**Software Security Knowledge: Training**

**Presented at the 23rd Systems and Software Technology Conference (SSTC), 16-19 May 2011, Salt Lake City, UT. Sponsored in part by the USAF. U.S. Government or Federal Rights License**
Agenda

8:00-8:45am  Software Security Knowledge about Applications Weaknesses
9:00-9:45am  Software Security Knowledge about Attack Patterns Against Applications

Training in Software Security

10:15-11:00am Software Security Practice
11:15-12:00am Supporting Capabilities

Assurance Cases
Secure Development & Secure Operations
CWE is Meant for People to Use
A complete body of knowledge covering the entire field of software engineering may be years away. However, the body of knowledge needed by professionals to create software free of common and critical security flaws has been developed, vetted widely and kept up to date. That is the foundation for a certification program in software assurance that can gain wide adoption. It was created in late 2008 by a consortium of national experts, sponsored by DHS and NSA, and was updated in late 2009. It contains ranked lists of the most common errors, explanations of why the errors are dangerous, examples of those errors in multiple languages, and ways of eliminating those errors. It can be found at http://cwe.mitre.org/top25.

Any programmer who writes code without being aware of those problems and is not capable of writing code free of those errors is a threat to his or her employers and to others who use computers connected to systems running his or her software.
The Certified Secure Software Lifecycle Professional (CSSLP) Certification Program will show software lifecycle stakeholders not only how to implement security, but how to glean security requirements, design, architect, test and deploy secure software.

An Overview of the Steps:

(ISC)² 5-day CSSLP CBK® Education Program
Educate yourself and learn security best practices and industry standards for the software lifecycle through the CSSLP Education Program. (ISC)² provides education your way to fit your life and schedule. Completing this course will, not only teach all of the necessary components to establish a security plan across your organization, but also prepare you for the CSSLP certification exam.
References

[ISO/IEC 9899:1999] Section 5.1.1.3, "Diagnostics"

[MITRE 07] CWE ID 563, "Unused Variable"; CWE ID 570, "Expression is Always False"; CWE ID 571, "Expression is Always True"

[Sutter 05] Item 1

[Seacord 05a] Chapter 8, "Recommended Practices"
Industry Uptake

CWE CAPEC

Making Security Measurable

Manually review code after security education

Manual code review, especially review of high-risk code, such as code that faces the Internet or parses data from the Internet, is critical, but only if the people performing the code review know what to look for and how to fix any code vulnerabilities they find. The best way to help understand classes of security bugs and remedies is education, which should minimally include the following areas:

- C and C++ vulnerabilities and remedies, most notably buffer overruns and integer arithmetic issues.
- Web-specific vulnerabilities and remedies, such as cross-site scripting (XSS).
- Database-specific vulnerabilities and remedies, such as SQL injection.
- Cryptographic vulnerabilities, such as weak random number generation.

Many vulnerabilities are programming language (C, C++, etc.) or domain-specific (web, database) and others can be categorized by vulnerability type, such as injection (XSS and SQL injection) or cryptographic (over random number generation) so specific training in these areas is advised.

Resources


Testing

Testing activities validate the secure implementation of a product, which reduces the likelihood of security bugs being released and discovered by customers or malicious users. The majority of SAFECode members have adopted the following testing techniques as part of their software development process.

Fuzz testing

Fuzz testing is a reliability and security testing technique that relies upon both intentionally malformed data and then having the software under test consume that malformed data to see how it responds. The science of fuzz testing is somewhat new but it is maturing rapidly. There is a small market for fuzz testing tools today, but in many cases software developers must build bespoke fuzz tools to suit specialized file and network data formats. Fuzz testing is an effective testing technique because it uncovers weaknesses in data handling code.

Hardware/Software Security Testing Methods, Version 2.0

- Common-cause Systems,symmetry, and Usability: http://capec.mitre.org/
The Security Development Lifecycle: MS08-078 and the SDL

Recent Posts
MS08-078 and the SDL
Announcing CAT.NET CTD and AntYSS V3 beta
SDL videos
BlueHat SDL Sessions Wrap-up
Secure Coding Secrets?

Tags
Common Criteria Crawl Walk Run
Privacy SDL SDL Pro Network
Security Assessment Security Bookshot
SDL threat modeling

News
Blogroll
BlueHat Security Briefings
The Microsoft Security Response Center
Michael Howard's Web Log
The Data Privacy Imperative
Security Vulnerability Research & Defense
Visual Studio Code Analysis blog
MSRC Ecosystem Strategy Team

Books / Papers / Guidance
The Security Development Lifecycle (Howard and Lipner)
Privacy Guidelines for Developing Software Products and Services
Microsoft Security Development Lifecycle (SDL) – Portal
Microsoft Security Development Lifecycle (SDL) – Process Guidance (Web)

MS08-078 and the SDL

Hi, Michael here.

Every bug is an opportunity to learn, and the security update that fixed the data binding bug that affected Internet Explorer users is no exception.

The Common Vulnerabilities and Exposures (CVE) entry for this bug is CVE-2008-8944.

Before I get started, I want to explain the goals of the SDL and the security work here at Microsoft. The SDL is designed as a multi-layered process to help systematically reduce security vulnerabilities; if one component of the SDL process fails to prevent or catch a bug, then some other component should prevent or catch the bug. The SDL also mandates the use of security defenses whose impact will be reflected in the "mitigations" section of a security bulletin, because we know that no software development process will catch all security bugs. As we have said many times, the goal of the SDL is to "Reduce vulnerabilities, and reduce the severity of what's missed."

In this post, I want to focus on the SDL-required code analysis, code review, fuzzing and compiler and operating system defenses and how they faired.

Background

The bug was an invalid pointer dereference in MSHTML.DLL when the code handles data binding. It's important to point out that there is no heap corruption and there is no heap-based buffer overrun!

When data binding is used, IE creates an object which contains an array of data binding objects. In the code in question, when a data binding object is released, the array length is not correctly updated leading to a function call into freed memory.

The vulnerable code looks a little like this (by the way, the real array name is ArrayOfObjectsFromI3 but I figured ArrayOfObjectsFromI3 is a little more descriptive for people not in the Internet Explorer team):

```c
int MaxIdx = ArrayOfObjectsFromI3.Size();
for (int i = 0; i < MaxIdx; i++) {
    if (!ArrayOfObjectsFromI3[i])
        continue;
    ArrayOfObjectsFromI3[i]->TransferFromSource();
}
```

Here's how the vulnerability manifests itself: if there are two data transfers with the same identifier (so MaxIdx is 2), and the first transfer updates the length of the ArrayOfObjectsFromI3 array when its work was done and releases its data binding object, the loop count would still be whatever MaxIdx was at the start of the loop.

This is a time-of-check-time-of-use (TOCTOU) bug that led to code calling into a freed memory block. The Common Weakness Enumeration (CWE) classification for this vulnerability is CWE-367.

The fix was to check the maximum iteration count on each loop iteration rather than once before the loop starts. This is the correct fix for a TOCTOU bug, move the check as close as possible to the action because it was time of check time of use.

Copyright © 2011 MITRE

Fuzz Testing

a time-of-check-time-of-use (TOCTOU) bug that led to code calling into a freed memory block. The Common Weakness Enumeration (CWE) classification for this vulnerability is CWE-367.
Our methodology for the Top 10 2007 was simple: take the MITRE Vulnerability Trends for 2006, and distill the Top 10 web application security issues. The ranked results are as follows:
Introduction

Code review is probably the single-most effective technique for identifying security flaws. When used together with automated tools and manual penetration testing, code review can significantly increase the cost-effectiveness of an application security verification effort.

This guide does not prescribe a process for performing a security code review. Rather, this guide focuses on the mechanics of reviewing code for certain vulnerabilities, and provides limited guidance on how the effort should be structured and executed. OWASP intends to develop a more detailed process in a future version of this guide.

Manual security code review provides insight into the "real risk" associated with insecure code. This is the single most important value from a manual approach. A human reviewer can understand the context for certain coding practices, and make a serious risk estimate that accounts for both the likelihood of attack and the business impact of a breach.

Why Does Code Have Vulnerabilities?

MITRE has catalogued almost 700 different kinds of software weaknesses in their CWE project. These are all different ways that software developers can make mistakes that lead to insecurity. Every one of these weaknesses is subtle and many are seriously tricky. Software developers are not taught about these weaknesses in school and most do not receive any training on the job about these problems.

These problems have become so important in recent years because we continue to increase connectivity and to add technologies and protocols at a shocking rate. Our ability to invent technology has seriously outstripped our ability to secure it. Many of the technologies in use today simply have not received any security scrutiny.

There are many reasons why businesses are not spending the appropriate amount of time on security. Ultimately, these reasons stem from an underlying problem in the software market. Because software is essentially a black-box, it is extremely difficult to tell the difference between good code and insecure code. Without this visibility, buyers won't pay more for secure code, and vendors would be foolish to spend extra effort to produce secure code.

One goal for this project is to help software buyers gain visibility into the security of software and start to effect change in the software market.

Nevertheless, we still frequently get pushback when we advocate for security code review. Here are some of the (unjustified) excuses that we hear for not putting more effort into security:

"We never get hacked (that I know of), we don't need security"
Some High-Level CWEs Are Now Part of the NVD CVE Information

**Overview**

SQL injection vulnerability in mod/banners/nextlist.php in Clansphere 2007.4 allows remote attackers to execute arbitrary SQL commands via the cat_id parameter to index.php in a banners action.

**Impact**

CWE Severity (version 2.0):

CVSS v2 Base score: 2.5 (High) AV:N/AC:L/AV:N/AC:2/AV:N/AC:1 (legend)

Impact Subscore: 6.4

Exploitability Subscore: 10.0

Access Vector: Network exploitable

Access Complexity: Low

Authentication: Not required to exploit


**References to Advisories, Solutions, and Tools**

External Source: BID (Excluded)

Name: 25770

Hyperlink: http://www.securityfocus.com/bid/25770

External Source: MILWORM (Excluded)

Name: 4443

Hyperlink: http://www.milworm.com/exploits/4443

**Vulnerable software and versions**

Configuration 1

- Clansphere, Clansphere, 2007.4

**Technical Details**

Vulnerability Type (View All)

SQL Injection (CWE-89)

**NVD XML feeds also include CWE**

**Vulnerability Type (View All)**

SQL Injection (CWE-89)

**CWE-89 Individual Dictionary Definition (Draft 5)**

**Description**

Failure to Sanitize Data into SQL Queries (aka 'SQL Injection' or 'Injection vulnerability')

**Summary**

This application failure to adequately filter input from user-controllable input. This can lead to such input being interpreted as SQL rather than arbitrary user data and executed as part of a dynamically generated SQL query. This is a specific form of an injection problem, one that directly affects SQL databases, in which SQL commands are embedded into data sent to input in order to effect the execution of a dynamically generated SQL statement.

**Likelihood of Exploit**

Confidentiality: Site, SQL database. Can be modified to include sensitive data, loss of confidentiality is a frequent problem with SQL injection vulnerabilities.

Authentication: SQL injection commands are entered as user names and passwords. It may be possible to use this information to log in to the application as an account with no previous knowledge of the password.

Authorization: If authentication information is injected into a SQL database, it may be possible to access the information through the execution of a SQL script. XSS injection vulnerability. 

Integrity: Just as it may be possible to read sensitive information, it is also possible to make changes to data and alter data within this database, which is accessible to the victim.

**Potential Mitigations**

Requirements specification: A non-SQL, style database which is not a subject to this flaw may be used.

Design: Follow the principle of least privilege when creating user accounts to a SQL database. Users should only have the minimum privilege necessary to use the account. If the requirements of the system indicate that a user need not read and modify their own data, then limit their privilege to only read data from others.

Design: Disallow any filtering done on the client side or the server side. Implementation: Implement SQL tricks using prepared statements that bind variables. Prepared statements that do not bind variables can be vulnerable to attacks.
NIST Special Publications:
SP500-268  CWE
SP500-269  CWE
SP800-53a  CVE, OVAL, CWE
SP800-115  CVE, CCE, CVSS, CWE

NIST Interagency Reports:
NISTIR-7435  CVE, CVSS, CWE
NISTIR-7628  CVE, CWE
(Idaho National Labs SCADA Report)

NSTB Assessments
Summary Report:
Common Industrial Control System Cyber Security Weaknesses

May 2010

Level 4
Enterprise Systems:
Business Planning and Logistics / Engineering Systems

Level 3
Operations Management:
System Management / Supervisory Control

Level 2
Supervisory Control Equipment:
Supervisory Control Functions / Site Monitoring and Local Display

Level 1
Control Equipment:
Protection and Local Control Devices

Level 0
Equipment Under Control:
Sensors and Actuators

ISC Web Application Client
ICS Business Application Client
Corporate Hosts
Replicated Database
Web Server
ICCP Server
OPC Server
Information Server Application Server

Historical Database
Real-time Database
Communications Processor

Supervisory Control LAN

RTU/DAM
Distributed Control
PLC

Corporate Network
Business Servers
LAN / WAN / DMZ

Remote Vendor or Engineer Access

Secure Control System/Enterprise Architecture
Table 27. Most common programming errors found in ICS code.

<table>
<thead>
<tr>
<th>Weakness Classification</th>
<th>Vulnerability Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-19: Data Handling</td>
<td>CWE-228: Improper Handling of Syntactically Invalid Structure</td>
</tr>
<tr>
<td></td>
<td>CWE-229: Improper Handling of Values</td>
</tr>
<tr>
<td></td>
<td>CWE-230: Improper Handling of Missing Values</td>
</tr>
<tr>
<td></td>
<td>CWE-20: Improper Input Validation</td>
</tr>
<tr>
<td></td>
<td>CWE-116: Improper Encoding or Escaping of Output</td>
</tr>
<tr>
<td></td>
<td>CWE-195: Signed to Unsigned Conversion Error</td>
</tr>
<tr>
<td></td>
<td>CWE-198: Use of Incorrect Byte Ordering</td>
</tr>
<tr>
<td>CWE-119: Failure to Constrain Operations within the Bounds of a Memory Buffer</td>
<td>CWE-120: Buffer Copy without Checking Size of Input (&quot;Classic Buffer Overflow&quot;)</td>
</tr>
<tr>
<td></td>
<td>CWE-121: Stack-based Buffer Overflow</td>
</tr>
<tr>
<td></td>
<td>CWE-122: Heap-based Buffer Overflow</td>
</tr>
<tr>
<td></td>
<td>CWE-125: Out-of-bounds Read</td>
</tr>
<tr>
<td></td>
<td>CWE-129: Improper Validation of Array Index</td>
</tr>
<tr>
<td></td>
<td>CWE-131: Incorrect Calculation of Buffer Size</td>
</tr>
<tr>
<td></td>
<td>CWE-170: Improper Null Termination</td>
</tr>
<tr>
<td></td>
<td>CWE-190: Integer Overflow or Wraparound</td>
</tr>
<tr>
<td></td>
<td>CWE-680: Integer Overflow to Buffer Overflow</td>
</tr>
<tr>
<td>CWE-398: Indicator of Poor Code Quality</td>
<td>CWE-454: External Initialization of Trusted Variables or Data Stores</td>
</tr>
<tr>
<td></td>
<td>CWE-456: Missing Initialization</td>
</tr>
<tr>
<td></td>
<td>CWE-457: Use of Uninitialized Variable</td>
</tr>
<tr>
<td></td>
<td>CWE-476: NULL Pointer Dereference</td>
</tr>
<tr>
<td></td>
<td>CWE-400: Uncontrolled Resource Consumption (&quot;Resource Exhaustion&quot;)</td>
</tr>
<tr>
<td></td>
<td>CWE-252: Unchecked Return Value</td>
</tr>
<tr>
<td></td>
<td>CWE-690: Unchecked Return Value to NULL Pointer Dereference</td>
</tr>
<tr>
<td></td>
<td>CWE-772: Missing Release of Resource after Effective Lifetime</td>
</tr>
<tr>
<td>CWE-442: Web Problems</td>
<td>CWE-22: Improper Limitation of a Pathname to a Restricted Directory (&quot;Path Traversal&quot;)</td>
</tr>
<tr>
<td></td>
<td>CWE-79: Failure to Preserve Web Page Structure (&quot;Cross-site Scripting&quot;)</td>
</tr>
<tr>
<td></td>
<td>CWE-89: Failure to Preserve SQL Query Structure (&quot;SQL Injection&quot;)</td>
</tr>
<tr>
<td>CWE-703: Failure to Handle Exceptional Conditions</td>
<td>CWE-431: Missing Handler</td>
</tr>
<tr>
<td></td>
<td>CWE-248: Uncaught Exception</td>
</tr>
<tr>
<td></td>
<td>CWE-755: Improper Handling of Exceptional Conditions</td>
</tr>
<tr>
<td></td>
<td>CWE-390: Detection of Error Condition Without Action</td>
</tr>
</tbody>
</table>
Top 25 Series – Summary and Links

As requested here are the links to all the posts on the Top 25 Most Dangerous Programming Errors. Please let us know if you have any suggestions or comments.

1 – Cross-Site Scripting (XSS)
2 – SQL Injection
3 – Classic Buffer Overflow
4 – Cross-Site Request Forgery (CSRF)
5 – Improper Access Control (Authorization)
6 – Reliance on Untrusted Inputs in a Security Decision
7 – Path Traversal
8 – Unrestricted Upload of Dangerous File Type
9 – OS Command Injection
10 – Missing Encryption of Sensitive Data
11 – Hardcoded Credentials
12 – Buffer Access with Incorrect Length Value
13 – PHP File Inclusion
14 – Improper Validation of Array Index
15 – Improper Check for Unusual or Exceptional Conditions
16 – Information Exposure Through an Error Message
17 – Integer Overflow Or Wraparound
18 – Incorrect Calculation of Buffer Size
19 – Missing Authentication for Critical Function
20 – Download of Code Without Integrity Check
21 – Incorrect Permission Assignment for Critical Response
22 – Allocation of Resources Without Limits or Throttling
23 – Open Redirect
24 – Use of a Broken or Risky Cryptographic Algorithm
25 – Race Conditions
SDL and the CWE/SANS Top 25

Bryan here. The security community has been buzzing since SANS and MITRE’s joint announcement earlier this month of their list of the Top 25 Most Dangerous Programming Errors. Now, I don’t want to get into a debate in this blog about whether this new list will become the new de facto standard for analyzing security vulnerabilities (or indeed, whether it already has become the new standard). Instead, I’d like to present an overview of how the Microsoft SDL maps to the CWE/SANS list, just as I did for May.

Michael and I have written coverage of the Top 25 and believe that the results to the Top 25 were developed independently root them out of the software analysis white paper and guidance around every major made many of the same steps for you to download and.

Below is a summary of how the SDL covers every one of them (race conditions and by multiple SDL requirements and tools to prevent or detect).
CWE Outreach: A Team Sport

May/June Issue of IEEE Security & Privacy...

Improving Software Security by Eliminating the CWE Top 25 Vulnerabilities

In January 2009, MITRE® and SANS issued the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors® to help make developers more aware of the bugs that can cause security compromises.

CWE-209: Improper Input Validation

The scope of secure software development includes ensuring that all input is validated against expected input.

CWE-119: Improper Error Handling

Improper error handling in software can expose applications to code execution, arbitrary file read, and other vulnerabilities.

CWE-89: Improper Neutralization of Special Elements in SQL Command

Improper neutralization of special elements in SQL commands can allow an attacker to perform SQL injection attacks.

CWE-668: Improper Access Control

Improper access control can allow a malicious attacker to access data that they are not authorized to access.

CWE-91: Improper Account Management

Improper account management in software can allow a malicious attacker to access data that they are not authorized to access.

CWE-80: Improper Input Validation

Improper input validation in software can allow a malicious attacker to execute arbitrary code or conduct other malicious activities.

CWE-565: Use of Error Prone Function

Use of error prone functions in software can lead to security vulnerabilities.

CWE-79: Improper Neutralization of Special Elements in SQL Command

Improper neutralization of special elements in SQL commands can allow an attacker to perform SQL injection attacks.

CWE-91: Improper Account Management

Improper account management in software can allow a malicious attacker to access data that they are not authorized to access.

CWE-89: Improper Neutralization of Special Elements in SQL Command

Improper neutralization of special elements in SQL commands can allow an attacker to perform SQL injection attacks.

CWE-668: Improper Access Control

Improper access control can allow a malicious attacker to access data that they are not authorized to access.

CWE-119: Improper Error Handling

Improper error handling in software can expose applications to code execution, arbitrary file read, and other vulnerabilities.

CWE-89: Improper Neutralization of Special Elements in SQL Command

Improper neutralization of special elements in SQL commands can allow an attacker to perform SQL injection attacks.

CWE-668: Improper Access Control

Improper access control can allow a malicious attacker to access data that they are not authorized to access.

CWE-79: Improper Neutralization of Special Elements in SQL Command

Improper neutralization of special elements in SQL commands can allow an attacker to perform SQL injection attacks.

CWE-91: Improper Account Management

Improper account management in software can allow a malicious attacker to access data that they are not authorized to access.

CWE-565: Use of Error Prone Function

Use of error prone functions in software can lead to security vulnerabilities.
[비즈니스 임팩트를 줄여주는 새로운 품질 관리 방법론]

y5를 사용하여, 소프트웨어 결함을 없애는 5가지 스텝은 아래와 같습니다.

1. 스캔 소프트웨어
2. 검출 결합 우선순위
3. 「ビジネ스インパクトから考える新しい品質管理」
4. 수정 우선순위는 결합부터
5. 리포트 결합수정상정

Coveity5을 사용하여, 소프트웨어 결함을 쉽게 제거하는 5단계는 아래에 설명합니다.

1. 스캔 소프트웨어
2. 검출 결함 우선순위
3. 맵핑 소프트웨어
4. 수정 우선순위
5. 리포트 결합수정상정
### Threat Classification Taxonomy Cross Reference View

This view contains a mapping of the WASC Threat Classification's Attacks and Weaknesses with MITRE's Common Weakness Enumeration, MITRE's Common Attack Pattern Enumeration and Classification, OWASP Top Ten 2010 RC1 (original mapping with OWASP Top Ten from Jeremiah Grossman & Bill Corry) and SANS/CWE and OWASP Top Ten 2007 and 2004 (original mapping from Dan Cornell, Denim Group).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WASC-01</td>
<td>Insufficient Authentication</td>
<td>287</td>
<td></td>
<td>642</td>
<td>A3 - Broken Authentication and Session Management, A4 - Insecure Direct Object References</td>
<td>A7 - Broken Authentication and Session Management, A4 - Insecure Direct Object Reference</td>
<td>A3 - Broken Authentication and Session management, A2 - Broken Access Control</td>
</tr>
<tr>
<td>WASC-02</td>
<td>Insufficient Authorization</td>
<td>284</td>
<td></td>
<td>285</td>
<td>A4 - Insecure Direct Object References, A7 - Failure to Restrict URL Access, A4 - Insecure Direct Object Reference</td>
<td>A10 - Failure to Restrict URL Access, A4 - Insecure Direct Object Reference</td>
<td>A2 - Broken Access Control</td>
</tr>
<tr>
<td>WASC-03</td>
<td>Integer Overflows</td>
<td>199</td>
<td>128</td>
<td>682</td>
<td>A10 - Insufficient Transport Layer Protection, A9 - Insecure Communications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WASC-04</td>
<td>Insufficient Transport Layer Protection</td>
<td>311 523</td>
<td>319</td>
<td>426</td>
<td>A5 - Malicious File Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WASC-05</td>
<td>Remote File Inclusion</td>
<td>98</td>
<td>233</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WASC-06</td>
<td>Format String</td>
<td>134</td>
<td>67</td>
<td>426</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WASC-07</td>
<td>Buffer Overflow</td>
<td>119 120</td>
<td>10 100</td>
<td>119</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WASC-08</td>
<td>Cross-site Scripting</td>
<td>79</td>
<td>18 19 63</td>
<td>79</td>
<td>A2 - Cross-Site Scripting</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WASC-09</td>
<td>Cross-site Request Forgery</td>
<td>352</td>
<td>62</td>
<td>352</td>
<td>A5 - Cross Site Request Forgery (XSS)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WASC-10</td>
<td>Denial of Service</td>
<td>400</td>
<td>119</td>
<td>404</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Test and vulnerability assessment

Testing applications for security defects should be an integral and organic part of any software testing process. During security testing, organizations should test to help ensure that the security requirements have been implemented and the product is free of vulnerabilities.

The SEF refers to the MITRE Common Weakness Enumeration\(^5\) (CWE) list and the Common Vulnerability Scoring System (CVSS) metrics to identify and assess vulnerabilities. CVSS metrics are used to measure the complexity of the attack vector, the impact of the exploit, and the availability and criticality of the system. CVSS metrics help organizations prioritize their vulnerability management activities.

Creating a security improvement plan includes:

- Investigating common development processes and the IBM Integrated Product Development process
- Emphasizing security awareness and requirements in the software development process
- Discussing test and vulnerability assessments

Security in Development: The IBM Secure Engineering Framework

1. Security in Development: The IBM Secure Engineering Framework
2. Investigating common development processes and the IBM Integrated Product Development process
3. Emphasizing security awareness and requirements in the software development process
4. Discussing test and vulnerability assessments
Making the Business Case for Software Assurance

Nancy R. Mead
Julie H. Allen
W. Arthur Conlin
Antonio Diommi
Jenny Harrison
Jeff Ingalsbe
James Rainey
Don Shoemaker

April 2009

SPECIAL REPORT
CMU/SEI-2009-SR-001

CERT Program
Unrestricted distribution subject to the copyright.

http://www.sei.cmu.edu

Software Engineering Institute

Carnegie Mellon

OVM: An Ontology for Vulnerability Management

Ju An Wang & Minho Guo
Southern Polytechnic State University
1100 South Marietta Parkway
Marietta, GA 30060
(01) 678-915-3718
jwang@spsu.edu

ABSTRACT

In order to reach the goals of the Information Security Automation Program (ISAP), we propose an ontological approach to capturing and utilizing the fundamental concepts in information security and their relationship, retrieving vulnerabilities data and reasoning about the cause and impact of vulnerabilities. Our ontology for vulnerability management (OVM) has been populated with all vulnerabilities in NVD [2] with additional inference rules, knowledge representation, and data-referencing mechanisms. With the seamless integration of common vulnerabilities and their related concepts such as attacks and countermeasures, OVM provides a powerful pathway to ranking ISAP successful.

Categories and Subject Descriptors

C.2.6 [Computer-Communication Networks]: General [Security and protection]; K.6.5 [Management of Computing and Information Systems]: Security and Protection;

General Terms

Ontology, Security, Vulnerability Analysis and Management

Keywords

Security, vulnerability, Semantic technology, Ontology, Vulnerability analysis

1. INTRODUCTION

The Information Security Automation Program (ISAP) is a U.S. government multi-agency initiative to enable automation and standardization of technical security operations [1]. In high-level goals include enabling standards based communication of vulnerability data, enhancing and managing configuration policies for various IT products, assessing information systems and reporting compliance status, using standard metrics to weight and aggregate potential vulnerability impact, and remaining identified vulnerabilities [1]. Secure computer systems ensure that confidentiality, integrity, and availability are maintained for users, data, and other information assets. Over the last few decades, a significant amount of knowledge has been accumulated in the area of information security. However, a lot of concepts in information security are vaguely defined and sometimes they have different semantics in different contexts, causing misinterpretation among stakeholders due to the language ambiguity. On the other hand, the standardization, design and development of security tools (1-5) require a systematic classification and definition of security concepts and techniques. It is important to have a clearly defined vocabulary and standardized language as means to accurately communicate system vulnerability information and their countermeasures among all the people involved. We believe that Semantic technology and ontology in particular, could be a useful tool for system security. Our research work has sought to define and develop web pages with simple means of information management.

An ontology is a specification of concepts and their relationships. Ontology represents knowledge in a formal and structured form. Therefore, ontology provides a better tool for communication, reasoning, and organization of knowledge. Ontology as a knowledge representation (KR) system based on Description Logics (DLs) [6], which is an umbrella name for a family of KR formalisms representing knowledge in various domains. The DL formalisms specify a knowledge domain as the "world" by first defining the relevant concepts of the domain, and then it uses these concepts to specify properties of objects and individuals accessing the domain [16-12]. Semantic technologies not only provide a tool for communication, but also a foundation for high-level reasoning and decision-making. Ontology, in particular, provides the potential of formal logic inference based on well-defined data and knowledge bases. Ontology captures the relationships between collected data and the explicit knowledge of concepts and relationships to deduce the implicit and inherent knowledge. As a matter of fact, a heavy-weight ontology can be defined as a formal logic system, as it includes facts and rules, concepts, concept taxonomies, relationships, properties, axioms and constraints.

A vulnerability is a security flaw, which arises from computer system design, implementation, maintenance, and operation. Research in the area of vulnerability analysis focuses on discovery of previously unknown vulnerabilities and quantification of the security of systems according to some metrics. Researchers at MITRE have provided a standard format for naming a security vulnerability, called Common Vulnerabilities and Exposures (CVE) [14], which assigns each vulnerability a unique identification number. We have designed a vulnerability ontology OVM (ontology for vulnerability management) populated with all existing vulnerabilities in NVD [2]. It supports research on reasoning about vulnerabilities and characterizations of vulnerabilities and their impact on computing systems. Vendors and users can use our ontology to support vulnerability analysis, tool development and vulnerability management.

The rest of this paper is organized as follows: Section 2 presents the architecture of our OVM; Section 3 discusses how to populate the OVM with vulnerability instances from NVD and other
A Policy-Based Vulnerability Analysis Framework

By

SOPHIE JEAN ENGLE
B.S. (University of Nebraska at Omaha) 2002

Dissertation
Submitted in partial satisfaction of the requirements for the degree of
DOCTOR OF PHILOSOPHY
in
Computer Science
in the
OFFICE OF GRADUATE STUDIES
of the
UNIVERSITY OF CALIFORNIA

Approved:

Professor Matt Bishop (Chair)

Professor S. Felix Wu

Professor Karl Levitt

Professor Sean Peiser

Committee in Charge
2010

Analysis-Based Verification: A Programmer-Oriented Approach to the Assurance of Mechanical Program Properties

T. J. Halloran
May 27, 2010
CMU-ISR-10-112

Institute for Software Research
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Thesis Committee:
William E. Scherlis (Advisor)
James D. Herold
Mary Shaw
Joshua J. Bloch, Google, Inc.

Copyright © 2010 T. J. Halloran

This material is based upon work supported by the following agencies: NASA NNX08AD68A and NASA NNX08AD65A, Lockheed Martin, PB0867299.

ARCS PACI/3925/06-A IBM Sponsor 10-2006. The views expressed in this document are those of the author and do not necessarily represent the official policies, either expressed or implied, of the sponsor, the U.S. Government, or Carnegie Mellon University.
Linkage with Fundamental Changes in Enterprise Security Initiatives

Twenty Critical Controls for Effective Cyber Defense: Consensus Audit Guidelines

What the 20 CSC Critics say...

20 Critical Security Controls - Version 2.0
- 20 Critical Security Controls - Introduction (Version 2.0)
- Critical Control 1: Inventory of Authorized and Unauthorized Devices
- Critical Control 2: Inventory of Authorized and Unauthorized Users
- Critical Control 3: Secure Configurations for Hardware and Software
- Critical Control 4: Secure Configurations for Network Devices
- Critical Control 5: Boundary Defense
- Critical Control 6: Maintenance, Monitoring, and Analysis of Systems
- Critical Control 7: Application Software Security
- Critical Control 8: Controlled Use of Administrative Privileges
- Critical Control 9: Controlled Access Based on Need to Know
- Critical Control 10: Security Management

Procedures and tools for implementing the control

Source code testing tools, web application security scanning tools, and object code testing tools have proven useful in securing application software, along with manual application security penetration testing by testers who have extensive programming knowledge as well as application penetration testing expertise. The Common Weakness Enumeration (CWE) initiative is utilized by many such tools to identify the weaknesses that they find. Organizations can also use CWE to determine which types of weaknesses they are most interested in addressing and removing. A broad community effort to identify the "Top 25 Most Dangerous Programming Errors" is also available as a minimum set of important issues to investigate and address during the application development process. When evaluating the effectiveness of testing for these weaknesses, the Common Attack Pattern Enumeration and Classification (CAPEC) can be used to organize and record the breadth of the testing for the CWEs as well as a way for testers to think like attackers in their development of test cases.

CWE and CAPEC included in Control 7 of the “Twenty Critical Controls for Effective Cyber Defense: Consensus Audit Guidelines”
Linkage with Fundamental Changes in Enterprise Security Initiatives

Enabling Distributed Security in Cyberspace

Building a Healthy and Resilient Cyber Ecosystem with Automated Collective Action

CWE and CAPEC included in “Enabling Distributed Security in Cyberspace: Building a Healthy and Resilient Cyber Ecosystem with Automated Collective Action”
Software Assurance
Community Resources and Information Clearinghouse
Sponsored by DHS National Cyber Security Division

Forums
The Software Assurance Program of the Department of Homeland Security, Cyber Security Division co-sponsors SwA Forums semi-annually with the Department of Defense and the National Institute for Standards and Technology. The purpose of the forums is to bring together members of government, industry, academia with vested interests in software assurance to discuss and grow knowledge, security, and reliability in software.

Forum Presentations
SwA Forum presentations that are released for publication are posted here:
- 13th Semi-Annual Software Assurance Forum - September 27-October 1, 2010
- 12th Semi-Annual Software Assurance Forum - March 9-12, 2010
- 11th Semi-Annual Software Assurance Forum - November 3-5, 2009
- 10th Semi-Annual Software Assurance Forum - March 10-12, 2009
- 9th Semi-Annual Software Assurance Forum - October 14-16, 2008

SwA Working Groups
In between SwA Forums, the DHS SwA Program hosts SwA Working Groups to provide venues for multi-domain collaboration in advancing software assurance in government and industry, and also updates from the SwA Working Groups are presented at the SwA Forums and to other relevant stakeholder groups. For more information on WG sessions, see the Events page on Build Security In.

- June 21-23, 2010 Working Group Session Agenda and Presentations
- September 14-16, 2010 Working Group Session Agenda and Presentations

Learn more about SwA Forums and Working Group Sessions on SwA Forums and Working Group Sessions FAQs and Frequently Asked Questions.

SwA Communities
SwA Forums & Working Groups
Workforce Education & Training
Processes & Practices
Technology, Tools & Product Eval.

Resources
Build Security In
SwA Tools Overview
CERT Secure Coding Standards
Common Attack Pattern Enumeration and Classification (CAPEC)
Common Weakness Enumeration (CWE)
The Data & Analysis Center for Software
Federal Plan for Cyber Security and Information Assurance Research and Development: Available for download on the National Coordination Office for Networking and Information Technology Research and Development site.
Function Extraction: Automated Behavior Computation for Aerospace Software Verification and Certification (PDF)
ISO/IEC SC22 OWG3 Guidance for Avoiding Vulnerabilities through Language Selection and Use
Questions?

tramartin@mitre.org