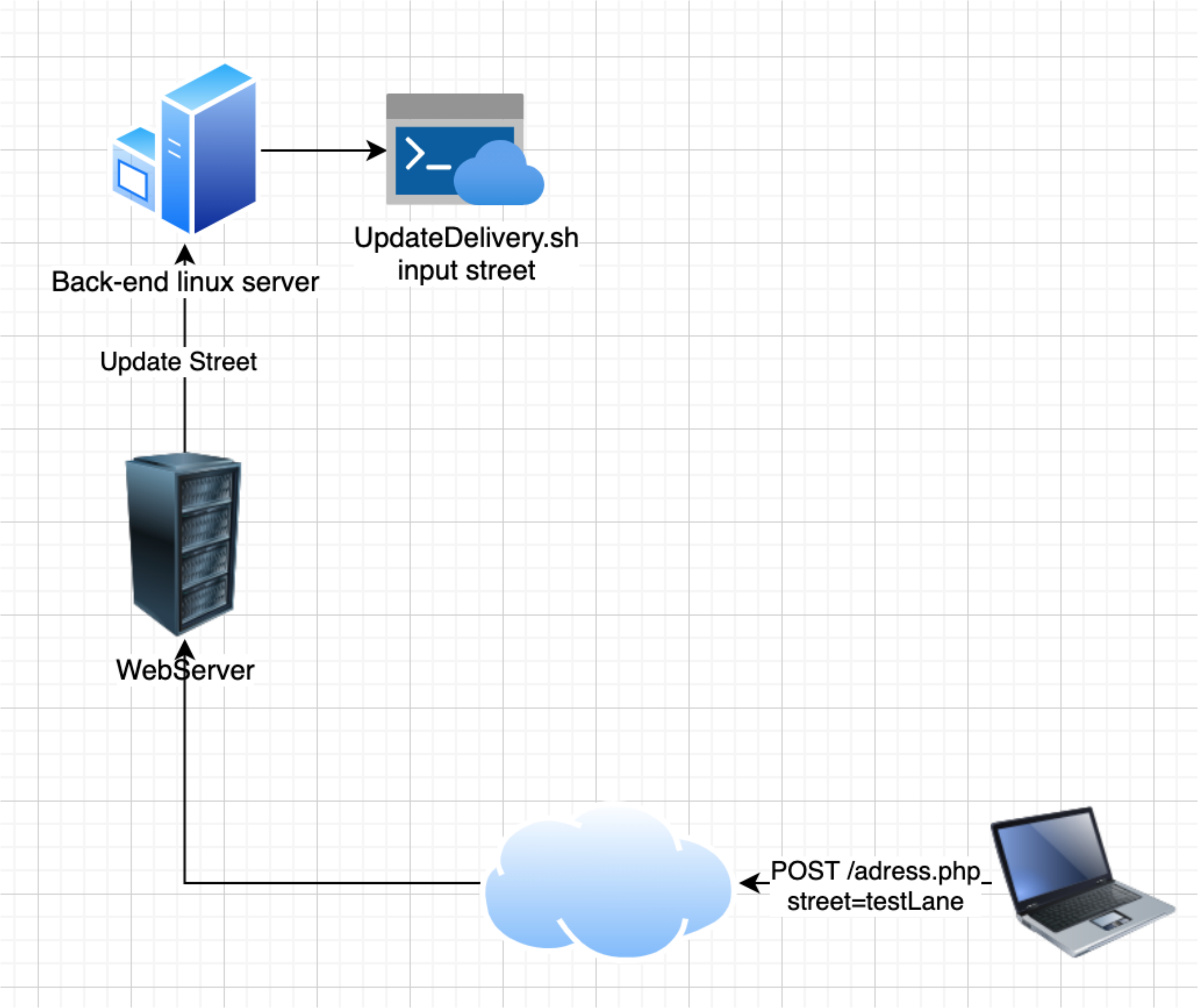
Command injection

What is it?

Command injection happens when we can control a parameter that gets passed into a shell. If that input is not handled safely and sanitised properly, we can insert a command into our input and have that executed by the shell. Depending on the capabilities and privilidges of that shell, we can execute various commands.

The theory sounds very simple however it's not simple at all to find this kind of vulnerability.



Attack strategy

The reason command injection is so hard to find is because we never really know which of our processes will trigger a back-end shell to execute. This means we will need to fuzz every parameter we find but you might be wondering what characters to fuzz with. To determine this, we first need to talk about which command separators can possibly be used and also which commands.

Separators

The following command separators work on both Windows and Unix-based systems:

- `&`

- `&&`
- `|`
- `||`

The following command separators work only on Unix-based systems:

- `;`
- Newline (`0x0a` or `\n`)

On Unix-based systems, you can also use backticks or the dollar character to perform inline execution of an injected command within the original command:

- `` ` `` injected command `` ` ``
- `$(` injected command `)`

Commands

Below is a summary of some commands that are useful on Linux and Windows platforms:

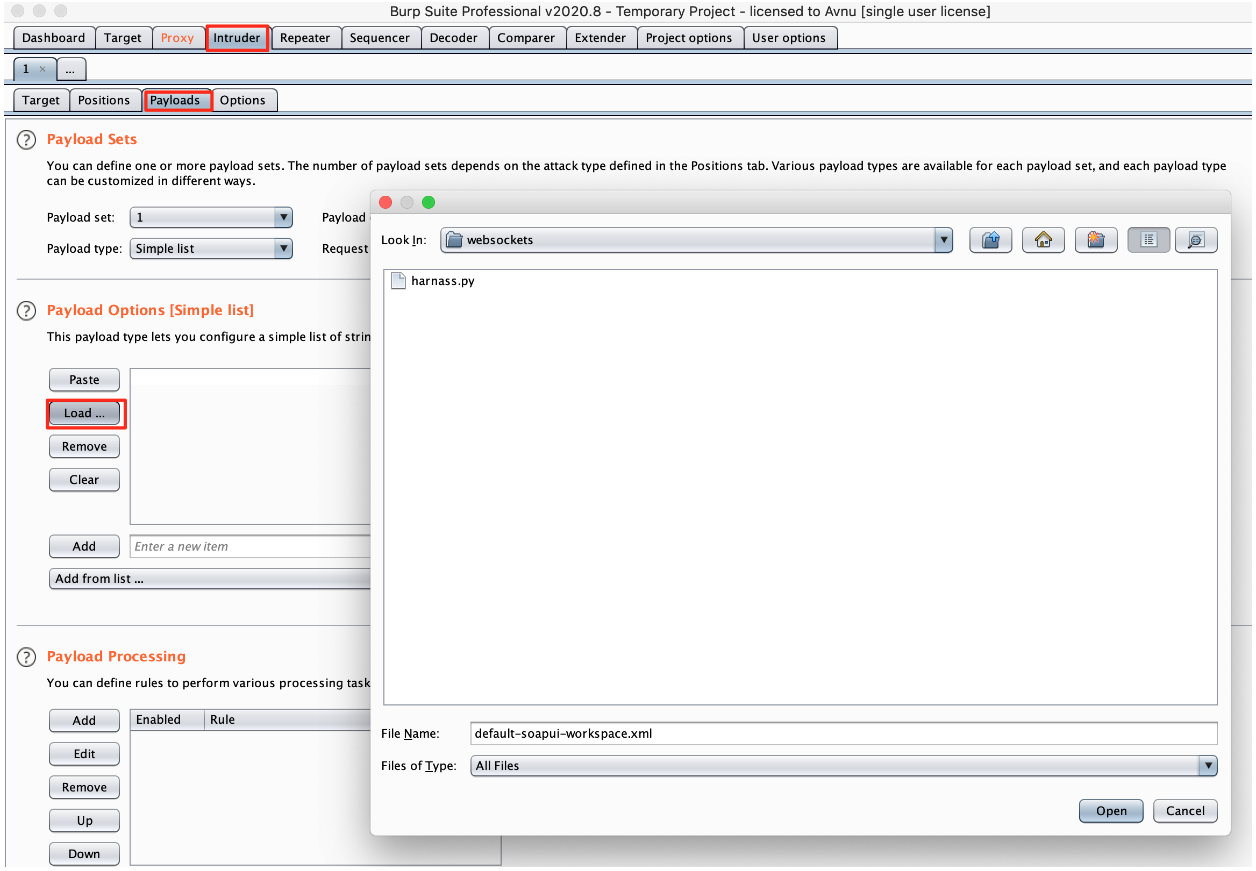
Linux and windows commands

Purpose of command	Linux	Windows
Name of current user	`whoami`	`whoami`
Operating system	`uname -a`	`ver`
Network configuration	`ifconfig`	`ipconfig /all`
Network connections	`netstat -an`	`netstat -an`
Running processes	`ps -ef`	`tasklist`

fuzzing list

Based on this, we can create a fuzzing list that contains all the seperators togheter with all the possible commands. I will leave this up to you as it should be a good exercise. Feel free to contact me on discord if you have issues with this however.

Fuzz every single parameter you can find with this worldlist by using burp intruder.



Burp Suite Professional v2020.8 - Temporary Project - licensed to Avnu [single user license]

Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1

Payload type: Simple list

Payload Options [Simple list]

This payload type lets you configure a simple list of strin

Payload Processing

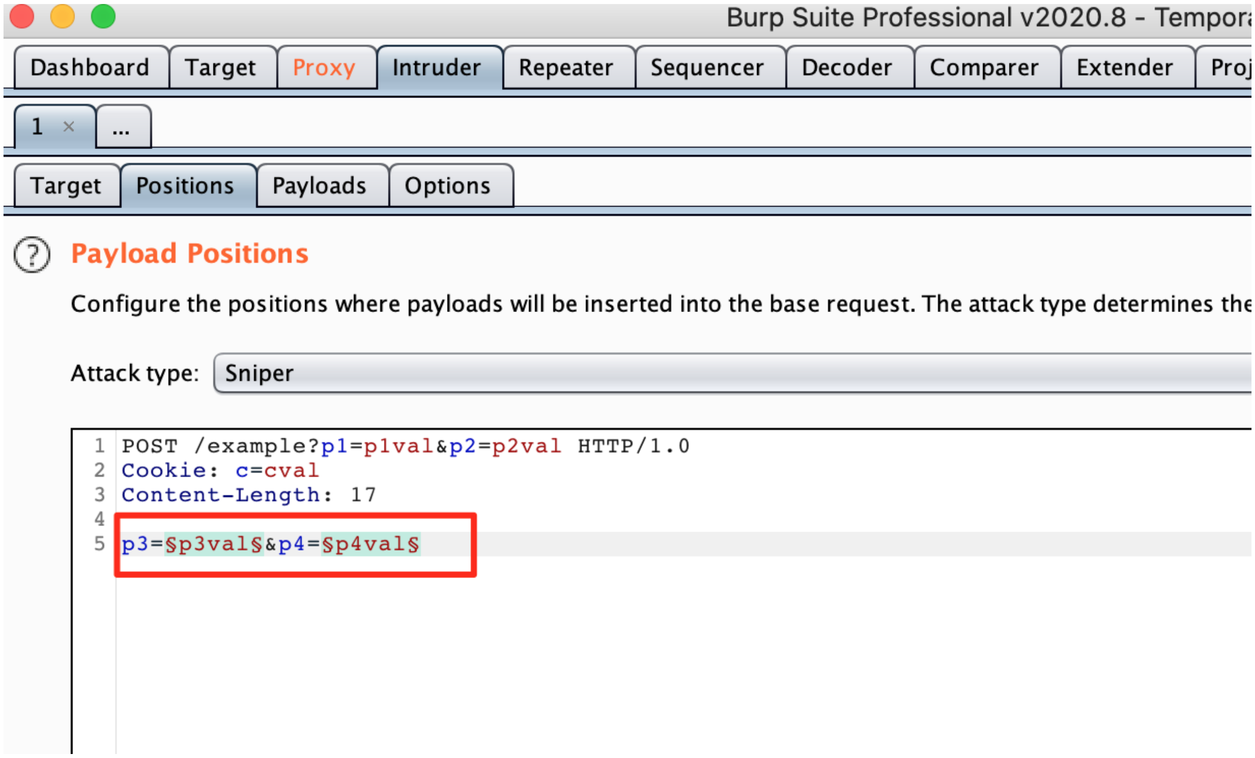
You can define rules to perform various processing task

Look In: websockets

harnass.py

File Name: default-soapui-workspace.xml

Files of Type: All Files



Configure the positions where payloads will be inserted into the base request. The attack type determines the

Attack type: Sniper

```
POST /example?p1=p1val&p2=p2val HTTP/1.0
Cookie: c=cval
Content-Length: 17

p3=§p3val§&p4=§p4val§
```

As you can see in the screenshot, we load in our fuzzing list and mark the parameters we want to test.

Blind command injection

We can test for blind command injection by launching a request that will execute a ping command to the loopback adress.

```
& ping -c 10 127.0.0.1 &
```

Again, add this to your fuzzing list and mind the response time for this attack vector. If it's longer than 10 seconds, we probably have a blind command injection but be mindful as lag might occur and give a false positive.