Introduction to Sessions

A user session can be defined as a sequence of requests originating from the same client and the associated responses during a specific time period. Modern web applications need to maintain user sessions to keep track of information and status about each user. User sessions facilitate the assignment of access or authorization rights, localization settings, etc., while users interact with an application, pre, and post-authentication.

HTTP is a stateless communication protocol, and as such, any request-response transaction is unrelated to other transactions. This means that each request should carry all needed information for the server to act upon it appropriately, and the session state resides on the client's side only.

For the reason above, web applications utilize cookies, URL parameters, URL arguments (on GET requests), body arguments (on POST requests), and other proprietary solutions for session tracking and management purposes.

Session Identifier Security

A unique **session identifier** (Session ID) or token is the basis upon which user sessions are generated and distinguished.

We should clarify that if an attacker obtains a session identifier, this can result in session hijacking, where the attacker can essentially impersonate the victim in the web application.

An attacker can obtain a session identifier through a multitude of techniques, not all of which include actively attacking the victim. A session identifier can also be:

- Captured through passive traffic/packet sniffing
- Identified in logs
- Predicted
- Brute Forced

Now let us focus on session identifier security for a minute.

A session identifier's security level depends on its:

- `Validity Scope` (a secure session identifier should be valid for one session only)
- `Randomness` (a secure session identifier should be generated through a robust random number/string generation algorithm so that it cannot be predicted)
- `Validity Time` (a secure session identifier should expire after a certain amount of time)

The established programming technologies (PHP, JSP, etc.) generate session identifiers that "comply" with the above *validity scope*, *randomness*, and *validity time* requirements. If you come across a custom session identifier generation implementation, proceed with extreme caution and be as exhaustive as possible in your tests.

A session identifier's security level also depends on the location where it is stored:

- `URL`: If this is the case, the HTTP *Referer* header can leak a session identifier to other websites. In addition, browser history will also contain any session identifier stored in the URL.
- `HTML`: If this is the case, the session identifier can be identified in both the browser's cache memory and any intermediate proxies
- `sessionStorage`: SessionStorage is a browser storage feature introduced in HTML5. Session identifiers stored in sessionStorage can be retrieved as long as the tab or the browser is open. In other words, sessionStorage data gets cleared when the *page session* ends. Note that a page session survives over page reloads and restores.
- `localStorage`: LocalStorage is a browser storage feature introduced in HTML5. Session identifiers stored in localStorage can be retrieved as long as localStorage does not get deleted by the user. This is because data stored within localStorage will not be deleted when the browser process is terminated, with the exception of "private browsing" or "incognito" sessions where data stored within localStorage are deleted

Table of Contents

Introduction to Sessions

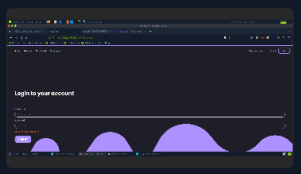
Session Attacks

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



Interact Terminate Reset

Life Left: 42m +

```
by the time the last tab is closed.
```

Session identifiers that are managed with no server interference or that do not follow the secure "characteristics" above should be reported as weak.

Session Attacks

This module will cover different types of session attacks and how to exploit them. These are:

- `Session Hijacking`: In session hijacking attacks, the attacker takes advantage of insecure session identifiers, finds a way to obtain them, and uses them to authenticate to the server and impersonate the victim.
- `Session Fixation`: Session Fixation occurs when an attacker can fixate a (valid) session identifier. As you can imagine, the attacker will then have to trick the victim into logging into the application using the aforementioned session identifier. If the victim does so, the attacker can proceed to a Session Hijacking attack (since the session identifier is already known).
- `XSS (Cross-Site Scripting)` <-- With a focus on user sessions
- `CSRF (Cross-Site Request Forgery)`: Cross-Site Request Forgery (CSRF or XSRF) is an attack that forces an end-user to execute inadvertent actions on a web application in which they are currently authenticated. This attack is usually mounted with the help of attacker-crafted web pages that the victim must visit or interact with. These web pages contain malicious requests that essentially inherit the identity and privileges of the victim to perform an undesired function on the victim's behalf.
- `Open Redirects` <-- With a focus on user sessions: An Open Redirect vulnerability occurs when an attacker can redirect a victim to an attacker-controlled site by abusing a legitimate application's redirection functionality. In such cases, all the attacker has to do is specify a website under their control in a redirection URL of a legitimate website and pass this URL to the victim. As you can imagine, this is possible when the legitimate application's redirection functionality does not perform any kind of validation regarding the websites which the redirection points to.

Module Targets

We will refer to URLs such as `http://xss.htb.net` throughout the module sections and exercises. We utilize virtual hosts (vhosts) to house the web applications to simulate a large, realistic environment with multiple webservers. Since these vhosts all map to a different directory on the same host, we have to make manual entries in our `/etc/hosts` file on the Pwnbox or local attack VM to interact with the lab. This needs to be done for any examples that show scans or screenshots using an FQDN.

To do this quickly, we could run the following (be reminded that the password for your user can be found inside the `my_credentials.txt` file, which is placed on the Pwnbox's Desktop):

Introduction to Sessions

```
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
```

After this command, our `/etc/hosts` file would look like the following (on a newly spawned Pwnbox):

Introduction to Sessions

```
MisaelMacias@htb[/htb]$ cat /etc/hosts

# Your system has configured 'manage_etc_hosts' as True.
# As a result, if you wish for changes to this file to persist
# then you will need to either
# a.) make changes to the master file in /etc/cloud/templates/hosts.debian.tmpl
# b.) change or remove the value of 'manage_etc_hosts' in
#     /etc/cloud/cloud.cfg or cloud-config from user-data
#
127.0.1.1 htb-9zftpkslke.htb-cloud.com htb-9zftpkslke
127.0.0.1 localhost

# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts

<TARGET IP> xss.htb.net csrf.htb.net oredirect.htb.net minilab.htb.net
```

You may wish to write your own script or edit the hosts file by hand, which is fine.

If you spawn a target during a section and cannot access it directly via the IP be sure to check your hosts file and update any entries!

Module exercises that require vhosts will display a list that you can use to edit your hosts file after spawning the target VM at the bottom of the respective section.

Let's now dive into each of the previously mentioned session attacks and vulnerabilities in detail.

Next ➜ Mark Complete & Next