Fix X-Content-Type-Options Header Missing

Now we go to fix X-Content-Type-Options Header Missing. I was found this article have instruction to fix this issue.
https://stackoverflow.com/questions/52034082/how-to-set-header-x-content-type-options-nosniff-in-springboot-application

First, we need and dependency below into pom.xml

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
        <!-- <version>2.3.12.RELEASE</version> -->
    </dependency>
```

Second, we create a new class, this class need inside source code folder.
This is my directory "devsecops/WebSecurityConfig.java". So package will be the same with project name.

```
package com.devsecops;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
   @Override
   protected void configure(HttpSecurity http) throws Exception {
      http.csrf().disable();
   }
}
```

According to this code, You will disable csrf protection. So you can remove http.csrf().disable(); out of code for secure measurement.
Another way if you have false positive alerts like below and you want to ignore the warnings then you can do the following, we can also add our own rules to ignore it. Now, you run command below.

```
docker run -v $(pwd):/zap/wrk/:rw -t owasp/zap2docker-weekly zap-api-scan.py -t $applicationURL:$PORT/v3/api-docs -f openapi -g zap-rules
```

In this command, "-g" tag is used to indicate the set of rules that will be used during the scan. Here, "zap-rules" refers to the set of rules that are bundled with OWASP ZAP, an open-source web application security scanner. These rules are used to detect and report the security vulnerabilities present in the API endpoints of an application.

You can see below is a sample rule after run command above

```
# zap-api-scan rule configuration file
# Change WARN to IGNORE to ignore rule or FAIL to fail if rule matches
# Active scan rules set to IGNORE will not be run which will speed up the scan
# Only the rule identifiers are used - the names are just for info
# You can add your own messages to each rule by appending them after a tab on each line.
```

```
10003   WARN    (Vulnerable JS Library (Powered by Retire.js) - Passive/release)
10009   WARN    (In Page Banner Information Leak - Passive/beta)
10010   WARN    (Cookie No HttpOnly Flag - Passive/release)
10011   WARN    (Cookie Without Secure Flag - Passive/release)
10015   WARN    (Re-examine Cache-control Directives - Passive/release)
10017   WARN    (Cross-Domain JavaScript Source File Inclusion - Passive/release)
10019   WARN    (Content-Type Header Missing - Passive/release)
10020   WARN    (Anti-clickjacking Header - Passive/release)
10021   WARN    (X-Content-Type-Options Header Missing - Passive/release
Example false positive:
WARN-NEW: X-Content-Type-Options Header Missing [100000] x 4
    http://192.168.207.129:31242/increment/10 (200)
    http://192.168.207.129:31242/compare/10 (200)
    http://192.168.207.129:31242/ (200)
    http://192.168.207.129:31242/v3/api-docs (200)
```

Now, run this command:

```
docker run -v $(pwd):/zap/wrk/:rw -t owasp/zap2docker-weekly zap-api-scan.py -t $applicationURL:$PORT/v3/api-docs -f openapi -c zap-rules -r zap_report.html
```

The `-c` flag in the command you provided is used to specify a configuration file containing rules for the scan. The rules file is used to configure which rules should be run during the scan and how they should be handled.

In our case, the rules file is named `zap-rules`. It contains a list of rules with their corresponding rule identifiers and their handling configuration. The handling configuration can be set to either `WARN`, `IGNORE`, or `FAIL`.

For example, rule number `10003` has a handling configuration of `WARN` and its name is "Vulnerable JS Library (Powered by Retire.js)". This means that if this rule matches during the scan, it will generate a warning message.

To write your own rules file, you can create a text file with a `.conf` extension and use the following format:

```
<rule_id> <handling_configuration> <rule_name>

For example:
10001       WARN        Example Rule 1
10002       IGNORE        Example Rule 2
```

Note: keep the space between each value with the tab button, do not use space button.

Now you can create file like below to create custom rule to ignore false positive.
The first value is the rule ID, you can see the report above, we have the ID 100000, the second value and we configure it as IGNORE, the last value is the url we see in the report fox.

```
# zap-api-scan rule configuration file
# Change WARN to IGNORE to ignore rule or FAIL to fail if rule matches
# Active scan rules set to IGNORE will not be run which will speed up the scan
# Only the rule identifiers are used - the names are just for info
# You can add your own messages to each rule by appending them after a tab on each line.
100000  IGNORE  http://192.168.207.129:31242
```

Reference: OWASP ZAP – ZAP - API Scan (zaproxy.org)

https://www.zaproxy.org/docs/docker/api-scan/#rules-file-format

Run again:

```
docker run -v $(pwd):/zap/wrk/:rw -t owasp/zap2docker-weekly zap-api-scan.py -t $applicationURL:$PORT/v3/api-docs -f openapi -c zap-rules -r zap_report.html
```

ADDTION CREATE CUSTOM RULE:

For example, if you have a configuration file named `my_zap_rules.conf`, you can use the `-c` tag to specify that configuration file:

```
docker run -v $(pwd):/zap/wrk/:rw -t owasp/zap2docker-weekly zap-api-scan.py \
-t $applicationURL:$PORT/v3/api-docs -f openapi -c my_zap_rules.conf \
-r zap_report.html
```

A configuration file is a set of rules and configurations that you can use to tailor ZAP's scanning to your application. It can contain various settings such as the scope of the scan, authentication settings, and other customizations.

Each rule in a configuration file defines a specific security test that will be performed by ZAP. For example, a rule can be defined to check for SQL injection vulnerabilities, cross-site scripting (XSS) vulnerabilities, or other security issues.

Here's an example of a rule that checks for SQL injection vulnerabilities:

SQL injection check

```
alertSqlInjection=\
   enabled\
   alertThresholdHigh=3\
   alertThresholdMedium=2\
   attackStrength=medium\
   ignorePublicOrigin=true\
   ignoreTimeThresholdInSecs=3600\
   maxErrors=-1\
   matchRegex=\
   onlyForward=false\
   parameterName=\
   poa=false\
   useUnionBasedSqlInjection=true\
   useErrorBasedSqlInjection=true\
   useBooleanBasedSqlInjection=true\
   usePiggyBackedSqlInjection=true\
   useOutOfBandSqlInjection=true\
   useSecondOrderSqlInjection=true
```

This rule checks for SQL injection vulnerabilities in the application. It specifies various settings such as the attack strength, whether to use union-based, error-based, or boolean-based SQL injection techniques, and whether to use out-of-band techniques.
The SQL Injection check in the provided configuration file checks for SQL injection vulnerabilities in the application being tested by ZAP. This vulnerability allows an attacker to inject malicious SQL code into a query, which can result in data loss, data manipulation or unauthorized access to sensitive data.

The configuration file specifies various settings for the SQL Injection check, which can be matched against the requests and responses made by the application being tested. Here's an explanation of some of the important settings in the rule:

`enabled`: This setting specifies whether the rule is enabled or disabled. When set to , the rule will be used during the scan.`enabled`
`attackStrength`: This setting defines the strength of the SQL injection attack. It can be set to either , or . The strength determines the complexity of the payload generated by ZAP. A higher strength increases the likelihood of detecting vulnerabilities but can also result in more false positives.`low``medium``high`
`useUnionBasedSqlInjection`: When set to , ZAP will use union-based SQL injection, which is a technique that allows an attacker to retrieve data from the database.`true`
`useErrorBasedSqlInjection`: When set to , ZAP will use error-based SQL injection, which is a technique that relies on error messages generated by the database to retrieve data.`true`
`useBooleanBasedSqlInjection`: When set to , ZAP will use boolean-based SQL injection, which is a technique that relies on the true or false responses generated by the database to retrieve data.`true`
`usePiggyBackedSqlInjection`: When set to , ZAP will use piggy-backed SQL injection, which is a technique that injects malicious code into fields that are not directly used in the query, but are included in the response.`true`

During the scan, ZAP will use the settings specified in the configuration file to generate payloads and test the application for SQL injection vulnerabilities. If ZAP detects a vulnerability, it will generate an alert, which can be viewed in the ZAP GUI or in the report generated after the scan.

Example XSS check in a ZAP configuration file:

```
alertXss=\
   enabled\
   alertThresholdHigh=3\
   alertThresholdMedium=2\
   attackStrength=medium\
   ignorePublicOrigin=true\
   ignoreTimeThresholdInSecs=3600\
   maxErrors=-1\
   matchRegex=\
   onlyForward=false\
   parameterName=\
   poa=false\
   scanHeadersAllRequests=false\
   scanHeadersInAttack=false\
   showEvidence=true\
   stripScriptTags=false\
   testAllTags=false\
   variant=0
```

This rule checks for cross-site scripting (XSS) vulnerabilities in the application being tested by ZAP. An XSS vulnerability allows an attacker to inject malicious code into a web page that can steal data or modify the behavior of the page.

The configuration file specifies various settings for the XSS check, which can be matched against the requests and responses made by the application being tested. Here's an explanation of some of the important settings in the rule:
`enabled`: This setting specifies whether the rule is enabled or disabled. When set to `enabled`, the rule will be used during the scan.
`attackStrength`: This setting defines the strength of the XSS attack. It can be set to either `low`, `medium` or `high`. The strength determines the complexity of the payload generated by ZAP. A higher strength increases the likelihood of detecting vulnerabilities but can also result in more false positives.
`stripScriptTags`: When set to `false`, ZAP will not remove script tags from the response before testing for XSS vulnerabilities. If the application is vulnerable to XSS attacks, ZAP will generate an alert with details of the attack.
`testAllTags`: When set to `false`, ZAP will only test input fields for XSS vulnerabilities. If set to `true`, ZAP will test all tags for vulnerabilities, including tags that are normally not used for user input.

Learn more:
SQL Injection Prevention - OWASP Cheat Sheet Series
zap-extensions/SqlInjectionScanRule.java at main · zaproxy/zap-extensions · GitHub