

1 2



9 0

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE
COIMBRA

Segurança em Tecnologias da Informação

Practical Assignment #3

Scenario 1:

Web security testing



Scenario 2:

Web application firewall



2021/2022

Mestrado em Engenharia Informática

PL2 Joana Brás

PL2 Pedro Rodrigues

2021179983

2018283166

joanabras@student.dei.uc.pt

pedror@student.dei.uc.pt

1. Introdução	4
2. Cenário de Rede	4
3. Software	5
4. Instalações e Configurações	6
4.1. OWASP ZAP	6
4.2. OWASP Juice Shop	6
4.3. Apache Web Server (WAF)	6
4.3.1 Configuração Geral	6
4.3.2 ModSecurity	8
5. Web Application Security Testing	9
5.1 OWASP ZAP	10
5.1.1 OWASP ZAP Automated Scan	10
5.1.2 OWASP ZAP Active Scan	12
5.1.3 OWASP ZAP Fuzz Scan	13
5.2 Exploração Manual	15
5.2.1 WSTG-INFO: Information Gathering	15
5.2.1.1 WSTG-INFO-01: Conduct Search Engine Discovery Reconnaissance for Information Leakage	15
5.2.1.2 WSTG-INFO-02: Fingerprint Web Server	16
5.2.1.3 WSTG-INFO-03: Review Webserver Metafiles for Information Leakage	16
5.2.1.4 WSTG-INFO-04: Enumerate Applications on Webserver	17
5.2.1.5 WSTG-INFO-05: Review Webpage Comments and Metadata for Information Leakage	18
5.2.1.6 WSTG-INFO-06: Identify Application Entry Points	19
5.2.1.7 WSTG-INFO-07: Map Execution Paths Through Application	20
5.2.1.8 WSTG-INFO-08: Fingerprint Web Application Framework	20
5.2.1.9 WSTG-INFO-09: Fingerprint Web Application	21
5.2.1.10 WSTG-INFO-10: Map Application Architecture	21
5.2.2. WSTG-CONF: Configuration and Deployment Management Testing	21
5.2.2.1 WSTG-CONF-01: Test Network Infrastructure Configuration	21
5.2.2.2 WSTG-CONF-02: Test Application Platform Configuration	22
5.2.2.3 WSTG-CONF-03: Test File Extensions Handling for Sensitive Information	24
5.2.2.4 WSTG-CONF-04: Review Old Backup and Unreferenced Files for Sensitive Information	24
5.2.2.5 WSTG-CONF-05: Enumerate Infrastructure and Application Admin Interfaces	25
5.2.2.6 WSTG-CONF-06: Test HTTP Methods	25
5.2.2.7 WSTG-CONF-07: Test HTTP Strict Transport Security	26
5.2.2.8 WSTG-CONF-08: Test RIA Cross Domain Policy	26
5.2.2.9 WSTG-CONF-09: Test File Permission	26
5.2.2.10 WSTG-CONF-10: Test for Subdomain Takeover	27
5.2.2.11 WSTG-CONF-11: Test Cloud Storage	27

5.2.3 WSTG-IDNT: Identity Management Testing	27
5.2.3.1 WSTG-IDNT-01: Test Role Definitions	28
5.2.3.2 WSTG-IDNT-02: Test User Registration Process	28
5.2.3.3 WSTG-IDNT-03: Test Account Provisioning Process	29
5.2.3.4 WSTG-IDNT-04: Testing Account Enumeration and Guessable User Account	30
5.2.3.5 WSTG-IDNT-05: Testing for Weak or Unenforced Username Policy	
31	
5.2.4 WSTG-ATHN: Authentication Testing	31
5.2.4.1 WSTG-ATHN-01: Testing for Credentials Transported over an Encrypted Channel	32
5.2.4.2 WSTG-ATHN-02: Testing for Default Credentials	32
5.2.4.3 WSTG-ATHN-03: Testing for Weak Lock Out Mechanism	32
5.2.4.4 WSTG-ATHN-04: Testing for Bypassing Authentication Schema	33
5.2.4.5 WSTG-ATHN-05: Testing for Vulnerable Remember Password	34
5.2.4.6 WSTG-ATHN-06: Testing for Browser Cache Weaknesses	34
5.2.4.7 WSTG-ATHN-07: Testing for Weak Password Policy	35
5.2.4.8 WSTG-ATHN-08: Testing for Weak Security Question Answer	36
5.2.4.9 WSTG-ATHN-09: Testing for Weak Password Change or Reset Functionalities	36
5.2.4.10 WSTG-ATHN-10: Testing for Weaker Authentication in Alternative Channel	36
5.2.5. WSTG-ATHZ: Authorization Testing	37
5.2.5.1 WSTG-ATHZ-01: Testing Directory Traversal File Include	37
5.2.5.1 WSTG-ATHZ-02: Testing for Bypassing Authorization Schema	37
5.2.5.1 WSTG-ATHZ-03: Testing for Privilege Escalation	39
5.2.5.1 WSTG-ATHZ-04: Testing for Insecure Direct Object References	39
5.2.6 WSTG-SESS: Session Management Testing	40
5.2.6.1 WSTG-SESS-01: Testing for Session Management Schema	40
5.2.6.2 WSTG-SESS-02: Testing for Cookies Attributes	40
5.2.6.3 WSTG-SESS-03: Testing for Session Fixation	41
5.2.6.4 WSTG-SESS-04: Testing for Exposed Session Variables	41
5.2.6.5 WSTG-SESS-05: Testing for Cross Site Request Forgery	42
5.2.6.6 WSTG-SESS-06: Testing for Logout Functionality	43
5.2.6.7 WSTG-SESS-07: Testing Session Timeout	43
5.2.6.8 WSTG-SESS-08: Testing for Session Puzzling	44
5.2.7 WSTG-INPV: Input Validation Testing	44
5.2.7.1 WSTG-INPV-01: Testing for Reflected Cross Site Scripting	45
5.2.7.2 WSTG-INPV-05: Testing for SQL Injection	46
5.2.8 WSTG-ERRH: Testing for Error Handling	46
5.2.8.1 WSTG-ERRH-01: Testing for Error Code	47
5.2.8.1 WSTG-ERRH-02: Testing for Stack Traces	47

5.2.9 WSTG-CRYP: Testing for Weak Cryptography	48
5.2.10 WSTG-BUSL: Business Logic Testing	48
5.2.11 WSTG-CLNT: Client-Side Testing	48
5.2.7.1 WSTG-CLNT-01: Testing for DOM-Based Cross Site Scripting	49
6. Web Application Firewall (WAF)	49
7. Conclusão	51
8. Referências	51

1. Introdução

O presente relatório aborda uma série de componentes necessários para o terceiro trabalho prático da disciplina de Segurança em Tecnologias da Informação. Este tem vários objetivos, sendo os principais:

1. Utilização de ferramentas de procura de vulnerabilidades e de penetração a fim de não só perceber como detectar falhas de segurança numa aplicação web mas também como explorar essas falhas e tirar proveito destas. A aplicação web proposta para este trabalho, “Juice Shop”, foi criada pela OWASP (Open Web Application Security Project) com fins educativos, promovendo assim o ensino e a literacia da comunidade, e também servindo para exemplo de como o “ZAP” (software criado pela OWASP) pode ser usado para fazer a análise de vulnerabilidades de um dado sistema. Neste trabalho também é incentivado o uso da distribuição de Linux - “Kali Linux” - pelo facto de já ter pré instaladas múltiplas ferramentas de “penetration testing” que podem ser úteis para a exploração de vulnerabilidades encontradas pelo ZAP. O procedimento de exploração de vulnerabilidades deve seguir dentro do possível as guidelines do [WSTG](#) (Web Security Testing Guide).
2. Após a identificação e exploração de vulnerabilidades através do software disponível para o efeito, é também pedido que seja utilizado o apache server através do seu módulo “ModSecurity” (reverse proxy) como uma “Web Application Firewall” (WAF), a fim de tentar proteger ao máximo as vulnerabilidades encontradas na etapa anterior. O apache server funcionando como WAF deve ser completamente transparente atuando apenas nos casos onde alguma das regras pode causar uma vulnerabilidade na aplicação web (“Juice Shop”) atuando nesses casos bloqueando o ataque.

2. Cenário de Rede

Para efeitos de visualização e melhor percepção do cenário que nos foi proposto, construímos este diagrama simplificado que mostra os principais aspectos do trabalho nomeadamente as máquinas envolvidas para o teste de funcionalidades, o software que foi instalado em cada uma destas e os seus endereços IP.

Por uma questão de performance o OWASP ZAP foi instalado na host machine visto que estando iria necessitar de imensos recursos da máquina tornando o processo de análise bastante lento. Já o Kali Linux ficou numa VM visto que é um sistema operativo diferente pelo que pode correr virtualizado, não necessitando nós de uma máquina física para o efeito. No caso do servidor web a correr a “Juice Shop” optamos por o colocar numa máquina virtual a correr debian, juntamente com o apache server a servir de firewall (WAF), visto que sendo este um proxy e estando a proteger uma só aplicação não nos fez sentido estar a criar uma máquina dedicada. No que toca às ligações entre as máquinas, todas elas encontram-se ligadas à mesma rede “NAT” (criada pelo software de virtualização VMWare).

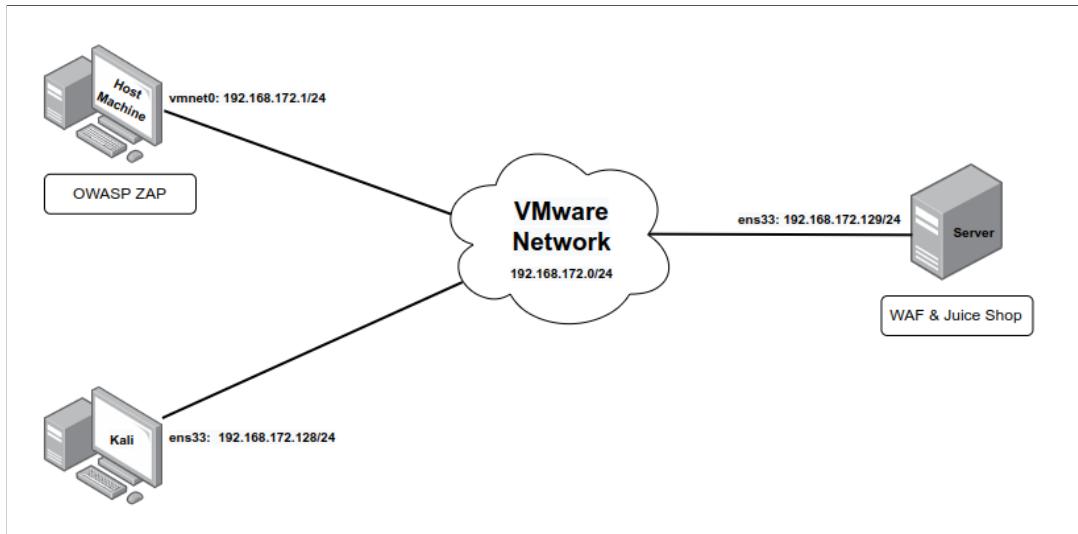


Figura 1 - Cenário de rede

3. Software

Para a implementação das funcionalidades necessárias fizemos uso do seguinte software:

- OWASP ZAP (Host Machine)
- Kali Linux & Penetration Testing Tools (Kali)
- Apache2 Web Server with “ModSecurity” reverse proxy (Server)
- OWASP Juice Shop Web Application (Server)

Este foi instalado nas diversas *Virtual Machines* usadas para a realização deste projeto. O software de virtualização utilizado para a sua criação (como referido anteriormente) foi o "VMware".

4. Instalações e Configurações

De uma forma geral, neste trabalho não foram necessárias grandes configurações visto que foram utilizadas soluções de software cujas configurações default se adaptaram perfeitamente às nossas necessidades para este trabalho, com a exceção do apache web server, cujos ficheiros de configuração tiveram de ser alterados para o efeito. Segue-se o procedimento de configuração de cada uma das componentes.

4.1. OWASP ZAP

No caso do ZAP, após a sua instalação, apenas configuramos nos scans ativos dois módulos adicionais instalados a partir da secção de add-ons: “SQL Injection Scanner”, “Active Scanner Rules”, “FuzzDB Files”, “FuzzDB Ofensive” e “Wappalyzer - Technology Detection” a fim de tentar obter mais informação mais detalhada nos scans ativos que realizamos.

4.2. OWASP Juice Shop

Em termos da aplicação web “OWASP Juice Shop” apenas foi feita a instalação através do software docker e do container existente no [docker hub](#) que já contém a aplicação Web configurada. De seguida foi corrido o container definindo o porto “3000” onde a “Juice Shop” irá escutar por pedidos HTTP. Note-se que através do apache web server que instalamos e estando este a funcionar como proxy, a “Juice Shop” poderá ser acedida através no porto “80” ou “443”. Abaixo encontra-se a configuração efetuada para correr a “Juice Shop”, bem como o processo de instalação através do container.

```
# Install
sudo apt-get install docker docker.io
sudo usermod -aG docker "${USER}"
docker pull bkimminich/juice-shop
# Run
docker run --rm -d - p 3000:3000 bkimminich/juice-shop
```

4.3. Apache Web Server (WAF)

4.3.1 Configuração Geral

Para a configuração da WAF começamos por instalar o apache web server e os módulos necessários para o seu correto funcionamento proxy HTTP. No bloco abaixo encontram-se os comandos efetuados:

```
sudo apt-get install apache2
sudo a2enmod proxy
sudo a2enmod proxy_html
sudo a2enmod proxy_http
sudo systemctl restart apache2
```

Após este processo de instalação dos módulos necessários foi necessário editar alguns ficheiros de configuração do apache para que este funcionasse da forma que pretendíamos. Os ficheiros “`default-ssl.conf`” e “`000-default.conf`”, presentes na pasta cuja diretoria é “`/etc/apache2/sites-available`” são *templates* de suporte para a configuração do Apache. Estes foram *disabled* e os criados (“`apache-ssl.conf`” e “`apache.conf`”) foram *enabled*, isto na mesma pasta mencionada anteriormente. Este passo garantiu que os ficheiros de configuração criados prevaleciam sobre os *default*. Os comandos usados foram os seguintes:

```
sudo a2dissite default-ssl
sudo a2dissite 000-default
sudo a2ensite apache-ssl
sudo a2ensite apache
```

As configurações presentes nos ficheiros referidos anteriormente e que permitem o funcionamento do apache como proxy encontram-se abaixo. Note-se que a aplicação “`Juice Shop`” não se encontra preparada para funcionar em HTTPS, no entanto optamos também por editar o ficheiro de configuração do apache que adiciona suporte para SSL, caso seja necessário mais tarde. Além disso, note-se que a “`Juice Shop`” no nosso cenário de rede encontra-se a funcionar na mesma máquina que o apache pelo que o redirect é feito para o URL “`http://localhost:3000/`”.

- No ficheiro “`apache.conf`”:

```
<VirtualHost *:80>
    ProxyPass / http://localhost:3000/
    ProxyPassReverse / http://localhost:3000/

    # Regras da WAF
    SecRuleEngine On
</VirtualHost>
```

- No ficheiro “`apache-ssl.conf`”:

```
<IfModule mod_ssl.c>
    <VirtualHost *:443>
        ProxyPass / http://localhost:3000/
        ProxyPassReverse / http://localhost:3000/

        # Regras da WAF
        SecRuleEngine On
    </VirtualHost>
</IfModule>
```

4.3.2 ModSecurity

Após a instalação do apache web server e sua configuração com proxy HTTP passamos à instalação do módulo “ModSecurity” a fim de implementar as funcionalidades de WAF. No snippet seguinte encontram-se os passos/comandos necessários à instalação deste módulo.

```
sudo apt-get install libapache2-mod-security2 -y
sudo a2enmod headers
sudo systemctl restart apache2
```

Estando instalado o módulo necessário para o funcionamento do apache como WAF apenas restou configurar a firewall à medida do necessário. Para a sua configuração começamos por copiar os conteúdos presentes no ficheiro de template “[/etc/modsecurity/modsecurity.conf-recommended](#)” para um novo ficheiro de configurações “[/etc/modsecurity/modsecurity.conf](#)”. De seguida procedemos à edição deste ficheiro, alterando o seguinte parâmetros para dar enable/disable das regras firewall à medida das necessidades.

```
# -- Rule engine initialization -----
# Enable ModSecurity, attaching it to every transaction. Use detection
# only to start with, because that minimises the chances of post-installation
# disruption.

# To turn off the firewall blocking features, change this setting to "DetectionOnly"
SecRuleEngine On
```

Tendo o modsecurity ativo, também procedemos à instalação de um conjunto de regras desenhado pela OWASP, o “[OWASP-CRS](#)”, para tal, efetuamos as configurações que se encontram descritas no snippet abaixo.

```

# Remove default "Core Rule Set"
sudo rm -rf /etc/modsecurity/rules

# Install git and clone "OWASP-CRS"
sudo apt-get install git
sudo cd /etc/modsecurity
sudo git clone https://github.com/coreruleset/coreruleset

# Setup the new CRS
sudo mv coreruleset/crs-setup.conf.example crs/crs-setup.conf
sudo mv coreruleset/rules ..

```

Para ativação destas mesmas regras foi ainda necessário alterar outros ficheiros de configuração nomeadamente:

- Nos ficheiros “`apache-ssl.conf`” e “`apache.conf`” para habilitar a utilização das regras na WAF, adicionamos a linha “`SecRuleEngine On`” à semelhança do que foi mostrado anteriormente na configuração do ModSecurity.
- No “`/etc/apache/mods-available/security2.conf`” a fim de incluir o novo rule set acrescentamos a seguinte configuração:

```

<IfModule security2_module>
    SecDataDir /var/cache/modsecurity
    Include /etc/modsecurity/*.conf
    Include /etc/modsecurity-crs/rules/*.conf
</IfModule>

```

- No “`/etc/modsecurity/crs/crs-setup.conf`” alteramos o nível de “paranoia” do novo rule set para 3. Este nível permite definir a agressividade das regras utilizadas na filtragem dos pedidos. Achamos que 3 era a melhor setting visto que um valor superior poderia contribuir para que fossem rejeitados pedidos fidedignos. Abaixo é apresentada a configuração colocada.

```

SecAction \
    "id: 9000", \
    phase:1 \
    nolog, \
    pass, \
    tnone, \
    setvar:tx.paranoia_level=3"

```

5. Web Application Security Testing

Tendo instalado todo o software necessário para a realização do trabalho passamos

então à fase de testagem da aplicação web seguindo as guidelines propostas pelo WTSG. Começamos numa primeira fase por nos servir do ZAP onde através de múltiplos scans e fuzzing conseguimos não só mapear toda a aplicação web mas também encontrar pontos fracos na aplicação que eram mais ou menos suscetíveis de serem atacados por pessoas experientes na área. Após o mapeamento e exploração inicial procedemos a uma análise mais manual onde utilizamos na mesma o ZAP e outras ferramentas que achamos relevantes para, percorrendo todos os pontos relevantes no WTSG V4.1, tentar efetuar/explorar as falhas encontradas previamente contribuindo com exemplos relevantes para numa fase seguinte, estando a WAF implementada, tentar novamente avaliar o desempenho desta face às vulnerabilidades encontradas.

5.1 OWASP ZAP

O ZAP (Zed Attack Proxy), como referido anteriormente, é uma ferramenta desenvolvida pela OWASP com o fim de detectar vulnerabilidades em aplicações web. Nas seções seguintes vamos apresentar os resultados obtidos da utilização dos múltiplos scanners presentes nesta aplicação. Começaremos pelos resultados preliminares obtidos através de um “[Automated Scan](#)”, que mapeia todo o esqueleto da aplicação pesquisa usando a “[Spider](#)” potenciais vulnerabilidades. De seguida analisaremos os resultados obtidos após efetuar um “[Active Scan](#)”, que faz uso de múltiplas queries (algumas sendo add-ons que instalamos) para tentar identificar vulnerabilidades na aplicação. Por último apresentaremos os resultados obtidos quando feito um “[Fuzz Scan](#)”, cujo principal objetivo é tentar invocar “undefined behaviour” na aplicação que a compromete de alguma forma, permitindo escalar privilégios.

5.1.1 OWASP ZAP Automated Scan

Após efetuar o automated scan à “[Juice Shop](#)” o ZAP construiu um mapa de toda a aplicação e encontrou diversas situações de risco com maior ou menor gravidade, recorrendo à ferramenta “[Spider](#)”. Alguns dos problemas mais graves encontrados, entre outros foram:

- 1. SQL Injection:** Este foi o problema mais grave encontrado pelo ZAP durante o scan efetuado. O ataque em questão explora o facto de queries à base de dados utilizada pela aplicação não estarem suficientemente protegidas, sendo possível interagir diretamente com estas e adulterando o seu funcionamento original. Efetivamente, o ZAP detectou algumas situações onde, especialmente em campos de login/inserção de texto, era possível utilizar técnicas de SQL injection ultrapassando as proteções e accedendo a regiões protegidas. Uma das strings utilizada no automated scan que expôs a vulnerabilidade foi “[`\(``](#)”.

2. **Cross-Domain Misconfiguration:** Este problema pertence à categoria de problemas de risco médio encontrados durante o scan. O ataque em questão explora o facto da aplicação permitir que diversas páginas (com diferentes domínios) possam enviar e receber pedidos entre elas sem haver quaisquer restrições acerca de entre quais domínios este comportamento é desejável. Isto pode ser restringido/mitigado utilizando headers HTTP criados para o efeito (CORS headers) de acordo com a política pretendida. Efetivamente durante o scan foram encontradas várias evidências onde o header necessário para esta proteção permite um acesso generalizado a qualquer domínio. “`Access-Control-Allow-Origin: *`”.
3. **Content Security Policy (Header Not Set):** Tal como o problema anterior, este problema também apresenta risco médio. A potencial vulnerabilidade aqui explorada prende-se com o facto de não estar definida uma política de controlo dos recursos. Desta forma, podem ser carregados/executados, inadvertidamente, scripts (ou outro conteúdo) de domínios permitidos pela aplicação, contendo conteúdo malicioso através do browser do cliente. Através da inclusão do header “`Content-Security-Policy:`” e explicitando a política de segurança, podem ser assim mitigados ataques de XSS que tipicamente ocorrem nestas situações. Efetivamente, durante o scan o ZAP encontrou diversas situações onde este header não estava definido e onde ocorriam acessos a conteúdos de outros URLs. Outros problemas relacionados com a falta deste header também foram encontrados como é possível visualizar nas tabelas da seção seguinte.
4. **Missing Anti-clickjacking Header:** Este problema também é detectado com sendo de risco médio. A vulnerabilidade que pode ser explorada à semelhança da anterior prende-se com o facto de não estar incluído um header que define a forma certas componentes da página devem ser rendered (como por exemplo frames). O header que está em falta é o “`X-Frame-Options`” e que em diversos exemplos o ZAP detetou que não estava presente.
5. **Session ID in URL Rewrite:** Por fim temos este problema, também de nível médio, também detectado pelo ZAP. A potencial vulnerabilidade explorada nesta situação está relacionada com o facto de na aplicação ser possível identificar num URL campos como “`userid`” o que permite a um potencial atacante ter acesso por exemplo a tokens de sessão, podendo-se fazer passar desta forma por outro utilizador e efetuar operações na conta deste sem necessitar de login ou senha do mesmo.

Todos estes problemas foram emitidos como alertas durante a primeira fase do scan, de seguida foi realizado um scan ativo que evidenciou ainda mais algumas destas vulnerabilidades, e por vezes ainda encontrou novas.

5.1.2 OWASP ZAP Active Scan

Após o ZAP ter mapeado completamente toda a aplicação, tentamos utilizar a ferramenta de scan ativo a fim de explorar as falhas expostas no passo anterior. No fim de cada uma das nossas tentativas o ZAP construiu uma relatório com as falhas encontradas, evidências (sob forma de ataques que tentou realizar) e uma pequena descrição dos problemas com sua resolução.

O relatório completo com todas as vulnerabilidades encontradas segue anexado com este documento, no entanto as imagens seguintes retiradas do relatório exemplificam os problemas encontrados pelo ZAP num dos nossos scans.

Summary of Alerts

Risk Level	Number of Alerts
High	3
Medium	12
Low	9
Informational	8

Figura 2 - Sumário das falhas encontradas pelo ZAP no Active Scan

Alerts

Name	Risk Level	Number of Instances
Advanced SQL Injection - AND boolean-based blind - WHERE or HAVING clause	High	2
Cloud Metadata Potentially Exposed	High	1
SQL Injection - SQLite	High	2
Absence of Anti-CSRF Tokens	Medium	2
CSP: Wildcard Directive	Medium	15
CSP: script-src unsafe-inline	Medium	2
CSP: style-src unsafe-inline	Medium	8
Content Security Policy (CSP) Header Not Set	Medium	1412
Cross-Domain Misconfiguration	Medium	345
HTTP Only Site	Medium	1
Missing Anti-clickjacking Header	Medium	1165
Proxy Disclosure	Medium	404
Session ID in URL Rewrite	Medium	2214
Vulnerable JS Library	Medium	1
XSLT Injection	Medium	30
Application Error Disclosure	Low	13
CSP: Notices	Low	9
Cookie No HttpOnly Flag	Low	11
Cookie with SameSite Attribute None	Low	12
Cookie without SameSite Attribute	Low	1
Cross-Domain JavaScript Source File Inclusion	Low	372
Private IP Disclosure	Low	9
Timestamp Disclosure - Unix	Low	105
X-Content-Type-Options Header Missing	Low	4137
Charset Mismatch	Informational	1
Content Security Policy (CSP) Report-Only Header Found	Informational	5
Cookie Slack Detector	Informational	105
Information Disclosure - Sensitive Information in URL	Informational	19
Information Disclosure - Suspicious Comments	Informational	41
Loosely Scoped Cookie	Informational	13
Re-examine Cache-control Directives	Informational	26
User Agent Fuzzer	Informational	2422

Figura 3 - Tabela com todas falhas encontradas pelo ZAP no Active Scan (tipo e risco)

5.1.3 OWASP ZAP Fuzz Scan

Por fim, antes de avançar para uma exploração mais manual resolvemos utilizar ainda a funcionalidade de fuzzer do ZAP a fim de explorar algumas secções da aplicação que continham uma componente de interação com o utilizador. Tentando atacar o login, por exemplo, através de bases de dados de palavras passe, nomes de utilizador / email (adquiridas a partir dos add-ons) conseguimos obter a senha de acesso de uma das contas de administrador. Para mais, tentamos ainda através do fuzzer e das bases de dados de SQL queries disponíveis detectar diversas situações onde era possível explorar os ataques de SQL injection referidos anteriormente, no entanto em muitas situações a própria aplicação era capaz de

bloquear os ataques. Em suma, esta exploração através do Fuzzer deu-nos alguma intuição para a exploração manual que efetuamos.

Como exemplos de “fuzzes” que efetuamos, mostramos abaixo os resultados obtidos no processo de descoberta da password de um administrador e SQL injection num campo de login. A descoberta do nome deste usuário foi feita empiricamente, visualizando na aplicação uma “review” a um sumo na qual constava o seu email “admin@juice-sh.op”. A password encontrada neste processo foi “admin123”.

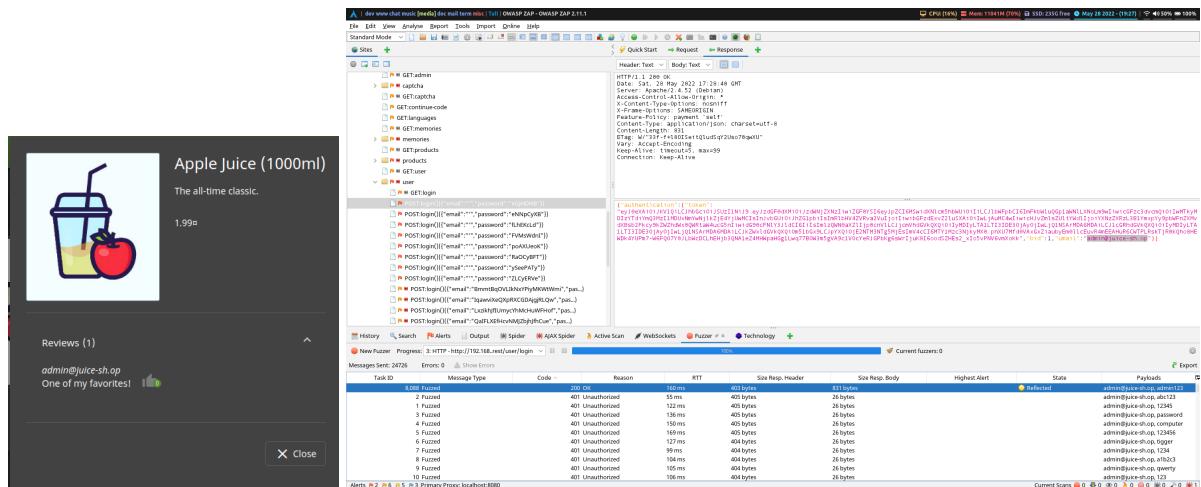


Figura 4 - Resultado do Fuzz Attack - Descoberta de password de um admin user

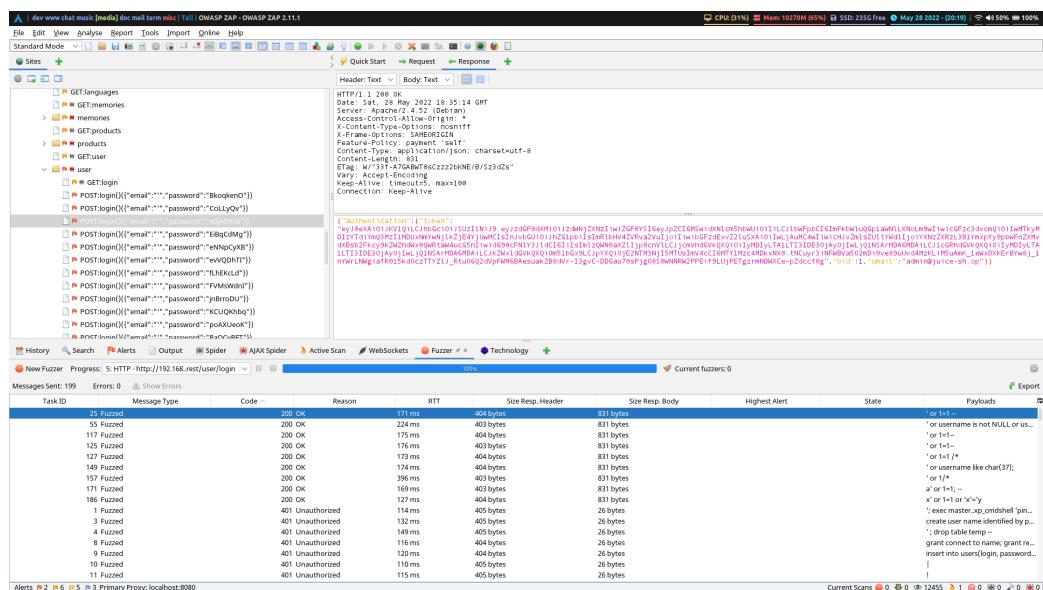


Figura 5 - Resultado do Fuzz Attack - SQL Injection

5.2 Exploração Manual

Tendo efetuado uma análise exploratória com o ZAP, passamos então a uma análise mais manual em que “metemos as mãos” nas várias componentes da aplicação, numa tentativa de melhor identificar as falhas de segurança já descobertas na etapa anterior e em certos casos encontrar evidências concretas que possam ser utilizadas para numa fase posterior como apoio à correção das vulnerabilidades encontradas na aplicação web “Juice Shop”. Para melhor estruturar e guiar o nosso processo de recolha de evidências e testagem, fizemos uso de um conjunto de guidelines propostas pelo WSTG no documento com o mesmo nome, versão 4.1.

Os subcapítulos seguintes mostram os resultados obtidos a partir da nossa análise de cada um dos pontos tocados neste documento, sempre que tal foi possível e pertinente no contexto da aplicação que estamos a analisar (Juice Shop).

5.2.1 WSTG-INFO: Information Gathering

Esta secção toca em todos os aspectos relacionados com a recolha de informação que é possível fazer e que está relacionada com a sua arquitetura e componentes/mecanismos internos. Sobretudo pretende-se estudar a robustez da aplicação no que toca ao conhecimento que um atacante pode adquirir sobre o funcionamento da plataforma e que potenciem o encontro de vulnerabilidades.

5.2.1.1 WSTG-INFO-01: Conduct Search Engine Discovery Reconnaissance for Information Leakage

Neste tópico o principal objetivo prendia-se com a análise da capacidade de motores de busca serem capazes de indexar os vários recursos dentro da aplicação web e construir uma espécie de “mapa” desta. Este tipo de indexação feita pelos motores de busca, tipicamente conhecida como “crawling”, pode ser de certa forma mitigada através da criação de um ficheiro especial na aplicação web denominado de “`robots.txt`”. Este ficheiro é responsável por listar todos os “paths” dentro da aplicação em que não seja desejado que um motor de busca possa indexar, com vista a proteger secções privadas da aplicação. Caso este trabalho não seja realizado corremos o risco de deixar a público informação valiosa acerca da aplicação que pode ser explorada por potenciais atacantes.

No caso da nossa aplicação, estando esta a correr num ambiente controlado e isolado da internet, não é possível serem utilizados os diversos motores de busca existentes por forma a efetuar “crawling” à “Juice Shop” pelo que no nosso caso podemos dar este tópico como **não aplicável** no contexto do nosso trabalho.

5.2.1.2 WSTG-INFO-02: Fingerprint Web Server

O “[Fingerprinting](#)” de um web server consiste na obtenção de informação acerca das componentes internas que permitem o serviço funcionar. Desta forma a identificação das versões de frameworks, livrarias, serviços externos, etc..., faz parte da tarefa de “[Fingerprinting](#)”. Esta informação pode tornar-se relevante para um potencial atacante, visto que, sabendo que porventura uma dada ferramenta numa versão em particular apresenta uma vulnerabilidade e sabendo que essa mesma ferramenta está a ser utilizada no âmbito da aplicação web, a aplicação fica por consequência automaticamente vulnerável.

Neste caso pretende-se obter informação acerca do web server que está a ser utilizado para dar host à aplicação “[Juice Shop](#)”. Efetuando um pedido HTTP à aplicação e observando a resposta dada facilmente obtemos a informação de que o servidor a ser utilizado no nosso caso é o [servidor Apache](#). Também obtemos outras informações relevantes como a sua versão e o sistema operativo onde este está a correr (Debian GNU/Linux). De facto visto que no nosso cenário de rede temos o Apache a funcionar como proxy entre o cliente e a aplicação que está a correr no container, faz sentido que seja este o detectado. Na imagem abaixo mostramos o header da resposta obtida quando fizemos o pedido HTTP (“[GET http://192.168.172.129/ HTTP/1.1](#)”).

```
HTTP/1.1 200 OK
Date: Sun, 29 May 2022 10:22:03 GMT
Server: Apache/2.4.52 (Debian)
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Sun, 29 May 2022 10:14:26 GMT
ETag: W/"7c3-1810f4f5734"
Content-Type: text/html; charset=UTF-8
Content-Length: 1987
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
```

Figura 6 - Resposta ao pedido HTTP efetuado à aplicação. É possível observar algumas informações relativas ao servidor web que está a ser utilizado (3 linha),

5.2.1.3 WSTG-INFO-03: Review Webserver Metafiles for Information Leakage

Como já referido anteriormente, as aplicações web por norma guardam um ficheiro denominado “[robots.txt](#)” que contém uma lista de caminhos dentro da aplicação que não se deseja ver indexados ([crawled](#)) por motores de busca. No entanto, caso

um atacante tenha acesso a este ficheiro pode facilmente descobrir estas regiões e dirigir o seu ataque para as mesmas (visto que certamente devem ter informações sensíveis). Este ficheiro pode ser facilmente obtido fazendo um `wget`, `curl` ou até mesmo usando um browser (inspect sources) caso as devidas precauções não sejam tomadas. Também devemos realçar que este ficheiro não é a melhor forma para definir restrições acerca do conteúdo pode/deve ser acedido por terceiros, pelo facto de obviamente dar imensa informação acerca do esqueleto da aplicação.

No caso da “Juice Shop”, conseguimos obter facilmente o ficheiro “`robots.txt`” da forma que foi descrita acima. Após uma análise deste, descobrimos que existe uma região na aplicação “`/ftp`” que não seria indexada pelos motores de busca. No entanto, tentando manualmente aceder a este recurso (usando o browser para o efeito) percebemos que não existe nenhum tipo de proteção a quem o pode aceder, tratando-se de uma falha grave de segurança. Nas imagens abaixo mostramos o “`robots.txt`” e a vulnerabilidade que foi encontrada através de uma simples leitura deste.



```

HTTP/1.1 200 OK
Date: Sun, 29 May 2022 10:21:43 GMT
Server: Apache/2.4.52 (Debian)
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
Content-Type: text/plain; charset=utf-8
Content-Length: 28
ETag: W/"1c-8HgF6mNyhsSFK0pascC9uB0WjX0"
Vary: Accept-Encoding

User-agent: *
Disallow: /ftp
  
```

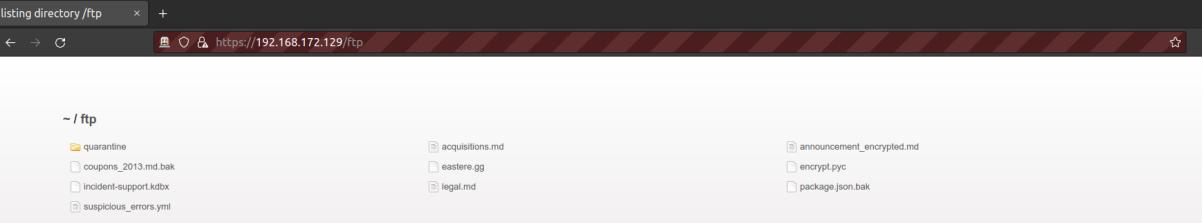


Figura 7 - Recolha de informação do ficheiro robots.txt (vulnerável) e exploração da falha - Acesso a uma área reservada (ficheiro “adquisitions.md” contém informação confidencial)

5.2.1.4 WSTG-INFO-04: Enumerate Applications on Webserver

Como referido anteriormente o “`Fingerprinting`” de um web server permite-nos obter informação bastante útil acerca do funcionamento/estado/versão deste, sendo

potencialmente relevante para um atacante que use esta informação com fins maliciosos. Além disso, não só este fingerprinting é importante mas também uma completa enumeração de todos os serviços que estão a correr no servidor, portos abertos, etc... pode contribuir para uma maior facilidade do atacante aceder a recursos protegidos (maior superfície de ataque).

No caso da “Juice Shop”, conseguimos através do ZAP e através de uma ferramenta de mapeamento `nmap` identificar as aplicações que estavam a correr no servidor e que estavam directamente relacionadas com esta. De facto conseguimos identificar aplicações e portos de outros serviços a correr na mesma máquina como as figuras abaixo podem demonstrar.

Technology	Version	Categories	Website	Implies	CPE
Apache	2.4.52	Web servers	http://apache.org		cpe:/a:apache:http_server
Cart Functionality		Ecommerce	https://www.wappalyzer.com/technologies/ecommerce/...		
Cloudflare		CDN	https://cfndns.com	Cloudflare	
Debian		CDN	http://www.cloudflare.com		
Google Font API		Operating systems	https://debian.org		cpe:/o:debian:debian_linux
Google Hosted Libraries		Font scripts	http://google.com/fonts		
jQuery	2.2.4 or 3.3.1	CDN	https://www.javascript.google.com/speed/libraries		
jQuery		Miscellaneous	http://jigrader.com		
Material Design Lite	1.3.0	JavaScript libraries	https://jquery.com		cpe:/a:jquery:jquery
Osano		UI frameworks	https://jgrmmlt.com		
SoundCloud		Cookie compliance	https://www.osano.com		
		Widgets	https://developers.soundcloud.com/docs/api/html5-widg...		

Figura 8 - Aplicações/Frameworks/Serviços associados à Juice Shop"

Figura 9 - Lista de open ports obtida pelo scan com o `nmap` (sudo nmap --send-eth -e eth0 -sV -f -n 192.168.172.129 -v -p 1-65535 -Pn --disable-arp-ping)

5.2.1.5 WSTG-INFO-05: Review Webpage Comments and Metadata for Information Leakage

A análise ao código fonte de uma aplicação web pode ser uma falha de segurança visto que expõe os seus potenciais pontos fracos à vista de atacantes. Desta forma, deve-se zelar por, no limite, não incluir no código fonte informações que caso sejam vistas por atacantes possam orientar de certa forma o seu ataque para zonas vulneráveis. Uma das falhas que pode ser cometida (partindo do pressuposto que temos acesso de alguma forma ao código fonte da aplicação) é a inclusão de

comentários explícitos que tenham observações sobre a forma de funcionamento, utilização ou até mesmo sobre o método de acesso a certos recursos privados (passwords de teste, usernames de teste, ...).

No decorrer da nossa análise da “Juice Shop”, recorrendo ao ZAP, verificamos que com as suas expressões regulares, este foi capaz de encontrar algumas situações onde existem comentários no código fonte que são suspeitos de conter informação que potenciem um ataque. Na imagem abaixo apresentamos as evidências encontradas pelo ZAP.

The screenshot shows the ZAP interface with a list of findings. One finding is highlighted, detailing a 'Suspicious Comments' issue. The evidence is a query with status 200, WASC ID 13, and source 'Passive (10027 - Information Disclosure - Suspicious Comments)'. The description states: 'The response appears to contain suspicious comments which may help an attacker. Note: Matches made within script blocks or files are against the entire content not only comments.' The 'Other Info' section shows a pattern used: `bQUERY\b` was detected in an element starting with `use strict";(self.webpackChunkfrontend=self.webpackChunkfrontend||[]).push([["179"],{}]);var j=d[2313],t=d[5e3],k=d[5]"; see evidence field for the suspicious comment/snippet.' The 'Solution' section advises removing all comments that may help an attacker and fixing any underlying problems they refer to. The 'Reference' section is empty. At the bottom, there are alerts and a primary proxy status.

Figura 10 - Evidências de comentários suspeitos encontrados pelo ZAP no código fonte da aplicação “Juice Shop”

5.2.1.6 WSTG-INFO-06: Identify Application Entry Points

A identificação de pontos de entrada na aplicação é uma componente importante da recolha de informação acerca da aplicação web, visto que, expõe pontos fracos que podem ser utilizados para aceder a áreas privilegiadas. Um exemplo comum de situações facilitadoras que auxiliam os atacantes a encontrar estes pontos são a passagem de informações como "session id" ou "session token", no header de um HTTP request, que facilmente pode ser capturado (sniffed).

No caso da “Juice Shop”, encontramos situações em que estes identificadores estavam a ser passados como nos headers HTTP. A situação abaixo evidencia um exemplo encontrado na análise feita com o ZAP em que num “POST” request foram passados vários elementos incluindo o id de sessão.

```
POST http://192.168.172.129/socket.io/?EIO=4&transport=polling&t=04Fk4S1&sid=PR0vx9HweaKJof5RAAn0 HTTP/1.1
Host: 192.168.172.129
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:100.0) Gecko/20100101 Firefox/100.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Content-type: text/plain;charset=UTF-8
Content-Length: 1
Origin: https://192.168.172.129
Connection: keep-alive
Referer: https://192.168.172.129/
Cookie: language=en; welcomebanner_status=dismiss
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
```

Figura 9 - Evidencia a passagem de um “Session ID” num header HTTP, o que pode comprometer a aplicação web (potencial entry point).

5.2.1.7 WSTG-INFO-07: Map Execution Paths Through Application

O completo mapeamento dos vários caminhos possíveis dentro de uma aplicação é importante pois permite-nos perceber que recursos existem e quais deles é que podem ser mais suscetíveis/vulneráveis a ataques. Desta forma é imperativo testar não só os vários trajetos possíveis dentro da aplicação mas também os fluxos de dados e eventuais race conditions que possam acontecer a fim de minimizar os riscos de ataque da aplicação.

No caso da “Juice Shop”, foi possível fazer este mapeamento (através do ZAP) que revelou algumas potenciais falhas de segurança e permitiu perceber os principais fluxos dentro da aplicação. Assim podemos considerar que existe uma falha de segurança nesta situação visto que do mapeamento efetuado o ZAP encontrou vulnerabilidades como mostra a imagem abaixo.

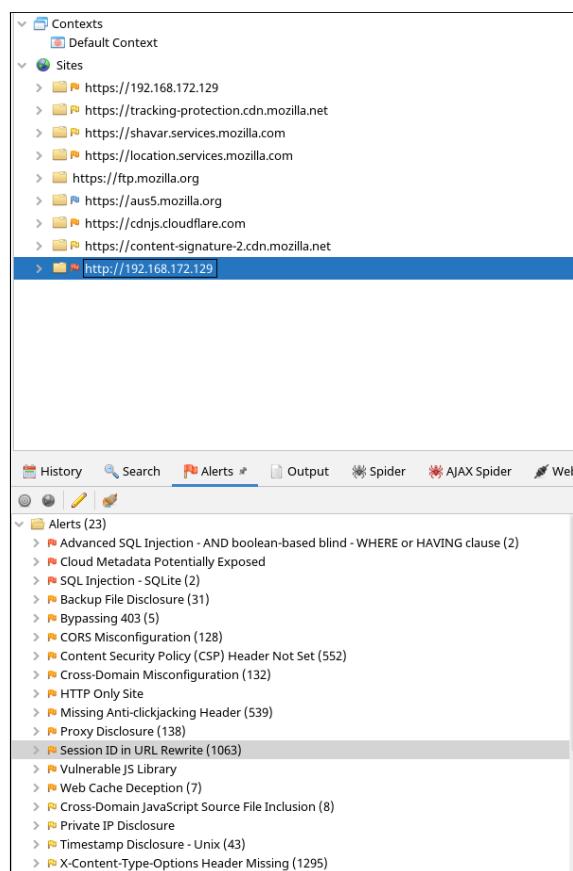


Figura 11 - Mapeamento da aplicação “Juice Shop” e vulnerabilidades encontradas.

5.2.1.8 WSTG-INFO-08: Fingerprint Web Application Framework

Este ponto mais uma vez faz referência ao “Fingerprinting” de frameworks utilizadas por uma web aplicação, a fim de encontrar vulnerabilidades nestas que comprometam a aplicação web no seu global.

No caso da “Juice Shop”, já encontramos várias evidências deste problema que apresentamos nos pontos acima, pelo que não vamos repetir esta análise. Note-se que a aplicação web está a correr através de um docker container e atrás de um proxy http pelo que se torna mais difícil obter informação acerca das componentes desta. As figuras 7 e 8, apresentam a informação que mesmo assim conseguimos obter através do ZAP e do `nmap`.

5.2.1.9 WSTG-INFO-09: Fingerprint Web Application

Este ponto mais uma vez faz referência ao “Fingerprinting” de aplicações web que sejam utilizadas como dependências. Estas aplicações como por exemplo: Wordpress, phpBB, Mediawiki, etc.. são amplamente utilizadas como base de outras aplicações web e são conhecidas várias vulnerabilidades em certas versões das mesmas.

No âmbito da análise da “Juice Shop”, que tenhamos conhecimento, nenhuma das aplicações anteriores é utilizada pelo que podemos considerar este tópico como não aplicável no contexto deste trabalho. Caso alguma destas aplicações tenha passado despercebida, consideramos a análise realizada nos pontos anteriores como suficientemente elucidativa da situação.

5.2.1.10 WSTG-INFO-10: Map Application Architecture

No caso deste ponto consideramos que todos os aspetos mais relevantes no contexto da aplicação que estamos a analisar “Juice Shop” já foram tocados. Desta forma não vamos analisar novamente este assunto, podendo o leitor analisar o ponto “WSTG-INFO-07” para contexto.

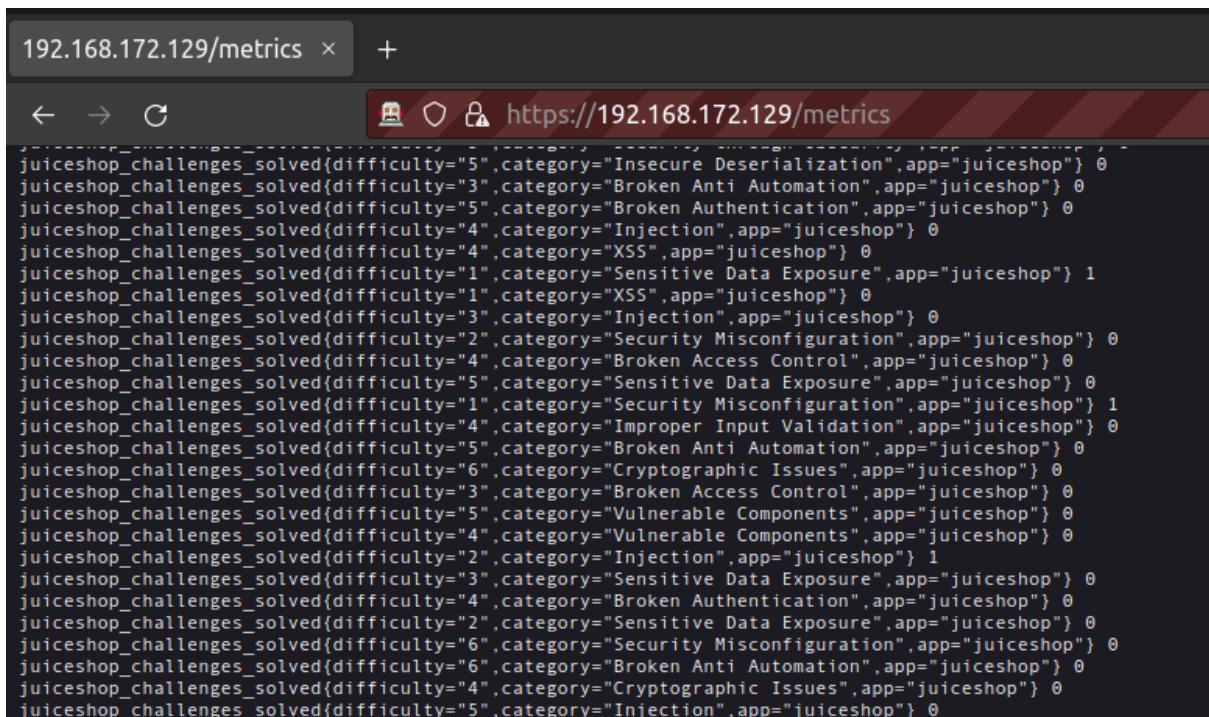
5.2.2. WSTG-CONF: Configuration and Deployment Management Testing

Esta secção toca em todos os aspetos relacionados com a configuração que é possível fazer às várias componentes de uma web application por forma a garantir a sua segurança e confidencialidade, especialmente quando deployed na internet. Em suma, pretende-se analisar como estas configurações têm impacto na segurança.

5.2.2.1 WSTG-CONF-01: Test Network Infrastructure Configuration

Este teste refere-se a qualquer falha em qualquer tipo de ficheiro de configuração da aplicação e provar que há uma falha que pode ser explorada. Como tal, é necessário analisar todos os pontos que possam provocar esse tipo de ataque, revendo as configurações todas por toda a rede e confirmar que não são vulneráveis. Além disso, é necessário validar que as frameworks e sistemas estão seguras e não estão vulneráveis por manutenção negligenciada do software ou uso de definições e credenciais existentes por defeito. Estes problemas põem-se especialmente quando o software está deployed em rede.

No âmbito da análise da “Juice Shop”, consideramos que este ponto não é fácil de analisar, visto que o deployment foi feito apenas localmente/isolado pelo que não deverá haver muitas falhas relacionadas com a configuração de rede (que é bastante minimalista). Porém, segundo o WSTG, um dos pontos tocados neste teste é usado exemplo prende-se com a configuração de rede permitir a um atacante remoto aceder e partilhar o código fonte com terceiros, o que aumenta a probabilidade de acontecerem ataques. Este aspeto foi realçado na secção anterior “WSTG-INFO” no entanto para reiterar apresentamos um exemplo onde é possível aceder a uma listagem no código que identifica as vulnerabilidades da aplicação e localizada no endpoint “/metrics” (descoberto por web scraping da aplicação). Embora seja uma feature deste tipo de aplicação, a existência desta lista, pretendemos realçar com isto a importância de não ser possível numa aplicação real poder obter estas informações através de scrapping, sendo esta uma configuração a ter em conta no processo de deployment na internet.



```

192.168.172.129/metrics × +
← → C https://192.168.172.129/metrics

juiceshop_challenges_solved{difficulty="5",category="Insecure Deserialization",app="juiceshop"} 0
juiceshop_challenges_solved{difficulty="3",category="Broken Anti Automation",app="juiceshop"} 0
juiceshop_challenges_solved{difficulty="5",category="Broken Authentication",app="juiceshop"} 0
juiceshop_challenges_solved{difficulty="4",category="Injection",app="juiceshop"} 0
juiceshop_challenges_solved{difficulty="4",category="XSS",app="juiceshop"} 0
juiceshop_challenges_solved{difficulty="1",category="Sensitive Data Exposure",app="juiceshop"} 1
juiceshop_challenges_solved{difficulty="1",category="XSS",app="juiceshop"} 0
juiceshop_challenges_solved{difficulty="3",category="Injection",app="juiceshop"} 0
juiceshop_challenges_solved{difficulty="2",category="Security Misconfiguration",app="juiceshop"} 0
juiceshop_challenges_solved{difficulty="4",category="Broken Access Control",app="juiceshop"} 0
juiceshop_challenges_solved{difficulty="5",category="Sensitive Data Exposure",app="juiceshop"} 0
juiceshop_challenges_solved{difficulty="1",category="Security Misconfiguration",app="juiceshop"} 1
juiceshop_challenges_solved{difficulty="4",category="Improper Input Validation",app="juiceshop"} 0
juiceshop_challenges_solved{difficulty="5",category="Broken Anti Automation",app="juiceshop"} 0
juiceshop_challenges_solved{difficulty="6",category="Cryptographic Issues",app="juiceshop"} 0
juiceshop_challenges_solved{difficulty="3",category="Broken Access Control",app="juiceshop"} 0
juiceshop_challenges_solved{difficulty="5",category="Vulnerable Components",app="juiceshop"} 0
juiceshop_challenges_solved{difficulty="4",category="Vulnerable Components",app="juiceshop"} 0
juiceshop_challenges_solved{difficulty="2",category="Injection",app="juiceshop"} 1
juiceshop_challenges_solved{difficulty="3",category="Sensitive Data Exposure",app="juiceshop"} 0
juiceshop_challenges_solved{difficulty="4",category="Broken Authentication",app="juiceshop"} 0
juiceshop_challenges_solved{difficulty="2",category="Sensitive Data Exposure",app="juiceshop"} 0
juiceshop_challenges_solved{difficulty="6",category="Security Misconfiguration",app="juiceshop"} 0
juiceshop_challenges_solved{difficulty="6",category="Broken Anti Automation",app="juiceshop"} 0
juiceshop_challenges_solved{difficulty="4",category="Cryptographic Issues",app="juiceshop"} 0
juiceshop_challenges_solved{difficulty="5",category="Injection",app="juiceshop"} 0

```

Figura 12 - Código Fonte de uma página da aplicação (descoberta por scrapping) que evidencia problemas que esta tem.

5.2.2.2 WSTG-CONF-02: Test Application Platform Configuration

Este ponto toca um aspeto importante que deve ter sido em conta no deployment de uma aplicação web. De facto, se a aplicação está pronta para consumo do utilizador, cuidados devem ser tomados para não incluir no payload software, código ou

qualquer outro tipo de recursos que não seja necessário para o seu funcionamento. É frequente em ambiente de desenvolvimento serem testadas muitas tecnologias e muitas features que acabam por não se refletir no produto final. As vezes coisas simples como “debug logs” são inseridos para facilitar este processo. Efetivamente, deve ser feita uma análise para garantir que estes componentes não ficam na versão final, por questões de performance e sobretudo por questões de segurança, visto que uma feature que está na versão de produção e que não é utilizada só aumenta a superfície de ataque (especialmente se estiver deprecated por alguma razão).

No caso da “Juice Shop”, identificamos uma situação onde uma API se encontra “deprecated” e que mesmo assim se encontra ativa na versão de produção. Tudo isto se passa na secção de “reclamações” onde existe um botão de upload que embora aparentemente só permita dar upload de ficheiros de extensão “.zip” e “.pdf”, inspecionando o código verificamos que é possível também dar upload de ficheiros “.xml”. Tentando um upload de um ficheiro deste tipo verificamos que a aplicação emite uma mensagem de erro nas “DevTools”, dizendo que esta feature se encontra “deprecated”. Nas imagens abaixo encontram-se evidências que recolhemos.

```

allowedMimeType: [
    'application/pdf',
    'application/xml',
    'text/xml',
    'application/zip',
    'application/x-zip-compressed',
    'multipart/x-zip'
]

```

Figura 12 - Uso de uma API deprecated em ambiente de produção

5.2.2.3 WSTG-CONF-03: Test File Extensions Handling for Sensitive Information

Em servidores web, é comum usar-se extensões de ficheiros para determinar que tecnologias, linguagens e plugins são requeridos. Normalmente verifica-se que tipo de ficheiros o utilizador quer dar upload e no caso de receber um tipo diferente, pode gerar resultados diversos por parte do servidor web. Usar extensões de ficheiro standard, apesar de estar de acordo com as normas de web, permite expor as tecnologias subjacentes usadas no dispositivo web e simplifica bastante determinar qual o cenário ideal de ataque específico a cada dispositivo. Uma má configuração do servidor web pode levar também a que informação confidencial e credenciais de acesso sejam expostas.

No caso da “Juice Shop”, tentando fazer um “POST” request HTTP, através da aplicação “postman”, verificamos que esta não consegue lidar bem com o tipo de ficheiro que está a ser submetido. Na evidência que incluímos abaixo, tentamos fazer um upload de um ficheiro no endpoint “/file-upload” que não está incluído nas extensões suportadas pela aplicação (.sh - shell script) e que mesmo assim é aceite, emitindo código “204 – No content”. Este comportamento não é o desejado, visto que deveria haver uma filtragem mais fina dos ficheiros.

The screenshot shows a Postman interface with the following details:

- URL: http://192.168.172.129/file-upload
- Method: POST
- Body tab selected
- Form-data section: file (Value: deploy.sh)
- Headers tab (8 items):
 - Content-Type: application/x-www-form-urlencoded
 - Content-Length: 278
 - Host: 192.168.172.129
 - Connection: keep-alive
 - Accept: */*
 - User-Agent: PostmanRuntime/7.29.0
 - Accept-Encoding: gzip, deflate
 - Cookie: session=...
- Test Results tab: Status: 204 No Content, Time: 30 ms, Size: 278 B, Save Response
- Message: You successfully solved a challenge: Upload Type (Upload a file that has no .pdf or .zip extension.)

Figura 13 - Upload de um ficheiro com extensão indesejada com sucesso!

5.2.2.4 WSTG-CONF-04: Review Old Backup and Unreferenced Files for Sensitive Information

Numa aplicação web, embora a maioria dos ficheiros presentes sejam de facto necessários para o seu correto funcionamento, é bastante comum encontrar ficheiros que servem como backup/versão temporária em várias aplicações web. Estes ficheiros não devem ser incluídos visto que podem fornecer a atacantes

informações acerca da infraestrutura da aplicação e as alterações que sofreu ao longo do tempo.

No caso da “Juice Shop”, através do ZAP, encontramos bastantes situações onde tais ficheiros de backup foram encontrados. Isto compromete severamente a segurança, pelo que estes ficheiros deveriam ter sido removidos. Na imagem abaixo apresentamos evidências destes ficheiros de backup.

The screenshot shows a ZAP interface with a list of network requests. The first request is highlighted in blue. The details pane on the right contains the following information:

- Evidence: A backup of [http://192.168.172.129/ftp/quarantine] is available at [http://192.168.172.129/ftp/quarantine%20-%20Copy%20(2)]
- CWE ID: 530
- WASC ID: 34
- Source: Active (10095 - Backup File Disclosure)
- Description: A backup of the file was disclosed by the web server
- Other Info:
http://192.168.172.129/ftp/quarantine
- Solution:
Do not edit files in-situ on the web server, and ensure that un-necessary files (including hidden files) are removed from the web server.
- Reference:
<https://cwe.mitre.org/data/definitions/530.html>
https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/02-Configuration_and_Deployment_Management_Testing/04-Review_Old_Backup_and_Unreferenced_Files_for_Sensitive_Information.html

Figura 14 - Várias evidências de ficheiros de backup presentes na aplicação e que são desnecessários na produção. (ZAP)

5.2.2.5 WSTG-CONF-05: Enumerate Infrastructure and Application Admin Interfaces

Este ponto toca nas interfaces de utilizador e de administrador que por vezes tem funcionalidades distintas, permitindo a um administrador ter mais controlo da aplicação do que um usuário padrão. A importância deste ponto passa por garantir que essas funcionalidades não são facilmente acedidas por um utilizador sem privilégios para tal.

No caso da “Juice Shop”, não visualizamos diferenças entre as interfaces de administrador e utilizador padrão pelo que consideramos este tópico como **não aplicável** no contexto do nosso trabalho. No entanto, devemos realçar que não existe nenhuma camada extra de segurança/página especial para autenticação de um administrador, o que na nossa perspetiva não é desejável.

5.2.2.6 WSTG-CONF-06: Test HTTP Methods

O protocolo HTTP permite que sejam usadas uma série de ações no servidor web (também denominadas métodos ou verbos). Os mais comuns são “GET” e “POST”, mas temos outros como “HEAD”, “PUT”, “DELETE”, “CONNECT”, “OPTIONS” e “TRACE”. Todos estes métodos servem para aceder a informação providenciada por um servidor web.

No caso da “Juice Shop”, em nenhum dos headers encontramos “Allow: GET, HEAD, POST, TRACE, OPTIONS”, sendo que por isso encontramos uma falha de segurança.

5.2.2.7 WSTG-CONF-07: Test HTTP Strict Transport Security

Existe um mecanismo denominado “HTTP Strict Transport Security” (HSTS) que permite a uma aplicação web informar o browser que nunca deve estabelecer uma ligação com domínios específicos num header especial de resposta.

No caso da “[Juice Shop](#)”, não encontramos nos headers que vimos nenhuma implementação HSTS através do header “[Strict-Transport-Security](#)”: portanto considera-se uma falha de segurança. No entanto, esta vulnerabilidade encontra-se fora do âmbito do projeto pelo que vamos considerar este tópico como **não aplicável**.

5.2.2.8 WSTG-CONF-08: Test RIA Cross Domain Policy

O “Rich Internet Applications” (RIA) permite troca de dados e serviços entre domínios de forma controlada, usando por exemplo Oracle Java, Silverlight e Adobe Flash. No entanto, por vezes os ficheiros de políticas estão mal configurados, o que permite ataques Cross-site Request Forgery e permite a terceiros aceder a conteúdo privado dos utilizadores.

Esta tecnologia não se encontra no programa do “[Juice Shop](#)”, portanto é um teste que não é possível de se fazer no âmbito do projeto (ou seja, **não aplicável**).

5.2.2.9 WSTG-CONF-09: Test File Permission

Este ponto foca o acesso a ficheiros da aplicação web cujas permissões são demasiado genéricas, comprometendo assim a confidencialidade dos mesmos. Não só se refere a permissões ao nível do “[filesystem](#)”, mas também se refere à necessidade de proteger o acesso a certos ficheiros com “[access tokens](#)” para que só os utilizadores que tiverem esse token poderem aceder ao recurso.

No âmbito de uma das alíneas anteriores da secção “WSTG-INFO”, já tinha sido referida a existência de uma secção no endpoint “[/ftp](#)” onde era possível aceder a ficheiros que podemos considerar privilegiados. Abaixo evidenciamos um ficheiro que encontramos nessa região e que não tendo proteções nem a nível do “[filesystem](#)” nem a nível de “token” contém informações confidenciais expostas a um atacante mais atento.

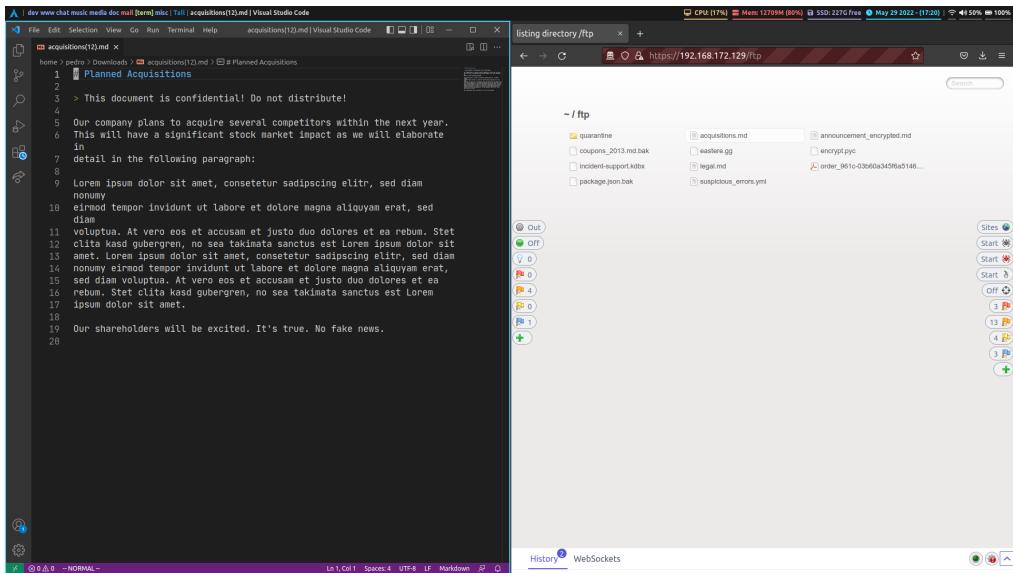


Figura 15 - Documento confidencial sem qualquer proteção de acesso.

5.2.2.10 WSTG-CONF-10: Test for Subdomain Takeover

Este teste é feito a uma vulnerabilidade do subdomínio externo do DNS, quando este está configurado para apontar para um serviço ou recurso externo não existente ou não ativo, ou quando a verificação do subdomínio não é apropriadamente efetuada.

Se caso algum dos pontos anteriores se verificar, o utilizador fica exposto a uma variedade de ataques possíveis, por exemplo, ser vítima de conteúdo malicioso, phising, roubo das credenciais de sessão do utilizador, entre outros.

No âmbito do nosso projeto, não estamos a utilizar DNS, visto estarmos a fazer todo o trabalho localmente. Logo este teste **não é aplicável**.

5.2.2.11 WSTG-CONF-11: Test Cloud Storage

A “Cloud Storage” refere-se a um serviço que permite às aplicações e serviços web armazenar e aceder a objetos. Não ter controle apropriado sobre os acessos a esse mesmo armazenamento pode levar à exposição e até mesmo alteração de conteúdo privado dos utilizadores.

Respetivamente ao nosso projeto, visto não estarmos a usar Cloud Storage na nossa aplicação, que se encontra está localmente, este teste **não é aplicável**.

5.2.3 WSTG-IDNT: Identity Management Testing

Esta secção toca em todos os aspectos relacionados com a verificação/atribuição de identidade de alguém que se regista/usa uma aplicação web. Também diz respeito à

atribuição de “roles” para melhor separação das permissões de acesso a certas regiões restritas.

5.2.3.1 WSTG-IDNT-01: Test Role Definitions

No contexto de uma aplicação web é bastante comum existirem diferentes tipos de “*roles*” de acordo com as posições que cada um dos utilizadores tem no contexto da aplicação. É geralmente expectável que existam pelo menos dois tipos de “*roles*”: Utilizador Padrão e Administrador. A existência de mais é aconselhável caso haja uma hierarquia mais estratificada, mas caso os “*roles*” básicos não existam podemos considerar que pode ocorrer falhas de segurança onde utilizadores não privilegiados executam funções que não deveriam conseguir executar.

No caso da “*Juice Shop*”, encontramos estes dois tipos de “*roles*” pelo que não temos nada de grave a apontar nesse sentido.

5.2.3.2 WSTG-IDNT-02: Test User Registration Process

Neste ponto pretende-se testar o processo de registo de um utilizador numa aplicação web. De facto, o processo do registo deve ser rigoroso por forma a permitir que as informações que são fornecidas quando o registo são suficientes para identificar a pessoa em questão mais tarde. O aumento da facilidade no registo de um utilizador pode ser visto como uma falha de segurança visto que se este não fornece informações suficientes, alguém facilmente pode adquirir um conjunto de informação básico e forjar a autenticação.

No caso da “*Juice Shop*”, consideramos que o processo de registo não pede informações suficientes ao utilizador para que se garanta que numa autenticação posterior se consiga verificar com a maior certeza possível a sua identidade. Na nossa perspetiva um email e uma pergunta de segurança, nos dias de hoje não são suficientes para garantir a maior proteção contra fraude possível.

The screenshot shows a 'User Registration' form. It includes fields for 'Email *', 'Password *' (with a note: 'Password must be 5-40 characters long.' and character count '0/20'), 'Repeat Password *' (with character count '0/40'), 'Security Question *' (with note: 'This cannot be changed later!'), and 'Answer *'. A 'Show password advice' toggle is present. The 'Register' button is at the bottom, along with a link 'Already a customer?'

Figura 16 - Processo de registo de um utilizador na aplicação.

5.2.3.3 WSTG-IDNT-03: Test Account Provisioning Process

Neste ponto pretende-se testar o processo de validação que ocorre durante um registo de um novo utilizador numa aplicação web. De facto, na medida do possível deve-se tentar que as informações inseridas pelos utilizadores sejam verdadeiras. Por exemplo, algo tão simples como verificar se o mail que o utilizador inseriu no registo é válido/existe pode contribuir para uma melhor segurança da sua conta e da aplicação no global.

No caso da “[Juice Shop](#)”, verificamos por exemplo com o campo de email no registo de um novo utilizador, que a aplicação faz uma verificação se o email é corretamente sinteticamente. No entanto, na nossa perspetiva deveria ser enviado um email após o registo a fim de ativar a conta pois o email mesmo estando correto sintaticamente pode nem sequer existir. Na imagem seguinte tentamos mostrar um exemplo dessa mesma situação.

The screenshot shows a 'User Registration' form with the following fields:

- Email ***: ola@stiefixe (highlighted in red)
- Password ***: (highlighted in red)
- Repeat Password ***: (highlighted in red)
- Show password advice**: A toggle switch is turned off.
- Security Question ***: Mother's maiden name? (highlighted in red)
- Answer ***: afafaf

Below the form is a blue button labeled '+ Register' and a link 'Already a customer?'.

Figura 17 - Processo de registo de um utilizador com um email cujo domínio não existe.

5.2.3.4 WSTG-IDNT-04: Testing Account Enumeration and Guessable User Account

Este teste serve para verificar por força bruta uma combinação de nome de utilizador e palavra-passe. Para isso, testa-se diferentes combinações de utilizador até que um username esteja bloqueado, e, neste momento, passa a tentar adivinhar a “password” ao qual está associado.

The screenshot shows a 'Login' form with the following fields:

- Email ***: adsfasdf (highlighted in red)
- Password ***: (highlighted in red)

Below the form is a blue button labeled ' Log in' and a checkbox 'Remember me'. At the bottom, there is a link 'Forgot your password?' and a link 'Not yet a customer?'.

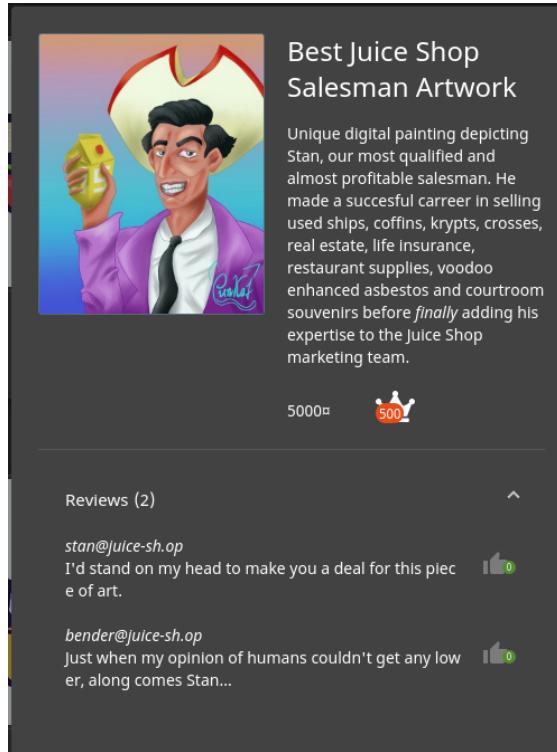


Figura 18 - Evidências da possibilidade de adivinhar contas de utilizadores da aplicação quer por teste de força bruta quer por observação de “reviews”

5.2.3.5 WSTG-IDNT-05: Testing for Weak or Unenforced Username Policy

A vulnerabilidade associada a este teste consiste em tentar enumerar usernames base que são facilmente adivinháveis, por exemplo, usando a primeira letra do primeiro nome de uma pessoa concatenada com o último nome desse mesmo utilizador.

Neste caso, como não existe nenhum login através de username o teste **não é aplicável**.

5.2.4 WSTG-ATHN: Authentication Testing

Esta secção toca em todos os aspectos relacionados com todo o tipo de autenticação que é possível realizar numa aplicação web. De uma forma geral aborda temas como a qualidade de passwords, o seu transporte, forma de como são guardadas e como são alteradas e utilizadas. Estes vários tópicos são extremamente relevantes pois este é um dos principais pontos de falha de segurança na maioria das aplicações hoje em dia.

5.2.4.1 WSTG-ATHN-01: Testing for Credentials Transported over an Encrypted Channel

Neste ponto pretende-se testar a forma de como as credenciais são transportadas nas comunicações efetuadas entre cliente e servidor, a fim de perceber se os mecanismos mais comuns de segurança estão a ser implementados para garantir que estas não são facilmente acessíveis a potenciais atacantes que estejam à escuta no canal de comunicação podendo apoderar-se destas (caso estejam em plain text). Principalmente, este teste visa averiguar a utilização do protocolo HTTPS nas comunicações.

No caso da “Juice Shop”, como esta aplicação foi concebida para trabalhar em cima do protocolo HTTP não tendo nenhum suporte para HTTPS, vamos considerar este tópico como **não aplicável** no contexto do nosso trabalho.

5.2.4.2 WSTG-ATHN-02: Testing for Default Credentials

Neste ponto pretende-se testar a robustez de uma aplicação no que toca à existência de contas com credenciais default que podem comprometer seriamente a segurança da aplicação, fornecendo a potenciais atacantes um ponto de entrada na mesma. De facto em muitas aplicações web existem contas que são criadas para fins de teste demonstração ou até mesmo é dada a opção aos utilizadores de criar uma conta e apenas mais tarde dar update às credenciais que foram atribuídas de forma default. Como referimos, é uma falha de segurança que pode ser explorada por atacantes através de um “fuzz attack” recorrendo a um dicionário com nomes contas mais comuns e suas palavras chave.

No caso da “Juice Shop”, efetuamos um “fuzz attack” extensivo testando várias combinações de emails e passwords. Os resultados são apresentados na imagem abaixo onde podemos ver que na duração do fuzz a única vulnerabilidade encontrada apenas ocorria quando não era inserido nenhum valor nos campos de autenticação.

Messages Sent: 144864	Errors: 0	Show Errors	Export						
Task ID	Message Type	Code	Reason	RTT	Size Resp. Header	Size Resp. Body	Highest Alert	State	Payloads
144848 Fuzzed		401	Unauthorized	126 ms	404 bytes	26 bytes			email@example.com
144849 Fuzzed		401	Unauthorized	122 ms	404 bytes	26 bytes			email@example.com
144850 Fuzzed		401	Unauthorized	109 ms	404 bytes	26 bytes			email@example.com
144851 Fuzzed		401	Unauthorized	114 ms	404 bytes	26 bytes			email@example.com
144852 Fuzzed		401	Unauthorized	111 ms	404 bytes	26 bytes			email@example.com
144853 Fuzzed		401	Unauthorized	123 ms	404 bytes	26 bytes			email@example.com
144854 Fuzzed		401	Unauthorized	119 ms	404 bytes	26 bytes			email@example.com
144855 Fuzzed		401	Unauthorized	113 ms	404 bytes	26 bytes			email@example.com
144856 Fuzzed		401	Unauthorized	120 ms	404 bytes	26 bytes			email@example.com
144857 Fuzzed		401	Unauthorized	114 ms	404 bytes	26 bytes			email@example.com
144858 Fuzzed		401	Unauthorized	109 ms	404 bytes	26 bytes			email@example.com
144859 Fuzzed		401	Unauthorized	105 ms	404 bytes	26 bytes			email@example.com
144860 Fuzzed		401	Unauthorized	105 ms	404 bytes	26 bytes			email@example.com
144861 Fuzzed		401	Unauthorized	102 ms	404 bytes	26 bytes			email@example.com
144862 Fuzzed		401	Unauthorized	104 ms	404 bytes	26 bytes			email@example.com
144863 Fuzzed		401	Unauthorized	97 ms	404 bytes	26 bytes			email@example.com

Figura 19 - Fuzz attack realizado na tentativa de encontrar contas default.

5.2.4.3 WSTG-ATHN-03: Testing for Weak Lock Out Mechanism

Neste ponto pretende-se testar se uma aplicação web apresenta um mecanismo de bloqueio ao fim de N tentativas de inserção de credenciais falhadas. Efetivamente, muitas das aplicações hoje em dia já apresentam bastantes mecanismos para

prevenir ataques de tentativa e erro “fuzzing” ou outros, que são efetuados por máquinas (robôs). Mecanismos como o “captcha” funcionam neste sentido, e evitando que um atacante possa utilizar a plataforma para fazer brute-force numa tentativa de encontrar senhas/contas de utilizadores.

No caso da “Juice Shop”, como podemos verificar na figura 19, obviamente reparamos que a aplicação não apresenta mecanismos para dar resposta a estes ataques, sendo isto uma vulnerabilidade bastante grave.

5.2.4.4 WSTG-ATHN-04: Testing for Bypassing Authentication Schema

Neste ponto pretende-se testar a capacidade de ultrapassar os mecanismos de segurança da aplicação e aceder a áreas que apenas através de um login poderiam ser acedidas.

No caso da “[Juice Shop](#)”, conseguimos de facto ultrapassar os mecanismos de login, explorando o facto de que a aplicação funciona em HTTP. Para o conseguir, começamos por efetuar um login com um utilizador criado para o efeito “[lolxd@gmail.com](#)” e visualizamos no ZAP o conteúdo de resposta do servidor onde em “[plaintext](#)” aparecia o token de sessão do mesmo. De seguida apenas nos limitamos a inserir esse token de sessão numa query efetuada, conseguindo de facto aceder a informações que só conseguimos visualizar caso existisse um login ativo deste utilizador.

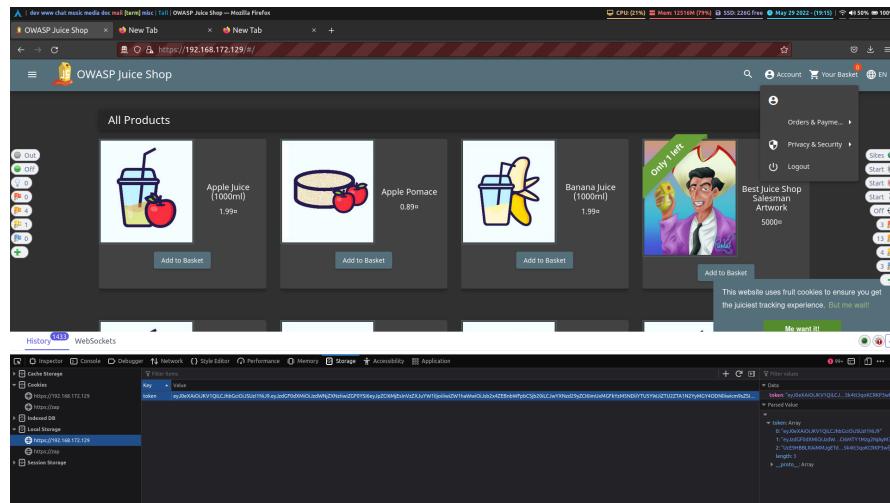


Figura 20 - Ultrapassagem do login, utilizando um token de sessão obtido por sniffing dos pacotes HTTP.

5.2.4.5 WSTG-ATHN-05: Testing for Vulnerable Remember Password

Neste ponto pretende-se testar a capacidade de aceder a passwords guardadas pelo utilizador localmente, sobretudo utilizando funcionalidades como “remember me”. Este tipo de funcionalidades pode comprometer a segurança visto que, caso haja algum acesso feito por um atacante à máquina onde o browser utilizado para aceder à aplicação está a correr é fácil aceder à “local storage” e ver este tipo de informações.

No caso da “Juice Shop”, verificamos que de facto aquando o clicar do botão remember me são guardadas informações na “local storage” que caso sejam acedidas podem comprometer a segurança da conta do utilizador. A imagem abaixo evidencia essa situação, onde o email do utilizador foi guardado aquando o clique no botão “remember me”, persistindo no browser entre as várias sessões e potenciando ataques de “fuzz” contra este utilizador.

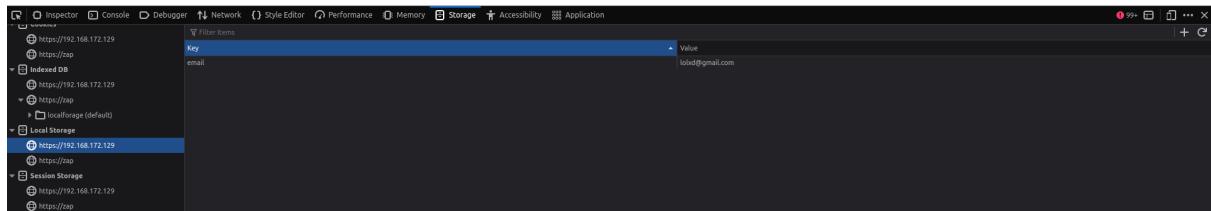


Figura 21 - Informações guardadas na localStorage - "remember me"

5.2.4.6 WSTG-ATHN-06: Testing for Browser Cache Weaknesses

Neste ponto pretende-se testar a segurança de uma aplicação web relativamente aos conteúdos que guarda em cache durante uma sessão. De facto é importante avaliar se durante a execução de uma aplicação esta vai guardando algum conteúdo sensível na cache do browser, que à semelhança da localStorage pode ser alvo de ataque e acesso por parte de hackers. É importante garantir que esta informação guardada em cache não persiste após o término da sessão de um dado utilizador.

No caso da “Juice Shop”, verificamos que de uma forma geral a cache não é utilizada para guardar informações sensíveis. Nos casos que esta cache é utilizada foi tomado o cuidado de incluir os headers HTTP necessários para garantir a não persistência dos conteúdos (como podemos observar na evidência abaixo).

```

HTTP/1.1 200 OK
Date: Sun, 29 May 2022 10:27:45 GMT
Server: Apache/2.4.52 (Debian)
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Sun, 22 May 2022 12:20:52 GMT
ETag: W/"805-180ebb690a0"
Content-Type: image/svg+xml
Content-Length: 2053
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive

```

Figura 22 - Uso de headers de cache control por parte da Juice Shop

5.2.4.7 WSTG-ATHN-07: Testing for Weak Password Policy

Neste ponto pretende-se avaliar a política de segurança que uma dada aplicação tem no registo de um dado utilizador, no que toca às restrições da password. Como já referido anteriormente, a qualidade da password é um dos principais pontos de falha que muitos atacantes exploram. Uma password com boa qualidade é imperativo para garantir a segurança do próprio e de todos os outros utilizadores. As aplicações devem zelar por garantir que as passwords que os vários utilizadores inserem cumprem com os requisitos mínimos que evitem sistemas estado da arte de quebra de passwords (john the ripper, etc...).

No caso da “Juice Shop”, como já foi referido, conseguimos através de um “fuzz attack” descobrir a password de um administrador.

Olhando para os requisitos de password impostos pela aplicação percebemos que estes são muito frágeis, obrigando apenas a ter uma password com um número de caracteres entre 5 e 40 , limitando-se a sugerir aos utilizadores boas práticas em vez de as impor (como podemos ver na imagem abaixo).

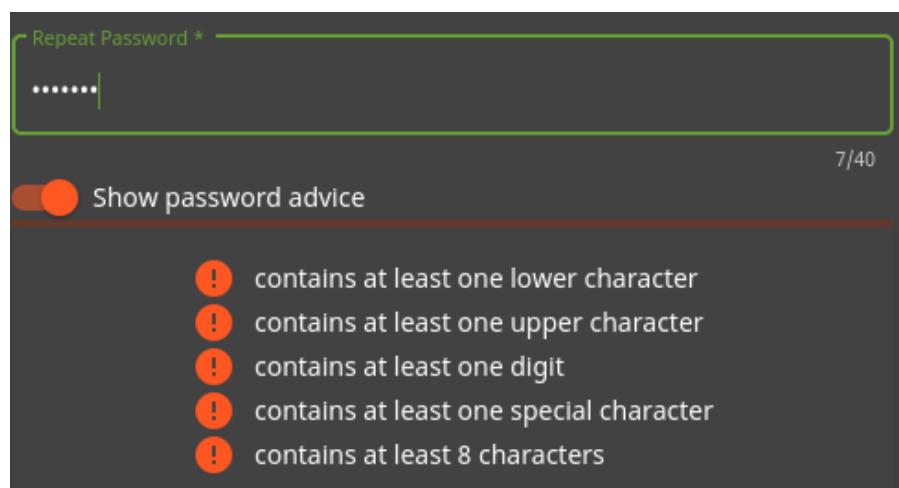


Figura 23 - Política utilizada na criação de passwords na Juice Shop

5.2.4.8 WSTG-ATHN-08: Testing for Weak Security Question Answer

Neste ponto, à semelhança de pontos anteriores, pretende-se testar a robustez das funcionalidades de “*security question*” que muitas aplicações têm. De facto, a existência de security questions é um método excelente para acrescentar uma camada extra de segurança. No entanto, caso as questões sejam demasiado standard (já fornecidas) corremos novamente o risco de sofrer ataques de “*fuzzing*”.

No caso da “*Juice Shop*”, é possível definir questões de segurança aquando a criação de conta, o que é uma vantagem. O problema está no facto destas questões de segurança serem bastante conhecidas e susceptíveis a ataques de fuzzing. Uma forma de colmatar isto seria isto seria dar ao utilizador a possibilidade de ele gerar as suas próprias perguntas e respostas.

The screenshot shows a registration form. In the 'Security Question *' field, the question 'Mother's maiden name?' is entered. A note below the field states 'This cannot be changed later!'. In the 'Answer *' field, there is a placeholder text 'Please provide an answer to your security question.' The entire form is set against a dark background.

Figura 24 - Secção de questões de segurança na página de registo

5.2.4.9 WSTG-ATHN-09: Testing for Weak Password Change or Reset Functionalities

Quando um utilizador pretende mudar ou restaurar a sua password, em vez de existir um administrador para executar esses pedidos, a aplicação tem um serviço que faz essa mudança automaticamente. Quando mudam, geralmente é dentro da aplicação. Quando são restauradas, ou são mostradas na aplicação ou são enviadas por email. Isto pode indicar que a password por estar armazenada sob formato texto ou descriptada. O teste consiste em determinar a resistência da aplicação à mudança ou restauro da aplicação.

No caso da “*Juice Shop*”, o restauro da password é feito apenas dentro da aplicação e não existe nenhuma confirmação com o utilizador. Este procedimento não é o mais seguro.

5.2.4.10 WSTG-ATHN-10: Testing for Weaker Authentication in Alternative Channel

Por vezes é possível que existam canais alternativos de autenticação de contas de utilizador. Esses canais alternativos podem comprometer a segurança da conta.

Este não é um teste à segurança destes canais alternativos, mas sim um registo de todas as possibilidades de vulnerabilidades que comprometem as contas dos utilizadores.

No caso da “Juice Shop”, no entanto, não é aplicável porque não há canal alternativo de autenticação.

5.2.5. WSTG-ATHZ: Authorization Testing

Esta secção toca em todos os aspetos relacionados com a autorização de acesso a recursos utilizados pela aplicação web, como por exemplo ficheiros. De uma forma especial são analisadas as situações onde é possível ultrapassar a segurança e aceder a recursos privados.

5.2.5.1 WSTG-ATHZ-01: Testing Directory Traversal File Include

Neste ponto pretende-se avaliar mais uma vez a robustez da aplicação no que toca a situações em que expõe aos utilizadores mecanismos que estes podem utilizar para navegar o “filesystem” do servidor onde está a correr, muitas vezes inadvertidamente. De facto existem casos onde um mau design dos endpoints pode levar a que os utilizadores a partir de uma simples query, por exemplo <http://example.com/getUserProfile.jsp?item=../../../../etc/passwd> consigam aceder a ficheiros como “/etc/passwd” onde são guardadas as hashes das passwords de todas as contas de utilizadores no sistema operativo do servidor. Este método de aceder à informação não é de todo desejável e deve ser bloqueado.

No caso da “Juice Shop”, o ZAP encontrou várias situações onde através de estratégias destas (também chamadas de dot-dot-slash attacks) conseguiu aceder a ficheiros guardados na máquina a dar host. A figura abaixo exemplifica a situação.

The screenshot shows a ZAP interface with a list of detected issues:

- Absence of Anti-CSRF Tokens (3)
- Backup File Disclosure (62)
- Bypassing 403 (10)
 - GET: http://192.168.172.129/%20/ftp/coupons_2013.md.bak%20/
 - GET: http://192.168.172.129/%20/ftp/eastere.gpg%20/
 - GET: http://192.168.172.129/%20/ftp/encrypt.pyc%20/
 - GET: http://192.168.172.129/%20/ftp/package.json.bak%20/
 - GET: http://192.168.172.129/%20/ftp/suspicious_errors.yml%20/
 - GET: http://192.168.172.129:3000/%2e/ftp/coupons_2013.md.bak
 - GET: http://192.168.172.129:3000/%2e/ftp/eastere.gg
 - GET: http://192.168.172.129:3000/%2e/ftp/encrypt.pyc
 - GET: http://192.168.172.129:3000/%2e/ftp/package.json.bak
 - GET: http://192.168.172.129:3000/%2e/ftp/suspicious_errors.yml
- CORS Misconfiguration (274)
- CSP: Wildcard Directive
- CSP: style-src unsafe-inline
- Content Security Policy (CSP) Header Not Set (3270)
- Cross-Domain Misconfiguration (485)
- HTTP Only Site (21)

Evidence:
CVE ID: 0
WASC ID: 0
Source: Active (40038 - Bypassing 403)
Description: Bypassing 403 endpoints may be possible; the scan rule sent a payload that caused the response to be accessible (status code 200).
Other Info:
<http://192.168.172.129:3000/ftp/eastere.gg>
Solution:
Reference:
<https://www.acunetix.com/blog/articles/a-fresh-look-on-reverse-proxy-related-attacks/>
<https://blackhat.com/us-18/Wed-August-8/us-18-Orange-Tai-Breaking-Parser-LogicTake-Your-Path-Normalization-Off-And-Pop-0days-Out-2.pdf>

Figura 25 - Situações detectadas pelo ZAP, onde ocorreu bypass e onde foi possível aceder a ficheiros no filesystem da máquina.

5.2.5.1 WSTG-ATHZ-02: Testing for Bypassing Authorization Schema

Neste ponto pretende-se avaliar a possibilidade de ultrapassar o esquema de autorização de acesso à informação que não pertence ao utilizador em questão.

Efetivamente, caso a aplicação web apresente alguma vulnerabilidade, ocorrem situações em que atacantes através de modificações de alguns ids conseguem aceder a contas de outros e visualizar suas informações. Claramente isto não é desejável pelo que deve ser protegido. Este tipo de ataques é conhecido por “horizontal escalation”

No caso da “Juice Shop”, identificamos pelo menos uma situação onde é possível ultrapassar o esquema de autenticação, permitindo a um utilizador ver o que está no cesto de compras de outro, necessitando apenas que modificar a variável “`bid`” na session storage do browser como mostrado na evidência abaixo.

The screenshot shows two parts of a web application interface. The top part is a dark-themed shopping basket page titled "Your Basket (olxd@gmail.com)". It lists four items: Banana Juice (1000ml) with 2 units at 1.99€, Eggfruit Juice (500ml) with 1 unit at 8.99€, Apple Pomace with 1 unit at 0.89€, and Green Smoothie with 1 unit at 1.99€. The bottom part is a developer tools window showing the "Storage" tab. It displays the session storage with the following items:

Key	Value
<code>bid</code>	6
<code>itemTotal</code>	15.850000000000001

The screenshot shows the same application interface after modifying the session storage. The basket now contains three items: Apple Juice (1000ml) with 2 units at 1.99€, Orange Juice (1000ml) with 3 units at 2.99€, and Eggfruit Juice (500ml) with 1 unit at 8.99€. The total price is now 21.94€. The developer tools storage tab shows the updated data:

Key	Value
<code>bid</code>	1
<code>itemTotal</code>	21.94

Figura 26 - Visualização do cesto de outro utilizador ao modificar a variável “`bid`” na “session storage” do browser.

5.2.5.1 WSTG-ATHZ-03: Testing for Privilege Escalation

O principal objetivo deste ponto é descrever se é possível para um utilizador escalar os seus privilégios para um nível mais alto. Também é verificado quando se consegue aceder a uma conta de administrador presente na aplicação web. Este tipo de ataques é conhecido como “vertical escalation” de privilégios

No caso da “Juice Shop”, este ponto já foi confirmado como vulnerabilidade, quando se testou se era possível fazer ataques do tipo "SQL Injection", na secção 5.1.3, ou no “fuzz” que efetuamos e que nos permitiu o acesso às credenciais do administrador “`admin@juice-sh.op`”.

5.2.5.1 WSTG-ATHZ-04: Testing for Insecure Direct Object References

A vulnerabilidade conhecida como Insecure Direct Object References ocorre quando é possível aceder a dados presentes na base de dados através de inputs dos utilizadores.

No caso da “Juice Shop”, já provamos que esta é uma vulnerabilidade através de diversos ataques de SQL Injection que nos permitiram interferir diretamente com os objetos da base dados “SQLite” que estão a correr no contexto da aplicação. Na imagem abaixo apresentamos uma evidência de um ataque deste tipo que nos possibilita efetuar modificações ao conteúdo da base de dados.

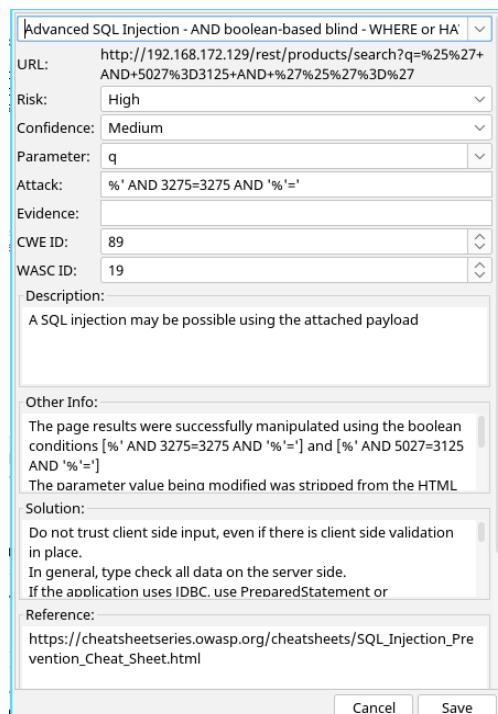


Figura 27 - SQL Injection em campos de inserção de input

5.2.6 WSTG-SESS: Session Management Testing

Esta secção toca em todos os aspectos relacionados com os mecanismos de manutenção de sessão que os browsers implementam através de cookies, local storage, cache, etc... De uma forma geral prende-se avaliar com os seguintes testes se algum destes mecanismos põe em causa a segurança da aplicação ou expõe informações privilegiadas.

5.2.6.1 WSTG-SESS-01: Testing for Session Management Schema

Neste ponto pretende-se avaliar se os mecanismos utilizados pela aplicação web, nomeadamente no que toca ao uso de “[cookies](#)”. De facto os cookies podem ser usados de diversas formas em ataques: para overflow, para manipulação, para recolha de informação acerca da aplicação etc... Assim é extremamente relevante não só utilizar os mecanismos de controlo através de headers HTTP e definir uma correta política de utilização dos mesmos, mas também evitar o uso destes desnecessariamente o que contribui para o ganho de informação que um atacante pode ter na sua análise.

No contexto da “[Juice Shop](#)”, identificamos diversas situações onde cookies estão a ser utilizados para controlo de informação de sessão (especialmente na zona do cesto de compras e definição da linguagem do site). No que toca à headers, da nossa análise não verificamos nenhuma irregularidade, no tempo de vida dos cookies e nos mecanismos de cache. No entanto devemos apontar que as operações sobre cookies no contexto desta aplicação estão a ocorrer sob HTTP (não havendo transporte encriptado) e o ZAP identificou várias situações onde existe um uso desnecessário de cookies que pode contribuir para brechas na segurança da aplicação. Para além disso, no processo de “[Set-Cookie](#)” reparamos que nem sempre estão a ser utilizados os melhores atributos, como vamos analisar no ponto seguinte.

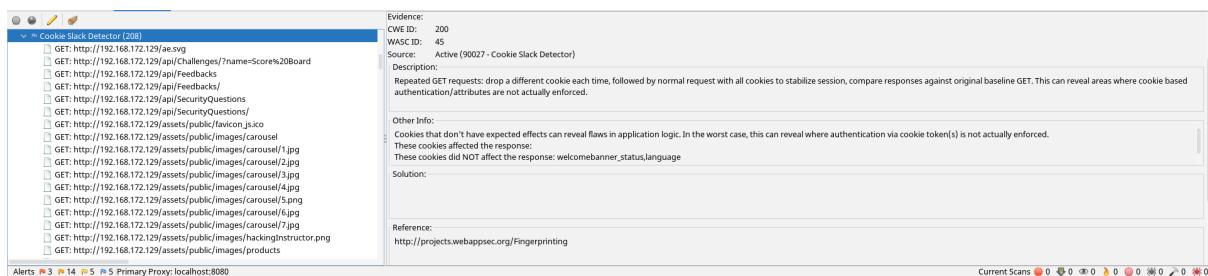


Figura 28- Potenciais vulnerabilidades no sistema de cookies encontradas pelo ZAP

5.2.6.2 WSTG-SESS-02: Testing for Cookies Attributes

Como já foi referido anteriormente o uso de “[cookies](#)” é relevante visto que o HTTP é um protocolo Stateless, sendo necessário mecanismos deste género para manter o estado. Neste ponto pretende-se testar se os cookies criados,

nomeadamente através da diretiva “`Set-Cookie`” apresentam também outros atributos para garantir uma maior segurança.

No contexto da “`Juice Shop`”, identificamos algumas situações onde não são usados (erroneamente utilizados) parâmetros de segurança como por exemplo o “`HttpOnly`” e “`Path`” que servem para ajudar a prevenir ataques de “`client-side-XSS`” e restringir a região para onde os cookies podem ser enviados, respetivamente. Podemos verificar na evidência abaixo retirada da “`Cookie Jar`” da aplicação, que o parâmetro “`HttpOnly`” encontra-se a “`false`” e o “`Path`” é demasiado genérico (“`/`” - root).

Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed	Data
192.168.172.129	/	Mon, 29 May 2022 ...	72	False	False	None	Sun, 29 May 2022 ...	language: "en"
192.168.172.129	/	Mon, 29 May 2022 ...	10	False	False	None	Sun, 29 May 2022 ...	Created: "Sun, 29 May 2022 10:47:18 GMT" Expires: "Sun, 29 May 2022 10:47:28 GMT" HttpOnly: true Path: "/" SameSite: "None" Size: 10
192.168.172.129	/	Mon, 30 May 2022 ...	769	False	False	None	Sun, 29 May 2022 ...	Expires / Max-Age: "Mon, 29 May 2023 10:47:18 GMT"
192.168.172.129	/	Mon, 29 May 2022 ...	27	False	False	None	Sun, 29 May 2022 ...	HostOnly: true HttpOnly: false Last Accessed: "Sun, 29 May 2022 21:47:09 GMT" Path: "/" SameSite: "None" Size: 10

Figura 29- Olhando para a “Cookie Jar” verificamos que alguns cookies não têm atributos bem definidos e que são relevantes.

5.2.6.3 WSTG-SESS-03: Testing for Session Fixation

Fixação da sessão ocorre quando os valores das cookies da sessão são mantidos antes e depois da autenticação. Normalmente ocorre quando se pretende guardar informação relativa, por exemplo, aos itens de um carrinho de compras online antes do utilizador entrar na sua conta.

A vulnerabilidade ocorre quando é permitido a uma pessoa externa forçar estas cookies de sessão no browser da vítima, fazendo-se passar por ela mesma.

No contexto da “`Juice Shop`”, não conseguimos visualizar vulnerabilidades deste género. No entanto, também é algo que sai do nosso objetivo de análise. Por outro lado, o que de facto identificámos é que os valores das cookies parecem atualizar-se antes e após a autenticação do utilizador na aplicação web.

5.2.6.4 WSTG-SESS-04: Testing for Exposed Session Variables

Este teste consiste em verificar se, como o nome indica, as variáveis de sessão de utilizador se encontram expostas, como, por exemplo, cookies, IDs de sessão, entre outros. Esta vulnerabilidade ocorre bastante quando estas variáveis estão a passar do browser do cliente para o servidor da aplicação.

No caso de uma pessoa externa ter acesso a estas variáveis, a vítima fica exposta a um roubo de identidade, isto é, o atacante pode agir como se fosse a vítima.

No contexto da “[Juice Shop](#)”, temos evidências de que os pedidos de pacotes de HTTP estão a ser interceptados por uma fonte externa, onde o ID de sessão é visível, sendo que por isso é uma vulnerabilidade.

The screenshot shows the OWASP ZAP interface. The left pane displays a tree of requests, including various GET and POST methods for user authentication and session management. The right pane shows the detailed response for one of these requests. In the 'Body.Text' tab, the session ID '40{"sid": "Bpqlo85fh8k2tHzIAAuZ"}' is clearly visible in the response content. Below the main interface, there is a status bar with CPU: 8%, Mem: 12963M (81%), SSD: 225G free, Date: May 29 2022 - (22:30), and battery level at 100%.

Figura 30 - Evidência de que é possível visualizar o ID da sessão no pedido de transferência de pacotes HTTP

5.2.6.5 WSTG-SESS-05: Testing for Cross Site Request Forgery

Este tipo de ataques (Cross Site Request Forgery - CSRF) força um utilizador a enviar pedidos HTTP para um destino sem ser a sua intenção, conhecimento ou consentimento a fim de fazer certas ações como sendo a vítima. Estes ocorrem quando são usados URLs e formulários de ação previsíveis e repetitivos. Este ataque permite ao atacante ter conhecimento de todos os dados de utilizador. Se a vítima for um administrador, poderá comprometer a aplicação web por inteiro. A forma de evitar este tipo de ataques é ter tokens que previnam estas vulnerabilidades.

No contexto da “[Juice Shop](#)”, encontrou-se evidências de que alguns desses tokens que deveriam prevenir estes ataques não estão a ser usados, sendo que por isso, podem comprometer o sistema inteiro.

A mensagem que foi originada neste programa (“No Anti-CSRF tokens were found in a HTML submission form.”), evidenciada na imagem seguinte, comprova que o sistema está de facto vulnerável a este tipo de ataques.

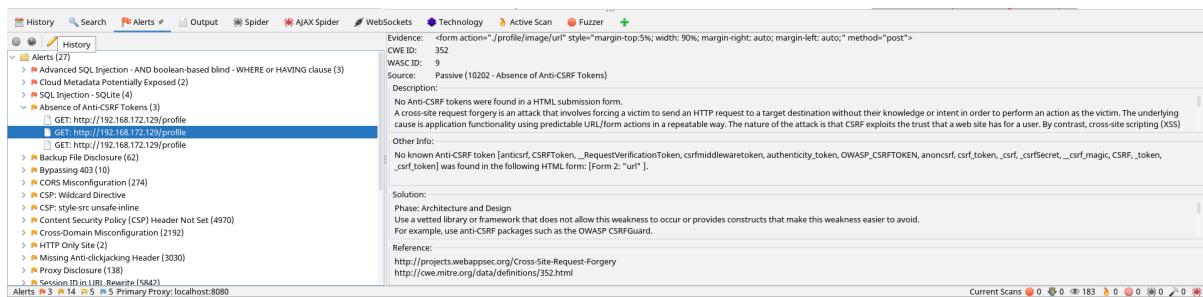


Figura 31 - Mensagem originada no programa “ZAP”, que mostra que não há forma de prevenção de ataques do tipo CSRF.

5.2.6.6 WSTG-SESS-06: Testing for Logout Functionality

A funcionalidade de logout é por si auto-explicativa. O utilizador tem a opção de sair da sua conta a qualquer momento e o botão de “Logout” tem de estar presente em todas as páginas ao qual o utilizador acede.

O teste pretende provar se o UI de logout de facto funciona e se o cache do site não guarda o site no estado em que o utilizador tinha a sessão ativa, isto é, se o utilizador ao sair da sua conta e voltar ao estado anterior da página, a nova página não se encontra num estado em que o utilizador tinha a sessão ativa.

No contexto da “Juice Shop”, o botão “Logout” parece funcionar corretamente, apagando informações confidenciais da sessão do utilizador. No entanto, um possível aspeto que pode ser considerado como feature ou possível vulnerabilidade é a opção de “Remember me” do servidor web, em que é guardado localmente o email do utilizador. Este aspeto foi abordado previamente.

5.2.6.7 WSTG-SESS-07: Testing Session Timeout

A funcionalidade conhecida como “Session Timeout” consiste, como o próprio nome indica, na desconexão do utilizador à aplicação web se este se encontra inativo por algum tempo.

Este mecanismo providencia aos atacantes menos tempo para tentar adivinhar uma sessão de ID de um outro utilizador válida. No entanto, se este consegue encontrar um ID de sessão válida, o atacante pode simular atividade por parte da vítima, o que invalida a funcionalidade de “Session Timeout”.

No contexto da “Juice Shop”, não detetámos nenhuma evidência de que esta funcionalidade está ativa ou implementada. Assim podemos assumir que há uma falha de segurança, já que hoje em dia os ataques informáticos são mais recorrentes, o que torna a possibilidade de ataques por haver esta vulnerabilidade mais elevada.

5.2.6.8 WSTG-SESS-08: Testing for Session Puzzling

A vulnerabilidade conhecida como “[Sessions Puzzling](#)” consiste no uso das mesmas variáveis de sessão para mais do que um destino. Um atacante pode aceder às páginas web de uma forma não antecipada pelos developers, com o propósito de variáveis de sessão usadas num contexto serem aplicadas noutra.

Se esta falha se verificar, o atacante pode tomar a identidade da vítima, elevar os privilégios de outros utilizadores maliciosos, manipular valores do lado do site web de forma imprevisível ou indetectável, entre outros.

Uma potencial forma de encontrar estas falhas seria analisar todo o código source da aplicação e verificar pontos sensíveis.

No contexto da “[Juice Shop](#)”, não conseguimos identificar situações destas nos testes que fizemos. No entanto, isso está fora do âmbito da nossa análise, pelo que consideramos que não é aplicável no nosso caso.

5.2.7 WSTG-INPV: Input Validation Testing

Esta secção é dedicada a todo o tipo de testes relacionados com validação de input por parte da aplicação web. De facto, tudo o que está relacionado com input vindo do utilizador pode ser considerado um ponto mais frágil da aplicação web, visto que o carácter estocástico do que pode ser inserido em campos deste tipo é imenso, sendo difícil, e diríamos nós impossível prever todas as formas possíveis de ataques e proteger, adequadamente, estes campos. Os ataques mais comuns nesta área passam por inserção de certas strings ou código que são interpretados pela aplicação de uma forma inesperada e que permitem ao atacante modificar ou aceder a informação privilegiada, ultrapassar os mecanismos de autenticação do servidor ou até mesmo controlar a máquina na qual o servidor está a operar (Injection). Desta forma torna-se imperativo garantir que é feita uma testagem extensiva por forma a minimizar as falhas que possam ocorrer.

No contexto do nosso trabalho verificamos que existem bastantes pontos nas normas do WSTG que não se aplicam à nossa aplicação “[Juice Shop](#)”, por essa tecnologia não ser utilizada (como é o caso do LDAP, NoSQL, MS Access, etc..), ou por acharmos que saia um pouco do âmbito do nosso projeto e da nossa análise (Remote File Inclusion, Server Side Template Injection). Assim resolvemos apresentar apenas algumas evidências de falhas de segurança na aplicação a fim de demonstrar que a nossa “[Juice Shop](#)”, apresenta diversas falhas nesta vertente. Note-se que muitas destas falhas já foram referidas anteriormente (descobertas pelo ZAP inclusivé) neste documento pelo que vamos optar por fazer uma análise breve.

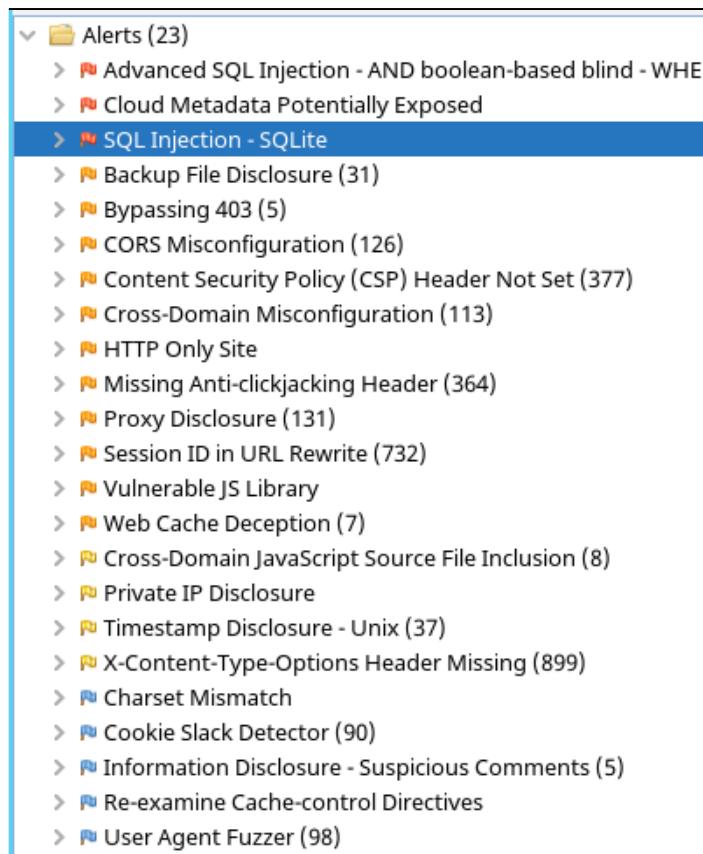


Figura 32 - Panorama geral - Falhas encontradas pelo ZAP

5.2.7.1 WSTG-INPV-01: Testing for Reflected Cross Site Scripting

Neste ponto pretende-se testar se a aplicação web é resiliente contra ataques do tipo “[Reflected XSS](#)”. Neste tipo de ataques o que ocorre é uma injeção de um payload com código executável pelo browser numa resposta HTTP. Note-se que este ataque não é guardado do lado da aplicação (não é persistente) no entanto é suficiente para impactar utilizadores que tenham aberto um link malicioso ou forjado para a aplicação.

No contexto da “[Juice Shop](#)”, conseguimos identificar uma situação onde esta vulnerabilidade se encontra. Efetuando compras com um dado utilizador e acedendo ao separador onde são listadas as compras que este já fez, conseguimos perceber que no URL é exposto um id como parâmetro, no momento em que este decide clicar no icon do camião a fim de fazer tracking da encomenda (“<https://192.168.172.129/#/track-result?id=961c-15a805c2b1c86f40>”). Ora isto é suscetível a ataques de XSS, bastando simplesmente alterar o id para uma string com código javascript como por exemplo “[`<iframe src="javascript:alert\(`xss`\)">`](#)”. Na imagem abaixo é apresentada a evidência deste ataque.

Figura 33 - Ataque de Reflected XSS.

5.2.7.2 WSTG-INPV-05: Testing for SQL Injection

Neste ponto pretende-se testar a resiliência do site a ataques do tipo SQL Injection. Em secções anteriores já explicamos no que consiste este ataque pelo que apenas nos vamos limitar a apresentar mais uma evidência de uma vulnerabilidade da nossa “Juice Shop” no que toca a este tipo de situações. Na imagem abaixo podemos verificar que conseguimos ultrapassar (através de fuzzing) a autenticação recorrendo a SQL Injection no campo email (na página login).

Task ID	Message Type	Code	Reason	RTT	Size Resp. Header	Size Resp. Body	Highest Alert	State	Payloads
25 Fuzzed		200 OK		117 ms	404 bytes	831 bytes			' or 1=1 --
55 Fuzzed		200 OK		112 ms	403 bytes	831 bytes			' or username is NULL or username=
117 Fuzzed		200 OK		107 ms	404 bytes	831 bytes			' or '
125 Fuzzed		200 OK		109 ms	402 bytes	831 bytes			' or 1=1--
127 Fuzzed		200 OK		103 ms	404 bytes	831 bytes			' or 1=1/*
149 Fuzzed		200 OK		114 ms	404 bytes	831 bytes			' or username like char(37);
157 Fuzzed		200 OK		116 ms	402 bytes	831 bytes			' or 1/*
171 Fuzzed		200 OK		122 ms	403 bytes	831 bytes			' or 1=1 --
186 Fuzzed		200 OK		109 ms	404 bytes	831 bytes			' or 1=1 or 'x='y

Figura 34 - Ataque de SQL Injection (fuzzing login).

5.2.8 WSTG-ERRH: Testing for Error Handling

Esta secção toca em como os erros podem ser um ponto fraco de uma aplicação se não tratados com cuidado, pela possibilidade de expor informação relevante que, em conjunto com outro tipo de ataques, pode comprometer a aplicação web.

5.2.8.1 WSTG-ERRH-01: Testing for Error Code

Neste ponto pretende-se testar a forma de como as mensagens de erro emitidas por uma aplicação web podem dar informações aos atacantes acerca da infraestrutura de suporte desta como por exemplo bases de dados. Efetivamente, através de uma simples interação com aplicação, e no evento de ocorrer um erro deve-se zelar para que as mensagens de erro sejam o menos possível sugestivas da forma como o sistema por detrás funciona e suas propriedades (no caso de uma base de dados, por exemplo acesso ao schema, etc..) e devem ser ajustadas para que o utilizador da aplicação perceba o erro que está a cometer sem ganhar demasiada informação acerca de como esse erro “foi gerado”.

No contexto da “Juice Shop”, já demonstramos várias evidências onde o erro devolvido é sugestivo da infraestrutura a correr por detrás. Um exemplo claro disso é nos ataques de SQL injection onde a mensagem de erro adiciona no payload informação acerca da query que falhou, de onde pode ser extraída informação acerca das tabelas existentes na base de dados.

```
```json
{
 "error": {
 "message": "SQLITE_ERROR: near \"\"; syntax error",
 "stack": "Error: SQLITE_ERROR: near \"\"; syntax error",
 "errno": 1,
 "code": "SQLITE_ERROR",
 "sql": "SELECT * FROM Products WHERE ((name LIKE '%'('%' OR description LIKE '%'('%') AND deletedAt IS NULL) ORDER BY name"
 }
}
```

```

Figura 35 - Mensagem de erro demasiado informativa - SQL Injection.

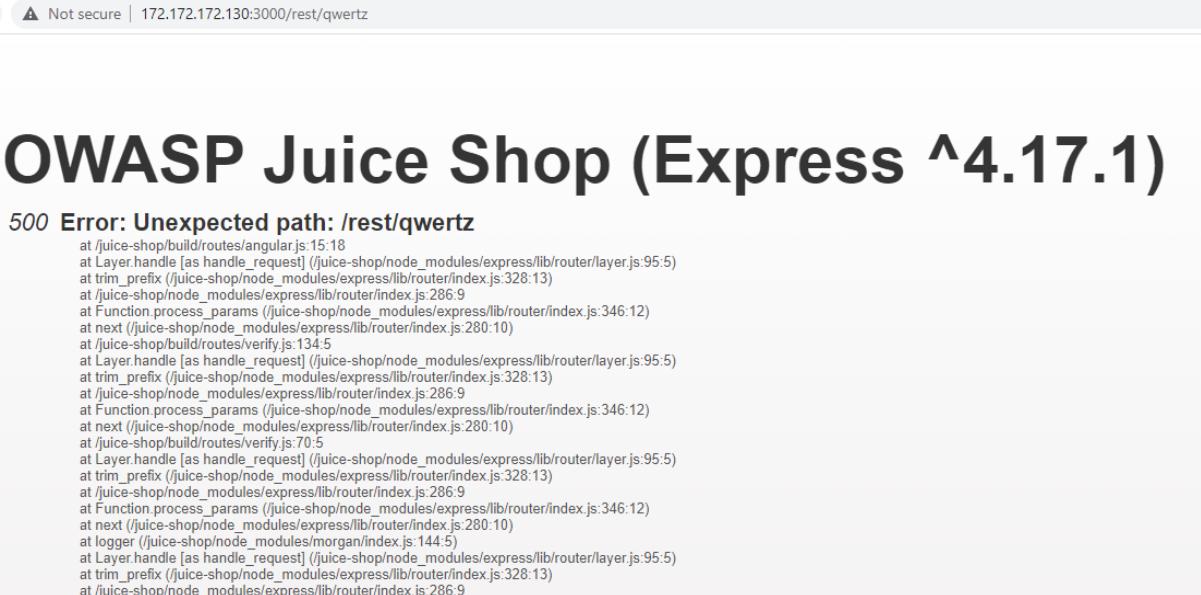
5.2.8.1 WSTG-ERRH-02: Testing for Stack Traces

Todos os sites geram, de uma forma ou outra, uma mensagem de erro. É comum esperar que os utilizadores de uma aplicação web escrevam o que é expectável deles num certo campo (por exemplo escrever um número inteiro acima de zero num campo onde é requerida a idade).

O erro originado pela consideração apenas do “caminho feliz” (onde o utilizador introduz apenas o que é expectável) pode denunciar vários aspectos que um atacante usaria para criar uma cadeia de eventos que comprometem a segurança da aplicação, por exemplo, como a API é processada internamente (isto é, como os objetos são referenciados internamente - chamado “stack traces”), verificar versões e tipos de aplicações e serviços usados no servidor web, controlar a reação do sistema a certas exceções usando a lógica estabelecida à volta do “caminho feliz”, entre outros.

A mitigação desta possível falha de segurança revolve à volta de considerar outros caminhos que não apenas aquele em que o utilizador não tenta provocar um erro propositadamente.

No contexto da “Juice Shop”, foi forçado um erro, ao adicionar a extensão “/rest/qwertz” ao URL da aplicação web, da qual, se feito o login como administrador (referenciado anteriormente), origina o erro seguinte, onde mostra claramente um “stack trace”. Assim, através deste erro, um atacante pode ter informação crucial para aceder à aplicação web, tornando-se uma vulnerabilidade indireta (já que por si só, esta falha de segurança não compromete o sistema).



The screenshot shows a browser window with the address bar containing "Not secure | 172.172.172.130:3000/rest/qwertz". The main content area displays the following:

OWASP Juice Shop (Express ^4.17.1)

500 Error: Unexpected path: /rest/qwertz

```
at /juice-shop/build/routes/angular.js:15:18
at Layer.handle [as handle_request] (/juice-shop/node_modules/express/lib/router/layer.js:95:5)
at trim_prefix (/juice-shop/node_modules/express/lib/router/index.js:328:13)
at /juice-shop/node_modules/express/lib/router/index.js:286:9
at Function.process_params (/juice-shop/node_modules/express/lib/router/index.js:346:12)
at next (/juice-shop/node_modules/express/lib/router/index.js:280:10)
at /juice-shop/build/routes/verify.js:134:5
at Layer.handle [as handle_request] (/juice-shop/node_modules/express/lib/router/layer.js:95:5)
at trim_prefix (/juice-shop/node_modules/express/lib/router/index.js:328:13)
at /juice-shop/node_modules/express/lib/router/index.js:286:9
at Function.process_params (/juice-shop/node_modules/express/lib/router/index.js:346:12)
at next (/juice-shop/node_modules/express/lib/router/index.js:280:10)
at /juice-shop/build/routes/verify.js:70:5
at Layer.handle [as handle_request] (/juice-shop/node_modules/express/lib/router/layer.js:95:5)
at trim_prefix (/juice-shop/node_modules/express/lib/router/index.js:328:13)
at /juice-shop/node_modules/express/lib/router/index.js:286:9
at Function.process_params (/juice-shop/node_modules/express/lib/router/index.js:346:12)
at next (/juice-shop/node_modules/express/lib/router/index.js:280:10)
at logger (/juice-shop/node_modules/morgan/index.js:144:5)
at Layer.handle [as handle_request] (/juice-shop/node_modules/express/lib/router/layer.js:95:5)
at trim_prefix (/juice-shop/node_modules/express/lib/router/index.js:328:13)
at /juice-shop/node_modules/express/lib/router/index.js:286:9
```

Figura 36 - Stack trace originado pela adição da extensão “/rest/qwertz” ao URL da aplicação web “Juice Shop”.

5.2.9 WSTG-CRYP: Testing for Weak Cryptography

As vulnerabilidades associadas a este tipo de testes referem-se ao protocolo HTTPS.

No contexto da “Juice Shop”, este protocolo não se encontra implementado, pelo que é impossível efetuar este teste.

5.2.10 WSTG-BUSL: Business Logic Testing

Esta secção pretende incentivar as pessoas que fazem testes de vulnerabilidades a sistemas a pensar mais além do pensamento convencional e de métodos tradicionais de ataques. Não há um teste específico que se possa fazer e por estarmos num primeiro contacto com esta lista de possíveis vulnerabilidade é-nos complicado perceber possíveis procedimentos fora do normal. Como tal, consideramos esta secção fora do contexto do projeto.

5.2.11 WSTG-CLNT: Client-Side Testing

Esta secção é dedicada a todo o tipo de testes relacionados com teste de funcionalidades que ocorrem do lado do cliente, ou seja, execuções de código no

browser do cliente, websockets, CSS Injection etc... Nesta secção optamos por fazer uma análise breve visto que muitos dos problemas, mais uma vez já foram referidos, ou capturados pelo ZAP. Assim decidimos apenas mostrar um exemplo de uma situação onde existem vulnerabilidades deste tipo (que nos pareceu mais relevante no contexto da cadeira), como poderemos ver na próxima secção.

5.2.7.1 WSTG-CLNT-01: Testing for DOM-Based Cross Site Scripting

O DOM-Based Cross Site Scripting consiste na injeção de código por parte do utilizador no servidor web que é executado do lado do browser em locais que, por exemplo, JavaScript é implementado de forma insuficiente ou incompleta. O DOM (Document Object Model) é representação estrutural de documentos num browser. Um ataque DOM-Based XSS ocorre quando, por exemplo, uma função de JavaScript é especialmente desenhada para que os elementos presentes no DOM sejam manipulados conforme o atacante pretenda.

No contexto da “Juice Shop”, foi injetada a string “`<iframe src="javascript:alert(`hello There`)">`” no campo de pesquisa da aplicação web, a qual originou um alerta, sob a forma de um pop-up do lado do cliente com a frase “hello There”, o que indica que de facto é possível efetuar ataques DOM-Based Cross Site Scripting (XSS).

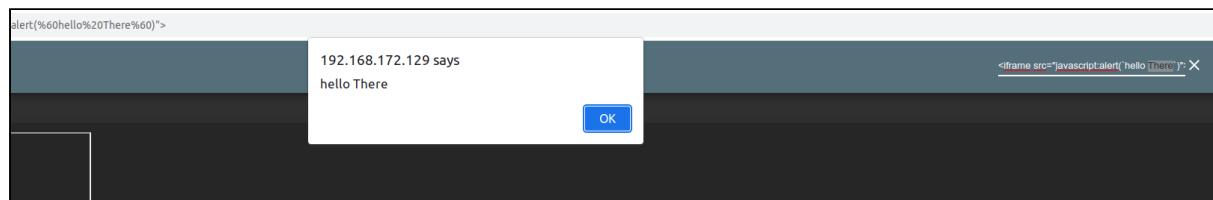


Figura 37 - Exemplo de um ataque DOM-Based XSS no campo de search

6. Web Application Firewall (WAF)

Após a realização dos de todos os testes propostos pelas guidelines do WSTG, chegamos à fase de ativação da WAF já configurada, a fim de testar a sua eficácia no bloqueio dos ataques correspondentes às vulnerabilidades que fomos encontrando ao longo do trabalho. Algumas das vulnerabilidades encontradas, como por exemplo as relacionadas com “information leaking”, “configuration”, etc., não são alvo de análise pela firewall visto que não são situações contra as quais esta possa proteger. Desta forma, as situações que testamos bastante prendem-se essencialmente com os ataques de Injection, XSS, entre outros (que o zap foi encontrando), ou seja ataques em que é possível construir expressões regulares para detectar certos padrões no payload de um pedido que possam ser sugestivas de um ataque. Como já referimos antes utilizamos o “core rule set” do OWASP a fim de ter uma maior quantidade de regras para proteção.

A título de exemplo, apresentamos uma das evidências que encontramos de como a WAF bloqueou um ataque de SQL Injection. Obviamente foi possível arranjar mais nomeadamente aquando o fuzzing de login forms, DOM-XSS, entre outros, mas achamos que este era um exemplo interessante.

É possível observar que antes da ativação da firewall existiam várias strings que quando inseridas no form de login permitiam (através de SQL Injection) ultrapassar a autenticação e entrar na conta do administrador.

| Task ID | Message Type | Code | Reason | RTT | Size Resp. Header | Size Resp. Body | Highest Alert | State | Payloads |
|------------|--------------|--------|--------|--------|-------------------|-----------------|----------------------|-------|----------|
| 25 Fuzzed | | 200 OK | | 117 ms | 404 bytes | 831 bytes | ' or 1=1 -- | | |
| 55 Fuzzed | | 200 OK | | 112 ms | 403 bytes | 831 bytes | ' or username is ... | | |
| 117 Fuzzed | | 200 OK | | 101 ms | 404 bytes | 831 bytes | ' or 1=1- | | |
| 125 Fuzzed | | 200 OK | | 105 ms | 403 bytes | 831 bytes | ' or 1=1- | | |
| 127 Fuzzed | | 200 OK | | 103 ms | 404 bytes | 831 bytes | ' or 1=1 /* | | |
| 149 Fuzzed | | 200 OK | | 114 ms | 404 bytes | 831 bytes | ' or username li... | | |
| 157 Fuzzed | | 200 OK | | 116 ms | 403 bytes | 831 bytes | ' or 1/* | | |
| 171 Fuzzed | | 200 OK | | 122 ms | 403 bytes | 831 bytes | a' or 1=1; -- | | |
| 186 Fuzzed | | 200 OK | | 105 ms | 404 bytes | 831 bytes | x' or 1=1 or 'x'=y | | |

Figura 38 - Exemplo de fuzzing que encontrou vulnerabilidades de SQL Injection no login, permitindo acesso à conta de administrador.

Após a implementação da WAF e experimentando com uma das strings utilizadas para login anteriormente e que correspondia a uma SQL Injection verificamos que o servidor apache a funcionar como reverse-proxy (WAF) intercepta e rejeita o pedido protegendo assim a “Juice Shop” contra esta vulnerabilidade. Note-se que o texto a vermelho apresenta a informação devolvida pelo apache ao rejeitar o pedido, com o código “403 Forbidden”.

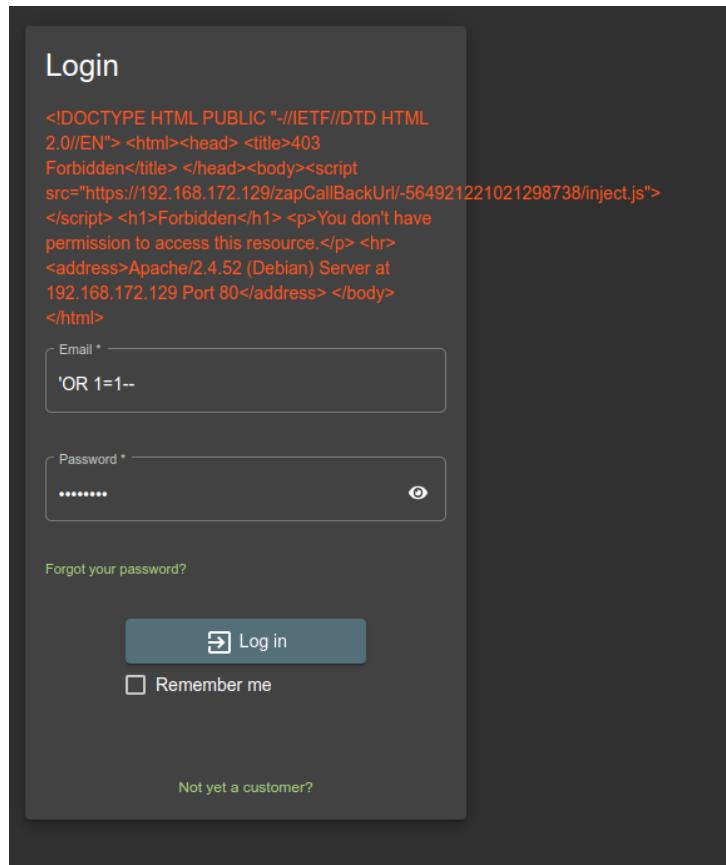


Figura 39 - WAF bloqueia ataque de SQL Injection (WSTG-INPV-01)

7. Conclusão

Com este trabalho tivemos a possibilidade de analisar vários pontos que podem ser vulneráveis numa aplicação web e como testar algum destes, usando como auxílio as guidelines do WSTG. Reconhecemos que esta análise é algo superficial e que certamente com um pouco mais atenção podemos ser capazes de encontrar mais vulnerabilidades mas no geral os objetivos foram atingidos. Também tivemos a possibilidade de experimentar software como o ZAP, utilitário do kali linux, e apache (a funcionar como WAF) o que foi interessante e alargou o nosso conhecimento acerca das ferramentas existentes que podemos vir a utilizar no futuro.

8. Referências

- <https://owasp.org/www-project-web-security-testing-guide/v41/>
- <https://www.linode.com/docs/guides/securing-apache2-with-modsecurity/>
- https://www.digitalocean.com/community/tutorials/how-to-use-apache-http-server-as-reverse-proxy-using-mod_proxy-extension
- <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>
- <https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP>

- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options>
- <https://pwning.owasp-juice.shop/>