# Rapport : <https://docs.google.com/document/d/1aixZk1XMCT3ZVU2H785c70kICEAaUP8ucUQEvO0xDvA/edit>

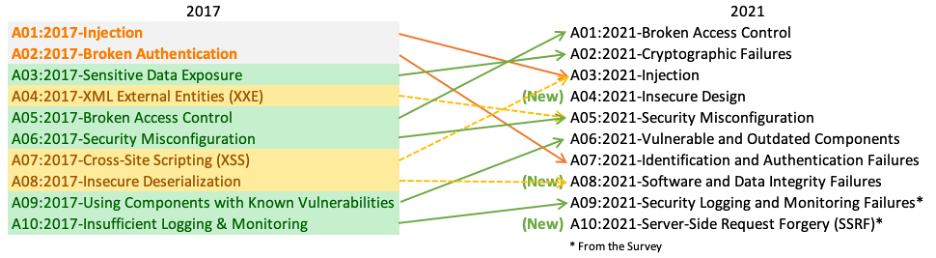
# Repartition de la charge de travail

| Faille | Application vulnérable | Catégorie OWASP | Attaquant |
| --- | --- | --- | --- |
| Injection SQL | WebGoat | Injection (A03) | Ayoub✔️ |
| Spoofing Cookie | Webgoat | Broken Access Control (A01) | Ayoub✔️ |
| Encoding | Webgoat | Cryptographic Failure (A02) | Ayoub✔️ |
| XXE | WebGoat | Security Misconfiguration (A05) | Ayoub✔️ |
| XSS | WebGoat | Injection Cross Site Scripting (A03)  Insecure Design (A04) ? | Ayoub✔️ |
| Vulnerable Component | WebGoat | Vuln&OutdatedComponent (A06) | Jérémie |
| Authentication Bypasses | WebGoat | Identify & auth.Failures(A7) | Jérémie |
| Insecure Deserialization | WebGoat | Software&Data Integrity (A08) | Jérémie |
| Logging Security | WebGoat | Security Logging Failures (A9) | Jérémie |
| Cross-Site Request Forgeries | WebGoat | Server Side Request Forgery (A10) | Jérémie |
| Absence de jeton CSRF | Bwapp | Software&Data Integrity (A08) | Valentine ✔️ |
| Session ID in URL | Bwapp | Identify & auth.Failures(A7) | Valentine ✔️ |
| Insecure DOR (reset secret) | Bwapp | (A04) Insecure Design | Valentine✔️ |
| PHP Eval Function | Bwapp | Vuln&OutdatedComponent (A06) | Valentine✔️ |
| SSRF | Bwapp | Server Side Request Forgery (A10) | Valentine ✔️ |
| Logout Management | Bwapp | Broken Access Control (A01) | Simon ✔️- |
| Base64 Encoding | Bwapp | Cryptographic Failure (A02) | Simon ✔️- |
| Unvalidated Redirects & Forwards | Bwapp | Insecure Design (A04) | Simon ✔️- |
| robots.txt | Bwapp | Security Misconfiguration (A05) | Simon ✔️- |
| OS Command Injection | Bwapp | Injection (A03) | Simon ✔️- |

* [A01 Contrôles d'accès défaillants : 2](https://owasp.org/Top10/fr/A01_2021-Broken_Access_Control/)
* [A02 Défaillances cryptographiques : 1](https://owasp.org/Top10/fr/A02_2021-Cryptographic_Failures/)
* [A03 Injection : 2](https://owasp.org/Top10/fr/A03_2021-Injection/)
* [A04 Conception non sécurisée : 1](https://owasp.org/Top10/fr/A04_2021-Insecure_Design/)
* [A05 Mauvaise configuration de sécurité : 2](https://owasp.org/Top10/fr/A05_2021-Security_Misconfiguration/)
* [A06 Composants vulnérables et obsolètes : 2](https://owasp.org/Top10/fr/A06_2021-Vulnerable_and_Outdated_Components/)
* [A07 Identification et authentification de mauvaise qualité : 2](https://owasp.org/Top10/fr/A07_2021-Identification_and_Authentication_Failures/)
* [A08 Manque d'intégrité des données et du logiciel : 2](https://owasp.org/Top10/fr/A08_2021-Software_and_Data_Integrity_Failures/)
* [A09 Carence des systèmes de contrôle et de journalisation : 0](https://owasp.org/Top10/fr/A09_2021-Security_Logging_and_Monitoring_Failures/)
* [A10 Falsification de requête côté serveur (SSRF): 2](https://owasp.org/Top10/fr/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/)

10 : Val  
9 : Jerem  
8 : Val  
7 : Jerem  
6 : Jerem  
5 : Ayoub  
4 : Simon  
3 : Ayoub  
2 : Simon  
1 : Simon

# Failles



# Bwapp

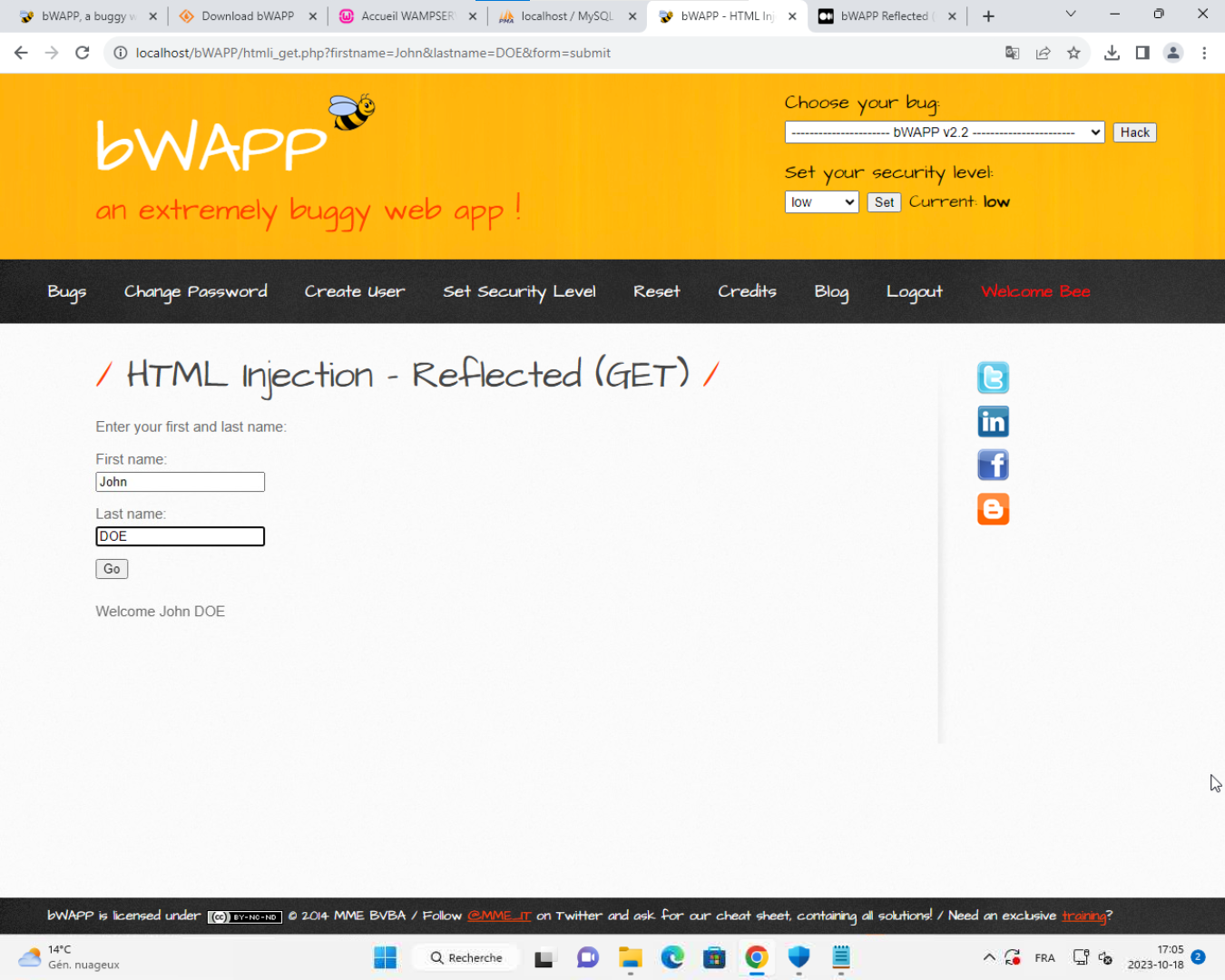
# 

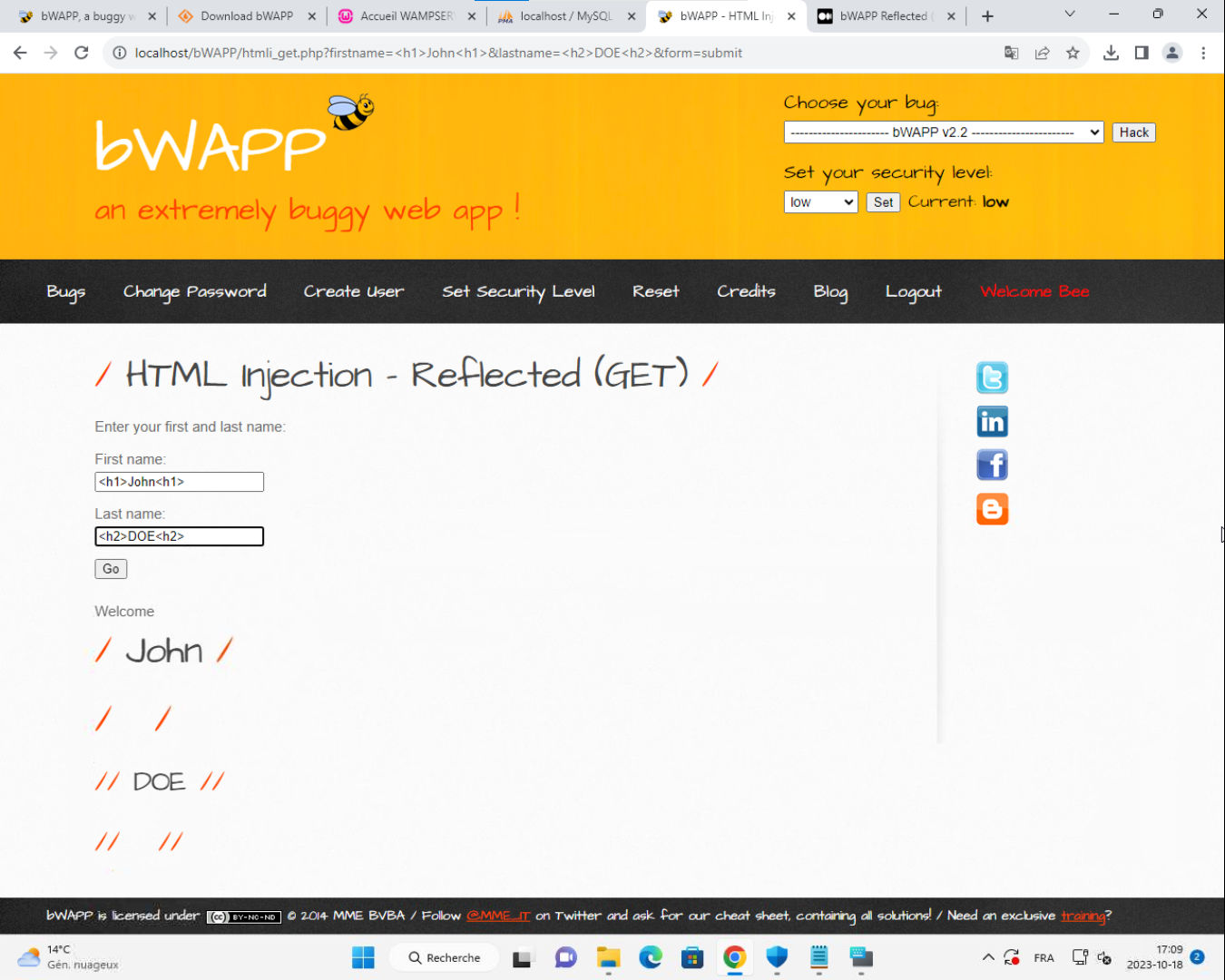
Absence de jetons CSRF

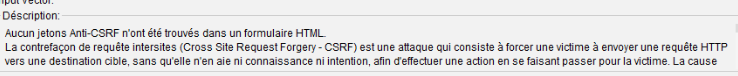
CSP Header not set

# Injection HTML

Low Security : Interprétation du code HTML indiqué dans First et Last Name







Correction des failles

https://cheatsheetseries.owasp.org/cheatsheets/Injection\_Prevention\_Cheat\_Sheet.html

**Installation de WebGoat**

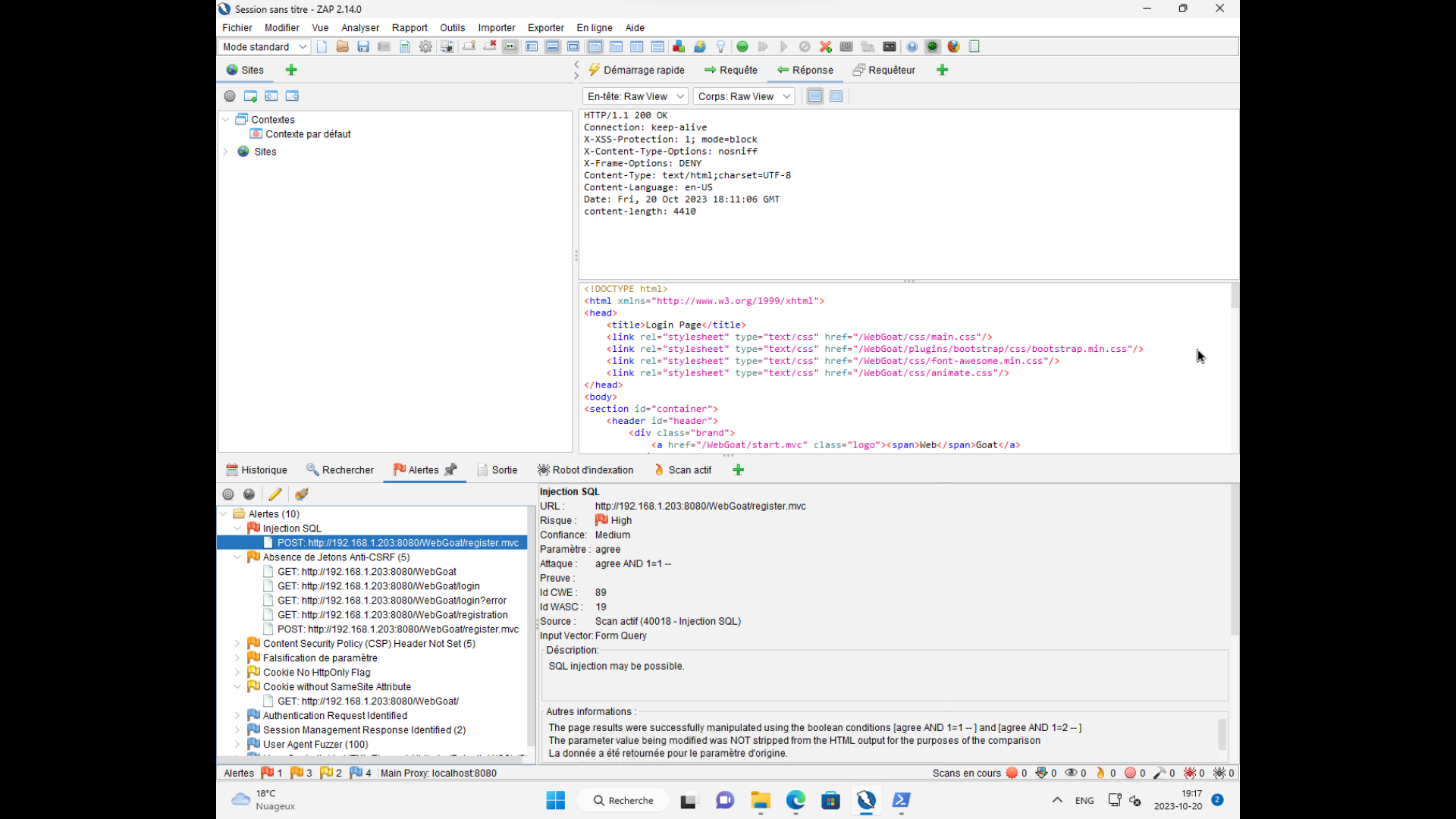
**WebGoat** est facile à installer via l’utilitaire Docker , il faut pull le conteneur depuis le Docker Hub puis le run.

1. **Injection**

**1.Injection SQL**

**Explication**

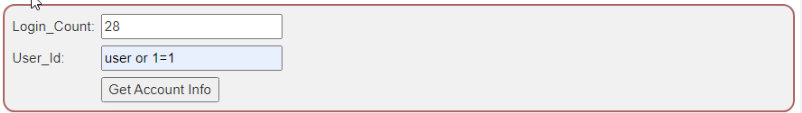
Une faille SQL

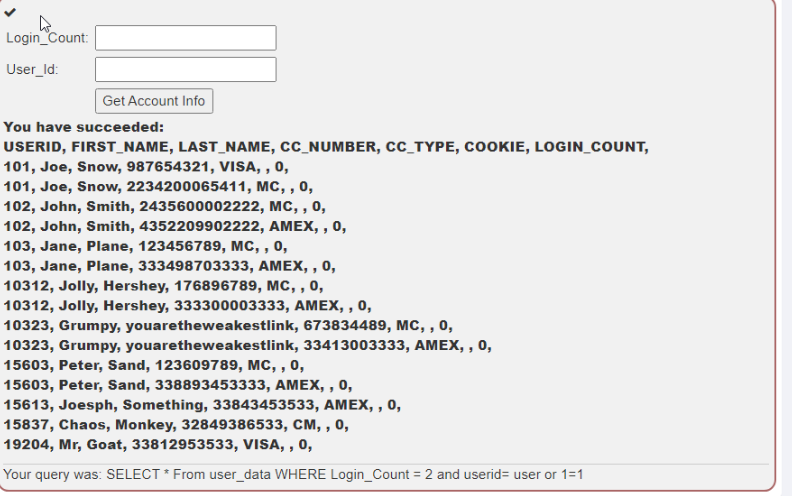
****

**Test de la faille**

Nous allons tester une commande dans le champs (balise form HTML)

Nous avons un form HTML





On récupère les données

**Correction de la faille de sécurité**

Pour corriger la faille on peut avoir des requêtes préparées et accepter uniquement les lettres et les chiffres à l’aide d’expression régulières

requête préparée :

const username\_form= document.getElementbyID(“username”)

const password\_form= document.getElementbyID(“password”)

const regex = **[a-zA-Z0-9]\***

const request = regex.test(username\_form);

const request\_2 = regex.test(password\_form);

const sql = SELECT \* FROM users where username=? AND password= ?

if (request && request 2) {

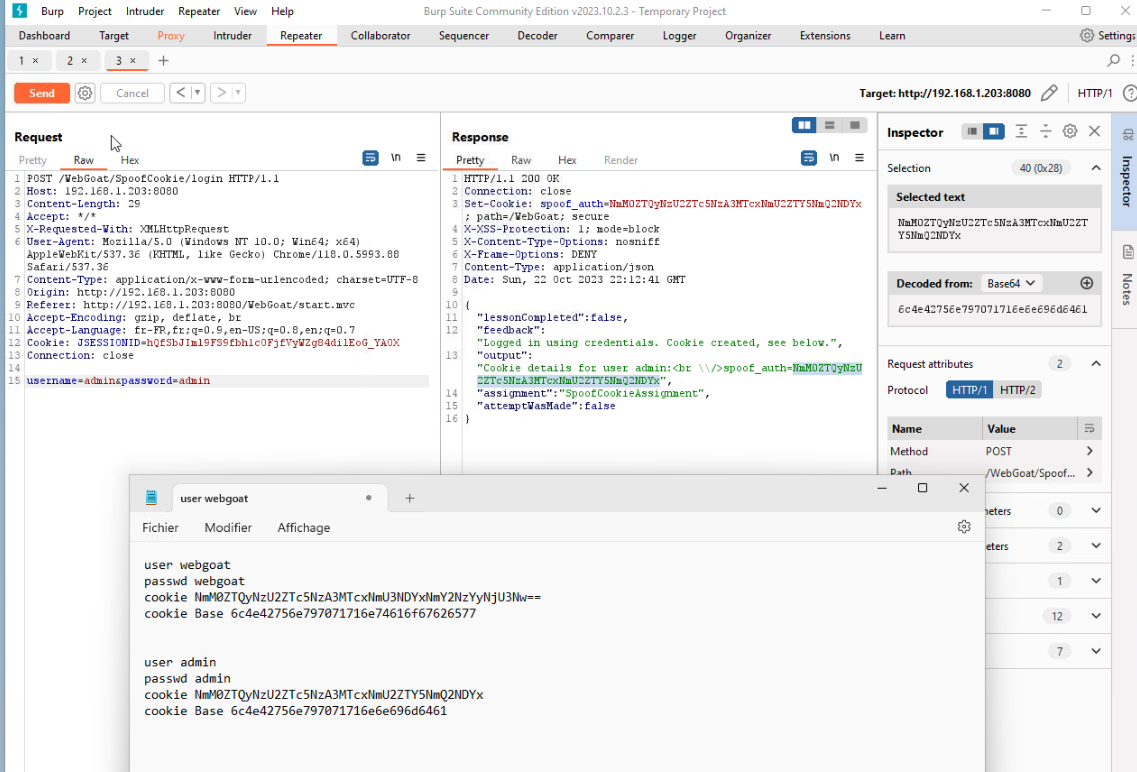
db.query([sql,username\_form,password\_form])

} else {

alert("le username et/ou le mot de passe ne sont pas valide.");

}

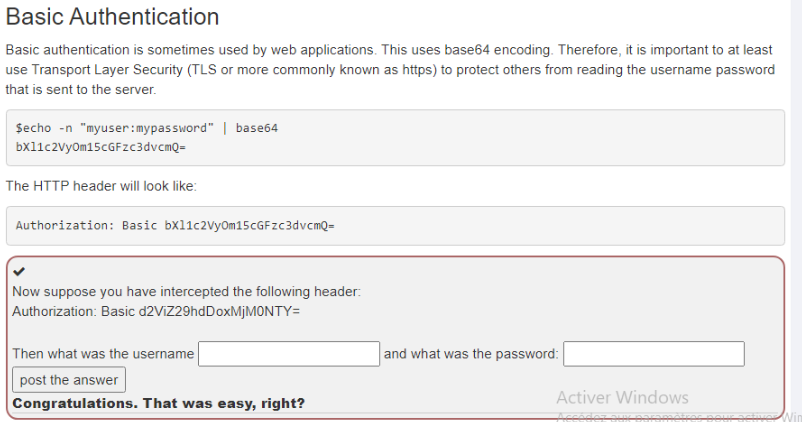
1. **Cookie Spoofing**

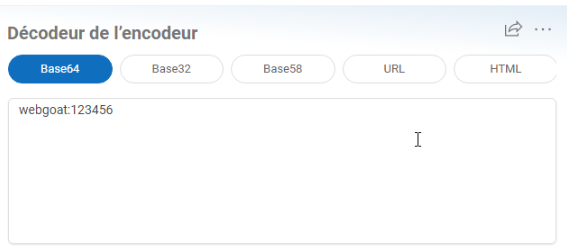
****

1. **Chiffrement et Encodage**

Dans les applications web, l’encodage standard est en base64. Le problème d’un encodage ou chiffrement est qu’il est

**Encoding Base64**

****

****

**Correction de la faille de sécurité :**

Pour fixer cette faille il faut vérifier les hash. Il faut donc hasher le mot de passe avant de l’envoyer et le serveur doit comparer les hash. Cela permet d’éviter que les mots de passe circulent directement.

1. **XXE**

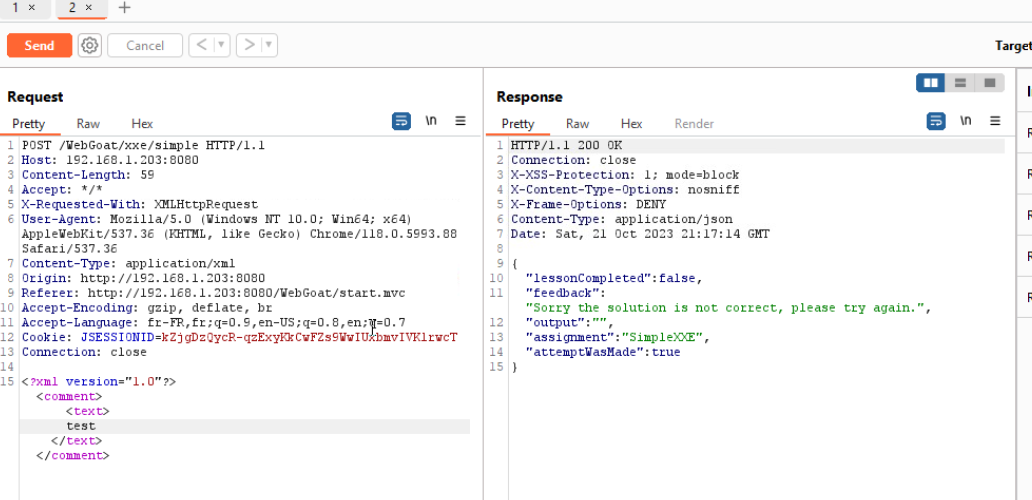
**XML External Entity**

XML, définit comment un document doit être structuré pour l'échange de données.On peut spécifier différents champs de type différents. On envoie des informations en format XML dans des requêtes HTTP (GET/POST). En Modifiant la requête avant de l’envoyer on peut récupérer des informations supplémentaire.

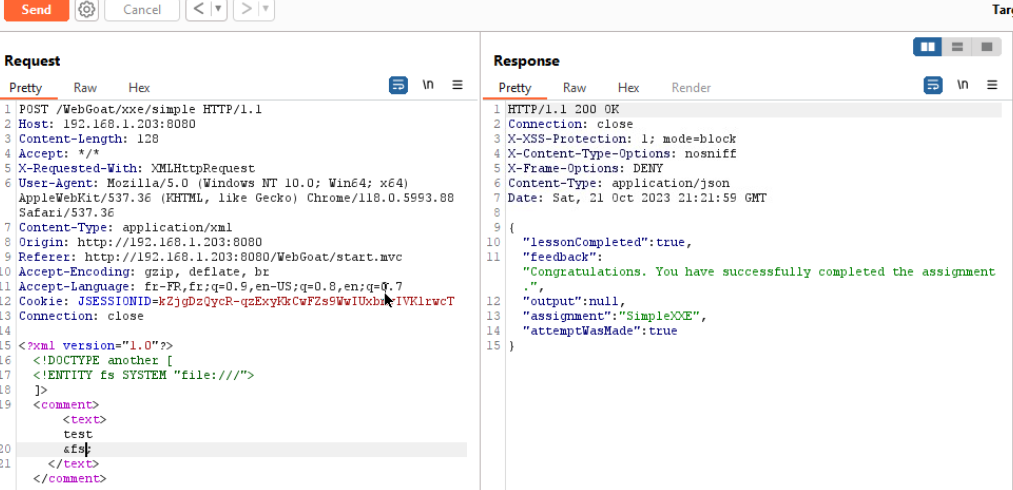
****

On intercepte la requête

On la renvoie dans le repeater



Il reste à ajouter les champs qui nous intéresse

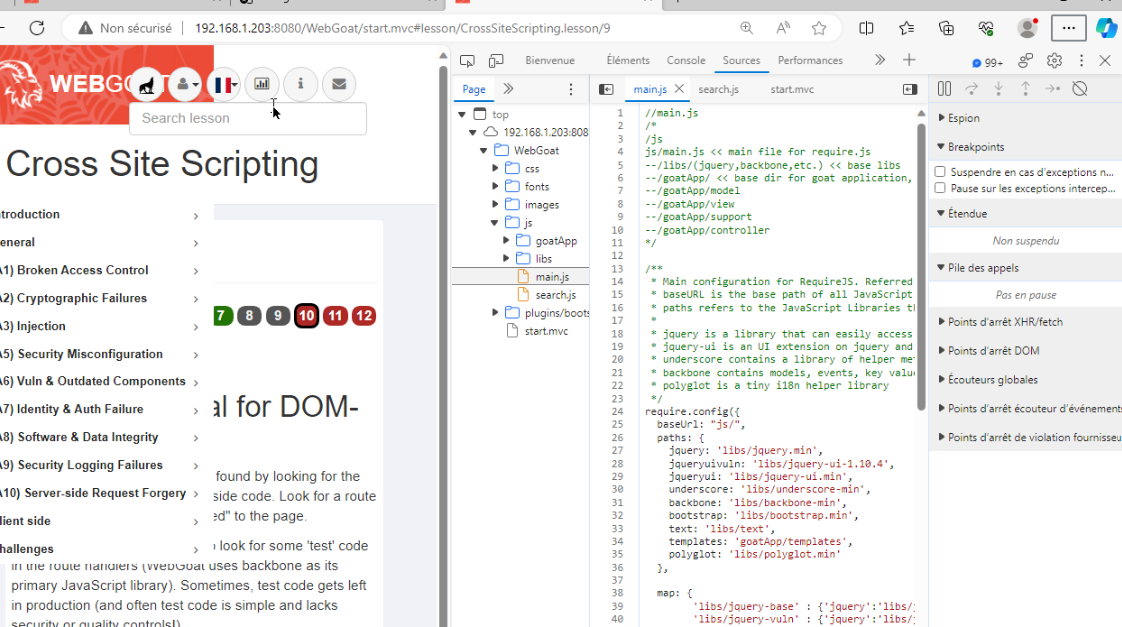


**Correction de la faille de sécurité :**

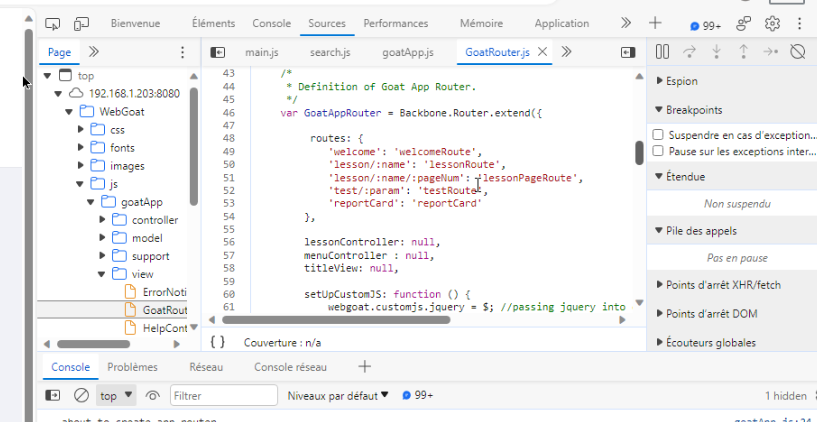
Il faut empêcher

1. **XSS Vulnerability**

Ce sont des failles concernant le Javascript. Il est possible d’exécuter du code javascript pour récupérer des informations auquel on ne devrait pas avoir accès

****

**On récupère un accès aux routes**

****

On peut récupérer des données issue de test (code en production)

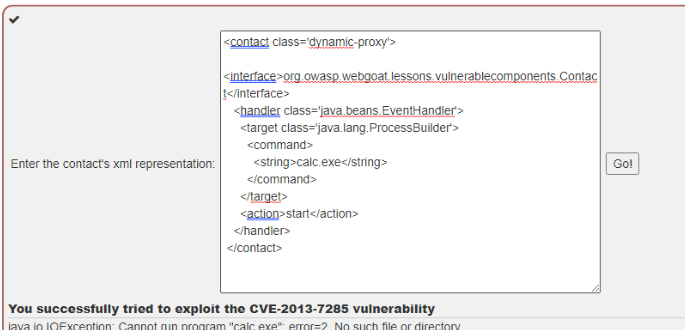
1. **Vulnerable Components**

Cette leçon traite de l'exploitation de la vulnérabilité XStream CVE-2013-7285. Cette vulnérabilité permet une exécution de code à distance, car XStream permet la création d'objets Java arbitraires. Par conséquent, il est possible de créer un java.lang.ProcessBuilder et d'exécuter une commande au sein de l'application Java actuelle.

CVE-2013-7285 est une vulnérabilité de sécurité qui peut être exploitée par des attaquants pour exécuter du code malveillant à distance dans une application Java vulnérable. Pour ce faire, les attaquants peuvent manipuler les données entrantes de manière à créer un objet java.lang.ProcessBuilder, qui peut être utilisé pour exécuter des commandes système potentiellement dangereuses.



En essayant le premier payload nous arrivons à exploiter la faille en créant un nouvel objet. Pour ensuite avoir une RCE (Remote Code Execution) , il va falloir modifier notre objet XML.



Ici nous avons exploité l’objet Contact de telle sorte qu’il exécute une commande sur l'infrastructure attaquée. Cette vulnérabilité est critique car elle permet l'exécution de code sur une machine distante par le biais d’un composant vulnérable. Une des conséquences pourrait être la mise en place d’un Reverse Shell.

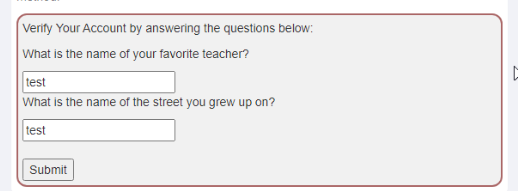
**Correction de la faille de sécurité :**

Pour corriger la faille de sécurité liée à XStream mentionnée (CVE-2013-7285), vous devez mettre à jour XStream vers une version qui corrige la vulnérabilité. La version de XStream 1.4.11 est la première version à corriger cette faille, mais il est recommandé d'utiliser la version la plus récente pour bénéficier de toutes les mises à jour de sécurité et des correctifs.

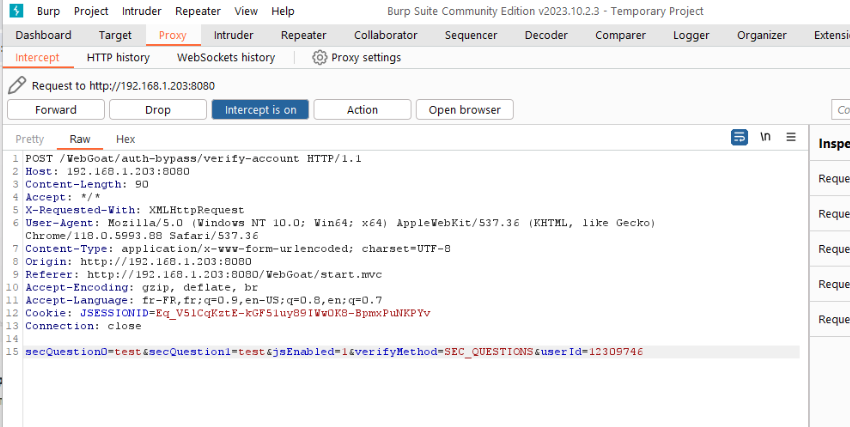
1. **Authentication Bypasses**

Ici nous avons une authentification à deux facteurs avec une question secrète. Nous essayons de passer la question secrète en sniffant la requête et en modifiant les paramètres de la requête.

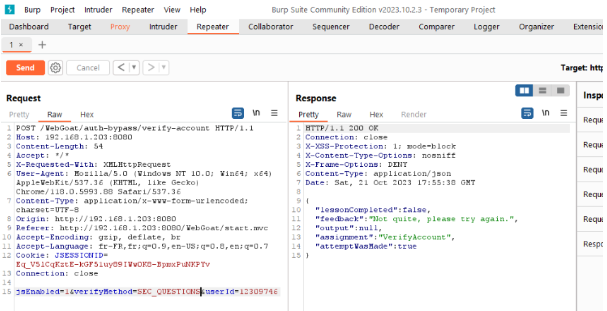
Nous démarrons l’outil Burpsuite pour intercepter la requête quand nous allons appuyer sur le bouton “Submit”.



Voici la requête interceptée :



Comme le suggère l’exemple sur WebGoat nous allons supprimer les paramètres secQuestion dans la requête pour bypasser l’authentification à deux facteurs :



Malheureusement , en regardant le code Java fournit par WebGoat nous remarquons que le serveur regarde si la requête comprend les mots “secQuestion”, si ces mots ne sont pas présents le serveur ne lit pas notre requête.

Réessayons en falsifiant la requête pour que tous les prérequis du serveur soient vérifiés :



Effectivement , on constate que nous avons bien passé les filtres du serveur.

**Correction de la faille de sécurité :**

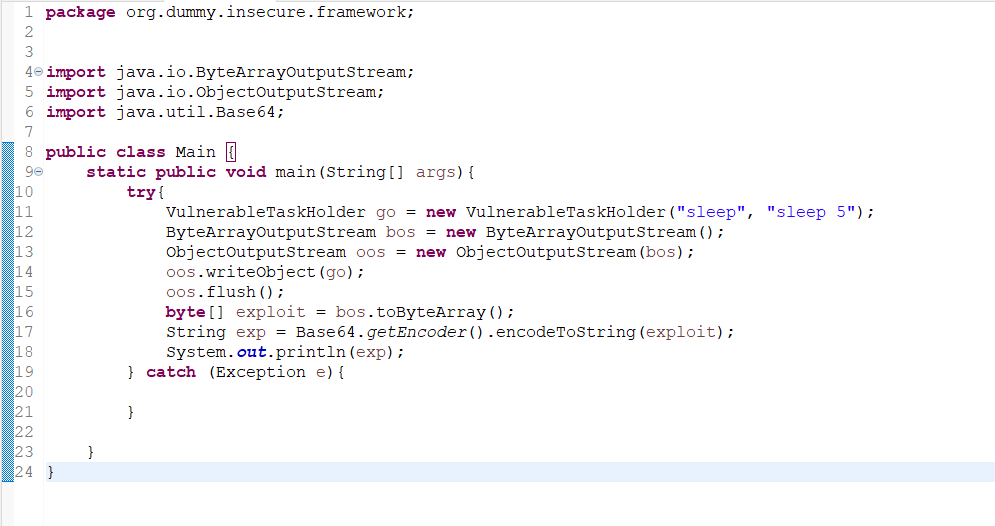
Pour corriger la faille il faut chiffrer la requête.

1. **Insecure Deserialization**

Ici notre objet sérialisé est un String qui va être sérialisé en Objet Java. Il faut que l’on compile notre payload (Classe Java) en base64. Une fois notre payload fabriqué nous allons exploiter la vulnérabilité Insecure Deserialization.

On récupère le payload VulnerableTaskHolder présent sur le Git de WebGoat : [WebGoat/src/main/java/org/dummy/insecure/framework/VulnerableTaskHolder.java at main · WebGoat/WebGoat · GitHub](https://github.com/WebGoat/WebGoat/blob/main/src/main/java/org/dummy/insecure/framework/VulnerableTaskHolder.java)

Une fois le payload récupéré , nous allons créer une classe Main en Java pour créer l’exploit qui va permettre d’exploiter la vulnérabilité Insecure Déserialization.



Ci-dessus nous transformons le payload fournit par WebGoat en l’encodant en base64. Ceci nous permet de créer un token sous forme de String qui est basé sur le temps. Voici le string créé :

rO0ABXNyADFvcmcuZHVtbXkuaW5zZWN1cmUuZnJhbWV3b3JrLlZ1bG5lcmFibGVUYXNrSG9sZGVyAAAAAAAAAAICAANMABZyZXF1ZXN0ZWRFeGVjdXRpb25UaW1ldAAZTGphdmEvdGltZS9Mb2NhbERhdGVUaW1lO0wACnRhc2tBY3Rpb250ABJMamF2YS9sYW5nL1N0cmluZztMAAh0YXNrTmFtZXEAfgACeHBzcgANamF2YS50aW1lLlNlcpVdhLobIkiyDAAAeHB3DgUAAAfnChUPBSQeeZlgeHQAB3NsZWVwIDV0AAVzbGVlcA==

Vu que la chaîne de caractères est basé sur le temps , WebGoat nous demande d’attendre 10 minutes et de re-générer une chaine de caractère pour prouver que nous avons généré le String sur l’instant :



Après 10 minutes on fournit le nouveau token ce qui permet de compléter le challenge.

**Correction de la faille de sécurité :**

L'OWASP indique que la meilleure façon de prévenir les attaques par désérialisation non sécurisée est de ne jamais accepter d'objets sérialisés provenant d'utilisateurs non fiables. En alternative, vous pouvez utiliser des outils de sérialisation qui n'autorisent que les types de données primitifs.

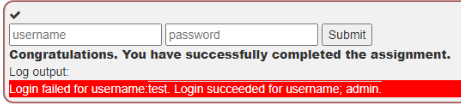
1. **Logging Security**

WebGoat nous fournit un élément HTML Form dans lequel on va essayer d’avoir le bon message de log en faisant du LogSpoofing.

****

Nous arrivons à empoisonner les logs en injectant dans le champs “username” :

“test. Login succeeded for username: admin”

****

Cela permet d’empoisonner les messages de logs pour avoir la sortie espéré.

**Correction de la faille de sécurité :**

Comme vous pouvez le constater, le LogSpoofing peut devenir un problème lorsque les utilisateurs tentent de falsifier les logs. Il existe diverses méthodes pour le faire, en plus d'une soumission de formulaire. Pensez par exemple aux paramètres d'URL ou aux charges JSON élaborées. Par conséquent, il est important de :

* Appliquer une désinfection d'entrée appropriée.
* Vous assurer de pouvoir établir l'authenticité de la source et mettre en place des contrôles d'intégrité pour détecter toute altération des logs.
* Veiller à ce qu'un utilisateur ne puisse pas injecter des logs depuis n'importe quelle source.
* Veiller à ce que le stockage des logs soit protégé.

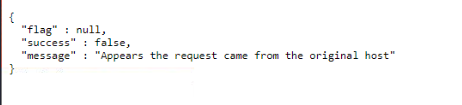
1. **Cross-Site Request Forgeries**

La faille CSRF (Cross-Site Request Forgery) est une attaque qui se produit lorsque un site web malveillant exploite la confiance d'un utilisateur envers un site de confiance pour effectuer des actions non désirées sur ce site de confiance. En d'autres termes, un attaquant incite un utilisateur à effectuer involontairement une action sur un site sans que l'utilisateur en ait conscience. Une attaque CSRF fonctionne car les requêtes du navigateur incluent automatiquement tous les cookies, y compris les cookies de session.

WebGoat nous fournit l’interface suivante pour les CSRF :

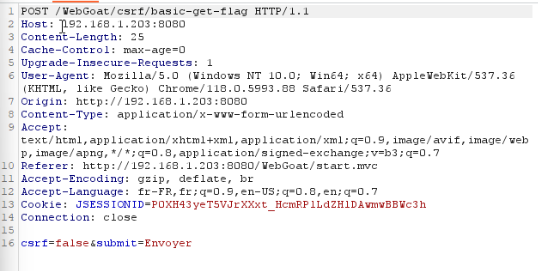


Quand on clique sur envoyer on peut voir un message qui nous dit qu’on a pas réussi le challenge mais qu’il va falloir lancer cette requête pour ouvrir cette page depuis une source inconnue :

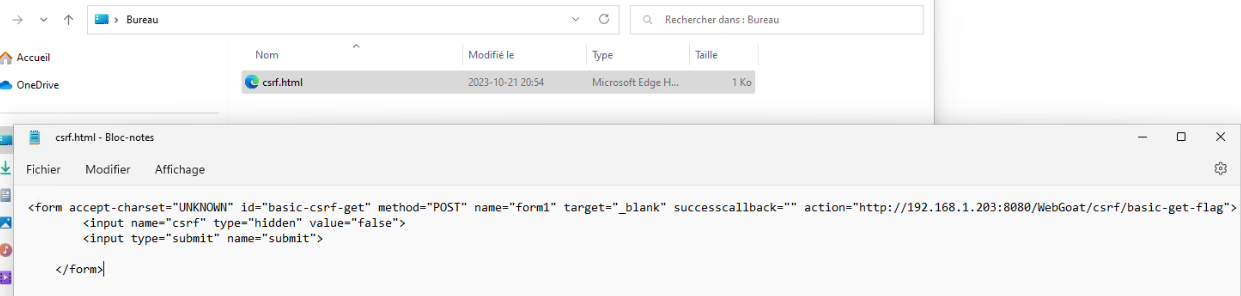


Nous allons créer un nouveau Form HTML , pour falsifier l’accès depuis une source inconnue.

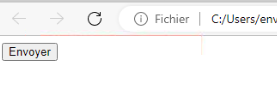
Tout d’abord , voici l’interception de la requête avec BurpSuite:



Nous recréons le Form HTML depuis une source différente :

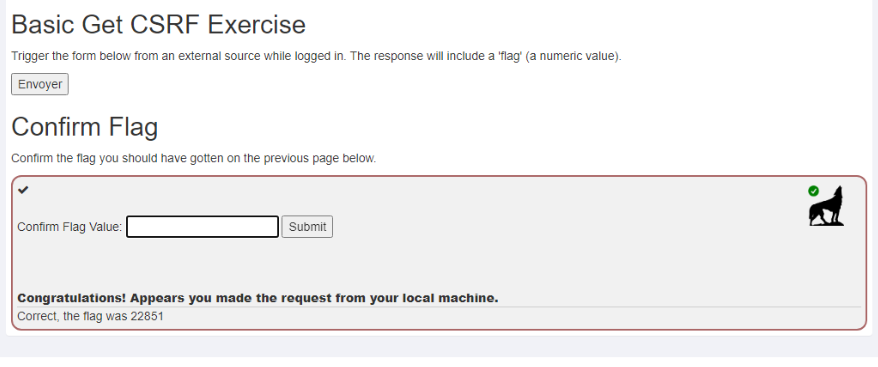


Voici la page Web que l’on obtient :



En appuyant sur “Envoyer” , nous arrivons à faire en sorte que l’utilisateur connecté sur notre page enclenche des actions non-désirés à son insu :





**Correction de la faille de sécurité :**

La technique de correction la plus courante est l’utilisation de token anti-CSRF ce qui permet d’identifier un client légitime du client malveillant.

Un jeton CSRF (Cross-Site Request Forgery) est une valeur permettant de prouver que vous envoyez une requête à partir d'un formulaire ou d'un lien généré par le serveur. En d'autres termes, lorsque le serveur envoie un formulaire au client, il y attache une valeur aléatoire unique (le jeton CSRF) que le client doit renvoyer. Lorsque le serveur reçoit la requête à partir de ce formulaire, il compare la valeur du jeton reçue avec la valeur précédemment générée. Si elles correspondent, il considère que la requête est valide.

| A01: Broken Access Control | Gestion de la déconnexion  Usurpation des cookies |
| --- | --- |
| A02: Cryptographic Failures | Encodage Base64  Hashage |
| A03: Injection | Injection SQL  Injection de commande dans l’OS  Faille XXS |
| A04: Insecure Design | Insecure DOR  Redirections non valides |
| A05: Security Misconfiguration | Robots.txt  XML External Entity |

| A06: Vulnerable and Outdated Components | Fonction eval en PHP  XStream CVE-2013-7285 |
| --- | --- |
| A07: Identification and Authentication Failures | Information de session dans l’URL  Contournement de l’authentification |
| A08: Software and Data Integrity Failures | Cross Site Request Forgery  Désérialisation non sécurisée |
| A09: Security Logging and Monitoring Failures | Sécurité des journaux 1  Sécurité des journaux 2 |
| A10: Server-Side Request Forgery | Scan des ports  Accès aux ressources du réseau interne |