ISO/IEC JTC 1/SC 22/OWGV N 0119

# A new type of Working Group used for a new SC22 Working Group

## OWG: Vulnerability

John Benito

JTC 1/SC 22 WG14 Convener
INCITS CT 22 Vice Chairman
JTC 1/SC 22 OWG:V Convener

# 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.

# Vulnerability Template

- The body of Technical Report describes vulnerabilities in a generic manner, including:
  - Brief description of application vulnerability
  - Cross-reference to enumerations, e.g. CWE
  - Categorizations by selected characteristics
  - Description of failure mechanism, i.e. how coding problem relates to application vulnerability
  - Points at which the causal chain could be broken
  - Assumed variations among languages
  - Ways to avoid the vulnerability or mitigate its effects
- Annexes will provide language-specific treatments of each vulnerability.

# Description of 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 sentinels (start point and end point) for 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 or a structure as the count mechanism instead of the sentinel values

# Cross-reference to enumerations

- CWE:
  - 193. Off-by-one Error

# Description of failure mechanism

- 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 calculation errors or references to illegal locations, resulting in potentially unbounded behaviour.
- Off-by-one errors are not exploited as often in attacks because they are difficult to identify and exploit externally, but the calculation errors and boundary-condition errors can be severe.

# Ways to avoid the vulnerability

- Off-by-one errors are a common defect 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.
- 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 (eg 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.
- Coding standards can be written such that either the sentinel values or the length of all arrays is used. Ideally length should be a calculated function of the indices.

# 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
- The project has two officers
  - – Convener/Project Editor, John Benito
  - – Secretary, Jim Moore

# OWG: Vulnerability Status

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

# Meeting Schedule for OWG:V

- Meeting #6 2007-10-1/3 INCITS/Plum Hall, Kona, Hawaii, USA
- Meeting #7 2007-12-12/14 INCITS/SEI, Pittsburgh, PA, USA
- Meeting #8 2008-04-09/11 NEN/ACE, Amsterdam, NL
- Meeting #9 2008-07 INCITS/Blue Pilot, Washington DC, USA
- Meeting #10 2008-10 – Stuttgart, Germany

# 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 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
- *Predictability*: Products where high confidence in the result of the computation is desired
- *Assurance*: Products to be developed for dependability or other important characteristics

# 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.

# 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 a document that is ready for the first SC 22 ballot (registration).
- On track to publish in 2009.