

Date: 2019-09-03 Author: Ian Rash Index: #6

Now we are at the point where everything is coming together and now we need to know WHAT we can do with the device.

Getting a list of CGI

At this point, we have a working client (for the most part), now we need commands to run! We are going to explore a variety of ways we can get more info on the commands that are available to us.

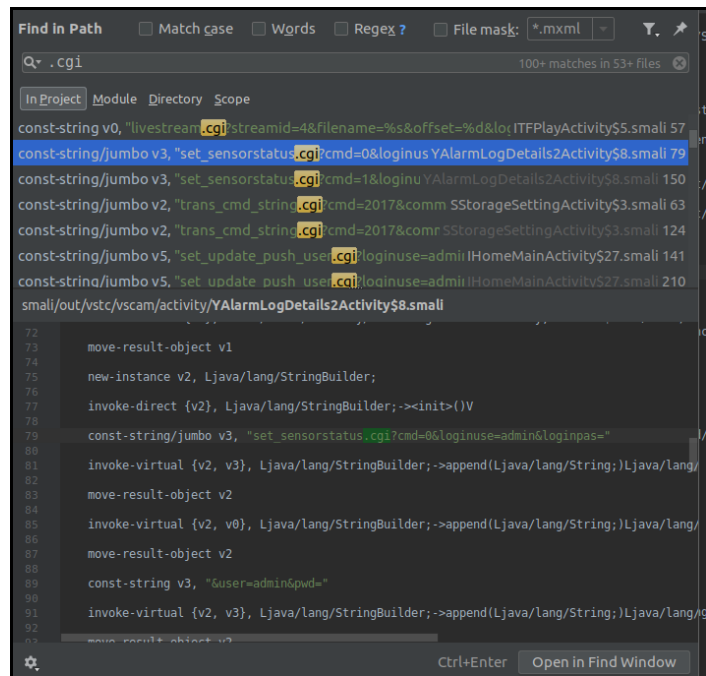
Wireshark and Testing Methodology

We can use Wireshark as a sniffer to get the request strings we want. We just open the APP as usual, and try each and every single option, button, etc. Try to do anything that might generate a GET. Then filter the GET packets out in Wireshark.

Decompiling APK

We can use Android Studio's "Debug and Profile APK" feature to disassemble the APK into Smali. I didn't know anything about Smali before messing around with this project, so pardon me if I get things wrong, I never really learned the language, I just used intuition on most of this.

Using the "Find in Path" feature in Android Studio let's us search the project for any mention of .cgi. While some results aren't going to be exactly as we want, we will get to see a majority of the surface level functions we can use.



If we actually look at some of them we can see some code that provides us with more info.

```
new-instance v1, Ljava/lang/StringBuilder;

invoke-direct {v1}, Ljava/lang/StringBuilder;.<init>()V

const-string/jumbo v2, "set_sensorname.cgi?&sensorid="
invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;.<append>(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v1

invoke-virtual {v1, p2}, Ljava/lang/StringBuilder;.<append>(I)Ljava/lang/StringBuilder;

move-result-object v1

const-string v2, "&sensorid0="
invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;.<append>(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v1

invoke-virtual {v1, p3}, Ljava/lang/StringBuilder;.<append>(I)Ljava/lang/StringBuilder;

move-result-object v1

const-string v2, "&sensorid1="
```

I don't need to really know smali to know what I'm seeing here. We have a java class, `StringBuilder`, building a CGI string to send out. By looking at these segments of code by the CGI strings we have found, we can also mine more information on what arguments some of these commands use.

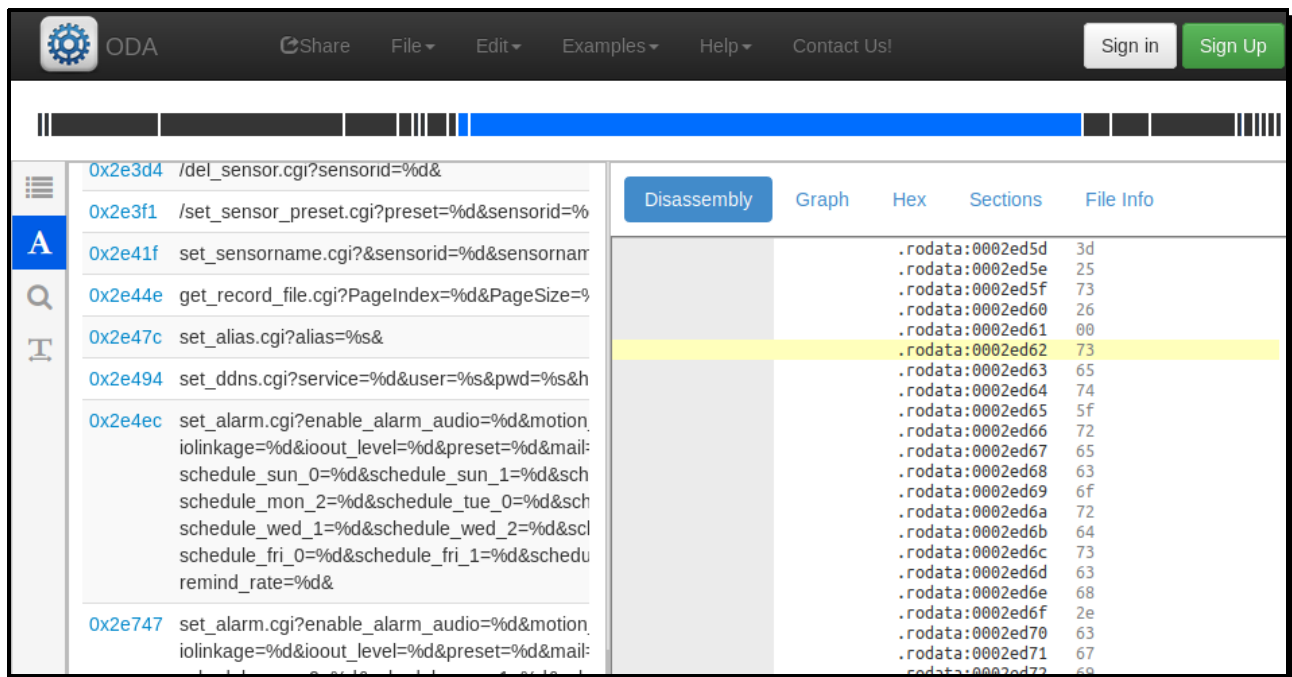
Decompling SO

I noticed after a while an interesting pattern I kept seeing after it made the CGI string, it would call a method:

```
vstc2/nativecaller/NativeCaller;->TransferMessage
```

However, no amount of googling lead me closer to finding out what it did, and worse, I couldn't find the method ANYWHERE in the smali. I must have searched for hours.

Finally when I did find it, it was just a stub method, there was no code to it at all! Then after a bit more googling I learned that methods gained from Shared Object files won't show up in a Smali decompile, since it's only decompiling the Java, not the assembly. When looking through the shared objects, I found one called `libvstc2_jni.so`, which seemed to be what I was looking for. I tried decompiling it on Linux but I couldn't find a good tool to do it with and ended up using `onlinedisassembler` which worked pretty well as I mainly just used it for it's string search, rather than wanting to pour over ARM ASM.



Decompiling Firmware

I'd also like to decompile the firmware on the camera, but unfortunately, I couldn't find a good method to pull a copy off the device. I was, however, able to get a hold of an update, (Finally I've been waiting for months for a firmware update I could listen into.), which doesn't contain any CGI info, but does contain some other interesting stuff I'll detail later.

Googling

Ripping through Shared Objects, Smali, and Wireshark captures is fun and all, but sometimes you just want real human English answers. I spent a lot of time googling things during this writeup, as well as when I first sat down with the device months ago.

Finally after getting so far, I wanted to see if there was ANY information out there about this API, so I set out with a couple choice google searches, and an understanding of what "down the rabbit hole" truly means.

I started with googling "vstarcam api", which lead me to an interesting [SDK page](#). Once there I quickly located a CGI manual, let's open it up and take a look inside.

get_record_file.cgi

描述: 获取录像文件名字

认证: 管理者

语法: 录像文件名字

<部分机型支持>

record_num0: 多少个录像文件

record_name0: 录像文件名字

get_factory_parm.cgi

描述: 获取厂家消息

认证: 管理者

语法: 录像文件名字

factory_server: 厂家域名消息

factory_user: 厂家域名用户名

factory_passwd: 厂家域名密码

factory_heatbeat: 厂家域名心跳包

factory_port: 厂家域名端口

factory_status: 厂家域名状态

媒体流相关 **CGI**

Oh no, the manual is in Chinese, but at least we are getting closer.

I tried "vstarcam sdk" and ended up finding an [english manual](#). I also noticed this [forum post](#), which also talked about the poor security of the VStarCam firmware.

In this post he talks about some juicy things, one of which I discovered in an update I went through.

There was also [another post](#) about the camera, although it mentions it by a different name, unsurprisingly, the picture even is the same one I have. This guy went deep into the nitty gitty on some of the exploits, some of which I'd like to try myself now that I know about them.

Also, after referencing the manual, I did not notice any commands I didn't find already, or see any that weren't listed.

Extras

After studying this thing for months, I FINALLY received a firmware update. I wasted no time sniffing the connection and learning some cool stuff.

40	5.433425727	38.146.78.24	vstarcam	UDP	264 6176 → 10562 Len=220
41	5.434566535	vstarcam	38.146.78.24	UDP	62 10562 → 6176 Len=10
42	5.481981949	vstarcam	38.146.78.24	UDP	62 10562 → 6176 Len=4
43	5.572567054	38.146.78.24	vstarcam	UDP	62 6176 → 10562 Len=4
44	5.661867744	vstarcam	38.146.78.24	UDP	62 10562 → 6176 Len=4

▶ Frame 40: 264 bytes on wire (2112 bits), 264 bytes captured (2112 bits) on interface 0
 ▶ Ethernet II, Src: Buffalo_9d:78:20 (cc:e1:d5:9d:78:20), Dst: B-LinkE1_0b:db:b4 (48:02:2a:0b:db:b4)
 ▶ Internet Protocol Version 4, Src: 38.146.78.24 (38.146.78.24), Dst: vstarcam (192.168.11.140)
 ▶ User Datagram Protocol, Src Port: 6176 (6176), Dst Port: 10562 (10562)
 ▶ Data (220 bytes)
 ▶ VSS-Monitoring ethernet trailer, Source Port: 9224

0000	48 02 2a 0b db b4 cc e1 d5 9d 78 20 08 00 45 00	H:*		x	E
0010	00 f8 00 00 40 00 35 11 04 17 26 92 4e 18 c0 a8	@	5	..&	N
0020	0b 8c 18 20 29 42 00 e4 b0 96 f1 d0 00 d8 d1 00	...)	B	..	
0030	00 0b 01 0a 00 00 cc 00 00 00 47 45 54 20 2f 61		..GET	/a	
0040	75 74 6f 5f 64 6f 77 6e 6c 6f 61 64 5f 66 69 6c	uto_down	load_fil		
0050	65 2e 63 67 69 3f 73 65 72 76 65 72 3d 64 6f 72	e.cgi?se	rver=dor		
0060	61 65 6d 6f 6e 2e 69 70 63 61 6d 2e 73 6f 26 66	aemon.ip	cam.so&		
0070	69 6c 65 3d 2f 66 66 38 39 64 64 36 62 37 36 34	ile=/ff8	9dd6b764		
0080	66 38 39 38 61 39 66 38 30 65 34 30 63 64 63 31	f898a9f8	0e40cdc1		
0090	33 65 32 63 63 26 74 79 70 65 3d 30 26 72 65 73	3e2cc&ty	pe=0&res		
00a0	65 76 65 72 65 64 31 3d 26 72 65 73 65 76 65 72	evered1=	&resever		
00b0	65 64 32 3d 26 72 65 73 65 76 65 72 65 64 33 3d	ed2=&res	evered3=		
00c0	26 72 65 73 65 76 65 72 65 64 34 3d 26 6c 6f 67	&resever	ed4=&log		
00d0	69 6e 75 73 65 3d 61 64 6d 69 6e 26 6c 6f 67 69	inuse=ad	min&logi		
00e0	6e 70 61 73 3d 70 61 73 73 77 6f 72 64 26 75 73	npas=pas	sword&us		
00f0	65 72 3d 61 64 6d 69 6e 26 70 77 64 3d 70 61 73	er=admin	&pwd=pas		
0100	73 77 6f 72 64 26 24 08	sword&	S		

When it started the download, it used the auto_download_file.cgi script. With it we can supply a server, and a file to download. Potentially abusable! We will definitely cover this in a later part.

54	6.750107010	vstarcam	d34ss8m87o4huh.cloudfront.net	TCP	68 41918 → 41918 [ACK] Seq=1 Ack=1 Win=14600
55	6.796366464	vstarcam	d34ss8m87o4huh.cloudfront.net	HTTP	161 GET /ff89dd6b764f898a9f80e40cdc13e2cc HTTP/1.1
56	6.815683155	d34ss8m87o4huh.cloudfront.net	vstarcam	TCP	68 http(80) → 41918 [ACK] Seq=1 Ack=94 Win=29184

Frame 55: 161 bytes on wire (1288 bits), 161 bytes captured (1288 bits) on interface 0
 Ethernet II, Src: B-LinkE1_0b:db:b4 (48:02:2a:0b:db:b4), Dst: Buffalo_9d:78:20 (cc:e1:d5:9d:78:20)
 Internet Protocol Version 4, Src: vstarcam (192.168.11.140), Dst: d34ss8m87o4huh.cloudfront.net (54.230.119.231)
 Transmission Control Protocol, Src Port: 41918 (41918), Dst Port: http (80), Seq: 1, Ack: 1, Len: 93
 Hypertext Transfer Protocol
 VSS-Monitoring ethernet trailer, Source Port: 28815

000	cc e1 d5 9d 78 20 48 02 2a 0b db b4 08 00 45 00	x	H	*	E
010	00 91 3f ed 40 00 40 06 7f 78 c0 a8 0b 8c 36 e6	..?.	@	..	x.....6
020	77 e7 a3 be 00 50 01 90 03 43 2a ae 4a 89 80 18	w....P	..	C	*.J....
030	1c 84 b6 07 00 00 01 01 08 0a 00 4f 3c ab 37 ee		..0	<	7
040	3e 1d 47 45 54 20 2f 66 66 38 39 64 64 36 62 37	>.	GET	/f	f89dd6b7
050	36 34 66 38 39 38 61 39 66 38 30 65 34 30 63 64	64f898a9	f80e40cd		
060	63 31 33 65 32 63 63 20 48 54 54 50 2f 31 2e 31	c13e2cc	HTTP/1.1		
070	0d 0a 48 6f 73 74 3a 20 64 6f 72 61 65 6d 6f 6e	..Host:	doraemon		
080	2e 69 70 63 61 6d 2e 73 6f 0d 0a 43 6f 6e 6e 65	.ipcam.s	o	Conne	
090	63 74 69 6f 6e 3a 43 6c 6f 73 65 0d 0a 0d 0a 70	ction:Cl	ose	p	
0a0	8f				

I immediately grabbed a copy of the file and went to town in binwalk, extracting all files so I could take a look. One of the first files I noticed, ipcam.sh, had an interesting thing inside it. This is the file verbatim.

```

export PATH=/system/system/bin:$PATH
#telnetd
export LD_LIBRARY_PATH=/system/system/lib:/mnt/lib:$LD_LIBRARY_PATH
mount -t tmpfs none /tmp -o size=3m

/system/system/bin/brushFlash
/system/system/bin/updata
/system/system/bin/wifidaemon &
/system/system/bin/upgrade &

```

When I saw this I thought, "Oh man, someone left telnetd on and had to turn it off in a later update."

After downloading previous updates from caches online, I found that this was indeed true. It is also backed up by [this article](#), specifically in the section "CVE-2017-8224 - Backdoor account".

I thought that was super funny.

Another interesting find was some of the developer names were accidentally left in some binaries, by way of a home directory listing.

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	ELF, 32-bit LSB executable, ARM, version 1 (SYSV)
17271	0x4377	Unix path: /home/dengyibao/share/sdk/3518e_a0/Hi3518_SDK_V1.0.A.0//mpp2/lib/libmpi.so
1247608	0x130978	HTML document header

```
lan@gaming-computer:~/Documents/crystal/vstarcam-c7824wip-investigation/software/_CH-sys-48.53.64.60.bin.extracted$ cat /tmp/encoder
```

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	ELF, 32-bit LSB executable, ARM, version 1 (SYSV)
11798	0x2E16	Unix path: /home/laishaojun/share/hisi/Hi3518_SDK_V1.0.8.1//mpp/lib/libmpi.so
657504	0xA0860	HTML document header
658144	0xA0A00	Unix path: /home/laishaojun/share/hisi/Hi3518_SDK_V1.0.8.1//mpp/lib/libmpi.so

I also found another interesting tidbit, the password they use to zip is hardcoded into binaries added in some of the updates.

```
lan@gaming-computer:~/Documents/crystal/vstarcam-c7824wip-investigation/software/_CH-sys-48.53.64.60.bin.extracted/system/system/bin$ cat /tmp/wifidaemon
```

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	ELF, 32-bit LSB executable, ARM, version 1 (SYSV)
58379	0xE40B	Unix path: /lib/modules/ko/wdt.ko
50568	0xEC98	Unix path: /system/system/bin/unzip1 -o -P vstarcam!@#\$% /tmp/www.zip -d /system > /tmp/app.txt
51500	0xF03C	Unix path: /sys/class/net/ra0
54087	0xFA57	Unix path: /lib/modules/wifi/mtutil7601Usta.ko
54968	0xFDC8	Unix path: /etc/Wireless/RT2870STA/RT2870STA.dat
57760	0x108B0	Unix path: /system/system/bin/sysversion.txt

```
lan@gaming-computer:~/Documents/crystal/vstarcam-c7824wip-investigation/software/_CH-sys-48.53.64.60.bin.extracted/system/system/bin$
```

TAGGED SKILLS:

- 🛠️ Exploitation
- 🔭 Exploit PoC Development
- 🔒 Security Research
- 📄 C