CRLF Injection

ISEC3004 - Group 21

Structure

The Team

1. Menick Thanthrige Sanugi Upenya Perera - 20542524
2. Pramoda Methmali Gunarathne - 20531218
3. Harshi Kasundi Bandaranayake - 20583387
4. Vishmi Chalanika Kalansooriya- 20532279
5. Pasindu Lakpriya Weerakoon - 20149925

CRLF - Defined

CRLF - Carriage Return Line Feeds (or more popularly known as /r and /n respectively) are used to signify termination of a line. Windows OS require both CR and LF to signify the end of a line whereas Linux/Unix will only require a LF to do so.

Whatever the action might be, whenever a user interacts with the Internet, the web browser sends requests to the web server. The web server then replies the browser with an HTTP response where the HTTP header and the actual website content will be separated by the CRLF characters. Carriage Return Line Feeds helps your browser understand where a new HTTP header may begin or end.

What is CRLF Injection Attack?

A CRLF injection attack exploits vulnerabilities at the application layer where an attacker attempts to inject CRLF character sequence into a web application via input methods such as URL, user input or HTTP parameters when it is not expected.

CRLF injection vulnerabilities often occurs when input data is,

- Not neutralized
- Incorrectly neutralized
- Unsanitized

Attackers inject intentionally crafted special CRLF text streams with malicious intent that can trick the vulnerable web application to perform potentially harmful actions resulting in security breaches, disclosure and/or corruption of valuable data.

There are 2 types of CRLF Injections...

HTTP response splitting

Easily exploitable when the server fails to properly sanitize user input and attackers can include their own CRLF character sequence, forcing the server to perform particular action or create different responses to send back to the user.

Log injection

Also known as log splitting and log poisoning. The attacker falsifies log files entries by adding extra lines of instructions after an EOL (End Of the Line). Log injections can be used to create corrupt data making it unusable, change data by making fake entries which may be used to divert attention from the real attack.

How to Detect CRLF Log Injection attack?

Because the log injection vulnerabilities will persistently store the malicious payload in the database or file on the web application server side, it will cause persistent harm to the web application.

- We can use LogInjector which is an effective method for detecting Log injection vulnerabilities.

- LogInjector detects web application log injection vulnerabilities through a dynamic testing approach based on feedback-guided mutation.

- LogInjector employs the extended crawler to locate log-injectable and view-logs interfaces.

- The goal of mutation is to allow injected code to retain execution semantics while also avoiding the web application's input-filter checks. We can add prefix, add suffix, tag transformation, attribute transformation and overall transformation to mutate.

How does CRLF Injection work ?

How log poisoning works.

There are several types of Log Injection attacks.

1.One way is to tamper with a log and render it useless, or the attacker may counterfeit a log and alter the contents to produce fake log entries.

2. The second way is to to initiate an XSS attack via the log when it is examined and this happens due to the vulnerabilities in a web application.

3. An attacker can also create a log injection by inserting commands that a parser could run after reading the log.

How does CRLF Injection work ?

How HTTP response splitting works in combination with Cross-site Scripting (XSS)

Step 1. The Attacker incorporates a false HTTP response header. As a result, the web browser interprets this as a response that has ended and starts parsing a new response.

Step 2. Creates a phony HTTP response. The new reaction starts at this point.

Step 3. Includes one more false HTTP response header. The web browser needs this to correctly parse the information.

Step 4. Adds one more false HTTP response header. Ex. Content Length : 25 -the web browser will only parse the following 25 bytes as a result.

Step 5. Inserts page content using an XSS.

Step 6. The web browser disregards the original material that originates from the web server due to the Content-Length header.

Exploitation - Demo

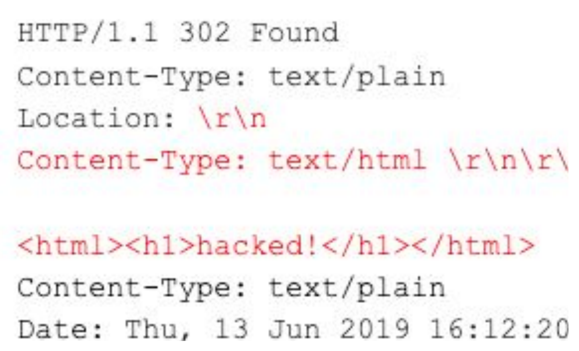
- The HTTP response splitting exploitation is done and the link is attached below.
- The exploitation for log injection attack is also attached with the solution but our main focus will be for HTTP response splitting.

Patching - Demo

- The patching for HTTP response splitting is done by using input validation.

Impacts of CRLF Attack

1. If the user/hacker successfully manipulates the 'Location Header' and make it empty and feed in carriage return and line feed as shown below:



```
HTTP/1.1 302 Found
Content-Type: text/plain
Location: \r\n
Content-Type: text/html \r\n\r\

<html><h1>hacked!</h1></html>
Content-Type: text/plain
Date: Thu, 13 Jun 2019 16:12:20
```

- The web browser/ client will ignore this header, implying that the web browser will parse the remaining headers. As a result, no redirect will be performed, indicating that the preceding attack was successful.
- Without proper sanitization of the location header values, we cannot filter characters such as \r and \n.

Impacts of CRLF Attack

2. The Set-Cookie header is used in HTTP responses to request that a cookie be saved by the browser. By using the set cookie value, a hacker can manipulate to set malicious cookies on the victim’s web browser.

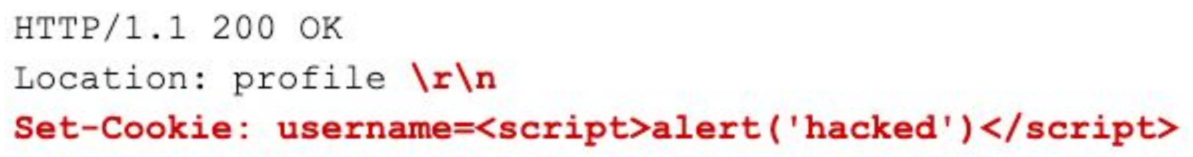
- This can lead to Session Fixation, Phishing attacks, Cross-site scripting attack and page Hijacking.

```
https://example.com/%0d%0aSet-Cookie%3A%20username%3D%3Cscript%3Ealert
(%27hacked%27)%3C%2Fscript%3E
```

- %0D%0A is a newline character in a URL-encoded HTTP response, which is typically represented in code as "\r\n" (CRLF).The attacker is attempting to insert a Set-Cookie header immediately after the original cookie by using the newline character as a line separator. As a result, a malicious script is added to the victim's page in order to hack it.

Impacts of CRLF Attack

- The HTTP response from the server would indicate that the set-cookie redirects to the malicious script.



```
HTTP/1.1 200 OK
Location: profile \r\n
Set-Cookie: username=<script>alert('hacked')</script>
```

- As a result, the client's username will be hacked with the injection of javascript (that can redirect to a hacked web browser attempting to collect and gain user's sensitive information or attack the server) showing the client an alert saying 'hacked' executing the script in the backend.

How to mitigate the effects of a CRLF injection attack?

The below methods can be used to mitigate a CRLF attack done through log injection and HTTP response splitting possibly leading to XSS attacks.

1. Limit the implementation of HTTP Request on the web server.

- We can do this manually or through the command line.
- To configure manually:
  - Click **Configuration > Virtual Servers > Server Settings > Request Limits.**
- To configure request limits through the command line:

```
user> enable-request-limits --user=admin --password-file=admin.pwd
--host=serverhost --port=8989 --config=config1 --vs=config1_vs_1
```

- **Pros** → A 404 range of errors will occur if an HTTP request URL length, query string, and request size limit are exceeded.

How to mitigate the effects of a CRLF injection attack?

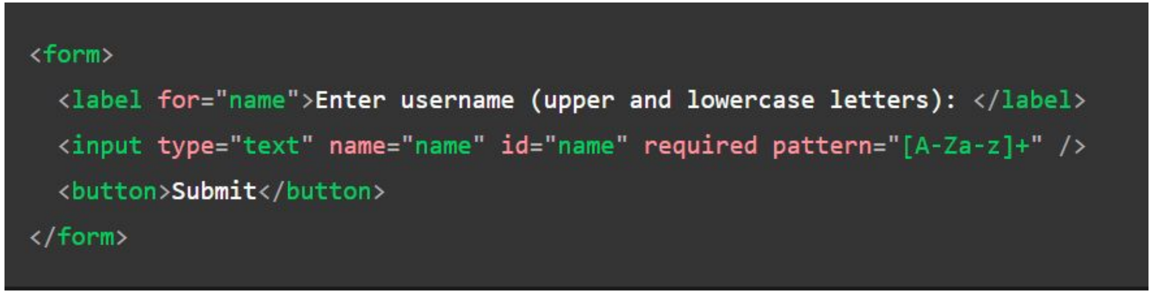
2. Implementation of client-side input validation to prevent malicious scripts from being entered such as defined location headers and HTML content being manipulated.

❖ There are two methods:

a. Built-in form Validation - There are certain validation attributes being considered and if there is any violation, it is considered invalid.

● Some of the elements that are considered here are:

- Pattern element that checks if a specific regular expression entered adheres to a specific pattern defined.

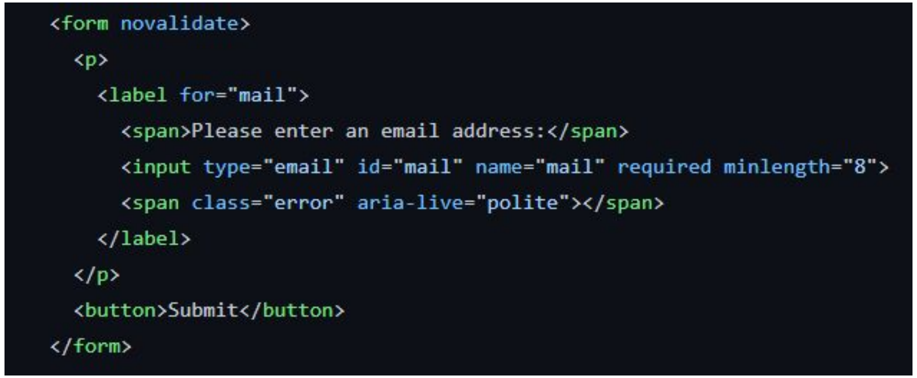


```
<form>
  <label for="name">Enter username (upper and lowercase letters): </label>
  <input type="text" name="name" id="name" required pattern="[A-Za-z]+" />
  <button>Submit</button>
</form>
```

Here, we check if the entered username equals to any regex character mentioned inside the brackets, [A-Za-z]+

How to mitigate the effects of a CRLF injection attack?

- The minLength element where this specifies the minimum length of textual data of strings a client can enter.



```
<form novalidate>
  <p>
    <label for="mail">
      <span>Please enter an email address:</span>
      <input type="email" id="mail" name="mail" required minlength="8">
      <span class="error" aria-live="polite"></span>
    </label>
  </p>
  <button>Submit</button>
</form>
```

Here, it specific that the entered email by the user must be only 8 characters long else display an error message.

Cons → However, input validation is too difficult and ineffective for some types of data such as comments.
Pros → Input validation can help protect against malicious data sent to the server via a website or application in an HTTP request by a malicious actor.

How to mitigate the effects of a CRLF injection attack?

b. Javascript validation - Validation by coding using javascript. This validation is completely customizable where we can customize error messages.

- This method uses Constraint Validation API - It is a set of attributes,methods and properties available that is allowed in the Document object model API.
- The ValidityState interface represents the validity states that a form control element can have in relation to the constraints that have been defined. They work together to explain whether and why a value fails to validate.
- Few of the Extensions to other interfaces are :

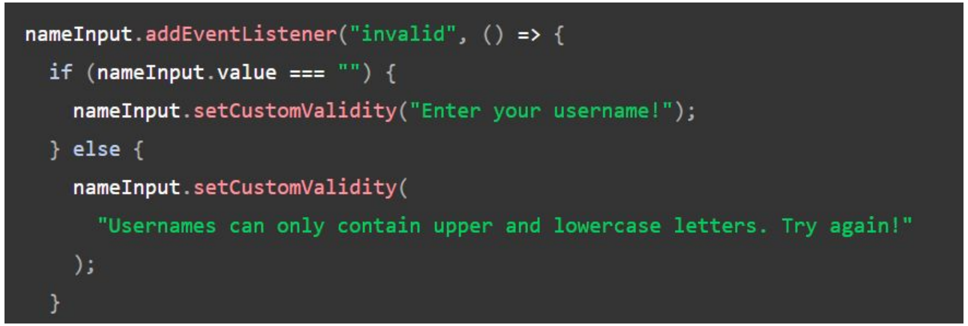
- HTMLInputElement → represents an input element
- HTMLOutputElement → represents an output element.

- Few of the Instance methods that can be used:

- checkValidity() → Compares the value of the element to its constraints. If the value is invalid, an invalid event is fired at the element and false is returned; otherwise, true is returned.

How to mitigate the effects of a CRLF injection attack?

- setCustomValidity() *(message)* → When a message is set, the validity state is set to invalid and it sets an error message to be shown to the user. In order to clear this state,we can call the function with an empty string as an argument. The custom error message is cleared in this case, the element is deemed valid, and no message is displayed.



```
nameInput.addEventListener("invalid", () => {
  if (nameInput.value === "") {
    nameInput.setCustomValidity("Enter your username!");
  } else {
    nameInput.setCustomValidity(
      "Usernames can only contain upper and lowercase letters. Try again!"
    );
  }
```

Considering the example given in slide 18 - we use the setCustomValidity method here to valid the username inside javascript.

Cons → An attacker who disables JavaScript or uses a Web Proxy can bypass JavaScript input validation performed on the client.

How to mitigate the effects of a CRLF injection attack?

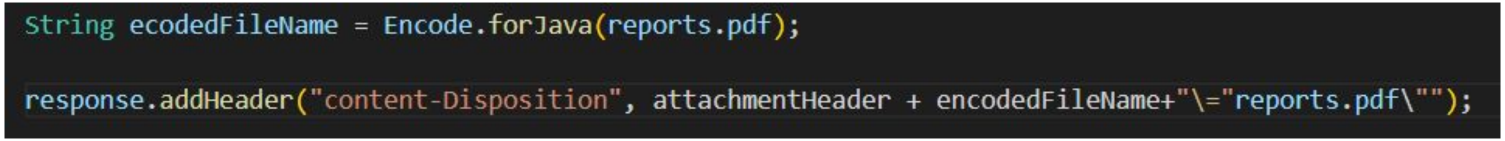
3. Sanitize and neutralize all user-supplied data, or encode output in HTTP headers correctly by scanning using Veracode and trusted third party library such as OWASP's Encoder jar to fix this issue.

- What is Veracode scan? Veracode provides an automated, on-demand application security testing solution that is both accurate and cost-effective.
- In OWASP, we can use maven and add the following dependency to get the Encoder.

```
<dependency>

<groupId>org.owasp.encoder</groupId>

<artifactId>encoder</artifactId>

<version>1.2.1</version>

<dependency>
```

How to mitigate the effects of a CRLF injection attack?

- Then we can encode the value and provide to escape the malicious character like CR, LF in addHeader() method as shown below using encoder.



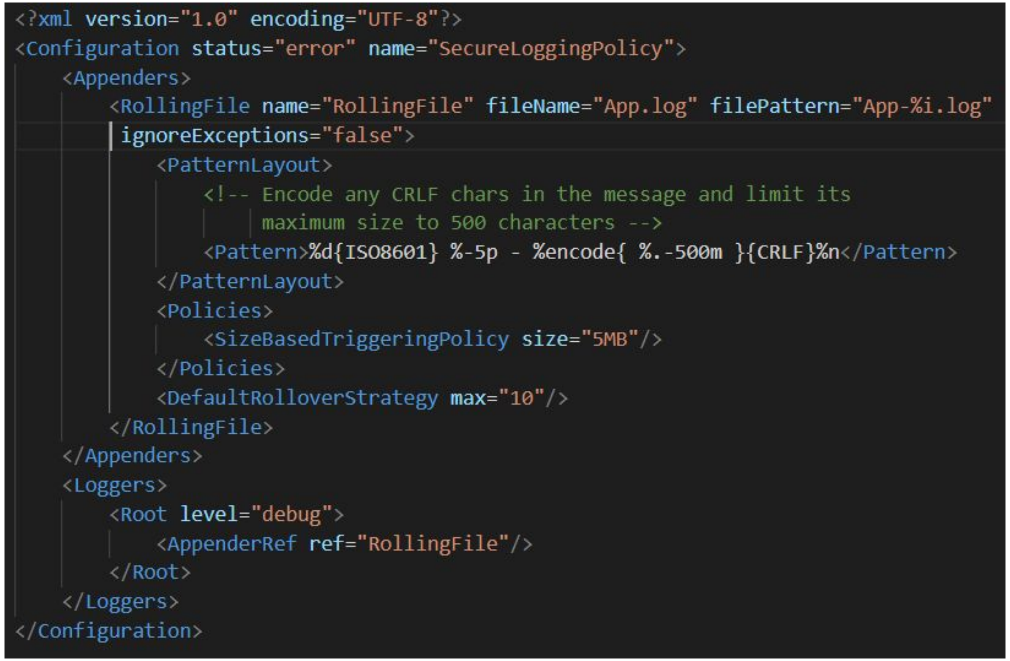
```
String ecodedFileName = Encode.forJava(reports.pdf);

response.addHeader("content-Disposition", attachmentHeader + encodedFileName+"\="reports.pdf\"");
```

- This method can demonstrate us that the scan is complete ensuring that no \r or \n is executed as they will be escaped by the above method.
- **Make sure that any input validation done on the client side is also done on the server side.**

Pros → We encrypt the data send in HTTP headers. If the attacker attempts to supply the CR and LF codes, they are effectively scrambled, resulting in a massive CRLF infusion sway.

How to mitigate the effects of a CRLF injection attack?

4. Use defenses such as: to prevent an attacker from writing malicious content into the application log.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="error" name="SecureLoggingPolicy">
    <Appenders>
        <RollingFile name="RollingFile" fileName="App.log" filePattern="App-%i.log"
        ignoreExceptions="false">
            <PatternLayout>
                <!-- Encode any CRLF chars in the message and limit its
                    maximum size to 500 characters -->
                <Pattern>%d{ISO8601} %-5p - %encode{ %.-500m }{CRLF}%n</Pattern>
            </PatternLayout>
            <Policies>
                <SizeBasedTriggeringPolicy size="5MB"/>
            </Policies>
            <DefaultRolloverStrategy max="10"/>
        </RollingFile>
    </Appenders>
    <Loggers>
        <Root level="debug">
            <AppenderRef ref="RollingFile"/>
        </Root>
    </Loggers>
</Configuration>
```

- Filter the user input to prevent the injection of Carriage Return (CR) or Line Feed (LF) characters by using regular expression such as filePattern.
- Limit the size of the user input value used to generate the log message, here it is -500m message size limit.
- When viewing log files in a web browser, ensure that all XSS defenses are enabled.

Conclusion

Thus it can be concluded that,

- CRLF is a vulnerability that can lead to potential gateway for other harmful attacks such as cross-site scripting, web cache, phishing and session fixation.
- The exploitation with http response splitting is done to show the attack and we have attached the log injection exploitation as well.
- Patching method that was done for both HTTP response splitting and Log injection is input validation.
- Mitigation methods are mostly reliable when its performed on both the server side and client side.

References

(Reference - slide 7 LogInjector: Detecting Web Application Log Injection Vulnerabilities)

(Reference - Slide 15 https://docs.oracle.com/cd/E19146-01/821-1828/gebyq/index.html)

(Reference - Slide 16 - 19 https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation)

(Reference - Slide 20-21https://webspidertech.wordpress.com/2018/07/31/veracode-flaws-crlf-http-response-splitting-cwe-113-java/)

(Reference - Slide 22 https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_in_Java_Cheat_Sheet.html

https://www.veracode.com/security/java/cwe-117)

(Other references for demo - https://poison.digi.ninja/)