Security Issues Formalization

V. T. Dimitrov

University of Sofia, Faculty of Mathematics and Informatics,
5 James Bourchier Blvd, 1164, Sofia, Bulgaria

E-mail: cht@fmi.uni-sofia.bg

Software bugs are primary security issues. Semantic templates and Software fault patterns are tools for software bugs specification tools. Software weaknesses are described in formatted text. There is no widely accepted formal notation for that purpose. This paper shows how UML can be used for formal specification of weakness on the example of CWE-119.

Keywords: software weaknesses, formal specification, UML

1. Introduction

In [MITRE, 2016], the term "software bug" applies to the following concepts:

- **Weakness** (Common Weakness Enumeration - CWE): A type of mistake in software that, in proper conditions, could contribute to the introduction of vulnerabilities within that software. This term applies to mistakes regardless of whether they occur in implementation, design, or other phases of the SDLC. [MITRE CWE, 2016]
- **Vulnerability** (Common Vulnerabilities and Exposures - CVE): An occurrence of a weakness (or multiple weaknesses) within software, in which the weakness can be used by a party to cause the software to modify or access unintended data, interrupt proper execution, or perform incorrect actions that were not specifically granted to the party who uses the weakness. [MITRE CVE, 2016]
- **Attack** (Common Attack Pattern Enumeration and Classification - CAPEC): A well-defined set of actions that, if successful, would result in either damage to an asset, or undesirable operation. [MITRE CAPEC, 2016]

An attacker has to find and exploit a weakness, exposed by a weakness, and realize the vulnerability.

These are important concepts that are related but different. A weakness is a static presence existing in software systems -- it might stay in software and never cause any problems until it is exploited by an attacker, and when the attacker finds out the weakness (es) and exploit it (them), the vulnerability of this software is exposed.

2. Why don't Look Deeper?

The following questions can provoke further thinking and discussions throughout the Software Assurance community and beyond:

- What formal methods can be used to help formalize CWEs with required accuracy and precision and at the same time allow for further extensions?
- To what granularity should CWEs be formalized? Finer granularity means more flexibility (especially when new weaknesses are identified, the extracted commonalities can reduce the re-invent work) but more effort to create them; coarser granularity indicates the easy-to-use weakness items while we need to re-invent the wheel every time.
- How can the formalized CWEs be used and in which domains? For education and training? To prevent vulnerabilities? To integrate into software IDEs, test tools, and tools that generate test tools?
- How can an automatic system be constructed to record newly identified vulnerabilities and classify them by CWEs? With better formalization and finer granularity of CWE definitions (which also means limited dictionary for weaknesses, better taxonomy of vulnerabilities), text mining could be the potential technique to mapping CVEs to CWEs at least semi-automatically.

In response to the above query is the focus of this research. The main fact is that the vulnerability is a realization of a weakness of the software. This gives is in the following two aspects:

1. A realization of a vulnerability happens through and attack or attacks, i.e. there is the dynamic aspect.
2. The exploited weakness itself is a property of the software, i.e. this is the static aspect.

The description of the dynamics of the attack can be done with Communicating sequential processes (CSP) [Wikipedia CSP, 2016], while the property of the software (static) can be described with Z notation [Wikipedia Z notation, 2016]. An alternative is to use UML [OMG, 2016] class and object diagrams for the static properties of the weaknesses and activity diagrams for dynamic ones. The paper is focused on the last approach.

Following the Semantic Templates idea [Wu, 2011] is explored in details. The idea there is to build a data base with knowledge about the vulnerabilities using the available repositories. In other words, first the repositories are annotated according the software template. Then, in each part of the template, semantic nets are organized with 3 kinds of arrows. The main idea of Wu is that one vulnerability can be represented by several weaknesses in each component of the template. However, the problems are much more. For example, the leading idea is that Semantic Templates can be extracted automatically or semi-automatically from their natural language descriptions, but how clear and descriptive are these descriptions is an open question to do that. There are available instruments, but we doubt this is the proper direction to follow.

The first task is: Describe vulnerabilities as chains of weaknesses. What Wu does is connecting a given vulnerability with a concrete (root) weakness and from there develops the template. This means that she misses the attack. While indeed, the attack is performed following a scheme, it is dynamic and rather the vulnerability is a successfully conducted attack, and not a property of the software. The latter is a weakness of the software.

The vulnerabilities have to be described as attacks and not as chains of weaknesses.

If it is needed to describe the relationships between the weaknesses, as it is in Yan's dissertation, it is better to use the UML notation. Diagrams are preferred nowadays. Even, using UML to define a diagram that reflects the relationships between the weaknesses. The dynamics of the vulnerabilities can also be presented with diagrams.

Anyway, in UML the software is described with the three models of classes, states, and interactions. The static aspect of the relationships between the weaknesses can be presented with a class diagram or a specialized such. The dynamics in time of a separate weakness can be presented with a state diagram. The attacks and the connections between the weaknesses can be described with activity diagrams and other kinds of diagrams from the interactions model. Note that there are no good tools for re-engineering but there is work done in this direction.

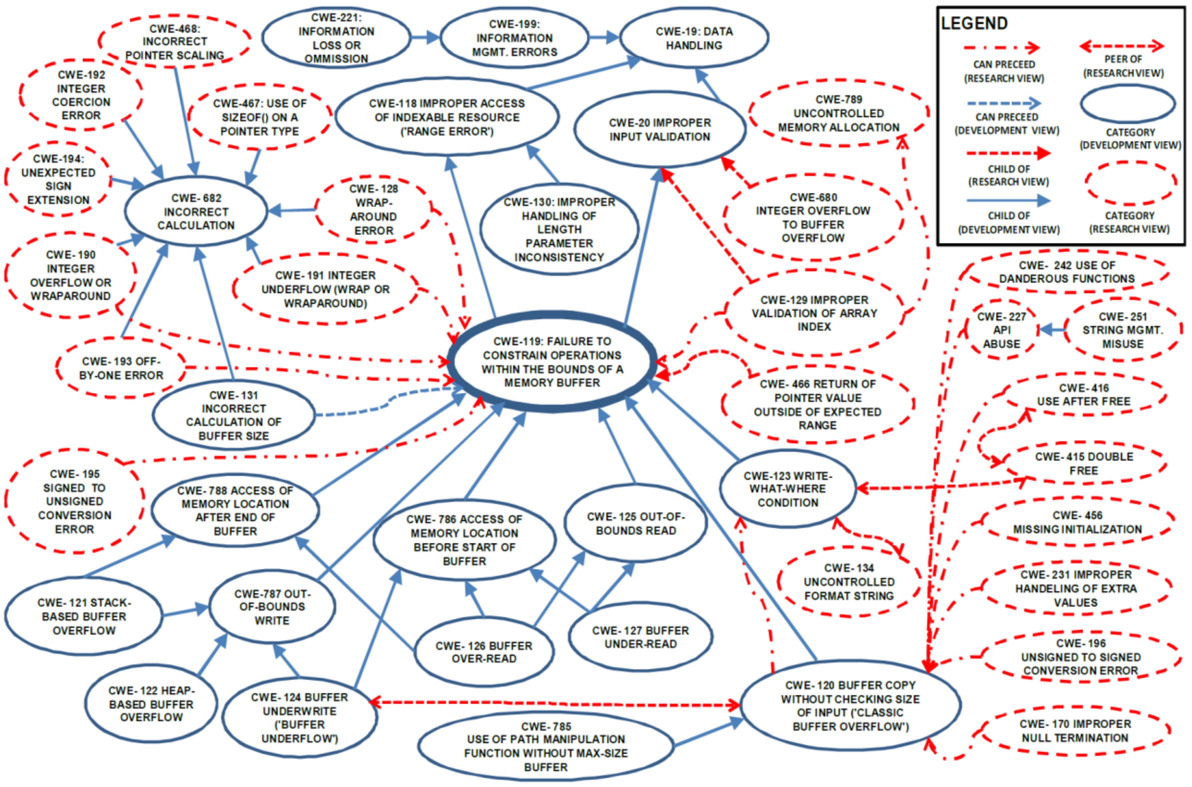
LEGEND
CAN PRECEED (RESEARCH VIEW)
PEER OF (RESEARCH VIEW)
CAN PRECEED (DEVELOPMENT VIEW)
CATEGORY (DEVELOPMENT VIEW)
CHILD OF (RESEARCH VIEW)
CHILD OF (DEVELOPMENT VIEW)
CATEGORY (RESEARCH VIEW)
CWE-468: INCORRECT POINTER SCALING
CWE-221: INFORMATION LOSS OR OMMISSION
CWE-199: INFORMATION MGMT. ERRORS
CWE-19: DATA HANDLING
CWE-192 INTEGER COERCION ERROR
CWE-467: USE OF SIZEOF() ON A POINTER TYPE
CWE-118 IMPROPER ACCESS OF INDEXABLE RESOURCE ('RANGE ERROR')
CWE-20 IMPROPER INPUT VALIDATION
CWE-789 UNCONTROLLED MEMORY ALLOCATION
CWE-194: UNEXPECTED SIGN EXTENSION
CWE-682 INCORRECT CALCULATION
CWE-128 WRAP-AROUND ERROR
CWE-130: IMPROPER HANDLING OF LENGTH PARAMETER INCONSISTENCY
CWE-680 INTEGER OVERFLOW TO BUFFER OVERFLOW
CWE-190 INTEGER OVERFLOW OR WRAPAROUND
CWE-191 INTEGER UNDERFLOW (WRAP OR WRAPAROUND)
CWE-129 IMPROPER VALIDATION OF ARRAY INDEX
CWE-242 USE OF DANDEROUS FUNCTIONS
CWE-227 API ABUSE
CWE-251 STRING MGMT. MISUSE
CWE-193 OFF-BY-ONE ERROR
CWE-119: FAILURE TO CONSTRAIN OPERATIONS WITHIN THE BOUNDS OF A MEMORY BUFFER
CWE-466 RETURN OF POINTER VALUE OUTSIDE OF EXPECTED RANGE
CWE-416 USE AFTER FREE
CWE-131 INCORRECT CALCULATION OF BUFFER SIZE
CWE-415 DOUBLE FREE
CWE-195 SIGNED TO UNSIGNED CONVERSION ERROR
CWE-788 ACCESS OF MEMORY LOCATION AFTER END OF BUFFER
CWE-786 ACCESS OF MEMORY LOCATION BEFORE START OF BUFFER
CWE-125 OUT-OF-BOUNDS READ
CWE-123 WRITE-WHAT-WHERE CONDITION
CWE-456 MISSING INITIALIZATION
CWE-134 UNCONTROLLED FORMAT STRING
CWE-231 IMPROPER HANDELING OF EXTRA VALUES
CWE-121 STACK-BASED BUFFER OVERFLOW
CWE-787 OUT-OF-BOUNDS WRITE
CWE-126 BUFFER OVER-READ
CWE-127 BUFFER UNDER-READ
CWE-196 UNSIGNED TO SIGNED CONVERSION ERROR
CWE-122 HEAP-BASED BUFFER OVERFLOW
CWE-124 BUFFER UNDERWRITE ('BUFFER UNDERFLOW')
CWE-785 USE OF PATH MANIPULATION FUNCTION WITHOUT MAX-SIZE BUFFER
CWE-120 BUFFER COPY WITHOUT CHECKING SIZE OF INPUT ('CLASSIC BUFFER OVERFLOW')
CWE-170 IMPROPER NULL TERMINATION

Figure 1. Buffer Overflow-related CWEs Extracted in the Preparation and Collection Phase from [Wu, 2011]

3. Formal Specification of CWEs with UML Class Diagrams

The easiest and most attractive way is to create a class diagram, in which the weaknesses are classes and the relationships between them are presented as associations. Figure 1 represents Wu's strategy.

The mental model is useful for reading the database contents. It is also useful to follow the links among CWEs. However, it is impossible to generate from the mental model test or verification code. Semantic Templates are at knowledge base level and do not impose further pragmatic exploration of its descriptions.

So, the following three diagrams (Figure 2 - 4) are extracted from Figure 1. The idea is to separate "the mental model" of the original diagram into two views: a research view and a development view.

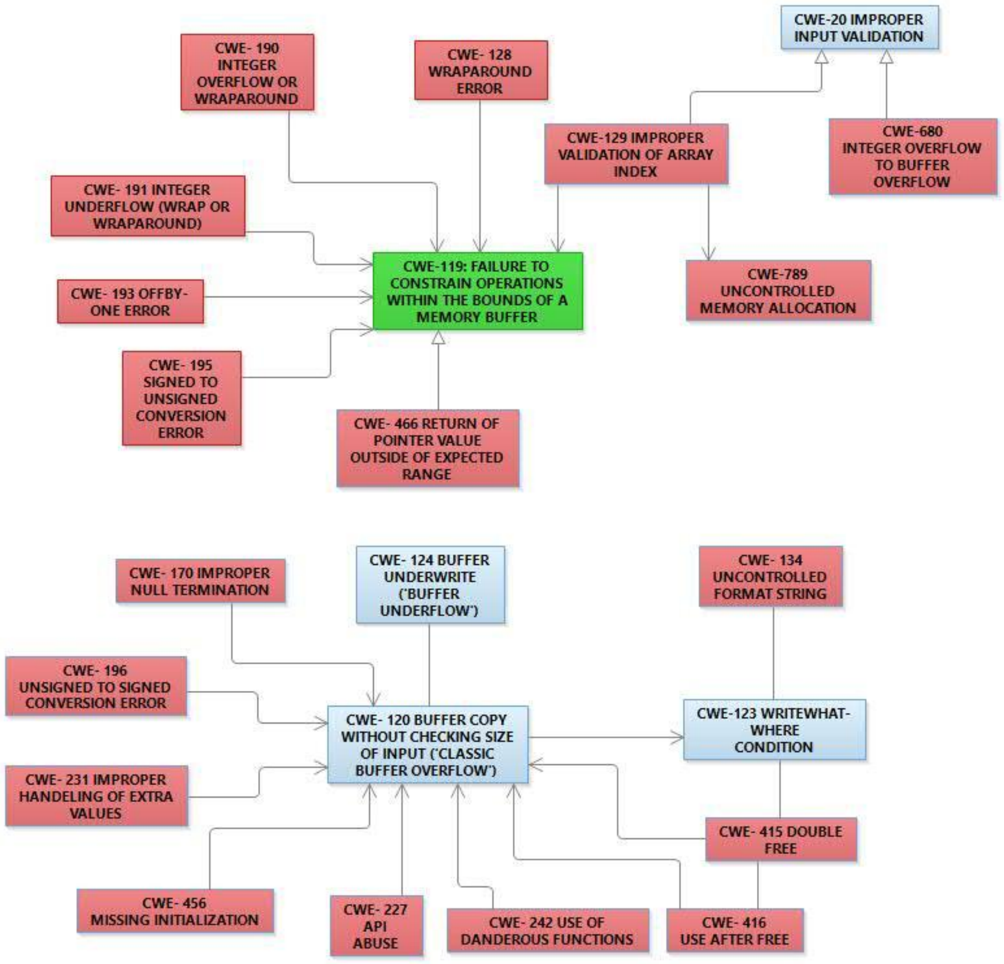
CWE-20 IMPROPER INPUT VALIDATION
CWE- 190 INTEGER OVERFLOW OR WRAPAROUND
CWE- 128 WRAPAROUND ERROR
CWE-129 IMPROPER VALIDATION OF ARRAY INDEX
CWE-680 INTEGER OVERFLOW TO BUFFER OVERFLOW
CWE- 191 INTEGER UNDERFLOW (WRAP OR WRAPAROUND)
CWE-119: FAILURE TO CONSTRAIN OPERATIONS WITHIN THE BOUNDS OF A MEMORY BUFFER
CWE-789 UNCONTROLLED MEMORY ALLOCATION
CWE- 193 OFFBY-ONE ERROR
CWE- 195 SIGNED TO UNSIGNED CONVERSION ERROR
CWE- 466 RETURN OF POINTER VALUE OUTSIDE OF EXPECTED RANGE
CWE- 170 IMPROPER NULL TERMINATION
CWE- 124 BUFFER UNDERWRITE ('BUFFER UNDERFLOW')
CWE- 134 UNCONTROLLED FORMAT STRING
CWE- 196 UNSIGNED TO SIGNED CONVERSION ERROR
CWE- 120 BUFFER COPY WITHOUT CHECKING SIZE OF INPUT ('CLASSIC BUFFER OVERFLOW')
CWE-123 WRITEWHAT-WHERE CONDITION
CWE- 231 IMPROPER HANDELING OF EXTRA VALUES
CWE- 415 DOUBLE FREE
CWE- 456 MISSING INITIALIZATION
CWE- 227 API ABUSE
CWE- 242 USE OF DANDEROUS FUNCTIONS
CWE- 416 USE AFTER FREE

Figure 2. CWE-119 Research View

The research view is presented in Figure 2 as one diagram showing both the inheritance and the precedence relationships.

The development view is further separated into 2 diagrams: an inheritance diagram (Figure 3) and a precedence diagram (Figure 4). This way the model is more readable, but more elaboration is needed to make sure these kinds of views are best.

The three diagrams are UML class diagrams; created with IBM Rational Software Architect (RSA) [Wikipedia RSA, 2016]. However, they can be transformed into specialized diagrams – UML & OCL, RSA allows creation of specialized metamodels for new diagram types.

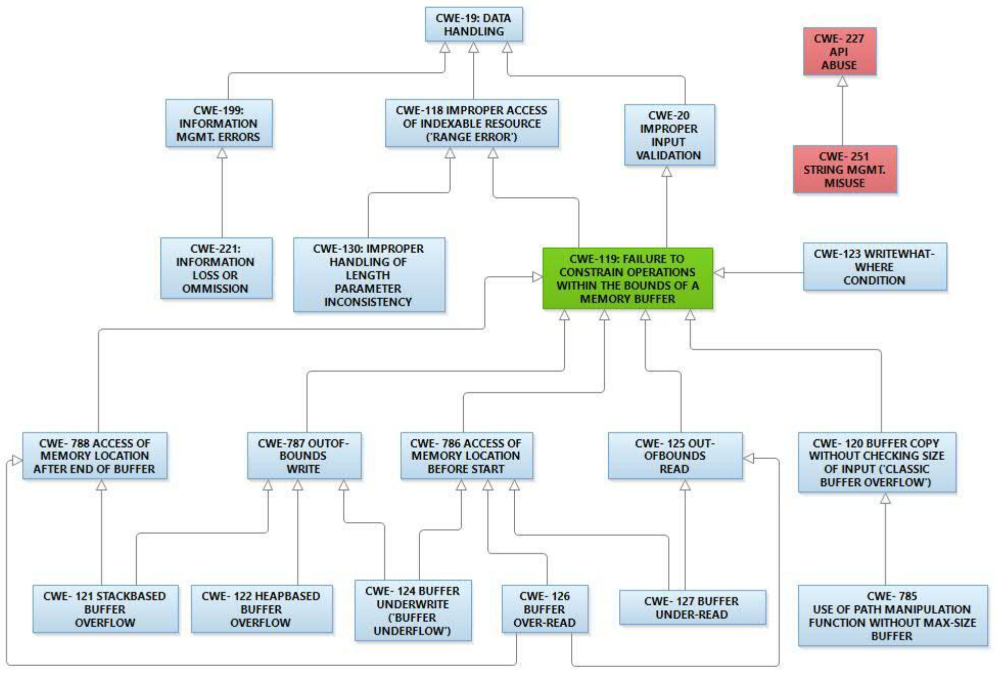
CWE-19: DATA HANDLING
CWE- 227 API ABUSE
CWE-199: INFORMATION MGMT. ERRORS
CWE-118 IMPROPER ACCESS OF INDEXABLE RESOURCE ('RANGE ERROR')
CWE-20 IMPROPER INPUT VALIDATION
CWE- 251 STRING MGMT. MISUSE
CWE-221: INFORMATION LOSS OR OMMISSION
CWE-130: IMPROPER HANDLING OF LENGTH PARAMETER INCONSISTENCY
CWE-119: FAILURE TO CONSTRAIN OPERATIONS WITHIN THE BOUNDS OF A MEMORY BUFFER
CWE-123 WRITEWHAT-WHERE CONDITION
CWE- 788 ACCESS OF MEMORY LOCATION AFTER END OF BUFFER
CWE-787 OUTOF-BOUNDS WRITE
CWE- 786 ACCESS OF MEMORY LOCATION BEFORE START
CWE- 125 OUT-OFBOUNDS READ
CWE- 120 BUFFER COPY WITHOUT CHECKING SIZE OF INPUT ('CLASSIC BUFFER OVERFLOW')
CWE- 121 STACKBASED BUFFER OVERFLOW
CWE- 122 HEAPBASED BUFFER OVERFLOW
CWE- 124 BUFFER UNDERWRITE ('BUFFER UNDERFLOW')
CWE- 126 BUFFER OVER-READ
CWE- 127 BUFFER UNDER-READ
CWE- 785 USE OF PATH MANIPULATION FUNCTION WITHOUT MAX-SIZE BUFFER

Figure 1. CWE-119 Development View with Inheritance

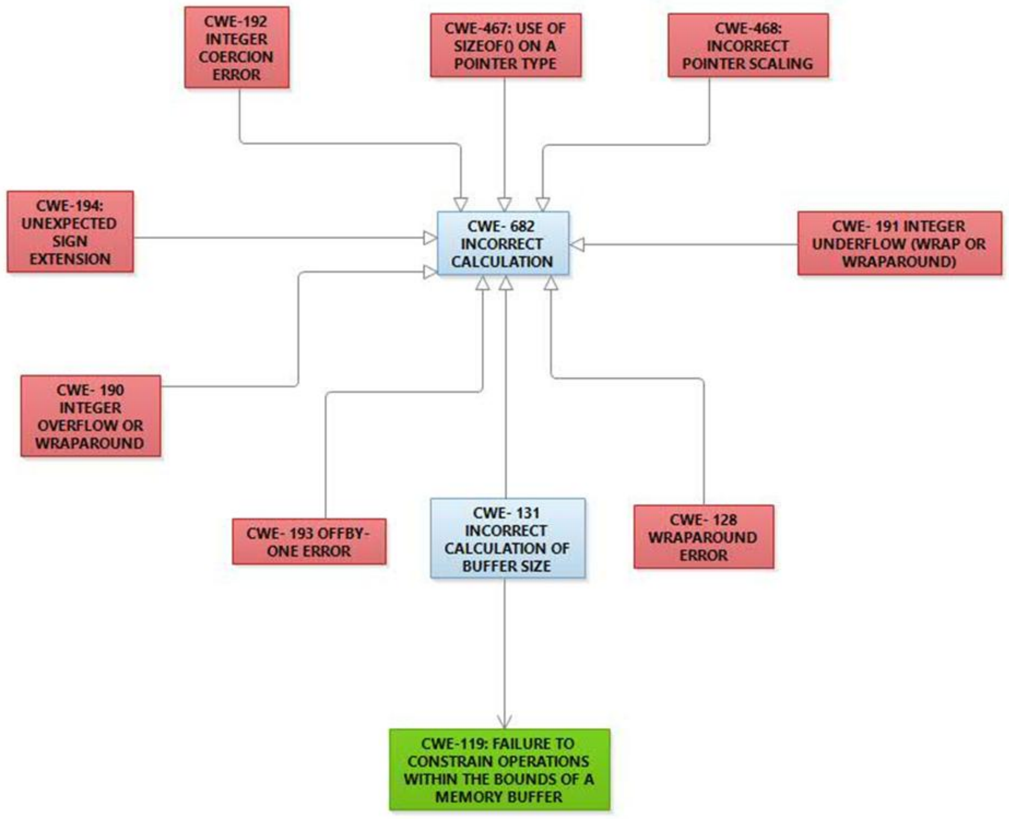
CWE-192 INTEGER COERCION ERROR
CWE-467: USE OF SIZEOF() ON A POINTER TYPE
CWE-468: INCORRECT POINTER SCALING
CWE-194: UNEXPECTED SIGN EXTENSION
CWE- 682 INCORRECT CALCULATION
CWE- 191 INTEGER UNDERFLOW (WRAP OR WRAPAROUND)
CWE- 190 INTEGER OVERFLOW OR WRAPAROUND
CWE- 193 OFFBY-ONE ERROR
CWE- 131 INCORRECT CALCULATION OF BUFFER SIZE
CWE- 128 WRAPAROUND ERROR
CWE-119: FAILURE TO CONSTRAIN OPERATIONS WITHIN THE BOUNDS OF A MEMORY BUFFER

Figure 2. CWE-119 Development View with Precedence

Note that Figure 1 from Yan's dissertation is extracted from CWE, but this is not a trivial task. That is why Yan's work has been used as starting point. The complete mental model for CWE-119 could be created by retrieving data directly the XML, extracted from the CWE database. Figure 5 presents the current, still not complete diagram of the CWE-119 Relationships mental model.

The mental model follows by hand the links given in CWEs database. Every CWE accessible via some path of links starting from CWE-119 must be presented in the diagram. Big problem is how to position the different CWEs with different roles on the diagram in some clear and readable way, but the hope is that suitable recommendations would be developed for that.

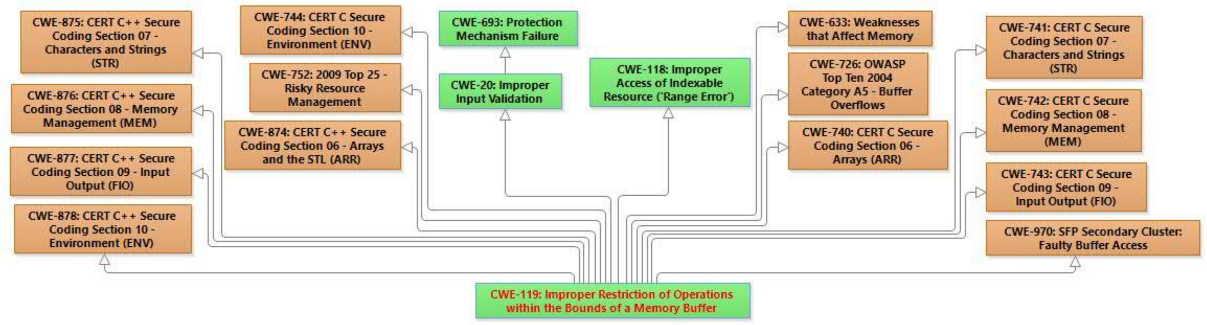
CWE-875: CERT C++ Secure Coding Section 07 - Characters and Strings (STR)
CWE-744: CERT C Secure Coding Section 10 - Environment (ENV)
CWE-693: Protection Mechanism Failure
CWE-633: Weaknesses that Affect Memory
CWE-741: CERT C Secure Coding Section 07 - Characters and Strings (STR)
CWE-752: 2009 Top 25 - Risky Resource Management
CWE-20: Improper Input Validation
CWE-118: Improper Access of Indexable Resource ('Range Error')
CWE-726: OWASP Top Ten 2004 Category A5 - Buffer Overflows
CWE-876: CERT C++ Secure Coding Section 08 - Memory Management (MEM)
CWE-742: CERT C Secure Coding Section 08 - Memory Management (MEM)
CWE-874: CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR)
CWE-740: CERT C Secure Coding Section 06 - Arrays (ARR)
CWE-877: CERT C++ Secure Coding Section 09 - Input Output (FIO)
CWE-743: CERT C Secure Coding Section 09 - Input Output (FIO)
CWE-878: CERT C++ Secure Coding Section 10 - Environment (ENV)
CWE-970: SFP Secondary Cluster: Faulty Buffer Access
CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer

Figure 5. CWE-119 Relationships - not complete yet

4. Conclusion

One big problem for representation of CWEs in UML diagrams is the need for automatization. The CWEs are represented in XML database, but some information is in text format. The purpose of Semantic templates is to extract formatted and unformatted information from the database.

CWEs representation in UML diagrams is good for better visual acceptance by humans. UML is well known notation in the community. There are many tools that support UML. Even more, specialized profiles for CWEs could be developed in UML.

It is convenient to generate code from UML in many tools. Such generators could be developed in the UML tools to generate test and verification sequences for CWEs.

In this paper possibilities for representation of CWEs in UML diagrams have been discussed. UML is a notation for software specification. It is formalized with metaclasses by OMG. UML permits the software to be specified at the appropriate level of details in any case. If it is needed OCL can be used for further accuracy.

CWEs are organized in abstraction several levels. UML permits such a taxonomy organization with class and object diagrams. The class diagrams organize abstraction levels and the object diagrams stay on the base.

UML diagrams could be used for training, education and software development. Their notation is primary developed to humans. UML diagrams can be used as a source for generation of test sequences, verification code etc. that support the software development, testing and verification. Tools that support UML, like RSA, are usually open for that purpose.

Tools for semiautomatic identification of vulnerabilities and their classification to the CWEs could be implemented in object-oriented classification style using the artificial intelligence for text recognition and understanding. The work in that direction is huge and many research must be done but Semantic Templates are good starting point.

References

MITRE About MITRE. URL: http://www.mitre.org (accessed 01.10.2016).

MITRE CAPEC Common Attack Pattern Enumeration and Classification. URL: http://capec.mitre.org (accessed 01.10.2016).

MITRE CVE Common Vulnerabilities and Exposure. URL: http://cve.mitre.org (accessed 01.10.2016).

MITRE CWE Common Weakness Enumeration. URL: http://cwe.mitre.org (accessed 01.10.2016).

OMG What is UML: URL: http://www.uml.org (accessed 01.10.2016).

Wikipedia Communicating sequential processes. URL: http://en.wikipedia.org/wiki/Communicating_sequential_processes (accessed 01.10.2016).

Wikipedia Rational — Software Architect. URL: http://en.wikipedia.org/wiki/Rational_Software_Architect (accessed 01.10.2016).

Wikipedia Z notation. URL: http://en.wikipedia.org/wiki/Z_notation (accessed 01.10.2016).

Wu Y. Using Semantic Templates to Study Vulnerabilities Recorded in Large Software Repositories // Dissertation, Omaha, Nebraska. 2011.