THE MITRE CORPORATION

THE MAEC™ LANGUAGE

OVERVIEW

DESIREE BECK, IVAN KIRILLOV, PENNY CHASE, MITRE
JUNE 12, 2014

Malware Attribute Enumeration and Characterization (MAEC™) is a standardized language for sharing structured information about malware based upon attributes such as behaviors, artifacts, and attack patterns.

By eliminating the ambiguity and inaccuracy that currently exists in malware descriptions and by reducing reliance on signatures, MAEC aims to improve human-to-human, human-to-tool, tool-to-tool, and tool-to-human communication about malware; reduce potential duplication of malware analysis efforts by researchers; and allow for the faster development of countermeasures by enabling the ability to leverage responses to previously observed malware instances.

Acknowledgements

The authors would like to thank the MAEC Community for its input and help in reviewing this document.

Trademark Information

Warnings

MITRE PROVIDES MAEC "AS IS" AND MAKES NO WARRANTY, EXPRESS OR IMPLIED, AS TO THE ACCURACY, CAPABILITY, EFFICIENCY, MERCHANTABILITY, OR FUNCTIONING OF MAEC. IN NO EVENT WILL MITRE BE LIABLE FOR ANY GENERAL, CONSEQUENTIAL, INDIRECT, INCIDENTAL, EXEMPLARY, OR SPECIAL DAMAGES, RELATED TO MAEC OR ANY DERIVATIVE THEREOF, WHETHER SUCH CLAIM IS BASED ON WARRANTY, CONTRACT, OR TORT, EVEN IF MITRE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.[1]

Feedback

The MAEC development team welcomes any feedback regarding the MAEC Overview. Please send any comments, questions, or suggestions maec@mitre.org.[2]

[1] For detailed information see [TOU].

[2] For more information about the MAEC Language, please visit [MAEC].

Table of Contents

1 Introduction

Malicious software – also called "malware" – has existed in one form or another since the advent of the first PC virus in 1971. It is presently responsible for a variety of malicious activities, ranging from the vast majority of spam email distribution via botnets to the theft of sensitive information via targeted social engineering attacks. Whether the attackers are script kiddies, “hacktivists,” criminals, or nation states, all use malware of some variety to negatively impact or gain access to an organization’s network. 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 as such, 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: even a single instance of undetected malware 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 malware that makes use of USB as its infection vector. Consequently, the main focus of the majority of anti-malware efforts to date has been on preventing damaging effects through early detection.

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

Today, effective malware detection and mitigation requires a variety of analysis and detection methods, and many different vendor or tool-specific data models have evolved as a result. Such data models are especially diverse in the manner in which they capture and describe higher level characteristics of malware such as behaviors. As a result, interpreting and correlating information from an assortment of disparate sources can be a difficult task.

The goal of the Malware Attribute Enumeration and Characterization (MAEC™, pronounced “mike”) effort is to provide a basis for transforming malware research and response. MAEC aims to eliminate the ambiguity and inaccuracy that currently exists in malware descriptions and to reduce reliance on signatures. In this way, MAEC seeks to improve human-to-human, human-to-tool, tool-to-tool, and tool-to-human communication about malware, reduce potential duplication of malware analysis efforts by researchers, and allow for the faster development of countermeasures by enabling the

ability to leverage responses to previously observed malware instances. As will be illustrated, the MAEC Language enables correlation, integration, and automation.

1.1 The MAEC Language

International in scope and free for public use, MAEC is a standardized language for sharing structured information about malware based upon attributes such as behaviors, artifacts, and attack patterns.

As shown in Figure 1-1, MAEC in its current state is composed of a data model that spans several interconnected schemas, thus representing the grammar that defines the language. These schemas permit different forms of MAEC output to be generated, which can be considered as specific uses of the MAEC grammar.

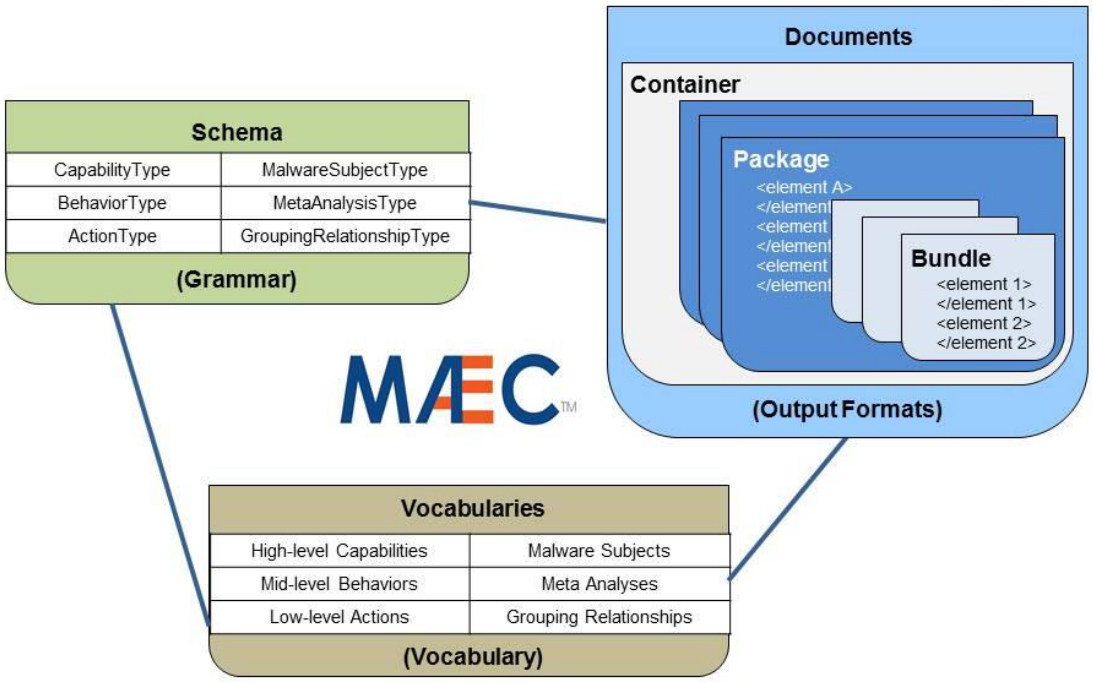

Figure 1-1. MAEC overview

As we will discuss later in the document, the MAEC Container, MAEC Package, and MAEC Bundle schemas are targeted at different use cases and thus capture different types of malware-related information.

Initially, the MAEC language may appear complex. *However, MAEC compatibility requires the implementation of only as much of MAEC that is needed or desired for a particular use case*. This will be illustrated in this document through the presentation of high level use cases and explicit examples.

1.2 Relationships to Other Languages and Formats

The MAEC Language directly imports and uses components of both the DHS-led and MITRE-developed Cyber Observable eXpression (CybOX) language [CYBOX] and the IEEE ICSG's Malware Metadata Exchange Format (MMDEF) [MMDEF]. Details are provided below.

In addition, complete threat analysis requires details of the specific vulnerabilities, weaknesses, and platforms targeted for exploitation by a malware instance, all of which can be captured in MAEC via references to other standards efforts, including Common Vulnerabilities and Exposures (CVE) [CVE], Common Weakness Enumeration (CWE) [CWE], and Common Platform Enumeration (CPE) [CPE]. In fact, an organization performing cyber threat analysis may choose to use Structured Threat Information eXpression (STIX™) [STIX], which was developed explicitly to characterize a rich set of cyber threat information in a standardized and structured manner; STIX can describe malware instances using MAEC characterizations through use of its MAEC schema extension[3]. Please refer to the "Ties to Existing Standards" web page for further information [TIE].

1.2.1 Cyber Observable eXpression (CybOX)

The Cyber Observable eXpression (CybOX) is a standardized language for the specification, capture, characterization and communication of events or stateful properties that are observable in a given operational domain. Cyber observables apply to numerous domains: threat assessment and characterization, malware characterization, operational event management, logging, cyber situational awareness, incident response, digital forensics, and cyber threat information sharing, among others.

A cyber observable is a *measurable event* or *stateful property* in the cyber context. Examples of measurable events include registry key creation, file deletion, and the sending of an HTTP GET request; examples of stateful properties include the MD5 hash of a file, the value of a registry key, and the name of a process.

Malware characterization with MAEC relies on the common implementation (structure and content) that CybOX provides for expressing cyber observables across and among MAEC's full range of use cases. Thus, whereas MAEC provides coverage of malware analysis context, indicators, behaviors, and capabilities, CybOX provides the underpinnings necessary to broadly cover actions and objects used in the malware context.

[3] The Characterizing Malware with STIX and MAEC white paper [MAEC$_S$] provides more details on the relationship between MAEC and STIX and when each should be used in the context of malware characterization.

MAEC makes prominent use of the CybOX Object and Action entities, among others. A simplified overview of the CybOX core schema that defines these entities is shown in Figure 1-2; the CybOX components that MAEC uses are shown in green.

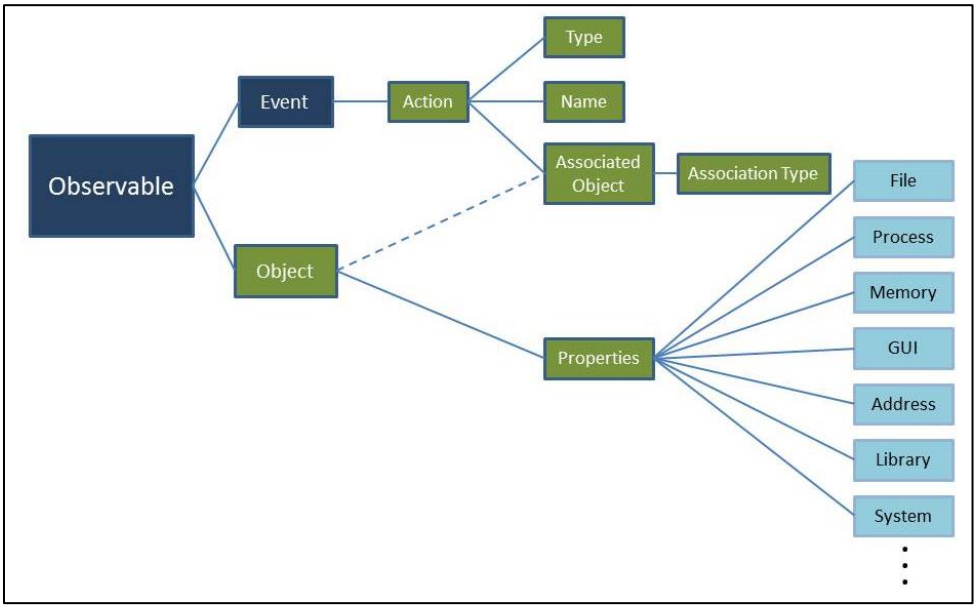

Figure 1-2. Cyber Observable eXpression (CybOX) schema simple overview

The CybOX "Properties" construct is an abstract placeholder for various predefined object types (e.g., File, Process, Memory), as defined in their own schemas. Such object schemas, shown in light blue in Figure 1-2, are maintained independently of the core CybOX schema.

1.2.2 Malware Metadata Exchange Format (MMDEF)

The Malware Metadata Exchange Format (MMDEF) [MMDEF] is being developed by the IEEE's Industry Connections Security Group (ICSG). The development of the original schema was led primarily by a group of Anti-Virus (AV) product vendors for the purpose of developing some way to augment shared malware samples with additional metadata. As such, it permits the characterization of some static features like hashes and file names, along with some basic behavioral features.

MAEC utilizes one major component from MMDEF v1.2. In particular, it uses the "metadata:fieldDataEntry" type for capturing information about malware prevalence in the wild.

1.3 Document Overview

We begin in Section 2 with an overview of the MAEC data models, and we provide high level use cases in Section 3. Open issues and challenges are discussed in Section 4, and

we enumerate sources of additional information on MAEC in Section 5. A glossary of terms and references are provided in the appendices.

2 Overview of the MAEC Data Models

The MAEC Language is defined by three data models, each of which is implemented in its own XML schema ([$SPEC_B$],[$SPEC_P$][$SPEC_C$]). There is also a default vocabularies schema ([$SPEC_V$]), which defines a default set of controlled vocabularies used within MAEC. As illustrated in Figure 2-1, "MAEC Bundle" is the (lowest) Tier 1 data model; "MAEC Package" is the (middle) Tier 2 data model; and "MAEC Container" is the (highest) Tier 3 data model. All three data models offer a stand-alone output format, so a lower level model can be used without the higher tier data model (although each model level requires all lower tiers). This three-tiered structure provides flexibility in the type and amount of information that can be shared.

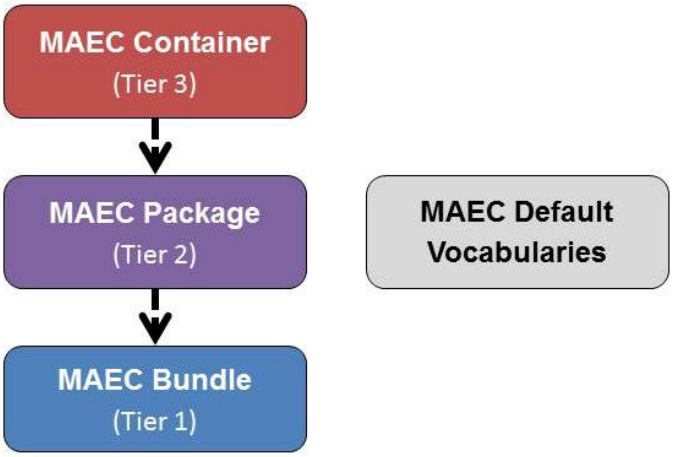

Figure 2-1. MAEC data models

In the subsections below, we will give an overview of each of the three data models. To motivate the use of the defined elements of the data models most effectively, we will work from lowest tier (MAEC Bundle) to highest tier (MAEC Container).

2.1 Overview of the MAEC Bundle Data Model

The MAEC Bundle data model provides the ability to capture and share data obtained from the analysis of a single malware instance. In terms of its most elemental structure, the MAEC Bundle data model can be thought of as having three interconnected layers, as shown in Figure 2-2. While the MAEC Bundle data model also includes other components, the Actions, Behaviors, and Capabilities are key pieces of its underlying structure, and so we discuss each in further detail below.

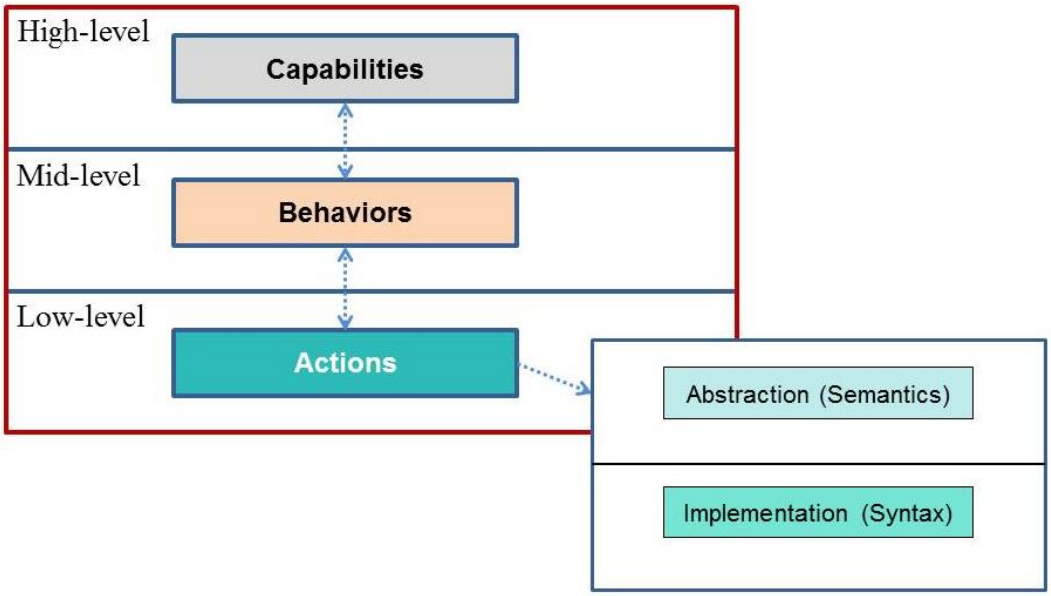

Figure 2-2. MAEC Bundle data model overview

2.1.1 Low-Level Actions

At the lowest layer, MAEC Actions answer the question of "what" the malware instance does on a system or network, and thus they characterize hardware accesses, network activity, and system state changes performed by malware. As such, MAEC Action entities describe attributes tied to the basic functionality and low-level operation of malware, including system state changes such as the insertion of a registry key or the creation of a file. Therefore, likely sources of such data include static analysis, dynamic analysis of malware binaries through sandboxes, and host-based and network-based intrusion detection and prevention systems (IDPS).

Actions can describe a wide range of activities, although initially we have concentrated on defining those that can be achieved through low-level API calls. By design, MAEC Actions are relatively devoid of any significant intention: while they answer the question of "what" the malware does, they don't answer the question of "why" the malware performed the actions in the first place.

Actions are represented at both semantic and syntactic levels in order to abstract actions from their implementations. Such abstracted Actions allow for the construction of a more concise grammar and also facilitate correlation between malware instances that may do similar things at this level but with vastly different implementations (such as malware targeted at different platforms). On the other hand, the implementation of a particular action or action type may provide insight that can be valuable for correlation or even attribution.

2.1.2 Mid-Level Behaviors

The more interesting structure of the MAEC Bundle data model begins at the middle layer, which we term Behaviors. Behaviors are aimed at organizing and defining the purpose behind low-level Actions, whether in groups or as singletons. Thus, Behaviors serve to describe "how" a malware instance operates at significant level of abstraction, and can therefore represent discrete components of malware functionality at a level that is useful for analysis, triage, and detection.

For instance, the description of a registry entry created or modified by malware can be useful for establishing its presence on a system. However, it does not give any insight into why the malware created or manipulated the registry entry. Such a registry entry inserted or modified by a malware instance could be associated with different behaviors. For example, the registry entry could be used to ensure that the malware gets executed when the system boots, or it could be used as a simple flag to indicate that the system has been infected. As we will discuss in the next section, including the necessary components for characterizing such mid-level Behaviors in the MAEC Language allows for the accurate description of the possible high-level intent or goals (i.e., Capabilities) that are behind the low-level Actions being performed by malware.

2.1.3 High-Level Capabilities

At the more conceptual and upper-most layer, MAEC defines Capabilities[4]. Similar to the relationship between Behaviors and Actions, Capabilities serve to organize groups of Behaviors, and therefore they offer a standard way of capturing the set of high-level abilities that a malware instance possesses. However, the key difference between Behaviors and Capabilities is that while Behaviors are intended to describe "how" a malware instance operates, Capabilities are meant to state "what" it is capable of doing. In this sense, a Behavior may serve to describe a particular implementation of a Capability in a malware instance.

For example, ensuring that a malware instance is executed at start-up (e.g., by creating a binary copy of the malware somewhere on the local hard disk and/or by creating a particular registry entry) is a Behavior that is typically part of a '*Persistence*' Capability. Other examples of Capabilities include '*Propagation*', '*Self-Defense*', and '*Data Theft*'. Because there is a relatively low upper bound on the number of possible capability types, MAEC Capabilities can be useful in terms of understanding the functionality of malware at a very high level.

Once higher order classifications are made, we envision that the Capabilities taxonomy will have "views" intended for different target audiences. For example, forensic analysts may only be interested in looking at malware payload Capabilities and Behaviors, while a

[4] MAEC Capabilities were previously called "MAEC Mechanisms."

SOC analyst might want to view Capabilities and Behaviors related to command and control.

2.1.4 Example Mapping

As a very simple example of how a malicious activity can be mapped between the MAEC Bundle levels, let's say that a malware instance calls the Windows "CreateFile" API to create the file "xyz.dll." This event would first be mapped to the '*Create File*' Action, and after further investigation, we might conclude that this file was created as a means of instantiating a malicious binary on a system, thus mapping to a '*Malicious Binary Instantiation*' Behavior. Finally, the '*Malicious Binary Instantiation*' Behavior could be considered part of a malware '*Persistence*' Capability. This is illustrated in Figure 2-3 below.

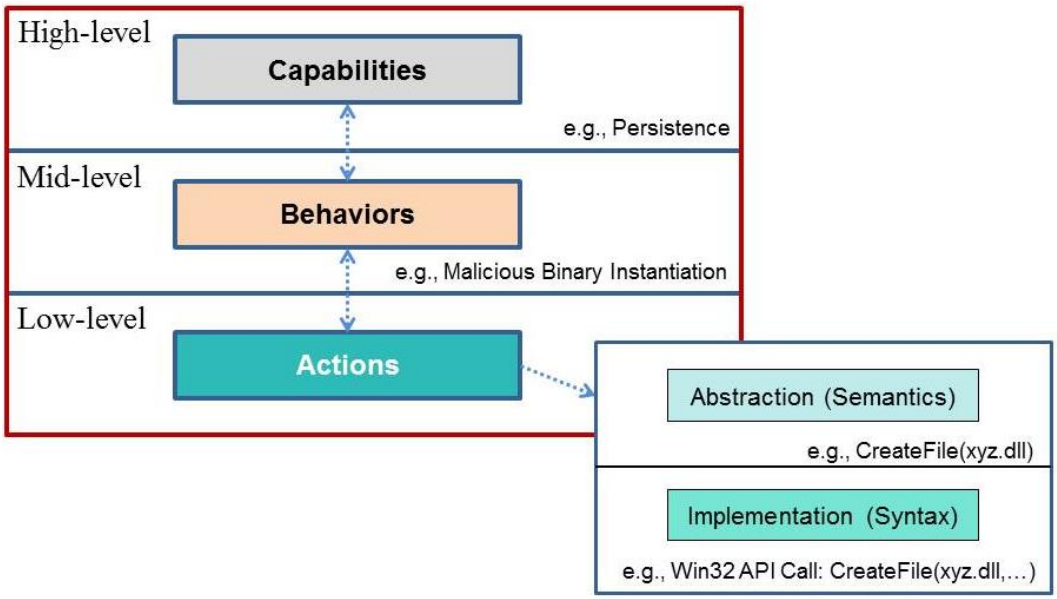

Figure 2-3. MAEC data model mapping example

2.1.5 The MAEC Bundle Output Format

The MAEC Bundle XML schema (namesake of the MAEC Bundle data model) is currently the standard output format[5] that can be used to describe a *single* malware instance as a MAEC Bundle schema instance. As shown in Figure 2-4, the MAEC Bundle schema serves as a container and transport mechanism for use in storing and subsequently sharing MAEC-encoded information about malware, which may include MAEC Actions, Behaviors, and Capabilities as well as other attributes obtained from the characterization of a malware instance.

[5] Other output formats, such as JSON, are being considered for future implementations.

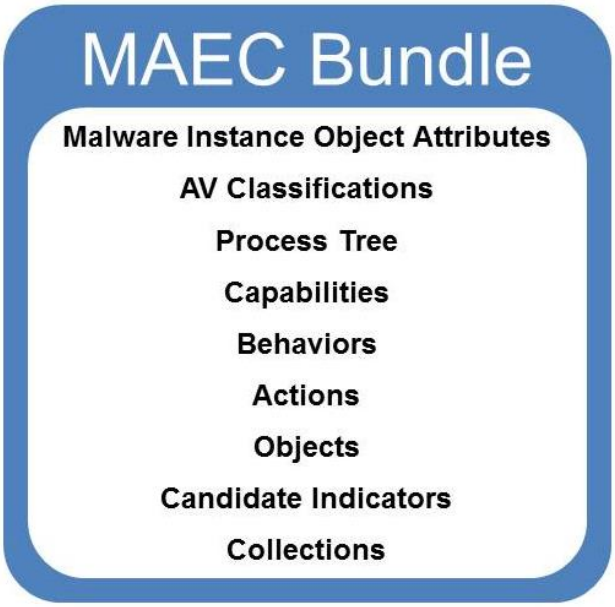

Figure 2-4. MAEC Bundle schema overview

A MAEC Bundle is very flexible and can be used to describe anything from a particular insertion method (composed of several low-level Actions and mid-level Behaviors) to any or all of the attributes listed in Figure 2-4. A MAEC Bundle can contain intelligence-derived indicators as well as other signatures and patterns useful in network and host-based intrusion detection.

High level definitions of the basic components of the MAEC Bundle schema are given below. We give further details of the components of the MAEC Bundle schema in [$SPEC_B$].

- Malware Instance Object Attributes – Captures details of the malware instance that the MAEC Bundle characterizes using its enumerations and schema. Most commonly, this is a file object with a few attributes, such as name, size, and cryptographic hashes.
- AV Classifications – Captures any Anti-Virus scanner tool classifications of the malware instance.
- Process Tree – Specifies the observed process tree of execution for the malware instance.
- Capabilities – Encompasses all of the MAEC Capabilities in the MAEC Bundle. Each Capability entity can contain information such as properties, Strategic and Tactical Objectives associated with the Capability (defined next), related Behaviors, and relationships to other Capabilities.
    - Strategic Objective – Capture the details of a Capability with additional granularity. A Capability can have one or more Strategic Objectives that it attempts to carry out.

    - Tactical Objective – Capture the details of a Strategic Objective with additional granularity. A Strategic Objective can have one or more Tactical Objectives associated with it, which further define the Strategic Objective.
- Behaviors – Encompasses all of the MAEC Behaviors in the MAEC Bundle. Each Behavior entity can contain information such as a textual description of the Behavior, related Actions, and relationships to other Behaviors.
- Actions – Encompasses all of the MAEC Actions in the MAEC Bundle. Each Action entity can contain information such as the type of Action that it represents (e.g., '*create file*', '*copy file*'), discovery method and associated tools, and relationships to other Actions.
- Objects – Encompasses all of the MAEC Objects in the MAEC Bundle. Each Object entity can contain information such as the type of Object that it represents (e.g., '*file*', '*process*'), specific properties of the Object (e.g., '*file name*', '*process name*'), and relationships to other Objects.
- Candidate Indicators – Encompasses all of the MAEC Candidate Indicators in the Bundle. Each Candidate Indicator entity can contain information such as importance, author, description, and target information.
- Collections – Encompasses all of the MAEC Collections in the MAEC Bundle: Behavior Collections, Action Collections, Object Collections, and Candidate Indicator Collections. Each Collection entity can contain information such as a text description of the Collection, a characterization of how the elements are related, and a list of the Behaviors/Actions/Objects/Candidate Indicators themselves.

A MAEC Bundle can be used to encompass a set of malware attributes with a particular significance (e.g., a persistence-related Behavior), or it can simply serve as a generic container for MAEC-characterized malware data pertaining to a single malware instance. Therefore, it can be used with as little or as much information as desired; any further meaning beyond the explicit data stored in the MAEC Bundle is determined by its producer.

Note that instead of capturing all malware attributes associated with a malware instance in a single MAEC Bundle, a user may have a need to use multiple MAEC Bundles. For example, a user may choose to use several different tools in the analysis of the malware instance, each of which creates a separate MAEC Bundle. In this case, the collection of MAEC Bundles can be shared as part of a MAEC Package, described next in Section 2.2. However, it is also perfectly valid to merge multiple MAEC Bundles into a single MAEC Bundle, if so desired. Identifiers should be preserved so that after merging, it will be possible to determine the original source Bundle of a particular entity through its identifier.

2.2 Overview of the MAEC Package Data Model

Before we discuss the MAEC Package data model, we must define the Malware Subject entity. A Malware Subject contains details of a particular malware instance (e.g., a file as

identified by MD5 and/or SHA1 hash), any minor variants of the same instance that may have been observed (e.g., the same file but with different names), along with all of the analyses that were performed on the instance, any findings generated from the analyses, and any other metadata. As such, the Malware Subject is MAEC's representation of a malware instance and all of the known data associated with it.

The MAEC Package data model enables a user to share MAEC characterized data for one or more Malware Subjects; in most such cases, the Malware Subjects are related in some manner. For example, the Malware Subjects captured in a MAEC Package might include files that are created or dropped during a dynamic analysis, variants of the same malware family, or files that are identified as being similar by a clustering algorithm. In addition to encompassing a collection of MAEC Bundles associated with each of the Malware Subjects, the MAEC Package data model also defines elements that enable the sharing of analysis and relationship information.

Figure 2-5 gives a pictorial overview of a MAEC Package. As shown, a MAEC Package encompasses one or more Malware Subjects, each which includes its own analysis metadata and MAEC Bundles (i.e., Findings Bundles, defined below). Also captured within the MAEC Package are any relationships between pairs of Malware Subjects, along with relationship information for the *entire* collection of Malware Subjects (i.e., Grouping Relationships, defined below).

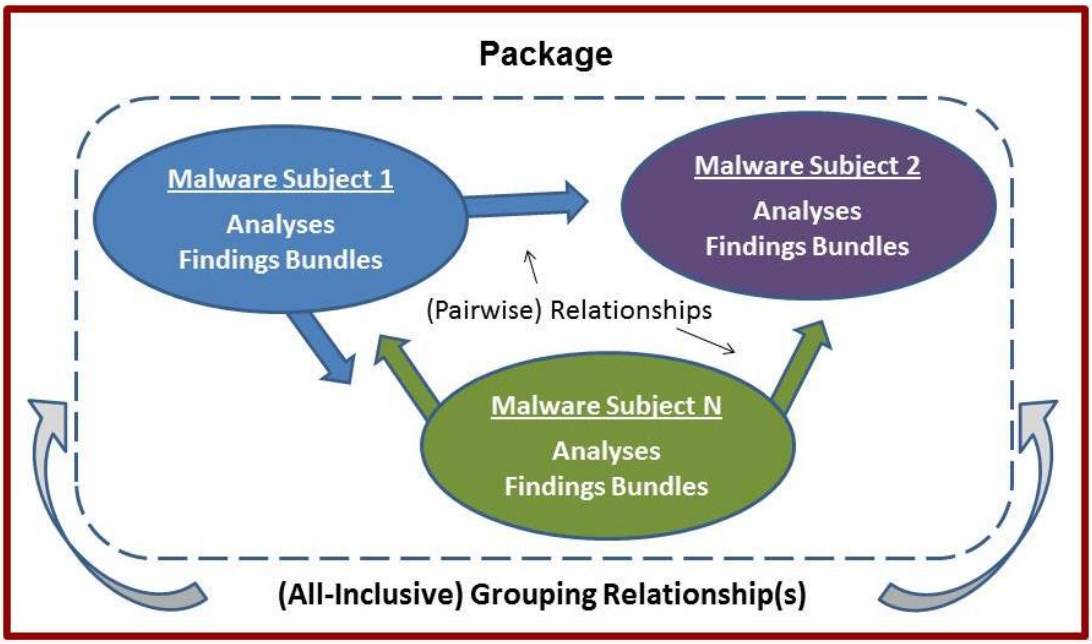

Figure 2-5. MAEC Package data model overview

Explicit definitions are given in [SPEC$_P$], with high level definitions defined in the next section describing the MAEC Package output format.

2.2.1 The MAEC Package Output Format

The MAEC Package XML schema is currently the standard output format that can be used to describe *one or more* Malware Subjects using MAEC's enumerations and schema. As illustrated in Figure 2-6, the content of a MAEC Package includes a set of Malware Subjects and Grouping Relationship information, where the content of a Malware Subject includes additional information: Malware Instance Object Attributes, labels, configuration details, development environment details, minor variant information, field data, analysis information, MAEC Bundles associated with the Malware Subject, information about the relationships between the Malware Subject of focus and other Malware Subjects, and compatible platform information. In essence, a MAEC Package enables MAEC Bundle management, allowing users to share multiple MAEC Bundles and associated metadata for one or more Malware Subjects.

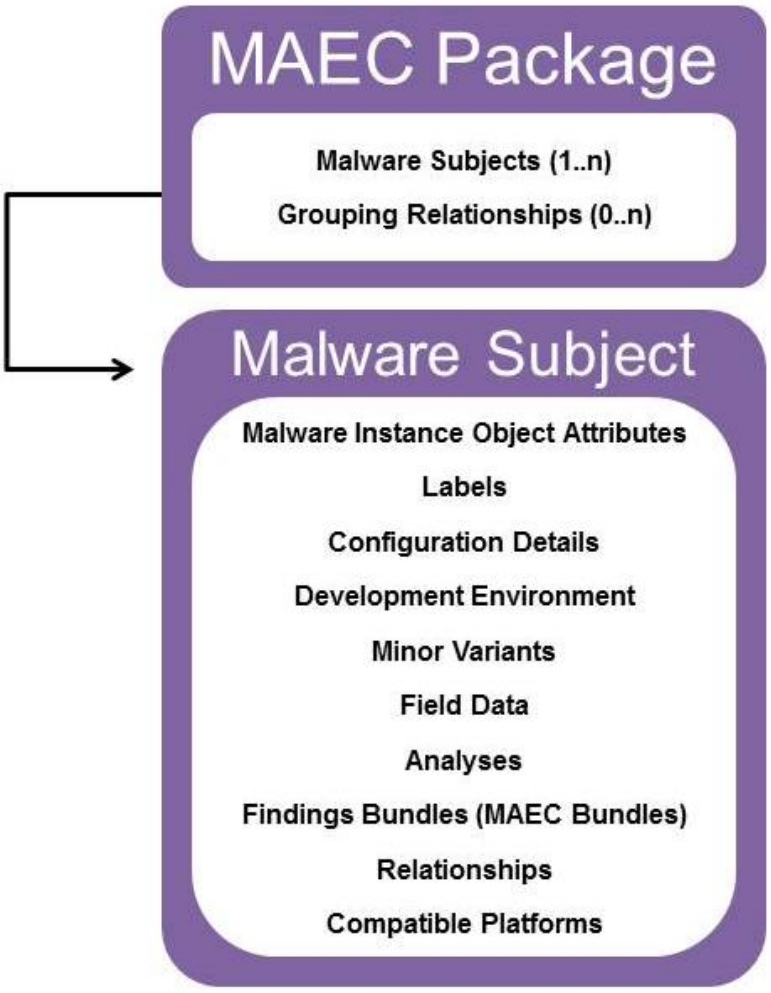

Figure 2-6. MAEC Package schema overview

The structure of the MAEC Package schema is provided in detail in [$SPEC_P$], but we give high level definitions for the basic components below:

- Malware Subject – represents a single malware object (most commonly a file) and its associated metadata:
    - Malware Instance Object Attributes – Details of the specific properties of the malware instance characterized by the Malware Subject; for example, its MD5 hash. Note that this information may be repeated in a MAEC Bundle if the MAEC Bundle is to be self-contained.
    - Label – specifies a commonly accepted label to describe the Malware Subject, e.g., "worm." More than one label may be specified through the use of multiple instances of this field.
    - Configuration Details – Captures details of the configuration specified for the Malware Subject, such as configuration parameters.
    - Development Environment – Captures details of the development environment used in the creation of the malware instance characterized by the Malware Subject.
    - Minor Variants – Captures any observed minor variants of the malware instance characterized by the Malware Subject, such as identical files with different names.
    - Field Data – Captures field data and prevalence information relating to the malware instance characterized by the Malware Subject.
    - Analyses – Captures analysis-related details for the Malware Subject such as analyst, source, summary, and tool information. Analyses can reference one or more individual MAEC Bundles to denote that the findings of the analysis are captured in the Bundles.
    - Findings Bundles - Set of MAEC Bundles pertaining to the Malware Subject of focus. For example, these MAEC Bundles could capture the output of different tools, some data obtained through manual malware analysis, etc. The term "Findings_Bundles" is used rather than simply "Bundles" to imply that the content was derived from analysis.
    - Relationships – Captures uni-directional relationships between the Malware Subject of focus and other Malware Subjects. Examples include '*downloaded by*,' '*dropped by*,' '*downloads*,' and '*drops*.'
    - Compatible Platform – Specifies a single platform with which the Malware Subject is compatible (i.e., on which the Malware Subject can execute). More than one compatible platform may be specified through the use of multiple instances of this field.
- Grouping Relationship – Specifies the particular relationship between all of the Malware Subjects encompassed in the MAEC Package. Example relationships include '*same malware family*' and '*clustered together*' (possibly by a malware analysis clustering algorithm).

2.3 Overview of the MAEC Container Data Model

The MAEC Container data model enables a user to share any collection of MAEC characterized data. Currently, the data model simply enables a user to share a collection

of MAEC Packages. The data model will be expanded and further defined to include other components as dictated by future needs.

2.3.1 The MAEC Container Output Format

The MAEC Container XML schema is currently the standard output format that can be used to share any collection of MAEC data. As illustrated in Figure 2-7, the content of a MAEC Container simply includes a set of MAEC Packages.

Figure 2-7. MAEC Container schema overview

3 High Level Use Cases for the MAEC Language

At its highest level, MAEC is a domain-specific language for non-signature based malware characterization. Because MAEC provides a common vocabulary and grammar for the malware domain, it follows that the majority of the use cases for MAEC are motivated by the unambiguous and accurate communication of malware attributes enabled by MAEC.

To illustrate this in more detail, we provide high level use cases in four general areas[6]: malware analysis, cyber threat analysis, intrusion detection, and incident management. In this section, we generally do not specify whether the information is captured in a single MAEC Bundle or multiple MAEC Bundles contained in a MAEC Package; rather, we are simply illustrating the utility that MAEC provides in terms of characterizing malware.

3.1 Malware Analysis

As shown in Figure 3-1, MAEC will typically be used to encode the data obtained from malware analysis. In such a scenario, a malware instance is analyzed automatically or manually using either dynamic or static methods. The results are then captured using the MAEC schema and either a single MAEC Package (with one or more MAEC Bundles) or one or more standalone MAEC Bundles are generated to communicate the analysis results. As we will also briefly discuss, MAEC Packages and MAEC Bundles can also be used to help with visualization, to capture data for storage in analysis-oriented repositories, and as a means for standardizing tool output.

3.1.1 Static and Dynamic Malware Analysis

The analysis of malware using static and dynamic/behavioral methods is critical for understanding the malware's inner workings. Information obtained from such analyses can be used for malware detection, mitigation, the development of countermeasures, and as a means of triage for determining whether further analysis is necessary.

In terms of static analysis, MAEC can be used to capture the particular details that are extracted from a malware instance. Details can range from the static attributes of a malware instance binary, such as information on the packers that the instance was packed with, to interesting code snippets obtained from the manual reverse engineering of the instance binary code.

[6] Please see [$EXAM_D$] for detailed examples and very specific use cases that explicitly illustrate the use of MAEC Bundles and MAEC Packages.

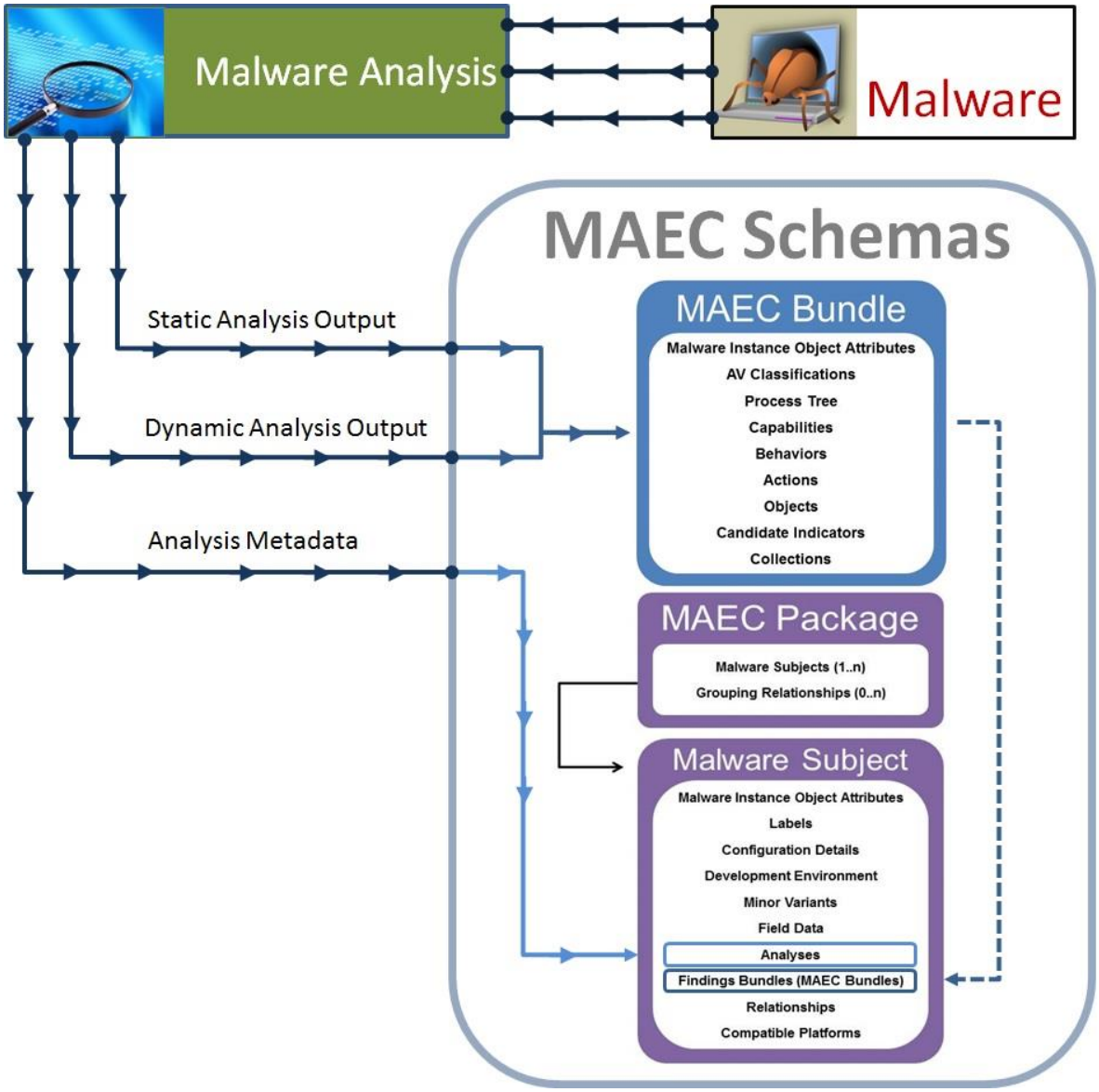

Figure 3-1. Malware Analysis with MAEC

With regard to dynamic analysis, MAEC can be used to capture details of the particular actions exhibited by executing the malicious binary or code. This can be done at multiple levels of abstraction, starting with the lowest level (which is most commonly captured as some form of native system API call) and extending to higher levels describing a particular unit of malicious functionality, such as keylogging or vulnerability exploitation.

For both static and dynamic analysis, MAEC can capture information on each analysis as a separate item, including the particular findings of the analysis, information on any tools that were used, and other associated data such as the details of the analysis environment. As such, MAEC permits all of the analyses for a malware instance to be described in a standard fashion and captured in a single document, the MAEC Package.

3.1.2 Malware Visualization

In addition to capturing the output of one or more malware analyses, a MAEC Package or MAEC Bundle can also be used as a standard format to create visualizations of malware behavior. Such visualizations permit clear assessment of the low-level Actions, mid-level Behaviors, and high-level Capabilities performed by malware and facilitate comparison between two or more malware instances.

While no visualization tools currently exist to display MAEC content, we expect that future tools will provide much needed insight to analysts for quickly identifying similarities between malware instances and between analysis outputs from different tools.

3.1.3 Analysis-Oriented Malware Repositories

Malware repositories oriented toward analysis often have very specific requirements, and it is common for security organizations to use their own custom schemas for storage of data in such repositories. From a malware analysis standpoint at a local level, custom repositories can serve their purpose. However, sharing or exporting data from custom repositories can be difficult and time-consuming due to the need to translate between multiple proprietary schemas, and the usefulness of a custom repository as a long-term analysis resource can be limited if the schema is not suitably expressive.

MAEC is well-suited for use as a common intermediate format for mapping between dissimilar malware repository schemas so that analysis information stored in disparate repositories can be shared, allowing teams or organizations to quickly leverage each other's analysis results. Furthermore, for appropriate database architectures, using the MAEC data model in malware repositories would not only enable information sharing but would also permit improved data-mining due to MAEC's structuring and labeling of malware attributes (MAEC can serve as a physical or logical data model, depending upon the architecture). For example, an analyst could query a MAEC-based repository for malware instances that exhibit a particular MAEC-defined Action, Behavior, or Capability.

3.1.4 Standardized Tool Output

Like human analysts, malware analysis tools that automatically generate reports lack consistency in reporting. Not only does this make it difficult to correlate output between tools, it also makes it difficult to evaluate the breadth of coverage of individual tools. These issues would be mitigated, and ingestion of tool results into analysis-oriented repositories made easier, if malware analysis tools were to adopt MAEC as a common output format (depicted in Figure 3-2).

Given MAEC's extensive support for capturing the output of both static and dynamic malware analysis, it follows that MAEC could be used as a standard output format for such tools. Native support for MAEC in this manner is already present in several tools, and

there are also existing translator utilities for converting certain tool outputs into MAEC. Further details are available on the MAEC Web site [MAEC].

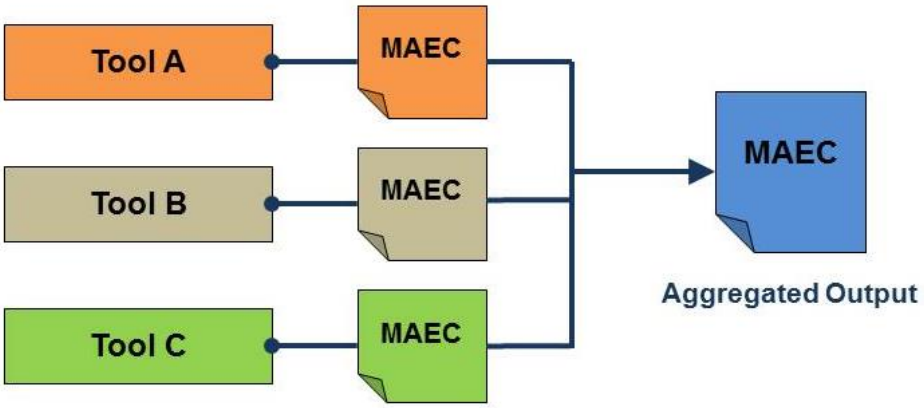

Figure 3-2. Standardized Analysis Tool Output with MAEC

Standardized tool output using MAEC could also be used as objective criteria in the assessment of anti-malware tools. In this sense, a tool would be assessed on the basis of its ability to detect all of the MAEC-defined attributes associated with a particular malware type or class. If a tool could not detect *all* of the MAEC-defined attributes associated with a particular malware type, then it could not claim to be capable of detecting that malware type or class.

3.2 Cyber Threat Analysis

Beyond analysis of a particular malware instance, an organization defending against cyber adversaries often engages in the broader task of cyber threat analysis – the collection and analysis of cyber attack and threat information in relation to the organization's potential vulnerabilities. Cyber threat information includes analysis results of malware instances, along with additional threat data such as intent and kill-chain information and adversary tools, techniques, and procedures. Given a corpus of threat data, skilled cyber analysts must identify patterns of related activities, attribute activities to particular threat actors, identify and implement mitigation strategies, and anticipate future launches of previously-seen and similar attacks.

For successful cyber threat analysis, detailed analysis information about the malware instances must be obtained. For example, triage procedures may reveal information such as spear-phishing email headers or URLs to malicious websites, while in-depth malware analysis may uncover command and control domain names and IP addresses. Although today's malware reporting may include such details, currently there is usually no standardization between reports, and reports do not typically reference relevant standards (e.g., CVE). As a result, security operations staff and others charged with protecting systems from cyber threats may find it difficult to judge the true threat that malware represents. However, capturing this information in MAEC will result in a threat

being more readily understood and evaluated because the information will be more consistent across analysts and incidents. Furthermore, MAEC's standardized encoding of the Capabilities exhibited by a malware instance will allow for the accurate discernment of the threat that the malware poses to an organization and its infrastructure.

3.2.1 Malware Threat Scoring System

This linkage between MAEC and other standards efforts (see Section 1.2) could also allow for the creation of a malware threat scoring system, similar to that of the Common Vulnerability Scoring System (CVSS) [CVSS] for software vulnerabilities. MAEC's link to relevant 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.) and degree of entrenchment/propagation, for example.

3.2.2 Attribution

In cyber threat analysis, it is often useful to characterize the tools, techniques, and procedures 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 helpful for the purposes of attribution. Accordingly, with malware being one of the most prevalent tools used 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 capability, as its standard vocabulary and grammar permits the accurate identification of malware attributes observed in previous attacks, thus allowing for the construction of an accurate link between attackers and their malware toolset, based on previously observed and characterized malware.

3.3 Intrusion Detection

Effective intrusion detection is central to keeping networks safe from malicious actors. Using MAEC to characterize malware based on its attributes provides actionable information for malware detection and assessment: more specifically, low-level Objects and Actions, mid-level Behaviors, and high-level Capabilities enable malware detection.

Unlike a physical signature, a single MAEC characterization, represented by a MAEC Bundle or MAEC Package, can provide data that can be used to detect multiple malware instances. Because there are a finite number of ways of implementing a particular software behavior (for instance, keylogging), particularly at the assembly level, there is likely to be an intersection of such attributes between multiple malware instances. Therefore, the MAEC characterization of a single malware instance – to include behavior-based indicators to detect the presence of the malware – can permit an intrusion detection system to detect malware families and even otherwise un-related malware that have certain attributes in common with the malware instance.

MAEC-characterized data can also be used for host-based malware detection via translation into the Open Vulnerability and Assessment Language (OVAL) [OVAL] or Open Indicators of Compromise (OpenIOC) [IOC] formats.

3.4 Incident Management

When a cyber incident occurs, a defending organization must coordinate their response among a team of analysts and decision makers. In some cases, the organization may solicit help from Computer Security Incident Response Teams (CSIRTs), law enforcement, Internet Service Providers (ISPs), or product vendors. Regardless of the underlying threat, when numerous people or parties are involved, even within the same organization, effective incident management is extremely important. As we discuss below, a uniform malware reporting format, standardized malware repositories, and the ability to verify remediation procedures – all based on the MAEC data model – greatly enhance malware-related incident management efforts.

3.4.1 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. Furthermore, reported information often excludes key malware attributes that may be useful for mitigation and detection purposes (e.g., the specific vulnerability that is exploited). Certainly, the value of malware reporting to end-users is significantly degraded without an encompassing, common format.

MAEC's standardized vocabulary and grammar for use in malware reporting facilitates the creation of a separate, uniform reporting format. 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 different authors and organizations. Also, because current reporting is typically captured in free-form text format, the structure provided by MAEC offers additional capabilities such as machine-based manipulation and automated ingest of malware reporting data.

3.4.2 Malware Repositories

As discussed in Section 3.1.3, there is typically disparity among the malware repository schemas currently in use by different organizations, with essentially every security organization using their own custom schema. This makes effective sharing of analysis information difficult, even if both parties want to share analyses and other data.

MAEC provides a solution. As discussed previously, the MAEC schema is well-suited to be used as a common, standardized, intermediate format for mapping between dissimilar malware repository schemas so that analysis information stored in disparate repositories can be shared.

3.4.3 Remediation

One of the current realities of cyber security is that malware detection and prevention of infection is not always possible, especially with new and targeted malware threats. Consequently, remediation of malware infections has become increasingly important. Unfortunately, most conventional AV tools and utilities are not capable of removing every trace of a detected malware instance. 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, potentially resulting in a misallocation of remediation resources. Even worse, an incomplete remediation could render the system unstable, as well as prone to future infection.

MAEC provides a means for communicating the exact artifacts and low-level attributes associated with a malware instance, permitting greatly improved remediation of malware infections. Using MAEC, administrators can perform manual remediation based on the data contained in a MAEC Bundle or Package, or they can verify the remediation performed by another tool by checking for the existence artifacts captured in a MAEC Bundle or Package.

4 Requirements for the MAEC Language

The following requirements were developed based on the goals of MAEC and the needs outlined in the use cases described in Section 3. These requirements apply to the MAEC Language itself and establish the MAEC Language as the standardized framework for expressing structured information about malware. At the highest level are the Basic Requirements, which capture the essence of the MAEC goals and use cases. Each of these requirements is further expanded and refined into general content requirements and individual classes of schema requirements: MAEC Bundle requirements, MAEC Package requirements, and MAEC Container requirements.

4.1 Basic Requirements

The basic requirements listed in this section form the foundation of the MAEC Language and are further refined and expanded upon in Section 4.2, Detailed Requirements.

4.1.1 Capturing Malware Instance Identity

- The language MUST be capable of capturing the identity of the malware instance(s) being characterized.

4.1.2 Expressing Analysis Results

- The language MUST be capable of expressing the results of dynamic analysis that was performed on one or more malware instances.
- The language MUST be capable of expressing the results of static analysis that was performed on one or more malware instances.
- The language MUST be capable of expressing the results of manual analysis that was performed on one or more malware instances.
- The language MUST be capable of expressing the results of automated analysis that was performed on one or more malware instances.

4.1.3 Content Exchange

- The language MUST allow for the exchange of a MAEC Bundle as a single unit of content.
- The language MUST allow for the exchange of a MAEC Package as a single unit of content.
- The language MUST allow for the exchange of a MAEC Container as a single unit of content.

4.2 Detailed Requirements

The detailed requirements expand upon the general requirements listed in the previous section.

4.2.1 General Content Requirements

These general requirements apply to all content[7] written in the MAEC Language.

- The language MUST require that all content specify the language version with which it complies.
- The language MUST allow all content to specify when it was created.
- The language MUST allow content to contain additional information that is relevant to the creation of the document.
- All major components of the language MUST be reusable.
- Components of the language MUST have identifiers that are unique in the scope of the document.

4.2.2 MAEC Bundle Requirements

These requirements apply to MAEC Bundles and further refine the basic requirements listed above.

- The language MUST allow for the exchange of a MAEC Bundle as a single unit of content.
- A MAEC Bundle MUST allow for the capture of the identity of the malware instance which it characterizes.
- A MAEC Bundle MUST allow for the capture of metadata for specifying the general type of analysis-derived results which it expresses.
- A MAEC Bundle SHOULD be capable of capturing all analysis-derived malware characteristics for a malware instance.

4.2.3 MAEC Package Requirements

These requirements apply to MAEC Packages and further refine the basic requirements listed above.

- A MAEC Package MUST allow for the capture of the identity of one or more malware instances which it characterizes.
- A MAEC Package MUST be capable of capturing information about the analysis tools used to generate analysis results, including the tool name and version information.
- A MAEC Package MUST allow for the capture of the relationships between the Malware Subjects which it characterizes.
- A MAEC Package MUST allow for the capture of the analysis-derived findings for each malware instance that it characterizes.
- A MAEC Package and its MAEC Malware Subjects SHOULD be capable of capturing any metadata related to the malware instances which they characterize, including but not limited to analysis metadata and field data.

[7] That is, any data expressed in MAEC and output in the form of a MAEC document.

4.2.4 MAEC Container Requirements

These requirements apply to MAEC Containers and further refine the basic requirements listed above.

- A MAEC Container MUST allow for the capture of one or more groups of characterized malware instances.

5 Open Issues & Future Work

In this section, we outline the primary issues and future work that we foresee going forward with the next iterations of MAEC development beyond the current version. We welcome the community's input and involvement in resolving these and other issues, as well as proposing other additions or changes for future versions.

Many of MAEC's initial implementation issues described in [WP] have been successfully resolved. For example, in order to avoid a single, monolithic schema, we have divided the data models into three different components – Bundle, Package, and Container – and we have separately defined MAEC's default vocabularies. As another example, within the MAEC Bundle, we defined the Process Tree to capture temporal and order-based relationships between attributes. However, as we outline below, challenges remain as we move forward with further MAEC development.

5.1 Defining Additional Actions

In the current version of MAEC, Actions have been defined as they correspond to API calls of the Windows operating system. It would not be difficult to expand MAEC to capture similar library calls associated with non-Windows platforms. However, identifying and enabling MAEC to capture low-level actions that are not associated with explicit library calls will be more challenging.

5.2 Defining Behaviors

In the current and earlier versions of MAEC, it was fairly straightforward to identify low level Actions of malware through the recording of the API calls performed during some form of instrumented binary execution (e.g., sandboxing). However, due to their sheer volume and abstract nature, it is much more difficult to interpret those results so that they have meaning.

In MAEC, deriving meaning from low-level Actions requires the definition of mid-level Behaviors, a task that poses several difficulties. For example, should Behaviors be defined independently of explicit Actions? Or, should Behaviors be defined by identifying sets of related Actions? Possibly, a combination of methods will be most effective. Regardless, the number of Behaviors that will be ultimately be defined will be large, and as with all large enumerations and taxonomies, challenging to manage.

5.3 Associating Actions, Behaviors, and Capabilities

In MAEC v4.1, we have made a good start at defining both low-level Actions and high-level Capabilities. However, beyond the difficulty of defining Behaviors (see Section 5.2), there is the difficulty of associating specific Behaviors with sets of Actions. For instance, how explicit should such definitions be? Should they simply represent an observed set of

Actions for a particular Behavior, or can they instead encompass a pattern (or set of patterns) that more generally expresses the semantics of the Behavior?

Similarly, we must also define a method of associating particular Capabilities with sets of Behaviors. Only by having the ability to link Actions to Behaviors to Capabilities can actionable, higher level meaning be relayed in a MAEC document. We anticipate both of these tasks to be especially challenging.

6 Additional Documents and Information

As shown in Figure 6-1, this document (highlighted in yellow) is one of several overview, specification, and supporting documents associated with the MAEC Language. Icons are used to indicate whether the material is contained in an actual document ( ) or captured on a Web page ( ).

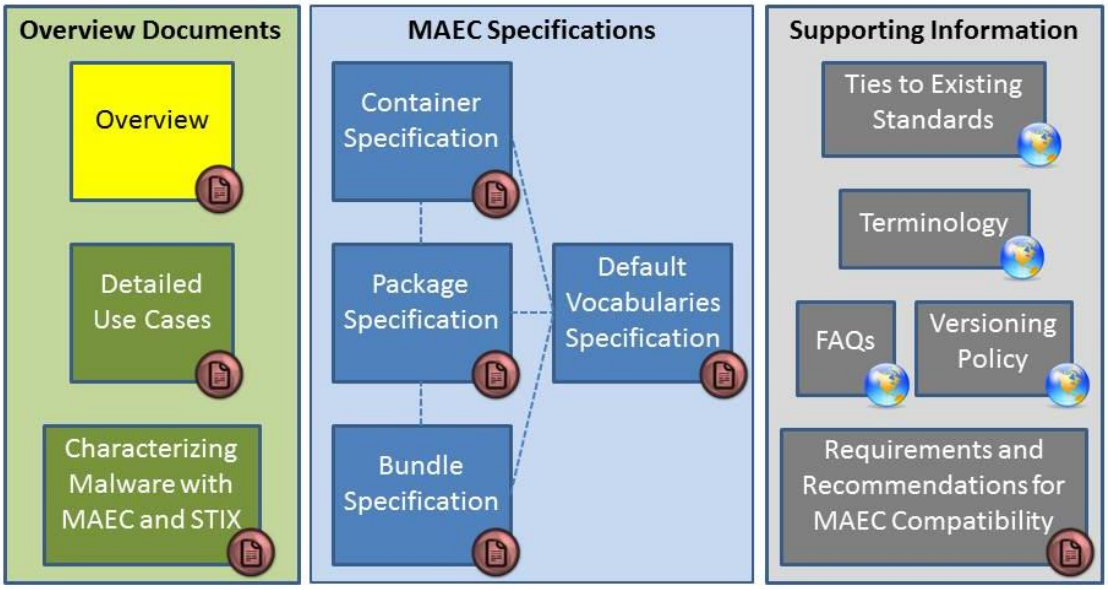

Figure 6-1. MAEC Language v4.1 documents

All documents and other content can be found on the MAEC Website [MAEC]. A summary of and link to each information source is provided below:

- Overview: Introduces and motivates MAEC, provides an overview of the MAEC language, and presents a collection of high level use cases [$MAEC_O$]. (This document.)
- Detailed Use Cases: Provides explicit examples to illustrate how MAEC can be used to capture malware information stemming from various forms of malware analysis [$EXAM_D$].
- Characterizing Malware with MAEC and STIX: Describes the use of MAEC and STIX in the context of malware characterization and malware metadata exchange [$MAEC_S$].
- Container Specification: Specification for the MAEC Container data model [$SPEC_C$].
- Package Specification: Specification for the MAEC Package data model [$SPEC_P$].
- Bundle Specification: Specification for the MAEC Bundle data model [$SPEC_B$]

- Default Vocabulary Specification: Specification for the MAEC Default Vocabularies [$SPEC_V$].
- Ties to Existing Standards: Provides an overview of how MAEC is related to MMDEF, CybOX, CPE, CVE, and STIX.
- Terminology: Contains terms associated with malware and malware analysis, as well as terminology that is specific to MAEC.
- FAQs: Frequently asked questions about MAEC including questions about the language, use, relationships to other efforts, and the MAEC community.
- Versioning Policy: Details the current methodology for determining whether a revision will require a major version change, a minor version change, or an update version change. Note that the MAEC schemas and default vocabularies are versioned independently of the MAEC Language, and their version numbers may or may not coincide with each other or with that of the MAEC Language.
- Requirements and Recommendations for MAEC Compatibility: Specifies requirements for MAEC-compatible tools, services, and repositories.

Appendix A – Terminology

This section contains standard terminology used throughout this document that is associated with malware and malware analysis, as well as terminology that is specific to MAEC.

A.1 General Malware Terminology

- **Artifact**: an entity remaining on a system after the execution of malware that is the result of the behaviors performed by 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 (e.g., files, directories), registry keys, and open network ports.
- **Attribute**: a characteristic of malware that can be used as a descriptor. For MAEC, attributes can include low-level Actions, mid-level Behaviors, the categories of the high-level Capabilities, and metadata.
- **Attack Pattern:** a description of a common method for exploiting a computer system (or "information system"), which can include the attacker's perspective and guidance on ways to mitigate their effect. Example: Exploiting incorrectly configured SSL security levels.
- **AV Classification**: a single classification of a malware instance by an anti-virus (AV) tool.
- **Dynamic Analysis**: analysis of malware that is performed by executing the malware, typically in a sandbox environment.
- **Finding:** A particular fact (attribute) associated with a malware instance that is discovered through analysis. In MAEC, a finding is captured in a MAEC Bundle.
- **Malware Instance**: a specific, single copy of malware; in MAEC, a malware instance refers to the object whose Behaviors, Actions, Objects, Process Tree, and Candidate Indicators are characterized in the MAEC Bundle. Accordingly, in a MAEC Package, the object whose total set of attributes, including analytical findings and metadata, is characterized in the Malware Subject.
- **Malware Pattern:** a collection of attributes common to a set of malware instances (e.g., families (malware that share a significant portion of their code and/or have evolved from each other) or classes (malware that share significant features). A single malware pattern may potentially have many varying malware instances, families, or classes associable with it.
- **Metadata**: a set of data that describes and gives information about other data. Examples: malware sample relationship information; field data associated with a malware sample (e.g., prevalence information, first-seen date).
- **Payload:** the specific malware attributes unrelated to insertion, infection, armoring, obfuscation, and self-defense. Therefore, payload of a malware instance can be thought of as the actions taken after the successful infection of a system, and is

directly tied into the purpose behind the malware instance. This is potentially one of the categories in MAEC's high-level Capabilities.

- **Propagation:** the mechanism and vector used by malware for the purpose of spreading to other machines. Considered a MAEC Capability and potentially a sub-category of payload.
- **Static Analysis**: analysis of malware that is performed without executing the code. Static analysis tools include disassemblers and string extractors.
- **Taxonomy**: the practice of classifying malware or aspects of malware according to type or function. Examples: network reconnaissance, propagation, insertion method.
- **Tool:** A software application or device that analyzes or detects malware through various methods. Examples: a static analysis tool, dynamic analysis tool, signature based scanner, or heuristic based scanner.
- **Type:** any of the anti-virus (AV)/security community's commonly used monikers for groupings of malware that share some common characteristic. Examples: virus, Trojan, worm, backdoor, key logger, rootkit, bot, etc.
- **Vector:** the specific method used to infect a system with malware. Examples: software vulnerability exploitation or social-engineering.

A.2 MAEC-Specific Terminology

- **Action:** a CybOX entity extended by MAEC for capturing a system state change or a similar operation that represents the fundamental low-level operation of malware. Some examples include the creation of a file, deletion of a registry key, and the sending of some data on a socket.
- **Behavior:** a MAEC entity for capturing 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 one or more Actions. Example: the consequence of inserting a registry key (Action) is allowing malware to become resident at system start-up (Behavior).
- **Candidate Indicator**: a MAEC entity that specifies the particular components that may signify the presence of the malware instance on a host system or network. For instance, the existence of a particular Registry Key Object, or observation of a particular '*send http get request*' Action.
- **Capability**: a high-level MAEC entity for capturing a particular goal of the author(s) of a malware instance and used to organize groups of Behaviors. Example: ensuring that malware is executed at start-up is a Behavior that is typically part of a "Persistence" Capability.
- **Collection**: a MAEC entity that serves as a container for abstract grouping of MAEC Bundle level entities, i.e., Behaviors, Actions, Objects, and Candidate Indicators. Used in some of the dynamic analysis tool translators for binning actions that operate on the same type of Object.

- **Document:** some instance of MAEC output, whether it be a MAEC Bundle, MAEC Package, or MAEC Container. A document may characterize one or more malware instances.
- **Grouping Relationship**: a MAEC entity that serves as a relationship for defining the grouping between the Malware Subjects captured in a MAEC Package. Examples: '*same malware family*' and '*part of intrusion set*.'
- **Malware Subject**: a MAEC entity for capturing the total set of attributes pertaining to a single malware instance, including any corresponding analyses, field data, findings via MAEC Bundles, and relationships to other Malware Subjects.
- **Object**: a CybOX entity for capturing the characteristics of a specific cyber-relevant object, including its particular properties and relationships to other Objects. Examples: file, registry key, or process (which are typically captured in MAEC using further defined object models).
- **Process Tree**: a MAEC entity for capturing the the hierarchy of processes created, modified, or affected by the execution of the malware instance. The fields of a MAEC Process Tree include parent process information and spawned or injected process information, along with any initiated Actions.

Appendix B – References

References made in this document are listed below.

B.1 MAEC Documents

[$MAEC_O$] MAEC Overview
http://maec.mitre.org/about/docs/MAEC_Overview.pdf

[$MAEC_S$] Characterizing Malware with MAEC and STIX
http://maec.mitre.org/about/docs/Characterizing_Malware_MAEC_and_STIX_v1.0.pdf

[$SPEC_B$] MAEC Bundle Specification
http://maec.mitre.org/language/version4.1/MAEC_Bundle_Spec_v4_1.pdf

[$SPEC_P$] MAEC Package Specification
http://maec.mitre.org/language/version4.1/MAEC_Package_Spec_v2_1.pdf

[$SPEC_C$] MAEC Container Specification
http://maec.mitre.org/language/version4.1/MAEC_Container_Spec_v2_1.pdf

[$SPEC_V$] MAEC Default Vocabularies Specification
http://maec.mitre.org/language/version4.1/MAEC_Vocabs_Spec_v1_1.pdf

[REQ] Requirements and Recommendations for MAEC Compatibiity
http://maec.mitre.org/compatible/Requirements_for_MAEC_Compatibility_V1.1.pdf

B.2 MAEC Web Pages

[$EXAM_W$] MAEC v4.1 Release Examples
http://maec.mitre.org/language/version4.1/#samples

[$EXAM_G$] MAEC Examples (GitHub repository)
https://github.com/MAECProject/schemas/tree/master/examples

[MAEC] MAEC Web Site
https://maec.mitre.org

[$MAEC_C$] MAEC Community
https://maec.mitre.org/community/index.html

[$MAEC_L$] MAEC Discussion List Signup
http://maec.mitre.org/community/discussionlist.html

[$MAEC_H$] MAEC Handshake (send email to maec@mitre.org for access)
https://handshake.mitre.org/

[REL4] MAEC v4.1 Release
https://maec.mitre.org/language/version4.1/

[TERM] MAEC Terminology
http://maec.mitre.org/about/terminology.html

[TIES] Ties to Existing Standards
http://maec.mitre.org/about/standards.html

[FAQ] MAEC FAQ
http://maec.mitre.org/about/faqs.html

[TOU] MAEC Terms of Use
https://maec.mitre.org/about/termsofuse.html

[VER] Versioning Policy
http://maec.mitre.org/language/versioning_policy.html

B.3 MAEC Schema

[REL_B] MAEC Bundle Model
https://maec.mitre.org/language/version4.1/maec_bundle_schema.xsd

[REL_P] MAEC Package Model
https://maec.mitre.org/language/version4.1/maec_package_schema.xsd

[REL_C] MAEC Container Model
https://maec.mitre.org/language/version4.1/maec_container_schema.xsd

[REL_D] MAEC Default Vocabularies
https://maec.mitre.org/language/version4.1/maec_default_vocabularies.xsd

B.4 MAEC Development

[DEV] MAEC GitHub Repositories
https://github.com/MAECProject/

[DEV_P] MAEC Python Library
https://github.com/MAECProject/python-maec

[DEV_S] MAEC Schema Development
https://github.com/MAECProject/schemas

[DEV_U] MAEC Utilities
https://github.com/MAECProject/utils

B.5 Other References

[CPE] Common Platform Enumeration (CPE)
http://nvd.nist.gov/cpe.cfm (Official CPE Dictionary)
http://csrc.nist.gov/publications/PubsNISTIRs.html (CPE Specifications)

[CUCKOO] Cuckoo Sandbox
http://www.cuckoosandbox.org/

[CVE] Common Vulnerabilities and Exposures (CVE)
http://cve.mitre.org

[CVSS] Common Vulnerability Scoring System
http://www.first.org/cvss

[CYBOX] Cyber Observable eXpression (CybOX)
http://cybox.mitre.org

[IOC] Open Indicators of Compromise (OpenIOC)
http://openioc.org/

[MMDEF] IEEE ICSG's Malware Metadata Exchange Format
http://standards.ieee.org/develop/indconn/icsg/mmdef.html

[OVAL] Open Vulnerability and Assessment Language (OVAL)
http://oval.mitre.org

[RFC2119] RFC 2119 – Key words for use in RFCs to Indicate Requirement Levels
http://www.ietf.org/rfc/rfc2119.txt

[STIX] Structured Threat Information eXpression (STIX)
http://stix.mitre.org