Common Attack Pattern Enumeration and Classification — CAPEC™

A Community Knowledge Resource for Building Secure Software

CAPEC is a publicly available catalog of attack patterns along with a comprehensive schema and classification taxonomy created to assist in the building of secure software. By structuring the definition of attack patterns as part of a broader integrated software assurance knowledge architecture that includes similar knowledge-structuring efforts such as Common Weakness Enumeration (CWE™), Common Vulnerabilities and Exposures (CVE®), Malware Attribute Enumeration and Characterization (MAEC™), and Cyber Observable eXpression (CybOX™), CAPEC supports the needs of developers, testers, and educators to build secure software and assists in enhancing security throughout the software development lifecycle.

Attack patterns examples:

- HTTP response splitting
- SQL injection
- XSS in HTTP query strings
- Session fixation
- Phishing
- Filter failure through buffer overflow
- Removing or short-circuiting guard logic
- Lifting data embedded in client distributions
- Subvert code-signing facilities
- Reflection attack in an authentication protocol
- Cause web server misclassification
- Rainbow table password cracking
- Forced deadlock
- Cache poisoning
- Restful privilege escalation

Challenge

To build secure software, builders must ensure that they have protected every relevant potential vulnerability. Yet, to attack software, attackers often have to find and exploit only a single exposed vulnerability. To identify and mitigate relevant vulnerabilities in software, the development community needs more than just good software engineering and analytical practices, a solid grasp of software security features, and a powerful set of tools. All of these things are necessary but not sufficient.

To respond effectively, the community needs to think outside of the box and have a firm grasp of the attacker's perspective and the approaches used to exploit software systems. CAPEC provides this information to the community in order to help enhance security throughout the software development lifecycle and to support the needs of developers, testers, and educators.

Solution

"Attack Patterns" are descriptions of common methods for exploiting software providing the attacker's perspective and guidance on ways to mitigate their effect. They derive from the concept of design patterns applied in a destructive rather than constructive context and are generated from in-depth analysis of specific real-world exploit examples.

This information when captured in such a formalized way can bring considerable value for software security considerations through all phases of the software development lifecycle and other security-related activities including identification of relevant security requirements, misuse, and abuse; context for architectural risk analysis and guidance for security architecture; context for appropriate risk-based and penetration testing; lessons learned from security incidents into preventative guidance; and identification of appropriate prescriptive organizational policies and standards. While many other tools (e.g., misuse/abuse cases, security requirements, threat models, knowledge of common weaknesses and vulnerabilities, coding rules, and attack trees) are also useful for building secure software, attack patterns play a unique role amid this larger architecture of software security knowledge and techniques.

Attack Patterns Are:

- Powerful mechanisms to capture and communicate the attacker's perspective.
- Descriptions of common methods for exploiting software.
- Derived from the concept of design patterns applied in a destructive rather than constructive context and are generated from in-depth analysis of specific real-world exploit examples.

Initially released in 2007, the CAPEC List continues to evolve with public participation and contributions to form a structured mechanism for identifying, collecting, refining, and sharing attack patterns among the cyber security community.

Participation Requested

Developers, testers, educators, and other community members are invited to participate in this growing community effort by submitting new attack patterns for inclusion in the CAPEC List and joining our CAPEC Community Email Discussion List on the CAPEC Web site.

CAPEC™
capec.mitre.org

The MITRE Corporation maintains CAPEC and its public Web site and provides impartial technical guidance to the CAPEC Community throughout the process to ensure CAPEC serves the public interest.

202 Burlington Road, Bedford, MA 01730-1420
www.mitre.org
MITRE

Example Attack Pattern

Name	HTTP Response Splitting
Typical Severity	High
Description	HTTP Response Splitting causes a vulnerable web server to respond to a maliciously crafted request by sending an HTTP response stream such that it gets interpreted as two separate responses instead of a single one. This is possible when user-controlled input is used unvalidated as part of the response headers. An attacker can have the victim interpret the injected header as being a response to a second dummy request, thereby causing the crafted contents to be displayed and possibly cached. To achieve HTTP Response Splitting on a vulnerable web server, the attacker: 1. Identifies the user-controllable input that causes arbitrary HTTP header injection. 2. Crafts a malicious input consisting of data to terminate the original response and start a second response with headers controlled by the attacker. 3. Causes the victim to send two requests to the server. The first request consists of maliciously crafted input to be used as part of HTTP response headers and the second is a dummy request so that the victim interprets the split response as belonging to the second request.
Attack Prerequisites	User-controlled input used as part of HTTP header Ability of attacker to inject custom strings in HTTP header Insufficient input validation in application to check for input sanity before using it as part of response header
Typical Likelihood of Exploit	Medium
Methods of Attack	Injection Protocol Manipulation
Examples-Instances	In the PHP 5 session extension mechanism, a user-supplied session ID is sent back to the user within the Set-Cookie HTTP header. Since the contents of the user-supplied session ID are not validated, it is possible to inject arbitrary HTTP headers into the response body. This immediately enables HTTP Response Splitting by simply terminating the HTTP response header from within the session ID used in the Set-Cookie directive. CVE-2006-0207
Attacker Skill or Knowledge Required	High - The attacker needs to have a solid understanding of the HTTP protocol and HTTP headers and must be able to craft and inject requests to elicit the split responses.
Resources Required	None
Probing Techniques	With available source code, the attacker can see whether user input is validated or not before being used as part of output. This can also be achieved with static code analysis tools If source code is not available, the attacker can try injecting a CR-LF sequence (usually encoded as %0d%0a in the input) and use a proxy such as Paros to observe the response. If the resulting injection causes an invalid request, the web server may also indicate the protocol error.
Indicators-Warnings of Attack	The only indicators are multiple responses to a single request in the web logs. However, this is difficult to notice in the absence of an application filter proxy or a log analyzer. There are no indicators for the client
Solutions and Mitigations	To avoid HTTP Response Splitting, the application must not rely on user-controllable input to form part of its output response stream. Specifically, response splitting occurs due to injection of CR-LF sequences and additional headers. All data arriving from the user and being used as part of HTTP response headers must be subjected to strict validation that performs simple character-based as well as semantic filtering to strip it of malicious character sequences and headers.
Attack Motivation-Consequences	Execute unauthorized code or commands Gain privileges/assume identity
Context Description	HTTP Response Splitting attacks take place where the server script embeds user-controllable data in HTTP response headers. This typically happens when the script embeds such data in the redirection URL of a redirection response (HTTP status code 3xx), or when the script embeds such data in a cookie value or name when the response sets a cookie. In the first case, the redirection URL is part of the Location HTTP response header, and in the cookie setting, the cookie name/value pair is part of the Set- Cookie HTTP response header.
Injection Vector	User-controllable input that forms part of output HTTP response headers
Payload	Encoded HTTP header and data separated by appropriate CR-LF sequences. The injected data must consist of legitimate and well-formed HTTP headers as well as required script to be included as HTML body.
Activation Zone	API calls in the application that set output response headers.
Payload Activation Impact	The impact of payload activation is that two distinct HTTP responses are issued to the target, which interprets the first as response to a supposedly valid request and the second, which causes the actual attack, to be a response to a second dummy request issued by the attacker.
CIA Impact	Confidentiality Impact: High Integrity Impact: High Availability Impact: Low
Related Weaknesses	CWE113 - HTTP Response Splitting - Targeted CWE74 - Injection - Secondary CWE697 - Insufficient Comparison - Targeted CWE707 - Improper Enforcement of Message or Data Structure - Targeted CWE713 - OWASP Top Ten 2007 Category A2 - Injection Flaws - Targeted
Relevant Security Requirements	All client-supplied input must be validated through filtering and all output must be properly escaped.
Related Security Principles	Reluctance to Trust
Related Guidelines	Never trust user-supplied input.
References	G. Hoglund and G. McGraw. Exploiting Software: How to Break Code. Addison-Wesley, February 2004.
For enhanced descriptions of this example CAPEC-ID, see http://capec.mitre.org/data/definitions/34.html.	

Trusted Automated eXchange of Indicator Information — TAXII™
Enabling Cyber Threat Information Exchange

TAXII defines a set of services and message exchanges that, when implemented, enable sharing of actionable cyber threat information across organization and product/service boundaries. TAXII, through its member specifications, defines concepts, protocols, and message exchanges to exchange cyber threat information for the detection, prevention, and mitigation of cyber threats. TAXII is not a specific information sharing initiative or application and does not attempt to define trust agreements, governance, or other non-technical aspects of cyber threat information sharing. Instead, TAXII empowers organizations to achieve improved situational awareness about emerging threats, and enables organizations to easily share the information they choose with the partners they choose.

TAXII use cases include:

- **Public Alerts or Warnings**
- **Private Alerts and Reports**
- **Push and Pull Content Dissemination**
- **Set-up and Management of Data Sharing Between Parties**

TAXII and STIX

TAXII is the preferred method of exchanging information represented using the Structured Threat Information Expression (STIX™) language, enabling organizations to share structured cyber threat information in a secure and automated manner.

Challenge

The gathering and use of detailed cyber intelligence is the best defense against today's determined cyber adversaries. "Cyber intelligence" — or the collecting, analyzing, and countering of cyber security threat information — starts with gathering information about attacks, such as spear-phishing email header and content, urls to malicious links, and malware analysis-derived artifacts like Command and Control (C2) domain names and IP addresses. With a corpus of threat data, skilled cyber analysts can group patterns of similar activity, attribute activity to certain threat actors, quickly identify and implement mitigation strategies, and anticipate the launch of similar attacks in the future.

To fully realize the benefits of cyber intelligence, organizations need to share cyber threat data, if not defensive strategies and more, with trusted partners. Current cyber threat information sharing, however, is often either a time-consuming, manual process or a limited-scope automation effort tied to particular cyber threat information sharing community or technology.

Solution

TAXII fills this void. The TAXII services and message exchanges are designed to enhance interoperability of different cyber security solutions and vendors are encouraged to incorporate support for TAXII within their cyber security products and services. By supporting TAXII, vendors enhance the value of their solutions by allowing their customers to leverage actionable intelligence from multiple sources.

TAXII's goal is to help add automation to the processes of existing cyber threat information sharing communities and to help establish new communities of sharing by simplifying the technical aspects of cyber threat information exchange. It is recognized that sharing communities are highly diverse and cannot be reduced to a single sharing model. For this reason, TAXII uses a modular design that can accommodate a wide array of sharing models. Individual services in TAXII are optional for any given implementation, allowing enterprises to include only the services for their particular sharing model.

TAXII™

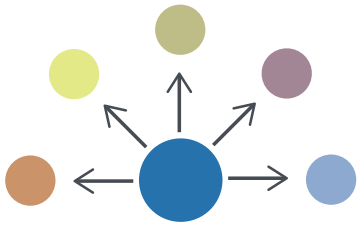
taxii.mitre.org

TAXII is a U.S. Department of Homeland Security–led effort of the office of Cybersecurity and Communications. MITRE, operating as DHS's FFRDC, manages the TAXII website, community engagement, and discussion lists to enable open and public collaboration with all stakeholders.

Sharing models supported by TAXII include (but are not limited to):

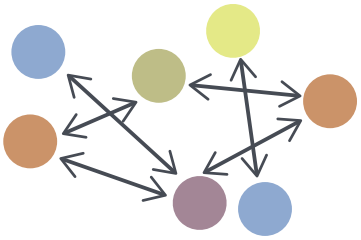
Source-Subscriber

A single entity publishes information out to a group of consumers. This is a common model in commercial environments, where the data source is a vendor and the subscribers purchase access to the vendor's information. This is also a common model for free alerts from some authoritative source.



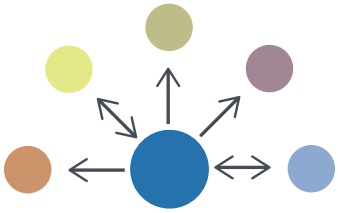
Peer-to-Peer

A group of data producers and data consumers establish direct relationships with each other. The group may have a single governing policy, but all sharing exchanges are between individuals.



Hub-and-Spoke

A group of data producers and consumers share information with each other, but instead of sending directly, the information is sent to a central hub, which then handles dissemination to all the other spokes as appropriate. This model can be viewed as being similar to e-mail distribution lists, where a sender provides a message to a mailing-list service, which then forwards the message on to all the members of the list.



Push or Pull Sharing

TAXII supports both push and pull messaging in all models, allowing sharing scenarios where data consumers are automatically provided with new data, or where the consumer can request updates at times of their choosing. Data producers in a TAXII architecture can choose whether data consumers can pull data from the producer, whether data is pushed from the producer, or whether a mixture of the two methods is supported.

Lightweight, Non-Disruptive Design

Existing sharing communities often have established an infrastructure for storing and managing threat information. TAXII is designed to enable the exchange of this information without impacting existing data management infrastructure. TAXII defines network-level messages and services, but does not impose significant requirements on behavior below the network layer. As such, TAXII is intended to be layered on top of existing data management schemes with minimal disruption. For similar reasons, enterprises without existing infrastructure are free to use their own favored data management schemes, confident that such schemes can integrate with TAXII services and messages.

Cyber threat information is frequently sensitive and organizations may be highly selective as to what information is shared with specific parties. The information that factors in to such decisions can vary from organization to organization. Rather than attempting to standardize such behavior, TAXII focuses on ensuring secure transport of the information over the wire and leaves decisions as to what is shared with whom to the back-end infrastructure of the enterprise. TAXII imposes no requirements or limits on sharing decisions and allows organizations to decide what information is visible to individual requesters using their native decision processes.

TAXII leverages existing protocols and specifications wherever possible. The TAXII core services are designed in a fashion that is neutral with regard to network protocols and data formats. TAXII defines bindings to specific network protocols and data formats separately from the core services. Implementers can select the bindings they wish to use or even define their own. The fact that all bindings share the same understanding of the TAXII services and messages a party that can only support a very constrained set of protocols or formats can still make use of the services and messages of TAXII, and thus would have a window for receiving threat information from a significantly larger set of sources.

Feedback Requested

TAXII Community members can make contributions to TAXII development and manage issue tracking for the TAXII specifications, schemas, and supporting information by joining the TAXII Community at https://taxii.mitre.org/community/. Members of the cyber security community are invited to participate in this growing community effort.

Structured Threat Information eXpression — STIX™

A Structured Language for Cyber Threat Intelligence Information

STIX is a collaborative, community-driven effort to define and develop a structured language to represent cyber threat information. The STIX Language conveys the full range of potential cyber threat information and strives to be fully expressive, flexible, extensible, automatable, and as human-readable as possible. All interested parties are welcome to participate in evolving STIX as part of its open, collaborative community.

STIX use cases include:

- **Analyzing Cyber Threats**
- **Specifying Indicator Patterns for Cyber Threats**
- **Managing Cyber Threat Prevention and Response Activities**
- **Sharing Cyber Threat Information**

STIX and TAXII

Trusted Automated eXchange of Indicator Information (TAXII™) is the preferred method of exchanging information represented using the STIX Language, enabling organizations to share structured cyber threat information in a secure and automated manner.

Challenge

Organizations today must maintain a "cyber threat intelligence" capability as a key part of their defense against determined cyber adversaries. Examples of cyber intelligence include understanding and characterizing information such as what sort of attack actions have occurred and are likely to occur; how can these actions be detected and recognized; how can they be mitigated; who are the relevant threat actors; what are they trying to achieve; what are their capabilities, in the form of tactics, techniques, and procedures (TTP) they have leveraged over time and are likely to leverage in the future; what sort of vulnerabilities, misconfigurations, or weaknesses they are likely to target; what actions have they taken in the past; etc.

A key component of success for this capability is information sharing with partners, peers, and others they select to trust. But while cyber threat intelligence and information sharing can help focus and prioritize the use of the immense volumes of complex cyber security information organizations face today, they have a foundational need for common, structured representations of this information to make it tractable.

Solution

STIX is a community-driven solution to this, providing structured representations of cyber threat information that is expressive, flexible, extensible, automatable, and readable. STIX enables the sharing of comprehensive, rich, "high-fidelity" cyber threat information across organizational, community, and product/service boundaries. STIX extends simple indicator sharing to enable the management and exchange of significantly more expressive sets of indicators as well as other full-spectrum cyber threat information.

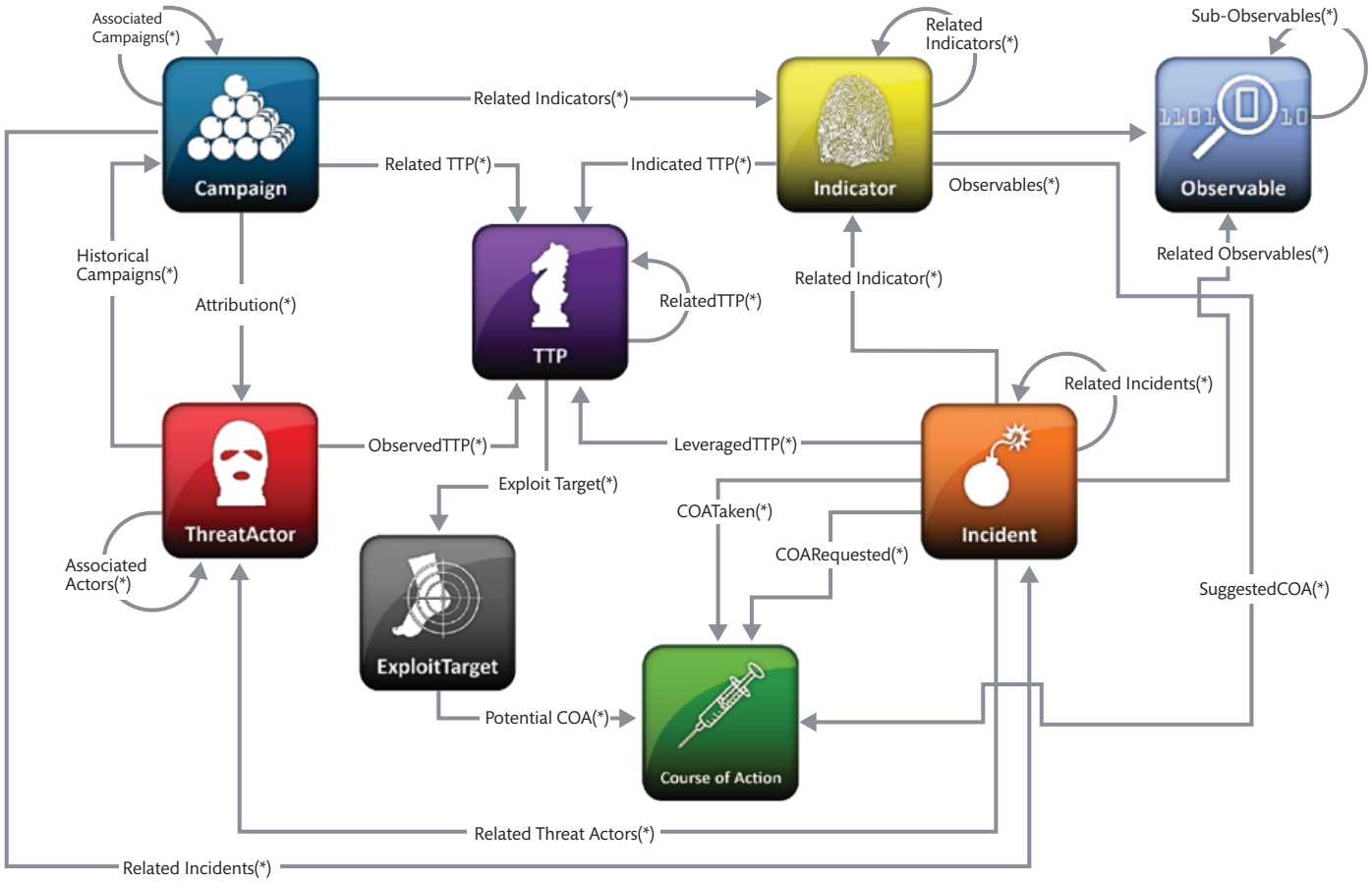
STIX Language

The STIX Language is a community effort being developed in collaboration with any and all interested parties for the specification, capture, characterization, and communication of standardized cyber threat information. It does this in a structured fashion to support more effective cyber threat management processes and application of automation.

STIX provides a common mechanism for addressing structured cyber threat information across and among a wide range of use cases improving consistency, efficiency, interoperability,

STIX™

STIX is a U.S. Department of Homeland Security–led effort of the office of Cybersecurity and Communications. MITRE, operating as DHS's FFRDC, manages the STIX website, community engagement, and discussion lists to enable open and public collaboration with all stakeholders.



STIX Architecture

and overall situational awareness. In addition, STIX provides a unifying architecture tying together a diverse set of cyber threat information including:

- Cyber Observables (e.g., a registry key is created, network traffic occurs to specific IP addresses, email from a specific address is observed, etc.)
- Indicators (potential observables with attached meaning and context)
- Incidents (instances of specific adversary actions)
- Adversary Tactics, Techniques, and Procedures (including attack patterns, malware, exploits, kill chains, tools, infrastructure, victim targeting, etc.)
- Exploit Targets (e.g., vulnerabilities, weaknesses or configurations)
- Courses of Action (e.g., incident response or vulnerability/weakness remedies)
- Cyber Attack Campaigns (sets of Incidents and/or TTP with a shared intent)
- Cyber Threat Actors (identification and/or characterization of the adversary)

To enable such an aggregate solution to be practical for any single use case, existing structured languages may be leveraged where appropriate such as Cyber Observable Expression (CybOX™), Malware Attribute Enumeration and Characterization (MAEC™), Common Attack Pattern Enumeration and Classification (CAPEC™), etc., and numerous flexibility mechanisms are designed into the language.

In particular, almost everything in this definitively-structured language is optional such that any single use case could leverage only the portions of STIX that are relevant for it — from a single field to the entire language or anything in between — without being overwhelmed by the rest.

Feedback Requested

STIX Community members make contributions to STIX development and manage issue tracking for the STIX schemas, tools, specifications, and supporting information by joining the STIX Community at https://stix.mitre.org/community/. Members of the cyber security community are also invited to participate in this growing community effort.

Open Vulnerability and Assessment Language — OVAL®

A Community-Developed Language for Determining Vulnerability and Configuration Issues on Computer Systems

OVAL is an international, information security community effort to standardize how to assess and report upon the machine state of computer systems. OVAL includes a language to encode system details, and an assortment of content repositories held throughout the community.

The language provides a framework for making assertions about a machine's state by standardizing the three main steps of the assessment process: representing the machine state of a system for testing; analyzing the system for the presence of the specified machine state (vulnerability, configuration, patch state, etc.); and reporting the results of this assessment.

The repositories are collections of publicly available and open content that utilize the language.

OVAL Language

The OVAL community has developed three schemas written in Extensible Markup Language (XML) to serve as the framework and vocabulary of the OVAL Language. These schemas correspond to the three steps of the assessment process: an OVAL System Characteristics schema for representing machine state, an OVAL Definition schema for expressing a specific machine state, and an OVAL Results schema for reporting the results of an assessment. The community has also developed numerous platform-specific component schemas for specifying and representing a system's machine state on those platforms.

OVAL Repository

Content written in the OVAL Language—XML documents such as OVAL Definitions files—is located in the many repositories found within the community. One such repository, the OVAL Repository hosted by The MITRE Corporation, is the central meeting place for the OVAL Community to discuss, analyze, store, and disseminate OVAL Definitions. Each definition in the OVAL Repository determines whether a specified software vulnerability, configuration issue, program, or patch is present on a system.

OVAL Adoption

Vendor organizations adopt OVAL by incorporating OVAL into their information security products and services, while users support OVAL by deploying products and services that have adopted OVAL to further enhance the security of their enterprises. A product or service is considered an OVAL Adopter if it uses OVAL as appropriate for communicating details of vulnerabilities, patches, security configuration settings, and other machine states.

To be an official OVAL Adopter a product or service must complete the OVAL Adoption Program Process:

- Make a declaration to adopt one or more OVAL Adoption Capabilities.
- Implement the OVAL Adoption Capabilities in the product or service.
- Complete the OVAL Adoption Questionnaire that provides an overview of how the OVAL Adoption Capabilities have been integrated into the product or service.

See the "Requirements and Recommendations for OVAL Adoption and Use" document and the formal OVAL Adoption Program description on the OVAL Web site for details.

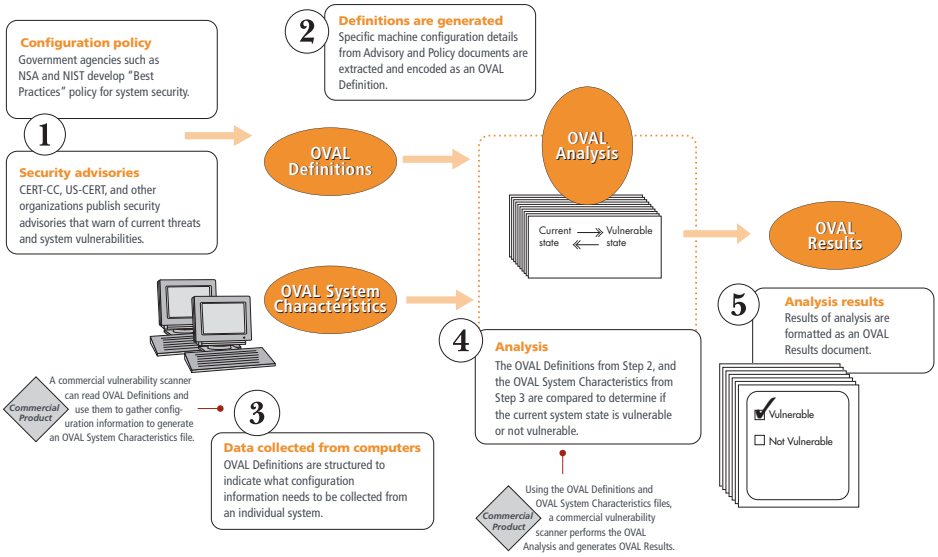
OVAL Community

The information security community contributes to the development of OVAL by participating in the creation of the OVAL Language on the OVAL Developer's Forum and by writing definitions for the OVAL Repository through the OVAL Repository Forum. An OVAL Board consisting of representatives from a broad spectrum of industry

OVAL Celebrating 10 Years

The MITRE Corporation maintains OVAL and its public Web site, manages the adoption program, and provides impartial technical guidance to the OVAL Board and Community throughout the process to ensure OVAL serves the public interest.

MITRE
202 Burlington Road, Bedford, MA 01730-1420
www.mitre.org



How the three core components of the OVAL Language work together during a standard vulnerability assessment process

and government organizations from around the world oversees and approves the OVAL Language and monitors the posting of the definitions hosted in the OVAL Repository. This means that OVAL, which is funded by the Office of Cybersecurity and Communications at the U.S. Department of Homeland Security for the benefit of the community, reflects the insights and combined expertise of the broadest possible collection of security and system administration professionals worldwide.

OVAL Definitions

OVAL Definitions are machine-readable, gold standard tests that definitively determine whether the specified software vulnerability, configuration issue, program, or patch is present on a system. There are four main classes of OVAL Definitions:

OVAL Vulnerability Definitions
Tests that determine the presence of vulnerabilities on systems.

OVAL Compliance Definitions
Tests that determine whether the configuration settings of a system meets a security policy.

OVAL Inventory Definitions Tests that determine whether a specific piece of software is installed on a system.

OVAL Patch Definitions Tests that determine whether a particular patch is appropriate for a system.

OVAL Definitions include metadata, a high-level summary, and the detailed definition. Definition metadata provides the OVAL-ID, status of the definition (Draft, Interim, or Accepted), the CVE name or other reference on which the definition (or definitions) is based, the version of the official OVAL Definition Schema the definition works with, a brief description of the security issue covered in the definition, the main author, and a list of the significant contributors to the development of the definition.

The high-level summary includes the following: "Vulnerable software exists," which states the specific operating system (OS), the name of the file with the vulnerability in it, application version, and patch status; and "Vulnerable configuration," which indicates if the service is running or not, specific configuration settings, and workarounds.

The detailed portion of definitions provides the logic for checking for the system characteristics (OS installed, settings in the OS, software applications installed, and settings in applications) to indicate that vulnerable software exists, and configuration attributes (registry key values, file system attributes, and configuration files) to indicate that a vulnerable configuration exists.

Any member of the OVAL Community may submit OVAL Definitions, as detailed on the OVAL Web site. OVAL Content hosted in other OVAL repositories is also often available to the public.

Malware Attribute Enumeration and Characterization — MAEC™

A Structured Language for Attribute-Based Malware Characterization

MAEC is a structured language for encoding and communicating high-fidelity information about malware based upon attributes such as behaviors, artifacts, and attack patterns.

By eliminating the ambiguity and inaccuracy that currently exists in malware descriptions and by reducing reliance on signatures, MAEC aims to:

- **Improve human-to-human, human-to-tool, tool-to-tool, and tool-to-human communication about malware**
- **Allow for the faster development of countermeasures by enabling the ability to leverage responses to previously observed malware instances**
- **Reduce potential duplication of malware analysis efforts by researchers**

Challenge

Modern methods for detecting and combating malware often rely on the characterization of malware attributes and behaviors. The use of static and dynamic analysis techniques allows for an encompassing profile of malware to be constructed based upon its disassembled binary and observed run-time behavior.

Yet, the lack of an accepted standard for unambiguously characterizing malware means that there is no clear method for communicating the specific malware attributes detected in malware by the analyses, nor for enumerating its fundamental makeup. The results are non-interoperable and disparate malware reporting between organizations, disjointed or inaccurate malware attribution, the duplication of malware analysis efforts, increased difficulty in determining the severity of a malware threat, and a greater period of time between malware infection and detection/response, among others.

Solution

MAEC solves these problems. The characterization of malware using abstract patterns offers a wide range of benefits over the usage of physical signatures, and allows for the accurate encoding of how malware operates and the specific actions that it performs. Such information can not only be used for malware detection, but also for assessing the end-goal the malware is pursuing and the corresponding threat that it represents.

Focusing on the attributes and behaviors of malware facilitates detection and analysis of emerging, sophisticated malware threats that circumvent the traditional signature-based and heuristic approaches. Characterizing malware in a standard way supports collaboration across organizations and the identification of common behavior, functionality, and code bases across instances of malware.

MAEC achieves this end result by utilizing three community-developed components to define the standardized MAEC Language:

- Element dictionaries
- Schemas for defining vocabulary syntax
- Standard output formats based on schemas

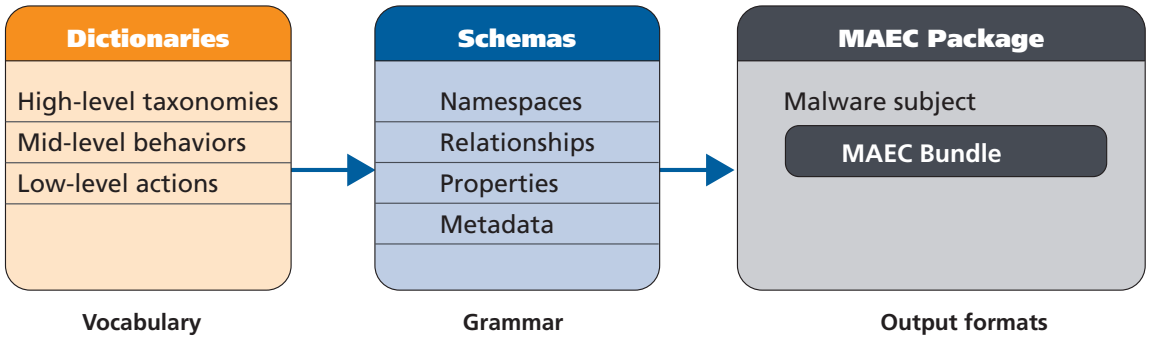
MAEC Language

MAEC is being developed as a formal language for characterizing attributes and behaviors of all types of malware. Initially MAEC will focus on characterizing the most common malware types, including Trojans, worms, and rootkits, but will ultimately be applicable to more esoteric malware types. As a language, MAEC will have a grammar and vocabulary that provide a standard means of communicating information about malware attributes.

MAEC™
maec.mitre.org

The MITRE Corporation maintains MAEC and its public Web site presence and provides impartial technical guidance to the MAEC Community throughout the process to ensure MAEC serves the public interest.

MITRE
202 Burlington Road, Bedford, MA 01730-1420
www.mitre.org



MAEC's core components include a vocabulary, grammar, and forms of standardized output.

MAEC Dictionaries – a series of dictionaries for defining three distinct levels of malware elements—low-level actions, mid-level behaviors, and high-level mechanisms.

MAEC Schemas – a syntax for the vocabulary of actions, behaviors, and taxonomies, and an interchange format for structured information about these elements.

MAEC Output Formats – standard output formats that can be used for particular use cases, including the description of a malware instance, malware intrusion set, or malware families in terms of MAEC's dictionaries and schemas.

MAEC Use Cases

As a domain-specific language for the characterization of malware, MAEC has a broad range of uses, especially with regards to malware analysis and anti-malware operations. The following are just a few of the use cases that MAEC will support:

Analysis-Oriented Use Cases

- Common Vocabulary for Malware Analysis
- Enhanced Data Sharing Between Malware Repositories
- Objective Criteria for Anti-malware Tool Assessment

Operations-Oriented Use Cases

- Uniform Malware Reporting Format
- Malware Detection
- Malware Threat Assessment
- Malware Response
- Malware/Attacker Correlation

Feedback Requested

MAEC Community members can make contributions to MAEC development and manage issue tracking for the MAEC schemas, utilities, specifications, and supporting information by joining the MAEC Community at https://maec.mitre.org/community/. Members of the cyber security community are invited to participate in this growing community effort.

Cyber Observable eXpression — CybOX™
A Structured Language for Cyber Observables

CybOX provides a common structure for representing cyber observables across and among the operational areas of enterprise cyber security that improves the consistency, efficiency, and interoperability of deployed tools and processes, as well as increases overall situational awareness by enabling the potential for detailed automatable sharing, mapping, detection, and analysis heuristics.

Examples of cyber observables include:

- **A Registry Key is created**
- **A File is deleted**
- **A Mutex exists**
- **Specific HTTP Get Request received**
- **A file has a specific MD5 hash**
- **Data is sent to an address on a socket**
- **Network traffic occurs to specific IP addresses**
- **Email from a specific address is observed**
- **Application logs show communication on certain ports**
- **A service's configuration is changed**
- **A remote thread is created**

International in scope and free for public use, CybOX is a structured language for the specification, capture, characterization, and communication of events or stateful properties that are observable in the operational domain. A wide variety of high-level cyber security use cases rely on such information including event management/ logging, malware characterization, intrusion detection, incident response/management, attack pattern characterization, indicator sharing, etc. CybOX provides a common structure for representing cyber observables across and among these use cases thereby improving consistency, efficiency, interoperability, and overall situational awareness for the enterprise.

Challenge

The concept of observable events or properties in the operational cyber realm is a central underlying element of many of the different activities involved in cyber security. Until recently, no uniform structured mechanism existed for specifying, capturing, characterizing, or communicating these cyber observables. Each activity area, each use case and often each supporting tool vendor uses its own unique approach that inhibits consistency, efficiency, interoperability and overall situational awareness.

Solution

CybOX is a standardized language for representing cyber observables, whether dynamic events or stateful properties that are observable in the operational cyber domain. CybOX is not targeted at a single cyber security use case but rather is intended to be flexible enough to offer a common solution for all cyber security use cases requiring the ability to deal with cyber observables. It is also intended to be flexible enough to allow both the high-fidelity description of instances of cyber observables that have been measured in an operational context as well as more abstract patterns for potential observables that may be targets for observation and analysis apriori. By specifying a common structured schematic mechanism for these cyber observables, CybOX enables detailed automatable sharing, mapping, detection and analysis heuristics.

CybOX is targeted to support a wide range of relevant cyber security domains including:

- Threat assessment and characterization (detailed attack patterns)
- Malware characterization
- Operational event management
- Logging

CybOX is a U.S. Department of Homeland Security–led effort of the office of Cybersecurity and Communications. MITRE, operating as DHS's FFRDC, manages the CybOX website, community engagement, and discussion lists to enable open and public collaboration with all stakeholders.

- Cyber situational awareness
- Incident response
- Indicator sharing
- Digital forensics
- Etc.

Through utilization of the standardized CybOX Language, relevant observable events or properties can be captured and shared, defined in indicators and rules, or used to adorn the appropriate portions of attack patterns and malware profiles in order to tie the logical pattern constructs to real-world evidence of their occurrence or presence for attack detection and characterization. Incident response and management can then take advantage of all of these capabilities to investigate occurring incidents, improve overall situational awareness and improve future attack detection, prevention and response.

Supported Use Cases

CybOX is intended to be flexible enough to provide a common foundation for a wide diversity of cyber security use cases requiring the ability to deal with cyber observables. For most use cases, the utilization of CybOX should be indirect with primary focus on the use case domain-specific standard or solution which leverages CybOX as an enabler. See table at right for examples of current use cases.

Feedback Requested

CybOX Community members can make contributions to CybOX development and manage issue tracking for the CybOX schemas, utilities, specifications, and supporting information by joining the CybOX Community at https://cybox.mitre.org/community/. Members of the cyber security community are invited to participate in this growing community effort.

Supported Use Case	Relevant Process	Domain Specific Standard
Analyze event data from diverse set of sensors of different types and different vendors	Event Management	CybOX
Detect malicious activity utilizing attack patterns	Attack Detection	Common Attack Pattern Enumeration and Classification (CAPEC™)
Detect malicious activity utilizing malware behavior characterizations	Attack Detection	Malware Attribute Enumeration and Characterization (MAEC™)
Enable automated attack detection signature rule generation	Attack Detection	CybOX, MAEC, CAPEC, STIX
Characterize malicious activity utilizing attack patterns	Incident Response/ Management	CAPEC, STIX
Identify new attack patterns	Threat Characterization	CAPEC
Prioritize existing attack patterns based on tactical reality	Security Testing and Secure Development	CAPEC, STIX
Characterize malware behavior	Malware Analysis	MAEC
Guide malware analysis utilizing attack patterns	Malware Analysis	MAEC, CAPEC
Detect malware effects	Attack Detection and Incident Response/ Management	STIX, MAEC, Open Vulnerability and Assessment Language (OVAL®)
Enable collaborative attack indicator sharing	Information Sharing	STIX, TAXII
Empower and guide incident management utilizing attack patterns and malware characterizations	Incident Response/ Management	STIX, CAPEC, MAEC, CybOX
Enable consistent, useful and automation-capable incident alerts	Incident Response/ Management	STIX, MAEC, CAPEC, CEE
Enable automatic application of mitigations specified in attack patterns	Incident Response/ Management	STIX
Enable incident information sharing	Incident Response/ Management	STIX
Support correlation between observed properties and malicious indicators as part of digital forensics	Digital Forensics	Digital Forensics XML (DFXML), STIX, MAEC, CAPEC
Capture digital forensics analysis results	Digital Forensics	Ongoing work to refine DFXML based on CybOX
Capture digital forensics provenance information	Digital Forensics	Ongoing work to refine DFXML based on CybOX
Enable collaborative sharing of digital forensics information	Digital Forensics	Ongoing work to refine DFXML based on CybOX, STIX, TAXII
Enable explicit and implicit sharing controls for cyber observable information	Information Sharing	STIX, CybOX, TAXII
Enable new levels of meta-analysis on operational cyber observables	Cyber Situational Awareness	CybOX, STIX

Common Weakness Scoring System — CWSS™

Scoring the Severity of Software Weaknesses

CWSS is a mechanism for scoring the severity of Common Weakness Enumeration (CWE™) entries discovered in an enterprise's software applications, especially when used in conjunction with the Common Weakness Risk Analysis Framework (CWRAF™).

CWSS can also be used by individual developers to prioritize unfixed weaknesses within their own software.

Challenge

When a security analysis of a software application is performed, such as when using an automated code assessment tool, developers often face hundreds or thousands of individual findings for individual weaknesses that are identified in their code. In certain circumstances, a software weakness can lead to an exploitable vulnerability. For example, a buffer overflow weakness might arise from an input routine where the programmer does not properly validate the length of an input buffer. This weakness only contributes to a vulnerability if the input can be influenced by a malicious party, and if that malicious input can inserted into a buffer that is smaller than the malicious input.

Due to the high volume of reported weakness findings, developers are forced to prioritize which issues they should investigate and fix first. Similarly, when assessing design and architecture choices and their associated weaknesses, there needs to be a method for prioritizing them relative to each other and with the other issues in the application. Finally, software consumers want to know what issues they should worry about versus others, and what to ask about to get a more secure product from their vendors and suppliers.

Further complicating the problem, the importance of a weakness usually depends on the business or mission needs that the software is supporting, the kinds of technologies in use, and the threat environment.

In short, people need to be able to reason and communicate about the relative importance of different weaknesses. While various scoring methods are used today, they are either ad hoc or inappropriate for use against the evaluation of software security.

Solution

CWSS provides a mechanism for scoring weaknesses in a consistent, flexible, open manner while enabling an organization to reflect the context of their business domain(s). It is a collaborative, community-based effort that is addressing the needs of stakeholders across government, academia, and industry. CWSS is a part of the project, co-sponsored by the Software Assurance program in the office of Cybersecurity and Communications of the U.S. Department of Homeland Security (DHS).

CWSS:

- Provides a common framework for prioritizing security errors ("weaknesses") that are discovered in software applications
- Provides a quantitative measurement of the unfixed weaknesses that are present within a software application
- Can be used by developers to prioritize unfixed weaknesses within their own software
- In conjunction with CWRAF, can be used by consumers to identify the most important weaknesses for their business domains, in order to inform their acquisition and protection activities as one part of the larger process of achieving software assurance.

Learn More – https://cwe.mitre.org/cwss

CWSS™

cwe.mitre.org/cwss

The MITRE Corporation maintains CWSS on the Common Weakness Enumeration (CWE™) website and provides impartial technical guidance to the CWSS community throughout the process to ensure CWSS serves the public interest.

202 Burlington Road, Bedford, MA 01730-1420
www.mitre.org
MITRE

Common Weakness Risk Analysis Framework — CWRAF™

Prioritizing the Severity of Software Weaknesses in Your Own Organization

CWRAF is a way for organizations to apply the Common Weakness Scoring System (CWSS™) using specialized scenarios, or "vignettes," in order to prioritize those Common Weakness Enumeration (CWE™) entries that are most relevant to their own businesses, missions, and deployed technologies.

CWRAF provides a framework for scoring software weaknesses in a consistent, flexible, open manner, while accommodating context of an organization's business domain(s). It is a collaborative, community-based effort that is addressing the needs of its stakeholders across government, academia, and industry. CWRAF is a part of the Common Weakness Enumeration (CWE™) project, co-sponsored by the Software Assurance program in the office of Cybersecurity and Communications of the U.S. Department of Homeland Security (DHS).

CWRAF benefits:

- Includes a mechanism for measuring risk of security-relevant software development errors ("weaknesses") in a way that is closely linked with the potential impact to an organization's business or mission.
- Supports the automatic selection and prioritization of relevant weaknesses, customized to the specific needs of the organization's business or mission.
- Can be used by organizations in conjunction with CWSS to identify the most important weaknesses for their business domains, in order to inform their acquisition and protection activities as one part of the larger process of achieving software assurance.
- Leverages the construct in CWE's Common Consequences information, where all CWEs, if manifested in an exploitable manner, result in the attacker being able to cause one or more of the following technical impacts: modify data; read data; DoS: unreliable execution; DoS: resource consumption; execute unauthorized code or commands; gain privileges /assume identity; bypass protection mechanism; and hide activities.

Vignettes Explained

CWRAF and CWSS allow users to rank classes of weaknesses independent of any particular software package, in order to prioritize them relative to each other (e.g., "buffer overflows are higher priority than memory leaks"). This method of prioritization, sometimes referred to as a "Top-N list," is used by the CWE/SANS Top 25, OWASP Top Ten, and similar efforts. CWRAF and CWSS allow users to create their own custom Top-N lists.

Within CWRAF, a vignette provides a shareable, formalized way to define a particular environment or operational context, i.e., the role that software plays within that environment, and an organization's priorities with respect to software security of that piece of software. It identifies essential resources and capabilities, as well as their importance relative to security principles such as confidentiality, integrity, and availability.

Vignettes allow CWSS to support diverse audiences who may have different requirements for how to prioritize weaknesses. CWSS scoring can occur within the context of a vignette.

There are currently 23 vignettes that are being actively developed for CWRAF within the categories, or "domains," of Banking/Finance, Chemical, e-Commerce, Emergency Services, Energy, e-Voting, Human Resources, National Defense, Public Health, Social Media, and Telecommunications. The CWRAF community will help to refine these and develop others over time including for Food & Water, Manufacturing, Homeland Security, Government (other), Teleworking, and Shipping/Transportation. Feedback is welcome.

Learn More - https://cwe.mitre.org/cwraf

CWRAF™

cwe.mitre.org/cwraf

The MITRE Corporation maintains CWRAF on the Common Weakness Enumeration (CWE™) website and provides impartial technical guidance to the CWRAF community throughout the process to ensure CWRAF serves the public interest.

202 Burlington Road, Bedford, MA 01730-1420
www.mitre.org
MITRE

Common Weakness Enumeration — CWE™

A Community-Developed Dictionary of Software Weakness Types

CWE, targeted to developers and security practitioners, is a formal list of software weaknesses that:

- Serves as a common language for describing software security weaknesses in architecture, design, or code.
- Serves as a measuring stick for software security tools targeting these weaknesses.
- Provides a common baseline definition for weakness identification, mitigation, and prevention efforts.
- Is industry-endorsed via the CWE Community and CWE-Compatible Products.

Some Common Types of Software Weaknesses:

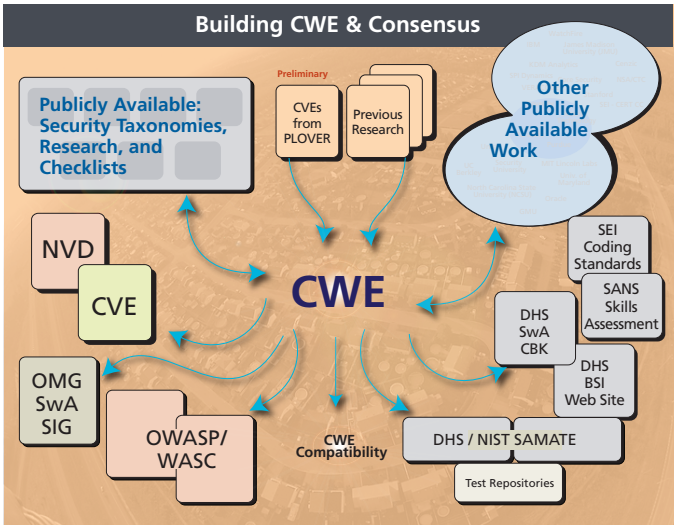
- Buffer overflows, format strings, etc.
- Structure and validity problems
- Common special element manipulations
- Channel and path errors
- Handler errors
- User interface errors
- Pathname traversal and equivalence errors
- Authentication errors
- Resource management errors
- Insufficient verification of data
- Code evaluation and injection
- Randomness and predictability

Challenge

Software acquirers want assurance that the software products they are obtaining are reviewed for known types of exploitable security weaknesses, and the acquisition groups in large government and private organizations are moving forward to use these types of reviews as part of future contracts. Until recently the tools and services that could be used for this type of review were new at best and there were no nomenclature, taxonomies, or standards to define the capabilities and coverage of them. That made it difficult to comparatively decide which tool or service was best suited for a particular job. What was needed was a standard list and classification of software security weaknesses to serve as a unifying language of discourse and a measuring stick for tools and services.

Solution

CWE is a community-developed formal list or dictionary of common software weaknesses. Leveraging the diverse thinking on this topic from academia, the commercial sector, and government, CWE unites the most valuable breadth and depth of content and structure to serve as a standard definition. Our objective is to help shape and mature the code security assessment industry and also dramatically accelerate the use and utility of software assurance capabilities for organizations in reviewing the software systems they acquire or develop.



Working from these collections—as well as those contained in the other information sources listed on the CWE Web site—we developed the current draft of the CWE List, which includes over 600 separate weaknesses and we have created a branding and compliance program to acknowledge and validate tools and services using CWE identifiers.

Community

The following organizations are actively contributing to the development of CWE: Apple, Cenzec, Core Security, HP, GrammaTech, Klocwork, IBM, Parasoft, Veracode,

cwe.mitre.org

The MITRE Corporation maintains CWE and its public Web site, manages the compatibility program, and provides impartial technical guidance to the CWE Community throughout the process to ensure CWE serves the public interest.

202 Burlington Road, Bedford, MA 01730-1420
www.mitre.org
MITRE

CWE Entries include:

- name of the weakness type
- description of the type
- alternate terms for the weakness
- description of the behavior of the weakness
- description of the exploit of the weakness
- likelihood of exploit for the weakness
- description of the consequences of the exploit
- potential mitigations
- node relationship information
- source taxonomies
- code samples for the languages/architectures
- CVE identifiers of vulnerabilities for which that type of weakness exists
- references

CWE ID 415	Double Free
Description	The product calls free() twice on the same memory address, potentially leading to modification of unexpected memory locations.
Likelihood of Exploit	Low to Medium
Common Consequences	Access control: Doubly freeing memory may result in a write-what-where condition, allowing an attacker to execute arbitrary code.
Potential Mitigations	Architecture and Design: Choose a language that provides automatic memory management. Implementation: Ensure that each allocation is freed only once. After freeing a chunk, set the pointer to NULL to ensure the pointer cannot be freed again. In complicated error conditions, be sure that clean-up routines respect the state of allocation properly. If the language is object oriented, ensure that object destructors delete each chunk of memory only once. Implementation: Use a static analysis tool to find double free instances.
Demonstrative Examples	Example 1: The following code shows a simple example of a double free vulnerability. Double free vulnerabilities have two common (and sometimes overlapping) causes: - Error conditions and other exceptional circumstances - Confusion over which part of the program is responsible for freeing the memory Although some double free vulnerabilities are not much more complicated than the previous example, most are spread out across hundreds of lines of code or even different files. Programmers seem particularly susceptible to freeing global variables more than once. Example 2: While contrived, this code should be exploitable on Linux distributions which do not ship with heap-chunk check summing turned on.
Observed Examples	CVE-2002-0059 - Double free from malformed compressed data. CVE-2003-0545 - Double free from invalid ASN.1 encoding. CVE-2003-1048 - Double free from malformed GIF. CVE-2004-0642 - Double free resultant from certain error conditions. CVE-2004-0772 - Double free resultant from certain error conditions. CVE-2005-0891 - Double free from malformed GIF. CVE-2005-1689 - Double free resultant from certain error conditions.
Node Relationships	Child Of - Operation on Resource in Wrong Phase of Lifetime (666) in View (1000) Child Of - Duplicate Operations on Resource (675) in View (1000) Child Of - Resource Management Errors (399) in View (699) Peer Of - Use After Free (416) in View (699 & 1000) Peer Of - Write-what-where Condition (123) in View (700) Child Of - Indicator of Poor Code Quality (398) in View (700) Child Of - Weaknesses that Affect Memory (633) in View (631) Child Of - CERT C Secure Coding Section 08 – Memory Management (MEM) (742) in View (734) Member Of - Weaknesses Examined by SAMATE (630) in View (630) Peer Of - Signal Handler Race Condition (364) in View (1000)
Source Taxonomies	PLOVER - DFREE - Double-Free Vulnerability 7 Pernicious Kingdoms - Double Free CLASP - Doubly freeing memory CERT C Secure Coding - MEM00-C - Allocate and free memory in the same module, at the same level of abstraction CERT C Secure Coding - MEM01-C - Store a new value in pointers immediately after free() CERT C Secure Coding - MEM31-C - Free dynamically allocated memory exactly once CERT C Secure Coding - MEM00-C - Allocate and free memory in the same module, at the same level of abstraction CERT C Secure Coding - MEM01-C - Store a new value in pointers immediately after free() CERT C Secure Coding - MEM31-C - Free dynamically allocated memory exactly once
Applicable Platforms	C C++
White Box Definitions	A weakness where code path has: 1. start statement that relinquishes a dynamically allocated memory resource 2. end statement that relinquishes the dynamically allocated memory resource

Symantec, CAST, EC-Council, EMC, Japan's Information-technology Promotion Agency, ISC2, NIST, and Red Hat.

We are also leveraging the work, ideas, and contributions of researchers at Armorize Technologies, Carnegie Mellon's CERT/CC, CERIAS/Purdue University, Cigital, KDM Analytics, Kestrel Technology, Oracle, OWASP, SANS Institute, SkillBridge, UNISYS, WASC, and WhiteHat Security. See the CWE Web site for a complete list of participants and how your organization can contribute.

CWE List

International in scope and free for public use, CWE provides a unified, measurable set of software weaknesses that will enable more effective discussion, description, selection, and use of software security tools and services that can find these weaknesses in source code.

The CWE List is currently offered in many views including:

Dictionary - an alphabetic view of the list's enumerated weaknesses

Classification Tree - provides access to individual weaknesses with more simplicity to various potential users through classification layering

Graphical - allows users to better understand individual weaknesses in the classification tree through their broader context and relationships

Slices-by-Topic - provide selective subsets of CWE by language or some other attribute

XML/XSD of CWE content — in toto or by slice — is also available. Additional formats and views will be added in the future. Visit the CWE Web site for the latest information.

MITRE

Learn More – https://cwe.mitre.org

Common Vulnerabilities and Exposures — CVE®

The Standard for Information Security Vulnerability Names

CVE is a dictionary of common names for publicly known information security vulnerabilities. CVE's common identifiers—called CVE Identifiers—make it easier to share data across separate network security databases and tools, and provide a baseline for evaluating the coverage of an organization's security tools.

CVE is:

- **One name for one vulnerability or exposure**
- **One standardized description for each vulnerability or exposure**
- **A dictionary rather than a database**
- **The way for disparate databases and tools to "speak" the same language**
- **The way to interoperability and better security coverage**
- **A basis for evaluation among tools and databases**
- **Free for public download and use**
- **Industry-endorsed via the CVE Editorial Board and CVE-Compatible Products**

Why CVE

CVE was launched in 1999 when most information security tools used their own databases with their own names for security vulnerabilities. At that time there was no significant variation among products and no easy way to determine when the different databases were referring to the same problem. The consequences were potential gaps in security coverage and no effective interoperability among the disparate databases and tools. In addition, each tool vendor used different metrics to state the number of vulnerabilities or exposures they detected, which meant there was no standardized basis for evaluation among the tools.

CVE's common, standardized identifiers provided the solution to these problems. CVE is now the industry standard for vulnerability names. CVE Identifiers provide reference points for data exchange so that information security products and services can speak with each other. They also provide a baseline for evaluating the coverage of tools and services so that users can determine which tools are most effective and appropriate for their organization's needs. In short, products and services compatible with CVE provide better coverage, easier interoperability, and enhanced security.

How CVE Works

The process of creating a CVE Identifier begins with the discovery and report of a potential security vulnerability. The information is then assigned a CVE Identifier by a CVE Numbering Authority (CNA) and posted on the CVE List on the CVE Web site by the CVE Editor. As part of its management of CVE, The MITRE Corporation functions as Editor and Primary CNA. The CVE Editorial Board oversees this process.

CVE in Use

As the industry standard, CVE Identifiers are used in numerous information security products and services from around the world. These "CVE-Compatible" products include vulnerability databases; security advisories and archives; vulnerability notification, assessment, and remediation products; intrusion detection, management,

202 Burlington Road, Bedford, MA 01730-1420
www.mitre.org

MITRE

monitoring, and response products; incident management products; data/event correlation products; educational materials; firewalls; patch management products; policy compliance products; and security information management tools.

The U.S. National Vulnerability Database (NVD) of CVE fix information (http://nvd.nist.gov)—sponsored by the office of Cybersecurity and Communications at the U.S. Department of Homeland Security and operated by the National Institute of Standards and Technology (NIST)—is based on and synchronized with the CVE List. NVD also includes Security Content Automation Protocol (SCAP) mappings for CVE-IDs. SCAP is a method for using specific standards to enable automated vulnerability management, measurement, and policy compliance evaluation (e.g., FISMA compliance) and CVE is one of the open community standards SCAP uses for enumerating, evaluating, and measuring the impact of software problems and reporting results. The use of CVE by U.S. agencies was also recommend by NIST in two official documents in 2002, and in June 2004, the U.S. Defense Information Systems Agency (DISA) issued a task order for information assurance applications that requires the use of products that use CVE Identifiers.

CVE also helped to create new initiatives: MITRE's Common Weakness Enumeration (CWE™) dictionary of software weaknesses is based in part on the 55,000+ CVE Identifiers on the CVE List, and its Open Vulnerability and Assessment Language (OVAL®), a community-developed language for determining vulnerability and configuration issues on computer systems using community-developed XML schemas and definitions, bases its OVAL Vulnerability Definitions primarily on CVE Identifiers.

And in 2011, the International Telecommunication Union's (ITU-T) Cybersecurity Rapporteur Group, which is the telecom/information system standards body within the treaty-based 150-year-old intergovernmental organization, adopted CVE as a part of its "Cybersecurity Information Exchange Framework (X.CYBIEF)" by issuing *Recommendation ITU-T X.1520 Common Vulnerabilities and Exposures (CVE)*, that is based upon CVE's current Compatibility Requirements, and any future changes to the document will be reflected in subsequent updates to X.CVE.

Each CVE Identifier Includes

- CVE Identifier number (i.e., "CVE-1999-0067").
- Brief description of the security vulnerability or exposure.
- Any pertinent references (i.e., vulnerability reports and advisories or OVAL-ID).

CVE Community

CVE is an international information security community effort. In addition to the contributions of the CVE Editorial Board and the CVE Sponsor, numerous organizations from around the world have made their products CVE-Compatible, have included CVE Identifiers in their security advisories, and/or have adopted or promoted the use of CVE.

CVE Editorial Board The CVE Editorial Board, which includes members from numerous information security-related organizations from around world such as commercial security tool vendors, members of academia, research institutions, government agencies, and other prominent security experts, oversees which vulnerabilities or exposures are included in the CVE List.

CVE Sponsor CVE is sponsored by the office of Cybersecurity and Communications at the U.S. Department of Homeland Security.

CVE-Compatible Products and Services Numerous organizations from around the world have made their information security products and services "CVE-Compatible" by incorporating CVE Identifiers. Refer to the CVE Compatibility section of the CVE Web site for a list of official CVE-Compatible Products and Services and Declarations to Be CVE-Compatible.