INFORME DE AUDITORIA WEB GOAT

ALEXANDRA OANE

INDICE

1. ÁMBITO Y ALCANCE DE LA AUDITORÍA

1.1. INTRODUCCIÓN

La presente auditoria se ha realizado para la aplicación web “WebGoat” (http://127.0.0.1:8080/WebGoat).

Inicialmente se realizó un registro del usuario para poder acceder. A continuación se realizó la búsqueda de los puertos abiertos, el sistema operativo y los lenguajes de programación empleados en la aplicación web.

Los puertos abiertos: El puerto es 80.

El sistema operativo: No he podido encontrar el sistema operativo.

Lenguajes de programación: el lenguaje de programación es javascript.

Al acceder a WebGoat, se accedió a las distintas secciones situadas en la página principal.
A continuación se realizo la descarga e instalación de ZAP, una de las herramientas empleadas para realizar la auditoria. También se ha usado nmap,

Además durante la realización del informe, se ha buscado información de Sql Injection para poder realizar los apartados. Los apartados opcionales ha sido necesario resolverlos para poder obtener las soluciones de los ejercicios posteriores en algunos de los casos.

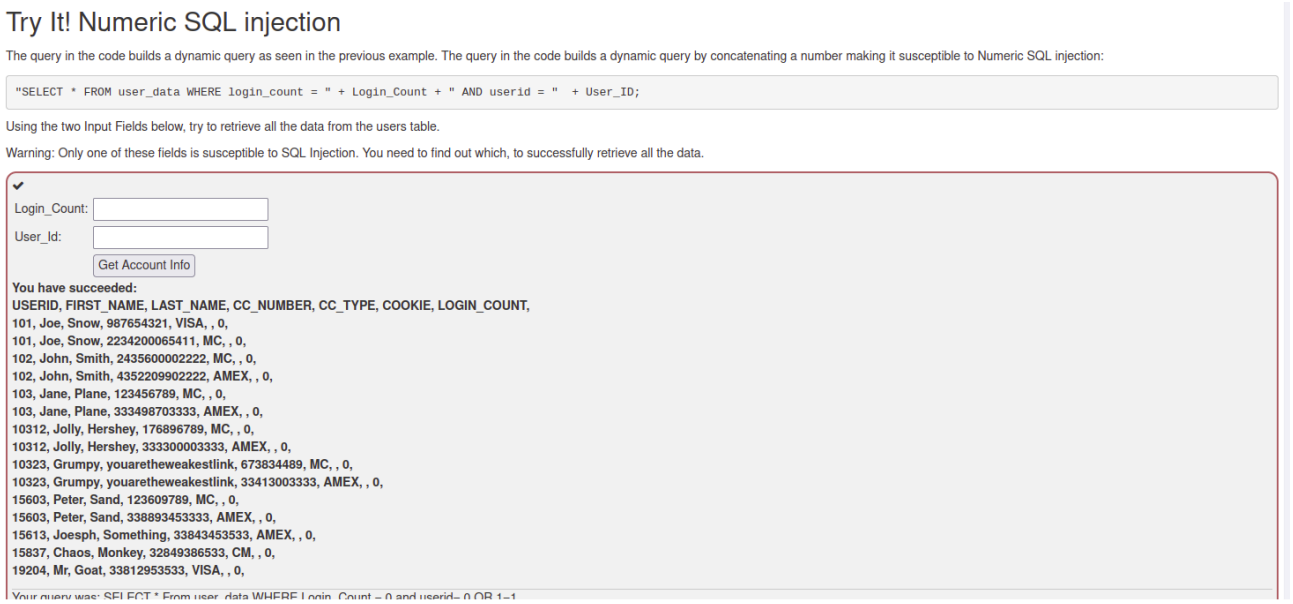
2. INFORME EJECUTIVO

2.1. A1 SQL Injection – Apartado 10

2.1.1. Proceso realizado

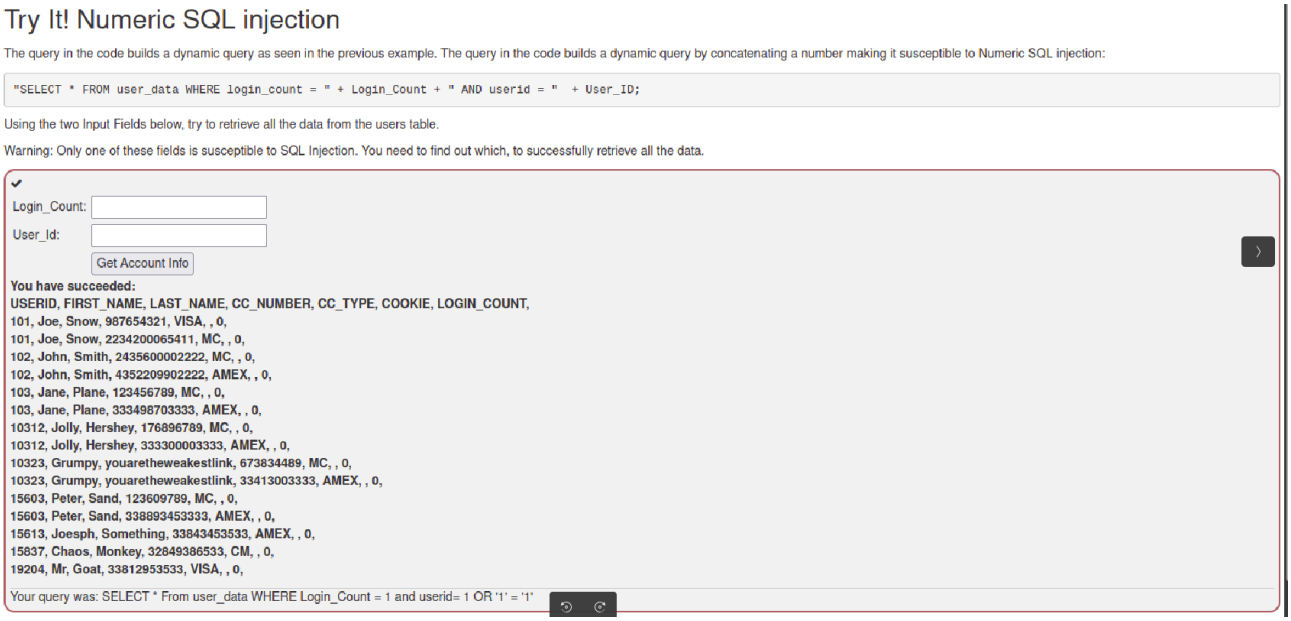
Al inicio del ejercicio se observa que es necesario resolver apartados anteriores para poder obtener la solución.
En este apartado se debe averiguar Login_Count y User_id. Para poder acceder se realizamo una búsqueda de información sobre Numeric SQL Injection. Tras intentar la consulta con los distintos campos y probando distintos datos finalmente introduciendo en los campos:

- Login_Count= 0
- User_id= 0 OR 1+1



Se observa que introduciendo una segunda opción con los siguientes datos:

- Login_Count= 1
- User_id= 1 OR ‘1’ = ‘1’”



2.1.2. Vulnerabilidades

Las vulnerabilidades que se observan en este apartado, es una serie de fallo que presenta la aplicación web, probando manualmente distintos datos en un orden determinado para poder acceder a los datos de la base de datos.

2.1.3. Conclusiones

Para poder acceder a todos los datos de los empleados presentados en la tabla en el apartado 2, se comprueba que se produce un fallo que permite acceder a los nombres completos y datos de sus tarjetas bancarias.

2.1.4. Recomendaciones

Se recomienda reforzar la seguridad de la aplicación web.

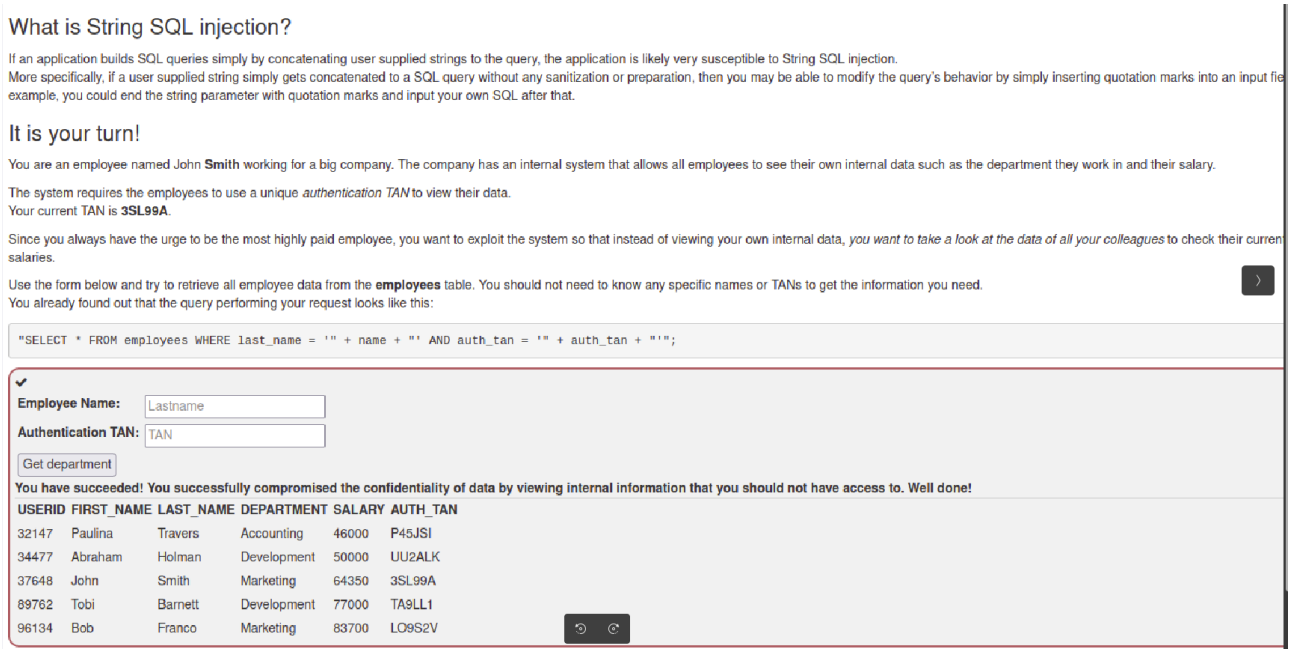
2.2. A1 SQL Injection - Apartado 11

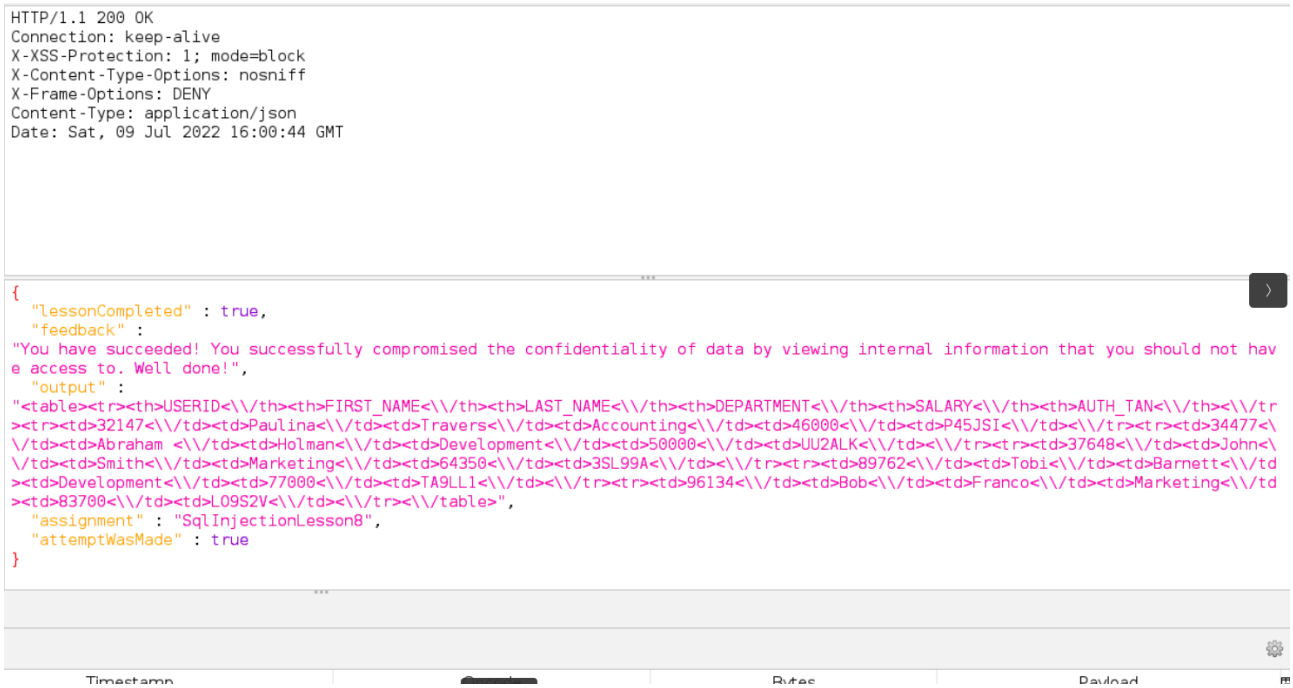
2.2.1. Proceso realizado

En este apartado se pide realizar una cadena de inyección SQL. La forma que nos presentan de ayuda para poder acceder siguiendo el orden establecido, se han insertado los siguientes datos:

EMPLOYEE NAME: 0
Authentication TAN: 0 OR 1=1

El resultado ha sido vulnerar la seguridad y poder acceder a la información interna confidencial de los empleados, sin tener acceso.
También se intentó acceder insertando datos que se presentaban en la tabla, pero no ha sido posible.
Insertando apellidos de los empleados en “employee name” y en “authentification TAN”
‘OR ‘1’ = 1’ (ejemplo), no se ha podido acceder.
Dando el valor “0” inicial y en el segundo apartado la combinación de “0” ó 1=1, es en el único caso en el que se ha producido el fallo.





2.2.2. Vulnerabilidades

En este apartado se observa que al insertar la combinación numérica de 0 y 1, dando estos valores se produce un fallo que permite acceder a datos confidenciales de los empleados.

2.2.3. Conclusiones

Se ha podido acceder a los datos confidenciales de la base de datos, de los empleados. Este fallo se produce al realizar una cadena de inyección SQL, introduciendo los valores numéricos 0 y 1.

2.2.4. Recomendaciones

Se recomienda reforzar la seguridad.

2.3. Intenta obtener toda la información que puedas de la base de datos utilizando los fallos disponibles en la sección A1 SQL Injection.

2.3.1. Proceso realizado

Se ha tratado de realizar la búsqueda de toda la información posible de la base de datos usando los fallos posibles y no ha sido posible.

2.3.2. Vulnerabilidades

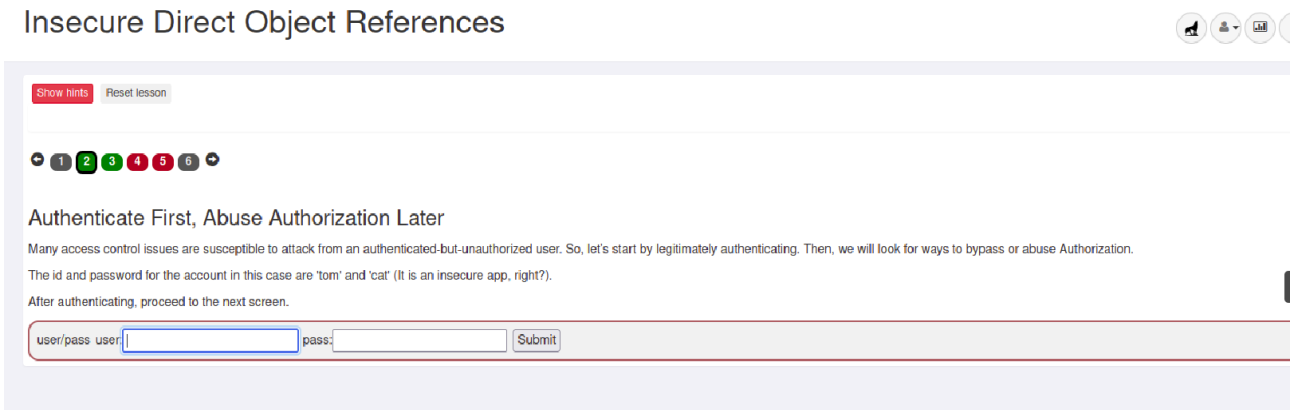
2.3.3. Conclusiones

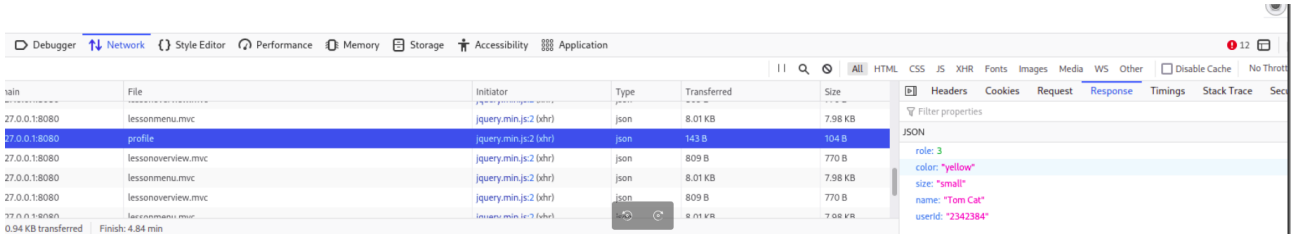
2.3.4. Recomendaciones

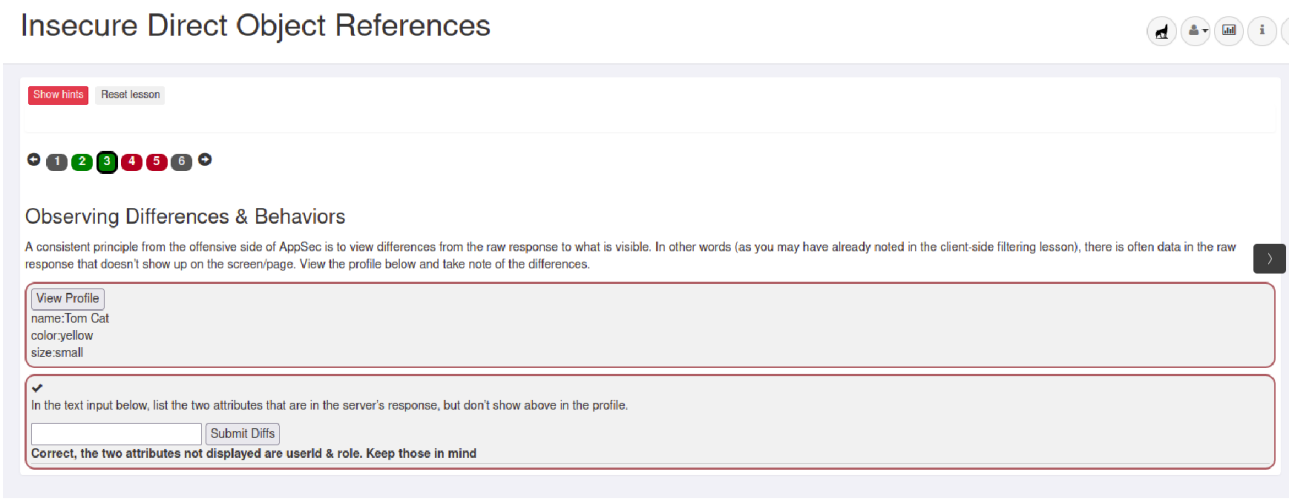
2.4. A5 Insecure Direct Object References - Apartado 2 y Apartado 3

2.4.1. Proceso realizado

En este apartado, se retrocedió al ejercicio anterior para introducir los datos en los campos que se señalan. Dentro de la aplicación web de WebGoat se realizó la inspección a través de “more tools”.
Se observa en el campo de “response” los datos necesarios para poder realizar el apartado 3.
Los dos atributos que se solicitan en el ejercicio pero no están presentes, se pueden obtener accediendo al campo anteriormente mencionado.
Además también se ha podido observar que también se muestra el userID.







2.4.2. Vulnerabilidades

Las vulnerabilidades que se observan en este apartado, sin la necesidad de abrir “Inspector”, se puede averiguar. Pero con la ayuda de la herramienta se ha podido acceder a los siguientes apartados. Obteniendo los atributos de señalan en el ejercicio.

2.4.3. Conclusiones

Con la ayuda de la herramienta de desarrollador “Inspector”, buscando entre las distintas secciones se ha podido encontrar desde Network y Response, los dos atributos solicitados.

2.4.4. Recomendaciones

Se recomienda reforzar la seguridad de las referencias de los objetos.

2.5. A5 Insecure Direct Object References - Apartado 4

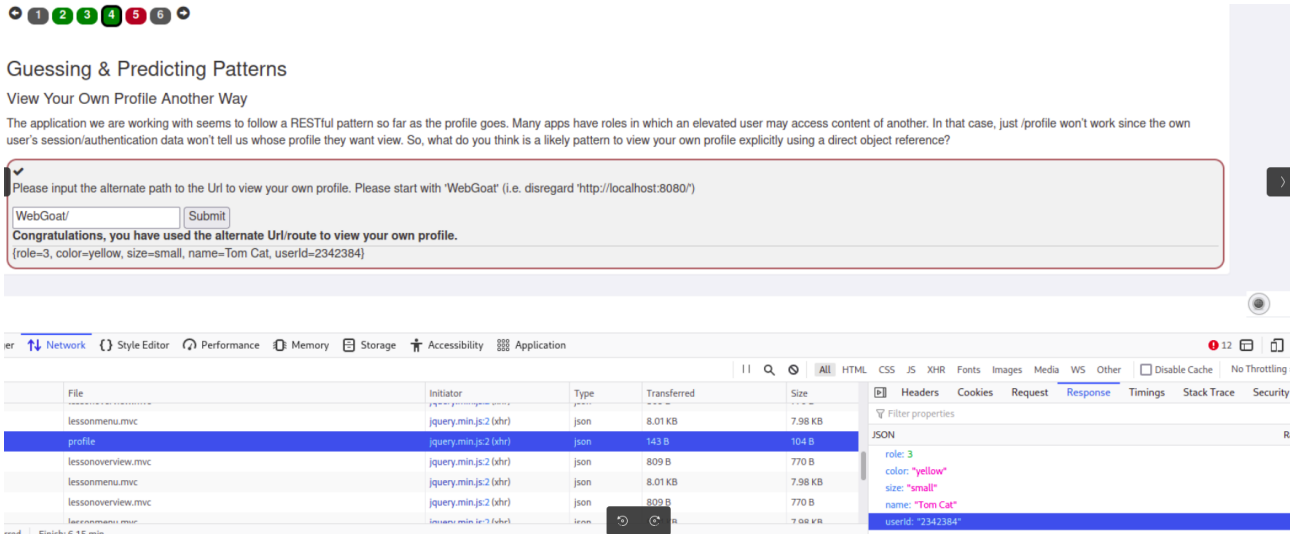
2.5.1. Proceso realizado

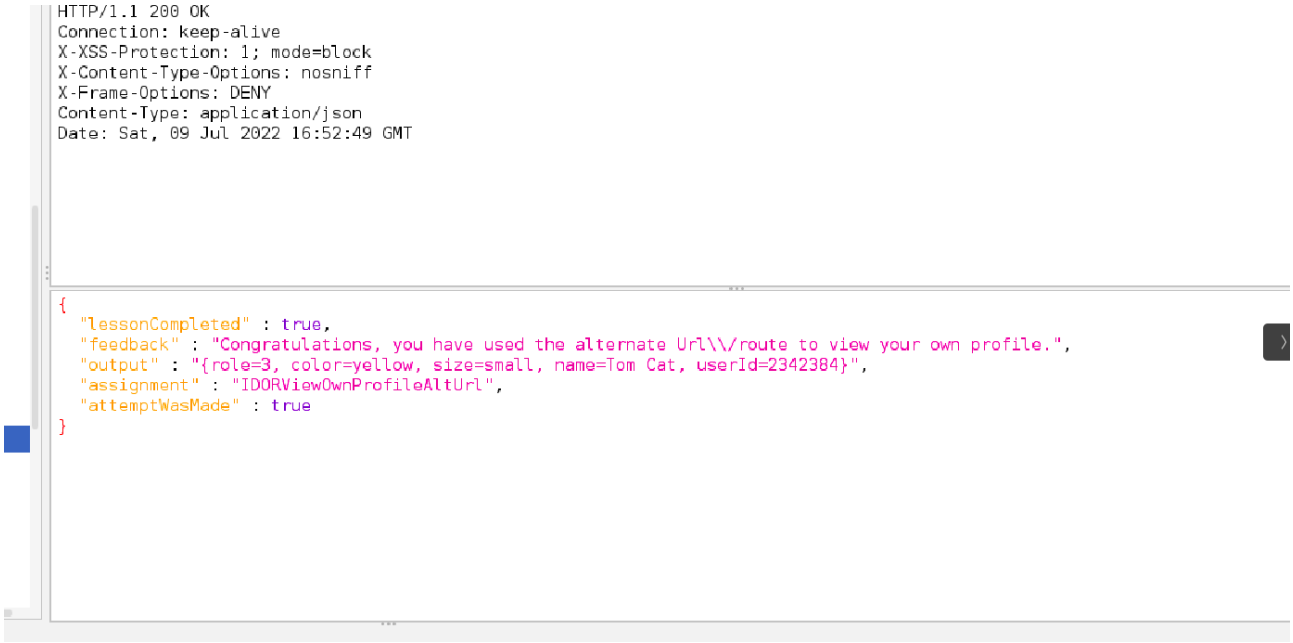
En este apartado con la obtención de los datos de los apartados anteriores en la dirección que se solicita se obtiene en el apartado de “Headers”.

webgoat/IDOR/profile/numero del userid que en este caso es 2342384

injection → developer tools → network

En la sección de network y en el apartado response se puede observar los datos que se muestran.





2.5.2. Vulnerabilidades

Se ha observado a través de la obtención de los datos anteriores y con la ayuda de la herramienta, el fallo se produce por los problemas del control de acceso por un usuario autenticado pero no autorizado.

2.5.3. Conclusiones

Se han podido extraer los distintos datos a través de la herramienta de desarrollador.

2.5.4. Recomendaciones

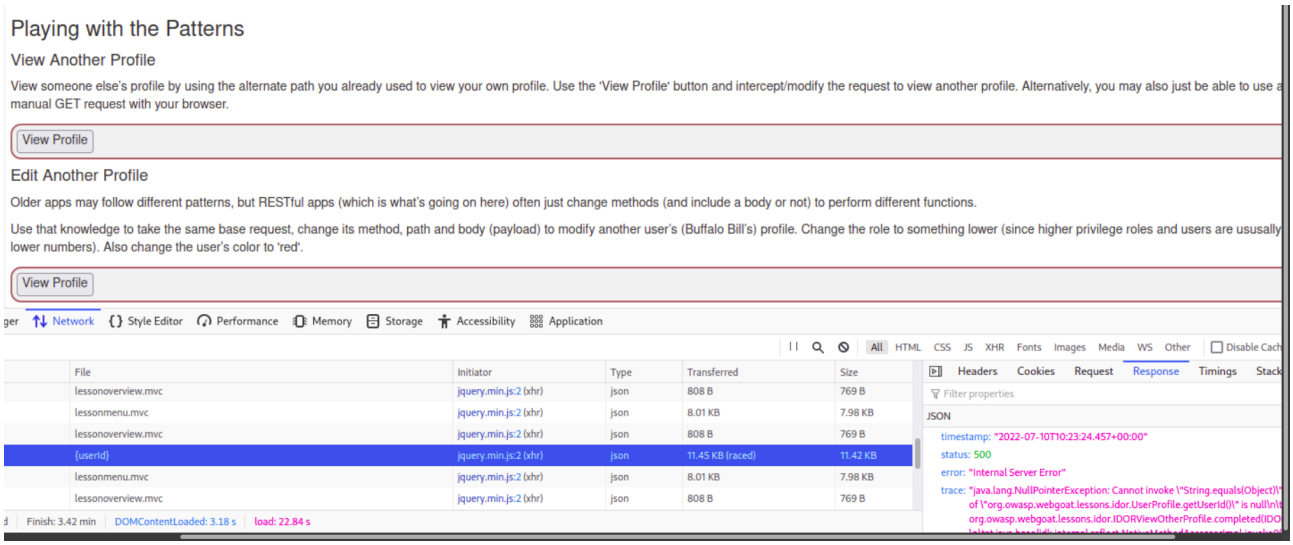
Reforzar la seguridad, realizando configuraciones más restrictivas.

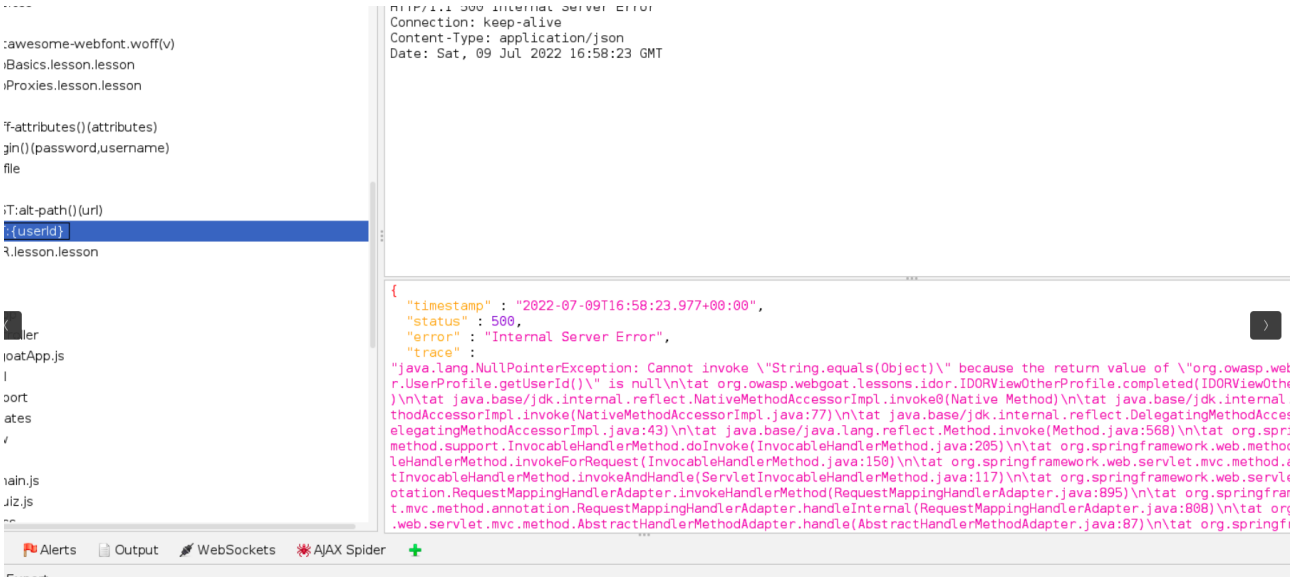
2.6. A5 Insecure Direct Object References - Apartado 5

2.6.1. Proceso realizado

En este apartado al tratar de acceder a los dos perfiles se puede observar, se ha tratado de realizar el ejercicio pero no ha sido posible averiguar la solución.

status 500 y error "Internal Server Error"





2.6.2. Vulnerabilidades

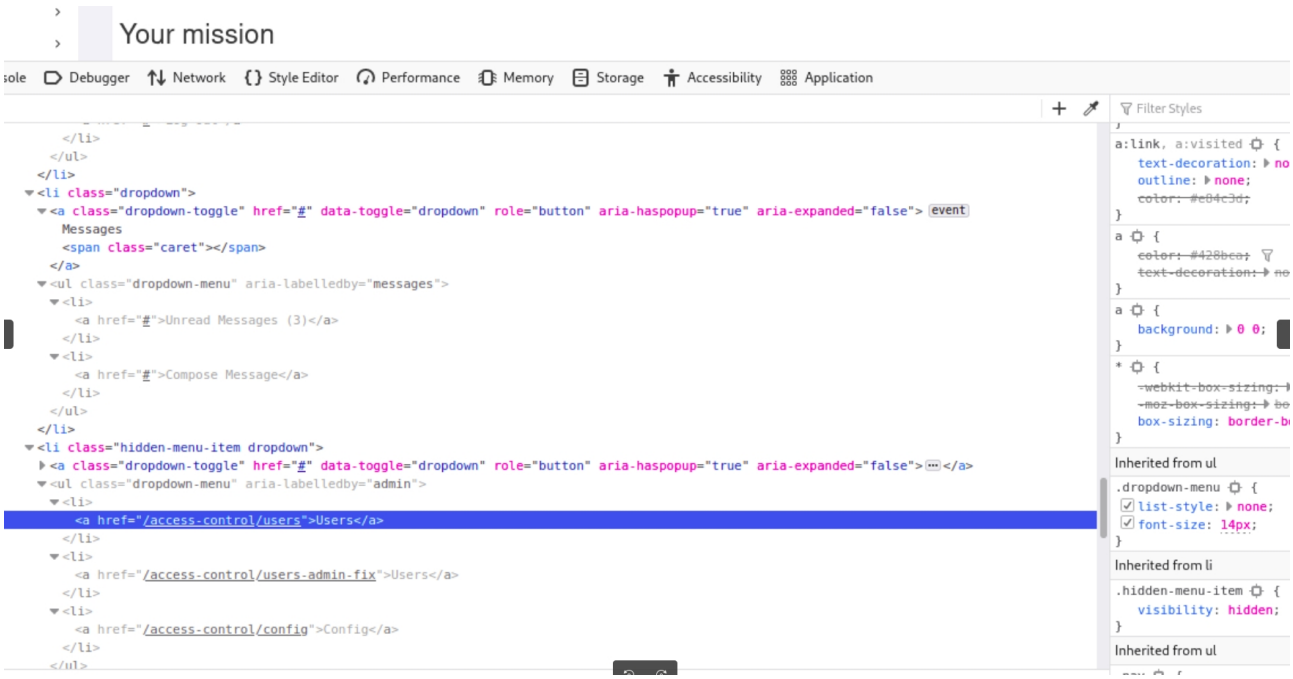
2.6.3. Conclusiones

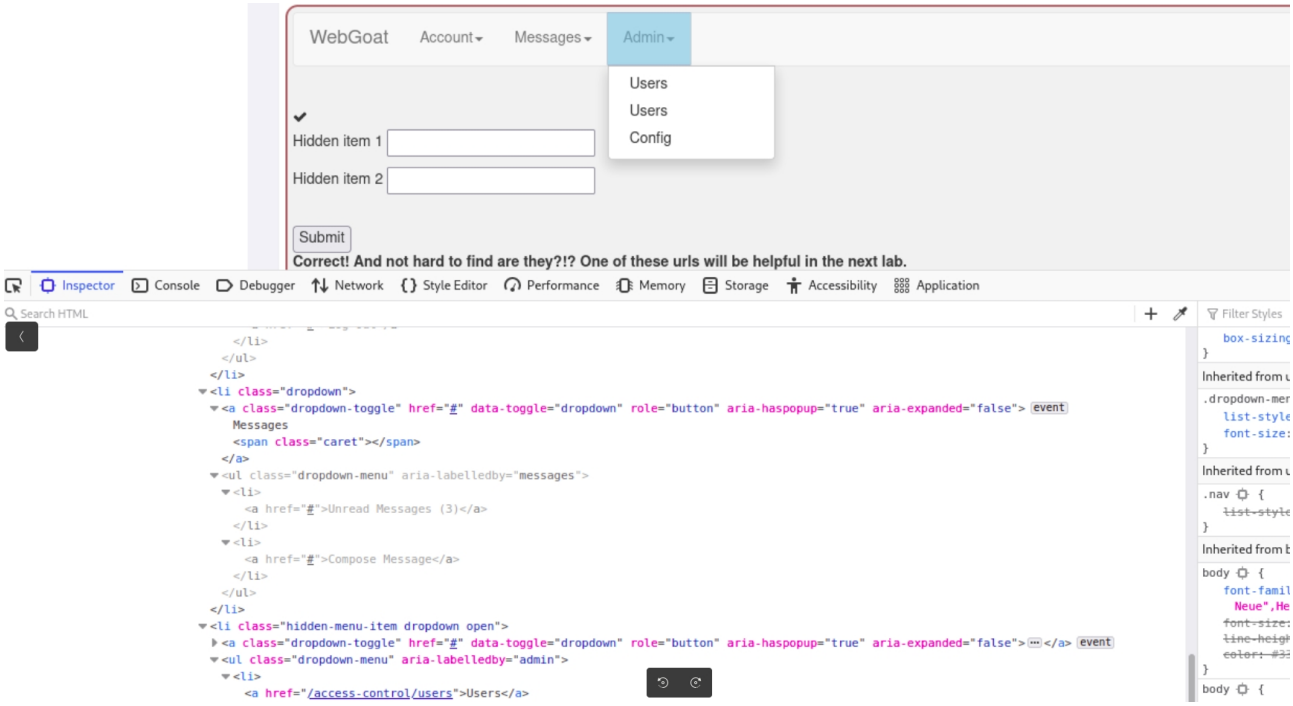
2.6.4. Recomendaciones

2.7. A5 Missing Function Level Access Control - Apartado 2

2.7.1. Proceso realizado

En este apartado se pide averiguar dos objetos que se encuentran ocultos. Tras realizar varios intentos de manera aleatoria, los distintos resultados eran incorrectos.
Se accedió a través de “Inspector” y se realizó una búsqueda, en una de las secciones se pudo acceder a la parte de admin y dar la visibilidad a los campos ocultos.
Al lado de “Messages” se puede comprobar que se encuentra el campo de “Administrador”.





2.7.2. Vulnerabilidades

Las vulnerabilidades que se observan en este apartado debido a un error que presenta, es poder acceder a la visibilidad de los campos ocultos. Se obtiene de una manera sencilla buscando en los datos de la consola.

2.7.3. Conclusiones

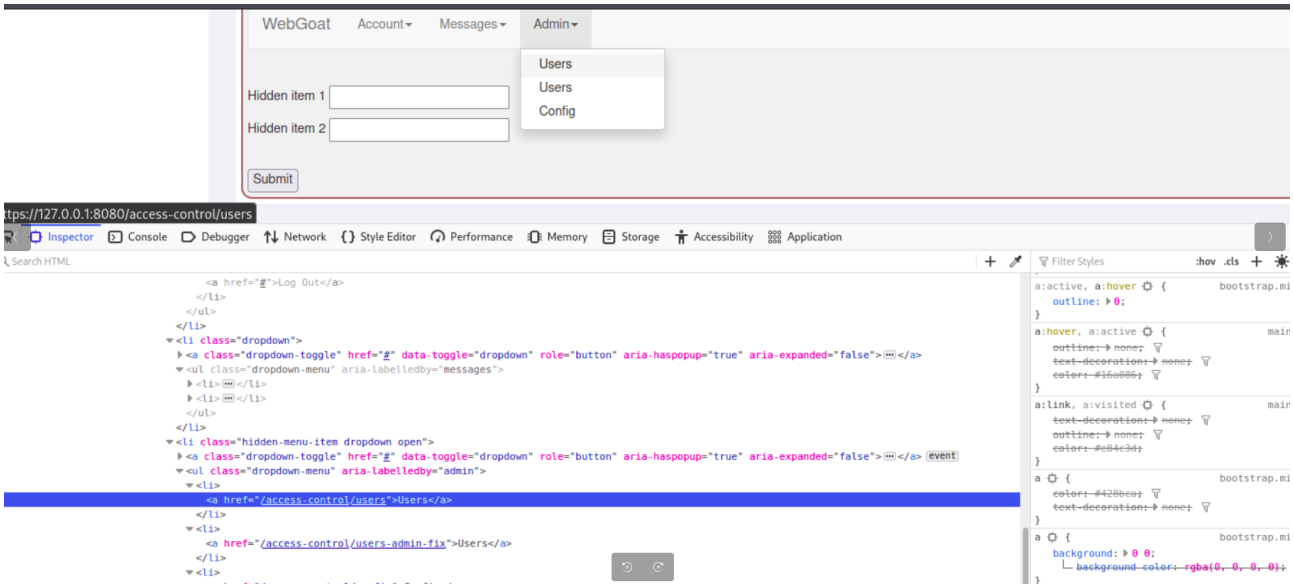
Se ha podido acceder a los campos ocultos con la ayuda de las herramientas de desarrollador.

2.7.4. Recomendaciones

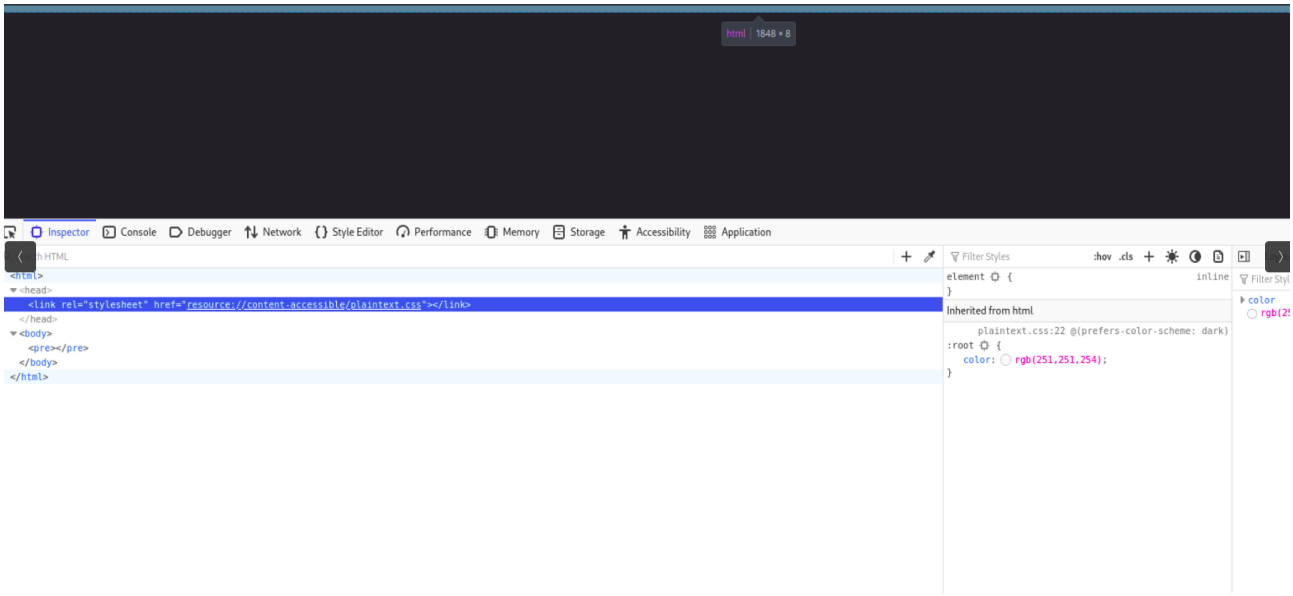
Se recomienda reforzar la visibilidad de los campos que se quieren ocultar a los posibles atacantes.

2.8. A5 Missing Function Level Access Control - Apartado 3

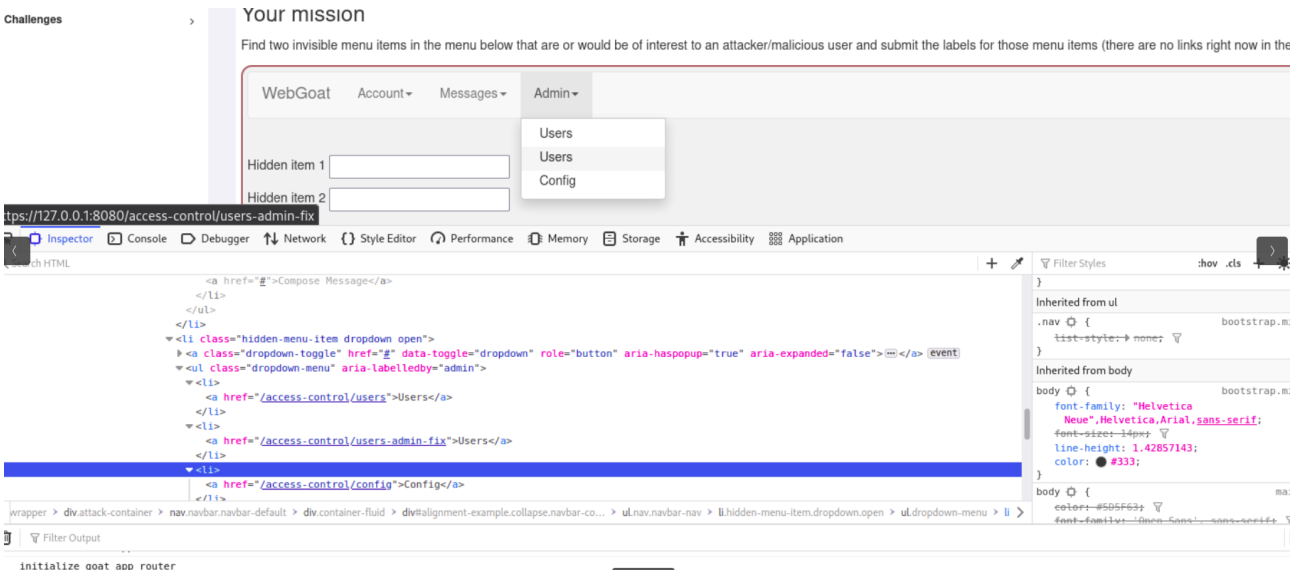
2.8.1. Proceso realizado

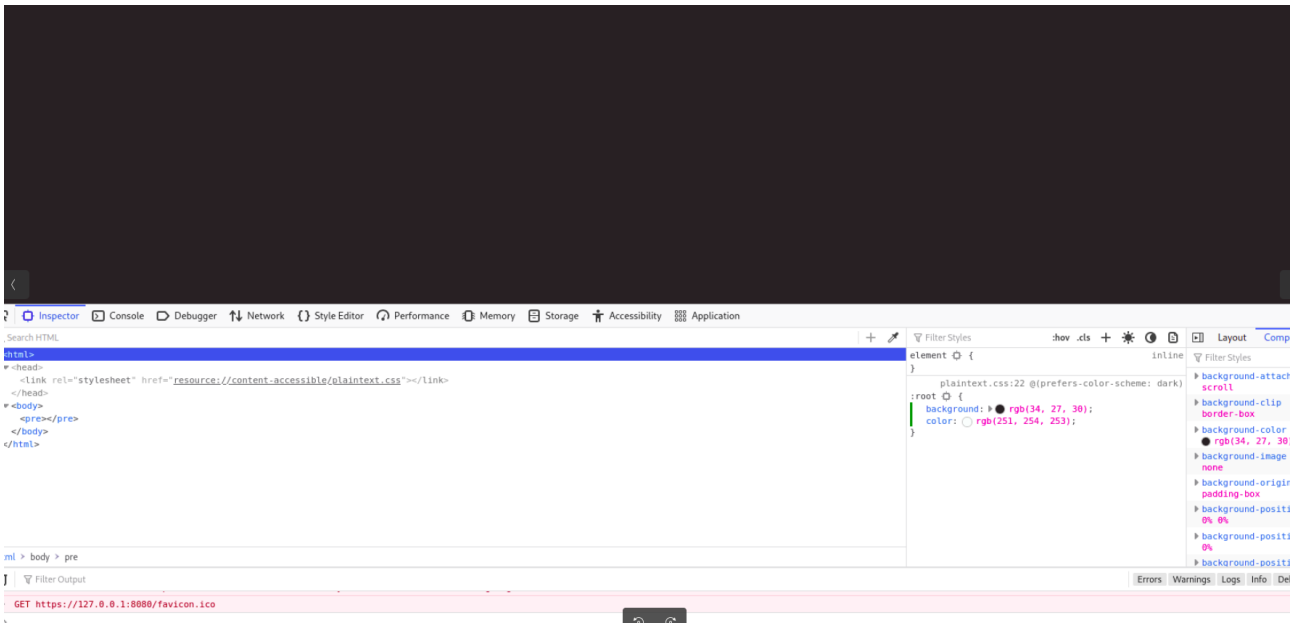


En el siguiente apartado, se solicita que se encuentre dos objetos que no se muestran.En un primer momento se realizaron pruebas con distintos datos, al no dar resultado, se usó “Inspector” y se fue abriendo y comprobando cada uno de los resultados. En este apartado no se ha podido encontrar el Hash.

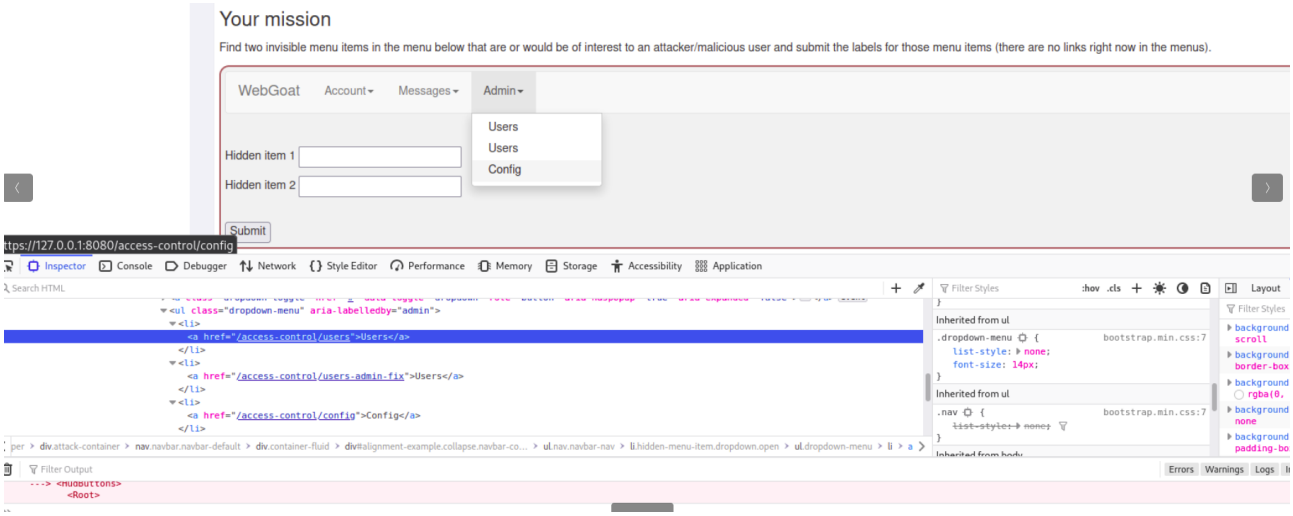


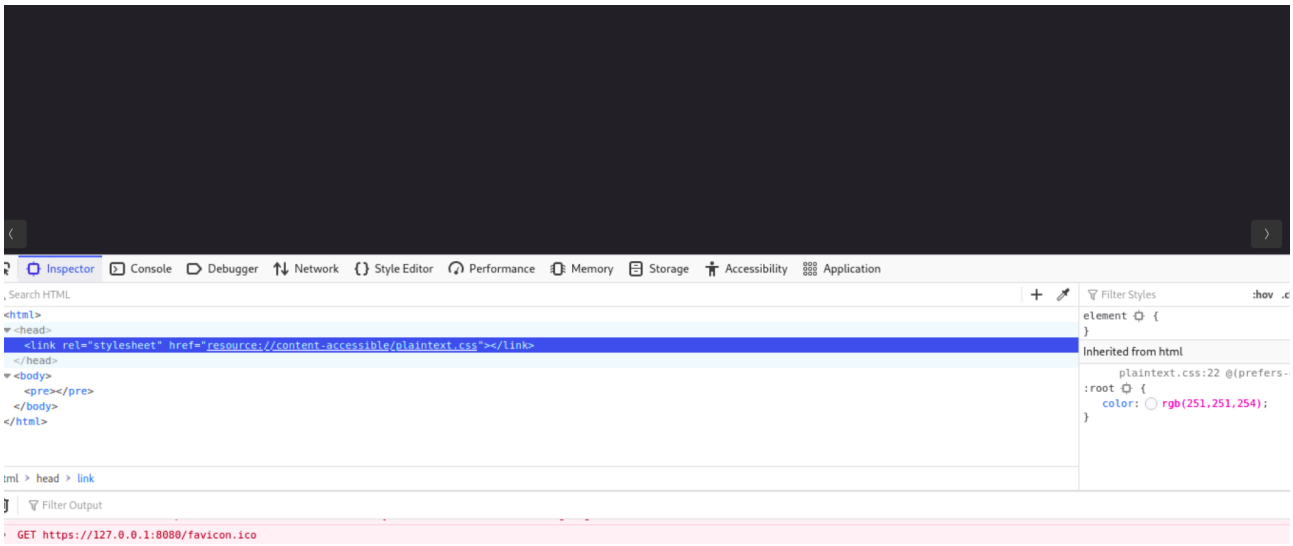
Se comprueba con la siguiente opción control users admin, en este caso también se observa que no hay ningún dato que se proporcione.





Y también se comprueba con la última opción, configuración. En este caso tampoco se observa ningún dato que permita acceder.





2.8.2. Vulnerabilidades

2.8.3. Conclusiones

Se ha tratado de realizar toda la búsqueda e inspección pero no ha sido posible averiguar el Hash.

2.8.4. Recomendaciones

2.9. A7 Cross Site Scripting - Apartado - Apartado 7

2.9.1. Proceso realizado

No se ha podido realizar el apartado.

2.9.2. Vulnerabilidades

2.9.3. Conclusiones

2.9.4. Recomendaciones

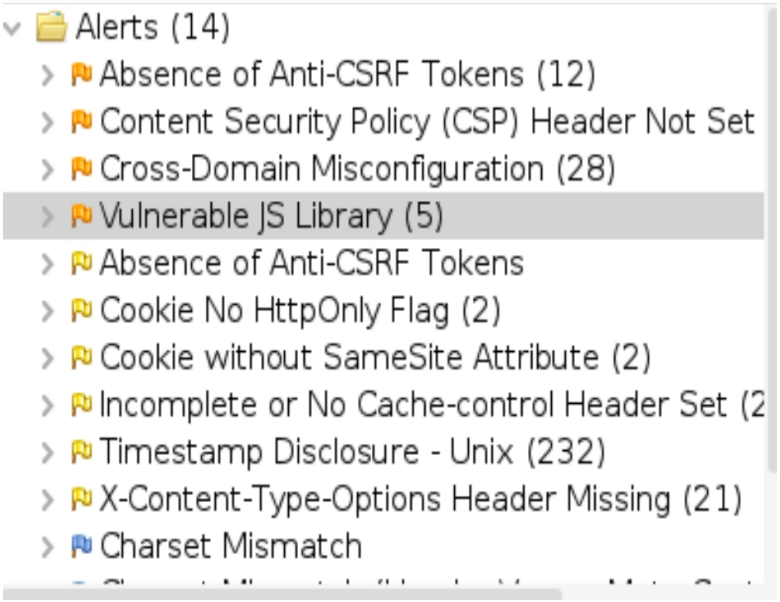
3. DESCRIPCIÓN DEL PROCESO DE AUDITORIA

3.1. Reconocimiento/Information gathering

Se ha monitorizado y rastreado toda la posible información para poder acceder a la información confidencial y la posibilidad de vulnerar la seguridad e insertar datos.
En los distintos apartados accediendo a la herramientas de developer y con la ayuda de ZAP se pueden realizar algunos de los ejercicios.

3.2. Explotación de vulnerabilidades detectadas

Durante el proceso de reconocimiento se han detectado inicialmente 14 alertas, de las cuales 4 son de tipo medio y 10 son de tipo bajo.



CSP: Wildcard Directive

URL: http://127.0.0.1:37903/
Risk: Medium
Confidence: Medium
Parameter:
Attack:
Evidence: default-src 'none'; script-src 'self'; connect-src 'self'; child-src 'self'; img-src 'self' data:; font-src 'self' data:; style-src 'self'
CWE ID: 693
WASC ID: 15
Source: Passive (10055 - CSP)

Description:

frame-ancestors, form-action

The directive(s): frame-ancestors, form-action are among the directives that do not fallback to default-src, missing/excluding them is the same as allowing anything.

Other Info:

Solution:

Ensure that your web server, application server, load balancer, etc. is properly configured to set the Content-Security-Policy header.

Reference:

http://www.w3.org/TR/CSP2/
http://www.w3.org/TR/CSP/
http://caniuse.com/#search=content+security+policy

Alert Tags:

Key	Value
OWASP_2021_A05	https://owasp.org/Top10/A05_2021-Security_Misconfiguration/
OWASP_2017_A06	https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfigur

Cross-Domain Misconfiguration

URL: https://firefox.settings.services.mozilla.com/v1/buckets/monitor/collections/changes/changeset?collection=partitioning-exempt-urls&bucket=main&_expected=0
Risk: Medium
Confidence: Medium
Parameter:
Attack:
Evidence: Access-Control-Allow-Origin: *
CWE ID: 264
WASC ID: 14
Source: Passive (10098 - Cross-Domain Misconfiguration)

Description:

Web browser data loading may be possible, due to a Cross Origin Resource Sharing (CORS) misconfiguration on the web server

Other Info:

The CORS misconfiguration on the web server permits cross-domain read requests from arbitrary third party domains, using unauthenticated APIs on this domain. Web browser implementations do not permit arbitrary third parties to read the response from authenticated APIs, however. This reduces the risk somewhat. This misconfiguration could be used attacker to access data that is available in an unauthenticated manner, but which uses some other form of security, such as IP address white-listing.

Solution:

Ensure that sensitive data is not available in an unauthenticated manner (using IP address white-listing, for instance).
Configure the "Access-Control-Allow-Origin" HTTP header to a more restrictive set of domains, or remove all CORS headers entirely, to allow the web browser to enforce the Same Policy (SOP) in a more restrictive manner.

Reference:

https://vulncat.fortify.com/en/detail?id=desc.config.dotnet.html5_overly_permissive_cors_policy

Alert Tags:

Key	Value
OWASP_2021_A01	https://owasp.org/Top10/A01_2021-Broken_Access_Control/
OWASP_2017_A05	https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control.htm

Vulnerable JS Library

URL: http://127.0.0.1:8080/WebGoat/js/libs/jquery.min.js
Risk: Medium
Confidence: Medium
Parameter:
Attack:
Evidence: /*! jQuery v3.4.1
CWE ID: 829
WASC ID:
Source: Passive (10003 - Vulnerable JS Library)

Description:

The identified library jquery, version 3.4.1 is vulnerable.

Other Info:

CVE-2020-11023
CVE-2020-11022

Solution:

Please upgrade to the latest version of jquery.

Reference:

https://blog.jquery.com/2020/04/10/jquery-3-5-0-released/

Alert Tags:

Key	Value
OWASP_2017_A09	https://owasp.org/www-project-top-ten/2017/A9_2017-Using_Components_with_Kno
OWASP_2021_A06	https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/

Absence of Anti-CSRF Tokens

URL: http://127.0.0.1:8080/WebGoat/login
Risk: Low
Confidence: Medium
Parameter:
Attack:
Evidence: <form action="/WebGoat/login" method='POST' style="width: 200px;">
CWE ID: 352
WASC ID: 9
Source: Passive (10202 - Absence of Anti-CSRF Tokens)

Description:

No Anti-CSRF tokens were found in a HTML submission form.
A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform
victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that

Other Info:

No known Anti-CSRF token [anticsrf, CSRFToken, __RequestVerificationToken, csrfmiddlewaretoken, authenticity_token, OWASP_CSRFTOKEN, anoncsrf, csrf_token, _csrf, _
__csrf_magic, CSRF, _token, _csrf_token] was found in the following HTML form: [Form 1: "exampleInputEmail1" "exampleInputPassword1"].

Solution:

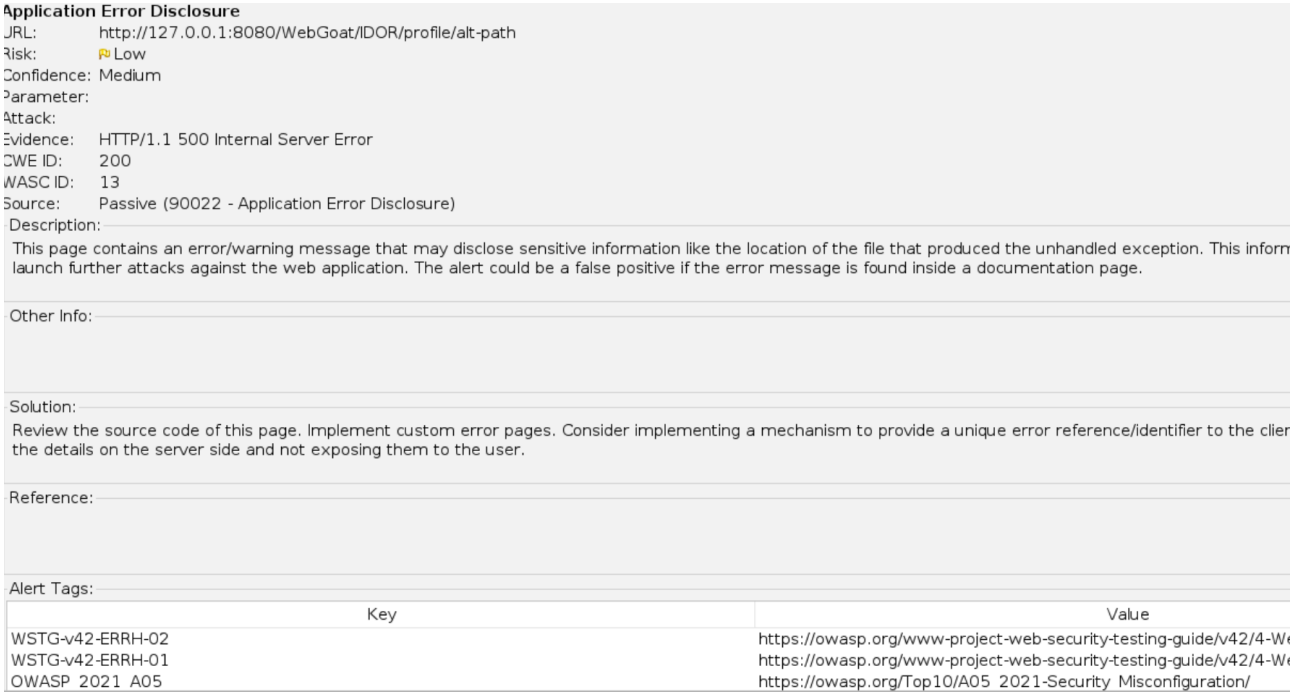
Phase: Architecture and Design
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
For example, use anti-CSRF packages such as the OWASP CSRFGuard.

Reference:

http://projects.webappsec.org/Cross-Site-Request-Forgery
http://cwe.mitre.org/data/definitions/352.html

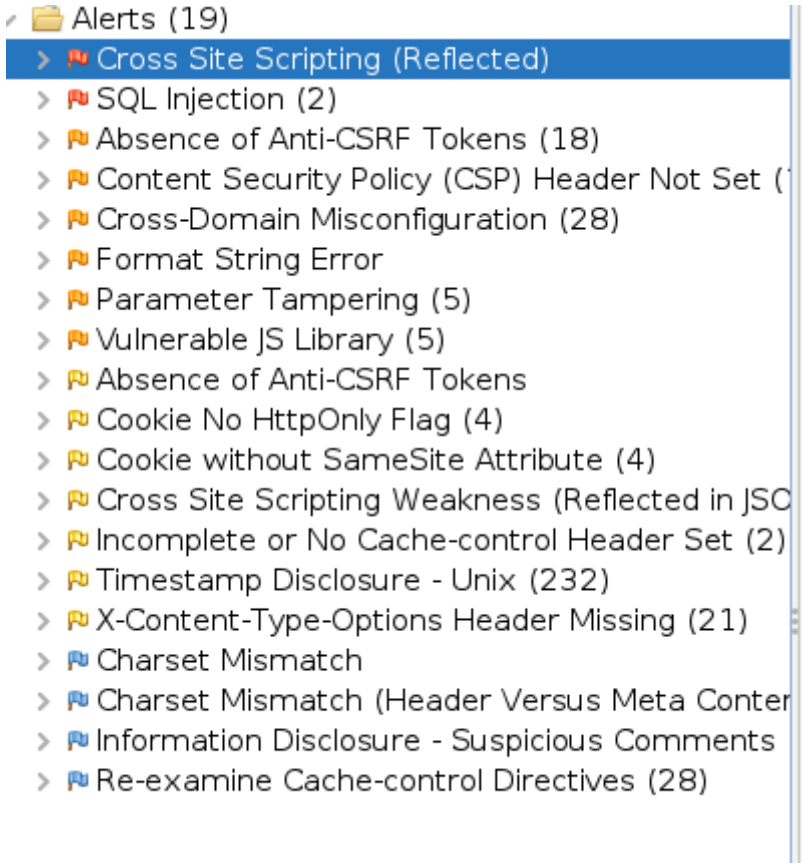
Alert Tags:

Key	Value
OWASP_2021_A01	https://owasp.org/Top10/A01_2021-Broken_Access_Control/
WSTG-v42-SESS-05	https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Appli
OWASP_2017_A05	https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Con



3.3. Post-explotación

Tras realizar en ZAP un escaneo se han detectado 19 alertas. De las cuales 2 son de nivel alto, 6 alertad de nivel medio y 11 son de nivel bajo.



Alertas de nivel alto.

Cross Site Scripting (Reflected)

URL: http://127.0.0.1:8080/WebGoat/CrossSiteScripting/attack5a?QTY1=1&QTY2=1&QTY3=1&QTY4=1&field1=%3Cimg+src%3Dx+onerror%3Dprompt%28%29%3E&field2=111
Risk: High
Confidence: Medium
Parameter: field1
Attack: <img src=x onerror=prompt()>
Evidence: <img src=x onerror=prompt()>
CWE ID: 79
WASC ID: 8
Source: Active (40012 - Cross Site Scripting (Reflected))

Description:

Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.

Other Info:

Solution:

Phase: Architecture and Design
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.

Reference:

http://projects.webappsec.org/Cross-Site-Scripting
http://cwe.mitre.org/data/definitions/79.html

Alert Tags:

Key	Value
OWASP_2021_A03	https://owasp.org/Top10/A03_2021-Injection/
WSTG-v42-INPV-01	https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_...
OWASP_2017_A07	https://owasp.org/www-project-top-ten/2017/A7_2017-Cross-Site_Scripting_(XSS).html

SQL Injection

URL: http://127.0.0.1:8080/WebGoat/access-control/user-hash
Risk: High
Confidence: Medium
Parameter: userHash
Attack: userHash OR 1=1 --
Evidence:
CWE ID: 89
WASC ID: 19
Source: Active (40018 - SQL Injection)

Description:

SQL injection may be possible.

Other Info:

The page results were successfully manipulated using the boolean conditions [userHash AND 1=1 --] and [userHash OR 1=1 --]
The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison
Data was NOT returned for the original parameter.

Solution:

Do not trust client side input, even if there is client side validation in place.
In general, type check all data on the server side.
If the application uses JDBC, use PreparedStatement or CallableStatement, with parameters passed by '?'

Reference:

https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

Alert Tags:

Key	Value
OWASP_2021_A03	https://owasp.org/Top10/A03_2021-Injection/
WSTG-v42-INPV-05	https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_...
OWASP_2017_A01	https://owasp.org/www-project-top-ten/2017/A1_2017-Injection.html

Alertas de nivel medio

Absence of Anti-CSRF Tokens

URL: http://127.0.0.1:8080/WebGoat/login
Risk: Medium
Confidence: Low
Parameter:
Attack:
Evidence: <form action="/WebGoat/login" method='POST' style="width: 200px;">
CWE ID: 352
WASC ID: 9
Source: Passive (10202 - Absence of Anti-CSRF Tokens)

Description:

No Anti-CSRF tokens were found in a HTML submission form.
A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has

Other Info:

No known Anti-CSRF token [anticsrf, CSRFToken, __RequestVerificationToken, csrfmiddlewaretoken, authenticity_token, OWASP_CSRFTOKEN, anoncsrf, csrf_token, _csrf, _csrfSecret, __csrf_magic, CSRF, _token, _csrf_token] was found in the following HTML form: [Form 1: "exampleInputEmail1" "exampleInputPassword1"].

Solution:

Phase: Architecture and Design
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
For example, use anti-CSRF packages such as the OWASP CSRFGuard.

Reference:

http://projects.webappsec.org/Cross-Site-Request-Forgery
http://cwe.mitre.org/data/definitions/352.html

Alert Tags:

Key	Value
OWASP_2021_A01	https://owasp.org/Top10/A01_2021-Broken_Access_Control/
WSTG-v42-SESS-05	https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_...
OWASP_2017_A05	https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control.html

Content Security Policy (CSP) Header Not Set

URL: http://127.0.0.1:8080/WebGoat/login
Risk: Medium
Confidence: High
Parameter:
Attack:
Evidence:
CWE ID: 693
WASC ID: 15
Source: Passive (10038 - Content Security Policy (CSP) Header Not Set)

Description:

Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets,

Other Info:

Solution:

Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header, to achieve optimal browser support: "Content-Security-Policy" for Chrome 25+, Firefox 23+ and Safari 7+, "X-Content-Security-Policy" for Firefox 4.0+ and Internet Explorer 10+, and "X-WebKit-CSP" for Chrome 14+ and Safari 6+.

Reference:

https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing_Content_Security_Policy
https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html
http://www.w3.org/TR/CSP/

Alert Tags:

Key	Value
OWASP_2021_A05	https://owasp.org/Top10/A05_2021-Security_Misconfiguration/
OWASP_2017_A06	https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration.html

Cross-Domain Misconfiguration

URL: https://location.services.mozilla.com/v1/country?key=91e66841-a83b-487f-9b5d-e460f5225ebf
Risk: Medium
Confidence: Medium
Parameter:
Attack:
Evidence: Access-Control-Allow-Origin: *
CWE ID: 264
WASC ID: 14
Source: Passive (10098 - Cross-Domain Misconfiguration)

Description:

Web browser data loading may be possible, due to a Cross Origin Resource Sharing (CORS) misconfiguration on the web server

Other Info:

The CORS misconfiguration on the web server permits cross-domain read requests from arbitrary third party domains, using unauthenticated APIs on this domain. Web browser implementations do not permit arbitrary third parties to read the response from authenticated APIs, however. This reduces the risk somewhat. This misconfiguration could be used by an attacker to access data that is available in an unauthenticated manner, but which uses some other form of security, such as IP address white-listing.

Solution:

Ensure that sensitive data is not available in an unauthenticated manner (using IP address white-listing, for instance).
Configure the "Access-Control-Allow-Origin" HTTP header to a more restrictive set of domains, or remove all CORS headers entirely, to allow the web browser to enforce the Same Origin Policy (SOP) in a more restrictive manner.

Reference:

https://vulncat.fortify.com/en/detail?id=desc.config.dotnet.html5_overly_permissive_cors_policy

Alert Tags:

Key	Value
OWASP_2021_A01	https://owasp.org/Top10/A01_2021-Broken_Access_Control/
OWASP_2017_A05	https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control.html

Format String Error

URL: http://127.0.0.1:8080/WebGoat/access-control/user-hash
Risk: Medium
Confidence: Medium
Parameter: userHash
Attack: ZAP%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s
Evidence:
CWE ID: 134
WASC ID: 6
Source: Active (30002 - Format String Error)

Description:

A Format String error occurs when the submitted data of an input string is evaluated as a command by the application.

Other Info:

Potential Format String Error. The script closed the connection on a /%s

Solution:

Rewrite the background program using proper deletion of bad character strings. This will require a recompile of the background executable.

Reference:

https://owasp.org/www-community/attacks/Format_string_attack

Alert Tags:

Key	Value
OWASP_2021_A03	https://owasp.org/Top10/A03_2021-Injection/
OWASP_2017_A01	https://owasp.org/www-project-top-ten/2017/A1_2017-Injection.html

3.4. Posibles mitigaciones

3.5. Herramientas utilizadas

Las herramientas que se han utilizado para poder realizar la práctica son:

La interfaz gráfica ZAP
Herramientas de Developer