ISO/IEC JTC 1/SC 22/
OWGV N0140

OWG: Vulnerability

ISO working group on Guidance for Avoiding Vulnerabilities through language selection and use

John Benito, Convener
Jim Moore, Secretary

19 June 2008

OWG: Vulnerability Summary

- We are making progress!
  - meetings scheduled out over a year
  - Participation is good and is made up of a wide variety of technical expertise.
- Have passed initial SC22 ballot.
- On track to publish in 2009.
- HOWEVER…
  - We need the assistance of language working groups to develop language-specific annexes

The Problem

- Any programming language has constructs that are imperfectly defined, implementation dependent or difficult to use correctly.
- As a result, software programs sometimes execute differently than intended by the writer.
- In some cases, these vulnerabilities can be exploited by hostile parties.
  - – Can compromise safety, security and privacy.
  - – Can be used to make additional attacks.

Complicating Factors

- The choice of programming language for a project is not solely a technical decision and is not made solely by software engineers.
- Some vulnerabilities cannot be mitigated by better use of the language but require mitigation by other methods, e.g. review, static analysis.

An example

- While buffer overflow examples can be rather complex, it is possible to have very simple, yet still exploitable, stack based buffer overflows:
- An Example in the C programming language:

```
#include <string.h>
#define BUFSIZE 256

int main(int argc, char **argv) {
 char buf[BUFSIZE];

 strcpy(buf, argv[1]);
}
```

Example

- ❑ Buffer overflows generally lead to the application halting or crashing.
- ❑ Other attacks leading to lack of availability are possible, that can include putting the program into an infinite loop.
- ❑ Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy.

OWG:Vulnerability Product

- A type III Technical Report
  - A document containing information of a different kind from that which is normally published as an International Standard
- Project is to work on a set of *common mode* failures that occur across a variety of languages
  - Not all vulnerabilities are common to all languages, that is, some manifest in just a language
- The product will not contain *normative* statements, but information and suggestions

OWG:Vulnerability Product

- No single programming language or family of programming languages is to be singled out
  - As many programming languages as possible should be involved
  - Need not be just the languages defined by ISO Standards

Approach to Identifying Vulnerabilities

- *Empirical approach:* Observe the vulnerabilities that occur in the wild and describe them, e.g. buffer overrun, execution of unvalidated remote content
- *Analytical approach:* Identify potential vulnerabilities through analysis of programming languages
  - This just might help in identifying tomorrows vulnerabilities.

Audience

- *Safety*: Products where it is critical to prevent behavior which might lead to human injury, and it is justified to spend additional development money
- *Security*: Products where it is critical to secure data or access, and it is justified to spend additional development money
- *Mission-Critical*: Products where it is important to prevent behaviour that can lead to losses
- *Modeling and Simulation*: Products which require unusual reliability because the cost of computation is high

OWG: Vulnerability Progress

- Eight meetings have been held, hosted by
  - US
  - Italy
  - Canada
  - UK
  - Netherlands
- Meetings planned through 2008, hosted by
  - US (editors meeting)
  - Germany
- E-Mail reflector, Wiki and Web site are used during and between meetings
- More information
  - http://aitc.aitcnet.org/isai/

OWG: Vulnerability Participants

- Canada
- Germany
- Italy
- Japan
- France
- United Kingdom
- USA – CT 22
- SC 22/WG 9
- SC 22/WG14
- MDC (Mumps)
- SC 22/WG 5, INCITS J3 (Fortran)
- SC 22/WG 4, INCITS J4 (Cobol)
- ECMA (C#, C++CLI)
- RT/SC Java
- MISRA C/C++
- CERT

OWG: Vulnerability Status

- Response to NP Ballot comments is completed, see SC 22 N4027
- Project is organized and on schedule to produce a document in 2009
- Current draft passed the first SC 22 ballot (PDTR registration)
- The project has two officers
  - – Convener/Project Editor, John Benito
  - – Secretary, Jim Moore

Measure of Success

- Provide guidance to users of programming languages that:
  - Assists them in improving the predictability of the execution of their software even in the presence of an attacker
  - Informs their selection of an appropriate programming language for their job
- Provide feedback to programming language standardization groups, resulting in the improvement of programming language standards.

Vulnerability Template

- The body of Technical Report describes vulnerabilities in a generic manner, including:
  - Brief description of application vulnerability
  - Cross-reference to enumerations and other classifications, e.g. CWE
  - Description of failure mechanism, i.e. how coding problem relates to application vulnerability
  - Applicable language characteristics
  - Avoiding or mitigating the vulnerability
  - Implications for standardization
- Annexes will provide language-specific treatments of each vulnerability.
- The following slides provide an example using XZH, the Off-by-One Error

Description of application vulnerability

- A product uses an incorrect maximum or minimum value that is 1 more or 1 less than the correct value. This usually arises from one of a number of situations where the bounds as understood by the developer differ from the design, such as;
  - Confusion between the need for “<” and “<=” or “>” and “>=” in a test.
  - Confusion as to the index range of an algorithm, such as beginning an algorithm at 1 when the underlying structure is indexed from 0, beginning an algorithm at 0 when the underlying structure is indexed from 1 (or some other start point) or using the length of a structure as the bounds instead of the sentinel values.
  - Failing to allow for storage of a sentinel value, such as the ‘\0’ string terminator that is used by the C and C++ programming languages.

Continued…

- These issues arise from mistakes in mapping the design into a particular language, in moving between languages (such as between C-based languages where all arrays start at 0 and other languages where arrays often start at 1), and when exchanging data between languages with different default array sentinel values.
- The issue also can arise in algorithms where relationships exist between components, and the existence of a sentinel value changes the conditions of the test.
- The existence of this possible flaw can also be a serious security hole as it can permit someone to surreptitiously provide an unused location (such as 0 or the last element) that can be used for undocumented features or hidden channels).

Cross-reference to enumerations

- CWE:
  - 193. Off-by-one Error
- [May add MISRA C and C++, CERT, JSF, others]

Mechanism of Failure

- An off-by-one error could lead to.
  - an out-of bounds access to an array (buffer overflow),
  - an incomplete comparisons and calculation mistakes,
  - a read from the wrong memory location, or
  - an incorrect conditional.
- Such incorrect accesses can cause cascading errors or references to illegal locations, resulting in potentially unbounded behaviour.
- Off-by-one errors are not often exploited in attacks because they are difficult to identify and exploit externally, but the cascading errors and boundary-condition errors can be severe.

Applicable Language Characteristics

- As this vulnerability arises because of an algorithmic error by the developer, it can in principle arise in any language; however, it is most likely to occur when:
  - the language relies on the developer having implicit knowledge of structure start and end indices (e.g., knowing whether arrays start at 0 or 1 – or indeed some other value)
  - where the language relies upon explicit sentinel values to terminate variable length arrays

Avoiding the Vulnerability or Mitigating its Effects

- Software developers can avoid the vulnerability or mitigate its ill effects in the following ways:
  - Off-by-one errors are a common bug that is also a code quality issue. As with most quality issues, a systematic development process, use of development/analysis tools and thorough testing are all common ways of preventing errors, and in this case, off-by-one errors.

Continued…

- Where references are being made to structure indices and the languages provide ways to specify the whole structure or the starting and ending indices explicitly (e.g., Ada provides xxx'First and xxx'Last for each dimension), these should be used always. Where the language doesn't provide these, constants can be declared and used in preference to numeric literals.
- Where the language doesn’t encapsulate variable length arrays, encapsulation should be provided through library objects and a coding standard developed that requires such arrays to only be used via those library objects, so the developer does not need to be explicitly concerned with managing sentinel values

Implications for Standardization

- Languages should provide encapsulation for arrays that:
  - Prevent the need for the developer to be concerned with explicit sentinel values.
  - Provide the developer with symbolic access to the array start, end and iterators.

OWG: Vulnerability Summary

- We are making progress!
  - meetings scheduled out over a year
  - Participation is good and is made up of a wide variety of technical expertise.
- Have passed initial SC22 ballot.
- On track to publish in 2009.
- HOWEVER…
  - We need the assistance of language working groups to develop language-specific annexes

Discussion

- Should not go to ballot without language-specific annexes because the presence of the annexes would change the main document:
  - Examples might move to annexes.
  - There may be resistance to changing the main document after it has been successfully balloted.
  - Adding annexes after the initial TR is approved would give the impression of instability.
- Doing prototype of language-specific annexes
  - Select a subset of descriptions and write a sample annex
  - To experiment with formats
  - To look at what kind of changes to the main document would be appropriate
- The OWGV should probably provide a template as a starting point for a language-specific annex.
- Alan Burns will provide leadership. Participants: Barnes, Ploedereder, Vardanega, Michell, Schonberg, Rosen.
- August review should be cc-ed to working groups as well as the November PDTR.