MAEC™ 5.0 Specification

Core Concepts

October 9, 2017

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. As such, 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. It is believed that MAEC can enable these capabilities, thereby transforming malware research and response.

1.1. The MAEC Language

MAEC is a standardized language for sharing structured information about malware. The MAEC data model can be represented as a connected graph of nodes and edges where MAEC top level objects define the nodes and MAEC relationships define the edges. A relationship is a link between MAEC objects that describes how the objects are related.

As shown in Figure 1-1, MAEC defines several top level objects: Behaviors, Malware Actions, Malware Families, Malware Instances, and Collections. Relationships between objects (including STIX cyber observable objects) are depicted by directed edges in the diagram: embedded relationships (those that are specified directly on a top-level object as an object property) are labeled in **black font** (labels correspond to the property names), and direct relationships are labeled using a `blue` background (labels correspond to literal values for the relationship type). See Section 6.1 for more information about direct relationships; embedded relationships for each top-level object are specified in Section 5.

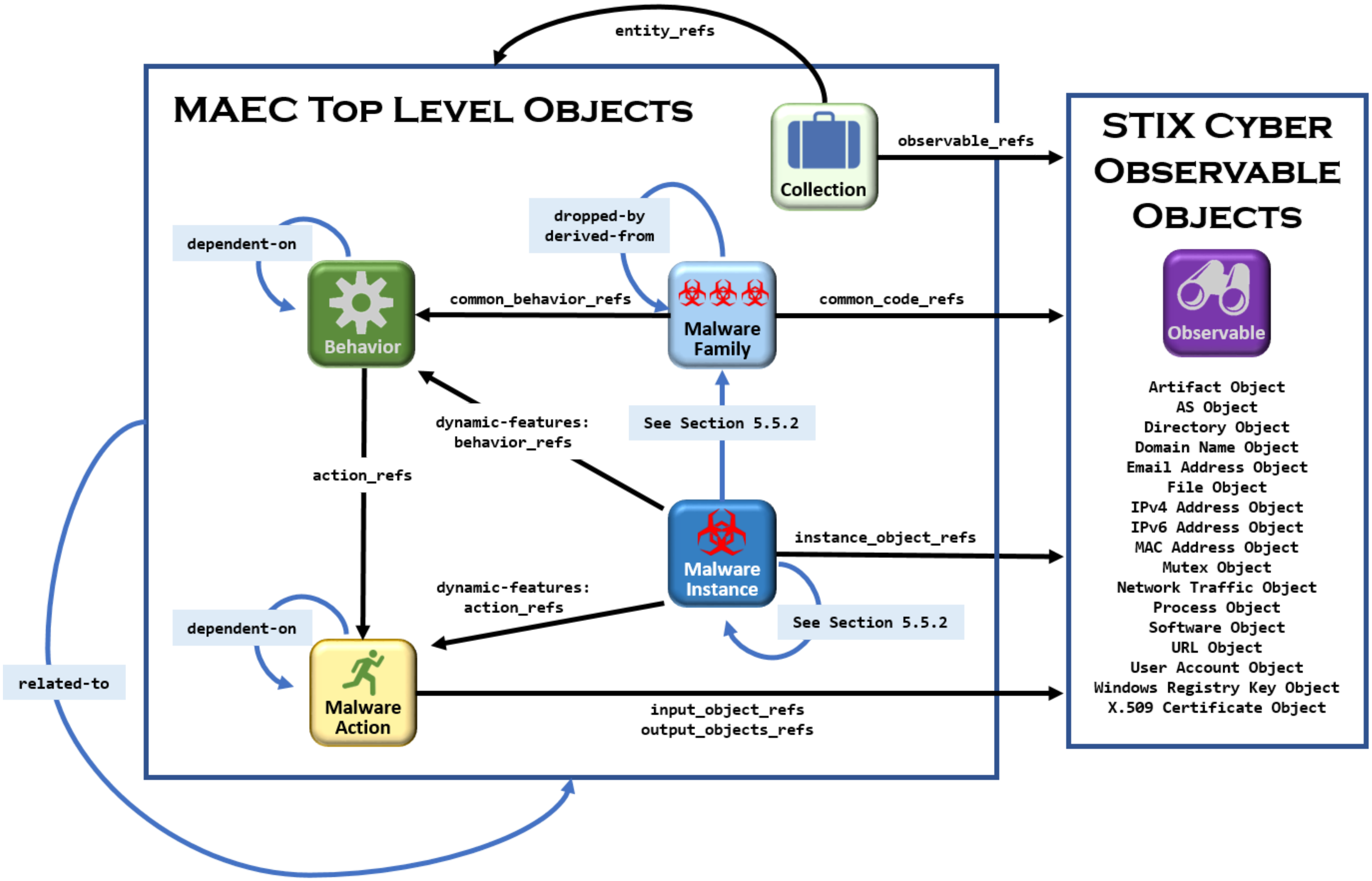

Figure 1-1. MAEC Overview

1.2. MAEC 4 vs. MAEC 5

MAEC 5.0 represents a significant refactoring and simplification of previous MAEC data models. Broadly, the MAEC 5.0 development goals were to simplify and refactor any components that were overly complicated or nested, to deprecate any components that were not used, and to align with the design principles of STIX 2.

The following list summarizes the primary differences between MAEC versions 4 and 5:

- **Architectural Changes**
  - Graph-based data model. The MAEC 4 data model was a combination of graph-like and non-graph like components. With MAEC 5, we have shifted towards a much more graph-based approach with the definition of MAEC top-level objects (i.e., nodes of the graph) and MAEC relationships (i.e., edges of the graph).
  - JSON serialization. All previous versions of MAEC were serialized as XML - with MAEC 5 we moved to a JSON/JSON schema based serialization, which significantly reduces the size and complexity of MAEC documents, while also allowing for better integration with various types of applications.
  - One output format. MAEC 4 defined three different output formats: the MAEC Bundle, Package, and Container. MAEC 5 deprecates the MAEC Bundle and Container in favor of the MAEC Package, which is now the only MAEC output format.
- **New Entities**
  - Malware Family top-level object. MAEC 5 defines a new object for capturing properties associated with Malware Families, including those that are common to all members of the family.
  - Signature metadata type. MAEC 5 defines a new type for capturing metadata about signatures and rules (e.g., YARA rules) that may have triggered during the analysis of a malware instance.
  - Binary obfuscation type. MAEC 5 defines a new type for capturing details of how a malware binary may be obfuscated, such as with a packer or simple XOR encoding, to include layering of obfuscation methods (for example, if a binary is obfuscated with two different methods).
- **Refactored and Simplified Entities**
  - Malware Subject type. Renamed to “Malware Instance” for clarity, this type has been significantly refactored, and now captures all of its analysis results as either embedded entities or direct references to other MAEC top-level objects, without the need for an intermediary type.
  - Behavior type. This type has remained semantically consistent, but now includes an extensive default vocabulary for names and allows for external references to data sources such as ATT&CK [ATT&CK] for specifying the particular technique used in implementing the Behavior.

- AV Classification type. This type has been refactored to be more compatible with the output of VirusTotal, and it can now be specified on any STIX Cyber Observable File Object that is included in a MAEC Package.
- Analysis type. Renamed to “Analysis Metadata” for clarity, this type has been significantly simplified and no longer includes the full textual report specified for the analysis or a direct link to the specific results of the analysis, but it still retains the ability to capture useful general metadata about an analysis that was performed.

1.3. Relationships to Other Languages and Formats

The MAEC Language directly imports and uses components of the OASIS Structured Threat Information Expression (STIX™) language [OASIS], [STIX-1], [STIX-2], [STIX-3], [STIX-4]. An organization performing cyber threat analysis could use the STIX Malware Object; however, MAEC is intended to provide a comprehensive, structured way of capturing detailed information about malware samples, and is therefore targeted primarily towards malware analysts. STIX, meanwhile, is meant to capture a broad spectrum of cyber-threat related information, including basic information on malware, which makes it applicable to a more diverse audience.

1.3.1. STIX Cyber Observables

Malware characterization with MAEC relies on the common implementation (structure and content) that STIX Cyber Observables provide for expressing cyber observables across and among MAEC’s full range of use cases. Thus, whereas MAEC provides coverage of malware analysis context, behaviors, and capabilities, STIX Cyber Observables provide the underpinnings necessary to broadly cover objects, such as files and network connections, used in the malware context.

Cyber Observables are defined by two documents in the STIX specification. *STIX™ Version 2.0. Part 3: Cyber Observable Core Concepts* ([STIX-3]) describes and defines Cyber Observable Core Concepts, which are the parts of STIX that are specific to representation of cyber observables. *STIX™ Version 2.0. Part 4: Cyber Observable Objects* ([STIX-4]) contains a library of Cyber Observable Objects, i.e., definitions for the types of objects that can be observed.

1.3.2. STIX Vocabularies

To avoid duplicating content, MAEC makes use of existing STIX vocabularies wherever applicable. In such cases, a direct reference to the corresponding STIX Vocabulary is provided in the property description.

1.4. Terminology

The keywords “**MUST**”, “**MUST NOT**”, “**REQUIRED**”, “**SHALL**”, “**SHALL NOT**”, “**SHOULD**”, “**SHOULD NOT**”, “**RECOMMENDED**”, “**MAY**”, and “**OPTIONAL**” in this document are to be interpreted as described in [RFC2119].

1.5. Normative References

[STIX-1] STIX™ Version 2.0. Part 1: STIX Core Concepts.
https://docs.google.com/document/d/1IcA5KhglNdyX3tO17bBluC5nqSf70M5qgK9nuAoYJgw

[STIX-2] STIX™ Version 2.0. Part 2: STIX Objects
https://docs.google.com/document/d/1S5XhY6F5OT599b0OuHtUf8IBzFvNY8RysFHIj93DgsY

[STIX-3] STIX™ Version 2.0. Part 3: Cyber Observable Core Concepts
https://docs.google.com/document/d/1PSGv6Uvo3YyrK354cH0cvdn7gGedbhYJkgNVzwW9E6A

[STIX-4] STIX™ Version 2.0. Part 4: Cyber Observable Objects
https://docs.google.com/document/d/1DdS-NrVTjGJ3wvCJ7dbSlhYeiaWS6G6dOXu2F3POpUs

[STIX-Vocab1] STIX™ Version 2.0. Part 3: Cyber Observable Core Concepts, Encryption Algorithm Vocabulary
https://docs.google.com/document/d/1ti4Ei_ii_Uc4izHNZlYmBP9NgD5-iVWC--y-3HmGZyg/edit#heading=h.h5b9uravt8oh

[RFC2119] Bradner, S., “"Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, http://www.rfc-editor.org/info/rfc2119.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014. http://www.rfc-editor.org/info/rfc7159.txt.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, http://www.rfc-editor.org/info/rfc3986.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, http://www.rfc-editor.org/info/rfc4122.

[ISO10646] "ISO/IEC 10646:2014 Information technology -- Universal Coded Character Set (UCS)", 2014. [Online]. Available: http://standards.iso.org/ittf/PubliclyAvailableStandards/c063182_ISO_IEC_10646_2014.zip

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, http://www.rfc-editor.org/info/rfc3339.

1.6. Non-Normative References

[**OASIS**] OASIS Cyber Threat Intelligence (CTI) Technical Committee (TC). https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=cti

[MAEC Vocab] MAEC 5.0 Vocabularies. [Online]. Available: MAEC 5.0 Vocabularies.

[ATT&CK] Adversarial Tactics, Techniques & Common Knowledge. [Online]. Available: ATT&CK.

[MAEC] Malware Attribute Enumeration and Characterization. [Online]. Available: MAEC

[CVSS] Common Vulnerability Scoring System SIG. [Online]. Available: CVSS

1.7. Document Organization

This document begins with an examination of four high-level use cases relevant to MAEC. That is followed with a summary of common data types which is followed immediately by a chapter elucidating specific MAEC data types. Then Cyber Observable object extensions are discussed, followed by MAEC top level objects. The document concludes with a discussion of the standard MAEC output format.

1.7.1. Example Formatting

Note that certain properties are highlighted in **bold** in the examples provided in sections 4-7; these properties are especially pertinent to the example as they are part of the data model component being illustrated by the example.

2. 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 three general areas: malware analysis, cyber threat analysis, and incident management.

2.1. 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 a MAEC Package to communicate the analysis results. As we will also briefly discuss, MAEC Packages 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.

2.1.1. 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 how the instanced was packed, to interesting code snippets obtained from the manual reverse engineering of the instance binary code.

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 type of analysis performed, 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.

2.1.2. Malware Visualization

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

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.

2.1.3. 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.

2.1.4. 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.

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].

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.

2.2. 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.

2.2.1. Malware Threat Scoring System

This linkage between MAEC and other standards efforts (see Section 1.3) 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 persistence, for example.

2.2.2. 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.

2.2.3. Malware Provenance

Understanding and tracking the source and evolution of malware families over time, as well as trying to understand the characteristics of a malware instance that might be useful in identifying its provenance, are important parts of the anti-malware lifecycle. MAEC is useful in both cases. Malware family evolution can be tracked via MAEC's graph-based data model, while the lineage of malware instances can be modeled by leveraging top-level relationships between MAEC entities. With regard to the latter, MAEC defines a standard set of malware properties, such as strings, for both malware instances and families, which can serve as artifacts that are directly associated with provenance.

2.3. 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.

2.3.1. 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 vocabularies 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.

2.3.2. Malware Repositories

As discussed in Section 2.1.3, 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.

2.3.3. 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 Package, or they can verify the remediation performed by another tool by checking for the existence of artifacts captured in a MAEC Package.

3. Common Data Types

This section defines the common data types used throughout MAEC. These types will be referenced by the “Type” column in the tables of other sections. This section defines the names and permitted values of common types that are used in MAEC; however, it does not define the meaning of any properties using these types. These types may be further restricted elsewhere in the document.

Data Type	Description
`boolean`	A value of `true` or `false`.
`dictionary`	A set of key/value pairs.
`external-reference`	A non-MAEC identifier or reference to other related external content.
`identifier`	An identifier (ID) for a MAEC Top Level Object, Relationship Object, or Package.
`list`	A sequence of values ordered based on how they appear in the list. The phrasing “`list` of type `<type>`” is used to indicate that all values within the list **MUST** conform to the specified type.
`hex`	An array of octets (8-bit bytes) as hexadecimal.
`integer`	A number without any fractional or decimal part.
`float`	A double-precision number.
`open-vocab`	A value from a MAEC open vocabulary.
`string`	A series of Unicode characters.

`timestamp`	A time value (date and time).
`stix-observable-objects`	A dictionary of STIX Cyber Observable Objects.
`object-ref`	A reference to a STIX Cyber Observable Object.

3.1. Boolean

Type Name: `boolean`

The `boolean` data type has two possible values: `true` or `false`.

The JSON MTI serialization uses the JSON boolean type [RFC7159], which is a literal (unquoted) `true` or `false`.

Examples

```
{
  ...
  "is_encoded": true,
  ...
}
```

3.2. Dictionary

Type Name: `dictionary`

The `dictionary` data type captures an arbitrary set of key/value pairs.

Dictionary keys:

- **MUST** be unique in each dictionary.
- **MUST** be in ASCII.
- Are limited to the characters a-z (lowercase ASCII), A-Z (uppercase ASCII), numerals 0-9, hyphen (-), and underscore (_).
- **SHOULD** be no longer than 30 ASCII characters in length.
- **MUST** have a minimum length of 3 ASCII characters.
- **MUST** be no longer than 256 ASCII characters in length.

- **SHOULD** be lowercase.

Dictionary values **MUST** be valid common data types.

Examples
```
{
  ...
  "attributes": {
     "file type":"pdf",
     "encryption algorithm":"rc4"
  }
  ...
}
```

3.3. External Reference

Type Name: `external-reference`

The `external-reference` data type describes pointers to information represented outside of MAEC. For example, a Malware Instance object could use an external reference to indicate an ID for that malware in an external database or a report could use references to represent source material.

The JSON MTI serialization uses the JSON object type [RFC7159] when representing `external-reference`.

3.3.1. Properties

Property Name	Type	Description
`source_name` (required)	`string`	The source within which the `external-reference` is defined (system, registry, organization, etc.).
`description` (optional)	`string`	A human readable description.
`url` (optional)	`string`	A URL reference to an external resource [RFC3986].

external_id (optional)	string	An identifier for the external reference content.

3.3.2. Requirements

- In addition to the **source_name** property, at least one of the **external_id**, **url**, or **description** properties **MUST** be present.

Examples

```
{
  ...
  "references": [
     {
        "source_name": "ACME Threat Intel",
        "description": "Threat report",
        "url": "http://www.example.com/threat-report.pdf"
     }
  ]
  ...
}
```

```
{
  ...
  "references": [
     {"url":"https://collaborate.mitre.org/maec/index.php/Behavior:45"},
     {"url":"https://collaborate.mitre.org/maec/index.php/Behavior:45/13"}
  ]
  ...
}
```

3.4. Identifier

Type Name: identifier

The identifier data type universally and uniquely identifies a MAEC Top Level Object, Relationship Object, or Package. Identifiers (IDs) **MUST** follow the form *object-type--UUIDv4*, where *object-type* is the exact value (all type names are lowercase strings, by definition) from the type property of the object being identified or referenced and where the *UUIDv4* is an RFC 4122-compliant Version 4 UUID. The UUID **MUST** be generated according to the algorithm(s) defined in RFC 4122, Section 4.4 (Version 4 UUID) [RFC4122].

The JSON MTI serialization uses the JSON string type [RFC7159] when representing identifier.

Examples

```
{
  ...
  "behaviors": [
    {
      "type": "behavior",
      "id": "behavior--c2f01ec8-42ff-403e-9e76-b4e8a1ffe1b8",
      "name": "persist after system reboot"
    }
  ]
  ...
}
```

3.5. List

Type Name: `list`

The `list` data type defines an ordered sequence of values. The phrasing "`list` of type `<type>`" is used to indicate that all values within the list **MUST** conform to the specific type. For instance, `list` of type `integer` means that all values of the list must be of the `integer` type. This specification does not specify the maximum number of allowed values in a list, however every instance of a list **MUST** have at least one value. Specific MAEC object properties may define more restrictive upper and/or lower bounds for the length of the list.

Empty lists are prohibited in MAEC and **MUST NOT** be used as a substitute for omitting the property if it is optional. If the property is required, the list **MUST** be present and **MUST** have at least one value.

The JSON MTI serialization uses the JSON array type [RFC7159], which is an ordered list of zero or more values.

Examples

```
{
  ...
  "action_refs": [
    "malware-action--c095f1ab-0847-4d89-92ef-010e6ed39c20",
    "malware-action--80f3f63a-d5c9-4599-b9e4-2a2bd7210736",
    "malware-action--5643f634-fff9-4b39-34a4-76fed73d0dd6"
  ],
  ...
}
```

3.6. Hexadecimal

Type Name: `hex`

The `hex` data type encodes an array of octets (8-bit bytes) as hexadecimal. The string **MUST** consist of an even number of hexadecimal characters, which are the digits '0' through '9' and the letters 'a' through 'f'.

Examples
```
{
  ...
  "file_offset":"0400af88"
  ...
}
```

3.7. Integer

Type Name: `integer`

The `integer` data type represents a number without any fractional or decimal part. Unless otherwise specified, all integers **MUST** be capable of being represented as a signed 64-bit value ([-(2**63)+1, (2**63)-1]). Additional restrictions **MAY** be placed on the type as described where it is used.

In the JSON MTI serialization, integers are represented by the JSON number type [RFC7159].

Examples
```
{
  ...
  "count": 8,
  ...
}
```

3.8. Float

Type Name: `float`

The `float` data type represents an IEEE 754 [IEEE 754-2008] double-precision number (e.g., a number with a fractional part). However, because the values ±Infinity and NaN are not representable in JSON, they are not valid values in STIX.

In the JSON MTI serialization, floating point values are represented by the JSON number type [RFC7159].

Examples

```
{
  ...
  "distance": 8.321,
  ...
}
```

3.9. Open Vocabulary

Type Name: `open-vocab`

The `open-vocab` data type is represented as a `string`. For properties that use this type, there will be a list of suggested values to define the property (see [MAEC Vocab]). The value of the property **SHOULD** be chosen from the open vocabulary but **MAY** be any other string value. Values that are not from the open vocabulary **SHOULD** be all lowercase (where lowercase is defined by the locality conventions) and **SHOULD** use hyphens instead of spaces or underscores as word separators.

A consumer that receives MAEC content with one or more `open-vocab` terms not defined in the open vocabulary **MAY** ignore those values.

The JSON MTI serialization uses the JSON string type [RFC7159] when representing `open-vocab`.

Examples

Example using a value from an open vocabulary:

```
{
  ...
  "structural_features": {
```

```
      "name":"code-compression",
      ...
  }
  ...
}
```

Example using a custom value:

```
{
 ...
  "structural_features": {
      "name":"some-odd-code-obfuscation",
      ...
  }
  ...
}
```

3.10. String

Type Name: `string`

The `string` data type represents a finite-length string of valid characters from the Unicode coded character set [ISO10646]. Unicode incorporates ASCII and the characters of many other international character sets.

The JSON MTI serialization uses the JSON string type [RFC7159], which mandates the UTF-8 encoding for supporting Unicode.

Examples

```
{
  ...
  "name":"add-windows-hook",
  ...
}
```

3.11. Timestamp

Type Name: `timestamp`

The `timestamp` data type defines how timestamps are represented in MAEC.

The JSON MTI serialization uses the JSON string type [RFC7159] when representing `timestamp`.

3.11.1. Requirements

- The timestamp property **MUST** be a valid RFC 3339-formatted timestamp [RFC3339] using the format `YYYY-MM-DDTHH:mm:ss[.s+]Z` where the "s+" represents 1 or more sub-second values. The brackets denote that subsecond precision is optional, and that if no digits are provided, the decimal place **MUST NOT** be present.
- The timestamp **MUST** be represented in the UTC timezone and **MUST** use the "Z" designation to indicate this.

Examples

```
{
  ...
  "submission_date": "2016-01-20T12:31:12.12345Z",
  ...
}
```

3.12. Observable Objects

Type Name: `stix-observable-objects`

The `stix-observable-objects` data type is a dictionary (see the `dictionary` data type) where the keys are used as references to the values, which are STIX Observable Objects. Each key in the dictionary **SHOULD** be a non-negative monotonically increasing integer, starting at the value 0 and incrementing by 1, and represented as a string within the JSON MTI serialization. However, implementers **MAY** elect to use an alternate key format.

Examples

The following example illustrates the capture of a STIX Network Traffic Object and an associated IPv4 Address Object.

```
{
 "0": {
   "type": "ipv4-addr",
   "value": "198.51.100.2"
 },
```

```
 "1": {
   "type": "network-traffic",
   "dst_ref": "0"
 }
}
```


3.13. Object Reference

Type Name: `object-ref`

The `object-ref` data type specifies a reference to a STIX Observable Object captured in the MAEC Package **observable_objects** property (`stix-observable-objects`). The reference **MUST** be valid within the scope of the local Package and **MUST** reference a STIX Cyber Observable of one of the following types:

- `artifact`
- `autonomous-system`
- `directory`
- `domain-name`
- `email-addr`
- `email-message`
- `file`
- `ipv4-addr`
- `ipv6-addr`
- `mac-addr`
- `network-traffic`
- `process`
- `software`
- `url`
- `user-account`
- `windows-registry-key`
- `x509-certificate`

Examples
The following example illustrates the referencing of a malware binary (represented as a STIX Cyber Observable File Object) by a Malware Instance.


```
{
  "type":"package",
  "id":"package--7892dac8-c416-35c6-bc5c-7b6dcf576f91",
```

```
  "schema_version":"5.0",
  "maec_objects": [
     {
        "type":"malware-instance",
        "id":"malware-instance--b965814d-0c2e-4e01-b8a5-d8c32bb038e6",
        "instance_object_refs": ["0"]
     }
  ],
  "observable_objects": {
     "0": {
        "type":"file",
        "hashes":{"MD5":"4472ea40dc71e5bb701574ea215a81a1"},
        "size":25536
     }
  }
}
```


4. MAEC Types

These MAEC 5.0 types are used by MAEC's top level objects (TLOs), that is, the entities captured at the top level of a MAEC Package. Types are presented in alphabetical order.

4.1. API Call Type

Type Name: `api-call`

The `api-call` type serves as a method for characterizing API Calls, as implementations of Malware Actions.

4.1.1. Properties

Property Name	Type	Description
address (optional)	`hex`	Captures the hexadecimal address of the API call in the binary.
return_value (optional)	`string`	Captures the return value of the API call.

`parameters` (optional)	`dictionary`	Captures a list of function parameters. Each key in the dictionary **MUST** be a string that captures the exact name of the parameter, and each corresponding key value **MUST** be a string that captures the corresponding parameter value. For parameter values that can be represented by a constant, e.g., GENERIC_WRITE, the constant rather than the literal **SHOULD** be used. For cases where the parameter cannot be represented by a constant, the literal (as reported by the tool producing the data) **MUST** be used.
`function_name` (required)	`string`	Captures the full name of the API function called, e.g., `CreateFileEx`.

Examples

Action with Parameter Constants

```
{
  "type":"package",
  "id":"package--7892dac8-c416-35c6-bc5c-7b6dcf576f91",
  "schema_version":"5.0",
  "maec_objects": [
     {
        "type":"malware-action",
        "id":"malware-action--c095f1ab-0847-4d89-92ef-010e6ed39c20",
        "name":"delete file",
        "output_object_refs": ["3"],
        "api_call": {
           "address":"040089aa",
           "return_value":"0400f258",
           "parameters": {
              "lpFileName":"C:\\Temp\\badfile.pptx",
              "dwDesiredAccess":"GENERIC_WRITE",
              "dwShareMode":"FILE_SHARE_READ",
              "lpSecurityAttributes":"NULL",
              "dwCreationDisposition":"CREATE_NEW",
              "dwFlagsAndAttributes":"FILE_ATTRIBUTE_NORMAL",
              "hTemplateFile":"00000000"
           },
```

```
            "function_name":"CreateFileEx"
         }
      }
   ]
}
```

Action with Parameter Literals


```
{
  "type":"package",
  "id":"package--6e8a76ff-9ffa-419e-8ad4-8a165e86f171",
  "schema_version":"5.0",
  "maec_objects": [
     {
        "type":"malware-action",
        "id":"malware-action--2dc56470-bef0-4a32-910f-760a5d62be2b",
        "name":"delete file",
        "input_object_refs": ["1"],
        "api_call": {
           "address":"040089aa",
           "return_value":"1",
           "parameters": {
              "lpFileName":"C:\\Temp\\badfile.pptx",
              "dwDesiredAccess":"40000000",
              "dwShareMode":"0x00000001",
              "lpSecurityAttributes":"0",
              "dwCreationDisposition":"1",
              "dwFlagsAndAttributes":"128",
              "hTemplateFile":"00000000"
           },
           "function_name":"DeleteFile"
        }
     }
  ]
}
```


4.2. Analysis Metadata Type

Type Name: `analysis-metadata`

The `analysis-metadata` type captures metadata associated with the analyses performed on a malware instance, such as the tools used and the analysts who performed the analysis.

4.2.1. Properties

Property Name	Type	Description
`is_automated` (required)	`boolean`	Captures whether the analysis was fully automated (i.e., no human analyst in the loop). If this property is set to `true`, the **`analysts`** property **MUST NOT** be included.
`start_time` (optional)	`timestamp`	Captures the date/time that the analysis was started.
`end_time` (optional)	`timestamp`	Captures the date/time that the analysis was completed.
`last_update_time` (optional)	`timestamp`	Captures the date/time that the analysis was last updated.
`confidence` (optional)	`integer`	Captures the relative measure of confidence in the accuracy of the analysis results. The confidence value **MUST** be a number in the range of 0-100.
`analysts` (optional)	`list` of type `string`	Captures the names of analysts who performed the analysis.
`analysis_type` (required)	`open-vocab`	Captures the type of analysis performed. The value for this property **SHOULD** come from the `analysis-type-ov` vocabulary.
`comments` (optional)	`list` of type `string`	Captures comments regarding the analysis that was performed. A comment **SHOULD** be attributable to a specific analyst and **SHOULD** reflect particular insights of the author that are significant from an analysis standpoint.
`tool_refs` (optional)	`list` of type `object-ref`	References the tools used in the analysis of the Malware Instance.

		The objects referenced **MUST** be of STIX Cyber Observable type `software` and **MUST** be specified in the **`observable_objects`** property of the Package.
`analysis_environment` (optional)	`dictionary`	Captures any metadata, such as the host virtual machine, associated with the analysis environment used to perform the dynamic analysis of the Malware Instance. Each key in the dictionary **SHOULD** come from the `analysis-environment-ov`, and each corresponding key value **SHOULD** be a valid `object-ref` or `list` of `object-ref`. This property **MUST NOT** be included if **`analysis_type`** is set to a value of `static`.
`description` (optional)	`string`	Captures a textual description of the analysis performed.
`conclusion` (optional)	`open-vocab`	Captures the conclusion of the analysis, such as whether the binary was found to be malicious. The value for this property **SHOULD** come from the `analysis-conclusion-ov` vocabulary.
`references` (optional)	`list` of type `external-reference`	Captures any references to reports or other data sources pertaining to the analysis.

Examples

```
{
  "type":"package",
  "id":"package--7892dac8-c416-35c6-bc5c-7b6dcf576f91",
  "schema_version":"5.0",
  "maec_objects":[
    {
      "type":"malware-instance",
```

```
      "id":"malware-instance--b965814d-0c2e-4e01-b8a5-d8c32bb038e6",
      "instance_object_refs":[
        "0"
      ],
      "name":{
        "value":"MalwareB.1.1",
        "confidence":80
      },
      "analysis_metadata":[
        {
          "is_automated":false,
          "start_time":"2017-02-05T12:15:00Z",
          "end_time":"2017-02-05T12:20:00Z",
          "last_update_time":"2017-02-05T12:20:00Z",
          "confidence":75,
          "analysts":[
            "John Doe",
            "Jane Doe"
          ],
          "analysis_type":"dynamic",
          "analysis_environment":{
            "operating-system":"2",
            "host-vm":"3",
            "installed-software":[
              "4",
              "5"
            ]
          },
          "comments":[
            "The decryption key is: Infected---key+-34512",
            "Analysis required increase of default timeout value"
          ],
          "tool_refs":[
            "1"
          ],
          "description":"Basic automated sandbox analysis.",
          "conclusion":"malicious"
        }
      ]
    }
  ],
  "observable_objects":{
    "0":{
      "type":"file",
```

```
      "hashes":{
        "MD5":"4472ea40dc71e5bb701574ea215a81a1"
      },
      "size":25536
    },
    "1":{
      "type":"software",
      "name":"Cuckoo Sandbox",
      "version":"2.0"
    },
    "2":{
      "type":"software",
      "name":"Windows 7",
      "vendor":"Microsoft"
    },
    "3":{
      "type":"software",
      "name":"Virtualbox",
      "version":"5.0.40",
      "vendor":"Oracle"
    },
    "4":{
      "type":"software",
      "name":"Office 2010",
      "vendor":"Microsoft",
      "version":"14.0.4"
    },
    "5":{
      "type":"software",
      "name":"Java",
      "vendor":"Oracle",
      "version":"1.8.0_40"
    }
  }
}
```


4.3. Binary Obfuscation Type

Type Name: `binary-obfuscation`

The `binary-obfuscation` type captures metadata on the methods that a binary may be obfuscated with, such as executable packers or XOR encryption. This includes obfuscation of the entire binary as well as its constituent pieces, such as strings.

4.3.1. Properties

Property Name	Type	Description
method (required)	open-vocab	Captures the method used to obfuscate the binary. The value for this property **SHOULD** come from the obfuscation-method-ov vocabulary.
layer_order (optional)	integer	Captures the ordering of the obfuscation method with respect to other obfuscation methods (if known), as a positive integer. For example, if a binary was packed and then XOR encrypted, the **layer_order** property of the packing layer would equal 1 and the **layer_order** property of the XOR encryption layer would equal 2.
encryption_algorithm (optional)	open-vocab	Captures the name of the encryption algorithm used by the obfuscation method (if applicable). The values for this property **SHOULD** come from the STIX encryption-algo-ov vocabulary [STIX-Vocab1].
packer_name (optional)	string	Specifies the name of the packer (if applicable).
packer_version (optional)	string	Specifies the version of the packer (if applicable).
packer_entry_point (optional)	hex	Specifies the entry point address of the packer (if applicable).
packer_signature (optional)	string	Specifies the matching signature detected for the packer (if applicable).

Examples


```
{
  "type":"package",
  "id":"package--2d42dac8-c416-42c6-bc5c-7b6dcf576fc5",
  "schema_version":"5.0",
  "maec_objects": [
     {
        "type":"malware-instance",
        "id":"malware-instance--19863c16-503e-493f-8841-16c68e39c26e",
```

```
        "instance_object_refs": ["0"],
        "static_features": {
           "obfuscation_methods": [
              {
                 "method":"packing",
                 "layer_order":1,
                 "packer_name":"UPX"
              },
              {
                 "method":"encryption",
                 "layer_order":2,
                 "encryption_algorithm":"XOR"
              }
           ]
        }
     }
  ]
}
```

4.4. Capability Type

Type Name: `capability`

The `capability` type captures details of a Capability implemented by a malware instance. A Capability corresponds to a high-level ability that a malware instance possesses, such as persistence or anti-behavioral analysis. Malware Instances and Families may share Capabilities; however, the associated Behaviors implementing the Capabilities will often differ. Therefore, Capabilities are defined inline to Malware Instances and Malware Families rather than as top level objects that are subsequently referenced.

4.4.1. Properties

Property Name	Type	Description
name (required)	`open-vocab`	Captures the name of the Capability. The values for this property **SHOULD** come from the `capability-ov` vocabulary. When used as part of a refined Capability, the values for this property **SHOULD** come from the `refined-capability-ov` vocabulary.

refined_capabilities (optional)	list of type capability	Captures a refinement of the Capability, recursively using capability.
description (optional)	string	Captures a textual description of the Capability.
attributes (optional)	dictionary	Captures attributes of the Capability as key/value pairs. Each key in the dictionary **MUST** be a string that captures the name of the attribute and **SHOULD** come from the common-attribute-ov vocabulary. Each corresponding key value **MUST** be a string or list of strings that captures the corresponding attribute values.
behavior_refs (optional)	list of type identifier	Captures the IDs of Behaviors that implement the Capability. Each referenced entity **MUST** be of type behavior and each Behavior **MUST** be present in the current Package.
references (optional)	list of type external-reference	Captures external references to ATT&CK Tactics and other entities that may be associated with the Capability.

Examples

```
{
  "type":"package",
  "id":"package--2d42dac8-c416-42c6-bc5c-7b6dcf576fc5",
  "schema_version":"5.0",
  "maec_objects":[
    {
      "type":"malware-instance",
      "id":"malware-instance--19863c16-503e-493f-8841-16c68e39c26e",
      "instance_object_refs":[
        "0"
      ],
      "labels":[
        "mass-mailer",
        "worm"
      ],
      "capabilities":[
        {
          "name":"persistence",
          "refined_capabilities":[
            {
```

```
              "name":"continuous-execution"
            },
            {
              "name":"system-re-infection"
            }
          ],
          "description":"The instance persists after a system reboot.",
          "attributes":{
            "persistence-scope":[
              "self",
              "other malware/components"
            ],
            "technique":"creates registry key"
          },
          "behavior_refs":[
            "behavior--1",
            "behavior--2"
          ],
          "references":[
            {
              "source_name":"ATT&CK",
              "description":"Persistence",
              "url":"https://attack.mitre.org/wiki/Persistence"
            }
          ]
        }
      ]
    }
  ]
}
```

4.5. Dynamic Features Type

Type Name: `dynamic-features`

The `dynamic-features` type captures the dynamic features (i.e., those associated with the semantics of the executed code, of a malware instance).

4.5.1. Properties

Property Name	Type	Description
`behavior_refs` (optional)	`list` of type `identifier`	Captures the IDs of Behaviors exhibited by the Malware Instance. Each referenced entity **MUST** be of type `behavior`.
`action_refs` (optional)	`list` of type `identifier`	Captures the IDs of Actions discovered for the Malware Instance. Each referenced entity **MUST** be of type `malware-action`. This property is intended for capturing Actions that are discovered through static analysis, reverse engineering, or other methods and therefore **MUST NOT** be used to reference any of the Actions that are included in the **`process_tree`** property. As such, the Actions referenced by this property are mutually exclusive with respect to the Actions referenced by the **`process_tree`** property.
`network_traffic_refs` (optional)	`list` of type `object-ref`	Captures any network traffic recorded for the Malware Instance. The Object(s) referenced **MUST** be of STIX Cyber Observable type `network-traffic` OR `artifact` (for including binaries of captured traffic such as PCAPs) and **MUST** be specified in the **`observable_objects`** property of the Package.
`process_tree` (optional)	`list` of type `process-tree-node`	Captures the Process Tree observed during the execution of the Malware Instance. This property may also capture Actions that are executed by a process and captured by dynamic analysis/sandboxing and therefore **MUST NOT** be used to reference any of the Actions that are included in the **`action_refs`** property. As such, the Actions referenced by this property are mutually exclusive with respect to the Actions referenced by the **`action_refs`** property.

4.5.2. Requirements

- At least one of **behavior_refs** or **action_refs** or **network_traffic_refs** or **process_tree** **MUST** be included when using this type.

Examples

```
{
  "type":"package",
  "id": "package--2d42dac8-c416-42c6-bc5c-7b6dcf576fc5",
  "schema_version":"5.0",
  "maec_objects": [
     {
        "type":"malware-instance",
        "id":"malware-instance--19863c16-503e-493f-8841-16c68e39c26e",
        "instance_object_refs": ["0"],
        "dynamic_features": {
           "behavior_refs": ["behavior--1", "behavior--2"],
           "action_refs": ["malware-action--1", "malware-action--2"],
           "network_traffic_refs": ["4"],
           "process_tree": [
              {
                 "process_ref":"1",
                 "ordinal_position":0
              }
           ]
        }
     },
     {
        "type":"behavior",
        "id":"behavior--1",
        "name":"persist after system reboot",
        "description":"System reboot persistence via registry startup",
        "action_refs": ["malware-action--1"]
     }
  ],
  "observable_objects": {
     "0": {
        "type": "file",
        "hashes": {"MD5": "66e2ea40dc71d5ba701574ea215a81f1"}
     },
     "1": {
        "type": "process",
```

```
        "pid": "1234"
      },
      "2": {
        "type": "process",
        "pid": "2345"
      },
      "3": {
        "type": "domain-name",
        "value": "example.com"
      },
      "4": {
        "type": "network-traffic",
        "dst_ref": "0",
        "protocols": [
          "ipv4",
          "tcp",
          "http"
        ]
      }
  }
}
```

4.6. Field Data Type

Type Name: `field-data`

The `field-data` type captures field data, such as the time that the malware instance or family was first observed, associated with a malware instance or family.

4.6.1. Properties

Property Name	Type	Description
`delivery_vectors` (optional)	`list` of type `open-vocab`	Captures the vectors used to distribute/deploy the Malware Instance. The values for this property **SHOULD** come from the `delivery-vector-ov` vocabulary.
`first_seen` (optional)	`timestamp`	Captures the date/time that the malware instance was first seen by the producer of the Malware Instance Object.

last_seen (optional)	timestamp	Captures the date/time that the malware instance was last seen by producer of the Malware Instance Object.

4.6.2. Requirements

- At least one of **delivery_vectors** or **first_seen** or **last_seen** **MUST** be included when using this type.

Examples


```
{
  "type":"package",
  "id":"package--6864e55f-5f5f-451a-843e-8c66913ae116",
  "schema_version":"5.0",
  "maec_objects": [
     {
        "type":"malware-family",
        "id":"malware-family--8ff5814d-0c2e-5601-b8a5-d0032bb03847",
        "name": {
           "value": "Cryptolocker",
           "confidence": 85
        },
        "field_data": {
           "delivery_vectors":["trojanized-link", "downloader"],
           "first_seen":"2013-09-05T00:00:00Z",
           "last_seen":"2017-01-05T00:00:00Z"
        }
     }
  ]
}
```


4.7. Malware Development Environment Type

Type Name: `malware-development-environment`

The `malware-development-environment` captures details of the development environment used in developing the malware instance, such as information on any tools that were used.

4.7.1. Properties

Property Name	Type	Description
`tool_refs` (optional)	`list` of type `object-ref`	References the tools used in the development of the malware instance. The Objects referenced **MUST** be of STIX Cyber Observable type `software` and **MUST** be specified in the **observable_objects** property of the Package.
`debugging_file_refs` (optional)	`list` of type `object-ref`	References debugging files associated with the malware instance, such as PDB files. The Objects referenced **MUST** be of STIX Cyber Observable type `file` and **MUST** be specified in the **observable_objects** property of the Package.

4.7.2. Requirements

- At least one of **`tool_refs`** or **`debugging_file_refs`** **MUST** be included when using this type.

Examples

```
{
  "type":"package",
  "id":"package--2f5d32d0-2f41-48a1-b272-fa5f0390dbd3",
  "schema_version":"5.0",
  "maec_objects": [
     {
        "type":"malware-instance",
        "id":"malware-instance--90153d4d-092e-1601-b8a5-11312bb0388d",
        "instance_object_refs":["0"],
        "name": {
           "value": "RansomW.cb",
           "confidence": 20
        },
        "static_features": [
           {
              "development_environment": [
                 {
```

```
                    "tool_refs": ["4"]
                 }
              ]
           }
        ]
     }
  ],
  "observable_objects": {
     "0": {
        "type": "file",
        "hashes": {"MD5": "66e2ea40dc71d5ba701574ea215a81f1"}
     },
     "4": {
        "type":"software",
        "name":"gcc"
     }
  }
}
```

4.8. Name Type

Type Name: `name`

The `name` type captures the name of a malware instance, family, or alias, as well as the source and relative confidence in the name.

4.8.1. Properties

Property Name	Type	Description
`value` (required)	`string`	Captures the name of the malware instance, family, or alias.
`source` (optional)	`external-reference`	Captures the internal or external source of the **`value`** property (i.e., the name).
`confidence` (optional)	`integer`	Captures the relative confidence in the accuracy of the assigned name. The confidence value **MUST** be a number in the range of 0-100.

Examples

```
{
```

```
  "type":"package",
  "id":"package--d7b38d7d-f587-4556-a786-0cd2ee10bf5d",
  "schema_version":"5.0",
  "maec_objects": [
     {
        "type":"malware-instance",
        "id":"malware-instance--90153d4d-092e-1601-b8a5-11312bb0388d",
        "name": {
           "value": "Conficker.A",
           "source": {
              "source_name": "Conficker Threat Intel",
              "description": "Analysis details of Conficker by Amanda Analyst",
              "url": "http://www.example.com/threat-report.pdf"
           },
           "confidence": 80
        }
     }
  ]
}
```

4.9. Process Tree Node Type

Type Name: `process-tree-node`

The `process-tree-node` type captures a single node in a process tree, as recorded for a Malware Instance.

4.9.1. Properties

Property Name	Type	Description
`process_ref` (required)	`object-ref`	References the Process Object, contained in the Package, which represents the process and its relevant metadata. The Object referenced **MUST** be of STIX Cyber Observable type `process` and **MUST** be specified in the **`observable_objects`** property of the Package.
`parent_action_ref` (optional)	`identifier`	Captures the ID of the Action that created or injected the process.

		The referenced entity **MUST** be of type `malware-action`.
`ordinal_position` (optional)	`integer`	Captures the ordinal position of the process with respect to the other processes spawned or injected by the malware. This value **MUST** be a non-negative integer. For specifying the root process of the process tree, a value of `0` **MUST** be used.
`initiated_action_refs` (optional)	`list` of type `identifier`	Captures the IDs of the Actions initiated by the process. Each referenced entity **MUST** be of type `malware-action`.

Examples

```
{
  "type":"package",
  "id": "package--2d42dac8-c416-42c6-bc5c-7b6dcf576fc5",
  "schema_version":"5.0",
  "maec_objects": [
     {
        "type":"malware-instance",
        "id":"malware-instance--19863c16-503e-493f-8841-16c68e39c26e",
        "instance_object_refs": ["0"],
        "dynamic_features": {
           "behavior_refs": ["behavior--1", "behavior--2"],
           "process_tree": [
              {
                 "process_ref":"1",
                 "ordinal_position":0
               }
           ]
        }
     }
  ],
  "observable_objects": {
     "0": {
        "type": "file",
        "hashes": {"MD5": "66e2ea40dc71d5ba701574ea215a81f1"}
     },
     "1": {
        "type": "process",
```

```
            "pid": 1234
        },
        "2": {
            "type": "process",
            "pid": 2345
        },
        "3": {
            "type": "process",
            "pid": 5678
        }
    }
}
```

4.10. Relationship Distance Type

Type Name: `relationship-distance`

The `relationship-distance` type captures a distance score and associated metadata between the source and target in a MAEC relationship.

4.10.1. Properties

Property Name	Type	Description
`distance_score` (required)	`float`	Captures the distance score between the source and target in the relationship. This is most commonly represented as a floating point value between zero and one (with a higher value representing a greater distance).
`algorithm_name` (optional)	`string`	Captures the name of the algorithm or tool used in calculating the distance score specified in the **`distance_score`** property.
`algorithm_version` (optional)	`string`	Captures the version of the algorithm or tool used in calculating the distance score specified in the **`distance_score`** property.
`metadata` (optional)	`dictionary`	Specifies a dictionary of additional metadata around the distance score, as a set of key/value pairs. Dictionary keys and their corresponding values **MUST** be of type `string`.

Examples

```
{
  "type":"package",
  "id":"package--0987dac8-2316-52c6-6fbc-074ef8876fdd",
  "schema_version":"5.0",
  "maec_objects": [
     {
        "type":"malware-instance",
        "id":"malware-instance--b965814d-0c2e-4e01-b8a5-d8c32bb038e6",
        "instance_object_refs": ["0"]
     },
     {
        "type":"malware-instance",
        "id":"malware-instance--bacd8340-83bd-94ad-0111-f029304ced90",
        "instance_object_refs": ["1"]
     }
  ],
  "observable_objects": {
     "0": {
        "type":"file",
        "hashes": {"MD5":"4472ea40dc71e5bb701574ea215a81a1"}
     },
     "1": {
        "type":"file",
        "hashes": {"MD5":"39C8E9953FE8EA40FF1C59876E0E2F28"}
     }
  },
  "relationships": [
     {
        "type":"relationship",
        "source_ref":"malware-instance--b965814d-0c2e-4e01-b8a5-d8c32bb038e6",
        "target_ref":"malware-instance--bacd8340-83bd-94ad-0111-f029304ced90",
        "relationship_type":"has-distance",
        "metadata": {
           "distance": {
             "distance_score":0.92,
             "algorithm_name":"clusterAlgorithm-abc",
             "algorithm_version":"6.1",
             "metadata":{"foo":"bar"}
           }
        }
     }
  ]
}
```

4.11. Signature Metadata Type

Type Name: `signature-metadata`

The `signature-metadata` type captures metadata associated with a signature (for example, a YARA rule) that may have been triggered during the analysis of a malware instance.

4.11.1. Properties

Property Name	Type	Description
`signature_type` (required)	`string`	Captures the type of the signature, i.e., the language or platform it is written for. For example, “snort”, for the Snort network intrusion detection system (NIDS). The name of the language or platform **SHOULD** be in lowercase, with any whitespace replaced with dashes (“-”).
`name` (optional)	`string`	Captures the name provided for the signature (if applicable).
`description` (optional)	`string`	Captures a textual description of the signature.
`author` (optional)	`string`	Captures the name of the author of the signature.
`reference` (optional)	`external-reference`	Captures an external reference associated with the signature.
`severity` (optional)	`string`	Captures a measure of severity associated with the detection of the signature.
`external_id` (optional)	`string`	Captures an external identifier associated with the signature.

4.11.2. Requirements

- In addition to **`signature_type`**, at least one of the **`name`** or **`description`** properties **MUST** be included when using this type.

Examples

```
{
  "type":"package",
  "id":"package--2d42dac8-c416-42c6-bc5c-7b6dcf576fc5",
  "schema_version":"5.0",
  "maec_objects": [
     {
        "type":"malware-instance",
        "id":"malware-instance--19863c16-503e-493f-8841-16c68e39c26e",
        "instance_object_refs": ["0"],
        "triggered_signatures": [
           {
              "signature_type":"yara",
              "description":"Ransomware",
              "author":"John Doe",
              "reference":{
                 "url": "http://foo.bar"
              },
              "severity":"9.0"
           },
           {
              "signature_type":"cuckoo",
              "description":"Anti-sandbox sleep",
              "author":"Jane Doe",
              "reference":{
                 "url":"http://bar.foo"
              },
              "severity":"5.0"
           }
        ]
     }
  ]
}
```

4.12. Static Features Type

Type Name: `static-features`

The `static-features` type captures features associated with a malware instance (a binary file) not related to the semantics of the code.

4.12.1. Properties

Property Name	Type	Description
strings (optional)	list of type string	Captures any strings that were extracted from the malware instance.
obfuscation_methods (optional)	list of type binary-obfuscation	Captures metadata associated with methods used to obfuscate the malware instance (e.g., packers, encryptors).
certificates (optional)	list of type object-ref	References any software certificates used to sign the malware instance. The Objects referenced **MUST** be of STIX Cyber Observable type x509-certificate and **MUST** be specified in the **observable_objects** property of the Package.
file_headers (optional)	list of type object-ref	References any file headers (e.g., PE file headers) extracted from the malware instance. The Objects referenced **MUST** be of STIX Cyber Observable type file and **MUST** be specified in the **observable_objects** property of the Package.
configuration_parameters (optional)	dictionary	Captures any configuration parameters specified for the malware instance. Each key in the dictionary **MUST** be of type string and **SHOULD** come from the malware-configuration-parameter-ov vocabulary, which is based on the data reported by the Malware Configuration Parser (MWCP) tool developed by the Department of

		Defense Cyber Crime Center (DC3). Each corresponding key value **MUST** also be of type `string`, and should capture the actual value of the configuration parameter.
`development_environment` (optional)	`malware-development-environment`	Captures details of the development environment used to create the malware instance.

4.12.2. Requirements

- At least one of **`strings`** or **`obfuscation_methods`** or **`certificates`** or **`file_headers`** or **`configuration_parameters`** or **`development_environment`** properties **MUST** be included when using this type.

Examples

```
{
  "type":"package",
  "id":"package--b7be50bd-6348-4226-bef9-4c3510f698f7",
  "schema_version":"5.0",
  "maec_objects": [
     {
        "type":"malware-instance",
        "id":"malware-instance--90153d4d-092e-1601-b8a5-11312bb0388d",
        "name": {
           "value": "Malcode.13",
           "confidence": 50
        },
        "static_features": {
           "strings": ["This string is key.", "This is another string in the instance"],
           "obfuscation_methods": [
              {
                 "method":"packing",
                 "ordering":1,
                 "packer_name":"UPX"
              },
              {
                 "method":"encryption",
                 "ordering":1,
                 "encryption_algorithm":"XOR"
              }
           ],
```

```
            "configuration_parameters": [
               {
                  "name":"magic-number",
                  "value":"0x674dfe60abee3234"
               },
               {
                  "name":"directory",
                  "value":"C:\\Users\\<username>\\Desktop"
               }
            ],
            "development_environment": {
               "tool_refs": ["4"],
               "debugging_file_refs": ["6"]
            }
         }
      }
   ]
}
```

4.13. Cyber Observable Object Extensions

The following are MAEC-specific extensions defined for STIX Cyber Observable Objects that are used in the context of MAEC.

4.13.1. AV Classification Extension

Type Name: `x-maec-avclass`

The `x-maec-avclass` extension captures information on anti-virus (AV) tool classifications for a particular file. Note that unlike other extensions, the base type of this extension is `list`, with each entry in the list (of type `dictionary`) representing a single AV classification. This custom extension **MUST** only be used in conjunction with the STIX Cyber Observable File Object [STIX-4].

4.13.1.1. Properties

Property Name	Type	Description
scan_date (required)	`timestamp`	Captures the date and time of the scan. This property can be used to track how scans change over time.

submission_date (optional)	`timestamp`	Captures the date and time that the binary was submitted for scanning.
is_detected (required)	`boolean`	Captures whether the AV tool specified in the `x-maec-avclass` extension has detected the malware instance.
classification_name (optional)	`string`	Captures the classification assigned to the malware instance by the AV tool.
av_name (optional)	`string`	Captures the name of the AV tool that generated the classification.
av_vendor (optional)	`string`	Captures the name of the vendor of the AV tool that generated the classification.
av_version (optional)	`string`	Captures the version of the AV tool that generated the classification.
av_engine_version (optional)	`string`	Captures the version of the AV engine used by the AV tool that generated the classification.
av_definition_version (optional)	`string`	Captures the version of the AV definitions used by the AV tool that generated the classification.

Examples


```
{
  "type":"package",
  "id":"package--e2ea70f1-02af-4560-8712-34e1d138393e",
  "schema_version":"5.0",
  "observable_objects": {
     "0": {
        "type":"file",
        "name":"a92e5b2bae.exe",
        "hashes": {"MD5":"a92e5b2bae0b4b3a3d81c85610b95cd4"},
        "extensions": {
           "x-maec-avclass": [
              {
                 "scan_date":"2010-05-15T03:38:44Z",
                 "is_detected":false,
                 "av_name":"Security Essentials",
                 "av_vendor":"Microsoft",
```

```
                    "av_engine_version":"4.2.3",
                    "av_definition_version":"032415-0011"
                },
                {
                    "scan_date":"2010-05-18T12:43:12Z",
                    "is_detected":true,
                    "classification_name":"Trojan.Zeus",
                    "av_vendor":"McAfee"
                }
            ]
        }
    }
  }
}
```


5. MAEC Top Level Objects

This section defines the set of MAEC top-level objects (TLOs), i.e., those entities captured at the top level of a MAEC Package (see Section 7). Properties common to all top-level objects are highlighted in their respective property tables in grey.

5.1. Behavior

Type Name: `behavior`

A Behavior corresponds to the specific purpose behind a particular snippet of code, as executed by a malware instance. Examples include keylogging, detecting a virtual machine, and installing a backdoor. Behaviors may be composed of one or more Malware Actions, thereby providing context to these Actions.

5.1.1. Properties

Property Name	Type	Description
type (required)	`string`	The value of this property **MUST** be `behavior`.
id (required)	`identifier`	Specifies a unique ID for the Behavior.

name (required)	open-vocab	Captures the name of the Behavior. The values for this property **SHOULD** come from the behavior-ov open vocabulary.
description (optional)	string	Specifies a textual description of the Behavior.
timestamp (optional)	timestamp	Captures the local or relative time at which the Behavior occurred or was observed.
attributes (optional)	dictionary	Captures attributes of the Behavior as name/value pairs. Dictionary keys used in this property **SHOULD** come from the common-attribute-ov vocabulary. Each corresponding key value **MUST** be of type string and **SHOULD** come from an associated vocabulary, if applicable. For example, if the key is encryption-algorithm, its corresponding value **SHOULD** come from the STIX encryption-algo-ov vocabulary [STIX-Vocab1].
action_refs (optional)	list of type identifier	Captures Actions that serve as an implementation of the Behavior. Each list item specifies the unique ID of the Action being referenced; accordingly, each referenced item **MUST** be of type malware-action. Each Action **MUST** be present in the current Package. The ordering of the references in the list denotes the sequential ordering of the Actions with respect to the Behavior; that is, Actions at the beginning of the list **MUST** have occurred before those later in the list.

technique_refs (optional)	list of type external-reference	References any techniques used to implement the Behavior; for example, DLL Search Order Hijacking. Each reference **SHOULD** point to a valid ATT&CK [ATT&CK] Technique or similar entity.

5.1.2. Relationships

The table shows relationships explicitly defined between the Behavior object and other objects. Relationships are not restricted to those listed below.

Embedded Relationships			
action_refs	malware-action		
Common Relationships			
related-to			
Source	**Relationship Type**	**Target**	**Description**
behavior	dependent-on	behavior	Specifies that the behavior is dependent on the successful execution of another.
behavior	discovered-by	software	Specifies that the behavior was discovered by a particular tool, as a represented by a STIX Cyber Observable Software Object.

Examples

```
{
  "type":"package",
  "id":"package--2d42dac8-c416-42c6-bc5c-7b6dcf576fc5",
  "schema_version":"5.0",
  "maec_objects":[
    {
      "type":"behavior",
      "id":"behavior--2099d4c1-0e8a-49d2-8d32-f0427e1ff817",
      "name":"persist-after-system-reboot",
      "action_refs":[
        "malware-action--c095f1ab-0847-4d89-92ef-010e6ed39c20",
        "malware-action--80f3f63a-d5c9-4599-b9e4-2a2bd7210736"
```

```
      ],
      "attributes":{
        "persistence-scope" : "system wide"
      },
      "technique_refs":[
        {
          "source_name":"att&ck",
          "description":"registry run keys/start folder",
          "external_id":"t1060"
        }
      ]
    },
    {
      "type":"malware-action",
      "id":"malware-action--c095f1ab-0847-4d89-92ef-010e6ed39c20",
      "name":"create file",
      "output_object_refs":[
        "0"
      ]
    },
    {
      "type":"malware-action",
      "id":"malware-action--80f3f63a-d5c9-4599-b9e4-2a2bd7210736",
      "name":"create registry key value",
      "output_object_refs":[
        "1"
      ]
    }
  ],
  "observable_objects":{
    "0":{
      "type":"file",
      "hashes":{
        "MD5":"4472ea40dc71e5bb701574ea215a81a1"
      },
      "size":25536,
      "name":"foo.dll",
      "parent_directory_ref":"2"
    },
    "1":{
      "type":"windows-registry-key",
      "key":"HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\Run",
      "values":[
        {
```

```
          "name":"Foo",
          "value":"C:\\Windows\\System32\foo.dll"
        }
      ]
    },
    "2":{
      "type":"directory",
      "path":"C:\\Windows\\System32"
    }
  }
}
```

5.2. Collection

Type Name: `collection`

A Collection captures a set of MAEC entities (e.g., Malware Instances, Behaviors, etc.) or STIX Cyber Observables that are related or associated in some way.

5.2.1. Properties

Property Name	Type	Description
type (required)	`string`	The value of this property **MUST** be `collection`.
id (required)	`identifier`	Specifies a unique ID for the Collection.
description (optional)	`string`	Specifies a textual description of the Collection.
association_type (required)	`open-vocab`	Specifies how the contents of the Collection are associated. The values for this property **SHOULD** come from the `entity-association-ov` vocabulary.

`entity_refs` (optional)	`list` of type `identifier`	Specifies a set of one or more MAEC entities that are contained in the Collection. Each item specifies the unique ID of the entity being referenced. All entities **MUST** be present in the current Package. This property is mutually exclusive with regard to the **`observable_refs`** property and both properties **MUST NOT** be present in the same Collection.
`observable_refs` (optional)	`list` of type `object-ref`	Specifies a set of one or more STIX Cyber Observable Objects that are contained in the Collection. All Cyber Observable Objects **MUST** be present in the current Package. This property is mutually exclusive with regard to the **`entity_refs`** property and both properties **MUST NOT** be present in the same Collection.

5.2.2. Requirements

- One of **`entity_refs`** or **`observable_refs`** **MUST** be included when using this object.

5.2.3. Relationships

The table shows relationships explicitly defined between the Collection object and other objects. Relationships are not restricted to those listed below.

Embedded Relationships	
entity_refs	`behavior`, `collection`, `malware-action`, `malware-family`, `malware-instance`, `relationship`
observable_refs	`artifact`, `autonomous-system`, `directory`, `domain-name`, `email-addr`, `email-message`, `file`, `ipv4-addr`, `ipv6-addr`, `mac-addr`, `mutex`, `network-traffic`, `process`, `software`, `url`, `user-account`, `windows-registry-key`, `x509-certificate`

Common Relationships
related-to

Examples


```
{
  "type":"package",
  "id":"package--12fbdac8-c416-42c6-cc5c-7b84cf576fc5",
  "schema_version":"5.0",
  "maec_objects": [
     {
        "type":"collection",
        "id":"collection--739df9c1-93ab-49d2-73f0-f0427e1ff817",
        "association_type":"observed together",
        "entity_refs": [
           "malware-instance--4c46cb42-8e83-4bbb-acf8-e09c1311093b",
           "malware-instance--f19859bf-26e4-415e-a1be-41c0486d406d",
           "malware-instance--4a58d70a-9d25-4c80-a114-28036705d026"
        ]
     }
  ]
}
```


5.3. Malware Action

Type Name: `malware-action`

A Malware Action represents an abstraction on a system-level API call (or similar entity) called by the malware instance during its execution, and thereby corresponds to the lowest-level dynamic operation of the malware instance. Actions do not contain any associated context as to why they were performed, as this level of detail and abstraction is documented by Behaviors. Examples of Actions include the creation of a particular file on disk and the opening of a port. Actions are commonly captured and reported by dynamic malware analysis tools (i.e., sandboxes).

5.3.1. Properties

Property Name	Type	Description

type (required)	string	The value of this property **MUST** be malware-action.
id (required)	identifier	Specifies a unique ID for the Malware Action.
name (required)	open-vocab	Captures the name of the Malware Action. The values for this property **SHOULD** come from the malware-action-ov vocabulary.
is_successful (optional)	boolean	Specifies whether the Malware Action was successful in its execution.
description (optional)	string	Captures a basic textual description of the Malware Action.
timestamp (optional)	timestamp	Captures the local or relative time(s) at which the Malware Action occurred or was observed.
input_object_refs (optional)	list of type object-ref	References STIX Observable Objects used as input(s) to the Malware Action. The Object(s) referenced **MUST** be specified in the **observable_objects** property of the Package.
output_object_refs (optional)	list of type object-ref	References STIX Observable Objects resulting as output(s) from the Malware Action. The Object(s) referenced **MUST** be specified in the **observable_objects** property of the Package.
api_call (optional)	api-call	Captures attributes of the specific API call that was used to implement the Malware Action.

5.3.2. Relationships

The table shows relationships explicitly defined between the Malware Action object and other objects. Relationships are not restricted to those listed below.

Embedded Relationships

input_object_refs	artifact, autonomous-system, directory, domain-name, email-addr, email-message, file, ipv4-addr, ipv6-addr, mac-addr, mutex, network-traffic, process, software, url, user-account, windows-registry-key, x509-certificate
output_object_refs	artifact, autonomous-system, directory, domain-name, email-addr, email-message, file, ipv4-addr, ipv6-addr, mac-addr, mutex, network-traffic, process, software, url, user-account, windows-registry-key, x509-certificate

Common Relationships

related-to

Source	Relationship Type	Target	Description
malware-action	dependent-on	malware-action	Specifies that the action is dependent on the successful execution of another.
malware-action	discovered-by	software	Specifies that the action was discovered by a particular tool, as a represented by a STIX Cyber Observable Software Object.

Examples

Basic Create File Action

```
{
  "type":"package",
  "id":"package--7892dac8-c416-35c6-bc5c-7b6dcf576f91",
  "schema_version":"5.0",
  "maec_objects": [
     {
        "type":"malware-action",
        "id":"malware-action--c095f1ab-0847-4d89-92ef-010e6ed39c20",
        "name":"create file",
        "is_successful":true,
        "output_object_refs": ["4"],
        "timestamp": "2016-01-20T12:31:12.12345Z"
     }
  ],
  "observable_objects": {
    "4": {
       "type":"file",
       "hashes":{"MD5":"4472ea40dc71e5bb701574ea215a81a1"},
```

```
      "size":25536,
      "name":"foo.dll"
    }
  }
}
```

Read Registry Key Value Action with Implementation

```
{
  "type":"package",
  "id":"package--0072dac8-c416-35c6-bc5c-7b6dcf576def",
  "schema_version":"5.0",
  "maec_objects": [
     {
        "type":"malware-action",
        "id":"malware-action--e754b078-4185-4eba-a06c-7b2b6c6bd0a5",
        "name":"read registry key value",
        "input_object_refs": ["3"],
        "implementation": {"api_function_name" : "RegQueryValueEx"},
        "timestamp": "2016-01-20T12:31:12.12345Z"
     }
  ],
  "observable_objects": {
     "3": {
        "type":"windows-registry-key",
        "key":"hkey_local_machine\\system\\bar\\foo",
        "values": [
           {
              "name":"Foo",
              "data":"qwerty",
              "data_type":"REG_SZ"
           }
        ]
     }
  }
}
```

Load Library Action

```
{
  "type":"package",
  "id":"package--c10f51a5-8bcb-4f94-bfa2-8a81db056926",
  "schema_version":"5.0",
  "maec_objects":[
    {
```

```
      "id":"action--7fcf121f-e66e-418a-bea1-ce8e8b642da9",
      "type":"malware-action",
      "name":"load library",
      "timestamp":"2017-08-04T12:57:36.143937",
      "input_object_refs":[
        "0"
      ]
    }
  ],
  "observable_objects":{
    "0":{
      "type":"file",
      "name":"advapi32.dll"
    }
  }
}
```

5.4. Malware Family

Type Name: `malware-family`

A Malware Family is a set of malware instances that are related by common authorship and/or lineage. Malware Families are often named and may have components such as strings that are common across all members of the family.

5.4.1. Properties

Property Name	Type	Description
type (required)	`string`	The value of this property **MUST** be `malware-family`.
id (required)	`identifier`	Specifies a unique ID for the Malware Family.
name (required)	`name`	Captures the name of the Malware Family, as specified by the producer of the MAEC Package.

aliases (optional)	list of type name	Captures aliases for the Malware Family. For cases where the alias comes from an external source, the name of the source **SHOULD** be provided.
labels (optional)	list of type open-vocab	Specifies one or more commonly accepted labels to describe the members of the Malware Family, e.g. "worm." The values for this property **SHOULD** come from the malware-label-ov vocabulary.
description (optional)	string	Captures a basic, textual description of the Malware Family.
field_data (optional)	field-data	Specifies field data about the Malware Family, such as first seen and last seen dates, as well as delivery vectors.
common_strings (optional)	list of type string	Specifies any strings common to all members of the Malware Family.
common_capabilities (optional)	list of type capability	Specifies a set of one or more Capabilities that are common to all members of the Malware Family.
common_code_refs (optional)	list of type object-ref	References code snippets that are shared between all of the members of the Malware Family. The Object(s) referenced **MUST** be of STIX Cyber Observable type artifact and **MUST** be specified in the **observable_objects** property of the Package.
common_behavior_refs (optional)	list of type identifier	Specifies a set of one or more Behaviors that are common to all of the members of the Malware Family. Each item specifies the unique ID of the Behavior being referenced; accordingly, each referenced item **MUST** be of type behavior.

references (optional)	list of type external-reference	Captures external references to the Malware Family.

5.4.2. Relationships

These are the relationships explicitly defined between the Malware Family object and other objects. Relationships are not restricted to those listed below.

Embedded Relationships			
common_code_refs	artifact		
common_behavior_refs	behavior		
Common Relationships			
related-to			
Source	**Relationship Type**	**Target**	**Description**
malware-family	dropped-by	malware-family	Indicates that the source malware family is dropped by the target malware family.
malware-family	derived-from	malware-family	Indicates that the code base of the source malware family is a derived from the code base of the target malware family.

Examples

Basic Malware Family

```
{
  "type":"package",
  "id":"package--f53adac8-c416-42c6-6fbc-7b6ef8876fc5",
  "schema_version":"5.0",
  "maec_objects": [
     {
        "type":"malware-family",
        "id":"malware-family--df91014d-0c2e-4e01-b8a5-d8c32bb038e6",
        "name": {
```

```
            "value":"Zeus",
            "confidence":90
         }
      }
   ]
}
```

Expanded Malware Family

```
{
  "type":"package",
  "id":"package--b73adac8-3416-66c6-6fbc-096ef8876fc5",
  "schema_version":"5.0",
  "maec_objects": [
     {
        "type":"malware-family",
        "id":"malware-family--df91014d-0c2e-4e01-b8a5-d8c32bb038e6",
        "name": {
           "value":"Zeus",
           "confidence":90
        },
        "aliases": [
           {
              "value":"ZBot",
              "source":"McAfee",
              "confidence":80
           }
        ],
        "labels": ["bot", "downloader", "trojan"],
        "common_capabilities": [
           {
              "name":"persistence",
              "refined_capabilities":[{"name":"continuous execution"}]
           }
        ],
        "common_behavior_refs": ["behavior--ac15b814-868b-43fd-a89b-91e463293f2b"]
     },
     {
        "type":"malware-instance",
        "id":"malware-instance--b965814d-0c2e-4e01-b8a5-d8c32bb038e6",
        "instance_object_refs": ["0"],
        "name": {
           "value":"Zeus 1.3",
           "confidence":80
        }
```

```
    },
    {
      "type":"behavior",
      "id":"behavior--ac15b814-868b-43fd-a89b-91e463293f2b",
      "name":"persist after system reboot"
    }
  ],
  "observable_objects": {
    "0": {
      "type":"file",
      "hashes": {"MD5":"4472ea40dc71e5bb701574ea215a81a1"},
      "size":25536,
      "name":"foo.dll"
    }
  },
  "relationships": [
    {
      "type":"relationship",
      "id":"relationship--74ae7da8-784d-4a00-aad1-e40c65c78b98",
      "source_ref":"malware-instance--b965814d-0c2e-4e01-b8a5-d8c32bb038e6",
      "target_ref":"malware-family--df91014d-0c2e-4e01-b8a5-d8c32bb038e6",
      "relationship_type":"variant-of"
    }
  ]
}
```


Multiple Related Malware Families


```
{
  "type":"package",
  "schema_version":"5.0",
  "maec_objects": [
    {
      "type":"malware-family",
      "id":"malware-family--07b74c48-d2ef-4e51-943f-ac37274c9a00",
      "name": {
        "value":"Gameover Zeus",
        "confidence":80
      }
    },
    {
      "type":"malware-family",
      "id":"malware-family--e9855981-1e45-4ef1-8989-1272052b0ed5",
```

```
        "name": {
           "value":"Cryptolocker",
           "confidence":80
        }
     }
  ],
  "relationships": [
     {
        "type":"relationship",
        "id":"relationship--151035ce-95bb-4716-a046-7487055f2f7c",
        "source_ref":"malware-family--e9855981-1e45-4ef1-8989-1272052b0ed5",
        "target_ref":"malware-family--07b74c48-d2ef-4e51-943f-ac37274c9a00",
        "relationship_type":"dropped-by"
     }
  ]
}
```

5.5. Malware Instance

Type Name: `malware-instance`

A Malware Instance can be thought of as a single member of a Malware Family that is typically packaged as a binary. This type allows for the characterization of the binaries associated with a Malware Instance along with any corresponding analyses, associated Capabilities, Behaviors, and Actions, and relationships to other Malware Instances.

5.5.1. Properties

Property Name	Type	Description
`type` (required)	`string`	The value of this property **MUST** be `malware-instance`.
`id` (required)	`identifier`	Specifies a unique ID for the Malware Instance.
`instance_object_refs` (required)	`list` of type `object-ref`	References the Cyber Observable Objects that characterize the packaged code (typically a binary) associated with the Malware Instance Object. For most use cases, the object referenced **SHOULD** be of STIX Cyber Observable type `file`. Objects

		referenced **MUST** be specified in the **observable_objects** property of the Package. For cases where multiple STIX Observable File Objects are referenced by this property, each Object **MUST** have the same hash value (via the **hashes** property) but **MAY** have different file names (via the **name** property).
name (optional)	`name`	Captures the name of the Malware Instance, as specified by the producer of the MAEC Package.
aliases (optional)	`list` of type `name`	Captures any aliases for the name of the Malware Instance, as reported by sources other than the producer of the MAEC document (e.g., AV vendors).
labels (optional)	`list` of type `open-vocab`	Specifies commonly accepted labels used to describe the Malware Instance, e.g. "trojan." The values for this property **SHOULD** come from the `malware-label-ov` vocabulary.
description (optional)	`string`	Captures a basic, textual description of the Malware Instance.
field_data (optional)	`field-data`	Specifies field data about the Malware Instance, such as first seen and last seen dates, as well as delivery vectors.
os_execution_envs (optional)	`list` of type `open-vocab`	Specifies the operating systems that the Malware Instance executes on. The values for this property **SHOULD** come from the `operating-system-ov` vocabulary.

architecture_execution_envs (optional)	`list` of type `open-vocab`	Specifies the processor architectures that the Malware Instance executes on. The values for this property **SHOULD** come from the `processor-architecture-ov` vocabulary.
capabilities (optional)	`list` of type `capability`	Specifies a set of one or more Capabilities possessed by the Malware Instance.
os_features (optional)	`list` of type `open-vocab`	Specifies any operating system-specific features used by the Malware Instance. Each item in the list specifies a single feature. The values for this property **SHOULD** come from the `os-features-ov` vocabulary.
dynamic_features (optional)	`dynamic-features`	Captures features associated with the semantics of the code executed by the Malware Instance, such as Malware Actions and Behaviors.
static_features (optional)	`static-features`	Captures features associated with the binary that aren't related to the semantics of the executed code, such as strings and packer information.
analysis_metadata (optional)	`list` of type `analysis-metadata`	Captures metadata associated with the analyses performed on the Malware Instance, such as the tools that were used.
triggered_signatures (optional)	`list` of type `signature-metadata`	Captures metadata associated with any signatures or rules (e.g., YARA) that were triggered during the analysis of the malware instance.

5.5.2. Relationships

The table shows relationships explicitly defined between the Malware Action object and other objects. Relationships are not restricted to those listed below.

Embedded Relationships			
`instance_object_refs`	`file`		
Common Relationships			
`related-to`			
Source	**Relationship Type**	**Target**	**Description**
`malware-instance`	`ancestor-of`	`malware-instance`	Indicates that the source malware instance is an ancestor of the target malware instance.
`malware-instance`	`downloaded-by`	`malware-family`, `malware-instance`	Indicates that the source malware instance is downloaded by the target malware instance or family.
`malware-instance`	`dropped-by`	`malware-family`, `malware-instance`	Indicates that the source malware instance is dropped by the target malware instance or family.
`malware-instance`	`derived-from`	`malware-family`, `malware-instance`	Indicates that the code base of the source malware instance is a derived from the code base of the target malware instance or family.
`malware-instance`	`extracted-from`	`malware-instance`	Indicates that the source malware instance is extracted from the target malware instance.
`malware-instance`	`has-distance`	`malware-instance`	Indicates that the source malware instance has some distance (with respect to similarity) to the target malware instance.
`malware-instance`	`installed-by`	`malware-family`, `malware-instance`	Indicates that the source malware instance is installed by the target malware instance or family.

`malware-instance`	`variant-of`	`malware-family`, `malware-instance`	Indicates that the source malware instance is a variant of the target malware instance or family.

Examples


```
{
  "type":"package",
  "id":"package--773adac8-2316-42c6-6fbc-9cdef8876fc5",
  "schema_version":"5.0",
  "maec_objects": [
     {
        "type":"malware-instance",
        "id":"malware-instance--b965814d-0c2e-4e01-b8a5-d8c32bb038e6",
        "instance_object_refs": ["0"],
        "name": {
           "value":"Zeus 1.3",
           "confidence":50
        },
        "capabilities": [{"name":"anti-detection"}],
        "analysis_metadata": [
           {
              "analysis_type":"in-depth",
              "description": "ran sample through sandbox"
           }
        ]
     }
  ],
  "observable_objects": {
     "0": {
        "type":"file",
        "hashes":{"MD5":"4472ea40dc71e5bb701574ea215a81a1"},
        "size":25536,
        "name":"foo.dll"
     }
  }
}
```


6. MAEC Relationships

MAEC Relationships are used to describe edges between MAEC top-level objects. Property information, relationship information, and examples are provided for the MAEC Relationship Object below.

6.1. Relationship

Type Name: `relationship`

The Relationship Object captures relationships between two entities in a MAEC Package. If MAEC TLOs are considered "nodes" or "vertices" in the graph, the Relationship Object represent "edges". Explicit relationships between MAEC Top Level Objects are provided above in Section 5. Note that MAEC relationships cannot be the source or target of another relationship.

MAEC defines many relationship types to link together some TLOs. These relationships are contained in the "Relationships" table under each TLO definition. Relationship types defined in the specification **SHOULD** be used to ensure consistency. An example of a specification-defined relationship is that a `malware-instance` is `downloaded-by` a `malware-instance`. That relationship type is listed in the Relationships section of the Malware Instance TLO definition.

MAEC also allows relationships from any TLO to any TLO that have not been defined in this specification. These relationships **MAY** use the generic `related-to` relationship type or **MAY** use a custom relationship type. As an example, a user might want to link `malware-instance` directly to a `collection`. They can do so using `related-to` to say that the Malware is related to the Collection but not describe how, or they could use `has-common-artifacts` (a custom name they determined) to indicate more detail.

6.1.1. Common Relationships

Each MAEC top-level object has its own set of relationship types that are specified in the definition of that TLO. The following common relationship types are defined for all TLOs.

Relationship Type	Source	Target	Description
`related-to`	*`<MAEC Top-level Object>`*	*`<MAEC Top-level Object of any type>`*	Asserts a non-specific relationship between two TLOs. This relationship can be used when none of the other predefined relationships are appropriate.

6.1.2. Specification-Defined Relationship Summary

This relationship summary table is provided as a convenience. If there is a discrepancy between this table and the relationships defined with each of the TLOs, then the relationships defined with the TLOs **MUST** be viewed as authoritative.

Source	Type	Target
`behavior`	`dependent-on`	`behavior`
`behavior`	`discovered-by`	`software`
`malware-action`	`dependent-on`	`malware-action`
`malware-action`	`discovered-by`	`software`
`malware-family`	`dropped-by`	`malware-family`
`malware-family`	`derived-from`	`malware-family`
`malware-instance`	`ancestor-of`	`malware-instance`
`malware-instance`	`has-distance`	`malware-instance`
`malware-instance`	`installed-by`	`malware-family`
`malware-instance`	`installed-by`	`malware-instance`
`malware-instance`	`derived-from`	`malware-family`
`malware-instance`	`derived-from`	`malware-instance`
`malware-instance`	`variant-of`	`malware-family`
`malware-instance`	`variant-of`	`malware-instance`
`malware-instance`	`downloaded-by`	`malware-family`

`malware-instance`	`downloaded-by`	`malware-instance`
`malware-instance`	`dropped-by`	`malware-family`
`malware-instance`	`dropped-by`	`malware-instance`
`malware-instance`	`extracted-from`	`malware-instance`

6.1.3. Properties

Property Name	Type	Description
`type` (required)	`string`	The value of this property **MUST** be `relationship`.
`id` (required)	`identifier`	Specifies a unique ID for the Relationship.
`source_ref` (required)	`identifier`	Specifies a reference to the ID of the entity in the MAEC document that corresponds to the source in the source-target relationship. The referenced entity **MUST** be present in the current Package.
`target_ref` (required)	`identifier`	Specifies a reference to the ID of the entity in the MAEC document that corresponds to the target in the source-target relationship. The referenced entity **MUST** be present in the current Package.
`timestamp` (optional)	`timestamp`	Specifies a timestamp that states when the relationship was created.
`relationship_type` (required)	`string`	Specifies the type of relationship being expressed. This value **SHOULD** be an exact value listed in the relationships for the source and target top-level object, but **MAY** be any string. The value of this field **MUST** be in ASCII and is limited to characters a–z (lowercase ASCII), 0–9, and dash (-).
`metadata` (optional)	`dictionary`	Specifies a dictionary of additional metadata around the relationship.

		Standard dictionary keys include the following: ● `distance`: used for capturing any distance related metadata. The corresponding value for this key **MUST** be an object of type `relationship-distance`. Custom entries in the dictionary **MAY** also be included. Each custom entry **MUST** have a key of type `string` and the key **MUST** be in ASCII and is limited to characters a–z (lowercase ASCII), 0–9, and dash (-). Each custom entry **MUST** have a key value that is a valid common datatype, as defined in Section 3.

Examples

Malware Instances (downloaded)

```
{
  "type":"package",
  "id":"package--0987dac8-2316-52c6-6fbc-074ef8876fdd",
  "schema_version":"5.0",
  "maec_objects": [
     {
        "type":"malware-instance",
        "id":"malware-instance--b965814d-0c2e-4e01-b8a5-d8c32bb038e6",
        "instance_object_refs": ["0"]
     },
     {
        "type":"malware-instance",
        "id":"malware-instance--bacd8340-83bd-94ad-0111-f029304ced90",
        "instance_object_refs": ["1"]
     }
  ],
  "observable_objects": {
     "0": {
        "type":"file",
        "hashes": {"MD5":"4472ea40dc71e5bb701574ea215a81a1"}
     },
     "1": {
        "type":"file",
```

```
        "hashes": {"MD5":"39C8E9953FE8EA40FF1C59876E0E2F28"}
      }
  },
  "relationships": [
      {
        "type":"relationship",
        "id":"relationship--dcc7d8d4-91c0-412a-8d09-a030ab19e0f1",
        "source_ref":"malware-instance--b965814d-0c2e-4e01-b8a5-d8c32bb038e6",
        "target_ref":"malware-instance--bacd8340-83bd-94ad-0111-f029304ced90",
        "relationship_type":"downloaded-by"
      }
  ]
}
```


Malware Instances (distance score)


```
{
  "type":"package",
  "id":"package--dbd7a6ae-9dfc-48a2-9e6e-bf85f0c8613b",
  "schema_version":"5.0",
  "maec_objects": [
      {
        "type":"malware-instance",
        "id":"malware-instance--c90945ec-ea66-4c61-9bd4-72e66aeb464e",
        "instance_object_refs": ["0"]
      },
      {
        "type":"malware-instance",
        "id":"malware-instance--ce40a5c7-f3af-4b64-90e2-2884194192ab",
        "instance_object_refs": ["1"]
      }
  ],
  "observable_objects": {
      "0": {
        "type":"file",
        "hashes": {"MD5":"aafdea40dc71e5bb701574ea215a81a1"}
      },
      "1": {
        "type":"file",
        "hashes": {"MD5":"3ABCE9953FE8EA40FF1C59876E0E2F28"}
      }
  },
  "relationships": [
      {
```

```
        "type":"relationship",
        "id":"relationship--0bc99c9c-8765-4a36-b416-bbdde178b5a4",
        "source_ref":"malware-instance--c90945ec-ea66-4c61-9bd4-72e66aeb464e",
        "target_ref":"malware-instance--ce40a5c7-f3af-4b64-90e2-2884194192ab",
        "relationship_type":"has-distance",
        "metadata":{"distance":{"distance_score":"0.35",
                                "algorithm_name":"FooDist"}}
     }
  ]
}
```


7. MAEC Package

Type Name: `package`

The `package` is the standard output format that can be used to capture *one or more* Malware Instances or Malware Families and the entities associated with them: Capabilities, Behaviors, Actions, Cyber Observable Objects, and Collections and Relationships.

7.1. Properties

Property Name	Type	Description
type (required)	`string`	The value of this property **MUST** be `package`.
id (required)	`identifier`	Specifies a unique ID for this Package.
schema_version (required)	`string`	Specifies the version of the MAEC specification used to represent the content in this Package. The value of this property **MUST** be `5.0` for Packages containing MAEC Objects defined in this specification.
maec_objects (required)	`list` of type *<MAEC Object>*	Specifies MAEC Objects. Objects in this list **MUST** be valid MAEC Top-level Objects.
observable_objects (optional)	`stix-observable-objects`	Specifies a dictionary of STIX Cyber Observable Objects relevant to the MAEC Package.

		This dictionary **MUST** contain all Cyber Observable Objects associated with the MAEC Package, including those that are referenced by other Cyber Observable Objects.
`relationships` (optional)	`list` of type `relationship`	Specifies a set of one or more MAEC Relationships. Each entry in this list **MUST** be of type `relationship`.

Examples

```
{
  "type":"package",
  "id":"package--0987dac8-2316-52c6-6fbc-074ef8876fdd",
  "schema_version":"5.0",
  "maec_objects": [
     {
        "type":"malware-instance",
        "id":"malware-instance--b965814d-0c2e-4e01-b8a5-d8c32bb038e6",
        "instance_object_refs": ["0"]
     },
     {
        "type":"malware-instance",
        "id":"malware-instance--bacd8340-83bd-94ad-0111-f029304ced90",
        "instance_object_refs": ["1"]
     }
  ],
  "observable_objects": {
     "0": {
        "type":"file",
        "hashes": {"MD5":"4472ea40dc71e5bb701574ea215a81a1"}
     },
     "1": {
        "type":"file",
        "hashes": {"MD5":"39C8E9953FE8EA40FF1C59876E0E2F28"}
     }
  },
  "relationships": [
     {
        "type":"relationship",
        "source_ref":"malware-instance--b965814d-0c2e-4e01-b8a5-d8c32bb038e6",
```

```
        "target_ref":"malware-instance--bacd8340-83bd-94ad-0111-f029304ced90",
        "relationship_type":"downloaded-by"
      }
  ]
}
```

8. Appendix - MAEC Idioms

Most of the MAEC 4.1 idioms are incorporated in the MAEC 5.0 specification as examples. Those that are not are given below.

8.1. Static Analysis Capture

This Idiom describes the process of capturing the results of static analysis performed on some malware instance, such as through the use of a PE file analysis tool.

In this scenario, a malicious PE binary has been analyzed with the freely available PEiD tool. This tool provides information about the entry point and subsystem defined in the PE headers of the file, as well as the version of the linker used in linking the code.

```
{
  "type":"package",
  "id":"package--a4dadac8-ddf6-67c6-6fbc-8854f8876fc5",
  "schema_version":"5.0",
  "maec_objects":[
    {
      "type":"malware-instance",
      "id":"malware-instance--b965814d-0c2e-4e01-b8a5-d8c32bb038e6",
      "instance_object_refs":["0"],
      "static_features":{
        "file_headers":["1"]
      },
      "analysis_metadata":[
        {
          "is_automated":"false",
          "analysts":["Frankie Li"],
          "analysis_type":"static",
          "description":"A basic static triage of the subject binary using PEiD.",
          "tool_refs":["2"],
          "references":[{"url":"http://www.sans.org/reading_room/whitepapers/malicious/
```

```
                                      detailed-analysis-advanced-persistent-threat-malware_33814"}]
        }
      ]
    }
  ],
  "observable_objects":{
    "0":{
      "type":"file",
      "hashes":{"MD5":"4EC0027BEF4D7E1786A04D021FA8A67F"},
      "size":196608,
      "name":"dg003_improve_8080_V132.exe",
      "mime_type":"vnd.microsoft.portable-executable"
    },
    "1":{
      "type":"file",
      "hashes":{"MD5":"4EC0027BEF4D7E1786A04D021FA8A67F"},
      "mime_type":"vnd.microsoft.portable-executable",
      "extensions":{
        "windows-pebinary-ext":{
          "pe_type":"exe",
          "optional_header":{
            "major_linker_version":6,
            "minor_linker_version":0,
            "address_of_entry_point":36418,
            "subsystem_hex":"02"
          }
        }
      }
    },
    "2":{
      "type":"software",
      "name":"PEiD",
      "version":"0.94"
    }
  }
}
```

8.2. Dynamic Analysis Capture

This Idiom describes the process of capturing the results of dynamic analysis performed on some malware instance, such as through the use of a malware sandbox tool.

In this scenario, a malicious PE binary has been analyzed using the freely available ThreatExpert sandbox service, which provides information about the low-level actions that the PE binary performs when executed. For the sake of brevity, the example below focuses on two actions reported by the sandbox: the creation of a file, and the creation of a mutex.


```
{
  "type":"package",
  "id":"package--24afdac1-1536-d7c6-ffbc-ddb4f8876fc5",
  "schema_version":"5.0",
  "maec_objects":[
    {
      "type":"malware-instance",
      "id":"malware-instance--b965814d-0c2e-4e01-b8a5-d8c32bb038e6",
      "instance_object_refs":["0"],
      "dynamic_features":{
        "action_refs":[
          "malware-action--e6ecdda7-6a70-4320-8e54-5c956c778b7b",
          "malware-action--e5e6fd60-77ea-4489-a801-f2b56bfccb22"
        ]
      },
      "analysis_metadata":[
        {
          "is_automated":"true",
          "analysis_type":"dynamic",
          "tool_refs":["1"]
        }
      ]
    },
    {
      "type":"malware-action",
      "id":"malware-action--e6ecdda7-6a70-4320-8e54-5c956c778b7b",
      "name":"create file",
      "output_object_refs":["2"]
    },
    {
      "type":"malware-action",
      "id":"malware-action--e5e6fd60-77ea-4489-a801-f2b56bfccb22",
      "name":"create mutex",
      "output_object_refs":["3"]
    }
  ],
  "observable_objects":{
    "0":{
      "type":"file",
```

```
      "hashes":{
        "MD5":"B6C39FF68346DCC8B67AA060DEFE40C2",
        "SHA1":"D55B0FB96FAD96D203D10850469489FC03E6F2F7"
      },
      "size":210564,
      "mime_type":"vnd.microsoft.portable-executable"
    },
    "1":{
      "type":"software",
      "name":"ThreatExpert",
      "vendor":"ThreatExpert"
    },
    "2":{
      "type":"file",
      "name":"Zcxaxz.exe",
      "size":332288,
      "mime_type":"vnd.microsoft.portable-executable"
    },
    "3":{
      "type":"mutex",
      "name":"redem-Mutex"
    }
  }
}
```

8.3. In-depth Analysis Capture

This Idiom describes the process of capturing results of in-depth malware analysis, such as that which characterizes the capabilities or behaviors exhibited by the malware.

In this scenario, a malicious PE binary has been manually analyzed using a disassembler tool. As part of this analysis, it was discovered that the malware instance contained a keylogging capability, as well as a Windows-hook based behavior that implements the capability. Because there is no Windows hook object defined in STIX 2.0 Cyber Observables, it is necessary to define a custom object ("x-windows-hook").

```
{
  "type":"package",
  "id":"package--65cddac1-1536-11c6-ffbc-df24f8876fc5",
  "schema_version":"5.0",
  "maec_objects":[
    {
      "type":"malware-instance",
```

```
      "id":"malware-instance--b965814d-0c2e-4e01-b8a5-d8c32bb038e6",
      "instance_object_refs":["0"],
      "capabilities":{
        "name":"collection",
        "refined_capabilities":[
          {
            "name":"input-peripheral-capture",
            "behavior_refs":["behavior--cfb4d731-c6e2-4c8e-808d-111e1ba66962"]
          }
        ]
      },
      "analysis_metadata":[
        {
          "is_automated":"false",
          "analysis_type":"static"
        }
      ]
    },
    {
      "type":"behavior",
      "id":"behavior--cfb4d731-c6e2-4c8e-808d-111e1ba66962",
      "name":"capture-keyboard-input",
      "action_refs":["malware-action--e6ecdda7-6a70-4320-8e54-5c956c778b7b"]
    },
    {
      "type":"malware-action",
      "id":"malware-action--a48e58bb-f35d-4bf6-bb16-0e74061ac47e",
      "name":"add-windows-hook",
      "output_object_refs":["1"]
    }
  ],
  "observable_objects":{
    "0":{
      "type":"file",
      "hashes":{"MD5":"B6C39FF68346DCC8B67AA060DEFE40C2"},
      "size":210564,
      "mime_type":"vnd.microsoft.portable-executable"
    },
    "1":{
      "type":"x-windows-hook",
      "hook_type":"WH_KEYBOARD_LL"
    }
  }
}
```