Polypyus

The Firmware Historian

Jan Friebertshäuser, Florian Kosterhon, Jiska Classen, Matthias Hollick
Secure Mobile Networking Lab - SEEMOO
Technische Universität Darmstadt, Germany

Motivation

InternalBlue Bluetooth Experimentation Framework

- Monitoring and modification of management traffic.
- Firmware manipulation during runtime.
- Additional projects for firmware diffing, patching in C, emulation, etc.
- … and it runs on many devices!

Android 6–11
Samsung Galaxy S series +
Google Nexus series

iOS 12–14
iPhone 6–11 (12)

Linux (BlueZ)
Eval kits + Raspberry Pis

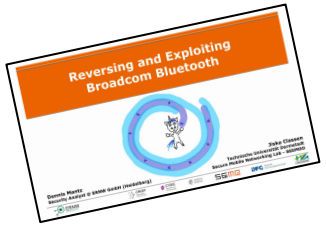
macOS High Sierra–Big Sur
MacBooks + iMacs

https://github.com/seemoo-lab/internalblue

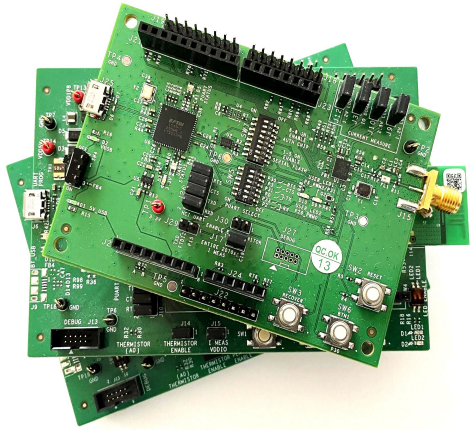
ACM **WiSec '19**
ACM **MobiSys '19**

Leaked Symbols

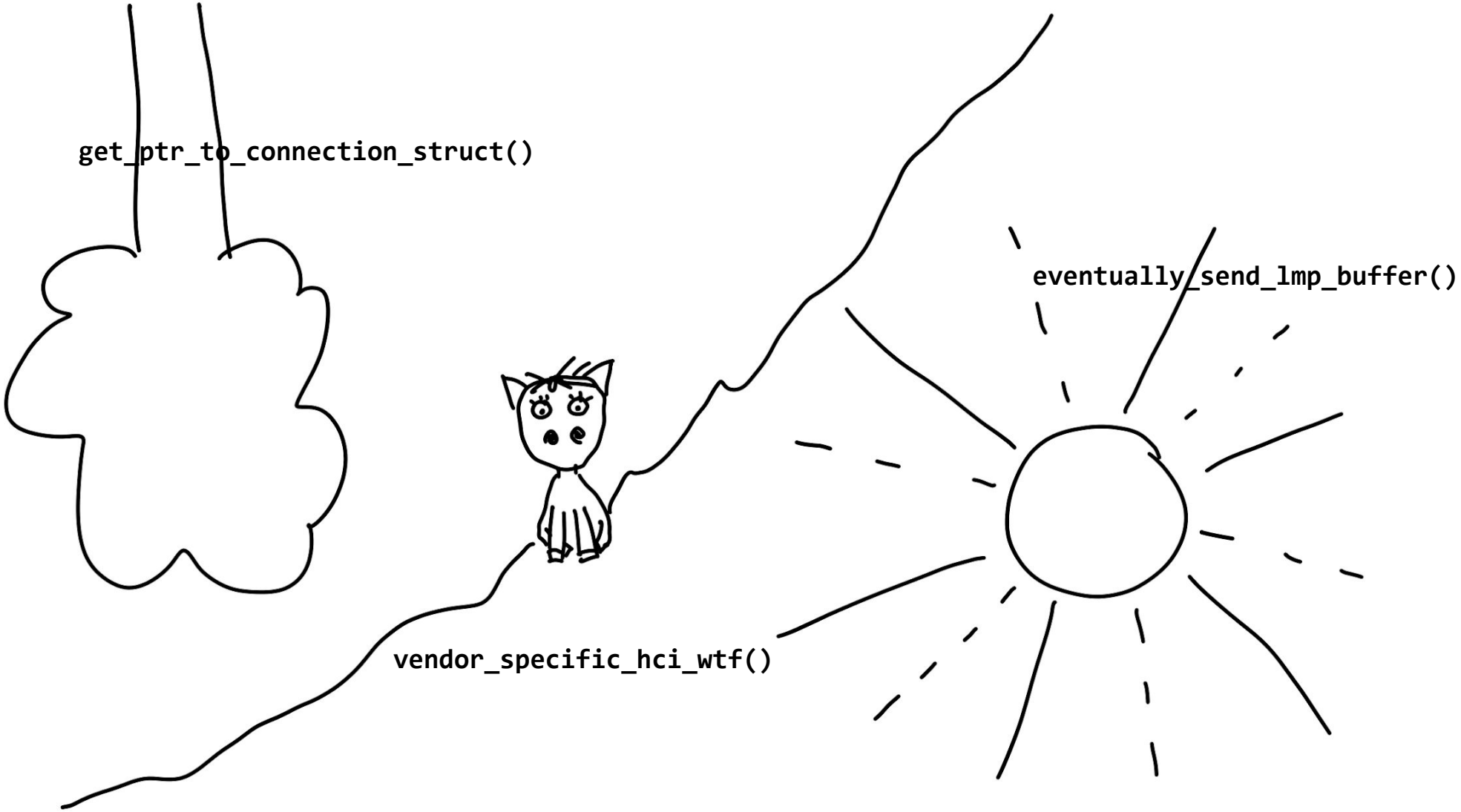
- Cypress WICED Studio 6.2–6.4 accidentally included function and global variable names for a few development kits.
- One of these chips is also contained in the MacBook Pro 2016.
- Found this after manually reverse-engineering the Google Nexus 5 firmware for a couple of months…



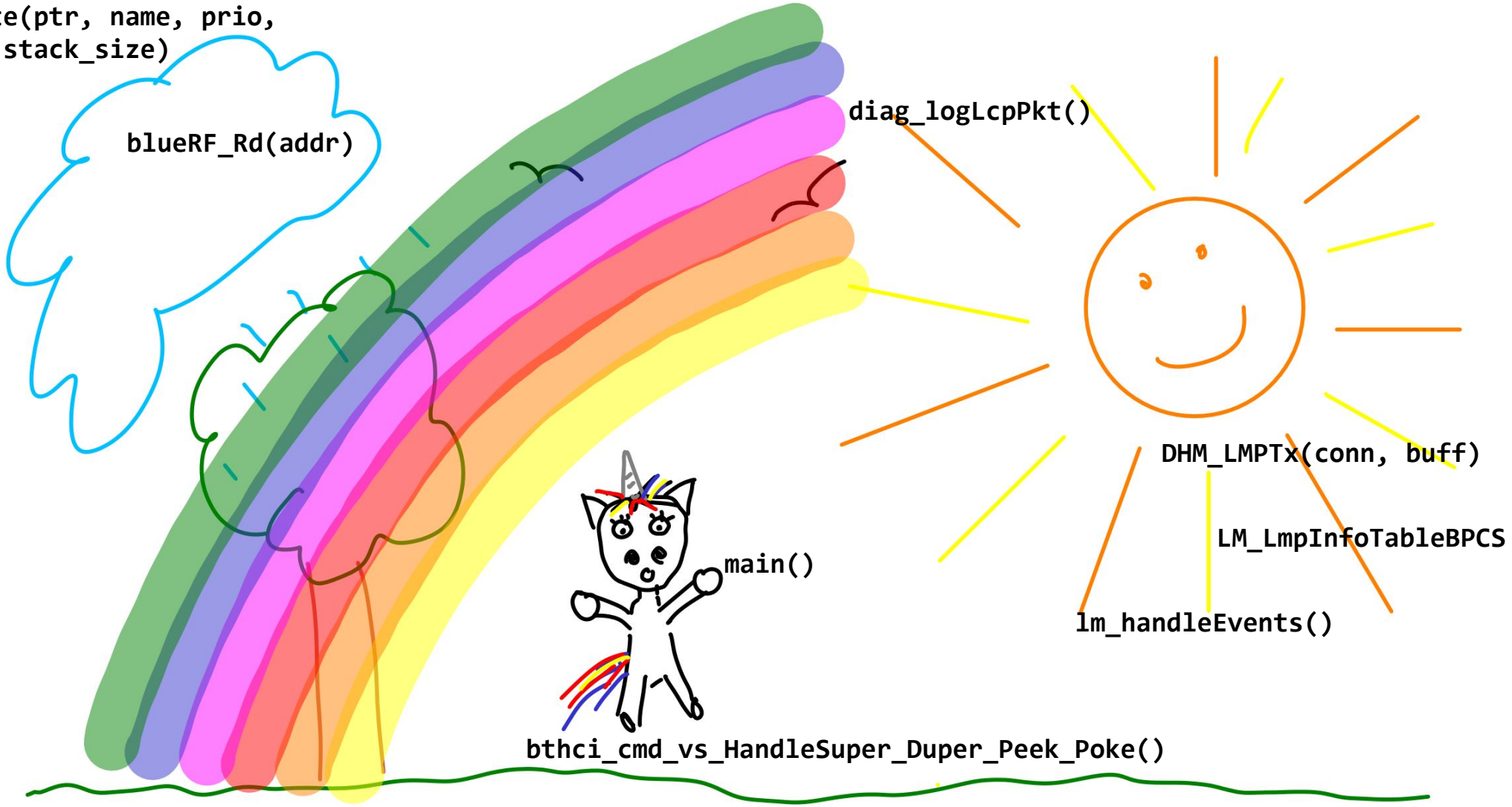
REcon ’19



Reverse Engineering without Symbols



Reverse Engineering with Symbols



Binary Diffing?

When comparing symbol names between evaluation kit firmwares, many of them are common and should be found using binary differs.

Firmware A	Firmware B	Unique Symbols	Common Symbols
BCM20703A2	CYW20719B1	12 161	5655
BCM20703A2	CYW20735B1	8402	5496
BCM20703A2	CYW20819A1	8215	5332
CYW20719B1	CYW20735B1	4790	10 435
CYW20719B1	CYW20819A1	6115	9515
CYW20735B1	CYW20819A1	1927	9570

Problem Analysis

Tried various binary differs, very poor results :(

- Disassemblers miss function starts (and ends).
- Binary differs only diff parts that were identified as code/function.
- Binary differs cannot utilize statistics like call graphs with missing functions.

... so I found myself hand-picking significant byte sequences to search for relevant functions in other firmware versions.

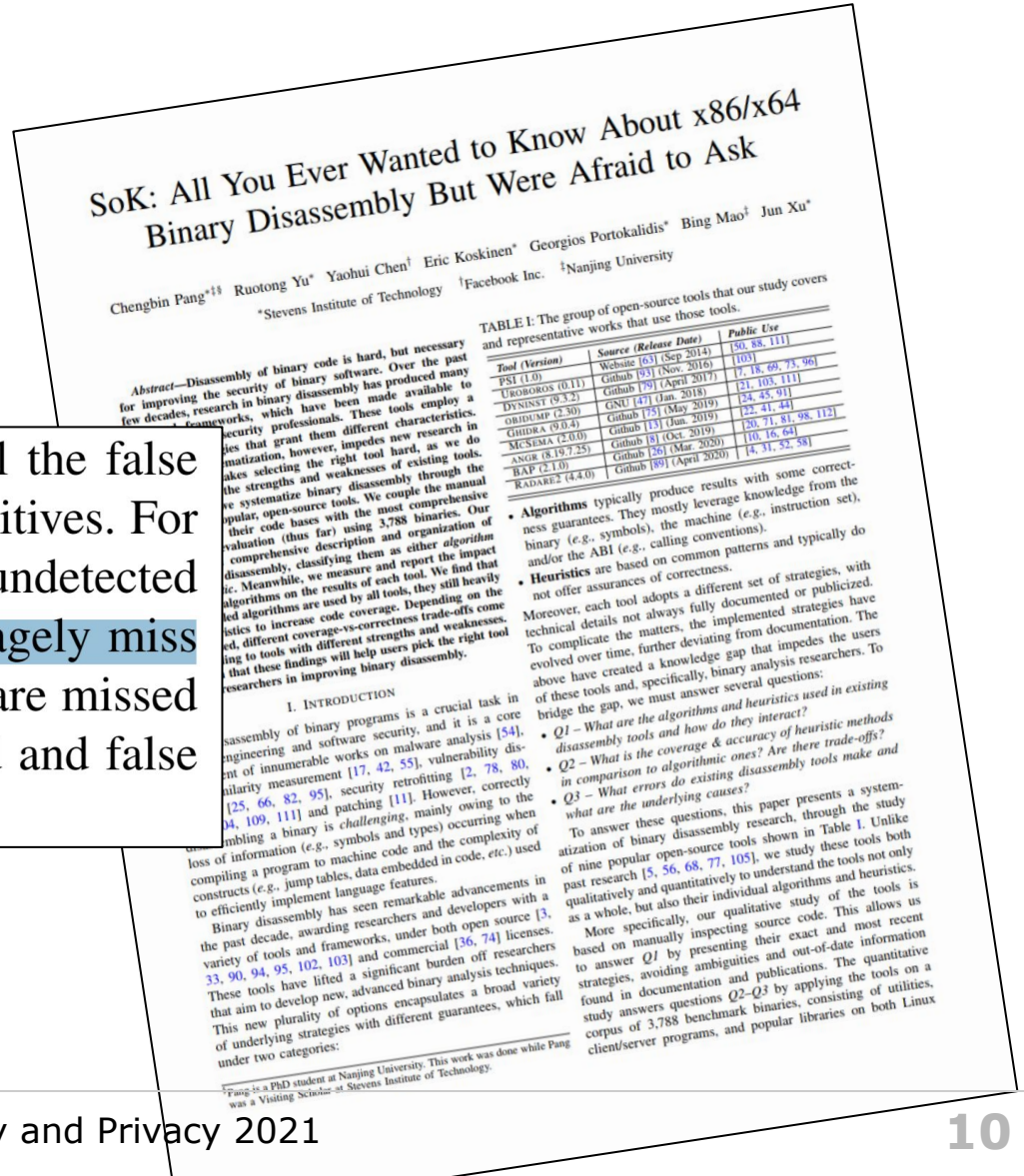
WHY YOU NO DIFF??!?

```
                    mm_freeACLBuffer
0x17b4c: 00 28          cmp  r0, #0
0x17b4e: 02 d0          beq  locret_17b56
0x17b50: 08 38          subs r0, #8
0x17b52: f2 f7 43 b8    b.w  mem_Release
                      locret_17b56
0x17b56: 70 47          bx   lr
```

```
                    loc_d0dc:
0x0d0dc: 00 28          cmp  r0, #0
0x0d0de: 02 d0          beq  locret_d0e6
0x0d0e0: 08 38          subs r0, #8
0x0d0e2: f4 f7 f7 bf    b.w  mem_Release
                      locret_d0e6
0x0d0e6: 70 47          bx   lr
```

Come on, function identification has been solved years ago!!!

The reasons of false negatives are consistent. All the false negatives by linear tools are side-effects of false positives. For recursive tools, most false negatives are caused by undetected function: as shown in Table X, recursive tools averagely miss 25.0% of the functions. The remaining instructions are missed mainly because certain jump tables are not resolved and false positives over-run the legitimate instructions.



SoK: All You Ever Wanted to Know About x86/x64 Binary Disassembly But Were Afraid to Ask

Chengbin Pang*‡§ Ruotong Yu* Yaohui Chen† Eric Koskinen* Georgios Portokalidis* Bing Mao‡ Jun Xu*

*Stevens Institute of Technology †Facebook Inc. ‡Nanjing University

TABLE I: The group of open-source tools that our study covers and representative works that use those tools.

Tool (Version)	Source (Release Date)	Public Use
PSI (1.0)	Website [63] (Sep 2014)	[50, 88, 111]
UROBOROS (0.11)	Github [93] (Nov. 2016)	[103]
DYNINST (9.3.2)	Github [79] (April 2017)	[7, 18, 69, 73, 96]
OBJDUMP (2.30)	GNU [47] (Jan. 2018)	[21, 103, 111]
GHIDRA (9.0.4)	Github [75] (May 2019)	[24, 45, 91]
MCSEMA (2.0.0)	Github [13] (Jun. 2019)	[22, 41, 44]
ANGR (8.19.7.25)	Github [8] (Oct. 2019)	[20, 71, 81, 98, 112]
BAP (2.1.0)	Github [26] (Mar. 2020)	[10, 16, 64]
RADARE2 (4.4.0)	Github [89] (April 2020)	[4, 31, 52, 58]

Abstract—Disassembly of binary code is hard, but necessary for improving the security of binary software. Over the past few decades, research in binary disassembly has produced many ... frameworks, which have been made available to security professionals. These tools employ a ...ies that grant them different characteristics. ...matization, however, impedes new research in ...akes selecting the right tool hard, as we do ...the strengths and weaknesses of existing tools. ...e systematize binary disassembly through the ...opular, open-source tools. We couple the manual ...their code bases with the most comprehensive ...valuation (thus far) using 3,788 binaries. Our ...comprehensive description and organization of ...disassembly, classifying them as either *algorithm* ...ic. Meanwhile, we measure and report the impact ...algorithms on the results of each tool. We find that ...ed algorithms are used by all tools, they still heavily ...istics to increase code coverage. Depending on the ...ed, different coverage-vs-correctness trade-offs come ...ing to tools with different strengths and weaknesses. ...that these findings will help users pick the right tool ...researchers in improving binary disassembly.

I. INTRODUCTION

...sassembly of binary programs is a crucial task in ...engineering and software security, and it is a core ...ent of innumerable works on malware analysis [54], ...ilarity measurement [17, 42, 55], vulnerability dis- [25, 66, 82, 95], security retrofitting [2, 78, 80, ...4, 109, 111] and patching [11]. However, correctly ...mbling a binary is *challenging*, mainly owing to the loss of information (*e.g.*, symbols and types) occurring when compiling a program to machine code and the complexity of constructs (*e.g.*, jump tables, data embedded in code, *etc.*) used to efficiently implement language features.

Binary disassembly has seen remarkable advancements in the past decade, awarding researchers and developers with a variety of tools and frameworks, under both open source [3, 33, 90, 94, 95, 102, 103] and commercial [36, 74] licenses. These tools have lifted a significant burden off researchers that aim to develop new, advanced binary analysis techniques. This new plurality of options encapsulates a broad variety of underlying strategies with different guarantees, which fall under two categories:

- **Algorithms** typically produce results with some correctness guarantees. They mostly leverage knowledge from the binary (*e.g.*, symbols), the machine (*e.g.*, instruction set), and/or the ABI (*e.g.*, calling conventions).
- **Heuristics** are based on common patterns and typically do not offer assurances of correctness.

Moreover, each tool adopts a different set of strategies, with technical details not always fully documented or publicized. To complicate the matters, the implemented strategies have evolved over time, further deviating from documentation. The above have created a knowledge gap that impedes the users of these tools and, specifically, binary analysis researchers. To bridge the gap, we must answer several questions:

- *Q1 – What are the algorithms and heuristics used in existing disassembly tools and how do they interact?*
- *Q2 – What is the coverage & accuracy of heuristic methods in comparison to algorithmic ones? Are there trade-offs?*
- *Q3 – What errors do existing disassembly tools make and what are the underlying causes?*

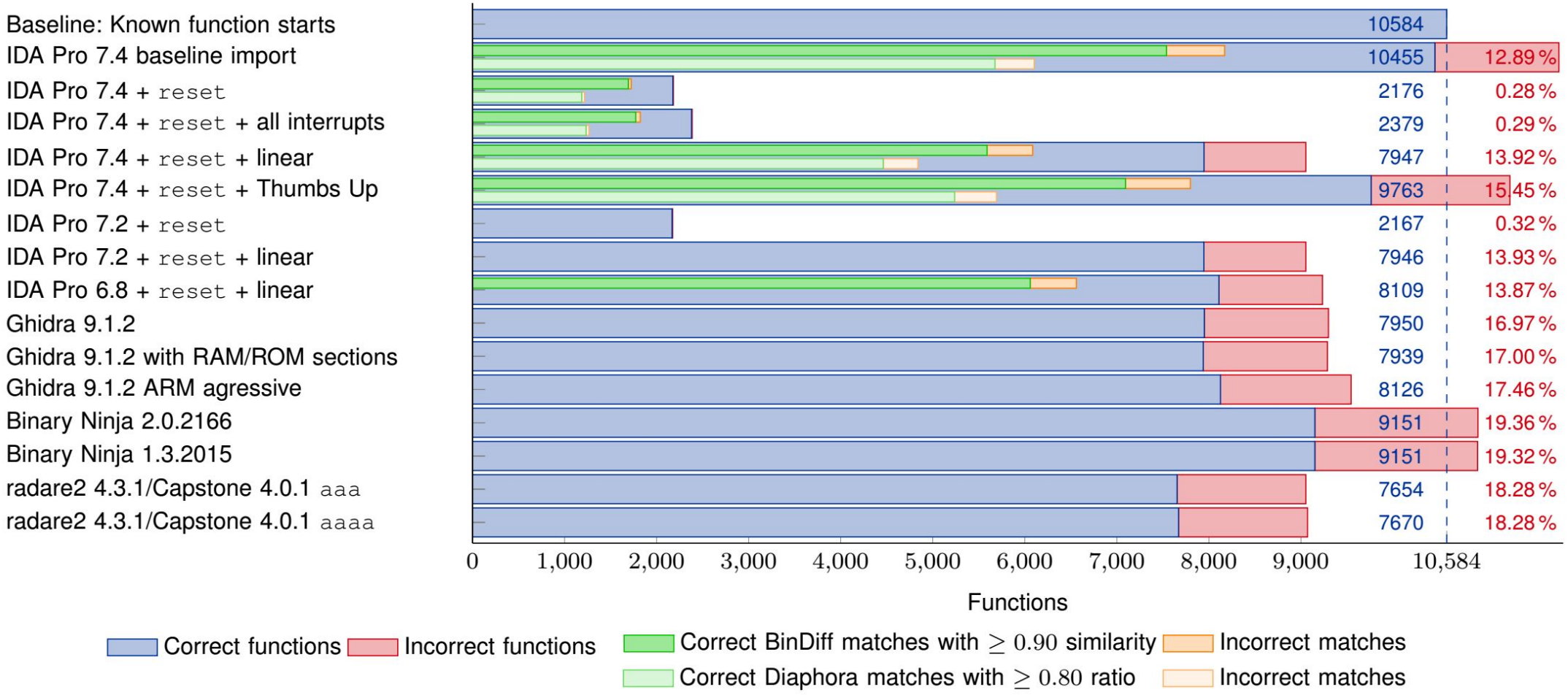
To answer these questions, this paper presents a systematization of binary disassembly research, through the study of nine popular open-source tools shown in Table I. Unlike past research [5, 56, 68, 77, 105], we study these tools both qualitatively and quantitatively to understand the tools not only as a whole, but also their individual algorithms and heuristics.

More specifically, our qualitative study of the tools is based on manually inspecting source code. This allows us to answer *Q1* by presenting their exact and most recent strategies, avoiding ambiguities and out-of-date information found in documentation and publications. The quantitative study answers questions *Q2–Q3* by applying the tools on a corpus of 3,788 benchmark binaries, consisting of utilities, client/server programs, and popular libraries on both Linux

‡Pang is a PhD student at Nanjing University. This work was done while Pang was a Visiting Scholar at Stevens Institute of Technology.

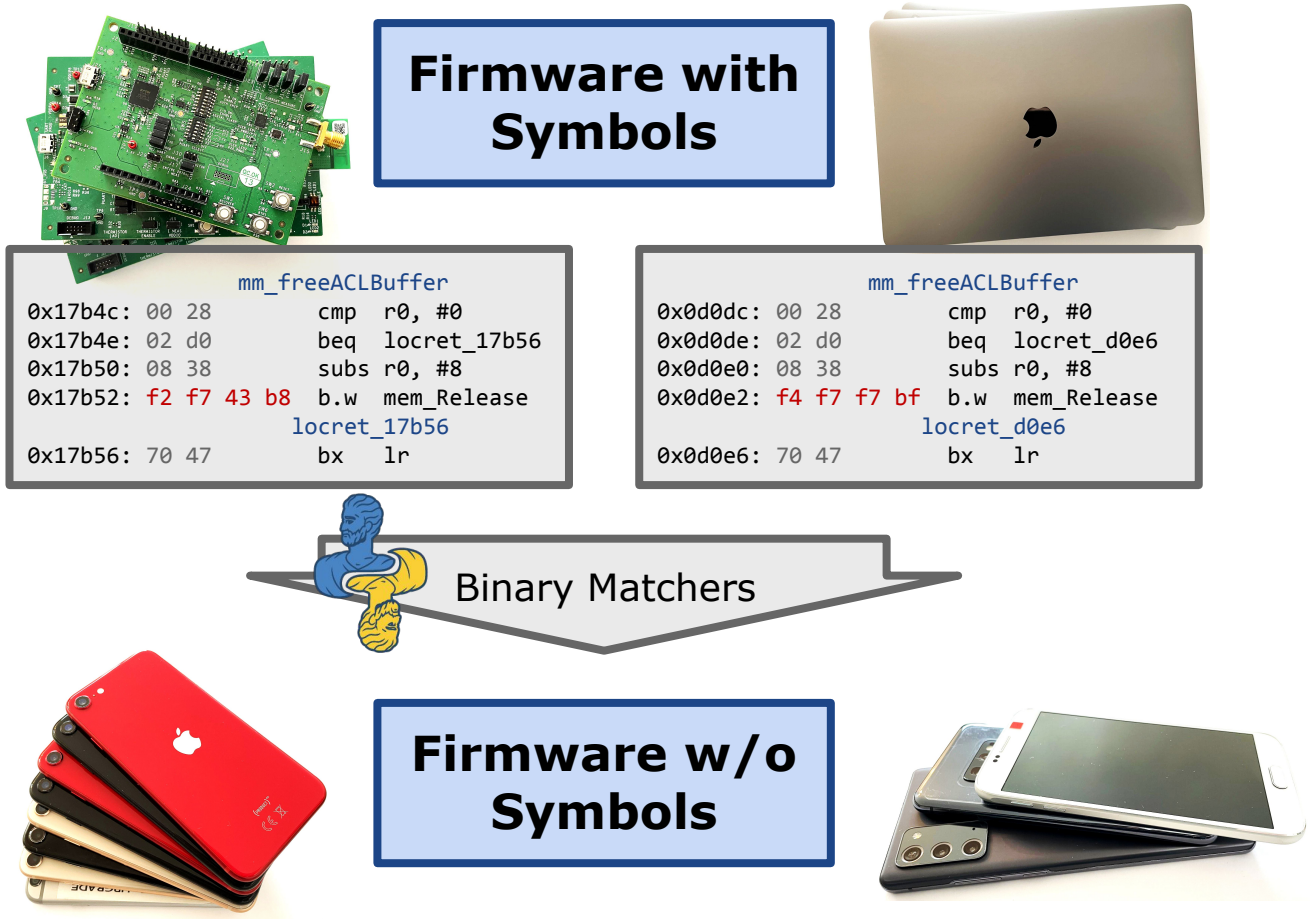
See https://arxiv.org/abs/2007.14266, 42nd IEEE Symposium on Security and Privacy 2021

Binary Diffing vs. Disassembler Performance



Polypyus
Raw Binary Diffing

Polypyus Bindiffer



Very fast binary differ that learns from a history of binaries and applies them to other binaries within seconds.

- Learn history from previously reverse-engineered firmware or leaked symbols.
- Works on raw binary format.

  Standalone!
  No IDA/… required.

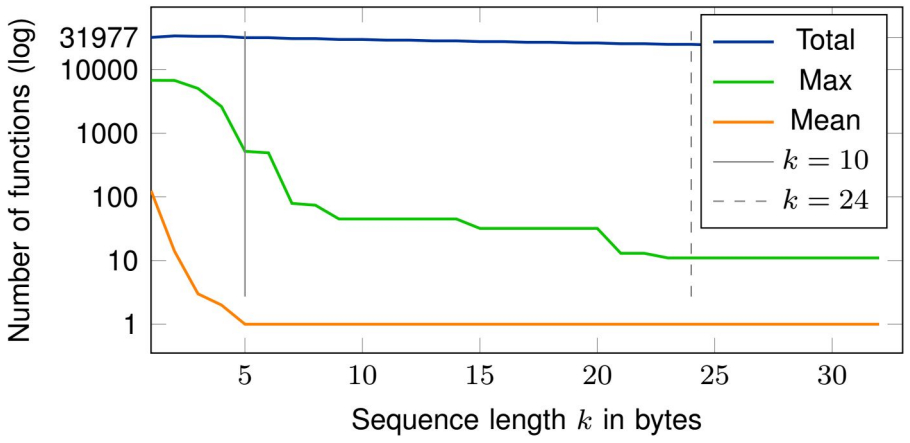
Demo Video

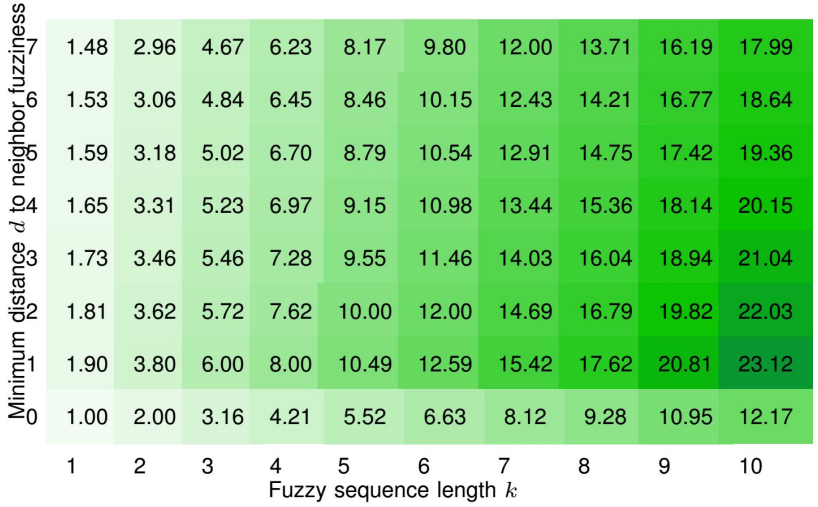
Function Similarity Matcher Cost Function

- Similar functions are used to create a matcher.
- A matcher should have a minimum length.

  `return 1;` is a valid function but …
- A matcher needs wildcard bytes.

  Branches, references, …
- Prevent long sequences of wildcard bytes within a matcher!





Function Identification

- Optional feature: learn common function prologues.

  First 8 bytes of a function, typically the same `push, mov, ...` instructions etc.

- Apply similar function prologues to identify functions.
- Indirectly improves diffing results by third-party binary differs.

```
sub_18CC4
0x18cc4: 2d e9 f0 41  push {r4-r8,lr}
0x18cca: 04 46        mov r4, r0
                      ...
```

```
sub_1CF7E
0x1cf7e: 70 b5        push {r4-r6,lr}
0x1cf80: 04 46        mov r4, r0
                      ...
```

```
sub_2F8B0
0x2f8b0: 70 b5        push {r4-r6,lr}
0x2f8b2: 05 46        mov r5, r0
                      ...
```

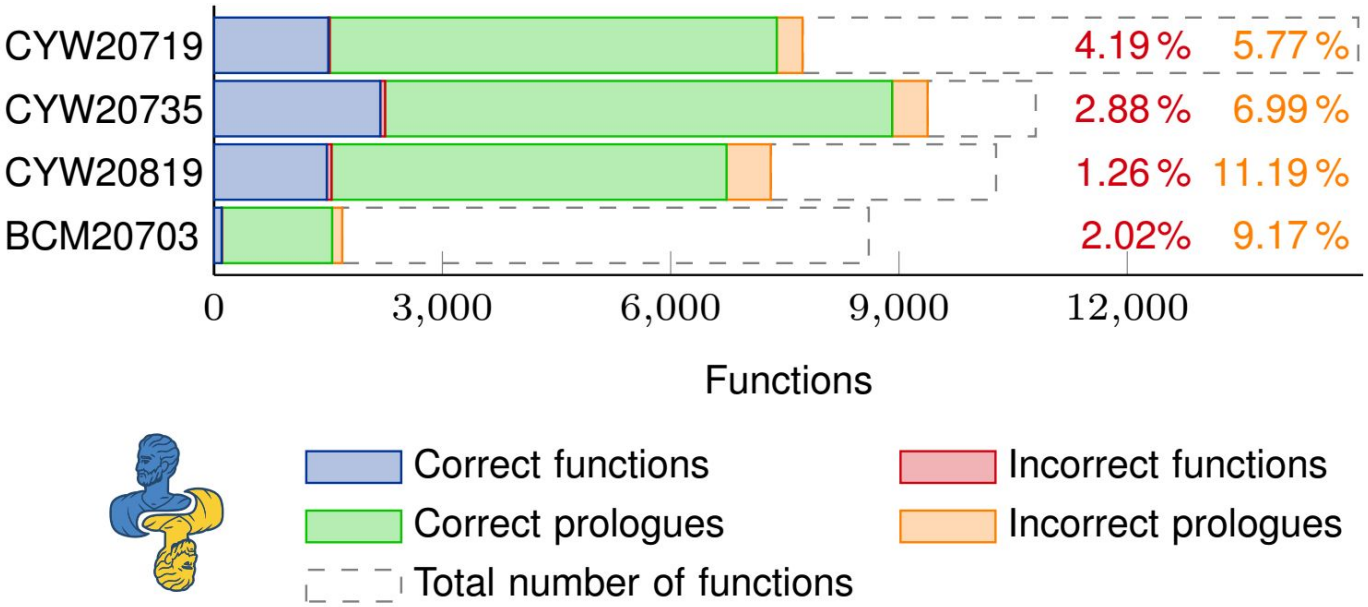
```
sub_18D8C
0x18d8c: 2d e9 f0 41  push {r4-r8,lr}
0x18d90: 04 46        mov r4, r0
                      ...
```

```
sub_1CDC0
0x1cdc0: 70 b5        push {r4-r6,lr}
0x1cdc2: 04 46        mov r4, r0
                      ...
```

```
sub_2615C
0x2615c: 70 b5        push {r4-r6,lr}
0x2615e: 05 46        mov r5, r0
                      ...
```

Polypyus Result Quality

- Very high function correctness :)
- 8 byte function prologues work okayish.
- All on a raw binary without disassembler.



Performs well within the same architecture :)

Chip	Device	Build Date	Symbols	Known Functions	ROM Size	ARM Core	Full Matches	+ Starts
BCM2046A2	iMac Late 2009	2007	—	—	0x31c00	ARM7TDMI-S (?)	0	10
BCM2070B0	MacBook 2011, Thinkpad T420	Jul 9 2008	—	—	0x57800	ARM7TDMI-S (?)	0	9
BCM20702A1	Asus USB Dongle	Feb (?) 2010	—	—	0x5fc00	ARM7TDMI-S	0	33
BCM4335C0	Google Nexus 5	Dec 11 2012	—	—	0x8f000	Cortex M3	266	2392
BCM4345B0	iPhone 6	Jul 15 2013	—	—	0xb3000	Cortex M3	371	2989
BCM20703A1	MacBook Pro early 2015	Dec 23 2013	—	—	0xc7000	Cortex M3	708	4350
BCM43430A1	Raspberry Pi 3/Zero W	Jun 2 2014	—	—	0x8f400	Cortex M3	209	1859
BCM4345C0	Raspberry Pi 3+/4	Aug 19 2014	—	—	0xc1c00	Cortex M3	307	2546
BCM4358A3	Samsung Galaxy S6, Nexus 6P	Oct 23 2014	—	—	0x8f000	Cortex M3	275	1978
BCM4345C1	iPhone SE	Jan 27 2015	—	—	0xb7000	Cortex M3	295	2477
BCM4364B0	MacBook/iMac 2017–2019	Aug 21 2015	—	—	0xd4000	Cortex M3	340	2804
BCM4355C0	iPhone 7	Sep 14 2015	—	—	0x90000	Cortex M3	231	1838
BCM20703A2	MacBook/iMac 2016–2017	Oct 22 2015	✓	8603	0xc7000	Cortex M3	101	1583
BCM4347B0	Samsung Galaxy S8	Jun 3 2016	—	—	0xf4800	Cortex M4	414	2720
BCM4347B1	iPhone 8/X/XR	Oct 11 2016	—	—	0xfc000	Cortex M3	695	3599
CYW20719B1	Evaluation board	Jan 17 2017	✓	15 036	0x1d2000	Cortex M4	1519	6214
CYW20735B1	Evaluation board	Jan 18 2018	✓	10 791	0x14f000	Cortex M4	2241	7129
CYW20819A1	Evaluation board	May 22 2018	✓	10 276	0xf6800	Cortex M4	1543	5772
BCM4375B1	Samsung Galaxy S10/S20	Apr 13 2018	—	Canaries	0x130000	Cortex M4 (?)	14	362
BCM4378B1	iPhone 11/SE2	Oct 25 2018	Strings	Canaries	0x132800	Cortex M4 (?)	15	380

Polypyus is provided with inputs from four symbolicated firmwares. Matching results are for function identification and starts with *Polypyus* defaults: 24 B minimum length for function identification and optional 8 B prologues for function starts.

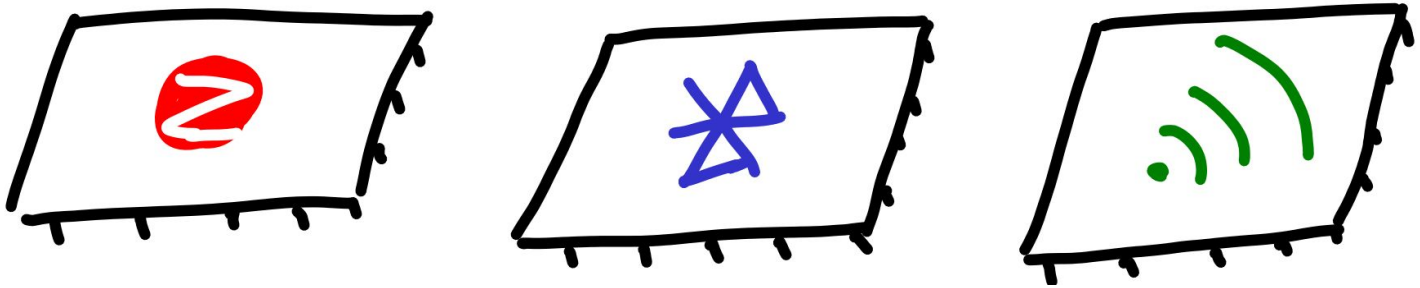
PDOM

```
~/Documents/WICED-Studio-6.2$ find . -iname *pdom
./.metadata/.plugins/org.eclipse.cdt.core/20719-B1_Bluetooth.1496794091727.pdom
./.metadata/.plugins/org.eclipse.cdt.core/20729-B0_ZigBee.1472077731428.pdom
./.metadata/.plugins/org.eclipse.cdt.core/BCM20706-A2.1462220119850.pdom
./.metadata/.plugins/org.eclipse.cdt.core/20739-B1_Bluetooth_ZigBee.1496794165427.pdom
./.metadata/.plugins/org.eclipse.cdt.core/20719-B0_Bluetooth.1498694110221.pdom
./.metadata/.plugins/org.eclipse.cdt.core/43012-C0_Bluetooth.1502317446335.pdom
./.metadata/.plugins/org.eclipse.cdt.core/43xxx_Wi-Fi.1472674019671.pdom
./.metadata/.plugins/org.eclipse.cdt.core/BCM20735-B0.1462220135459.pdom
./.metadata/.plugins/org.eclipse.cdt.core/20721-B1_Bluetooth.1522425295912.pdom
./.metadata/.plugins/org.eclipse.cdt.core/20706-A2_Bluetooth.1472077679756.pdom
./.metadata/.plugins/org.eclipse.cdt.core/20729-B1_ZigBee.1498694475383.pdom
./.metadata/.plugins/org.eclipse.cdt.core/BCM20739-B0.1462220149391.pdom
./.metadata/.plugins/org.eclipse.cdt.core/20719-B0_Bluetooth.1472077700287.pdom
./.metadata/.plugins/org.eclipse.cdt.core/43xxx_WiFi.1472663273205.pdom
./.metadata/.plugins/org.eclipse.cdt.core/BCM20719-B0.1470329520648.pdom
./.metadata/.plugins/org.eclipse.cdt.core/20735-B1_Bluetooth.1522425326768.pdom
./.metadata/.plugins/org.eclipse.cdt.core/43xxx_Wi-Fi.1496793890291.pdom
./.metadata/.plugins/org.eclipse.cdt.core/BCM20729-B0.1471993583020.pdom
./.metadata/.plugins/org.eclipse.cdt.core/20735-B0_Bluetooth.1472077718968.pdom
```

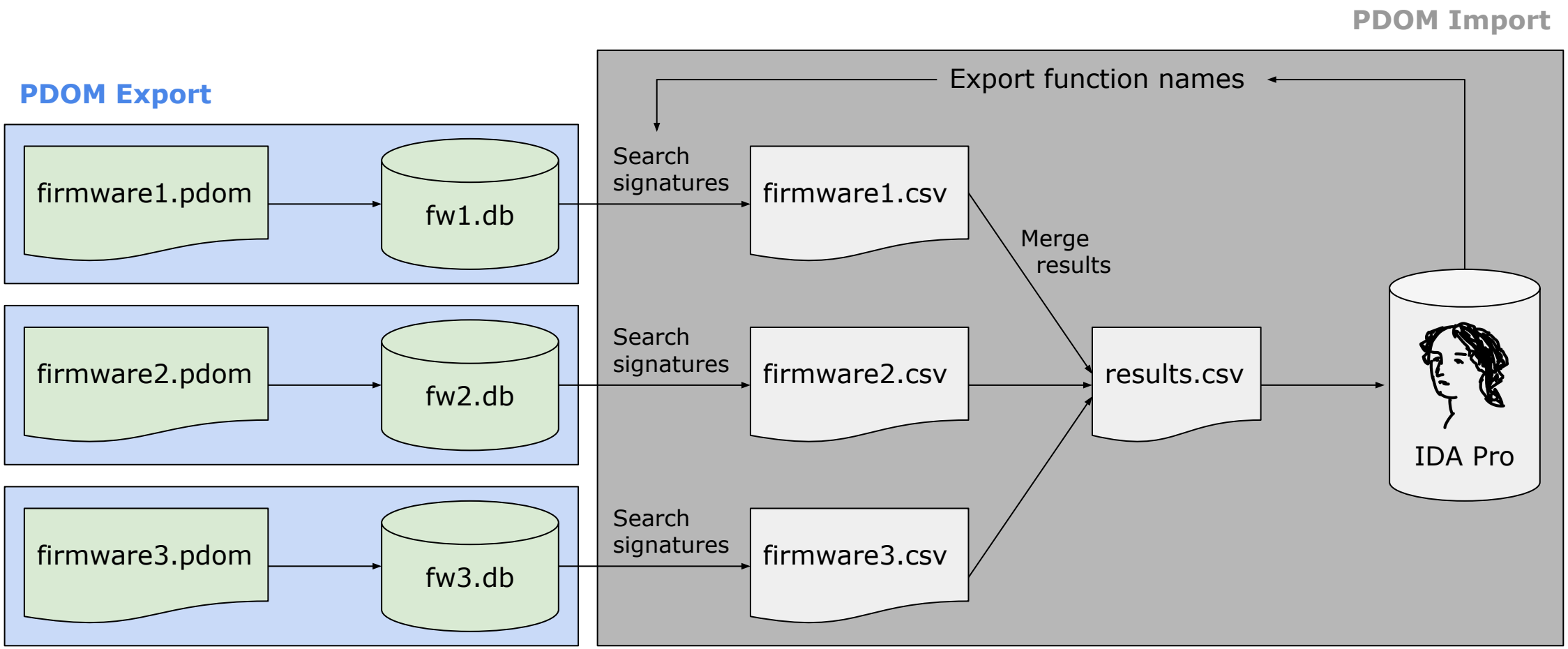
What is PDOM?

- PDOM: **Persistent Document Object Model**
- Special format generated by the Eclipse compiler, e.g. to speed up search for function name completion etc.
- Can have different verbosity levels of cached information.

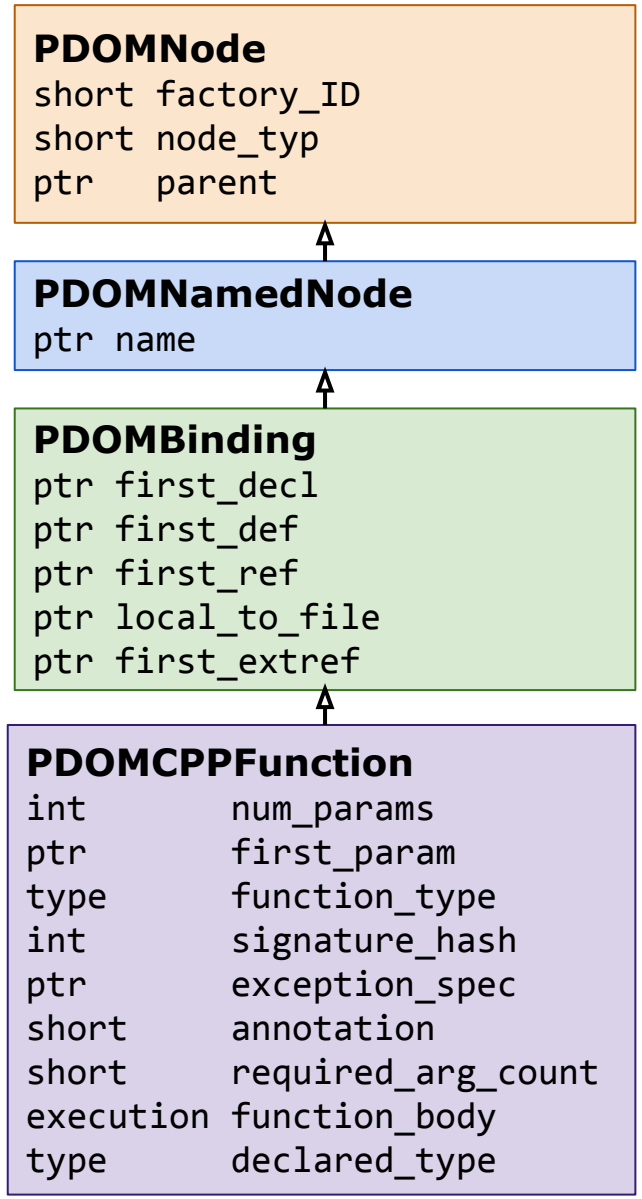
… whatever, Cypress forgot to delete them from their SDK 🎉
Even more Eclipse projects than chips officially supported by WICED Studio!



Utilizing PDOM in Reverse

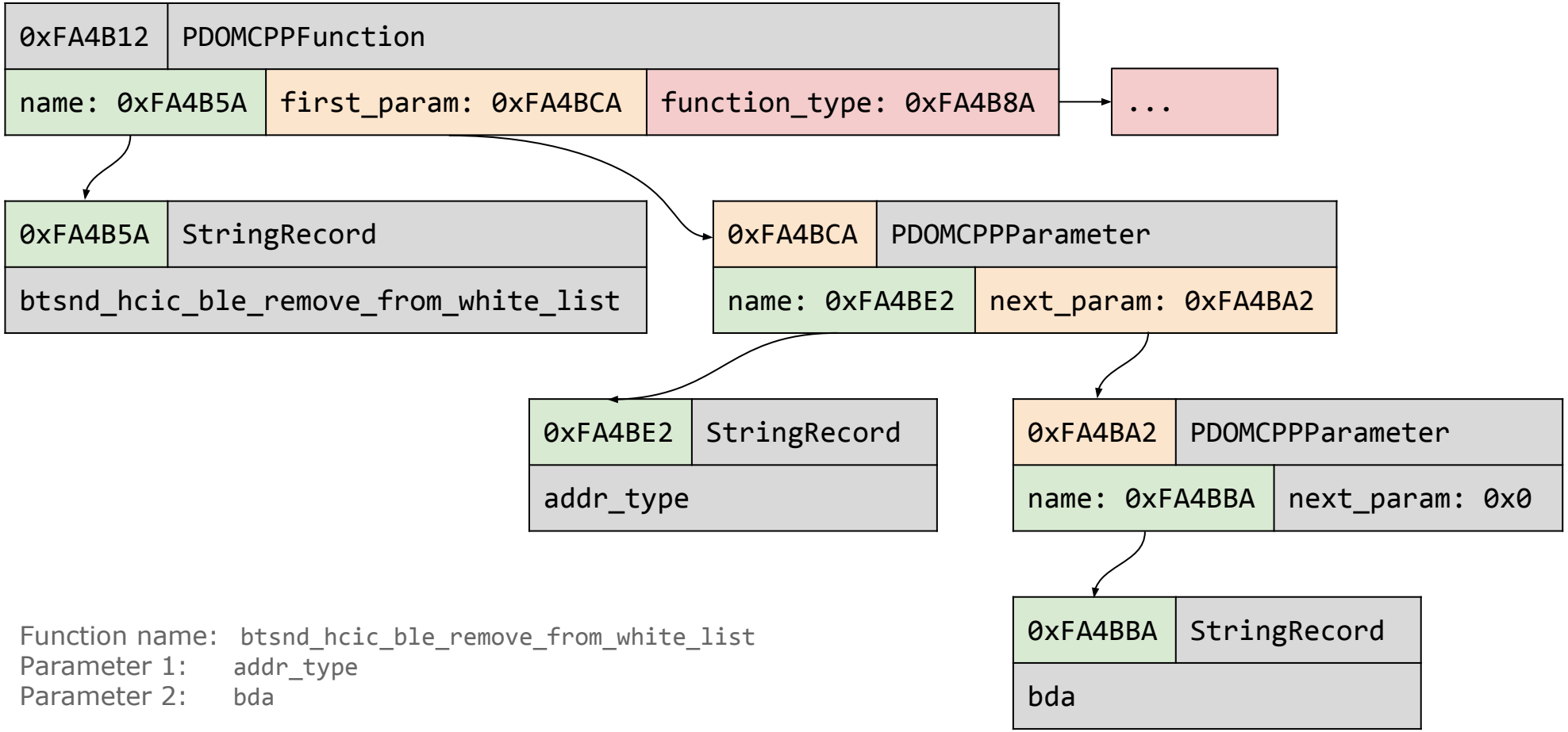


Basic PDOM Structure



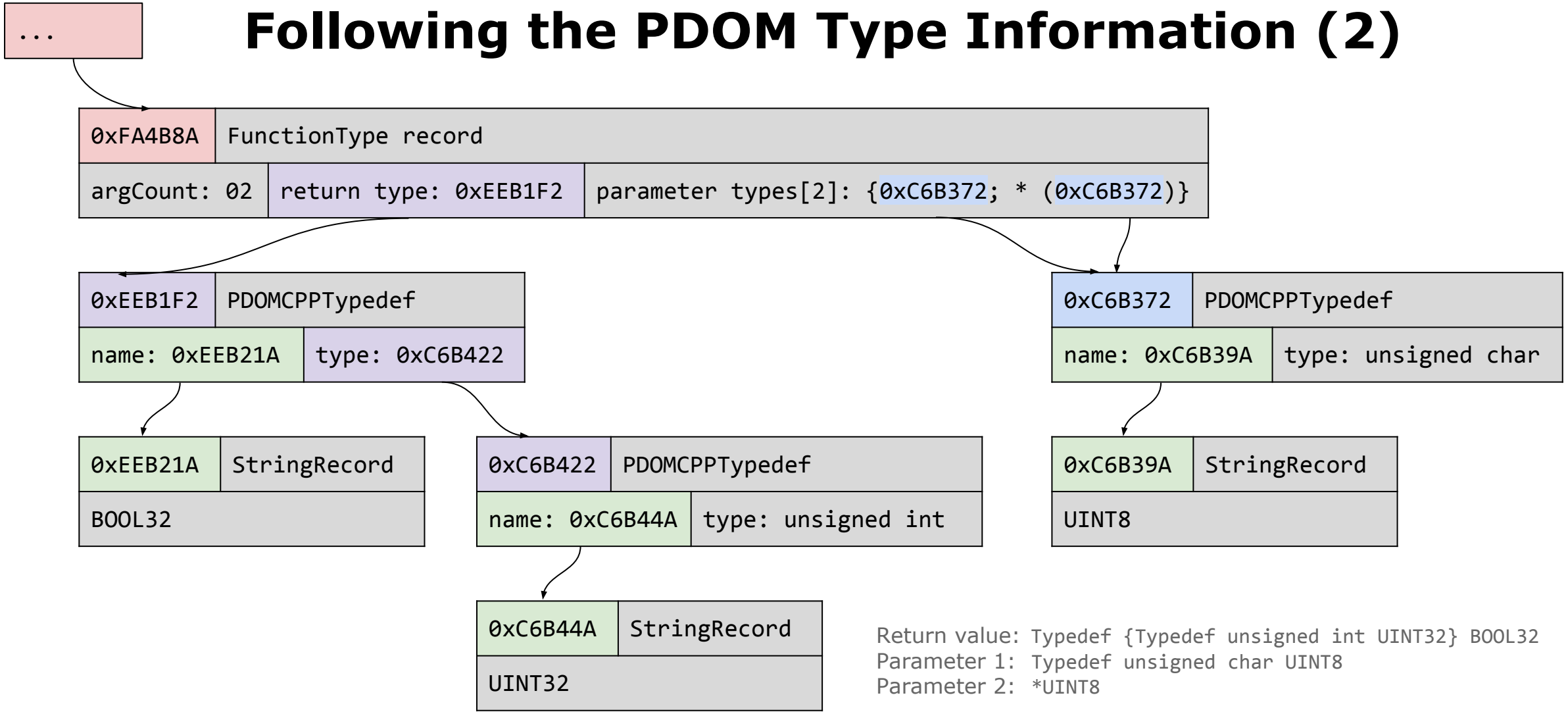
Offset	Value	Bytes
0	factory_ID	00 01
2	node_typ	01 07
4	parent	00 11 9D 98
8	name	00 1F 49 6B
12	first_decl	00 1F 49 7E
	...	
32	num_params	00 00 00 02
36	first_param	00 1F 49 79
	...	
64	declared_type	00 00 00 00 00 00

Following the PDOM Type Information (1)



Function name: `btsnd_hcic_ble_remove_from_white_list`
Parameter 1: `addr_type`
Parameter 2: `bda`

Following the PDOM Type Information (2)



Return value: `Typedef {Typedef unsigned int UINT32} BOOL32`
Parameter 1: `Typedef unsigned char UINT8`
Parameter 2: `*UINT8`

Reconstructed function signature:
`BOOL32 btsnd_hcic_ble_remove_from_white_list(UNIT8 addr_type, UINT8* bda)`

Combining Multiple PDOMs

Reference PDOM:

```
BOOLEAN BTM_BleUpdateAdvFilterPolicy(BTM_BLE_AFP adv_policy)
BOOLEAN BTM_BleUpdateAdvFilterPolicy(UNKNOWN adv_policy)
BOOL32  BTM_BleUpdateAdvFilterPolicy(UNKNOWN adv_policy)

BOOL32  BTM BleUpdateAdvFilterPolicy(BTM_BLE_AFP adv_policy)
```

Apply more precise information to reference PDOM

Priority	Input	BCM20735-B1	BCM20719-B1	BCM20706-A2
1	One signature	2888	4320	1816
2	Signature in reference database	858	829	369
3	Most parameters	65	83	39
4	Most matches in PDOM	743	1035	587
5	Random	40	97	22
Total		**4594**	**6364**	**2833**

Successful Function Signature Imports

Eclipse PDOM Input		CYW20735B1		CYW20719B1		CYW20706A2	
Chip	**Build Date**	**Same Names**	**Signature Similarity**	**Same Names**	**Signature Similarity**	**Same Names**	**Signature Similarity**
CYW20706A2	May 2 2016	2236	74.20 %	2377	72.44 %	1817	88.51 %
CYW20706A2	Aug 24 2016	1334	91.95 %	998	87.46 %	446	100 %
CYW20719B0	Aug 4 2016	2345	94.90 %	2595	93.94 %	1819	69.97 %
CYW20719B0	Aug 24 2016	767	95.56 %	820	95.79 %	382	70.53 %
CYW20719B0	Jun 28 2017	1138	98.42 %	863	97.58 %	410	76.55 %
CYW20719B1	Jun 7 2017	1271	97.75 %	994	100 %	431	80.35 %
CYW20721B1	May 30 2018	1040	97.94 %	788	99.19 %	235	95.03 %
CYW20729B0	Aug 23 2016	3476	81.24 %	5749	79.43 %	2603	69.49 %
CYW20735B0	May 2 2016	2255	80.31 %	2500	78.58 %	1805	68.73 %
CYW20735B0	Aug 24 2016	2132	97.34 %	2030	95.16 %	1211	76.78 %
CYW20735B1	May 30 2018	1249	100 %	991	97.35 %	417	78.66 %
CYW20739B0	May 2 2016	2294	80.21 %	2542	78.62 %	1811	68.81 %
20729B0 Zigbee	Aug 24 2016	79	93.16 %	263	90.81 %	43	93.65 %
20729B1 Zigbee	Jun 19 2017	171	93.28 %	376	92.71 %	50	94.56 %
20739B1 Zigbee	Jun 7 2017	1547	93.17 %	1007	95.33 %	439	72.75 %
43xxx Wi-Fi	5pm, Aug 31 2016	673	88.22 %	1153	88.46 %	281	89.55 %
43xxx Wi-Fi	8pm, Aug 31 2016	398	92.49 %	687	92.24 %	94	71.48 %
43xxx Wi-Fi	Jun 7 2017	1714	84.10 %	2354	87.32 %	1008	85.38 %
43012C0 Wi-Fi	Aug 9 2017	1011	96.06 %	707	93.00 %	311	91.55 %
		4594/11 810	91.07 %	6364/14 725	90.28 %	2833/8603	81.18 %
Combined Same Name/Total Functions		**38.90 %**		**43.22 %**		**32.93 %**	

Persistent Document Object Model (PDOM) files extracted from *Cypress WICED Studio 6.2* and *6.4*. Not all chips listed here are publicly available as evaluation kit, thus, some binaries are missing in the function identification analysis. Projects that were built multiple times contain partially redundant PDOM files.

Before

```
int __fastcall gatt_update_app_use_link_flag(int a1, int a2, int a3, int a4){
      int result; // r0
       result = gatt_update_app_hold_link_status(a1);
       if ( a4 )  {
            if ( a2 ){
                  result = *(unsigned __int16 *)(a2 + 28);
                  if ( result == 4 ){
                        result = BTM_GetHCIConnHandle();
                        if ( result != 0xFFFF ){
                                    if ( a3 || (result = gatt_num_apps_hold_link(a2)) == 0 )
                                    result = GATT_SetIdleTimeout(a2 + 13);
                        }
                  }
            }
      }
 return result;
}
```

After

```
void __cdecl gatt_update_app_use_link_flag(tGATT_IF gatt_if, tGATT_TCB *p_tcb, BOOL32 is_add, BOOL32 check_acl_link){
  gatt_update_app_hold_link_status(gatt_if);
  if ( check_acl_link
      && p_tcb
      && *(_WORD *)((char *)&p_tcb[5].__vtbl + 3) == 4
      && BTM_GetHCIConnHandle() != 0xFFFF
      && (is_add || !gatt_num_apps_hold_link(p_tcb)) )
  {
      GATT_SetIdleTimeout((char *)&p_tcb[2].__vtbl + 3);
  }
}
```

Demo Video??

Q&A

Twitter: @naehrdine, @seemoolab

jiska@bluetooth.lol

https://github.com/seemoo-lab/polypyus

Full Workflow

IDA Pro + Polypyus

- Start recursive analysis at reset vector.
- Import Polypyus results including prologues.
- Perform linear sweep.

IDA Pro without Polypyus

- Perform linear sweep.

Polypyus even helps identifying functions that IDA Pro's linear sweep cannot identify.