Analysis Results

WebGoat-main.zip

Report Date	2024-09-10 20:35:36
Report Author	admin
Classification Method	OWASP Top 10 2021
Product Version	11.0-SNAPSHOT+f428841a
Rules Version	11-SNAPSHOT.130321

Confidentiality Note

01 Project Information

Project Name

WebGoat-main.zip

UUID

185e14e9-0a35-4d27-8a31-4f6275f3bdcc

Project in DerScanner

Dynamics by vulnerabilities

Vulnerabilities are divided by severity level: critical, medium, low and info.

CRITICAL LEVEL	MEDIUM LEVEL	LOW LEVEL	INFORMATION
Likely to lead to compromise confidential data and violation of the integrity of the system.	May be less likely to lead to compromising confidential data and violating the integrity of the system, or are less serious security	Can become a potential security risk.	Signal a violation of good programming practice.

First of all, pay attention to vulnerabilities of critical and medium levels.

Dynamics by rating

The app score is calculated on a scale from 0 to 5. Score is calculated based on the number of critical and medium level vulnerabilities. The impact of critical vulnerabilities is greater than that of medium level vulnerabilities, and does not take into account the amount of code. Medium level vulnerabilities are taken into account based on their frequency and total number of source code lines.

Scan History

	Date and Time	Status	Languages	Lines of Code	Number of Vulnerabilities					Score
					Critical	Medium	Low	Info	Total	
1/1	2024-09-04 09:36:07	completed	Config files, JavaScript, Java, Scala, Kotlin, VBScript, HTML5, T-SQL, PL/SQL	364 602	130	592	375	199	1 296	0.1/5.0

02

Scan Information

1/1 2024-09-04 09:36:07
11-SNAPSHOT.130256

Scan Statistics

Status	completed
Score	0.1/5.0
Duration	0:15:57
Lines of Code	364 602

Vulnerabilities

Critical	Medium	Low	Info	Total
130	592	375	199	1 296

Language	Status	Duration	Lines of Code	Number of Vulnerabilities				
				Critical	Medium	Low	Info	Total
Config files	completed	0:00:13	195 189	30	146	49	0	225
JavaScript	completed	0:13:10	127 615	8	88	178	132	406
JVM languages	completed	0:02:18	18 352	86	126	148	31	391
VBScript	completed	0:00:01	11 541	0	0	0	0	0
HTML5	completed	0:00:10	11 017	6	224	0	36	266
T-SQL	completed	0:00:01	446	0	0	0	0	0
PL/SQL	completed	0:00:01	442	0	8	0	0	8

Language Statistics

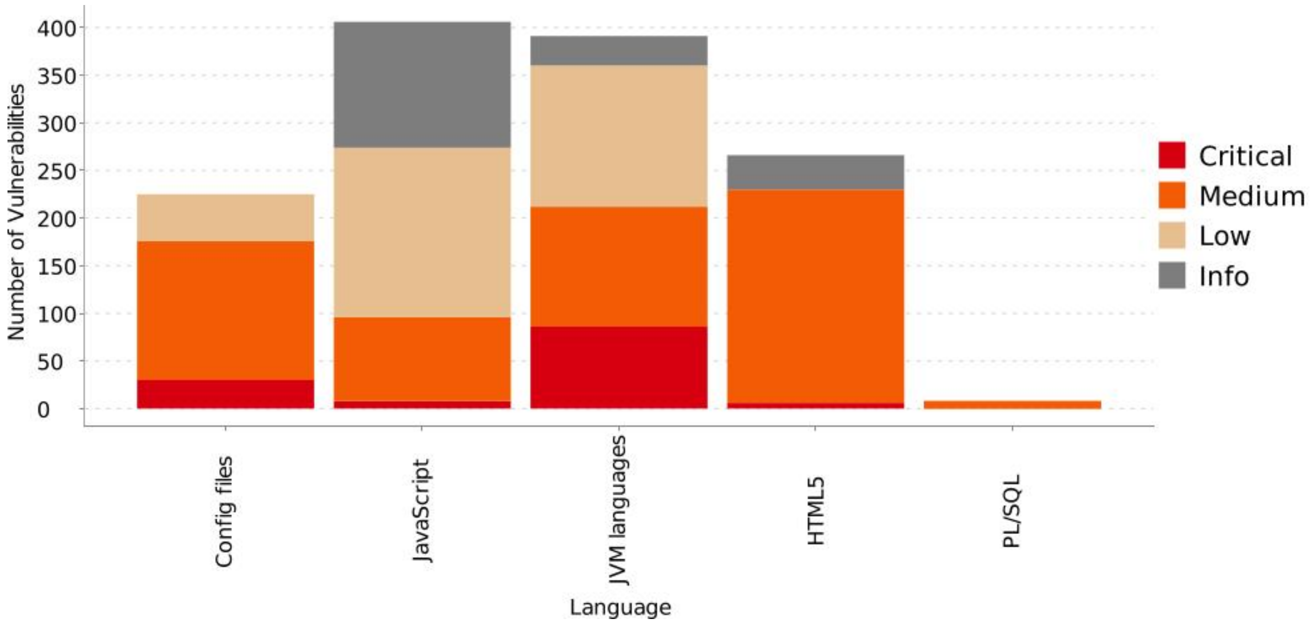
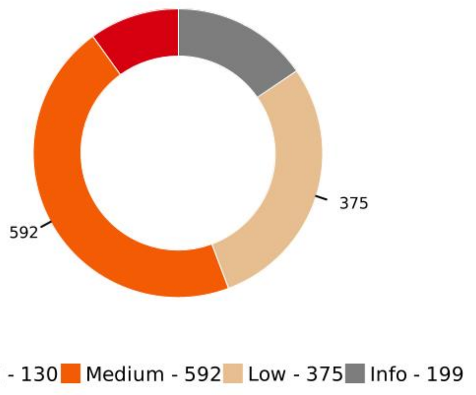
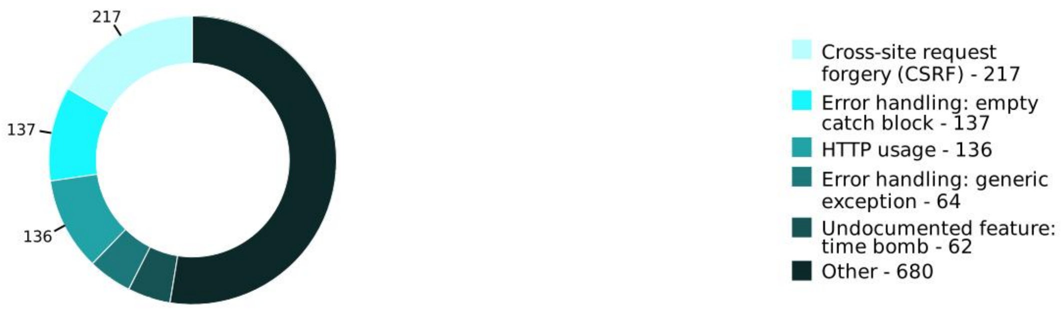


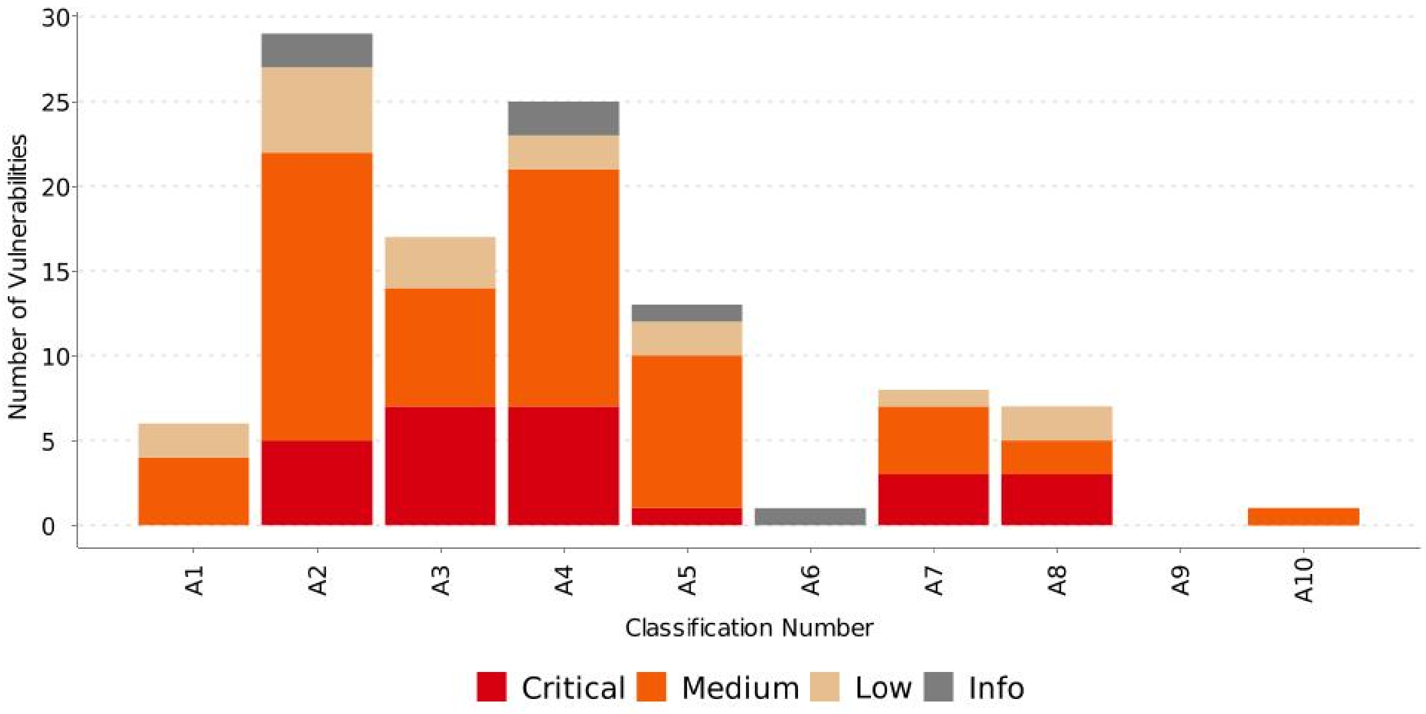
Diagram of identified vulnerabilities



Vulnerability Types



Classification by OWASP Top 10 2021



	Vulnerabilities					Occurrences				
	Critical	Medium	Low	Info	Total	Critical	Medium	Low	Info	Total
A1	0	4	2	0	6	0	15	2	0	17
A2	5	17	5	2	24	29	161	132	5	327
A3	7	7	3	0	16	86	24	5	0	115
A4	7	14	2	2	21	44	74	19	5	142
A5	1	9	2	1	9	1	110	60	4	175
A6	0	0	0	1	1	0	0	0	10	10
A7	3	4	1	0	7	13	85	8	0	106
A8	3	2	2	0	7	10	213	12	0	235
A9	0	0	0	0	0	0	0	0	0	0
A10	0	1	0	0	1	0	1	0	0	1

Vulnerability List

Vulnerabilities are displayed accordingly to export settings: **11 selected**

Actual: **11 of 1296**

A1 — Broken Access Control

Critical vulnerabilities		**0**
Medium-level vulnerabilities		**2***
Path manipulation	Java	2
✓ WebGoat-main/src/main/java/org/owasp/webgoat/lessons/challenges/challenge7/MD5.java:49		Confirmed
✓ WebGoat-main/src/main/java/org/owasp/webgoat/lessons/pathtraversal/ProfileUploadBase.java:42		Confirmed
Low-level vulnerabilities		**0**
Info-level vulnerabilities		**0**

A10 — Server-Side Request Forgery (SSRF)

Critical vulnerabilities	**0**
Medium-level vulnerabilities	**0**
Low-level vulnerabilities	**0**
Info-level vulnerabilities	**0**

* Rejected vulnerabilities are not taken into account

A2 Cryptographic Failures

Critical vulnerabilities		0
Medium-level vulnerabilities		2*
External information leak	JavaScript	1
● WebGoat-main/src/main/resources/webgoat/static/js/libs/ace.js:19459		Confirmed
External information leak	Java	1
● WebGoat-main/src/main/java/org/owasp/webgoat/lessons/jwt/JWTRefreshEndpoint.java:118		Confirmed
Low-level vulnerabilities		0
Info-level vulnerabilities		0

A3 Injection

Critical vulnerabilities		4*
SQL injection	Java	2
● WebGoat-main/src/main/java/org/owasp/webgoat/lessons/sqlinjection/advanced/SqlInjectionLesson6a.java:74		Confirmed
● WebGoat-main/src/main/java/org/owasp/webgoat/lessons/sqlinjection/introduction/SqlInjectionLesson5b.java:86		Confirmed
XML external entity (XXE) injection	Java	1
● WebGoat-main/src/main/java/org/owasp/webgoat/lessons/xxe/CommentsCache.java:105		Confirmed
Reflected XSS	Java	1
● WebGoat-main/src/main/java/org/owasp/webgoat/lessons/xss/CrossSiteScriptingLesson5a.java:93		Confirmed
Medium-level vulnerabilities		3*
Path manipulation	Java	2
● WebGoat-main/src/main/java/org/owasp/webgoat/lessons/challenges/challenge7/MD5.java:49		Confirmed
● WebGoat-main/src/main/java/org/owasp/webgoat/lessons/pathtraversal/ProfileUploadBase.java:42		Confirmed

* Rejected vulnerabilities are not taken into account

A3 Injection

Medium-level vulnerabilities			
Resource injection		Java	1
✓	WebGoat-main/src/main/java/org/owasp/webgoat/lessons/challenges/challenge7/Assignment7.java:92		Confirmed
Low-level vulnerabilities			0
Info-level vulnerabilities			0

A4 Insecure Design

Critical vulnerabilities			0
Medium-level vulnerabilities			4*
Path manipulation		Java	2
✓	WebGoat-main/src/main/java/org/owasp/webgoat/lessons/challenges/challenge7/MD5.java:49		Confirmed
✓	WebGoat-main/src/main/java/org/owasp/webgoat/lessons/pathtraversal/ProfileUploadBase.java:42		Confirmed
External information leak		JavaScript	1
✓	WebGoat-main/src/main/resources/webgoat/static/js/libs/ace.js:19459		Confirmed
External information leak		Java	1
✓	WebGoat-main/src/main/java/org/owasp/webgoat/lessons/jwt/JWTRefreshEndpoint.java:118		Confirmed
Low-level vulnerabilities			0
Info-level vulnerabilities			0

A5 Security Misconfiguration

Critical vulnerabilities			1*
XML external entity (XXE) injection		Java	1
✓	WebGoat-main/src/main/java/org/owasp/webgoat/lessons/xxe/CommentsCache.java:105		Confirmed

* Rejected vulnerabilities are not taken into account

A5		Security Misconfiguration
Medium-level vulnerabilities		2*
External information leak	Java	1
✓ WebGoat-main/src/main/java/org/owasp/webgoat/lessons/jwt/JWTRefreshEndpoint.java:118		Confirmed
External information leak	JavaScript	1
✓ WebGoat-main/src/main/resources/webgoat/static/js/libs/ace.js:19459		Confirmed
Low-level vulnerabilities		0
Info-level vulnerabilities		0

A6	Vulnerable and Outdated Components
Critical vulnerabilities	0
Medium-level vulnerabilities	0
Low-level vulnerabilities	0
Info-level vulnerabilities	0

A7	Identification and Authentication Failures
Critical vulnerabilities	0
Medium-level vulnerabilities	0
Low-level vulnerabilities	0
Info-level vulnerabilities	0

A8	Software and Data Integrity Failures
Critical vulnerabilities	0
Medium-level vulnerabilities	2*

* Rejected vulnerabilities are not taken into account

A8	Software and Data Integrity Failures	
Medium-level vulnerabilities		
Cross-site request forgery (CSRF)	Java	1
✓ WebGoat-main/src/main/java/org/owasp/webgoat/container/WebSecurityConfig.java:86		Confirmed
Cross-site request forgery (CSRF)	HTML5	1
✓ WebGoat-main/src/main/resources/lessons/bypassrestrictions/html/BypassRestrictions.html: 19		Confirmed
Low-level vulnerabilities		0
Info-level vulnerabilities		0

* Rejected vulnerabilities are not taken into account

Analysis Results

A3

Reflected XSS (Java)

Description

The reflected XSS or client type XSS attack is possible.
Cross-site scripting is one of the most common types of attacks on web applications. XSS attacks take seventh place in the “OWASP Top 10 2017” list of ten most significant vulnerabilities in web applications.
The main phase of any XSS attack is an imperceptible for the victim execution of a malicious code in the context of the vulnerable application. For this purpose, the functionality of the client application (browser) is used that allows to automatically execute scripts embedded in web page code. In most cases, these malicious scripts are implemented in JavaScript.
Consequences of an XSS attack vary from violations of application functionality to complete loss of user data confidentiality. The malicious code during the XSS attack can steal user HTTP-cookie, which gives an attacker the ability to make requests to the server on behalf of the user.
OWASP suggests the following classification of XSS attacks:

- Server type XSS occurs when data from an untrusted source is included in the response returned by the server. The source of such data can be both user input and server database (where it had been previously injected by an attacker who exploited vulnerabilities in the server-side application).
- Client type XSS occurs when the raw data from the user input contains code that changes the Document Object Model (DOM) of the web page received from the server. The source of such data can be both the DOM and the data received from the server (e.g., in response to an AJAX request).

Typical server type attack scenario:

1. Unvalidated data, usually from a HTTP request, gets into the server part of the application.
2. The server dynamically generates a web page that contains the unvalidated data.
3. In the process of generating a web page, server does not prevent the inclusion of an executable code that can be executed in the client (browser), such as JavaScript code language, HTML-tags, HTML-attributes, Flash, ActiveX, etc., in the page code.
4. The victim’s client application displays the web page that contains the malicious code injected via data from an untrusted source.
5. Since malicious code is injected in the web page coming from the known server, the client part of the application (browser) executes it with the rights set for the application.
6. This violates the same-origin policy, according to which the code from the one source must not get an access to resources from another source.

Client type attacks are executed in a similar way with the only difference that the malicious code is injected during the phase of the client application work with the document object model received from the server.

Example

In the following example, the JSP code reads the valueof the eid parameter (employee ID) and displays it on the page:
<% String eid = request.getParameter("eid"); %>
Employee ID: <%= eid %>
If the user follows the attacker’s link containing malicious script as the value of eid, this script will be executed in the browser, in particular, sending cookies, session IDs or other victim’s valuable information to the attacker.
Example for Scala application based on the Play framework: the raw method of the play.templates.JavaExtensions class returns a text without template escaping.

Recommendations

- Implement a validation mechanism. Whitelist is more secure but less flexible than the blacklist. The blacklist must at least include the characters “&”, “<”, “>”, and quotation marks.
- Many web application servers provide their own mechanisms of protection against XSS, but they may not be considered sufficient. There is no guarantee that the application will run in conjunction with the server that updates these mechanisms timely and completely.

Links

1. OWASP: Cross-site Scripting (XSS)
2. CWE-79: Improper Neutralization of Input During Web Page Generation
3. Types of Cross-Site Scripting - OWASP
4. OWASP: XSS Prevention Cheat Sheet
5. OWASP Top 10-2017 A7-Cross-Site Scripting (XSS)
6. CWE CATEGORY: OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure
7. CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)
8. CWE-81: Improper Neutralization of Script in an Error Message Web Page
9. CWE-83: Improper Neutralization of Script in Attributes in a Web Page

Vulnerability Entries

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/xss/CrossSiteScriptingLesson5a.java:93

Level	Critical

Status Confirmed

```
90      .output(cart.toString())
91      .build();
92 } else {
93   return success(this)
94      .feedback("xss-reflected-5a-success-alert")
95      .output(cart.toString())
96      .build();
```

Trace

@org.springframework.web.bind.annotation.GetMapping("/CrossSiteScripting/attack5a") @org.springframework.web.bind.annotation.ResponseBody public org.owasp.webgoat.container.assignments.AttackResult completed(@org.springframework.web.bind.annotation.RequestParam java.lang.Integer QTY1, @org.springframework.web.bind.annotation.RequestParam java.lang.Integer QTY2, @org.springframework.web.bind.annotation.RequestParam java.lang.Integer QTY3, @org.springframework.web.bind.annotation.RequestParam java.lang.Integer QTY4, @org.springframework.web.bind.annotation.RequestParam java.lang.String field1, @org.springframework.web.bind.annotation.RequestParam java.lang.String field2)

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/xss/CrossSiteScriptingLesson5a.java:55#1

```
52
53  @GetMapping("/CrossSiteScripting/attack5a")
54  @ResponseBody
55  public AttackResult completed(
56      @RequestParam Integer QTY1,
57      @RequestParam Integer QTY2,
58      @RequestParam Integer QTY3,
59 ...
60      userSessionData.setValue("xss-reflected1-complete", "false");
61      return failed(this).feedback("xss-reflected-5a-failure").output(cart.toString()).build();
62    }
63   }
64 }
```

return

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/xss/CrossSiteScriptingLesson5a.java:93

```
90      .output(cart.toString())
91      .build();
92 } else {
93   return success(this)
94      .feedback("xss-reflected-5a-success-alert")
95      .output(cart.toString())
96      .build();
```

A3

SQL injection (Java)

Description

SQL injection is possible. This can be exploited to bypass the authentication mechanism, access all database entries, or execute malicious code with application rights.
Client side code injection attacks take the first place in the “OWASP Top 10 2017” web application vulnerabilities ranking and the seventh place in the “OWASP Mobile Top 10 2014”. ranking. The level of potential damage from such an attack depends on the user input validation performance and file protection mechanisms.
SQL Injection occurs when a database query is based on data from an untrusted source (e.g., user input). In the absence of proper validation an attacker can modify the query to execute malicious SQL query.
The most common variants of SQL injection:

- Direct addition of malicious code into a string variable, based on which the SQL query is generated.
- Premature termination of the correct SQL command via the “– ” sequence of characters (interpreted as the beginning of a comment). The contents of the string after this sequence will be ignored during the execution of SQL command.
- Addition of the “;” character (interpreted as the end of the command), and other malicious commands (request splitting) to the input string variable.
- Password guessing via the sequential execution of SQL queries.

Example

Let the application substitute the user entered city name to the SQL query. For example, for the input string “London” we have:
SELECT * FROM Orders WHERE City = 'London'
An attacker can enter the following query:
London'; drop table Orders--
In this case, a query will be built that searches through the table and then deletes the table:
SELECT * FROM Orders WHERE City = 'London';drop table Orders-- '

Recommendations

- Do not make assumptions about the type or amount of data entered by a user.
- Implement a mechanism of validation for data entered by a user.
- Escape special characters (“;”, “–”, “/*”, “*/”, “‘”; the exact list depends on the database type).
- Use stored procedures to validate user input along with the mechanism of parameters filtering.

Links

1. OWASP Top 10 2017-A1-Injection
2. OWASP: SQL Injection
3. WASC-19: SQL Injection
4. CAPEC-66: SQL Injection
5. Understanding SQL Injection – cisco.com
6. CWE CATEGORY: OWASP Top Ten 2017 Category A1 - Injection
7. CWE-89

Vulnerability Entries

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/sqlinjection/advanced/SqlInjectionLesson6a.java:74

Level	Critical
Status	Confirmed

```
71 try (Statement statement =
72     connection.createStatement(
73         ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY)) {
74   ResultSet results = statement.executeQuery(query);
75
76   if ((results != null) && results.first()) {
77     ResultSetMetaData resultsMetaData = results.getMetaData();
```

Trace

@org.springframework.web.bind.annotation.
PostMapping("/SqlInjectionAdvanced/attack6a")
@org.springframework.web.bind.annotation.
ResponseBody
public org.owasp.webgoat.container.assignments.
AttackResult completed(@org.springframework.
web.bind.annotation.RequestParam("userid_6a")
java.lang.String userId)

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/sqlinjection/advanced/SqlInjectionLesson6a
java:56#60

```
53
54 @PostMapping("/SqlInjectionAdvanced/attack6a")
55 @ResponseBody
56 public AttackResult completed(@RequestParam(value = "userid_6a") String userId) {
57   return injectableQuery(userId);
58   // The answer: Smith' union select userid,user_name, password,cookie,cookie, cookie,
userid from
59   // user_system_data --
60 }
61
62 public AttackResult injectableQuery(String accountName) {
63   String query = "";
```

Statement.executeQuery()

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/sqlinjection/advanced/SqlInjectionLesson6a
java:74

```
try (Statement statement =
   connection.createStatement(
       ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY)) {
  ResultSet results = statement.executeQuery(query);

  if ((results != null) && results.first()) {
    ResultSetMetaData resultsMetaData = results.getMetaData();
```

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/sqlinjection/introduction/SqlInjectionLesson5b.java:86

Level	Critical
Status	Confirmed

```
// String query = "SELECT * FROM user_data WHERE Login_Count = " + login_count + " and userid
// = " + accountName, ;
try {
  ResultSet results = query.executeQuery();

  if ((results != null) && (results.first() == true)) {
    ResultSetMetaData resultsMetaData = results.getMetaData();
```

Trace

Connection.prepareStatement()

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/sqlinjection/introduction/SqlInjectionLesson5b.java:65

```
String queryString = "SELECT * From user_data WHERE Login_Count = ? and userid= " +
accountName;
try (Connection connection = dataSource.getConnection()) {
  PreparedStatement query =
65     connection.prepareStatement(
66         queryString, ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.
CONCUR_READ_ONLY);
67
68   int count = 0;
```

PreparedStatement.executeQuery()

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/sqlinjection/introduction/SqlInjectionLesson5b.java:86

```
83 // String query = "SELECT * FROM user_data WHERE Login_Count = " + login_count + " and
userid
84 // = " + accountName, ;
85 try {
86   ResultSet results = query.executeQuery();
87
88   if ((results != null) && (results.first() == true)) {
89     ResultSetMetaData resultsMetaData = results.getMetaData();
```

A3

A5

XML external entity (XXE) injection (Java)

Description

XXE (XML eXternal Entity) attack is possible. An attacker can cause failures in the application work or gain access to sensitive data.
XML provides a mechanism to enable including third-party files' content into the file via the entity mechanism defined in the DTD (Document Type Definitions). If the external entity is defined in the XML header, the developer is able to use its contents in XML file. Herein validation of external entities at XML parsing phase is not performed.
If the application works with the XML file received from an untrusted source (for example, from the data entered by a user), the attacker is able to inject malicious or not provided by the application external entity into the XML file, and thus disrupt the functionality of the application.

Example

In the following example, an attacker includes the external entity corresponding to a local file that contains confidential information (passwords) into the XML file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" > ]>
<foo>&xxe;</foo>
```

In the following example, an attacker includes the external entity of “infinite” volume (system pseudo-random number generator) into the XML file, which causes application freezes or crashes due to lack of memory:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE s [
<!ENTITY x "/dev/random">
]>
<foo>
&x;
&x;
[...]
</foo>
```

Recommendations

- Implement user input data validation mechanism.
- To work with XML files use libraries that provide protection against XXE attacks.
- Use the following settings of the XML factory, parser, or reader: factory.setFeature("http://xml.org/sax/features/external-general-entities", false); , factory.setFeature("http://xml.org/sax/features/external-parameter-entities", false);.
- If the DOCTYPE declaration of inline type is not required, this feature should be disabled: factory.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);.

Links

1. OWASP Top 10 2017-A1-Injection
2. OWASP: Testing for XML Injection
3. How we got read access on Google’s production servers – blog.detectify.com
4. Identifying Xml eXternal Entity vulnerability (XXE) – Philippe Arteau / blog.h3xstream.com
5. CWE CATEGORY: OWASP Top Ten 2017 Category A1 - Injection

Vulnerability Entries

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/xxe/CommentsCache.java:105

Level	Critical
Status	Confirmed

```
102   xif.setProperty(XMLConstants.ACCESS_EXTERNAL_SCHEMA, ""); // compliant
103 }
104

105 var xsr = xif.createXMLStreamReader(new StringReader(xml));

106
107 var unmarshaller = jc.createUnmarshaller();
108 return (Comment) unmarshaller.unmarshal(xsr);
```

Trace

@org.springframework.web.bind.annotation.PostMapping(path

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/xxe/ContentTypeAssignment.java:60#86

```
57
58 @PostMapping(path = "xxe/content-type")
59 @ResponseBody

60 public AttackResult createNewUser(
61     HttpServletRequest request,
62     @RequestBody String commentStr,
63     @RequestHeader("Content-Type") String contentType) {
64 ...
65   }
66
67   return attackResult;
68 }

69
70 private boolean checkSolution(Comment comment) {
71   String[] directoriesToCheck =
```

XMLInputFactory.createXMLStreamReader()

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/xxe/CommentsCache.java:105

```
102   xif.setProperty(XMLConstants.ACCESS_EXTERNAL_SCHEMA, ""); // compliant
103 }
104

105 var xsr = xif.createXMLStreamReader(new StringReader(xml));

106
107 var unmarshaller = jc.createUnmarshaller();
108 return (Comment) unmarshaller.unmarshal(xsr);
```

A8

Cross-site request forgery (CSRF) (HTML5)

Description

Cross-Site Request Forgery (CSRF) is possible.
Cross-Site Request Forgery (CSRF) attacks rank eighth on the OWASP Top 10 2013. CSRF is a type of attack that occurs when a malicious website, email or blog forces a user’s browser to perform an action on another site where the user is logged in.
Possible scenario of an attack:
The victim goes to a site created by the attacker, and a request is secretly sent on his behalf to another server (for example, a payment system server) that performs some kind of malicious operation (e.g., transferring money to the attacker’s account). In order to carry out this attack, the victim must be authenticated on the server to which the request is sent and the request must not require any confirmation from the user, which cannot be ignored or forged by the attacking script.

Example

In the following example, the web application allows administrators to create new accounts:

```
<form method="POST" action="/new_user">
  Name of new user: <input type="text" name="username">
  Password for new user: <input type="password" name="user_passwd">
    <input type="submit" name="action" value="Create User">
</form>
```

An attacker can create a malicious web site that contains the following code:

```
<form method="POST" action="http://www.example.com/new_user">
```

```
  <input type="hidden" name="username" value="hacker">
  <input type="hidden" name="user_passwd" value="hacked">
</form>
<script>
  document.usr_form.submit();
</script>
```

If the administrator visits a malicious web site during an open session on the vulnerable site, unbeknown to him/her an account will be created, which an attacker will use later.
Most browsers with every HTTP request transfer the referer header containing the address of the web site from which the transition occurs. However, since the attacker can overwrite the referer contents, this header does not help to counteract CSRF-attacks.

Recommendations

- If the application uses cookies, include in each form a secret value that can be validated on the server to verify the legitimacy of the request. The identifier (token) must be unique for each request, and not for the session. The token should not be easy to guess; it needs to be protected as well as the session token, for instance, via TLS.
- Use the framework provided mechanisms of protection against CSRF.
- Use additional mechanisms for verifying the request legitimacy, e.g., CAPTCHA, re-authentication, one-time tokens.
- Send the session ID not only as a cookie, but also as the value of the hidden field. The server must verify that these values match. An attacker will not be able to modify the value of the session identifier due to the same origin policy.
- Limit the session time. CSRF-attacks are successful only if they are carried out while the victim’s session on the vulnerable web site is valid. Reducing the session time reduces the likelihood of CSRF.

The described techniques only protect against CSRF, but not against cross-site scripting (XSS).

Links

1. OWASP Top 10 2013-A8-Cross-Site Request Forgery (CSRF)
2. CWE-352: Cross-Site Request Forgery (CSRF)
3. Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet - OWASP
4. CWE-1034

Vulnerability Entries

WebGoat-main/src/main/resources/lessons/bypassrestrictions/html/BypassRestrictions.html:19

Level	Medium
Status	Confirmed

```
<div class="attack-container">
    <div class="assignment-success"><i class="fa fa-2 fa-check hidden" aria-hidden="true"
></i></div>
    <div class="container-fluid">

        <form class="attack-form" accept-charset="UNKNOWN" name="fieldRestrictions"

              method="POST"
              action="BypassRestrictions/FieldRestrictions">

```

A8

Cross-site request forgery (CSRF) (Java)

Description

Cross Site Request Forgery (CSRF) is possible.
Cross Site Request Forgery attacks take the eighth place in the “OWASP Top 10 2013” web application vulnerabilities ranking. CSRF is a type of attack that occurs when a malicious web site, email, blog, instant message, or program causes a user’s web browser to perform an unwanted action on a trusted site for which the user is currently authenticated.
A possible attack scenario:
A victim visits the website created by attacker. Then the request is sent to another server (e.g. the server of the payment system) from victim’s face and carrying out some malicious action (e.g., transfer money to the account of the attacker). In order to implement this attack the victim should be authenticated on the server to send the request, and this request should not require any confirmation from the user that cannot be ignored or tampered with the attacking script.

Example

In the following example, the web application allows administrators to create new accounts:

```
RequestBuilder rb = new RequestBuilder(RequestBuilder.POST, "/new_user");
```

```
body = addToPost(body, new_username);
body = addToPost(body, new_passwd);
rb.sendRequest(body, new NewAccountCallback(callback));
```

An attacker can create a malicious web site that contains the following code:

```
RequestBuilder rb = new RequestBuilder(RequestBuilder.POST, "http://www.
example.com/new_user");
body = addToPost(body, "attacker";
body = addToPost(body, "haha");
rb.sendRequest(body, new NewAccountCallback(callback));
```

If the administrator visits a malicious website during an open session on the vulnerable site, unbeknown to him/her an account will be created, which an attacker will use later.

Most browsers with every HTTP request transfer the referer header containing the address of the website from which the transition occurs. However, since the attacker can overwrite the referer contents, this header does not help to counteract CSRF-attacks.

Recommendations

- If the application uses cookies, include in each form a secret value that can be validated on the server to verify the legitimacy of the request. One of the possible options is to use a random request ID.

In the following example, the web application uses a random request ID:

```
 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));
```

The identifier (token) must be unique for each request, and not for the session. The token should not be easy to guess; it needs to be protected as well as the session token, for instance, via TLS.

- Use the framework provided mechanisms of protection against CSRF.
- Use additional mechanisms for verifying the request legitimacy, e.g., CAPTCHA, re-authentication, one-time tokens.
- Send the session ID not only as a cookie, but also as the value of the hidden field. The server must verify that these values match. An attacker will not be able to modify the value of the session identifier due to the same origin policy.

- Limit the session time. CSRF-attacks are successful only if they are carried out while the victim’s session on the vulnerable web site is valid. Reducing the session time reduces the likelihood of CSRF.

The described techniques only protect against CSRF, but not against cross-site scripting (XSS).

Links

1. OWASP Top 10 2013-A8-Cross-Site Request Forgery (CSRF)
2. CWE-352: Cross-Site Request Forgery (CSRF)
3. Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet - OWASP
4. CWE-1034

Vulnerability Entries

WebGoat-main/src/main/java/org/owasp/webgoat/container/WebSecurityConfig.java:86

Level	Medium
Status	Confirmed

```
83      oidc.loginPage("/login");
84    })
85 .logout(logout -> logout.deleteCookies("JSESSIONID").invalidateHttpSession(true))
86 .csrf(csrf -> csrf.disable())
87 .headers(headers -> headers.disable())
88 .exceptionHandling(
89    handling ->
```

A2

A4

A5

External information leak (Java)

Description

System configuration information leak is possible. This can help an attacker to plan an attack.
Debug information and error messages can be written to the log, displayed to the console, or sent to the user depending on the system settings. In some cases, an attacker can make a conclusion about the system vulnerabilities from the error message. For example, a database error can indicate insecurity against SQL injection attacks. Information about the version of the operating system, application server and system configuration can also be of value to the attacker.
In this case, we are talking about the external leak: information about the system is transferred to another machine over the network. External leaks are more dangerous than internal ones.
This applies both to web applications and mobile applications.

Example

In the following example, the application allows the external exclusion leak via HTTP response:

```
protected void doPost (HttpServletRequest req, HttpServletResponse res) throws IOException {
   //...
   PrintWriter out = res.getWriter();
   try {
      //...
   } catch (Exception e) {
      out.println(e.getMessage());
   }
}
```

In this case, the error message may contain information about the operating system, installed applications or features of the system configuration.
A similar example for Android: the application sends a broadcast message with the stack trace of handled exception to all the registered receivers.

```
try {
   //...
} catch (Exception e) {
   String exception = Log.getStackTraceString(e);
   Intent i = new Intent();
   i.setAction("SEND_EXCEPTION");
   i.putExtra("exception", exception);
   view.getContext().sendBroadcast(i);
}
```

Recommendations

- Exclude detailed information about the system and its configuration from the error messages.
- If possible, do not send error messages using broadcast messaging.

Links

1. OWASP Top 10 2017-A3-Sensitive Data Exposure
2. OWASP Top 10 2017-A6-Security Misconfiguration
3. OWASP Top 10 2013-A5-Security Misconfiguration
4. OWASP Top 10 2013-A6-Sensitive Data Exposure
5. CWE-497: Exposure of System Data to an Unauthorized Control Sphere
6. CWE CATEGORY: OWASP Top Ten 2017 Category A5 - Broken Access Control
7. CWE CATEGORY: OWASP Top Ten 2017 Category A6 - Security Misconfiguration
8. CWE-200: Exposure of Sensitive Information to an Unauthorized Actor
9. CWE-209: Generation of Error Message Containing Sensitive Information

Vulnerability Entries

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/jwt/JWTRefreshEndpoint.java:118

Level	Medium
Status	Confirmed

```
115   }
116   return ok(failed(this).feedback("jwt-refresh-not-tom").feedbackArgs(user).build());
117 } catch (ExpiredJwtException e) {
118   return ok(failed(this).output(e.getMessage()).build());
119 } catch (JwtException e) {
120   return ok(failed(this).feedback("jwt-invalid-token").build());
121 }
```

Trace

SQLException.getMessage()

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/sqlinjection/introduction/SqlInjectionLesson8
java:105

```
102   }
103 } catch (SQLException e) {
104   return failed(this)
105       .output("<br><span class='feedback-negative'>" + e.getMessage() + "</span>")
106       .build();
107 }
108
```

ResponseEntity.ok()

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/jwt/JWTRefreshEndpoint.java:118

```
115   }
116   return ok(failed(this).feedback("jwt-refresh-not-tom").feedbackArgs(user).build());
117 } catch (ExpiredJwtException e) {
118   return ok(failed(this).output(e.getMessage()).build());
119 } catch (JwtException e) {
120   return ok(failed(this).feedback("jwt-invalid-token").build());
121 }
```

A1

A3

A4

Path manipulation (Java)

Description

Using data from an untrusted source when working with the file system may give an attacker access to important system files.
By manipulating variables that reference files with “dot-dot-slash (../)” sequences and its variations or by using

absolute file paths, it may be possible to access arbitrary files and directories stored on file system including application source code or configuration and critical system files.

Example

In the following example, the application uses the value of the HTTP request parameter to specify the name of the file that is to be deleted. An attacker can set the string ../../tomcat/conf/server.xml as a parameter and thus delete the configuration file.

```
String rName = request.getParameter("reportName");
File rFile = new File("/usr/local/apfr/reports/" + rName);
rFile.delete()
```

Recommendations

- Create a white list of acceptable names from which the user can choose. Do not use values entered by the user without validation.

Links

1. OWASP Top 10 2017-A1-Injection
2. OWASP Top 10 2017-A5-Broken Access Control
3. OWASP Top 10 2013-A4-Insecure Direct Object References
4. CWE-73: External Control of File Name or Path
5. Path Traversal - OWASP
6. CWE CATEGORY: OWASP Top Ten 2017 Category A1 - Injection
7. CWE-23
8. CWE-36
9. Restrict path access to prevent path traversal
10. A01:2021 - Broken Access Control
11. CWE-35: Path Traversal
12. A03:2021 - Injection

Vulnerability Entries

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/challenges/challenge7/MD5.java:49

Level	Medium
Status	Confirmed

```
46 } else {
47   for (String element : args) {
48     try {
49       System.out.println(MD5.getHashString(new File(element)) + " " + element);
50     } catch (IOException x) {
51       System.err.println(x.getMessage());
52     }
```

Trace

/** * Command line program that will take files as arguments and output the MD5 sum for each file. * * @param args * command line arguments * @since ostermillerutils 1.00.00 */
public static void main(java.lang.String[] args)

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/challenges/challenge7/MD5.java:43#55

```
40  * @param args command line arguments
41  * @since ostermillerutils 1.00.00
42  */
43 public static void main(String[] args) {
44   if (args.length == 0) {
45     System.err.println("Please specify a file.");
46   } else {
47 ...
48       }
49     }
50   }
51 }
52
53 /**
54  * Gets this hash sum as an array of 16 bytes.
```

new File()

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/challenges/challenge7/MD5.java:49

```
46 } else {
47   for (String element : args) {
```

```
48    try {
49      System.out.println(MD5.getHashString(new File(element)) + " " + element);
50    } catch (IOException x) {
51      System.err.println(x.getMessage());
52    }
```

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/pathtraversal/ProfileUploadBase.java:42

Level	Medium
Status	Confirmed

```
39 File uploadDirectory = cleanupAndCreateDirectoryForUser();
40
41 try {
42   var uploadedFile = new File(uploadDirectory, fullName);
43   uploadedFile.createNewFile();
44   FileCopyUtils.copy(file.getBytes(), uploadedFile);
45
```

Trace

@org.springframework.web.bind.annotation.PostMapping(value

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/pathtraversal/ProfileUpload.java:36#40

```
33    consumes = ALL_VALUE,
34    produces = APPLICATION_JSON_VALUE)
35 @ResponseBody
36 public AttackResult uploadFileHandler(
37    @RequestParam("uploadedFile") MultipartFile file,
38    @RequestParam(value = "fullName", required = false) String fullName) {
39   return super.execute(file, fullName);
```

```
40 }

41
42 @GetMapping("/PathTraversal/profile-picture")
43 @ResponseBody
```

new File()

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/pathtraversal/ProfileUploadBase.java:42

```
39 File uploadDirectory = cleanupAndCreateDirectoryForUser();
40
41 try {

42   var uploadedFile = new File(uploadDirectory, fullName);

43   uploadedFile.createNewFile();
44   FileCopyUtils.copy(file.getBytes(), uploadedFile);
45
```

A3

Resource injection (Java)

Description

Using data from an untrusted source to identify the resource allows an attacker to view or modify protected system resources.
The injection when working with resources (resource injection) occurs when an attacker can specify the identifier to access the system resources (for example, the port number for the network resource access). This allows him/her in particular to transfer valuable data to a third party server.
Injection vulnerabilities take the first place in the “OWASP Top 10 2017” web-application vulnerabilities ranking.

Example

In the following example, the application uses the HTTP request parameter when creating socket:

```
String remotePort = request.getParameter("remotePort");
ServerSocket srvr = new ServerSocket(remotePort);
Socket skt = srvr.accept();
```

In a similar example for an Android application, the URL derived from the intent is

```
used to load pages into WebView.
WebView webview = new WebView(this);
setContentView(webview);
String url = this.getIntent().getExtras().getString("url");
webview.loadUrl(url);
```

Recommendations

- Create a whitelist of valid resource IDs and allow a user to select from this list and not to set his/her own value.
- If maintaining a whitelist is too difficult because of the large number of valid IDs, create a whitelist of characters allowed in identifiers. Blacklist in this case is ineffective, as it is likely to initially be incomplete, or sooner or later cease to be relevant.
- If nonetheless blacklist is chosen as a validation mechanism, make sure that it takes into account all the possible encodings and special character values (different for different operating systems). Changing the list should be simple when changing the requirements for validation.

Links

1. OWASP Top 10 2017-A1-Injection
2. OWASP Top 10 2013-A1-Injection
3. OWASP Top 10 2013-A4-Insecure Direct Object References
4. CWE-99: Improper Control of Resource Identifiers (‘Resource Injection’)
5. CWE CATEGORY: OWASP Top Ten 2017 Category A1 - Injection
6. CWE-1030

Vulnerability Entries

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/challenges/challenge7/Assignment7.java:92

Level	Medium
Status	Confirmed

```
89         .recipient(username)
90         .time(LocalDateTime.now())
91         .build();
92    restTemplate.postForEntity(webWolfMailURL, mail, Object.class);
93   }
94 }
95 return success(this).feedback("email.send").feedbackArgs(email).build();
```

Trace

@org.springframework.web.bind.annotation.PostMapping("/challenge/7") @org.springframework.web.bind.annotation.ResponseBody public org.owasp.webgoat.container.assignments.AttackResult sendPasswordResetLink(@org.springframework.web.bind.annotation.RequestParam java.lang.String email, jakarta.servlet.http.HttpServletRequest request) throws java.net.URISyntaxException

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/challenges/challenge7/Assignment7.java:74#96

```
71
72 @PostMapping("/challenge/7")
73 @ResponseBody
74 public AttackResult sendPasswordResetLink(@RequestParam String email,
HttpServletRequest request)
75     throws URISyntaxException {
76   if (StringUtils.hasText(email)) {
77     String username = email.substring(0, email.indexOf("@"));
78 ...
79     }
80   }
81   return success(this).feedback("email.send").feedbackArgs(email).build();
82 }
83
84 @GetMapping(value = "/challenge/7/.git", produces = MediaType.
APPLICATION_OCTET_STREAM_VALUE)
```

```
85 @ResponseBody
```

RestTemplate.postForEntity()

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/challenges/challenge7/Assignment7.java:92

```
89          .recipient(username)
90          .time(LocalDateTime.now())
91          .build();

92     restTemplate.postForEntity(webWolfMailURL, mail, Object.class);

93   }
94 }
95 return success(this).feedback("email.send").feedbackArgs(email).build();
```

A2

A4

A5

External information leak (JavaScript)

Description

System configuration information leak is possible. This can help an attacker to plan an attack.
Debug information and error messages can be written to the log, displayed to the console, or sent to the user depending on the system settings. In some cases, an attacker can make a conclusion about the system vulnerabilities from the error message. For example, a database error can indicate insecurity against SQL injection attacks. Information about the version of the operating system, application server and system configuration can also be of value to the attacker.
In this case, we are talking about the external leak: information about the system is transferred to another machine over the network. External leaks are more dangerous than internal ones.
This applies both to web applications and mobile applications.

Example

In the following example, the application allows external information leakage by outputting an error to the web console:
console.log(error);
The error message in this case may contain information about the operating system, installed applications and system configuration features.

Recommendations

- Exclude detailed information about the system and its configuration from the error messages.
- If possible, do not send error messages using broadcast messaging.

Links

1. OWASP: Top 10 2017-A3-Sensitive Data Exposure
2. OWASP: Top 10 2017-A6-Security Misconfiguration
3. OWASP: Top 10 2013-A5-Security Misconfiguration
4. OWASP: Top 10 2013-A6-Sensitive Data Exposure
5. CWE-497: Exposure of System Data to an Unauthorized Control Sphere
6. CWE CATEGORY: OWASP Top Ten 2017 Category A5 - Broken Access Control
7. CWE CATEGORY: OWASP Top Ten 2017 Category A6 - Security Misconfiguration
8. CWE-200: Exposure of Sensitive Information to an Unauthorized Actor
9. CWE-209: Generation of Error Message Containing Sensitive Information

Vulnerability Entries

WebGoat-main/src/main/resources/webgoat/static/js/libs/ace.js:19459

Level	Medium
Status	Confirmed

```
19456       this.$worker.postMessage({event: event, data: {data: data.data}});
19457     }
19458     catch(ex) {
19459       console.error(ex.stack);
19460     }
19461 };
19462
```

03

About OWASP Top 10 2021

Report classifies the level of vulnerability by **OWASP Top 10 2021**. The Open Web Application Security Project (OWASP) is an online community which creates freely-available articles, methodologies, documentation, tools, and technologies in the field of web application security. One of its main projects is OWASP Top 10 aiming to raise awareness about application security by identifying some of the most critical risks that organizations face. The Top 10 project is referenced by many standards, books, tools, and organizations, including MITRE, PCI DSS, DISA, FTC.

Note that some vulnerabilities may belong to the number of categories or to none at all. To see the full list of vulnerabilities, choose the **By severity** classification method.

A1

Broken Access Control

Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits.

A2

Cryptographic Failures

Numerous web applications do not properly protect sensitive data falling under privacy laws or regulations, such as passwords, credit card numbers, health records, or personal information. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.

A3

Injection

Injection flaws (such as SQL, NoSQL, OS command, ORM, LDAP, and EL or OGNL) occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing

A4

Insecure Design

Insecure design is a broad category representing different weaknesses, expressed as “missing or ineffective control design”. An insecure design cannot be fixed by a perfect implementation as by definition, needed security controls were never created to defend data without proper authorization. User-supplied data must always be validated, filtered, or sanitized by the application.

against specific attacks. One of the factors that contribute to insecure design is the lack of business risk profiling inherent in the software or system being developed, and thus the failure to determine what level of security design is required.

A5

Security Misconfiguration

Security misconfiguration can happen at any level of an application stack, including the network services, platform, web server, application server, database, frameworks, custom code, and pre-installed virtual machines, containers, or storage. To ensure security, a proper configuration is essential as well as regular updates.

A6

Vulnerable and Outdated Components

Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using outdated components or components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.

A7

Identification and Authentication Failures

Confirmation of the user's identity, authentication, and session management is critical to protect against authentication-related attacks. Authentication flaws enable attackers to compromise passwords, keys, or session tokens, or assume other users’ identities.

A8

Software and Data Integrity Failures

Software and data integrity failures relate to code and infrastructure that does not protect against integrity violations. An example of this is where an application relies upon plugins, libraries, or modules from untrusted sources, repositories, and content delivery networks (CDNs). An insecure CI/CD pipeline can introduce the potential for unauthorized access, malicious code, or system compromise. Lastly, many applications now include auto-update functionality, where updates are downloaded without sufficient integrity verification and applied to the previously trusted application. Attackers could potentially upload their own updates to be distributed and run on all installations. Another example is where objects or data are encoded or serialized into a structure that an attacker can see and modify is vulnerable to insecure deserialization.

A9

Security Logging and Monitoring Failures

Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract or destroy data.

A10

Server-Side Request Forgery (SSRF)

SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL).