CWE User Experience Working Group Meeting

October 25, 2023

Agenda

This meeting is being recorded :-)

- **Purpose**
- **Housekeeping**
- **Primary topics**
  - Red Hat CWE Blog (Co-chair presentation)
  - Technical Weaknesses (Member presentation)
  - Open Discussion
- **Reminders and Adjourn**

UEWG: Purpose

- **Mission:** Identifying areas where CWE content, rules, guidelines, and best practices must improve to better support stakeholder community, and work collaboratively to fix them
- **Periodic reporting of activities to CWE Board**
  - (next quarterly Board meeting TBD Q4-2023)
- **Please solicit participations from your contacts**
  - Contact: cwe@mitre.org & capec@mitre.org

Housekeeping

- **CWE UEWG Meeting frequency**
  - Meetings have been changed and are now set to occur on the last Wednesday of the month

- **The CWE Program is continuously seeking feedback on UEWG activities and priorities during these sessions or via email: cwe@mitre.org**

Topic 1

Red Hat CWE Blog

Co-chair - Przemyslaw Roguski

Red Hat's CWE journey

CWE/CAPEC User Experience Working Group

Przemysław Roguski

Red Hat

What prompted Red Hat to publish the "Red Hat's CWE journey" and "Weakness risk-patterns: A Red Hat way to identify poor software practices in the secure development lifecycle" blog posts?

- Limited knowledge of how the CWE program can be used in a daily security work
- Lack of the weakness understanding
- Weakness vs Vulnerability frequent misunderstanding

Red Hat’s CWE beginning and recent changes in the Red Hat CWE approach:

- Red Hat attained CWE compatibility at the end of 2012
- Custom grouping was used until 2022
- From the beginning of 2022, Red Hat's default CWE coverage has been based on the CWE-699 Software Development view

Reactive and proactive CWE usage:

- More efficiently described the nature of vulnerabilities affecting Red Hat products
- Better, more accurate weakness monitoring
- More useful weakness analysis and statistics
- CWE classifications used in Secure Development Lifecycle (SDL)
  - Threat Models
  - Static Application Security Testing (SAST)
  - Penetration Testing

Always something can be done better:

- CWE assignment to CVE improvements
  - RCM working group
- Repeated weaknesses in software

Resources:

https://www.redhat.com/en/blog/red-hat-cwe-journey

https://www.redhat.com/en/blog/weakness-risk-patterns

linkedin.com/company/red-hat

facebook.com/redhatinc

youtube.com/user/RedHatVideos

twitter.com/RedHat

Red Hat

Topic 2

Technical Weaknesses

UEWG Member - John Keane

Why Software Security Is **Not** Different from Software Quality

John J Keane Jr

jkeanejr@aol.com


A Presentation by
The
Software Angel
of Death

OWASP
Application Security Verification Standard 4.0.3

Financial regulators do not tolerate external financial audits with no access to the books, sample transactions, or the people performing the controls. Industry and governments must demand the same standard of transparency in the software engineering field.

We strongly encourage the use of security tools within the development process itself. DAST and SAST tools can be used continuously by the build pipeline to find easy to find security issues THAT SHOULD NEVER BE PRESENT.

OWASP Mobile Application Security Verification Standard 2.0.0

MASVS-CODE: Code Quality

Mobile apps have many data entry points, including the UI, IPC, network, and file system, which might receive data that has been inadvertently modified by untrusted actors.

By treating this data as untrusted input and properly verifying and sanitizing it before use, developers can prevent classical injection attacks, such as SQL injection, XSS, or insecure deserialization. However, other common coding vulnerabilities, such as memory corruption flaws, are hard to detect in penetration testing but easy to prevent with secure architecture and coding practices.

Developers should follow best practices such as the OWASP Software Assurance Maturity Model (SAMM) and NIST.SP.800-218 Secure Software Development Framework (SSDF) to avoid introducing these flaws in the first place.

This category covers coding vulnerabilities that arise from external sources such as app data entry points, the OS, and third-party software components. Developers should verify and sanitize all incoming data toprevent injection attacks and bypass of security checks. They should also enforce app updates and ensure that the app runs up-to-date platforms to protect users from known vulnerabilities.

But First A Trip To The Dark Ages of Computing

- Buffer Overflows were understood and partially publicly documented as early as 1972
  - Computer Security Technology Planning Study laid out the technique:
    - The code performing this function does not check the source and destination addresses properly, permitting portions of the monitor (Kernel) to be overlaid by the user. **This can be used to inject code into the monitor (kernel) that will permit the user to seize control of the machine**.
  - Actively discussed during Graduate Studies
  - Focus on Performance Only
  - Not recognized as potential SECURITY issue at that time
- The earliest documented hostile exploitation of a buffer overflow was in 1988.
  - It was one of several exploits used by the Morris worm to propagate itself over the Internet.

2016

- MITRE Translates Software Quality Issues into a Common Language | The MITRE Corporation
- https://www.electronicdesign.com/industrial-automation/article/21805867/whats-the-difference-between-cwe-and-cqe.
  - MITRE's John Marien notes, “The Common Weakness Enumeration (CWE) lists quality issues that can be exploited. One or more weaknesses can create a vulnerability. Yet beyond these security-relevant weaknesses, there's a large set of quality issues not covered by CWE.”

Private Conversation With John Marien

- In 2016, 60% of Security Weaknesses were directly attributable to Quality Weaknesses
- In 2022, it appears that is now 85%

What Else Happened In 2016?

- Publication of “Software Quality Assurance”
  - Reliability is an important prerequisite to software security. It is hard to conceive of a secure system that is unreliable, but easy to think of a reliable system that may be unsecure….
  - Security must be a quality concern for software engineers building any software system that manages stakeholder resources, including intellectual property and identity information.
- Software Security - an overview | ScienceDirect Topics

Using FORTIFY Taxonomy-2019

- https://vulncat.fortify.com/en
- Searched On
  - Availability – 74 Weaknesses
  - Reliability – 77 Weaknesses
  - Safety – 62 Weaknesses
  - Maintainability – 31 Weaknesses
  - Performance – 175 Weaknesses
  - Bad Practices – 138 Weaknesses
  - Supportability – 46 Weaknesses
  - Dependability – 91 Weaknesses
- Created Separate Worksheet for Each Technical Weakness
- 442 Unique Weaknesses
  - Mapped to Related Security Weakness
- 694 Total Weaknesses
- 276 Security Weaknesses Mapped to 1 Technical Weakness
  - XSLT Injection Mapped To Performance
- 166 Security Weaknesses Mapped From 2 to 5 Weaknesses
  - Unsafe Reflection Maps to Safety, Reliability, Performance, Bad Practice and Dependability

SAFETY

Buffer Overflow	Integer Overflow	Race Condition: App Download
Buffer Overflow: Format String	JSON Path Manipulation	Race Condition: File System Access
Buffer Overflow: Format String (%f/%F)	LDAP Injection	Race Condition: Signal Handling
Buffer Overflow: Off-by-One	LDAP Manipulation	Resource Injection
Buffer Overflow: Signed Comparison	Log Forging	Setting Manipulation
Code Correctness: Arithmetic Operation on Boolean	Log Forging (debug)	SQL Injection
Code Correctness: Erroneous Synchronization	Memory Leak	SQL Injection: Poor Validation
Code Correctness: Memory Free on Stack Variable	Memory Leak: Reallocation	String Termination Error
Code Correctness: Premature Thread Termination	NoSQL Injection: DynamoDB	Type Mismatch: Integer to Character
Command Injection	NoSQL Injection: MongoDB	Type Mismatch: Negative to Unsigned
Dangerous Function	Null Dereference	Type Mismatch: Signed to Unsigned
Dangerous Function: strcpy()	Often Misused: Exception Handling	Unchecked Return Value
Dead Code	Often Misused: Strings	Undefined Behavior
Dead Code: Expression is Always false	Out-of-Bounds Read	Uninitialized Variable
Dead Code: Expression is Always true	Out-of-Bounds Read: Off-by-One	Unreleased Resource
Denial of Service	Out-of-Bounds Read: Signed Comparison	Unreleased Resource: Synchronization
Double Free	Path Manipulation	Unsafe Reflection
Format String	Path Manipulation: Base Path Overwriting	Use After Free
Format String: Argument Number Mismatch	Poor Style: Variable Never Used	XML External Entity Injection
Format String: Argument Type Mismatch	Process Control	XML Injection
Illegal Pointer Value		XPath Injection

AVAILABILITY

Access Control: Database	Insecure Storage: Insufficient Keychain Protection	Privacy Violation: iOS Property List
ASP.NET Misconfiguration: Persistent Authentication	Insecure Storage: Lacking Data Protection	Privacy Violation: Keyboard Caching
Axis Misconfiguration: Service Enumeration	Insecure Storage: Lacking Keychain Protection	Privacy Violation: Screen Caching
Biometric Authentication: Insufficient Touch ID Protection	Insecure Storage: Passcode Policy Unenforced	Process Control
Cache Management: Headers	Insecure Storage: Unspecified Keychain Access Policy	Process Control: Invoker Servlet
Cache Management: Insecure Policy	Insecure Transport: Secure Section Access Not SSL-Enabled	Race Condition: Roaming Data Access
Cache Management: Language	J2EE Bad Practices: Sockets	Server-Side Script Injection
Cache Management: User-Agent	J2EE Misconfiguration: Insufficient Session ID Length	Server-Side Template Injection
Credential Management: Hardcoded Username	Key Management: Empty Encryption Key	SQL Injection
Cross-Session Contamination	Key Management: Empty PBE Password	System Field Overwrite
Cross-Site Scripting: Inter-Component Communication	Key Management: Hardcoded Encryption Key	System Information Leak: Struts 2
Cross-Site Scripting: Inter-Component Communication (Cloud)	Key Management: Hardcoded PBE Password	Unchecked Return Value
Cross-Site Scripting: Persistent	Key Management: Null Encryption Key	Unreleased Resource
Cross-Site Scripting: Poor Validation	Key Management: Null PBE Password	Unreleased Resource: Android SQLite Database
Cross-Site Scripting: Reflected	Object Injection	Unreleased Resource: Database
Database Bad Practices: Use of Restricted Accounts	Obsolete	Unreleased Resource: Files
Dead Code: Unused Parameter	Obsolete: Deprecated by ESAPI	Unreleased Resource: Streams
Dynamic Code Evaluation: Unsafe Deserialization	Password Management	Unreleased Resource: Unmanaged Object
Expression Language Injection	Password Management: Empty Password	Unsafe JNI
HTML5: CORS Functionality Abuse	Password Management: Hardcoded Password	Unsafe Native Invoke
HTML5: CORS Prolonged Caching of Preflight Response	Password Management: Password in Comment	User or System Dependent Program Flow
HTML5: CORS Unsafe Methods Allowed	PCI Privacy Violation	Weak Cryptographic Hash
Insecure Deployment: Unpatched Application	Privacy Violation	Weak Encryption
Insecure Storage: Externally Available Keychain	Privacy Violation: Android Internal Storage	Web Server Misconfiguration: Weak Authentication

One BIG Surprise From FORTIFY Taxonomy

- The Spreadsheet Showed Significant Overlap Between Safety (MISRA) and Availability
  - More Availability Than Safety
- Safety and Availability Worksheets Color-coded
- Data Transferred to New Worksheet and Stacked with Availability on Top
- Data Sorted Alphabetically and then with Duplicates Removed.
- NO Safety Findings Remained
  - Safety is a subset of Availability
  - No Debate as to whether MISRA is a “Security” standard
- Answered Concern Expressed By Senior Executive

BIGGER SURPRISES

- Automated tools and online scans are unable to complete more than half of the ASVS without human assistance. If comprehensive test automation for each build is required, then a combination of custom unit and integration tests, along with build initiated online scans are used. Business logic flaws and access control testing is only possible using human assistance. These should be turned into unit and integration tests. OWASP Application Security Verification Standard 4.0.3

- DoD Cybersecurity Test and Evaluation Guidebook Starting on page 16, Section 3.7.1.1

  "RMF A&A is necessary but not sufficient to ensure that a system can operate in a cyber-contested environment…it does not test how well the sescurity capabilities in an integrated aspect work in the presence during mission execution."

-

The Bottom Line

Fixing Technical Weaknesses Provides Direct Benefit To The Security of The Code

There is Enough Publicly Available Information To Support The Premise That Software

Security

Is Not Much Different From Software

Quality

BACKUP

Specific Example

- Legacy Electronic Health Record (AHLTA)
- Poor Response Time Resulting In Bad User Experience
  - Multiple Transactions Failing to Meet Key Performance Parameters
- Security Weaknesses (CAT 1) uncorrected from earlier pilot
- Contract to
  - Meet or Exceed KPPs
  - Fix All Cat 1s and Introduce No New Ones
- CAST and Fortify Baseline Scans Provided From My IV&V Team
- Extensive Early Meetings To Decompose CAST Results

STEP 1

- Very Early Use of CAST – 2014
- Calculate the Percentage of Violations Against The Total Number of Rules Assessed
  - 1392970 Rules Assessed
  - 80638 Violations
  - 5.7%
- Calculate The Number of Clean Rules
  - 145 Distinct Rules
  - 10 Rules with 1 Finding
- Later Scans Had Several Rules With 0 Findings

STEP 2 Health Factors Example

Criterion	Total KO (Failures)
Architectural Design	3725
Changeability	10204
Documentation	23084
Performance	285
Programming Practices	5636
Robustness	6462
Security	1381
Technical Quality Index	32752
Transferability	27498

Step 2 Continued

- Color Code Health Factors in Spreadsheet
  - Security Always RED
- Rack and Stack With Security On The Bottom Of The Stack
- SORT Alphabetically And Observe Duplicate Entries
- SORT Again To Remove Duplicates
  - Except for 2 Encryption Findings In One Project, Security (Red) Disappeared In Every Report

SECURITY

Avoid using Inner Classes	258	Avoid using 'System.printStackTrace()' within a try catch block	7	Avoid using native Methods (JNI)	0
Avoid using fields (non Static final) from other Classes	182	Avoid using 'System.printStackTrace()' outside a try catch block	6	Use only Hibernate API to access to the database	0
Avoid declaring and throwing an exception and not throwing it	130	Avoid missing default in switch statements	5	Avoid direct Class inheritance from java.lang.Throwable	0
Avoid declaring Instance Variables without defined access type	112	Avoid using Hashtable	4	Avoid using finalize()	0
Avoid public/protected setter for the generated identifier field	88	Avoid using Vector	4	Collection-typed attributes getter must be defined with the correct interface	0
Avoid catching an exception of type Exception, RuntimeException, or Throwable	66	Avoid to use this within constructor in multi-thread environment	3	Never exit a finally block with a return, break, continue, or throw	0
Declare as Static all methods not using instance fields	63	Avoid empty finally blocks	3	Track Classes referencing database objects	0
Avoid using references to the id in the persistent class's method equals()	62	Avoid to use Log.debug() without calling Log.isDebugEnabled()	3	Avoid many-to-many association	0
Avoid large number of String concatenation	57	Persistent classes should Implement hashCode() and equals()	2	Avoid catch blocks with assertion	0
Avoid non serializable Entities	54	Avoid cyclical calls and inheritances between packages	2	Avoid String initialization with String object (created using the 'new' keyword)	0
Avoid declaring Inner Classes	53	Avoid directly instantiating a class used as a Spring bean	2	Avoid using 'java.lang.Runtime.exec()'	0
Avoid instanceof in methods that override or implement Object.equals(), Comparable.compareTo()	45	Avoid thread creation for application running on application server	2	Avoid table and column names that are too long (portability)	0
Avoid throwing an exception in a catch block without chaining it	45	Avoid double checked locking	2	Avoid using 'java.lang.System.getenv()'	0
All types of a serializable class must be serializable	42	The exception Exception should never been thrown. Always Subclass Exception and throw the Subclassed Classes.	2	Avoid using 'System.gc'	0
Avoid fields in servlet classes that are not final static	16	Avoid using 'java.lang.Error'	2	Avoid using DriverManager	0
Avoid using deprecated objects	12	Avoid using 'sun.*' Classes	2	Avoid Classes with a High Public Data Ratio	0
Avoid declaring Non Final Class Variables with Public or Package access type	11	Avoid using incorrect XML parsing model	2	Avoid using 'java.System.exit()'	0
Persistent class method's equals() and hashCode() must access its fields through getter methods	10	Avoid non thread safe singleton	1	Avoid instantiating Boolean	0
Avoid declaring Final Instance Variables that are not dynamically initialized	7	Avoid empty catch blocks	0	Avoid too many packages referencing Mainframe	0
Avoid static field of type collection	7	Pages should use error handling page	0	Avoid using 'System.err' and 'System.out' outside a try catch block	0
Avoid using 'java.io.File'	7	Avoid database tables associated to more than one Entity	0	Avoid using 'System.err' and 'System.out' within a try catch block	0

PERFORMANCE

Avoid instantiations inside loops	72		Avoid String concatenation in loops	1
Declare as Static all methods not using instance fields	63		Avoid Artifacts with a Complex SELECT Clause	1
Avoid large number of String concatenation	57		Use lazy fetching for collection	0
Avoid indirect String concatenation inside loops	28		Avoid using SQL queries inside a loop	0
Avoid the use of InstanceOf inside loops	20		Avoid Artifacts with Group By	0
Avoid method invocation in a loop termination expression	13		Avoid Artifacts with High Depth of Nested Subqueries	0
Avoid static field of type collection	7		Avoid select-before-update when the table is not associated to an UPDATE trigger	0
Avoid using Dynamic instantiation	4		Never use array to map a collection	0
Avoid using Hashtable	4		Avoid SELECT *" queries"	0
Avoid using Vector	4		Avoid using finalize()	0
Avoid direct or indirect remote calls inside a loop	3		Collection must be the same between getter and setter	0
Avoid to use Log.debug() without calling Log.isDebugEnabled()	3		Avoid Artifacts with High RAW SQL Complexity	0
Avoid using incorrect XML parsing model	2		Avoid String initialization with String object (created using the 'new' keyword)	0
Avoid Artifacts with queries on more than 4 Tables	1		Avoid using 'System.gc'	0
Avoid Artifacts with SQL statement including subqueries	1		Avoid using DriverManager	0
Prefer UNION ALL to UNION	1		Avoid instantiating Boolean	0

Accomplishments

- CAST Results Used To Prioritize Technical Work
- FORTIFY Used To Identify CAT I Issues
- At End of 1 Year POP:
  - All Prior Cat 1 Weaknesses Corrected and No New Ones Introduced
  - All KPPs met or exceeded in test lab.
- Limitations
  - Could Not Fix The Serious Internal Architecture Flaws
  - Not Enough Time Nor Money To Fix Everything Else In One Year

Topic 3

Open Discussion

Chris Coffin

Miscellaneous Topics

- **Other thoughts or topics?**

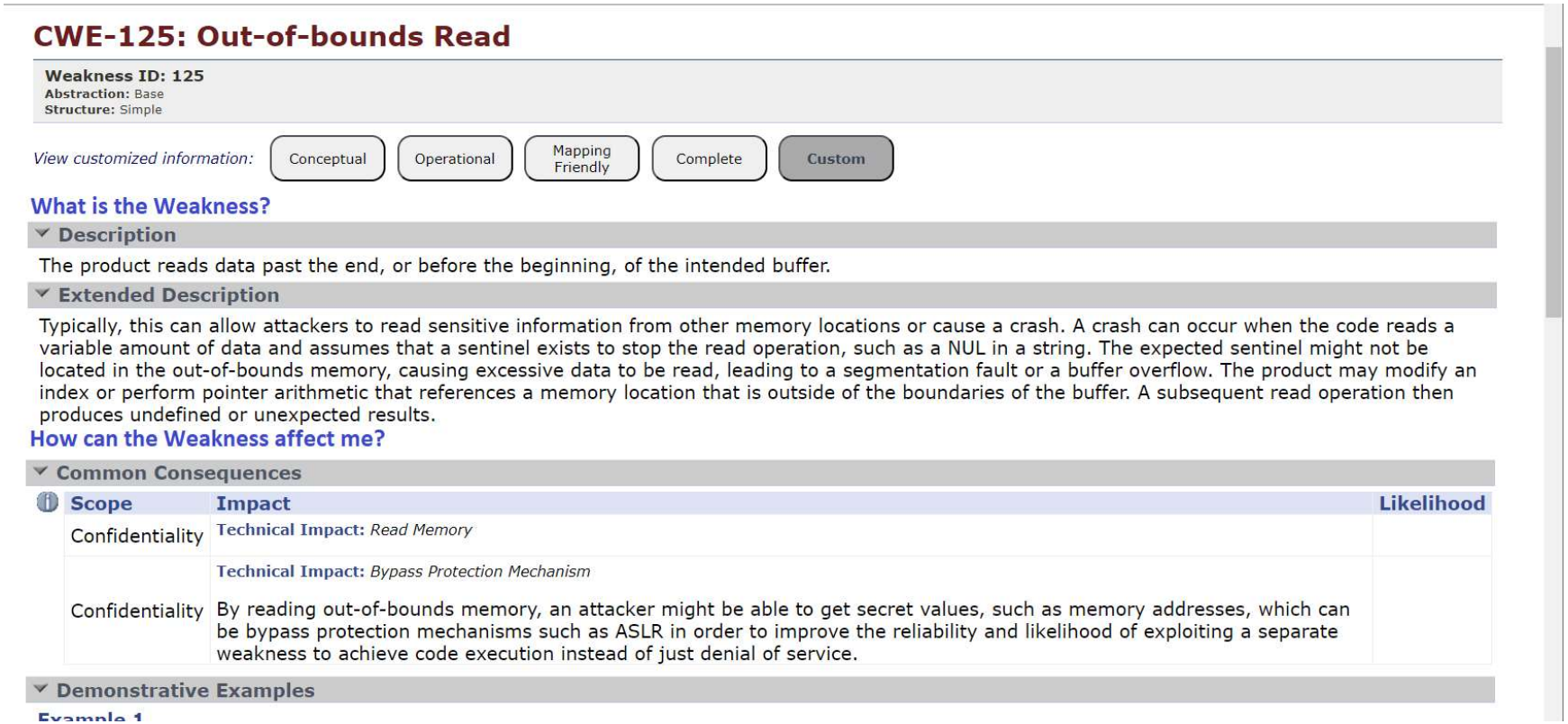
Next Meeting – November 29 @ 12pm

PLEASE CONTACT WITH ANY QUESTIONS OR THOUGHTS

CWE@MITRE.ORG

Backups

CWE Element/Information Presentation – Mockup for New CWE Users



CWE-125: Out-of-bounds Read

Weakness ID: 125
Abstraction: Base
Structure: Simple

View customized information: Conceptual | Operational | Mapping Friendly | Complete | Custom

What is the Weakness?

Description

The product reads data past the end, or before the beginning, of the intended buffer.

Extended Description

Typically, this can allow attackers to read sensitive information from other memory locations or cause a crash. A crash can occur when the code reads a variable amount of data and assumes that a sentinel exists to stop the read operation, such as a NUL in a string. The expected sentinel might not be located in the out-of-bounds memory, causing excessive data to be read, leading to a segmentation fault or a buffer overflow. The product may modify an index or perform pointer arithmetic that references a memory location that is outside of the boundaries of the buffer. A subsequent read operation then produces undefined or unexpected results.

How can the Weakness affect me?

Common Consequences

Scope	Impact	Likelihood
Confidentiality	**Technical Impact:** *Read Memory*	
Confidentiality	**Technical Impact:** *Bypass Protection Mechanism* By reading out-of-bounds memory, an attacker might be able to get secret values, such as memory addresses, which can be bypass protection mechanisms such as ASLR in order to improve the reliability and likelihood of exploiting a separate weakness to achieve code execution instead of just denial of service.	

Demonstrative Examples

Example 1

CWE Element/Information Presentation – Mockup for New CWE Users

index or perform pointer arithmetic that references a memory location that is outside of the boundaries of the buffer. A subsequent read operation then produces undefined or unexpected results.

How does this Weakness relate to others?

Relationships

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer
ParentOf	V	126	Buffer Over-read
ParentOf	V	127	Buffer Under-read
CanFollow	B	822	Untrusted Pointer Dereference
CanFollow	B	823	Use of Out-of-range Pointer Offset
CanFollow	B	824	Access of Uninitialized Pointer
CanFollow	B	825	Expired Pointer Dereference

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name
MemberOf	C	1218	Memory Buffer Errors

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

Relevant to the view "CISQ Quality Measures (2020)" (CWE-1305)

Relevant to the view "CISQ Data Protection Measures" (CWE-1340)

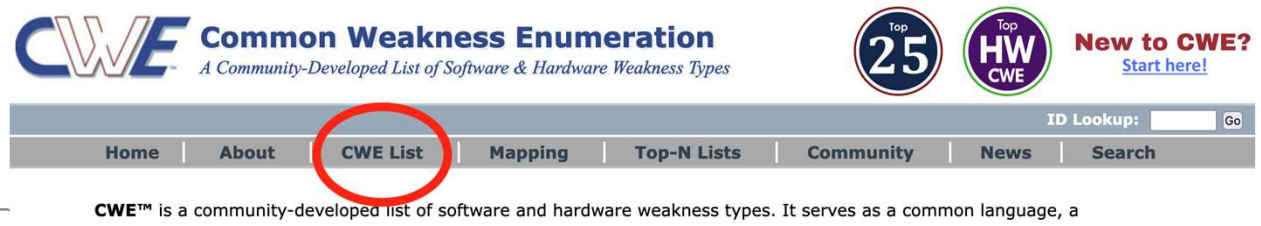
Where can I get more information?

References

[REF-1034] Raoul Strackx, Yves Younan, Pieter Philippaerts, Frank Piessens, Sven Lachmund and Thomas Walter. "Breaking the memory secrecy assumption". ACM. 2009-03-31. <https://dl.acm.org/doi/10.1145/1519144.1519145>. *URL validated: 2023-04-07.*

[REF-1035] Fermin J. Serna. "The info leak era on software exploitation". 2012-07-25. <https://media.blackhat.com/bh-us-

Grouping CWEs



- **CWE entries are currently "grouped" in different ways to provide useful subsets of the CWE corpus for different purposes:**
  - Views: a subset of CWE entries that provides a way of examining CWE content. The two main view structures are Slices (flat lists) and Graphs (containing relationships between entries), examples include:
    - CWE-1194: Hardware Design
    - CWE-699: Software Development
    - CWE-1400: Comprehensive Categorization for Software Assurance Trends
    - CWE-1003: Weaknesses for Simplified Mapping of Published Vulnerabilities (NVD)
  - Categories: a CWE entry that contains a set of other entries that share a common characteristic. A category is not a weakness, but rather a structural item that helps users find weaknesses that share the stated common characteristic.
    - CWE-1199: General Circuit and Logic Design Concerns
  - ~ Overall Hierarchy
    - CWE-1000: Research Concepts contains all CWE entries in one hierarchical structure

Grouping CWEs, cont.

- What groupings are most useful to new or casual CWE users? Experienced users?
- How can groupings be better presented/discovered/identified to the user?
- Should new users be guided to groupings of CWEs for learning about CWE? (e.g., links in user stories)
- Are there additional groupings that we are missing? Too many?

CSV Single Colon Separators Within Column Data

- **A double colon is used to separate csv fields/columns data, while a single colon is used to separate multi-value data within fields/columns**
- **The Observed Examples field contains Reference and link data that includes a url in some cases (colon within "http://...")**
- **Should a note be added to the download data that warns the user of this, or should we look into an alternative separator?**
- **Example taken from CWE - CWE-41: Improper Resolution of Path Equivalence (4.12) (mitre.org)**
  - ::REFERENCE:CVE-2000-1114:DESCRIPTION:Source code disclosure using trailing dot:LINK:https://www.cve.org/CVERecord?id=CVE-2000-1114::REFERENCE:CVE-2002-1986:DESCRIPTION:Source code disclosure using trailing

CWE User Pain Points

- **Pain point topics that the group is aware of or would like to discuss**
- **For those on the call, what were your biggest questions or concerns when beginning to use CWE?**
- **Are there common questions that CWE users have that are not covered in the current FAQ?**

- **Other potential opportunities:**
  - Features we could expand or improve to make CWE consumption easier?
  - Maybe engage the community in one or more ways to solicit this kind of feedback (see topic #3)

- **Other thoughts?**

Community Engagement Strategy

- **Develop a strategy for engaging the CWE user community for feedback**
- **What are the best methods to query the community on topics such as the pain points covered in topic #2**
- **What communication methods should be employed?**
  - E.g., polls, emails, web, social media
- **Should we target specific user types?**

- **Other thoughts?**

CWE Video Tips Series

- **Current video ideas:**
  - How to search CWE for a weakness
  - How to display only the information that you need with presentation filters
  - What is a weakness (vs a vulnerability)
  - How are weaknesses organized
  - What is a category (how is it different than a pillar)
  - What are views
  - How and why to use the research view
  - Use cases for CWE (could user stories be used?)
  - How do I submit an idea for a new weakness
  - How can I improve the quality of existing weaknesses

New to CWE – Future Content

- **The New to CWE content audience is different from what has been catered to previously**
- **The audience is the casual or new user to CWE or even the manager who makes security funding decisions**
- **The team has previously drafted material for the New to CWE audience that covers the CWE hierarchy**
  - Not yet released material
  - Do members agree that this topic should be covered for New to CWE?
- **Are there other topics that UEWG members feel strongly about or believe should be covered given the intended audience?**
- **Should there be a close coupling of the topics covered here with the CWE Video Tips series?**

CWE Naming and Vulnerability Mapping

- **Being thinking about solutions for common and well-known issues surrounding use of CWE names and how to more easily map vulnerabilities to CWEs**
- **Current CWE structure is difficult to understand and use**
- **Community needs better root cause information for vulnerabilities**
- **Does CWE naming need a change or update to support easier mapping?**
  - Remove CWE names for Views and/or Categories?
  - New naming that embeds a structure (e.g., CWE-1234-1)