Fundamental Vulnerabilities

Larry Wagoner

February 23, 2013

Many so-called vulnerabilities or weaknesses in software are not vulnerabilities, per se, but symptoms or attack methods that exploit weaknesses. For instance, consider a buffer overflow. To exploit a buffer overflow, an attacker provides input specially crafted to exceed the capacity of a buffer so that the return address is overwritten. A "buffer overflow" is how the attacker exploits what are a series of weaknesses or vulnerabilities in the software that allow the exploitation to occur. A buffer overflow is not the weakness or vulnerability in the software that allows this exploit to occur.

In a paper entitled "Vulnerabilities Analysis."[1], Matt Bishop describes how a buffer overflow can be decomposed into primitive conditions that must hold for the vulnerability to exist. The primitive conditions he identified for a buffer overflow are:

- C1. Failure to check bounds when copying data into a buffer.
- C2. Failure to prevent the user from altering the return address.
- C3. Failure to check that the input data was of the correct form (user name or network address).
- C4. Failure to check the type of the words being executed (data loaded, not instructions).

These preconditions are the weaknesses or vulnerabilities that must exist for a buffer overflow exploit to occur.

Bishop further states that invalidating any of these conditions in the following ways would prevent an attacker from exploiting this vulnerability:

- C1'. If the attacker cannot overflow the bounds, the control flow will continue in the text
- (instruction) space and not shift to the loaded data.
- C2'. If the return address cannot be altered, then even if the input overflows the bounds, the control flow will resume at the correct place.
- C3'. As neither a user name nor a network address is a valid sequence of machine instructions on most UNIX systems, this would cause a program crash and not a security breach.
- C4'. If the system cannot execute data, the return into the stack will cause a fault.

For instance, if bounds checking (C1) were to be in place when copying the data into a buffer, then even though the input data was not of the correct form and the return address could be altered, the attack could not be accomplished. By understanding the component factors of a buffer overflow, a thoughtful approach can be used as to which of the factors is the best one to invalidate.

[1] Bishop, Matt. "Vulnerabilities analysis." Proceedings of the Recent Advances in Intrusion Detection. 1999, http://nob.cs.ucdavis.edu/bishop/papers/1999-raid/1999-vulclass.pdf . Additional work in this area is contained in Bishop, Matt, et al. A taxonomy of buffer overflow preconditions. Tech Report: CSE-2010-1, Computer Science, UC Davis, 2010.

These four primitive conditions are not unique to buffer overflows. For instance, C3, failure to check input data, is a primitive condition for many vulnerabilities/attack methods.

The concept of primitive conditions as the basis for vulnerabilities in software provides an understanding of the underlying causes as to why a vulnerability/attack method exists and can be exploited.

These primitive conditions can be used to determine *fundamental vulnerabilities (FVs)*. Fundamental vulnerabilities are the root causes underlying the vulnerabilities/weaknesses that are in software. The difference between a primitive precondition is simply the way that each is stated. A FV is a statement of fact of a situation. A precondition is stated as a failure to do something.

Software languages make trade-offs to allow certain features or functionality. FVs should not be considered as something that is wrong. FVs are simply a statement of fact that a situation exists or a choice has been made. For instance, hardware architectures vary. There are many reasons for this and having many different hardware architectures leads to vulnerabilities when software is run on different architectures and the variances are not addressed. This can be stated as the FV:

- Hardware is not standardized (i.e. Size of short, int, long differ between platforms)

So even though we cannot expect that hardware architectures will be standardized to alleviate this problem, the fact is that hardware architectures vary and this underlies some vulnerabilities, it is a FV.

Mitigating a FV may stop all instances of a particular exploitation of a vulnerability or multiple vulnerabilities. Another FV may only stop some portion of the exploit occurrences of a single vulnerability. There may be reasons to select the mitigation of one FV over another.

Since FVs are simply a statement of a situation, by understanding the FVs, more knowledge and structure can be added to the choices made by language designers that balance performance, flexibility, security, functionality, usability, and so on. Identification of these root causes would allow targeted efforts at stopping multiple categories of vulnerabilities and expose new vulnerabilities or categories of vulnerabilities.

Analysis to Determine Fundamental Vulnerabilities

FVs of CWE-121 Stack-based Buffer Overflow

The fundamental vulnerabilities for CWE-121, Stack-based Buffer Overflow, can be directly found by building upon Bishop's work. Recall that Bishop proposed four pre-conditions:

- C1. Failure to check bounds when copying data into a buffer.
- C2. Failure to prevent the user from altering the return address.
- C3. Failure to check that the input data was of the correct form (user name or network address).
- C4. Failure to check the type of the words being executed (data loaded, not instructions).

These can be restated directly as FVs:

- Array bounds check before array access does not exist or is faulty

- Function return address is not immutable (return address can be altered by user)
- Input verification does not exist or is faulty
- Code and data are indistinguishable in memory (Code and data are segregated in memory)

FVs of CWE-369 Divide by Zero

Many named CWE vulnerabilities encompass a variety of exploitation means. For instance, CWE-369, Divide by Zero, has several fundamental vulnerabilities underlying it, but not all apply to all instances of a division by zero.

A division by zero can be prevented by a check of the divisor before the operation is performed. As the operation is being performed, an exception could be generated to handle the occurrence. This leads to the following FV:

- Divisor is not checked for zero value before division operation is performed

The value of zero for the divisor variable could occur for many reasons. One reason is that the variable was never initialized and assumed the value of zero. This could be caused by input to the program that forces the execution of the program along a path that skips the initialization step. Alternatively, the divisor variable may be initialized, but input may cause the divisor variable to assume a value of zero. In each of these cases the divisor variable will be zero, but how it assumed that value can vary greatly. Therefore, there are many primitive conditions, but not every primitive condition applies to all instances.

The following FVs underlie some instances of division by zero:

- Initialization of variable is not performed
- Input verification does not exist or is faulty
- Program logic is faulty
- Exception is not generated in response to a fault condition
- Exception is not acted upon

Derivation of FVs through CWE analysis

Determining FVs can be thought of as answering three basic questions:

- What could have been done before to prevent the issue from happening?
- What could have been done during the occurrence of the issue to stop the issue from happening?
- What could have been done after the occurrence of the issue to negate what happened?

A few CWEs will now be analyzed to demonstrate how these questions could be used to determine FVs. Note that only a few CWEs are presented as examples as the goal is to determine the set of FVs, not to analyze all CWEs to determine their underlying FVs.

FVs of CWE-128 Wrap-around Error

- Check that an integer computation will not overflow available space does not exist or is faulty
- Exception is not generated in response to a fault condition

- Exception is not acted upon

FVs of CWE-311 Missing Encryption of Sensitive Data

- Sensitive data is exposed to unauthorized entity
- Cryptographic algorithm does not exist or is faulty

FVs of CWE-416 Use After Free

- Automatic management of buffers does not occur
- Exception is not generated in response to a fault condition

FVs of CWE-137 Representation errors

This is a broad category of weaknesses that are introduced when inserting or converting data from one representation to another. As a result there are many FVs:

- Different format types exist for numbers (e.g. character '5' and numerical 5)
- Dynamic typing is used
- Data is converted from one data type to another
- Data type is converted from one data size (within the same data type) to another
- Hardware is not standardized

FVs of CWE-798 Use of Hard-coded Credentials

The use of hard coded credentials is the general problem of sensitive data being available to an attacker. In order for the credentials to be useful to the attacker, they must be either hardcoded and/or reusable. In addition the sensitive data must be accessible by the unauthorized person. Therefore the FVs would be:

- Sensitive data is exposed to unauthorized entity (e.g. person or process)
- Sensitive data is hardcoded/reusable for use by a security function

One to One with CWE

Several CWEs translate into a single FV:

FVs of CWE-20 Improper input validation

- Input checks do not exist or are faulty

FVs of CWE-561 Dead Code

- Code exists in a program that is not on any execution path

List of Fundamental Vulnerabilities

The previous analyses and other analyses have been used to construct an initial set of FVs that are listed in the following table. The list is a start, it is definitely not complete. It is also not clear whether entries such as:

- Persons writing or managing the code development did not use generally accepted software development best practices

belong in this list of FVs. Entries such as these are fundamental issues that are the root of vulnerabilities, but these may be straying too far from tangible issues with programming languages.

More analysis will need to be conducted to ensure that the set of FVs is as accurate and complete as possible.

	Fundamental Vulnerability	Comment/Rationale	CWE reference entry
1.	Application does not have a dedicated resource pool	Resource availability would be unpredictable leading to resource exhaustion	
2.	Array bounds check before array access does not exist or is faulty		129: Unchecked Array Indexing
3.	Authentication check does not exist or is faulty		287: Improper Authentication
4.	Authentication credential is spoofed		290: Authentication Bypass by Spoofing
5.	Automatic management of buffers does not occur		
6.	Binary compilation is not functionally equivalent to its source	Either compiler mistake or intentional compiler miscompilation	14: Compiler Removal of Code to Clear Buffers
7.	Check that an integer computation will not overflow size of the integer does not exist or is faulty	Computation result exceeds size of data type	190: Integer Overflow or Wraparound
8.	Code and data are indistinguishable in memory	Code and data are not segregated in memory	
9.	Code exists in a program that is not on any execution path		561: Dead Code
10.	Cryptographic algorithm does not exist or is faulty		311: Missing Encryption of Sensitive Data
11.	Data is converted from one data type to another		681: Incorrect Conversion between Numeric Types
12.	Data type is converted from one data size (within the same data type) to another	e.g. converting 16 bit integer to 8 bit integer e.g. short, long int; single byte, multi-byte characters	681: Incorrect Conversion between Numeric Types

13.	Deprecated construct is used	e.g. as a language evolves, outdated construct is available, but not removed	
14.	Different format types exist for numbers	e.g. character '5' and numerical 5	
15.	Divisor is not checked for zero value before division operation is performed	Applies to both division and modulo operators	369: Divide By Zero
16.	Dynamic typing is used		
17.	Dynamically loaded code is used without authentication	e.g. use of dynamically linked resource	494: Download of Code Without Integrity Check
18.	Exception is not acted upon	Fault condition exception can be generated, but not acted upon	CWE-248: Uncaught Exception
19.	Exception is not generated in response to a fault condition		CWE-391: Unchecked Error Condition
20.	Function has a side effect		
21.	Function prototype is not used to specify function interface		CWE-628: Function Call with Incorrectly Specified Arguments
22.	Function return address is not immutable	Return address can be altered	
23.	Hardware is not standardized	Size of short, int, long differ between platforms	
24.	History and provenance is not available for use at authentication points	No basis for determining the integrity of dynamically linked or used resource	
25.	Incorrect use of a language construct or function due to ease of incorrect use	e.g. confusion of "=" and "==" This is definitely an issue, but is it something that should be included?	
26.	Index is calculated that is outside of the indexable resource		118 Improper Access of Indexable Resource ('Range Error')
27.	Initialization of variable is not performed	Assignment of value to variable is not performed before use in operation	665: Improper Initialization
28.	Input verification does not exist or is faulty		20. Improper input validation
29.	Interface between languages is in contrast with one or both languages		
30.	Limitation on the execution of code is insufficient to protect system	e.g. untrusted code may be dynamically linked and executed using admin privileges of main program	
31.	Multiple exceptions without prioritization generated from	e.g. No hierarchical order to exceptions	

	a single event		
32.	Multiple operations are needed to complete common functionality	e.g. cwe-227 calling chdir after calling chroot –this leads to only one part of a necessary operation being accomplished	
33.	Object passed by reference		
34.	Ownership of a resource expires	e.g. memory containing sensitive information can then be read by some other program	
35.	Persons writing or managing the code development did not use generally accepted software development best practices	This is definitely an issue, but is it something that should be included?	
36.	Persons writing or managing the code development were unaware of security issues	This is definitely an issue, but is it something that should be included?	
37.	Production code and debug code are indistinguishable from each other		489: Leftover Debug Code
38.	Program logic is faulty		
39.	Race condition for shared resource exists		362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')
40.	Relative pathname is used		23: Relative Path Traversal
41.	Resource assigned a permission allowing an unauthorized person to access		282: Improper Ownership Management
42.	Resource comes from an untrusted source		399: Resource Management Errors
43.	Resource is owned		399: Resource Management Errors
44.	Resource is shared without access control		402: Transmission of Private Resources into a New Sphere ('Resource Leak')
45.	Resource permissions are incorrect	e.g. unauthorized users allowed to use	732 Incorrect Permission Assignment for Critical Resource
46.	Resource remains allocated but is never used again		404: Improper Resource Shutdown or Release
47.	Resource transaction and consumption are not synced		399: Resource Management Errors
48.	Resource use does not have an imposed limit		CWE 770: Allocation of Resources without Limits of Trottling
49.	Security check is not performed local to the application	e.g. security check is performed on client for a server application	
50.	Sensitive data is exposed to unauthorized entity	(e.g. person or process)	200: Information Exposure

51.	Sensitive data is hardcoded/reusable for use by a security function		798: Use of Hard-coded Credentials
52.	Signature to verify integrity of dynamically loaded code is not available		
53.	Signed and unsigned data types are converted from one the other		195: Signed to Unsigned Conversion Error 196: Unsigned to Signed Conversion Error
54.	Signed integer is used where an unsigned integer could be used		
55.	Stack is used when calling a function	e.g. pointers to local variables can exist after the return	
56.	String termination sentinel character is not immutable	There is a duality of a string and a null terminated array	CWE-463: Deletion of Data Structure Sentinel
57.	There are multiple privilege levels available	This allows a permission that is not the minimum needed to be used when performing an operation	
58.	There is a difference between the actual size of a resource and the recorded size (due to the need for a sentinel)	e.g. string length is calculated incorrectly	
59.	There is a disconnect between a pointer and the resource that it represents		416: Use After Free 465: Pointer Issues
60.	There is an insufficient or non-existent restriction on a file access or command execution		
61.	There is non-constant scaling of pointers		465: Pointer Issues 469: Use of Pointer Subtraction to Determine Size 682: Incorrect Calculation
62.	There is syntactic ambiguity in the language		
63.	Type checking is weak or non-existent		
64.	Value returned from a call does not exist or is not as expected		394: Unexpected Status Code or Return Value