SkyPenguin Labs | Tackling The Grounds
Breaking Into IoT Devices

Welcome to the first ever research paper published by SkyPenguinLabs! This research paper is a whole dedicated paper for IoT devices specifically aimed at AppleTV, Google Chrome cast, Google Nest, Roku TV, Amazon FireStick and other various devices. The point of this research document is to proof the SkyLine programming language and its capabilities when operating with standard research as well as what can be used for general IoT research. SkyPenguinLabs has published this document in hopes to change the perspective of IoT devices and give people some new profound technologies and ideas for the world around us! Note that there will be a section in here which talks about ChatGPT for security research and even go further into AI/ML purposes. However, we plan for that to be at the very end of the document and will ensure that users try to at least understand the core fundamentals of IoT security research before hand.

This document will also try to focus on the world of web with IoT devices. For example, you will see a ton of dissection around APIs within these little devices and will even go over some basic functionalities, scanning and fuzzing with specific tools. It may be worth my time to also note that the contributors to this research can range from 1 single person (the primary author: Totally_Not_A_Haxxer) or a entire contributing team. The reason I did this is because I felt that again, maybe we can bring some new light to the world of IoT devices- hell, we might even explore the world of vehicles if we can- but that is just an idea. As mentioned, the primary focus will be listed.

I guess this should be my time to actually go ahead and start the research paper. Below is going to be a list of the requirements that we will be using to conduct this research by brand name and what we will be using to conduct basic research.

SkyPenguin Labs | Tackling The Grounds
0x001 Equipment & Requirements

This research paper is fucking massive, so, I suggest that we actually go through the items we will be using for research based on devices. If the equipment is not used universally and is only used for say discovering and understanding a server or protocol, then when the analysis comes, we can go from there and actually introduce that piece of software. That being said, lets go ahead and list a few sections off the bat. Below are the sub-sections for each brand.

Section 0x001 (Setup): Equipment & Requirements | Google

For this portion of the research paper which will actually be our first round of exploration, we will be using some standard frameworks going into this.

- P1: THE DEVICE OF CHOICE

The device that we actually decided to choose to research was going to be a standard Google cast specifically the Google Chromecast 4K HD which will be the most standard Google chromecast device that can be used.

Additional research argues that the smaller version of the Google chromecast (the USB like device) is more used: but we are going to be choosing to go with this device because this may just be the more "larger" one to explore.

- P2: LAYING OUT FRAMEWORKS FOR DISCOVERY

The next thing that we need to list out is going to be the frameworks we are going to use for general recon. Given this research paper is just here to explore all of the devices and whats inside of them more than we are going for general exploitation, I would like to pinpoint some of the primary frameworks we will be using for exploring and understanding the different parts of the cast. Some of these tools, titled with tags like [U] stand for frameworks or tools that will be used directly across all devices. Note that this research paper will not go over installing any of these tools as it differs per operating system and also what you already do and do not have installed on your system. Please refer to the original documentation of every tool listed here for more information.

Below you will see some bullet points that point all of this out firmly.

A. **Wireshark** [U]- Any security researcher working with IoT may obviously go to choose Wireshark. The reason we will be using Wireshark for general discovery on actually all devices is because most of these IoT devices are going to be operating using protocols like BLE, SSDP, UPnP, DAAP, UDP, AirPlay and various other protocols. Some of these protocols like mDNS will need to be heavily analyzed and later on will require custom programs to parse the protocols. For now, Wireshark will actually be our best friend here for general discovery and understanding of the network layer within the device. As the research here poses, we can assume that a Google Chromecast will be using SSDP ALOT based on the devices features and functionalities.

B. **NMAP** [U]- As you can also imagine, we will be using nmap for service discovery, script scanning and more. Basically just want to go over the services and try to fingerprint anything interesting.

C. **PostMan** [U] - Postman is something we are going to be using for further analysis and discovery later on in the idea of research if its needed at all. This is just going to be for the API side of things if we actually ever need to go that far into the analysis.

D. **BurpSuite** [U] - Burp is also going to be something we will be using to replay back some really complex requests that may require anything additional. During the base research that was already conducted on these devices before this document even existed, burp was not needed but honestly is suggested to use. Replaying specific requests can be a serious pain if its not done correctly or there is no script to repeat it. So, if you are following along, suggestion that you use burp.

E. **DevTools** [U] - Along with Postman, we will be using dev tools to see if the resources or APIs are having any other connections to be used. I do not think dev tools is really going to be used in any part of discovery, but assuming that other frameworks and tools do not work- this could be our last resort here.

- P3: LAYING OUT FRAMEWORKS FOR PHYSICAL ANALYSIS

This section is going to remain empty for now since physical layers were not the purpose of this research. While the physical research is important, there were some legal limits to attacking proprietary software that the original contributors or authors of this research paper did not have. Of course, we should not necessarily be saying - legality should stop a researcher, but we would like to respect the companies and brands being researched and label this paper as a ethical security research document.

Physical analysis tools could be super useful and should be gone over in the future.

Section 0x001 (Setup): Equipment & Requirements | Roku

- P1: THE DEVICE OF CHOICE

The device chosen for the Roku section of things was the Roku Premiere | HD/4K/HDR Streaming Media Player. We chose this device because of how small, portable and unique the device is.

- P2: LAYING OUT FRAMEWORKS FOR DISCOVERY

The frameworks to explore Roku were primarily the same as the ones that are generally universal to this. This is because Roku on the web side of things is not super complex and is not one of the devices that likes to hide everything and is actually much more open than one might think. That being said, we can go ahead and move onto this section. For those wondering, I would also like to state that the tools used for Roku are staying limited for a reason- most of the implementations used for further research are all proprietary due to one primary factor. Keeping it safe.

The SkyLine programming language which was a huge support beam for tools that powered and proofed this research document was announced closed on September 13th 2023. The reason being is because of the algorithms, signatures, patterns, scanners and other sets of code that are able to bypass security systems and general detection utilities for malicious content. This is the best quality about the language which means that if the developers have decided to close it down, that the backend of the language as well as payloads will remain unknown.

Section 0x001 (Setup): Equipment & Requirements | Apple

- P1: THE DEVICE OF CHOICE

Apple was a bit annoying, I felt like the research for IoT was going to be limited since iPhones were already too researched and that AI smart home devices such as the Amazon Alexa were going to be off the limit. Instead, I decided to pick up the first generation of AppleTVs.

Why? This device was chosen for a specific reason, its protocols, implementations, services and more. Apple is well known to be one of the most proprietary brands out there, anything from their hardware down to their software and even languages (similar to Google) is all their own. Everything is just made by Apple even protocols have their own implementations.

It was to my assumption that we would need to use custom utilities, frameworks, protocols, cracked software and leaked source code to our advantage here and actually understand how the device works from the inside out. This will also add more " how to " and "real world" to this document. Not to mention we also have to explain and understand proprietary file formats and even unique file formats like the BPLIST and PLIST formats for data storage. The initial research did not gather much, but I did gather a good amount of information about the device.

- P2: LAYING OUT FRAMEWORKS FOR DISCOVERY

It was firmly understood and expected that working with any form of Apple device was going to be an absolute pain in the ass. So, we realized that we may need much more complex utilities and also need a much more complex understanding of IoT devices and their physical layers as we go on. However, we also need to cover the network layer and digital aspect of the device as well.

Most of the frameworks and other tools used such as marshaling were built using the SkyLine or Go programming language. This included building parsers and junk generators for PLIST or BPLIST files after figuring out some of the files were damaged when they were generated and sent from the server. We found this out by trying to use Apples custom SDKs for parsing those files and got absolutely no results- the same goes for plistutils, go-plist, py-plist and other various utils in other languages. Of course, they worked when you found demo BPLIST and PLIST files online, but the ones the servers on the AppleTV sent would seem to not be parsed - but anyway, we will get into that later.

1. **Avahi-Utils** - We will be using the Avahi for mDNS/DNS-SD parsing and general discovery. This is because it was found that AppleTV seems to be using mDNS/DNS-SD much more than various other IoT devices and after further research realizing that AppleTV seems (along with various other Apple devices) to use mDNS so much more and rely on it much more.

2. **Strings, xxd, hxd, head, tail, etc.** [U]- This should have been an obvious one. But we will be needing programs like string, head, xxd, hd, and others. This is because we need to analyze some of the files that were slapped as responses from the server. SOOOOO- I went ahead and decided to include other programs like this within this

singular point. We are also going to need to see what the servers will do when responding to specific events in binary modes.

Section 0x001 (Setup): Equipment & Requirements | Amazon

- P1: THE DEVICE OF CHOICE

Similar to the Apple devices, choosing a device to research for Amazon was also a bit of a pain for multiple reasons. It was discovered that in the research, amazon has a ton of devices but only a few smart home devices that interface with say TVs, unlike Google. That being said, I decided to choose a device that would actually make sense, something that was small and light but also served a general purpose that made it easy to travel and interface with directly and even interact with. I also chose this device for its amazing and widespread use of the UPnP side of things.

That device you ask? `Amazon Fire TV Stick with Alexa Voice Remote (includes TV controls), free & live TV without cable or satellite, HD streaming device`

This device made sense just for how small it was yet easy to setup and work with. I felt that this also had its own unique side of things for research.

- P2: LAYING OUT FRAMEWORKS FOR DISCOVERY

Similar to Apple and other brands, we decided to keep the frameworks for discovery low on this one. We decided to also include other enumeration tools that we have built on our own which used the SkyLine programming language and the Go programming language, but we will get into that when we finish the research.

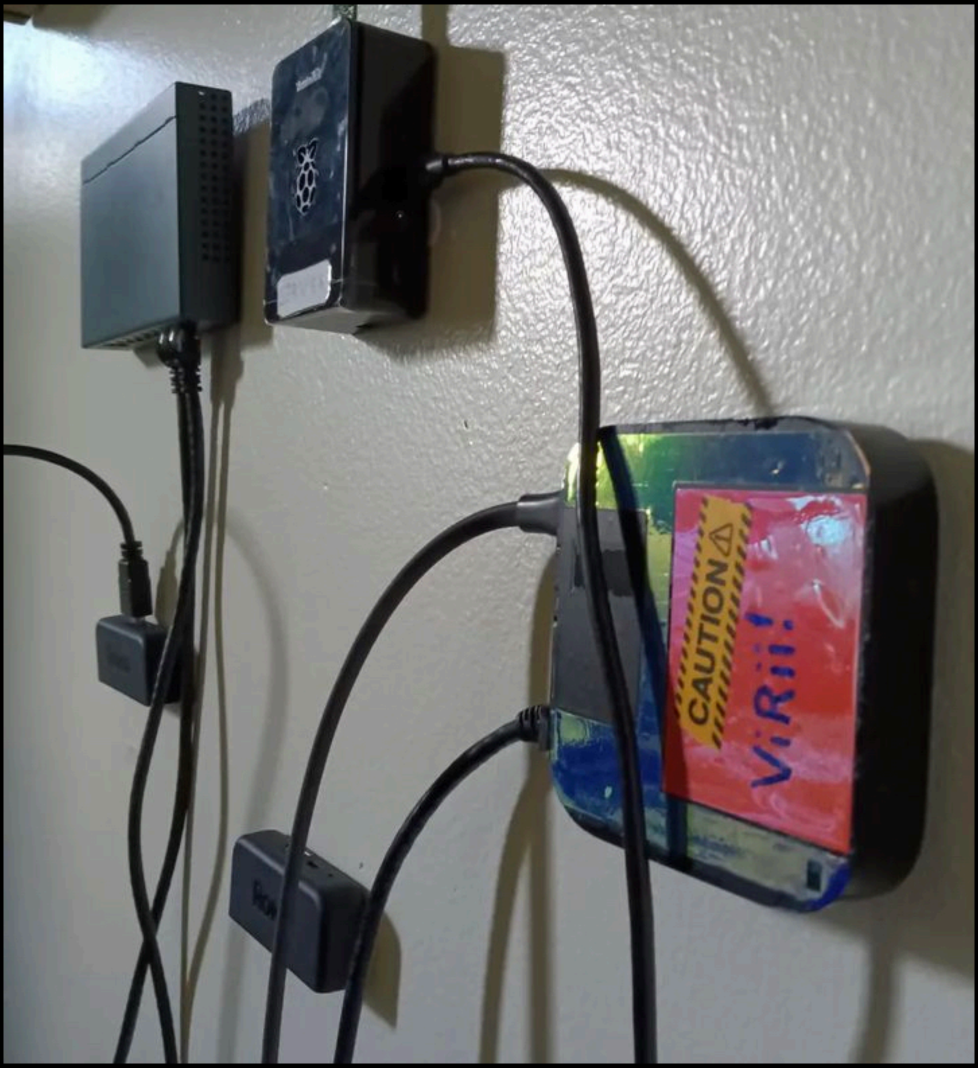
SkyPenguin Labs | Tackling The Grounds

0x002 Environment Setup

In order to ensure that testing takes place in a well connected environment (physical environment) I have decided to start connecting devices and started to also create something known as an IoT wall. This is basically just multiple devices connected to the same network and plug setup on the same side of my rooms wall. Below is a list of devices that will be used for the testing and device purposes. Pretty much a summary.

A. Raspberry PI 4B - This will actually be acting as a server, any data or findings will be held and remotely grabbed from this server. The reason we decided to have this as included is because my home PC has too much garbage on it and the RPI is fit with custom software that utilizes the SkyLine programming language and another proprietary language known as Radical Processing Core for CPU heavy tasks such as making large requests or formatting large outputs and translating specific sets of data. The reason data translation may need to be fit is to be translated for specific frameworks or tools such as NMAP. Fitting for this scan may be important for future research.

B. Google Chromecast: The Google Chromecast will also be on the wall being indicated and attached to nothing else but a simple plug. Sadly, the Cast itself does not support



ethernet like other devices such as the AppleTV.

C.AppleTV: The AppleTV will be connected to a plug with its power switch and will have a simple ethernet cable attached to the switch which was also used.

D.TP-Link TL-SG108 8 Port Gigabit Unmanaged Ethernet Network Switch: This switch is just used as am extender for any ethernet connections. This way the server can also have a much more stable connection as well as other devices around it and that way we do not need to use a bunch of long cables from the router.

E.FireStick: The firestick , similar to the Chromecast will not be connected to the TV and will just be wired on. I have thought about not attaching this device similar to the chromecast to the wall because I would like to unstick it if I need to stick it onto a screen for more visual payloads such as ones that may turn the TV off, kill applications and more.

F. Roku TV(s): We are using two different TVs because of the further research that was done on these devices and also wanting to trace the affects of attacking two different devices at once also hearing that Roku may be implementing extra security measures. I did the same with Amazon FireStick and the Google Chromecast where I also got ahold of two other Firesticks.

All of these were hooked up together and you may not see some of the other devices- but an image below shows what the general IoT wall setup looked like. Note that again, this environment was supposed to be pretty clean and generally setup. For any other device that was added to be apart of the research was not included in this wall but was included separately in another environment.

This is the image of the devices that have been strapped to the wall. You may also notice that only two devices are connected to ethernet which actually traces back to the previous statement. It may also again be worth noting that devices such as the Google chromecast and Firestick were not physically connected to the TVs or screens yet as we are just trying to make sure that the device connected to the network and actually worked properly. After initial setup, I confirmed that all devices did not have developer mode on as the point in this research is to try and gain access to device information from endpoints that do not require developer mode on. I would also like to mention that maybe it is plausible to turn developer mode on to see if you can leverage or discover any other endpoints that may exist within the device which in turn can be used to further manipulate the device and try to gain more information. For now, this is all we need to be setup as everything else such as Wireshark, Burp, Postman, Avahi, and other various utils were also used.

Before we get further into it, its again worth noting that this research document is just that, it is a research document. This document was not written to harm the reputation of any one brand or company and was not written to harm any users. All devices were obtained ethically and were donated or received with the free will from the donatee where they were fully understanding of what I would be doing to these devices or what the point in needing them was. Speaking of, thank you to the people that donated these devices and allowed me to use them freely, this means quite a ton!

That being said, let us go ahead and move onto a whole other section of this document and talk about the primary reason as to using proprietary tools and the bigger reason as to why this document exists.

SkyPenguin Labs | Tackling The Grounds

0x003 Documental Reasoning

I would like to go a bit more into this document and its purpose. This document was originally created by Totally_Not_A_Haxxer for general research. This was done for multiple reasons, originally, a research paper was published on Hakin9 about security research for AppleTVs and dissecting their APIs. The author of that paper (Totally_Not_A_Haxxer) felt quite weird when that paper was also limited, stripped a bit, contained and also locked down by the company.

In turn, the author decided to start this paper: A paper that was not only accessible to everyone and not just someone who needed to sign up on the platform but also something that was very much more technical, less limited, could be more flexible and less "walk through-y" if that makes sense. So, that being said, this document is basically the full version of the one slapped down to Hakin9. Most of the issues I as the author had with the document being submitted was the fact that I felt I had to walk users through everything and that was kind of my bad, not to mention, I felt that the document was very un-informal and could have been done better. Regardless, I am happy Hakin9 published that, but its time for something new and much more unique.

Double Publishing Warning: The paper submitted to Hakin9 was submitted under the name Totally_Not_A_Haxxer (the same owner of this paper). Its important to understand that this paper was released as a community effort and did not take from the paper from Hakin9. Not to mention, Hakin9 and employees at Hakin9 did not make the writer aware of any rules and regulations for submitting research and was also not made aware of any ToS as far as double publishing and re-submitting work on other platforms. Please do understand that this paper is completely separate and if anything most likely existed and was in development before the previous document submitted to Hakin9.

That being said, we can move onto the actual purpose of this document. This document was designed to make research much more open and give people a direct guide to real world experience with IoT laws and giving you a good example of what security research can look like as a team effort. This paper was also submitted as a way to help the SkyPenguin-Labs and associated communities grow by being able to work together to create proper efforts! Outside of that, there was also a private reason as to why this

document existed and acting here. The next section on the next page will actually go deeper into testing and experimenting.

SkyPenguin Labs | Tackling The Grounds

0x003 Documental Reasoning (P2)

This one is going to be a bit weird since no person at this time is aware the SkyLine programming language exists. Well, there are some people. But it is not a language that has been promoted, it just exists- that is it. Anyway, this document is a good proof for the language for two prime reasons which are listed below.

1. **The Caster Framework**: The Caster framework is the framework that automates close to 100% of the research made within this document. Anything from firmware analysis to the whole idea of picking apart binary files from the AirPlay server in AppleTVs to remotely controlling RokuTVs by abusing the ECP protocol which you will learn about more in this document. The original framework was written in the Go programming language, until SkyLine was ready to take on new limits. The Caster framework is one of the very first technologies written using SkyLine and throughout this document you will notice that scripts written in the language are used to explore and make requests to specific APIs and also conduct some simple research about the documents.

2. **Topic Specific Research:** SkyLine as a programming language was created for one reason, to improve the general world of cyber security. This can be anything from binary exploitation to very basic web recon and data formatting and translation. Hell, there is even a proprietary game hacking library being built for specific (types) emphasis on the "types" of games < not naming specific games >. This is also one of the more deeper aspects. Our world already has languages that can operate with cyber security related tasks. You can open up IRB (Ruby console) and start immediately hacking. You can download the Python library "scapy" and get down to network hacking. We understand firmly that there is over 20 years worth of development and efforts put into libraries for other languages and frameworks like metasploit written using languages like Ruby. However, SkyLine aims to be better and more unique by compiling all of the efforts put into other languages while sticking to its base. This research document is a very good example of what SkyLine is absolutely horrible at and what SkyLine does best at.

That being said, this document will not only be super helpful in the future but will also exist in hopes to inspire others to be more technical about research. There is a literal reason this document is so massive- because real research does not just cover a small section, it covers everything it can.

SkyPenguin Labs | Tackling The Grounds

0x003 Documental Reasoning (P3)

The next thing is actually quite sad that I have to go over this but felt like it was worth bringing into the document. It seems as if people in the cyber security space are starting to use ChatGPT along with other text based ML technologies to automate their security research. This document at the very end of the research will use GPT to automate some of the discovery and understanding of specific data formats going against general resources. The reason this sparked my interest is because an amazing friend of mine actually is doing a presentation about using ChatGPT for security reasons and trying to find ways as to why using GPT does generally suck for security research and should never be used. This document aims to proof out a few reasons and the main one is floating around bias materiel.

During numerous testing phases and also technical writing, I decided to test out GPT for security research trying to understand the hype. So, I got a small little device from around my home, something that was IoT and started to use GPT for various reasons. For example, I got a list of resources revolving around a file format that was proprietary and was actually reverse engineered by a few researchers. I then started the reverse engineering process myself using GPT for analysis of the data that was being found and found that BIAS was the biggest issue.

Moreover, I also realized that when taking approximately 30 different research articles that were written in August-September of 2023 about security research in various fields that they all had some form of bias. For example, when talking about buffer overflows in a binary exploitation article: it was noted by the "author" that encrypting strings is impossible to find during runtime and decompiling a binary was not possible- this made absolutely no sense considering that frameworks such as IDA and even Ghidra will decompile binary files. I mean there are technical definitions everywhere as well saying - "A decompiler, like a disassembler, works via reverse engineering. The decompiler translates a compiled code or an executable file into high-level code."

The reason this document aims to prove how horrible GPT and various other models can be for the world of cyber security is because of the misleading of beginners. If a beginner uses something like ChatGPT, despite the warning from OpenAI saying - " this model may occasionally output bias content ", they will still believe what they are reading is real.

There have been numerous demonstrations of the models capabilities but from what I am seeing and as this document will also try to prove, GPT is most of the times bias. Even when taking peoples articles, as a technical and general writer, I understand that humans leave patterns and also leave some form of style that is unique- following that style is easy

to detect when its static across 90 other authors. We should put "author" in air quotes because I do not think that any real author generates content. Then again, that is bias and this is a research paper- so please do not take anything in this section literal.

SkyPenguin Labs | Tackling The Grounds

0x004 A brief Docu Technical Intro

Now we are going to start the technical research soon, so I thought it was important to give you a brief overview of how this document is going to work out and what we are going to do.

I think the first most important thing to knock out of the way is that this document is not a guide, it is not beginner friendly. If you do not understand the standard TCP/IP and OSI models of networking and do not understand how protocols such as UDP and SSDP work, then I highly suggest you turn back and go further into those before talking about this document.

This document will try to break down other protocols like the mDNS and AirPlay protocols but will try not to go deep into it. Not to mention, it is important that we also look into the world of implementations of protocols such as the Zeroconf protocol that Apple supposedly customized and made their own.

This document will also not demonstrate tools, we will not be going deep into the details of how to use NMAP, Wireshark, Postman, Burp suite, ZAP and will not go deep into tools like strings, xxd, hxd, etc. The aim of this document is to stay within the zone of research where the only tools being talked about are going to be tools written in the SkyLIne programming language and that is about it. This can include anything from hex dumpers to string view tools as this research is powered by scripts written in the SkyLine programming language.

At the end of this document we will talk about the framework that automates all of this research, Caster. Caster is written now in the SkyLine programming language that also relies on an external module in Go which uses the GoPacket library to parse and send out packets such as ARP requests, SSDP requests and responses, as well as capture and parse UDP and DAAP messages over the network.

This document will aim to separate each section by brand name and will also have its own ranges of subsections exploring different web systems, protocols and more! Note that this document will also be showing some resources at the end and talk about some basic OSINT for devices.

Note: Extra research might be added to this document. For example, if a user donates a flight control module for a drone, or submits a request to research drones, that research will be shown here and also written here in a separate section starting with - " [EXT] ".

SkyPenguin Labs | Tackling The Grounds

0x005 Exploring Google Devices

Diving deep into this research paper, I would like to actually talk about the Google Chrome cast and various other devices that may belong to Google. So, we are actually going to go ahead and separate this into sections in the case that maybe more Google devices are researched in the future.

0x005 Exploring Google Devices - Section 1

Section 1 is the official start of this research paper. In this section, we will be exploring the Google chromecast and the standard one that was mentioned in the environmental setups. So, we should most likely get right into it. This section is going to be utilizing NMAP for further recon.

The first thing we should do is actually go ahead and check along the side of its network and run a few nmap scans to check what services are running on this unique device.

The reason I suggest using nmap before trying to start any functionality to the device is typically scanning that network might give us a good idea of where to look first and maybe what else there is to explore. The following command below was used to scan the network of the Cast.

```
sudo nmap -v -A -p- -sC -d 10.0.0.8
```

To dissect the command, I will be going through this a bit more explaining why I decided to use this command and its flags. Note that the target address for this cast is `10.0.0.8`.

A. Flag (-v): If you are not new to nmap, this wonderful flag is to make the program output much more verbose. The reason I chose this for this scan was to give me more information on what is going on behind the scenes and if there is an error then we can actually go ahead and either tune script settings or use another program to solve this issue.

B. Flag (-A): This option is needed because it will give us a ton more information on the systems and services being run as it enables OS detection, version detection, script scanning, and traceroute. Running this option gives us more information about the device especially if there is a signature.

C. Flag (-p-): Basically just telling nmap to scan all possible ports. IoT devices like the Google cast, firestick and AppleTV are known to host their services and servers on ports larger than the standard 1024 port scan range. So we try to enumerate and scan for every port possible.

D. Flag (-sC): Wanting to run basic script scanning for NSE and load basic scripts on default scan. I decided to give this a shot in the dark because if NSE actually comes up with something in the scan then it can be pretty helpful for further exploration and understanding of the device.

E. Flag (-d): Enabling debug will help us understand a bit more about the program and add debug output from the scan. This can again help us with more information for tuning the scan if something goes wrong in the same way that verbose may.

After the scan is finished, we are set with an insane amount of output from the program and so much that it takes more than 20 mouse scrolls to even get to the port section. So, to go ahead and save room- I will go ahead and just slap the details right in here to the PDF and we can go over the ports after in summary.

NMAP SCAN DATA BREAK [START OF SCAN FILE OUTPUT]

```
Not shown: 65525 closed tcp ports (reset)
PORT      STATE SERVICE         REASON         VERSION
6466/tcp  open  ssl/unknown     syn-ack ttl 64
| ssl-cert: Subject: commonName=atvremote/38:86:F7:37:36:67/
dnQualifier=boreal/boreal/Chromecast HD
| Issuer: commonName=atvremote/38:86:F7:37:36:67/dnQualifier=boreal/
boreal/Chromecast HD
| Public Key type: rsa
| Public Key bits: 2048
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2023-03-09T05:00:00
| Not valid after:  2038-01-19T03:14:07
| MD5:   af2b 24e9 45e6 277b 2020 0a8b b75a 0f24
| SHA-1: b07b 30d4 ab9f b40c fff5 1cf6 c43a 747a a98e 6d21
| -----BEGIN CERTIFICATE-----
| MIIDKjCCAhKgAwIBAgIGAYbF2678MA0GCSqGSIb3DQEBCwUAMEwxJDAiBgNVBC4T
| G2JvcmVhbC9ib3JlYWwvQ2hyb21lY2FzdCBIRDEkMCIGA1UEAxMbYXR2cmVtb3Rl
| LzM4Ojg2OkY3OjM3OjM2OjY3MB4XDTIzMDMwOTA1MDAwMFoXDTM4MDExOTAzMTQw
| N1owTDEkMCIGA1UELhMbYm9yZWFsL2JvcmVhbC9DaHJvbWVjYXN0IEhEMSQwIgYD
| VQQDExthdHZyZW1vdGUvMzg6ODY6Rjc6Mzc6MzY6NjcwggEiMA0GCSqGSIb3DQEB
| AQUAA4IBDwAwggEKAoIBAQCKoR7/SjkHwA6CuVDivaX1DyyP5a4DPYOkZ0mtGyCi
| EAdHf38gsaFFgs9GoBU7uj1OygMeTrOe0EcPtsmSvcHyfL5jzj4GZ5xcuMn1rO1n
| D95gEpjRtlbQJvuChkfDWRxwnTL4ZkJLPtJLSOUtyEpiQzmAfa4ryXaNJhA3F+eO
| UA5XCtHgM6dJMa2khRH1lETf2N84E67FmIDlysQx/eGKUXluUk3A5oy24Mq6G41L
| Ggxr1vvoMfGY0LXSsuM+Tr6hwfsBmAVf0nq2+zitKjuWceqGwZO7TeXX1RJGBYH3
| mC4jxipRbUq1Z8MMt7zHo1RsSRFSENiWgpojNvRR7OR3AgMBAAGjEjAQMA4GA1Ud
| DwEB/wQEAwIHgDANBgkqhkiG9w0BAQsFAAOCAQEAVjXXsk7cdVAjFlv1wMEc2gSU
| iaRVfbQUZQURUg1FUsaVPHF5ZslknF4bsGXOyDUVWXEdHUhAkKM6PIFEsE6sMf8e
| biD2qJwF3230UfCpBGtRKd0URaepbzFpWDbYUr+f3PkhlhIQNzqNH5WPACiApwTF
| 7g/wmhWT7Z8ImSv3CBopEQV0PlIew7F/jpyHYVS/L18nmo5OSkWGiAT/pUQYExLl
| fl82b23EeVldeXPv023HlyeA+j84Ks8wGcPld/s6lbqFNDKAG5VE6bxnjJ70oWx0
| 1qFOCNSQjZjtuu14TZ3z5+5exMT3KdPpYuDdUZ5Prb06g5L8gGxAwCvVSYUcpQ==
|_-----END CERTIFICATE-----
|_ssl-date: TLS randomness does not represent time
6467/tcp  open  ssl/unknown      syn-ack ttl 64
| ssl-cert: Subject: commonName=atvremote/38:86:F7:37:36:67/
dnQualifier=boreal/boreal/Chromecast HD
| Issuer: commonName=atvremote/38:86:F7:37:36:67/dnQualifier=boreal/
boreal/Chromecast HD
| Public Key type: rsa
| Public Key bits: 2048
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2023-03-09T05:00:00
| Not valid after:  2038-01-19T03:14:07
| MD5:   af2b 24e9 45e6 277b 2020 0a8b b75a 0f24
| SHA-1: b07b 30d4 ab9f b40c fff5 1cf6 c43a 747a a98e 6d21
| -----BEGIN CERTIFICATE-----
| MIIDKjCCAhKgAwIBAgIGAYbF2678MA0GCSqGSIb3DQEBCwUAMEwxJDAiBgNVBC4T
| G2JvcmVhbC9ib3JlYWwvQ2hyb21lY2FzdCBIRDEkMCIGA1UEAxMbYXR2cmVtb3Rl
| LzM4Ojg2OkY3OjM3OjM2OjY3MB4XDTIzMDMwOTA1MDAwMFoXDTM4MDExOTAzMTQw
| N1owTDEkMCIGA1UELhMbYm9yZWFsL2JvcmVhbC9DaHJvbWVjYXN0IEhEMSQwIgYD
| VQQDExthdHZyZW1vdGUvMzg6ODY6Rjc6Mzc6MzY6NjcwggEiMA0GCSqGSIb3DQEB
| AQUAA4IBDwAwggEKAoIBAQCKoR7/SjkHwA6CuVDivaX1DyyP5a4DPYOkZ0mtGyCi
| EAdHf38gsaFFgs9GoBU7uj1OygMeTrOe0EcPtsmSvcHyfL5jzj4GZ5xcuMn1rO1n
| D95gEpjRtlbQJvuChkfDWRxwnTL4ZkJLPtJLSOUtyEpiQzmAfa4ryXaNJhA3F+eO
| UA5XCtHgM6dJMa2khRH1lETf2N84E67FmIDlysQx/eGKUXluUk3A5oy24Mq6G41L
| Ggxr1vvoMfGY0LXSsuM+Tr6hwfsBmAVf0nq2+zitKjuWceqGwZO7TeXX1RJGBYH3
| mC4jxipRbUq1Z8MMt7zHo1RsSRFSENiWgpojNvRR7OR3AgMBAAGjEjAQMA4GA1Ud
| DwEB/wQEAwIHgDANBgkqhkiG9w0BAQsFAAOCAQEAVjXXsk7cdVAjFlv1wMEc2gSU
| iaRVfbQUZQURUg1FUsaVPHF5ZslknF4bsGXOyDUVWXEdHUhAkKM6PIFEsE6sMf8e
| biD2qJwF3230UfCpBGtRKd0URaepbzFpWDbYUr+f3PkhlhIQNzqNH5WPACiApwTF
| 7g/wmhWT7Z8ImSv3CBopEQV0PlIew7F/jpyHYVS/L18nmo5OSkWGiAT/pUQYExLl
| fl82b23EeVldeXPv023HlyeA+j84Ks8wGcPld/s6lbqFNDKAG5VE6bxnjJ70oWx0
| 1qFOCNSQjZjtuu14TZ3z5+5exMT3KdPpYuDdUZ5Prb06g5L8gGxAwCvVSYUcpQ==
|_-----END CERTIFICATE-----
|_ssl-date: TLS randomness does not represent time
8008/tcp  open  http?            syn-ack ttl 64
|_http-title: Site doesn't have a title (text/html).
8009/tcp  open  ssl/ajp13?       syn-ack ttl 64
| ssl-cert: Subject: commonName=78e099a0-c0de-a07d-906f-22e2ef274f7b
| Issuer: commonName=78e099a0-c0de-a07d-906f-22e2ef274f7b
| Public Key type: rsa
| Public Key bits: 2048
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2023-09-28T20:16:02
| Not valid after:  2023-09-30T20:16:02
| MD5:   c366 09a1 2d73 10b3 a94c 7883 7e7a bce2
| SHA-1: 10c2 8895 3a1e 3ae8 4500 6170 1f85 07f2 4102 144a
| -----BEGIN CERTIFICATE-----
| MIIC2jCCAcKgAwIBAgIEEoVSyjANBgkqhkiG9w0BAQsFADAvMS0wKwYDVQQDDCQ3
| OGUwOTlhMC1jMGRlLWEwN2QtOTA2Zi0yMmUyZWYyNzRmN2IwHhcNMjMwOTI4MjAx
| NjAyWhcNMjMwOTMwMjAxNjAyWjAvMS0wKwYDVQQDDCQ3OGUwOTlhMC1jMGRlLWEw
| N2QtOTA2Zi0yMmUyZWYyNzRmN2IwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK
| AoIBAQCtbBEiicHOpWuypn7I6cAVSD4FF1hIPovR/OuF5q3RBh93urKLWDM/ppSM
| 4qJCLBNMG3T8MZKQkY73ipcksYilruXS7To8CVhaLqyFLE0BsaJGg4MNxrM+/n3+
| AFvn0/y9ACEpmvTNndApJO92os3ZqS35ggbN+zHWUSJoCvLk1UH52PDk5IBB7bpg
| eDM1T4xJixqWdhZkF3sMV5vzphdsoj9YCtrnfFJ7Ryrft5hcF/3qN3Gv5X09SSh2
| ueZ3oLx0yUibrD+y9oRu5AV/2fy22LhpufjW+ILsglIL2rxcMJI5J3Bv5z4uTrEk
| q38I5fQmI2wfJXKozzedP37L4ifRAgMBAAEwDQYJKoZIhvcNAQELBQADggEBAAd4
| PEqqc0aT3kAQOLuSkZ+N85mRJjuD+F5sy16QrCs6ea5kcsJ51k+0tbgbvLFzPmY5
| Fw2/tw3z0UwwlUwlY2t3wHtgbJUN70S97U3iwUpoQQLXGcTklw9dCc0WsbOYybl/
| FTOTnG54NwKj28GyNC/K0lk9VV5+rHQ9HKpu6M1dMkehi+Bl916JqkRk8VO5v+Ny
| L2/bbtj+sEbQKb9bCkBxkLBelh6eo0k9MEQSNyFdwo3BiDayEU9g55jxkZofrYft
| ZddFHdUUd0L74IRF8IWLPLSuSqRuUXsDqzU9VyBdRKdfWJxdDH7zqM+5/G/t7cF6
| rcMwFEasrpuyw422Ve0=
|_-----END CERTIFICATE-----
|_ssl-date: TLS randomness does not represent time
|_ajp-methods: Failed to get a valid response for the OPTION request
8443/tcp  open  ssl/https-alt?   syn-ack ttl 64
| http-cisco-anyconnect:
|_  ERROR: Failed to connect to SSL VPN server
| ssl-cert: Subject: commonName=GN3FTGQ FA8FCA948530/
organizationName=Google Inc/stateOrProvinceName=California/
countryName=US/organizationalUnitName=Cast/localityName=Mountain
View
| Issuer: commonName=Chromecast ICA 30 (ATV)/organizationName=Google
Inc/stateOrProvinceName=California/countryName=US/
organizationalUnitName=Cast/localityName=Mountain View
| Public Key type: rsa
| Public Key bits: 2048
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2019-03-05T08:00:00
| Not valid after:  2042-08-12T08:40:17
| MD5:   2ef0 c7d4 54fb 1dd3 e601 4742 3a07 b6f6
| SHA-1: 8752 6286 e8ab b6af 4cc1 ebd2 204d 8c13 8a13 62a8
| -----BEGIN CERTIFICATE-----
| MIIDuDCCAqCgAwIBAgIHR04zRlRHUTANBgkqhkiG9w0BAQsFADCBgDELMAkGA1UE
| BhMCVVMxEzARBgNVBAgMCkNhbGlmb3JuaWExFjAUBgNVBAcMDU1vdW50YWluIFZp
| ZXcxEzARBgNVBAoMCkdvb2dsZSBJbmMxDTALBgNVBAsMBENhc3QxIDAeBgNVBAMM
| F0Nocm9tZWNhc3QgSUNBIDMwIChBVFYpMB4XDTE5MDMwNTA4MDAwMFoXDTQyMDgx
| MjA4NDAxN1owfTELMAkGA1UEBhMCVVMxEzARBgNVBAgMCkNhbGlmb3JuaWExFjAU
| BgNVBAcMDU1vdW50YWluIFZpZXcxEzARBgNVBAoMCkdvb2dsZSBJbmMxDTALBgNV
| BAsMBENhc3QxHTAbBgNVBAMMFEdOM0ZUR1EgRkE4RkNBOTQ4NTMwMIIBIjANBgkq
| hkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEArakDhfc0csQbxr+zewNA4jJfXu4+JipP
| C1ajHgE4PSenMaOWCoYoC1SbXasul5kozu7aigsqLJGbR9QeCtyn6BD/WkRGVlT7
| n6X1eBtEH2eOelE9R3u0vPWsqe+oPaXZ7LjKzzX+3T8VIHSbSW7QN0dzvWKkHXYf
| fLNlI/S1FXLk2OZaZjck6rEBAELLWusmke/M1gSuyTS8RZW5H5VlXeDKCThnTDYw
| yopYI+qBhL+IzqqN2zBsLnEjLVXj8sMfjKy4hC2g/jktCnnLTAL35FZ/aYAsCy4+
| Vi8741t8fYdgKJV5OXq3FITUUIjsf7fPmFLC63pmTvXB4ZbDyhWHNwIDAQABozkw
| NzAJBgNVHRMEAjAAMAsGA1UdDwQEAwIHgDAdBgNVHSUEFjAUBggrBgEFBQcDAgYI
| KwYBBQUHAwEwDQYJKoZIhvcNAQELBQADggEBAArHy6xVivIIslqj67W/DAca+MJy
| srHsJ7V1aa6jl3Xg1yMdo+cQ3Ls349ARktlEDEpVSfasiBJyWHAb8jEuRVJVN+9j
| XzqYUu+4w/0xmDM2JIkqEXgo49jVoeMvnSYvGghv+i6dck3lnQ1PEXdhaoS0JWq2
| TkhU9McwahAS9DP6mzy+wj7jDckrMfUsuE46u0wq+8Rv1ZXvOh00xPrW2oSvzrVb
| A2W8g+/o+nZFF9k1o1TLQfgfxTyr06IN0rndAcReAsw6hn8ScCso2VF2u9dAR8ES
| aH8zBEnJKf8zBDHPyRLzHgQW05nbHgCPsAvJ95mwK+HzaqtiJJ+Oaa0WBT8=
|_-----END CERTIFICATE-----
| ssl-date:
|_  ERROR: Unable to obtain data from the target
9000/tcp  open  ssl/cslistener?  syn-ack ttl 64
| ssl-date:
|_  ERROR: Unable to obtain data from the target
10001/tcp open  ssl/scp-config?  syn-ack ttl 64
| ssl-date:
|_  ERROR: Unable to obtain data from the target
10101/tcp open  ssl/ezmeeting-2? syn-ack ttl 64
| ssl-date:
|_  ERROR: Unable to obtain data from the target
33157/tcp open  tcpwrapped       syn-ack ttl 64
45829/tcp open  tcpwrapped       syn-ack ttl 64
MAC Address: 32:F3:22:DF:0F:D3 (Unknown)
No exact OS matches for host (If you know what OS is running on it,
see https://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=7.92%E=4%D=9/29%OT=6466%CT=1%CU=37934%PV=Y%DS=1%DC=D%G=Y%M
=32F322
OS:%TM=65168904%P=x86_64-pc-linux-
gnu)SEQ(SP=107%GCD=1%ISR=10C%TI=Z%CI=Z%II
OS:=I%TS=A)OPS(O1=M5B4ST11NW6%O2=M5B4ST11NW6%O3=M5B4NNT11NW6%O4=M5B4
ST11NW6
OS:
%O5=M5B4ST11NW6%O6=M5B4ST11)WIN(W1=FE88%W2=FE88%W3=FE88%W4=FE88%W5=F
E88%
OS:W6=FE88)ECN(R=Y%DF=Y%T=40%W=FAF0%O=M5B4NNSNW6%CC=Y%Q=)T1(R=Y%DF=Y
%T=40%S
OS:=O%A=S+
%F=AS%RD=0%Q=)T2(R=N)T3(R=N)T4(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%O=%R
OS:D=0%Q=)T5(R=Y%DF=Y%T=40%W=0%S=Z%A=S+
%F=AR%O=%RD=0%Q=)T6(R=Y%DF=Y%T=40%W=
OS:0%S=A%A=Z%F=R%O=%RD=0%Q=)T7(R=Y%DF=Y%T=40%W=0%S=Z%A=S+
%F=AR%O=%RD=0%Q=)U
OS:1(R=Y%DF=N%T=40%IPL=164%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)IE
(R=Y%DF
OS:I=N%T=40%CD=S)

Uptime guess: 12.879 days (since Sat Sep 16 07:16:06 2023)
```

```
Network Distance: 1 hop
TCP Sequence Prediction: Difficulty=263 (Good luck!)
IP ID Sequence Generation: All zeros

TRACEROUTE
HOP RTT     ADDRESS
1   7.26 ms 10.0.0.8
Final times for host: srtt: 7259 rttvar: 7031  to: 100000
```

NMAP SCAN DATA BREAK [END OF FILE]

Well, the scan seems to be pretty fucking large. So, lets go ahead and highlight some primary information that nmap was able to pick up. If we do not find anything, we might want to scan in a different format. Even then, sometimes running other scripts may also pose a huge impact for our scan

First Port Set (port = 6466)

Alright, so this one was a bit interesting. The first thing I noticed right away was actually going to be some of the data that was given as an output in accordance to the device itself such as the common name. So, lets go through it.

```
6466/tcp  open  ssl/unknown      syn-ack ttl 64
| ssl-cert: Subject: commonName=atvremote/38:86:F7:37:36:67/
dnQualifier=boreal/boreal/Chromecast HD
| Issuer: commonName=atvremote/38:86:F7:37:36:67/dnQualifier=boreal/
boreal/Chromecast HD
```

This data here states that the ssl-cert has a subject with values "commonName" and "dnQualifier". So, we can already see that we are for one targeting the correct device as we can pull names such as "Chromecast HD" out of it and we can assume that this port is reserved for any commands sent from the remote which is indicated by `atvremote`.

The field `atvremote` with a quick Google search will commonly show "Apple TV" but, since we know we are not scanning a AppleTV and we know chromecasts are not working with AppleTV remotes or directly interfacing with any services on an AppleTV- then we can easily infer that ATV stands for Android TV and of course this is a remote for the Android TV operating system.

Another quick google search and even searching in device settings will help you understand that this device is infact using the ATV-OS (Android TV OS).

Now we should be able to move onto other ports, we also notice that the next port that is scanned gives us similar information. Note that the scan also outputs some crypto and SSL information which can be important in other serious research more than what we are going for.

Second Port Set (port = 6467)

```
6467/tcp  open  ssl/unknown      syn-ack ttl 64
| ssl-cert: Subject: commonName=atvremote/38:86:F7:37:36:67/
dnQualifier=boreal/boreal/Chromecast HD
| Issuer: commonName=atvremote/38:86:F7:37:36:67/dnQualifier=boreal/
boreal/Chromecast HD
| Public Key type: rsa
| Public Key bits: 2048
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2023-03-09T05:00:00
| Not valid after:  2038-01-19T03:14:07
| MD5:   af2b 24e9 45e6 277b 2020 0a8b b75a 0f24
| SHA-1: b07b 30d4 ab9f b40c fff5 1cf6 c43a 747a a98e 6d21
| -----BEGIN CERTIFICATE-----
| MIIDKjCCAhKgAwIBAgIGAYbF2678MA0GCSqGSIb3DQEBCwUAMEwxJDAiBgNVBC4T
| G2JvcmVhbC9ib3JlYWwvQ2hyb21lY2FzdCBIRDEkMCIGA1UEAxMbYXR2cmVtb3Rl
| LzM4Ojg2OkY3OjM3OjM2OjY3MB4XDTIzMDMwOTA1MDAwMFoXDTM4MDExOTAzMTQw
| N1owTDEkMCIGA1UELhMbYm9yZWFsL2JvcmVhbC9DaHJvbWVjYXN0IEhEMSQwIgYD
| VQQDExthdHZyZW1vdGUvMzg6ODY6Rjc6Mzc6MzY6NjcwggEiMA0GCSqGSIb3DQEB
| AQUAA4IBDwAwggEKAoIBAQCKoR7/SjkHwA6CuVDivaX1DyyP5a4DPYOkZ0mtGyCi
| EAdHf38gsaFFgs9GoBU7uj1OygMeTrOe0EcPtsmSvcHyfL5jzj4GZ5xcuMn1rO1n
| D95gEpjRtlbQJvuChkfDWRxwnTL4ZkJLPtJLSOUtyEpiQzmAfa4ryXaNJhA3F+eO
| UA5XCtHgM6dJMa2khRH1lETf2N84E67FmIDlysQx/eGKUXluUk3A5oy24Mq6G41L
| Ggxr1vvoMfGY0LXSsuM+Tr6hwfsBmAVf0nq2+zitKjuWceqGwZO7TeXX1RJGBYH3
| mC4jxipRbUq1Z8MMt7zHo1RsSRFSENiWgpojNvRR7OR3AgMBAAGjEjAQMA4GA1Ud
| DwEB/wQEAwIHgDANBgkqhkiG9w0BAQsFAAOCAQEAVjXXsk7cdVAjFlv1wMEc2gSU
| iaRVfbQUZQURUg1FUsaVPHF5ZslknF4bsGXOyDUVWXEdHUhAkKM6PIFEsE6sMf8e
| biD2qJwF3230UfCpBGtRKd0URaepbzFpWDbYUr+f3PkhlhIQNzqNH5WPACiApwTF
| 7g/wmhWT7Z8ImSv3CBopEQV0PlIew7F/jpyHYVS/L18nmo5OSkWGiAT/pUQYExLl
| fl82b23EeVldeXPv023HlyeA+j84Ks8wGcPld/s6lbqFNDKAG5VE6bxnjJ70oWx0
| 1qFOCNSQjZjtuu14TZ3z5+5exMT3KdPpYuDdUZ5Prb06g5L8gGxAwCvVSYUcpQ==
|_-----END CERTIFICATE-----
|_ssl-date: TLS randomness does not represent time
```

This information is pretty much the same other than a few changing elements. Nothing really sticking out here. We should just move onto the next two ports actually since those services arent chunked together.

Third Port Set (port = 8008/tcp)

Fourth Port Set (port = 8009/tcp)

It would be important to understand these ports, sadly, nmap seems to be kind of confused as to what this port exactly is based on its scan which may indicate a few false positives for services that is possible.

```
8008/tcp  open  http?            syn-ack ttl 64
|_http-title: Site doesn't have a title (text/html).
8009/tcp  open  ssl/ajp13?       syn-ack ttl 64
| ssl-cert: Subject: commonName=78e099a0-c0de-a07d-906f-22e2ef274f7b
| Issuer: commonName=78e099a0-c0de-a07d-906f-22e2ef274f7b
| Public Key type: rsa
| Public Key bits: 2048
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2023-09-28T20:16:02
| Not valid after:  2023-09-30T20:16:02
| MD5:   c366 09a1 2d73 10b3 a94c 7883 7e7a bce2
| SHA-1: 10c2 8895 3a1e 3ae8 4500 6170 1f85 07f2 4102 144a
| -----BEGIN CERTIFICATE-----
| MIIC2jCCAcKgAwIBAgIEEoVSyjANBgkqhkiG9w0BAQsFADAvMS0wKwYDVQQDDCQ3
| OGUwOTlhMC1jMGRlLWEwN2QtOTA2Zi0yMmUyZWYyNzRmN2IwHhcNMjMwOTI4MjAx
| NjAyWhcNMjMwOTMwMjAxNjAyWjAvMS0wKwYDVQQDDCQ3OGUwOTlhMC1jMGRlLWEw
| N2QtOTA2Zi0yMmUyZWYyNzRmN2IwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK
| AoIBAQCtbBEiicHOpWuypn7I6cAVSD4FF1hIPovR/OuF5q3RBh93urKLWDM/ppSM
| 4qJCLBNMG3T8MZKQkY73ipcksYilruXS7To8CVhaLqyFLE0BsaJGg4MNxrM+/n3+
| AFvn0/y9ACEpmvTNndApJO92os3ZqS35ggbN+zHWUSJoCvLk1UH52PDk5IBB7bpg
| eDM1T4xJixqWdhZkF3sMV5vzphdsoj9YCtrnfFJ7Ryrft5hcF/3qN3Gv5X09SSh2
| ueZ3oLx0yUibrD+y9oRu5AV/2fy22LhpufjW+ILsglIL2rxcMJI5J3Bv5z4uTrEk
| q38I5fQmI2wfJXKozzedP37L4ifRAgMBAAEwDQYJKoZIhvcNAQELBQADggEBAAd4
| PEqqc0aT3kAQOLuSkZ+N85mRJjuD+F5sy16QrCs6ea5kcsJ51k+0tbgbvLFzPmY5
| Fw2/tw3z0UwwlUwlY2t3wHtgbJUN70S97U3iwUpoQQLXGcTklw9dCc0WsbOYybl/
| FTOTnG54NwKj28GyNC/K0lk9VV5+rHQ9HKpu6M1dMkehi+Bl916JqkRk8VO5v+Ny
| L2/bbtj+sEbQKb9bCkBxkLBelh6eo0k9MEQSNyFdwo3BiDayEU9g55jxkZofrYft
| ZddFHdUUd0L74IRF8IWLPLSuSqRuUXsDqzU9VyBdRKdfWJxdDH7zqM+5/G/t7cF6
| rcMwFEasrpuyw422Ve0=
|_-----END CERTIFICATE-----
```

The first port is a basic TCP port and seems to connect fine given the previous scan output, but what exactly is this port. I am going to go ahead and check to see if we can connect to this port via HTTP and plug it into curl. Maybe we can find something here. Assuming nmap is correct, we can formulate the following CURL command.

```
curl -v -X GET http://10.0.0.8:8008/
```

```
Note: Unnecessary use of -X or --request, GET is already inferred.
*   Trying 10.0.0.8:8008...
* Connected to 10.0.0.8 (10.0.0.8) port 8008 (#0)
> GET / HTTP/1.1
> Host: 10.0.0.8:8008
> User-Agent: curl/7.74.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 404 Not Found
< Content-Length:0
< Content-Type:text/html
<
* Connection #0 to host 10.0.0.8 left intact
```

This amazing output does not say too much, but we can actually see that it connects just fine and does not truly have a huge issue other than the path not being found. We can assume that this connection might be a HTTP server. Especially after some initial researching showing that port 8008 is a common alternative for HTTP servers. We should most likely not be inferring that this is HTTP but it definitely accepts the connection to some form of method. So, we can at least somewhat assume this. This might be a port to look into.

Port 8009 on the other hand is a bit different. This actually is an SSL/TLS type connection and holds some more information that may be helpful for us to note out. Note that the SSL-cert field in the sub-field "commonName" holds the value

```
commonName=78e099a0-c0de-a07d-906f-22e2ef274f7b
```

This seems to look like a standard UUID which is used a ton in IoT devices especially for APi endpoints and various other services. We should again keep that noted.

Now we can kind of move down the list. This service is also important to keep in mind because this could legit be a AJP13 service that is running, but its also important to understand a bit about AJP13 and its possible misconfiguration. An amazing article on HackArticles pointed out some information that is important for us to note.

https://diablohorn.com/2011/10/19/8009-the-forgotten-tomcat-port/

AJP is a wire protocol. It an optimized version of the HTTP protocol to allow a standalone web server such as Apache to talk to Tomcat. Historically, Apache has been much faster than Tomcat at serving static content. The idea is to let Apache serve the static content when possible, but proxy the request to Tomcat for Tomcat related content. The ajp13 protocol is packet-oriented. A binary format was presumably chosen over the more readable plain text for reasons of performance. The web server communicates with the servlet container over TCP connections. To cut down on the expensive process of socket creation, the web server will attempt to maintain persistent TCP connections to the servlet container, and to reuse a connection for multiple request/response cycles

This can be helpful if we want to go a step further and see if this server is correct and accurate. So, without stoping, we can run a pretty simple command through nmap to check

```
nmap -n -p 8009 -sV --script ajp-auth,ajp-headers,ajp-methods,ajp-
```

for AJP information and see if we can enumerate the service further given that nmap was not too sure about this service. The command below is used for further enumeration. The result from nmap is shown below.

```
PORT     STATE SERVICE    VERSION
8009/tcp open  ssl/ajp13?
|_ajp-methods: Failed to get a valid response for the OPTION request
```

We could in the future try various options if the service did fail and see if there are any quicker ways to enumerate this. But this may mean that nmap was not fully correct on the service or one of the methods is just not in service. Again, maybe there is a function to trigger to test the buttons and services later.

Moving on, we can go back to our scan and see some interesting information about the services and other things we may have seen.

Fifth Port Set (port = 8443)

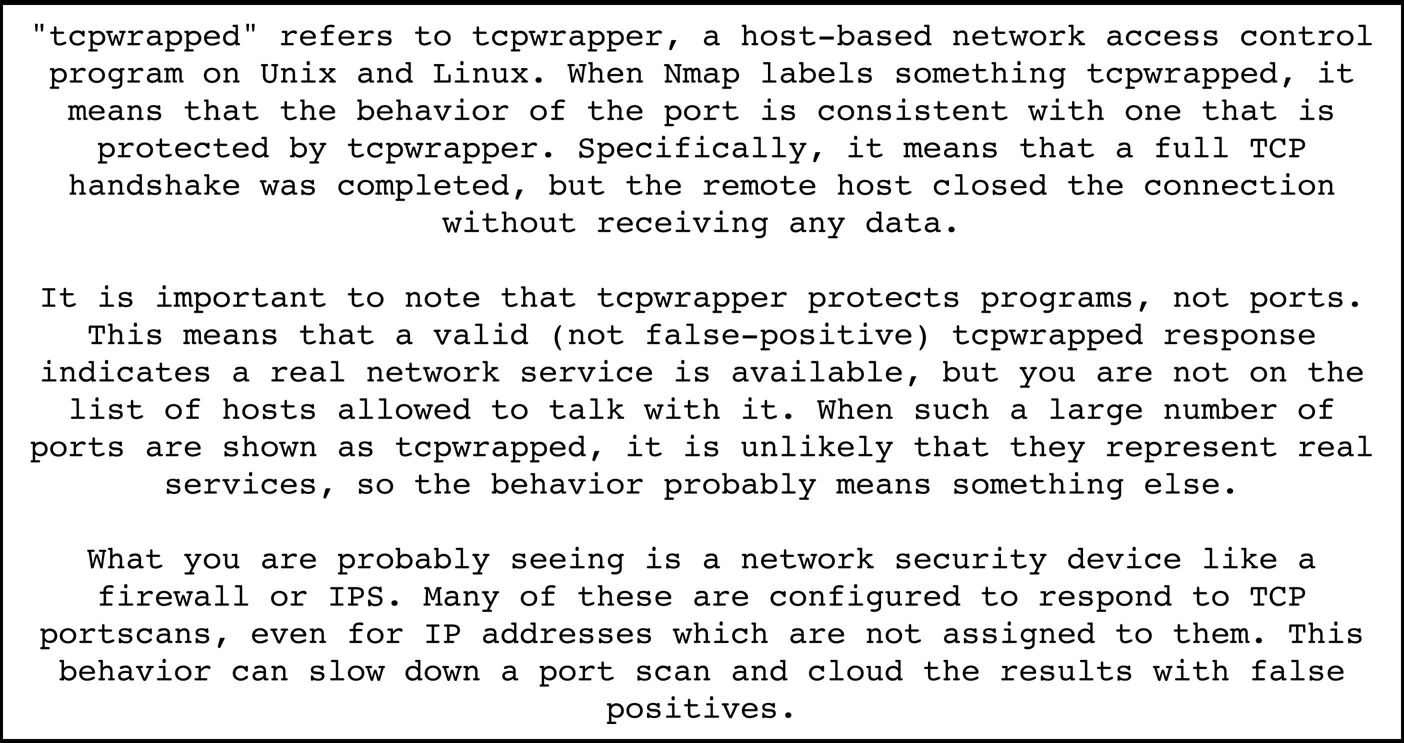
```
8443/tcp  open  ssl/https-alt?   syn-ack ttl 64
| http-cisco-anyconnect:
|_  ERROR: Failed to connect to SSL VPN server
| ssl-cert: Subject: commonName=GN3FTGQ FA8FCA948530/organizationName=Google Inc/stateOrProvinceName=California/countryName=US/organizationalUnitName=Cast/localityName=Mountain View
| Issuer: commonName=Chromecast ICA 30 (ATV)/organizationName=Google Inc/stateOrProvinceName=California/countryName=US/organizationalUnitName=Cast/localityName=Mountain View
| Public Key type: rsa
```

This port is what I would assume to be the more "https" / "ssl" version of port 8008. So, my thought process here is that maybe the device is going to be hosting two servers for separate functionalities, maybe the device is hosting some form of functionality that requires some authorization headers and other various sets of information. We also notice that the names are a bit different to, we have some common names and issuer fields that are a bit different.

Last Port Set

```
33157/tcp open  tcpwrapped       syn-ack ttl 64
45829/tcp open  tcpwrapped       syn-ack ttl 64
MAC Address: 32:F3:22:DF:0F:D3 (Unknown)
No exact OS matches for host (If you know what OS is running on it,
see https://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=7.92%E=4%D=9/29%OT=6466%CT=1%CU=37934%PV=Y%DS=1%DC=D%G=Y%M
=32F322
OS:%TM=65168904%P=x86_64-pc-linux-
gnu)SEQ(SP=107%GCD=1%ISR=10C%TI=Z%CI=Z%II
OS:=I%TS=A)OPS(O1=M5B4ST11NW6%O2=M5B4ST11NW6%O3=M5B4NNT11NW6%O4=M5B4
ST11NW6
OS:
%O5=M5B4ST11NW6%O6=M5B4ST11)WIN(W1=FE88%W2=FE88%W3=FE88%W4=FE88%W5=F
E88%
OS:W6=FE88)ECN(R=Y%DF=Y%T=40%W=FAF0%O=M5B4NNSNW6%CC=Y%Q=)T1(R=Y%DF=Y
%T=40%S
OS:=O%A=S+
%F=AS%RD=0%Q=)T2(R=N)T3(R=N)T4(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%O=%R
OS:D=0%Q=)T5(R=Y%DF=Y%T=40%W=0%S=Z%A=S+
%F=AR%O=%RD=0%Q=)T6(R=Y%DF=Y%T=40%W=
OS:0%S=A%A=Z%F=R%O=%RD=0%Q=)T7(R=Y%DF=Y%T=40%W=0%S=Z%A=S+
%F=AR%O=%RD=0%Q=)U
OS:1(R=Y%DF=N%T=40%IPL=164%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)IE
(R=Y%DF
OS:I=N%T=40%CD=S)
```

The last port set is actually considering that there are two seperate ports for the TCPwrapped service which was actually explained pretty well for those that are not aware of what TCPwrapped is. Below is shown from the live overflow link.



"tcpwrapped" refers to tcpwrapper, a host-based network access control program on Unix and Linux. When Nmap labels something tcpwrapped, it means that the behavior of the port is consistent with one that is protected by tcpwrapper. Specifically, it means that a full TCP handshake was completed, but the remote host closed the connection without receiving any data.

It is important to note that tcpwrapper protects programs, not ports. This means that a valid (not false-positive) tcpwrapped response indicates a real network service is available, but you are not on the list of hosts allowed to talk with it. When such a large number of ports are shown as tcpwrapped, it is unlikely that they represent real services, so the behavior probably means something else.

What you are probably seeing is a network security device like a firewall or IPS. Many of these are configured to respond to TCP portscans, even for IP addresses which are not assigned to them. This behavior can slow down a port scan and cloud the results with false positives.

There are a few issues with this, because this means that our scan might have been detected especially given methods and options such as `-A` and `-sV` were used in the scan. Given that tcpwrapped can be known as a firewall, then it will catch noisy scans and those two options are asking for a world of hurt when it comes down to scanning which may also explain the confusion in our scan. If worse comes to worst, we can just make a slower scan and use options that may not be caught by the IDS or firewall (based on the info above)
Alright, by now we have gathered a bunch of helpful information and the most helpful being information on the services and attempted enumeration on very specific services. To sum up, we have the following information about the device.

Ports	Service	Device Information	
6466	tcp || ssl unknown	MAC: 32:F3:22:DF:0F:D3	
6467	tcp || ssl unknown	UUID (assuming): commonName=78e099a0-c0de-a07d-906f-22e2ef274f7b	
8008	http || tcp	Remote: atvremote/ 38:86:F7:37:36:67	
8009	ssl/ajp13? || tcp	OS: Android TV (ATV)	
8443	http(s) || ssl/https-alt	Remote: ATVREMOTE	
9000	ssl/cslistener?	Remote address? 38:86:F7:37:36:67	
10001		IP: 10.0.0.8	
10101			
33157			
45829			

This information was actually pretty helpful and can help us in a wide variety of scans. If we really wanted to or needed to, we can take this information for a bit of an informal dive. Which actually is our next section.

0x005 Exploring Google Devices - Section 2

This section is the `Attacking Google With Google` section of the document. In this section, we will be discussing some more information finding and gathering with the Google chromecast using the information we have found with the most basic nmap scan and some basic enumeration whilst plugging it all into the very thing that created this device- Google.

Google is by far one of your best friends when going to look for information of a device- if someone tells you they are a hacker- they best know how to Google.

Right now, I am particularly interested in the devices ability to host and interact with information external to the device itself and interact with data from other devices. We will get more into setting up the device and so on from there, but right now, I am purely interested in finding more out about the web services such as port 8008, 8009 and other or any known vulnerabilities for other services this Cast may be running. So, of course, lets get to Google dorking shall we!

Note: For those who do not know, Google dorking is the ability to use Google to find and query for very specific information using very specific search parameters. For example, we can use `filetype: pdf` to search for anything revolving around a pdf with our search query or even just in general find random PDFs sitting on Google.
First, I want to see what might be hiding when we are using HTTP so lets look for services on the whole realm of HTTP with a cast.

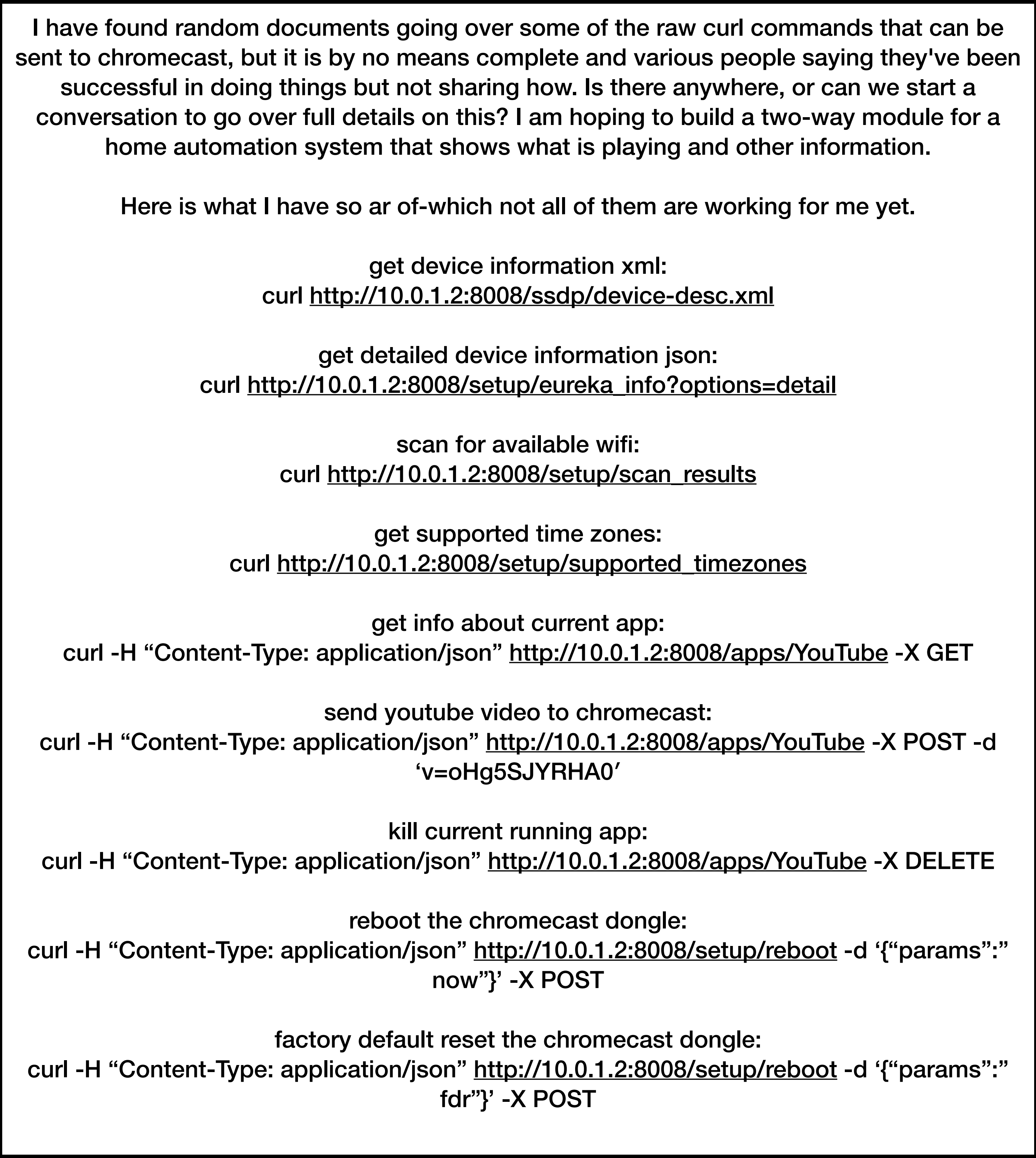
After searching and typing the following query - we get some interesting documentation.

```
inurl unofficial documentation intitle chromecast
```

The reason we are searching for unofficial documentation is because this can help speed up our research process into understanding what is and what is not happening behind the scenes for someone who may have already beaten us to it. Well, in this case someone did and just gave us a ton of good incite!

A post on a thread from XDA forums states the following information.

Wow, what a huge hit for information- looking further into it we can see that there are some differences here. From further reading and documentation issues, it was stated that in 2015 being able to do this may pose some form of authorization issue with the API especially



I have found random documents going over some of the raw curl commands that can be sent to chromecast, but it is by no means complete and various people saying they've been successful in doing things but not sharing how. Is there anywhere, or can we start a conversation to go over full details on this? I am hoping to build a two-way module for a home automation system that shows what is playing and other information.

Here is what I have so ar of-which not all of them are working for me yet.

get device information xml:
```
curl http://10.0.1.2:8008/ssdp/device-desc.xml
```

get detailed device information json:
```
curl http://10.0.1.2:8008/setup/eureka_info?options=detail
```

scan for available wifi:
```
curl http://10.0.1.2:8008/setup/scan_results
```

get supported time zones:
```
curl http://10.0.1.2:8008/setup/supported_timezones
```

get info about current app:
```
curl -H “Content-Type: application/json” http://10.0.1.2:8008/apps/YouTube -X GET
```

send youtube video to chromecast:
```
curl -H “Content-Type: application/json” http://10.0.1.2:8008/apps/YouTube -X POST -d ‘v=oHg5SJYRHA0′
```

kill current running app:
```
curl -H “Content-Type: application/json” http://10.0.1.2:8008/apps/YouTube -X DELETE
```

reboot the chromecast dongle:
```
curl -H “Content-Type: application/json” http://10.0.1.2:8008/setup/reboot -d ‘{“params”:”now”}’ -X POST
```

factory default reset the chromecast dongle:
```
curl -H “Content-Type: application/json” http://10.0.1.2:8008/setup/reboot -d ‘{“params”:”fdr”}’ -X POST
```

when it came down to rebooting and factory resetting the device. Of course this in itself posed its issues but it is important that maybe look into this.

When setting up our device with some information from the post and upon further research, it is apparent that 90% of those URLs do not work anymore most likely because Google has decided to patch them and that unauthenticated command requests are not the safest thing in the world! So, now what? Well, we can work with the URLs we do have.

As mentioned in the previous state, some of these URLs need to be going through a step by step process such as walking through one authentication step, then authorization step, then initiation step and then a data step- so, lets go ahead and get down what actually works.

The first thing I decided to try out was the ones for device information- because the more device information the better. So, my better judgment gave me the idea of messing around with SSDP related paths. The first path will be

http://10.0.1.2:8008/ssdp/device-desc.xml

Of course we will be formatting this with our own address, but when we send off the command and output the information, we get a very very weird response which gives us some good info on the device. Shown below is some XML output

```
<?xml version="1.0"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <URLBase>http://10.0.0.8:8008</URLBase>
  <device>
    <deviceType>urn:dial-multiscreen-org:device:dial:1</deviceType>
    <friendlyName>SomeRandomName</friendlyName>
    <manufacturer>Google</manufacturer>
    <modelName>Chromecast HD</modelName>
    <UDN>uuid:78e099a0-c0de-a07d-906f-22e2ef274f7b</UDN>
    <iconList>
      <icon>
        <mimetype>image/png</mimetype>
        <width>98</width>
        <height>55</height>
        <depth>32</depth>
        <url>/setup/icon.png</url>
      </icon>
    </iconList>
    <serviceList>
      <service>
        <serviceType>urn:dial-multiscreen-org:service:dial:1</
serviceType>
        <serviceId>urn:dial-multiscreen-org:serviceId:dial</
serviceId>
        <controlURL>/ssdp/notfound</controlURL>
        <eventSubURL>/ssdp/notfound</eventSubURL>
        <SCPDURL>/ssdp/notfound</SCPDURL>
      </service>
    </serviceList>
  </device>
</root>
```

Wow! Would you look at that, we actually got some amazing information to pick apart about the devices and can confirm a few things about the information we gathered.

A. Base Server: The base URL for the SSDP configuration must be set to the HTTP server on port 8008! We can see this as we understand this device is forming the information and holding information for SSDP related data which we can confirm even further when we get into using Wireshark.

B. UDN: The UDN or Unique Device Name which was not too far off from our whole "uuid" but we can actually picture this being used in some scenarios where two casts may have the same friendly name but could be distinguished by their UDN especially if there is a ton of service discovery noise happening.

C. Service Information: One of the ports that were listed on there was for some form of Secure Copy Protocol (SCP) configuration which read (scpconfig). This is actually important to us because we might be able to further enumerate and gain access to the devices internal file system if we can get some more information and bypass restrictions. Of course this is more " exploitation "- but the SSDP settings confirms this with the SCPDURL.

This information actually confirms aa ton for us and gives us a lot to work with when we start working with the more dynamic side of things and network side of things- especially touching on SSDP, UDP and various other protocols!