Reconstruction of Common Weakness Enumeration

Yan Wu
University of Nebraska at Omaha

Irena Bojanova
University of Maryland University College

Yaacov Yesha
University of Maryland, Baltimore County

April 2, 2014
STC 2014

Agenda

* Common Weakness Enumeration (CWE) [1]
* Semantic Template [2]
* Software Fault Pattern (SFP) [3]
* Directions for CWE formalization

Common Weakness Enumeration (CWE)

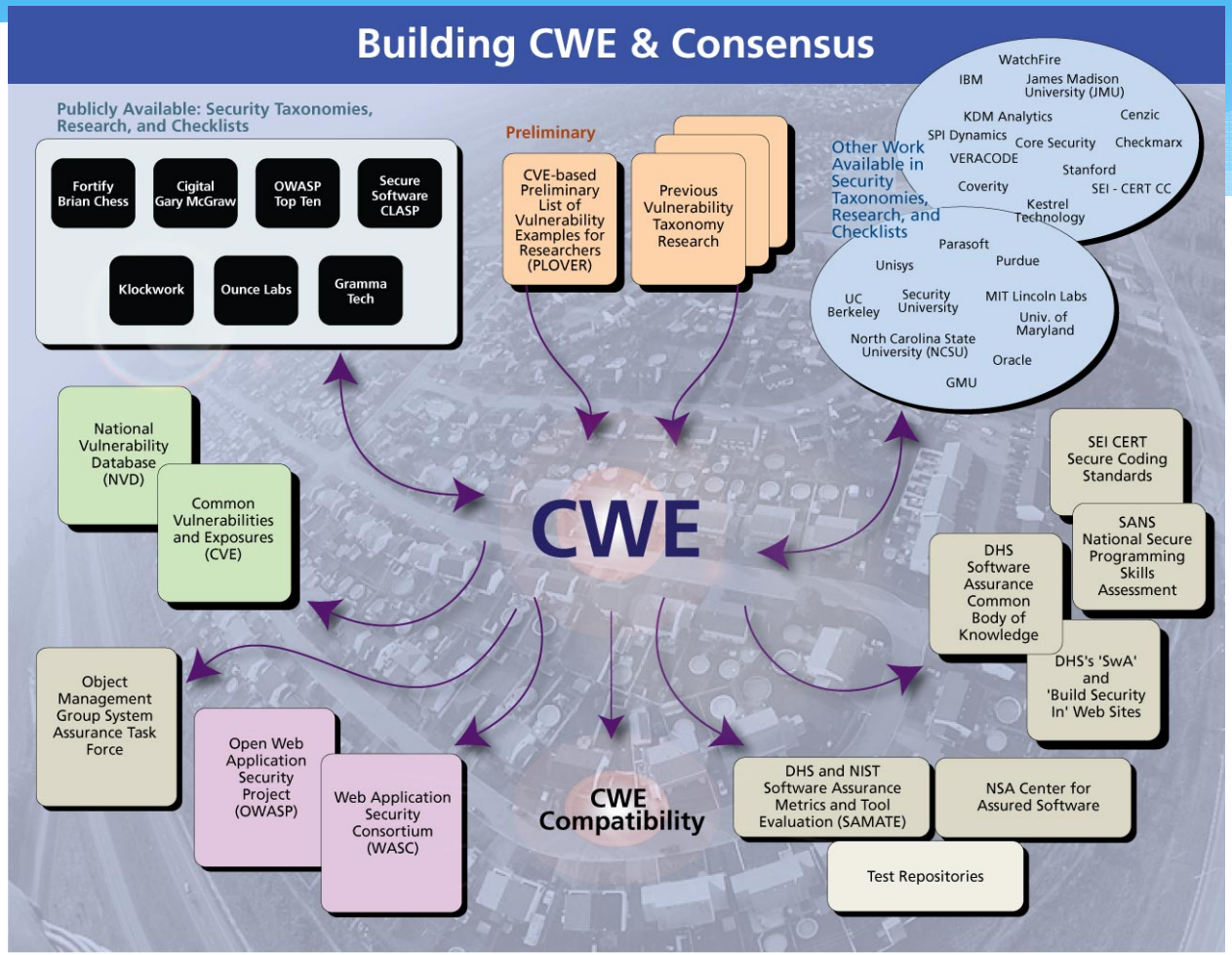
* Know what makes your software vulnerable to attacks
* Software – should be free of known weaknesses that compromise security
* What is meant by software having no known weaknesses?
* How to evaluate tools and services for finding weaknesses?

→Need of repository of software weakness types

CWE – collection of software weakness types

- Each software weakness has a name and a number
- Each type has a definition and additional information

CWE Efforts Context and Community



http://cwe.mitre.org/about/images/lg_consensus.jpg

Use of CWE

* CWE – for use by those who:
  * Create software
  * Analyze software for security flaws
  * Provide tools & services for finding & defending against security flaws in software
* CWE Compatibility and Effectiveness Program :
  1) CWE Searchable
  2) CWE Output
  3) Mapping Accuracy
  4) CWE Documentation
  5) CWE Coverage
  6) CWE Test Results
* Designations for products or services:
  - CWE Compatible – meet 1) to 4)
  - CWE Effective – meet all 1) to 6)

Examples of CWEs

CWEs

include flaws, faults, bugs, vulnerabilities, and other errors in software code, design, architecture, or implementation

Examples of CWEs:

* Buffer overflows
* Structure and validity problems
* Common special element manipulations
* User interface errors
* Pathname traversal and equivalence errors
* Authentication errors
* Resource management errors
* Code evaluation and injection

Structure of CWE

Major types of CWE-IDs:

1) Category -- aggregates by types of weaknesses
   Ex: CWE-355: User Interface Security Issues
2) Compound Element – aggregates by group of events
   Ex: CWE-476: NULL Pointer Dereference
3) View – predefined perspectives
   Ex: CWE-1000: Research Concepts
4) Weakness – the covered weakness
   Ex: CWE-311: Failure to Encrypt Sensitive Data

Structure of CWE (cont.)

Each CWE entry includes :

* CWE Identifier Number/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 Identifier numbers of vulnerabilities for which that type of weakness exists
* References

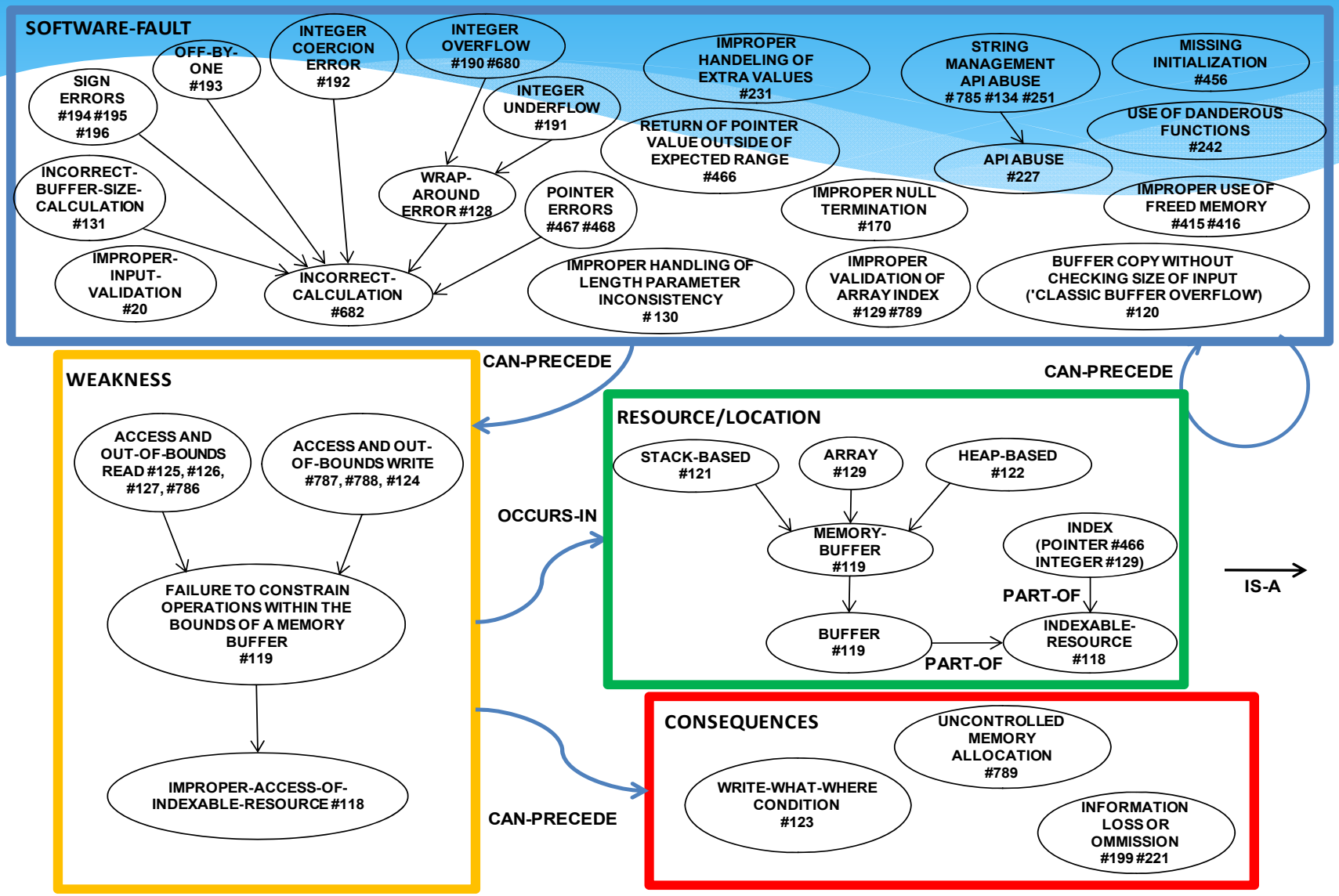
Why Reconstruction?

* Overlap, missing information, huge volume of CWE specification document
* Natural language descriptions lead to misunderstandings
* Information overload of vulnerability information in software repositories and, paradoxically
* Lack of techniques to systematically annotate and analyze this data.

How to Reconstruct

* Semantic Template
  * Classify
  * Extract commonalities
  * Reorganize information into four groups
  * Support by vulnerabilities and feedback to annotate vulnerabilities
* Software Fault Pattern
  * Classify
  * Identify patterns
  * Test cases generator
* Formalization of CWEs

Semantic Template



Benefits of Reconstruction by Semantic Template

* Provide a framework for constructing a richer context around existing vulnerability information
* Easier to understand and work with than original CWE
* A repeatable and intuitive schema for the representation and management of vulnerability information
* Provide actionable metrics and measures
* Semi-automated annotation of vulnerabilities

Software Fault Pattern (SFP)

* Focus on computation
* Support automation
* Enable mathematical reasoning
* Ensure systematic coverage

Software Fault Pattern (SFP)

* A SPF is a common pattern with one or more associated pattern rules, representing a family of faulty computations.
* Organized by the primary SFP definition which refers to the secondary cluster and is arranged into invariant core and variation points
* Achieved:
  * 21 primary clusters
  * 62 secondary clusters
  * 36 unique SPFs
  * 632 CWEs that have been categorized
  * 310 are identified as discernible CWEs.

CWE Formalization

* The CWE formalization work is joint work with Paul E. Black

CWE Formalization

* CWE formalization is needed for precise determination whether or not the software in question has the weakness.
* In addition to being precise, it has to be accurate, so that it will faithfully represent the meaning intended by the community.
* Three CWEs were selected formalization, in order to have examples of the issues involved.

CWE Formalization - Existing Definitions

The 3 CWE selected and their description summaries (from CWE Common Weakness Enumeration https://cwe.mitre.org/):

* CWE-307: Improper Restriction of Excessive Authentication Attempts

" The software does not implement sufficient measures to prevent multiple failed authentication attempts within in a short time frame, making it more susceptible to brute force attacks."

* CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer

"The software performs operations on a memory buffer, but it can read from or write to a memory location that is outside of the intended boundary of the buffer."

CWE Formalization - Existing Definitions

* CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

” The software constructs all or part of an OS command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended OS command when it is sent to a downstream component.”

CWE Formalization
- Our Formalization of CWE-307

* CWE-307 Improper Restriction of Excessive Authentication Attempts
* Our definition: "The software does not limit the number of failed authentication attempts or may allow more than a specified number of failed authentication attempts within a specified time period."
* Analysis: Our formal definition provides a precise meaning to the terms "sufficient measures" and "more susceptible to brute force attacks." in the existing definition.

CWE Formalization
- Our Formalization of CWE-119

* CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer
* Our definition: "The software can access through a buffer a memory location not allocated to that buffer."
* Analysis: Our definition gives a precise meaning to the term "intended boundary" in the existing definition. This meaning is determined by the semantics of the programming language.

CWE Formalization
- Our Formalization of CWE-78

* CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
* Our definition: “There exist inputs having the same trusted part that result in code symbols in the output program that differ in a way that is not included in a description of a given syntax of allowed different sequences of code symbols.”

CWE Formalization
- Our Formalization of CWE-78

* Analysis: We obtained our definition by substantially modifying one of the definitions of injection weakness in a paper by by Ray and Ligatti [4], that also defined the notion of code symbols in the output program. We introduced our modification because, as observed in that paper by Ray and Ligatti, that definition results in cases of software without an injection weakness being considered having that weakness.

CWE Formalization
- Our Formalization of CWE-78

* That definition of injection weakness in that paper by Ray and Ligatti is allowing variation in code symbols in the output for inputs having the same trusted part.
* Our definition differs from their definition by including the notion of a syntax that describes what influence the untrusted part of the input is allowed to have on code in the software output

References

[1] CWE Common Weakness Enumeration https://cwe.mitre.org/

[2] Y. Wu, R. A. Gandhi, and H. Siy. "Using semantic templates to study vulnerabilities recorded in large software repositories." 2010 ICSE Workshop on Software Engineering for Secure Systems, SESS '10, pages 22-28, New York, NY, USA, 2010. ACM.

[3] N. Mansourov and D. Campara. System Assurance Beyond Detecting Vulnerabilities A volume in The MK/OMG Press, Elsevier Inc., 2011, 7.3 Software fault pattern.

[4] Donald Ray and Jay Ligatti, Defining code-injection attacks Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '12), Pages 179-190.