Obtaining Session Identifiers without User Interaction

There are multiple attacking techniques through which a bug bounty hunter or penetration tester can obtain session identifiers. These attacking techniques can be split into two categories:

1. Session ID-obtaining attacks without user interaction
2. Session ID-obtaining attacks requiring user interaction

This section will focus on Session ID-obtaining attacks that do not require user interaction.

Obtaining Session Identifiers via Traffic Sniffing

Traffic sniffing is something that most penetration testers do when assessing a network's security from the inside. You will usually see them plugging their laptops or Raspberry Pis into available ethernet sockets. Doing so allows them to monitor the traffic and gives them an idea of the traffic going through the network (segment) and the services they may attack. Traffic sniffing requires the attacker and the victim to be on the same local network. Then and only then can HTTP traffic be inspected by the attacker. It is impossible for an attacker to perform traffic sniffing remotely.

You may have noticed that we mentioned HTTP traffic. This is because HTTP is a protocol that transfers data unencrypted. Thus if an attacker is monitoring the network, they can catch all kinds of information such as usernames, passwords, and even session identifiers. This type of information will be more challenging and, most of the time, impossible to obtain if HTTP traffic is encrypted through SSL or IPsec.

To sum up, obtaining session identifiers through traffic sniffing requires:

- The attacker must be positioned on the same local network as the victim
- Unencrypted HTTP traffic

There are numerous packet sniffing tools. In this module, we will use Wireshark. Wireshark has an inbuilt filter function that allows users to filter traffic for a specific protocol such as HTTP, SSH, FTP, and even for a particular source IP address.

Let us practice session hijacking via traffic sniffing against a web application. This web application is the target we can spawn on the exercise at the end of this section.

Navigate to the end of this section and click on `Click here to spawn the target system!` or the `Reset Target` icon. Use the provided Pwnbox or a local VM with the supplied VPN key to reach the target application and follow along. Don't forget to configure the specified vhost (`xss.htb.net`) to access the application.

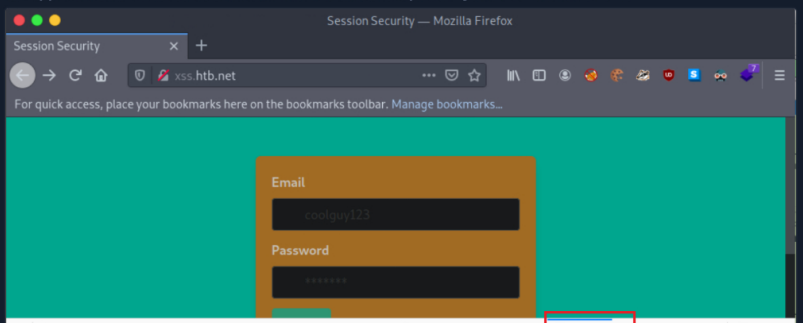
A quick way to specify this (and any other) vhost in your attacking system is the below:

Obtaining Session Identifiers without User Interaction

```
MisaelMacias@htb[/htb]$ IP=ENTER SPAWNED TARGET IP HERE
MisaelMacias@htb[/htb]$ printf "%s\t%s\n\n" "$IP" "xss.htb.net csrf.htb.net oredirect.htb.net
```

Part 1: Simulate the attacker

Navigate to `http://xss.htb.net` and, using Web Developer Tools (Shift+Ctrl+I in the case of Firefox), notice that the application uses a cookie named `auth-session` most probably as a session identifier.



? Go to Questions

Table of Contents

Introduction to Sessions

Session Attacks

Session Hijacking
Session Fixation
Obtaining Session Identifiers without User Interaction
Cross-Site Scripting (XSS)
Cross-Site Request Forgery
Cross-Site Request Forgery (GET-based)
Cross-Site Request Forgery (POST-based)
XSS & CSRF Chaining
Exploiting Weak CSRF Tokens
Additional CSRF Protection Bypasses
Open Redirect
Remediation Advice

Skills Assessment

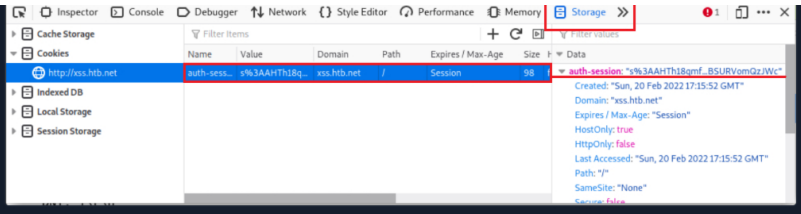
Session Security - Skills Assessment

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left

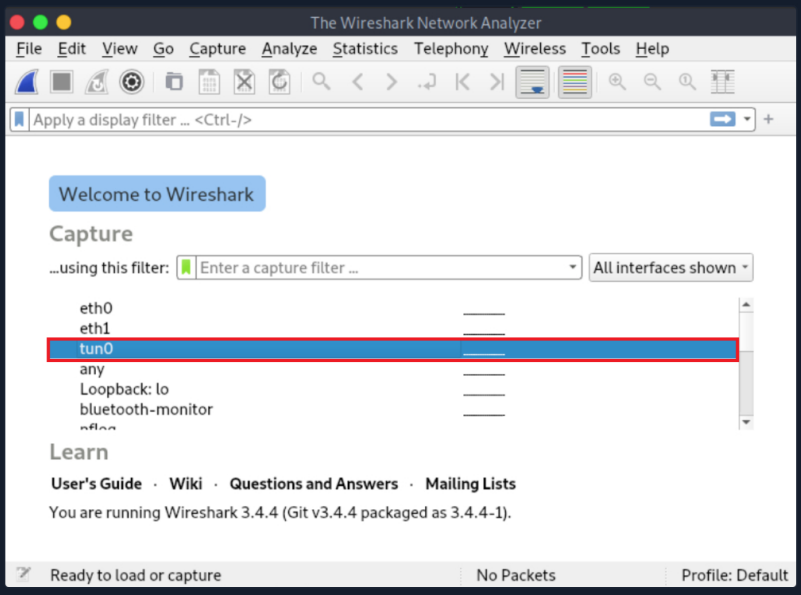


Now fire up Wireshark to start sniffing traffic on the local network as follows.

Obtaining Session Identifiers without User Interaction

```
MisaelMacias@htb[/htb]$ sudo -E wireshark
```

You will come across the below.



Right-click "tun0" and then click "Start capture"

Part 2: Simulate the victim

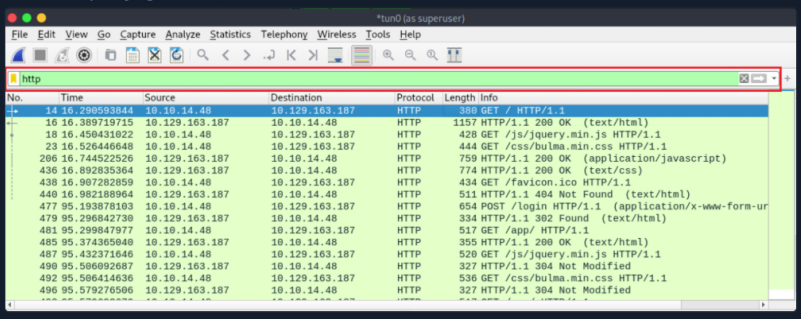
Navigate to `http://xss.htb.net` through a `New Private Window` and login to the application using the credentials below:

- `Email: heavycat106`
- `Password: rocknrol`

This is an account that we created to look into the application!

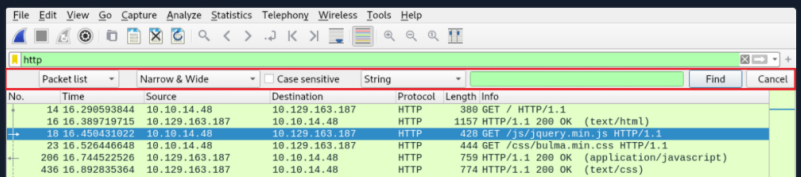
Part 3: Obtain the victim's cookie through packet analysis

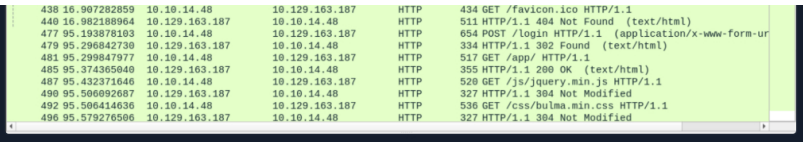
Inside Wireshark, first, apply a filter to see only HTTP traffic. This can be done as follows (don't forget to press Enter after specifying the filter).



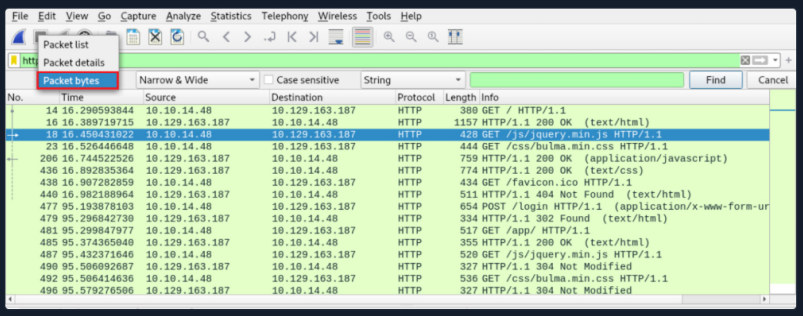
Now search within the Packet bytes for any `auth-session` cookies as follows.

Navigate to `Edit` -> `Find Packet`



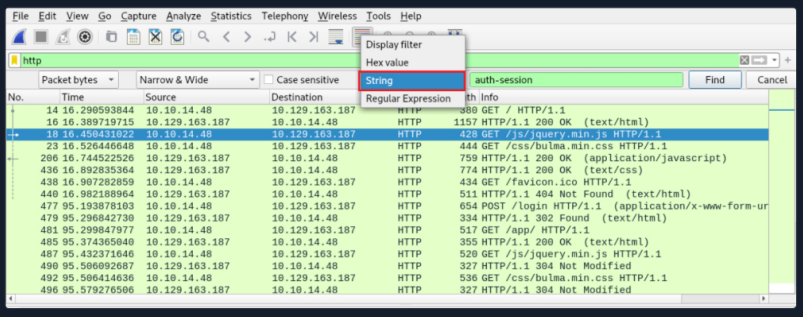


Left-click on `Packet list` and then click `Packet bytes`

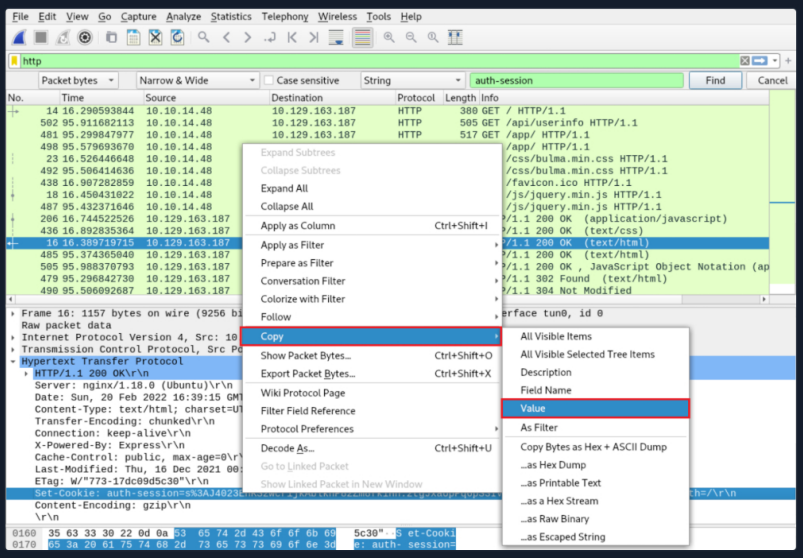


Select `String` on the third drop-down menu and specify `auth-session` on the field next to it. Finally, click `Find`.

Wireshark will present you with the packets that include an `auth-session` string.

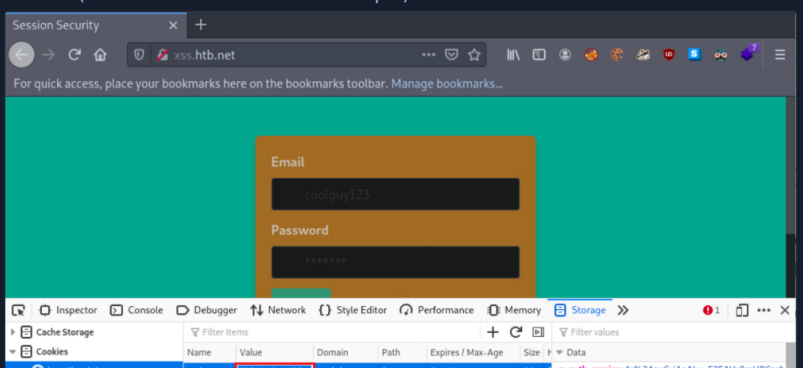


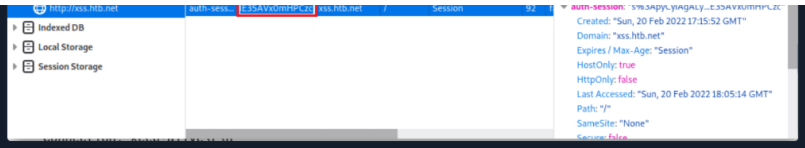
The cookie can be copied by right-clicking on a row that contains it, then clicking on `Copy` and finally clicking `Value`.



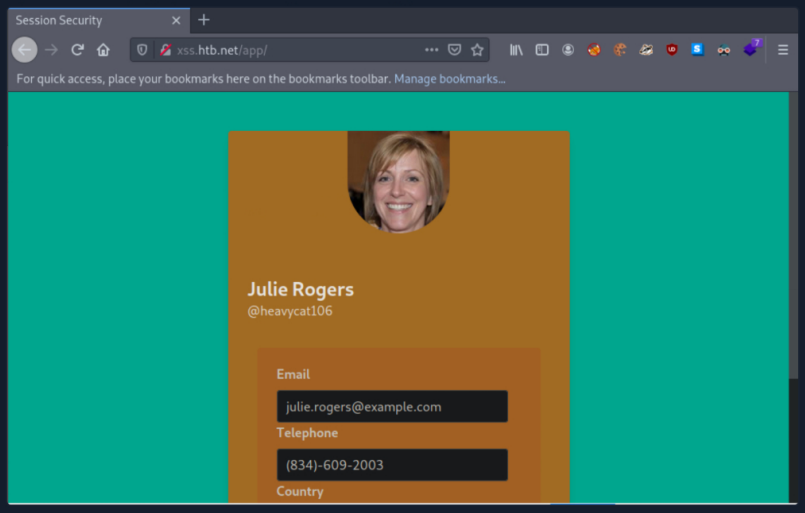
Part 4: Hijack the victim's session

Back to the browser window using which you first browsed the application (not the Private Window), open Web Developer Tools, navigate to *storage*, and change your current cookie's value to the one you obtained through Wireshark (remember to remove the `auth-session=` part).





If you refresh the page, you will see that you are now logged in as the victim!



Obtaining Session Identifiers Post-Exploitation (Web Server Access)

Note: The below examples cannot be replicated in this section's lab exercise!

During the post-exploitation phase, session identifiers and session data can be retrieved from either a web server's disk or memory. Of course, an attacker who has compromised a web server can do more than obtain session data and session identifiers. That said, an attacker may not want to continue issuing commands that increase the chances of getting caught.

PHP

Let us look at where PHP session identifiers are usually stored.

The entry `session.save_path` in `PHP.ini` specifies where session data will be stored.

Obtaining Session Identifiers without User Interaction

```
MisaelMacias@htb[/htb]$ locate php.ini
MisaelMacias@htb[/htb]$ cat /etc/php/7.4/cli/php.ini | grep 'session.save_path'
MisaelMacias@htb[/htb]$ cat /etc/php/7.4/apache2/php.ini | grep 'session.save_path'
```

```
[root@tilix ~]$ locate php.ini
/etc/php/7.4/apache2/php.ini
/etc/php/7.4/cli/php.ini
/usr/lib/php/7.4/php.ini-development
/usr/lib/php/7.4/php.ini-production
/usr/lib/php/7.4/php.ini-production.cli
/var/www/html/opencart/upload/php.ini
[root@tilix ~]$ cat /etc/php/7.4/cli/php.ini | grep 'session.save_path'
;     session.save_path = "N;/path"
;     session.save_path = "N;MODE;/path"
;session.save_path = "/var/lib/php/sessions"
;       (see session.save_path above), then garbage collection does *not*
[root@tilix ~]$ cat /etc/php/7.4/apache2/php.ini | grep 'session.save_path'
;     session.save_path = "N;/path"
;     session.save_path = "N;MODE;/path"
;session.save_path = "/var/lib/php/sessions"
;       (see session.save_path above), then garbage collection does *not*
[root@tilix ~]$
```

In our default configuration case it's `/var/lib/php/sessions`. Now, please note a victim has to be authenticated for us to view their session identifier. The files an attacker will search for use the name convention `sess_<sessionID>`.

How a PHP session identifier looks on our local setup.

Name	Value	Domain	Path	Expir…	Size	HttpO…	Secure	Same…	Same…	Priority
PHPSESSID	s6kitq8d3071rmlvbfitpim9mm	192.1…	/	Session	35					Medium

The same PHP session identifier but on the webserver side looks as follows.

Obtaining Session Identifiers without User Interaction

```
MisaelMacias@htb[/htb]$ ls /var/lib/php/sessions
MisaelMacias@htb[/htb]$ cat //var/lib/php/sessions/sess_s6kitq8d3071rmlvbfitpim9mm
```

```
[root@tilix ~]$ ls /var/lib/php/sessions
sess_s6kitq8d3071rmlvbfitpim9mm
[root@tilix ~]$ cat //var/lib/php/sessions/sess_s6kitq8d3071rmlvbfitpim9mm
username|s:5:"admin";#
[root@tilix ~]$
```

As already mentioned, for a hacker to hijack the user session related to the session identifier above, a new cookie must be created in the web browser with the following values:

- cookie name: PHPSESSID
- cookie value: s6kitq8d3071rmlvbfitpim9mm

Java

Now, let us look at where Java session identifiers are stored.

According to the Apache Software Foundation:

"The `Manager` element represents the *session manager* that is used to create and maintain HTTP sessions of a web application.

Tomcat provides two standard implementations of `Manager`. The default implementation stores active sessions, while the optional one stores active sessions that have been swapped out (in addition to saving sessions across a server restart) in a storage location that is selected via the use of an appropriate `Store` nested element. The filename of the default session data file is `SESSIONS.ser`."

You can find more information here.

.NET

Finally, let us look at where .NET session identifiers are stored.

Session data can be found in:

- The application worker process (aspnet_wp.exe) - This is the case in the *InProc Session mode*
- StateServer (A Windows Service residing on IIS or a separate server) - This is the case in the *OutProc Session mode*
- An SQL Server

Please refer to the following resource for more in-depth details: Introduction To ASP.NET Sessions

Obtaining Session Identifiers Post-Exploitation (Database Access)

In cases where you have direct access to a database via, for example, SQL injection or identified credentials, you should always check for any stored user sessions. See an example below.

Code: sql

```sql
show databases;
use project;
show tables;
select * from users;
```

```
Reading table information for completion of table and column names
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mydatabase         |
| mysql              |
| opendb             |
| performance_schema |
| project            |
+--------------------+
6 rows in set (0.000 sec)

MariaDB [(none)]> use project;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [project]> show tables;
```

```
+----------------+
| Tables_in_project |
+----------------+
| all_sessions   |
| files          |
| login          |
| sessions       |
| users          |
| users_data     |
+----------------+
6 rows in set (0.000 sec)

MariaDB [project]> select * from users;
+----+-----------------------------------------+-----------------+----------------------------------+
| id | username                                | email           | password                         |
+----+-----------------------------------------+-----------------+----------------------------------+
|  1 | test                                    | test@gmail.com  | 098f6bcd4621d373cade4e832627b4f6 |
|  4 | admin                                   | admin@admin.pwn | 21232f297a57a5a743894a0e4a801fc3 |
|  5 | \'OR 1=1 -- -\'                         | mysqli_test@test.com | d6a6bc0db10694a2d90e3a69648f3a03 |
|  6 | "><img src=x onerror=prompt(document.cookie)> | xsstest@gmail.com | 9cf3dec7a89bc77aed78ef06f2dba6c0 |
|  7 | test11                                  | test11@gmail.com | 098f6bcd4621d373cade4e832627b4f6 |
+----+-----------------------------------------+-----------------+----------------------------------+
5 rows in set (0.000 sec)

MariaDB [project]>
```

Here we can see the users' passwords are hashed. We could spend time trying to crack these; however, there is also a "all_sessions" table. Let us extract data from that table.

Code: sql

```sql
select * from all_sessions;
select * from all_sessions where id=3;
```

```
MariaDB [project]> select * from all_sessions;
+----+-------------------+-------------------------+
| id | name              | session                 |
+----+-------------------+-------------------------+
|  1 | John              | Johnsspecialh4x0rc00kie |
|  2 | Ben               | Benssup3rs3cretc00kie   |
|  3 | Developer(Admin)  | develop3rl33tsp00ks     |
+----+-------------------+-------------------------+
3 rows in set (0.000 sec)

MariaDB [project]> select * from all_sessions where id=3;
+----+-------------------+-------------------------+
| id | name              | session                 |
+----+-------------------+-------------------------+
|  3 | Developer(Admin)  | develop3rl33tsp00ks     |
+----+-------------------+-------------------------+
1 row in set (0.000 sec)

MariaDB [project]>
```

Here we have successfully extracted the sessions! You could now authenticate as the user "Developer."

It is about time we cover Session ID-obtaining attacks requiring user interaction. In the following sections, we will cover:

- XSS (Cross-Site Scripting) <-- With a focus on user sessions
- CSRF (Cross-Site Request Forgery)
- Open Redirects <-- With a focus on user sessions

Enable step-by-step solutions for all questions

Questions

Answer the question(s) below to complete this Section and earn cubes!

Download VPN Connection File

Target(s): Click here to spawn the target system!

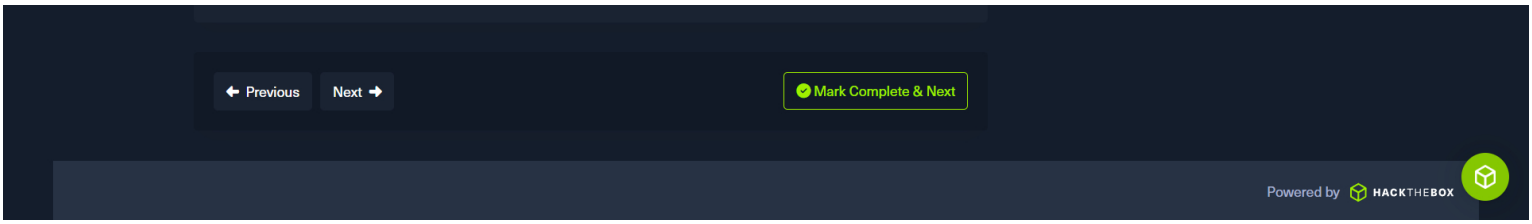
vHosts needed for these questions:

- xss.htb.net

+1 If xss.htb.net was an intranet application, would an attacker still be able to capture cookies via sniffing traffic if he/she got access to the company's VPN? Suppose that any user connected to the VPN can interact with xss.htb.net. Answer format: Yes or No

Yes

Submit


Previous
Next
Mark Complete & Next
Powered by HACKTHEBOX