a result, it is often the case that a single malware instance has multiple, disparate CARO-based identifiers[1] thus furthering confusion regarding malware identity and subsequently reducing the value of the identifier.

A large part of the problem with a system like the one created by CARO is that it attempts to encode malware attributes as part of the identifier [15]. It is very difficult to include all of the important information regarding a malware instance inside its identifier without making the identifier too large and cumbersome to effectively utilize. Likewise, there is no way of determining which critical attributes should be present in the identifier and which to leave out, as certain attributes are relevant only to certain parties. It appeared that a better solution would be the use of standardized, non-attribute based malware identifiers.

2.2 CME

In the fall of 2004, MITRE began work on the Common Malware Enumeration effort. The goal of CME was to provide single, common identifiers for new and prevalent virus threats, in order to reduce public confusion during malware incidents. Like other MITRE security standards efforts, the intent was to collaborate with industry and develop a consensus-oriented approach. This community effort was not an attempt to replace the vendor names used for viruses and other forms of malware, but rather was

[1] W32/MyWife.d@MM!M24.
http://vil.nai.com/vil/content/v_138027.htm

W32.Blackmal.E@mm.
http://www.symantec.com/security_response/writeup.jsp?docid=2006-011712-2537-99

intended to facilitate a shared, neutral indexing capability for malware.

In the first quarter of 2005 an initial capability was stood up. Anti-virus vendors submitted malware samples to a submission server with some metadata, the CME Board (composed of AV researchers and MITRE engineers) decided when a CME identifier should be generated, and the CME team identified the mapping between this sample and vendor names and created the content for the CME web site. For a period of time, CME seemed to fulfill its purpose, with identifiers being issued for prevalent malware incidents, and likewise being referenced in popular reporting about such malware [17].

2.3 Malware 2.0

In early 2007, it was becoming readily apparent that a paradigm shift was taking place in terms of new malware releases. Instead of a small number of widely targeted, interspersed malware instances, there was a pronounced move towards a large volume of narrowly targeted, highly-related malware [4]. Accordingly, this influx of short-lived malware [5] served to circumvent signature-based detection techniques through its sheer volume.

It was therefore not surprising to see malware detection becoming increasingly reliant on heuristics and other non-signature based techniques [20]. As such, the use of a sample-based approach for mapping vendor names to unique CME identifiers was no longer sensible for heuristically-identified malware instances. Consequently, the last CME ID issued was assigned to the malware instance commonly referred to as the "Storm" worm, and the CME-related efforts were transitioned to supporting the DHS/DoD/NIST Software Assurance Malware Working Group.

Based on the changing malware landscape and our experiences with CME, it became clear that a formal language was needed for communicating the fundamental characteristics of malware. Describing malware in such a fashion would mitigate the challenges posed by *armoring* and *obfuscation* techniques, which hinder the development of physical signature and heuristics. Likewise, it would allow for improved malware detection and mitigation by way of providing a common format for relating any information garnered through analysis or observation.

2.4 DHS/DoD/NIST Malware Working Group

Shortly after the issuance of the last CME ID in early 2007, the DHS/DoD/NIST Software Assurance Malware Working Group was stood up with representatives from MITRE and the Anti-Spyware Coalition (ASC) as co-chairs. The Malware Working Group was created to develop a framework of descriptive attributes of malware in order to:

- Improve communication by characterizing patterns (attributes and behaviors) to identify and describe malware *types* and instances
- Enable users to make informed decisions, i.e., lead to more stringent user acceptance criteria if potentially malicious code is to be installed with user knowledge
- Enable legal definitions of malware, spyware, adware
- Provide objective criteria for anti-malware tool assessments

2.5 MAEC

This work by the Software Assurance Working Group laid the foundation for the Malware Attribute Enumeration and Characterization (MAEC) effort, with the initial focus being on a single use case, namely attempting to develop a legally defensible definition of malware. This was driven by the desire to control the installation of spyware and adware and the ASC's success in using the concept of "potentially unwanted behaviors" to put the discussion in a more neutral setting. In talking to members of the security community involved with malware, we saw the need to broaden the Malware Working Group's set of use cases in order to provide greater uniformity in reporting malware, standardize the results of malware analysis, and support the development of databases describing adversary tactics, techniques, and procedures.

3. Objectives

3.1 Non-signature Malware Characterization

Under this broader concept, MAEC's main function is to serve as a standard method of characterizing malware based on its behaviors, artifacts, and attack patterns. This will allow for the description and identification of malware based on distinct patterns of attributes rather than a single metadata entity (which is the method commonly employed in signature-based detection).

Characterizing malware using a standardized method is not a new concept; however, the primary means of doing so are still rooted in the development of some form of signature (e.g. physical, functional, etc.) and are narrow in scope. On the other hand, more detailed characterizations that exist tend to rely on narrative text to describe the outputs of dynamic and static analysis.

MAEC's focus on structured attribute-based characterization provides several capabilities that the aforementioned methods do not possess. These capabilities stem from MAEC's existence as a domain-specific language, with an encompassing and unambiguous vocabulary and grammar. Such a vocabulary and grammar will provide the means for standardized characterization and communication of any malware attributes, and will therefore enable the following capabilities.

3.1.1 Malware Grouping

By identifying the distinct attributes present in malware instances, the similarity of two or more malware samples can be gauged. In this fashion, malware with the same attributes can be grouped together. This will permit accurate association of functionally identical polymorphic/metamorphic malware samples, and therefore help eliminate the duplication of analysis efforts.

Similarly, such characterization enables the description of malware *variants* based on small differences. This permits the partitioning of malware instances based on shared and unique attributes. Malware families and their variants can therefore be identified by defining the base group of attributes common to a family, as explained in 3.3.

3.1.2 Blended Malware Threat Description

An increasing number of malware instances have characteristics that can be ascribed to multiple *types* of malware [13]. Currently, no common method exists for communicating that these instances share these normally separate characteristics; instead, they are simply typed based on their perceived dominant attributes. However, such blended malware threats can be accurately described on the basis of their constituent attributes, thereby eliminating any confusion as to their true function.

3.1.3 Mitigation of Armoring Techniques

Attribute and behavioral based malware characterization will also mitigate the challenges posed by *polymorphism*, *metamorphism*, encryption, packing, and other obfuscation and armoring techniques. These techniques focus on circumventing specific detection methods through various means; however, the core functionality, or semantics, of a malware instance will always remain intact, and can therefore be characterized based on its attributes.

3.2 Unambiguous Malware Typing

In general, classifying an object based on some pre-defined category is often useful for conducting systematic studies. For this reason, malware has been binned into types since its emergence as a serious security threat. However, without a standardized format for accurately identifying the characteristics that define these aforementioned malware types, such typing has been fairly arbitrary and inconsistent. As a result, this has lead to confusion about what exactly a malware type entails, and has subsequently diminished the value of malware typing.

This top-down, tree-structured approach to malware typing is also not appropriate for the description of blended malware threats, as detailed in Section 3.1.2. Having such a fixed tree for typing works only if the malware instance being typed contains attributes that are representative of a single malware type; however, once the malware contains attributes that may belong to two or more malware types, such a tree will fail to accurately classify the malware.

Likewise, such inaccurate and ambiguous typing methods can be problematic in conjunction with heuristic malware detection, where a generic malware type is reported instead of an exact malware instance. Since this reported malware type is information that can be used for threat assessment and response, an incorrectly reported type may lead to an inadequate response and augmented damaging effects. For example, a heuristic engine that incorrectly classifies a network worm as a trojan based on faulty typing may delay the normal quarantine operation that may be part of an organization's response to network worms and lead to further infections.

Therefore, due to the fact that the various types of malware in the wild have distinct sets of shared attributes, one aim of MAEC is to permit empirical and unambiguous malware typing based on these attributes. To accomplish this, a large sample set of malware would be characterized using MAEC. Afterwards, the malware instances that share a predefined number of certain types of high-level attributes (i.e. mechanisms) would be grouped together; each such group would therefore represent a malware type. While there is likely to be some level of overlap when composing such groupings, giving priority to certain shared attributes for tie-breaks should eliminate most such overlaps.

3.3 Improved Malware Pedigree Tracking

As with malware typing, the usage of MAEC in describing malware instances will permit the identification of the minimum set of lower-level attributes (i.e. actions and behaviors) necessary and sufficient for characterizing a particular malware family. This will enable the accurate identification of new instances of existing malware families through the comparison of their MAEC-characterized attributes with the unique set of attributes specific to a malware family. Such identification will also facilitate the identification of code sharing across families. In this manner, MAEC should enable accurate and unambiguous malware pedigree tracking.

3.4 Legal Malware Definitions

MAEC's enumeration of malware attributes and their corresponding end-goals will permit the establishment of the behaviors and attributes that have been commonly observed as belonging to malicious software. Along with further analysis of individual malware samples, this will enable the creation of legal definitions of malware.

For example, back-up software that copies files from a user's hard disk to an off-site web server is typically classified as benign if it is installed by the user and the aforementioned actions are performed at the behest of the user. A program that performs the same actions can be classified as malicious if it is intended to steal files by being installed and executed transparently, without the authorization and permission of the user. Thus, in this second case, transparency of *insertion* and non-user initiated execution are the attributes that differentiate benign versus malicious intent, and these important attributes would be included in the MAEC characterization of the program.

However, the majority of software behaviors can be classified as being either benign or malicious, with the context of their use quite often being the sole differentiator between the two. A program, with the same exact MAEC characterization as the malicious example described above, could also be used for benign reasons by a system administrator to back-up a user's system without the user's knowledge.

Clearly, MAEC could be used to define the groups of behaviors and attributes that have the *potential* to be malicious based on past observation. If a behavior has been observed multiple times in malware and has been identified as being consistently used for information theft, chances are that any other program that implements that same behavior is malicious too, as long as it implements the other known malicious behaviors. However, the concrete establishment of the intent of a piece of software based on its attributes and behaviors is a complex problem and one that is outside the scope of MAEC.

4. Scope

MAEC is being developed as a language for addressing all known types, variants and manifestations of malware. Although initial work focuses on supporting the characterization of the most commonly discussed types of malware (such as, drive-by-downloaders, trojans, worms, kernel rootkits, etc.), MAEC will be applicable to the characterizing of any of the more esoteric malware types. This can range from malware that is currently only a proof-of-concept, such as a hypervisor rootkit, to firmware and malware introduced into a product through its development activities and components, tools and library supply-chain.

Accordingly, MAEC is intended to standardize malware characterization and communication, and subsequently provide the wider security community with a useful platform for conducting and integrating their anti-malware operations. MAEC is not meant to supply unique identifiers for malware instances, nor subsume the work done by the AV industry in creating malware detection mechanisms and anti-malware countermeasures.

5. Proposed Framework

As a language and format for attribute-based malware characterization, MAEC's core components include a vocabulary, grammar, and form of standardized output (Figure 1, below).

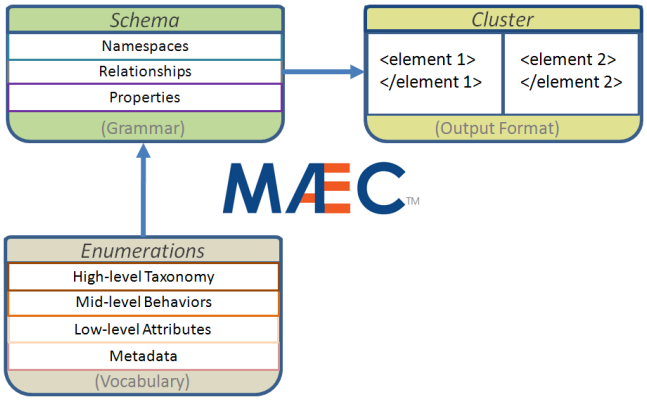


Figure 1: MAEC Overview

The enumerated vocabulary is composed of three distinct levels of malware attributes, as well as any metadata. MAEC's schema is effectively a grammar and defines the structure of the enumerated elements and the relationships between them. Finally, the MAEC cluster is a standardized format for the output of any MAEC characterized data.

5.1 MAEC's Enumerated Attributes

At its heart, MAEC will be composed of three tiers of enumerations of malware attributes (Figure 2, below). Each tier will consist of a finite number of attributes and will likely decompose into multiple levels of abstraction for more accurate malware characterization. Since software can only perform the actions that can be achieved through execution of the finite instructions provided by the underlying system's Instruction Set Architecture (ISA), and because these instructions consequently have a fixed number of arguments, it follows that these attributes are finite and enumerable. Therefore, MAEC's enumerated attributes will include the detailed actions and behaviors performed by the software, as well as the mechanisms that these actions and behaviors serve to implement.

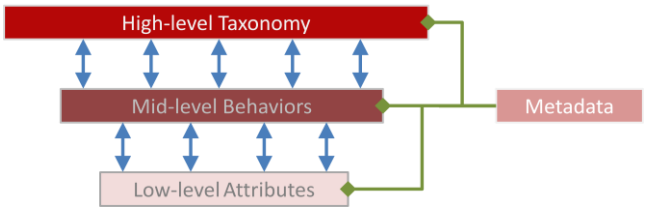


Figure 2: MAEC's Tiers of Enumerated Elements

MAEC's ability to communicate high-fidelity information about malware will be directly tied into its ability to accurately characterize the numerous types of malware in existence, as well as those created in the future. Therefore, at a minimum, MAEC must be able to describe any low-level actions performed by malware. However, its utility would be significantly degraded if it did not also have the ability to group such information into higher-level representations of malware behavior.

5.1.1 Low-level Attributes

At its lowest level, MAEC will describe attributes tied to the basic functionality and low-level operation of malware. This can include *observable* entities such as system state changes (e.g. the insertion of a registry key), as well as any features extracted through the disassembly of malicious binaries (e.g. specific assembly instructions). Therefore, likely sources of such data include static analysis, dynamic analysis of malware binaries through *sandboxes*, network and host-based IDS, and IPS.

The specific level of coverage and its composition at this level is still an open issue. There are certainly MAEC use-cases relating to the characterization of assembly code and other features specific to static malware analysis. It would likewise be useful to have the ability to describe the algorithms employed by malware and the nature of their operation. However, it may make the most sense to enumerate and support such attributes at the mid, behavioral level, defined in the next section.

As the community defines MAEC, the enumeration and definition of these low-level attributes will need to be done first, as this will likely make it easier to gain consensus on the mid-level behaviors. It would also be useful to leverage the work performed by the IEEE's ICSG [22] in creating a format for exchanging certain types of low-level malware attributes and metadata.

5.1.2 Mid-level Behaviors

At the middle tier, MAEC's language will abstract the aforementioned low-level attributes for the purpose of defining mid-level behaviors. The definition of such behaviors gives insight into the consequences of the actions performed by the low-level attributes, and allows a higher-level representation of these actions to be constructed.

For instance, the description of a registry key created or modified by malware is a fairly abstract bit of data that can be useful for detection purposes. However, it does not provide any insight into *why* the malware created or manipulated the registry entry. Such a registry entry could have many possible uses, including being used to ensure that the malware gets executed at system start-up, or as a simple flag to indicate that the system has been infected. Including the necessary components for characterizing such mid-level behaviors in the MAEC language will allow for the accurate description of the intent or goal behind system state changes and other low-level malware attributes.

Since these mid-level behaviors serve to connect the tiers of the low-level attributes and high-level taxonomy, and are the least well-defined of the three (representing consequences rather than distinct attributes or mechanisms), this enumeration will likely be the most difficult to construct and gain consensus on. This development process will likely be iterative and require many passes in order to accurately cement the relationship between these behaviors, the low-level attributes that they abstract, and the high-level taxonomy that directly references them from above.

5.1.3 High-level Taxonomy

At the more conceptual and high level, MAEC's vocabulary will allow for the construction of a taxonomy that abstracts clusters of mid-level malware behaviors based upon the achievement of a higher order classification or grouping. We envision that such a taxonomy will have views (i.e. unique layouts) intended for different target audiences – for instance, forensic analysts may

only be interested in looking at observable malware *payload* behaviors, etc.

To expound upon the example given for the mid-level behaviors (Section 5.1.2), ensuring that malware is executed at start-up is a behavior that is typically part of a *persistence* mechanism. This behavior is often accompanied by the instantiation of a binary copy of the malware on the local machine. Therefore, in MAEC's top-level taxonomy, these two mid-level behaviors would be defined as belonging to the class of persistence mechanism.

Once MAEC's mid-level behaviors have been defined, the MAEC community will be in a position to begin the process of creating the high-level taxonomy and constructing the appropriate and necessary behavioral linkages. Accordingly, it would be useful to leverage the work done on defining categories of high-level malware mechanisms, such as that performed by SANS' Internet Storm Center [19]. Some of these mechanisms we have described using our own definitions (Appendix A.II).

5.1.4 Metadata

In order to include all pertinent information regarding malware and to fully describe the common actions of malware and the rationale behind them, MAEC will characterize malware-appropriate metadata. This can range from metadata associated with malware behaviors, like the transparency of the insertion mechanism used, to the more common types of metadata that are associated with malware artifacts such as file hashes. However, defining the exact nature of malware-related metadata is an open question, and one that is best answered with the involvement of the larger security community.

5.2 The MAEC Schema

MAEC's enumerations of behaviors and other attributes are necessary for the establishment of a vocabulary for characterizing malware. Therefore, the primary intent of MAEC's schema is to define a syntax for this vocabulary. Likewise, the schema serves to create an interchange format for the MAEC language, and can also be utilized as a baseline for the creation of malware repositories or intermediate format for the sharing of information between repositories.

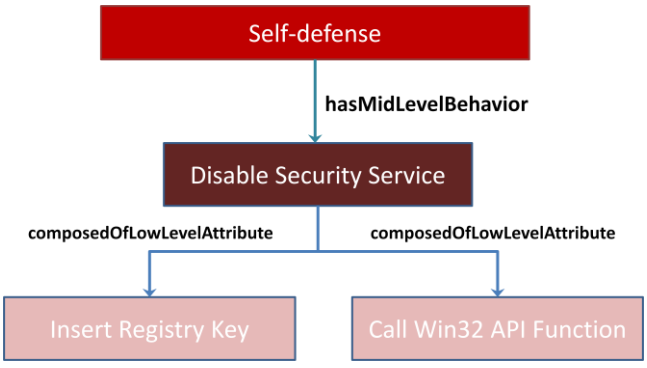


Figure 3: Example of Structure Imparted Through MAEC's Schema

An example of several very simplistic relationships that MAEC's schema would impart is shown in Figure 3, above. Here, the schema defines (through the 'hasMidLevelBehavior' relationship) that the high-level self-defense mechanism has an associated mid-level behavior of disabling a security service. This behavior, in turn, is defined as being composed of two low-level attributes, namely the insertion of a registry key, and a specific Win32 API function call, through the 'composedOfLowLevelAttribute' relationship.

The schema will be used to define relationships beyond those dictated in MAEC's three-layer hierarchy. For instance, a high-level mechanism could make use of multiple laterally associated mid-level behaviors. In this case, the schema will provide the means for correctly defining this lateral relationship. There may be multiple ways of defining such relationships, but only one should be specified in the schema, in order to ensure coherence between MAEC characterizations.

Likewise, the size and partitioning of MAEC's schema is another open issue. It is likely that a single, complete MAEC schema will be of substantial size and complexity, thus making it difficult to learn and effectively utilize. As such, this would compromise the adoption of MAEC by analysts and researchers due to its substantial learning curve and unwieldiness. Therefore, partitioning the MAEC schema into multiple sub-schemas, based on use case, is a possible solution; however, this is another decision that is best made using input from the greater community.

The mechanisms for defining temporal relationships in malware would also be established as part of MAEC's schema. This would be particularly useful for describing malware that lies dormant before executing its payload, or performs some actions in a repeating sequence, and would permit the accurate identification of where an active malware instance is in terms of its execution flow.

Consequently, at a minimum, the schema will define the following elements.

5.2.1 Namespaces

Namespaces in MAEC's schema will represent the grouping of the behaviors, mechanisms, and other malware-related enumerated attributes of MAEC's language into well-defined classes. Where applicable, namespaces will follow those utilized by relevant Making Security Measurable (MSM) standards (see Section 7 below).

5.2.2 Properties

MAEC's schema will contain a general set of properties applicable to malware attributes and namespaces, with specific properties for behaviors. This can encompass things like the number of times a behavior occurs, whether it is the child or parent of another behavior, etc.

5.2.3 Relationships

MAEC's schema will include an established set of rules for defining the potentially multi-directional relationships among namespaces. For example, members of the namespace of mid-level behaviors can be composed of multiple members of the low-level attributes, while members of the low-level namespace can be associated with (but not composed of) only a single member of the mid-level behavior namespace.

5.3 The MAEC Cluster

The MAEC cluster (Figure 4, below) represents a standard output format of MAEC, with the purpose of encompassing any set of attributes obtained from the characterization of a malware instance. Therefore, it will serve as a container and transport mechanism for use in storing and subsequently sharing any MAEC-characterized information. A MAEC cluster could be used to describe anything from a particular insertion method

(composed of several low-level attributes and mid-level behaviors), to all of the attributes of a malware instance, to the key behaviors common to an entire malware family.

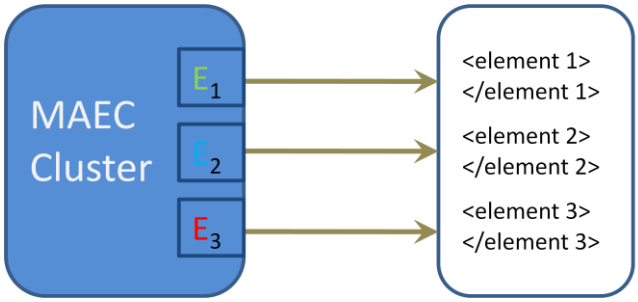


Figure 4: MAEC Cluster Overview

Although a MAEC cluster will be most useful when encompassing a set of malware attributes with a particular significance (like the insertion method or family behaviors mentioned above), it is intended to serve as a generic container for MAEC-characterized malware data. Therefore, it can be used with as little or as much information as desired; any further meaning beyond the explicit data stored in the cluster is defined by its producer.

An open issue related to the MAEC cluster is how to usefully incorporate large datasets and enumerations in MAEC. In many cases with malware analysis, there will be large volumes of strings and other static data extracted from malware. Such data is useful from both an analysis and detection standpoint, and would therefore be important for inclusion in a MAEC cluster. However, the incorporation of such data along with the MAEC characterization in a single cluster could greatly impair its readability. Therefore, it may be prudent to store such data in another, external MAEC cluster and then reference this secondary cluster inside the cluster containing the main analysis. An auxiliary cluster that uses some other storage method and is referenced by the main MAEC cluster is another possibility.

5.3.1 Output Format

```
<behavior id=1>
    <level>mid</level>
    <type>reconnaissance</type>
    <subtype>keyboardCheck</subtype>
    <attributes>
        <languageChecked>language:ukrainian</languageChecked>
            <successCondition>execution:stop</successCondition>
            <failureCondition>execution:continue</failureCondition>
    </attributes>
</behavior>
```

Figure 5: Notional MAEC Cluster – XML Representation

As shown in Figure 5 above (intended only to illustrate a possible form of MAEC XML output), MAEC will initially support XML as a data interchange format due to its ubiquity and propensity for being human-readable as well as machine-consumable. In the future, it is likely that MAEC will support Resource Description Framework (RDF) as well as JavaScript Object Notation (JSON) and other such lightweight formats.

6. Characterization Test Case

The computer worm best known as "Conficker" first appeared in the wild in November 2008, and has since been responsible for the creation of a sizable *botnet* [23]. Although this malware family does not implement any particularly novel mechanisms, it represents a highly complex, blended threat that makes use of a variety of attack *vectors*. There are also multiple variants of Conficker in existence, and its multiple layers of obfuscation and self-defense have made it one of the more difficult malware instances to analyze.

In terms of malware instances, Conficker represents a bit of an anomaly in that it was heavily analyzed and studied by the security and anti-malware communities. As such, there is a large amount of detailed information available about its structure and internal mechanisms. This is largely the reason that we chose Conficker as a test case for MAEC and a sanity check for how its three-tiered framework deals with real malware, as having these rich sources of information regarding the malware attributes and behaviors of a particular malware family would permit us to make a more accurate assessment in this regard.

Unlike Conficker, the vast majority of malware instances are not analyzed to any large degree. Therefore, we recognize that having this amount and type of information about a malware instance or a family is a highly uncommon occurrence. However, our hope was that the characterization of this unique malware instance would be useful for identifying any questions and/or problems related to our current concept of MAEC, while also helping to determine any critical missing pieces..

6.1 Initial Characterization

For our first iteration, we used SRI's detailed Conficker analysis [24], and attempted to piece together a full MAEC characterization of the Conficker.A variant. Unfortunately, we quickly discovered that extracting such information from prose plaintext is a tedious and time-consuming task, and that we would require multiple sources in order to construct a more complete characterization. Therefore, as a starting point, we decided to categorize SRI's top-level control flow for Conficker.A.

Although this process was fairly straightforward, due to the relative simplicity of the control flow, it brought up an interesting issue. The control flow's mixing of low-level attributes and mid-level behaviors made it difficult in some cases to accurately identify the boundaries between the high-level mechanisms being employed. This is because it is hard to judge the purpose of a low-level attribute (for instance "create random name in system32 directory") without any accompanying information; consequently, it is not possible to link a low-level action with the high-level mechanism it belongs to without knowing its function.

Of course, it is unlikely that every analysis will have a clear link between low-level attributes and mid-level behaviors. Likewise, not all users of MAEC will require such information. Therefore, this reiterated the notion that MAEC must be flexible enough to characterize malware using any information that happens to be available.

6.2 Second Characterization

For our second characterization, we stayed closer to our original intent to gain a broader sense of how MAEC might be applied. Accordingly, we utilized multiple data sources [25-30] in order to have a diverse set of information to work with that would permit the construction of a more complete characterization. This time around, we decided to focus on the second variant of Conficker seen in the wild, commonly referred to as "Conficker.B"; this variant includes a number of interesting mechanisms and behaviors not present in Conficker.A, such as self-defense.

Our process for this Conficker.B characterization consisted of two steps; first, we binned any attributes into the bottom two MAEC tiers (for the sake of simplicity we focused only on observables at the low level – see Section 6.2.1.1 for further discussion) Next, we attempted to link the mid-level behaviors as closely as possible with a high-level mechanism from our pre-defined set (Appendix A.II).

Overall, this bottom-up approach was fairly intuitive. By explicitly specifying the action performed by each low-level attribute (e.g., create, execute, modify, etc.), along with the object that it was performed on, we were able to better define and group the mid-level behaviors. Of course, this is based on the assumption that such behavioral information was present in the analysis.

Figure 5, below, shows the result of Conficker.B's Windows File Sharing *propagation* mechanism being characterized in this manner. Here, downward arrows symbolize the logical progression of the mechanism's execution, and dotted lines symbolize conditional elements that must be met for the execution to continue. Our complete characterization of Conficker.B done in this fashion can be found in Appendix B at the end of this document.

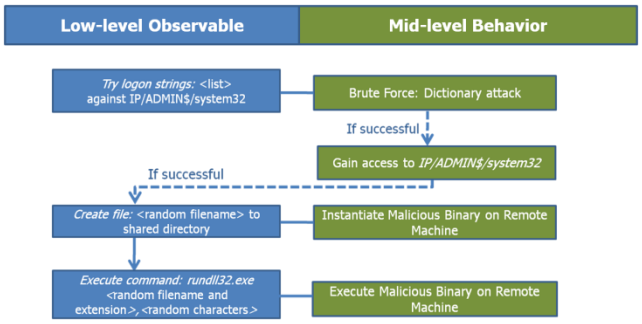


Figure 5: Conficker.B File Sharing Propagation Mechanism

Consequently, this allowed us to create a very basic grammar for the low-level observable attributes, which we tried to utilize consistently throughout our characterization. This grammar consists of an action, the type of object that the action is performed on, and the specific object which the malware instance performs the action on. For example, Conficker.B patches Windows' *netapi32.dll* to disable the vulnerability that it used as a vector for successfully inserting itself onto the compromised machine. Using our simple grammar, this action would be written as:

```
Modify File: \%system32%\netapi32.dll
```

Although we were largely successful in binning the analysis information we found into the three defined MAEC tiers, we had several observations and associated open questions throughout this process, as detailed in the following subsections. A more thorough discussion on a number of these topics stemming from our Conficker characterization, as well as other MAEC issues and challenges, can be found in Section 9.

6.2.1 General Observations

The following are general observations on MAEC and its conceptual features that were drawn from our characterization of Conficker.B. Most of these observations relate to the fact that MAEC and its elements must ultimately be defined in much greater detail for the construction of accurate malware characterizations to be fully operative.

6.2.1.1 Attributes

We first noticed that the level of detail of each of MAEC's tiers must be explicitly defined in order for attributes to be binned uniquely. For instance, in Figure 5 above, "try logon strings" is binned as a low-level attribute; however, this attribute could also have a number of commands/API calls associated with it, thereby actually making it a mid-level behavior (i.e. the purpose behind the commands/API calls). In this case, such information was not available in the analyses that we utilized, so we decided to make this a low-level attribute associated with a brute-force attack behavior; such cases of ambiguity must be properly addressed and resolved during the continued development of MAEC.

We also discovered that low-level attributes can encompass more than pure observables, as they can refer to changes of state, transient behaviors, and other features discovered through analysis. While we had originally taken this into account, our initial concept of MAEC made mention only of observables, for the sake of simplicity. Now that we had real-world data to analyze, it was clear that we needed to explicitly state which entities other than observables that low-level attributes can refer to.

Finally, it was readily apparent that we needed to overtly define the hierarchy within each of MAEC's three tiers, or else risk ambiguity in such characterizations. This is particularly true for the high-level tier, where we had questions regarding the accurate positioning of the mechanisms that we were binning the mid-level behaviors into. For instance, when referring to a specific propagation mechanism, such as the one that Conficker.B uses in conjunction with Windows File Sharing, it is clear that this mechanism belongs hierarchically as a branch off the general propagation mechanism. Similarly, armoring and obfuscation may lie under the umbrella of the *self-defense* mechanism.

6.2.1.2 Schema

With regards to the schema, we discovered that the analyses we used for our sources could effectively be partitioned into two categories; those that center on thorough analysis through reverse engineering, and those that have less detailed analysis and are more focused on detection. By having the multiple MAEC tiers (even if they're hard to populate), we are providing a framework for connecting these points of view. Thus, MAEC has the potential to ensure information flow between these two processes, which while being inter-connected, are seldom integrated.

6.2.2 Open Questions

The next few subsections describe several open questions that were the result of our characterization of Conficker.B. Many of these questions are associated with the specific implementation of MAEC and are discussed further in Section 9.1.

6.2.2.1 Attributes

After we noticed that low-level malware attributes can consist of entities other than observables, we also discovered that many analyses, especially those that go into considerable depth, characterize some form of machine code. Such code can provide valuable insight on the internal operation of malware, particularly with regards to obfuscation/encryption mechanisms and the like; it is therefore a useful attribute that has as much utility as other MAEC attributes. However, how to characterize such code is an

open question, especially considering that it can have multiple forms, be of substantial length, and is not enumerable.

Likewise, we found that several Conficker analyses made note of "*side-effects*," or the consequences of unintentional interactions between the malware and its execution environment. Such information can be considered a separate observable attribute, and would clearly be useful for detection, but the question of how to include it has yet to be answered, due to its environmental (i.e. platform) dependencies and the general difficulty in accurately correlating such information with malware behavior. One possible method would be to separate observables into categories of "first-order" and "second-order," with the first-order stemming directly from the execution of the malware, and the second-order being side-effects of execution in a specific environment.

6.2.2.2 Schema

After our characterization of Conficker.B, it was clear that MAEC's schema would need some way of defining logical and other types of relationships that can exist on their own or between multiple malware attributes. A simple type that we included in our characterization is the conditional relationship (see Figure 5 in Section 6.2 for an example). However, how to include these relationships without adding a significant layer of complexity to the schema has yet to be determined and is an open question.

7. Ties to MSM Standards

As part of MITRE's Making Security Measurable (MSM) effort[2], MAEC will make use of other relevant MSM standards, where appropriate (Figure 6, below).

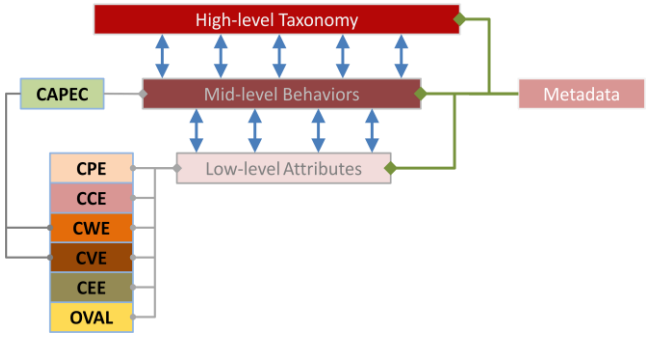


Figure 6: MAEC's Relationship with MSM Standards

A number of these standards can be utilized for more accurately characterizing malware, especially with regards to standardized reporting and assessment of threats and vulnerabilities in the operational environment. However, due to potential implementation complexities and the varying levels of maturity of these standards, it is likely that only a subset will initially be referenced by MAEC. Likewise, there exists an overlap in the namespace of observable data relating to malware, attack patterns (CAPEC), and logged events (CEE), that must addressed for these three standards be inter-operable.

7.1 CAPEC

MAEC will make use of the Common Attack Pattern Enumeration and Classification (CAPEC)[3] for describing the relevant attack patterns associated with the high-level malware taxonomy, such as those dealing with network reconnaissance, propagation, insertion, and *command & control*. MAEC's usage of CAPEC will allow for such behaviors to be defined through an industry standard attack pattern enumeration, thus ensuring that the attacker's perspective in implementing these behaviors is properly represented.

This association will also provide researchers with detailed information regarding the behavior's motivation (if included in the CAPEC entry). It is conceivable that such information could be utilized by researchers for determining the over-arching intentions of the malware author (by abstracting multiple CAPECs and other malware behaviors), as well as by developers for creating software with improved security against malware.

7.2 CEE

MAEC will use the event description language provided by Common Event Expression (CEE)[4] to describe logged events associated with malware activity. Such entries can be linked to specific malware behaviors and used to determine the presence of malware.

7.3 CWE

If it is determined that a malware instance exploits a particular software weakness, MAEC will link to its corresponding Common Weakness Enumeration (CWE)[5] entry. This linkage will allow for the generation of statistics with regard to the most common types of weaknesses being exploited by malware, thereby highlighting the areas where better security-oriented coding practices need to be implemented. It will also provide an attribute for use in characterization and correlation when a specific CVE or CCE isn't being targeted by malware.

7.4 CVE

MAEC will link to the Common Vulnerabilities and Exposures (CVE)[6] entry associated with a particular vulnerability exploited by malware. This will allow users to determine the nature of the vulnerability being exploited by the malware, as well as for automated fix and patch assessment through CVE-compatible tools. Likewise, MAEC's link to CVE will substantiate vulnerability-based threats by providing a concrete example of their exploitation, which will permit the prioritization of software vulnerability patching and associated threat assessment efforts.

7.5 CPE

For a standardized description of the software and hardware platforms targeted by malware, MAEC will make use of the respective Common Platform Enumeration (CPE)[7] entry associated with the platforms, permitting the tool-based identification of potential victim machines by IT administrators. Linking to CPE will also allow for assessment of the threat that a malware instance poses to organizational computing resources based on the platforms that it targets.

[2] http://makingsecuritymeasurable.mitre.org

[3] http://capec.mitre.org

[4] http://cee.mitre.org

[5] http://cwe.mitre.org

[6] http://cve.mitre.org

[7] http://cpe.mitre.org

7.6 CCE

The linkage of MAEC to the Common Configuration Enumeration (CCE)[8] will allow for the description of any of the vulnerabilities associated with malware that are not related to software flaws. This will allow for the host-based detection of specific configuration-related vulnerabilities exploited by malware, as well as the detection of general configuration issues that malware could potentially exploit. MAEC's link to CCE can also substantiate non-flaw based vulnerability threats by providing a concrete example of their exploitation, which will permit the prioritization of configuration vulnerability mitigation strategies and associated threat assessment efforts.

7.7 OVAL

Certain low-level malware attributes may represent attempts at software vulnerability exploitation, meaning that such entries can be linked to corresponding Open Vulnerability Assessment Language (OVAL)[9] definitions (if in existence). Such a connection would allow for improved malware threat mitigation, by tying in the ability to easily check for the host-based existence of a vulnerability that is directly associated with a particular malware instance.

Likewise, it can narrow down the potential malware variants capable of infecting a system by correlating the un-patched vulnerabilities present on a system with those linked to by MAEC characterizations. Similarly, it could enhance *remediation* by providing an automated means of checking for any remaining malware artifacts.

OVAL can also be used to determine malware presence based on comparison of multiple scans. A common malware behavior is to patch the particular vulnerability used to exploit a system after successful *infection*, so that detection of such a "silently" patched vulnerability can be used to establish the presence of malware.

8. Use Cases

As a domain-specific language for the characterization of malware, MAEC serves to provide a vocabulary and grammar for the encoding and decoding of information. It follows that the majority of the use cases for MAEC are motivated by the unambiguous and accurate communication of malware attributes that it enables.

While there are a number of ways that MAEC-encoded information can be utilized in some automated form, the majority of MAEC's use cases are human-oriented. That is, while MAEC will provide a foundational basis and structure for use in characterizing malware, we believe that such characterizations will be interpreted and utilized by humans more often than by machines.

8.1 Analysis Use Cases

As shown below in Figure 7, MAEC will typically be utilized to encode the data garnered from malware analysis. In such a scenario, malware would serve as an input to whatever method of analysis is being utilized. The results of the analysis would then be classified using MAEC's enumerations and schema, and a MAEC cluster would be generated for use in transportation and communication of the new data.

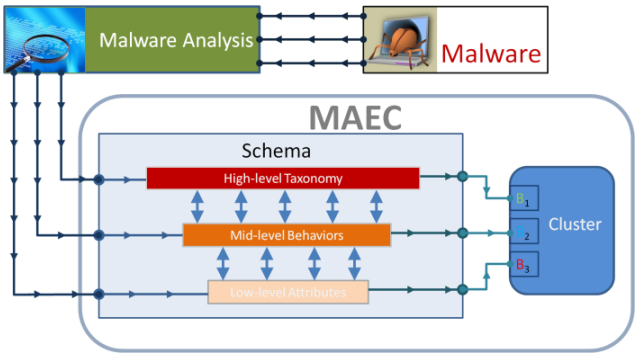


Figure 7: Typical Analysis MAEC Usage Scenario

The analysis of malware using static and dynamic/behavioral methods is becoming increasingly important for the purpose of understanding the inner workings of malware. Such information can be utilized for malware detection, mitigation, the development of countermeasures, and further analysis. However, the lack of a common vocabulary for analysis makes it difficult to compare and utilize the results of analyses performed by different analysts and tools.

The encompassing attribute enumerations provided by MAEC, containing all possible malware attributes capable of being characterized through malware analysis, will enable the convergence of malware analysis results upon a common vocabulary. Utilization of such a vocabulary for malware analysis should eliminate the confusion and ambiguity resulting from the use of multiple disparate vocabularies for analysis results.

Likewise, through its high-level taxonomy, MAEC will provide a way of guiding and helping the analysis process. By including an enumeration of the mechanisms of malware behavior, MAEC will give analysts the ability to search for any behaviors and low-level attributes that correspond to these mechanisms, as well as to easily abstract previously discovered attributes and behaviors in multiple dimensions.

MAEC clusters could also be utilized as a standard format for use in the creation of visualizations of malware behavior [32]. Such visualizations would permit clear assessment of the low-level actions and mid-level behaviors performed by malware and facilitate natural comparison between two or more malware instances.

8.1.1 Behavioral/Dynamic Analysis Tool Wrapper

Current behavioral analysis tools provide a powerful way of characterizing malware activity through its observed behavior. However, there is often no commonality between the output of such tools, thus making comparison and analysis of data between divergent sandboxes difficult. Likewise, re-writing such tools so that their output coincides with a standard like MAEC would be unfeasible.

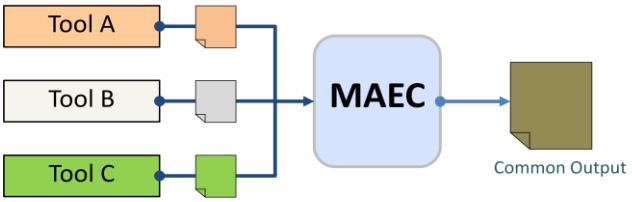


Figure 8: Multiple Tool Output to Common Format

[8] http://cce.mitre.org

[9] http://oval.mitre.org

Therefore, for such tools MAEC could be utilized to develop a wrapper for standardizing the output of any tool in existence (Figure 8, above). This would ensure data compatibility between divergent tools and facilitate the sharing of behavioral malware data.

Example Scenario

Say that a new sample, representing an unknown instance of malware, is submitted to a sandbox for behavioral analysis. The sandbox reports the analysis results in its own format, but also supports the use of MAEC for standardized output. Using the output of this MAEC-based wrapper, an analyst is able to compare the data generated with characterizations of previous malware instances performed on different sandboxes. This permits the analyst to establish that this is actually a slightly modified variant of a previously observed malware family.

8.1.2 Malware Repositories

Malware repositories oriented towards analysis often have very specific needs that require the use of a highly customized schema. This entails that sharing or exporting data from a repository defined by such a schema would be very difficult without the existence of a standardized system for the conversion of this data into a common format.

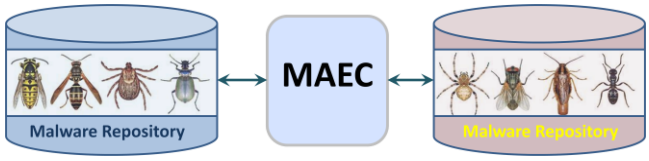


Figure 9: Repository Data Sharing Through MAEC

MAEC's schema could be used for mapping between the dissimilar schemas utilized in malware repositories. This would facilitate the sharing of analysis information stored in disparate repositories (Figure 10, above). Likewise, the usage of MAEC in malware repositories would permit improved data-mining due its structuring and labeling of malware attributes.

8.1.2.1 Improved Data-mining

Most current malware repository schemas are typically structured with little regard to the information contained inside malware analysis reports, with the entire report often being contained in a single text field inside the database. As such, it is usually possible only to do a string or regular expression based search for information that is highly specific to a single malware instance, such as the name of a file that it drops upon insertion. However, even such relatively simplistic queries are not possible across multiple repositories due to their divergent schemas.

This makes it very difficult to make any meaningful comparison between two malware instances, as there is only one field with a large number of varied and non-structured information to use in the comparison. As a repository grows and expands, this problem becomes even more profound, since the signal-to-noise ratio of simple text-based queries inevitably drops with a larger number of malware samples and their corresponding analysis information in the database.

By mapping a malware repository schema directly to MAEC's schema, any ambiguity between malware features and corresponding search queries could be eliminated. Since MAEC incorporates low-level attributes, as well as mid and high-level behaviors, the integration of a malware repository schema with MAEC would allow for querying the repository based on multiple tiers of discrete malware attributes, and would provide for individual attribute-level comparisons between multiple malware instances.

The aforementioned comparison of discrete malware attributes through MAEC could also be used as the basis for a malware similarity metric. In this manner, MAEC clusters specified for multiple malware instances could be compared, and a similarity score calculated based on the number of shared attributes at each of the three levels of abstraction.

Finally, organizations may not wish to directly export or share their malware repositories, but could instead allow the execution of search queries against them. MAEC could be used in this case to map between repository-independent queries and specific repositories. This would permit the execution of a single query against multiple repositories whose schemas may have little in common. The support of such federated queries is another important aspect of malware data-mining that MAEC would enable.

Example Scenario

For example, imagine searching a large unstructured repository for all malware that propagates via spammed infected email messages by using the keyword text of "SMTP." The results from such a query would include some relevant data relating to malware that does in fact use such a propagation mechanism, but could also have information about malware that connects to SMTP servers, attacks SMTP servers, or was originally received in an email message, just to name a few.

If the schema of the aforementioned repository were to be mapped to MAEC, querying the repository using the MAEC-defined category of (for example) "Propagation Vector: Email/SMTP" would retrieve the desired information, with completely accurate results. While this is a very simple query, it demonstrates the significant data mining capabilities provided by the usage of a domain-specific formal language.

8.1.3 Objective Criteria for Tool Assessments

MAEC's typing of malware based on discernable attributes can be utilized as objective criteria for use in the assessment of anti-malware tools. In this sense, a tool would be assessed on the basis of its support in detecting all of the attributes associated with a particular malware type. A tool that cannot detect certain MAEC-defined attributes associated with a particular malware type can miss any malware that contain such attributes, and therefore cannot objectively be defined as capable of detecting that type of malware.

8.2 Operational Use Cases

In the operational domain, threat analysis, intrusion detection, and incident management are processes that deal with all manners of cyber threats. MAEC, through its uniform encoding of malware attributes, will provide a standardized format for the incorporation of actionable information regarding malware in these processes. Accordingly, the successful integration of such a standard in the operational environment (Figure 10, below) will facilitate more accurate and enhanced malware triage, detection, and response.

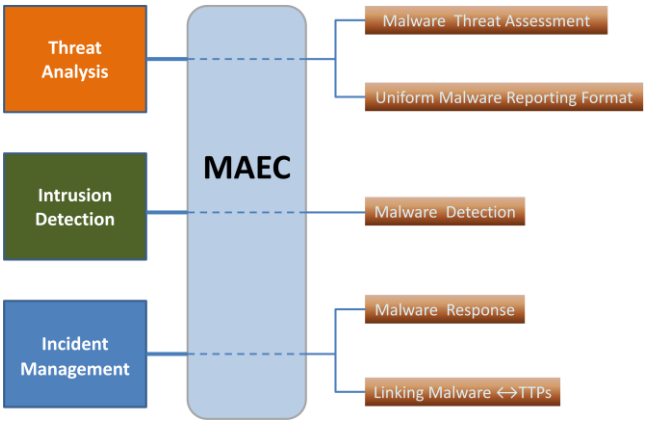


Figure 10: MAEC for use in Operations Security Management

8.2.1 *Uniform Malware Reporting Format*

Current malware reporting, while useful for determining the general type and nature of a malware instance, is inherently ambiguous due to the lack of a common structure and vocabulary. Likewise, it often excludes key malware attributes that may be useful for mitigation and detection purposes, such as the specific vulnerability being exploited. Clearly, the current value of malware reporting to end-users is significantly degraded without an encompassing, common format.

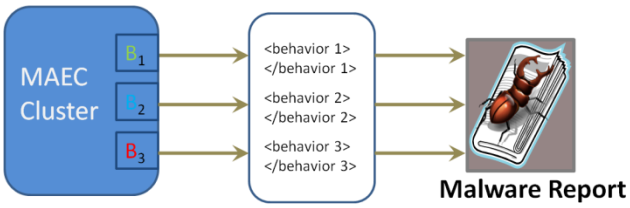


Figure 11: MAEC as a Uniform Malware Reporting Format

The use of MAEC's standardized vocabulary and grammar in malware reporting will facilitate the creation of a separate, uniform reporting format (Figure 11, above). Such a format will reduce confusion as to the nature of malware threats through the accurate and unambiguous communication of malware attributes, while also ensuring uniformity between reports composed by disparate authors and organizations.

Example Scenario

For example, say that a consumer of an AV product scans one of their machines and finds that a new piece of malware has been detected. Wishing to know more about the malware, they navigate to the site of their AV vendor and lookup the information regarding the instance that was found. Unfortunately, the information there is formatted as unstructured prose text and is missing key attributes. Likewise, it does not have any information about the vulnerability exploited by the malware which allowed it to get on the system in the first place.

With MAEC, such reporting would be structured in a meaningful way. This would permit the consumer in the above example to easily compare the behaviors of this detected instance with those of another. More importantly, it would allow them to quickly identify the vulnerability exploited, by linking to CVE, as well as the specifics of how the malware got on their system in the first place, so they can prevent this infection from re-occurring.

8.2.1.1 *Multiple Source Integration*

The majority of useful data in malware reports is currently made up of unstructured narrative text, making it very difficult to accurately compare or combine multiple reports. Therefore, the standard vocabulary and structure of the uniform malware reporting format enabled by MAEC will allow for accurate comparisons between multiple malware reports, without the need for converting each report to some intermediate representation.

As such, the use of MAEC in malware reporting will permit category-by-category comparison. This will not only allow for identification of differences and similarities among reports, but will also provide for the discernment of critical malware-related areas that need to be focused on or researched further.

8.2.2 *Malware Detection*

Characterizing malware based on its attributes with MAEC will permit the use of actionable information for malware detection. Specifically, MAEC's accurate identification of the observable low-level attributes associated with a malware instance will allow for the detection of malware at both host and infrastructure levels.

Figure 12: Malware Detection via Shared Attributes

As such, a single MAEC characterization of a malware instance, represented by a MAEC cluster, can provide data that can be used to detect multiple malware instances. Since there are only a finite number of ways of accomplishing a single software behavior (for instance, malware insertion), particularly at the assembly level, it is statistically likely that there will be an intersection of such attributes between multiple malware instances. Therefore, the MAEC characterization of a single instance can permit the detection of malware families and even otherwise un-related malware that have certain attributes in common with the instance (Figure 12, above).

8.2.2.1 *Host-based Detection*

MAEC's encoding of information regarding low-level system observables will allow for host-based malware detection. This can be accomplished through manual system inspection for MAEC-defined entities such as active processes that signal the presence of specific malware instances.

Likewise, utilization of such information contained in MAEC clusters will permit tool-based detection. For example, a tool could read a MAEC cluster and use the data contained inside to automatically generate OVAL queries for the purpose of detecting the existence of entities such as registry keys and files.

Similarly, using MAEC for characterizing malware will permit the definition of patterns of low-level attributes and mid-level behaviors utilized by malware. Such information could be used to establish new rule sets for heuristic-based detection, based on the observed sequences of behaviors.

Example Scenario

For example, say that a new, particularly nasty, malware instance has been analyzed and had a MAEC cluster created for it. This cluster details the artifacts and low-level observables specific to

the instance, and also gives a behavioral overview of what the malware attempts to do. Accordingly, no signature has yet been created for this malware.

Based on this behavioral overview, the security administrator of an organization determines that this malware has the potential to do serious harm to their computing environment, and that detection of it is a priority. The administrator then takes the MAEC cluster for the malware, and uses his MAEC-compatible tools to generate from it a number of OVAL queries for use in detecting the artifacts associated with the malware. These queries are then executed across the organization's computing infrastructure and the results used to assess whether or not this malware has been detected.

8.2.2.2 Network-based Detection

Although the use of malware-specific signatures in network intrusion detection and prevention systems (IDS/IPS) has become fairly widespread, there is still no general method for characterizing the relationship between distinct malware attributes and network infrastructure. Such a systematic way of linking malware behavior with outgoing network data could be used for infrastructure-based detection, as well as the accurate identification of observed malware behaviors. Likewise, the accurate characterization of incoming traffic as being associated with a malicious behavior can be used to detect and drop packets associated with behaviors like those employed by bots for command and control.

Therefore, MAEC's role as a standard language and format requires that it will be able to characterize any such applicable information regarding malware. In particular, MAEC will provide the capability for integrating and characterizing any attributes and patterns related to malicious network activity, thereby permitting network-based malware detection in combination with this data and IDS/IPS. In this regard, we envision the creation of MAEC enabled tools that could automatically generate signatures for specific malware behaviors.

Example Scenario

For example, say that a new malware instance has recently been introduced into the wild, and that it exploits an unknown network vulnerability as a means of insertion. However, several MAEC clusters of this malware have been prepared and integrated in a report describing the new instance. This report characterizes the specific types of network traffic generated by this malware in order to infect systems, as well as its other behavioral and low-level observable attributes.

Using the MAEC-encoded information contained in the report, an organization updates their IDS rule sets to block any traffic that looks similar to that of the malware insertion process. Likewise, this information is utilized to detect any machines that have already been infected by the instance. When the exploit is identified and the MAEC clusters for the malware instance updated to reflect this, the organization utilizes this information to patch all of their systems, thus making them immune to the insertion of the malware identified by the MAEC cluster.

8.2.3 Malware Threat Assessment

For IT administrators and others charged with protecting systems from cyber threats, one of the most useful aspects of malware reporting is data which details the specific threat that the malware represents. In particular, they are interested in details regarding the specific platforms, vulnerabilities, and weaknesses targeted by the malware. Such data, combined with information regarding the actions performed by the malware, would enable prioritization of malware assessment and change management efforts.

Although current malware reporting may include these useful characteristics, such information has no commonality between reports and does not link to other relevant standards. Therefore, this makes it difficult to ascertain the true threat that malware represents.

MAEC's linkage to OVAL, CPE, CVE, and CWE, will provide system administrators with the necessary information for determining the specific vulnerabilities and weaknesses targeted by malware. Accordingly, MAEC's encoding of mid-level behaviors and a high-level taxonomy will allow for the accurate discernment of the threat that it represents to their organization and infrastructure.

This linkage could also allow for the creation of a malware threat scoring system, similar to that of the Common Vulnerability Scoring System (CVSS) for software vulnerabilities[10]. As described above, MAEC's link to the relevant MSM standards as well as its characterization of mid and high-level malware features would provide the necessary data for accurately describing the attack vectors and payload of a malware instance. This data could then be used to score the potential impact of the malware based on pre-defined categories, such as payload type (e.g. data theft, bot-like behavior, etc.), propagation, and degree of *entrenchment*.

Of course, ascertaining such attributes requires a fair amount of analysis, something which is not done for the vast majority of malware samples. Therefore, it is likely that very few instances will link to MSM standards and include the necessary mid-level and high-level attributes. However, there are also patterns relating to malware behaviors and types that can be observed at the lowest level of attributes; defining such patterns through MAEC will allow an automated, cursory judgment of malware threats to be created using dynamic analysis engines and related methods.

Example Scenario

For example, say that a new malware instance has been seen in the wild, analyzed, and has had a report issued for it. The report uses MAEC for encoding the key attributes of interest, and therefore links to the CVE entry detailing the vulnerability exploited by the malware for insertion as well as the CPE entry for the software platform that it targets.

An IT administrator looks at the report, and determines based on its CPE linkage that several of their machines are running the platform targeted by the malware. However, these machines have already been patched against the vulnerability defined in the CVE entry linked in the report. Based on this information, the administrator is able to determine that this malware poses a low threat and detection of it based on the other MAEC-encoded information contained in the report has a low priority.

[10] http://www.first.org/cvss/

8.2.4 Malware Response

Malware detection alone is often not enough to ensure security and operability. Formulating a response to malware, although still a secondary function, has become increasingly important.

8.2.4.1 Remediation

One of the current realities with cyber security is that malware detection and prevention of infection is not always a possibility, especially with new and targeted malware threats. Unfortunately, most conventional AV tools and utilities are not capable of removing every trace of a detected malware instance [31]. Thus, even if the explicitly malicious portions of an infection are cleaned from a system (which is not always the case), the remaining pieces may lead to false positives in future scans, thereby potentially leading to a misallocation of remediation resources. Likewise, an incomplete remediation could render a system unstable and prone to future infection.

By providing the means for communicating the exact artifacts and low-level attributes associated with a malware instance, MAEC will permit greatly improved remediation of malware infections. Administrators could perform manual remediation based on the data contained in a MAEC cluster, or ascertain the remediation performed by another tool by checking for the existence of the aforementioned artifacts.

Accordingly, the incorporation of the temporal element in MAEC could prove to be extremely useful for remediation. In this sense, it would provide administrators with an accurate determination of how far along a malware instance is in its execution, and thus allow them to judge the steps that may or may not be necessary for remediation. This could be particularly useful for the remediation of malware that downloads new components or lies dormant for some period of time before executing its payload.

8.2.4.2 Countermeasures

The development of countermeasures against malware is an ongoing process that must be able to adapt to new techniques employed by malware authors. By correlating specific malware attributes and behaviors with specific counter-measures, MAEC will allow organizations to alter their defensive posture based on this link. Accordingly, new malware threats can be ascertained and counter-measures against them developed based on existing links between attributes and previously-developed countermeasures.

Example Scenario

For instance, say that an organization is concerned mostly with the security of their network; any unwanted traffic should not be able to get in or out. Since their security mechanisms and policies revolve around this consideration, the development of malware countermeasures is a high priority, and MAEC is therefore utilized to assist in this manner. This is accomplished by analyzing the MAEC characterizations of the various types of malware in the wild; this information is then used to block ports and services that are determined to be commonly employed by malware. Accordingly, it permits the development of countermeasures against specific malware mechanisms that could be particularly harmful, such as propagation and command & control.

8.2.3 Linking Malware TTPs

In the analysis of cyber incidents and attacks, it is often meaningful to characterize the tools, techniques, and procedures (TTPs) used in the attack as being part of a set belonging to a particular attacker. When correlated across multiple attacks, such a connection can be useful for the purposes of attribution.

With malware being one of the most prevalent tools employed by attackers, it would be useful to characterize specific malware instances as belonging to a set of tools used by specific attackers. MAEC would provide this function, as its standard vocabulary and grammar will permit the accurate identification of the malware attributes observed in previous attacks, and with the "attacker" being defined as a metadata attribute, this would allow for the construction of an accurate link based on previously observed and characterized malware.

9. Open Issues & Challenges

This section details the issues and challenges relating to MAEC that we have identified during its conceptual stage of development. The majority of these issues are with regards to implementation, and will therefore need to be properly researched and addressed. We welcome the community's input and involvement in resolving these and other issues relevant to the development and usage of MAEC. More information regarding participation, including how to sign up for MAEC's open discussion list, can be found on MAEC's website[11].

9.1 Implementation Issues

Due to MAEC's wide range of use cases, potentially large enumerations, and various attribute properties and relationships, the development process of the language has the potential to be exceedingly complex. Therefore, this process must be carefully planned in order to be successful. It may be necessary to initially focus on a specific subset of use cases, in order to reduce developmental complexity to a manageable level.

9.1.1 MAEC Schema Structuring

As MAEC has a broad set of use cases, it is clear that most parties making use of MAEC will be interested in utilizing only a subset of the language and its features. Therefore, instead of a single monolithic schema, it may make sense to have multiple schemas, based on their applicable use cases. This would potentially make MAEC useful to more parties, while also increasing the difficulty of development and consumption of MAEC-encoded information. As such, there are trade-offs in terms of complexity, usability, and other issues that must be considered before making any final decisions with regards to the structuring of MAEC's schema.

9.1.2 Temporal/Chronological Characterization

Like any piece of software, malware has an execution flow that is a function of its machine code and the environment in which it is executed. Since MAEC defines individual attributes relating to specific portions of execution, it needs to have some way of defining any temporal and order-based relationships between such attributes, as discussed in Section 6.2.2.1. This would permit the characterization of attributes based on absolute time, frequency, and order of execution.

Such characterization would especially be valuable for analysis and remediation (as briefly discussed in Section 8.2.4.1), but could also be used for detection. For example, the frequency of a botnet command and control command, if defined in a MAEC

[11] http://maec.mitre.org

cluster, could be correlated with network traffic and used as a means of malware detection.

9.1.3 Static Dataset Incorporation

As briefly discussed in Section 5.3, the integration of large static datasets in MAEC is an open issue. Although this could be achieved by including this information in 'child' MAEC clusters which would then be referenced by the 'parent' cluster, this introduces an extra layer of complexity that may not be worthwhile considering the relatively small nature of the problem. Another option would be to simply make this information optional for display in MAEC-based reporting, while still including it inside the MAEC cluster used for generating the report.

9.1.4 Assembly Code Characterization

Particularly with regards to static binary analysis/reverse-engineering, it would be useful for MAEC to have the ability to characterize assembly code, as touched upon in Sections 5.1.1. This would permit the identification of important algorithms, functions, and other non-observable entities that have been previously seen in malicious binaries, while also providing analysts with the ability to describe such code using a standard method. While existing at MAEC's lowest level, such code is not enumerable, but should still be able to be included in MAEC clusters.

Therefore, this issue has two aspects; namely, what format(s) can the code be in for inclusion, and the bigger question of how to characterize such code in a standardized fashion. As far as characterization, one option is to simply tie each block of interesting code to an enumerated behavior from MAEC's mid-level. However, there may be cases where there are notable instructions existing inside a code block that are worth characterizing separately. Such instructions must therefore also have some method of characterization; it is unlikely that their functionality could be enumerated, so this may end up taking the form of a custom string created by the analyst.

9.1.5 Malware Metadata

As briefly discussed in Sections 5.1.4, the specific definition of what constitutes malware metadata and how it relates to MAEC is something that must be defined. There are well-established types of metadata relating to malware that will certainly be included in MAEC, such as hashes, file lengths, and dates.

However, attributes not representative of core malware functionality, such as packing and encryption mechanisms, also have the potential to be considered as metadata. Therefore, this boundary must be determined in order to eliminate ambiguity in MAEC and create a well-defined set of enumerations.

9.2 General Challenges

The following issues are not directly related to MAEC, but rather represent some of the high-level challenges related to the development of any complex formal language.

9.2.1 Repeatability

Repeatability is a potential issue for any language or system intended to characterize such a broad, chaotic landscape as that of malware. The sheer number of potential attributes, combined with the convolution of most modern malware instances, and the different analysis techniques employed, entail that correlating characterizations of malware created using MAEC could be problematic. There is also the fact that malware can exhibit "emergent" behavior, unintended by its author(s), through some unforeseen ecological interaction [33].

In theory, a standard such as MAEC, if created with a well-defined and unambiguous vocabulary and grammar, should ensure repeatability of malware characterization when used to characterize the same malware instance with the same techniques. This should be especially true at the lowest level; after all, there is only one way to describe the insertion of a registry key and similar attributes.

However, at the higher levels, analysts may classify things differently, depending on their expertise, the observed interaction of the malware, and other factors. Therefore, MAEC and its core components should be implemented as explicitly as possible, to remove any potential guesswork and subjectivity from malware characterization.

10. Acknowledgements

The work contained in this paper was funded by DHS NCSD.

REFERENCES

[1] Symantec Internet Security Threat Report: 2008 Trends. http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_internet_security_threat_report_xiv_04-2009.en-us.pdf, 2009.

[2] Tracking Ghostnet: Investigating a Cyber-Espionage Network. http://www.scribd.com/doc/13731776/Tracking-GhostNet-Investigating-a-Cyber-Espionage-Network, 2009.

[3] Prince, B. *Symantec Sees rise in USB-Based Malware as Reports of U.S. Army Ban Surface*. http://www.eweek.com/c/a/Security/Symantec-Sees-Rise-in-USBBased-Malware-as-Reports-of-US-Army-Ban-Surface/, 2008.

[4] F-Secure Reports Amount of Malware Grew by 100% During 2007. http://www.f-secure.com/en_EMEA/about-us/pressroom/news/2007/fs_news_20071204_1_eng.html, 2007.

[5] Moscaritolo, A. *Most Malware Dies Within 24 Hours. SC Magazine*. http://www.scmagazineus.com/most-malware-dies-within-24-hours/article/146384/, 2009.

[6] Kruegel, C., Robertson, W., and Vigna, G. *Detecting Kernel-level Rootkits Through Binary Analysis*. Proceedings of the 20th Annual Computer Security Applications Conference, 91-100, 2004. DOI=http://dx.doi.org/10.1109/CSAC.2004.19

[7] Stinson, E., and Mitchell, J. *Characterizing Bots' Remote Control Behavior*. Proceedings of the 4th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment, 89-108, 2007.

[8] Christodorescu, M., Jha., S., and Kruegel, C. *Mining Specifications of Malicious Behavior*. Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2007.

[9] Fukushima, Y., Akihiro, S., Yoshiaki, H., and Kouichi, S. *Malware Detection Focusing on Behaviors of Process and its Implementation*, Joint Workshop on Information Security (JWIS2009), 2009.

[10] Vigna, G. *Static Disassembly and Code Analysis. Advances in Information Security: Malware Detection*, 27, 2007. DOI= http://dx.doi.org/10.1007/978-0-387-44599-1_2

[11] Bayer, U., Moser, A., Kruegel, C., and Kirda, E. *Dynamic Analysis of Malicious Code*. Journal of Computer Virology, 2:67-77, 2006. DOI= http://dx.doi.org/10.1007/s11416-006-0012-2

[12] Moore, A. P.; Ellison, R. J.; & Linger, R. C. *Attack Modeling for Information Security and Survivability* (CMU/SEI-2001-TN-001, ADA388771). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001.

[13] Messmer, E. *Blended Security Threats on the Rise, IBM Says*. http://www.infoworld.com/d/security-central/blended-security-threats-rise-ibm-says-346, 2008.

[14] Bontchev, V. *Current Status of the CARO Naming Scheme*. http://www.people.frisk-software.com/~bontchev/papers/naming.html, 2009.

[15] Gordon, S. *That Which We Call a Rose.A.* Virus Bulletin, 2003.

[16] Common Malware Enumeration (CME). http://cme.mitre.org/index.html, 2009.

[17] Vamosi, R. *Security Watch: Taking the Internet by Storm*. http://reviews.cnet.com/4520-3513_7-6725188-1.html?tag=nl.e404, 2007.

[18] IEEE Malware Working Group Wiki. http://ieee.sanasecurity.com/wiki/index.php/Main_Page, 2009.

[19] Zeitser, L. *Categories of Common Malware Traits*. http://isc.sans.edu/diary.html?storyid=7186, 2009.

[20] Hines, M. *Malware Boom Puts Pressure On Second-Tier AV Labs*. http://www.infoworld.com/d/security-central/malware-boom-puts-pressure-second-tier-av-labs-882, 2007.

[21] Anti-Spyware Coalition. http://www.antispywarecoalition.org/index.htm, 2009.

[22] IEEE-SA Industry Connections Malware Working Group. http://standards.ieee.org/prod-serv/indconn/icsgmal/index.html, 2009.

[23] Kerner, S. *Researchers: Conficker Still an Active Threat*. http://www.internetnews.com/security/article.php/3832846, 2009.

[24] An Analysis of Conficker. http://mtc.sri.com/Conficker/, 2009.

[25] Win32/Conficker.B – CA. http://www.ca.com/securityadvisor/virusinfo/virus.aspx?id=76852, 2009.

[26] Leder, F., and Werner, T. *Know Your Enemy: Containing Conficker*. http://www.honeynet.org/papers/conficker, 2009.

[27] Fitzgibbon, N. and Wood, M. *Conficker.C A Technical Analysis*. http://www.sophos.com/sophos/docs/eng/marketing_material/conficker-analysis.pdf, 2009.

[28] SANS ISC Conficker Diaries. http://isc.sans.edu/tag.html?tag=conficker, 2009.

[29] The Downadup Codex, Edition 2.0. http://www.symantec.com/connect/blogs/downadup-codex-edition-20, 2009.

[30] Conficker – Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Conficker, 2009.

[31] Anti-virus Comparative Malware Removal Test. http://www.av-comparatives.org/images/stories/test/removal/avc_removal_2009.pdf, 2009.

[32] Holz, T. *Honeyblog: Threadgraphs for Visualizing Malware Behavior*. http://honeyblog.org/archives/34-Thread-Graphs-for-Visualizing-Malware-Behavior.html, 2009.

[33] Crandall, J., Ensafi, R., Forrest, S., et. Al. *The Ecology of Malware*. NSPW'08, 2008. http://www.cs.unm.edu/~crandall/nspw08final.pdf

APPENDIX A: GLOSSARY

A.I: General Terminology

1) **Artifact:** an entity remaining on a system after the execution of malware that is the result of state changes brought on through its insertion, infection and payload. In terms of dynamic malware analysis, this can be thought of as the difference in system state before and after malware execution. Examples: file system objects (i.e. files, directories), registry keys, network ports, etc.
2) **Attribute:** a characteristic of malware that can be used as a descriptor. For MAEC, attributes can include low-level observables, mid-level behaviors, the categories of the high-level taxonomy, and metadata.
3) **Attack Pattern:** a description of a common method for exploiting software, which can include the attacker's perspective and guidance on ways to mitigate their effect. Example: Exploiting incorrectly configured SSL security levels.
4) **Behavior:** the end result of the execution of a specific set of instructions by malware. In this manner, a behavior can be thought of as the consequence of an action. Example: the consequence of inserting a registry key (action) is allowing malware to become resident at system start-up (behavior).
5) **Botnet:** the network of infected computers (or 'bots') created by worms and other forms of malware with embedded command & control mechanisms. Commonly used to send email spam, as well as originators of DDOS attacks.
6) **Entrenchment:** the subjective degree of how well-established and deeply-rooted a malware instance is designed to become on a system. This is most relevant to remediation, in the sense that a malware instance with a high degree of entrenchment (e.g. a rootkit) is generally more difficult to completely remove than an instance with a lesser degree of entrenchment.
7) **Metadata:** any malware data that by itself is not a distinct attribute. In this manner, metadata can be thought of information capable of more fully describing malware attributes.
8) **Observable**: any data that can be obtained by observing the host-level execution of malware on a system through some form of instrumentation. It is typically obtained through dynamic analysis methods and is dependent on the host operating system. This is the overarching category which artifacts fall under, but it also includes dynamic entities such as the processes spawned and network connections initiated by malware. Examples (broad): file system changes, GUI events, etc.
9) **Remediation:** the process of cleaning up and reverting back to a non-malicious system state after a malware infection. This can involve the removal of malicious binaries, registry keys, and other artifacts, as well as the reversion of artifacts not created by the malware to their state before the infection.
10) **Sandbox:** a behavioral/dynamic malware analysis system that can replicate most of the functionally of a target host while still maintaining a layer of separation between non-sandboxed machines and networks.
11) **Side-effect:** an observable that refers to the unintentional or unforeseen interaction between a malware instance and a particular software environment. For instance, Conficker.B's Windows network share propagation mechanism utilizes a brute-force password guessing behavior that has the potential to lockout user accounts based on group policy.
12) **Type:** any of the AV/security community's commonly used monikers for groupings of malware that share some common characteristic. Examples: virus, trojan, worm, backdoor, keylogger, rootkit, bot, etc.
13) **Variant:** a malware instance that shares some code, and therefore behaviors, with another instance or family of malware.
14) **Vector:** the specific method that malware uses to propagate itself. Examples: software vulnerability exploitation, social-engineering, etc.

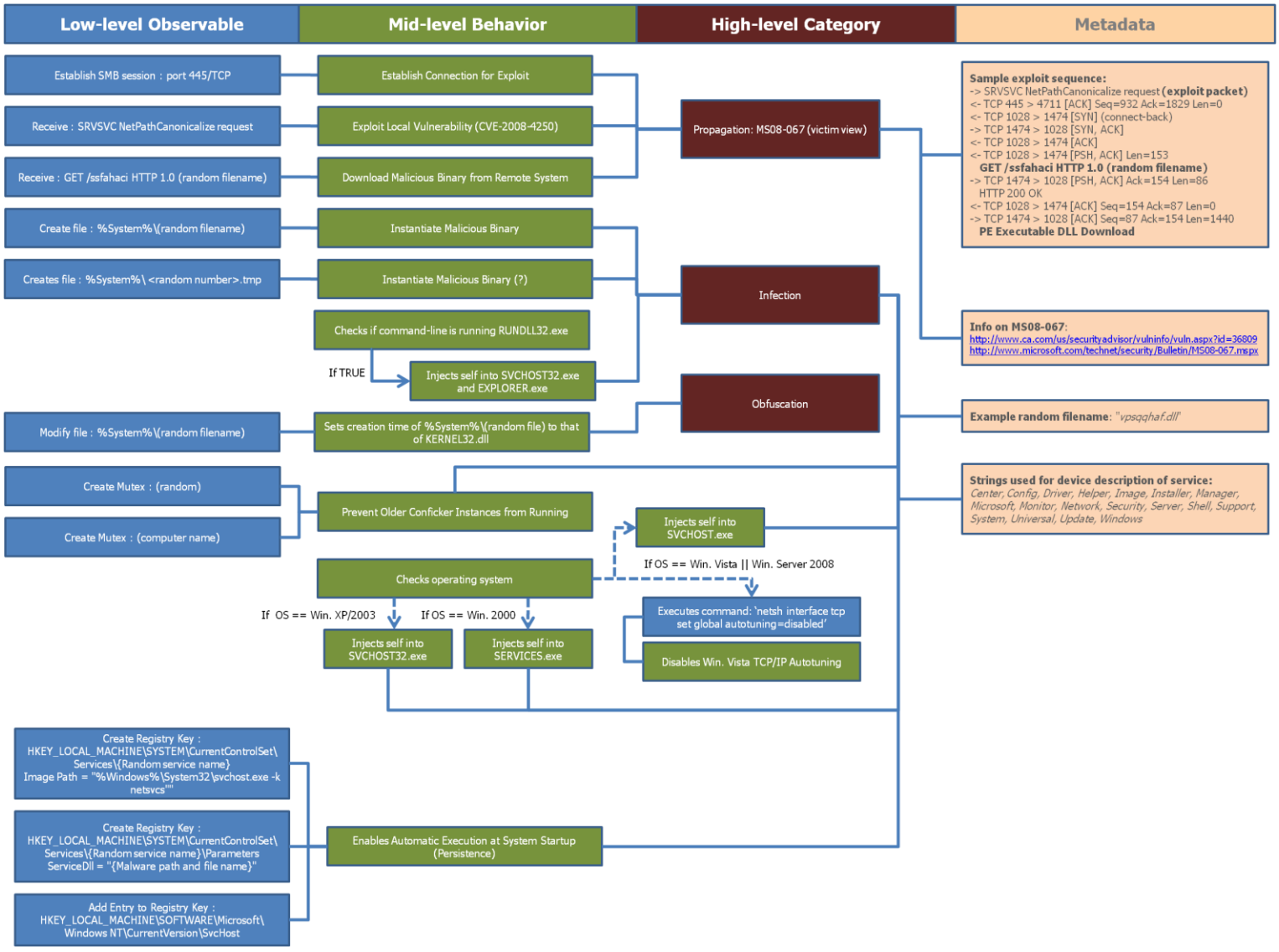
A.II: High-level Malware Processes and Mechanisms [19]

1) **Armoring** (mechanism)**:** a mechanism employed by malware for the purpose of impeding its analysis. Such mechanisms are typically targeted at specific analysis methods. This is potentially one of the categories in MAEC's high-level taxonomy. Examples: binary packing, anti-debugging mechanisms, virtual machine checks (dynamic analysis), etc.
2) **Command & Control** (mechanism)**:** a mechanism employed by malware for the purpose of executing commands specified by a remote attacker. Such mechanisms can be used for controlling a single, target machine to multitudes of "zombies" as in botnets. Examples: IRC-based, fast-flux, domain-generation, etc.
3) **Exfiltration** (process)**:** a mechanism employed by malware for the purpose of collecting and then transporting valuable information off of a system. While this is typically accomplished by uploading the data to a remote server, it can also be achieved by saving the data to an encrypted archive for physical retrieval. This is potentially one of the sub-categories of payload.
4) **Infection** (process)**:** the process by which malware instantiates or "installs" itself on a system; typically preceded by insertion. This is potentially one of the categories in MAEC's high-level taxonomy. Examples: writing a file to some directory, injecting a malicious binary into a process' address space, executing a malicious process, etc.
5) **Insertion** (process)**:** the process and vector used by malware for gaining initial entry into a system. This and infection are the only mechanisms common to all malware. This is potentially one of the categories in MAEC's high-level taxonomy. Examples: software vulnerability exploitation, social-engineering, etc.
6) **Metamorphism:** a mechanism used by malware for automatically re-coding itself each time it propagates or is otherwise distributed, primarily for the purpose of evading detection through physical signatures.

7) **Obfuscation** (mechanism)**:** a mechanism employed by malware for concealing its presence on a system. This is potentially one of the categories in MAEC's high-level taxonomy. Examples: utilizing the same name as a benign process, changing the last modified date of a malicious binary to that of a known binary, etc.
8) **Persistence** (process)**:** a process by which malware ensures continual execution on a system, independent of low-level system events such as shutdowns and reboots. This is potentially a sub-category of infection. Examples: insertion of an auto-run registry key, installation of a malicious binary as a system service, etc.
9) **Payload:** the specific malware attributes unrelated to insertion, infection, armoring, obfuscation, and self-defense. Therefore, a malware's payload can be thought of as the actions performed after the successful infection of a system, and is directly tied into the purpose behind the malware. This is potentially one of the categories in MAEC's high-level taxonomy.
10) **Polymorphism:** a mechanism employed by malware for the purpose of changing the appearance of its run-time code; some common methods include encryption and appending data. Like polymorphism, this is primarily intended to evade signature-based detection techniques.
11) **Propagation** (mechanism)**:** a mechanism utilized by malware for the purpose of spreading to other machines. This is potentially a sub-category of payload.
12) **Self-Defense** (mechanism)**:** a mechanism employed by malware that is intended to inhibit its removal after the successful infection of a system. This is potentially one of the categories in MAEC's high-level taxonomy. Examples: the removal/shutdown of AV products, the disabling of specific services, etc.

APPENDIX B: CONFICKER.B MAEC CHARACTERIZATION

B.I: Propagation, Infection, and Obfuscation Mechanisms



B.II: Self-Defense, Propagation, Payload, and Armoring Mechanisms