André Oudhof Follow
Dec 18, 2020 · 17 min read · Listen

PWNing my ISPs STBs

Remember that scene from the movie 'Hackers' where ZeroCool hacked into the TV station and changed the currently airing TV program to an episode of The Outer Limits? I think this is as close as I'll ever come to replicating that scene (short of actually hacking into a TV network since, you know.. that would be illegal).

Earlier this year I decided to take a look at the *Aminocom Aria 7* settopbox (STB) I received from my ISP Caiway / Delta to see if I could get shell access on the device and take a further look into its inner workings.

As it turned out I was not only able to get a shell on the device but also **take full control of all STBs in the IPTV network**, which allowed me to view details of the streams customers were watching, changing the channels, control volume and even streaming my own content to the devices.

This resulted in 5 CVEs:

- CVE-2020–10206 - Use of a Hard-coded Password in VNCserver in Amino Communications Aria 7
- CVE-2020–10207 - Use of Hard-coded Credentials in EntoneWebEngine
- CVE-2020–10208 - Command Injection in EntoneWebEngine
- CVE-2020–10209 - Command Injection in the CPE WAN Management Protocol (CWMP) registration
- CVE-2020–10210 - Hard-coded SSH keys

All findings have been resolved by Caiway/Delta and Aminocom after I informed them. Both had a very professional approach and took these issues seriously.

I've had a fiber connection in my home for years and also opted for the 'digital TV' package, which basically means I switched my main TV from a DVB-C setup to IPTV. The fiber connection in my home terminates in a FTU which has 2 LAN ports, 2 IPTV ports and 2 phone ports. All the device does is split the VLANs that are supplied over the fiber channel to the individual ports on the device. (I managed to root this device as well but that requires local / physical access, so not that spectacular and not part of this write up).

Originally the service would only allow a single STB to be connected, and since I also have a family I thought it would be wise not to mess with our main TV-setup. However, my ISP now offered a free second STB and enabled the second IPTV port, so I jumped on the opportunity and ordered one.

When I received the device the first thing I did (obviously) is open it up:

In order to prevent the device from receiving any firmware and configuration updates I did not want to connect the device to the internet / IPTV network until I had shell access to the device.

Serial interface

So first things first, lets find and connect the serial interface, so we can see what's happening:

```
CPU: 2x B53 (64-bit) [420f1000] 1503 MHz
SCB: 324 MHz
SYSIF: 751 MHz
DDR0 @ 1193MHz
GPIO: Turn the blue and red LED on
GPIO RESET disabled
SPLASH: starting
SPLASH BMEM init @ 40000000
PNG Info - Width = 720 Height = 480 ; index=0
PNG Info - Width = 720 Height = 480 ; index=1
PNG Info - Width = 1280 Height = 720 ; index=2
PNG Info - Width = 720 Height = 480 ; index=0
AVS: park check
AVS: temperature monitoring enabled
AVS: STB: V=0.995V, T=+29.206C, PV=0.862V, MV=1.001V, FW=30343978 [0.4.9.x]
board selection update!
board was ?, now a (flags:0xffe000f8)
AUTOBOOT [waitusb -o -d="USB Disk" && batch -e usbdisk0:/recovery/7260/start.txt]
USB device matching <USB Disk> not found!
Loader:zimg Filesys:raw Dev:emmcflash0.appkernel File: Options:vmalloc=434m bmem=440m@568m boot_dev=bolt active_module=
```

Boot with USB config file

So now that we have serial output I noticed that during boottime the device is looking for a specific file on a USB device. Obviously the first thing I did was create a file with that name on a USB-stick and reboot the device with the USB-stick attached.

It appeared the file required some special header and footer before it would be accepted by the device. Since I could not find any more information about this file (or the required header/footer) I started looking for other options to get access to the device.

Network traffic / cpemgt

I started Wireshark and connected my laptop to the ethernet port of the device.

Since the device is issuing DHCP requests I setup a DHCP server to supply it with an IP so maybe it would generate some traffic.

Nice, it appears to be looking for a domain CPEMGT and if that fails it falls back to cpemgt.entone.com . Then I setup a DNS server that replied to all DNS requests with my own IP in order to redirected all traffic to my laptop:

Now I could see the device initiated a HTTP POST request to my laptop. That's great, but I did not know what kind of response the device was expecting. To figure this out I tried sending the same request to the 'real' cpemgt.entone.com server in order to see what it would send back.

It appears to respond with a small JSON file that contains NOP (which I guessed meant No OPeration). So I then setup a webserver in order to send that exact response back to the device. It did indeed just do that: nothing!

Portscan

Before looking any further into this I decided to look at the results of a portscan I did of the device:

A lot of interesting services, including an SSH server, webserver and vulnerable version of gSOAP. I tried to login with some default credentials on the SSH server but that didn't work. Then I fired up hydra in order to bruteforce my way in but that wasn't working either.

gSOAP

One of the running services was a vulnerable version of gSOAP. Great, but that results in a catch22 situation because in order to create a working exploit I would first need shell access. And to get shell access I would need a working exploit. (Creating an exploit without a debugging environment / shell access is really difficult since you're essentially in the dark, you can't see the results of your exploit attempts.) I did manage to crash the TR069 process and control the instruction pointer:

(screenshot made after getting shell access)

Then I started looking at the webservices but those required authentication. Obviously I tried bruteforcing, gobuster etc. but that did not yield any results.

Finding similar firmware

Since I did not have a lot of information to go on I tried to find the firmware of the device online so I could get some more information about these services. I did some google-fu and found firmware for another Amino device. Since firmware of the same vendor often does not vary a lot between different devices I figured this might give some more insight in the device and services.

I found some credentials online and in the firmware that I tried to use in order to login using the SSH service found on port 10022 but those did not work.

Then I tried finding more information about the CPEMGT HTTP request I saw earlier since I figured there would be more options other than the NOP response I saw earlier. The file 'cpemgttools' looked interesting. The file contained a link to a document on Google Docs.

The document was world readable! Awesome, now I had detailed information about the responses I could send to the device.

So now that I knew what kind of response the device expects I tried to include a URL (to a configuration file) in my response to the device. The document mentioned the configuration file had to be signed with a private key (that I did not have), but I wanted to try anyway because it would not be the first time a signature is not properly verified. However, the file was not accepted.

CVE-2020–10209 — Command Injection in the CPE WAN Management Protocol (CWMP) registration

Then I tried some command injection in the URL:

Yay, looks like we have command injection!

Let's pop a reverse shell.

So now that I had a root shell I could start looking for other vulnerabilities. I made a copy of the flash memory in order to do some offline grepping and checking, and also had a look at the running processes.

Bootloader

Since I now had access to the bootpartition I could check what kind of header & footer the bootloader expected when using the /recovery/7620/start.txt file when booting. By simply using the 'strings' command we can see what the bootloader is checking:

After adding the header #ENTONE_BATCH_START and footer #ENTONE_BATCH_END to the file I now also had access to the bootloader. I briefly tried changing the 'init' parameter to start /bin/sh when booting but that didn't seem to work. Since I already had shell access I did not look any further into this.

SSH access

During the portscan I noticed an SSH server running on port 10022, but I was not able to login since default credentials (and combo lists) did not work. Also, the credentials I found in the other firmware version did not work.

Since I had access to the /etc/shadow file I tried bruteforcing the hashes. I actually threw quite an amount of GPU power at it and tried all known password lists with a lot of rulesets and tried bruteforcing up to a length of 8 including all chars. I was not able to crack the passwords.

CVE-2020–10210 — hard-coded SSH keys

Checking further the /root/.ssh/authorized_keys file contained a whopping 31 authorized keys! Aminocom informed me those keys are used in order for their customers (operators) to connect to the devices. I'm not sure why they would need 31 keys on the device, some of which appear to be really outdated. My guess is they added keys of all customers to the device, so they could roll out a single firmware file for all customers instead of supplying each customer with individual firmware. This would mean that any of the other ISP's could also access this device.

Since the filesystem is read-only I couldn't simply add my own public key to the authorized_keys file. So I copied the /etc directory to /tmp and modified the root password hash in the shadow file. Then I mounted the modified directory on top of the 'real' /etc directory and was able to login.

CVE-2020–10206 — Use of a Hard-coded Password in VNCserver

One of the running services was a VNC server on port 5900. This service was not listening on the external interface, so it could only be reached after connecting to the device via SSH. By decompiling the vncserver in Ghidra it was possible to determine the used password: envnc123 . The VNC server allows a user to remotely view the user interface as displayed over the HDMI port.

CVE-2020–10207 — Use of Hard-coded Credentials in EntoneWebEngine

Since I now had access to the filesystem I was interested in looking further into the webserver that was running on the device (port 10080 and 10443) and see if I could find the required credentials. I had already noticed a file called /tmp/fe.htpasswd in the /tmp directory. That contained the following line:

fe:entone.com:709c8dc50a254fbc88b1cb85bfc6487c

I opened op the EntoneWebEngine binary in Ghidra to check the authentication mechanism and ended up in the library libFem.so:

So there are the credentials in clear text, username is "**fe**" and password "**fe!@#**". As it turned out when accessing the server on port 10080 from localhost you don't even need credentials, you do need them for port 10443 though (which is the same server). I was now able to login on the webserver, but I was still in the dark about its actual functionality since the decompiled binary was not very easy to read. I tried gobuster once again and since I no longer got the 403 error I was now able to find some interesting things:

For instance, I was able to get some information about the device from the webserver:

By looking at the binary it was obvious there was a lot more functionality in this webserver and it would be easiest if I could get a look at a working setup and just see how the device interacted with the webserver by dumping/watching the traffic.

For this I needed to connect the STB to the IPTV network while still being able to store and execute commands in order to debug / intercept requests to the server. But connecting to the IPTV network might force a firmware update, and if the bug I found was already patched I would lose access to the device. I figured it probably wasn't patched and even if it was; I had a copy of the flash memory and access to the bootloader, so I might be able to gain access again.

So I connected the device to the IPTV network and let it boot and upgrade its firmware. When I tried the exploit it still worked. Woohoo :)

The problem was that in order to get access to the device I had to connect it to my laptop (so I could do the command injection during boot) instead of the IPTV network. So once I had a shell on the device I could no longer connect it to the IPTV network. I tried various things to let it reinitialize the network stack and do a partial boot after issuing my commands, but I could not get it to work. So I had to choose between a shell/executing commands on the device, or a working setup.

I started looking into how the previous command injection actually worked and found a file called /tmp/mnt/persist/cpemgt_last_register_stb_result that contained the command injection I had used:

I hadn't realized that the command injection actually would survive a reboot of the device. If this file exists it would try these settings first, and if that failed it would continue booting normally. So now I was able to store commands and have them executed once the device booted. Too bad that once the device was connected to the IPTV network a reverse shell would not reach my laptop since it's not connected to the internet but an IPTV network right? Or so I thought..

IPTV network

I didn't really have a look at the actual IPTV network yet and was mostly messing around with the STB. I plugged in the IPTV ethernet cable to my laptop to check what systems I could reach. What I saw was not what I expected:

There were a lot of ARP requests and multicast traffic from other devices. I expected to be in an isolated environment but this IPTV network turned out to one big LAN where all the connected devices are able to interact.

So when I did an arp-scan of the network thousands of devices started responding:

After doing a quick nmap scan of ports 10022,10080,10443 almost all devices appeared to have at lease some of these ports open. Now things got really interesting! I could connect to these devices and read the device settings using the webservice on ports 10080 and 10443. I also could change some of the settings, but I still didn't know all the options of the webservice.

I assumed the network would not have a connection to the internet, so it would be difficult to get a reverse shell on my device. This turned out to be a false assumption, because I could actually also open connections to the internet. The device was just behind a NAT firewall, so although I could not connect straight to the device from the internet I was able to make the device connect to my laptop over the internet, and then use that connection for a reverse tunnel back into the device.

All files stored in the /tmp/mnt/persist directory survive a reboot, so I had a place to store some files on the device. I created a small bash script that would mount a modified /etc/shadow file at the end of the boot process, and initiate a SSH session to my server using a publickey I stored in the persist directory.

You might wonder what's 'dropbear_modded'? It's a version of dropbear I modified to store all credentials that are used to login to the STB. I hoped some maintenance tool or administrator would use a password login to connect to the device every now and then so I could grab the plaintext password. Unfortunately no logins were detected :)

Once the device connected to my server at the end of the bootsequence I was able to connect using the reverse SSH tunnel, and now I had a fully functional STB while still connected using SSH.

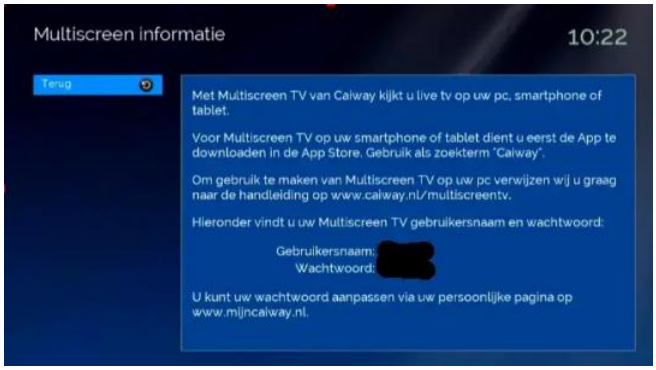
Now that I had a working setup I could run tcpdump to capture all requests to the webserver in order to gain better insight in all the functions. Looking at the tcpdump data it became clear that the internal webserver was used by the webbrowser on the device to change settings like the current channel, volume, enable teletext etc.

So with the information I had gathered I was now able to get detailed information of all devices in the network (what channel are people watching, at what volume, resolution etc). And I was able to change the channel and modify the volume of these devices remotely :)

This was great but now that I knew I could reach all these devices on the network I wanted to find a remote code execution exploit that would allow me to take full control of the devices.

Getting a customer's password

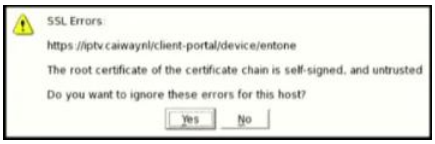
Something else that caught my eye was that one of the menu's on the device would show my customer number and a password that can be used for viewing TV online or using a mobile app. I was wondering where it got this information from, and how the authentication process worked.



While browsing files on the device I noticed the sys_config.txt file that contained a lot of settings, including a setting called "proxy".

```
# HDMI deep color format (0: off, 1: auto [default], 2: 30-bit, 3: 36-bit: 4: 48-bit)
deep_color_format=1

# HDMI color matrix
#   (0: auto [default], 1: BT.601, 2: BT.709, 3: BT.2020 NCL: 4: BT.2020 CL)
hdmi_color_matrix=0

# STB profile
# (0: Default, 1: Transcode)
stb_profile=0

# HDCP scheme (0:AUTO, 1: Force HDCPv1, 2: Force HDCPv2)
hdcp_scheme=0

# Proxy Setting
proxy_url=""
proxy_username=""
proxy_password=""

# TVI baud rate (0: 9600, 1: 19200, 2: 1200, 3: 2400, 4: 4800, 5: 38400, 6: 57600, 7: 115200)
tvi_baudrate=0

# TVI max intercharacter time (0-50 inclusive, in centiseconds)
tvi_interchar_time=0

# TVI binary mode (0: Off, 1: On)
tvi_binary_mode=0
```

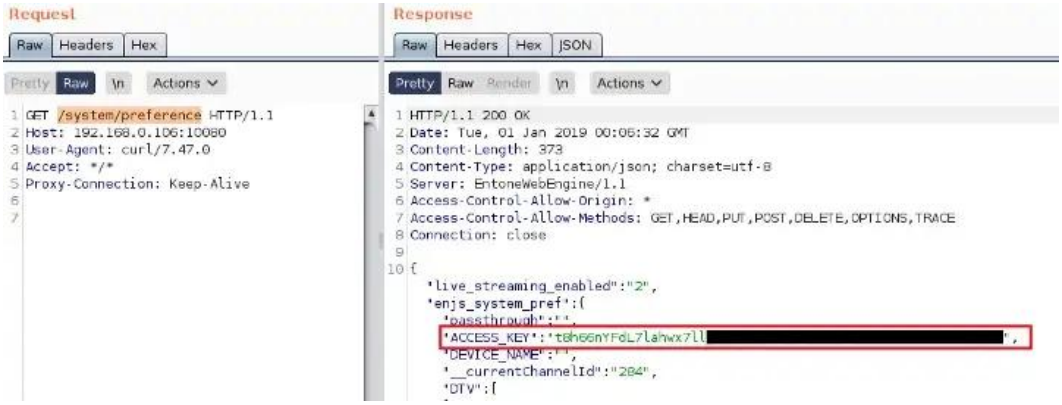
I changed the proxy to point to a Burp instance on my laptop (over the internet) and rebooted the device. I was then presented with a certificate error on my TV screen, and by pressing the 'OK' button on my remote I could simply accept the certificate and continue.

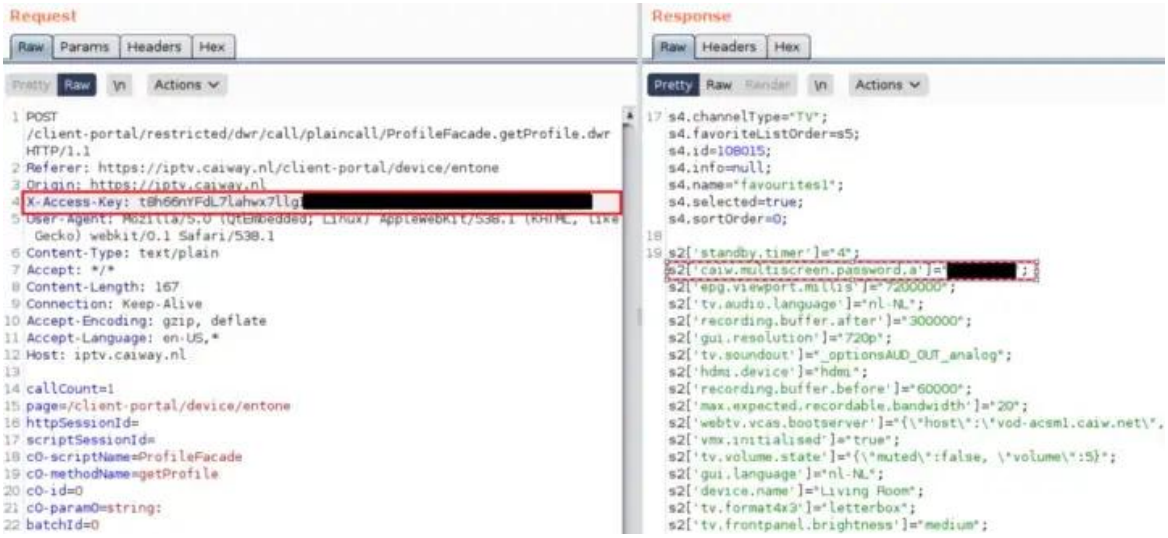


6824	https://iptv.caiway.nl	GET	/client-portal/device/entone
6825	https://iptv.caiway.nl	GET	/client-portal/fokuson/load/?module=ENGINE_2_0_3&buildid=3.23.15.3231520738&device=ENTONE
6826	https://iptv.caiway.nl	GET	/client-portal/modernizr.custom.js
6827	https://iptv.caiway.nl	GET	/client-portal/images/hour-glass-default.png
6828	https://iptv.caiway.nl	GET	/client-portal/favicon.ico
6829	https://iptv.caiway.nl	GET	/client-portal/fokuson/load/?module=CORE&module=FUI&module=ENVIRONMENT&buildid=3.23.15.3231520738&devi...
6830	https://iptv.caiway.nl	GET	/client-portal/json-facade/CommonPropertyFacade.getStringProperties?key=logical.channel.numbers.enabled&key=ch...
6831	https://iptv.caiway.nl	POST	/client-portal/rs/device/updatedevice2
6832	https://iptv.caiway.nl	GET	/client-portal/json-facade/ResourceFacade.getCoreOptions
6833	https://iptv.caiway.nl	GET	/client-portal/auth/rs/channels/bytype?subscribed=true&applyfavorites=true&applylocks=true&allchannelids=true
6834	https://iptv.caiway.nl	POST	/client-portal/restricted/dwr/call/plaincall/ProfileFacade.getProfile.dwr
6835	https://iptv.caiway.nl	POST	/client-portal/restricted/dwr/call/plaincall/CustomerFacade.getCustomerData.dwr
6836	https://iptv.caiway.nl	GET	/client-portal/rs/product/63
6837	https://iptv.caiway.nl	GET	/client-portal/rs/product/73
6838	https://iptv.caiway.nl	GET	/client-portal/fokuson/load?module=PVR&device=ENTONE&buildid=3.23.15.3231520738
6839	https://iptv.caiway.nl	GET	/client-portal/rs/widgets?resolution=720p&groupid=23&groupid=35
6840	https://iptv.caiway.nl	GET	/epg-cache/query?json=%7B%22channels%22:[10,11,111,176,184,213,24,254,279,281,284,285,305,308,309,310,311,...

After some investigation it turned out that an authentication key is stored on the device and is used to retrieve data from the Caiway website. For most data only the authentication key is sufficient and for some other data also a MAC address and/or hostname of the device are required. The data that can be retrieved contains the customer number, IPTV password, details about online recordings, and programming guide data.

Since all the required information is also available via the webserver on the device we can now impersonate any user and retrieve the customer number, password and get an overview of the user's recordings. We can also delete the recordings or add some new. And we can now use the retrieved password to watch pay-per-view or premium channels using another customer's account.





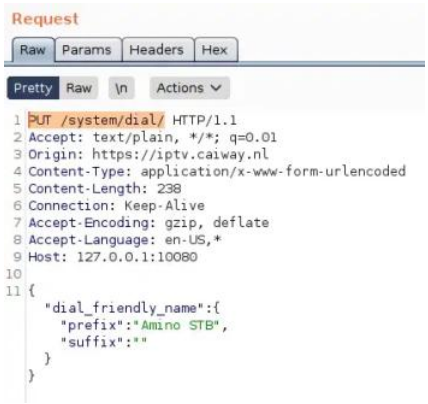
I was still looking for a way to remotely gain control over the STB. The sys_config.txt also contained network configuration settings, and I figured if one of the settings that I can change using the webserver is saved in this config_sys.txt file I might be able to inject newlines and overwrite the DNS or proxy settings.

None of the functions that I had already found until now saved any data in this config file, but by looking at the server binary in IDA / Ghidra I noticed that there were a lot more functions available then I had seen by looking at the traffic. The binary was a mess to read though, so I decided to create a quick wordlist with words that I extracted from the binary and ran Gobuster once more but now with a targeted wordlist.

CVE-2020–10208 — Command Injection in EntoneWebEngine

I then stumbled upon the endpoint '/api/dial'. And it stored the settings inside the sys_config.txt file. After trying injecting a newline it appeared that actually worked! Depending if the device would accept a second configuration line with DNS or proxy I should now be able to configure a remote device with my own proxy & DNS.

This is the unmodified PUT request:



Which results in the following lines in the sys_config.txt file:

```
cc_display_font_dual=0          # default: 0
cc_display_pen_size_dual=0              #default: 0
cc_display_pen_style_dual=0             #default: 0
deep_color_format=1
dial_friendly_name="Amino STB"
dial_friendly_suffix=""
dvbt_upgrade_freq=701000000
external_antenna_power=1
hdcp_scheme=0
```

Then, sending a modified request including newlines and escaped double quotes:

```
{
  "dial_friendly_name":{
    "prefix":"Amino STB\"\nNewVariable=\"NewValue",
    "suffix":""
  }
}
```

We get the following content in the sys_config.txt file:

```
cc_display_pen_style_dual=0             #default: 0
deep_color_format=1
dial_friendly_name="Amino STB"
NewVariable="NewValue"
dial_friendly_suffix=""
dvbt_upgrade_freq=701000000
external_antenna_power=1
hdcp_scheme=0
```

I tried changing the DNS and rebooted the device (btw rebooting can also be done remotely using the webinterface :)). To my surprise I was greeted with multiple syntax errors while booting the device (I had forgotten one of the double quotes). This sys_config.txt file was actually sourced (imported) into multiple other shellscripts during boot. This meant I could inject commands in the config file and they would be executed at boot time. So now we have remote code execution on all STBs connected to the IPTV network!

Sending PUT request with a command:


```
}
```

Results in the following sys_config.txt file:

```
cc_display_pen_style_dual=0             #default: 0
deep_color_format=1
dial_friendly_name="Amino STB"
NewVariable="$(echo w00000000000t >> /dev/console)"
dial_friendly_suffix=""
dvbt_upgrade_freq=701000000
external_antenna_power=1
```

And this is a snipped of the console output when booting:

```
******************** Run video_system_detect  13.49 ***************
******************** init_display 13.50 ********************
init_display
Color System information found in hwblk
w00000000000t
w00000000000t
####### Determining External RF modulator #######
External RFM not detected.
####### Done Determining External RF modulator #######
w00000000000t
Checking video frequency for single mode!
w00000000000t
for Europe
Pal Box, need detect scart switch
init scart for Aria7
w00000000000t
####### Disabled video fixed format #######
done init_board
```

I was pretty much done at this point, but there was one more thing I just wanted to do; being able to stream my own content to all STBs in the network :)

Once you have shell access that was pretty easy. I could simply store my videofile (an .mp4 for example) in the /tmp/mnt/persist directory of a device. Then I could start playing the file using the webserver:

So there you go, that's how I pwned all the STBs in my ISPs IPTV network!
Vulnerable Amino devices:

- AK45x series
- AK5xx series
- AK65x series
- Aria6xx series
- Aria7/AK7Xx series
- Kami7B

Updated firmware has been released for all devices.

Please feel free to connect with me on Linkedin: https://www.linkedin.com/in/l33t/