Formalizing Software Bugs

Irena Bojanova
UMUC, NIST

12/08/2014

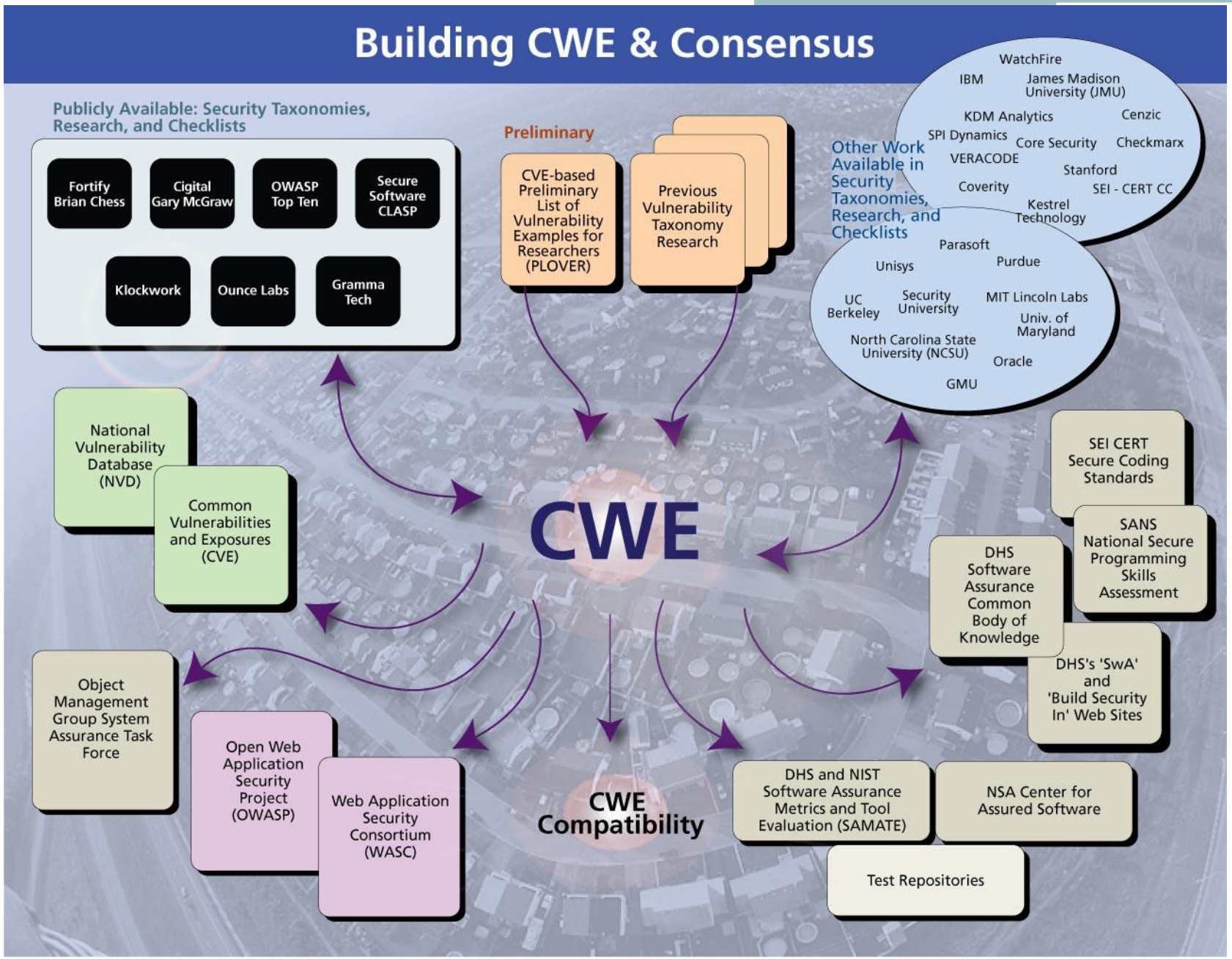
Software Bugs

- "Software bug" – a concept that applies to:
  - CWE – Common Weakness Enumeration
  - CVE – Common Vulnerabilities and Exposures
  - CAPEC – Common Attack Pattern Enumeration and Classification.

- Related, but different:
  - Weakness – a static presence in software – might never cause problems;
  - only when an attacker finds the weakness and exploits it
  - the vulnerability of this software is exposed.

- CWE & CVE + CAPEC:
  - Considerable efforts, providing foundational knowledge
  - However, not sufficient, accurate, and precise enough.

Building CWE & Consensus



Problems with CWE, CVE, & CAPEC

- CWE:
  - Not orthogonal, just an enumeration
  - Ambiguous definitions
  - Do not match well with classes reported by test tools.

Example: 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."

Problems CWE, CVE, & CAPEC (cont.)

- CVE:
  - Ambiguous definitions
  - Do not fully and precisely describe involved CWEs.

Example: CVE-160: Heartbleed.

"The (1) TLS and (2) DTLS implementations in OpenSSL 1.0.1 before 1.0.1g do not properly handle Heartbeat Extension packets, which allows remote attackers to obtain sensitive information from process memory via crafted packets that trigger a buffer over-read, as demonstrated by reading private keys, related to d1_both.c and t1_lib.c, aka the Heartbleed bug."

"Technical Details: Buffer Errors (CWE-119)"

→ Actual exploited is CWE-126, while only parent CWE-119 is listed.

→ CWE-130 is in CWE-126, but CWE-20 also can precede CWE-126.

Problems CWE, CVE, & CAPEC (cont.)

- CAPEC:
  - Do not fully show dynamics of activities leading to realization of attack
  - CVE, CWE, CAPEC inks could be confusing.

Example: Heartbleed – which CAPEC and CWEs?
From attack to weakness:

- Heartbleed exploits Buffer Overread (CWE-126)
  -> attack should be CAPEC 540: Overread Buffers
- CAPEC 540, “Related Weakness” is CWE-125, parent of CWE-126, lists CVE-160 (Heartbleed).

Form weakness to attack:

- CVE-160 (Heartbleed), “Technical Details”— CWE-119.
- CWE-119, “Related Attack Patterns” – no CAPEC 540, closest CAPEC-47.

→ It is a fact that CVE-160 is listed in CAPEC 540, but not in CAPEC-47.

Need For

- Precise descriptions of attacks (CAPECs) that lead to realization of vulnerabilities (CVEs), exposed by software weaknesses (CWEs).

- Research to explore formalization of CWEs, CVEs, & CAPECs.

CAPEC 540 – CAPEC 47

CAPEC-540: Overread Buffers

“An adversary attacks a target by providing input that causes an application to read beyond the boundary of a defined buffer. This typically occurs when a value influencing where to start or stop reading is set to reflect positions outside of the valid memory location of the buffer. This type of attack may result in exposure of sensitive information, a system crash, or arbitrary code execution.”

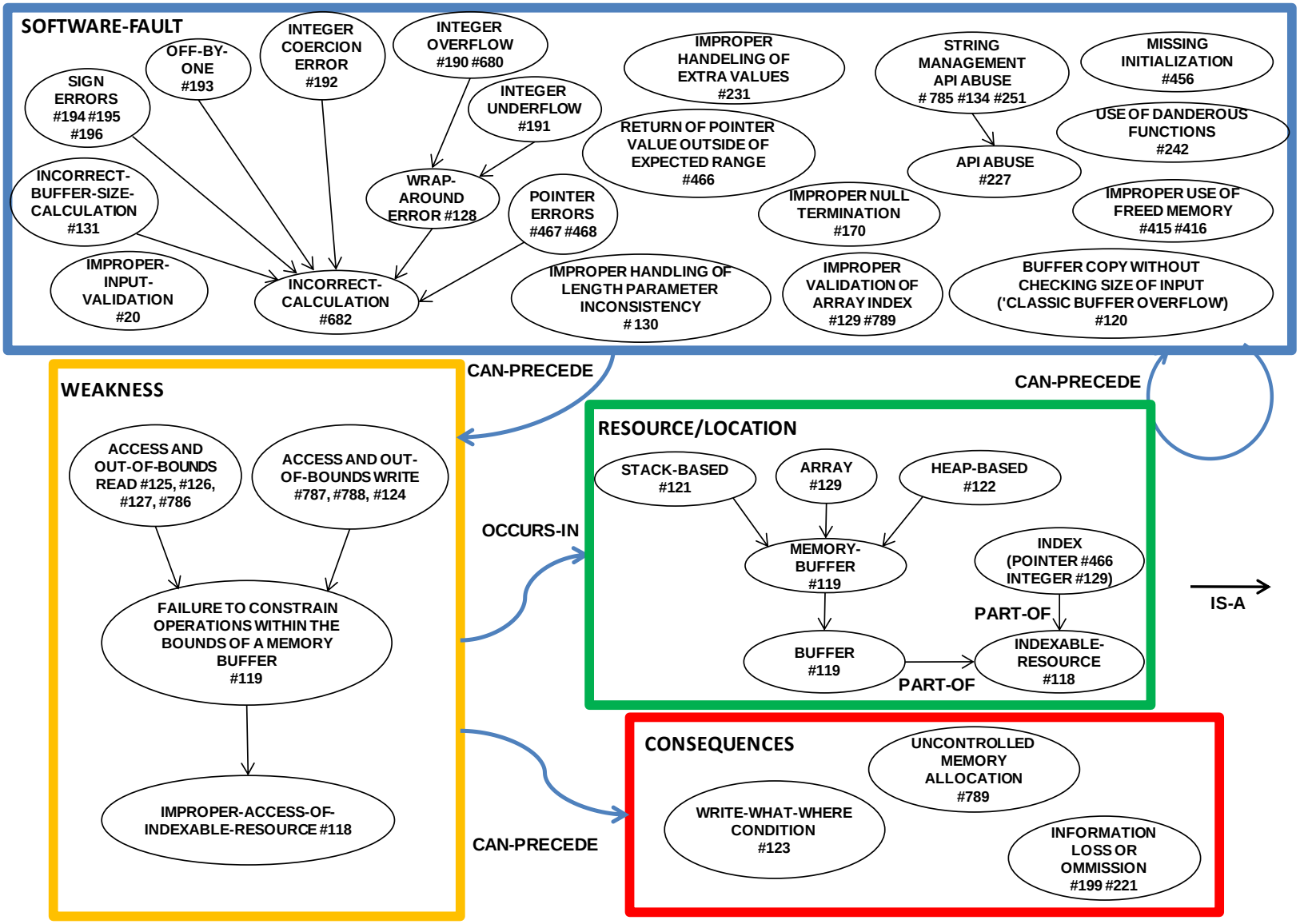
CAPEC-47: Buffer Overflow via Parameter Expansion

“In this attack, the target software is given input that the attacker knows will be modified and expanded in size during processing. This attack relies on the target software failing to anticipate that the expanded data may exceed some internal limit, thereby creating a buffer overflow.”

Previous Related Work

- Semantic Templates – Yan Wu's Dissertation
  - Classify
  - Extract commonalities
  - Reorganize information into four groups:
    - Software faults that lead to a weakness
    - 2) Resources that a weakness affects
    - 3) Weakness characteristics
    - 4) Consequences/failures resulting from the weakness.
  - Support by vulnerabilities and feedback to annotate vulnerabilities

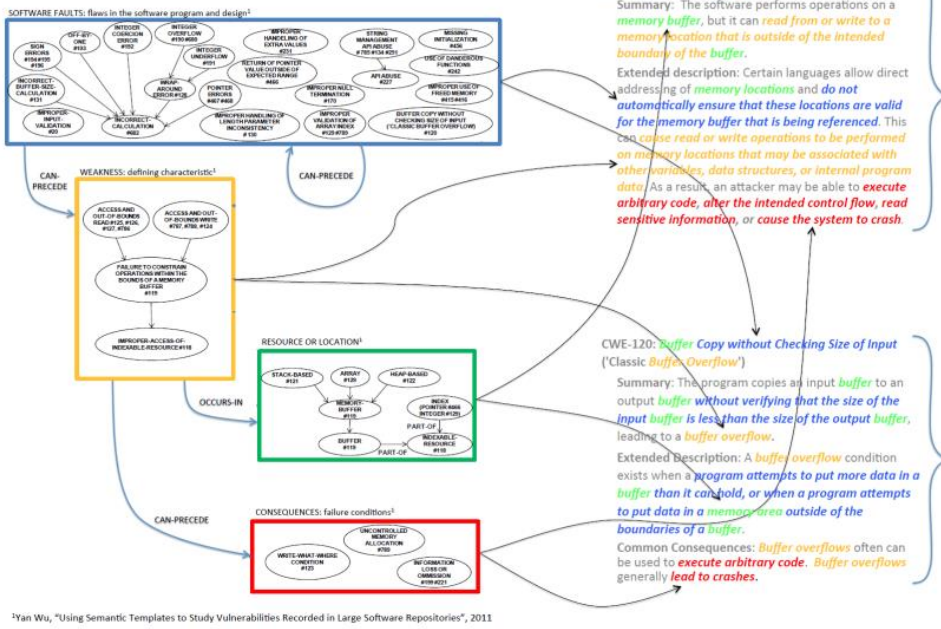
- Software Fault Patterns – KDM Analytics
  - Classify
  - Identify patterns
  - Test cases generator.



Buffer Overflow Semantic Template

Previous Related Work

Semantic Template



[1]Yan Wu, "Using Semantic Templates to Study Vulnerabilities Recorded in Large Software Repositories", 2011

Software Fault Pattern

Parameters	Buffer location		Access kind		Access position in relation to the buffer		Boundary exceeded	
	heap	stack	write	read	inside	outside	lower	upper
119 - Improper Restriction of Operations within the Bounds of a Memory Buffer	√	√	√	√		√	√	√
120 - Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	√	√	√		√		√	√
121 - Stack Overflow		√	√		√		√	√
122 - Heap Overflow	√		√		√		√	√
123 - Write-what-where Condition	√	√	√				√	√
124 - Buffer Underwrite ('Buffer Underflow')	√	√	√			√	√	
125 - Out-of-bounds read	√	√		√			√	√
126 - Buffer Over-read	√	√		√		√		√
127 - Buffer Under-read	√	√		√		√	√	

Note: chart is adapted

Tried to formally describe :

- Vulnerabilities as chains of exploited weaknesses
- Involved CWEs utilizing the corresponding SFPs.

Failed midway – realized need to specify also the dynamic aspect of the attack.

Formalization of CWE, CVE & CAPEC

- Formalization of CWEs will:
  - Clarify their definitions
  - Remove ambiguities among them.

  => As a result, reclassification of CWEs will be possible.
- Same reasoning applies to CVEs and CAPECs formalization.

- Important:
  - Base of CVEs are CWEs.
    Each CVE is a result of CAPEC(s) exploiting software CWEs.

  - Each CWE is a static software property.
  - Each CVE is a dynamic software property that occurs as a result of some CAPEC (s).

Considered Notations

- Static software properties: formal software specification models, languages, notations, and tools.
  - Z notation - standardized by ISO.
- Dynamic software properties – those used for process specification: process algebra, Petri nets, and state machines.
  - CSP – Communicating Sequential Processes
  - UML & OCL – Unified Modeling Language with Object Constraint Language.

- Z notation and CSP – strongly based on the mathematical notation without introducing strange constructions
- UML – provides good visual representation.

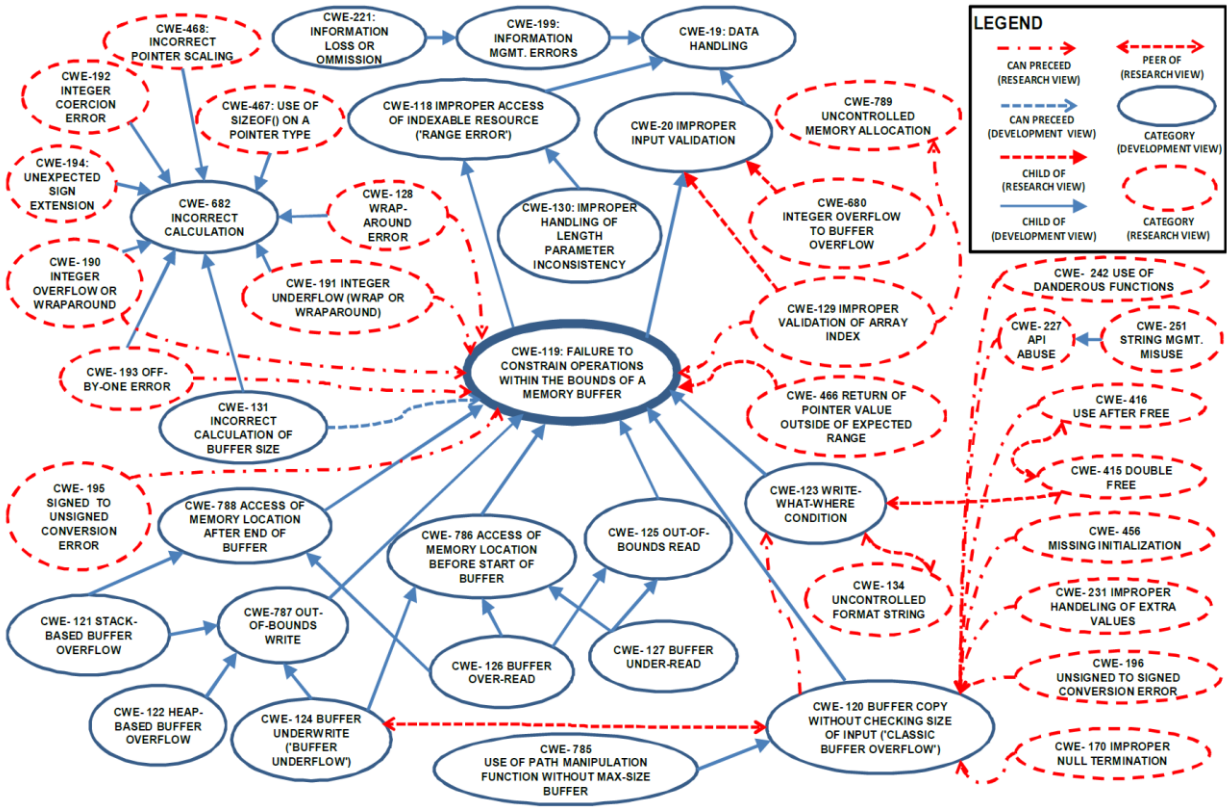
CWEs “Mental Models” (Step 1)

- CWE & CVE databases – represented in XML
- CWEs & CVEs text presentations – verbose and hardly navigable.

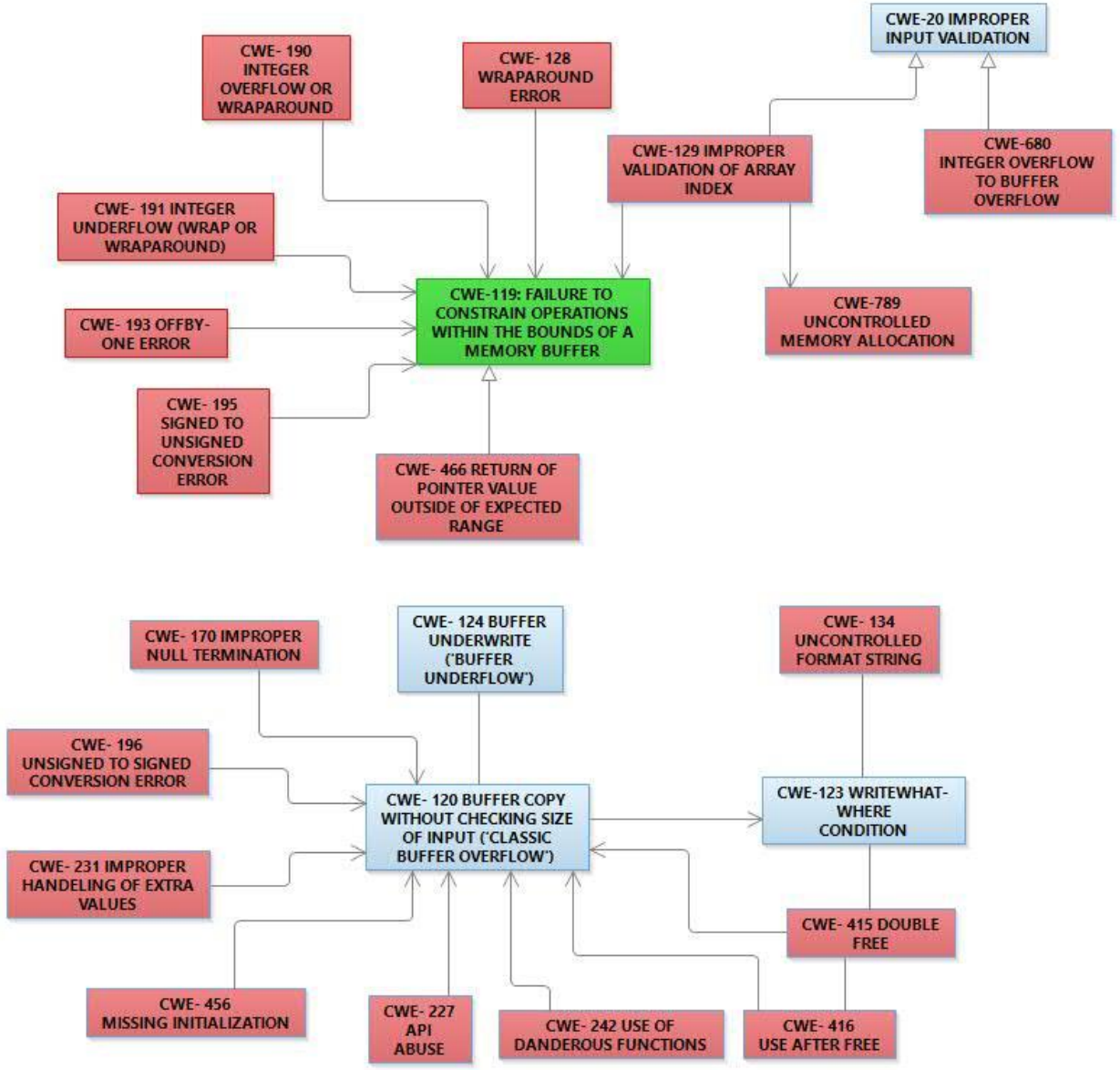
First Research Step:

- Formal graph representation of these databases
- “Mental models” of base CWEs – UML class diagrams:
  - CWEs –classes
  - Relationships between CWEs – associations

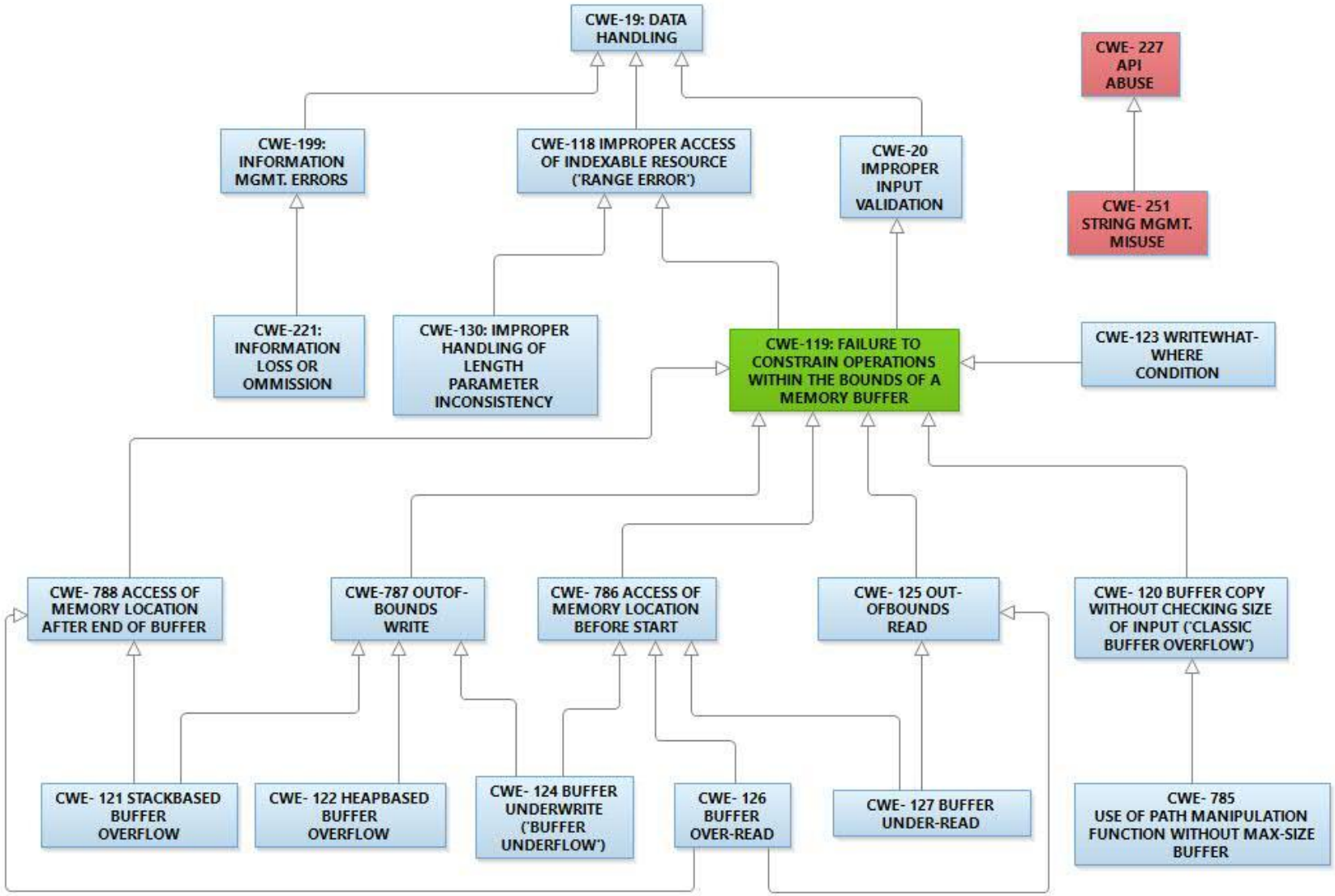
➔Later incorporate an OCL specification for each CWE.



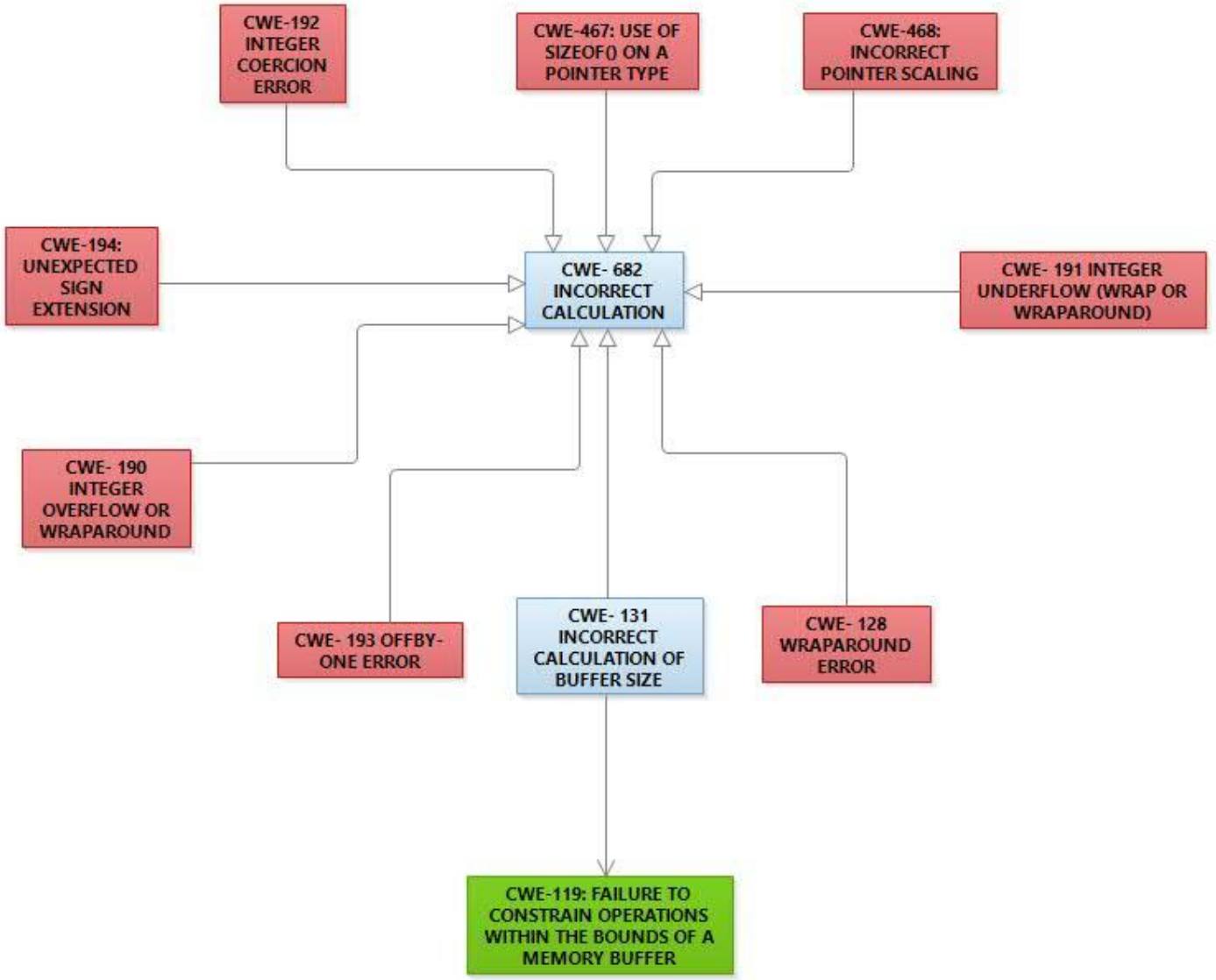
Buffer Overflow-related CWEs Extracted in the Preparation and Collection Phase (Figure 5 from Yan Wu's dissertation)



CWE-119 Research View – Inheritance and Precedence Relationships

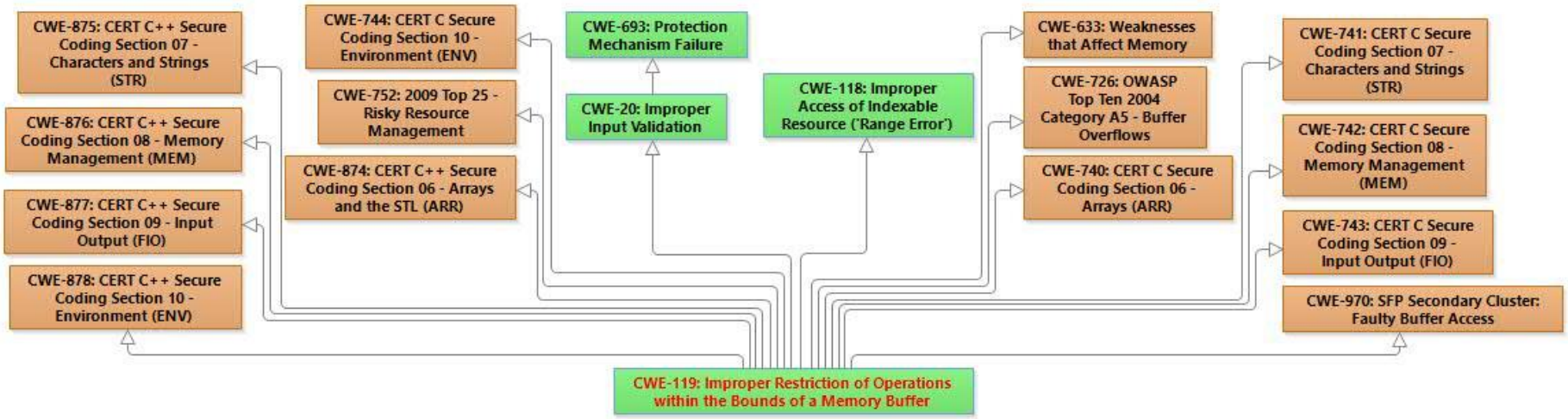


CWE-119 Development View – Inheritance Relationships



CWE-119 Development View – Precedence Relationships

CWE-119 Mental Model



CWE-119 Relationships - not complete yet
(extracted from CWE database XML representation)

CWEs Formal Specification (Step 2)

Second Research Step :

- CWEs formalization in Z notation
  → will remove ambiguities in their definitions
  → can lead to CWEs reclassification.
- Conversion of these Z notations in OCL specifications.

- UML representations – could be source to industry tools for code verification and tests generation (RSA, MS Visio).
- However, humans will not be able to use efficiently OCL, because of its complexity.
- Formalization in Z notation – open to theorem solvers and eventually to analyzers (e.g Alloy Analyzer).
- Z Word can be used for specification purposes and Z/Eves for verification.

CWE-128 in Z notation

CWE-128: Wrap-around Error: “Wrap around errors occur whenever a value is incremented past the maximum value for its type and therefore "wraps around" to a very small, negative, or undefined value.”

$$MAX\_INT : \mathbb{Z}$$
$$MIN\_INT : \mathbb{Z}$$

$$INT == \{i : \mathbb{Z} \mid MIN\_INT \leq i \land i \leq MAX\_INT\}$$

$$BAD\_INT : \mathbb{Z}$$

$$BAD\_INT < MIN\_INT \lor MAX\_INT < BAD\_INT$$

$$add, mul : INT \times INT \rightarrow INT \cup \{BAD\_INT\}$$

$$\forall i, j : INT \bullet add(i, j) = \textbf{if } i+j > MAX\_INT \textbf{ then } BAD\_INT \textbf{ else } i+j$$
$$\forall i, j : INT \bullet mul(i, j) = \textbf{if } i*j > MAX\_INT \textbf{ then } BAD\_INT \textbf{ else } i*j$$

CWE-119 in Z notation

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."

$$[Byte]$$

$$maxAddress : \mathbb{N}$$

Memory

$$contents : \mathrm{seq}\ Byte$$

$$\#contents = maxAddress + 1$$

read

$$\Xi Memory$$
$$i? : \mathbb{N}$$
$$r! : Byte$$

$$i? \in \mathrm{dom}\ contents$$
$$r! = contents(i?+1)$$

write

$\Delta Memory$
$i? : \mathbb{N}$
$w? : Byte$

$i? \in \mathrm{dom}\ contents$
$contents' = (1..i?) \lhd contents \frown \langle w? \rangle \frown ((i?+2)..\#contents) \lhd contents$

Program

$Memory$
$m, n : \mathbb{N}$
$BufferSpace : \mathbb{P}\ \mathbb{N}$

$m \leq n \leq maxAddress$
$BufferSpace = m..n$

badRun

$\Delta Program$
$m?, n? : \mathbb{N}$

$m = m? \land n = n?$
$\exists i : \mathbb{N} \bullet i \notin BufferSpace \land$
$((\exists r : Byte \bullet read[i/i?, r/r!]) \lor (\exists w : Byte \bullet write[i/i?, w/w?]))$

CVE/ CAPECs Formal Specification (Step 3)

Third Research Step:

- CVEs/CAPECs specification in CSP.
- CVEs/CAPECs specification in UML activity diagrams or BPPMN diagrams

→ will remove ambiguities in CVEs definitions

→ can lead to CWEs reclassification

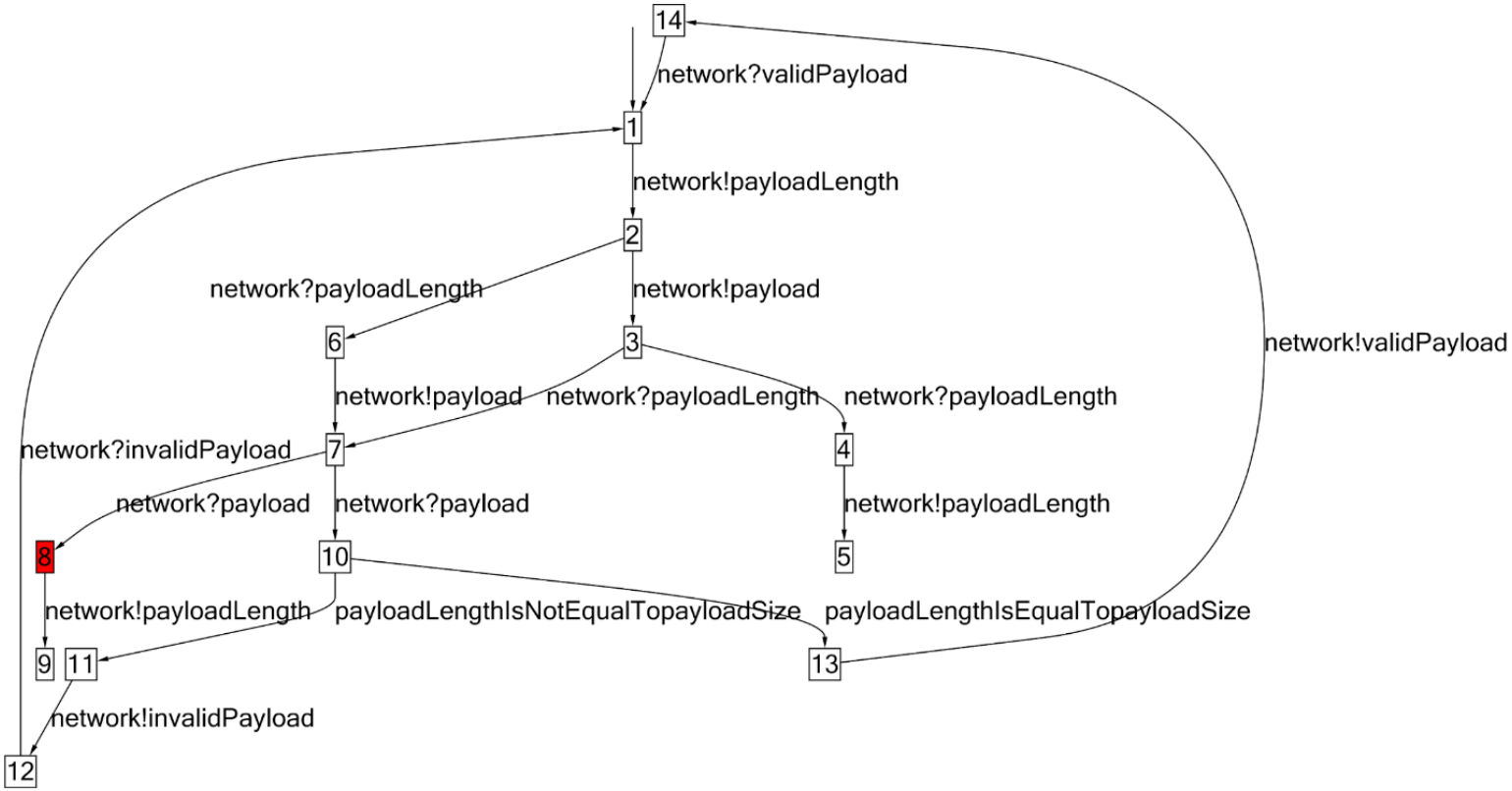
→ CAPECs could be assembled on higher levels.

- Humans accept UML and BPMN diagrams better
- But UML and BPMN diagrams could be not too precise, as they are open to many platforms and programming languages.
- UML activity diagrams with OCL or to BPMN diagrams can be used for tests code generation.
- Z notation and CSP specifications can be used in theorem solvers for verification of dynamic software properties.

CVE-2014-160/CAPEC-540 in CSP

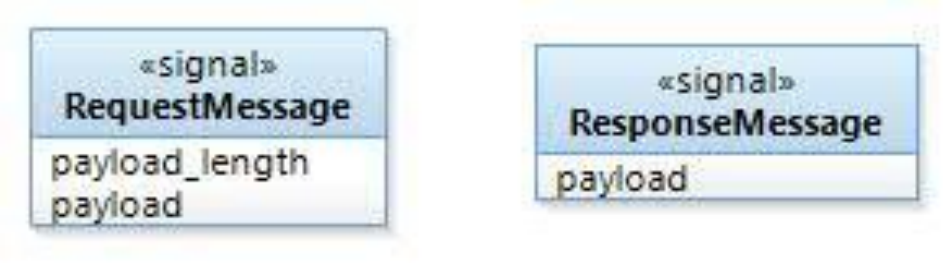
```
channel network 2;
enum {payloadLength, payload, validPayload, invalidPayload};
Attacker() = network!payloadLength -> network!payload -
>network?payloadResponse->Attacker();
CWE_126() = network?payloadLength -> network?payload->
      (payloadLengthIsEqualTopayloadSize->network!validPayload->CWE_126()
      [] payloadLengthIsNotEqualTopayloadSize->network!invalidPayload ->
      CWE_126());

System() = Attacker() ||| CWE_126();
```

CVE-160/ CAPEC 540 SCP Simulation



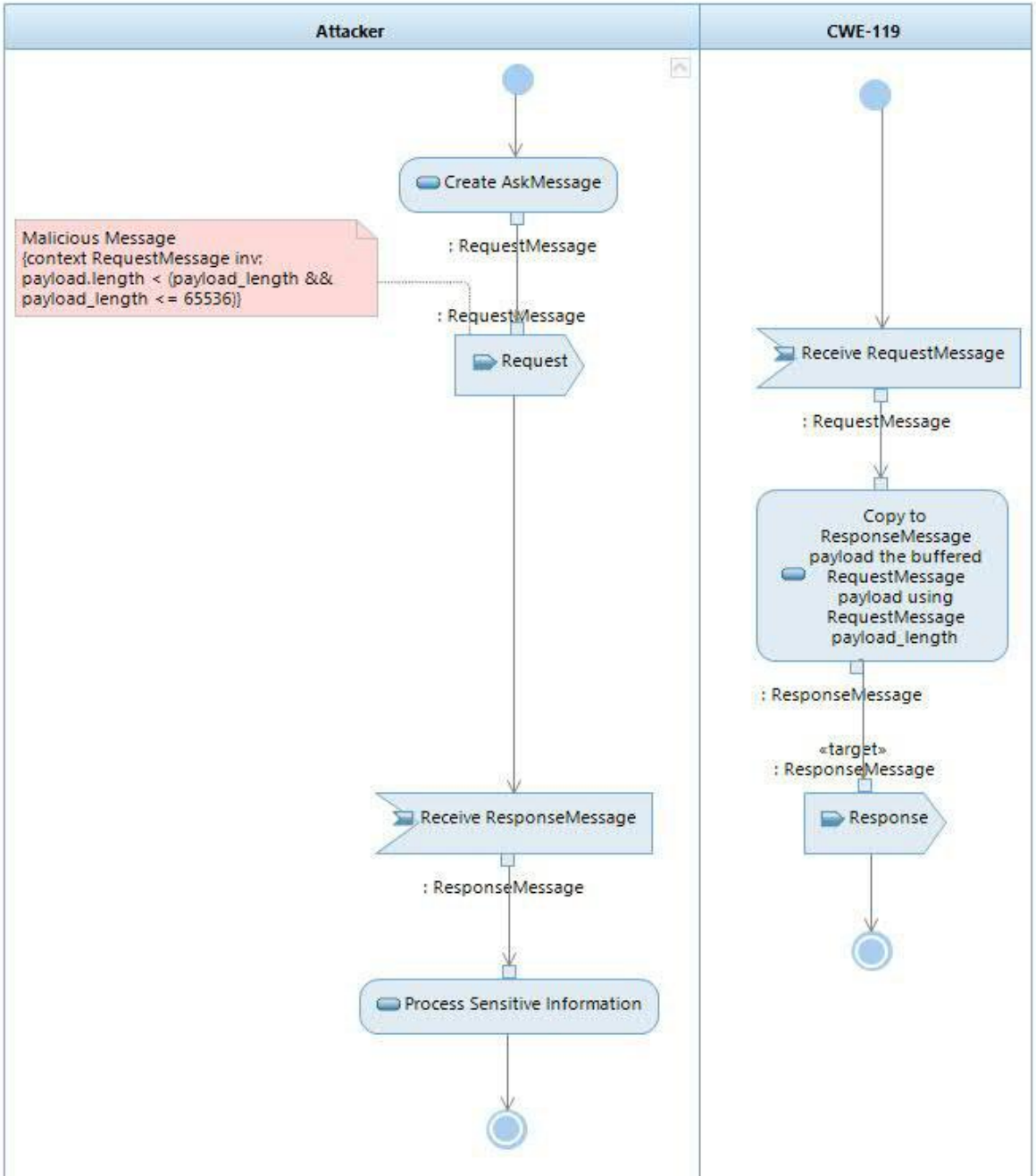
CVE 160/CAPEC 540 in UML & OCL

First, the two signals for a “heart beat” are defined in a class diagram.



The attribute types are not defined, because nowhere in corresponding CWEs or CVEs and CAPECs is mentioned anything about their types. They have to be C types as follows, independently of the attacker software:

```
payload_length: short;
payload: string;
```


Attacker
CWE-119
Create AskMessage
: RequestMessage
Malicious Message
{context RequestMessage inv:
payload.length < (payload_length &&
payload_length <= 65536)}
: RequestMessage
Request
Receive RequestMessage
: RequestMessage
Copy to ResponseMessage payload the buffered RequestMessage payload using RequestMessage payload_length
: ResponseMessage
«target»
: ResponseMessage
Response
Receive ResponseMessage
: ResponseMessage
Process Sensitive Information

Final Notes

- Pending:
  - Explore how OCL specifications can be used to connect the CWE Z-specification to the UML class diagram and the UML activity diagrams. → converting Z specifications into OCL.
  - Demonstrate how chains of CWEs can be specified with CAPECs in UML.

- There are industry efforts on legacy code reengineering. Their target are UML code representation for reuse and integration. UML is the point where CWEs, CVEs and CAPECs specifications can meet.