WERE 900K DEUTSCHE TELEKOM ROUTERS COMPROMISED BY MIRAI? (HTTPS://COMSECURIS.COM/BLOG/POSTS/WERE_900K_DEUTSCHE_TE

Tue, Nov 29, 2016 *Ralf*

reversing (https://comsecuris.com/blog/tags/reversing), exploitation (https://comsecuris.com/blog/tags/exploitation)

In the last 36 hours, news of a “cyber attack” against Deutsche Telekom DSL routers has been making headlines in German media. Customers have been asked to restart their devices to receive firmware updates, but little information on the actual cause has been made available, which has led to rumours and speculation.

The dominant theory proposed thus far was that a strain of the Mirai botnet family was responsible for the outage (https://isc.sans.edu/forums/diary/Port+7547+SOAP+Remote+Code+Execution+Attack+Against+DSL+Modems/21759) . What was unclear however is whether the affected DTAG home routers were compromised due to a command injection vulnerability recently published in the TR-064/TR-069 parsing (https://devicereversing.wordpress.com/2016/11/07/eirs-d1000-modem-is-wide-open-to-being-hacked/). In this blog post we explain why this attack was a denial of service attack rather than a compromise of the affected devices.

First, we obtained device firmware for the affected devices. Interestingly, all of them weighed on the order of approximately 4 Megabytes, much smaller than firmware updates for other SpeedPort routers. Furthermore, Binwalk wasn’t able to extract anything from them. Apparently, all of the devices affected have been manufactured by Arcadyan, an Taiwanese OEM of DSL routers with a rather large market share.

Arcadyan firmware files are (http://tecnicaquilmes.fullblog.com.ar/analizando-el-livebox-21-de-orange.html) commonly (https://sviehb.wordpress.com/2011/09/06/reverse-engineering-an-obfuscated-firmware-image-e01-unpacking/) obfuscated (http://www.devttys0.com/2014/02/reversing-the-wrt120n-firmware-obfuscation/) to prevent reverse-engineering efforts. None of the past deobfuscation techniques worked on the W921V we had settled on trying to analyze however.

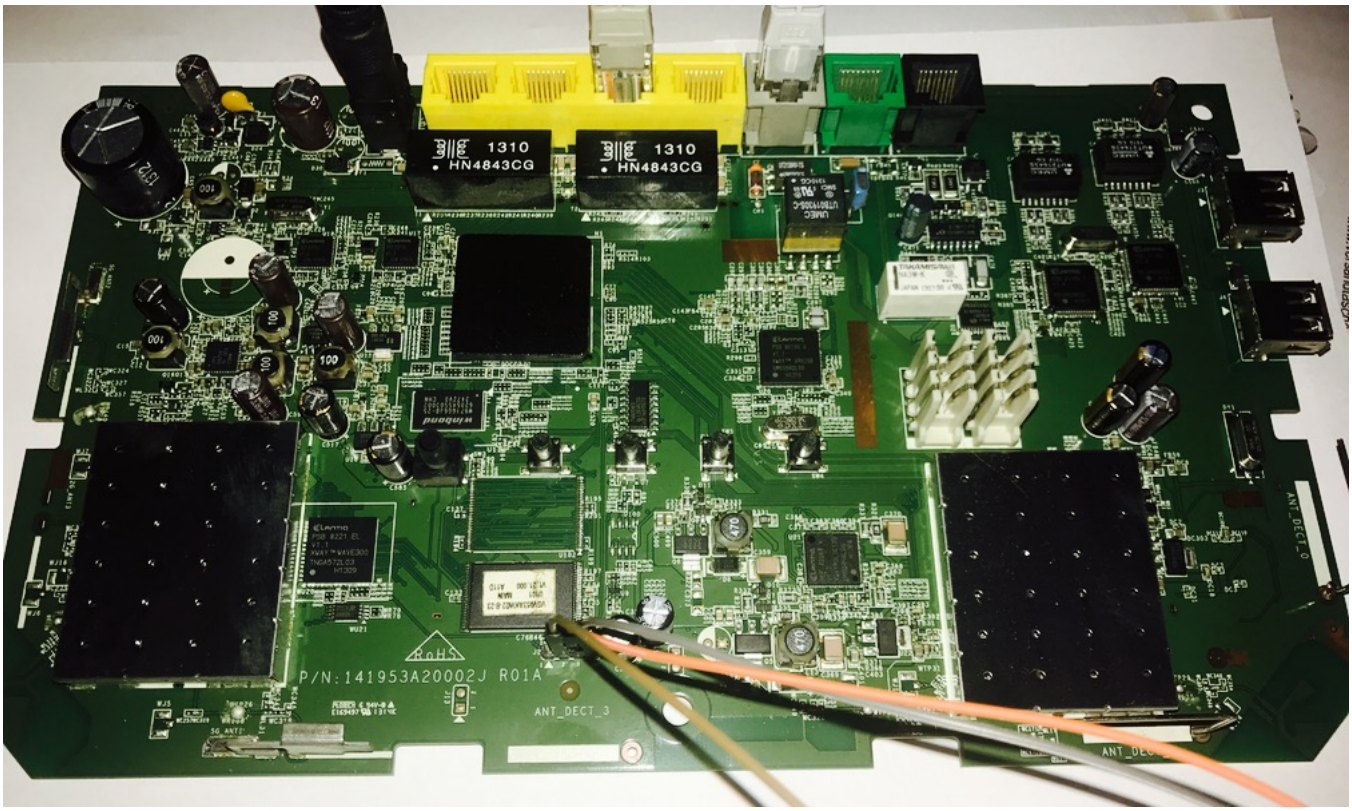
Looking further, we found a tool to extract DSL and ISDN firmware (https://github.com/openwrt/openwrt/tree/barrier_breaker/package/kernel/lantiq/ltq-vdsl-fw/src) in an OpenWRT repository and a thread from 2011 (http://www.ip-phone-forum.de/showthread.php?t=235678&page=11) on how to extract W921V firmware in which someone provided a tool for extracting the operating system image from said firmware.

While this did not work on current firmware versions, we saw in older firmware versions that this device is running a small Real-Time Operating System (RTOS) instead of a fully-fledged operating system such as Linux, undermining the theory that a command injection vulnerability would work against them. Indeed, this RTOS, which is rumoured to be called SuperTask due to the string

```
    ================ DUMP SuperTask Kernel State ============================
```

showing up in its binary, does neither have any shell to speak of nor does it have a wget binary.

Meanwhile having obtained one of the vulnerable router models, a W921V running firmware version 1.39.0000 we attached it to a Deutsche Telekom DSL line for testing. In the same IP Phone Forum thread given above, we saw that a serial console is available, so we opened the router and connected a UART cable to the 4 PIN header next to the unpopulated ANT_DECT_J.



The bootup log eliminated any doubt that this device was running Linux:

```
ROM VER: 1.1.4
CFG 06
NAND
NAND Read OK
[20] DDR Access auto data-eye tuning Rev 0.3a
DDR check ok... start booting...



========================================================================
Wireless ADSL IAD VR9 Loader v1.11.02 build Feb 20 2012 14:41:59
                    Arcadyan Technology Corporation
========================================================================
A2x VR9,
disable dcdc !!!
0xbf106a10 : 78
0xbf106a11 : 0
0xbf106a0c : 88
ST Micro NAND device: 32M
Searching primary BBT ... BBT found at page 65504
Searching mirror  BBT ... BBT found at page 65472
flash has 0 bad blocks

Copying boot params.....DONE

Get Primary to 1.....
Unzipping firmware at 0x80002000 ... with AREA[3][ZIP 1]  done
Ready to run firmware
STMicro 32MB 3,3V 8-bit found
----------------------------
NAND page size    : 512
NAND oob size     : 16
NAND block size   : 16384
NAND page/block   : 32
NAND block/chip   : 2048
NAND chip size    : 0x2000000
NAND page shift   : 9
NAND page mask    : 0xFFFF
NAND block shift  : 14
----------------------------

Scan BAD Block ...
0 Bad Block(s)

1
In c_entry() function ...
install_exception
Co config = 80048483
sys_irq_init ...
IrqConnect : IRQ = 53, Handler = 0x8001CFC0
VR9 is A21 chip !!!!, ifx_bsp_basic_mps_decrypt bf001f38
##### _ftext      = 0x80002000
##### _fdata      = 0x80C52D30
##### __bss_start = 0x80D95A70
##### end         = 0x8725ADC4
allocate_memory_after_end: alloc from 87262DD0 to 873A5B20, length=1322304
##### Backup Data from 0x80C52D30 to 0x87262DC4~0x873A5B04 len 1322304
##### Backup Data completed
##### Backup Data verified
[GPIO FLOW] SetGpio() Begin ..
BF103008 : bba67017
BF103004 : b01f23
BF10734C :CHIP LOC 71e13
BF107354 :ID0 8420019b
BF107358 :ID1 c01d70
LOT :ID 0x1d708420019b -->7232480627
PLL1 locked..fails 0.
ifx_gpio_init() !!!
ifx_gpio_pre_init() !!!
[KERN_INFO]IFX GPIO driver, version 1.2.10, (c)2009 Infineon Technologies AG
Register LED MODULE OK!!
Register BUTTON MODULE OK!!
[GPIO FLOW] SetGpio() End.
[INIT] System Log Pool startup ...
[INIT] MTinitialize ..
CPU Clock 500000000 Hz
mips_counter_frequency:250000000
r4k_offset: 0x0003d090(250000)
init_US_counter : time1 = 28 , time2 = 225065, diff 225037
US_counter = 112
set to constant US_counter = 112
 cnt1 2040003 cnt2 2042560, diff 2557
 cnt1 2886360 cnt2 2888164, diff 1804
[BOOT] : ifx_nand_init() ...
STMicro 32MB 3,3V 8-bit found
----------------------------
```

```
NAND page size    : 512
NAND oob size     : 16
NAND block size   : 16384
NAND page/block   : 32
NAND block/chip   : 2048
NAND chip size    : 0x2000000
NAND page shift   : 9
NAND page mask    : 0xFFFF
NAND block shift  : 14
----------------------------
Scan BAD Block ...
0 Bad Block(s)

[BOOT] : ifx_nand_init() ok
Runtime code version: 1.39.000
System startup...
[INIT] Memory COLOR 0, 8242880 bytes ..
[INIT] Memory COLOR 1, 2097152 bytes ..
[INIT] Memory COLOR 2, 9712304 bytes ..
InitCommSys: RESOURCE_BASE = 133, NUMRES = 640
InitCommSys: EVENT_BASE = 335, NUMEVT = 818
InitCommSys: MAILBOX_BASE = 6, NUMMBX = 64

rzMemory start: 0x82DA68FC, end 0x82FC6034, size 2225976
Build Day = Jul 13 2015
>>>>set_MP_pass_Magic to 4
Set flash memory layout to FL2MacAddr=84:9C:A6:F4:58:3A
CMP_MACADDR=84:9C:A6:F4:58:39
WLAN_MAC=84:9C:A6:F4:58:3D
Boot Parameters found !!!
Use Flash Data as COSIC Parameters
>>>>>>>>>>>>>>>>> HWOption set BMC Param : ucOscTrimValLOW 2f
>>>>>>>>>>>>>>>>> HWOption set BMC Param : g_ucOscTrimValHigh 0
Bootcode version: v1.11.02
Serial number: J316105185
Hardware version: 01A
WPA Key is from Boot Parameter 2882331268984712
pBootParams->password 34 36 39 (20)
Entered default password checking 8 8
fread_config_b> sizeof(struct III_Config_t) is 545848
fread_config_b> 1048576 bytes read from flash
fread_config_b 17854> nLen=8191988
Scz_Decompress_Buffer2Mem 935> *M=8191988
decomp_buf2buf_impl> buffer size *M=8191988
Decompression ratio = 0.0
fread_config_b> 631192 bytes decompressed successfully. 859e15e4

my CFGVersionMagic = 33343536
my CFGsize = 545856, my CFGDescSize = 46919
my Version = 1.39.000
My CfgHexSize:15492
My CFGDescSize:46919

old CFGVersionMagic on flash = 33343536
CFGsize on flash = 545856, CFGDescSize on flash = 46919
Version on flash = 1.39.000
===
===
Old CfgHexSize = 15492, Old CFGDescSize = 46919
Unzipping from 8267F05C to 82673904 ... unzip_address sig 5d000080 pre, zipfile=0x8267f05c, unzipfile=0x82673904
unzip_address sig 8000005d end
[ZIP 1] allocate_memory_after_end: alloc from 873A5B20 to 873B99A0, length=81516
[ZIP 1 49492c30] done
Uncompressed size = 46917
Tail1 : END_III_Config_t

Size of Old CFG_DESC is :46917!!!
1.useCfgDesc:1
Unzipping from 8266FC68 to 82664510 ... unzip_address sig 5d000080 pre, zipfile=0x8266fc68, unzipfile=0x82664510
unzip_address sig 8000005d end
[ZIP 1] [ZIP 1 49492c30] done
Uncompressed size = 46917
2.useCfgDesc:1
Tail : END_III_Config_t

useCfgDesc:3

Restore Config file from ver: 1.39.000 !!!

to copy from gConfig to gSetting in restore_config
DS_Tail:[t_VGV953AKW]
RELEASE_BUF_AND_RETURN_OK     - restore_config_new() #17747, in vr9/config.c
     - restore_config_new() #17748, in vr9/config.c
restore config OK. nCfgDescLen=561356.
----------------------> Checking SysLog in
                                        Enable UART TX
Turn off COSIC function now...31000 0
```

```
Disable UART TX
Enable UART TX
usb_explore_task() COSIC is ready. time=31891
Disable UART TX
```

The question however remained: What caused the outage? Having ruled out the command injection we were wondering wether the requests to port 7547 sent out by the botnet were crashing the devices. Since DTAG was already blocking port tcp/7547 inbound, we had to simulate the attack from an internal interface. Connections to 192.168.2.1 (the default SpeedPort IP) were blocked, connecting to the IP of the outbound WAN interface from the internal network worked fine however. So we sent the request we had intercepted on port 7547 on another host to the device:

```
$ (cat ~/request.tr069.capture.bin;sleep 10)|nc -v 87.161.192.22 7547
Connection to 87.161.192.22 port 7547 [tcp/cwmp] succeeded!
$
```

Nothing happened – the connection simply stayed open for 10 seconds with no response (the Mirai worm uses a similar timeout). Nothing appeared in the system logs or on the serial console either.

Changing the line endings to DOS line endings allowed us to elicit a rather detailed response from the device:

```
$ (cat ~/request.tr069.capture.dos.bin;sleep 10)|nc -v 87.161.192.22 7547
HTTP/1.1 500 Internal Server Error
Content-Type: text/xml; charset="utf-8"
Content-Length: 419
Date: Tue, 29 Nov 2016 17:32:03 GMT
Server: DTW921V UPnP/1.0 Speedport W 921V 1.39.000
EXT:

<?xml version="1.0"?>
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"><s:
Body><s:Fault><faultcode>s:Client</faultcode><faultstring>UPnPError</faultstring><detail><UPnPError xmlns="urn:schemas-upnp-or
g:control-1-0"><errorCode>504</errorCode><errorDescription>SSL needed</errorDescription></UPnPError></detail></s:Fault></s:Body
></s:Envelope>
```

However, this did not produce any visible indication of a crash either. As we found other vulnerabilities in the process of investigating the above, we do have have some knowledge about how the device behaves in crash scenarios. We are in the process of communicating these issues to DTAG and will share further details potentially in the future.

So what might have happened then? Interestingly, simply flooding the router with the above requests (TCP/UDP) did show however that the device first becomes much slower in accepting TCP connections on the external IP. Finally, it stopped accepting TCP connections on that interface altogether. This aligns with DTAG customer reports that indicate that the issue temporarily went away by restarting the device. Due to the ongoing heavy Mirai botnet activity, the faulty behavior quickly reoccurred in practice.

We conclude that this phenomena is what caused the outage for some DTAG customers this weekend. From our point of view this means that various news outlets definitely released reports based on preliminary investigation information that was wrong.

Does this mean that DTAG customers are safe? Not necessarily. As mentioned, we did find other vulnerabilities along the process of this investigation. This is not surprising based on the general state of end user router equipment, which is an ongoing pain point since years. Moreover, as several others have pointed out as well, configuration protocols such as TR-069/TR-064 bear their own potential for abuse.

During our testing we found that firmwares are being aggressively updated by DTAG – even when the EasySupport option that supposedly causes automatic updates was disabled, causing us to periodically downgrade.

← Reverse Engineering and Exploiting Samsung's Shannon Baseband (https://comsecuris.com/blog/posts/shannon/)