Hewlett Packard
Enterprise

HPE Security Fortify Audit Workbench

Developer Workbook

com.drajer.ecrnowais-ecr-now_Trunk_2020-12-07 - Trunk

FORTIFY®

Table of Contents

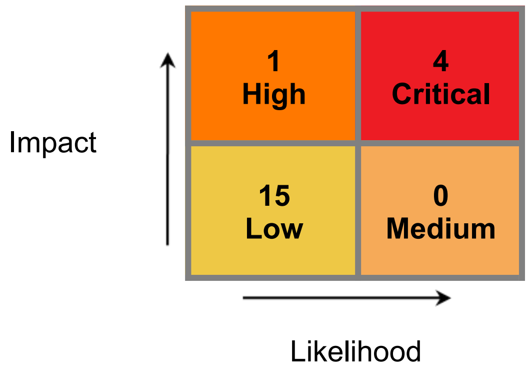
Executive Summary

This workbook is intended to provide all necessary details and information for a developer to understand and remediate the different issues discovered during the com.drajer.ecrnowais-ecr-now_Trunk_2020-12-07 - Trunk project audit. The information contained in this workbook is targeted at project managers and developers.

This section provides an overview of the issues uncovered during analysis.

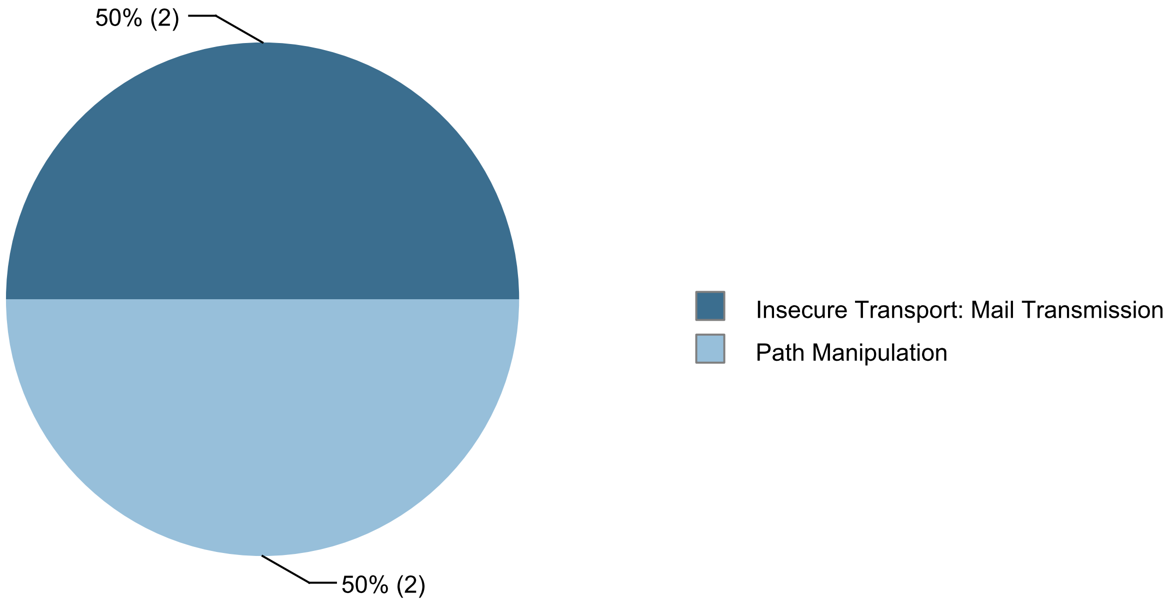
Project Name:	com.drajer.ecrnowais-ec now_Trunk_2020-12-07
Project Version:	Trunk
SCA:	Results Present
WebInspect:	Results Not Present
WebInspect Agent:	Results Not Present
Other:	Results Not Present

Issues by Priority



Impact		
	1 High	4 Critical
	15 Low	0 Medium
	Likelihood	

Top Ten Critical Categories



Project Description

This section provides an overview of the HPE Security Fortify scan engines used for this project, as well as the project meta-information.

SCA

Date of Last Analysis:	Dec 6, 2020, 9:12 AM	**Engine Version:**	17.20.0183
Host Name:	USMLVV3CTO0086	**Certification:**	VALID
Number of Files:	131	**Lines of Code:**	15,004

Issue Breakdown by Fortify Categories

The following table depicts a summary of all issues grouped vertically by Fortify Category. For each category, the total number of issues is shown by Fortify Priority Order, including information about the number of audited issues.

Category	Fortify Priority (audited/total)				Total Issues
	Critical	High	Medium	Low	
Header Manipulation	0	0 / 1	0	0 / 1	0 / 2
Insecure Transport: Mail Transmission	0 / 2	0	0	0	0 / 2
Path Manipulation	0 / 2	0	0	0	0 / 2
Poor Error Handling: Overly Broad Throws	0	0	0	0 / 6	0 / 6
Redundant Null Check	0	0	0	0 / 8	0 / 8

Results Outline

Header Manipulation (2 issues)

Abstract

Including unvalidated data in an HTTP response header can enable cache-poisoning, cross-site scripting, cross-user defacement, page hijacking, cookie manipulation or open redirect.

Explanation

Header Manipulation vulnerabilities occur when: 1. Data enters a web application through an untrusted source, most frequently an HTTP request. 2. The data is included in an HTTP response header sent to a web user without being validated. As with many software security vulnerabilities, Header Manipulation is a means to an end, not an end in itself. At its root, the vulnerability is straightforward: an attacker passes malicious data to a vulnerable application, and the application includes the data in an HTTP response header. One of the most common Header Manipulation attacks is HTTP Response Splitting. To mount a successful HTTP Response Splitting exploit, the application must allow input that contains CR (carriage return, also given by %0d or \r) and LF (line feed, also given by %0a or \n)characters into the header. These characters not only give attackers control of the remaining headers and body of the response the application intends to send, but also allows them to create additional responses entirely under their control. Many of today's modern application servers will prevent the injection of malicious characters into HTTP headers. For example, recent versions of Apache Tomcat will throw an `IllegalArgumentException` if you attempt to set a header with prohibited characters. If your application server prevents setting headers with new line characters, then your application is not vulnerable to HTTP Response Splitting. However, solely filtering for new line characters can leave an application vulnerable to Cookie Manipulation or Open Redirects, so care must still be taken when setting HTTP headers with user input. **Example:** The following code segment reads the name of the author of a weblog entry, `author`, from an HTTP request and sets it in a cookie header of an HTTP response.

```
String author = request.getParameter(AUTHOR_PARAM);
...
Cookie cookie = new Cookie("author", author);
     cookie.setMaxAge(cookieExpiration);
     response.addCookie(cookie);
```

Assuming a string consisting of standard alpha-numeric characters, such as "Jane Smith", is submitted in the request the HTTP response including this cookie might take the following form:

```
HTTP/1.1 200 OK
...
Set-Cookie: author=Jane Smith
...
```

However, because the value of the cookie is formed of unvalidated user input the response will only maintain this form if the value submitted for `AUTHOR_PARAM` does not contain any CR and LF characters. If an attacker submits a malicious string, such as "Wiley Hacker\r\nHTTP/1.1 200 OK\r\n...", then the HTTP response would be split into two responses of the following form:

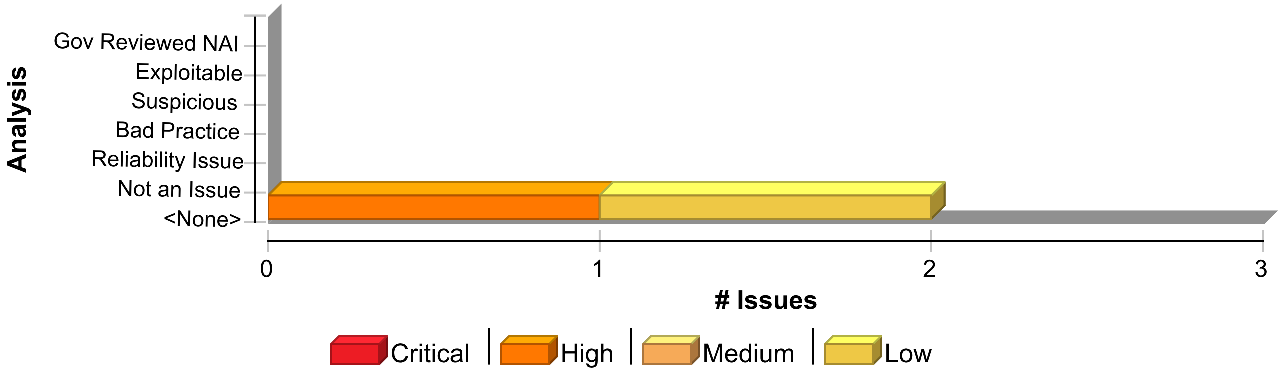
```
HTTP/1.1 200 OK
...
Set-Cookie: author=Wiley Hacker

HTTP/1.1 200 OK
...
```

Clearly, the second response is completely controlled by the attacker and can be constructed with any header and body content desired. The ability of attacker to construct arbitrary HTTP responses permits a variety of resulting attacks, including: cross-user defacement, web and browser cache poisoning, cross-site scripting and page hijacking. **Cross-User Defacement:** An attacker will be able to make a single request to a vulnerable server that will cause the server to create two responses, the second of which may be misinterpreted as a response to a different request, possibly one made by another user sharing the same TCP connection with the server. This can be accomplished by convincing the user to submit the malicious request themselves, or remotely in situations where the attacker and the user share a common TCP connection to the server, such as a shared proxy server. In the best case, an attacker may leverage this ability to convince users that the application has been hacked, causing users to lose confidence in the security of the application. In the worst case, an attacker may provide specially crafted content designed to mimic the behavior of the application but redirect private information, such as account numbers and passwords, back to the attacker. **Cache Poisoning:** The impact of a maliciously constructed response can be magnified if it is cached either by a web cache used by multiple users or even the browser cache of a single user. If a response is cached in a shared web cache, such as those commonly found in proxy servers, then all users of that cache will continue receive the malicious content until the cache entry is purged. Similarly, if the response is cached in the browser of an individual user, then that user will continue to receive the malicious content until the cache entry is purged, although only the user of the local browser instance will be affected. **Cross-Site Scripting:** Once attackers have control of the responses sent by an application, they have a choice of a variety of malicious content to provide users. Cross-site scripting is common form of attack where malicious JavaScript or other code included in a response is executed in the user's browser. The variety of attacks based on XSS is almost limitless, but they commonly include transmitting private data like cookies or other session information to the attacker, redirecting the victim to web content controlled by the attacker, or performing other malicious operations on the user's machine under the guise of the vulnerable site. The most common and dangerous attack vector against users of a vulnerable application uses JavaScript to transmit session and authentication information back to the attacker who can then take complete control of the victim's account. **Page Hijacking:** In addition to using a vulnerable application to send malicious content to a user, the same root vulnerability can also be leveraged to redirect sensitive content generated by the server and intended for the user to the attacker instead. By submitting a request that results in two responses, the intended response from the server and the response generated by the attacker, an attacker may cause an intermediate node, such as a shared proxy server, to misdirect a response generated by the server for the user to the attacker. Because the request made by the attacker generates two responses, the first is interpreted as a response to the attacker's request, while the second remains in limbo. When the user makes a legitimate request through the same TCP connection, the attacker's request is already waiting and is interpreted as a response to the victim's request. The attacker then sends a second request to the server, to which the proxy server responds with the server generated request intended for the victim, thereby compromising any sensitive information in the headers or body of the response intended for the victim. **Cookie Manipulation:** When combined with attacks like Cross-Site Request Forgery, attackers may change, add to, or even overwrite a legitimate user's cookies. **Open Redirect:** Allowing unvalidated input to control the URL used in a redirect can aid phishing attacks.

Recommendation

The solution to Header Manipulation is to ensure that input validation occurs in the correct places and checks for the correct properties. Since Header Manipulation vulnerabilities occur when an application includes malicious data in its output, one logical approach is to validate data immediately before it leaves the application. However, because web applications often have complex and intricate code for generating responses dynamically, this method is prone to errors of omission (missing validation). An effective way to mitigate this risk is to also perform input validation for Header Manipulation. Web applications must validate their input to prevent other vulnerabilities, such as SQL injection, so augmenting an application's existing input validation mechanism to include checks for Header Manipulation is generally relatively easy. Despite its value, input validation for Header Manipulation does not take the place of rigorous output validation. An application may accept input through a shared data store or other trusted source, and that data store may accept input from a source that does not perform adequate input validation. Therefore, the application cannot implicitly rely on the safety of this or any other data. This means the best way to prevent Header Manipulation vulnerabilities is to validate everything that enters the application or leaves the application destined for the user. The most secure approach to validation for Header Manipulation is to create a whitelist of safe characters that are allowed to appear in HTTP response headers and accept input composed exclusively of characters in the approved set. For example, a valid name might only include alpha-numeric characters or an account number might only include digits 0-9. A more flexible, but less secure approach is known as blacklisting, which selectively rejects or escapes potentially dangerous characters before using the input. In order to form such a list, you first need to understand the set of characters that hold special meaning in HTTP response headers. Although the CR and LF characters are at the heart of an HTTP response splitting attack, other characters, such as ':' (colon) and '=' (equal), have special meaning in response headers as well. After you identify the correct points in an application to perform validation for Header Manipulation attacks and what special characters the validation should consider, the next challenge is to identify how your validation handles special characters. The application should reject any input destined to be included in HTTP response headers that contains special characters, particularly CR and LF, as invalid. Many application servers attempt to limit an application's exposure to HTTP response splitting vulnerabilities by providing implementations for the functions responsible for setting HTTP headers and cookies that perform validation for the characters essential to an HTTP response splitting attack. Do not rely on the server running your application to make it secure. When an application is developed there are no guarantees about what application servers it will run on during its lifetime. As standards and known exploits evolve, there are no guarantees that application servers will also stay in sync.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Header Manipulation	2	0	0	2
Total	**2**	**0**	**0**	**2**

Header Manipulation — High

Package: com.drajer.sof.launch

sof/launch/LaunchController.java, line 303 (Header Manipulation) — High

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: javax.servlet.ServletRequest.getScheme()
From: com.drajer.sof.launch.LaunchController.launchApp
File: sof/launch/LaunchController.java:253

```
250  logger.info("Received Launch Parameter::::: {}", launch);
251  logger.info("Received FHIR Server Base URL:::: {}", iss);
```

Header Manipulation — High

Package: com.drajer.sof.launch

sof/launch/LaunchController.java, line 303 (Header Manipulation) — High

```
252  String uri =
253  request.getScheme()
254  + "://"
255  + request.getServerName()
256  + ("http".equals(request.getScheme()) && request.getServerPort() == 80
```

Sink Details

Sink: javax.servlet.http.HttpServletResponse.setHeader()
Enclosing Method: launchApp()
File: sof/launch/LaunchController.java:303
Taint Flags: WEB

```
300  authDetailsService.saveOrUpdate(launchDetails);
301  // response.sendRedirect(constructedAuthUrl);
302  response.setStatus(HttpServletResponse.SC_TEMPORARY_REDIRECT);
303  response.setHeader("Location", constructedAuthUrl);
304  }
305  } catch (Exception e) {
306  logger.error("Error in getting Authorization with Server");
```

Header Manipulation — Low

Package: com.drajer.sof.launch

sof/launch/LaunchController.java, line 303 (Header Manipulation) — Low

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: javax.servlet.ServletRequest.getServerPort()
From: com.drajer.sof.launch.LaunchController.launchApp
File: sof/launch/LaunchController.java:259

```
256  + ("http".equals(request.getScheme()) && request.getServerPort() == 80
257  || "https".equals(request.getScheme()) && request.getServerPort() == 443
258  ? ""
259  : ":" + request.getServerPort())
260  + request.getContextPath();
261  Integer state = random.nextInt();
262  logger.info("Random State Value==========> {}", state);
```

Sink Details

Sink: javax.servlet.http.HttpServletResponse.setHeader()

Header Manipulation — Low

Package: com.drajer.sof.launch

sof/launch/LaunchController.java, line 303 (Header Manipulation) — Low

Enclosing Method: launchApp()
File: sof/launch/LaunchController.java:303
Taint Flags: NUMBER, WEB

```
300  authDetailsService.saveOrUpdate(launchDetails);
301  // response.sendRedirect(constructedAuthUrl);
302  response.setStatus(HttpServletResponse.SC_TEMPORARY_REDIRECT);
303  response.setHeader("Location", constructedAuthUrl);
304  }
305  } catch (Exception e) {
306  logger.error("Error in getting Authorization with Server");
```

Insecure Transport: Mail Transmission (2 issues)

Abstract

Establishing an unencrypted connection to a mail server allows an attacker to carry out a man-in-the-middle attack and read all the mail transmissions.

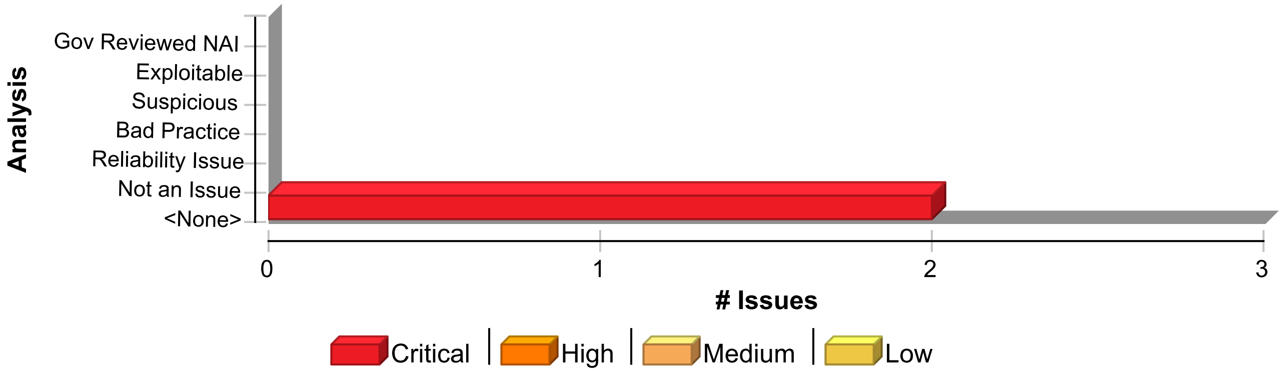
Explanation

Sensitive data sent over the wire unencrypted is subject to be read/modified by any attacker that can intercept the network traffic.

Recommendation

Most of the modern mail service providers offer encrypted alternatives on different ports that use SSL/TLS to encrypt all the data being sent over the wire or to upgrade an existing unencrypted connection to SSL/TLS. Always use these alternatives when possible.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Insecure Transport: Mail Transmission	2	0	0	2
Total	**2**	**0**	**0**	**2**

Insecure Transport: Mail Transmission	Critical
Package: com.drajer.routing.impl	
routing/impl/DirectResponseReceiver.java, line 67 (Insecure Transport: Mail Transmission)	**Critical**

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: getStore
Enclosing Method: readMail()
File: routing/impl/DirectResponseReceiver.java:67

Insecure Transport: Mail Transmission — Critical

Package: com.drajer.routing.impl

routing/impl/DirectResponseReceiver.java, line 67 (Insecure Transport: Mail Transmission) — Critical

Taint Flags:

```
64  Properties props = new Properties();
65  Session session = Session.getInstance(props, null);
66
67  Store store = session.getStore("imap");
68  int port = 143; // Integer.parseInt(prop.getProperty("port"));
69  logger.info("Connecting to IMAP Inbox");
70  store.connect(details.getDirectHost(), port, details.getDirectUser(),
details.getDirectPwd());
```

routing/impl/DirectResponseReceiver.java, line 134 (Insecure Transport: Mail Transmission) — Critical

Issue Details

Kingdom: Security Features
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: getStore
Enclosing Method: deleteMail()
File: routing/impl/DirectResponseReceiver.java:134
Taint Flags:

```
131  Properties props = new Properties();
132  Session session = Session.getInstance(props, null);
133
134  Store store = session.getStore("imap");
135  int port = 143; // Integer.parseInt(prop.getProperty("port"));
136  store.connect(host, username, password);
137
```

Path Manipulation (2 issues)

Abstract

Allowing user input to control paths used in file system operations could enable an attacker to access or modify otherwise protected system resources.

Explanation

Path manipulation errors occur when the following two conditions are met: 1. An attacker is able to specify a path used in an operation on the file system. 2. By specifying the resource, the attacker gains a capability that would not otherwise be permitted. For example, the program may give the attacker the ability to overwrite the specified file or run with a configuration controlled by the attacker. **Example 1:** The following code uses input from an HTTP request to create a file name. The programmer has not considered the possibility that an attacker could provide a file name such as "`../../tomcat/conf/server.xml`", which causes the application to delete one of its own configuration files.

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

Example 2: The following code uses input from a configuration file to determine which file to open and echo back to the user. If the program runs with adequate privileges and malicious users can change the configuration file, they can use the program to read any file on the system that ends with the extension `.txt`.

```
fis = new FileInputStream(cfg.getProperty("sub")+".txt");
amt = fis.read(arr);
out.println(arr);
```

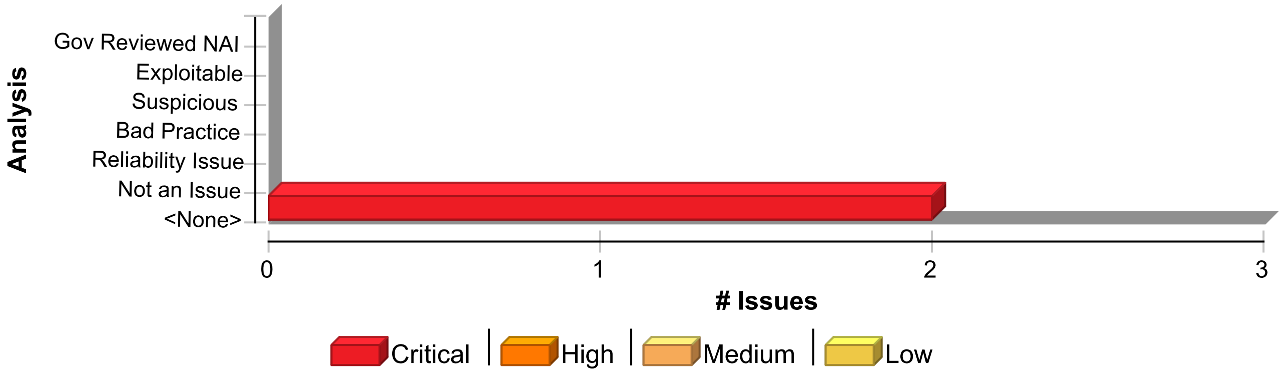
Some think that in the mobile world, classic vulnerabilities, such as path manipulation, do not make sense -- why would the user attack themself? However, keep in mind that the essence of mobile platforms is applications that are downloaded from various sources and run alongside each other on the same device. The likelihood of running a piece of malware next to a banking application is high, which necessitates expanding the attack surface of mobile applications to include inter-process communication. **Example 3:** The following code adapts Example 1 to the Android platform.

```
...
        String rName = this.getIntent().getExtras().getString("reportName");
        File rFile = getBaseContext().getFileStreamPath(rName);
...
        rFile.delete();
...
```

Recommendation

The best way to prevent path manipulation is with a level of indirection: create a list of legitimate resource names that a user is allowed to specify, and only allow the user to select from the list. With this approach the input provided by the user is never used directly to specify the resource name. In some situations this approach is impractical because the set of legitimate resource names is too large or too hard to keep track of. Programmers often resort to blacklisting in these situations. Blacklisting selectively rejects or escapes potentially dangerous characters before using the input. However, any such list of unsafe characters is likely to be incomplete and will almost certainly become out of date. A better approach is to create a whitelist of characters that are allowed to appear in the resource name and accept input composed exclusively of characters in the approved set.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Path Manipulation	2	0	0	2
Total	**2**	**0**	**0**	**2**

Path Manipulation — Critical

Package: com.drajer.routing.impl

routing/impl/DirectResponseReceiver.java, line 107 (Path Manipulation) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: javax.mail.Store.getFolder()
From: com.drajer.routing.impl.DirectResponseReceiver.readMail
File: routing/impl/DirectResponseReceiver.java:72

```
69  logger.info("Connecting to IMAP Inbox");
70  store.connect(details.getDirectHost(), port, details.getDirectUser(),
details.getDirectPwd());
71
72  Folder inbox = store.getFolder("Inbox");
73  inbox.open(Folder.READ_WRITE);
74
75  Flags seen = new Flags(Flags.Flag.SEEN);
```

Sink Details

Sink: java.io.File.File()
Enclosing Method: readMail()
File: routing/impl/DirectResponseReceiver.java:107
Taint Flags: NETWORK, XSS

```
104
105  try (InputStream stream = bodyPart.getInputStream()) {
106  byte[] targetArray = IOUtils.toByteArray(stream);
107  FileUtils.writeByteArrayToFile(new File(filename), targetArray);
```

Path Manipulation — Critical

Package: com.drajer.routing.impl

routing/impl/DirectResponseReceiver.java, line 107 (Path Manipulation) — Critical

```
108  }
109  File file1 = new File(filename);
110  FileBody fileBody = new FileBody(file1);
```

routing/impl/DirectResponseReceiver.java, line 109 (Path Manipulation) — Critical

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: javax.mail.Store.getFolder()
From: com.drajer.routing.impl.DirectResponseReceiver.readMail
File: routing/impl/DirectResponseReceiver.java:72

```
69  logger.info("Connecting to IMAP Inbox");
70  store.connect(details.getDirectHost(), port, details.getDirectUser(),
details.getDirectPwd());
71
72  Folder inbox = store.getFolder("Inbox");
73  inbox.open(Folder.READ_WRITE);
74
75  Flags seen = new Flags(Flags.Flag.SEEN);
```

Sink Details

Sink: java.io.File.File()
Enclosing Method: readMail()
File: routing/impl/DirectResponseReceiver.java:109
Taint Flags: NETWORK, XSS

```
106  byte[] targetArray = IOUtils.toByteArray(stream);
107  FileUtils.writeByteArrayToFile(new File(filename), targetArray);
108  }
109  File file1 = new File(filename);
110  FileBody fileBody = new FileBody(file1);
111
112  logger.info(
```

Poor Error Handling: Overly Broad Throws (6 issues)

Abstract

The method throws a generic exception making it harder for callers to do a good job of error handling and recovery.

Explanation

Declaring a method to throw `Exception` or `Throwable` makes it difficult for callers to do good error handling and error recovery. Java's exception mechanism is set up to make it easy for callers to anticipate what can go wrong and write code to handle each specific exceptional circumstance. Declaring that a method throws a generic form of exception defeats this system. **Example:** The following method throws three types of exceptions.

```
public void doExchange()
  throws IOException, InvocationTargetException,
         SQLException {
  ...
}
```

While it might seem tidier to write

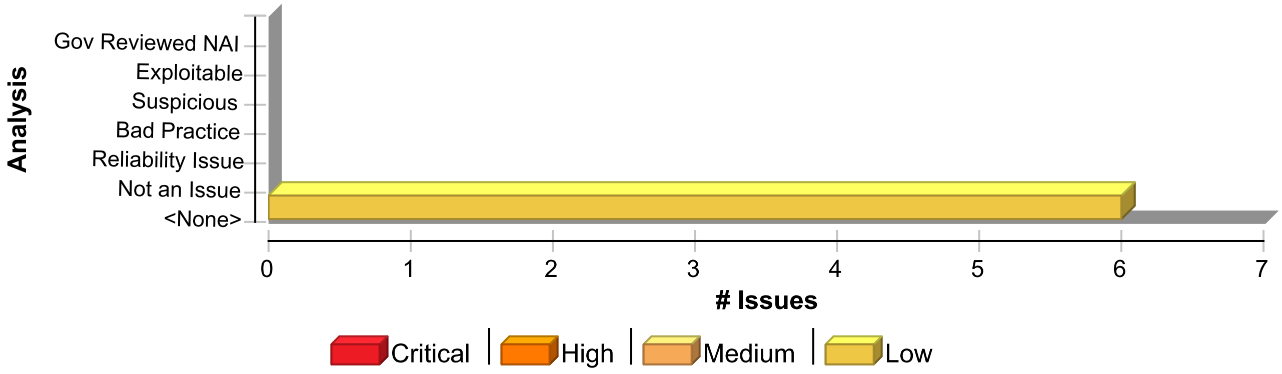
```
public void doExchange()
  throws Exception {
  ...
}
```

doing so hampers the caller's ability to understand and handle the exceptions that occur. Further, if a later revision of `doExchange()` introduces a new type of exception that should be treated differently than previous exceptions, there is no easy way to enforce this requirement.

Recommendation

Do not declare methods to throw `Exception` or `Throwable`. If the exceptions thrown by a method are not recoverable or should not generally be caught by the caller, consider throwing unchecked exceptions rather than checked exceptions. This can be accomplished by implementing exception classes that extend `RuntimeException` or `Error` instead of `Exception`, or add a try/catch wrapper in your method to convert checked exceptions to unchecked exceptions.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Poor Error Handling: Overly Broad Throws	6	0	0	6
Total	**6**	**0**	**0**	**6**

Poor Error Handling: Overly Broad Throws — Low

Package: com.drajer.ecrapp.config

ecrapp/config/WebSecurityConfig.java, line 24 (Poor Error Handling: Overly Broad Throws) — Low

Issue Details

Kingdom: Errors
Scan Engine: SCA (Structural)

Sink Details

Sink: Function: configure
Enclosing Method: configure()
File: ecrapp/config/WebSecurityConfig.java:24
Taint Flags:

```
21  private String tokenFilterClassName;
22
23  @Override
24  public void configure(WebSecurity web) throws Exception {
25  web.ignoring().antMatchers("/meta/**");
26  }
27
```

ecrapp/config/WebSecurityConfig.java, line 29 (Poor Error Handling: Overly Broad Throws) — Low

Issue Details

Kingdom: Errors
Scan Engine: SCA (Structural)

Sink Details

Sink: Function: configure
Enclosing Method: configure()
File: ecrapp/config/WebSecurityConfig.java:29
Taint Flags:

```
26  }
27
28  @Override
29  protected void configure(HttpSecurity http) throws Exception {
30  logger.info("*******************************************************************");
31  logger.info("Security Configuration" + tokenFilterClassName);
32  logger.info("*******************************************************************");
```

Poor Error Handling: Overly Broad Throws

Low

Package: com.drajer.routing.impl

routing/impl/DirectEicrSender.java, line 72 (Poor Error Handling: Overly Broad Throws)

Low

Issue Details

Kingdom: Errors
Scan Engine: SCA (Structural)

Sink Details

Sink: Function: sendMail
Enclosing Method: sendMail()
File: routing/impl/DirectEicrSender.java:72
Taint Flags:

```
69  }
70  }
71
72  public void sendMail(
73  String host,
74  String username,
75  String password,
```

routing/impl/DirectResponseReceiver.java, line 129 (Poor Error Handling: Overly Broad Throws)

Low

Issue Details

Kingdom: Errors
Scan Engine: SCA (Structural)

Sink Details

Sink: Function: deleteMail
Enclosing Method: deleteMail()
File: routing/impl/DirectResponseReceiver.java:129
Taint Flags:

```
126  }
127  }
128
129  public void deleteMail(String host, String username, String password) throws Exception {
130
131  Properties props = new Properties();
132  Session session = Session.getInstance(props, null);
```

Package: com.drajer.sof.launch

sof/launch/LaunchController.java, line 243 (Poor Error Handling: Overly Broad Throws)

Low

Issue Details

Kingdom: Errors

Poor Error Handling: Overly Broad Throws — Low

Package: com.drajer.sof.launch

sof/launch/LaunchController.java, line 243 (Poor Error Handling: Overly Broad Throws) — Low

Scan Engine: SCA (Structural)

Sink Details

Sink: Function: launchApp
Enclosing Method: launchApp()
File: sof/launch/LaunchController.java:243
Taint Flags:

```
240
241  @CrossOrigin
242  @RequestMapping(value = "/api/launch")
243  public void launchApp(
244  @RequestParam String launch,
245  @RequestParam String iss,
246  HttpServletRequest request,
```

sof/launch/LaunchController.java, line 315 (Poor Error Handling: Overly Broad Throws) — Low

Issue Details

Kingdom: Errors
Scan Engine: SCA (Structural)

Sink Details

Sink: Function: redirectEndPoint
Enclosing Method: redirectEndPoint()
File: sof/launch/LaunchController.java:315
Taint Flags:

```
312
313  @CrossOrigin
314  @RequestMapping(value = "/api/redirect")
315  public void redirectEndPoint(
316  @RequestParam String code,
317  @RequestParam String state,
318  HttpServletRequest request,
```

Redundant Null Check (8 issues)

Abstract

The program can potentially dereference a null pointer, thereby causing a null pointer exception.

Explanation

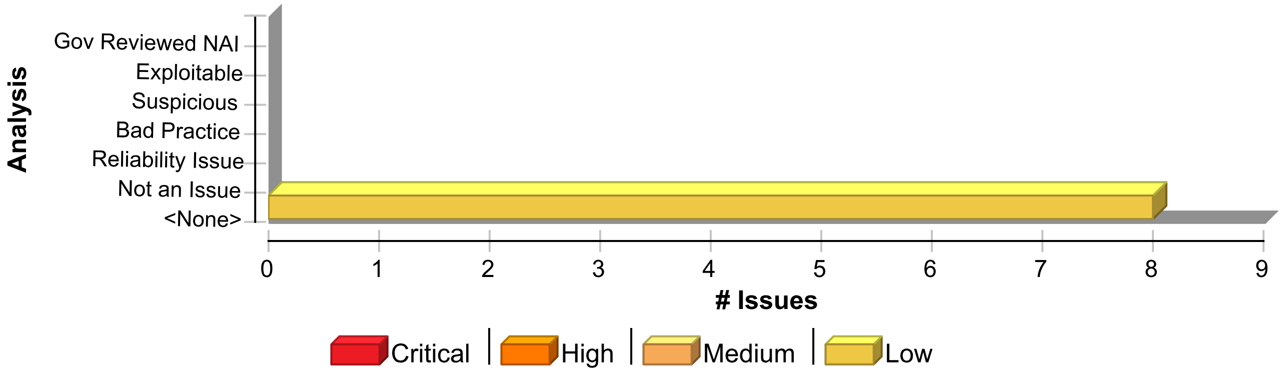
Null pointer exceptions usually occur when one or more of the programmer's assumptions is violated. A check-after-dereference error occurs when a program dereferences an object that can be `null` before checking if the object is `null`. Most null pointer issues result in general software reliability problems, but if attackers can intentionally trigger a null pointer dereference, they can use the resulting exception to bypass security logic or to cause the application to reveal debugging information that will be valuable in planning subsequent attacks. **Example:** In the following code, the programmer assumes that the variable `foo` is not `null` and confirms this assumption by dereferencing the object. However, the programmer later contradicts the assumption by checking `foo` against `null`. If `foo` can be `null` when it is checked in the `if` statement then it can also be `null` when it is dereferenced and might cause a null pointer exception. Either the dereference is unsafe or the subsequent check is unnecessary.

```
foo.setBar(val);
...
if (foo != null) {
    ...
}
```

Recommendation

Implement careful checks before dereferencing objects that might be `null`. When possible, abstract null checks into wrappers around code that manipulates resources to ensure that they are applied in all cases and to minimize the places where mistakes can occur.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Redundant Null Check	8	0	0	8
Total	**8**	**0**	**0**	**8**

Redundant Null Check — Low

Package: com.drajer.sof.service

sof/service/LoadingQueryDstu2Bundle.java, line 255 (Redundant Null Check) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Control Flow)

Sink Details

Sink: Dereferenced : medication
Enclosing Method: createDSTU2Bundle()
File: sof/service/LoadingQueryDstu2Bundle.java:255
Taint Flags:

```
252  Medication medication =
253  dstu2ResourcesData.getMedicationData(
254  context, client, launchDetails, dstu2FhirData, medReference);
255  Entry medicationEntry = new Entry().setResource(medication);
256  bundle.addEntry(medicationEntry);
257  if (medication != null) {
258  List<Medication> medicationList = new ArrayList<Medication>();
```

sof/service/TriggerQueryDstu2Bundle.java, line 221 (Redundant Null Check) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Control Flow)

Sink Details

Sink: Dereferenced : medication
Enclosing Method: createDSTU2Bundle()
File: sof/service/TriggerQueryDstu2Bundle.java:221
Taint Flags:

```
218  Medication medication =
219  dstu2ResourcesData.getMedicationData(
220  context, client, launchDetails, dstu2FhirData, medReference);
221  Entry medicationEntry = new Entry().setResource(medication);
222  bundle.addEntry(medicationEntry);
223  if (medication != null) {
224  List<Medication> medicationList = new ArrayList<Medication>();
```

sof/service/LoadingQueryDstu2Bundle.java, line 255 (Redundant Null Check) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Control Flow)

Sink Details

Sink: Dereferenced : medication

Redundant Null Check — Low

Package: com.drajer.sof.service

sof/service/LoadingQueryDstu2Bundle.java, line 255 (Redundant Null Check) — Low

Enclosing Method: createDSTU2Bundle()
File: sof/service/LoadingQueryDstu2Bundle.java:255
Taint Flags:

```
252  Medication medication =
253  dstu2ResourcesData.getMedicationData(
254  context, client, launchDetails, dstu2FhirData, medReference);
255  Entry medicationEntry = new Entry().setResource(medication);
256  bundle.addEntry(medicationEntry);
257  if (medication != null) {
258  List<Medication> medicationList = new ArrayList<Medication>();
```

sof/service/TriggerQueryDstu2Bundle.java, line 221 (Redundant Null Check) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Control Flow)

Sink Details

Sink: Dereferenced : medication
Enclosing Method: createDSTU2Bundle()
File: sof/service/TriggerQueryDstu2Bundle.java:221
Taint Flags:

```
218  Medication medication =
219  dstu2ResourcesData.getMedicationData(
220  context, client, launchDetails, dstu2FhirData, medReference);
221  Entry medicationEntry = new Entry().setResource(medication);
222  bundle.addEntry(medicationEntry);
223  if (medication != null) {
224  List<Medication> medicationList = new ArrayList<Medication>();
```

Package: com.drajer.sof.utils

sof/utils/R4ResourcesData.java, line 948 (Redundant Null Check) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Control Flow)

Sink Details

Sink: Dereferenced : medication
Enclosing Method: getCommonResources()
File: sof/utils/R4ResourcesData.java:948
Taint Flags:

```
945  Medication medication =
946  getMedicationData(context, client, launchDetails, r4FhirData, medReference);
```

Redundant Null Check — Low

Package: com.drajer.sof.utils

sof/utils/R4ResourcesData.java, line 948 (Redundant Null Check) — Low

```
947  BundleEntryComponent medicationEntry =
948  new BundleEntryComponent().setResource(medication);
949  bundle.addEntry(medicationEntry);
950  if (medication != null) {
951  List<Medication> medicationList = new ArrayList<>();
```

sof/utils/R4ResourcesData.java, line 992 (Redundant Null Check) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Control Flow)

Sink Details

Sink: Dereferenced : medication
Enclosing Method: getCommonResources()
File: sof/utils/R4ResourcesData.java:992
Taint Flags:

```
989  Medication medication =
990  getMedicationData(context, client, launchDetails, r4FhirData, medReference);
991  BundleEntryComponent medicationEntry =
992  new BundleEntryComponent().setResource(medication);
993  bundle.addEntry(medicationEntry);
994  if (medication != null) {
995  List<Medication> medicationList = new ArrayList<Medication>();
```

sof/utils/R4ResourcesData.java, line 948 (Redundant Null Check) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Control Flow)

Sink Details

Sink: Dereferenced : medication
Enclosing Method: getCommonResources()
File: sof/utils/R4ResourcesData.java:948
Taint Flags:

```
945  Medication medication =
946  getMedicationData(context, client, launchDetails, r4FhirData, medReference);
947  BundleEntryComponent medicationEntry =
948  new BundleEntryComponent().setResource(medication);
949  bundle.addEntry(medicationEntry);
950  if (medication != null) {
951  List<Medication> medicationList = new ArrayList<>();
```

Redundant Null Check

Low

Package: com.drajer.sof.utils

sof/utils/R4ResourcesData.java, line 992 (Redundant Null Check)

Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Control Flow)

Sink Details

Sink: Dereferenced : medication
Enclosing Method: getCommonResources()
File: sof/utils/R4ResourcesData.java:992
Taint Flags:

```
989  Medication medication =
990  getMedicationData(context, client, launchDetails, r4FhirData, medReference);
991  BundleEntryComponent medicationEntry =
992  new BundleEntryComponent().setResource(medication);
993  bundle.addEntry(medicationEntry);
994  if (medication != null) {
995  List<Medication> medicationList = new ArrayList<Medication>();
```