Google

What's Up with WhatsApp

A Detailed Walk Through of Reverse Engineering CVE-2019-3568

Maddie Stone
@maddiestone
Jailbreak Security Summit 2019

Who am I? - Maddie Stone (she/her)

- Security Researcher on Project Zero
  - Current Focus: In-the-wild use of 0-days
- Previously: Google's Android Security team
- Speaker at BlackHat USA, REcon, OffensiveCon, & more!
- BS in Computer Science, Russian, & Applied Math, MS in Computer Science

@maddiestone

Goal

The goal of this presentation is not to just tell you about the bug and exploit, but walk through the reversing process of how to learn through the bug.

Agenda aka walking through the RE process

- Basics about the bug
- Patch diffing tooling
- Static analysis
- Dynamic analysis with Frida
- Conclusion

What We Know about CVE-2019-3568

- Facebook's Advisory for CVE-2019-3568
  - *"A buffer overflow vulnerability in WhatsApp VOIP stack allowed remote code execution via specially crafted series of RTCP packets sent to a target phone number."*

What We Know about CVE-2019-3568

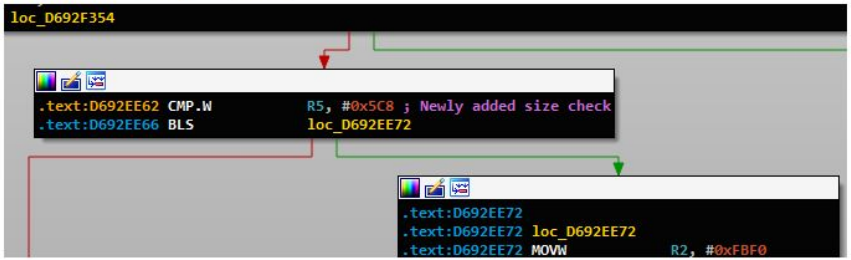
- Facebook's Advisory for CVE-2019-3568
  - *"A buffer overflow vulnerability in WhatsApp VOIP stack allowed remote code execution via specially crafted series of RTCP packets sent to a target phone number."*
- Checkpoint Research published blog highlighting two changes from the vuln version to the patched
  - "The NSO WhatsApp Vulnerability - This Is How It Happened"

What We Know about CVE-2019-3568

- Facebook's Advisory for CVE-2019-3568
  - *"A buffer overflow vulnerability in WhatsApp VOIP stack allowed remote code execution via specially crafted series of RTCP packets sent to a target phone number."*

Size Check #1

The patched function is a major RTCP handler function, and the added fix can be found right at its start. The added check verifies the length argument against a maximal size of 1480 bytes (0x5C8).



During our debugging session we confirmed that this is indeed a major function in the RTCP module and that it is called even before the WhatsApp voice call is answered.

changes from the vuln

Happened"

What We Know about CVE-2019-3568

- Facebook's Advisory for
  - *"A buffer overflow v*
    code execution via
    target phone numb

Size Check #2

In the flow between the two functions we can see that the same length variable is now used twice during the newly added sanitation checks (marked in blue):

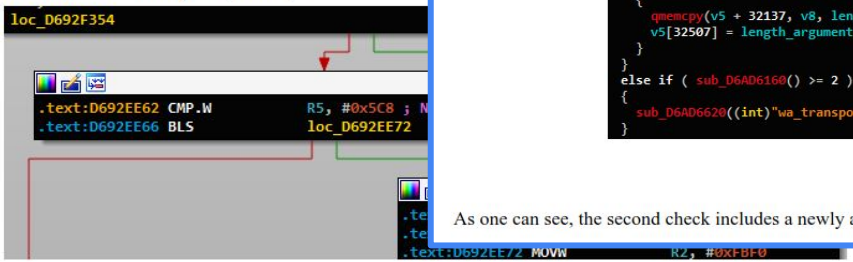
1. Validation that the packet's length field doesn't exceed the length.
2. Additional check that the length is one again <= 1480, right before a memory copy.

```
if ( packet_length_field <= length_argument )
{
  v18 = (void (__fastcall *)(int, int *, unsigned int, int, unsigned int))v5[4650];
  if ( v18 )
  {
    v19 = v5[4648];
    v20 = sub_D6ADAD08(v8[1]);
    v18(v19, v8, length_argument, v13, v20);
    sub_D69175B4(v8, length_argument, &v23);
    v21 = 12;
    if ( !v13 )
      v21 = 5;
    sub_D692C2DC(v5, v21, &v23, 4);
  }
  else if ( length_argument <= 0x5C8 && a5 && (v11 & 0xFE00) == 51200 )
  {
    qmemcpy(v5 + 32137, v8, length_argument);
    v5[32507] = length_argument;
  }
}
else if ( sub_D6AD6160() >= 2 )
{
  sub_D6AD6620((int)"wa_transport.cc", "RTCP payload length overflow %d, skip", packet_length_field);
}
```

As one can see, the second check includes a newly added log string that specifically say it is a sanitation check to avoid a possible overflow.

Size Check #1

The patched function is a major RTCP handler function, and the added
argument against a maximal size of 1480 bytes (0x5C8).



During our debugging session we confirmed that this is indeed a major function in the RTCP module and that it is called even before the WhatsApp voice call is answered.

Samples

- Vulnerable WhatsApp application
  - Version 2.19.133
  - 763ab8444e085bd26336408e72ca4de3a36034d53c3e033f8eb39d8d90997707
- Patched WhatsApp application
  - Version 2.19.134
  - ee09262fa8b535b5592960ca5ab41e194f632419f8a80ef2e41d36efdbe13f88

Patch Diffing Tooling

Tools under Test

- DarunGrim
- BinDiff
- Diaphora
- Radare2 (radiff2)

Do the Binary Diffing Tools Highlight This Change?

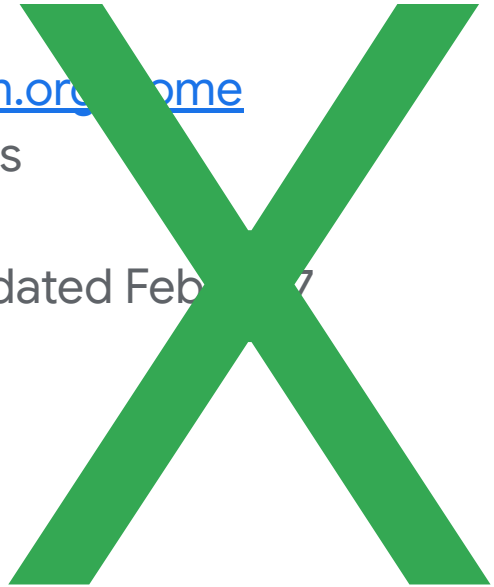
- “Size Check #1”
  - sub_51E34 in patched
- “Size Check #2”
  - sub_52D0C in patched

DarunGrim

- http://www.darungrim.org/Home
- Only runs on Windows
- Supports IDA 5.6
- Open source, last updated Feb 2017

DarunGrim

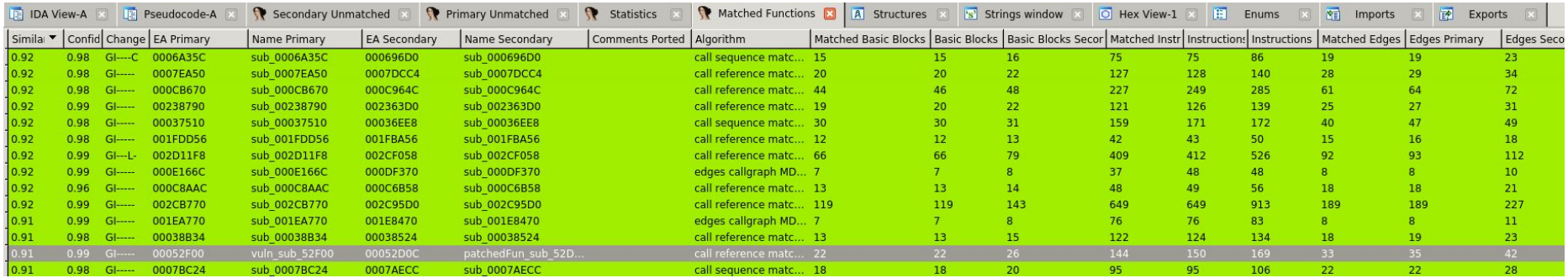
- http://www.darungrim.org/home
- Only runs on Windows
- Supports IDA 5.6
- Open source, last updated Feb 7



BinDiff

- https://www.zynamics.com/bindiff/manual/
- The OG
- Plugins for IDA 7.x
- Not open source

BinDiff

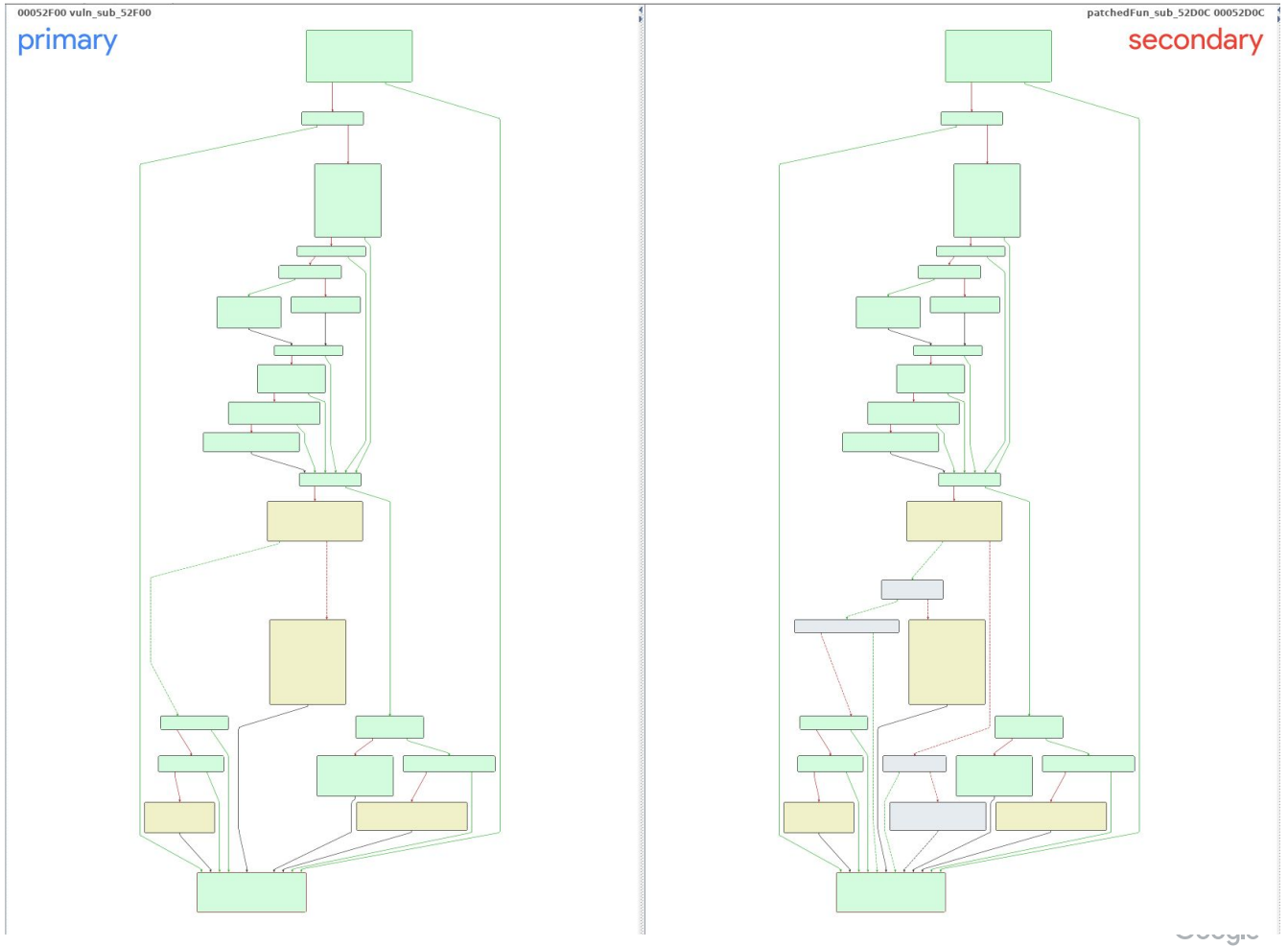


IDA View-A | Pseudocode-A | Secondary Unmatched | Primary Unmatched | Statistics | Matched Functions | Structures | Strings window | Hex View-1 | Enums | Imports | Exports

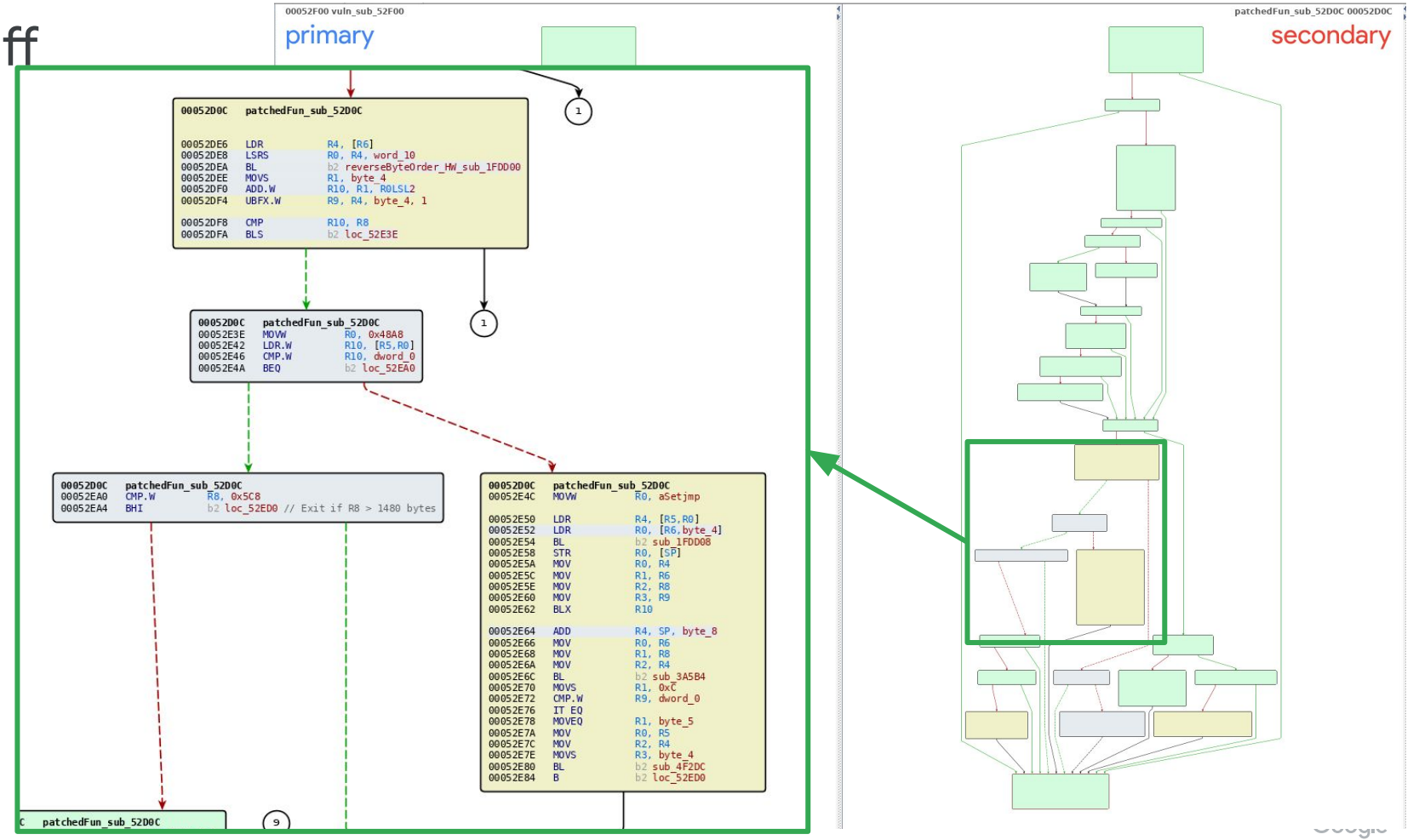
Similar	Confid	Change	EA Primary	Name Primary	EA Secondary	Name Secondary	Comments Ported	Algorithm	Matched Basic Blocks	Basic Blocks	Basic Blocks Secor	Matched Instr	Instructions	Instructions	Matched Edges	Edges Primary	Edges Seco
0.92	0.98	GI----C	0006A35C	sub_0006A35C	000696D0	sub_000696D0		call sequence matc...	15	15	16	75	75	86	19	19	23
0.92	0.98	GI-----	0007EA50	sub_0007EA50	0007DCC4	sub_0007DCC4		call reference matc...	20	20	22	127	128	140	28	29	34
0.92	0.98	GI-----	000CB670	sub_000CB670	000C964C	sub_000C964C		call reference matc...	44	46	48	227	249	285	61	64	72
0.92	0.99	GI-----	00238790	sub_00238790	002363D0	sub_002363D0		call reference matc...	19	20	22	121	126	139	25	27	31
0.92	0.98	GI-----	00037510	sub_00037510	00036EE8	sub_00036EE8		call sequence matc...	30	30	31	159	171	172	40	47	49
0.92	0.98	GI-----	001FDD56	sub_001FDD56	001FBA56	sub_001FBA56		call reference matc...	12	12	13	42	43	50	15	16	18
0.92	0.99	GI---L-	002D11F8	sub_002D11F8	002CF058	sub_002CF058		call reference matc...	66	66	79	409	412	526	92	93	112
0.92	0.99	GI-----	000E166C	sub_000E166C	000DF370	sub_000DF370		edges callgraph MD...	7	7	8	37	48	48	8	8	10
0.92	0.96	GI-----	000C8AAC	sub_000C8AAC	000C6B58	sub_000C6B58		call reference matc...	13	13	14	48	49	56	18	18	21
0.92	0.99	GI-----	002CB770	sub_002CB770	002C95D0	sub_002C95D0		call reference matc...	119	119	143	649	649	913	189	189	227
0.91	0.99	GI-----	001EA770	sub_001EA770	001E8470	sub_001E8470		edges callgraph MD...	7	7	8	76	76	83	8	8	11
0.91	0.98	GI-----	00038B34	sub_00038B34	00038524	sub_00038524		call reference matc...	13	13	15	122	124	134	18	19	23
0.91	0.99	GI-----	00052F00	vuln_sub_52F00	00052D0C	patchedFun_sub_52D...		call reference matc...	22	22	26	144	150	169	33	35	42
0.91	0.98	GI-----	0007BC24	sub_0007BC24	0007AECC	sub_0007AECC		call sequence matc...	18	18	20	95	95	106	22	22	28

- BinDiff opens 4 tabs automatically in IDA showing Matched/Unmatched Funcs, etc.
- Primary is the IDB you run BinDiff from and Secondary is the IDB you select.
  - Primary = vuln, Secondary = patched

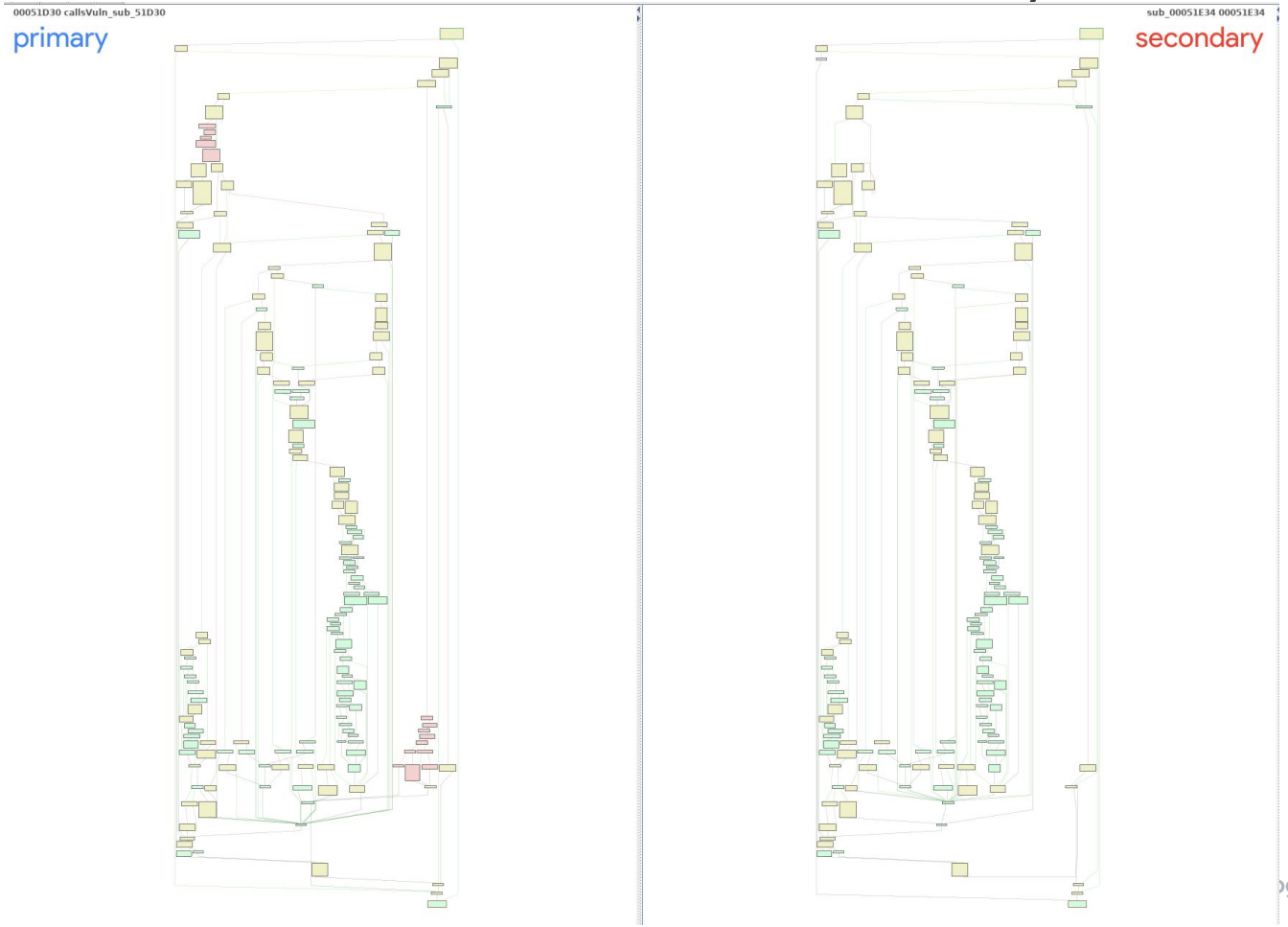
BinDiff


00052F00 vuln_sub_52F00
primary
patchedFun_sub_52D0C 00052D0C
secondary

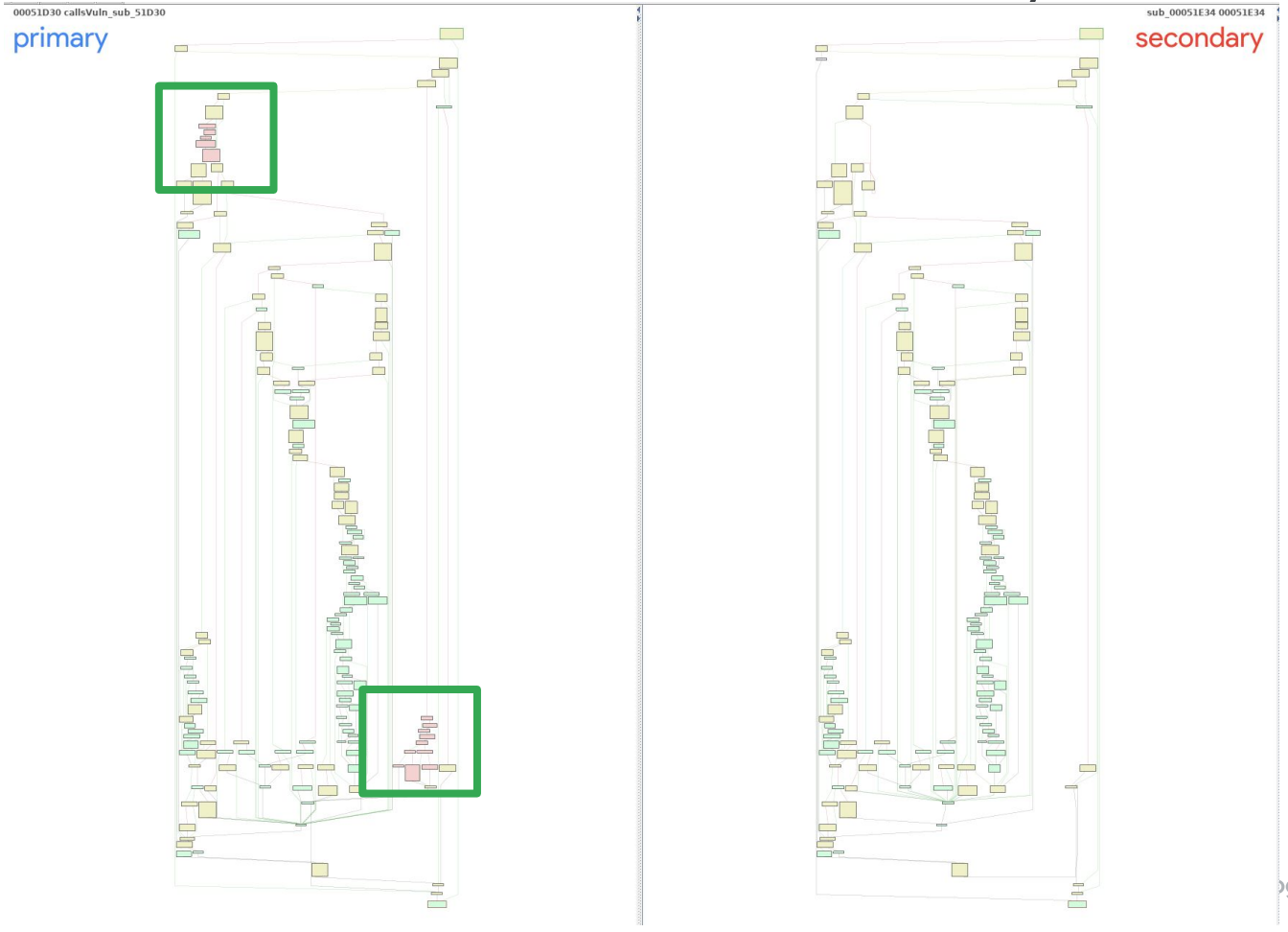
BinDiff


00052F00 vuln_sub_52F00
primary
patchedFun_sub_52D0C 00052D0C
secondary
00052D0C patchedFun_sub_52D0C
00052DE6 LDR R4, [R6]
00052DE8 LSRS R0, R4, word_10
00052DEA BL b2 reverseByteOrder_HW_sub_1FDD00
00052DEE MOVS R1, byte_4
00052DF0 ADD.W R10, R1, R0LSL2
00052DF4 UBFX.W R9, R4, byte_4, 1
00052DF8 CMP R10, R8
00052DFA BLS b2 loc_52E3E
1
00052D0C patchedFun_sub_52D0C
00052E3E MOVW R0, 0x48A8
00052E42 LDR.W R10, [R5,R0]
00052E46 CMP.W R10, dword_0
00052E4A BEQ b2 loc_52EA0
1
00052D0C patchedFun_sub_52D0C
00052EA0 CMP.W R8, 0x5C8
00052EA4 BHI b2 loc_52ED0 // Exit if R8 > 1480 bytes
00052D0C patchedFun_sub_52D0C
00052E4C MOVW R0, aSetjmp
00052E50 LDR R4, [R5,R0]
00052E52 LDR R0, [R6,byte_4]
00052E54 BL b2 sub_1FDD08
00052E58 STR R0, [SP]
00052E5A MOV R0, R4
00052E5C MOV R1, R6
00052E5E MOV R2, R8
00052E60 MOV R3, R9
00052E62 BLX R10
00052E64 ADD R4, SP, byte_8
00052E66 MOV R0, R6
00052E68 MOV R1, R8
00052E6A MOV R2, R4
00052E6C BL b2 sub_3A5B4
00052E70 MOVS R1, 0xC
00052E72 CMP.W R9, dword_0
00052E76 IT EQ
00052E78 MOVEQ R1, byte_5
00052E7A MOV R0, R5
00052E7C MOV R2, R4
00052E7E MOVS R3, byte_4
00052E80 BL b2 sub_4F2DC
00052E84 B b2 loc_52ED0
C patchedFun_sub_52D0C
9

BinDiff: Size Check #1 - Matches Functions Correctly



BinDiff: Size Check #1 - Matches Functions Correctly



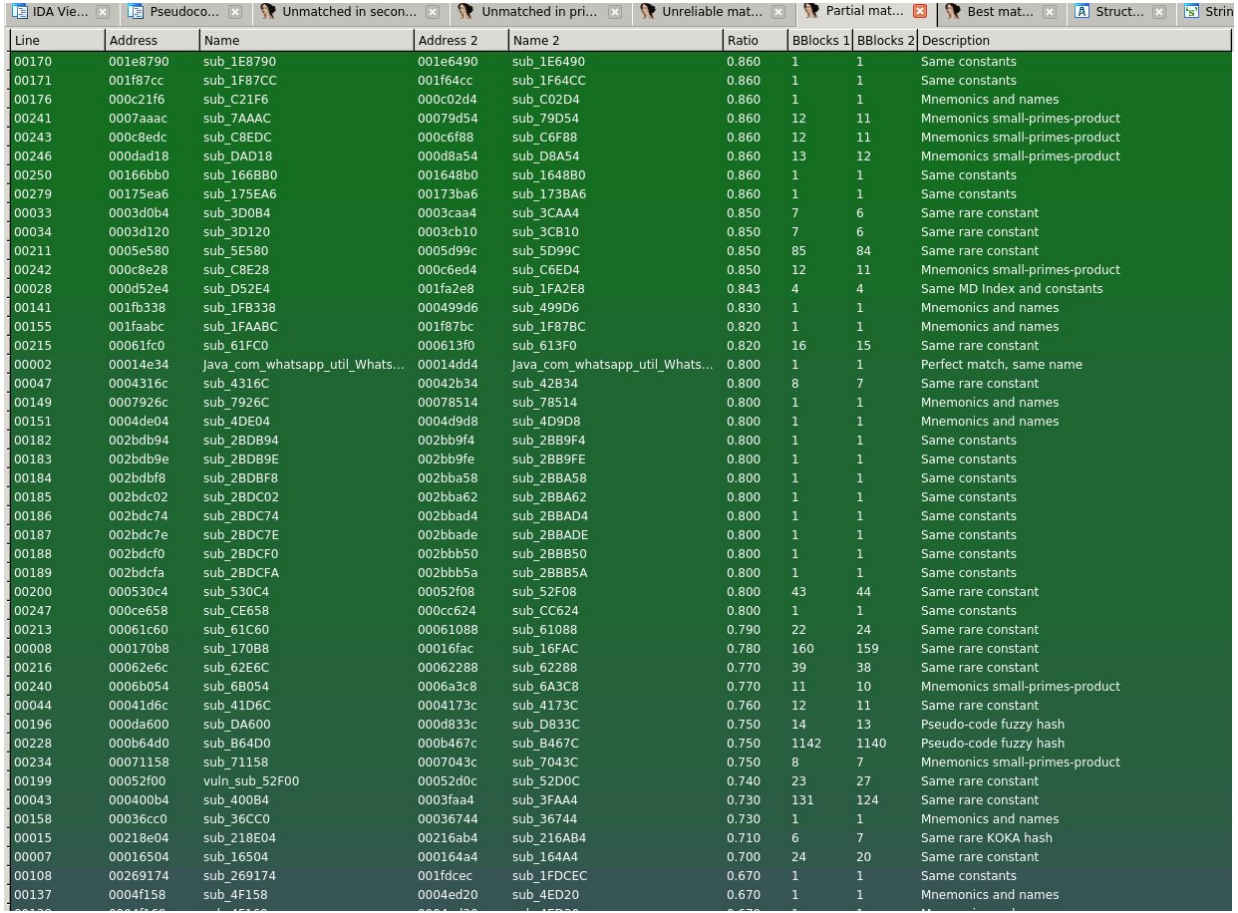
BinDiff

- The matching is good
- The UI for highlighting changes between the two functions is clear and obvious
- It is not obvious though which changes in the matched functions list may be important
- No support for decompiler
- UI is outside of IDA
- Seems to not get caught by name changes, offset changes, etc.

Diaphora

- diaphora.re
- Open-source and still supported (last update 2 weeks ago)
- Currently supports IDA 7.1-7.3
- Ghidra support in development and Binary Ninja support planned

Diaphora


IDA Vie... | Pseudoco... | Unmatched in secon... | Unmatched in pri... | Unreliable mat... | Partial mat... | Best mat... | Struct... | Strin

Line	Address	Name	Address 2	Name 2	Ratio	BBlocks 1	BBlocks 2	Description
00170	001e8790	sub_1E8790	001e6490	sub_1E6490	0.860	1	1	Same constants
00171	001f87cc	sub_1F87CC	001f64cc	sub_1F64CC	0.860	1	1	Same constants
00176	000c21f6	sub_C21F6	000c02d4	sub_C02D4	0.860	1	1	Mnemonics and names
00241	0007aaac	sub_7AAAC	00079d54	sub_79D54	0.860	12	11	Mnemonics small-primes-product
00243	000c8edc	sub_C8EDC	000c6f88	sub_C6F88	0.860	12	11	Mnemonics small-primes-product
00246	000dad18	sub_DAD18	000d8a54	sub_D8A54	0.860	13	12	Mnemonics small-primes-product
00250	00166bb0	sub_166BB0	001648b0	sub_1648B0	0.860	1	1	Same constants
00279	00175ea6	sub_175EA6	00173ba6	sub_173BA6	0.860	1	1	Same constants
00033	0003d0b4	sub_3D0B4	0003caa4	sub_3CAA4	0.850	7	6	Same rare constant
00034	0003d120	sub_3D120	0003cb10	sub_3CB10	0.850	7	6	Same rare constant
00211	0005e580	sub_5E580	0005d99c	sub_5D99C	0.850	85	84	Same rare constant
00242	000c8e28	sub_C8E28	000c6ed4	sub_C6ED4	0.850	12	11	Mnemonics small-primes-product
00028	000d52e4	sub_D52E4	001fa2e8	sub_1FA2E8	0.843	4	4	Same MD Index and constants
00141	001fb338	sub_1FB338	000499d6	sub_499D6	0.830	1	1	Mnemonics and names
00155	001faabc	sub_1FAABC	001f87bc	sub_1F87BC	0.820	1	1	Mnemonics and names
00215	00061fc0	sub_61FC0	000613f0	sub_613F0	0.820	16	15	Same rare constant
00002	00014e34	Java_com_whatsapp_util_Whats...	00014dd4	Java_com_whatsapp_util_Whats...	0.800	1	1	Perfect match, same name
00047	0004316c	sub_4316C	00042b34	sub_42B34	0.800	8	7	Same rare constant
00149	0007926c	sub_7926C	00078514	sub_78514	0.800	1	1	Mnemonics and names
00151	0004de04	sub_4DE04	0004d9d8	sub_4D9D8	0.800	1	1	Mnemonics and names
00182	002bdb94	sub_2BDB94	002bb9f4	sub_2BB9F4	0.800	1	1	Same constants
00183	002bdb9e	sub_2BDB9E	002bb9fe	sub_2BB9FE	0.800	1	1	Same constants
00184	002bdbf8	sub_2BDBF8	002bba58	sub_2BBA58	0.800	1	1	Same constants
00185	002bdc02	sub_2BDC02	002bba62	sub_2BBA62	0.800	1	1	Same constants
00186	002bdc74	sub_2BDC74	002bbad4	sub_2BBAD4	0.800	1	1	Same constants
00187	002bdc7e	sub_2BDC7E	002bbade	sub_2BBADE	0.800	1	1	Same constants
00188	002bdcf0	sub_2BDCF0	002bbb50	sub_2BBB50	0.800	1	1	Same constants
00189	002bdcfa	sub_2BDCFA	002bbb5a	sub_2BBB5A	0.800	1	1	Same constants
00200	000530c4	sub_530C4	00052f08	sub_52F08	0.800	43	44	Same rare constant
00247	000ce658	sub_CE658	000cc624	sub_CC624	0.800	1	1	Same constants
00213	00061c60	sub_61C60	00061088	sub_61088	0.790	22	24	Same rare constant
00008	000170b8	sub_170B8	00016fac	sub_16FAC	0.780	160	159	Same rare constant
00216	00062e6c	sub_62E6C	00062288	sub_62288	0.770	39	38	Same rare constant
00240	0006b054	sub_6B054	0006a3c8	sub_6A3C8	0.770	11	10	Mnemonics small-primes-product
00044	00041d6c	sub_41D6C	0004173c	sub_4173C	0.760	12	11	Same rare constant
00196	000da600	sub_DA600	000d833c	sub_D833C	0.750	14	13	Pseudo-code fuzzy hash
00228	000b64d0	sub_B64D0	000b467c	sub_B467C	0.750	1142	1140	Pseudo-code fuzzy hash
00234	00071158	sub_71158	0007043c	sub_7043C	0.750	8	7	Mnemonics small-primes-product
00199	00052f00	vuln_sub_52F00	00052d0c	sub_52D0C	0.740	23	27	Same rare constant
00043	000400b4	sub_400B4	0003faa4	sub_3FAA4	0.730	131	124	Same rare constant
00158	00036cc0	sub_36CC0	00036744	sub_36744	0.730	1	1	Mnemonics and names
00015	00218e04	sub_218E04	00216ab4	sub_216AB4	0.710	6	7	Same rare KOKA hash
00007	00016504	sub_16504	000164a4	sub_164A4	0.700	24	20	Same rare constant
00108	00269174	sub_269174	001fdcec	sub_1FDCEC	0.670	1	1	Same constants
00137	0004f158	sub_4F158	0004ed20	sub_4ED20	0.670	1	1	Mnemonics and names

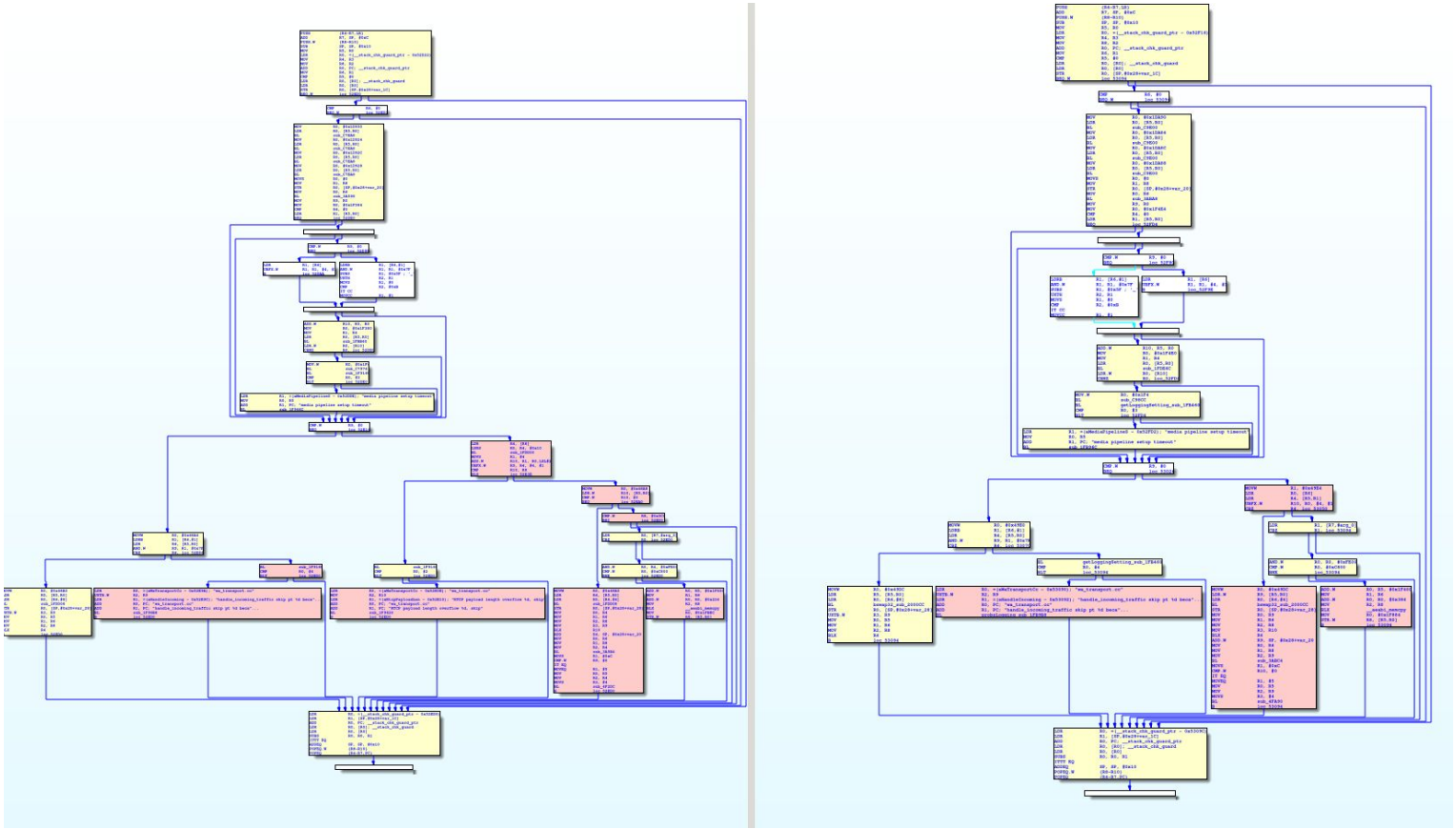
Diaphora

IDA Vie... | Pseudoco... | Unmatched in secon... | Unmatched in pri... | Unreliable mat... | Partial mat... | Best mat... | Struct... | Strin

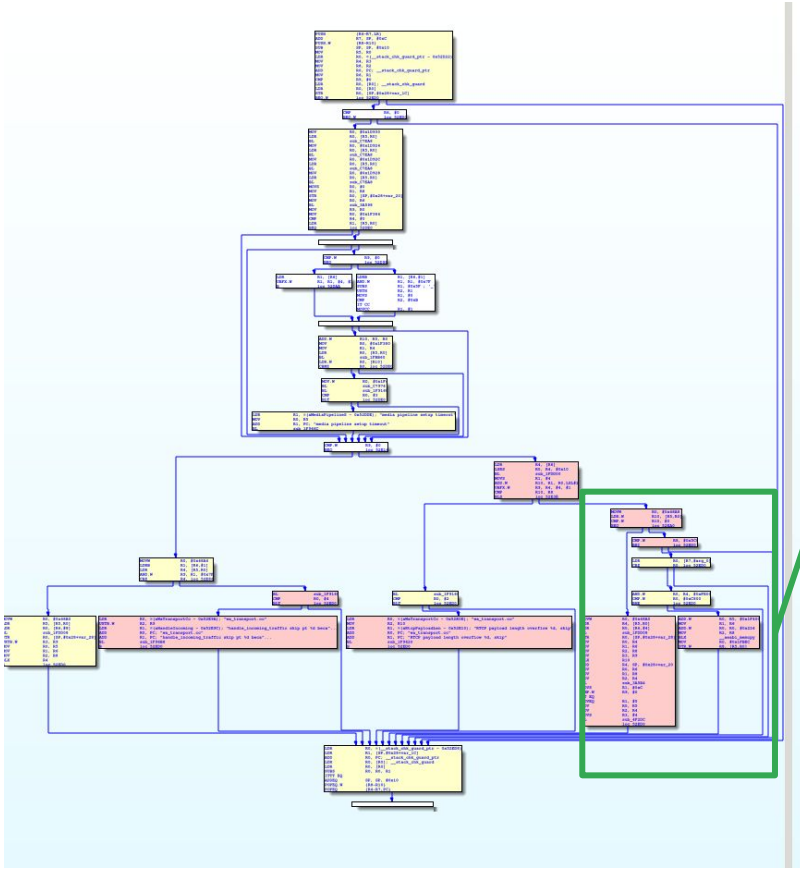
Line	Address	Name	Address 2	Name 2	Ratio	BBlocks 1	BBlocks 2	Description
00170	001e8790	sub_1E8790	001e6490	sub_1E6490	0.860	1	1	Same constants
00171	001f87cc	sub_1F87CC	001f64cc	sub_1F64CC	0.860	1	1	Same constants
00176	000c21f6	sub_C21F6	000c02d4	sub_C02D4	0.860	1	1	Mnemonics and names
00241	0007aaac	sub_7AAAC	00079d54	sub_79D54	0.860	12	11	Mnemonics small-primes-product
00243	000c8edc	sub_C8EDC	000c6f88	sub_C6F88	0.860	12	11	Mnemonics small-primes-product
00246	000dad18	sub_DAD18	000d8a54	sub_D8A54	0.860	13	12	Mnemonics small-primes-product
00250	00166bb0	sub_166BB0	001648b0	sub_1648B0	0.860	1	1	Same constants
00279	00175ea6	sub_175EA6	00173ba6	sub_173BA6	0.860	1	1	Same constants
00033	0003d0b4	sub_3D0B4	0003caa4	sub_3CAA4	0.850	7	6	Same rare constant
00034	0003d120	sub_3D120	0003cb10	sub_3CB10	0.850	7	6	Same rare constant
00211	0005e580	sub_5E580	0005d99c	sub_5D99C	0.850	85	84	Same rare constant
00242	000c8e28	sub_C8E28	000c6ed4	sub_C6ED4	0.850	12	11	Mnemonics small-primes-product
00028	000d52e4	sub_D52E4	001fa2e8	sub_1FA2E8	0.843	4	4	Same MD Index and constants
00141	001fb338	sub_1FB338	000499d6	sub_499D6	0.830	1	1	Mnemonics and names
00155	001faabc	sub_1FAABC	001f87bc	sub_1F87BC	0.820	1	1	Mnemonics and names
00215	00061fc0	sub_61FC0	000613f0	sub_613F0	0.820	16	15	Same rare constant
00002	00014e34	Java_com_whatsapp_util_Whats...	00014dd4	Java_com_whatsapp_util_Whats...	0.800	1	1	Perfect match, same name
00047	0004316c	sub_4316C	00042b34	sub_42B34	0.800	8	7	Same rare constant
00149	0007926c	sub_7926C	00078514	sub_78514	0.800	1	1	Mnemonics and names
00151	0004de04	sub_4DE04	0004d9d8	sub_4D9D8	0.800	1	1	Mnemonics and names
00182	002bdb94	sub_2BDB94	002bb9f4	sub_2BB9F4	0.800	1	1	Same constants
00183	002bdb9e	sub_2BDB9E	002bb9fe	sub_2BB9FE	0.800	1	1	Same constants
00184	002bdbf8	sub_2BDBF8	002bba58	sub_2BBA58	0.800	1	1	Same constants
00185	002bdc02	sub_2BDC02	002bba62	sub_2BBA62	0.800	1	1	Same constants
00186	002bdc74	sub_2BDC74	002bbad4	sub_2BBAD4	0.800	1	1	Same constants
00187	002bdc7e	sub_2BDC7E	002bbade	sub_2BBADE	0.800	1	1	Same constants
00188	002bdcf0	sub_2BDCF0	002bbb50	sub_2BBB50	0.800	1	1	Same constants
00189	002bdcfa	sub_2BDCFA	002bbb5a	sub_2BBB5A	0.800	1	1	Same constants
00200	000530c4	sub_530C4	00052f08	sub_52F08	0.800	43	44	Same rare constant
00247	000ce658	sub_CE658	000cc624	sub_CC624	0.800	1	1	Same constants
00213	00061c60	sub_61C60	00061088	sub_61088	0.790	22	24	Same rare constant
00008	000170b8	sub_170B8	00016fac	sub_16FAC	0.780	160	159	Same rare constant
00216	00062e6c	sub_62E6C	00062288	sub_62288	0.770	39	38	Same rare constant
00240	0006b054	sub_6B054	0006a3c8	sub_6A3C8	0.770	11	10	Mnemonics small-primes-product
00044	00041d6c	sub_41D6C	0004173c	sub_4173C	0.760	12	11	Same rare constant
00196	000da600	sub_DA600	000d833c	sub_D833C	0.750	14	13	Pseudo-code fuzzy hash
00228	000b64d0	sub_B64D0	000b467c	sub_B467C	0.750	1142	1140	Pseudo-code fuzzy hash
00234	00071158	sub_71158	0007043c	sub_7043C	0.750	8	7	Mnemonics small-primes-product
00199	00052f00	vuln_sub_52F00	00052d0c	sub_52D0C	0.740	23	27	Same rare constant
00043	000400b4	sub_400B4	0003faa4	sub_3FAA4	0.730	131	124	Same rare constant
00158	00036cc0	sub_36CC0	00036744	sub_36744	0.730	1	1	Mnemonics and names
00015	00218e04	sub_218E04	00216ab4	sub_216AB4	0.710	6	7	Same rare KOKA hash
00007	00016504	sub_16504	000164a4	sub_164A4	0.700	24	20	Same rare constant
00108	00269174	sub_269174	001fdcec	sub_1FDCEC	0.670	1	1	Same constants
00137	0004f158	sub_4F158	0004ed20	sub_4ED20	0.670	1	1	Mnemonics and names

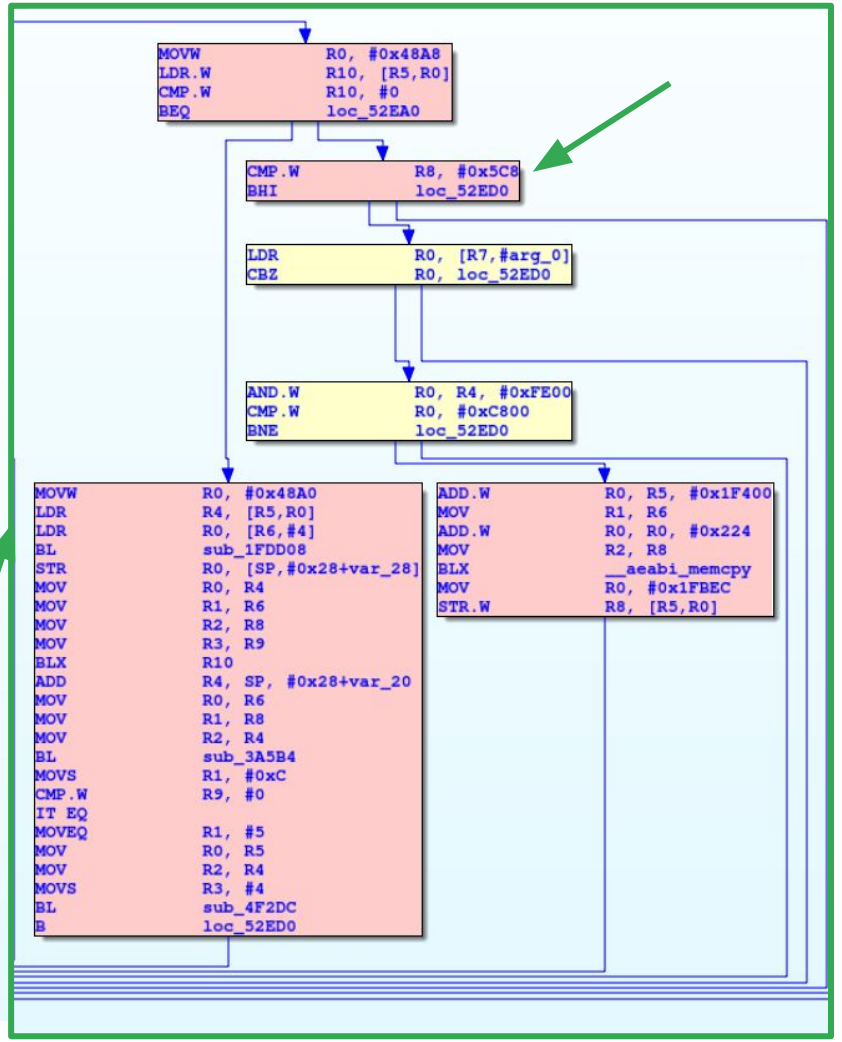
Google

Diaphora: Size Check #2

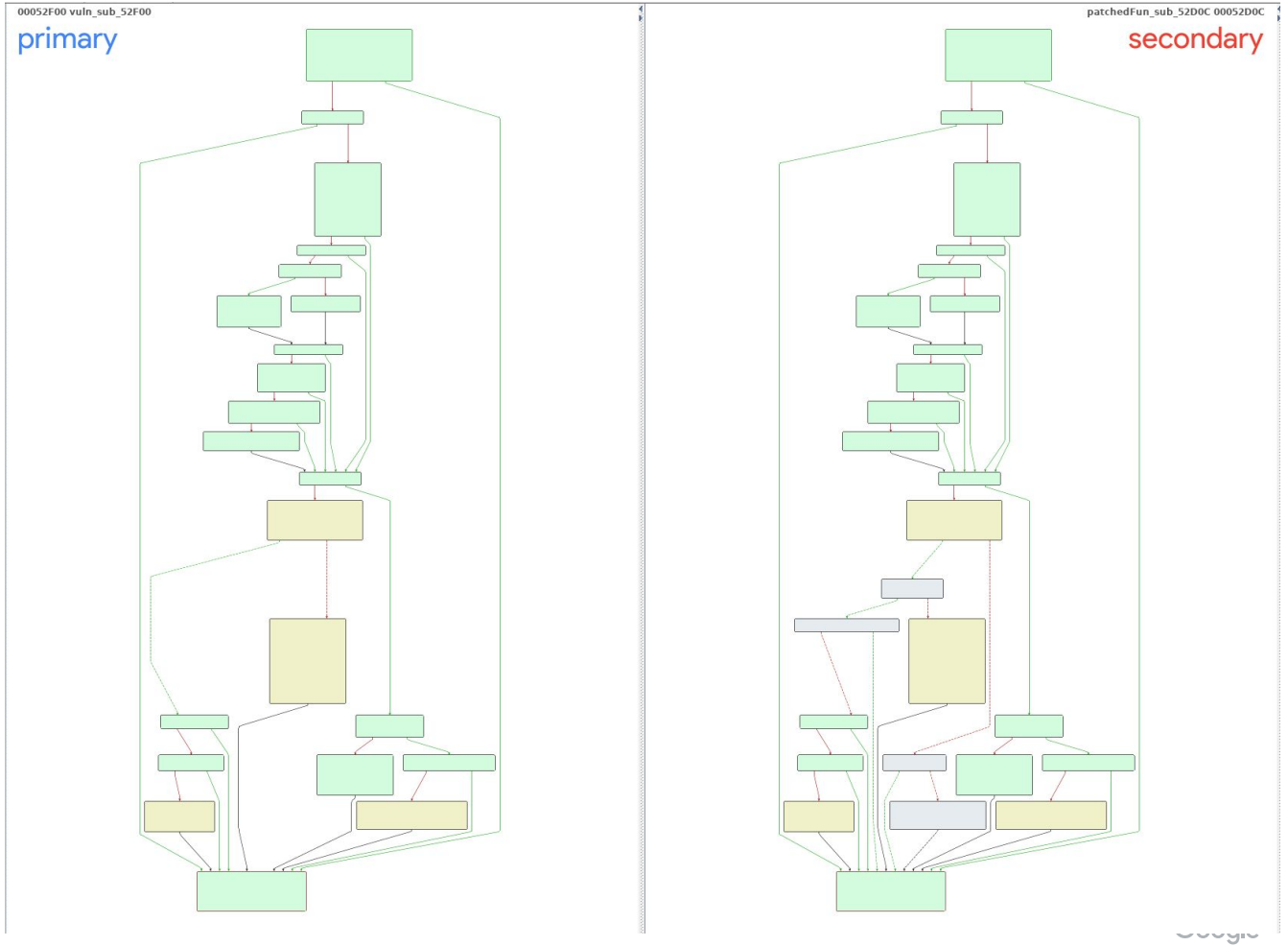


Diaphora: Size Check #2

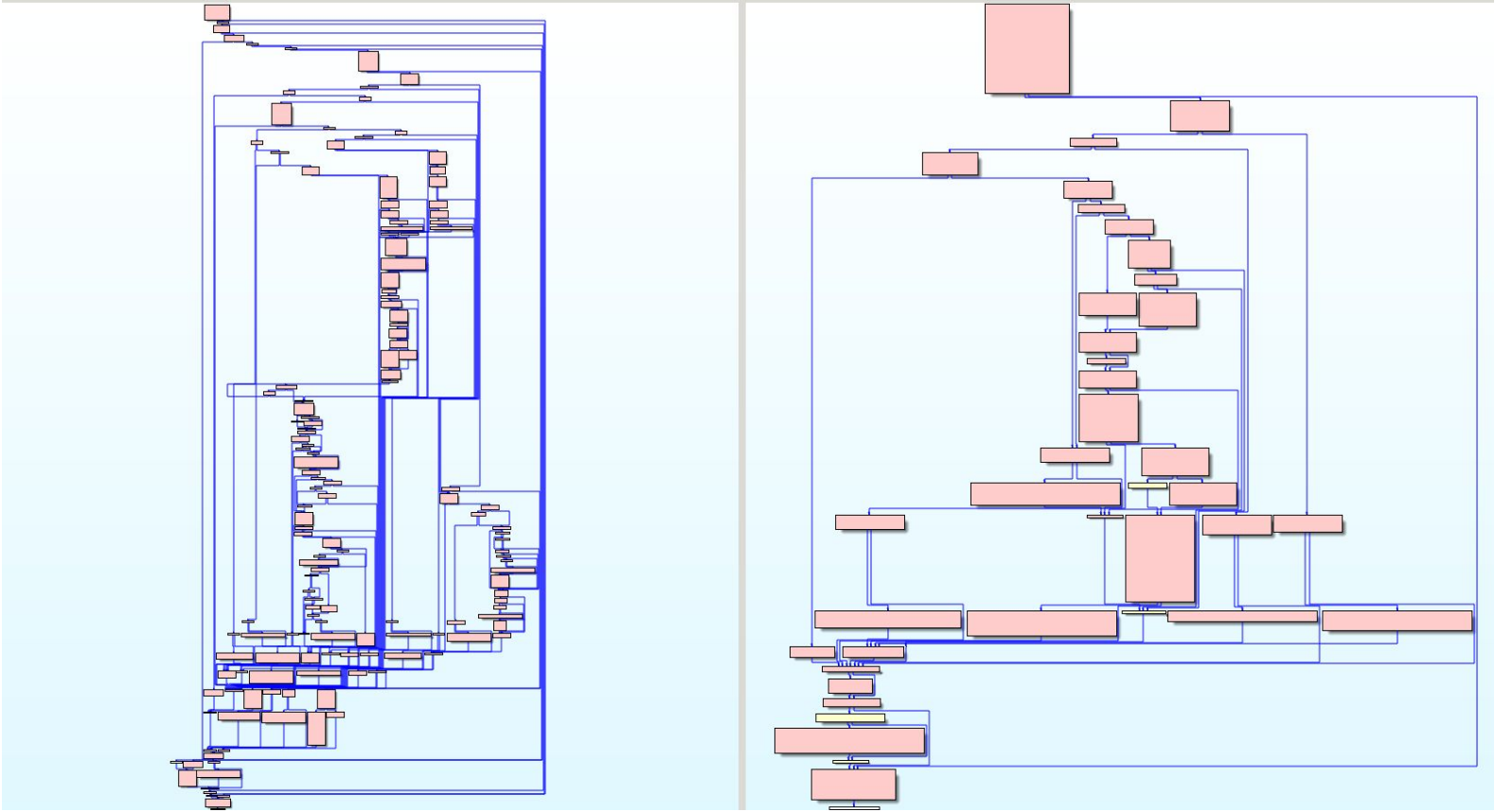




BinDiff: Size Check #2



Diaphora: Size Check #1 - Matches Wrong Functions



Diaphora Size Check #1 - Matches Wrong Function

Unreliable matches | Partial matches | Best matches | Imports | Exports

Line	Address	Name	Address 2	Name 2	Ratio	BBlocks 1	BBlocks 2	Description
00009	001fb478	sub_1FB478	0008d764	sub_8D764	0.500	1	4	Same constants
00008	000370b8	sub_370B8	00038934	sub_38934	0.470	14	4	Same constants
00006	0004c798	sub_4C798	002013c0	sub_2013C0	0.440	1	2	Same constants
00004	000cdb64	sub_CDB64	0028b334	sub_28B334	0.340	4	4	Same constants
00002	0004f1f8	sub_4F1F8	0007b738	sub_7B738	0.260	8	8	Same rare KOKA hash
00010	00057b94	sub_57B94	0004d9d8	sub_4D9D8	0.260	4	1	Same constants
00016	001fc5e8	sub_1FC5E8	001fbc8e	sub_1FBC8E	0.200	4	3	Same constants
00003	0007f260	sub_7F260	000391c0	sub_391C0	0.190	2	5	Same constants
00013	000506d0	sub_506D0	00050db8	sub_50DB8	0.190	93	16	Same rare constant
00014	000515e4	sub_515E4	00051e34	sub_51E34	0.190	40	151	Same rare constant
00007	0018eeac	sub_18EEAC	0006e4d0	sub_6E4D0	0.170	1	3	Same constants
00005	0008749c	sub_8749C	000144f4	sub_144F4	0.130	6	4	Same constants
00011	0004f32c	sub_4F32C	0004f0b0	sub_4F0B0	0.130	27	6	Same rare constant
00015	000c9e78	sub_C9E78	000d1100	sub_D1100	0.130	9	27	Same constants
00000	00051d30	callsVuln_sub_51D30	00051d30	sub_51D30	0.100	165	19	Same address and rare constant
00001	0007a3ac	sub_7A3AC	0007a3ac	sub_7A3AC	0.090	71	12	Same address and rare constant
00017	0006ea44	sub_6EA44	0006ee80	sub_6EE80	0.080	16	85	Same rare constant
00018	0007b104	sub_7B104	00079654	sub_79654	0.080	12	71	Same rare constant
00012	0004f864	sub_4F864	00050478	sub_50478	0.060	6	72	Same rare constant
00019	00087504	sub_87504	0025a684	sub_25A684	0.040	100	5	Same rare constant

Diaphora

- Matching wasn’t great
- Tends to get thrown off by naming, different offsets, etc.
- Has support for decompilation diffing, but rather basic
- Open source and currently developed!
- Integrated fully into IDA with support coming for other tools

Radare2

- https://github.com/radareorg/radare2
- https://radare.gitbooks.io/radare2book/content/
- Open source and currently developed (last commit was 2 hours ago!)
- Well documented

Radare is a portable reversing framework that can...

- Disassemble (and assemble for) many different architectures
- Debug with local native and remote debuggers (gdb, rap, webui, r2pipe, winedbg, windbg)
- Run on Linux, *BSD, Windows, OSX, Android, iOS, Solaris and Haiku
- Perform forensics on filesystems and data carving
- Be scripted in Python, Javascript, Go and more
- Support collaborative analysis using the embedded webserver
- Visualize data structures of several file types
- Patch programs to uncover new features or fix vulnerabilities
- Use powerful analysis capabilities to speed up reversing
- Aid in software exploitation

radare2 (radiff2)

- https://github.com/radareorg/radare2
- https://radare.gitbooks.io/radare2book/content/
- Open source and currently developed (last commit was 2 hours ago!)
- Well documented

And....

supports binary diffing via radiff2.

Radare is a portable reversing framework that can...

- Disassemble (and assemble for) many different architectures
- Debug with local native and remote debuggers (gdb, rap, webui, r2pipe, winedbg, windbg)
- Run on Linux, *BSD, Windows, OSX, Android, iOS, Solaris and Haiku
- Perform forensics on filesystems and data carving
- Be scripted in Python, Javascript, Go and more
- Support collaborative analysis using the embedded webserver
- Visualize data structures of several file types
- Patch programs to uncover new features or fix vulnerabilities
- Use powerful analysis capabilities to speed up reversing
- Aid in software exploitation

radiff2

`radiff2 patched_libwhatsapp.so vuln_libwhatsapp.so`

→ Results in 150,533 diffs

```
0x00052c18 41461d46e7f7bcfc064604f5f83000f569702946a7f18cfc2046fcf73dfa002e53d14ff21c61 =>
c0f20105002e00f09f80d6f88c004ff48051c0f778eed6f88c204ff40063d8f80010b9f1000f 0x00052c18
0x00052c3f f21862c0f20101c0f201026158 => f0010018bf0323cde900302046 0x00052c3f
0x00052c4d f27430a358d7f808a0c0f201000126a518265053ea010009d0b9f80000c0f30629a9f15f00c0b20b2815d31fe04ff214604ff21062b9f80010 =>
f48053fcf76cfbb0f1ff3f03901fddd7f808b044f6785103ab324604eb8b0041580020cde9000a2046fcf7bffa0546002d6fd13f484ff6a001 0x00052c4d
0x00052c89 00c0f201022058a258c1f30629a9f15f06104328d0f0b20a280ad804f5f83000f55e76304635f08ffd10b9 =>
01039d7844625a036819888a4206d11c210ce000200535c6f888005ce03749794409680b8800219a4208bf 0x00052c89
0x00052cb5 46aaf19fff04f5f830baf1000f08bf00f5c2654246d5e90001013041 => 214ff2e8624ff2ec66c0f20102c0f20106a3580d44a0595919a15040 0x00052cb5
0x00052cd3 01c5e900015ffa89f12046fef77ffd2046bde80007bde8f040fcf720ba4ff60c40c0f201002058d0f874120029cd =>
00a0512046294605f083fa4ff228704ff22c72c0f20100c0f201022158b9f1000fa3582944215003f10100a0502d 0x00052cd3
0x00052d02 f5f765fbf0b2b4e70000f0b503af2de9000784b0054674481c => 4ff4b0700bfb008505f11000adf155f905f11800acf1e5ff2e 0x00052d02
0x00052d1c
904678440e46002d00680068039000f0d180002e00f0ce804df63010c0f20100285875f0b3f84df62410c0f20100285875f0acf84df62c10c0f20100285875f0a5f84df62810c0f201002
85875f09ef8002041 =>
56f8200f06ebc0000830acf1ddff306844f24851615844f26c0200eb400011440c2206eb80008830c0f7a8ed316855f8240f013101f00f0131600f289cbf013028600025094804997844 0
0680068401a01bf28 0x00052d1c
0x00052d70 02903046e7f710fc81464ff28430c0f20100002c29 => 05b0bde8000ff0bdc0f758edb65738001859380 0f8 0x00052d70
0x00052d86 2bd051bbb9f1000f03d03168c1f3001108e0717801f07f => 380054563800b0b502af0d4644f2485114464258d2f8c8 0x00052d86
```

radiff2

`radiff2 patched_libwhatsapp.so vuln_libwhatsapp.so`

→ Results in 150,533 diffs

```
0x00052c18 41461d46e7f7bcfc064604f5f83000f569702946a7f18cfc2046fcf73dfa002e53d14ff21c61 =>
c0f20105002e00f09f80d6f88c004ff48051c0f778eed6f88c204ff40063d8f80010b9f1000f 0x00052c18
0x00052c3f f21862c0f20101c0f201026158 => f0010018bf0323cde900302046 0x00052c3f
0x00052c4d f27430a358d7f808a0c0f201000126a518265053ea010009d0b9f80000c0f30629a9f15f00c0b20b2815d31fe04ff214604ff21062b9f80010 =>
f48053fcf76cfbb0f1ff3f03901fddd7f808b044f6785103ab324604eb8b0041580020cde9000a2046fcf7bffa0546002d6fd13f484ff6a001 0x00052c4d
0x00052c89 00c0f201022058a258c1f30629a9f15f06104328d0f0b20a280ad804f5f83000f55e76304635f08ffd10b9 =>
01039d7844625a036819888a4206d11c210ce000200535c6f888005ce03749794409680b8800219a4208bf 0x00052c89
0x00052cb5 46aaf19fff04f5f830baf1000f08bf00f5c2654246d5e90001013041 => 214ff2e8624ff2ec66c0f20102c0f20106a3580d44a0595919a15040 0x00052cb5
0x00052cd3 01c5e900015ffa89f12046fef77ffd2046bde80007bde8f040fcf720ba4ff60c40c0f201002058d0f874120029cd =>
00a0512046294605f083fa4ff228704ff22c72c0f20100c0f201022158b9f1000fa3582944215003f10100a0502d 0x00052cd3
0x00052d02 f5f765fbf0b2b4e70000f0b503af2de9000784b0054674481c => 4ff4b0700bfb008505f11000adf155f905f11800acf1e5ff2e 0x00052d02
0x00052d1c
904678440e46002d00680068039000f0d180002e00f0ce804df63010c0f20100285875f0b3f84df62410c0f20100285875f0acf84df62c10c0f20100285875f0a5f84df62810c0f201002
85875f09ef8002041 =>
56f8200f06ebc0000830acf1ddff306844f24851615844f26c0200eb400011440c2206eb80008830c0f7a8ed316855f8240f013101f00f0131600f289cbf013028600025094804997844 0
0680068401a01bf28 0x00052d1c
0x00052d70 02903046e7f710fc81464ff28430c0f20100002c29 => 05b0bde8000ff0bdc0f758edb65738001859380 0f8 0x00052d70
0x00052d86 2bd051bbb9f1000f03d03168c1f3001108e0717801f07f => 380054563800b0b502af0d4644f2485114464258d2f8c8 0x00052d86
```

radiff2

```
radiff2 -AC -a arm Binaries/vuln_libwhatsapp.so Binaries/patched_libwhatsapp.so
```

⟶ Took 9.5 hours to run

```
fcn.002dfa50   102 0x2dfa50 | UNMATCH  (0.095588) | 0x2dd8b0    102 fcn.002dd8b0
fcn.002df528    54 0x2df528 |   MATCH  (0.944444) | 0x2dd388     54 fcn.002dd388
fcn.002df35c   450 0x2df35c | UNMATCH  (0.100000) | 0x2dd1bc    450 fcn.002dd1bc
fcn.002df2ac   166 0x2df2ac | UNMATCH  (0.094828) | 0x2dd10c    166 fcn.002dd10c
fcn.002cfe80 16912 0x2cfe80 |     NEW  (0.000000)
fcn.002def20   836 0x2def20 | UNMATCH  (0.089713) | 0x2dcd80    836 fcn.002dcd80
fcn.002df712     2 0x2df712 |     NEW  (0.000000)
fcn.002df788     2 0x2df788 |   MATCH  (1.000000) | 0x2dd5e8      2 fcn.002dd5e8
fcn.002df6da     2 0x2df6da |   MATCH  (1.000000) | 0x2dd572      2 fcn.002dd572
fcn.002dec70   636 0x2dec70 | UNMATCH  (0.110054) | 0x2dcad0    636 fcn.002dcad0
fcn.002de678   186 0x2de678 | UNMATCH  (0.060345) | 0x2dc4d8    186 fcn.002dc4d8
```

radiff2

```
radiff2 -AC -a arm Binaries/vuln_libwhatsapp.so Binaries/patched_libwhatsapp.so
```

→ Took 9.5 hours to run

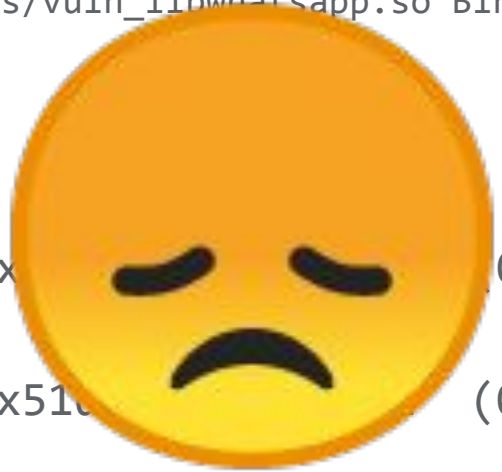
```
fcn.00052f00   430 0x52f00 |     NEW  (0.000000)

fcn.00051d30  3420 0x51d30 |     NEW  (0.000000)
```

radiff2

```
radiff2 -AC -a arm Binaries/vuln_libwhatsapp.so Binaries/patched_libwhatsapp.so
```

→ Took 9.5 hours to run



```
fcn.00052f00   430 0x                    0.000000)

fcn.00051d30  3420 0x51                  (0.000000)
```

Comparison :

	BinDiff	Diaphora	DarunGrim	Radare2
Matches the vuln vs patched funcs	2/2	1/2 Matched “Size Check #1” to wrong function		0/2
Clearly shows important changes in disasm (func to func)	Yes!	Meh?		No :(
Highlights important changes at file level	Out of the box? Nope. Maybe with more customizations?	Out of the box? Nope. Maybe with more customizations?		No :(

Comparison

:

	BinDiff	Diaphora	DarunGrim	Radare2
Matches the vuln vs patched funcs	2/2	1/2 Matched “Size Check #1” to		0/2
disasm (func to func)				
Highlights important changes at file level	Out of the box? Nope. Maybe with more customizations?	Out of the box? Nope. Maybe with more customizations?		No :(

Overall, I found BinDiff to be the most user friendly out of the box. However, doesn’t have the same support currently as Diaphora so mileage may vary if there are bugs, etc.

Static Analysis

Where we’re at

- We have two size checks added to the patched version.

Where we’re at

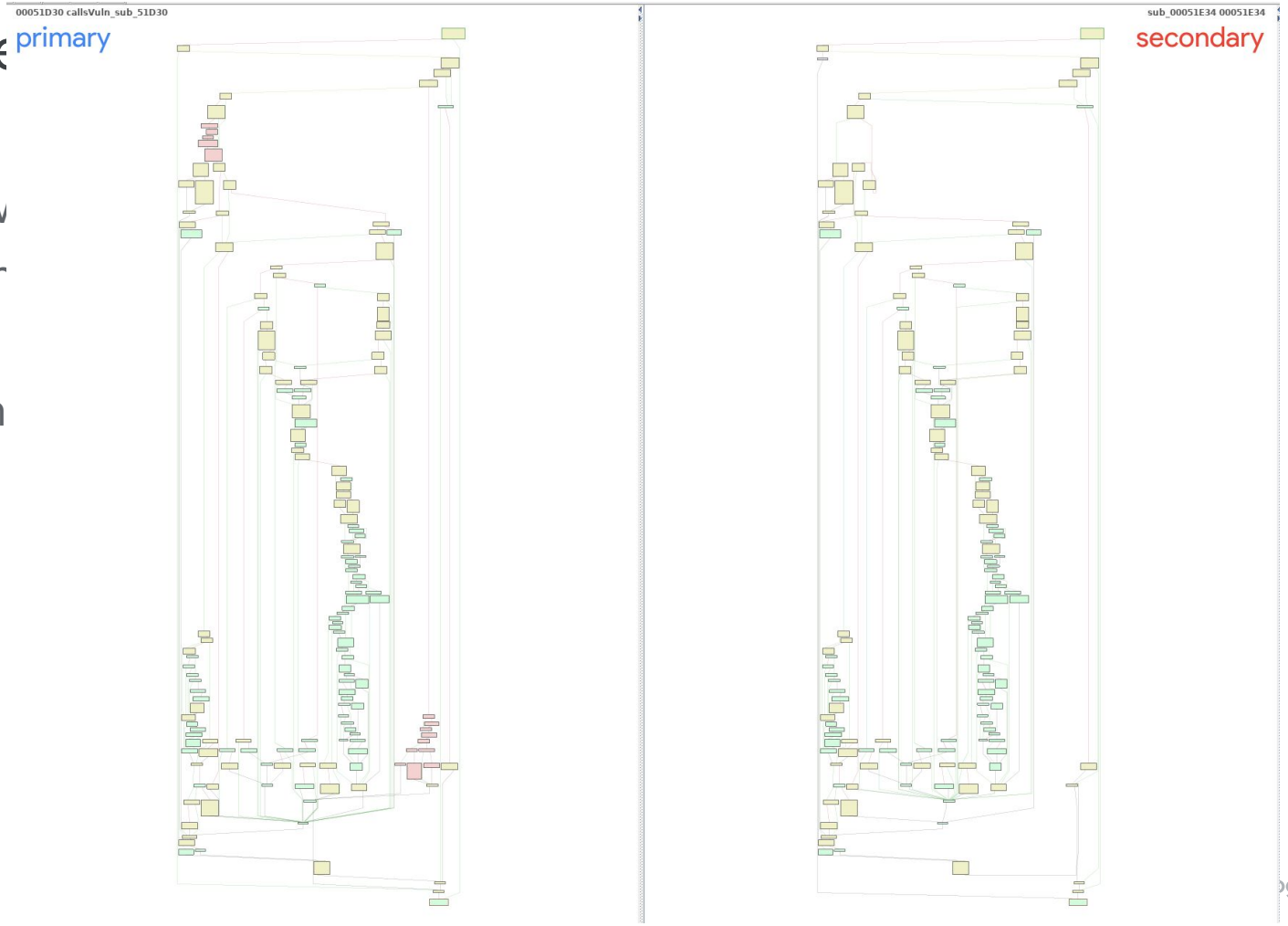
- We have two size checks added to the patched version.
- We know their corresponding functions in the vulnerable version of the library.

Where we’re at

- We have two size checks added to the patched version.
- We know their corresponding functions in the vulnerable version of the library.
- Bindiff highlighted that there are a few more changes in those two functions

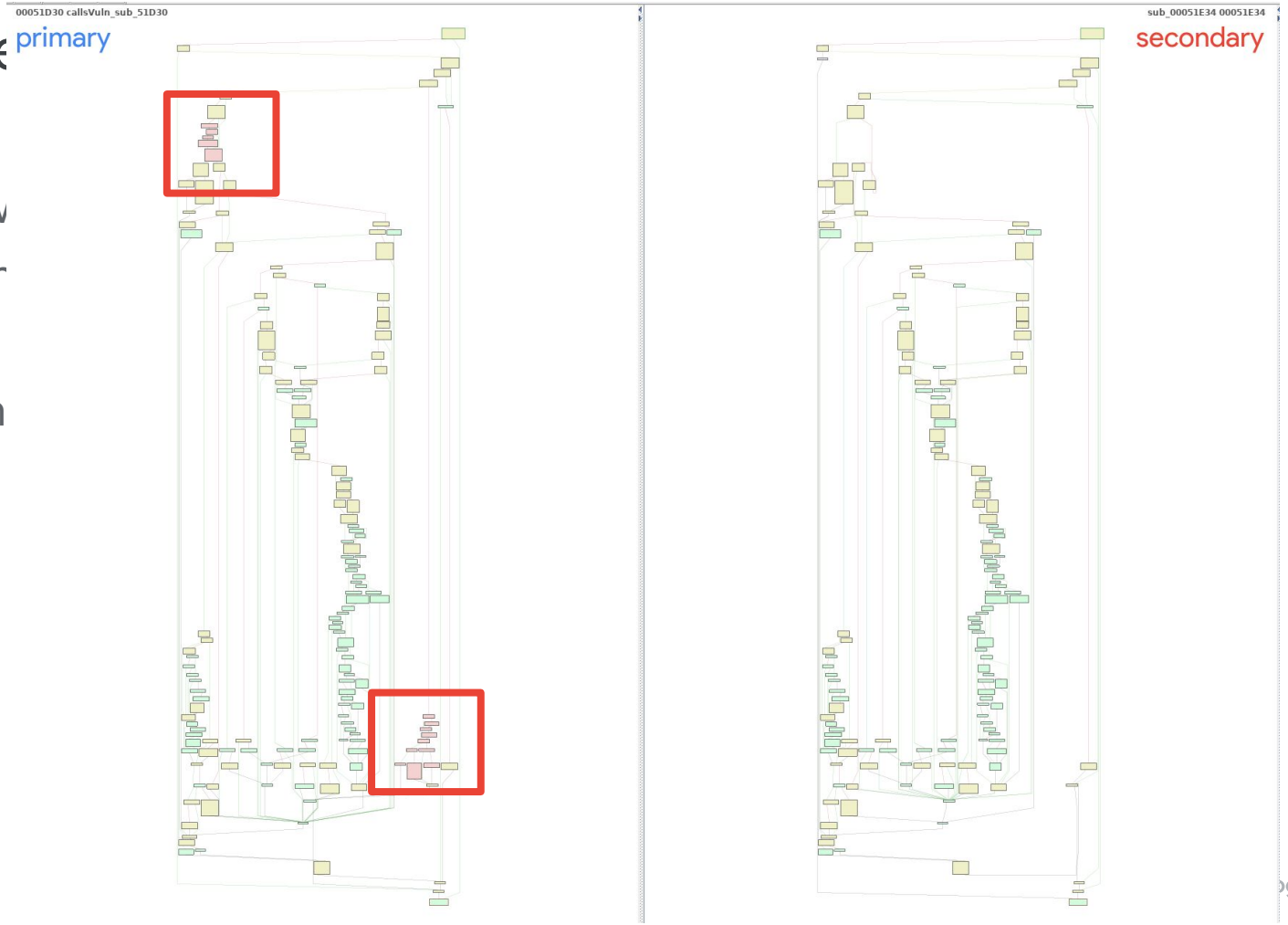
Where we're

- We have tw
- We know th
  library.
- Bindiff high
  functions



Where we're

- We have tw
- We know th
  library.
- Bindiff high
  functions



```
      buffer->dword1F894 = v28;
      ++buffer->numPacketsInBurst;
    }
    else if ( getLoggingSetting_sub_1FB460() >= 1 )
    {
      sub_1FB8D4("wa_transport.cc", " not enough space for buffer burst packet of length %d p
    }
    result = 1;
LABEL_80:
    *(_DWORD *)&v10[1].char0 = 0;
    return result;
  }
  numPacketsInBurst = buffer->numPacketsInBurst;
  if ( numPacketsInBurst )
  {
    if ( getLoggingSetting_sub_1FB460() >= 1 )
      sub_1FB8D4("wa_transport.cc", "processing a simulated burst of %d packets", numPacketsI
    buffer->dword1F894 = 0;
    buffer->numPacketsInBurst = 0;
    if ( numPacketsInBurst >= 1 )
    {
      v18 = 0;
      do
      {
        callsVuln_sub_51D30(
          arg1_containsPtrToBuffer,
          (unsigned __int16 *)buffer->pdword1F898[v18],
          *(_QWORD *)&buffer->pdword1F898[v18 + 1],
          (const void *)(*(_QWORD *)&buffer->pdword1F898[v18 + 1] >> 32),
          buffer->pdword1F898[v18 + 3],
          0);
        --numPacketsInBurst;
        v18 += 4;
      }
      while ( numPacketsInBurst );
```

Now what?

- What can we overwrite?

Now what?

- What can we overwrite?
- How do we exploit it?

Now what?

- What can we overwrite?
- How do we exploit it?
- How do we trigger it?

Now what?

- What can we overwrite?
- How do we exploit it?
- How do we trigger it?

Let’s do some static reversing!

Subroutines of Interest (arm32)

Function with vulnerable memcopy (size check #2):

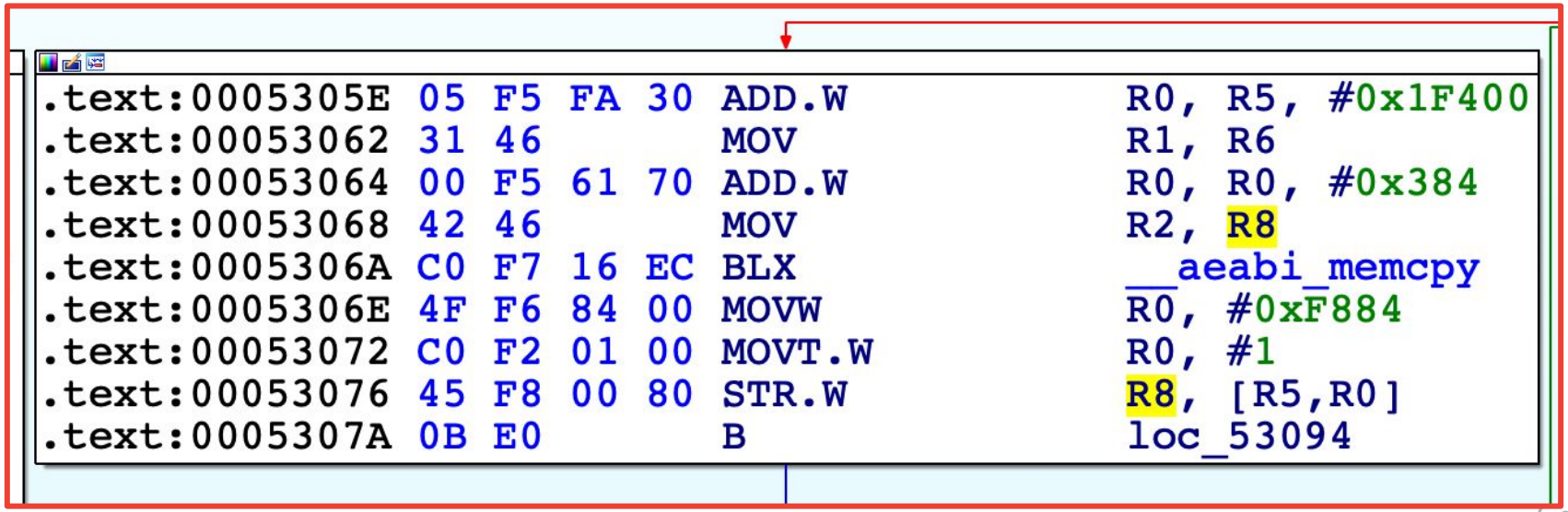
- Vulnerable: 0x52F00
- Patched: 0x52D0C

Function that calls func above (size check #1):

- Vulnerable: 0x51D30
- Patched: 0x51E34

What can we overwrite?

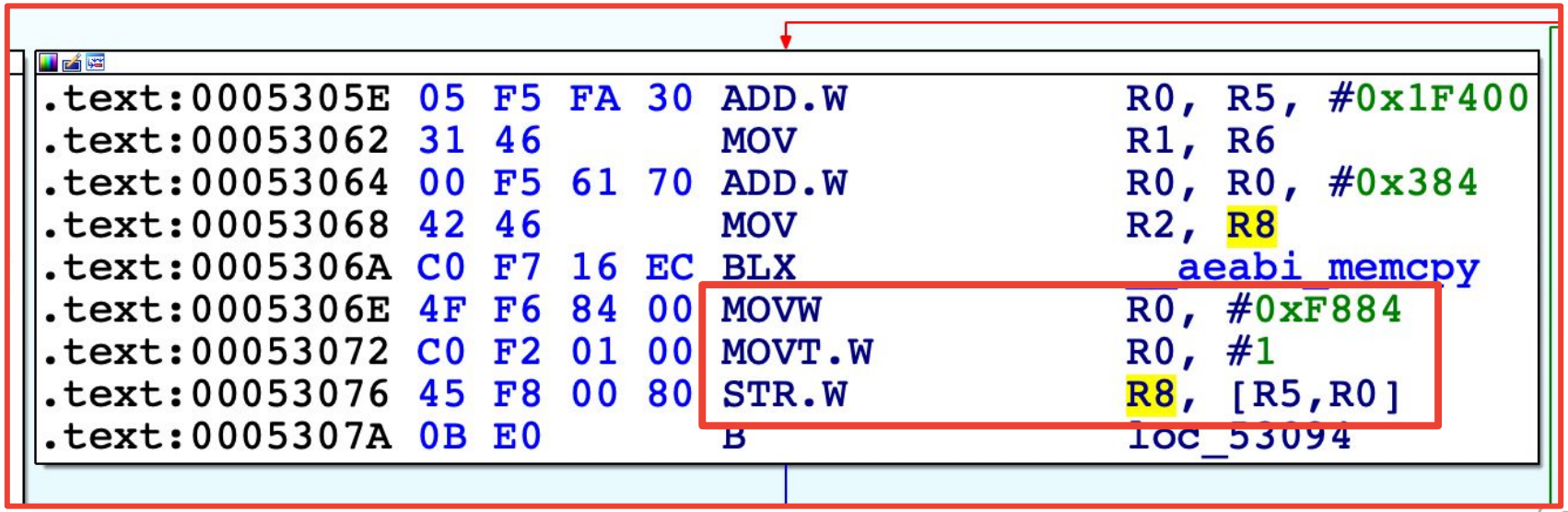
- In the vulnerable version (0x5306A):
  - `memcpy(buffer_arg0 + 0x1F7A4 , packet_arg1, length_arg2)`



What can we overwrite?

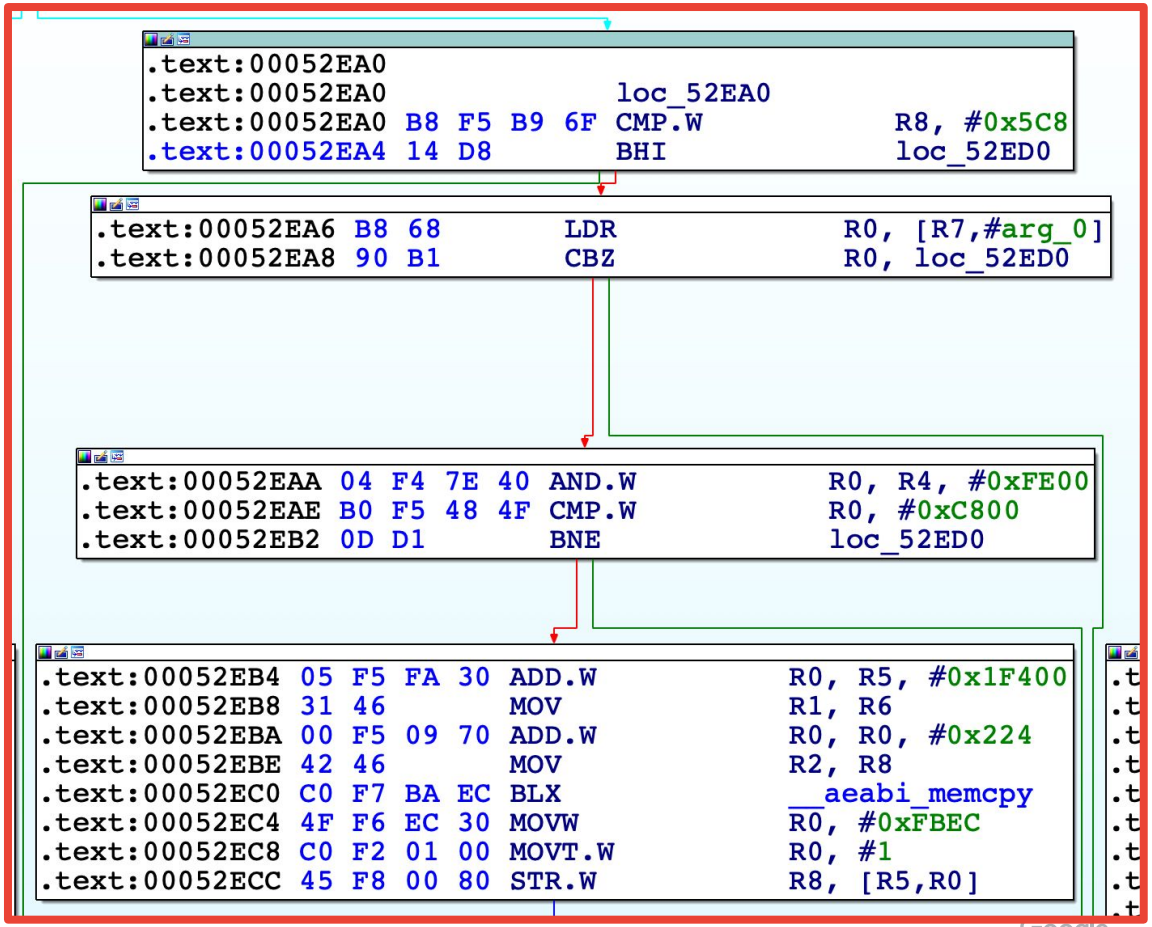
- In the vulnerable version (0x5306A):
  - `memcpy(buffer_arg0 + 0x1F7A4 ,`

Writes copy length to 0x100 bytes from the beginning of the copy.



What can we overwrite?

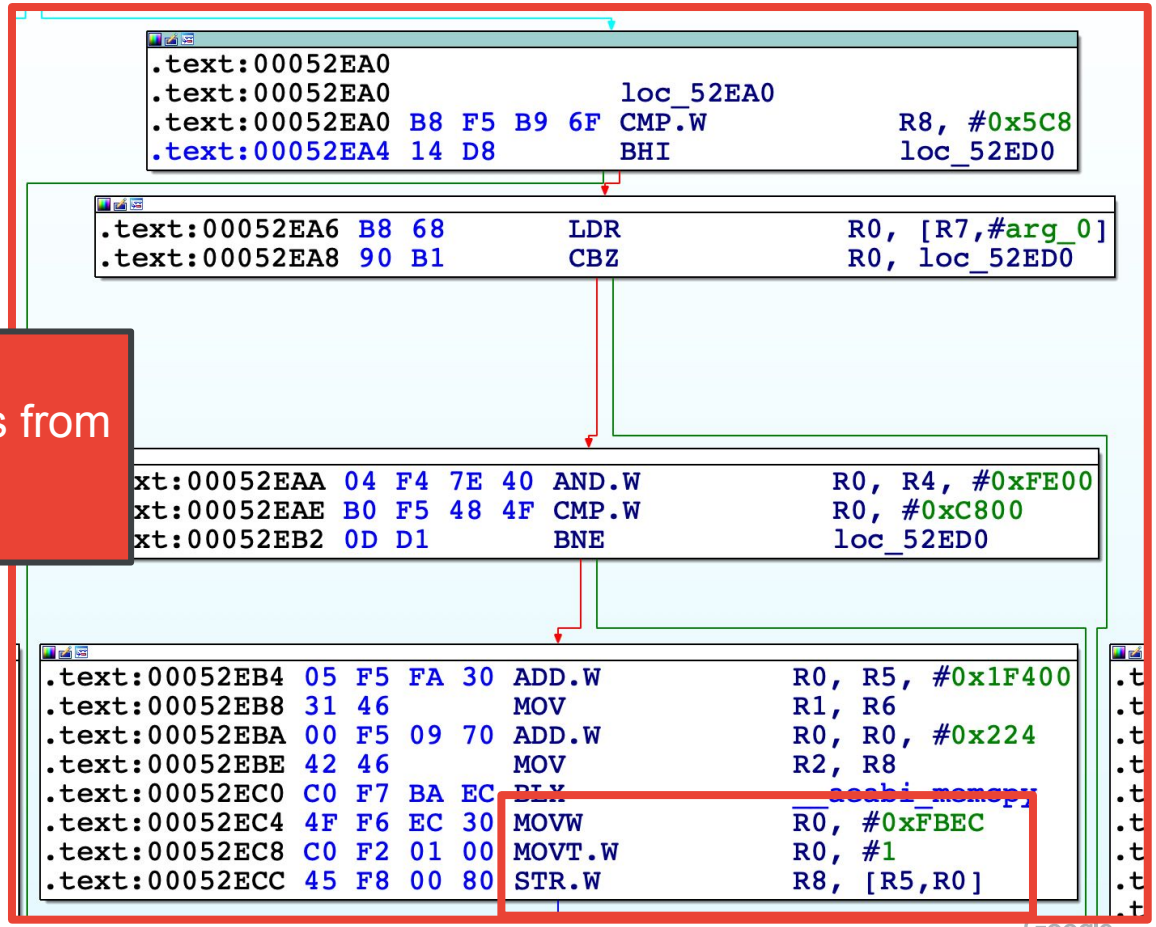
Patched version



What can we overwrite?

Patched version

Writes copy length to 0x5C8 bytes from the beginning of the copy.



What can we overwrite?

- Need to understand the structure where we're copying the data too.
- What's its size?
- Are we just likely to overwrite other members of the struct or do we need to look into what may be allocated after this struct?

Backing Up

- WhatsApp uses PJSIP, an open source product, for its video conferencing implementation
  - Thanks, Natalie! https://googleprojectzero.blogspot.com/2018/12/adventures-in-video-conferencing-part-3.html
- WhatsApp adds some customization on top of PJSIP, but includes lots of the same framework...including logging strings.
- Use this source code to help deduce the structs

How do we exploit it?

- Likely related to the burst packets processing that was removed in the patched version.
- Values for the burst packet processing are after where the packet can be copied
  - That means they can be overwritten

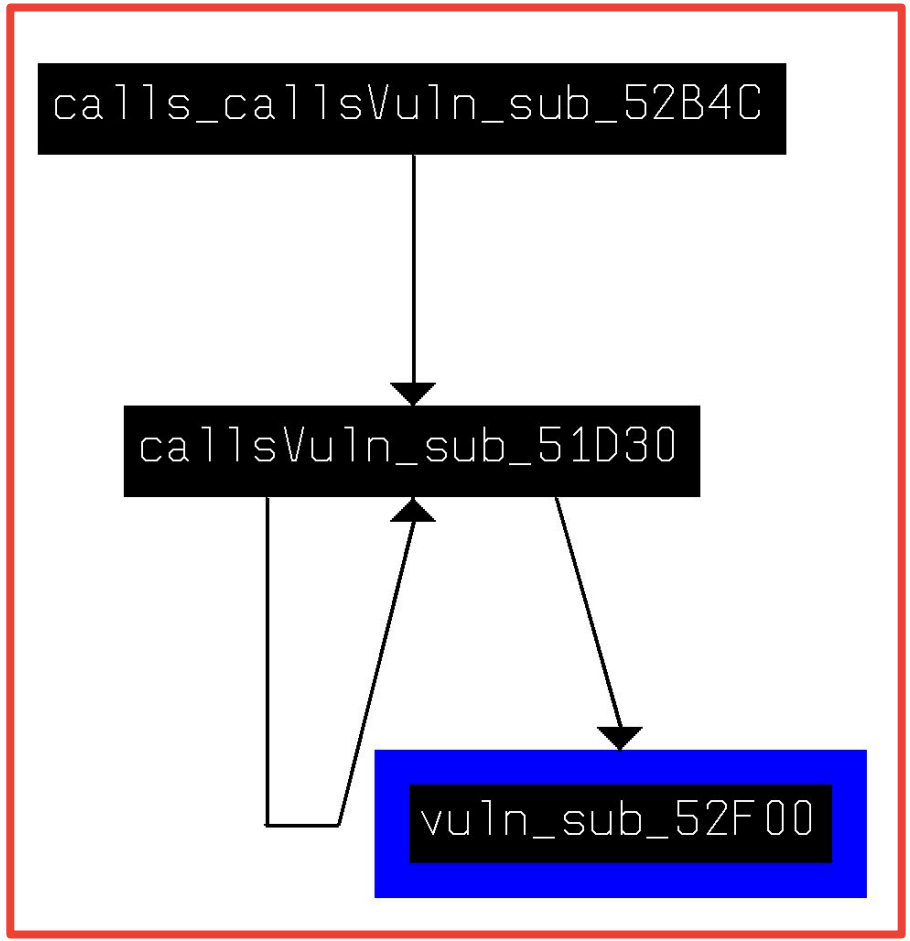
How do we trigger it?

What path calls the vulnerable memcpy?

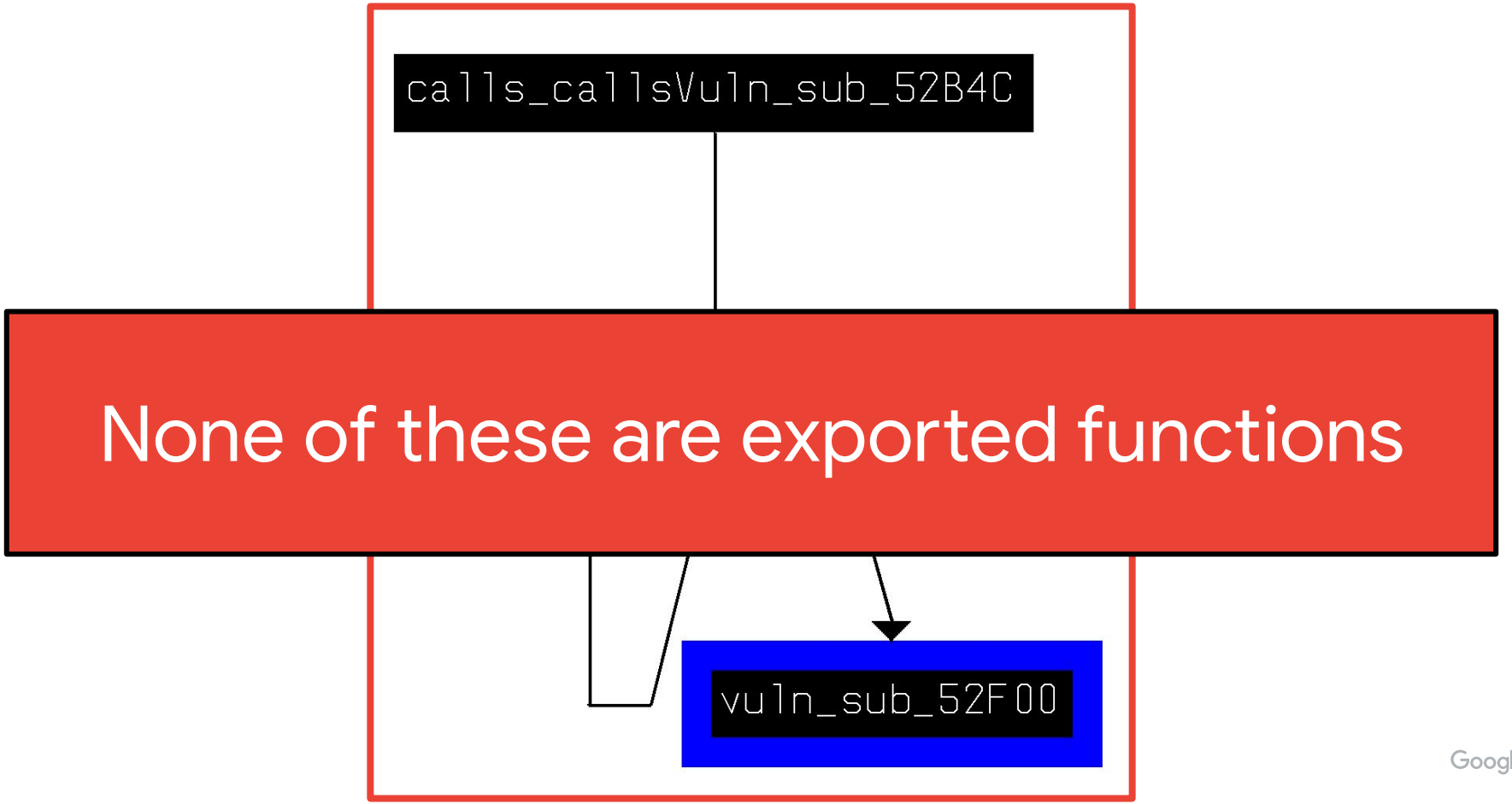
How do we trigger it?

Begin by tracing call references

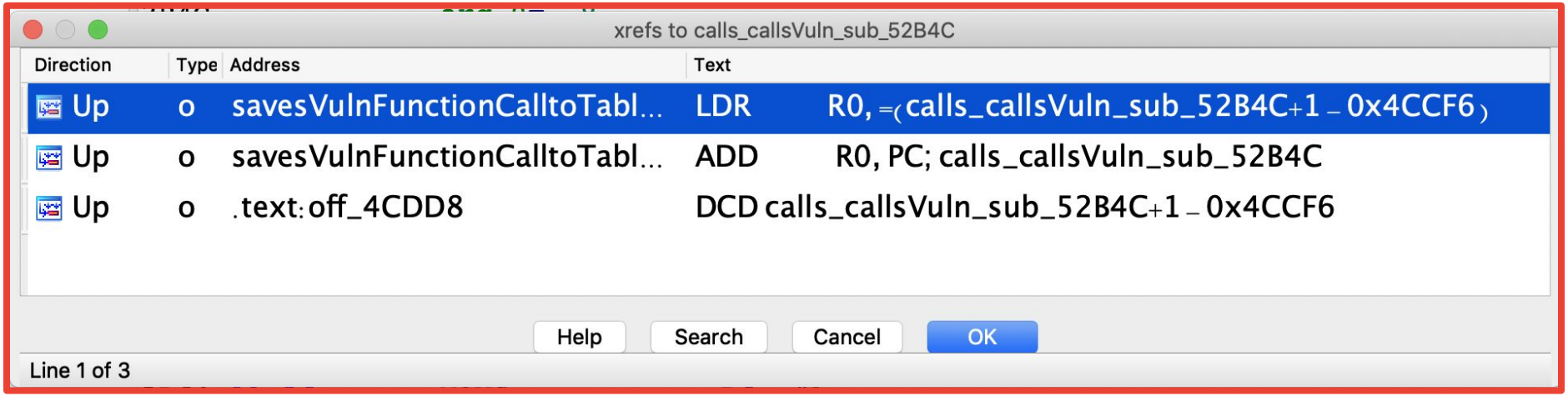
How do we trigger it?



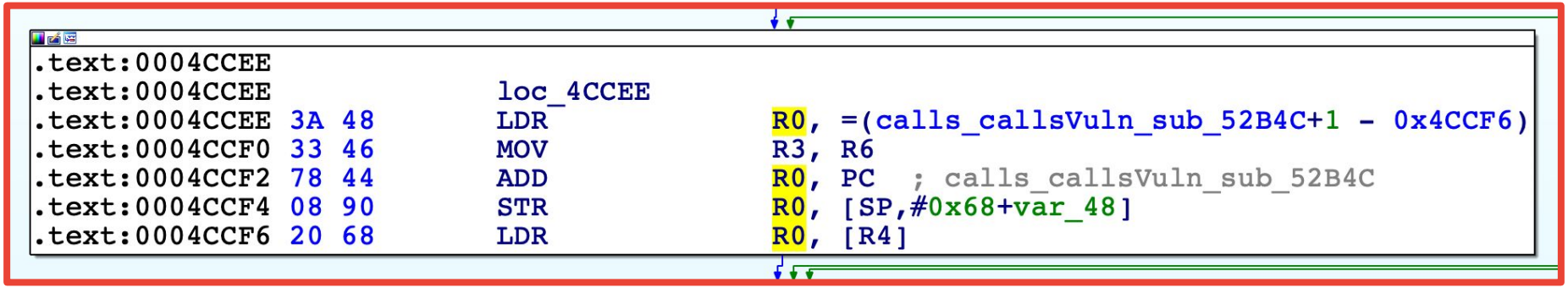
How do we trigger it?



How do we trigger it?



How do we trigger it?



How do we trigger it?

Use frida to show us the execution path

Dynamic Analysis with frida

What is frida?

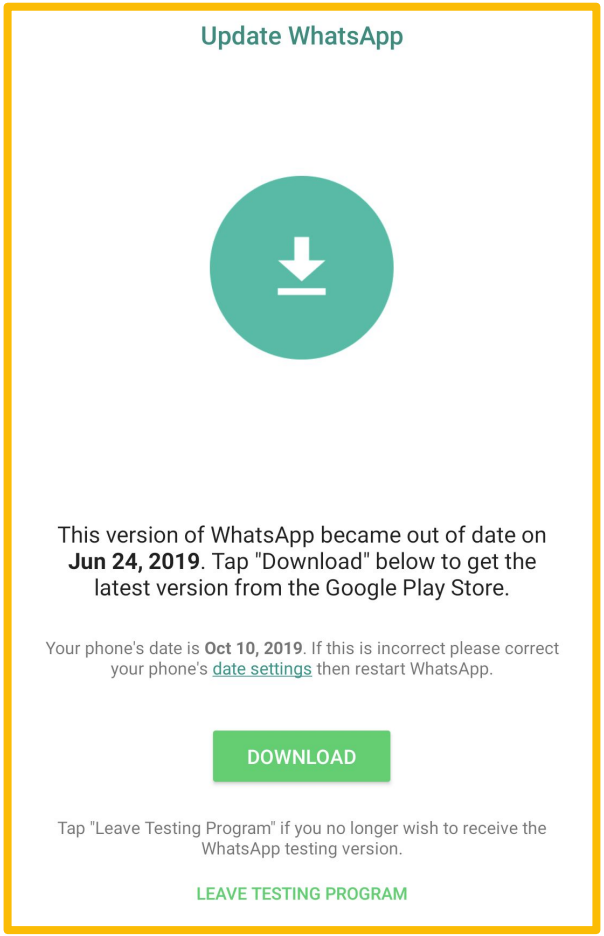
- frida.re is a dynamic instrumentation framework
- Runs on just about all platforms
- Actively developed, open source
- Run the frida-server on a rooted Android device to instrument/hook code running on the phone
    - Options to run on a non-rooted device, but a little more complex
- Write a combo Python & Javascript script to instrument the target, run from laptop

My setup

- Pixel 2 running PQ3A.190801.002 (P)
- Verizon test SIM
- Injecting from MacOS/Linux

Frida didn't originally work on Android 10 due to the linker being moved, but addressed on Tuesday

Running the vulnerable version of Whatsapp



Running the vulnerable version of Whatsapp

- Install the current version of Whatsapp
- Register and get the app fully started up
- Quit the app
- Save off the contents of `/data/data/com.whatsapp/` to your laptop
- Uninstall WhatsApp
- Disconnect the phone from WiFi and cellular
- Set the date of the device to a day when the version of interest was OK
- Using ADB, install the WhatsApp version of interest
- Using ADB, copy the saved files back into `/data/data/com.whatsapp/`
- Start the app, if it starts up correctly, turn on Wifi ensuring that "Automated app updates" and "Automated date and time" are both off
- Do not turn on cellular, this will override the date settings.

Hook the functions of interest

- To use frida to hook the functions of interest, we need a way to tell frida what functions to hook.
- For Android native libraries, can often use `Module.findExportByName`
  - But only if the function of interest is exported (like a JNI function)
- Our functions are not exported so we need to calculated the addresses of where they're loaded into memory
  - Know their offsets from the base from IDA
  - Use `Module.getBaseAddress` and then add the offset to the returned `NativePointer` to get the correct address

Find where the functions of interest are loaded

```
var libBaseAddr = Module.getBaseAddress("libwhatsapp.so");
var JNIOnload_addr = Module.getExportByName("libwhatsapp.so", "JNI_OnLoad");
var callsCallsVulnAddr_8A360 = libBaseAddr.add(0x8A360);
var callsVulnAddr_88DB0 = libBaseAddr.add(0x88DB0);
var vulnAddr_8A8B0 = libBaseAddr.add(0x8A8B0);
```

Find where the functions of interest are loaded

```
var libBaseAddr = Module.getBaseAddress("libwhatsapp.so");
var JNIOnload_addr = Module.getExportByName("libwhatsapp.so", "JNI_OnLoad");
var callsCallsVulnAddr_8A360 = libBaseAddr.add(0x8A360);
var callsVulnAddr_88DB0 = libBaseAddr.add(0x88DB0);
var vulnAddr_8A8B0 = libBaseAddr.add(0x8A8B0);
```

Get the address of where the library is loaded into memory.

Find where the functions of interest are loaded

```
var libBaseAddr = Module.getBaseAddress("libwhatsapp.so");
var JNIOnload_addr = Module.getExportByName("libwhatsapp.so", "JNI_OnLoad");
var callsCallsVulnAddr_8A360 = libBaseAddr.add(0x8A360);
var callsVulnAddr_88DB0 = libBaseAddr.add(0x88DB0);
var vulnAddr_8A8B0 = libBaseAddr.add(0x8A8B0);
```

If the function you want to hook is in the ELF's exports, you can simply use the `getExportByName` method.

Find where the functions of interest are loaded

```
var libBaseAddr = Module.getBaseAddress("libwhatsapp.so");
var JNIOnload_addr = Module.getExportByName("libwhatsapp.so", "JNI_OnLoad");
var callsCallsVulnAddr_8A360 = libBaseAddr.add(0x8A360);
var callsVulnAddr_88DB0 = libBaseAddr.add(0x88DB0);
var vulnAddr_8A8B0 = libBaseAddr.add(0x8A8B0);
```

Add the offset of the function from the base address.

Must use add() instead of + because otherwise JS thinks you want to do string operations rather than arithmetic ops.

Find where the functions of interest are loaded

```
var libBaseAddr = Module.getBaseAddress("libwhatsapp.so");
var JNIOnload_addr = Module.getExportByName("libwhatsapp.so", "JNI_OnLoad");
var callsCallsVulnAddr_8A360 = libBaseAddr.add(0x8A360);
var callsVulnAddr_88DB0 = libBaseAddr.add(0x88DB0);
var vulnAddr_8A8B0 = libBaseAddr.add(0x8A8B0);
```

Side note:
Yes, the offsets are different from the func addresses we talked about in the static analysis section. I did static analysis on the ARM32 lib without thinking that I'd be running the ARM64 one.

Find where the functions of interest are loaded

```
var libBaseAddr = Module.getBaseAddress("libwhatsapp.so");
var JNIOnload_addr = Module.getExportByName("libwhatsapp.so", "JNI_OnLoad");
var callsCallsVulnAddr_8A360 = libBaseAddr.add(0x8A360);
var callsVulnAddr_88DB0 = libBaseAddr.add(0x88DB0);
var vulnAddr_8A8B0 = libBaseAddr.add(0x8A8B0);

----------------------------------------------------------------------

libwhatsapp.so base address: "0x70e9bcb000" JNI_OnLoad: "0x70e9bfa5a4"

vuln (0x8A8B0): "0x70e9c558b0" callsVuln (0x88DB0): "0x70e9c53db0"
callsCallsVuln (0x8A360): "0x70e9c55360"
```

Let's hook the functions

```
Interceptor.attach(callsCallsVulnAddr_8A360, {
    onEnter: function (args) {
        console.log("In callsCallsVuln. Return addr: " +
                    JSON.stringify(this.returnAddress.sub(libBaseAddr)));
        return 0;
    }
});
```

Hook the function that is added to a callbacks table to print out its return address.

Let’s hook the functions

```
Interceptor.attach(vulnAddr_8A8B0, {
    onEnter: function (args) {
        console.log("** IN VULN SUB 0x8A8B0 **");
        console.log("Return addr: " +
                        JSON.stringify(this.returnAddress.sub(libBaseAddr)));
        console.log("Arg1 (Buffer): " + JSON.stringify(args[0]));
        console.log("Arg2 (Packet): " + JSON.stringify(args[1]));
        console.log("Arg3 (Len): " + JSON.stringify(args[2]));
        console.log(hexdump(args[1], {
            offset: 0,
            length: args[2].toInt32(),
            ansi:true
        }));
        return 0;
    }
});
```

While the call is ringing...

```
In callsCallsVuln. Return addr: "0x8f75c"
In callsVuln. Return addr: "0x8a468"
In callsCallsVuln. Return addr: "0x8f75c"
In callsVuln. Return addr: "0x8a468"
In callsCallsVuln. Return addr: "0x8f75c"
In callsVuln. Return addr: "0x8a468"
In callsCallsVuln. Return addr: "0x8f75c"
In callsVuln. Return addr: "0x8a468"
In callsCallsVuln. Return addr: "0x8f75c"
In callsVuln. Return addr: "0x8a468"
```

After we answer the call...

```
In callsCallsVuln. Return addr: "0x8f75c"
In callsVuln. Return addr: "0x8a468"
** IN VULN SUB 0x8A8B0 **
Return addr: "0x897ac"
Arg1 (Buffer): "0x70e120dc28"
Arg2 (Packet): "0x70fabb8038"
Arg3 (Len): "0x4a"

            0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
70fabb8038  81 ca 00 07 85 7e 02 d5 ed 2a b5 9d 88 62 1a 8a  .....~...*...b..
70fabb8048  83 7d 29 e7 5e ed 9f f2 f9 43 94 03 cc eb ad 3e  .}).^....C.....>
70fabb8058  c6 15 3e b3 7b 3a c1 a6 d1 59 ca 10 2f 03 c3 53  ..>.{:...Y../..S
70fabb8068  57 0f a8 e9 9a 58 bb 46 40 f4 41 2c 80 00 00 0a  W....X.F@.A,....
70fabb8078  c9 9e ed 5c 5b 26 e2 60 4f c6                    ...\[&.`O.
```

Now what? What's your goal of analyzing the bug?

- Understanding the vulnerability
  - Instrument the vuln function such that you change it's arguments to ones you control
- Hypothesize on what the exploit looked like
  - After understanding the vulnerability, moving up the change to see what you as the attacker can control on the other side of the WhatsApp server
- Variant analysis
  - Look for similar patterns (possibly using diffing like radiff2) through static analysis

Conclusion

Last thoughts

- No binary diffing tool out of the box will highlight *which* changes you're likely to care about. That will still take learning the tools to optimize their findings and doing some RE of your own.
- Using a variety of different RE techniques can help you get to the answer faster.
- When reversing code that use lots of callbacks, dynamic analysis can save lots of time.

THANK YOU!

@maddiestone

Google