JAVA-B Scan Report

Project Name	JAVA-B
Scan Start	Friday, December 20, 2019 5:37:17 AM
Preset	Checkmarx Default
Scan Time	00h:02m:44s
Lines Of Code Scanned	21057
Files Scanned	179
Report Creation Time	Friday, December 20, 2019 5:46:10 AM
Online Results	https://checkmarx.aetnagsc.com/CxWebClient/ViewerMain.aspx?scanid=1281496&projectid=6162
Team	Coventry
Checkmarx Version	8.8.0.72 HF19
Scan Type	Full
Source Origin	LocalPath
Density	9/10000 (Vulnerabilities/LOC)
Visibility	Public

Filter Settings

Severity

Included: High, Medium, Low, Information

Excluded: None

Result State

Included: Confirmed, Not Exploitable, To Verify, Urgent, Proposed Not Exploitable

Excluded: None

Assigned to

Included: All

Categories

Included:

Uncategorized	All
Custom	All
PCI DSS v3.2	All
OWASP Top 10 2013	All
FISMA 2014	All
NIST SP 800-53	All
OWASP Top 10 2017	All
OWASP Mobile Top 10 2016	All

Excluded:

Uncategorized	None
Custom	None
PCI DSS v3.2	None
OWASP Top 10 2013	None
FISMA 2014	None

NIST SP 800-53	None
OWASP Top 10 2017	None
OWASP Mobile Top 10 2016	None

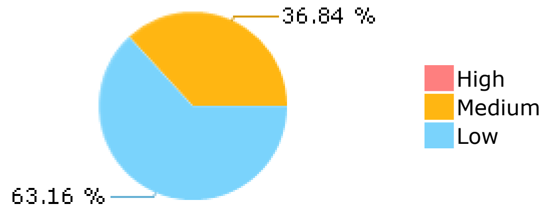
Results Limit

Results limit per query was set to 50

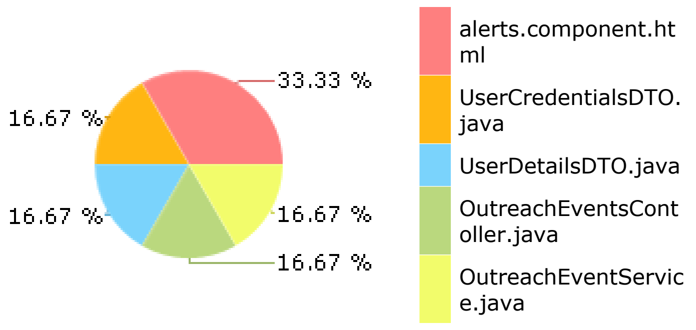
Selected Queries

Selected queries are listed in Result Summary

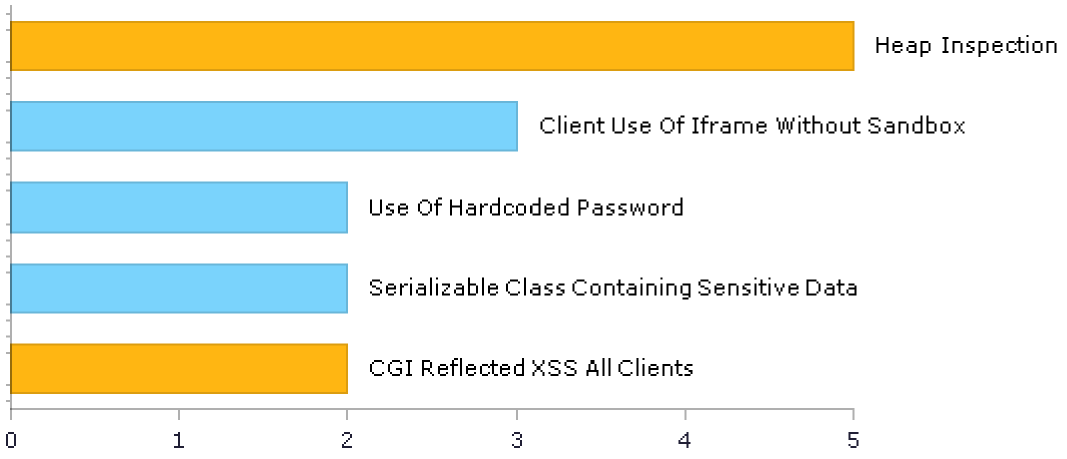
Result Summary



Most Vulnerable Files



Top 5 Vulnerabilities



Scan Summary - OWASP Top 10 2017

Further details and elaboration about vulnerabilities and risks can be found at: OWASP Top 10 2017

Category	Threat Agent	Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impact	Business Impact	Issues Found	Best Fix Locations
A1-Injection*	App. Specific	EASY	COMMON	EASY	SEVERE	App. Specific	0	0
A2-Broken Authentication*	App. Specific	EASY	COMMON	AVERAGE	SEVERE	App. Specific	2	2
A3-Sensitive Data Exposure*	App. Specific	AVERAGE	WIDESPREAD	AVERAGE	SEVERE	App. Specific	7	7
A4-XML External Entities (XXE)	App. Specific	AVERAGE	COMMON	EASY	SEVERE	App. Specific	0	0
A5-Broken Access Control*	App. Specific	AVERAGE	COMMON	AVERAGE	SEVERE	App. Specific	0	0
A6-Security Misconfiguration *	App. Specific	EASY	WIDESPREAD	EASY	MODERATE	App. Specific	4	4
A7-Cross-Site Scripting (XSS)*	App. Specific	EASY	WIDESPREAD	EASY	MODERATE	App. Specific	2	1
A8-Insecure Deserialization	App. Specific	DIFFICULT	COMMON	AVERAGE	SEVERE	App. Specific	0	0
A9-Using Components with Known Vulnerabilities*	App. Specific	AVERAGE	WIDESPREAD	AVERAGE	MODERATE	App. Specific	0	0
A10-Insufficient Logging & Monitoring	App. Specific	AVERAGE	WIDESPREAD	DIFFICULT	MODERATE	App. Specific	0	0

* Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

Scan Summary - OWASP Top 10 2013

Further details and elaboration about vulnerabilities and risks can be found at: OWASP Top 10 2013

Category	Threat Agent	Attack Vectors	Weakness Prevalence	Weakness Detectability	Technical Impact	Business Impact	Issues Found	Best Fix Locations
A1-Injection*	EXTERNAL, INTERNAL, ADMIN USERS	EASY	COMMON	AVERAGE	SEVERE	ALL DATA	0	0
A2-Broken Authentication and Session Management*	EXTERNAL, INTERNAL USERS	AVERAGE	WIDESPREAD	AVERAGE	SEVERE	AFFECTED DATA AND FUNCTIONS	2	2
A3-Cross-Site Scripting (XSS)*	EXTERNAL, INTERNAL, ADMIN USERS	AVERAGE	VERY WIDESPREAD	EASY	MODERATE	AFFECTED DATA AND SYSTEM	2	1
A4-Insecure Direct Object References*	SYSTEM USERS	EASY	COMMON	EASY	MODERATE	EXPOSED DATA	0	0
A5-Security Misconfiguration *	EXTERNAL, INTERNAL, ADMIN USERS	EASY	COMMON	EASY	MODERATE	ALL DATA AND SYSTEM	1	1
A6-Sensitive Data Exposure*	EXTERNAL, INTERNAL, ADMIN USERS, USERS BROWSERS	DIFFICULT	UNCOMMON	AVERAGE	SEVERE	EXPOSED DATA	7	7
A7-Missing Function Level Access Control*	EXTERNAL, INTERNAL USERS	EASY	COMMON	AVERAGE	MODERATE	EXPOSED DATA AND FUNCTIONS	0	0
A8-Cross-Site Request Forgery (CSRF)*	USERS BROWSERS	AVERAGE	COMMON	EASY	MODERATE	AFFECTED DATA AND FUNCTIONS	0	0
A9-Using Components with Known Vulnerabilities*	EXTERNAL USERS, AUTOMATED TOOLS	AVERAGE	WIDESPREAD	DIFFICULT	MODERATE	AFFECTED DATA AND FUNCTIONS	0	0
A10-Unvalidated Redirects and Forwards	USERS BROWSERS	AVERAGE	WIDESPREAD	DIFFICULT	MODERATE	AFFECTED DATA AND FUNCTIONS	0	0

* Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

Scan Summary - PCI DSS v3.2

Category	Issues Found	Best Fix Locations
PCI DSS (3.2) - 6.5.1 - Injection flaws - particularly SQL injection	0	0
PCI DSS (3.2) - 6.5.2 - Buffer overflows	0	0
PCI DSS (3.2) - 6.5.3 - Insecure cryptographic storage*****	0	0
PCI DSS (3.2) - 6.5.4 - Insecure communications*****	0	0
PCI DSS (3.2) - 6.5.5 - Improper error handling*****	2	2
PCI DSS (3.2) - 6.5.7 - Cross-site scripting (XSS)	2	1
PCI DSS (3.2) - 6.5.8 - Improper access control*****	0	0
PCI DSS (3.2) - 6.5.9 - Cross-site request forgery*****	0	0
PCI DSS (3.2) - 6.5.10 - Broken authentication and session management	2	2

***** Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

Scan Summary - FISMA 2014

Category	Description	Issues Found	Best Fix Locations
Access Control*	Organizations must limit information system access to authorized users, processes acting on behalf of authorized users, or devices (including other information systems) and to the types of transactions and functions that authorized users are permitted to exercise.	0	0
Audit And Accountability*	Organizations must: (i) create, protect, and retain information system audit records to the extent needed to enable the monitoring, analysis, investigation, and reporting of unlawful, unauthorized, or inappropriate information system activity; and (ii) ensure that the actions of individual information system users can be uniquely traced to those users so they can be held accountable for their actions.	0	0
Configuration Management*	Organizations must: (i) establish and maintain baseline configurations and inventories of organizational information systems (including hardware, software, firmware, and documentation) throughout the respective system development life cycles; and (ii) establish and enforce security configuration settings for information technology products employed in organizational information systems.	2	2
Identification And Authentication*	Organizations must identify information system users, processes acting on behalf of users, or devices and authenticate (or verify) the identities of those users, processes, or devices, as a prerequisite to allowing access to organizational information systems.	3	3
Media Protection	Organizations must: (i) protect information system media, both paper and digital; (ii) limit access to information on information system media to authorized users; and (iii) sanitize or destroy information system media before disposal or release for reuse.	5	5
System And Communications Protection	Organizations must: (i) monitor, control, and protect organizational communications (i.e., information transmitted or received by organizational information systems) at the external boundaries and key internal boundaries of the information systems; and (ii) employ architectural designs, software development techniques, and systems engineering principles that promote effective information security within organizational information systems.	0	0
System And Information Integrity*	Organizations must: (i) identify, report, and correct information and information system flaws in a timely manner; (ii) provide protection from malicious code at appropriate locations within organizational information systems; and (iii) monitor information system security alerts and advisories and take appropriate actions in response.	2	1

* Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

Scan Summary - NIST SP 800-53

Category	Issues Found	Best Fix Locations
AC-12 Session Termination (P2)	0	0
AC-3 Access Enforcement (P1)*	1	1
AC-4 Information Flow Enforcement (P1)	0	0
AC-6 Least Privilege (P1)	0	0
AU-9 Protection of Audit Information (P1)	0	0
CM-6 Configuration Settings (P2)	0	0
IA-5 Authenticator Management (P1)	0	0
IA-6 Authenticator Feedback (P2)	0	0
IA-8 Identification and Authentication (Non-Organizational Users) (P1)	0	0
SC-12 Cryptographic Key Establishment and Management (P1)	0	0
SC-13 Cryptographic Protection (P1)	0	0
SC-17 Public Key Infrastructure Certificates (P1)	0	0
SC-18 Mobile Code (P2)	3	3
SC-23 Session Authenticity (P1)*	0	0
SC-28 Protection of Information at Rest (P1)*	2	2
SC-4 Information in Shared Resources (P1)	5	5
SC-5 Denial of Service Protection (P1)*	1	1
SC-8 Transmission Confidentiality and Integrity (P1)	1	1
SI-10 Information Input Validation (P1)*	0	0
SI-11 Error Handling (P2)*	1	1
SI-15 Information Output Filtering (P0)*	2	1
SI-16 Memory Protection (P1)*	0	0

* Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

Scan Summary - OWASP Mobile Top 10 2016

Category	Description	Issues Found	Best Fix Locations
M1-Improper Platform Usage*	This category covers misuse of a platform feature or failure to use platform security controls. It might include Android intents, platform permissions, misuse of TouchID, the Keychain, or some other security control that is part of the mobile operating system. There are several ways that mobile apps can experience this risk.	0	0
M2-Insecure Data Storage*	This category covers insecure data storage and unintended data leakage.	0	0
M3-Insecure Communication*	This category covers poor handshaking, incorrect SSL versions, weak negotiation, cleartext communication of sensitive assets, etc.	0	0
M4-Insecure Authentication*	This category captures notions of authenticating the end user or bad session management. This can include: -Failing to identify the user at all when that should be required -Failure to maintain the user's identity when it is required -Weaknesses in session management	0	0
M5-Insufficient Cryptography*	The code applies cryptography to a sensitive information asset. However, the cryptography is insufficient in some way. Note that anything and everything related to TLS or SSL goes in M3. Also, if the app fails to use cryptography at all when it should, that probably belongs in M2. This category is for issues where cryptography was attempted, but it wasnt done correctly.	0	0
M6-Insecure Authorization*	This is a category to capture any failures in authorization (e.g., authorization decisions in the client side, forced browsing, etc.). It is distinct from authentication issues (e.g., device enrolment, user identification, etc.). If the app does not authenticate users at all in a situation where it should (e.g., granting anonymous access to some resource or service when authenticated and authorized access is required), then that is an authentication failure not an authorization failure.	0	0
M7-Client Code Quality*	This category is the catch-all for code-level implementation problems in the mobile client. That's distinct from server-side coding mistakes. This would capture things like buffer overflows, format string vulnerabilities, and various other code-level mistakes where the solution is to rewrite some code that's running on the mobile device.	0	0
M8-Code Tampering*	This category covers binary patching, local resource modification, method hooking, method swizzling, and dynamic memory modification. Once the application is delivered to the mobile device, the code and data resources are resident there. An attacker can either directly modify the code, change the contents of memory dynamically, change or replace the system APIs that the application uses, or	0	0

	modify the application's data and resources. This can provide the attacker a direct method of subverting the intended use of the software for personal or monetary gain.		
M9-Reverse Engineering*****	This category includes analysis of the final core binary to determine its source code, libraries, algorithms, and other assets. Software such as IDA Pro, Hopper, otool, and other binary inspection tools give the attacker insight into the inner workings of the application. This may be used to exploit other nascent vulnerabilities in the application, as well as revealing information about back end servers, cryptographic constants and ciphers, and intellectual property.	2	2
M10-Extraneous Functionality*****	Often, developers include hidden backdoor functionality or other internal development security controls that are not intended to be released into a production environment. For example, a developer may accidentally include a password as a comment in a hybrid app. Another example includes disabling of 2-factor authentication during testing.	0	0

***** Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

Scan Summary - Custom

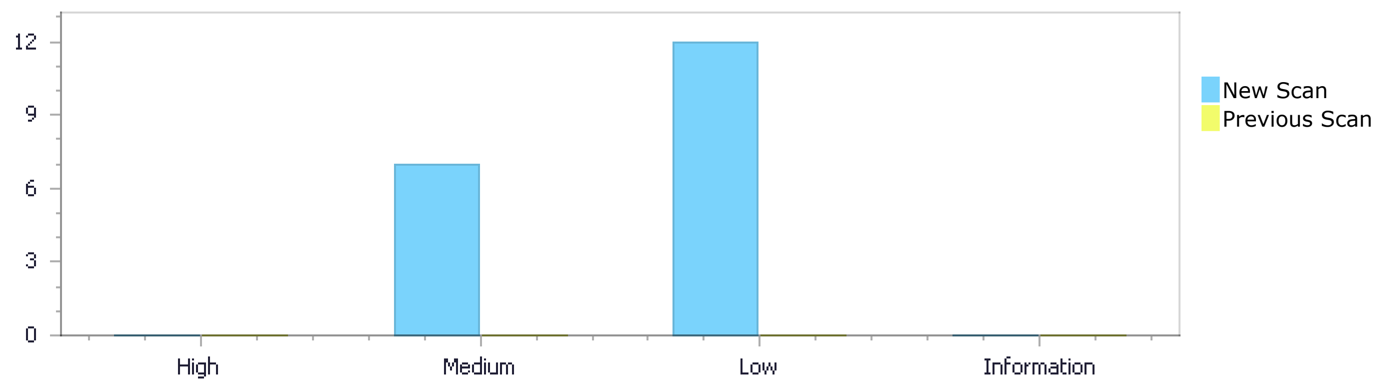
Category	Issues Found	Best Fix Locations
Must audit	0	0
Check	0	0
Optional	0	0

Results Distribution By Status

First scan of the project

	High	Medium	Low	Information	Total
New Issues	0	7	12	0	19
Recurrent Issues	0	0	0	0	0
Total	0	7	12	0	19

Fixed Issues	0	0	0	0	0



Results Distribution By State

	High	Medium	Low	Information	Total
Confirmed	0	0	0	0	0
Not Exploitable	0	0	0	0	0
To Verify	0	7	12	0	19
Urgent	0	0	0	0	0
Proposed Not Exploitable	0	0	0	0	0
Total	0	7	12	0	19

Result Summary

Vulnerability Type	Occurrences	Severity
Heap Inspection	5	Medium
CGI Reflected XSS All Clients	2	Medium
Client Use Of Iframe Without Sandbox	3	Low
Serializable Class Containing Sensitive Data	2	Low
Use Of Hardcoded Password	2	Low

Client Insufficient ClickJacking Protection	1	Low
Client Insufficient ClickJacking Protection	1	Low
Improper Exception Handling	1	Low
Improper Resource Access Authorization	1	Low
Information Exposure Through an Error Message	1	Low

10 Most Vulnerable Files

High and Medium Vulnerabilities

File Name	Issues Found
S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/controllers/OutreachEventsContoller.java	2
S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/services/OutreachEventService.java	2
S2AWSAssignments/Auth-Service/src/main/java/com/outreach/management/dto/UserDto.java	1
S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/dto/UserCredentialsDTO.java	1
S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/dto/UserDetailsDTO.java	1
S2AWSAssignments/Auth-Service/src/main/java/com/outreach/management/model/User.java	1
S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/collections/UserCredentialsCollections.java	1

Scan Results Details

Heap Inspection

Query Path:
Java\Cx\Java Medium Threat\Heap Inspection Version:2

Categories

OWASP Top 10 2013: A6-Sensitive Data Exposure
FISMA 2014: Media Protection
NIST SP 800-53: SC-4 Information in Shared Resources (P1)
OWASP Top 10 2017: A3-Sensitive Data Exposure

Description

Heap Inspection\Path 1:

Severity	Medium
Result State	To Verify
Online Results	https://checkmarx.aetnagsc.com/CxWebClient/ViewerMain.aspx?scanid=1281496&projectid=6162&pathid=5
Status	New

Method password; at line 11 of S2AWSAssignments/Auth-Service/src/main/java/com/outreach/management/dto/UserDto.java defines password, which is designated to contain user passwords. However, while plaintext passwords are later assigned to password, this variable is never cleared from memory.

	Source	Destination
File	S2AWSAssignments/Auth-Service/src/main/java/com/outreach/management/dto/UserDto.java	S2AWSAssignments/Auth-Service/src/main/java/com/outreach/management/dto/UserDto.java
Line	11	11
Object	password	password

Code Snippet

File Name	S2AWSAssignments/Auth-Service/src/main/java/com/outreach/management/dto/UserDto.java
Method	private String password;

```
....
11.      private String password;
```

Heap Inspection\Path 2:

Severity	Medium
Result State	To Verify
Online Results	https://checkmarx.aetnagsc.com/CxWebClient/ViewerMain.aspx?scanid=1281496&projectid=6162&pathid=6
Status	New

Method password; at line 17 of S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/dto/UserCredentialsDTO.java defines password, which is designated to contain user passwords. However, while plaintext passwords are later assigned to password, this variable is never cleared from memory.

	Source	Destination
File	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/dto/UserCredentialsDTO.java	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/dto/UserCredentialsDTO.java
Line	17	17
Object	password	password

Code Snippet

File Name	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/dto/UserCredentialsDTO.java
Method	private String password;

```
....
17.    private String password;
```

Heap Inspection\Path 3:

Severity	Medium
Result State	To Verify
Online Results	https://checkmarx.aetnagsc.com/CxWebClient/ViewerMain.aspx?scanid=1281496&projectid=6162&pathid=7
Status	New

Method password; at line 71 of S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/dto/UserDetailsDTO.java defines password, which is designated to contain user passwords. However, while plaintext passwords are later assigned to password, this variable is never cleared from memory.

	Source	Destination
File	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/dto/UserDetailsDTO.java	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/dto/UserDetailsDTO.java
Line	71	71
Object	password	password

Code Snippet

File Name	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/dto/UserDetailsDTO.java
Method	private String password;

```
....
71.    private String password;
```

Heap Inspection\Path 4:

Severity	Medium
Result State	To Verify
Online Results	https://checkmarx.aetnagsc.com/CxWebClient/ViewerMain.aspx?scanid=1281496&projectid=6162&pathid=8
Status	New

Method password; at line 31 of S2AWSAssignments/Auth-Service/src/main/java/com/outreach/management/model/User.java defines password, which is designated to contain user passwords. However, while plaintext passwords are later assigned to password, this variable is never cleared from memory.

	Source	Destination
File	S2AWSAssignments/Auth-Service/src/main/java/com/outreach/management/model/User.java	S2AWSAssignments/Auth-Service/src/main/java/com/outreach/management/model/User.java
Line	31	31
Object	password	password

Code Snippet

File Name	S2AWSAssignments/Auth-Service/src/main/java/com/outreach/management/model/User.java
Method	private String password;

```
....
31.      private String password;
```

Heap Inspection\Path 5:

Severity	Medium
Result State	To Verify
Online Results	https://checkmarx.aetnagsc.com/CxWebClient/ViewerMain.aspx?scanid=1281496&projectid=6162&pathid=9
Status	New

Method password; at line 14 of S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/collections/UserCredentialsCollections.java defines password, which is designated to contain user passwords. However, while plaintext passwords are later assigned to password, this variable is never cleared from memory.

	Source	Destination
File	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/collections/UserCredentialsCollections.java	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/collections/UserCredentialsCollections.java
Line	14	14
Object	password	password

Code Snippet

File Name	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/collections/UserCredentialsCollections.java
Method	public String password;

```
....
14.    public String password;
```

CGI Reflected XSS All Clients

Query Path:
Java\Cx\Java Medium Threat\CGI Reflected XSS All Clients Version:1

Categories

PCI DSS v3.2: PCI DSS (3.2) - 6.5.7 - Cross-site scripting (XSS)
OWASP Top 10 2013: A3-Cross-Site Scripting (XSS)
FISMA 2014: System And Information Integrity
NIST SP 800-53: SI-15 Information Output Filtering (P0)
OWASP Top 10 2017: A7-Cross-Site Scripting (XSS)

Description

CGI Reflected XSS All Clients\Path 1:

Severity	Medium
Result State	To Verify
Online Results	https://checkmarx.aetnagsc.com/CxWebClient/ViewerMain.aspx?scanid=1281496&projectid=6162&pathid=14
Status	New

Unvalidated input was found in line number 44 in S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/controllers/OutreachEventsContoller.java file. A possible XSS exploitation was found in println at line number 37.

	Source	Destination
File	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/controllers/OutreachEventsContoller.java	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/services/OutreachEventService.java
Line	44	38
Object	eventId	println

Code Snippet

File Name	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/controllers/OutreachEventsContoller.java
Method	public ResponseEntity<List<EventFeedBackDetailsCollections>> getEventFeedBackDetailsCollection(@PathVariable("eventId") String eventId) {

```
....
44.   public ResponseEntity<List<EventFeedBackDetailsCollections>>
getEventFeedBackDetailsCollection(@PathVariable("eventId") String
eventId) {
```

File Name	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/services/OutreachEventService.java
Method	public List<EventFeedBackDetailsCollections> getEventFeedBackDetailsCollection(String eventId) {

```
....
38.         System.out.println(eventId);
```

CGI Reflected XSS All Clients\Path 2:

Severity	Medium
Result State	To Verify

Online Results	https://checkmarx.aetnagsc.com/CxWebClient/ViewerMain.aspx?scanid=1281496&projectid=6162&pathid=15
Status	New

Unvalidated input was found in line number 44 in S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/controllers/OutreachEventsContoller.java file. A possible XSS exploitation was found in println at line number 37.

	Source	Destination
File	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/controllers/OutreachEventsContoller.java	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/services/OutreachEventService.java
Line	44	40
Object	eventId	println

Code Snippet

File Name	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/controllers/OutreachEventsContoller.java
Method	public ResponseEntity<List<EventFeedBackDetailsCollections>> getEventFeedBackDetailsCollection(@PathVariable("eventId") String eventId) {

```
....
44.   public ResponseEntity<List<EventFeedBackDetailsCollections>>
getEventFeedBackDetailsCollection(@PathVariable("eventId") String
eventId) {
```

File Name	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/services/OutreachEventService.java
Method	public List<EventFeedBackDetailsCollections> getEventFeedBackDetailsCollection(String eventId) {

```
....
40.         System.out.println(r);
```

Client Use Of Iframe Without Sandbox

Query Path:
JavaScript\Cx\JavaScript Low Visibility\Client Use Of Iframe Without Sandbox Version:1

Categories

NIST SP 800-53: SC-18 Mobile Code (P2)
OWASP Top 10 2017: A6-Security Misconfiguration

Description

Client Use Of Iframe Without Sandbox\Path 1:

Severity	Low
Result State	To Verify
Online Results	https://checkmarx.aetnagsc.com/CxWebClient/ViewerMain.aspx?scanid=1281496&projectid=6162&pathid=2
Status	New

The application loads an external library or source code file using iframe___187721294, at line 1 of S2AWSAssignments/OutReachmanagement/src/app/features/alerts/alerts.component.html. An attacker might be able to exploit this and cause the application to load arbitrary code.

	Source	Destination
File	S2AWSAssignments/OutReachmanagement/src/app/features/alerts/alerts.component.html	S2AWSAssignments/OutReachmanagement/src/app/features/alerts/alerts.component.html
Line	1	1
Object	iframe___187721294	iframe___187721294

Code Snippet

File Name	S2AWSAssignments/OutReachmanagement/src/app/features/alerts/alerts.component.html
Method	<iframe src="feedback-submit-success-alert.html"></iframe>

```
....
1.   <iframe src="feedback-submit-success-alert.html"></iframe>
```

Client Use Of Iframe Without Sandbox\Path 2:

Severity	Low
Result State	To Verify
Online Results	https://checkmarx.aetnagsc.com/CxWebClient/ViewerMain.aspx?scanid=1281496&projectid=6162&pathid=3
Status	New

The application loads an external library or source code file using iframe___1677272495, at line 2 of S2AWSAssignments/OutReachmanagement/src/app/features/alerts/alerts.component.html. An attacker might be able to exploit this and cause the application to load arbitrary code.

	Source	Destination
File	S2AWSAssignments/OutReachmanagement/src/app/features/alerts/alerts.component.html	S2AWSAssignments/OutReachmanagement/src/app/features/alerts/alerts.component.html
Line	2	2
Object	iframe___1677272495	iframe___1677272495

Code Snippet

File Name	S2AWSAssignments/OutReachmanagement/src/app/features/alerts/alerts.component.html
Method	<iframe src="feedback-unregistered-alert.html"></iframe>

```
....
2.   <iframe src="feedback-unregistered-alert.html"></iframe>
```

Client Use Of Iframe Without Sandbox\Path 3:

Severity	Low
Result State	To Verify

Online Results	https://checkmarx.aetnagsc.com/CxWebClient/ViewerMain.aspx?scanid=1281496&projectid=6162&pathid=4
Status	New

The application loads an external library or source code file using iframe___2110766868, at line 3 of S2AWSAssignments/OutReachmanagement/src/app/features/alerts/alerts.component.html. An attacker might be able to exploit this and cause the application to load arbitrary code.

	Source	Destination
File	S2AWSAssignments/OutReachmanagement/src/app/features/alerts/alerts.component.html	S2AWSAssignments/OutReachmanagement/src/app/features/alerts/alerts.component.html
Line	3	3
Object	iframe___2110766868	iframe___2110766868

Code Snippet

File Name	S2AWSAssignments/OutReachmanagement/src/app/features/alerts/alerts.component.html
Method	

```
....
3.
```

Serializable Class Containing Sensitive Data

Query Path:
Java\Cx\Java Low Visibility\Serializable Class Containing Sensitive Data Version:1

Categories

OWASP Top 10 2013: A6-Sensitive Data Exposure
OWASP Top 10 2017: A3-Sensitive Data Exposure

Description
Serializable Class Containing Sensitive Data\Path 1:

Severity	Low
Result State	To Verify
Online Results	https://checkmarx.aetnagsc.com/CxWebClient/ViewerMain.aspx?scanid=1281496&projectid=6162&pathid=16
Status	New

	Source	Destination
File	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/dto/UserCredentialsDTO.java	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/dto/UserCredentialsDTO.java
Line	17	8
Object	password	UserCredentialsDTO

Code Snippet

File Name	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/dto/UserCredentialsDTO.java
Method	private String password;

```
....
17.    private String password;
```

File Name	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/dto/UserCredentialsDTO.java
Method	@Data

```
....
8.   @Data
```

Serializable Class Containing Sensitive Data\Path 2:

Severity	Low
Result State	To Verify
Online Results	https://checkmarx.aetnagsc.com/CxWebClient/ViewerMain.aspx?scanid=1281496&projectid=6162&pathid=17
Status	New

	Source	Destination
File	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/dto/UserDetailsDTO.java	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/dto/UserDetailsDTO.java
Line	71	8
Object	password	UserDetailsDTO

Code Snippet

File Name	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/dto/UserDetailsDTO.java
Method	private String password;

```
....
71.    private String password;
```

File Name	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/dto/UserDetailsDTO.java
Method	@Data

```
....
8.   @Data
```

Use Of Hardcoded Password

Query Path:
Java\Cx\Java Low Visibility\Use Of Hardcoded Password Version:3

Categories

PCI DSS v3.2: PCI DSS (3.2) - 6.5.10 - Broken authentication and session management
OWASP Top 10 2013: A2-Broken Authentication and Session Management

FISMA 2014: Identification And Authentication
NIST SP 800-53: SC-28 Protection of Information at Rest (P1)
OWASP Top 10 2017: A2-Broken Authentication
OWASP Mobile Top 10 2016: M9-Reverse Engineering

Description
Use Of Hardcoded Password\Path 1:

Severity	Low
Result State	To Verify
Online Results	https://checkmarx.aetnagsc.com/CxWebClient/ViewerMain.aspx?scanid=1281496&projectid=6162&pathid=18
Status	New

The application uses a single, hard-coded password GRANT_TYPE_PASSWORD for authentication purposes, either using it to verify users' identities, or to access another remote system. This password at line 21 of S2AWSAssignments/Auth-Service/src/main/java/com/outreach/management/config/AuthorizationServerConfig.java appears in the code as plaintext, and cannot be changed without rebuilding the application.

	Source	Destination
File	S2AWSAssignments/Auth-Service/src/main/java/com/outreach/management/config/AuthorizationServerConfig.java	S2AWSAssignments/Auth-Service/src/main/java/com/outreach/management/config/AuthorizationServerConfig.java
Line	21	21
Object	GRANT_TYPE_PASSWORD	GRANT_TYPE_PASSWORD

Code Snippet

File Name	S2AWSAssignments/Auth-Service/src/main/java/com/outreach/management/config/AuthorizationServerConfig.java
Method	private static final String GRANT_TYPE_PASSWORD = "password";

```
....
21.    private static final String GRANT_TYPE_PASSWORD = "password";
```

Use Of Hardcoded Password\Path 2:

Severity	Low
Result State	To Verify
Online Results	https://checkmarx.aetnagsc.com/CxWebClient/ViewerMain.aspx?scanid=1281496&projectid=6162&pathid=19
Status	New

The application uses a single, hard-coded password plainClientCredentials for authentication purposes, either using it to verify users' identities, or to access another remote system. This password at line 34 of S2AWSAssignments/Auth-Service/src/test/java/com/devglan/rolebasedoauth2/RoleBasedOauth2ApplicationTests.java appears in the code as plaintext, and cannot be changed without rebuilding the application.

	Source	Destination
File	S2AWSAssignments/Auth-Service/src/test/java/com/devglan/rolebasedoauth2/RoleBasedOauth2Application	S2AWSAssignments/Auth-Service/src/test/java/com/devglan/rolebasedoauth2/RoleBasedOauth2Application

	Tests.java	Tests.java
Line	35	35
Object	plainClientCredentials	plainClientCredentials

Code Snippet

File Name	S2AWSAssignments/Auth-Service/src/test/java/com/devglan/rolebasedoauth2/RoleBasedOauth2ApplicationTests.java
Method	private static HttpHeaders getHeadersWithClientCredentials(){

```
....
35.         String plainClientCredentials="my-trusted-client:secret";
```

Client Insufficient ClickJacking Protection

Query Path:
JavaScript\Cx\JavaScript Low Visibility\Client Insufficient ClickJacking Protection Version:3

Categories

FISMA 2014: Configuration Management
NIST SP 800-53: SC-8 Transmission Confidentiality and Integrity (P1)

Description

Client Insufficient ClickJacking Protection\Path 1:

Severity	Low
Result State	To Verify
Online Results	https://checkmarx.aetnagsc.com/CxWebClient/ViewerMain.aspx?scanid=1281496&projectid=6162&pathid=1
Status	New

The application does not protect the web page S2AWSAssignments/OutReachmanagement/src/app/app.component.html from clickjacking attacks in legacy browsers, by using framebusting scripts.

	Source	Destination
File	S2AWSAssignments/OutReachmanagement/src/app/app.component.html	S2AWSAssignments/OutReachmanagement/src/app/app.component.html
Line	1	1
Object	CxJSNS_1531822548	CxJSNS_1531822548

Code Snippet

File Name	S2AWSAssignments/OutReachmanagement/src/app/app.component.html
Method	<div *ngIf="showNavBar === true">

```
....
1.  <div *ngIf="showNavBar === true">
```

Improper Resource Access Authorization

Query Path:
Java\Cx\Java Low Visibility\Improper Resource Access Authorization Version:3

Categories

FISMA 2014: Identification And Authentication
NIST SP 800-53: AC-3 Access Enforcement (P1)

[Description](#)

Improper Resource Access Authorization\Path 1:

Severity	Low
Result State	To Verify
Online Results	https://checkmarx.aetnagsc.com/CxWebClient/ViewerMain.aspx?scanid=1281496&projectid=6162&pathid=10
Status	New

	Source	Destination
File	S2AWSAssignments/Auth-Service/src/main/java/com/outreach/management/dao/RoleDao.java	S2AWSAssignments/Auth-Service/src/main/java/com/outreach/management/dao/RoleDao.java
Line	15	15
Object	roles	roles

Code Snippet

File Name	S2AWSAssignments/Auth-Service/src/main/java/com/outreach/management/dao/RoleDao.java
Method	Set<Role> find(@Param("roles") List<String> roles);

```
....
15.      Set<Role> find(@Param("roles") List<String> roles);
```

Client Insufficient ClickJacking Protection

Query Path:
Typescript\Cx\Typescript Low Visibility\Client Insufficient ClickJacking Protection Version:1

[Description](#)

Client Insufficient ClickJacking Protection\Path 1:

Severity	Low
Result State	To Verify
Online Results	https://checkmarx.aetnagsc.com/CxWebClient/ViewerMain.aspx?scanid=1281496&projectid=6162&pathid=11
Status	New

The application does not protect the web page S2AWSAssignments/OutReachmanagement/src/app/features/alerts/alerts.component.html from clickjacking attacks in legacy browsers, by using framebusting scripts.

	Source	Destination
File	S2AWSAssignments/OutReachmanagement/src/app/features/alerts/alerts.component.html	S2AWSAssignments/OutReachmanagement/src/app/features/alerts/alerts.component.html
Line	1	1
Object	CxJSNS_45628E27	CxJSNS_45628E27

Code Snippet

File Name	S2AWSAssignments/OutReachmanagement/src/app/features/alerts/alerts.component.html

Method	<iframe src="feedback-submit-success-alert.html"></iframe>

```
....
1.   <iframe src="feedback-submit-success-alert.html"></iframe>
```

Improper Exception Handling

Query Path:
Java\Cx\Java Low Visibility\Improper Exception Handling Version:0

Categories

PCI DSS v3.2: PCI DSS (3.2) - 6.5.5 - Improper error handling
NIST SP 800-53: SC-5 Denial of Service Protection (P1)

Description

Improper Exception Handling\Path 1:

Severity	Low
Result State	To Verify
Online Results	https://checkmarx.aetnagsc.com/CxWebClient/ViewerMain.aspx?scanid=1281496&projectid=6162&pathid=12
Status	New

Method find at line 15 of S2AWSAssignments/Auth-Service/src/main/java/com/outreach/management/dao/RoleDao.java performs an operation that could be expected to throw an exception, and is not properly wrapped in a try-catch block. This constitutes Improper Exception Handling.

	Source	Destination
File	S2AWSAssignments/Auth-Service/src/main/java/com/outreach/management/dao/RoleDao.java	S2AWSAssignments/Auth-Service/src/main/java/com/outreach/management/dao/RoleDao.java
Line	15	15
Object	roles	roles

Code Snippet

File Name	S2AWSAssignments/Auth-Service/src/main/java/com/outreach/management/dao/RoleDao.java
Method	Set<Role> find(@Param("roles") List<String> roles);

```
....
15.        Set<Role> find(@Param("roles") List<String> roles);
```

Information Exposure Through an Error Message

Query Path:
Java\Cx\Java Low Visibility\Information Exposure Through an Error Message Version:2

Categories

PCI DSS v3.2: PCI DSS (3.2) - 6.5.5 - Improper error handling
OWASP Top 10 2013: A5-Security Misconfiguration
FISMA 2014: Configuration Management
NIST SP 800-53: SI-11 Error Handling (P2)
OWASP Top 10 2017: A6-Security Misconfiguration

Description

Information Exposure Through an Error Message\Path 1:

Severity	Low
Result State	To Verify
Online Results	https://checkmarx.aetnagsc.com/CxWebClient/ViewerMain.aspx?scanid=1281496&projectid=6162&pathid=13
Status	New

Method createUser, at line 55 of S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/services/UserAuthenticationService.java, handles an Exception or runtime Error e. During the exception handling code, the application exposes the exception details to printStackTrace, in method createUser of S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/services/UserAuthenticationService.java, line 55.

	Source	Destination
File	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/services/UserAuthenticationService.java	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/services/UserAuthenticationService.java
Line	62	64
Object	e	printStackTrace

Code Snippet

File Name	S2AWSAssignments/outreachmanagementapi/src/main/java/com/outreach/management/services/UserAuthenticationService.java
Method	public UserDetailsCollections createUser(UserDetailsDTO userDetailsDTO) {

```
....
62.          } catch (Exception e) {
....
64.                e.printStackTrace();
```

Heap Inspection

Risk

What might happen

All variables stored by the application in unencrypted memory can potentially be retrieved by an unauthorized user, with privileged access to the machine. For example, a privileged attacker could attach a debugger to the running process, or retrieve the process's memory from the swapfile or crash dump file. Once the attacker finds the user passwords in memory, these can be reused to easily impersonate the user to the system.

Cause

How does it happen

String variables are immutable - in other words, once a string variable is assigned, its value cannot be changed or removed. Thus, these strings may remain around in memory, possibly in multiple locations, for an indefinite period of time until the garbage collector happens to remove it. Sensitive data, such as passwords, will remain exposed in memory as plaintext with no control over their lifetime.

While it may still be possible to retrieve data from memory, even if it uses a mutable container that is cleared, or retrieve a decryption key and decrypt sensitive data from memory - layering sensitive data with these types of protection would significantly increase the required effort to do so. By setting a high bar for retrieving sensitive data from memory, and reducing the amount and exposure of sensitive data in memory, an adversary is significantly less likely to succeed in obtaining valuable data.

General Recommendations

How to avoid it

When it comes to avoiding Heap Inspection, it is important to note that, given any read access to memory or a memory dump of an application, it is always likely to disclose some sensitive data to an adversary - these suggestions are part of defense-in-depth principles for protection of sensitive data in cases where such memory read access is successfully obtained. These recommendations will enable significant reduction in the lifespan and exposure of sensitive data in memory; however - given enough time, effort and unlimited access to memory, they will only go so far in protecting sensitive data being used by the application. The only way to handle Heap Inspection issues is to minimize and reduce data exposure, and obscure it in memory wherever possible.

- Do not store sensitive data, such as passwords or encryption keys, in memory in plain-text, even for a short period of time.
- Prefer to use specialized classes that store encrypted data in memory to ensure it cannot be trivially retrieved from memory.
- When required to use sensitive data in its raw form, temporarily store it in mutable data types, such as byte arrays, to reduce readability from memory, and then promptly zeroize the memory locations, to reduce exposure duration of this data while in memory.
- Ensure that memory dumps are not exchanged with untrusted parties, as even by ensuring all of the above - it may still be possible to reverse-engineer encrypted containers, or retrieve bytes of sensitive data from memory and rebuild it.

- In Java, do not store passwords in immutable strings - prefer using an encrypted memory object, such as SealedObject.

Source Code Examples

Java

Plaintext Password in Immutable String

```
class Heap_Inspection
{
  private String password;

  public void setPassword(String password)
 {
      this.password = password;
  }
}
```

Password Protected in Memory

```
class Heap_Inspection_Fixed
{
  private SealedObject password;

  public void setPassword(Character[] input)
 {
      Key key = getKeyFromConfiguration();
             Cipher c = Cipher.getInstance(CIPHER_NAME);
             c.init(Cipher.ENCRYPT_MODE, key);
             List<Character> characterList = Arrays.asList(input);
             password = new SealedObject((Serializable) characterList, c);
             Arrays.fill(input, '\0'); // Zero out input. Will also overwrite the values in
characterList by reference.
  }
}
```

CGI Reflected XSS All Clients

Risk

What might happen

Special crafted HTTP requests might retrieve system information and exploit the system through CGI (Common Gateway Interface).

Cause

How does it happen

The CGI specification provides opportunities to read files, acquire shell access, and corrupt file systems on server machines and their attached hosts.

Means of gaining access include: exploiting assumptions of the script, exploiting weaknesses in the server environment, and exploiting weaknesses in other programs and system calls.

The primary weakness in CGI scripts is insufficient input validation.

General Recommendations

How to avoid it

Refrain from binding user input to system variables if not necessary.
Validate and encode all user input.

Source Code Examples

Perl

Bad - The script blindly prints out the submitted comment

```perl
#!/usr/bin/perl
use CGI;

my $cgi = CGI->new();
my $text = $cgi->param('comment');

print $cgi->header();
print "You entered $text";
```

Good - The comment is being encoded before printed out

```perl
#!/usr/bin/perl
use CGI;
use HTML::Entities;

my $cgi = CGI->new();
my $text = $cgi->param('comment');

print $cgi->header();
print "You entered ", HTML::Entities::encode($text);
```

Client Insufficient ClickJacking Protection

Risk

What might happen

Clickjacking attacks allow an attacker to "hijack" a user's mouse clicks on a webpage, by invisibly framing the application, and superimposing it in front of a bogus site. When the user is convinced to click on the bogus website, e.g. on a link or a button, the user's mouse is actually clicking on the target webpage, despite being invisible.
This could allow the attacker to craft an overlay that, when clicked, would lead the user to perform undesirable actions in the vulnerable application, e.g. enabling the user's webcam, deleting all the user's records, changing the user's settings, or causing clickfraud.

Cause

How does it happen

The root cause of vulnerability to a clickjacking attack, is that the application's web pages can be loaded into a frame of another website. The application does not implement a proper frame-busting script, that would prevent the page from being loaded into another frame. Note that there are many types of simplistic redirection scripts that still leave the application vulnerable to clickjacking techniques, and should not be used.
When dealing with modern browsers, applications mitigate this vulnerability by issuing appropriate Content-Security-Policy or X-Frame-Options headers to indicate to the browser to disallow framing. However, many legacy browsers do not support this feature, and require a more manual approach by implementing a mitigation in Javascript. To ensure legacy support, a framebusting script is required.

General Recommendations

How to avoid it

Generic Guidance:

- Define and implement a a Content Security Policy (CSP) on the server side, including a frame-ancestors directive. Enforce the CSP on all relevant webpages.
- If certain webpages are required to be loaded into a frame, define a specific, whitelisted target URL.
- Alternatively, return a "X-Frame-Options" header on all HTTP responses. If it is necessary to allow a particular webpage to be loaded into a frame, define a specific, whitelisted target URL.
- For legacy support, implement framebusting code using Javascript and CSS to ensure that, if a page is framed, it is never displayed, and attempt to navigate into the frame to prevent attack. Even if navigation fails, the page is not displayed and is therefore not interactive, mitigating potential clickjacking attacks.

Specific Recommendations:

- Implement a proper framebuster script on the client, that is not vulnerable to frame-buster-busting attacks.
  - Code should first disable the UI, such that even if frame-busting is successfully evaded, the UI cannot be clicked. This can be done by setting the CSS value of the "display" attribute to "none" on either the "body" or "html" tags. This is done because, if a frame attempts to redirect and become the parent, the malicious parent can still prevent redirection via various techniques.
  - Code should then determine whether no framing occurs by comparing self === top; if the result is true, can the UI be enabled. If it is false, attempt to navigate away from the framing page by setting the top.location attribute to self.location.

Source Code Examples

JavaScript
Clickjackable Webpage

```
<html>
    <body>
     <button onclick="clicked();">
            Click here if you love ducks
        </button>
    </body>
</html>
```

Bustable Framebuster

```
<html>
    <head>
     <script>
            if ( window.self.location != window.top.location ) {
                window.top.location = window.self.location;
            }
        </script>
    </head>

    <body>
        <button onclick="clicked();">
            Click here if you love ducks
        </button>
    </body>
</html>
```

Proper Framebusterbusterbusting

```
<html>
    <head>
     <style> html {display : none; } </style>
        <script>
            if ( self === top ) {
                document.documentElement.style.display = 'block';
            }
            else {
                top.location = self.location;
            }
        </script>
    </head>

    <body>
        <button onclick="clicked();">
            Click here if you love ducks
        </button>
    </body>
</html>
```

Client Use Of Iframe Without Sandbox

Risk

What might happen

If an attacker can select the name of the library, or the location of the code file that is loaded by the application, they would be able to cause the application to execute arbitrary code. This effectively allows the attacker to control the code run by the application.

Cause

How does it happen

The application uses untrusted data to specify the library or code file, without proper sanitization. This causes the application to load any arbitrary code, as specified. The loaded code will then be executed.

General Recommendations

How to avoid it

- Do not dynamically load code libraries, especially not based on user input.
- If it is necessary to use untrusted data to select the library to be loaded, verify the selected library name matches a predefined set of whitelisted library names. Alternatively, use the value as an identifier to select from the whitelisted libraries.
- Validate any untrusted data used to load or process libraries or code files.

Source Code Examples

Improper Access Control (Authorization)

Weakness ID: 285 *(Weakness Class)* **Status:** Draft

Description

Description Summary

The software does not perform or incorrectly performs access control checks across all potential execution paths.

Extended Description

When access control checks are not applied consistently - or not at all - users are able to access data or perform actions that they should not be allowed to perform. This can lead to a wide range of problems, including information leaks, denial of service, and arbitrary code execution.

Alternate Terms

AuthZ:	"AuthZ" is typically used as an abbreviation of "authorization" within the web application security community. It is also distinct from "AuthC," which is an abbreviation of "authentication." The use of "Auth" as an abbreviation is discouraged, since it could be used for either authentication or authorization.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

Language-independent

Technology Classes

Web-Server: *(Often)*

Database-Server: *(Often)*

Modes of Introduction

A developer may introduce authorization weaknesses because of a lack of understanding about the underlying technologies. For example, a developer may assume that attackers cannot modify certain inputs such as headers or cookies.

Authorization weaknesses may arise when a single-user application is ported to a multi-user environment.

Common Consequences

Scope	Effect
Confidentiality	An attacker could read sensitive data, either by reading the data directly from a data store that is not properly restricted, or by accessing insufficiently-protected, privileged functionality to read the data.
Integrity	An attacker could modify sensitive data, either by writing the data directly to a data store that is not properly restricted, or by accessing insufficiently-protected, privileged functionality to write the data.
Integrity	An attacker could gain privileges by modifying or reading critical data directly, or by accessing insufficiently-protected, privileged functionality.

Likelihood of Exploit

High

Detection Methods

Automated Static Analysis

Automated static analysis is useful for detecting commonly-used idioms for authorization. A tool may be able to analyze related configuration files, such as .htaccess in Apache web servers, or detect the usage of commonly-used authorization libraries.

Generally, automated static analysis tools have difficulty detecting custom authorization schemes. In addition, the software's design may include some functionality that is accessible to any user and does not require an authorization check; an automated technique that detects the absence of authorization may report false positives.

Effectiveness: Limited

Automated Dynamic Analysis

Automated dynamic analysis may find many or all possible interfaces that do not require authorization, but manual analysis is required to determine if the lack of authorization violates business logic

Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.

Specifically, manual static analysis is useful for evaluating the correctness of custom authorization mechanisms.

Effectiveness: Moderate

These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules. However, manual efforts might not achieve desired code coverage within limited time constraints.

Demonstrative Examples

Example 1

The following program could be part of a bulletin board system that allows users to send private messages to each other. This program intends to authenticate the user before deciding whether a private message should be displayed. Assume that LookupMessageObject() ensures that the $id argument is numeric, constructs a filename based on that id, and reads the message details from that file. Also assume that the program stores all private messages for all users in the same directory.

(Bad Code)

Example Language: **Perl**

```
sub DisplayPrivateMessage {
my($id) = @_;
my $Message = LookupMessageObject($id);
print "From: " . encodeHTML($Message->{from}) . "<br>\n";
print "Subject: " . encodeHTML($Message->{subject}) . "\n";
print "<hr>\n";
print "Body: " . encodeHTML($Message->{body}) . "\n";
}

my $q = new CGI;
# For purposes of this example, assume that CWE-309 and
# CWE-523 do not apply.
if (! AuthenticateUser($q->param('username'), $q->param('password'))) {
ExitError("invalid username or password");
}

my $id = $q->param('id');
DisplayPrivateMessage($id);
```

While the program properly exits if authentication fails, it does not ensure that the message is addressed to the user. As a result, an authenticated attacker could provide any arbitrary identifier and read private messages that were intended for other users.

One way to avoid this problem would be to ensure that the "to" field in the message object matches the username of the authenticated user.

Observed Examples

Reference	Description
CVE-2009-3168	Web application does not restrict access to admin scripts, allowing authenticated users to reset administrative passwords.

CVE-2009-2960	Web application does not restrict access to admin scripts, allowing authenticated users to modify passwords of other users.
CVE-2009-3597	Web application stores database file under the web root with insufficient access control (CWE-219), allowing direct request.
CVE-2009-2282	Terminal server does not check authorization for guest access.
CVE-2009-3230	Database server does not use appropriate privileges for certain sensitive operations.
CVE-2009-2213	Gateway uses default "Allow" configuration for its authorization settings.
CVE-2009-0034	Chain: product does not properly interpret a configuration option for a system group, allowing users to gain privileges.
CVE-2008-6123	Chain: SNMP product does not properly parse a configuration option for which hosts are allowed to connect, allowing unauthorized IP addresses to connect.
CVE-2008-5027	System monitoring software allows users to bypass authorization by creating custom forms.
CVE-2008-7109	Chain: reliance on client-side security (CWE-602) allows attackers to bypass authorization using a custom client.
CVE-2008-3424	Chain: product does not properly handle wildcards in an authorization policy list, allowing unintended access.
CVE-2009-3781	Content management system does not check access permissions for private files, allowing others to view those files.
CVE-2008-4577	ACL-based protection mechanism treats negative access rights as if they are positive, allowing bypass of intended restrictions.
CVE-2008-6548	Product does not check the ACL of a page accessed using an "include" directive, allowing attackers to read unauthorized files.
CVE-2007-2925	Default ACL list for a DNS server does not set certain ACLs, allowing unauthorized DNS queries.
CVE-2006-6679	Product relies on the X-Forwarded-For HTTP header for authorization, allowing unintended access by spoofing the header.
CVE-2005-3623	OS kernel does not check for a certain privilege before setting ACLs for files.
CVE-2005-2801	Chain: file-system code performs an incorrect comparison (CWE-697), preventing defauls ACLs from being properly applied.
CVE-2001-1155	Chain: product does not properly check the result of a reverse DNS lookup because of operator precedence (CWE-783), allowing bypass of DNS-based access restrictions.

Potential Mitigations

Phase: Architecture and Design

Divide your application into anonymous, normal, privileged, and administrative areas. Reduce the attack surface by carefully mapping roles with data and functionality. Use role-based access control (RBAC) to enforce the roles at the appropriate boundaries.

Note that this approach may not protect against horizontal authorization, i.e., it will not protect a user from attacking others with the same role.

Phase: Architecture and Design

Ensure that you perform access control checks related to your business logic. These checks may be different than the access control checks that you apply to more generic resources such as files, connections, processes, memory, and database records. For example, a database may restrict access for medical records to a specific database user, but each record might only be intended to be accessible to the patient and the patient's doctor.

Phase: Architecture and Design

Strategy: Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, consider using authorization frameworks such as the JAAS Authorization Framework and the OWASP ESAPI Access Control feature.

Phase: Architecture and Design

For web applications, make sure that the access control mechanism is enforced correctly at the server side on every page. Users should not be able to access any unauthorized functionality or information by simply requesting direct access to that page.

One way to do this is to ensure that all pages containing sensitive information are not cached, and that all such pages restrict access to requests that are accompanied by an active and authenticated session token associated with a user who has the required permissions to access that page.

Phases: System Configuration; Installation

Use the access control capabilities of your operating system and server environment and define your access control lists accordingly. Use a "default deny" policy when defining these ACLs.

Relationships

Nature	Type	ID	Name	View(s) this relationship pertains to
ChildOf	Category	254	Security Features	**Seven Pernicious Kingdoms (primary)700**
ChildOf	Weakness Class	284	Access Control (Authorization) Issues	**Development Concepts (primary)699 Research Concepts (primary)1000**
ChildOf	Category	721	OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access	**Weaknesses in OWASP Top Ten (2007) (primary)629**
ChildOf	Category	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	**Weaknesses in OWASP Top Ten (2004) (primary)711**
ChildOf	Category	753	2009 Top 25 - Porous Defenses	**Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors (primary)750**
ChildOf	Category	803	2010 Top 25 - Porous Defenses	**Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors (primary)800**
ParentOf	Weakness Variant	219	Sensitive Data Under Web Root	**Research Concepts (primary)1000**
ParentOf	Weakness Base	551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization	**Development Concepts (primary)699** Research Concepts1000
ParentOf	Weakness Class	638	Failure to Use Complete Mediation	Research Concepts1000
ParentOf	Weakness Base	804	Guessable CAPTCHA	**Development Concepts (primary)699 Research Concepts (primary)1000**

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Missing Access Control
OWASP Top Ten 2007	A10	CWE More Specific	Failure to Restrict URL Access
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version: 1.5)*
1	Accessing Functionality Not Properly Constrained by ACLs	
13	Subverting Environment Variable Values	

17	Accessing, Modifying or Executing Executable Files
87	Forceful Browsing
39	Manipulating Opaque Client-based Data Tokens
45	Buffer Overflow via Symbolic Links
51	Poison Web Service Registry
59	Session Credential Falsification through Prediction
60	Reusing Session IDs (aka Session Replay)
77	Manipulating User-Controlled Variables
76	Manipulating Input to File System Calls
104	Cross Zone Scripting

References

NIST. "Role Based Access Control and Role Based Security". <http://csrc.nist.gov/groups/SNS/rbac/>.

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 4, "Authorization" Page 114; Chapter 6, "Determining Appropriate Access Control" Page 171. 2nd Edition. Microsoft. 2002.

Content History

Submissions

Submission Date	Submitter	Organization	Source
	7 Pernicious Kingdoms		Externally Mined

Modifications

Modification Date	Modifier	Organization	Source
2008-07-01	Eric Dalci	Cigital	External
	updated Time of Introduction		
2008-08-15		Veracode	External
	Suggested OWASP Top Ten 2004 mapping		
2008-09-08	CWE Content Team	MITRE	Internal
	updated Relationships, Other Notes, Taxonomy Mappings		
2009-01-12	CWE Content Team	MITRE	Internal
	updated Common Consequences, Description, Likelihood of Exploit, Name, Other Notes, Potential Mitigations, References, Relationships		
2009-03-10	CWE Content Team	MITRE	Internal
	updated Potential Mitigations		
2009-05-27	CWE Content Team	MITRE	Internal
	updated Description, Related Attack Patterns		
2009-07-27	CWE Content Team	MITRE	Internal
	updated Relationships		
2009-10-29	CWE Content Team	MITRE	Internal
	updated Type		
2009-12-28	CWE Content Team	MITRE	Internal
	updated Applicable Platforms, Common Consequences, Demonstrative Examples, Detection Factors, Modes of Introduction, Observed Examples, Relationships		
2010-02-16	CWE Content Team	MITRE	Internal
	updated Alternate Terms, Detection Factors, Potential Mitigations, References, Relationships		
2010-04-05	CWE Content Team	MITRE	Internal
	updated Potential Mitigations		

Previous Entry Names

Change Date	Previous Entry Name
2009-01-12	Missing or Inconsistent Access Control

BACK TO TOP

Client Insufficient ClickJacking Protection

Risk

What might happen

Clickjacking attacks allow an attacker to "hijack" a user's mouse clicks on a webpage, by invisibly framing the application, and superimposing it in front of a bogus site. When the user is convinced to click on the bogus website, e.g. on a link or a button, the user's mouse is actually clicking on the target webpage, despite being invisible.
This could allow the attacker to craft an overlay that, when clicked, would lead the user to perform undesirable actions in the vulnerable application, e.g. enabling the user's webcam, deleting all the user's records, changing the user's settings, or causing clickfraud.

Cause

How does it happen

The root cause of vulnerability to a clickjacking attack, is that the application's web pages can be loaded into a frame of another website. The application does not implement a proper frame-busting script, that would prevent the page from being loaded into another frame. Note that there are many types of simplistic redirection scripts that still leave the application vulnerable to clickjacking techniques, and should not be used.
When dealing with modern browsers, applications mitigate this vulnerability by issuing appropriate Content-Security-Policy or X-Frame-Options headers to indicate to the browser to disallow framing. However, many legacy browsers do not support this feature, and require a more manual approach by implementing a mitigation in Javascript. To ensure legacy support, a framebusting script is required.

General Recommendations

How to avoid it

Generic Guidance:

- Define and implement a a Content Security Policy (CSP) on the server side, including a frame-ancestors directive. Enforce the CSP on all relevant webpages.
- If certain webpages are required to be loaded into a frame, define a specific, whitelisted target URL.
- Alternatively, return a "X-Frame-Options" header on all HTTP responses. If it is necessary to allow a particular webpage to be loaded into a frame, define a specific, whitelisted target URL.
- For legacy support, implement framebusting code using Javascript and CSS to ensure that, if a page is framed, it is never displayed, and attempt to navigate into the frame to prevent attack. Even if navigation fails, the page is not displayed and is therefore not interactive, mitigating potential clickjacking attacks.

Specific Recommendations:

- Implement a proper framebuster script on the client, that is not vulnerable to frame-buster-busting attacks.
  - Code should first disable the UI, such that even if frame-busting is successfully evaded, the UI cannot be clicked. This can be done by setting the CSS value of the "display" attribute to "none" on either the "body" or "html" tags. This is done because, if a frame attempts to redirect and become the parent, the malicious parent can still prevent redirection via various techniques.
  - Code should then determine whether no framing occurs by comparing self === top; if the result is true, can the UI be enabled. If it is false, attempt to navigate away from the framing page by setting the top.location attribute to self.location.

Source Code Examples

Improper Exception Handling

Risk

What might happen

- An attacker could maliciously cause an exception that could crash the application, potentially resulting in a denial of service (DoS).
- Inadvertent application crashes may occur.

Cause

How does it happen

The application performs some operation, such as database or file access, that could throw an exception. Since the application is not designed to properly handle the exception, the application could crash.

General Recommendations

How to avoid it

Any method that could cause an exception should be wrapped in a try-catch block that:

- Explicitly handles expected exceptions
- Includes a default solution to explicitly handle unexpected exceptions

Source Code Examples

CSharp

Always catch exceptions explicitly.

```
try
{
// Database access or other potentially dangerous function
}
catch (SqlException ex)
{
// Handle exception
}
catch (Exception ex)
{
// Default handler for unexpected exceptions
}
```

Java

Always catch exceptions explicitly.

```
try
{
// Database access or other potentially dangerous function
}
catch (SQLException ex)
{
// Handle exception
}
catch (Exception ex)
{
// Default handler for unexpected exceptions
}
```

Information Exposure Through an Error Message

Risk

What might happen

Exposed details about the application’s environment, users, or associated data (for example, stack trace) could enable an attacker to find another flaw and help the attacker to mount an attack. This may also leak sensitive data, e.g. passwords or database fields.

Cause

How does it happen

The application handles exceptions in an insecure manner, including raw details directly in the error message. This could occur in various ways: by not handling the exception; printing it directly to the output or file; explicitly returning the exception object; or by configuration. These exception details may include sensitive information that could leak to the users due to the occurrence of the runtime error.

General Recommendations

How to avoid it

- Do not expose exception data directly to the output or users, instead return an informative, generic error message. Log the exception details to a dedicated log mechanism.
- Any method that could throw an exception should be wrapped in an exception handling block that:
  - Explicitly handles expected exceptions.
  - Includes a default solution to explicitly handle unexpected exceptions.
- Configure a global handler to prevent unhandled errors from leaving the application.

Source Code Examples

Serializable Class Containing Sensitive Data

Weakness ID: 499 *(Weakness Variant)* **Status:** Draft

Description

Description Summary

The code contains a class with sensitive data, but the class does not explicitly deny serialization. The data can be accessed by serializing the class through another class.

Extended Description

Serializable classes are effectively open classes since data cannot be hidden in them. Classes that do not explicitly deny serialization can be serialized by any other class, which can then in turn use the data stored inside it.

Time of Introduction

- Implementation

Applicable Platforms

Languages

Java

Common Consequences

Scope	Effect
Confidentiality	an attacker can write out the class to a byte stream, then extract the important data from it.

Likelihood of Exploit

High

Demonstrative Examples

Example 1

(Bad Code)

Example Language: **Java**

```
class Teacher {
private String name;
private String clas;
public Teacher(String name,String clas) {

//...
//Check the database for the name and address
this.SetName() = name;
this.Setclas() = clas;
}
}
```

Potential Mitigations

Phase: Implementation

In Java, explicitly define final writeObject() to prevent serialization. This is the recommended solution. Define the writeObject() function to throw an exception explicitly denying serialization.

Phase: Implementation

Make sure to prevent serialization of your objects.

Relationships

Nature	Type	ID	Name	View(s) this relationship pertains to
ChildOf	Weakness Class	485	Insufficient Encapsulation	**Development Concepts (primary)699 Research Concepts**

				(primary)1000
CanPrecede	Weakness Class	200	Information Exposure	Development Concepts699 Research Concepts1000

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Information leak through serialization

Content History

Submissions			
Submission Date	**Submitter**	**Organization**	**Source**
	CLASP		Externally Mined
Modifications			
Modification Date	**Modifier**	**Organization**	**Source**
2008-07-01	Eric Dalci	Cigital	External
	updated Time of Introduction		
2008-09-08	CWE Content Team	MITRE	Internal
	updated Common Consequences, Description, Relationships, Taxonomy Mappings		
2009-07-27	CWE Content Team	MITRE	Internal
	updated Demonstrative Examples		
Previous Entry Names			
Change Date	**Previous Entry Name**		
2008-04-11	Information Leak through Serialization		

BACK TO TOP

Use Of Hardcoded Password

Risk

What might happen

Hardcoded passwords expose the application to password leakage. If an attacker gains access to the source code, she will be able to steal the embedded passwords, and use them to impersonate a valid user. This could include impersonating end users to the application, or impersonating the application to a remote system, such as a database or a remote web service.
Once the attacker succeeds in impersonating the user or application, she will have full access to the system, and be able to do anything the impersonated identity could do.

Cause

How does it happen

The application codebase has string literal passwords embedded in the source code. This hardcoded value is used either to compare to user-provided credentials, or to authenticate downstream to a remote system (such as a database or a remote web service).
An attacker only needs to gain access to the source code to reveal the hardcoded password. Likewise, the attacker can reverse engineer the compiled application binaries, and easily retrieve the embedded password. Once found, the attacker can easily use the password in impersonation attacks, either directly on the application or to the remote system.
Furthermore, once stolen, this password cannot be easily changed to prevent further misuse, unless a new version of the application is compiled. Moreover, if this application is distributed to numerous systems, stealing the password from one system automatically allows a class break in to all the deployed systems.

General Recommendations

How to avoid it

- Do not hardcode any secret data in source code, especially not passwords.
- In particular, user passwords should be stored in a database or directory service, and protected with a strong password hash (e.g. bcrypt, scrypt, PBKDF2, or Argon2). Do not compare user passwords with a hardcoded value.
- Sytem passwords should be stored in a configuration file or the database, and protected with strong encryption (e.g. AES-256). Encryption keys should be securely managed, and not hardcoded.

Source Code Examples

Java

Hardcoded Admin Password

```
bool isAdmin(String username, String password) {
    bool isMatch = false;

    if (username.equals("admin")) {
        if (password.equals("P@ssw0rd"))
            return isMatch = true;
    }

    return isMatch;
}
```

No Hardcoded Credentials

```
bool isAdmin(String username, String password) {
    bool adminPrivs = false;

    if (authenticateUser(username, password)) {
        UserPrivileges privs = getUserPrivileges(username);

        if (privs.isAdmin)
            adminPrivs = true;
    }

    return adminPrivs;
}
```

Scanned Languages

Language	Hash Number	Change Date
Java	0129784306302162	9/30/2018
JavaScript	3602822811217894	9/30/2018
VbScript	1349101913133594	3/23/2018
PLSQL	02073854232X5614	9/30/2018
Typescript	1939975091058023	9/30/2018
Common	0206692917308612	9/30/2018