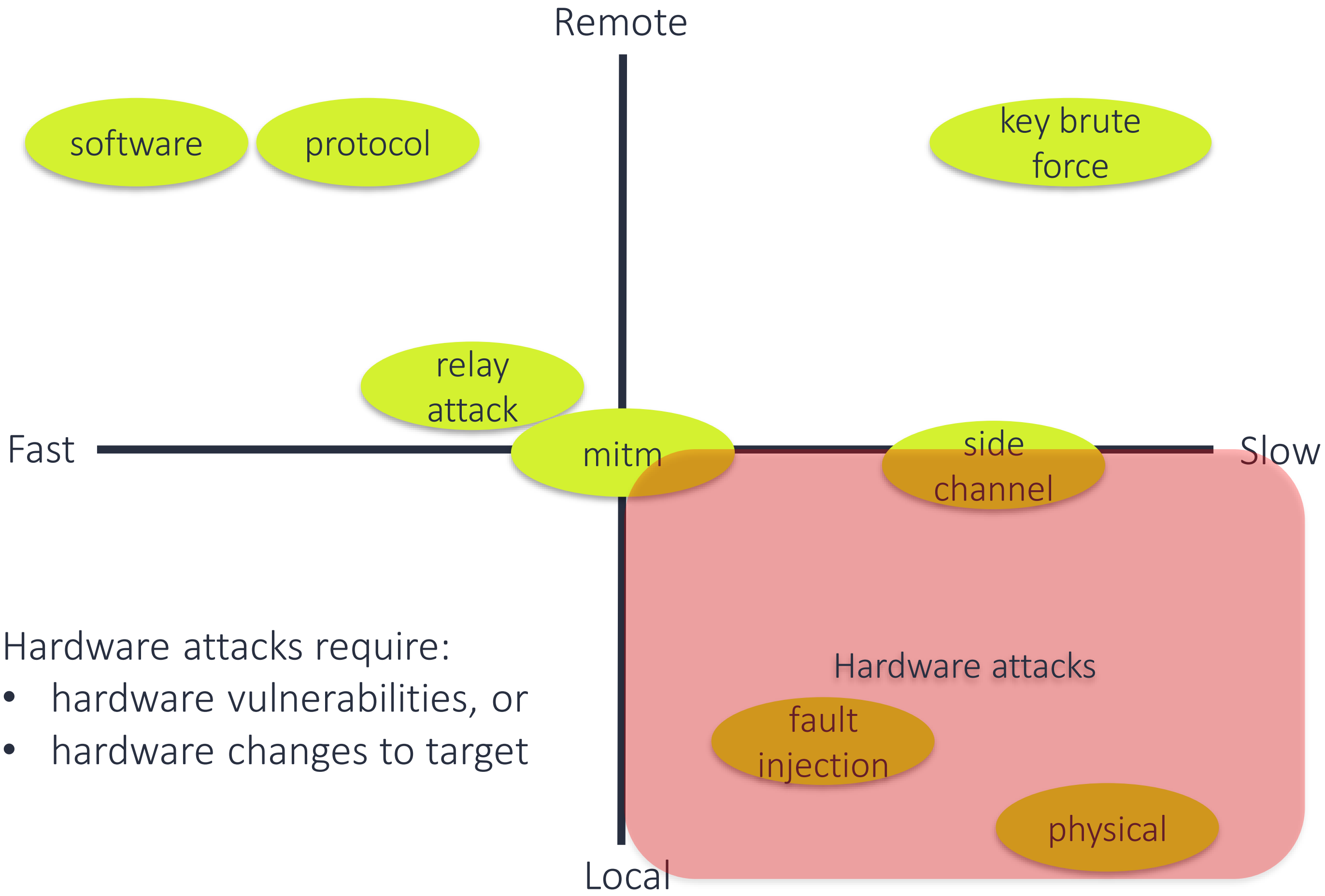
riscure

When Hardware Attacks scalE

Hardwear.io
27 September 2019

Attack exploitation space: time vs distance



Hardware attacks require:

- hardware vulnerabilities, or
- hardware changes to target

Attacker business case

$$p = n * (v - c_v) - c_f$$

p = profit

v = value

n = replications

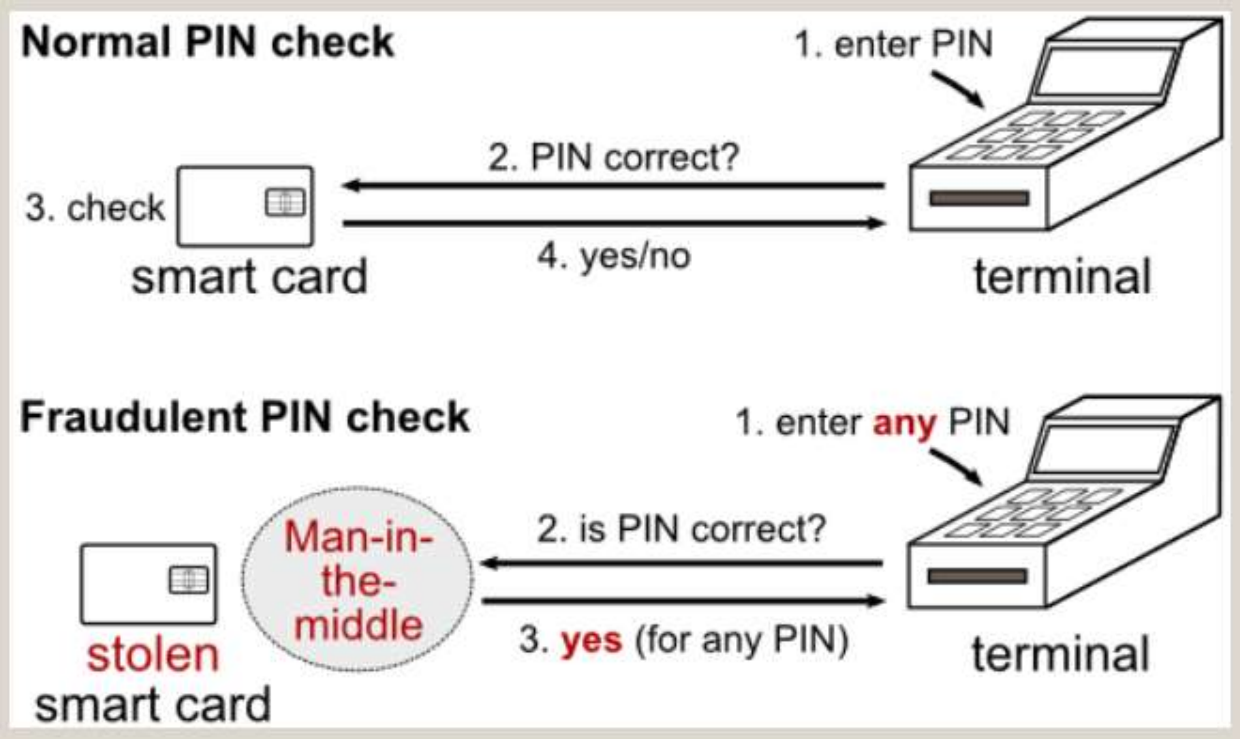
c_v = variable costs

c_f = fixed costs

Let’s analyze some known attacks

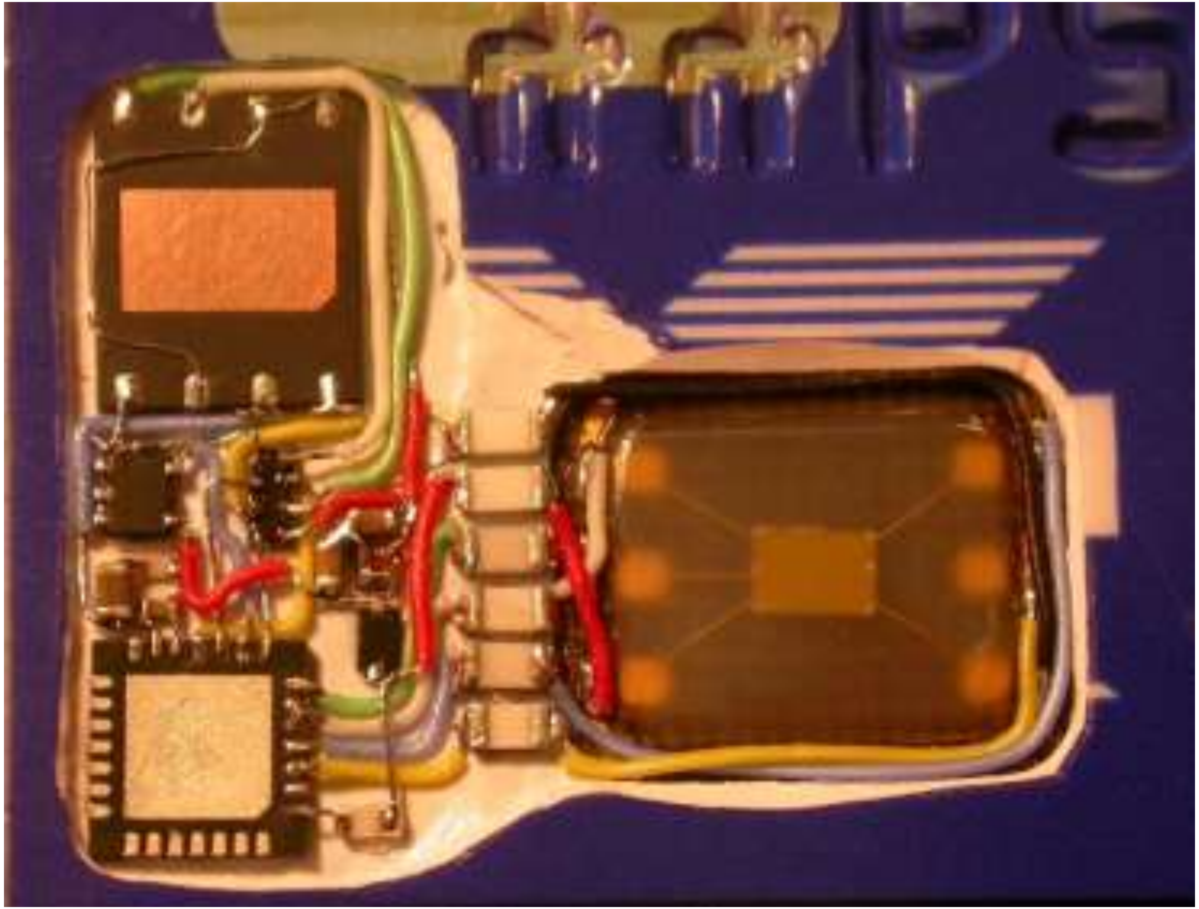
1. EMV Man-in-the-Middle
   Hardware attack to bypass PIN verification of stolen payment cards
2. Retail hack
   Network penetration attack to retrieve cardholder credentials
3. Card sharing
   Relay attack to avoid paying TV subscription fees

EMV Man-in-the-Middle (1)

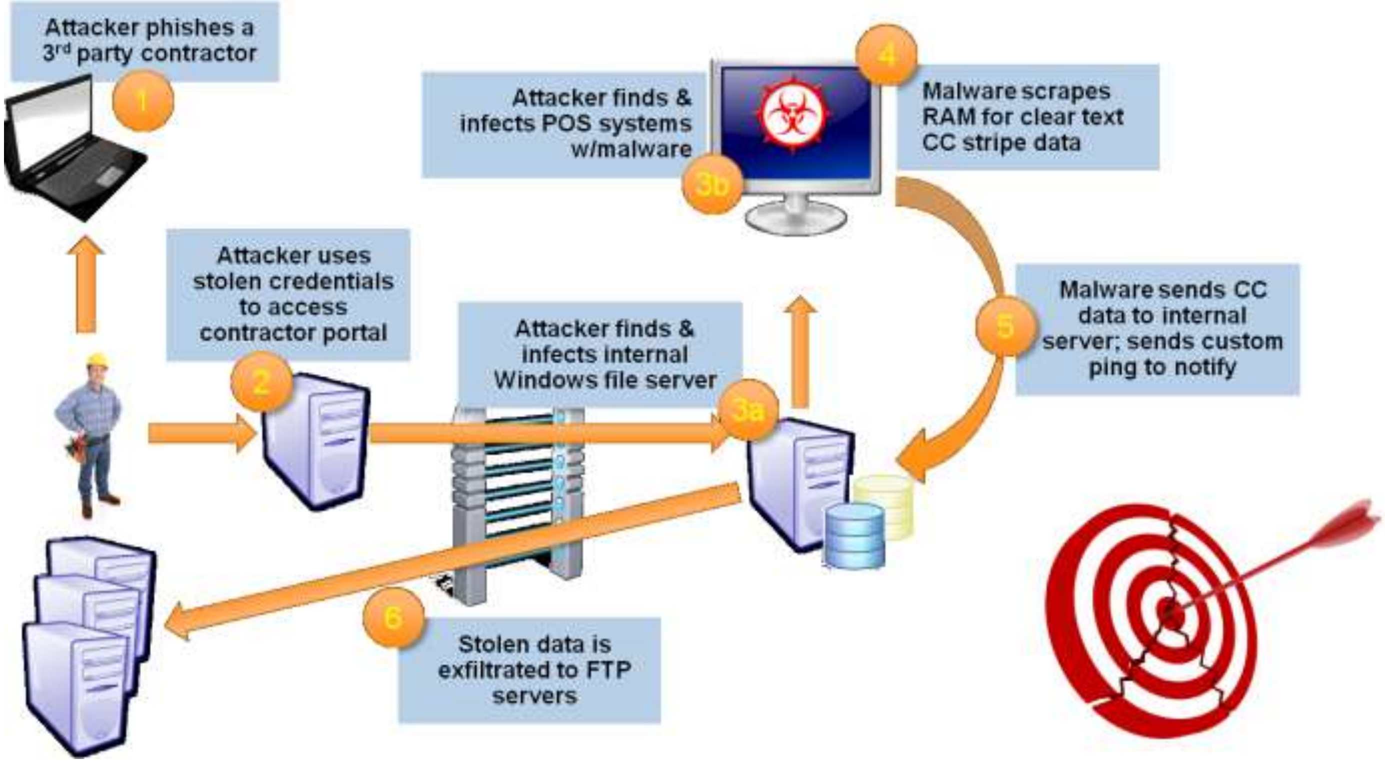


Source: https://www.cl.cam.ac.uk/research/security/banking/nopin/

EMV Man-in-the-Middle (2)



Retail hack



Attacker phishes a 3rd party contractor

Attacker uses stolen credentials to access contractor portal

Attacker finds & infects internal Windows file server

Attacker finds & infects POS systems w/malware

Malware scrapes RAM for clear text CC stripe data

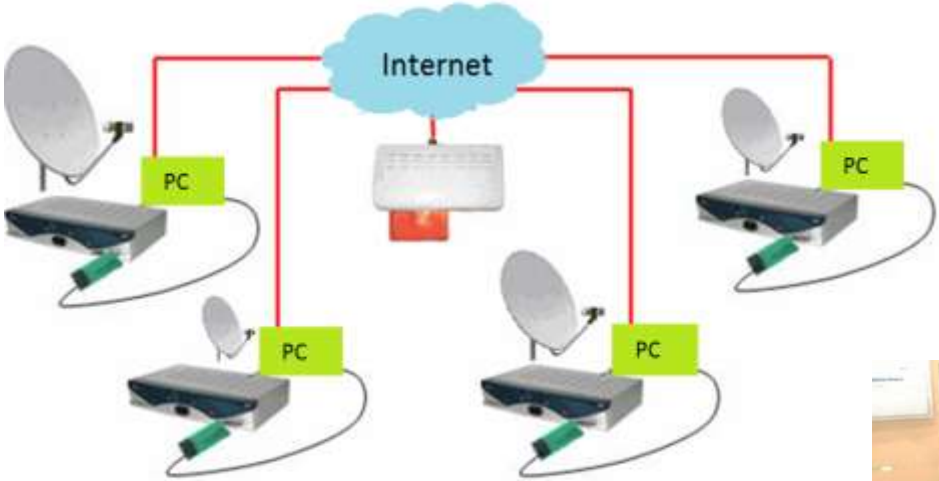
Malware sends CC data to internal server; sends custom ping to notify

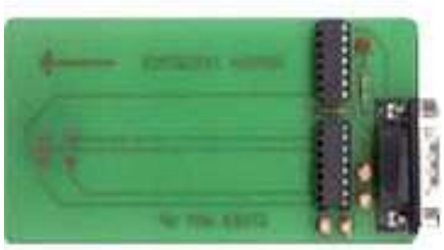
Stolen data is exfiltrated to FTP servers

Card sharing (1)

- Pay-TV decoders use smart cards to control video access
- Subscription is in smart card

Card sharing (2)





- Pay-TV decoders use smart cards to control video access
- Subscription is in smart card
- Distribution of session keys avoids need for individual subscriptions

Example attack business cases

Attack	Fixed Cost	Variable Cost	Value	Replications	Profit
EMV MitM	€ 30K	€ 100	€ 500	100	€ 10 K
Retail hack	€ 20K	€ 1	€ 25	10K	€ 220 K
Card sharing	€ 10K	€ 10	€ 100	1M	€ 90 M

Replications are key, but how is that bounded?

- Application size (e.g. #potential victims)
- Detection & mitigation
- Replication effort

To determine scalability, we need to quantify the replication effort

Attack phases and cost

What parameters determine the attack cost?

	Identification	Exploitation
What it is	**finding a vulnerability**	**replicate on target**
Frequency	once	repeated
Speed	How fast can we do this?	
Skill	Required knowledge / experience	
Equipment	Type of equipment	
Location	Where is the attacker?	

Fixed cost

Variable cost

Attack parameters

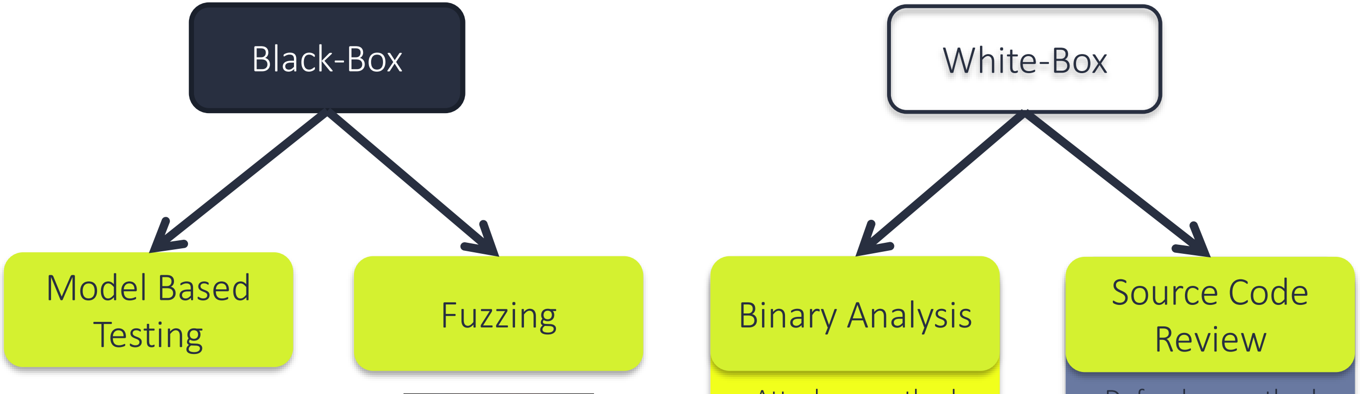
What are typical attack parameters?

	Identification		Exploitation	
Vulnerability	**Hardware**	**Software**	**Hardware**	**Software**
Speed	slow	slow	slow	fast
Skill	expert	expert	proficient	layman
Equipment	specialized	standard	specialized	none
Location	local	near	local	remote

Scalable attacks need software exploitation!

Scalable attack

How to find software vulnerabilities?



Black-Box

White-Box

Model Based Testing

Fuzzing

Binary Analysis

Source Code Review

Attackers method

Defenders method

Effectiveness

Most vulnerabilities are found white-box style!

Finding vulnerabilities in source code

Software packages typically

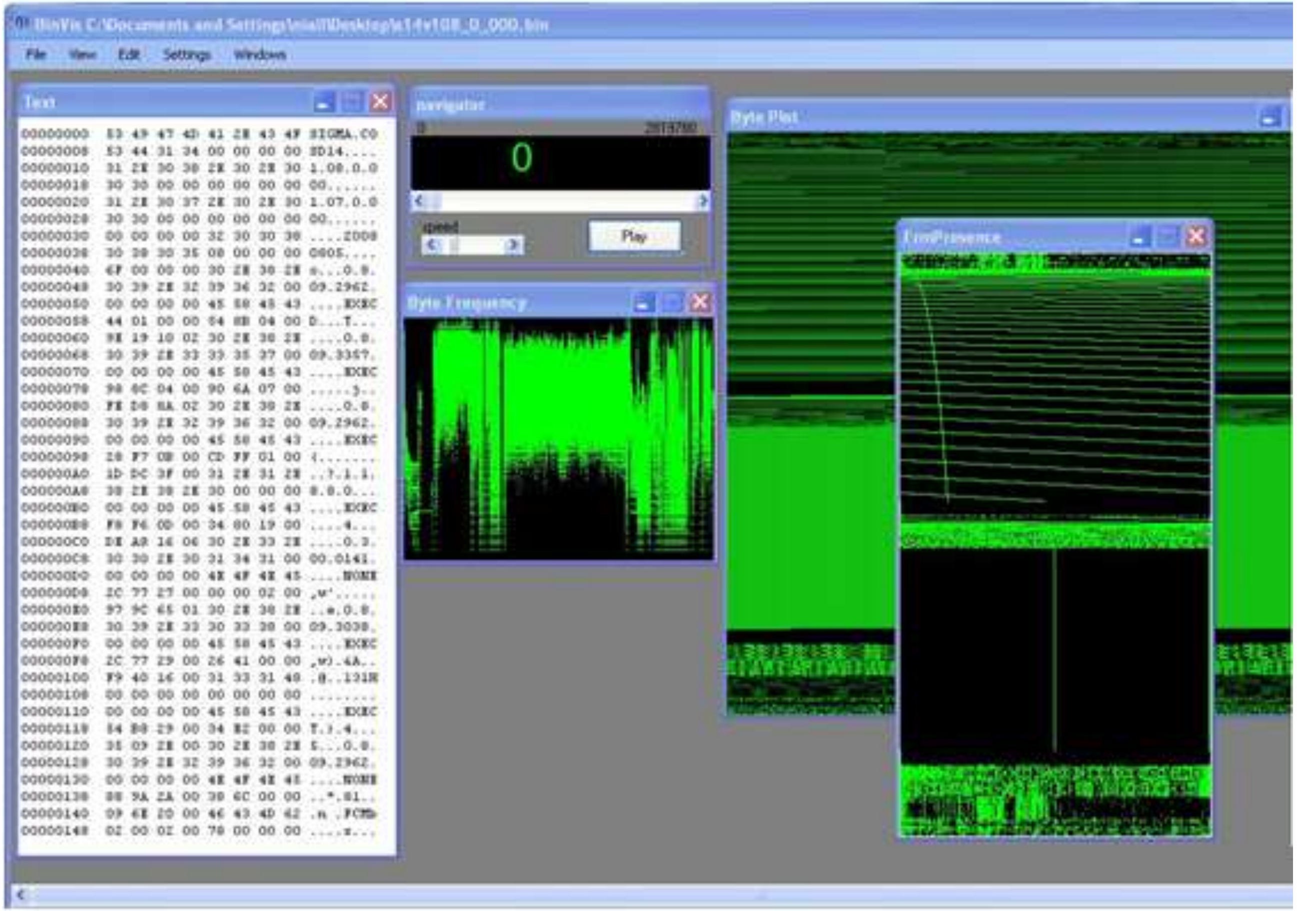
- vary between 10 and 10,000 KLoC
- have 0.1 up to 10 vulnerabilities per KLoC

→ **All products have software vulnerabilities**

Manual source code review performs at 100 LoC/hr

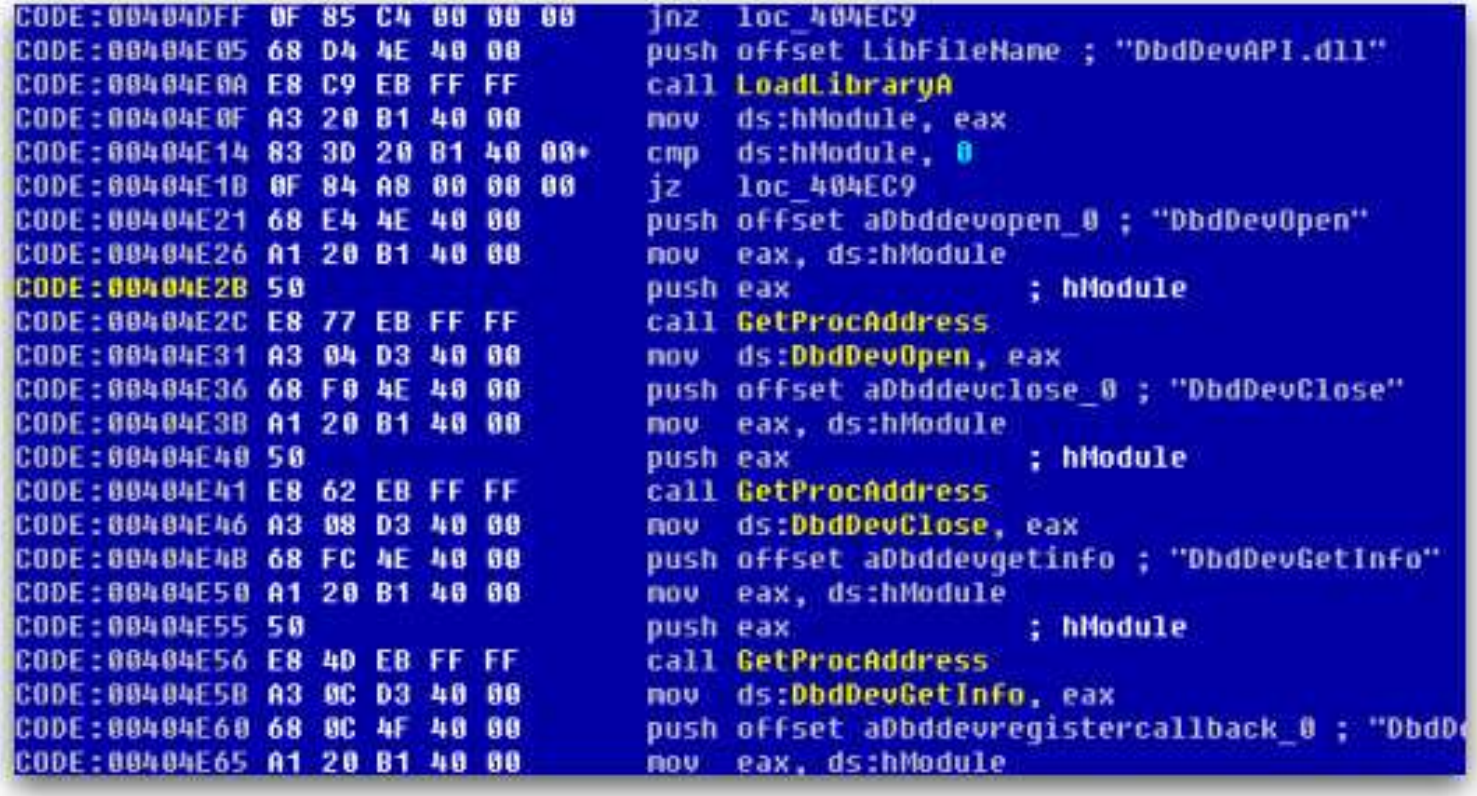
→ **Finding a vulnerability in source code may take just one day**

Binary analysis



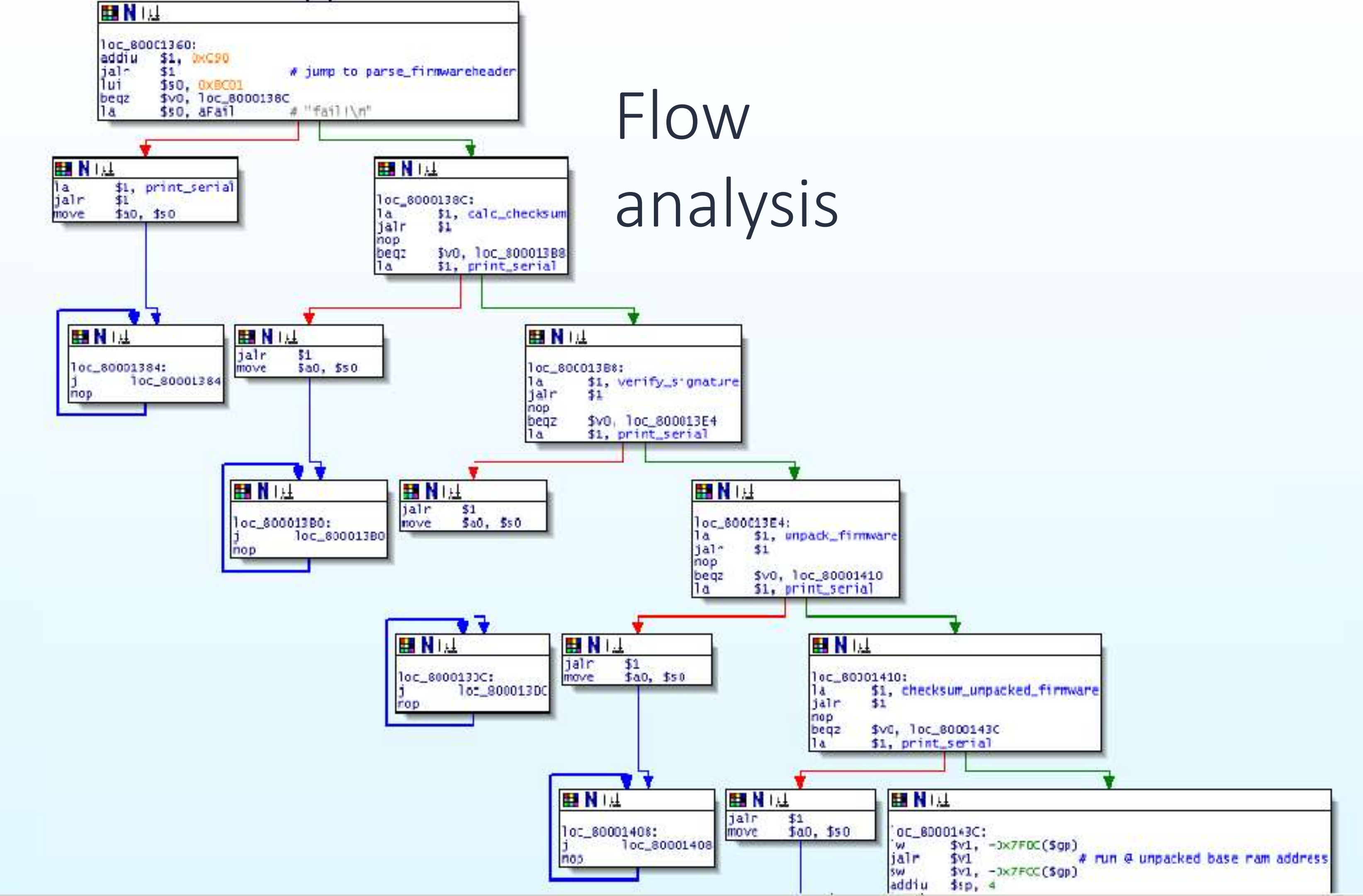
```
00000000  53 49 47 4D 41 2E 43 4F SIGMA.CO
00000008  53 44 31 34 00 00 00 00 SD14....
00000010  31 2E 30 38 2E 30 2E 30 1.08.0.0
00000018  30 30 00 00 00 00 00 00 00......
00000020  31 2E 30 37 2E 30 2E 30 1.07.0.0
00000028  30 30 00 00 00 00 00 00 00......
00000030  00 00 00 00 32 30 30 38 ....2008
00000038  30 38 30 35 00 00 00 00 0805....
00000040  6F 00 00 00 30 2E 30 2E o...0.0.
00000048  30 39 2E 32 39 36 32 00 09.2962.
00000050  00 00 00 00 45 58 45 43 ....EXEC
00000058  44 01 00 00 54 8B 04 00 D...T...
00000060  9E 19 10 02 30 2E 30 2E ....0.0.
00000068  30 39 2E 33 33 35 37 00 09.3357.
00000070  00 00 00 00 45 58 45 43 ....EXEC
00000078  98 6C 04 00 90 6A 07 00 .l...j..
00000080  FE D8 8A 02 30 2E 30 2E ....0.0.
00000088  30 39 2E 32 39 36 32 00 09.2962.
00000090  00 00 00 00 45 58 45 43 ....EXEC
00000098  28 F7 0B 00 CD FF 01 00 (.......
000000A0  1D DC 3F 00 31 2E 31 2E ..?.1.1.
000000A8  38 2E 38 2E 30 00 00 00 8.8.0...
000000B0  00 00 00 00 45 58 45 43 ....EXEC
000000B8  F8 F6 0D 00 34 80 19 00 ....4...
000000C0  DE A8 16 06 30 2E 33 2E ....0.3.
000000C8  30 30 2E 30 31 34 31 00 00.0141.
000000D0  00 00 00 00 4E 4F 4E 45 ....NONE
000000D8  2C 77 27 00 00 00 02 00 ,w'.....
000000E0  97 9C 65 01 30 2E 30 2E ..e.0.0.
000000E8  30 39 2E 33 30 33 38 00 09.3038.
000000F0  00 00 00 00 45 58 45 43 ....EXEC
000000F8  2C 77 29 00 26 41 00 00 ,w).&A..
00000100  F9 40 16 00 31 33 31 48 .@..131H
00000108  00 00 00 00 00 00 00 00 ........
00000110  00 00 00 00 45 58 45 43 ....EXEC
00000118  54 B8 29 00 34 E2 00 00 T.).4...
00000120  35 09 2E 00 30 2E 30 2E 5...0.0.
00000128  30 39 2E 32 39 36 32 00 09.2962.
00000130  00 00 00 00 4E 4F 4E 45 ....NONE
00000138  88 9A 2A 00 30 6C 00 00 ..*.0l..
00000140  09 6E 20 00 46 43 4D 62 .n .FCMb
00000148  02 00 02 00 78 00 00 00 ....x...
```

Disassemble



```
CODE:00404DFF 0F 85 C4 00 00 00    jnz  loc_404EC9
CODE:00404E05 68 D4 4E 40 00       push offset LibFileName ; "DbdDevAPI.dll"
CODE:00404E0A E8 C9 EB FF FF       call LoadLibraryA
CODE:00404E0F A3 20 B1 40 00       mov  ds:hModule, eax
CODE:00404E14 83 3D 20 B1 40 00+   cmp  ds:hModule, 0
CODE:00404E1B 0F 84 A8 00 00 00    jz   loc_404EC9
CODE:00404E21 68 E4 4E 40 00       push offset aDbddevopen_0 ; "DbdDevOpen"
CODE:00404E26 A1 20 B1 40 00       mov  eax, ds:hModule
CODE:00404E2B 50                   push eax             ; hModule
CODE:00404E2C E8 77 EB FF FF       call GetProcAddress
CODE:00404E31 A3 04 D3 40 00       mov  ds:DbdDevOpen, eax
CODE:00404E36 68 F0 4E 40 00       push offset aDbddevclose_0 ; "DbdDevClose"
CODE:00404E3B A1 20 B1 40 00       mov  eax, ds:hModule
CODE:00404E40 50                   push eax             ; hModule
CODE:00404E41 E8 62 EB FF FF       call GetProcAddress
CODE:00404E46 A3 08 D3 40 00       mov  ds:DbdDevClose, eax
CODE:00404E4B 68 FC 4E 40 00       push offset aDbddevgetinfo ; "DbdDevGetInfo"
CODE:00404E50 A1 20 B1 40 00       mov  eax, ds:hModule
CODE:00404E55 50                   push eax             ; hModule
CODE:00404E56 E8 4D EB FF FF       call GetProcAddress
CODE:00404E5B A3 0C D3 40 00       mov  ds:DbdDevGetInfo, eax
CODE:00404E60 68 0C 4F 40 00       push offset aDbddevregistercallback_0 ; "DbdD
CODE:00404E65 A1 20 B1 40 00       mov  eax, ds:hModule
```

Flow analysis



```
loc_800C1360:
addiu   $1, 0xC90
jal-    $1              # jump to parse_firmwareheader
lui     $s0, 0xBC01
beqz    $v0, loc_8000138C
la      $s0, aFail      # "fail!\n"
```

```
la      $1, print_serial
jalr    $1
move    $a0, $s0
```

```
loc_8000138C:
la      $1, calc_checksum
jalr    $1
nop
beqz    $v0, loc_800013B8
la      $1, print_serial
```

```
loc_80001384:
j       loc_80001384
nop
```

```
jalr    $1
move    $a0, $s0
```

```
loc_800C13B8:
la      $1, verify_s'gnature
jalr    $1
nop
beqz    $v0, loc_800013E4
la      $1, print_serial
```

```
loc_800013B0:
j       loc_800013B0
nop
```

```
jalr    $1
move    $a0, $s0
```

```
loc_800C13E4:
la      $1, unpack_firmware
jal^    $1
nop
beqz    $v0, loc_80001410
la      $1, print_serial
```

```
loc_800013)C:
j       lo:_800013DC
rop
```

```
jalr    $1
move    $a0, $s0
```

```
loc_80J01410:
la      $1, checksum_unpacked_firmware
jalr    $1
nop
beqz    $vC, loc_8000143C
la      $1, print_serial
```

```
loc_80001408:
j       loc_80001408
no3
```

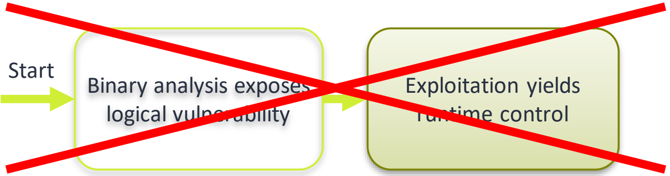
```
jalr    $1
move    $a0, $s0
```

```
'oc_8000143C:
'w      $v1, -Jx7FDC($gp)
jalr    $v1                  # run @ unpacked base ram address
sw      $v1, -Jx7FCC($gp)
addiu   $sp, 4
```

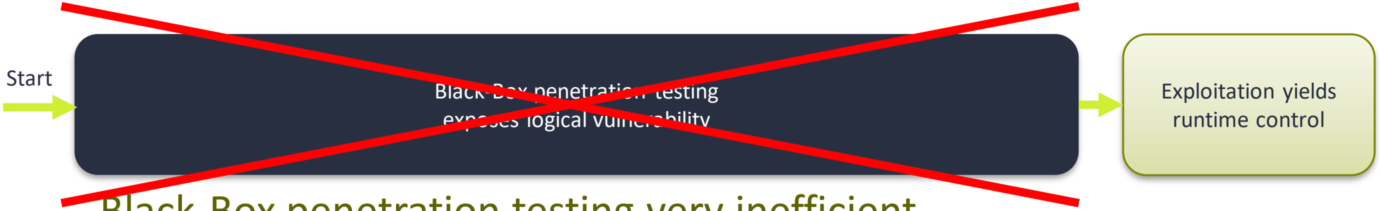
Software vulnerability hiding

- Given the widespread presence of vulnerabilities there is an increasing desire to mitigate risk
- Finding software vulnerabilities gets more difficult without access to source/binary code
- Access to device software is increasingly restricted:
    - PC software used to be accessible (e.g. exe files)
    - Smart phone software is only visible for root
    - Set-Top-Box software is hidden, and encrypted in transit
- How to attack a product protected with software encryption?

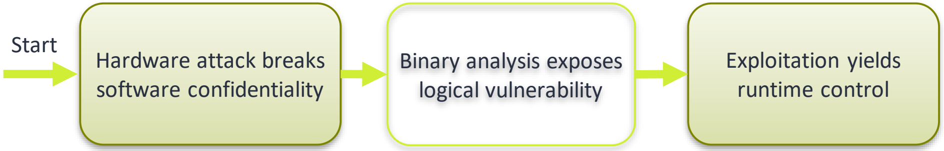
Attacking encrypted software


Start
~~Binary analysis exposes logical vulnerability~~
~~Exploitation yields runtime control~~

Encrypted software hides binary code


Start
~~Black-Box penetration testing exposes logical vulnerability~~
Exploitation yields runtime control

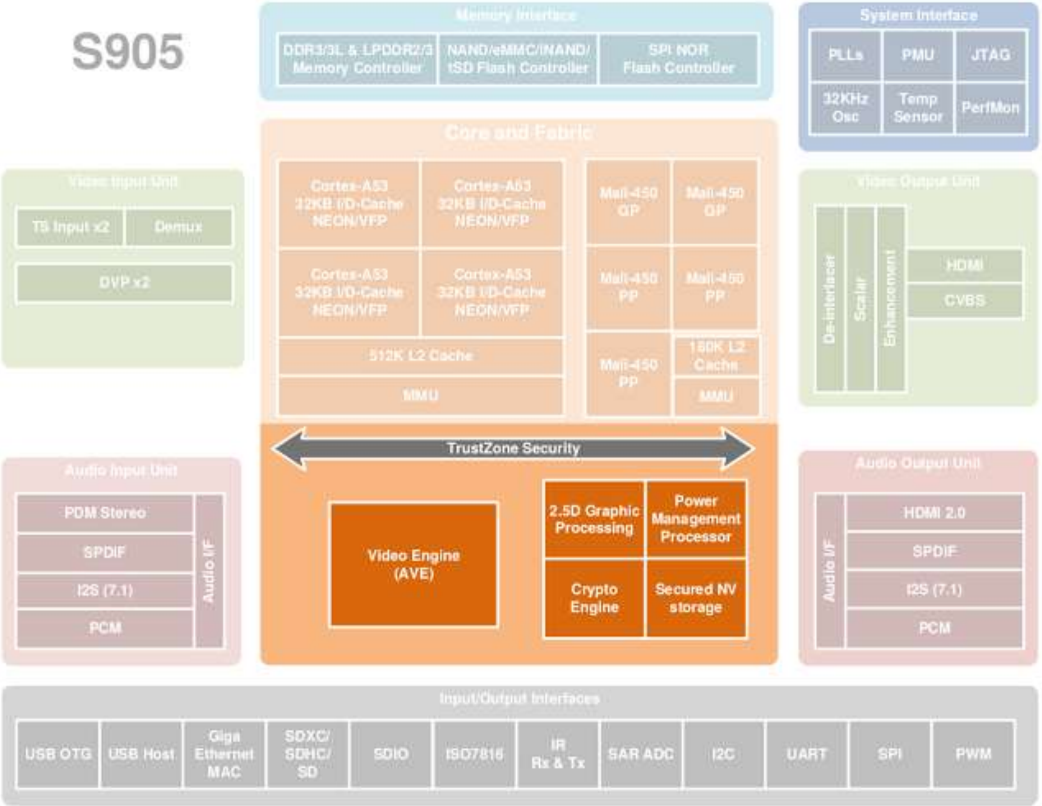
Black-Box penetration testing very inefficient


Start
Hardware attack breaks software confidentiality
Binary analysis exposes logical vulnerability
Exploitation yields runtime control

Hardware attack offers two-step alternative:

1. Break software confidentiality
2. White-box binary analysis exposes logical vulnerability

Design flaw in Pay-TV SoC

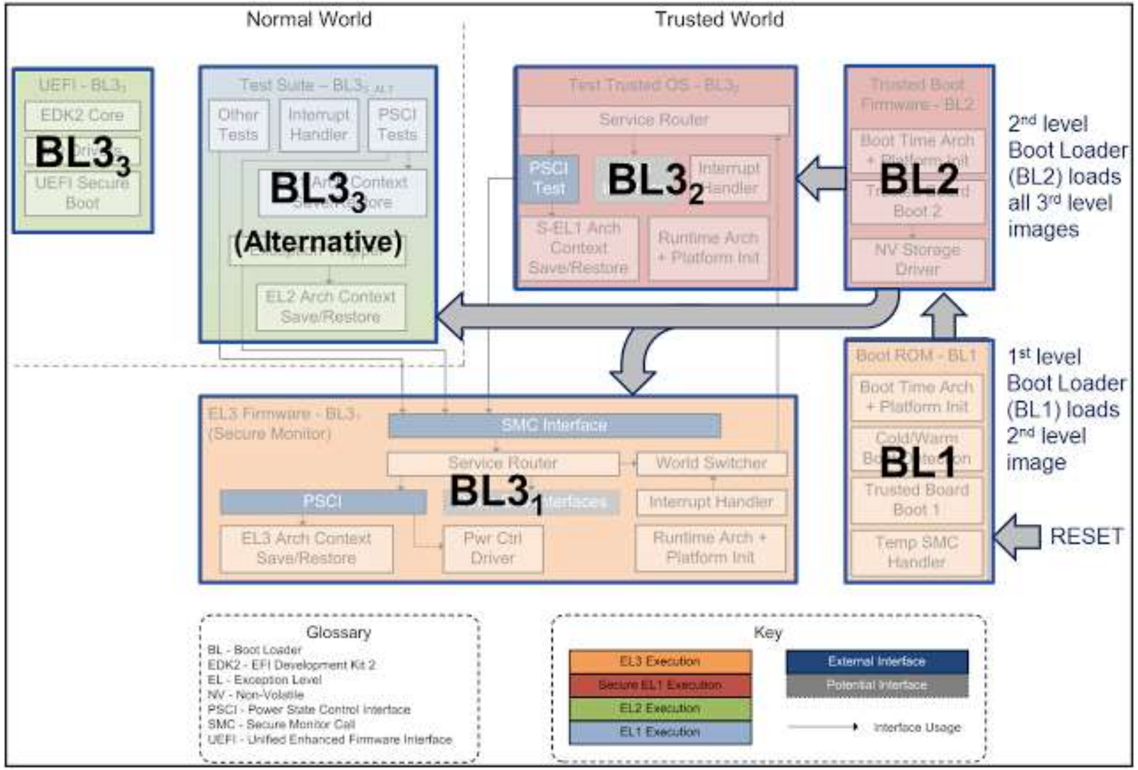


Security

- Trustzone based Trusted Execution Environment (TEE)
- Secured boot, encrypted OTP, internal control buses and storage
- Protected memory regions and electric fence data partition
- Hardware based Trusted Video Path (TVP) and secured contents (needs SecureOS software)

Source: http://www.fredericb.info/2016/10/amlogic-s905-soc-bypassing-not-so.html

Secure boot chain broken by backdoor



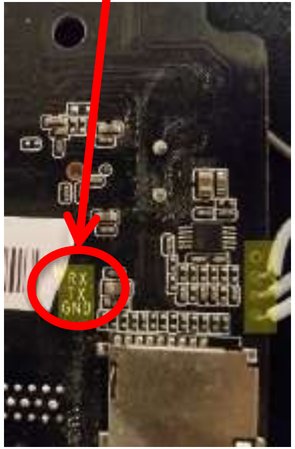
Glossary

BL - Boot Loader
EDK2 - EFI Development Kit 2
EL - Exception Level
NV - Non-Volatile
PSCI - Power State Control Interface
SMC - Secure Monitor Call
UEFI - Unified Enhanced Firmware Interface

2nd level Boot Loader (BL2) loads all 3rd level images

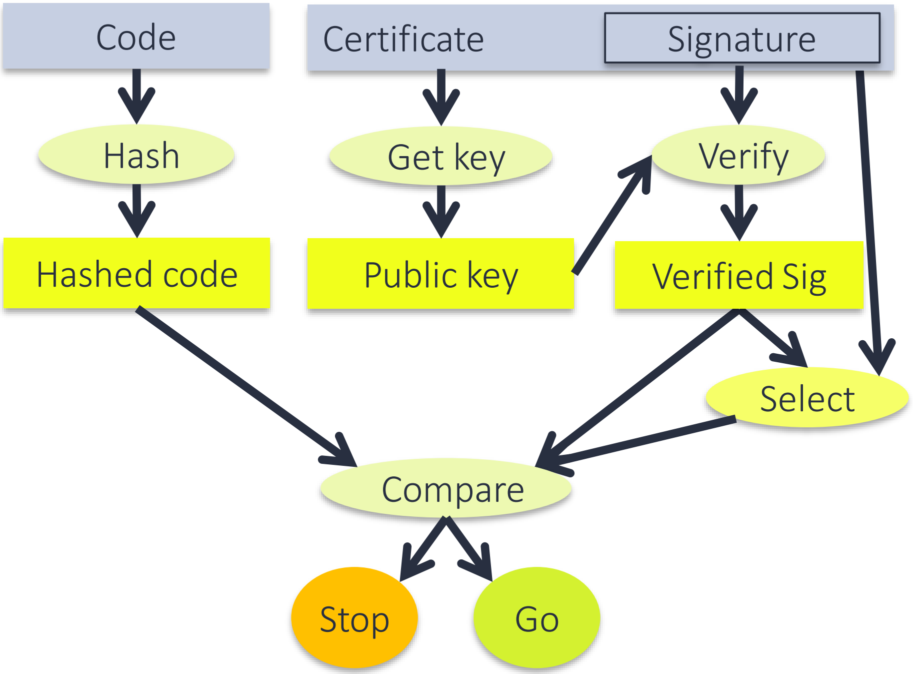
1st level Boot Loader (BL1) loads 2nd level image

Attacker used hardware weakness to dump Boot Loader image



Boot Loader header analysis

```
struct aml_img_header { // 64 bytes
    unsigned char magic[4];// "@AML"
    uint32_t total_len;
    uint8_t header_len;
    uint8_t unk_x9;
    uint8_t unk_xA;
    uint8_t unk_xB;
    uint32_t unk_xC;
    uint32_t sig_type;
    uint32_t sig_offset;
    uint32_t sig_size;
    uint32_t data_offset;
    uint32_t unk_x20;
    uint32_t cert_offset;
    uint32_t cert_size;
    uint32_t data_len;
    uint32_t unk_x30;
    uint32_t code_offset;
    uint32_t code_len;
    uint32_t unk_x3C;
  } aml_img_header_t;
```



sig_type provides backdoor that bypasses verification

Conclusions

- Scalable attacks need software exploitation
  - Hardware attacks are laborious
  - Software vulnerabilities are ubiquitous
  - Software exploits are easy to replicate
- Software encryption is inevitable for security
  - Binary analysis very successful in identifying vulnerabilities
  - Increasing number of products use encrypted software
- Hardware attacks are scalable when
  - Software is encrypted
  - Shallow bugs (detectable black-box style) are absent
  - Used in the identification step to extract software
  - Deep software vulnerabilities are present

Questions? contact:

Marc Witteman

witteman@riscure.com

Riscure B.V.
Frontier Building, Delftechpark 49
2628 XJ Delft
The Netherlands
Phone: +31 15 251 40 90
www.riscure.com

Riscure North America
550 Kearny St., Suite 330
San Francisco, CA 94108 USA
Phone: +1 650 646 99 79
inforequest@riscure.com

Riscure China
Room 2030-31, No. 989, Changle Road,
Shanghai 200031
China
Phone: +86 21 5117 5435
inforcn@riscure.com

riscure

driving your security forward

Riscure is hiring, visit https://www.riscure.com/careers/