

Binbloom Reloaded

Damien Cauquil |  virtualabs | PTS 2022

Introduction

Who am I ?



- Security Engineer (Quarkslab)
- Hardware/software RE
- Bluetooth Low Energy, sometimes 🙄

A bit of context

- Guillaume Heilles (**PapaZours**) published **binbloom** in 2020
- In 2021, he gave me **some insights of binbloom** before leaving
- A month after, I stumbled upon an **unknown firmware ...**

Unknown firmware, really ?

- Designed for **AArch64**
- **64-bit** architecture
- No idea of its **base address**
- **Not supported** by binbloom !

Wait, what's a base address ?

ELF file format

File walk-through

ANGE ALBERTINI
CORKAMI.COM



HEXADECIMAL DUMP

ASCII DUMP

```
7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 00 00 .ELF.....
02 00 03 00 01 00 00 00 60 00 00 00 40 00 00 00 .....~...@...
C0 00 00 00 00 00 00 00 34 00 20 00 01 00 28 00 +.....4.....(
04 00 03 00                                     ....
```

FIELDS

VALUES

EXPLANATION

1

e_ident
EI_MAG
EI_CLASS, EI_DATA
EI_VERSION
e_type
e_machine
e_version
e_entry
e_phoff
e_shoff
e_ehsize
e_phentsize
e_phnum
e_shentsize
e_shnum
e_shstrndx

0x7F, "ELF"
1 ELFCLASS32, 1 ELFDATA32
1 EX_CURRENT
2 ET_EXEC
3 EM_386
1 EX_CURRENT
0x8000060
0x40
0xC0
0x34
0x20
1
0x28
4
3*

3

CONSTANT SIGNATURE
32 BITS, LITTLE-ENDIAN
ALWAYS 1
EXECUTABLE
INTEL 386 (AND LATER)
ALWAYS 1
ADDRESS WHERE EXECUTION STARTS
PROGRAM HEADERS' OFFSET
SECTION HEADERS' OFFSET
ELF HEADER'S SIZE
SIZE OF A SINGLE PROGRAM HEADER
COUNT OF PROGRAM HEADERS
SIZE OF A SINGLE SECTION HEADER
COUNT OF SECTION HEADERS
INDEX OF THE NAMES' SECTION IN THE TABLE

2

p_type
p_offset
p_vaddr
p_paddr
p_filesz
p_memsz
p_flags

1 PT_LOAD
0
0x8000000
0x8000000
0xA0
0xA0
5 PF_R|PF_E

THE SEGMENT SHOULD BE LOADED IN MEMORY
OFFSET WHERE IT SHOULD BE READ
VIRTUAL ADDRESS WHERE IT SHOULD BE LOADED
PHYSICAL ADDRESS WHERE IT SHOULD BE LOADED
SIZE ON FILE
SIZE IN MEMORY
READABLE AND EXECUTABLE

Offset: 0x40/Address: 0x000040

```
01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
A0 00 00 00 A0 00 00 00 05 00 00 00 00 00 00 00 .....

```

Loading address

2	p_type	1 PT_LOAD	THE SEGMENT SHOULD BE LOADED IN MEMORY
	p_offset	0	OFFSET WHERE IT SHOULD BE READ
...	p_vaddr	0x80000000	VIRTUAL ADDRESS WHERE IT SHOULD BE LOADED
...	p_paddr	0x80000000	PHYSICAL ADDRESS WHERE IT SHOULD BE LOADED
	p_filesz	0xA0	SIZE ON FILE
	p_memsz	0xA0	SIZE IN MEMORY
	p_flags	5 PF_R PF_X	READABLE AND EXECUTABLE

Windows PE



Ange Albertini
corkami.com

Hexadecimal dump	ASCII dump	Fields	Values	Explanation
4D 5A 00 00-00 00 00-00 00 00-00 00 00 00 Offset: 0x38 00 00 00 00-00 00 00-00 00 00-40 00 00 00	MZ.....@...	e_magic e_lfanew	'MZ' 0x40	constant signature offset of the PE Header ❶
Offset: 0x40 50 45 00 00-4C 01 03 00-00 00 00-00 00 00 00 00 00 00 00-E0 00 02 01...	PE..L.....a...	Signature Machine NumberOfSections SizeOfOptionalHeader Characteristics	'PE', 0, 0 0x14c [intel 386] 3 0xe0 0x102 [32b EXE]	constant signature processor: ARM/MIPS/Intel/... number of sections ❷ relative offset of the section table ❷ EXE/DLL/...
Offset: 0x58 ...0B 01 00 00-00 00 00-00 00 00 00 00 00 00 00-00 00 40 00-00 10 00 00-00 00 00 00 00 00 00 00-00 00 40 00-00 10 00 00-00 02 00 00 00 00 00 00-00 00 04 00 00 00-00 00 00 00 00 00 40 00 00-00 02 00 00-00 00 00-02 00 00 00 00 00 00 00-00 00 00 00-00 00 00-00 00 00 00 00 00 00 00-10 00 00 00...@.....@.....	Magic AddressOfEntryPoint ImageBase SectionAlignment FileAlignment MajorSubsystemVersion SizeOfImage SizeOfHeaders Subsystem NumberOfRvaAndSizes	0x10b [32b] 0x1000 0x400000 0x1000 0x200 4 [NT 4 or later] 0x4000 0x200 2 [GUI] 16	32 bits/64 bits where execution starts ❸ address where the file should be mapped in memory where sections should start in memory ❷ where sections should start on file ❷ required version of Windows total memory space required total size of the headers ❸ driver/graphical/command line/... number of data directories ❹
...00 00 00 00-00 00 00 00-00 00 00 00 00 20 00 00-00 00 00 00-00 00 00 00 00 00 00 00 00-00 00 00 00-00 00 00 00 00	ImportsVA	0x2000	RVA of the imports ❹

ImageBase

.....	Magic	0x10b [32b]	32 bits/64 bits
.....	AddressOfEntryPoint	0x1000	where execution starts ①
.....@.....	ImageBase	0x400000	address where the file should be mapped in memory
.....	SectionAlignment	0x1000	where sections should start in memory ②
.....	FileAlignment	0x200	where sections should start on file ②
.....	MajorSubsystemVersion	4 [NT 4 or later]	required version of Windows
..@.....	SizeOfImage	0x4000	total memory space required
.....	SizeOfHeaders	0x200	total size of the headers ③
.....	Subsystem	2 [GUI]	driver/graphical/command line/...
.....	NumberOfRvaAndSizes	16	number of data directories ④

Raw firmware

```

0000:0000 FE FF FF FF BB 56 B6 39 20 5D F2 EB 1D C7 38 15 |þÿÿÿ»V¶9 ]ðë.Ç8.
0000:0010 5D 89 EA B3 8B 76 7B 1D AF 7A 59 D7 D2 EA 30 51 |].ê³.v{.˘zY×0ê0Q
0000:0020 8A 5B E0 C8 EF 8F 1C CF DE DF 9F F2 C9 9C 9E 27 |[àÈì..İþß.òÉ..'
0000:0030 64 E0 A7 EA AC 3C 72 E6 5C DB 15 7C 2E FA 54 5C |dà§ê~<ræ\Û. |.úT\
0000:0040 7A 55 D8 32 7B AF 04 AF F1 AA BE 49 88 F0 DE C3 |zU02{˘.˘ñª¼I.ðþÃ
0000:0050 37 32 4B C3 7A F7 CC 8A 28 35 16 08 1C F1 0D 7F |72KÃz÷Î.(5...ñ..
0000:0060 EB 1B 21 C4 4A 04 F1 32 6D 14 F5 FD 8D 7A F0 17 |ë. !ÄJ.ñ2m.õý.zð.
0000:0070 CB 69 F0 3D 27 D0 51 29 16 AF 08 C4 80 AF D5 16 |Ëið='ðQ).˘.Ä.˘õ.
0000:0080 CA 62 AE 75 C6 67 9F 43 0D BD 4D 91 BF 77 A8 DD |Êb@uÆg.C.½M.¿w˘Ý
0000:0090 61 C3 FD E7 09 01 79 1B 9F ED B3 F2 8D C2 C7 85 |aÃýç...y...í³ð.ÂÇ.
0000:00A0 C3 23 1C F6 41 33 3D C8 5C 8F 28 E9 B7 2B AF 4F |Ã#.öA3=E\.(é+˘0
0000:00B0 E8 2A 8A 5C BC 30 1D 9B F2 81 36 3A EF DB 6F E5 |è*.\¼0...ò.6:ïÛoâ
0000:00C0 A8 C4 E3 34 9B D4 12 55 E0 27 16 09 71 CC 3D B3 |˘Ãã4.Ô.Uà'..qİ=³
0000:00D0 64 3E D3 4C D7 4B C6 E4 C0 6A B2 9D BB B8 D0 00 |d>ÓL×KÆäÀj².»„Đ.
0000:00E0 C6 08 A1 D4 F6 D3 36 51 42 32 EA 20 07 46 55 1A |Æ. |ÔöÓ6QB2ê .FU.
0000:00F0 0F F9 1D C1 D0 13 8A BF E2 98 F8 5B A0 BB 62 87 |.ù.ÁĐ...¿â.ø[ »b.
0000:0100 35 28 5D 24 9F 7F DD 2E E8 26 6E 33 19 E8 A9 C7 |5(]$.˘.Ý.è&n3.è©Ç
0000:0110 4A 0C 72 89 56 49 23 98 9D 5D FA 58 21 5C 0D BC |J.r.VI#... ]úX!\.¼
0000:0120 ED 55 42 B9 B1 65 C5 B8 4C 9B 47 51 7A B5 11 7B |íUB¹±eÄ„L.GQzµ.{
0000:0130 F2 69 94 7A 7C EF BE 4A 46 53 EE B3 67 C4 8B 9A |òì.z |ĩ¼JFSî³gÃ..
0000:0140 B1 D9 2E 49 CE C2 80 71 D9 8E 35 8C 69 4D A4 68 |±Ù.IİÂ.qÙ.5.iM±h
0000:0150 B5 35 68 4D E2 17 DA 5B 3B 03 2E 98 80 BC 2B 33 |µ5hMâ.Ú[;...¼+3
0000:0160 65 C4 BC 46 61 17 4E FF 3E EB 90 08 E2 48 74 2E |eÄ¼Fa.Nÿ>ë..âHt.
0000:0170 93 9D AFD3 0D 64 6B A5 72 F2 63 F0 D5 61 1A 2E |..˘Ó.dk¥ròcðõa..
0000:0180 4B F6 A7 1B B8 A8 BD 8E 03 00 00 00 40 00 00 00 |Kö§.„˘½.....@...
0000:0190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....

```

Finding a needle in a haystack

Tools !

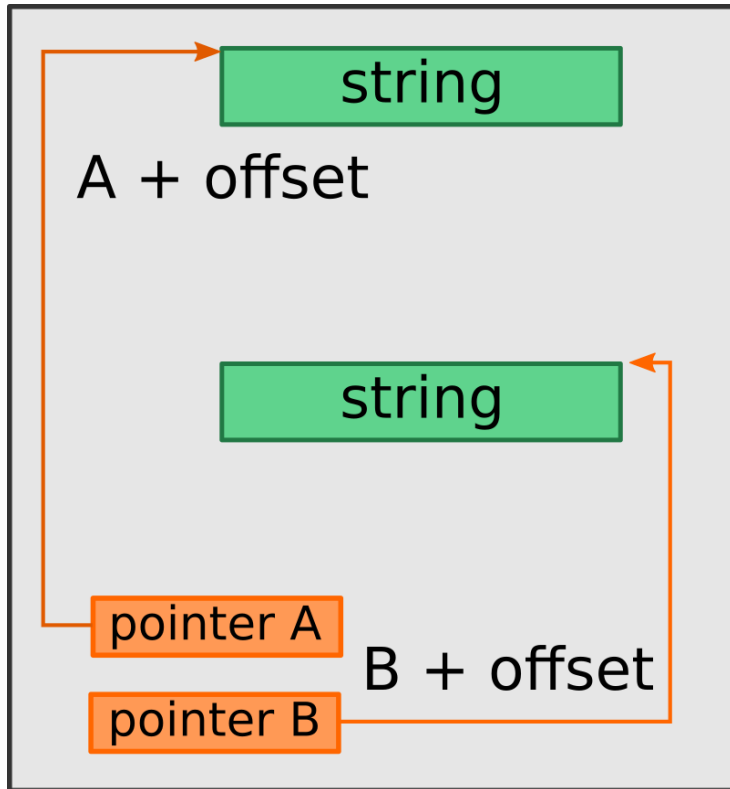


Basefind.py

- **Brute-force** on 32-bit address space

```
for base in xrange(args.min_addr, args.max_addr, args.page_size):
    if base % args.page_size == 0:
        print "Trying base address 0x%x" % base
    score = 0
    for ptr in ptr_table.keys():
        if ptr < base:
            #print "Removing pointer 0x%x from table" % ptr
            del ptr_table[ptr]
            continue
        if ptr >= (base + size):
            continue
        offset = ptr - base
        if offset in str_table:
            score += ptr_table[ptr]
```


Brute-force



- Tries every possible offset
- Looks for pointers pointing to **text strings**
- **Count valid pointers**

basefind.cpp

```
for( offset_t base = 0; base < 0xf0000000UL; base += 0x1000 )  
{  
    [ . . . ]  
}
```

Same in CPP 😅

rbasefind

- **Multi-threaded brute-force** on 32-bit address

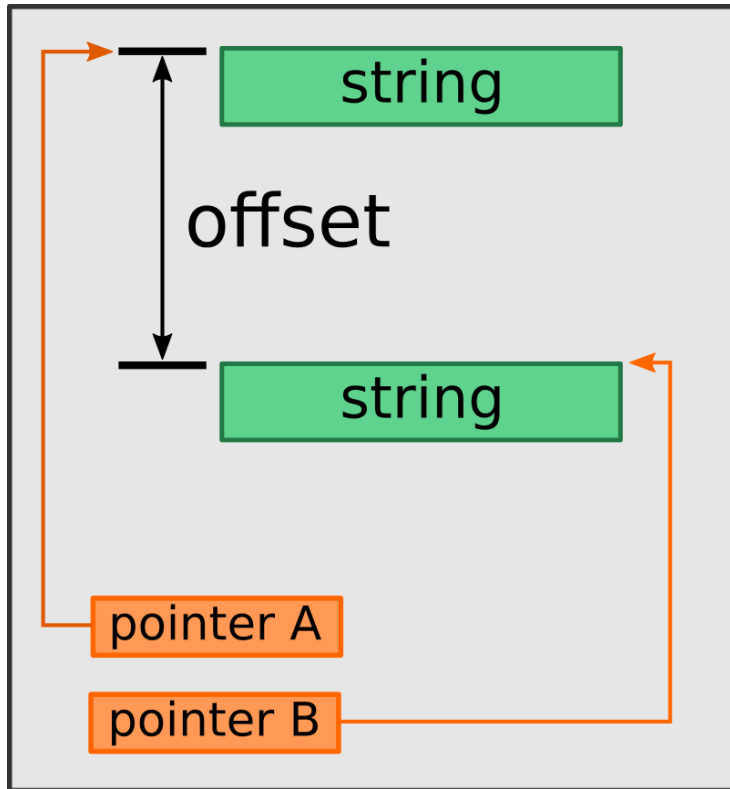
```
while current_addr <= interval.end_addr {  
    let mut news = FnvHashSet::default();  
    for s in strings {  
        match s.checked_add(current_addr) {  
            Some(add) => news.insert(add),  
            None => continue,  
        };  
    }  
    [...]  
    match current_addr.checked_add(config.offset) {  
        Some(_) => current_addr += config.offset,  
        None => break,  
    };  
    pb.inc();  
}
```

binbloom

- **Splits search space** in segments

```
uint32_t p2(uint32_t x) {  
    return 1 << (32 - __builtin_clz(x - 1));  
}  
  
/* ... */  
size = read_file(filename, &firmware);  
/* ... */  
mask_segment = ~(p2(size) - 1);  
mask_pointer = p2(size) - 1;  
nb_segments = mask_segment;  
while ((nb_segments & 1) == 0)  
    nb_segments = nb_segments >> 1;  
nb_segments++;
```

basefind2



- Computes **distance between strings**
- Looks for pointers with same distance
 $B - A = \text{offset}$
- **No brute-force !**

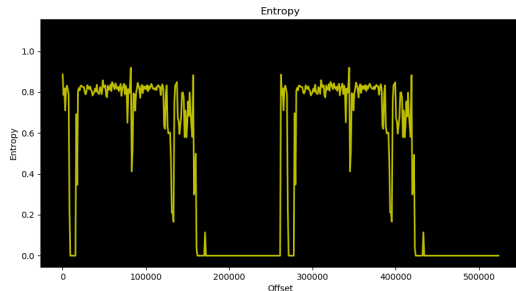
So, limited tools.

- Most of them perform a **brute-force search on 32-bit address space**
- Most of them rely on **text strings**
- **None of them supports 64-bit** architecture
(more than 1^{14} possibilities !)

**Burning the haystack
to get the needle**

1. Isolating data

Telling data and code apart



- Entropy can be used to **determine data segments**
- Data segments may contain **strings** and **pointers**
- Everything else is **code**

Avoiding useless data

- Do not consider **long series of 0x00 or 0xFF**
- Generally present in **unused areas** or used as **fillers**

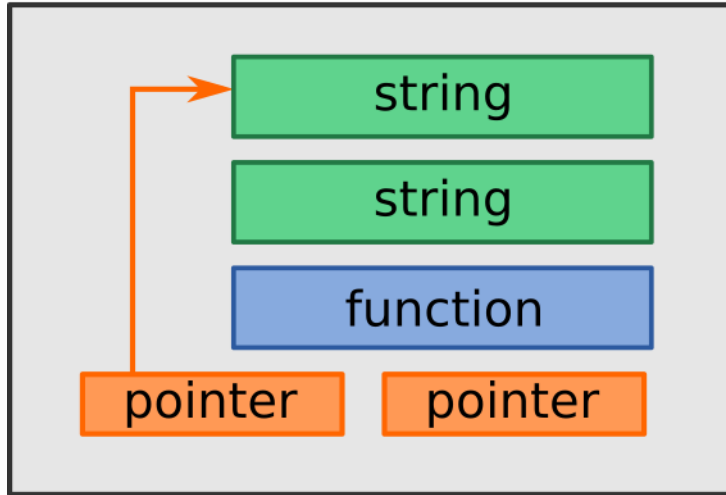
Points of Interest

- **Text strings**
- **Arrays** of similar values
- **Functions** (if possible)

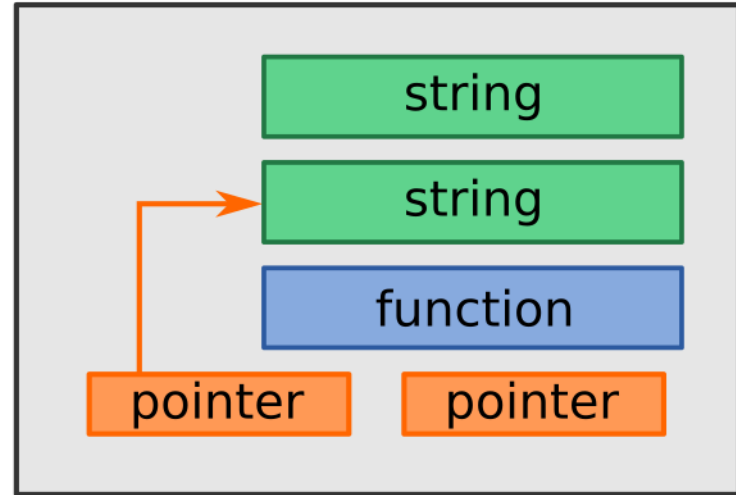
2. Inferring base address

Inference vs. brute-force

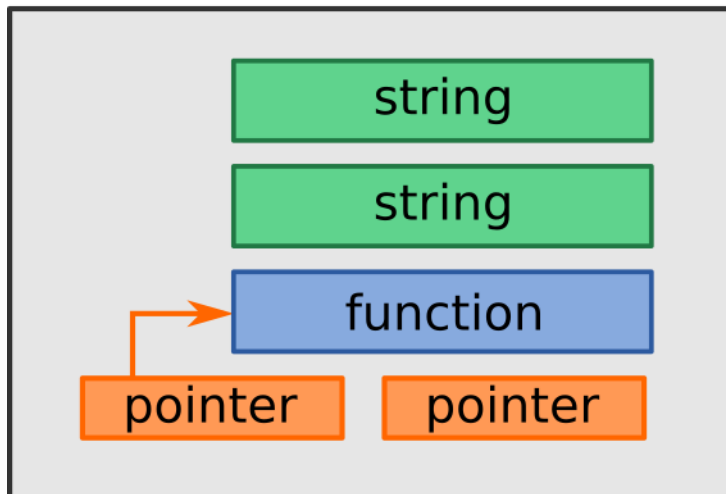
Candidate #1



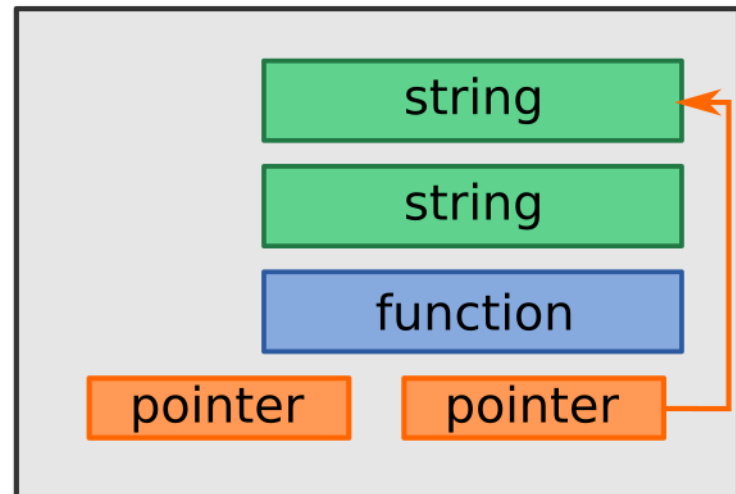
Candidate #2



Candidate #3



Candidate #4



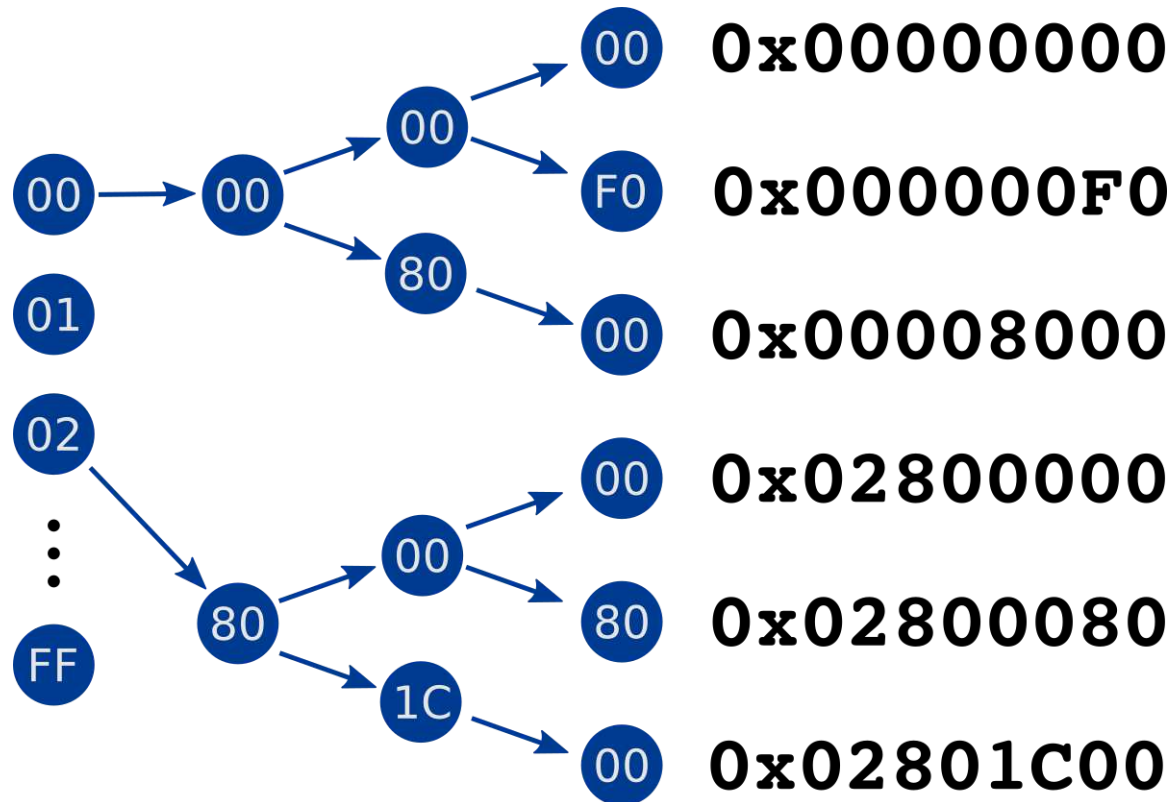
Inference vs. brute-force

- Search space is **reduced**
- Works on both **32-bit and 64-bit** arch firmwares
- Still a **lot of candidates** 🤔

Memory & performances issues

- Storing candidates in a list **is not efficient**
- Decided to **use a tree** rather than a list

Candidate addresses tree



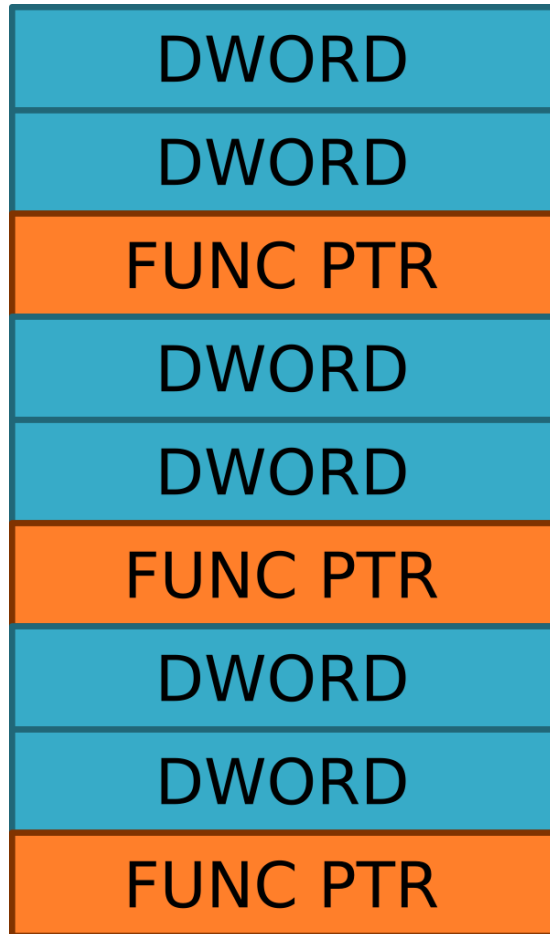
Candidate addresses tree

- **Searching/storing** only requires **8 operations**
- We **can prune the tree** to make room for new candidates

Evaluating candidate addresses

- looking for the address that will give the **best results**
- count **valid pointers**
- **Arrays** of valid pointers have **more weight**
- Compute **a score** for each candidate address

Structured data recognition



- Converting **arrays** of values **into structures**
- As a binbloom v1 legacy, only detects **UDS-related structures** 😊
- ... but may do **much more** !

Demo

Binbloom vs. others

Firmware	Binbloom	Binbloom2	Rbasefind
AE5R100V	11.33	3.019	0.916
bootloader ARM	5.48	0.183	5.40
ECU external flash firmware	5.78	5.69	6.17
IntegrityOS application	~	1.453	~
UBoot standalone application	8.228	0.723	1.462
STM32 firmware	5.232	0.03	0.064
Teensy firmware	5.686	0.068	0.053
Google Titan M firmware (2018)	9.664	1.288	10.23
Flash Air firmware	11.042	37.52	44.184

Performances (exec time in seconds)

Improvements

- Automatic architecture detection (with **cpu_rec**)
- **Entropy thresholds** may be defined per-architecture
- Automatic **function detection** per arch
- Better **structured data recognition** algorithm

Try Binbloom v2 !

<https://github.com/quarkslab/binbloom>

Conclusion

Conclusion

- We presented a more **generic base address search algorithm**
- **Binbloom v2** supports both **32-bit** and **64-bit** archs
- Still some room for improvements, so **stay tuned**

Thanks ! Questions ?

**Damien
Cauquil**

 quarkslab

 dcauquil@quarkslab.com

 virtualabs

 virtualabs