Making Security Measurable

10 November 2010

Sean Barnum
sbarnum@mitre.org
Today Everything’s Connected

Your System is attackable...

When this Other System gets subverted through an un-patched vulnerability, a mis-configuration, or an application weakness…
Cyber Threats Emerged Over Time

- **1980’s**
  - Password cracking
  - Exploiting known vulnerabilities
  - Burglaries
  - Packet spoofing
  - Back doors
  - Disabling audits
  - Internet social engineering attacks
  - GUI intruder tools

- **1990’s**
  - Automated widespread attacks
  - Executable code attacks (against browsers)
  - Automated probes/scans
  - Network mgmt. diagnostics
  - Sniffers

- **2000’s**
  - Automated widespread attacks
  - “Stealth”/advanced scanning techniques
  - Widespread attacks using NNTP to distribute attack
  - Widespread attacks on DNS infrastructure
  - Email propagation of malicious code

- **2010’s**
  - DDoS attacks
  - Increase in tailored worms
  - Sophisticated command & control
  - Diffuse spyware
  - Anti-forensic techniques
  - Home users targeted
  - Distributed attack tools
  - Increase in wide-scale Trojan horse distribution
  - Windows-based remote controllable Trojans (Back Orifice)
  - Techniques to analyze code for vulnerabilities without source code

**Attack Sophistication**
Solutions Also Emerged Over Time

1980’s
- Password cracking
- Exploiting known vulnerabilities
- Packet spoofing
- Burglaries

1990’s
- Password guessing
- Exploiting known vulnerabilities
- Burglaries

2000’s
- Automated probes/scans
- Email propagation of malicious code
- DDoS attacks
- Binary encryption
- Increase in tailored worms
- Sophisticated command & control
- Diffuse spyware

2010’s
- Increase in wide-scale Trojan horse distribution
- Windows-based remote controllable Trojans (Back Orifice)
- Techniques to analyze code for vulnerabilities without source code
Each new solution had to integrate with the existing solutions, so every enterprise ends up learning as they go and has a “unique” tapestry of solutions with “local practices.”
But a more supportable solution is possible with standardized approaches and the application of architecting principles.
Architecting Security with Information Standards for COIs

- Asset Management
- Vulnerability Management
- Configuration Management
- Threat Management
- System Development
- System Certification
- Intrusion Detection
- Incident Management
- Change Management
- Central Reporting


© 2010 The MITRE Corporation. All rights reserved.
What Do The Informational Building Blocks for “Architecting Security” Look Like?

• Standard ways for enumerating “things we care about”

• **Languages/Formats** for encoding/carrying high fidelity content about the “things we care about”

• **Repositories** of this content for use in communities or individual organizations

• **Adoption/branding and vetting** programs to encourage adoption by tools and services
The Building Blocks Are:

• Enumerations
  – Catalog the fundamental entities in IA, Cyber Security, and Software Assurance
    • Vulnerabilities (CVE), configuration issues (CCE), software packages (CPE), attack patterns (CAPEC), weaknesses in code/design/architecture (CWE)
• Languages/Formats
  – Support the creation of machine-readable state assertions, assessment results, and messages
    • Configuration/vulnerability/patch/asset patterns (XCCDF & OVAL), results from standards-based assessments (ARF), software security patterns (SBVR), event patterns (CEE), malware patterns (MAEC), risk of a vulnerability (CVSS), config risk (CCSS), weakness risk (CWSS), information messages (CAIF & *DEF)
• Knowledge Repositories
  – Packages of assertions supporting a specific application
    • Vulnerability advisories & alerts, (US-CERT Advisories/IAVAs), configuration assessment (NIST Checklists, CIS Benchmarks, NSA Configuration Guides, DISA STIGS), asset inventory (NIST/DHS NVD), code assessment & certification (NIST SAMATE, DoD DIACAP & eMASS)

Tools
  – Interpret IA, Cyber Security, and SwA content in context of enterprise network
  – Methods for assessing compliance to languages, formats, and enumerations
Vulnerability Information Sharing
(circa 1998-1999)

- Scanning Tools
- Priority Lists
- Alerts & Advisories
- Intrusion Detection Systems
- Vendor Patches
- Incident Response & Reporting
- Web Sites
- Research

© 2010 The MITRE Corporation. All rights reserved.
Vulnerability Information Sharing
(circa 1999+)

- Alerts & Advisories
- Priority Lists
- Scanning Tools
- Vendor Patches
- Intrusion Detection Systems
- Incident Response & Reporting
- Web Sites
- Research
- Vulnerability Information Sharing (circa 1999+)
- OVE List

© 2010 The MITRE Corporation. All rights reserved.
Vulnerability Type Trends:
A Look at the CVE List (2001 - 2007)
Removing and Preventing the Vulnerabilities Requires More Specific Definitions…CWEs

- **XSS**
  - Failure to Sanitize Directives in a Web Page (aka 'Cross-site scripting' (XSS)) (79)
  - Failure to Sanitize Script-Related HTML Tags in a Web Page (Basic XSS) (80)
  - Failure to Sanitize Directives in an Error Message Web Page (81)
  - Failure to Sanitize Script in Attributes of IMG Tags in a Web Page (82)
  - Failure to Sanitize Script in Attributes in a Web Page (83)
  - Failure to Resolve Encoded URI Schemes in a Web Page (84)
  - Doubled Character XSS Manipulations (85)
  - Invalid Characters in Identifiers (86)
  - Alternate XSS syntax (87)

- **buf**
  - Failure to Constrain Operations within the Bounds of an Allocated Memory Buffer (119)
  - Unbounded Transfer ('Classic Buffer Overflow') (120)
  - Write-what-where Condition (123)
  - Boundary Beginning Violation ('Buffer Underwrite') (124)
  - Out-of-bounds Read (125)
  - Wrap-around Error (128)
  - Unchecked Array Indexing (129)
  - Incorrect Calculation of Buffer Size (131)
  - Miscalculated Null Termination (132)
  - Return of Pointer Value Outside of Expected Range (466)

- **sql-inject**

- **dot**

- **php-include**

- **infoleak**

- **dos-malform**

- **link**

- **format-string**

- **crypt**

- **priv**

- **perm**

- **metachar**

- **int-overflow**

**Path Traversal (22)**
- Relative Path Traversal (23)
  - Path Traversal: `\...\filename` (29)
  - Path Traversal: `\dir\...\filename` (30)
  - Path Traversal: `\dir\...\filename` (31)
  - Path Traversal: `\...` (Triple Dot) (32)
  - Path Traversal: `\...\...` (Multiple Dot) (33)
  - Path Traversal: `\...\...` (34)
  - Path Traversal: `\...\...` (35)
- Absolute Path Traversal (36)
  - Path Traversal: `\absolute/\pathname/\here` (37)
  - Path Traversal: `\\absolute/\pathname/\here` (38)
  - Path Traversal: `C:\dirname` (39)
  - Path Traversal: `\UNC\\share\\name\\` (Windows UNC Share) (40)
If the weaknesses in software were as easy to spot and their impact as obvious as…

Missing Authentication for Critical Function (CWE-306)
Using Unpublished Web Service APIs (CAPEC-36)
Exploitable Software Weaknesses (a.k.a. Vulnerabilities)

Vulnerabilities can be the outcome of non-secure practices and/or malicious intent of someone in the development/support lifecycle.

The exploitation potential of a vulnerability is independent of the “intent” behind how it was introduced.

Intentional vulnerabilities are spyware & malicious logic deliberately imbedded (and might not be considered defects but they can make use of the same weakness patterns as unintentional mistakes)

Note: Chart is not to scale – notional representation – for discussions.
Common Weakness Enumeration (CWE)

- dictionary of weaknesses
  - weaknesses that can lead to exploitable vulnerabilities (i.e. CVEs)
  - the things we don’t want in our code, design, or architecture
  - web site with XML of content, sources of content, and process used

- structured views
  - currently provide hierarchical view into CWE dictionary content
  - will evolve to support alternate views

- open community process
  - to facilitate common terms/concepts/facts and understanding
  - allows for vendors, developers, system owners and acquirers to understand tool capabilities/coverage and priorities
  - utilize community expertise

http://cwe.mitre.org
Building software only require a few skills and basic understanding...
...but sailing ships in the open ocean and building commerce and defense capabilities based upon them requires understanding...
MSC00-CPP. Compile clearly at high warning levels

Added by Justin Recer, last edited by Justin Recer on Oct 08, 2008 (view change)
Labels: CERT@CERT. CERT@CERT

Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code.

According to C99/ISO/IEC 9899:1999 Section 5.1.1.3:

A conforming implementation shall produce at least one diagnostic message (identified in an implementation-defined manner) if a preprocessing translation unit or translation unit contains a violation of any syntax rule or constraint, even if the behavior is also explicitly specified as undefined or implementation-defined. Diagnostic messages need not be produced in other circumstances.

Assuming a conforming implementation, eliminating diagnostic messages will eliminate any syntax or constraint violations.

If suitable source code checking tools are available, use them regularly.

Exceptions

MSC00-30S: Compilers can produce diagnostic messages for correct code. This is permitted by C99/ISO/IEC 9899:1999, which allows a compiler to produce a diagnostic for any reason. It is usually preferable to write code to eliminate compiler warnings, but if the code is correct, it is sufficient to provide a comment explaining why the warning message does not apply. Some compilers provide ways to suppress warnings, such as suitable Formatted Comments, pragmas, which can be used sparingly when the programmer understands the implications of the warning but has a valid reason to use the biggest construct anyway.

Do not simply quiet warnings by adding type casts or other means. Instead, understand the reason for the warning and consider a better approach, such as using matching types and avoiding type casts whenever possible.

Risk Assessment

Eliminating violations of syntax rules and other constraints can eliminate serious software vulnerabilities that can lead to the execution of arbitrary code with the permissions of the vulnerable process.

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"
...some threats and hazards are unpredictable and dynamic...
...so new types of scanning for hazards and threats were created to make shipping safer and more dependable and secure in more places...
...but they also needed to “understand” current information about highly dynamic threats in order to operate safe, secure and reliably.
But they also needed to deal with the people that were out there trying to locate vulnerabilities and weaknesses in their technologies, processes, or practices...
...with defensive and offensive security capabilities.
Buffer Overflow (CWE-120) Exploit (CAPEC-123)

SQL Injection (CWE-89) Exploit (CAPEC-66)

Security Feature
PLOVER
(CWE
draft 1)

© 2010 The MITRE Corporation. All rights
reserved.
With all of these CWEs, where do you start?
Introduction

The 2010 CWE/SANS Top 25 Most Dangerous Software Errors is a list of the most widespread and critical programming errors that can lead to serious software vulnerabilities. They are often easy to find, and easy to exploit. They are dangerous because they will frequently allow attackers to completely take over the software, steal data, or prevent the software from working at all.

The Top 25 list is a tool for education and awareness to help programmers to prevent the kinds of vulnerabilities that plague the software industry, by identifying and avoiding all-too-common mistakes that occur before software is even shipped. Software customers can use the same list to help them to ask for more secure software. Researchers in software security can use the Top 25 to
2010 CWE/SANS Top 25 Programming Errors
(released 16 Feb 2010)
cwe.mitre.org/top25/

Sponsored by:
- National Cyber Security Division (DHS)

List was selected by a group of security experts from 34 organizations including:
- Academia: Purdue, Northern Kentucky University
- Government: CERT, NSA, DHS
- Software Vendors: Microsoft, Oracle, Red Hat, Apple, Juniper, McAfee, Symantec, Sun, RSA (of EMC)
- Security Vendors: Veracode, Fortify, Cigital, Mandiant, SRI, Breach, SAIC, Aspect,
- Security Groups: OWASP, WASC
### Insecure Interaction Between Components

These weaknesses are related to insecure ways in which data is sent and received between separate components, modules, programs, processes, threads, or systems.

For each weakness, its ranking in the general list is provided in square brackets.

<table>
<thead>
<tr>
<th>Rank</th>
<th>CWE ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CWE-79</td>
<td>Failure to Preserve Web Page Structure ('Cross-site Scripting')</td>
</tr>
<tr>
<td>2</td>
<td>CWE-89</td>
<td>Improper Sanitization of Special Elements used in an SQL Command ('SQL Injection')</td>
</tr>
<tr>
<td>4</td>
<td>CWE-352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>8</td>
<td>CWE-434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
</tr>
<tr>
<td>9</td>
<td>CWE-78</td>
<td>Improper Sanitization of Special Elements used in an OS Command ('OS Command Injection')</td>
</tr>
<tr>
<td>17</td>
<td>CWE-209</td>
<td>Information Exposure Through an Error Message</td>
</tr>
<tr>
<td>23</td>
<td>CWE-601</td>
<td>URL Redirection to Untrusted Site ('Open Redirect')</td>
</tr>
<tr>
<td>25</td>
<td>CWE-362</td>
<td>Race Condition</td>
</tr>
</tbody>
</table>

### Risky Resource Management

The weaknesses in this category are related to ways in which software does not properly manage the creation, usage, transfer, or destruction of important system resources.

<table>
<thead>
<tr>
<th>Rank</th>
<th>CWE ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>CWE-120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
</tr>
<tr>
<td>7</td>
<td>CWE-22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
</tr>
<tr>
<td>12</td>
<td>CWE-805</td>
<td>Buffer Access with Incorrect Length Value</td>
</tr>
<tr>
<td>13</td>
<td>CWE-754</td>
<td>Improper Check for Unusual or Exceptional Conditions</td>
</tr>
<tr>
<td>14</td>
<td>CWE-98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')</td>
</tr>
<tr>
<td>15</td>
<td>CWE-129</td>
<td>Improper Validation of Array Index</td>
</tr>
<tr>
<td>16</td>
<td>CWE-190</td>
<td>Integer Overflow or Wraparound</td>
</tr>
<tr>
<td>18</td>
<td>CWE-131</td>
<td>Incorrect Calculation of Buffer Size</td>
</tr>
<tr>
<td>20</td>
<td>CWE-494</td>
<td>Download of Code Without Integrity Check</td>
</tr>
<tr>
<td>22</td>
<td>CWE-770</td>
<td>Allocation of Resources Without Limits or Throttling</td>
</tr>
</tbody>
</table>

### Porous Defenses

The weaknesses in this category are related to defensive techniques that are often misused, abused, or just plain ignored.

<table>
<thead>
<tr>
<th>Rank</th>
<th>CWE ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>CWE-285</td>
<td>Improper Access Control (Authorization)</td>
</tr>
<tr>
<td>6</td>
<td>CWE-807</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
</tr>
<tr>
<td>10</td>
<td>CWE-311</td>
<td>Missing Encryption of Sensitive Data</td>
</tr>
<tr>
<td>11</td>
<td>CWE-798</td>
<td>Use of Hard-coded Credentials</td>
</tr>
<tr>
<td>19</td>
<td>CWE-306</td>
<td>Missing Authentication for Critical Function</td>
</tr>
<tr>
<td>21</td>
<td>CWE-732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
</tr>
<tr>
<td>24</td>
<td>CWE-327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
</tr>
</tbody>
</table>
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 compromise. The list is not exhaustive and is updated periodically to reflect new bugs.

Michael Howard

CWE-352: Cross-Site Request Forgeries

CORS-XSS, also known as (CSRF) vulnerabilities are relatively common form of Web attacks that allow an attacker to control a user's session and/or steal sensitive data from another user's session. A cross-site request forgery (CSRF) is a type of attack where an attacker can cause a user to perform an action on a Web site. The attacker can trick the user into clicking on a link or submitting a form with the attacker's malicious data. This can lead to unauthorized changes to the user's account or other sensitive data.
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 in time for May.

Michael and I have written coverage of the Top 25 and believe that the results to 25 were developed independently and are based on analysis white paper and best practice guidance around every major program. We have made many of the same recommendations for you to download and implement.

Below is a summary of how the SDL covers every one of them (race conditions are covered by multiple SDL requirements and tools to prevent or detect them). The table is sorted by CWE followed by Title. The table only shows tools and education.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Improper Input Validation</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>Improper Encoding or Escaping of Output</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>Failure to Preserve SQL Query Structure (aka SQL Injection)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>Failure to Preserve Web Page Structure (aka Cross-Site Scripting)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>Failure to Preserve OS Command Structure (aka OS Command Injection)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>319</td>
<td>Cleartext Transmission of Sensitive Information</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>352</td>
<td>Cross-site Request Forgery (aka CSRF)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>362</td>
<td>Race Condition</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>209</td>
<td>Error Message Information Leak</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>Failure to Constrain Memory Operations within the Bounds of a Memory Buffer</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>642</td>
<td>External Control of Critical State Data</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>External Control of File Name or Path</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>426</td>
<td>Untrusted Search Path</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>Failure to Control Generation of Code (aka ‘Code Injection’)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>494</td>
<td>Download of Code Without Integrity Check</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>404</td>
<td>Improper Resource Shutdown or Release</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>665</td>
<td>Improper Initialization</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>682</td>
<td>Incorrect Calculation</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>285</td>
<td>Improper Access Control (Authorization)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>259</td>
<td>Hard-Coded Password</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>732</td>
<td>Insecure Permission Assignment for Critical Resource</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>310</td>
<td>Use of Insufficiently Random Values</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>Execution with Unnecessary Privileges</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>602</td>
<td>Client-Side Enforcement of Server-Side Security</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>
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.
CWE Compatibility & Effectiveness Program (launched Feb 2007)

cwe.mitre.org/compatible/
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-2008-4844.

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 systemically 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 fared.

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 _anyPixmap, but I figured ArrayOfObjectsFromIE is a little more descriptive for people not in the Internet Explorer team.)

```c
int MaxIdx = ArrayOfObjectsFromIE.Size()-1;
for (int i=0; i <= MaxIdx; i++) {
    if (!_ArrayOfObjectsFromIE[i])
        continue;
        ArrayOfObjectsFromIE[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 ArrayOfObjectsFromIE 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, 2.

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 in this case, the action can happen in the loop.
OWASP Top Ten 2007 & 2010 use CWE refs
Some High-Level CWEs Are Now Part of the NVD CVE Information

Overview
SQL injection vulnerability in mods/banners/navlist.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
CVSS Severity (version 2.0):
   AV:N/AC:L/Au:N/C:P/I:P/A:P (legend)
   Impact Subscore: 6.4
   Exploitability Subscore: 10.0

Access Vector: Network exploitable
Access Complexity: Low
Authentication: Not required to exploit
Impact Type: Provides unauthorized access, Allows partial confidentiality, integrity, and availability violation, Allows unauthorized disclosure of information, Allows disruption of service

References to Advisories, Solutions, and Tools
External Source: BID (disclaimer)
Name: 25770
Hyperlink: http://www.securityfocus.com/bid/25770

External Source: MILWORM (disclaimer)
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)

CVE Standard Vulnerability Entry:
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5061

NVD XML feeds also include CWE

Vulnerability Type (View All)
SQL Injection (CWE-89)
Welcome to the NIST SAMATE Reference Dataset Project.

The purpose of the SAMATE Reference Dataset (SRD) is to provide users, researchers, a set of known security flaws. This will allow end users to evaluate tools and tool designs, source code, binaries, etc., i.e. from all the phases of the software life cycle (written to test or generated), and "academic" (from students) test cases. This dataset has bugs and vulnerabilities. The dataset intends to encompass a wide variety of compilers. The dataset is anticipated to become a large-scale effort, gathering test cases about the SRD, including goals, structure, test suite selection, etc.

**Browse, download, and search the SRD**
Anyone can browse or search test cases and download selected cases. Please click selected or all test cases. To find specific test cases, please click here.

**How to submit test cases**

---

**Source Code Security Analysis Tool Functional Specification Version 1.0**

Information Technology Laboratory (ITL), Software Diagnostics and Conformance Testing Division

29 January, 2007

Michael Kass
Michael Koo

National Institute of Standards and Technology
Information Technology Laboratory
Software Diagnostics and Conformance Testing Division
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.
- Common cryptographic errors and remedies.

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

Resources


Test

Testing activities validate the secure implementation of a product, which reduces the likelihood of security bugs being released and discovered by customers and malicious users. The majority of SAFECode members have adopted the full software security testing practices in their software development lifecycle. This is not to “test in security,” but rather to validate the robustness and security of the software products prior to making the product available to customers. Effective testing methods do find security bugs, especially for products that may not undergo critical security development process changes.

Fuzz testing

Fuzz testing is a reliability and security testing technique that relies on building test data that are designed with intentionally malformed data and then having the software under test consume the 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 testers to suit specialized file and network data formats. Fuzz testing is an effective testing technique because it uncovers weaknesses in data handling code.

Resources

- *Fuzz Testing of Application Reliability*, University of Wisconsin; http://pages.cs.wisc.edu/~harb/fuzz/fuzz.html
- *Fuzzing*: Brute Force Vulnerability Discovery, Sutton, Greens & Amini, Addison-Wesley
- *Common Attack Pattern Enumeration and Classification*, MITRE; http://cwe.mitre.org/
Idaho National Labs SCADA Report

SECURE CONTROL SYSTEM/ENTERPRISE ARCHITECTURE

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

May 2010

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

Corporate Network
ICS Web Application Client
ICS Business Application Client
Corporate Hosts

Level 3
Operations Management:
System Management / Supervisory Control

LAN / WAN / DMZ
ICCP Server
OPC Server
Information Server
Application Server
Replicated Database
Web Server

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

Supervisory Control LAN
Historical Database
Real-time Database
Communications Processor

Level 1
Control Equipment:
Protection and Local Control Devices

Control Network
RTU
Distributed Control
PLC

Level 0
Equipment Under Control:
Sensors and Actuators

I/O Network
Temperature Sensor
Pressure Sensor
Relay
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 (“Classic Buffer Overflow”)</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 (“Resource Exhaustion”)</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 (“Path Traversal”)</td>
</tr>
<tr>
<td></td>
<td>CWE-79: Failure to Preserve Web Page Structure (“Cross-site Scripting”)</td>
</tr>
<tr>
<td></td>
<td>CWE-89: Failure to Preserve SQL Query Structure (“SQL Injection”)</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>
CWE and CAPEC included in Control 7 of the “Twenty Critical Controls for Effective Cyber Defense: Consensus Audit Guidelines”

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.
Common Criteria v4 CCDB

- TOE to leverage CAPEC & CWE
- Also investigating how to leverage ISO/IEC 15026 NIAP Evaluation Scheme
- Above plus
- Also investigating how to leverage SCAP
Making Security Measurable

MITRE, in collaboration with government, industry, and academic stakeholders, is improving the measurability of security through enumerating baseline security data, providing standardized languages as means for accurately communicating the information, and encouraging the sharing of the information with users by developing repositories.

The other activities and initiatives listed here have similar concepts or compatible approaches to MITRE's. Together all of these efforts are helping to make security more measurable by defining the concepts that need to be measured, providing for high fidelity communications about the measurements, and providing for sharing of the measurements and the definitions of what to measure.

Enumerations

- Common Vulnerabilities and Exposures (CVE®) - common vulnerability identifiers
- Common Weakness Enumeration (CWE™) - list of software weaknesses types
- CAPEC - Common Attack Pattern Enumeration and Classification (CAPEC™) - list of common attack patterns
- CCE - Common Configuration Enumeration (CCE™) - common configuration identifiers
- CPE - Common Platform Enumeration (CPE™) - common platform identifiers
- CVSS/SANS Top 25 - consensus list of the 25 most dangerous programming errors

Center for Internet Security (CIS) Consensus Security Metrics Definitions - set of standard metrics and data definitions that can be used across organizations to collect and analyze data on security process performance and outcomes

Languages

- OVAL - Open Vulnerability and Assessment Language (OVAL®) - standard for determining vulnerability and configuration issues
- CCE - Common Event Expression (CCE™) - standardizes the way computer events are described, logged, and exchanged
- MAEC - Malware Attribute Enumeration and Characterization (MAEC™) - standardized language for attribute-based malware characterization
- Benchmark Development - resources for creating standards-based, structured, and automatable security guidance
- OVAL Interpreter - free tool for collecting information for testing, carrying out OVAL definitions, and presenting results of the tests
- Benchmark Editor™ - free tool that enhances and simplifies creation and editing of benchmark documents written in XCCDF and OVAL
- Recommendation Tracker™ - free tool that facilitates the development of automated security benchmarks
- Extensible Configuration Checklist Description Format (XCCDF) - specification language for uniform expression of security checklists, benchmarks, and other configuration guidance
- Open Checklist Interactive Language (OCIL®) - standardized language for expressing and evaluating non-automated security checks
- Common Vulnerability Scoring System (CVSS®) - open standard that conveys vulnerability severity and helps determine urgency and priority of response
- Policy Language for Assessment Results Reporting (PLANNER) - language for requesting and managing assessment results from tools, databases, and other products
- Assessment Results Format (ARF) - open language for exchanging per-device assessment results data between assessment tools, asset databases, and other systems
- Assessment Summary Results (ASR) - language for exchanging summarized assessment results data

Repositories

- National Vulnerability Database (NVD®) - U.S. National vulnerability database based on CVE that integrates all publically available vulnerability resources and references
- SEE repository - System Engineering Exchange tool that provides a collection of software assurance metrics
- Red Hat Repository - OVAL Patch Definitions corresponding to Red Hat Enterprise Linux security advisories
- Novel Repository - OVAL Definitions for SUSE Linux Enterprise compliance checking
- Debian Repository - OVAL Definitions corresponding to Debian security advisories
- National Checklist Program Repository - U.S. Government repository of publicly available security checklists/benchmarks
- Center for Internet Security (CIS) Benchmarks - best-practice security configurations accepted for compliance with FISMA, the ISO standard, GLB, SCX, HIPAA, and other regulatory requirements for information security
- DISA Security Technical Implementation Guides (STIGs) - U.S. Defense Information Systems Agency (DISA) STIGs are configuration standards for DoD information assurance and information assurance-enabled devices and systems
- Common Framework for Vulnerability Dosing and Response (CFDVR) - standard format for reporting and sharing vulnerability information among multiple organizations
- Federal Desktop Core Configuration (FDCC®) - OMB-mandated security configuration for Microsoft Windows Vista and XP operating system software
- United States Government Configuration Baseline (USGCB®) - security configuration baselines for IT products deployed across federal agencies

View the current collection of organizations, activities, and initiatives.

Disclaimer

This Web site is hosted by The MITRE Corporation. ©2010 The MITRE Corporation. CVE and OVAL are registered trademarks and the Making Security Measurable logo, CCE, CWE, CAPEC, CCE, MAEC, Benchmark Editor, and Recommendation Tracker are trademarks of The MITRE Corporation. All other trademarks are the property of their respective owners. Contact us at: makingsecuritymeasurable.mitre.org

Page Last Updated: September 02, 2010
Questions?

sbarnum@mitre.org