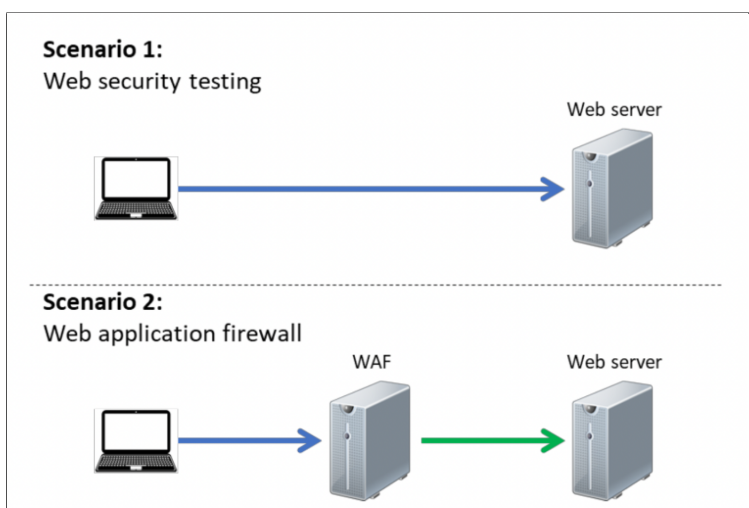


Web Application Security

Trabalho Prático 3



Mestrado em Engenharia Informática

Segurança em Tecnologias da Informação
2021 / 2022

PL2	Gabriel Fernandes	2018288117	gabrielf@student.dei.uc.pt
PL2	Miguel Rabuge	2018293728	rabuge@student.dei.uc.pt

Conteúdo

Introdução	2
Estrutura do trabalho prático	3
1 Rede, Servidores e Serviços	3
Phase 1 - Web Application Security Testing	4
2 Information Gathering	4
3 Configuration and Deployment Management Testing	5
4 Identity Management Testing	5
5 Authentication Testing	6
6 Authorization Testing	7
7 Session Management Testing	7
8 Input Validation Testing	8
9 Testing for Error Handling	9
10 Client Side Testing	10
Phase 2 - Web Application Firewall	12
11 Information Gathering	12
12 Configuration and Deployment Management Testing	12
13 Identity Management Testing	13
14 Authentication Testing	13
15 Authorization Testing	14
16 Session Management Testing	14
17 Input Validation Testing	14
18 Testing for Error Handling	14
19 Client Side Testing	15
Conclusão	16

Introdução

Neste trabalho prático procuramos testar a segurança de uma aplicação web, a OWASP JuiceShop. Este é um site desenhado com múltiplas vulnerabilidades para motivos educacionais. Na primeira parte deste relatório, iremos focar-nos na identificação de algumas destas vulnerabilidades, seguindo o OWASP Web Security Testing Guide (WSTG). Para tal, iremos também utilizar a ferramenta OWASP Zed Attack Proxy (ZAP) para executar alguns destes ataques. Considerando a quantidade de ataques possíveis, iremos focarmos em pelo menos um por secção, de modo a que possamos cobrir um espectro alargado de vulnerabilidades e ataques.

Estrutura do trabalho prático

1 Rede, Servidores e Serviços

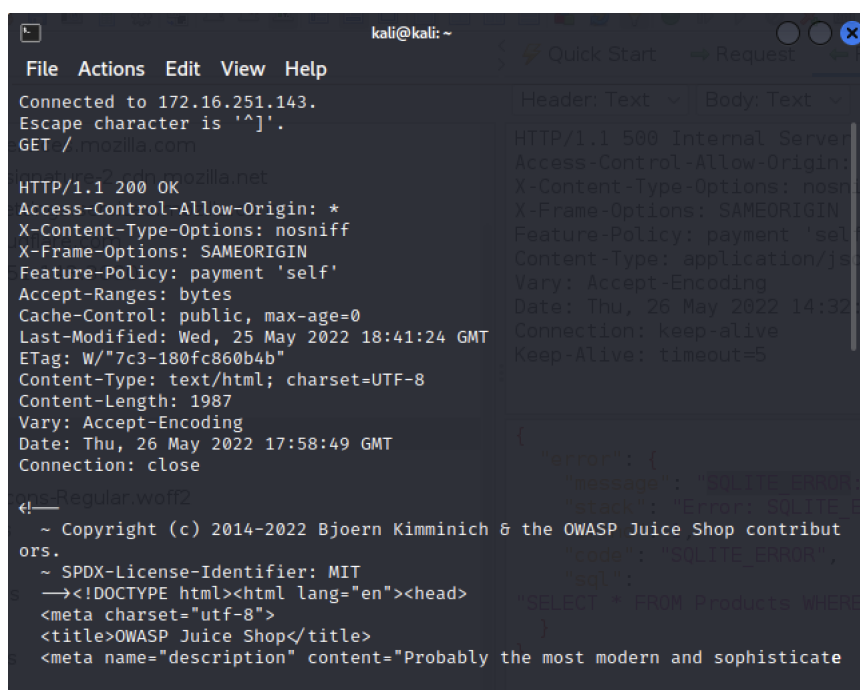
Em termos de rede, possuímos duas máquinas virtuais na mesma rede. A primeira com kali linux + OWASP ZAP e a segunda com debian, onde se encontra a JuiceShop Web Application, recorrendo a um docker container, e a Web Application Firewall (WAF) através de um servidor Apache 2 com ModSecurity. Os scripts para a instalação do zap, ativação da JuiceShop e configuração da WAF podem ser encontrados em [testing/zap.bash](#), [shop/shop.bash](#) e [shop/waf.bash](#), respetivamente.

Phase 1 - Web Application Security Testing

2 Information Gathering

Com o intuito de verificar se conseguíamos obter informação o tipo de server em que o site está hospedado, resolvemos efetuar um pedido ao server por via do **telnet**, efetuando o comando "**telnet (ip do server) (porto do server)**", seguido de "**GET /**" e dois *new lines*, tal como especificado no ponto WSTG-INFO-02 do manual. O output deste pedido pode ser visto na figura 1. Ao contrário do que é encontrado no manual, não conseguimos obter informações sobre o servidor ou o tipo de máquina onde este está a correr.

Ao fazermos login na aplicação, um **POST** vai ser enviado, levando a informação de login no *body* do pedido, no entanto esta informação vai em *plain text*, não oferecendo qualquer tipo de segurança ao utilizador e permite a um terceiro obter a informação de login deste. Assim sendo, estamos na presença de um possível *entry point*, como explicado no ponto WSTG-INFO-06 do manual. Este cenário pode ser observado na figura 2.



```
kali@kali: ~  
File Actions Edit View Help  
Connected to 172.16.251.143.  
Escape character is '^['.  
GET /  
HTTP/1.1 200 OK mozilla.net  
Access-Control-Allow-Origin: *  
X-Content-Type-Options: nosniff  
X-Frame-Options: SAMEORIGIN  
Feature-Policy: payment 'self'  
Content-Type: application/javascript  
Vary: Accept-Encoding  
Date: Thu, 26 May 2022 14:32:14 GMT  
Connection: keep-alive  
Keep-Alive: timeout=5  
<!-- Regular woff2  
~ Copyright (c) 2014-2022 Bjoern Kimminich & the OWASP Juice Shop contributors.  
~ SPDX-License-Identifier: MIT  
--><!DOCTYPE html><html lang="en"><head>  
<meta charset="utf-8">  
<title>OWASP Juice Shop</title>  
<meta name="description" content="Probably the most modern and sophisticated
```

Figura 1. Telnet para o ip e porto do site.

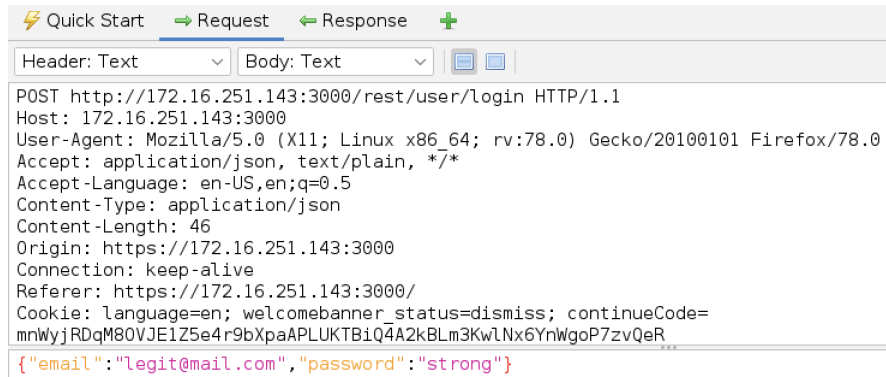


Figura 2. Pedido realizado aquando de um login na aplicação.

3 Configuration and Deployment Management Testing

Ao efetuarmos um *active scan* no ZAP, ficamos a conhecer uma diretoria na aplicação (*/ftp*) que contém, para além do ficheiro que contém os termos de uso (possível obter indo à aba "About us" da aplicação e clicando no link presente nesta página), tem também outros ficheiros cujos constituintes são confidenciais. Um exemplo é o ficheiro *acquisitions.md*. Esta falha enquadra-se com o abordado no ponto WSTG-CONF-04 do manual. Na figura 3 podem ser vistos os ficheiros e pastas guardadas na diretoria *ftp*.

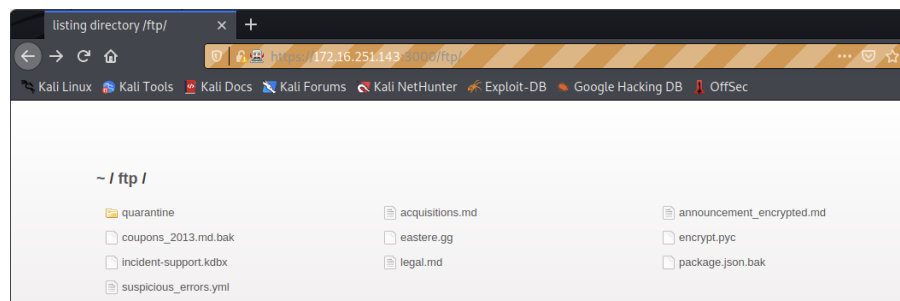


Figura 3. Diretoria *ftp* da aplicação acedida através do navegador.

4 Identity Management Testing

Aquando da criação de uma conta no site, apenas um email é pedido, que na verdade pode ser apenas uma string que pareça um email, pois nenhuma veri-

ificação de existência deste email é realizada. Assim sendo, muito facilmente se consegue automatizar o processo de criar contas "fake" na plataforma e o login com as mesmas, levando a um possível congestionamento ou até mesmo queda do site. Esta falha enquadra-se no ponto WSTG-IDNT-02 do manual.

5 Authentication Testing

Ao navegar pelo site facilmente encontramos um email que aparenta ser de um administrador. O email é **admin@juice-sh.op** e pode ser encontrado numa review ao produto "Apple Juice". Ao descobrir o email, tentámos entrar na conta a que este está associado. Visto que o email parece pertencer a um administrador, tentámos a password "admin" e variantes com números até que descobrimos a password correta, "admin123". Claramente estamos na presença de uma falha discutida no ponto WSTG-ATHN-02 do manual, dado que a password "admin123" parece ser uma default utilizada no desenvolvimento da aplicação.

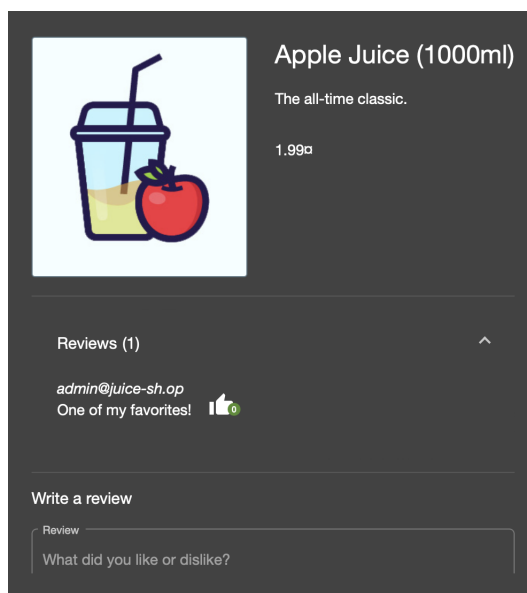


Figura 4. Email do administrador

Por outro lado, ao fazer um active scan ao login, através do ZAP, detetamos que existe uma possibilidade de SQL injection. De facto, como iremos ver mais à frente, ao colocar o email como " ' or 1=1 --" e a password como qualquer outra string, permite-nos entrar novamente como administrador (admin@juice-sh.op) na aplicação. Esta vulnerabilidade enquadra-se nos testes de *Authentication Schema Bypassing* no ponto WSTG-ATHN-04 do manual.

6 Authorization Testing

Em termos de authorization Testing, estando logged in, como administrador, por exemplo, fizemos um *Manual Penetration Test* às *reviews* dos utilizadores, na tab *Customer Feedback*. Pode verificar-se na figura 5 que o campo “Author” está disabled, de modo que não dá para editar. No entanto, se submetermos o form, passando pelo ZAP, verificamos que podemos agora editar o objeto, como podemos ver na figura 5. Deste modo, um utilizador que não tinha autorização para uma determinada funcionalidade (fazer uma review em nome de um utilizador que não ele mesmo), consegue ultrapassar essa condição editando o form interceptado, nomeadamente o campo “UserId”. É, portanto, uma vulnerabilidade de *Bypassing Authorization Schema* cujo ID é WSTG-ATHZ-02 no manual.

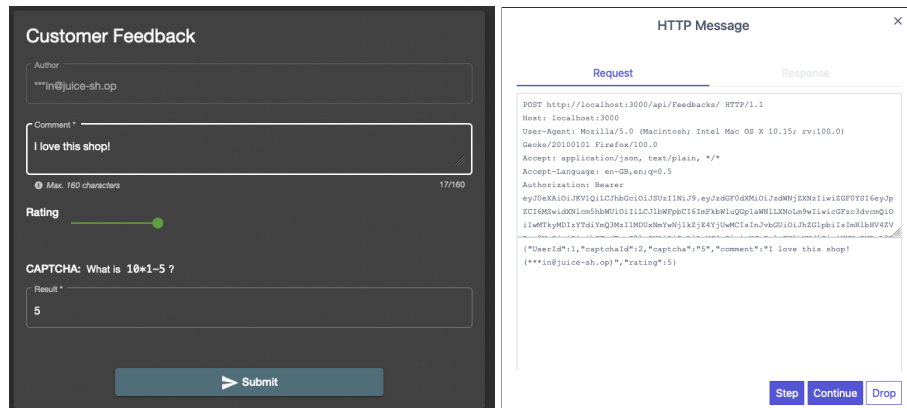
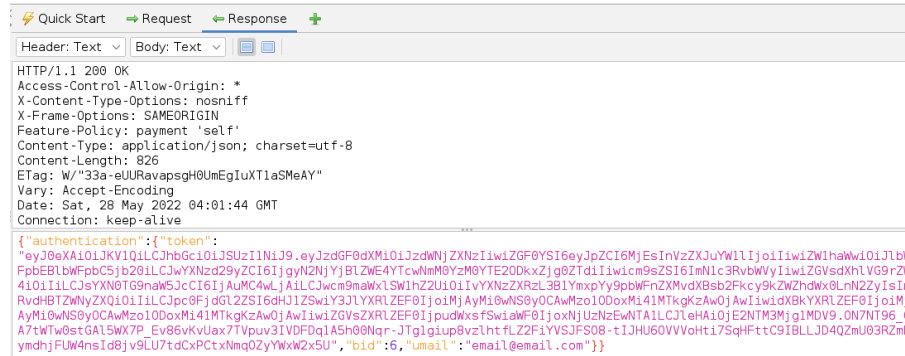
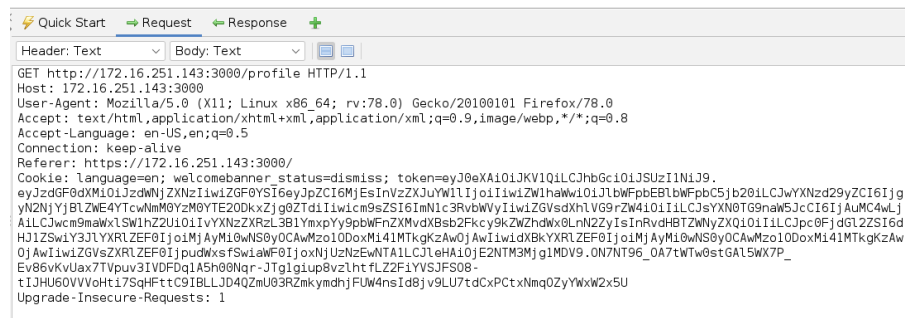


Figura 5. Review Form

7 Session Management Testing

Quando o utilizador efetua o login no site, é-lhe retornado um *token* que o identifica perante a aplicação. Este token é enviado em *plain text* no *body* da resposta do pedido de *login*, fazendo com que este seja facilmente interceptado por terceiros. Este *token* é depois utilizado em vários pedidos realizados pelo utilizador ao site. Assim, sendo, se o *token* for obtido por outra pessoa, essa vai conseguir fazer-se passar pelo utilizador e obtém acesso à aplicação sem sequer ter de criar uma conta. Esta falha enquadra-se no ponto WSTG-SESS-04 do manual. Um exemplo deste cenário pode ser observado nas figuras 6 e 7, onde na primeira vemos o *token* ser retornado pelo servidor e na segunda o mesmo a ser utilizado para autenticar o utilizador no pedido.

Figura 6. Resposta a um pedido válido de *login*.Figura 7. Pedido para ver o perfil de utilizador. (O *token* encontra-se na *Cookie*)

8 Input Validation Testing

Recorrendo ao OWASP ZAP, podemos iniciar um Fuzz attack ao login form, no qual tínhamos encontrado uma vulnerabilidade para SQLi. Deste modo, configuramos a ferramenta, nomeadamente o Fuzzer, para SQLi, como se pode observar na imagem 8. Assim, obtemos alguns ataques que conseguiram traspasar o sistema, como “ ’ or 1=1 --”, observável na figura 9 o que confirma a suspeita da vulnerabilidade de SQLi, apresentando-se como uma falha de validação do input, susceptível a SQL injections, como referido no ponto WSTG-INPV-05.

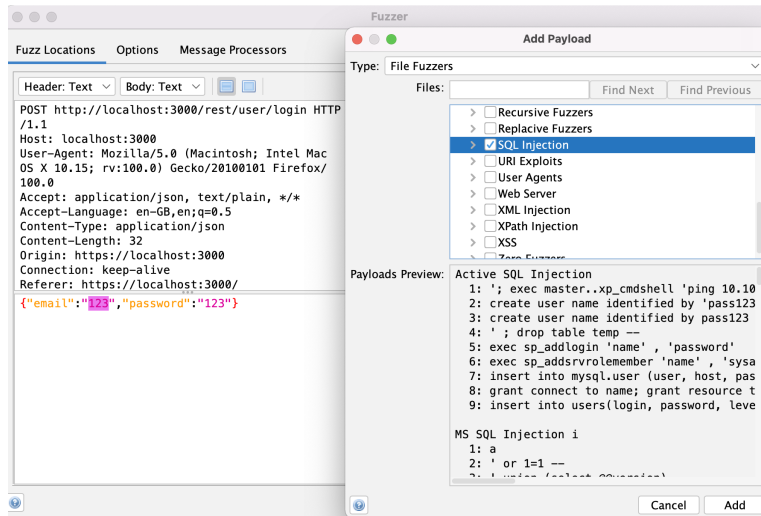


Figura 8. Fuzzer Setup

Task ID	Message Type	Code	Reason	RTT	Payloads	Size Resp. Header	Size Resp. Body
11	Fuzzed	200 OK		396 ms	' or 1=1 --	363 bytes	834 bytes
41	Fuzzed	200 OK		404 ms	' or username is not NULL or username = '	363 bytes	834 bytes
103	Fuzzed	200 OK		485 ms	' or 1=1--	363 bytes	834 bytes
111	Fuzzed	200 OK		398 ms	' or 1=1--	363 bytes	834 bytes
113	Fuzzed	200 OK		416 ms	' or 1=1 /*	363 bytes	834 bytes
135	Fuzzed	200 OK		396 ms	' or username like char(37);	363 bytes	834 bytes
143	Fuzzed	200 OK		366 ms	' or 1/*	363 bytes	834 bytes
157	Fuzzed	200 OK		372 ms	a' or 1=1; --	363 bytes	834 bytes

Figura 9. Fuzzer results

9 Testing for Error Handling

Relativamente a Error handling e Stack Trace dumping, através do active scan efetuado anteriormente ao login, podemos observar no body da response que se encontra um erro que não foi protegido, apresentando a stack trace e as queries utilizadas para a autenticação, como se pode ver na figura 10. Este erro, foi identificado pelo ZAP como *High Priority Alert* (SQLi), e foi conseguido ao colocar “'” no Email e “123” na Password. Deste modo, encontramos-nos perante uma vulnerabilidade de *Improper error Handling e Stack Trace*, identificados com os IDs WSTG-ERRH-01 e WSTG-ERRH-02.

```

HTTP/1.1 500 Internal Server Error
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
Content-Type: application/json; charset=utf-8
Vary: Accept-Encoding
Date: Fri, 27 May 2022 02:56:47 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"message": "SQLITE_ERROR: unrecognized token: \"202cb962ac59075b964b07152d234b70\\\"",
"stack":
  "Error\n    at Database.<anonymous> (/juice-shop/node_modules/sequelize/lib/dialects/sqlite/query.js:179:27)\n
  at Database.serialize (<anonymous>)\n    at /juice-shop/node_modules/sequelize/lib/dialects/sqlite/query.js:177:50\n
  \n    at new Promise (<anonymous>)\n    at Query.run (/juice-shop/node_modules/sequelize/lib/dialects/sqlite/query\n
  .js:177:12)\n    at /juice-shop/node_modules/sequelize/lib/sequelize.js:310:28\n    at runMicrotasks (<anonymous>)\n
  \n    at processTicksAndRejections (node:internal/process/task_queues:96:5)",
"name": "SequelizeDatabaseError",
"parent": {
  "errno": 1,
  "code": "SQLITE_ERROR",
  "sql":
    "SELECT * FROM Users WHERE email = '' AND password = '202cb962ac59075b964b07152d234b70' AND deletedAt IS NULL"
},
"original": {
  "errno": 1,
  "code": "SQLITE_ERROR",
  "sql":
    "SELECT * FROM Users WHERE email = '' AND password = '202cb962ac59075b964b07152d234b70' AND deletedAt IS NULL"
},
"sql":
  "SELECT * FROM Users WHERE email = '' AND password = '202cb962ac59075b964b07152d234b70' AND deletedAt IS NULL",
"message": "SQLITE_ERROR: unrecognized token: \"202cb962ac59075b964b07152d234b70\\\""
}

```

Figura 10. Error Stack Trace e Queries

10 Client Side Testing

Do ponto de vista de *Client Side Testing*, procuramos verificar a segurança da JuiceShop em relação a ataques XSS. Deste modo, fizemos um *fuzz attack* à *rest/product/search*, com vista a verificar se está protegida contra este tipo de ataques, como podemos ver na figura 11. Terminado o ataque, verificamos na figura 12 que a aplicação aceita, sem erro, alguns destes ataques. Ao introduzir um desses ataques na search bar: “XSS ”, verificamos que a aplicação é de facto vulnerável DOM XSS, como podemos observar na figura 13, identificados pelo ID WSTG-CLNT-01 do manual.

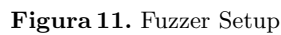
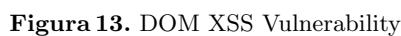


Figura 12. Fuzzer results



Phase 2 - Web Application Firewall

Nesta fase procuramos corrigir os erros que identificamos com recurso à WAF. Para tal, o nível de paranoia utilizado é o 3, dado que é aquele que nos aparenta ser mais adequado.

11 Information Gathering

Do ponto de vista de *Information Gathering*, efetuando o comando "telnet (ip do server) (porto do server)", obtemos o resultado observável na figura 14. Podemos constatar de imediato, que obtemos uma página de erro, 403 Forbidden, em vez de toda a informação que obtivemos sem a WAF, na figura 1. No entanto, desta vez, o tipo de servidor (apache) e sistema operativo (debian) é exposto através deste comando, pelo que não podemos afirmar que a WAF protegeu ataques no âmbito de *Information Gathering*, totalmente.



```
Trying 192.168.193.148...
Connected to 192.168.193.148.
Escape character is '^]'.
GET /
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access this resource.</p>
<hr>
<address>Apache/2.4.53 (Debian) Server at sti.local Port 80</address>
</body></html>
Connection closed by foreign host.
```

Figura 14. Telnet para o ip e porto do site com a WAF.

12 Configuration and Deployment Management Testing

Repetindo o mesmo ataque, tentando aceder ao diretório */ftp*, constatamos que a WAF não protege o acesso à mesma.

13 Identity Management Testing

Novamente, ao tentar criar um novo utilizador (e.g. sti@sti.sti), a aplicação permite criar este tipo de contas, pelo que a WAF não altera este facto.

14 Authentication Testing

Relativamente ao ponto WSTG-ATHN-02 de *Authentication Testing*, a WAF não protege a aplicação contra o ataque feito na *Phase 1*, dado que a informação sobre o email do administrador está exposta na página, e a password é relativamente óbvia. Tendo em conta que não há nenhuma proteção para *brute force* do login, do ponto de vista da WAF, é apenas feito um login legítimo. Relativamente à SQLi do login (“ ’ or 1=1 --”) verificamos que a WAF retorna uma página com o erro 403 Forbidden, como podemos ver na figura 15, mas permite *logins* legítimos.

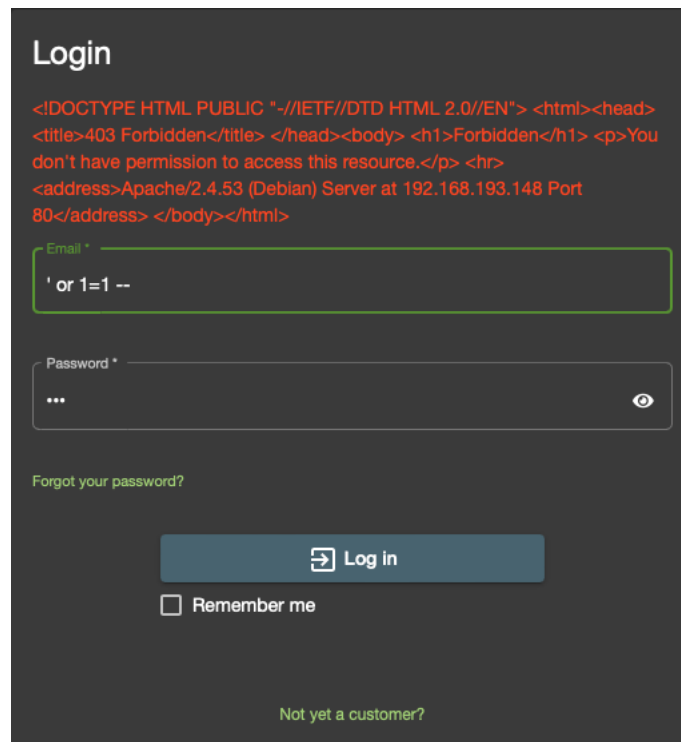


Figura 15. Proteção de SQL injection no login pela WAF.

15 Authorization Testing

Relativamente ao teste efetuado sobre *Authorization Testing*, constatamos que com o nível de paranoia 2, este ataque não é protegido, permitindo intercetar o *request* e alterá-lo, ao qual o servidor responde normalmente. Com o nível de paranoia 3, o ataque é protegido, no entanto uma review legítima deixa de ser permitida, pelo que esta funcionalidade deixa de existir na prática.

16 Session Management Testing

Em relação à exposição do *token*, a WAF não aparenta fazer nada, dado que o *token* continua a ser transmitido em *plain text* no *body* da resposta do servidor ao cliente aquando do *login*. Assim sendo, o problema apresentado na *Phase 1* persiste.

17 Input Validation Testing

Como referido anteriormente, testando SQL injections no login, verificamos que a WAF protege contra este tipo de ataques. Analogamente ao ataque efetuado, aplicando um *Fuzz Attack* de SQL injections ao Login, desta vez protegido com a WAF, verificamos que nenhum dos ataques teve sucesso, sendo todos bloqueados, emitindo o erro 403 Forbidden, como podemos observar na figura 16

Task ID	Message Type	Code	Reason	Payloads	RTT	Size Resp. Header	Size Resp. Body
100	Fuzzed	403	Forbidden	6	5 ms	216 bytes	280 bytes
101	Fuzzed	403	Forbidden	' '6	11 ms	216 bytes	280 bytes
102	Fuzzed	403	Forbidden	(6)	7 ms	216 bytes	280 bytes
103	Fuzzed	403	Forbidden	' or 1=1--	8 ms	216 bytes	280 bytes
104	Fuzzed	403	Forbidden	or 1=1	7 ms	216 bytes	280 bytes
105	Fuzzed	403	Forbidden	' or '1'='1	4 ms	216 bytes	280 bytes
106	Fuzzed	403	Forbidden	; or '1'='1	5 ms	216 bytes	280 bytes
108	Fuzzed	403	Forbidden	' or '7659'='7659	6 ms	216 bytes	280 bytes
110	Fuzzed	403	Forbidden	' --	6 ms	217 bytes	280 bytes
111	Fuzzed	403	Forbidden	' or 1=1--	5 ms	217 bytes	280 bytes
113	Fuzzed	403	Forbidden	' or 1=1 /*	6 ms	216 bytes	280 bytes
114	Fuzzed	403	Forbidden	or 1=1--	5 ms	216 bytes	280 bytes
115	Fuzzed	403	Forbidden	' or 'a'='a	9 ms	216 bytes	280 bytes
117	Fuzzed	403	Forbidden) or ('a'='a	10 ms	216 bytes	280 bytes
118	Fuzzed	403	Forbidden	admin' or '	13 ms	217 bytes	280 bytes
119	Fuzzed	403	Forbidden	' select * from information_schema.tables--	14 ms	216 bytes	280 bytes

Figura 16. Fuzzing de SQL injections no login com WAF

18 Testing for Error Handling

Após ativarmos a WAF, verificámos que o teste efetuado na *Phase 1* já não é possível, retornando um erro semelhante ao da figura 15. No entanto, pensamos que este comportamento é devido ao teste utilizar uma SQL injection e a WAF conseguir parar estas. Deste modo, nada nos garante que a WAF corrija erros que não estão devidamente *handled* pela aplicação.

19 Client Side Testing

Ao repetir o ataque da *Phase 1*, verificamos que muitos dos ataques XSS com sucesso anteriormente são bloqueados pela WAF, como podemos ver na figura 17. Porém, quando procuramos aplicar o mesmo ataque na search bar (“XSS ”), verificamos que a aplicação continua vulnerável a DOM XSS, pelo que a WAF não tem qualquer influência, dado que a o código do script nunca alcança o servidor, devido ao carácter “#”, dado que os browsers não dão forward dessa informação, executando o script apenas ao nível do cliente. No entanto, ao clicar no *hyperlink*, ao contrário da *Phase 1*, somos redirecionados para uma *Forbidden page*.

Task ID	Message Type	Code	Reason	Payloads	RTT	Size Res...	Size Resp. Body
74	Fuzzed	200	OK	<SCRIPT SRC==http://testsite.com/xss.js	76 ms	406 bytes	12,880 bytes
76	Fuzzed	200	OK	<IMG SRC=="javascript:alert('XSS')"	75 ms	406 bytes	12,880 bytes
77	Fuzzed	200	OK	<FRAME SRC==http://testsite.com/scriptlet.html <	63 ms	406 bytes	12,880 bytes
80	Fuzzed	200	OK	<SCRIPT >a=/XSS/ alert(a.source)</SCRIPT>	60 ms	406 bytes	12,880 bytes
94	Fuzzed	200	OK	res://c:\program%20files\adobe\acrobat%207.0\acrobat\acrobat.dll/#2/#210	16 ms	401 bytes	30 bytes
136	Fuzzed	200	OK	withidocument.__parent__alert(1)	18 ms	401 bytes	30 bytes
193	Fuzzed	200	OK	alert(1)	12 ms	401 bytes	30 bytes
191	Fuzzed	200	OK	res://c:\program%20files\adobe\acrobat%207.0\acrobat\acrobat.dll/#2/#210	34 ms	401 bytes	30 bytes
194	Fuzzed	200	OK	A=alert,A(1)	35 ms	401 bytes	30 bytes
195	Fuzzed	200	OK	+alert(0)+	44 ms	401 bytes	30 bytes
3	Fuzzed	403	Forbidden	XSS STYLE=xss:e/**/xpression(alert(XSS))>	77 ms	217 bytes	280 bytes
2	Fuzzed	403	Forbidden	XSS STYLE=xss:e/**/xpression(alert(XSS))>	170 ms	217 bytes	280 bytes
1	Fuzzed	403	Forbidden	</XSS STYLE=xss:expression(alert(XSS))>	201 ms	217 bytes	280 bytes
4	Fuzzed	403	Forbidden	XSS/~~/STYLE=xss:e/**/xpression(alert(XSS))>	38 ms	217 bytes	280 bytes
5	Fuzzed	403	Forbidden	"><script>alert(XSS)</script>	25 ms	217 bytes	280 bytes
6	Fuzzed	403	Forbidden	<body onload=a2=lyeval;a1=[x.a2.y('af'+ter)];.....=a1.x_c(1);;	44 ms	216 bytes	280 bytes
9	Fuzzed	403	Forbidden	<body/s/onload=x=[doc.parent.document].x.doc.write(1)	46 ms	216 bytes	280 bytes
8	Fuzzed	403	Forbidden	<body onload=a1=[x:document].....=a1.x_c.write(1);;	47 ms	216 bytes	280 bytes
7	Fuzzed	403	Forbidden	<body onload=a1=[x:this.parent.document].a1.x.write(1);>	52 ms	216 bytes	280 bytes
10	Fuzzed	403	Forbidden	<body/???/onload=x=[doc:parent[document]].x.doc.write(1)	51 ms	216 bytes	280 bytes

Figura 17. Fuzzing de XSS injections em /rest/products/search, com WAF

Conclusão

Com este trabalho pretendíamos inspecionar a JuiceShop relativamente a vulnerabilidades de segurança e posteriormente protegê-la com recurso a uma Web Application Firewall (WAF). Identificamos múltiplas vulnerabilidades na *Phase 1*, deixando muitas mais por diagnosticar, pelo que constatamos que a aplicação web é extremamente insegura. Como pode ser visto na *Phase 2*, alguns dos erros encontrados foram corrigidos, como por exemplo as SQL injections, com recurso à WAF. Conseguimos perceber também que esta tem as suas limitações, pelo que não consegue corrigir muitos dos erros identificados devido a natureza dos mesmos.