



LEADERSHIP FOR IT SECURITY & PRIVACY ACROSS HHS

HHS CYBERSECURITY PROGRAM

OFFICE OF THE CHIEF INFORMATION OFFICER

HC3 Intelligence Briefing Vulnerability/Patch Management for the Healthcare Enterprise

OVERALL CLASSIFICATION IS

UNCLASSIFIED

TLP:WHITE

06/06/2019

Agenda

- ▶ Introduction/Overview
- ▶ Vulnerability types
- ▶ Healthcare Enterprise Vulnerabilities
- ▶ Vulnerability Management Lifecycle
- ▶ Vulnerability Identification
- ▶ Vulnerability databases
- ▶ Common Vulnerabilities and Exposures
- ▶ National Vulnerability Database
- ▶ Common Weakness Enumeration
- ▶ Best practices
- ▶ References
- ▶ Questions



Image source: HealthITSecurity.com

Slides Key:



Non-Technical: managerial, strategic and high-level (general audience)



Technical: Tactical / IOCs; requiring in-depth knowledge (sysadmins, IRT)

Introduction

- ▶ What is a vulnerability?
 - A condition or state of being exposed to attack (hardware or software)
 - Sometimes colloquially referred to as an exploit (but in reality they are two different things)
 - There is no limit; new vulnerabilities are always being discovered
 - Exploitation of a vulnerability results in the compromise of confidentiality, availability and/or integrity of information resources
 - Could simply be a misconfiguration: <https://threatpost.com/files-exposed-record-misconfigs/145177/>
 - Vulnerabilities are not limited to technology
 - People (phishing, other social engineering)
 - Policy (gaps and obsolete/incomplete procedures)
 - In this presentation, we will be discussing technology vulnerabilities
- ▶ Why do vulnerabilities matter?
 - Many cyberattacks rely on specific vulnerabilities to succeed
 - Exceptions: Some DoS/DDoS and attack vectors such as social engineering/phishing



Cybersecurity: One in three breaches are caused by unpatched vulnerabilities



Introduction (continued)

- ▶ What is a patch?
 - An official, vendor-released modification to an application, operating system or other program that is designed to make it more secure and/or add features to it
- ▶ What is patch management?
 - National Institute for Standards and Technology (NIST): “Patch management is the process for identifying, acquiring, installing and verifying patches for products”
- ▶ Why conduct patch/vulnerability management?
 - Directly correlates with enterprise risk management
 - Patches can add additional features to software/applications
 - Regulatory/legal/insurance compliance
- ▶ What is a zero day? Why are zero days important?
 - An exploit/vulnerability that is not known to the vendors/developers
 - The vendor has known about this vulnerability for “x days”
 - Because zero days are unknown, they offer the attacker the element of surprise and an opportunity to gain a critical time advantage against defenders during an attack
- ▶ What is end-of-life software?
 - Software that is no longer supported by the vendor and therefore potentially exposed to new vulnerabilities



Image source: bluesolutions.co.uk



Vulnerability types

What are the different general vulnerability categories?

- ▶ **Network** – Typically related to generic network protocols or protocols related to infrastructure devices (switches, hubs, routers, certain types of servers); Often result in network disruption, data exfiltration or further penetration of the victim infrastructure, or data exfiltration
- ▶ **Software/Application** – Most common vulnerabilities; They involve manipulating the code that makes up an application or operating system to function in a way not intended by the developer; Often result in data exfiltration, system disruption or denial of access to information or further penetration on the victim infrastructure.
- ▶ **Personnel** – Personnel vulnerabilities, also known as insider threats; These are people who have regular or elevated access to an information due to their job responsibilities. They can be:
 - Careless/negligent employee
 - Careless/negligent third party
 - Agent of another organization
 - Disgruntled employee
 - Malicious insider
- ▶ **Physical** – Physical access must always be restricted to authorized personnel in order to minimize personnel vulnerabilities.
- ▶ Healthcare industry is effected by generic vulnerabilities as well as telehealth-related vulnerabilities...



Healthcare Enterprise Vulnerabilities

▶ Telehealth: Remote patient monitoring

- Wearable devices such as smart watches or wristbands to monitor and communicate vital signs; Devices such as a cane or walker, which can detect and alert when a patient has fallen; Stationary bedside or chairside devices which monitors patient vital signs
- Four components:
 - Sensors
 - Local data storage repository
 - Centralized repository
 - Diagnostic software/applications



Smart watches as telehealth devices to monitor vital signs, issue alerts and communicate to/from clinical care facility.
Image: Telecareware.com



A home patient monitor that measures blood pressure, pulse rate, blood oxygen saturation, weight, glucose level, prothrombin time, fluid status and temperature.
Image: 4mdmedical.com

▶ Remote patient monitoring security

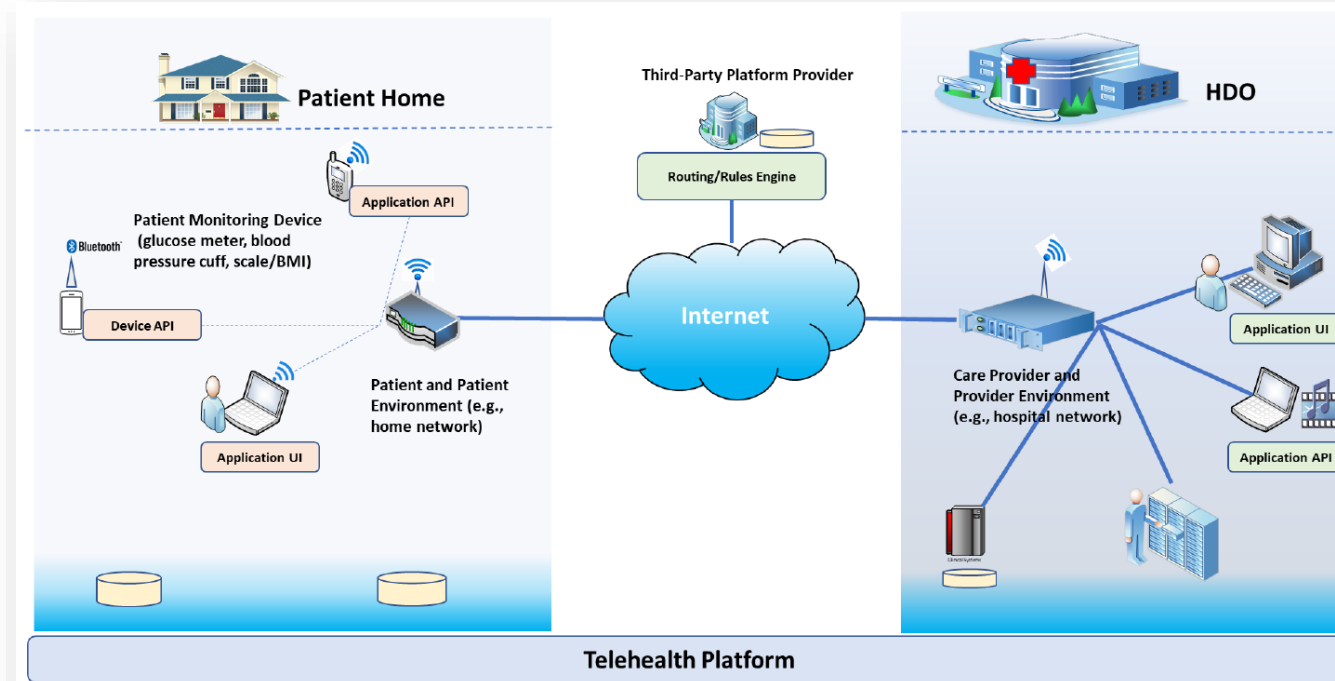
- Endpoint devices at patient's residence
 - Endpoints (home routers and systems)
 - Applications and devices
- Healthcare Delivery Organization (HDO)
 - Internal infrastructure
 - Routers, switches, hubs
 - Servers, applications, interfaces and insider threats



Healthcare Enterprise Vulnerabilities (continued)

▶ Telehealth: Store and forward

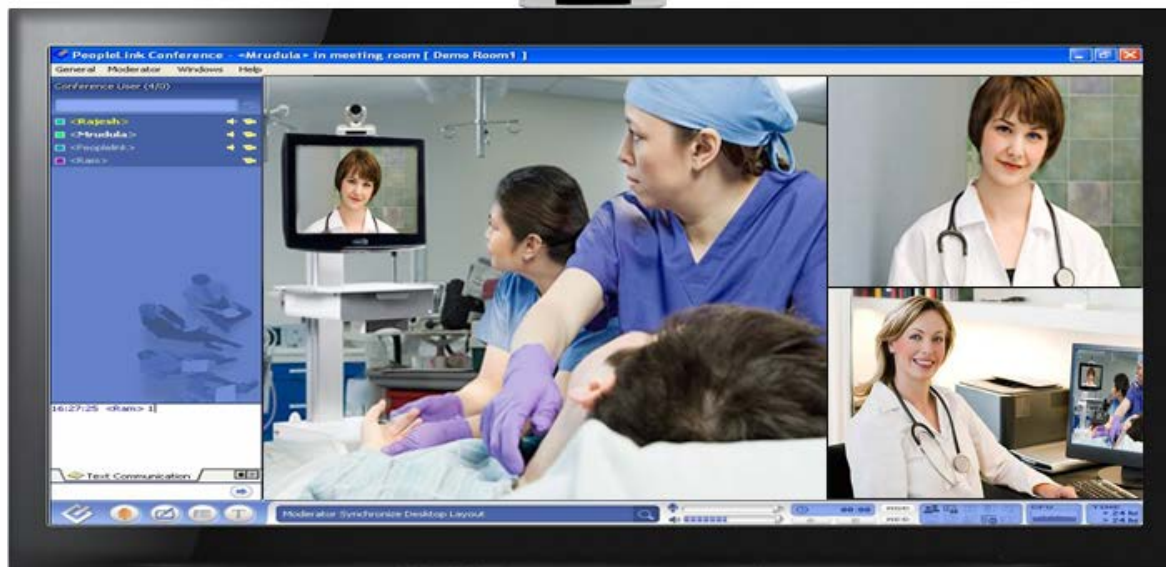
- Transmission of images/information between providers; Often used for further evaluation of data.
- Examples: X-rays and MRIs, digital photos, pre-recorded videos, other medical images
- Vulnerabilities associated with:
 - Internal device storage, directly attached storage, network attached storage, removable drives
 - Hardware, apps/programs, interfaces



Healthcare Enterprise Vulnerabilities (continued)

▶ Telehealth: Real-time audio/video

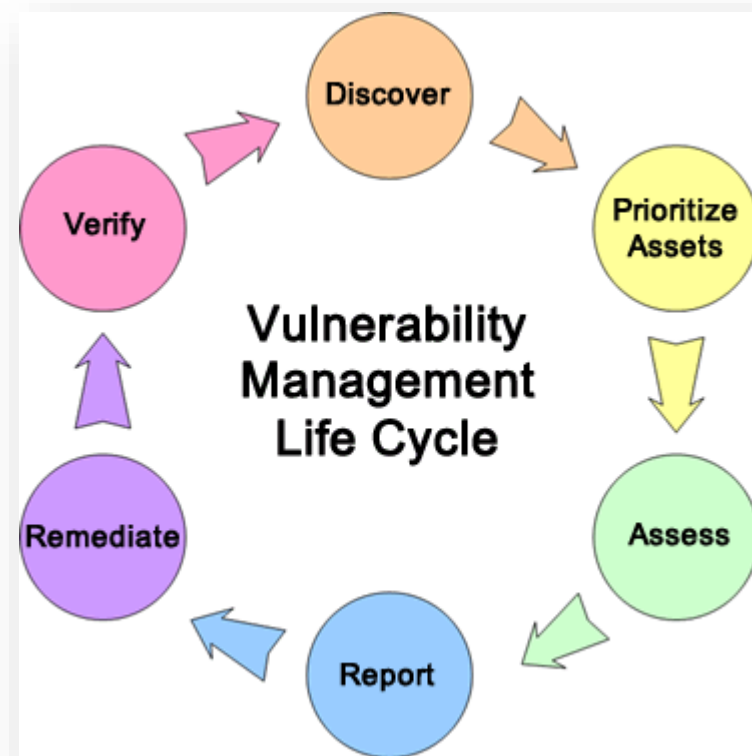
- Security of data in motion vs. data at rest
 - Encryption of links and repositories
 - Protocols (TCP, UDP, HTTP, DNS, SSL, TLS, RTP, IPsec, etc...)



Vulnerability management lifecycle

Per the Centers for Disease Control (CDC)

- ▶ **Discover:** Inventory all assets across the network and identify host details including operating system and open services to identify vulnerabilities. Develop a network baseline. Identify security vulnerabilities on a regular automated schedule.
- ▶ **Prioritize Assets:** Categorize assets into groups or business units, and assign a business value to asset groups based on their criticality to your business operation.
- ▶ **Assess:** Determine a baseline risk profile so you can eliminate risks based on asset criticality, vulnerability threat, and asset classification.
- ▶ **Report:** Measure the level of business risk associated with your assets according to your security policies. Document a security plan, monitor suspicious activity, and describe known vulnerabilities.
- ▶ **Remediate:** Prioritize and fix vulnerabilities in order according to business risk. Establish controls and demonstrate progress.
- ▶ **Verify:** Verify that threats have been eliminated through follow-up audits.



Source: <https://www.cdc.gov/cancer/npcr/tools/security/vmlc.htm>

Vulnerability Identification

Who identifies vulnerabilities?

- ▶ Vendors
- ▶ Independent researchers
- ▶ Bug bounties
- ▶ Adversaries/bad guys

How are vulnerabilities reported? Who are they reported to?

- ▶ Report to vendor
- ▶ Public disclosure (pressure vendor to act, possible public solution)
 - Timing
 - Who acts on it first? “Good guys” or “bad guys”?
- ▶ Sale on gray market
 - Researcher gets paid
 - Who acts on it first? “Good guys” or “bad guys”?
- ▶ Sale on black market
 - Researcher gets paid
 - Getting paid



Image source: Trend Micro



Vulnerability databases, etc...

- ▶ Common Vulnerabilities and Exposures (CVE)
 - Joint effort: DHS and MITRE
 - Repository to facilitate correlation and sharing of vulnerability data
 - <https://cve.mitre.org/>
- ▶ National Institute of Standards and Technology (NIST) National Vulnerability Database (NVD)
 - Government repository of standards-based vulnerability management data using the Security Content Automation Protocol (SCAP). This data enables automation of vulnerability management, security measurement, and compliance. The NVD includes databases of security checklist references, security-related software flaws, misconfigurations, product names, and impact metrics.
 - <https://nvd.nist.gov/>
- ▶ Carnegie Mellon University Computer Emergency Response Team (CERT) Coordination Center Vulnerability Notes Database
 - Includes “summaries, technical details, remediation information, and lists of affected vendors. Most vulnerability notes are the result of private coordination and disclosure efforts.”
 - NVD is recommended for further details on vulnerabilities
 - <https://www.kb.cert.org/vuls/>



Vulnerability databases, etc... (continued)

▶ Symantec/SecurityFocus BugTraq

- Stood up in 1999
- “high volume, full disclosure mailing list for the detailed discussion and announcement of computer security vulnerabilities” <https://www.securityfocus.com/bid/>

▶ Exploit-DB

- “CVE compliant archive of public exploits and corresponding vulnerable software, developed for use by penetration testers and vulnerability researchers.”
- “a repository for exploits and proof-of-concepts rather than advisories, making it a valuable resource for those who need actionable data right away.”
- Goal: “serve the most comprehensive collection of exploits gathered through direct submissions, mailing lists, as well as other public sources, and present them in a freely-available and easy-to-navigate database.”
- Includes Google Hacking database
- <https://www.exploit-db.com/>

▶ Seclist Full Disclosure

- <https://seclists.org/fulldisclosure/>

▶ Microsoft Security Update Guide


- <https://portal.msrc.microsoft.com/en-us/security-guidance>






Common Vulnerabilities and Exposures

Sponsored by CISA/DHS; Managed and maintained by MITRE



[CVE List](#) [CNAs](#) [WGs](#) [Board](#) [About](#) [News & Blog](#)



Go to for:
[CVSS Scores](#)
[CPE Info](#)
[Advanced Search](#)

[Search CVE List](#) [Download CVE](#) [Data Feeds](#) [Request CVE IDs](#) [Update a CVE Entry](#)

TOTAL CVE Entries: 117506

HOME > CVE > CVE-2012-0158 [Printer-Friendly View](#)

CVE-ID

CVE-2012-0158 [Learn more at National Vulnerability Database \(NVD\)](#)
[CVSS Severity Rating](#) • [Fix Information](#) • [Vulnerable Software Versions](#) • [SCAP Mappings](#) • [CPE Information](#)

Description

The (1) ListView, (2) ListView2, (3) TreeView, and (4) TreeView2 ActiveX controls in MSCOMCTL.COX in the Common Controls in Microsoft Office 2003 SP3, 2007 SP2 and SP3, and 2010 Gold and SP1; Office 2003 Web Components SP3; SQL Server 2000 SP4, 2005 SP4, and 2008 SP2, SP3, and R2; BizTalk Server 2002 SP1; Commerce Server 2002 SP4, 2007 SP2, and 2009 Gold and R2; Visual FoxPro 8.0 SP1 and 9.0 SP2; and Visual Basic 6.0 Runtime allow remote attackers to execute arbitrary code via a crafted (a) web site, (b) Office document, or (c) .rtf file that triggers "system state" corruption, as exploited in the wild in April 2012, aka "MSCOMCTL.COX RCE Vulnerability."

References

Note: [References](#) are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

- BID:52911
- [URL:http://www.securityfocus.com/bid/52911](http://www.securityfocus.com/bid/52911)
- CERT:TA12-101A
- [URL:http://www.us-cert.gov/cas/techalerts/TA12-101A.html](http://www.us-cert.gov/cas/techalerts/TA12-101A.html)
- [MISC:http://opensources.info/comment-on-the-curious-case-of-a-cve-2012-0158-exploit-by-chris-pierce/](http://opensources.info/comment-on-the-curious-case-of-a-cve-2012-0158-exploit-by-chris-pierce/)
- MS:MS12-027
- [URL:https://docs.microsoft.com/en-us/security-updates/securitybulletins/2012/ms12-027](https://docs.microsoft.com/en-us/security-updates/securitybulletins/2012/ms12-027)
- OVAL:oval.org.mitre.oval:def:15462
- [URL:https://oval.cisecurity.org/repository/search/definition/oval%3Aorg.mitre.oval%3Adef%3A15462](https://oval.cisecurity.org/repository/search/definition/oval%3Aorg.mitre.oval%3Adef%3A15462)
- SECTrack:1026899
- [URL:http://www.securitytracker.com/id?1026899](http://www.securitytracker.com/id?1026899)
- SECTrack:1026900
- [URL:http://www.securitytracker.com/id?1026900](http://www.securitytracker.com/id?1026900)
- SECTrack:1026902
- [URL:http://www.securitytracker.com/id?1026902](http://www.securitytracker.com/id?1026902)
- SECTrack:1026903
- [URL:http://www.securitytracker.com/id?1026903](http://www.securitytracker.com/id?1026903)
- SECTrack:1026904
- [URL:http://www.securitytracker.com/id?1026904](http://www.securitytracker.com/id?1026904)



Common Vulnerabilities and Exposures (cont)



- BID:52911
- URL:<http://www.securityfocus.com/bid/52911>
- CERT:TA12-101A
- URL:<http://www.us-cert.gov/cas/techalerts/TA12-101A.html>
- MISC:<http://opensources.info/comment-on-the-curious-case-of-a-cve-2012-0158-exploit-by-chris-pierce/>
- MS:MS12-027
- URL:<https://docs.microsoft.com/en-us/security-updates/securitybulletins/2012/ms12-027>
- OVAL:oval.org.mitre.oval:def:15462
- URL:<https://oval.cisecurity.org/repository/search/definition/oval%3Aorg.mitre.oval%3Adef%3A15462>
- SECTRACK:1026899
- URL:<http://www.securitytracker.com/id?1026899>
- SECTRACK:1026900
- URL:<http://www.securitytracker.com/id?1026900>
- SECTRACK:1026902
- URL:<http://www.securitytracker.com/id?1026902>
- SECTRACK:1026903
- URL:<http://www.securitytracker.com/id?1026903>
- SECTRACK:1026904
- URL:<http://www.securitytracker.com/id?1026904>
- SECTRACK:1026905
- URL:<http://www.securitytracker.com/id?1026905>
- XF:ms-activex-control-code-execution(74372)
- URL:<https://exchange.xforce.ibmcloud.com/vulnerabilities/74372>

Assigning CNA

Microsoft Corporation

Date Entry Created

20111213 Disclaimer: The [entry creation date](#) may reflect when the CVE ID was allocated or reserved, and does not necessarily indicate when this vulnerability was discovered, shared with the affected vendor, publicly disclosed, or updated in CVE.

Phase (Legacy)

Assigned (20111213)

Notes (Legacy)

Comments (Legacy)

Proposed (Legacy)

N/A

This is an entry on the [CVE List](#), which provides common identifiers for publicly known cybersecurity vulnerabilities.

SEARCH CVE USING KEYWORDS:

You can also search by reference using the [CVE Reference Maps](#).

For More Information: cve@mitre.org





National Vulnerability Database

NVD

► Operated by NIST

CVE-2012-0158 Detail

MODIFIED

This vulnerability has been modified since it was last analyzed by the NVD. It is awaiting reanalysis which may result in further changes to the information provided.

Description

The (1) ListView, (2) ListView2, (3) TreeView, and (4) TreeView2 ActiveX controls in MSCOMCTL.OCX in the Common Controls in Microsoft Office 2003 SP3, 2007 SP2 and SP3, and 2010 Gold and SP1; Office 2003 Web Components SP3; SQL Server 2000 SP4, 2005 SP4, and 2008 SP2, SP3, and R2; BizTalk Server 2002 SP1; Commerce Server 2002 SP4, 2007 SP2, and 2009 Gold and R2; Visual FoxPro 8.0 SP1 and 9.0 SP2; and Visual Basic 6.0 Runtime allow remote attackers to execute arbitrary code via a crafted (a) web site, (b) Office document, or (c) .rtf file that triggers "system state" corruption, as exploited in the wild in April 2012, aka "MSCOMCTL.OCX RCE Vulnerability."

Source: MITRE

Description Last Modified: 04/10/2012

Impact

CVSS v2.0 Severity and Metrics:

Base Score: 9.3 HIGH

Vector: (AV:N/AC:M/Au:N/C:C/I:C/A:C) (V2 legend)

Impact Subscore: 10.0

Exploitability Subscore: 8.6

Access Vector (AV): Network

Access Complexity (AC): Medium

Authentication (AU): None

Confidentiality (C): Complete

Integrity (I): Complete

Availability (A): Complete

Additional Information:

Victim must voluntarily interact with attack mechanism

Allows unauthorized disclosure of information

Allows unauthorized modification

Allows disruption of service

QUICK INFO

CVE Dictionary Entry:

CVE-2012-0158

NVD Published Date:

04/10/2012

NVD Last Modified:

10/12/2018





National Vulnerability Database(cont)

References to Advisories, Solutions, and Tools

By selecting these links, you will be leaving NIST webspace. We have provided these links to other web sites because they may have information that would be of interest to you. No inferences should be drawn on account of other sites being referenced, or not, from this page. There may be other web sites that are more appropriate for your purpose. NIST does not necessarily endorse the views expressed, or concur with the facts presented on these sites. Further, NIST does not endorse any commercial products that may be mentioned on these sites. Please address comments about this page to nvd@nist.gov.

Hyperlink	Resource
http://opensources.info/comment-on-the-curious-case-of-a-cve-2012-0158-exploit-by-chris-pierce/	
http://www.securityfocus.com/bid/52911	
http://www.securitytracker.com/id?1026899	
http://www.securitytracker.com/id?1026900	
http://www.securitytracker.com/id?1026902	
http://www.securitytracker.com/id?1026903	
http://www.securitytracker.com/id?1026904	
http://www.securitytracker.com/id?1026905	
http://www.us-cert.gov/cas/techalerts/TA12-101A.html	US Government Resource
https://docs.microsoft.com/en-us/security-updates/securitybulletins/2012/ms12-027	
https://exchange.xforce.ibmcloud.com/vulnerabilities/74372	
https://oval.cisecurity.org/repository/search/definition/oval%3Aorg.mitre.oval%3Adef%3A15462	

Technical Details

Vulnerability Type (View All)

- Code Injection (CWE-94)

Known Affected Software Configurations [Switch to CPE 2.2](#)

Configuration 1 ([hide](#))

⌵ **cpe:2.3:a:microsoft:office:2003:sp3:*:*:*:***

[Show Matching CPE\(s\)](#) ▼

⌵ **cpe:2.3:a:microsoft:office:2007:sp2:*:*:*:***

[Show Matching CPE\(s\)](#) ▼

⌵ **cpe:2.3:a:microsoft:office:2007:sp3:*:*:*:***

[Show Matching CPE\(s\)](#) ▼





Common Weakness Enumeration

▶ CWE-94: Description and Extended Description, Relationships and Modes of Introduction



Home > CWE List > CWE- Individual Dictionary Definition (3.2)

ID Lookup:

Home | About | CWE List | Scoring | Community | News | Search

CWE-94: Improper Control of Generation of Code ('Code Injection')

Weakness ID: 94
Abstraction: Class
Structure: Simple

Status: Draft

Presentation Filter:

▼ Description

The software constructs all or part of a code segment using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the syntax or behavior of the intended code segment.

▼ Extended Description

When software allows a user's input to contain code syntax, it might be possible for an attacker to craft the code in such a way that it will alter the intended control flow of the software. Such an alteration could lead to arbitrary code execution.

Injection problems encompass a wide variety of issues -- all mitigated in very different ways. For this reason, the most effective way to discuss these weaknesses is to note the distinct features which classify them as injection weaknesses. The most important issue to note is that all injection problems share one thing in common -- i.e., they allow for the injection of control plane data into the user-controlled data plane. This means that the execution of the process may be altered by sending code in through legitimate data channels, using no other mechanism. While buffer overflows, and many other flaws, involve the use of some further issue to gain execution, injection problems need only for the data to be parsed. The most classic instantiations of this category of weakness are SQL injection and format string vulnerabilities.

▼ Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that the user may want to explore.

- ▶ **Relevant to the view "Research Concepts" (CWE-1000)**
- ▶ **Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)**
- ▶ **Relevant to the view "Architectural Concepts" (CWE-1008)**
- ▶ **Relevant to the view "Development Concepts" (CWE-699)**

▼ Modes Of Introduction

The different Modes of Introduction provide information about how and when this weakness may be introduced. The Phase identifies a point in the software life cycle at which introduction may occur, while the Note provides a typical scenario related to introduction during the given phase.

Phase	Note
Architecture and Design	
Implementation	REALIZATION: This weakness is caused during implementation of an architectural security tactic.





Common Weakness Enumeration (cont)

► CWE-94: Applicable Platforms, Common Consequences and Likelihood of Exploit

▼ Applicable Platforms

The listings below show possible areas for which the given weakness could appear. These may be for specific named Languages, Operating Systems, Architectures, Paradigms, Technologies, or a class of such platforms. The platform is listed along with how frequently the given weakness appears for that instance.

Languages

Class: Interpreted (*Sometimes Prevalent*)

▼ Common Consequences

The table below specifies different individual consequences associated with the weakness. The Scope identifies the application security area that is violated, while the Impact describes the negative technical impact that arises if an adversary succeeds in exploiting this weakness. The Likelihood provides information about how likely the specific consequence is expected to be seen relative to the other consequences in the list. For example, there may be high likelihood that a weakness will be exploited to achieve a certain impact, but a low likelihood that it will be exploited to achieve a different impact.

Scope	Impact	Likelihood
Access Control	Technical Impact: <i>Bypass Protection Mechanism</i> In some cases, injectable code controls authentication; this may lead to a remote vulnerability.	
Access Control	Technical Impact: <i>Gain Privileges or Assume Identity</i> Injected code can access resources that the attacker is directly prevented from accessing.	
Integrity Confidentiality Availability	Technical Impact: <i>Execute Unauthorized Code or Commands</i> Code injection attacks can lead to loss of data integrity in nearly all cases as the control-plane data injected is always incidental to data recall or writing. Additionally, code injection can often result in the execution of arbitrary code.	
Non-Repudiation	Technical Impact: <i>Hide Activities</i> Often the actions performed by injected control code are unlogged.	

▼ Likelihood Of Exploit

Medium





Common Weakness Enumeration (cont)

► CWE-94: Example 1

▼ Demonstrative Examples

Example 1

This example attempts to write user messages to a message file and allow users to view them.

Example Language: PHP

(bad code)

```
$MessageFile = "cwe-94/messages.out";
if ($_GET["action"] == "NewMessage") {
    $name = $_GET["name"];
    $message = $_GET["message"];
    $handle = fopen($MessageFile, "a+");
    fwrite($handle, "<b>$name</b> says '$message'<hr>\n");
    fclose($handle);
    echo "Message Saved!<p>\n";
}
else if ($_GET["action"] == "ViewMessages") {
    include($MessageFile);
}
```

While the programmer intends for the MessageFile to only include data, an attacker can provide a message such as:

(attack code)

```
name=h4x0r
message=%3C?php%20system(%22/bin/l%20-l%22);?%3E
```

which will decode to the following:

(attack code)

```
<?php system("/bin/l%20-l");?>
```

The programmer thought they were just including the contents of a regular data file, but PHP parsed it and executed the code. Now, this code is executed any time people view messages. Notice that XSS ([CWE-79](#)) is also possible in this situation.





National Vulnerability Database(cont)

► Example 2

Example 2

edit-config.pl: This CGI script is used to modify settings in a configuration file.

```

Example Language: Perl (bad code)
use CGI qw(:standard);

sub config_file_add_key {
    my ($fname, $key, $arg) = @_;

    # code to add a field/key to a file goes here
}

sub config_file_set_key {
    my ($fname, $key, $arg) = @_;

    # code to set key to a particular file goes here
}

sub config_file_delete_key {
    my ($fname, $key, $arg) = @_;

    # code to delete key from a particular file goes here
}

sub handleConfigAction {
    my ($fname, $action) = @_;
    my $key = param('key');
    my $val = param('val');

    # this is super-efficient code, especially if you have to invoke

    # any one of dozens of different functions!

    my $code = "config_file_{$action}_key(\$fname, \$key, \$val)";
    eval($code);
}

$configfile = "/home/cwe/config.txt";
print header;
if (defined(param('action'))) {
    handleConfigAction($configfile, param('action'));
}
else {
    print "No action specified!\n";
}

```

The script intends to take the 'action' parameter and invoke one of a variety of functions based on the value of that parameter - config_file_add_key(), config_file_set_key(), or config_file_delete_key(). It could set up a conditional to invoke each function separately, but eval() is a powerful way of doing the same thing in fewer lines of code, especially when a large number of functions or variables are involved. Unfortunately, in this case, the attacker can provide other values in the action parameter, such as: add_key(",",","); system("/bin/l\$"); This would produce the following string in handleConfigAction(): config_file_add_key(",",","); system("/bin/l\$"); Any arbitrary Perl code could be added after the attacker has "closed off" the construction of the original function call, in order to prevent parsing errors from causing the malicious eval() to fail before the attacker's payload is activated. This particular manipulation would fail after the system() call, because the "_key(\\$fname, \\$key, \\$val)" portion of the string would cause an error, but this is irrelevant to the attack because the payload has already been activated.





National Vulnerability Database(cont)

► CWE-94: Description and Extended Description, Relationships and Modes of Introduction

▼ Potential Mitigations

Phase: Architecture and Design

Refactor your program so that you do not have to dynamically generate code.

Phase: Architecture and Design

Run your code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which code can be executed by your software.

Examples include the Unix chroot jail and AppArmor. In general, managed code may provide some protection.

This may not be a feasible solution, and it only limits the impact to the operating system; the rest of your application may still be subject to compromise.

Be careful to avoid [CWE-243](#) and other weaknesses related to jails.

Phase: Implementation

Strategy: Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

To reduce the likelihood of code injection, use stringent whitelists that limit which constructs are allowed. If you are dynamically constructing code that invokes a function, then verifying that the input is alphanumeric might be insufficient. An attacker might still be able to reference a dangerous function that you did not intend to allow, such as `system()`, `exec()`, or `exit()`.

Phase: Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Phase: Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Phase: Operation

Strategy: Compilation or Build Hardening

Run the code in an environment that performs automatic taint propagation and prevents any command execution that uses tainted variables, such as Perl's "-T" switch. This will force the program to perform validation steps that remove the taint, although you must be careful to correctly validate your inputs so that you do not accidentally mark dangerous inputs as untainted (see [CWE-183](#) and [CWE-184](#)).

Phase: Operation

Strategy: Environment Hardening

Run the code in an environment that performs automatic taint propagation and prevents any command execution that uses tainted variables, such as Perl's "-T" switch. This will force the program to perform validation steps that remove the taint, although you must be careful to correctly validate your inputs so that you do not accidentally mark dangerous inputs as untainted (see [CWE-183](#) and [CWE-184](#)).

▼ Memberships

This MemberOf Relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name
MemberOf		635	Weaknesses Originally Used by NVD from 2008 to 2016
MemberOf		752	2009 Top 25 - Risky Resource Management
MemberOf		884	CWE Cross-section
MemberOf		991	SFP Secondary Cluster: Tainted Input to Environment

▼ Notes

Research Gap

Many of these weaknesses are under-studied and under-researched, and terminology is not sufficiently precise.



Best practices

▶ Best Practices for Software Updates

- Enable automatic software updates whenever possible. This will ensure that software updates are installed as quickly as possible.
- Do not use unsupported end-of-life (EOL) software
 - Be continuously monitoring an enterprise infrastructure for outdates, obsolete and unsupported software
- Always visit vendor sites directly rather than clicking on advertisements or email links
- Avoid software updates while using untrusted networks
- Testing is critical for patch deployment, especially for enterprise critical systems
- For organizations: Build and maintain a mature patch management capability which encompasses all of the above across the enterprise



References

- ▶ Security Tip (ST04-006): Understanding Patches and Software Updates, July 14, 2009 | Last revised: September 28, 2018, Department of Homeland Security (DHS) Cybersecurity and Infrastructure Security Agency (CISA), <https://www.us-cert.gov/ncas/tips/ST04-006>
- ▶ SP 800-40 Rev. 3 : Guide to Enterprise Patch Management Technologies, July 2013, National Institute of Standards and Technology, <https://csrc.nist.gov/publications/detail/sp/800-40/rev-3/final>
- ▶ Configuration & Vulnerability Management News and Updates, National Institute of Standards and Technology, <https://www.nist.gov/topics/configuration-vulnerability-management>
- ▶ Vulnerability Management definition, National Institute of Standards and Technology, <https://csrc.nist.gov/glossary/term/Vulnerability-Management>
- ▶ Palmer, Danny, These are the top ten security vulnerabilities most exploited by hackers, ZDNet, March 19, 2019, <https://www.zdnet.com/article/these-are-the-top-ten-security-vulnerabilities-most-exploited-by-hackers-to-conduct-cyber-attacks/>
- ▶ Vulnerability Management Life Cycle, Centers for Disease Control and Prevention (CDC) National Program of Cancer Registries (NPCR), <https://www.cdc.gov/cancer/npcr/tools/security/vmlc.htm>
- ▶ Seven Ways Cyber Risk Goes Unchecked: The State of Healthcare Vulnerability Management, 2018, RiskSense, https://risksense.com/wp-content/uploads/2018/12/Brochure_The-State-of-Healthcare-Vulnerability-Management.pdf
- ▶ Mimran, Dudu, Time to Re-think Vulnerabilities Disclosure, April 14, 2015, <https://www.dudumimran.com/2015/04/time-to-re-think-vulnerabilities-disclosure.html>



References

- ▶ Microsoft Security Advisories and Bulletins, <https://docs.microsoft.com/en-us/security-updates/>
- ▶ CVE Details, <https://www.cvedetails.com/>
- ▶ Janus, Marta, The curious case of a CVE-2012-0158 exploit, August 6, 2013, Kaspersky SecureList, <https://securelist.com/the-curious-case-of-a-cve-2012-0158-exploit/37158/>
- ▶ National Vulnerability Database - CVE-2012-0158 Detail, National Institute of Standards and Technology, <https://nvd.nist.gov/vuln/detail/CVE-2012-0158>
- ▶ CVE-2012-0158 , MITRE Common Vulnerabilities and Exposures, <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-0158>
- ▶ CVE-2012-0158: Anatomy of a prolific exploit, Sophos, <https://www.sophos.com/en-us/medialibrary/PDFs/technical%20papers/CVE-2012-0158-An-Anatomy-of-a-Prolific-Exploit.PDF>
- ▶ Ranger, Steve, Cybersecurity: One in three breaches are caused by unpatched vulnerabilities, June 4, 2019, ZDNet, <https://www.zdnet.com/article/cybersecurity-one-in-three-breaches-are-caused-by-unpatched-vulnerabilities/>
- ▶ Exploit Database, <https://www.exploit-db.com/>
- ▶ Carnegie Mellon University, Software Engineering Institute, CERT Coordination Center, Vulnerability Notes Database, <https://www.kb.cert.org/vuls/>
- ▶ The Carnegie Mellon University Computer Emergency Response Team (CERT) Coordination Center Vulnerability Notes Database, <https://www.cdc.gov/cancer/npcr/tools/security/vmlc.htm>
- ▶ Vulnerability Management Lifecycle, Centers for Disease Control (CDC) National Program of Cancer Registries (NPCR) , <https://www.cdc.gov/cancer/npcr/tools/security/vmlc.htm>



References

- ▶ SANS Reading Room – Threats/Vulnerabilities, <https://www.sans.org/reading-room/whitepapers/threats/paper/34180>
- ▶ Common Weakness Enumeration, <https://cwe.mitre.org/index.html>
- ▶ National Vulnerability Database, National Institute of Standards and Technology, <https://nvd.nist.gov/>
- ▶ Common Vulnerabilities and Enumerations, MITRE, <https://cve.mitre.org/>
- ▶ Configuration & Vulnerability Management, <https://www.nist.gov/topics/configuration-vulnerability-management>
- ▶ Help Net Security, How organizations are managing vulnerability risks, <https://www.helpnetsecurity.com/2019/06/04/managing-vulnerability-risks/>



Questions

Upcoming Briefs

- ▶ Chipset vulnerabilities
- ▶ Legacy Systems in the Healthcare Enterprise

Product Evaluations

Recipients of this and other Healthcare Sector Cybersecurity Coordination Center (HC3) Threat Intelligence products are highly encouraged to provide feedback to HC3@HHS.GOV.

Requests for Information

Need information on a specific cybersecurity topic? Send your request for information (RFI) to HC3@HHS.GOV or call us Monday-Friday, between 9am-5pm (EST), at **(202) 691-2110**.

