



SCAP and SwA Automation Protocol

Bob Martin

June 21, 2010

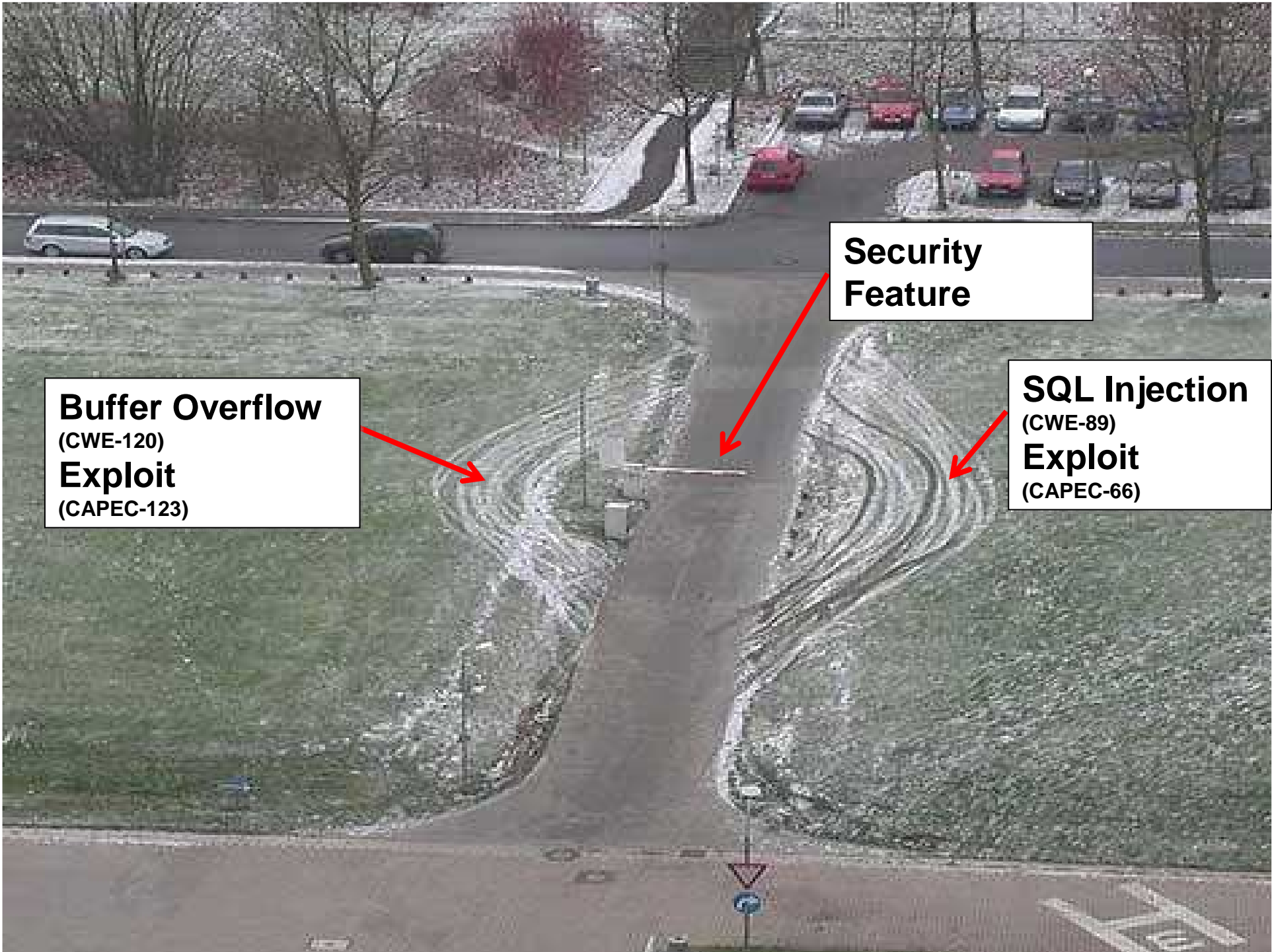


**Homeland
Security**

© 2010 MITRE

**Missing Authentication for
Critical Function (CWE-306)
Using Unpublished Web
Service APIs (CAPEC-36)**



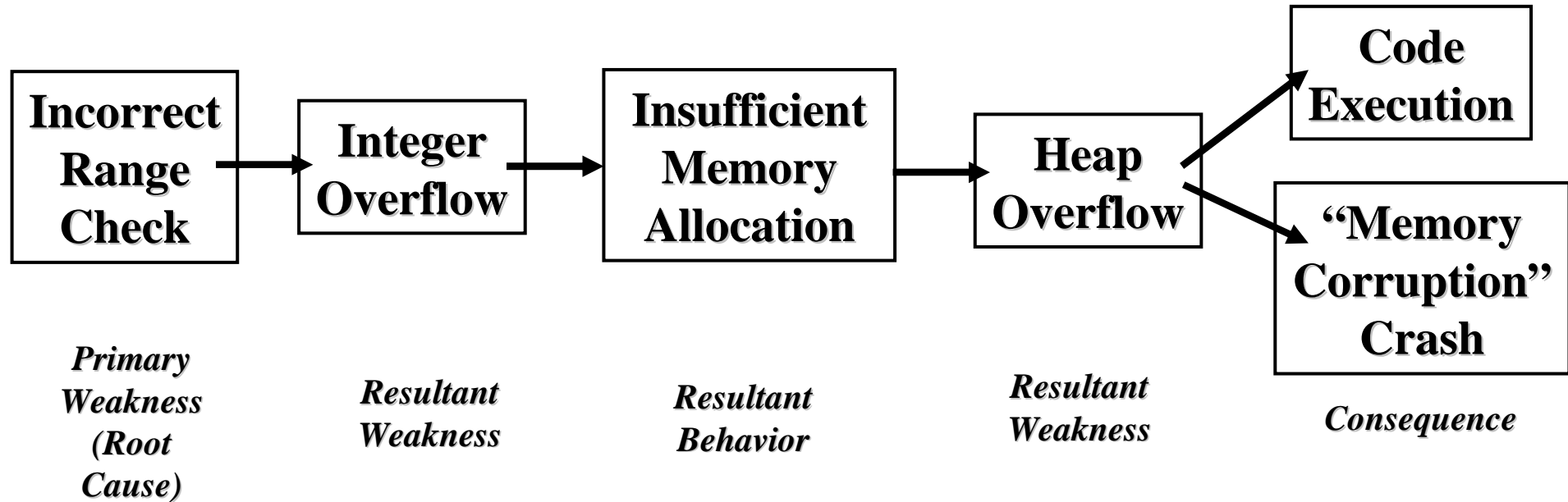


Buffer Overflow
(CWE-120)
Exploit
(CAPEC-123)

Security
Feature

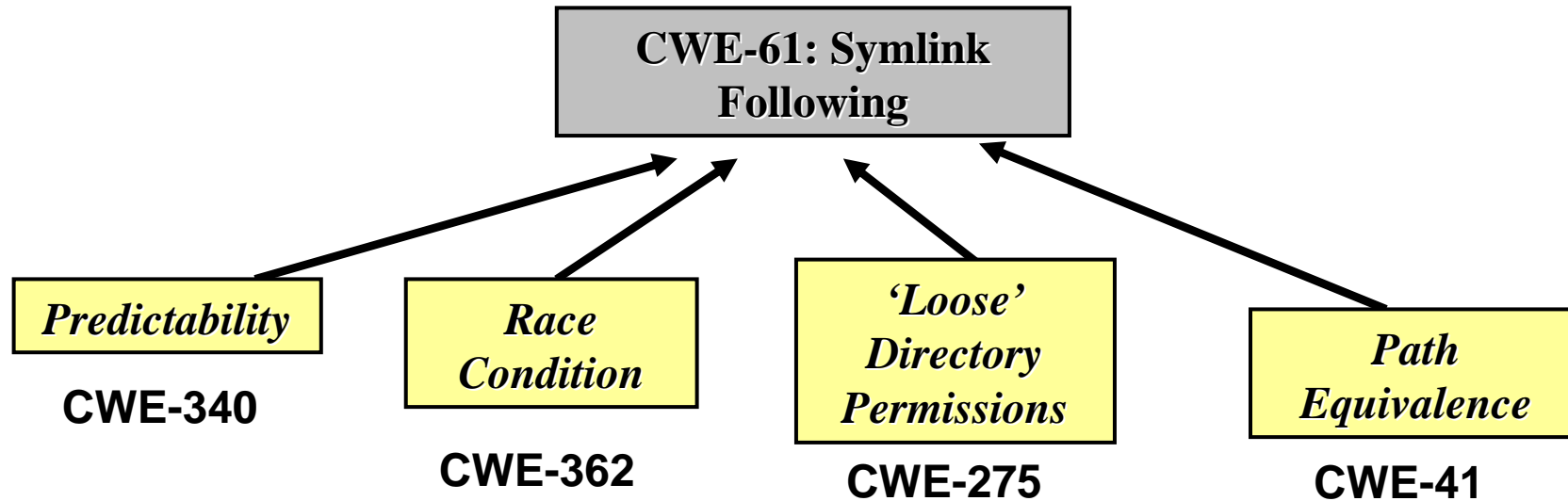
SQL Injection
(CWE-89)
Exploit
(CAPEC-66)

Many Vulnerabilities are Multi-Factor: Chains and Composites



One of the main problems with classification and terminology is that ANY behavior in the chain could be regarded as the vulnerability.

Composite: Symbolic Link Following



- **Filename can be predicted**
- **File can be created by other party before it is opened for writing**
- **File created in a shared directory with writable permissions**
- **Equivalence: a symlink can act an alternate name for a critical file**

Printable PDFs of Entire CWE Now Available



CWE Version 1.4

Edited by:
Steven M. Christey, Conor O. Harris, and Janis E. Kenderdine

Project Lead:
Robert A. Martin



CWE Version 1.4 Table of Contents

- 1
- 1
- 1
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 7
- 8
- 9
- 10
- 12
- 13
- 13
- 13
- 14
- 14
- 21
- 22
- 24
- 26
- 27
- 27
- 28
- 29
- 31
- 32
- 33
- 34
- 35
- 36
- 37
- 38
- 39
- 40
- 41
- 42
- 43
- 45
- 45
- 46
- 46
- 47
- 47
- 48
- 49
- 49
- 50
- 50
- 51
- 51
- 52
- 53
- 53
- iii

Table of Contents

CWE Version 1.4 CWE-1: Location

Status: Incomplete

roduced during the

Page	
699	1
699	13
699	13
699	695

Status: Draft

Page	
699	1
699	1
700	2
700	3
700	4
700	5
700	6
700	7
700	8
700	9
699	10
700	12
699	465
699	560
699	643
700	696

Status: Draft

Page	
699	1
699	1
699	5-40

CWE Version 1.4 Failure to Preserve SQL Query Structure (SQL Injection)

es.
minimize the SQL

Good Code

the boxes, the
ation user has the

Bad Code

database.

Bad Code

First of all, the
n SQL. If a user
hich may bypass
data / command
ble to alter the
possibly accessing
trophe are
programmer may
event any data /

ows SQL injection

ly encoded output.
e Java Beans,

on between
g, encoding, and
ability at every

ored procedures.
g. Do not
xec" or similar

CWE-89: Failure to Preserve SQL Query Structure (SQL Injection)

CWE Version 1.4 Index

- 7 - Characters and
- 8 - Memory Management
- 9 - Input Output (FIO),
- 0 - Environment (ENV),
- 1 - Signals (SIG), 736
- 2 - Error Handling (ERR),
- 9 - Miscellaneous (MSC),
- 0 - POSIX (POS), 738
- Comparison Errors, 200
- rmation, 338
- e Information, 342
- Side Security, 596
- e(), 578
- e, 211
- ossible Directory, 560
- 510
- 514
- Instead of Object
- Modification of Security-
- ar Buffers, 10
- 762
- ors), 247
- , 396
- ctory with Incorrect
- isecure Permissions,
- al Modifier, 521
- 750
- ation Leak, 244
- F), 373
- uring Sensitive
- r Injection), 611
- 6
- xception, 424
- Exception, 425
- l, 487
- 31
- t Timing Channel, 539

Index

CWE Version 1.4 Index

- uplicate Identifier, 692
- for Authorization
- to Detect NULL Pointer
- ary Authentication, 334
- rmine Size, 492
- tion, 670
- 1, 333
- thread-safe Manner,
- 490
- Argument, 564
- Comparison, 592
- 9
- tion, 319
- es, 651
- s, 287
- nism for Forgotten
- 41
- ERT C Secure Coding
- FE, 630
- (2004), 718
- (2007), 619
- n C, 652
- n C++, 653
- n Java, 655
- n PHP, 657
- NS Top 25 Most
- 739
- esign, 696
- plementation, 703
- irectories, 621
- 622
- rocesses, 622
- Q), 59
- 3
- ection), 107

Index

CWE Outreach: A Team Sport

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

CWE-732: Insecure Permission Assignment for Critical Resource

I've already touched on this several times here, but review all missions and ACLs on all objects you create in the file system configuration stores such as Windows Vista and later, and change any default ACL in the system or registry unless you intend to weaken the ACL.

CWE-330: Use of Insufficiently Random Values

Identify all the random number generators in your code and determine which, if any, generate passwords, or some other secret. Make sure the code generating random numbers is cryptographically random and not a deterministic pseudorandom generator like the C runtime `rand()` function. Using functions like `rand()` for, but not for cryptography.

CWE-250: Execution with Unnecessary Privileges

Identify all processes that run part of your solution and determine what privileges they need to operate correctly. If a process runs as root (on Linux, Unix, Mac OS X) or system (Windows) ask yourself, "Why?" Sometimes the answer is totally valid because the code must perform a privileged operation, but sometimes you don't know why it runs any other than, "That's the way it's always run!" If the code needs to operate at high privilege keep the time span within which the code is high privilege as small as possible—for example, opening a port below 1024 in a Linux application requires the code be run as root, but after that,

Basic Training

portant that define file and path information before accessing a file or path, and restrict what content or filename. As view, look for or accesses file and make sure name is appropriate to valid data. Remember "known good" content may be a

CWE-426: Untrusted

Old versions searched the root directory filenames, with problems if he had a weak password, weren't completely, weren't completely, no guarantee won't use searches or paths, but this is internationalization—for example, Vista, the API doesn't exist in version of Windows named "Why?" Sometimes the answer is totally valid because the code must perform a privileged operation, but sometimes you don't know why it runs any other than, "That's the way it's always run!" If the code needs to operate at high privilege keep the time span within which the code is high privilege as small as possible—for example, opening a port below 1024 in a Linux application requires the code be run as root, but after that,

CWE-94: Failure to Generation

It's common to see code injection vulnerabilities in JavaScript code that builds a string dynamically and passes it to `eval()` to execute. If the attacker controls the source string in any way, he or she can create a malicious payload. The simplest way to eradicate this kind of bug is to eradicate the use of `eval()`, but that could mean redesigning the application.

(XSS), CWE-79 is the real bug that makes CWE-116 worse. In the past, we took XSS bugs lightly, but now we see worms that can exploit XSS vulnerabilities in social networks such as MySpace (for example, the Sunny worm). Also, research into Web-related vulnerabilities has progressed substantially over the past few years, with new ways to attack systems regularly uncovered. For more XSS issues as defined by CWE-79, the best defense is to validate all incoming data. This has always been the right approach and will probably continue to be so for the foreseeable future. Developers can also add a layer of defense by encoding output derived from untrusted input (see CWE-116).

CWE-78: Failure to Preserve OS Command Structure

Many applications, particularly server applications, receive untrusted requests and use the data in them to interact with the underlying operating system. Unfortunately, this can lead to severe server compromise if the incoming data isn't analyzed—again, the best defense is to check the data. Also, running the potentially vulnerable application with low privilege can help contain the damage.

CWE-319: Cleartext Transmission of Sensitive Information

Sensitive data must obviously be protected at rest and while on the wire. The best solution to this vulnerability is to use a well-tested technology such as SSL/TLS or IPsec. Don't (ever!) create your own communication method and cryptographic defense. This weakness is related to CWE-327 ("Use of a Broken or Risky Cryptographic Algorithm"), so make sure you aren't using weak 40-bit RC4 or shared-key IPsec.

CWE-352: Cross-Site Request Forgery

Cross-site request forgery (also known as CSRF) vulnerabilities are a relatively new form of Web weakness caused, in part, by a bad Web application design. In short, this design doesn't verify that a request came from valid user code and is instead acting maliciously on the user's behalf. Generally, the best defense is to use a unique and unpredictable key for each user. Traditionally, verifying input doesn't mitigate this bug type because the input is valid.

CWE-362: Race Condition

Race conditions are timing problems that lead to unexpected behavior—for example, an application uses a filename to verify that a file exists and then uses the same filename to open that file. The problem is in the small time delay between the check and the file open, which attackers can use to change the file or delete or create it. The safest way to mitigate file system race conditions is to open the object and then use the resulting handle for further operations. Also, consider reducing the scope of shared objects—for example, temporary files should be local to the user and not shared with multiple user accounts. Correct use of synchronization primitives (mutexes, semaphores, critical sections) is similarly important.

CWE-642: External Control of Critical State

Unprotected state information, such as profile data or configuration, is subject to attack. It's important to protect it by using the appropriate control lists (ACLs) or permissions for persistent data and sophisticated cryptographic defenses, a hashed message authentication code (HMAC), for one-time data. You can use an HMAC for persistent data as well.

CWE-73: External Control of Filename or Path

Attackers might be able to specify arbitrary file data if they the data that's used as part or path name. It's critical

the very least, look for terms like "pwd" and "password" and make sure you have no hard-coded passwords or secret data in the code. You should also store this data in a secure location within the operating system. By secure, I mean protect it with an appropriate permission or encrypt it and protect the encryption key with an appropriate permission.

CWE-119: Failure to Constrain Memory Operations

The dreaded buffer overflow is a scourge of C and C++ and a vulnerability type has more headaches than buffer runs. The best way to solve the problem is to move away from C and C++ where it makes sense and use higher-level languages such as Ruby, C#, and Java because they don't offer direct access to memory. For C and C++, cautious developers should use "known bad" functions such as `strcpy`, `strcpy_s`, `strcat`, `strcat_s`, `sprintf`, and `gets` and use secure versions. Visual C++ has many weak APIs at compile time and you should strive to use them. Also, fuzz testing and static analysis can help identify potential buffer overruns. operating-system-level tools such as address space layout randomization and no execution stack can help reduce the chance of buffer overruns in exploited

CWE-642: External Control of Critical State

Unprotected state information, such as profile data or configuration, is subject to attack. It's important to protect it by using the appropriate control lists (ACLs) or permissions for persistent data and sophisticated cryptographic defenses, a hashed message authentication code (HMAC), for one-time data. You can use an HMAC for persistent data as well.

CWE-73: External Control of Filename or Path

Attackers might be able to specify arbitrary file data if they the data that's used as part or path name. It's critical

Basic Training

Editors: Richard Ford, rford@se.fi.edu
Michael Howard, mikehow@microsoft.com

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

(<http://cwe.mitre.org/top25/>). I was one of the many people

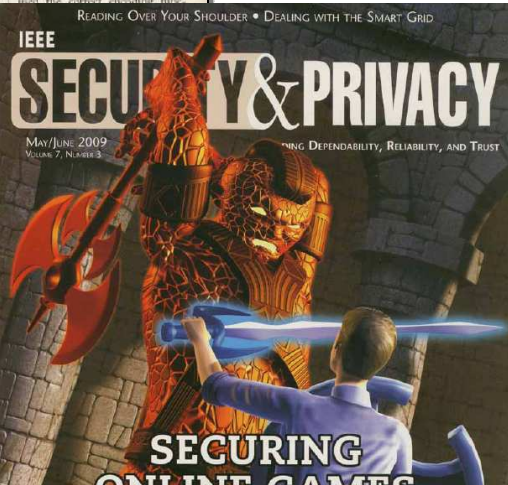
MICHAEL HOWARD
Microsoft

from industry, government, and academia who provided input to the document.

CWE, which stands for Common Weakness Enumeration, is a project sponsored by the National Cyber Security Division of the US Department of Homeland Security to classify security bugs. It assigns a unique number to weakness types such as buffer overruns or cross-site scripting bugs (for example, CWSy-327 is "Use of a Broken or Risky Cryptographic Algorithm"). Shortly after the Top 25 list's release, Microsoft unveiled a document entitled, "The Microsoft SDL and the CWE/SANS Top 25," to explain how Microsoft's security processes can help prevent the worst offenders (<http://blogs.msdn.com/sdl/archive/2009/01/27/sdl-and-the-cwe-sans-top-25.aspx>).

Full disclosure: I'm one of that document's coauthors, but my purpose here isn't to reargue the Microsoft piece. Rather, my goal is to describe some best practices that can help you eliminate the CWE Top 25 vulnerabilities in your own development environment and products. It's also important to understand that addressing the weak-

nesses in the list doesn't imply your software is secure from all forms of attack; there are plenty more vulnerability types to worry about.



Basic Training

68

Improving Software Security by Eliminating the CWE Top 25 Vulnerabilities

MICHAEL HOWARD

Special thanks to Robert A. Wierba of MITRE Corporation.

Handler Errors

- Deployment of Wrong Handler
- Missing Handler
- Dangerous Handler not Disabled During Sensitive Operations
- Unparsed Raw Web Content Delivery
- Incomplete Identification of Uploaded File Variables (PHP)
- Unrestricted File Upload

User Interface Errors

- UI Discrepancy for Security Feature
- Multiple Interpretations of UI Input
- UI Misrepresentation of Critical Information

Behavioral Problems

- Behavioral Change in New Version or Environment
- Expected Behavior Violation

Initialization and Cleanup Errors

- Insecure Default Variable Initialization
- External Initialization of Trusted Variable
- Run-soft on Failed Initialization
- Missing Initialization
- Incomplete Cleanup
- Improper Cleanup on Thrown Exception
- Improper Initialization (149)

Channel and Path Errors

- Channel Errors
- Failure to Protect Alternate Path
- Uncontrolled Search Path Element
- Unquoted Search Path on Element
- Untrusted Search Path

Error Handling

- Error Conditions, Return Values, Status Codes
- Failure to Use a Standard Error Handling Mechanism
- Failure to Catch All Exceptions in Servlet
- Not Calling Securely (Failing Open)
- Missing Custom Error Page

Pointer Issues

- Return of Pointer Value Outside of Expected Range
- Use of size off on a Pointer Type
- Incorrect Pointer Scaling
- Use of Pointer Subtraction to Determine Size
- Assignment of a Fixed Address to a Pointer
- Attempt to Access Child of a Non-structure Pointer

Time and State

- State Issues
 - Incomplete Internal State Destruction
 - State Synchronization Error
 - Mutable Objects Passed by Reference
 - Passing Mutable Objects to an Unchecked Method
 - External Control of Critical State Data (149)
- Session Fixation
- Concurrency Issues
- Temporary File Issues
- Covert Timing Channel
- Technology-Specific Time and State Issues
- Symbolic Name not Mapping to Correct Object
- Signal Errors
- Unrestricted Externally Accessible Lock
- Double-Checked Locking
- Insufficient Session Expiration
- Insufficient Synchronization
- Use of a Non-reentrant Function in an Unsynchronized Context
- Improper Control of a Resource Through its Lifetime
- Exposure of Resource to Wrong Sphere
- Incorrect Resource Transfer Between Spheres
- Use of a Resource after Expiration or Release
- External Influence of Sphere Definition
- Uncontrolled Recursion
- Redirect Without Path

Failure to Fulfill API Contract ('API Abuse')

- Failure to Clear Heap Memory Before Release (Heap Inspection)
- Call to Non-abstract API
- Use of Inherently Dangerous Function
- Multiple Binds to the Same Port
- J2EE Bad Practices: Direct Management of Connections
- Incorrect Check of Function Return Value
- Often Missed Arguments and Parameters
- Uncaught Exception
- Escalation with Unnecessary Privileges (130)
- Often Missed String Management
- J2EE Bad Practices: Direct Use of Sockets
- Unchecked Return Value
- Failure to Change Working Directory in chroot jail
- Reliance on DNS Lookups in a Security Decision
- Failure to Follow Specification
- Failure to Provide Specified Functionality

Web Problems

- Failure to Sanitize CRLF Sequences in HTTP Headers (HTTP Response Splitting)
- Inconsistent Interpretation of HTTP Requests (HTTP Request Smuggling)
- Improper Sanitization of HTTP Headers for Scripting Syntax
- Use of Non-Canonical URI Paths for Authentication Decisions

Indicator of Poor Code Quality

- NULL Pointer Dereference
- Incorrect Block Delimitation
- Omitted Break Statement in Switch
- Undefined Behavior for Input to API
- Use of Hard-coded, Security-relevant Constants
- Unsafe Function Call from a Signal Handler
- Signal Errors
- Return of Stack Variable Address
- Missing Default Case in Switch Statement
- Expression Issues
- Use of Obsolete Functions
- Use of Function with Inconsistent Implementations
- Unused Variable
- Dead Code
- Resource Management Errors
- Improper Resource Shutdown or Release (149)
- Empty Synchronized Block
- Exploit Call to Finalize()
- Reachable Assertion
- Use of Potentially Dangerous Function

Credentials Management

- Hard-Coded Password (130)
- Unvetted Password Change
- Missing Password Field Masking
- Weak Cryptography for Passwords
- Weak Password Requirements
- Not Using Password Aging
- Password Aging with Long Expiration
- Insecurely Protected Credentials
- Weak Password Recovery Mechanism or Forgotten Password

Insufficient Verification of Data Authenticity

- Single Validation Error
- Improper Verification of Cryptographic Signature
- Use of Lost Trusted Source
- Acceptance of Extraneous Untrusted Data With Trusted Data
- Improperly Trusted Remote DNS
- Insufficient Type Distinction
- Over-Reliance on Integrity Check Value
- Failure to Add Integrity Check Value
- Improper Validation of Integrity Check Value
- Trust of System Event Data
- Reliance on File Name or Extension of Externally-Supplied File
- Reliance on Obfuscation or Cryptoplexy of Security-Relevant Inputs without Integrity Checking

Privacy Violation

- Reliance on Cookies without Validation and Integrity Checking
- Client-Side Enforcement of Server-Side Security (149)
- Improperly Implemented Security Check for Standard
- Improper Authentication
- User Interface Security Issues
- Use of Insufficiently Random Values (130)
- Logging of Excessive Data
- Certificate Issues

Insufficient Encapsulation

- Mobile Code Issues/Missing Custom Error Page
 - Public (or world) Method Without Final (Object Block)
 - Use of Inner Class Containing Sensitive Data
 - Critical Public Variable Without Final Modifier
 - Unvetted Use of Code Without Integrity Check (149)
 - Any Declared Public, Final, and Static
 - Analysis() Method Declared Public
- Leftover Debug Code
- Use of Dynamic Class Loading
- clone() Method Without super.clone()
- Comparison of Classes by Name
- Data Leak Between Sessions
- Trust Boundary Violation

Cryptographic Issues

- Key Management Errors
- Missing Required Cryptographic Step
- Not Using a Random IV with CBC Mode
- Failure to Encrypt Sensitive Data
 - Incorrect Storage of Encryption Information
 - Clear Text Transmission of Sensitive Information (130)
 - Sensitive Code in a HTTPS Session Without Secure Attributes
- Removable One-Time Pad
- Inadequate Encryption Strength
- Use of a Broken or Flawed Cryptographic Algorithm (130)
- Use of RSA Algorithms without OAEP

Permissions, Privileges, and Access Controls

- Access Control Mechanism Issues (130)
- Permission Issues
 - Incorrect Default Permissions
 - Incorrectly Inherited Permissions
 - Incorrectly Preserved Inherited Permissions
 - Incorrect Execution Assigned Permissions
 - Improper Handling of Exec/Execv Method
 - Improper Preservation of Permissions
 - Exposed Unusually Access Method
 - Incorrect Permissions Assigned for Critical Resources (130)
 - Permission Race Condition During Resource Copy
- Privilege / Sandbox Issues
- Improper Ownership Management
- Unlearned User Management

Password in Configuration File

- Insufficient Compartmentalization
- Reliance on a Single Factor in a Security Decision
- Insufficient Psychological Acceptability
- Reliance on Security through Obscurity
- Protection Mechanism Failure
- Insufficient Logging
- Reliance on Cookies without Validation and Integrity Checking in a Security Decision

Data Handling

- Numeric Errors
 - Use of Incorrect Byte Ordering
 - Unchecked Array Indexing
 - Incorrect Conversion Between Numeric Types
 - Unchecked Sign Extension
 - Signed to Unsigned Conversion Error
 - Unsigned to Signed Conversion Error
 - Numeric Truncation Error
 - Incorrect Calculations (149)
 - Incorrect Calculation of Buffer Size
 - Integer Overflow or Wraparound
 - Integer Underflow (Wrap or Wraparound)
 - Off by one Error
 - Divide by Zero
- Modification of Assumed-Immutable Data (MAID)
 - Improper Input Validation (130)
 - Pathnames Traversal and Equivalence Errors
 - Process Control
 - Missing EM, Validation
 - Failure to Sanitize Data into a Different Plane (Typecast)
 - Improper Sanitization of Special Elements used in a Contextual ("Green and Spectral") (17)
 - Failure to Preserve Web Page Structure (Class-site Scripting) (17)
 - Improper Sanitization of Special Elements used in an HTML Contextual (SQL Injection) (149)
 - Failure to Sanitize Database (LDAP Queries) (LDAP Injection)
 - XML Injection (aka Blind XPath Injection)
 - Failure to Sanitize CRLF Sequences (CRLF Injection)
 - Uncontrolled Format String
 - Failure to Sanitize Special Elements into a Different Plane
 - Argument Injection or Modification
 - Improper Control of Resource Identification (Resource Injection)
 - Failure to Control destination of Code (Code Injection) (149)
 - Improper Sanitization of Special Elements
 - Technology Specific Input Validation Problems
 - Misinterpretation of Input
 - Unchecked Input for Loop Condition
 - Null Byte Interjection Error (Hex or Null Byte)
 - Direct Use of Unsafe JSP
 - Improper Output Sanitization for Logs
 - Failure to Control Operations within the Bounds of a Memory Buffer (17)
 - Use of Internally-Controlled Input to Select Classes or Code ("Heap's Reflection")
 - ASP.NET Misconfigurations: Not Using Input Validation Framework
 - URL Redirection to Untrusted Site (Open Redirect)
 - Variable Extraction Errors
 - Unvetted Function Hook Arguments
 - Internal Control of File Name or Path (17)
 - Improper Address Initialization in IOCTL, with METHOD, METHOD IO Control Code
 - Use of Path Manipulation Function with a Maximum-sized Buffer
- Information Leak (Information Disclosure)
 - Information Leak Through Sort Data
 - Process Leak Through Data Sources
 - Data Leaking Information Leaks
 - Out-of-Band Information Leak (149)
 - Cross-Boundary Crossing Information Leak
 - Information Leak
 - Process Environment Information Leak
 - Information Leak Through Debug Information
 - Sensitive Information Disclosed Before Release
 - Information Leak of System Data
 - Information Leak Through Calling
 - Information Leak Through Environment Variables
 - File and Directory Information Leaks
 - Information Leak Through Query Strings in GET Request
 - Information Leak Through Indexing of Private Data
 - Information Loss or Distortion
 - Combination Error, Denial of Service
- Improper Access of Indexable Resource ("Range Error")
- Type Errors
- Improper Casting or Casting of Output (17)
- String Errors
- Data Structure Issues
- Improper Handling of Syntactically Invalid Structure

OWASP Top Ten 2007 & 2010 use CWE refs

OWASP TOP 10



OWASP

The Open Web Application Security Project

OWASP Top 10 - 2010

The Ten Most Critical Web Application Security Risks

THE TEN MOST CRITICAL
APPLICATION SECURITY RISKS

2007 UPDATE

© 2002-2007 OWASP Foundation
This document is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike license

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:

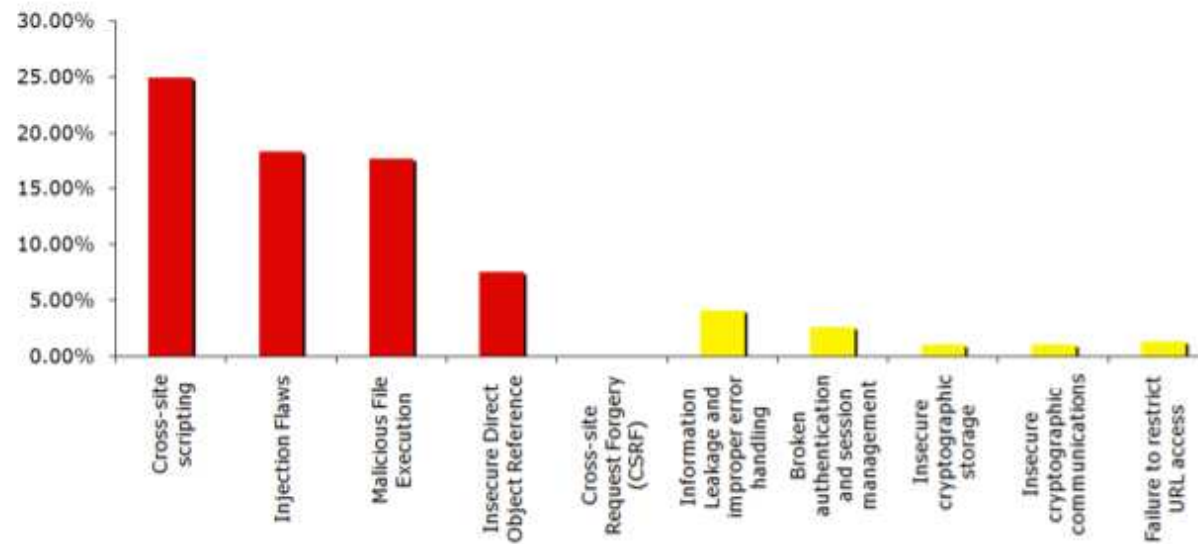
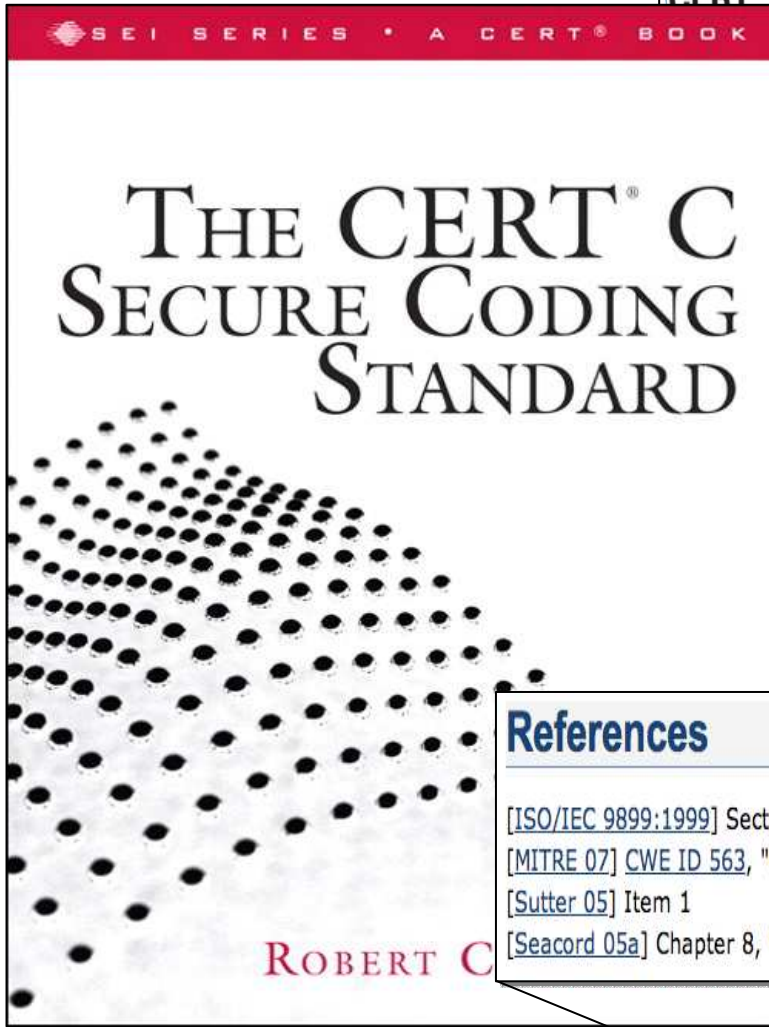


Figure 2: MITRE data on Top 10 web application vulnerabilities for 2006

1
lease



reative Commons (CC) Attribution Share-Alike
Free version at <http://www.owasp.org>



MSC00-CPP. Compile cleanly at high warning levels - CERT Secure Coding Standards

https://www.securecoding.cert.org/confluence/display/cplusplus/MSC00-CPP.+Compile+cleanly+at+high+warning+levels

Software Assurance Secure Systems Organizational Security Coordinated Response Training

ing Practices > ... > 49. Miscellaneous (MSC) > MSC00-CPP. Compile cleanly at high warning levels

Log In Sign Up

C++ Secure Coding Practices

MSC00-CPP. Compile cleanly at high warning levels

Added by [Justin Pincar](#), last edited by [Justin Pincar](#) on Oct 08, 2008 (view change) [SHOW COMMENT](#)
Labels: [unenforceable](#) [incomplete-cpp](#)

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 syntactic or constraint violations.

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

Exceptions

MSC00-EX1: 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 rewrite 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 suitably formatted comments or pragmas, which can be used sparingly when the programmer understands the implications of the warning but has good reason to use the flagged 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"

Related Sites

US-CERT

Go to "<http://cwe.mitre.org/data/definitions/570.html>"

© 2010 MITRE

Some High-Level CWEs Are Now Part of the NVD CVE Information

automation of vulnerability management, security measurement, and compliance (e.g. FISMA).

Resource Status

NVD contains:
26736 [CVE Vulnerabilities](#)
114 [Checklists](#)
91 [US-CERT Alerts](#)
1997 [US-CERT Vuln Notes](#)
2966 [OVAL Queries](#)
12410 [Vulnerable Products](#)

Last updated: 09/26/07
CVE Publication rate: 16 vulnerabilities / day

Email List

Select the email list(s) you wish to join, enter your e-mail address and press "Add" to receive NVD announcements or SCAP information.

NVD Announcements
 SCAP Announcements
 SCAP Discussion List
 XCCDF Discussion List

Workload Index

Vulnerability Workload Index: 9.06

About Us

NVD is a product of the NIST [Computer Security Division](#) and is sponsored by the Department of Homeland Security's [National Cyber Security Division](#). It supports the

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):
CVSS v2 Base score: 7.5 (High) (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.milw0rm.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>

Common Platform Enumeration:

NVD XML feeds also include CWE

Vulnerability Type (View All)
SQL Injection (CWE-89)

CWE Common Weakness Enumeration

A Community-Developed Dictionary of Software Weakness Types

Home > CWE List > CWE-89 Individual Dictionary Definition (Draft 9) View the CWE List

CWE-89 Individual Dictionary Definition (Draft 9)

Weakness ID: 89 (Weakness Base) **Status:** Incomplete

Description: **Summary**
The application fails to adequately filter SQL syntax from user-controllable input. This can lead to such input being interpreted as SQL, rather than ordinary user data and be executed as part of a dynamically generated SQL query. This is a specific form of an injection problem, one that explicitly affects SQL databases, in which SQL commands are injected into data-plane input in order to effect the execution of dynamically generated SQL statements.

Likelihood of Exploit: Very High

Common Consequences:
Confidentiality: Since SQL databases generally hold sensitive data, loss of confidentiality is a frequent problem with SQL injection vulnerabilities.
Authentication: If poor SQL commands are used to check user names and passwords, it may be possible to connect to a system as another user with no previous knowledge of the password.
Authorization: If authorization information is held in a SQL database, it may be possible to change this information through the successful exploitation of a SQL injection vulnerability.
Integrity: Just as it may be possible to read sensitive information, it is also possible to make changes or even delete this information with a SQL injection attack.

Potential Mitigations:
Requirements specification: A non-SQL style database which is not subject to this flaw may be chosen.
Design: Follow the principle of least privilege when creating user accounts to a SQL database. Users should only have the minimum privileges necessary to use their account. If the requirements of the system indicate that a user can read and modify their own data, then limit their privileges so they cannot read/write others' data.
Design: Duplicate any filtering done on the client-side on the server side.
Implementation: Implement SQL strings using prepared statements that bind variables. Prepared statements that do not bind variables can be vulnerable to attack.

Industry Uptake

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, database) 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

- A Process for Performing Security Code Reviews, Michael Howard, IEEE Security & Privacy July/August 2006.
<http://msdn.microsoft.com/en-us/library/aa302432.aspx>
- .NET Framework Security — Code Review;
<http://www.mscorlib.com/en-us/library/aa302432.aspx>
- **Common Weakness Enumeration, MITRE; <http://cwe.mitre.org/>**
- **Security Code Reviews;**
http://www.codesecurely.org/Wiki/view.aspx/Security_Code_Reviews
- Security Code Review — Use Visual Studio Bookmarks To Capture Security Findings; <http://blogs.msdn.com/alki/archive/2008/01/24/security-code-review-use-visual-studio-bookmarks-to-capture-security-findings.aspx>
- Security Code Review Guidelines, Adam Shostack;
<http://www.verber.com/mark/cs/security/code-review.html>
- OS WASP Top Ten; http://www.owasp.org/index.php/OWASP_Top_Ten_Project

Testing

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 following 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. These testing methods do find security bugs, especially for products that may not have undergone critical secure development process changes.

Fuzz testing

Fuzz testing is a reliability and security testing technique that relies on providing 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/~bart/fuzz/fuzz.html>
- Automated Whitebox Fuzz Testing, Michael Levin, Patrice Godefroid and Dave Molnar, Microsoft Research;
<ftp://ftp.research.microsoft.com/pub/tr/TR-2007-58.pdf>
- IANewsletter Spring 2007 “Look out! It’s the fuzzi!” Matt Warnock;
http://iac.dtic.mil/iatac/download/Vol10_No1.pdf
- Fuzzing: Brute Force Vulnerability Discovery, Sutton, Greene & Amini, Addison-Wesley.
- Open Source Security Testing Methodology Manual, ISECOM.
- **Common Attack Pattern Enumeration and Classification, MITRE;**
<http://capec.mitre.org/>

CWE
CAPEC



Fundamental Practices for Secure Software Development A Guide to the Most Effective Secure Development Practices in Use Today

OCTOBER 8, 2008

LEAD WRITER Michael Howard, Microsoft Corp.

CONTRIBUTORS

Gunter Blitz, SAP AG
Jerry Cochran, Microsoft Corp.
Matt Coles, EMC Corporation
Danny Dhillon, EMC Corporation
Chris Fagan, Microsoft Corp.
Cassio Goldschmidt, Symantec Corp.
Wesley Higaki, Symantec Corp.
Steve Lipner, Microsoft Corp.
Brad Minnis, Juniper Networks, Inc.
Hardik Parekh, EMC Corporation
Dan Reddy, EMC Corporation
Alexandr Seleznyov, Nokia
Reeny Sondhi, EMC Corporation
Janne Uusilehto, Nokia
Antti Vähä-Sipilä, Nokia



Recent Posts

MS08-078 and the SDL
 Announcing CAT.NET CTP and AntIXSS v3 beta
 SDL videos
 BlueHat SDL Sessions Wrap-up
 Secure Coding Secrets?

Tags

Common Criteria **Crawl Walk Run**
 Privacy **SDL** SDL Pro Network
 Security Assurance Security Blackhat
 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)
 Microsoft Security Development Lifecycle (SDL) - Process Guidance (.doc)

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

```
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

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](#).

September 2008 (5)
 August 2008 (2)
 July 2008 (8)
 June 2008 (4)

TOCTOU issues. We will update our training to address this.

Our static analysis tools don't find this because the tools would need to understand the re-entrant nature of the code.


Fuzz Testing

SAMATE Reference Dataset

http://samate.nist.gov/SRD/

AFC Home MII Home Search Map/Ph/Weather/Travel Bob's Bookmarks CVEnOVAL OVAL shared SPAMmngt

» sign in register | Search... GO



SRD Home View / Download Search / Download More Downloads Submit Test

Welcome to the NIST SAMATE Reference Dataset Project

The purpose of the SAMATE Reference Dataset (SRD) is to provide users, researchers, and developers with 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 includes known bugs and vulnerabilities. The dataset intends to encompass a wide variety of test cases, including known bugs and vulnerabilities. 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 here](#) to view selected or all test cases. To find specific test cases, please [click here](#).

How to submit test cases

NIST Draft Special Publication 500-268

Source Code Security Analysis Tool Functional Specification Version 1.0

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

29 January, 2007

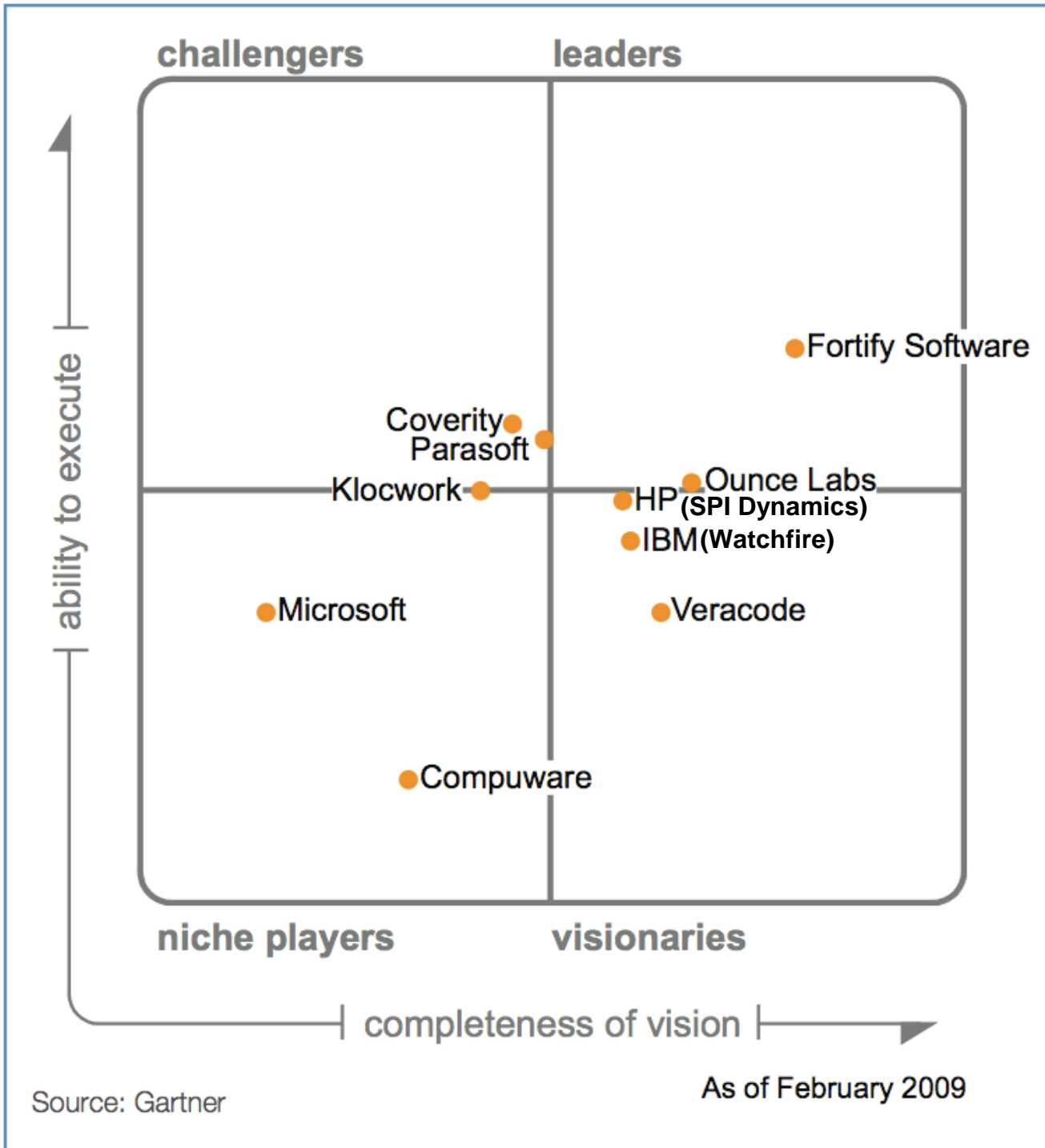
Michael Kass
Michael Koo

National Institute of Standards and Technology
Information Technology Laboratory
Software Diagnostics and Conformance Testing Division

Gartner Magic Quadrant for Static Application Security Testing Tools

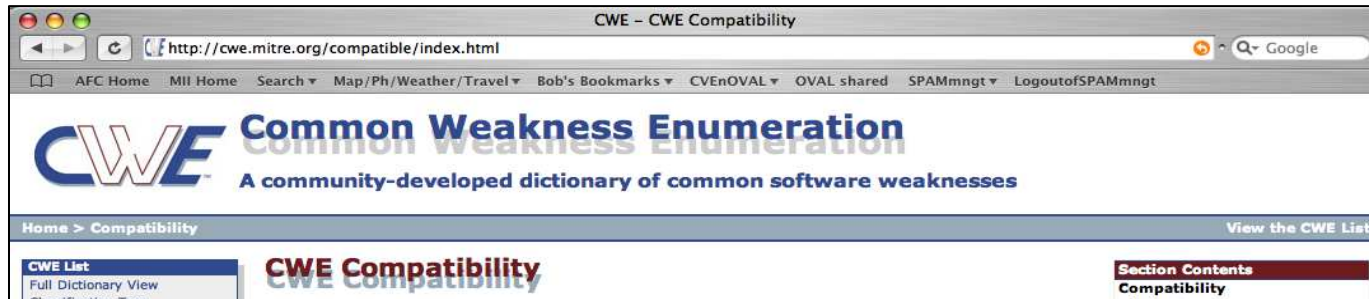
Plus Some Other Important Tool Players...

- Cenzic
- CAST Software
- Polyspace
- Security Innovation
- LDRA
- KDM Analytics
- SureLogic
- Programming Research Inc
- Armorize
- SofCheck
- GammaTech



CWE Compatibility & Effectiveness Program

(launched Feb 2007)



Organizations Participating

All organizations participating in the CWE Compatibility and Effectiveness Program are listed below, including those with CWE-Compatible Products and Services and those with Declarations to Be CWE-Compatible.

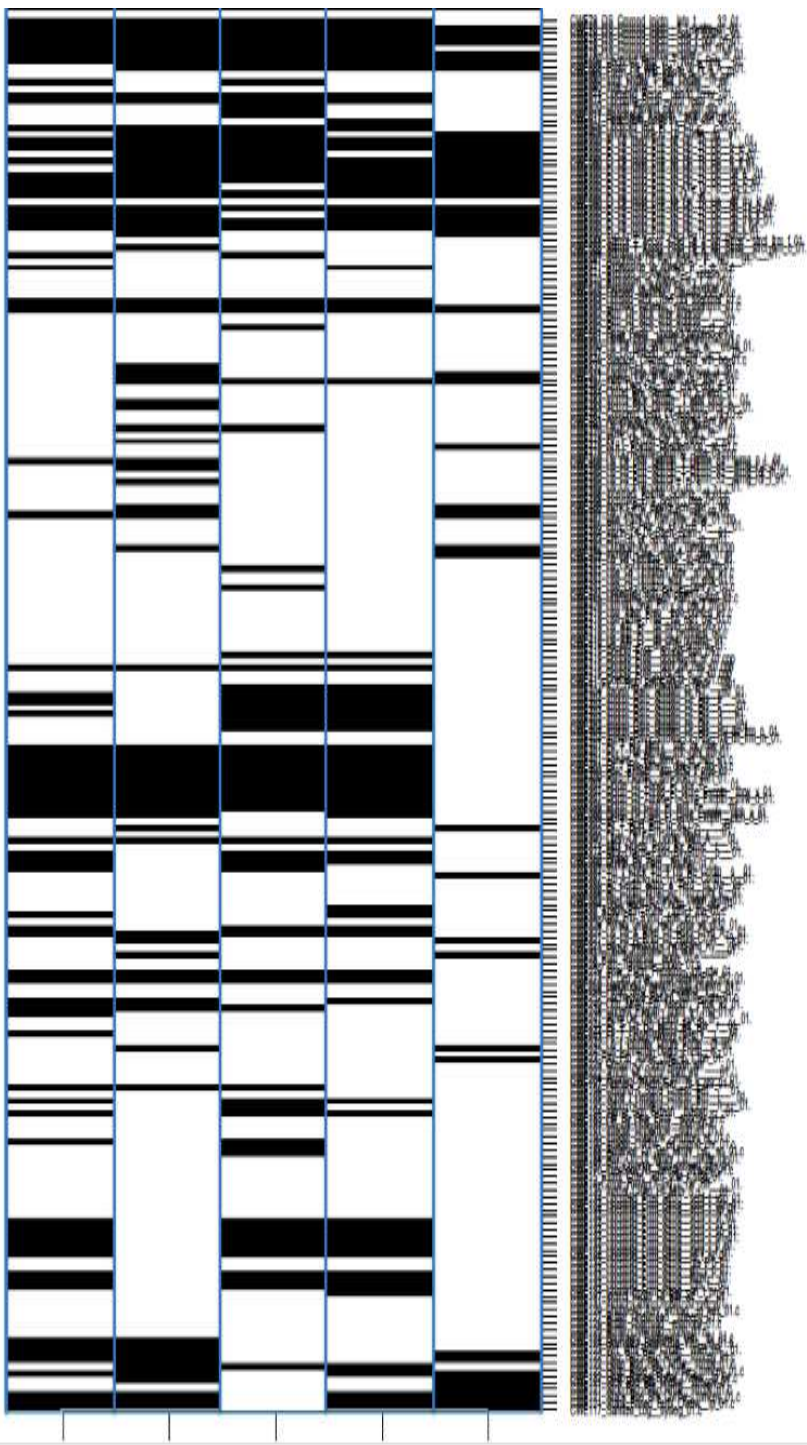
Products are listed alphabetically by organization name:

cwe.mitre.org/compatible/

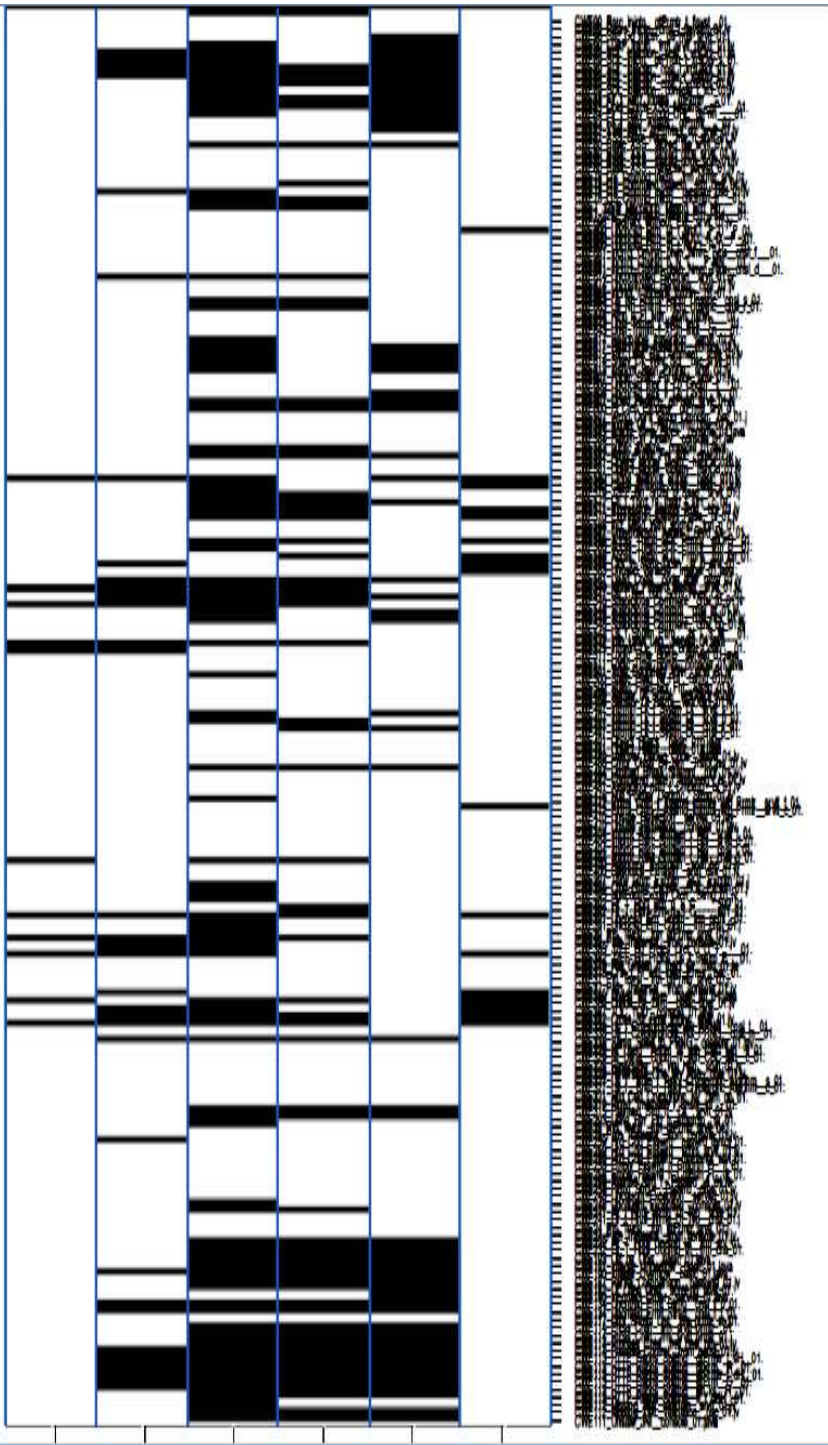
TOTALS
Organizations Participating: 28
Products & Services: 47

December 29, 2006

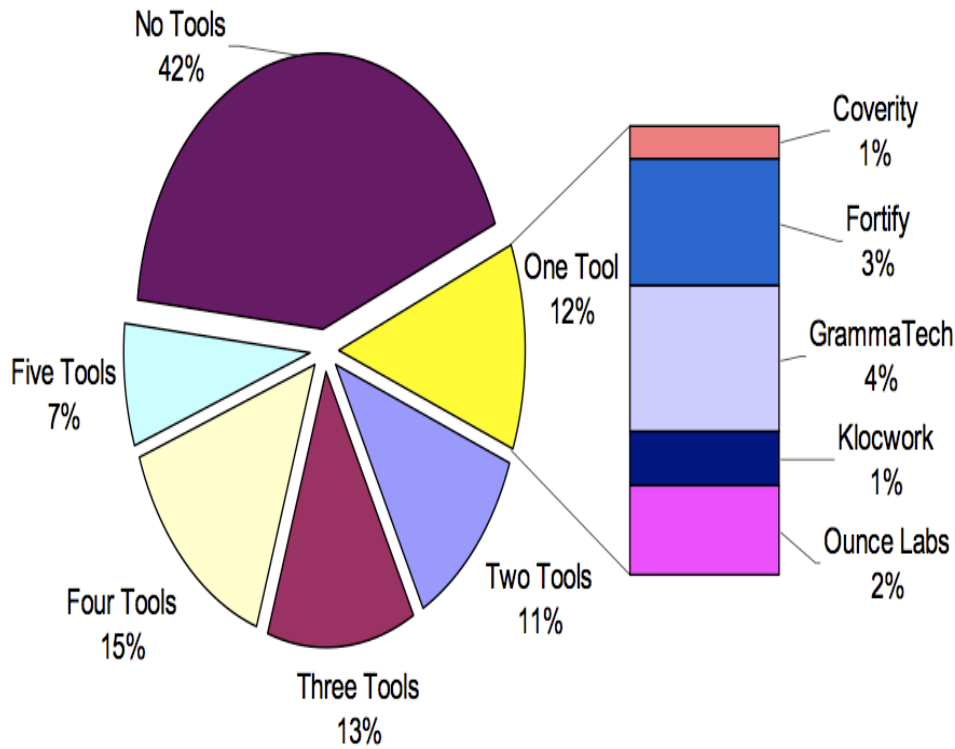
C Test Cases



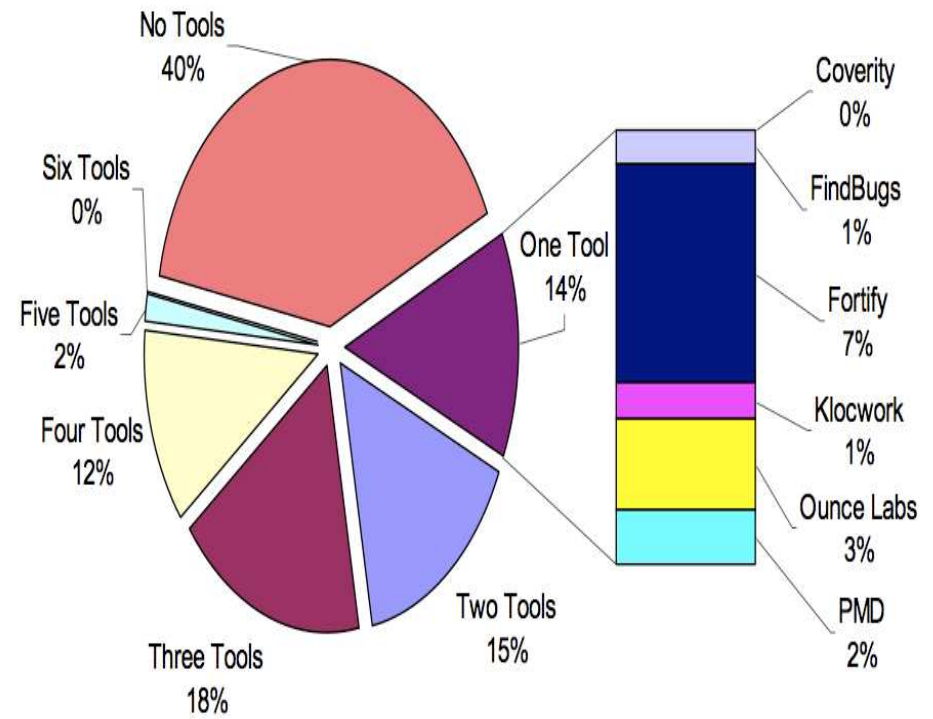
Java Test Cases



C/C++ "Breadth" Test Case Coverage



Java "Breadth" Test Case Coverage



2 March 2009 Jointly funded by:
Assistant Secretary of the Navy Chief System Engineer
197 Isaac Hull
Washington Navy Yard, DC
and Naval Ordnance Safety & Security Activity
Box 47, Bldg D-323
3817 Strauss Avenue Indian Head, MD 20640
Prepared by:Booz Allen Hamilton McLean, VA

Software Security Tools

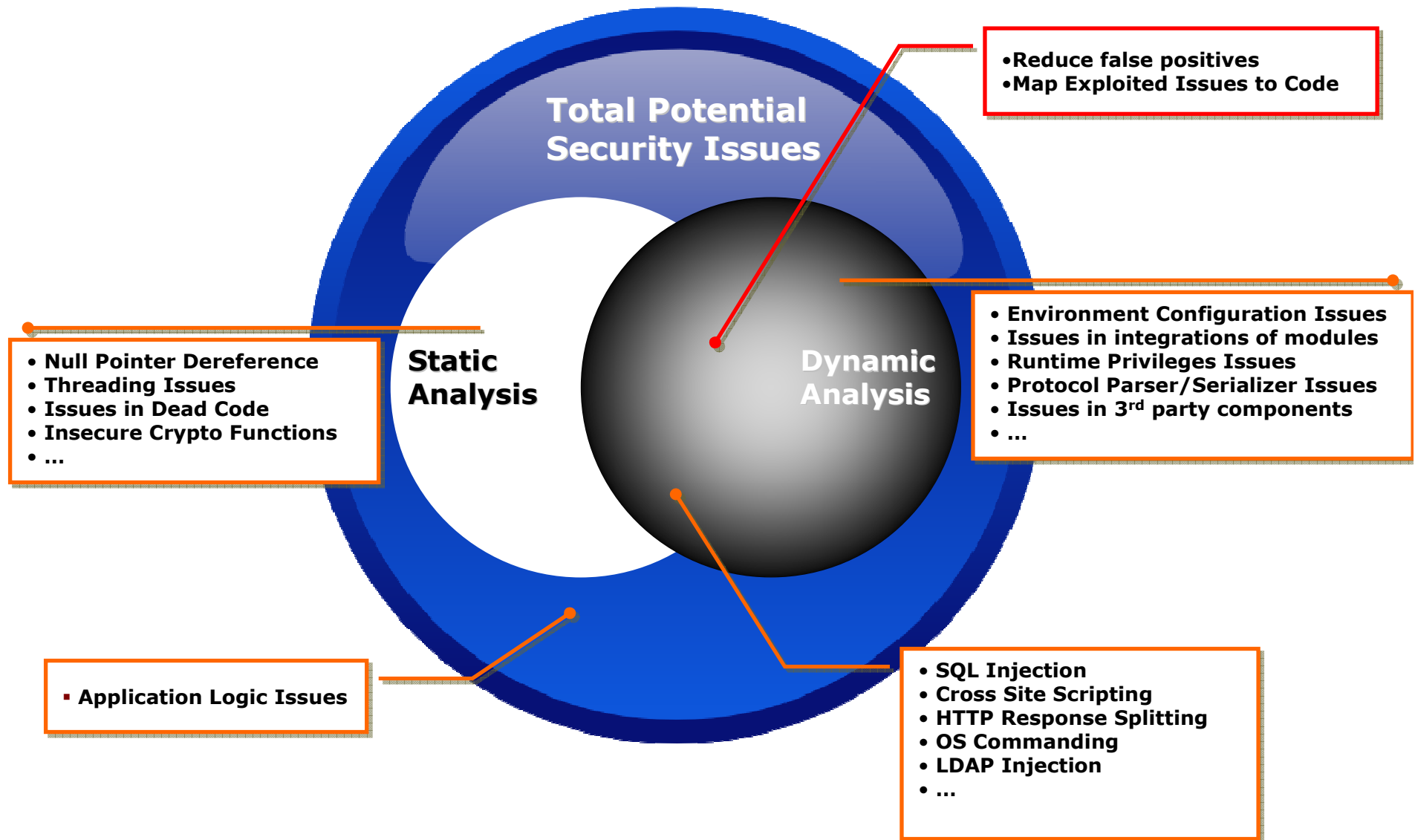
- Static Analysis
- Source Code Fault Injection
- Dynamic Analysis
- Architectural Analysis
- Pedigree Analysis
- Binary Code Analysis
- Disassembler Analysis
- Binary Fault Injection
- Fuzzing
- Malicious Code Detectors
- Bytecode Analysis

Software Security Assessment Tools Review

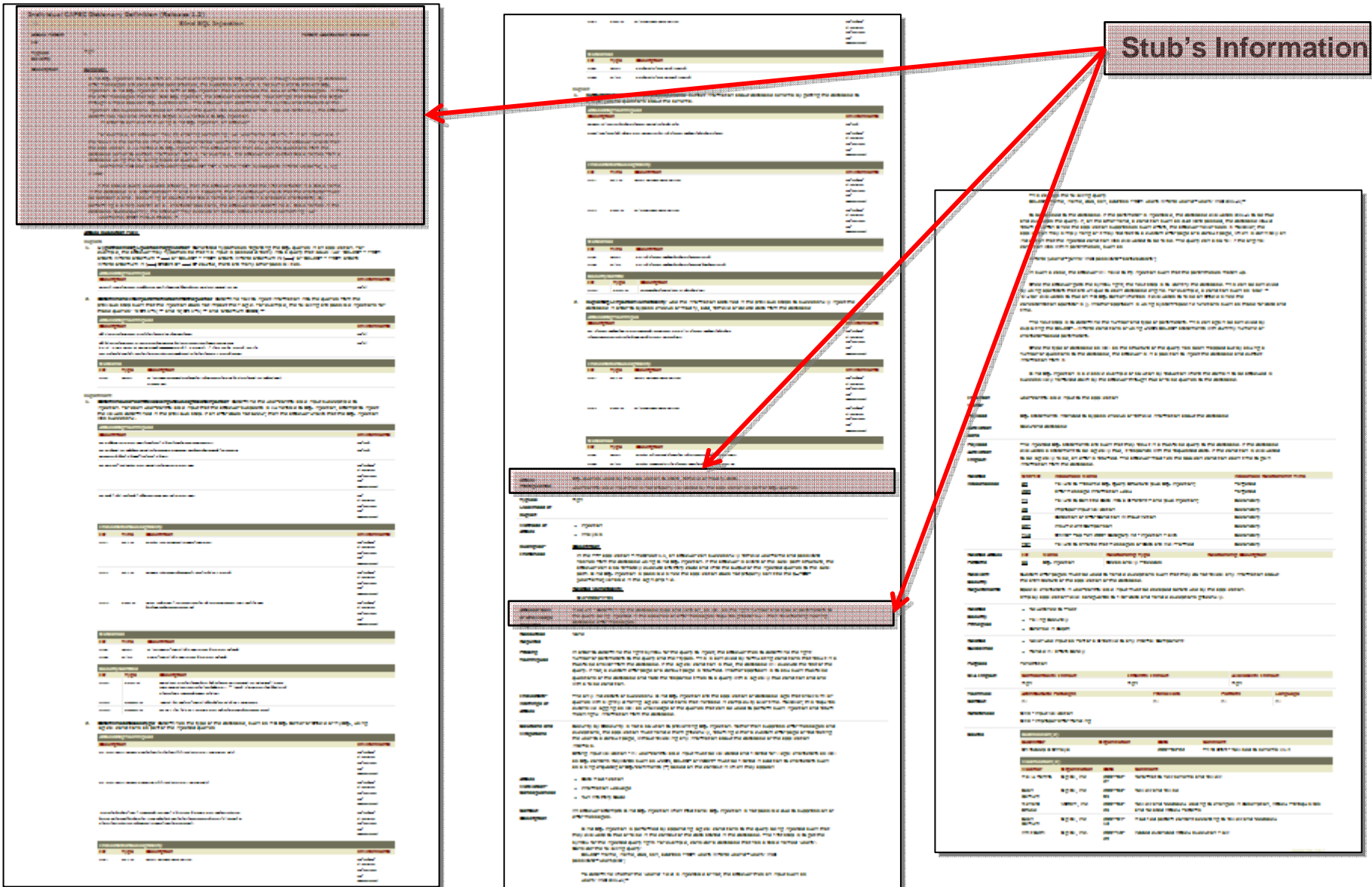
Summary of Evaluation	Static Analysis Code Scanning	Source Code Fault Injection	Dynamic Analysis	Architectural Analysis	Pedigree Analysis	Binary Code Analysis	Disassembler Analysis	Binary Fault Injection	Fuzzing	Malicious Code Detector	Byte Code Analysis
Key: X* - To be most beneficial X+ - In some cases X% - When possible X# - e.g., compilation											
When to Use											
Requirements				X							
Design				X*							
Implementation	X*	X	X	X	X*	X	X	X	X	X	X*
Testing	X	X*	X*	X	X	X	X	X*	X*	X	X
Production	X	X	X	X	X	X	X	X	X	X	X
Acquisition	X	X	X	X	X	X*	X*	X	X	X*	X
Required Skills (Understanding of...)											
Underlying source code		X									
Underlying development methodology			X#	X					X		
Implementing language	X	X					X%				
Binary						X+	X				
Bytecode											X
Testing methodology		X	X	X			X		X	X	X
Benefits											
Reduces cost over system life	X	X	X	X	X	X	X	X	X	X	X
Educates developers about secure programming	X			X	X						
Rechecks legacy code	X				X	X	X	X	X		X
Automates repetitive and tedious aspects of source code security audits	X				X						
Checks for good programming style	X										X
Increased test coverage		X						X	X		
Increased accuracy		X							X		
No need for source code			X	X		X	X	X	X	X	X
Improved accuracy and coverage		X	X					X	X		
Reduces the amount of testing necessary					X						
No disassembly	X	X	X	X	X	X		X	X		
Guaranteed the analysis is performed on the actual product			X+	X		X	X	X	X	X	X
Drawbacks											
No architectural-level flaws	X		X		X	X	X			X	X
Thorough understanding of the software			X	X				X	X		
Required expertise		X		X			X	X	X		
Requires use of open source software					X						
Lack of tool availability		X				X				X	
Licensing concerns							X				
Reliance on a primary vendor							X				
Additional analysis		X		X			X	X	X		X
Additional preparation		X	X	X					X		
Limited to a single language											X

Table 1: Evaluation Summary of Analyses for Source Code, Executables, and Intermediate Representations

Value of Aligning Multiple Perspectives



Complete CAPEC Entry Information



Linkage with Fundamental Changes in Enterprise Security Initiatives

Twenty Critical Controls for Effective Cyber Defense 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 Applications](#)
- [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 Security](#)
- [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: Data Protection](#)
- [Critical Control 11: Incident Response and Computer Forensics](#)
- [Critical Control 12: Business Continuity and Disaster Recovery](#)
- [Critical Control 13: Multi-Factor Authentication](#)
- [Critical Control 14: Security Awareness and Training](#)
- [Critical Control 15: Vendor Managed Security](#)
- [Critical Control 16: Information Security Policies](#)
- [Critical Control 17: Risk Assessment](#)
- [Critical Control 18: Security Assessments](#)
- [Critical Control 19: Security Incident Response and Computer Forensics](#)
- [Critical Control 20: Security Incident Response and Computer Forensics](#)

CAG: Critical Control 7: Application Software Security

<< previous control

Consensus Audit Guidelines

next control >>

How do attackers exploit the lack of this control?

Attacks against vulnerabilities in web-based and other application software have been a top priority for criminal organizations in recent years. Application software that does not properly check the size of user input, fails to sanitize user input by filtering out unneeded but potentially malicious character sequences, or does not initialize and clear variables properly could be vulnerable to remote compromise. Attackers can inject specific exploits, including buffer overflows, SQL injection attacks, and cross-site scripting code to gain control over vulnerable machines. In one attack in 2008, more than 1 million web servers were exploited and turned into infection engines for visitors to those sites using SQL injection. During that attack, trusted websites from state governments and other organizations compromised by attackers were used to infect hundreds of thousands of

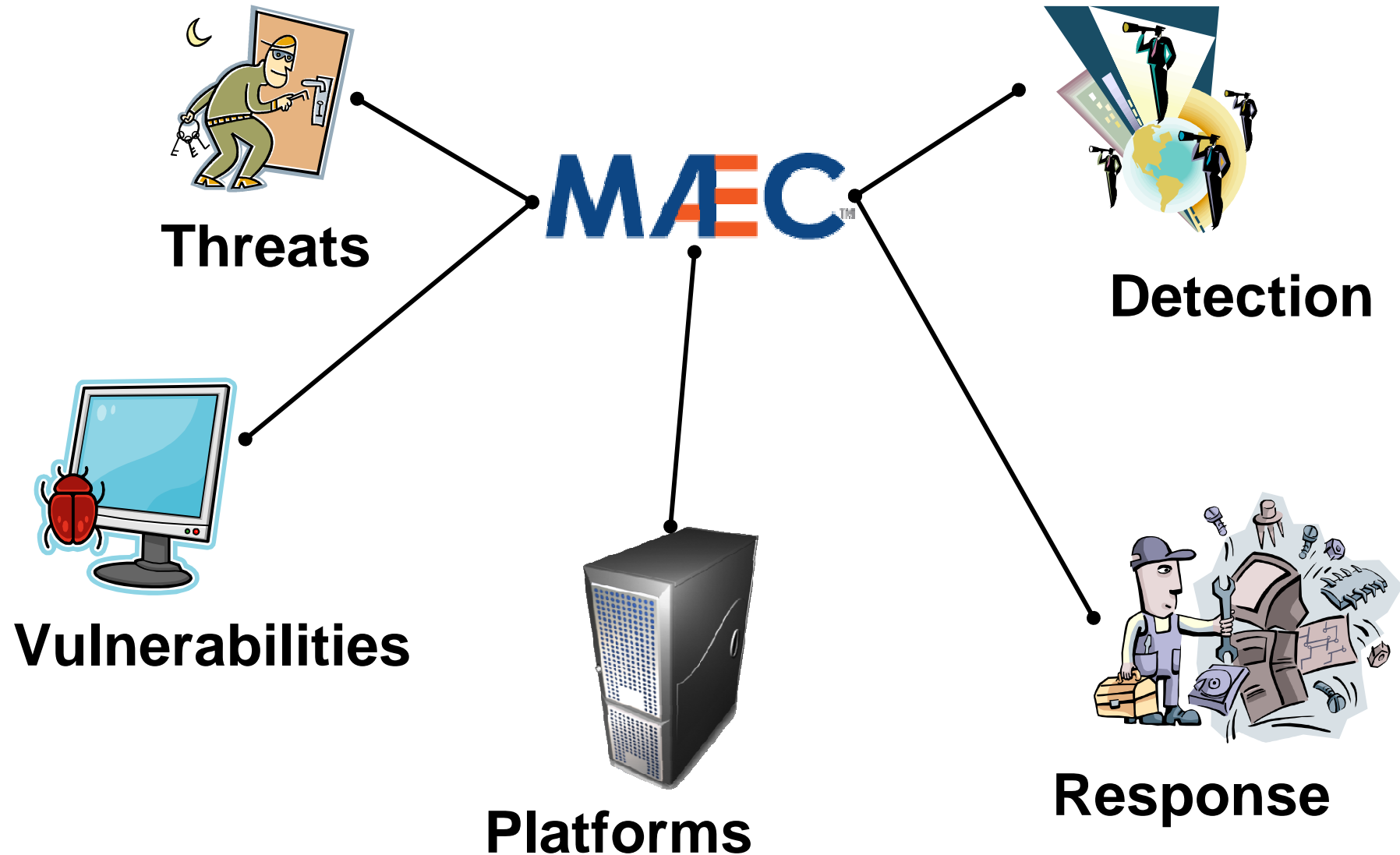
CWE and CAPEC included in Control 7 of the “Twenty Critical Controls for Effective Cyber Defense: Consensus Audit Guidelines”

Procedures and tools for implementing t

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.

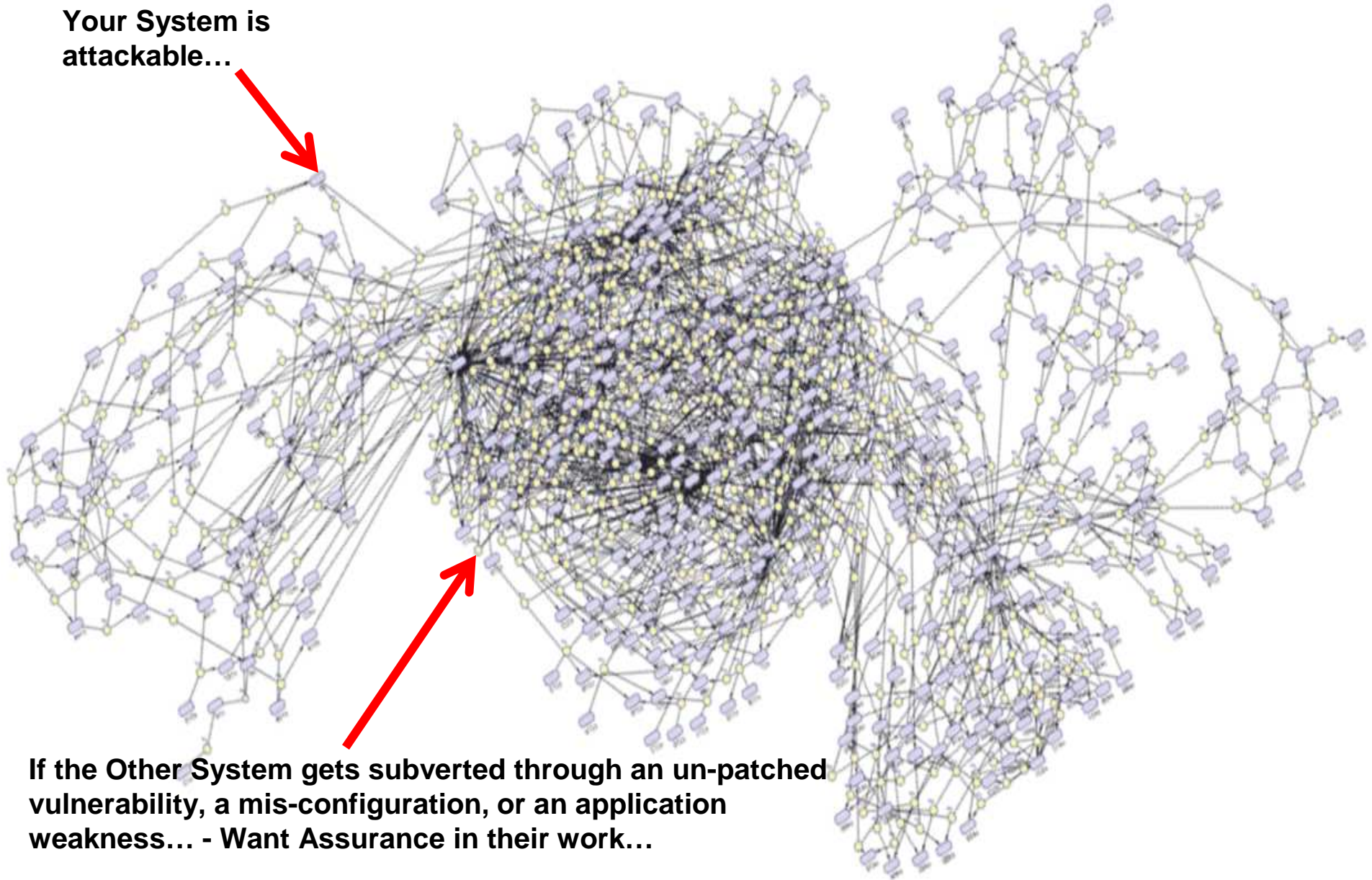


Correlate, Integrate, Automate



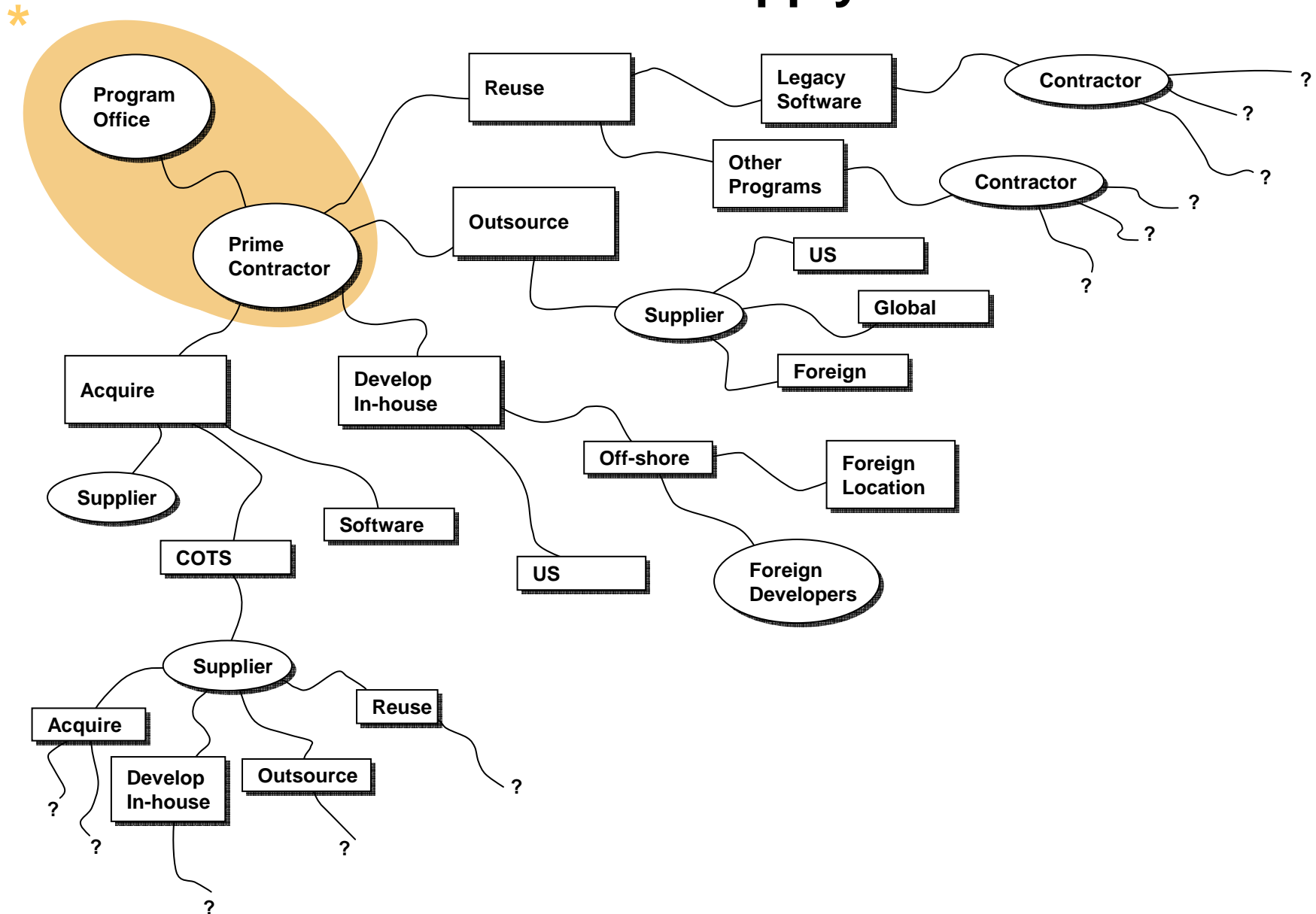
Today Everything's Connected

Your System is
attackable...



If the Other System gets subverted through an un-patched vulnerability, a mis-configuration, or an application weakness... - Want Assurance in their work...

The Software Supply Chain



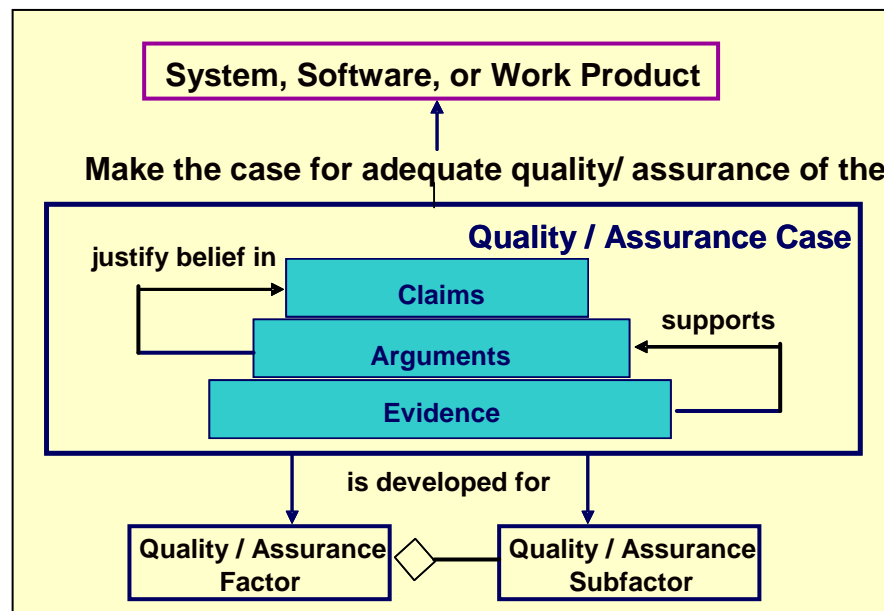
* “Scope of Supplier Expansion and Foreign Involvement” graphic in DACS www.softwaretchnews.com Secure Software Engineering, July 2005 article “Software Development Security: A Risk Management Perspective” synopsis of May 2004 GAO-04-678 report “Defense Acquisition: Knowledge of Software Suppliers Needed to Manage Risks”

ISO/IEC/IEEE 15026 Assurance Case

- Set of structured assurance claims, supported by evidence and reasoning (arguments), that demonstrates how assurance needs have been satisfied.
 - Shows compliance with assurance objectives
 - Provides an argument for the safety and security of the product or service.
 - Built, collected, and maintained throughout the life cycle
 - Derived from multiple sources

- Sub-parts

- A high level summary
- Justification that product or service is acceptably safe, secure, or dependable
- Rationale for claiming a specified level of safety and security
- Conformance with relevant standards & regulatory requirements
- The configuration baseline
- Identified hazards and threats and residual risk of each hazard / threat
- Operational & support assumptions



Attributes

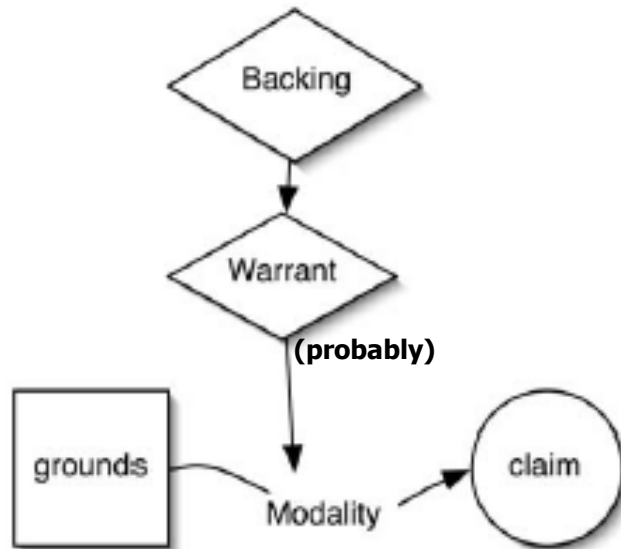
- Clear
- Consistent
- Complete
- Comprehensible
- Defensible
- Bounded
- Addresses all life cycle stages

Assurance Claims with Support by 'Substantial' Reasoning

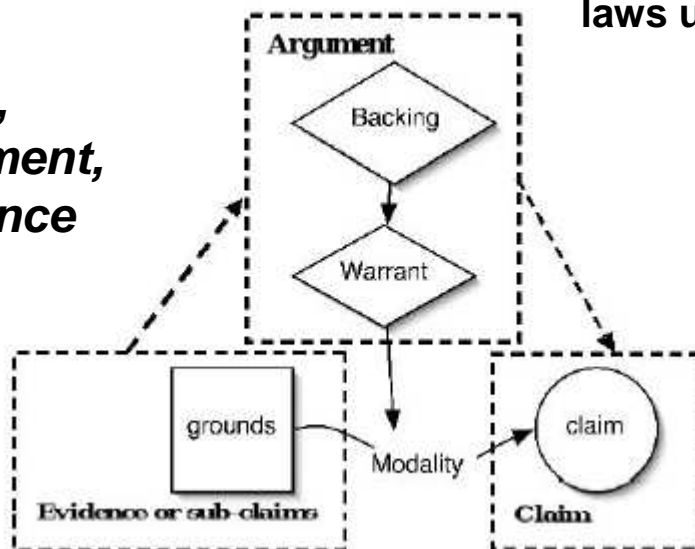


Stephen Toulmin, 1958

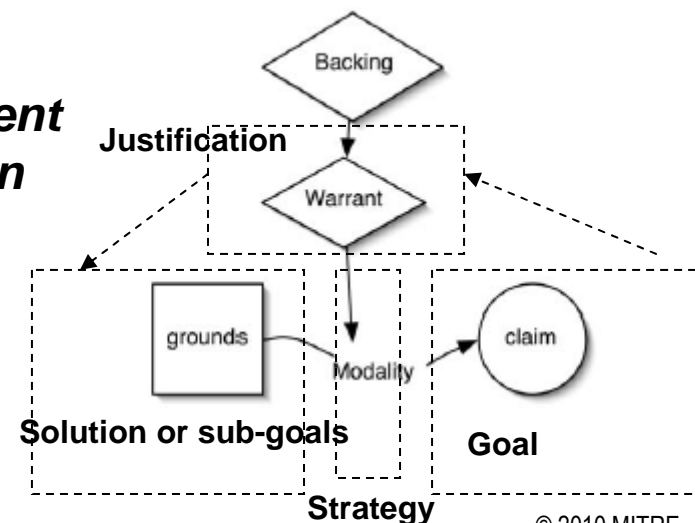
- Claims are assertions put forward for general acceptance
- The justification for claim is based on some grounds, the “specific facts about a precise situation that clarify and make good for a claim”
- The basis of the reasoning from the grounds (the facts) to the claim is articulated. Toulmin coined the term “warrant” for “substantial argument”. These are statements indicating the general ways of argument being applied in a particular case and implicitly relied on and whose trustworthiness is well established”.
- The basis of the warrant might be questioned, so “backing” for the warrant may be introduced. Backing might be the validation of the scientific and engineering laws used



CAE
*Claim,
 Argument,
 Evidence*

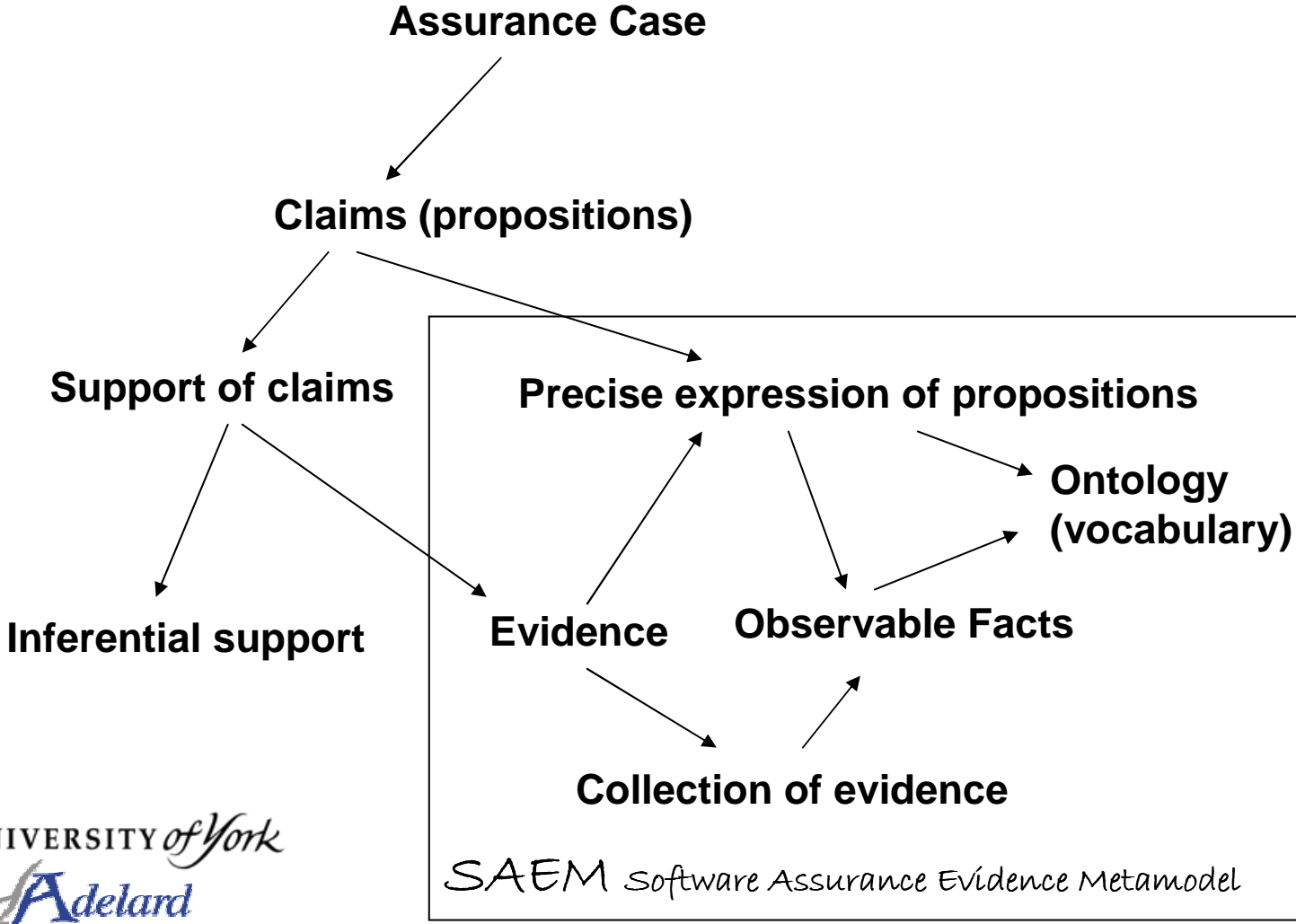


GSN
*Goal
 Statement
 Notation*



Object Management Group (OMG) Systems Assurance Task Force Claims-Evidence-Arguments Overview

ARM Argumentation Metamodel



SBVR
Semantic
Business
Vocabulary
& Rules

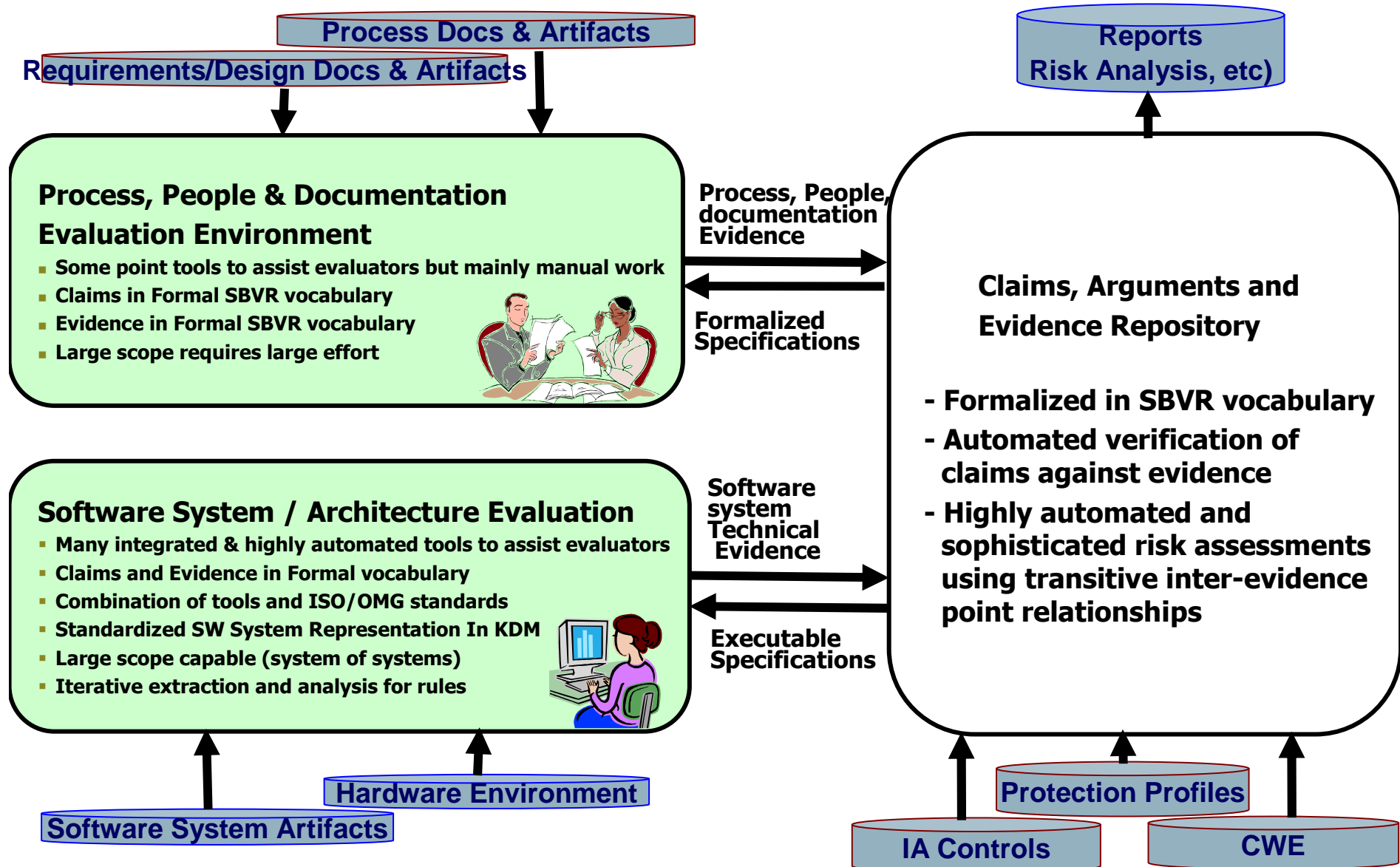


KDM Analytics

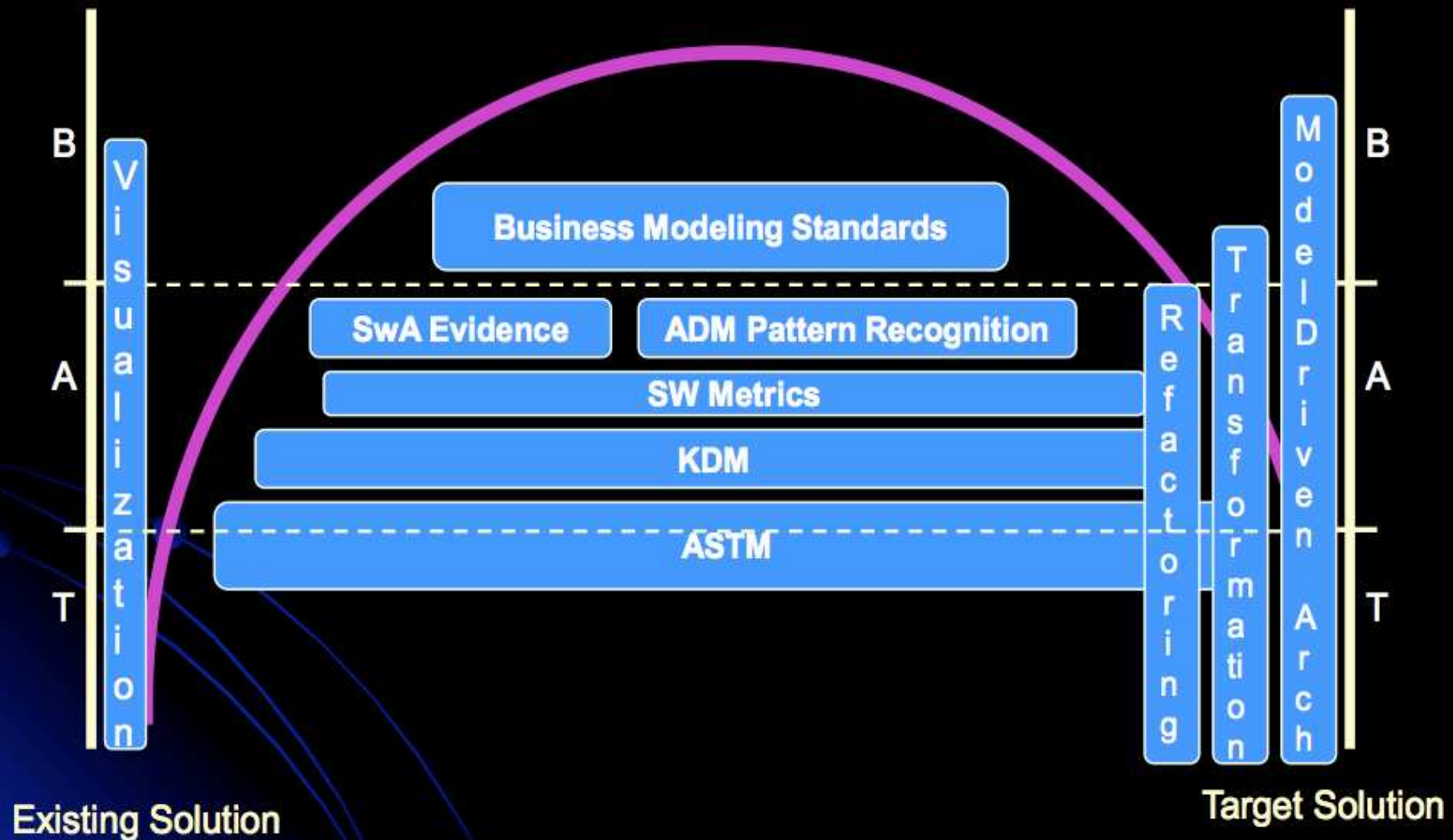
KDM Knowledge Discovery Metamodel

Software Assurance Ecosystem: The Formal Framework

The value of formalization extends beyond software systems to include related software system process, people and documentation



ADM Standards Span the Entire IT Architecture Spectrum





ISO IEC JTC 1/SC 27 NXXXX
 ISO/IEC JTC 1/SC 27/WG x NXXXXX
 REPLACES: N

ISO/IEC JTC 1/SC 27
 Information technology - Security techniques
 Secretariat: DIN, Germany

DOC TYPE: NB MWI Proposal for a technical report (TR)

TITLE: National Body New Work Item Proposal on "Secure software development and evaluation under ISO/IEC 15406 and ISO/IEC 18405"

SOURCE: INCITSACS 1, National Body of (US)

DATE: 2006-09-30

PROJECT: 15406 and 18405

STATUS: This document is circulated for consideration at the forthcoming meeting of SC 27/WG 3 to be held in Redmond (WA, USA) on 27 - 29 November 2006.

ACTION ID: ACT

DUE DATE:

DISTRIBUTION: P, D and L-Members
 W: Larry S.D. Z. Chairman
 M. De Soete, SC 27 Vice-Chair
 E. J. Humphreys, K. Naraenara, Y. Bafkim, M.-C. Kang, K. Ramnberg, WG-Consensus

MEDIUM: Live Intranet

NO. OF PAGES: xx

Secretariat ISO/IEC JTC 1/SC 27 -
 DIN Document Institute für Normung e. V., Burgstraße 6, 10772 Berlin, Germany
 Telephone: +49 30 201-10552; Facsimile: +49 30 2501-7233; E-mail: iso@vde.com
 HTTP://www.iso.org/secretariat

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

New Work Item Proposal
 NP submitting
PROPOSAL FOR A NEW WORK ITEM

Date of presentation of proposal YYYY-MM-DD	Proposer: ISO/IEC JTC 1/SC 27
Secretariat National Body	ISO/IEC JTC 1 N XXXX ISO/IEC JTC 1/SC 27 N

A proposal for a new work item shall be submitted to the secretariat of the ISO/IEC joint technical committee concerned with a copy to the ISO Central Secretariat.

Presentation of the proposal

Title: Secure software development and evaluation under ISO/IEC 15406 and ISO/IEC 18405

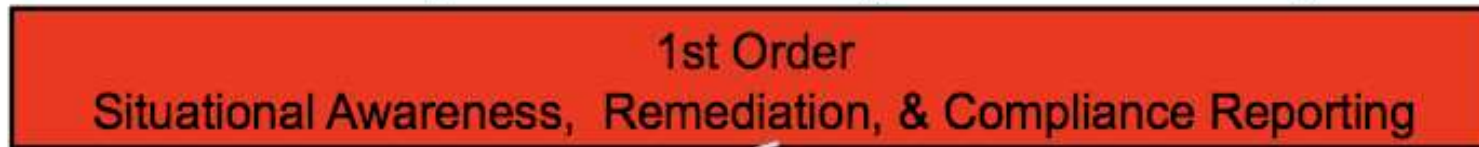
Scope

In the case where a target of evaluation (TOE) being evaluated, under ISO/IEC 15406 and ISO/IEC 18405, includes specific software portions, the TOE developer may optionally present the developer's technical rationale for mitigating software common attack patterns and related weaknesses as described in the latest revision of the Common Attack Pattern Enumeration and Classification (CAPEC) model table from <http://capec.mitre.org/>. The developer's technical rationale is expected to include a range of mitigation techniques, from architectural properties to design features, coding techniques, use of tools or other means.

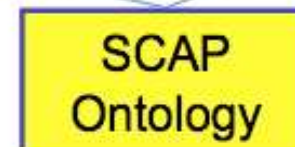
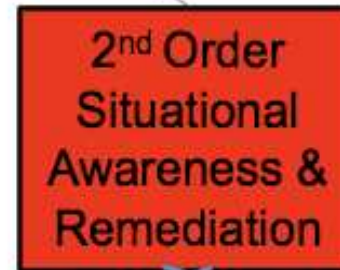
This Technical Report (TR) provides guidance for the developer and the evaluator on how to use the CAPEC as a technical reference point during the TOE development phase and in an evaluation of the TOE secure software under ISO/IEC 15406 and 18405, by addressing:

- A refinement of the IS 15406 Attack Potential evaluation table for software, taking into account the entries contained in the CAPEC and their characteristic.
- How the information for mitigating software common attack patterns and related weaknesses is used in an IS 15406 evaluation, in particular providing guidance on how to determine which attack patterns and weaknesses are applicable to the TOE, taking into consideration of:
 - the TOE technology;
 - the TOE security problem definition;
 - the interfaces the TOE exports that can be used by potential attackers;
 - the Attack Potentials that the TOE needs to provide resistance for.
- How the technical rationale provided by the developer for mitigating software common attack patterns and related weaknesses is used in the evaluation of the TOE design and the development of test cases.
- How the CAPEC and related Common Weakness Enumeration (CWE) weaknesses are used by the evaluator, who needs to consider at the applicant's attack patterns and be able to exploit specific related software weaknesses while performing the subsequent vulnerability analysis (AVA_VAN) activities on the TOE.
- How incomplete entries from the CAPEC are resolved during an IS 15406 evaluation.
- How the evaluator's attack and weakness analysis of the TOE incorporates other attacks and weaknesses not yet documented in the CAPEC.

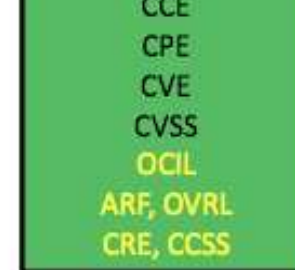
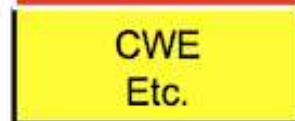
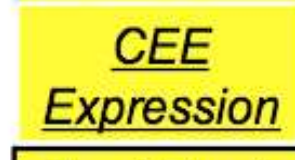
The TR also investigates specific elements from the ISO/IEC 15026 (and its revision) are applicable to the guidelines being developed in the TR within the context of IS 15406 and 18405.

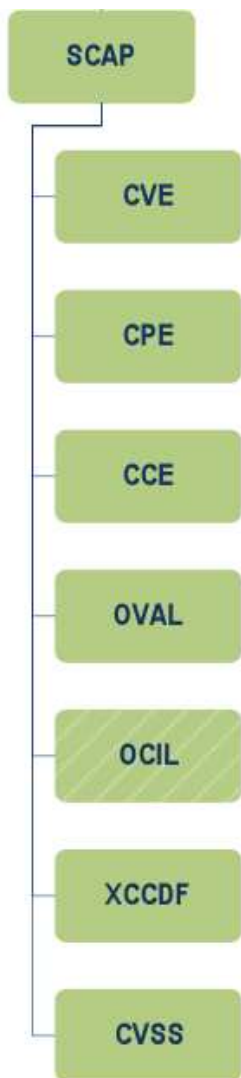


Receive 1st order for 'free' by virtue of 2nd order ontologies mapped through SCAP.



NIST Validation





SCAP 1.1 uses the following specifications:

- Extensible Configuration Checklist Description Format (XCCDF) 1.1.4, a language for authoring security checklists/benchmarks and for reporting results of checklist evaluation [QUI08]
- Open Vulnerability and Assessment Language (OVAL) 5.6, a language for representing system configuration information, assessing machine state, and reporting assessment results
- Open Checklist Interactive Language (OCIL) 2.0, a language for representing security checks that requires human feedback
- Common Platform Enumeration (CPE) 2.2, a nomenclature and dictionary of hardware, operating systems, and applications [BUT09]
- Common Configuration Enumeration (CCE) 5, a nomenclature and configurations
- Common Vulnerabilities and Exposures (CVE), a nomenclature and software flaws⁹
- Common Vulnerability Scoring System (CVSS) 2.0, an open speci severity of software flaw vulnerabilities [MEL07].

NIST
National Institute of
Standards and Technology
U.S. Department of Commerce

Special Publication 800-126
Revision 1 (DRAFT)

**The Technical Specification
for the Security Content
Automation Protocol (SCAP):
SCAP Version 1.1 (DRAFT)**

Recommendations of the National Institute
of Standards and Technology

Stephen Quinn
David Waltermire
Christopher Johnson
Karen Scarfone
John Banghart

4.	SCAP General Requirements and Conventions	4-1
4.1	Support for Legacy SCAP Versions	4-1
4.2	XCCDF Conventions and Requirements	4-1
4.2.1	Metadata Elements	4-1
4.2.2	Use of CPE Names	4-2
4.2.3	The <xccdf:Benchmark> Element	4-3
4.2.4	The <xccdf:Profile> Element	4-3
4.2.5	The <xccdf:Rule> Element	4-4
4.2.6	Allowed Check System Usage	4-5
4.2.7	XCCDF Test Results	4-10
4.3	OVAL Conventions and Requirements	4-12
4.3.1	Supported Previous Versions of OVAL (5.3, 5.4, and 5.5)	4-13
4.3.2	Support for Deprecated Constructs in OVAL	4-13
4.3.3	OVAL Schema Specification	
4.3.4	OVAL Results	
4.4	OCIL Conventions	
4.5	CPE Conventions	
4.6	CCE Conventions	
4.7	CVE Conventions	
4.8	CVSS Conventions	

**The Technical Specification
for the Security Content
Automation Protocol (SCAP):
SCAP Version 1.1 (DRAFT)**

Recommendations of the National Institute
of Standards and Technology

Stephen Quinn
David Waltermire
Christopher Johnson
Karen Scarfone
John Banghart

5.	SCAP Use Case Requirements.....	5-1
5.1	SCAP Data Streams.....	5-1
5.2	SCAP Configuration Verification.....	5-1
5.3	SCAP Vulnerability Assessment.....	5-3
5.3.1	SCAP Vulnerability Assessment Using XCCDF and OVAL	5-3
5.3.2	SCAP Vulnerability Assessment Using Standalone OVAL	5-4
5.3.3	OVAL Definitions and Vulnerability Assessment.....	5-4
5.4	Patch Validation	5-4
5.4.1	Using OVAL Definitions for Patch Validation	5-5
5.4.2	Referencing an OVAL Patch Data Stream.....	
5.5	SCAP Inventory Collection	

**The Technical Specification
for the Security Content
Automation Protocol (SCAP):
SCAP Version 1.1 (DRAFT)**

Recommendations of the National Institute
of Standards and Technology

Stephen Quinn
David Waltermire
Christopher Johnson
Karen Scarfone
John Banghart

SwAAP

CWE

CAPEC

MAEC

CWSS

OMG SAEM

OMG ARG

SAFES

"Food Label"

OMG SMM

ISO 15026

OMG KDM

OMG ASTM

● Software Assurance Automation Protocol (**SwAAP**)

- For measuring & enumerating software weaknesses and the assurance cases.

Common Weakness Enumeration (**CWE**),

Common Attack Pattern Enumeration & Classification (**CAPEC**),

Malware Attribute Enumeration & Characterization (**MAEC**),

Common Weakness Scoring System (**CWSS**),

OMG Software Assurance Evidence Metamodel (**OMG SAEM**),

OMG Argumentation Metamodel (**OMG ARG**),

Software Assurance Findings Expression Schema (**SAFES**),

NIST SAMATE's "Food Label",

OMG Structured Metrics Metamodel (**OMG SMM**),

ISO "Assurance Case" 15026 (**ISO 15026**),

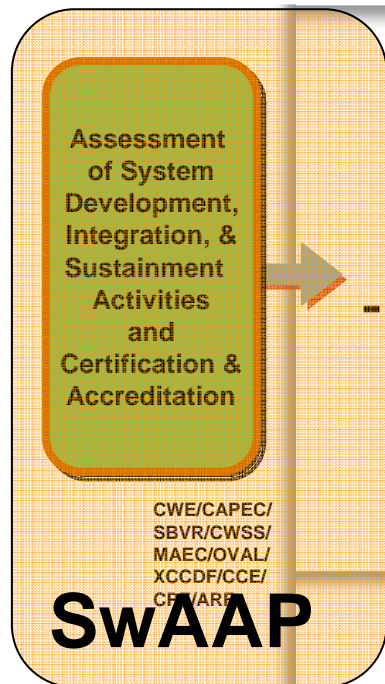
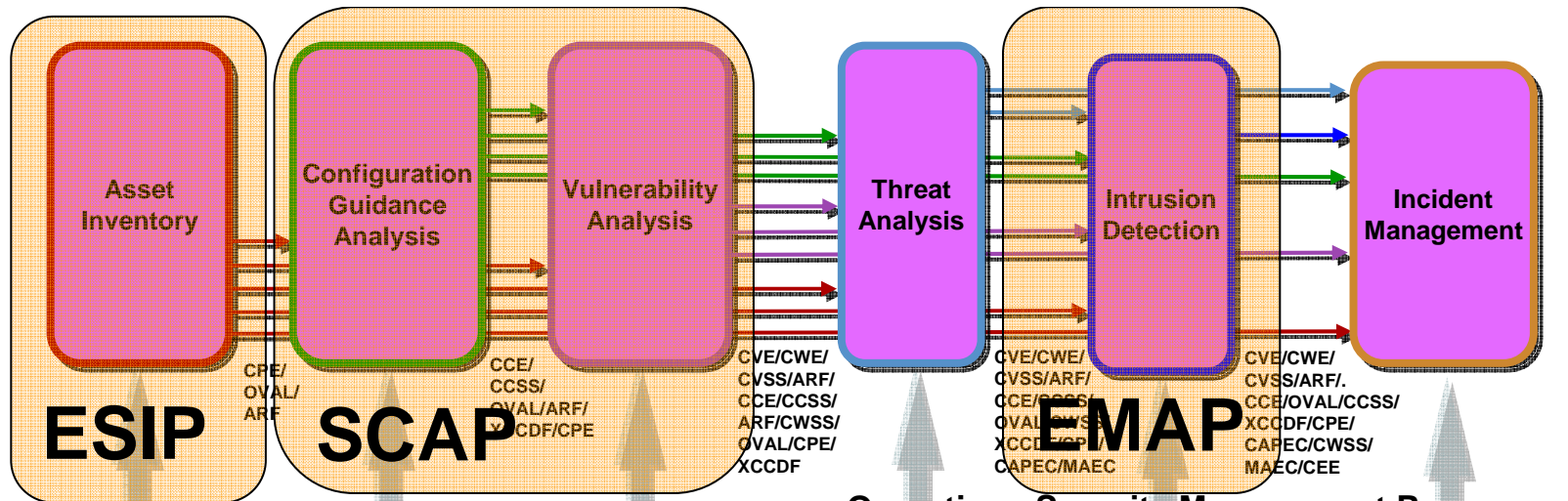
OMG Knowledge Discovery Metamodel (**OMG KDM**),

OMG Abstract Syntax Tree Metamodel (**OMG ASTM**)

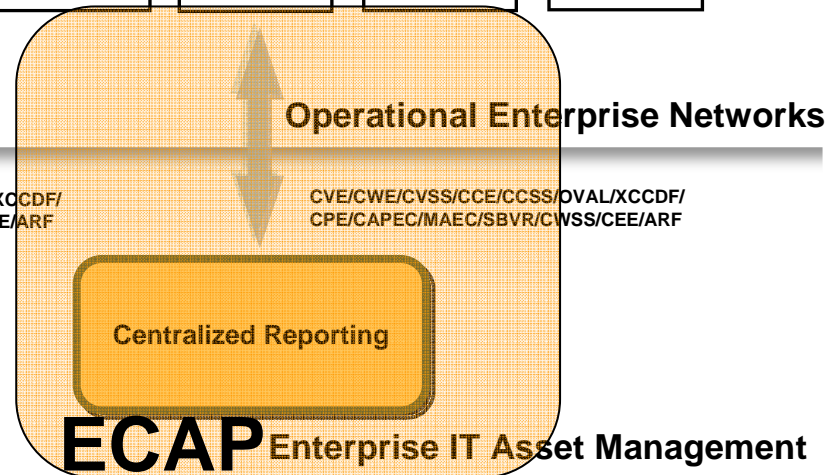
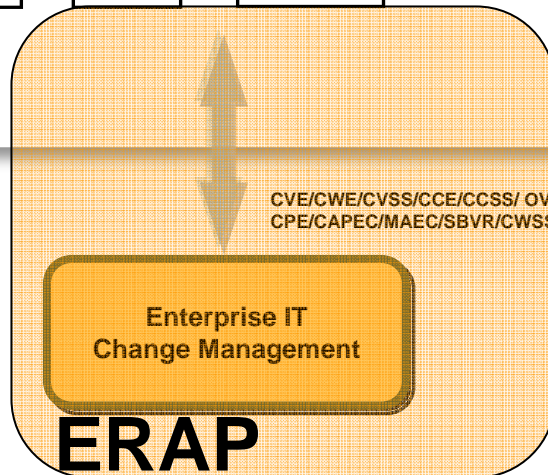
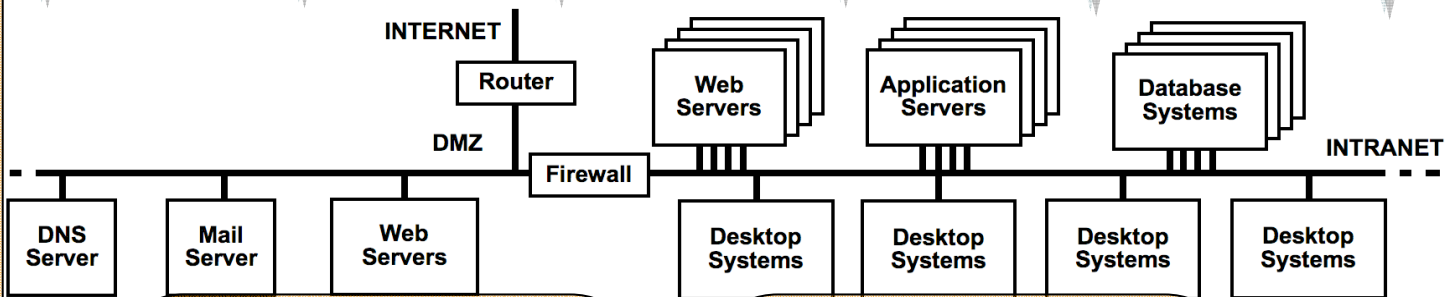
- plus SCAP to capture "accredited" system CPEs and CCE settings?
- OVAL checks for capturing "finger print" of software applications to address supply-chain risk measurement?

“Other” Automation Protocols (“O”AP)

- Event Management Automation Protocol (EMAP)
 - For reporting of security events. Common Event Expression (CEE), Malware Attribute Enumeration & Characterization (MAEC), and CAPEC.
- Enterprise Remediation Automation Protocol (ERAP)
 - For automated remediation of mis-configuration & missing patches. Common Remediation Enumeration (CRE) and Extended Remediation Information (ERI).
- Enterprise Compliance Automation Protocol (ECAP)
 - For reporting configuration compliance. Asset Reporting Format (ARF), Open Checklist Reporting Language (OCRL), etc.
- Enterprise System Information Protocol (ESIP)
 - For reporting of asset inventory information.



Development & Sustainment Security Management Processes



The image features a solid blue background. In the center, there are dark silhouettes of a dog and a cat. The dog is on the left, facing right, with its tail curved upwards. The cat is on the right, facing left, with its tail curved downwards. The word "Questions?" is written in white, bold, sans-serif font across the middle of the image, overlapping the silhouettes.

Questions?

ramartin@mitre.org