TOWARD FORMALIZATION OF SOFTWARE SECURITY ISSUES

VLADIMIR DIMITROV

FACULTY OF MATHEMATICS AND INFORMATICS, UNIVERSITY OF SOFIA, 5 JAMES BOURCHIER BLVD., 1164 SOFIA, BULGARIA

CHT@FMI.UNI-SOFIA.BG

ABSTRACT

CVE, CWE, and CAPEC databases and their relationships are shortly introduced. Focus on this paper is on formalization and more specific on weakness formalization. Software weaknesses are described as formatted text. There is no widely accepted formal notation for weakness specification. This paper shows how Z-notation can be used for formal specification of CWE-119.

A1. WHAT IS CWE? WHAT IS A "SOFTWARE WEAKNESS"? 1/2

Targeted at both the development community and the community of security practitioners, Common Weakness Enumeration (CWE) is a formal list or dictionary of common software weaknesses that can occur in software's architecture, design, code or implementation that can lead to exploitable security vulnerabilities. CWE was created to serve as a common language for describing software security weaknesses; serve as a standard measuring stick for software security tools targeting these weak-nesses; and to provide a common baseline standard for weakness identification, mitigation, and prevention efforts.

A1. WHAT IS CWE? WHAT IS A "SOFTWARE WEAKNESS"? 2/2

Software weaknesses are flaws, faults, bugs, vulnerabilities, and other errors in software implementation, code, design, or architecture that if left unaddressed could result in systems and networks being vulnerable to attack. Example software weaknesses include: 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; and randomness and predictability."

A2. WHAT IS THE DIFFERENCE BETWEEN A SOFTWARE VULNERABILITY AND SOFTWARE WEAKNESS?

Software weaknesses are errors that can lead to software vulnerabilities. A software vulnerability, such as those enumerated on the Common Vulnerabilities and Exposures (CVE) List, is a mistake in software that can be directly used by a hacker to gain access to a system or network.

A7. WHAT IS THE RELATIONSHIP BETWEEN CWE AND CAPEC?

While CWE is a list of software weakness types, Common Attack Pattern Enumeration and Classification (CAPEC) is a list of the most common methods attackers use to exploit vulnerabilities resulting from CWEs. Used together, CWE and CAPEC provide understanding and guidance to software development personnel of all levels as to where and how their software is likely to be attacked, thereby equipping them with the information they need to help them build more secure software.

A8. WHAT IS THE RELATIONSHIP BETWEEN CWE AND CVE?

MITRE began working on the issue of categorizing software weaknesses as early 1999 when it launched the Common Vulnerabilities and Exposures (CVE) List. As part of building CVE, MITRE's CVE Team developed a preliminary classification and categorization of vulnerabilities, attacks, faults, and other concepts beginning in 2005 to help define common software weaknesses. However, while sufficient for CVE those groupings were too rough to be used to identify and categorize the functionality offered within the offerings of the code security assessment industry. The CWE List was created to better address those additional needs.

A1. WHAT IS CVE?

CVE is a list of information security vulnerabilities and exposures that aims to provide common names for publicly known cyber security issues. The goal of CVE is to make it easier to share data across separate vulnerability capabilities (tools, repositories, and services) with this "common enumeration."

A8. WHAT IS A "VULNERABILITY"?

An information security vulnerability is a mistake in software that can be directly used by a hacker to gain access to a system or network. See the Terminology page for a complete explanation of how this term is used on the CVE Web site.

A9. WHAT IS AN "EXPOSURE"?

An information security exposure is a mistake in software that allows access to information or capabilities that can be used by a hacker as a stepping-stone into a system or network. See the Terminology page for a complete explanation of how this term is used on the CVE Web site.

WHAT IS AN "ATTACK PATTERN"?

An attack pattern is an abstraction mechanism for helping describe how an attack against vulnerable systems or networks is executed. Each pattern defines a challenge that an attacker may face, provides a description of the common technique(s) used to meet the challenge, and presents recommended methods for mitigating an actual attack. Attack patterns help categorize attacks in a meaningful way in an effort to provide a coherent way of teaching designers and developers how their systems may be attacked and how they can effectively defend them. The CAPEC List provides a formal list of known attack patterns.

WHAT IS FORMAL SPECIFICATION? 1/2

Formal specifications use mathematical notation to describe in a precise way the properties which an information system must have, without unduly constraining the way in which these properties are achieved. They describe what the system must do without saying how it is to be done. This abstraction makes formal specifications useful in the process of developing a computer system, because they allow questions about what the system does to be answered confidently, without the need to disentangle the information from a mass of detailed program code, or to speculate about the meaning of phrases in an imprecisely-worded prose description.

WHAT IS FORMAL SPECIFICATION? 2/2

A formal specification can serve as a single, reliable reference point for those who investigate the customer's needs, those who implement programs to satisfy those needs, those who test the results, and those who write instruction manuals for the system. Because it is independent of the program code, a formal specification of a system can be completed early in its development. Although it might need to be changed as the design team gains in understanding and the perceived needs of the customer evolve, it can be a valuable means of promoting a common understanding among all those concerned with the system.

CANDIDATE TOOLS FOR FORMAL SPECIFICATION OF SOFTWARE BUGS 1/3

Z-notation is a standard notation standardized by ISO. It is high level, based on mathematics notation. Z-notation is applicable on any level of abstraction. It is useful for software requirement specification and code verification. The problem with it is the limited set of tools supporting the notation – mainly at academia. Z-notation can be used for CWE and CVE specification (weaknesses and vulnerabilities).

CANDIDATE TOOLS FOR FORMAL SPECIFICATION OF SOFTWARE BUGS 2/3

Communicating Sequential Processes (CSP) is well-designed algebra. It is useful for distributed application specification and verification. It is behavioral in nature and can be used for CAPEC specification (attacks). Its problem is like that of Z-notation.

CANDIDATE TOOLS FOR FORMAL SPECIFICATION OF SOFTWARE BUGS 3/3

An alternative of Z-notation and CSP is UML. This notation is well supported in industry. Activity diagrams can be used for CAPEC specification, class diagrams and object diagrams for CWE and CVE can be used. Even more, special kind of CWE, CVE and CAPEC can be developed; code for software bug tests can be automatically generated. The problem is that all these possibilities must be carefully investigated.

RESEARCH APPROACHES

1. Develop a method for formal description of software attacks, exploited vulnerabilities, and involved CWEs.
2. Describe formally attacks exploiting some intriguing vulnerabilities: for example, Heartbleed or ShellShock.
3. Refine the involved CWEs, CVEs, and CAPECs definitions, so that they precisely and unambiguously described.

VULNERABILITY IS A REALIZATION OF A WEAKNESS OF THE SOFTWARE

This gives us 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 CSP, while the property of the software (static) can be described with Z-notation.

UML AS FORMALIZATION TOOL

In UML terminology, we need the three models: 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 reengineering but there is work done in this direction.

CWE-119: IMPROPER RESTRICTION OF OPERATIONS WITHIN THE BOUNDS OF A MEMORY BUFFER

CWE-119 is a class weakness. This means that it is an abstract description of a class of weaknesses that is independent of any language or technology. This is the most abstract description of a weakness.

CWE-119 description is usable, i.e. CWE users can use it.

CWE-119 Description Summary is: 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-119: IMPROPER RESTRICTION OF OPERATIONS WITHIN THE BOUNDS OF A MEMORY BUFFER 1/4

The subject of this weakness is the “software”. It can be an application, system software (operating system) or some of their components. The main idea is that the software run some operations in the memory on a bounded part of it (buffer), but these operations are performed (partly or as a whole) outside the buffer boundaries. The primitive operations subject of this weakness are memory read and write.

There are no specific location in the software for that weakness. It is possible the weakness cause to be located in one part of the software, but the actual read or write operations to be performed in another one part.

CWE-119: IMPROPER RESTRICTION OF OPERATIONS WITHIN THE BOUNDS OF A MEMORY BUFFER 2/4

The term “buffer” is very abstract one. In the programming languages terms it can be a variable created statically or dynamically in the software stack or heap, but it is possible the buffer to be in the code space as a method or function. The von Neumann architecture does not differentiate the memory in code and data space.

CWE-119: IMPROPER RESTRICTION OF OPERATIONS WITHIN THE BOUNDS OF A MEMORY BUFFER 3/4

CWE-119 Extended Description is: Certain languages allow direct addressing of memory locations and do not automatically ensure that these locations are valid for the memory buffer that is being referenced. This can cause read or write operations to be performed on memory locations that may be associated with other variables, data structures, or internal program data.

As a result, an attacker may be able to execute arbitrary code, alter the intended control flow, read sensitive information, or cause the system to crash.

Here, programming languages with pointers are mentioned. Pointer arithmetic is not obligatory one for this weakness. The data space is emphasized in the description but in some programming languages it is possible the pointer to point in code space.

Some of the impacts from successful exploitation of this weakness are listed.

CWE-119: IMPROPER RESTRICTION OF OPERATIONS WITHIN THE BOUNDS OF A MEMORY BUFFER 4/4

An alternate term is “memory corruption”: The generic term "memory corruption" is often used to describe the consequences of writing to memory outside the bounds of a buffer, when the root cause is some-thing other than a sequential copies of excessive data from a fixed starting location (i.e., classic buffer overflows or CWE-120). This may include issues such as incorrect pointer arithmetic, accessing invalid pointers due to incomplete initialization or memory release, etc.

The alternative term emphasizes on consequences from weakness exploitation rather on the cause problem. This is not CWE naming style.

Time of introduction of this weakness can be Architecture and Design, Implementation, or Operation.

It is applicable for programming languages without memory management support as C, C++, and assembler. The consequences of this weakness vary on the language, platform, and chip architecture.

FORMAL DESCRIPTION OF CWE-119 1/6

The basic subject of CWE-119 is the memory. It is a sequence of bytes. The basic type is the byte.

[Byte]

Memory address space starts from 0 and ends to some max address.

| maxAddress: ℕ

FORMAL DESCRIPTION OF CWE-119 2/6

Memory is finite sequence of bytes – possibly empty sequence. The finite sequences elements in Z-notation are numbered starting from 1, but computer memory bytes are indexed starting from 0.

```
Memory----------------------
| contents: seq Byte
| #contents = maxAddress + 1
|--------------------------------
```

FORMAL DESCRIPTION OF CWE-119 3/6

There are two operations with the memory: read and write that a program can perform. First one is read.

```
read--------------------
|ΞMemory
|i?: ℕ
|r!: Byte
|------------------------
|i? ∈ dom contents
|r! = contents(i?+1)
|-------------------------
```

FORMAL DESCRIPTION OF CWE-119 4/6

Second one is write.

```
write-------------------------
|ΔMemory
|i?: ℕ
|w?: Byte
|-------------------------------
|i? ∈ dom contents
|contents′ = (1..i?) ◁ contents ⁀ ⟨ w?⟩ ⁀ ((i?+2)..#contents) ◁ contents
|---------------------------------
```

FORMAL DESCRIPTION OF CWE-119 5/6

A program performs in memory on a buffer. The buffer is specified with its address space.

```
Program------------------------
|Memory
|m, n: ℕ
|BufferSpace: ℙ ℕ
|----------------------------------
|m ≤ n ≤ maxAddress
|BufferSpace = m..n
|-----------------------------------
```

FORMAL DESCRIPTION OF CWE-119 6/6

Every program run performs on a given buffer. The property bad run is CWE-119, i.e. read or write outside the buffer.

badRun-----------------------

|ΔProgram

|$m?, n?: \mathbb{N}$

|------------------------------

|$m = m? \land n = n?$

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

|-------------------------------

CONCLUSION

CWE-119, following this description, is an operation that accesses the memory outside the buffer space. This operation can be part of any other program operation.

CWE-119 specification is enough abstract and formal in comparison with its textual representation. On the other hand, the CWE-119 description is at very low level – the memory. Therefore, the level of the specification is not very high, but CWE-119 is formalized.

The more general question is “How suitable is the Z-notation for formal specification of CWEs?” More general conclusion to do from one example is not correct – more research is needed.

How the specification can be used? Z-notation supporting tools can extract automatically from the specification pre- and post-conditions. Formal verification tools can use these conditions to check the software for CWE-119. However, the devil is in the details – the problem is how to link the abstract buffer space with the real one of the programs. There are some ideas in that direction, but more research is needed.

QUESTIONS?