The Evolution of Program Analysis Approaches in the Era of AI

Alex Matrosov

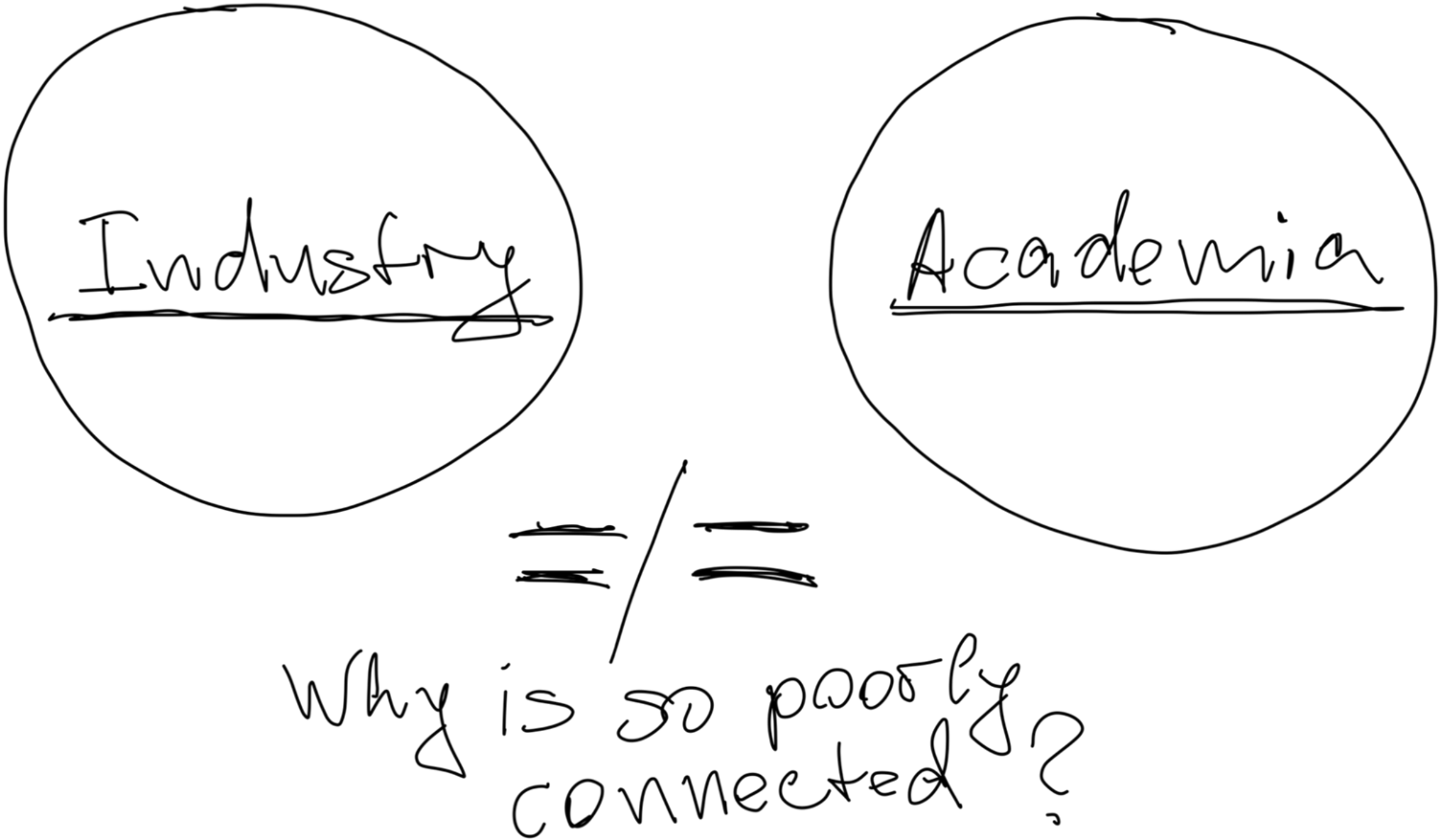
@matrosov

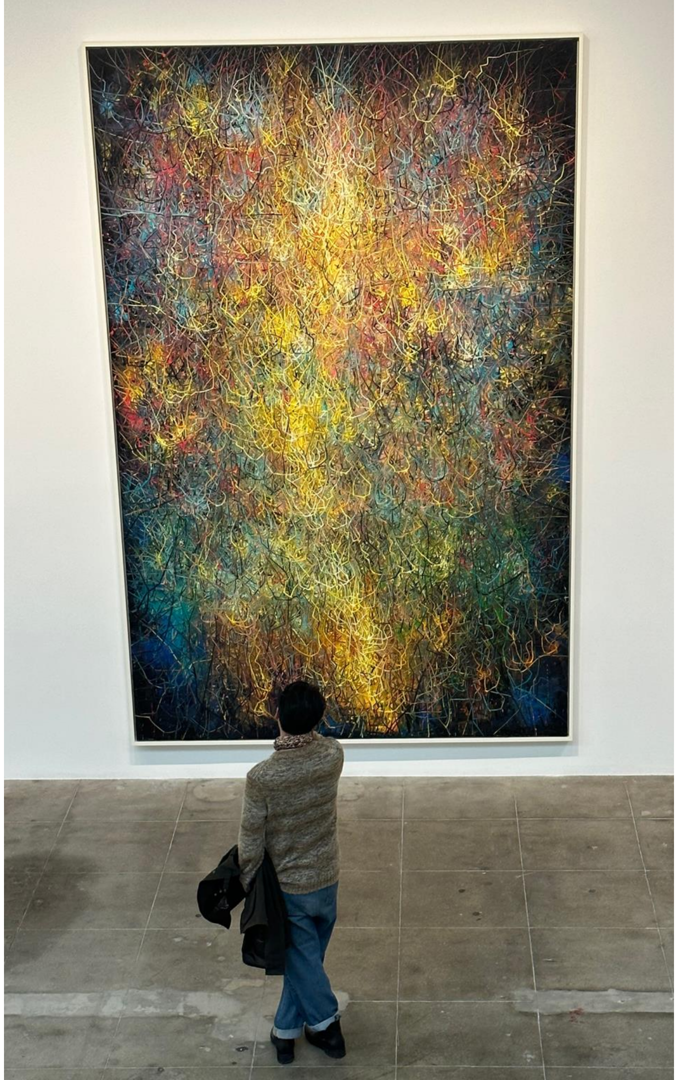
- Founder and CEO at BINARLY
- 20+ years doing all shades of binary program analysis
- Break a few times CPU’s and GPU’s
- Dedicating all my free time to surfing 🏄‍♂️


AIRE

This talk is not intended to cover the complete history of binary program analysis or reverse engineering.

I am describing the evolution of RE from the perspective of my personal experience over 20+ years.

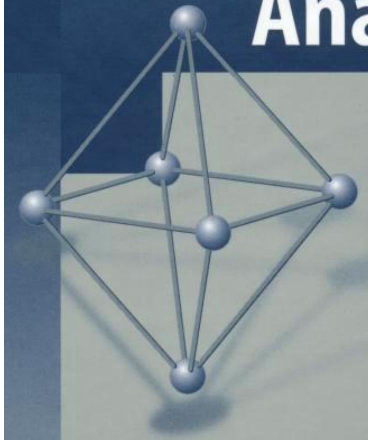

Industry
Academia
=/=
Why is so poorly connected?



STATE OF THE ART

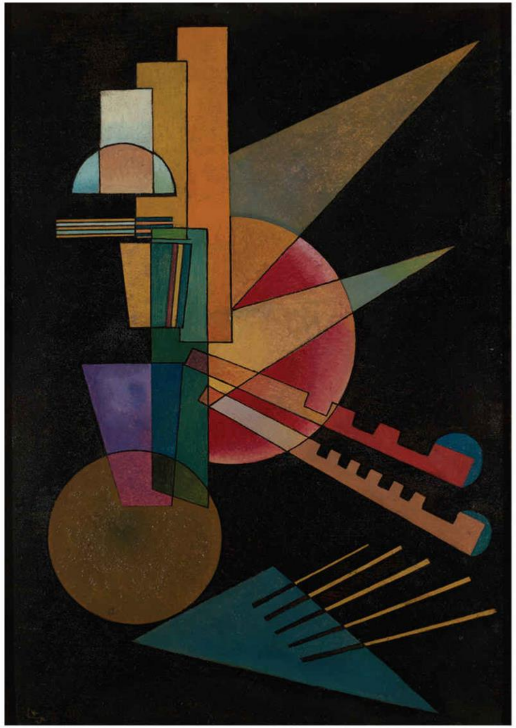
FLEMMING NIELSON
HANNE RIIS NIELSON
CHRIS HANKIN

Principles of Program Analysis



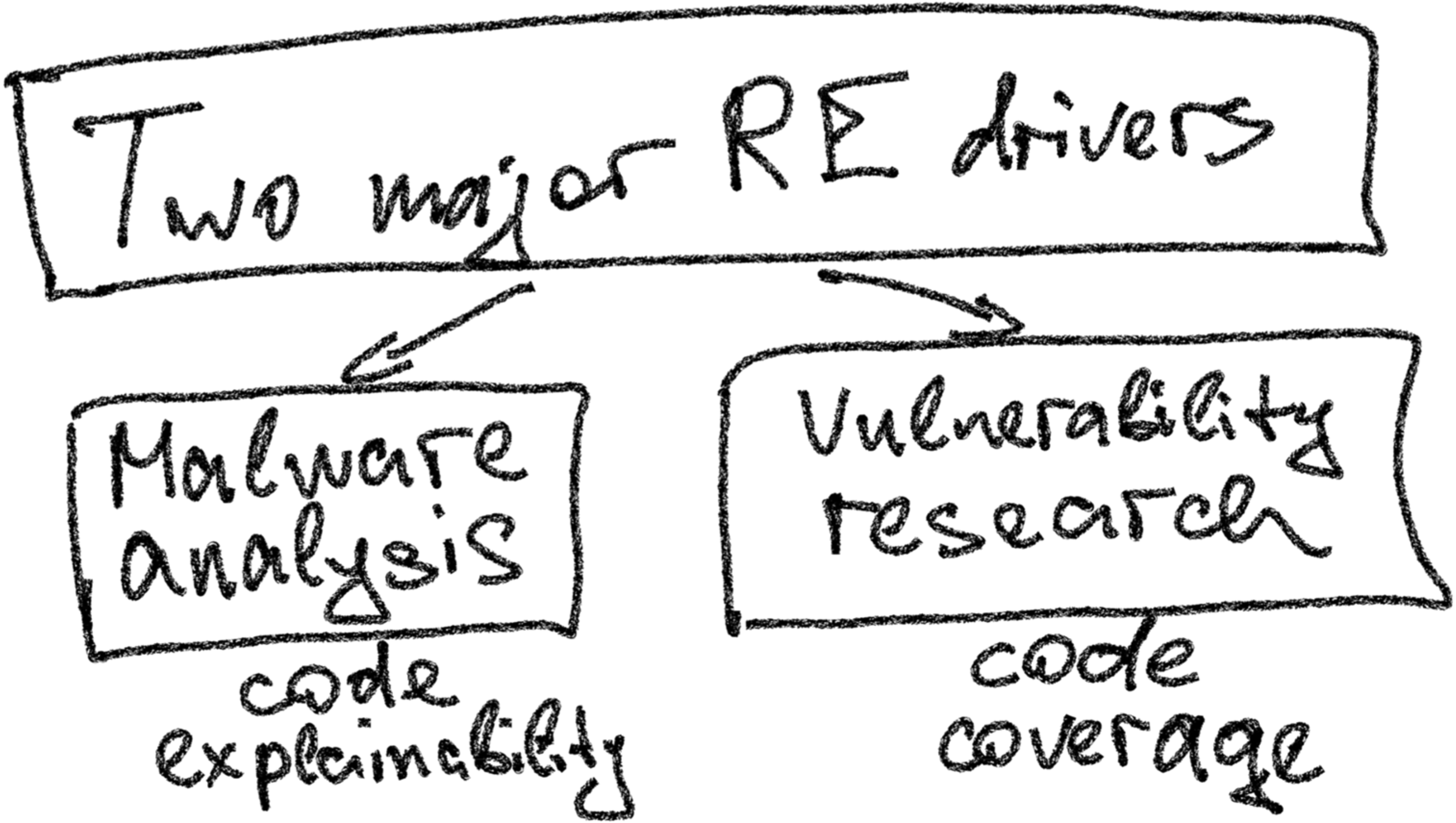
Springer

PRINCIPLES OF
ABSTRACT INTERPRETATION

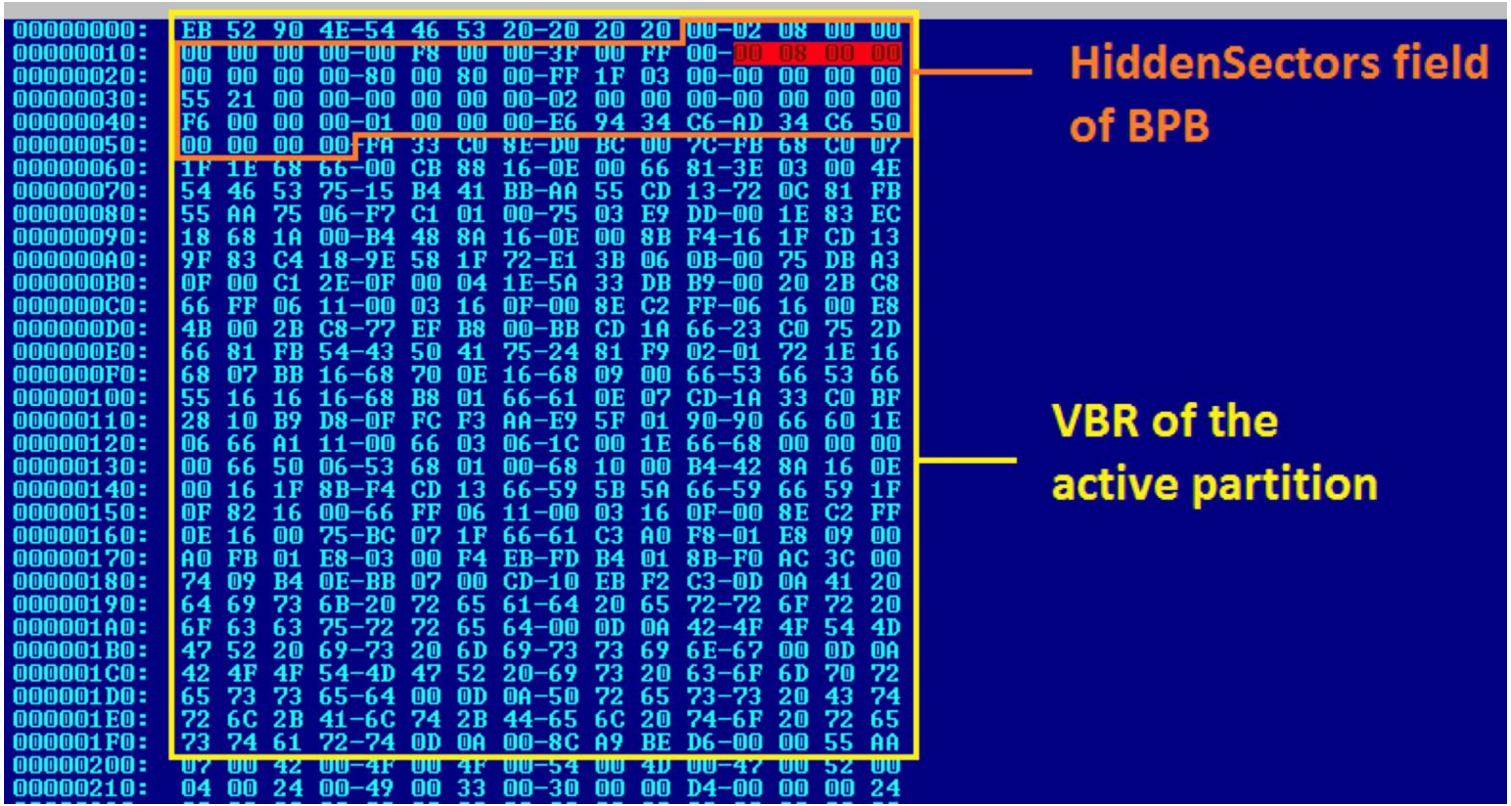


PATRICK COUSOT

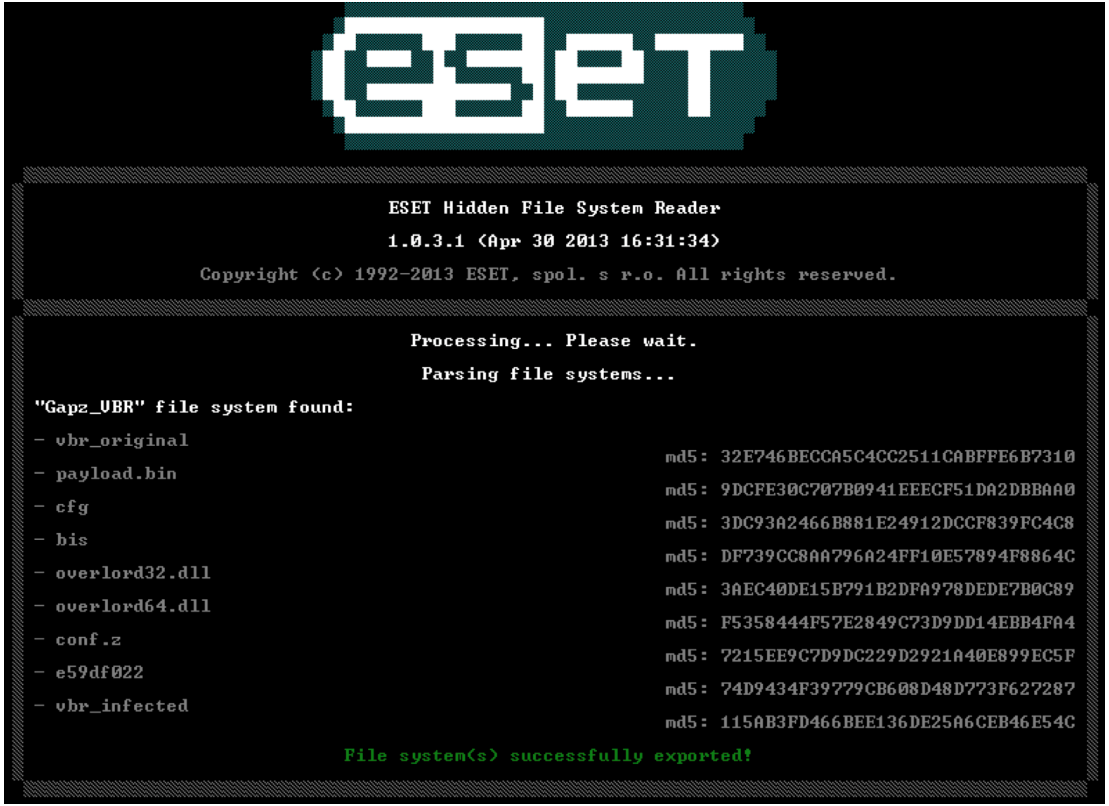

"It's all about where
your mind's at."
- Kelly Slater


Two major RE drivers
Malware analysis
Vulnerability research
code explainability
code coverage

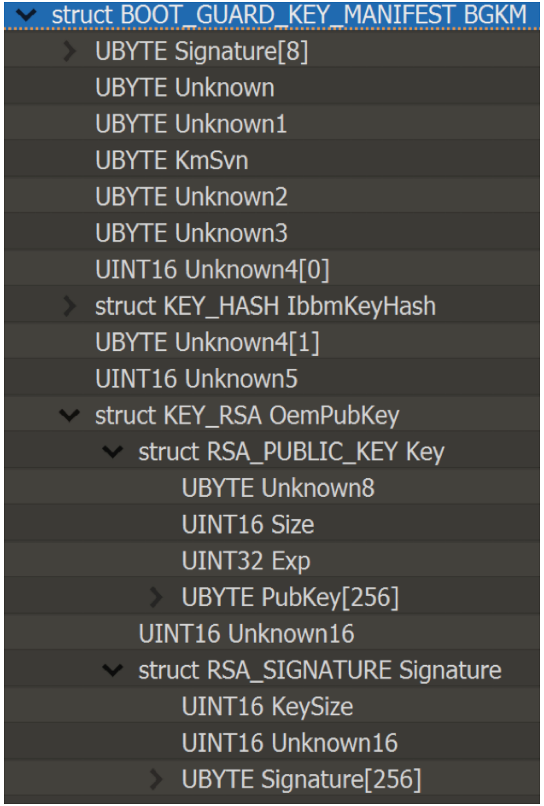
Unknown structured formats

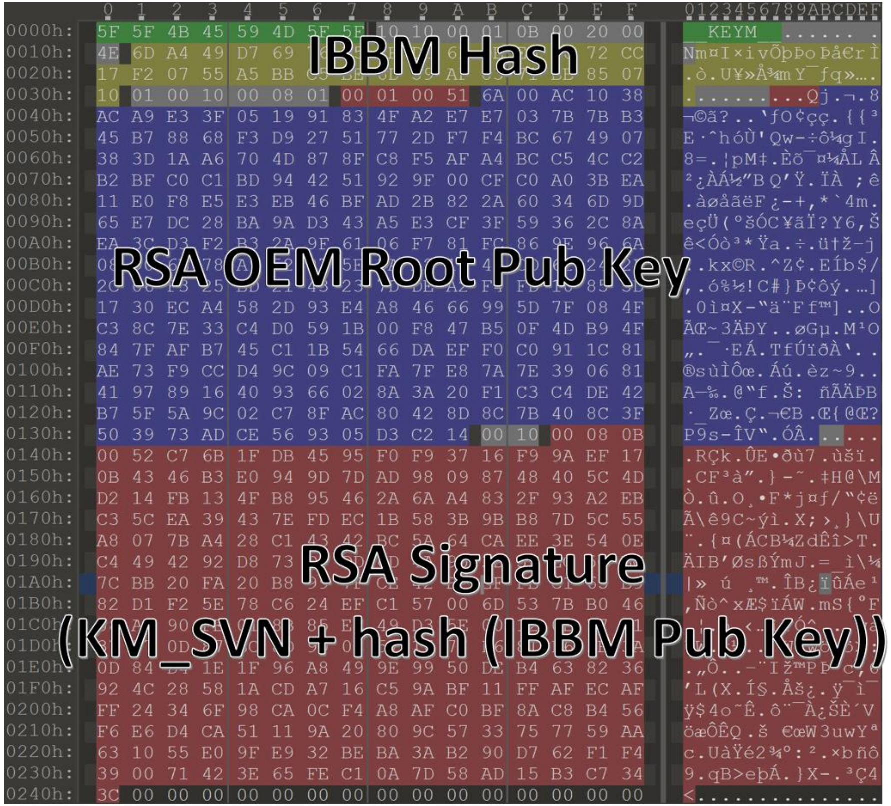


Signature-based automation

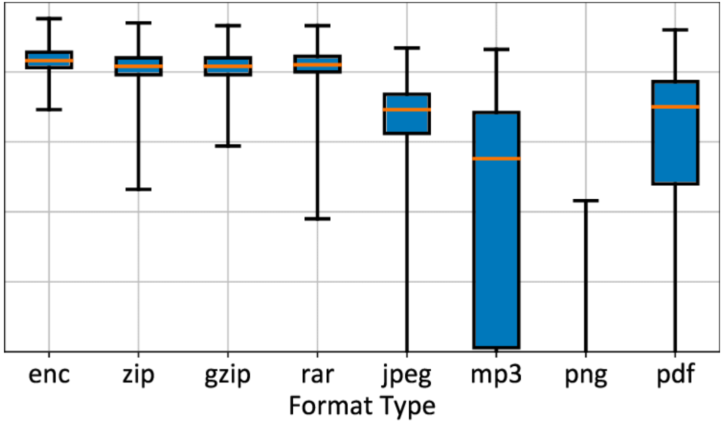


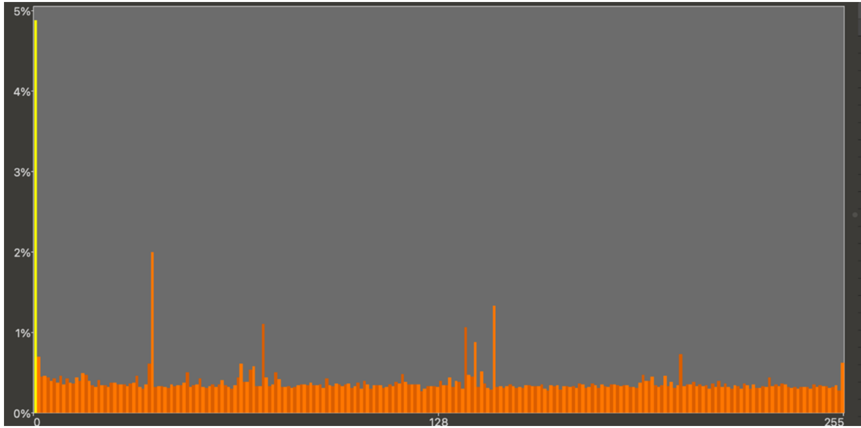
Undocumented structured formats





Entropy based ML models good for file format classification

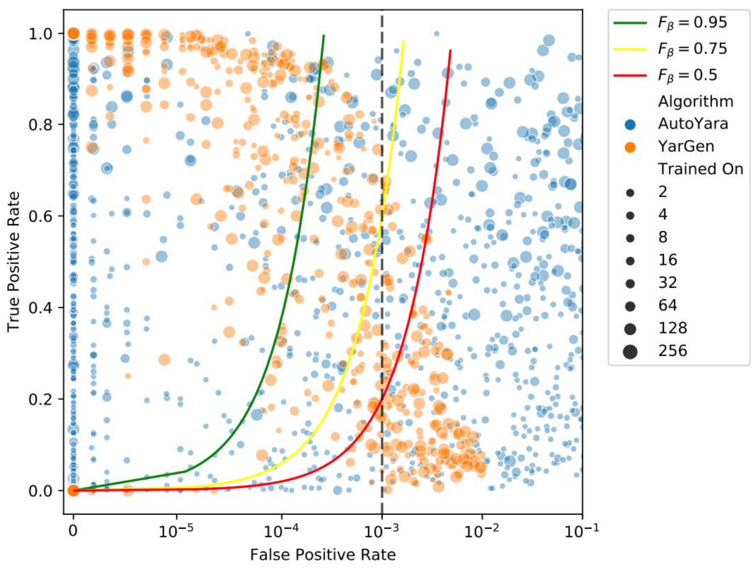




Dec	Hex	Char	Count	Percent
0	0h		3642	4.879%
1	1h		515	0.690%
2	2h		332	0.445%
3	3h		345	0.462%
4	4h		326	0.437%
5	5h		296	0.397%
6	6h		316	0.423%
7	7h		280	0.375%
8	8h		340	0.455%
9	9h		259	0.347%
10	Ah		321	0.430%
11	Bh		277	0.371%
12	Ch		267	0.358%
13	Dh		329	0.441%
14	Eh		294	0.394%
15	Fh		365	0.489%
16	10h		346	0.464%
17	11h		290	0.388%
18	12h		256	0.343%
19	13h		236	0.316%
20	14h		298	0.399%
21	15h		254	0.340%
22	16h		251	0.336%

Automated detection signature generation

Automatic Yara Rule Generation Using Biclustering



"Automatic Yara Rule Generation Using Biclustering" - 2020

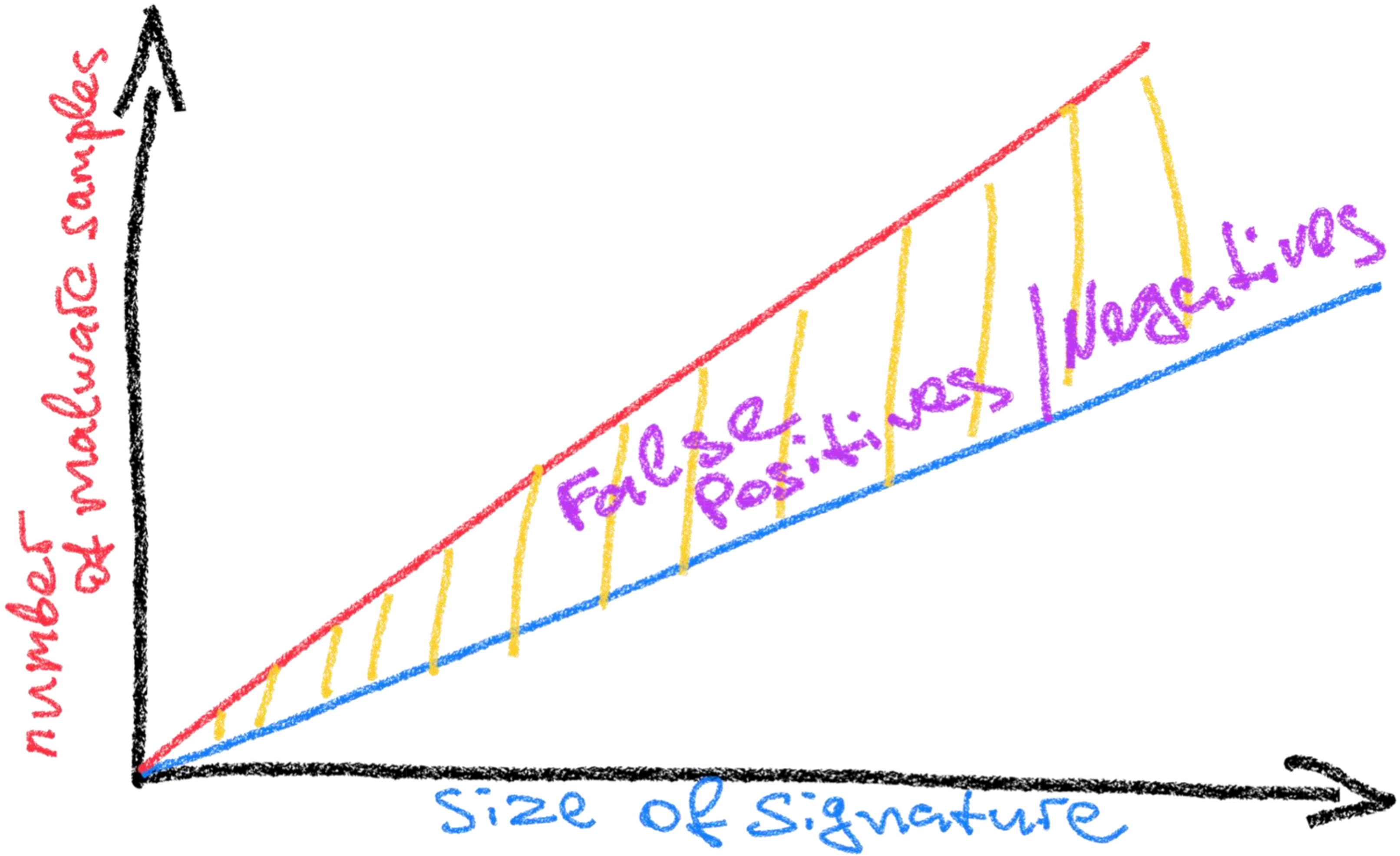
https://arxiv.org/pdf/2009.03779.pdf

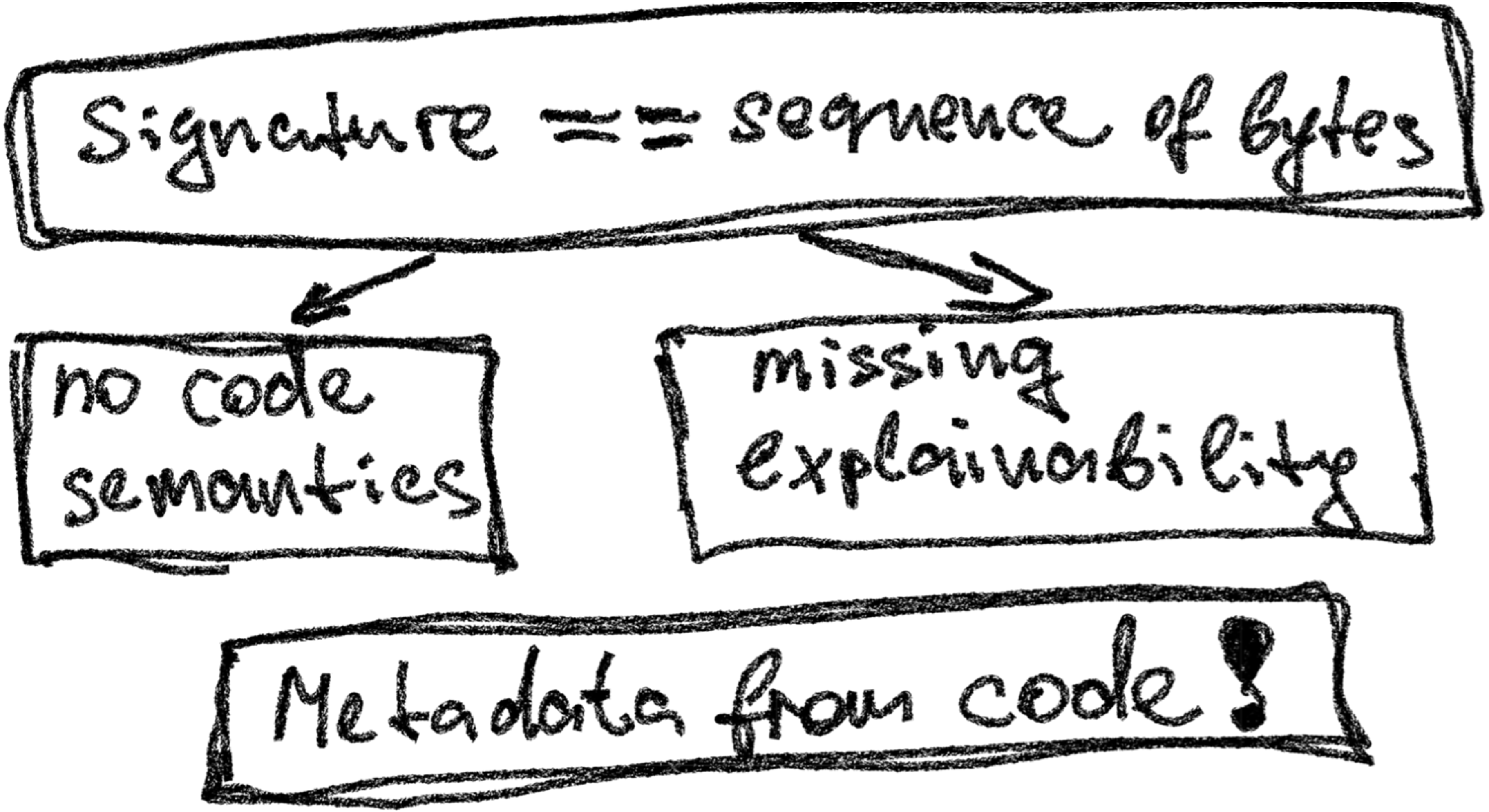
https://github.com/google/vxsig

Signature generation

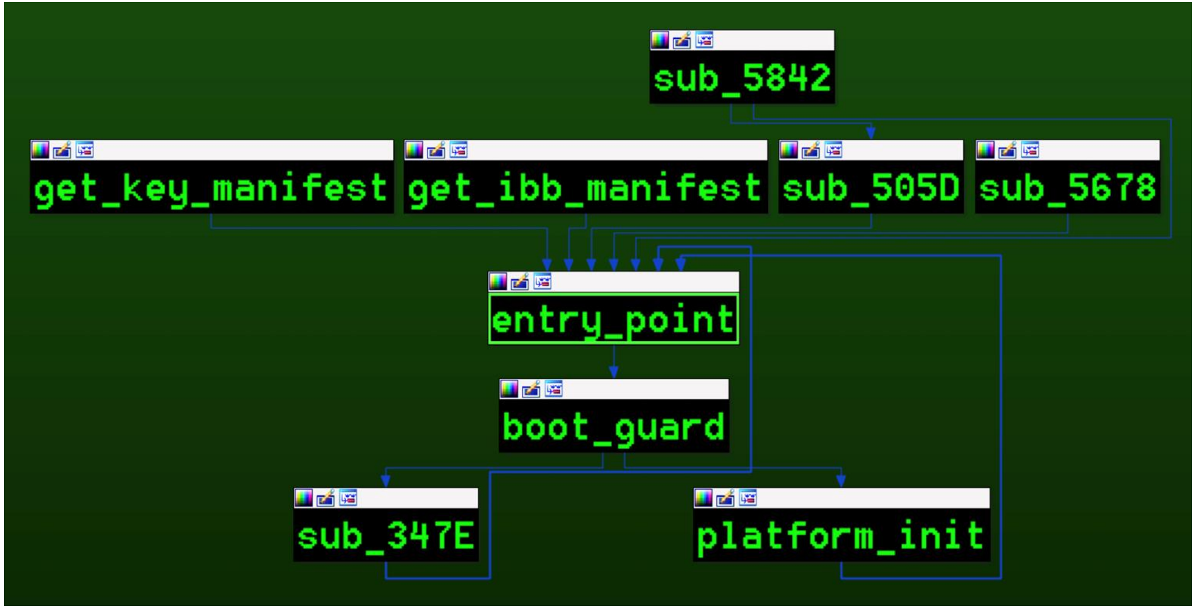
!=

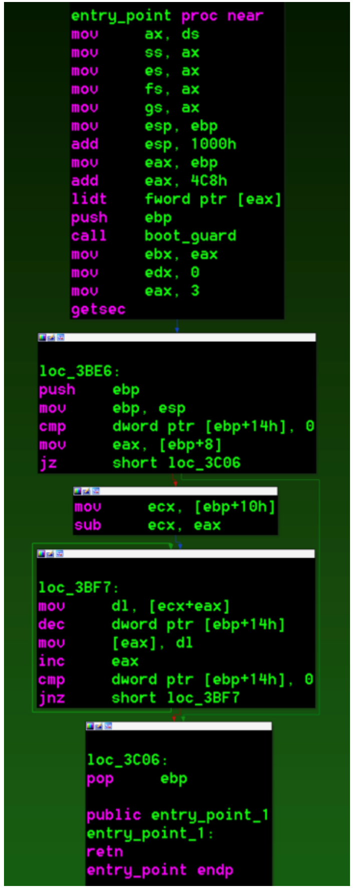
Signature Optimization


number of malware samples
False positives
Negatives
Size of signature


Signature == sequence of bytes
no code semantics
missing explainability
Metadata from code!

Disassembly CFG => Visual Graph Representation





REconstruction of complex C++ code still a problem in 2023

```
int __thiscall Rc4_GetBufferSize(_RC4_STRUCT *this)
{
  return (this->Reader->vTable->GetResBufSize)();
}
```

```
; int __thiscall Rc4_GetBufferSize(_RC4_STRUCT *this)
Rc4_GetBufferSize proc near               ; DATA XREF:
                mov     ecx, [ecx+4]
                mov     eax, [ecx]
                jmp     dword ptr [eax+10h]
Rc4_GetBufferSize endp
```

```
RC4_VTABLE      dd offset Rc4_GetReader ; DATA XREF: sub_1011E919+1E↑o
                dd offset Rc4_GetWriter
                dd offset ?Destroy@EventWaitNode@details@Concurrency@@QAEXXZ
                dd offset ?Sweep@EventWaitNode@details@Concurrency@@QAE_NXZ
                dd offset Rc4_GetBufferSize
                dd offset Rc4_IncreaseSize
                dd offset Rc4_Check
                dd offset Rc4_InitEmpty
                dd offset Rc4_Release
                dd offset Rc4_GetMuxName
```

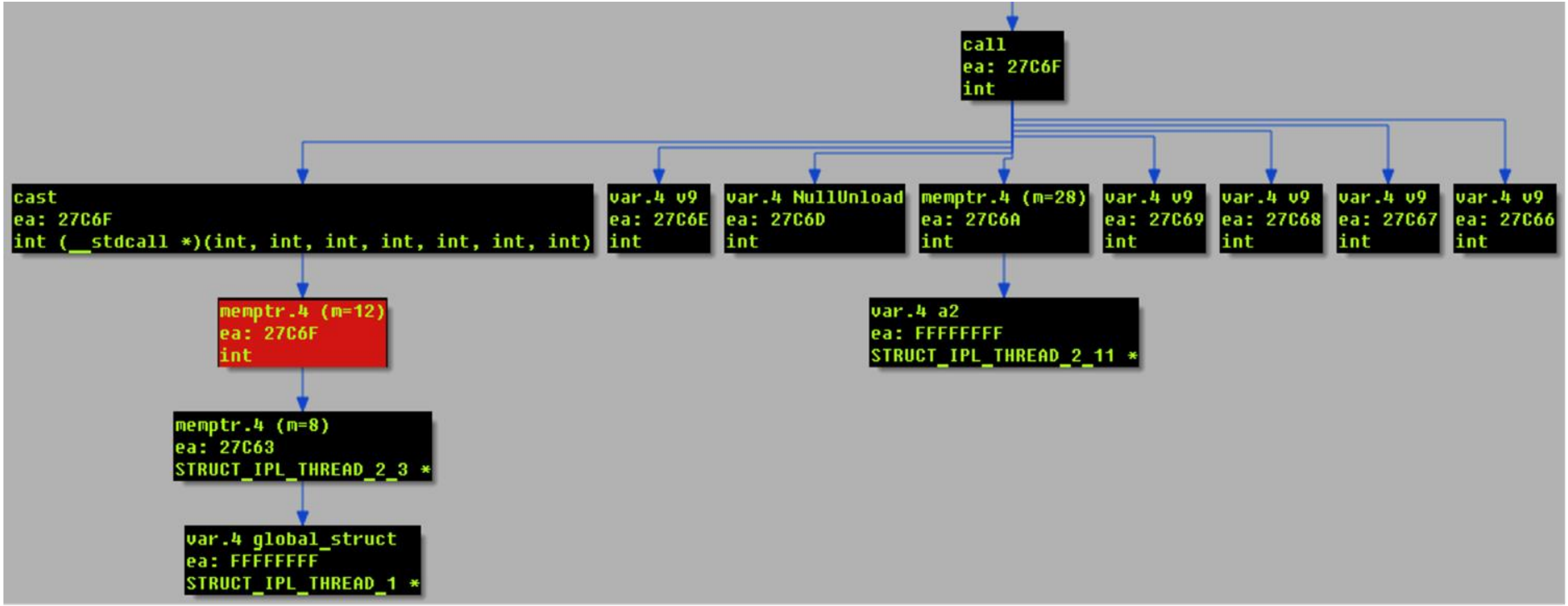
REIL: A platform-independent intermediate representation of disassembled code for static code analysis

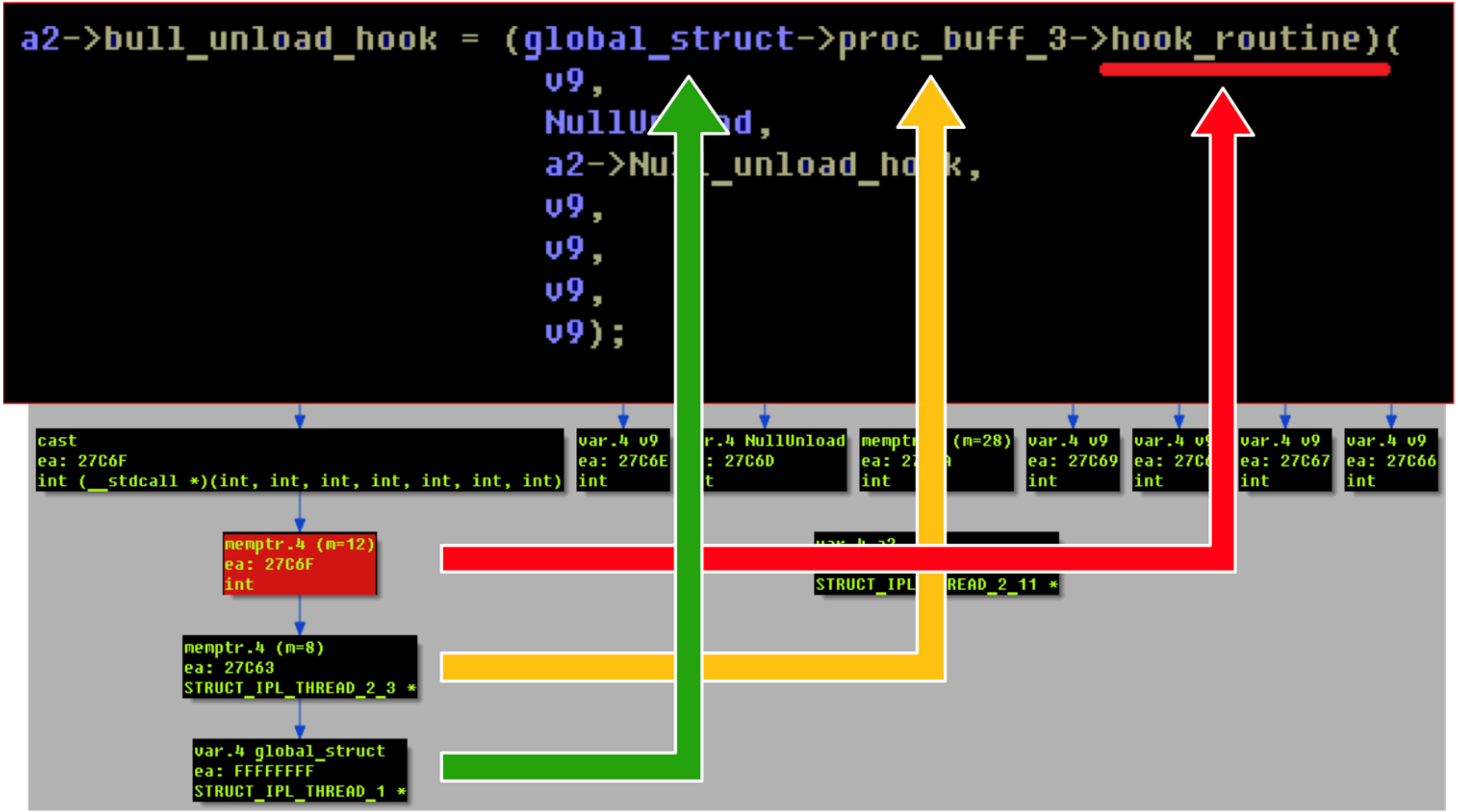
Thomas Dullien
zynamics GmbH
Bochum, Germany
thomas.dullien@zynamics.com

Sebastian Porst
zynamics GmbH
Bochum, Germany
sebastian.porst@zynamics.com

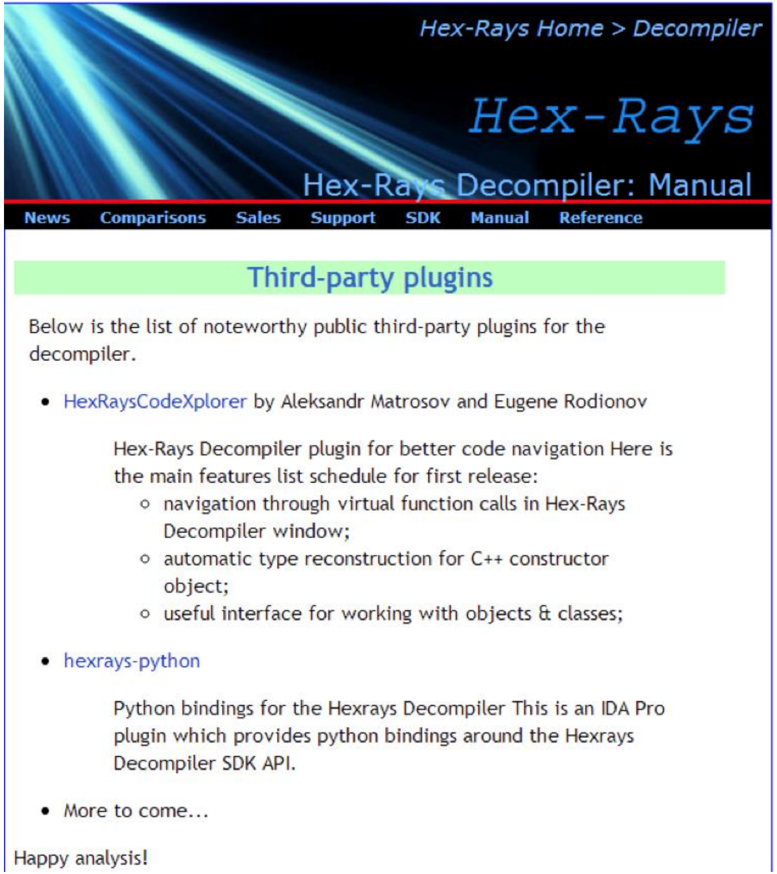
https://static.googleusercontent.com/media/www.zynamics.com/en//downloads/csw09.pdf

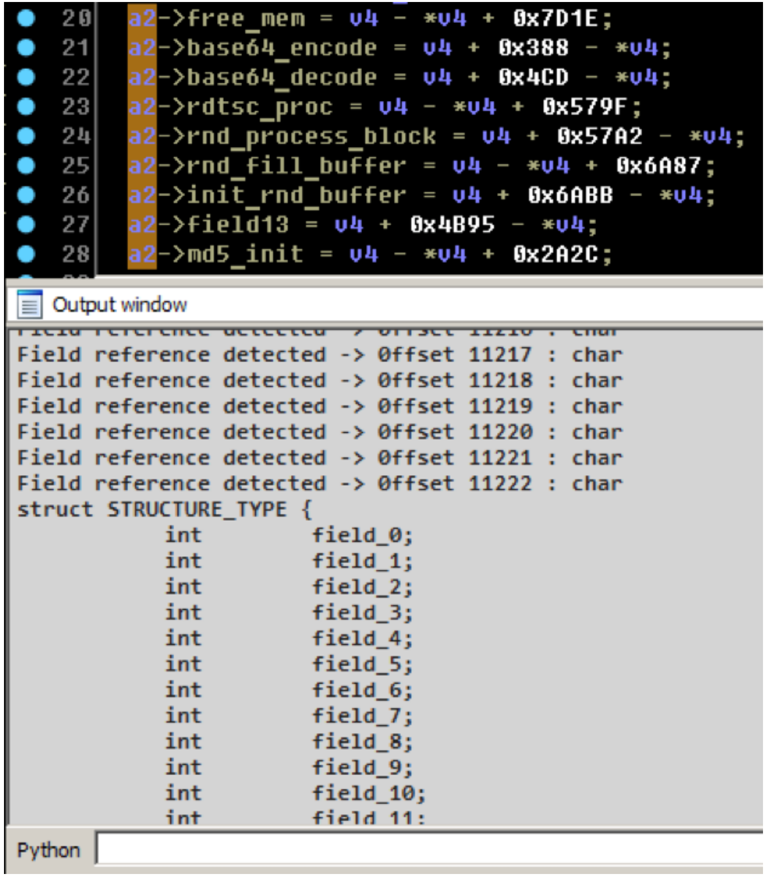
IR => Code Semantics




a2->bull_unload_hook = (global_struct->proc_buff_3->hook_routine)(
v9,
v9,
v9,
v9,
v9);
cast
ea: 27C6F
int (__stdcall *)(int, int, int, int, int, int, int)
var.4 v9
ea: 27C6E
int
NullUnload
: 27C6D
(m=28)
int
var.4 v9
ea: 27C69
int
int
var.4 v9
ea: 27C67
int
var.4 v9
ea: 27C66
int
memptr.4 (m=12)
ea: 27C6F
int
memptr.4 (m=8)
ea: 27C63
STRUCT_IPL_THREAD_2_3 *
var.4 global_struct
ea: FFFFFFFF
STRUCT_IPL_THREAD_1 *

HexRaysCodeXplorer v1.0: released in 2013 at REcon





Ghidra P-Code more suitable for RE needs vs Hex-Rays IR

```
180002aaa 42 0f b7 14   MOVZX   size,word ptr [RBX + last_index*0x2]
          53
                                           (unique, 0x3300, 8) = INT_MULT (register, 0x90, 8), (const, 0x2, 8)
                                           (unique, 0x3400, 8) = INT_ADD (register, 0x18, 8), (unique, 0x3300, 8)
                                           (unique, 0xbe80, 2) = LOAD (const, 0x1b1, 4), (unique, 0x3400, 8)
                                           (register, 0x10, 4) = INT_ZEXT (unique, 0xbe80, 2)
                                           (register, 0x10, 8) = INT_ZEXT (register, 0x10, 4)
180002aaf 4c 8d 1c 4d   LEA     R11,[0x8 + t*0x2]
          08 00 00 00
                                           (unique, 0x3480, 8) = INT_MULT (register, 0x8, 8), (const, 0x2, 8)
                                           (unique, 0x3580, 8) = INT_ADD (const, 0x8, 8), (unique, 0x3480, 8)
                                           (register, 0x98, 8) = COPY (unique, 0x3580, 8)
180002ab7 4c 03 dd      ADD     Decrypted,RBP
                                           (register, 0x200, 1) = INT_CARRY (register, 0x98, 8), (register, 0x28, 8)
                                           (register, 0x20b, 1) = INT_SCARRY (register, 0x98, 8), (register, 0x28, 8)
                                           (register, 0x98, 8) = INT_ADD (register, 0x98, 8), (register, 0x28, 8)
                                           (register, 0x207, 1) = INT_SLESS (register, 0x98, 8), (const, 0x0, 8)
                                           (register, 0x206, 1) = INT_EQUAL (register, 0x98, 8), (const, 0x0, 8)
                                           (unique, 0x12e80, 8) = INT_AND (register, 0x98, 8), (const, 0xff, 8)
                                           (unique, 0x12f00, 1) = POPCOUNT (unique, 0x12e80, 8)
                                           (unique, 0x12f80, 1) = INT_AND (unique, 0x12f00, 1), (const, 0x1, 1)
                                           (register, 0x202, 1) = INT_EQUAL (unique, 0x12f80, 1), (const, 0x0, 1)
180002aba 66 83 fa 7f   CMP     size,0x7f
                                           (register, 0x200, 1) = INT_LESS (register, 0x10, 2), (const, 0x7f, 2)
                                           (register, 0x20b, 1) = INT_SBORROW (register, 0x10, 2), (const, 0x7f, 2)
                                           (unique, 0x28f00, 2) = INT_SUB (register, 0x10, 2), (const, 0x7f, 2)
                                           (register, 0x207, 1) = INT_SLESS (unique, 0x28f00, 2), (const, 0x0, 2)
                                           (register, 0x206, 1) = INT_EQUAL (unique, 0x28f00, 2), (const, 0x0, 2)
                                           (unique, 0x12e80, 2) = INT_AND (unique, 0x28f00, 2), (const, 0xff, 2)
                                           (unique, 0x12f00, 1) = POPCOUNT (unique, 0x12e80, 2)
                                           (unique, 0x12f80, 1) = INT_AND (unique, 0x12f00, 1), (const, 0x1, 1)
                                           (register, 0x202, 1) = INT_EQUAL (unique, 0x12f80, 1), (const, 0x0, 1)
```

Ghidra P-Code more suitable for RE needs vs Hex-Rays IR

```
0000000000180002AAA  movzx edx,word ptr [rbx + r10*0x2]
  unique[0x3300:8] = R10 * 0x2
  unique[0x3400:8] = RBX + unique[0x3300:8]
  unique[0xbe80:2] = *[ram]unique[0x3400:8]
  EDX = zext(unique[0xbe80:2])
  RDX = zext(EDX)

0000000000180002AAF  lea r11,[0x8 + rcx*0x2]
  unique[0x3480:8] = RCX * 0x2
  unique[0x3580:8] = 0x8 + unique[0x3480:8]
  R11 = unique[0x3580:8]

0000000000180002AB7  add r11,rbp
  CF = carry(R11, RBP)
  OF = scarry(R11, RBP)
  R11 = R11 + RBP
  SF = R11 s< 0x0
  ZF = R11 == 0x0
  unique[0x12e80:8] = R11 & 0xff
  unique[0x12f00:1] = popcount(unique[0x12e80:8])
  unique[0x12f80:1] = unique[0x12f00:1] & 0x1
  PF = unique[0x12f80:1] == 0x0

0000000000180002ABA  cmp dx,0x7f
  CF = DX < 0x7f
  OF = sborrow(DX, 0x7f)
  unique[0x28f00:2] = DX - 0x7f
  SF = unique[0x28f00:2] s< 0x0
  ZF = unique[0x28f00:2] == 0x0
  unique[0x12e80:2] = unique[0x28f00:2] & 0xff
  unique[0x12f00:1] = popcount(unique[0x12e80:2])
  unique[0x12f80:1] = unique[0x12f00:1] & 0x1
  PF = unique[0x12f80:1] == 0x0
```

The most comprehensive IR for RE is developed by Binary Ninja

```
000011bc   EFI_STATUS sub_11bc(EFI_HANDLE* ImageHandle, struct EFI_SYSTEM_TABLE* SystemTable)

   0 @ 000011bc  __saved_rsi = rsi
   1 @ 000011bc  rsp = &__saved_rsi
   2 @ 000011be  rsp = &var_38
   3 @ 000011c2  rax = [SystemTable + 0x58].q
   4 @ 000011c6  r8 = SignalExitBootServicesNotifier8
   5 @ 000011cd  r10 = [SystemTable + 0x60].q
   6 @ 000011d1  r9 = 0
   7 @ 000011d4  [gRT1].q = rax
   8 @ 000011db  ImageHandle = 0x201
   9 @ 000011e0  [gRT2].q = rax
  10 @ 000011e7  rax = gSignalExitBootServicesEvent8
  11 @ 000011ee  [gST].q = SystemTable
  12 @ 000011f5  SystemTable = 8
  13 @ 000011f9  [gBS1].q = r10
  14 @ 00001200  [gBS2].q = r10
  15 @ 00001207  var_18 = gSignalExitBootServicesEvent8
  16 @ 0000120c  rax, ImageHandle, SystemTable, r8, r9, r10, r11, xmm4, xmm5 = call([r10 + 0x50].q, ImageHandle, SystemTable, r8, r9, stack = &var_38)
  17 @ 00001210  rax = gSignalVirtualAddressChangeEvent8
  18 @ 00001217  r9 = 0
  19 @ 0000121a  var_18 = gSignalVirtualAddressChangeEvent8
  20 @ 0000121f  r8 = SignalVirtualAddressChangeNotifier8
```

```
000011bc   EFI_STATUS sub_11bc(EFI_HANDLE* ImageHandle, struct EFI_SYSTEM_TABLE* SystemTable)

   0 @ 000011bc  __saved_rsi#1 = rsi#0
   1 @ 000011bc  rsp#1 = &__saved_rsi
   2 @ 000011be  rsp#2 = &var_38
   3 @ 000011c2  rax#1 = [SystemTable#0 + 0x58].q @ mem#0
   4 @ 000011c6  r8#1 = SignalExitBootServicesNotifier8
   5 @ 000011cd  r10#1 = [SystemTable#0 + 0x60].q @ mem#0
   6 @ 000011d1  r9#1 = 0
   7 @ 000011d4  [gRT1].q = rax#1 @ mem#0 -> mem#1
   8 @ 000011db  ImageHandle#1 = 0x201
   9 @ 000011e0  [gRT2].q = rax#1 @ mem#1 -> mem#2
  10 @ 000011e7  rax#2 = gSignalExitBootServicesEvent8
  11 @ 000011ee  [gST].q = SystemTable#0 @ mem#2 -> mem#3
  12 @ 000011f5  SystemTable#1 = 8
  13 @ 000011f9  [gBS1].q = r10#1 @ mem#3 -> mem#4
  14 @ 00001200  [gBS2].q = r10#1 @ mem#4 -> mem#5
  15 @ 00001207  var_18#1 = gSignalExitBootServicesEvent8
```

Decompilation != Silver Bullet

```
void __fastcall sub_180001000(void *a1, char a2, unsigned __int64 a3)
{
  if ( a3 )
    memset(a1, a2, a3);
}
```

BinaryNinja

3.3.3996 (e34a955e)

```
void sub_180001000(char* arg1, char arg2, int64_t arg3)
{
    if (arg3 != 0)
    {
        char* rdi_1 = arg1;
        int64_t rax = arg2;
        for (int64_t rcx = arg3; rcx != 0; rcx = (rcx - 1))
        {
            *rdi_1 = rax;
            rdi_1 = &rdi_1[1];
        }
    }
}
```

Ghidra

10.2.2 (9813cde2)

```
void FUN_180001000(undefined *param_1,undefined param_2,longlong par

{
  if (param_3 != 0) {
    for (; param_3 != 0; param_3 = param_3 + -1) {
      *param_1 = param_2;
      param_1 = param_1 + 1;
    }
  }
  return;
}
```

The decompilation of Golang is a disaster

```
__int64 __fastcall stringToGuid(
        __int64 a1,
        __int64 a2,
        __int64 a3,
        __int64 a4,
        __int64 a5,
        __int64 a6,
        __int64 a7,
        __int64 a8,
        __int64 a9,
        __int64 a10,
        __int64 a11,
        __int64 a12,
        __int64 a13,
        __int64 a14,
        __int64 a15,
        __int64 a16,
        __int64 a17,
        unsigned __int64 a18,
        __int64 a19,
        __int64 a20,
        __int64 a21,
        __int64 a22)
{
  __int64 v22; // x28
  __int64 v23; // x0
  __int64 v24; // x1
  unsigned __int64 v25; // x2
  unsigned __int64 v26; // x0
  __int64 v27; // x3
  __int64 v28; // x4
  __int64 v29; // x5
  __int64 v30; // x6
  __int64 v31; // x7
  __int64 v33; // [xsp+8h] [xbp-60h]
  __int64 v34; // [xsp+10h] [xbp-58h]
  __int64 v35; // [xsp+18h] [xbp-50h]
  __int64 v36; // [xsp+20h] [xbp-48h]
  unsigned __int64 v37; // [xsp+50h] [xbp-18h]
  __int64 v38; // [xsp+58h] [xbp-10h]

  while ( &a9 <= *(v22 + 16) )
  {
    a10 = a1;
    a11 = a2;
    runtime_morestack_noctxt_abi0();
    a1 = a10;
    a2 = a11;
  }
  a10 = a1;
  v34 = strings_Replace();
  v38 = runtime_stringtoslicebyte(0LL, v23, v24);
  v37 = v25;
  v26 = encoding_hex_Decode();
  if ( v37 < v26 )
    runtime_panicSliceAcap();
  return bytesToGUID(v38, v26, v37, v27, v28, v29, v30, v31, v33, v34, v35, v36);
}
```

The decompilation of Rust is a disaster

```
  v7 = atomic_load(&uefi::parsers::bg::KEYM_TAG::he79a90e6d0585aaf);
  if ( v7 != 2 )
    once_cell::imp::OnceCell$LT$T$GT$::initialize::h0ec87ed7e59f3858(
      &uefi::parsers::bg::KEYM_TAG::he79a90e6d0585aaf,
      &uefi::parsers::bg::KEYM_TAG::he79a90e6d0585aaf);
  regex::re_bytes::Regex::find_iter::hef832e90fc36d1e1(&unk_10019C008, a1, a2);
  v45 = v41;
  v46 = v42;
  v47 = v43;
  v48 = v44;
  _$LT$alloc..vec..Vec$LT$T$GT$$u20$as$u20$alloc..vec..spec_from_iter..SpecFromIter$LT$T$C$I$GT$$GT$::from_iter::h28d513b9e8021fff(&v45);
  v8 = v40;
  result = alloc::slice::merge_sort::h80d655c35f129dbe();
  if ( v40 >= 2 )
  {
    v31 = v40 - 1;
    v32 = (unsigned __int64 *)(v38 + 8);
    v8 = 1LL;
    do
    {
      v10 = *v32;
      if ( *v32 != *(_QWORD *)(v38 + 8 * v8 - 8) )
        *(_QWORD *)(v38 + 8 * v8++) = v10;
      ++v32;
      --v31;
    }
    while ( v31 );
  }
  if ( !v8 )
  {
LABEL_42:
    if ( v39 )
      result = __rust_dealloc(v38, 8 * v39);
    a3[2] = 0LL;
    a3[3] = 0LL;
    *a3 = 11LL;
    a3[1] = 1LL;
    return result;
  }
  v11 = 0LL;
  v12 = 8 * v8;
  v13 = 1LL;
  v14 = 14LL;
  v36 = 14LL;
  v37 = 1LL;
  while ( 1 )
  {
    result = *(_QWORD *)(v38 + v11);
    if ( result > a2 )
      core::slice::index::slice_start_index_len_fail::h2a4533b191a8042c();
```

Next directions for REsearch

- Utilize more Data Flow Analysis, industry is too focused on Control Flow Analysis and missing out on Data Semantics.

- Data and Code Reconstruction required specific methods to preserve code and its dependencies, as well as fast methods of querying this data. Datalog can be used to represent data and code in a deductive database, but it requires a large amount of memory.

- Infer ML models based on code semantics, not byte sequences, which lack context.


Use NewAE Probes Only.
Amp
Volt. 20:1
ARM
Press for SAFE
Hold for ARM
Status
Ready
Pulse
Fault
Open
Replace probe tip.
Hold ARM to reset.
W520
Equipment to be used by trained personnel only.
RoHS
M.2 KEY E
SATA

SIGN IN

The Register®

{* SECURITY *}

One month after Black Hat disclosure, HP's enterprise kit still unpatched

What could go wrong with leaving firmware open after world's biggest hacker convention talk?

https://www.theregister.com/2022/09/13/firmware_bugs_hp

Firmware Repeatable Failures

Vendor	Vulnerabilities	Number of Issues	BINARLY ID	CVE ID	CVSS score
intel / ami	PEI Memory Corruption (Arbitrary Code Execution)	3	BRLY-2022-027 BRLY-2022-009 BRLY-2022-014	CVE-2022-28858 CVE-2022-36372 CVE-2022-32579	8.2 High 8.2 High 7.2 High
	DXE Arbitrary Code Execution	1	BRLY-2022-015	CVE-2022-34345	7.2 High
	SMM Memory Corruption (Arbitrary Code Execution)	2	BRLY-2022-003 BRLY-2022-016	CVE-2022-27493 CVE-2022-33209	7.5 High 8.2 High
hp	SMM Memory Corruption (Arbitrary Code Execution)	6	BRLY-2022-010 BRLY-2022-011 BRLY-2022-012 BRLY-2022-013 BRLY-2021-046 BRLY-2021-047	CVE-2022-23930 CVE-2022-31644 CVE-2022-31645 CVE-2022-31646 CVE-2022-31640 CVE-2022-31641	8.2 High 7.5 High 8.2 High 8.2 High 7.5 High 7.5 High

Firmware Security Repeatable Failures

12🔥

Black Hat 2022
Disclosure Date

HP
Multiple unfixed vulnerabilities

AMI
Multiple unfixed vulnerabilities industry-wide

Lenovo
Multiple unfixed AMI vulnerabilities

10🔥

Insyde
7 new High-Impact CVE's in Insyde reference code

Vulnerabilities in the Insyde (industry-wide impact)

BRLY	CVE	CVSS v3	Description
BRLY-2022-017	CVE-2022-36338	7.5 High	SMM callout vulnerability in SMM driver (SMM arbitrary code execution)
BRLY-2022-018	CVE-2022-35894	6.0 Medium	SMM memory leak vulnerability in SMM driver (SMRAM read)
BRLY-2022-019	CVE-2022-36337	7.7 High	The stack buffer overflow vulnerability in DXE driver
BRLY-2022-020	CVE-2022-35407	7.7 High	The stack buffer overflow vulnerability in DXE driver
BRLY-2022-021	CVE-2022-35897	7.7 High	The stack buffer overflow vulnerability in DXE driver
BRLY-2022-022	CVE-2022-35408	7.5 High	SMM callout vulnerability in SMM driver (SMM arbitrary code execution)
BRLY-2022-023	CVE-2022-36448	8.2 High	SMM memory corruption vulnerability in Software SMI handler
BRLY-2022-024	CVE-2022-35895	8.2 High	SMM memory corruption vulnerability in SMM driver (SMRAM write)
BRLY-2022-025	CVE-2022-35896	6.0 Medium	SMM memory leak vulnerability in SMM driver (SMRAM read)
BRLY-2022-026	CVE-2022-35893	8.2 High	SMM memory corruption vulnerability in SMM driver (SMRAM write)

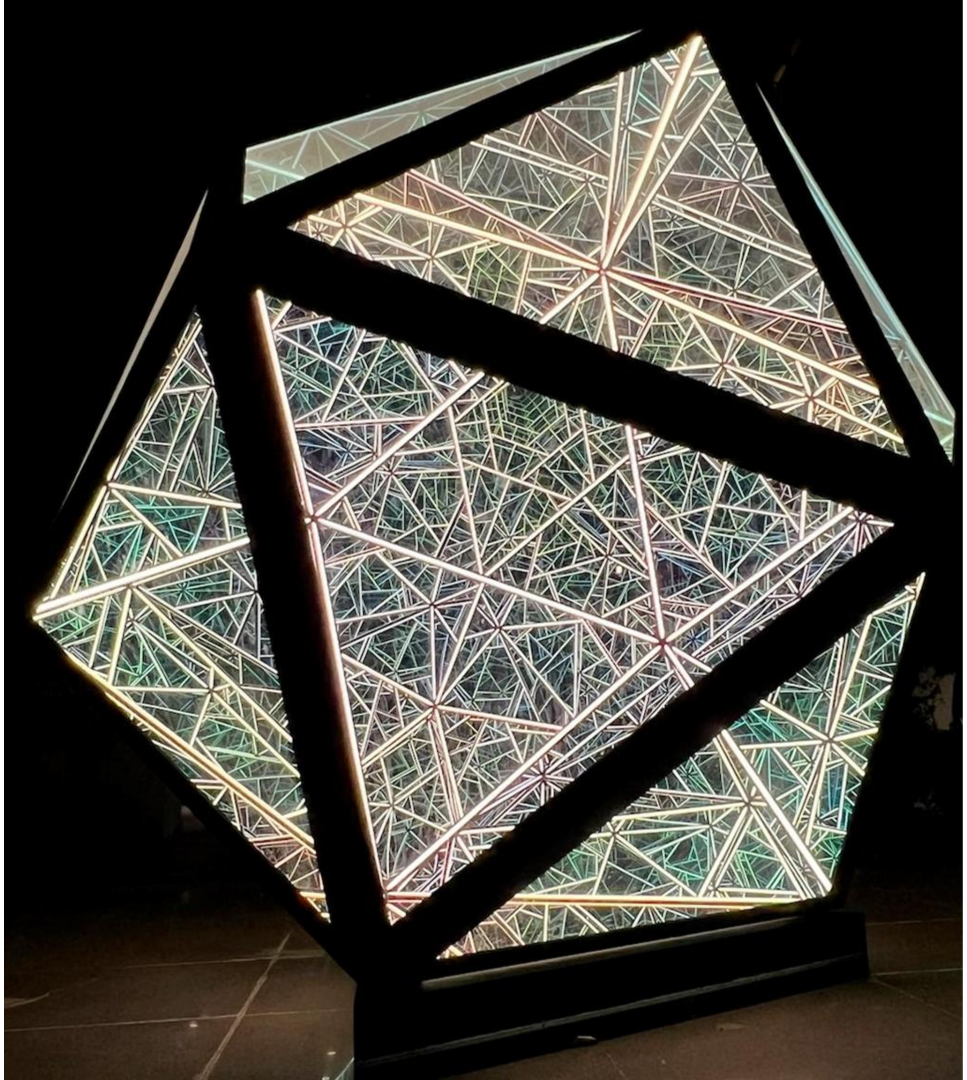
BINARLY 🔬

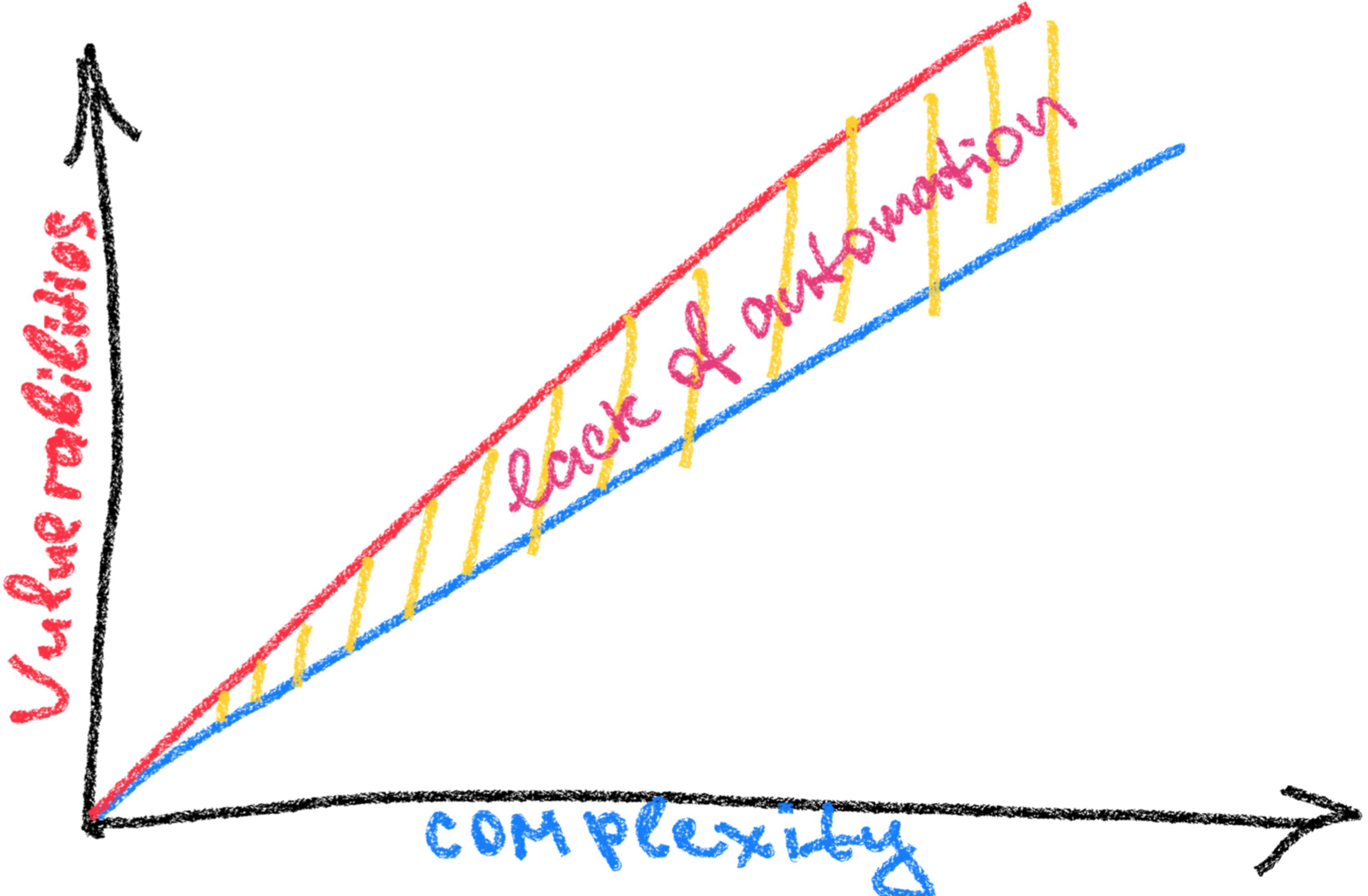
@binarly_io

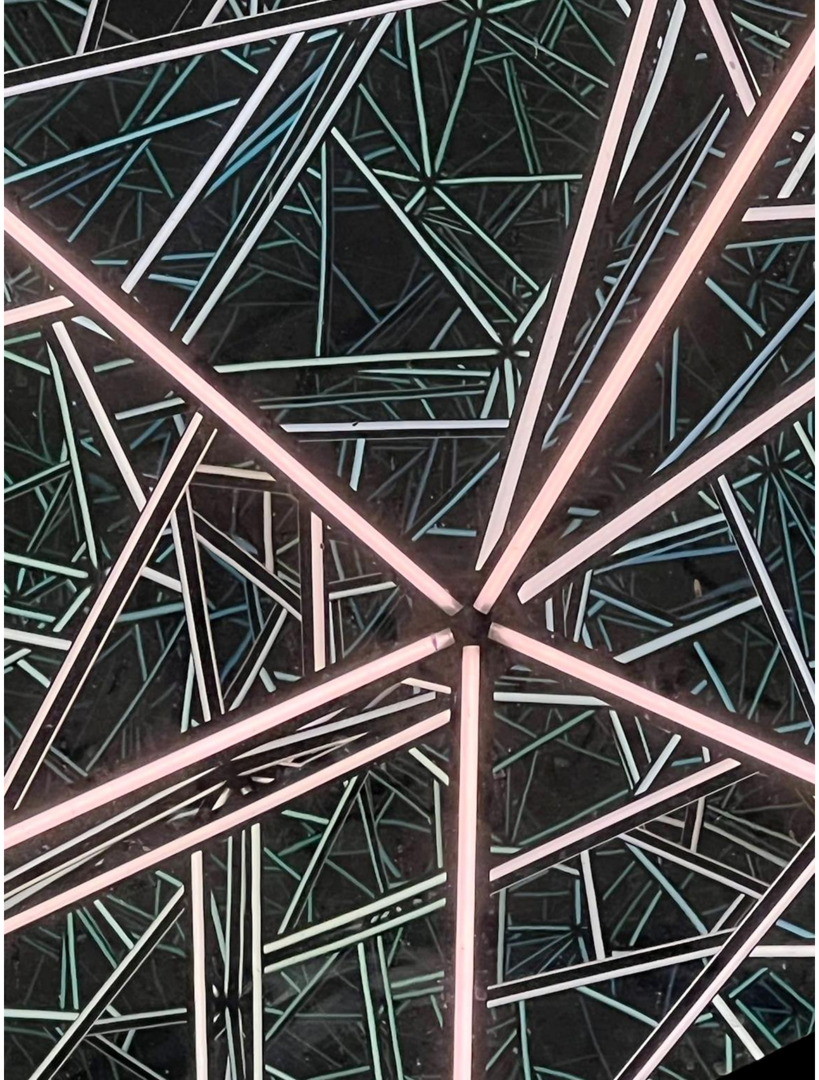
💎The REsearch year in numbers:
💥Total number of vulnerabilities reported - 228🔥
💥Affected silicon vendors - Intel, AMD, Qualcomm
💥Affected IBVs - Insyde, AMI
💥Affected device vendors - MS, HP, HPE, Dell, Lenovo, Intel, Fujitsu, Framework, Atos, Aruba, Cisco, Juniper ...

3:07 PM · Dec 28, 2022 · **33.9K** Views

https://binarly.io/advisories




vulnerabilities
lack of automation
complexity



Revisiting Automated Bug Hunting

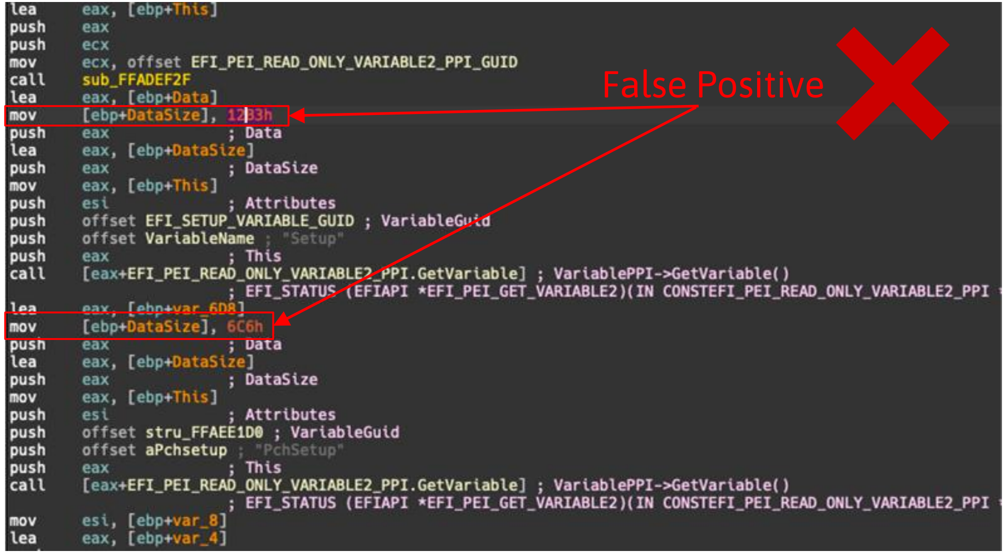
- **Progression of our past work:**
  "efiXplorer: Hunting for UEFI Firmware Vulnerabilities at Scale with Automated Static Analysis" [1]
- **Scalable approach based on vulnerability models; combination of:**
  1. **Lightweight static analysis**
  2. **Under-constrained symbolic execution**

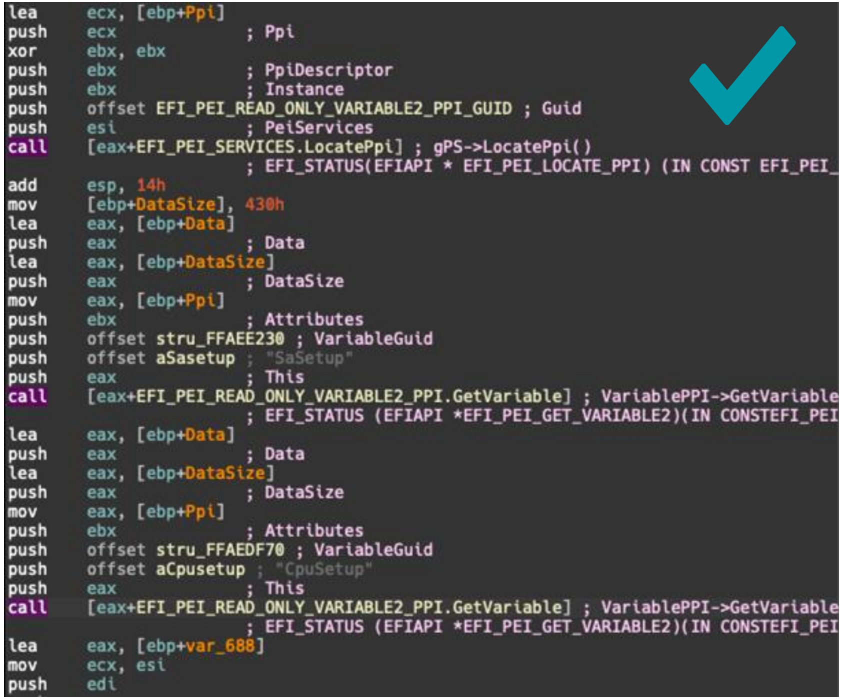
1: https://i.blackhat.com/eu-20/Wednesday/eu-20-Labunets-efiXplorer-Hunting-For-UEFI-Firmware-Vulnerabilities-At-Scale-With-Automated-Static-Analysis.pdf

Limitations of current approaches

<u>**With great scalability, comes a (great) potential for false positives!**</u>

Address	Type
00000000FFAE2BFD	pei_get_variable_buffer_overflow
00000000FFAE8894	pei_get_variable_buffer_overflow





Limitations of current approaches

Limitations of existing approaches:

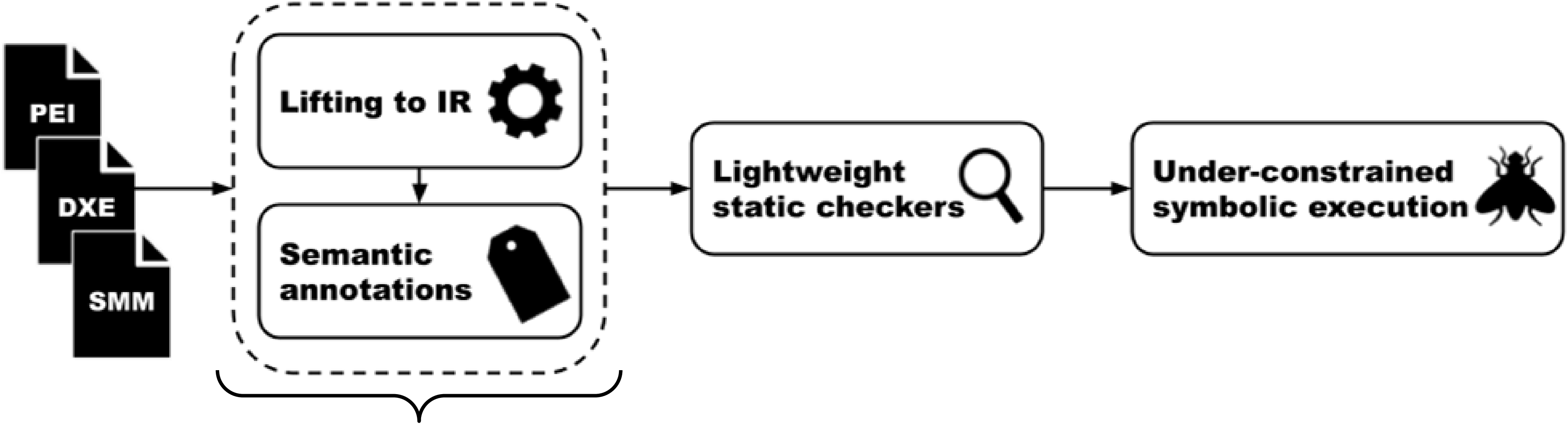
- Large number of false positives
- Mostly based on syntactic properties (pattern matching on disassembly)
- Highlighted in research by SentinelOne (Brick[2]):
  - Pattern matching on decompiler output
  - But: requires decompiler (Hex-Rays) & will not scale

Binarly team approach:

- Leverage semantic properties
- Use lightweight code pattern *checkers* to provide hints for deeper analysis

2: https://www.sentinelone.com/labs/another-brick-in-the-wall-uncovering-smm-vulnerabilities-in-hp-firmware/

Analysis pipeline



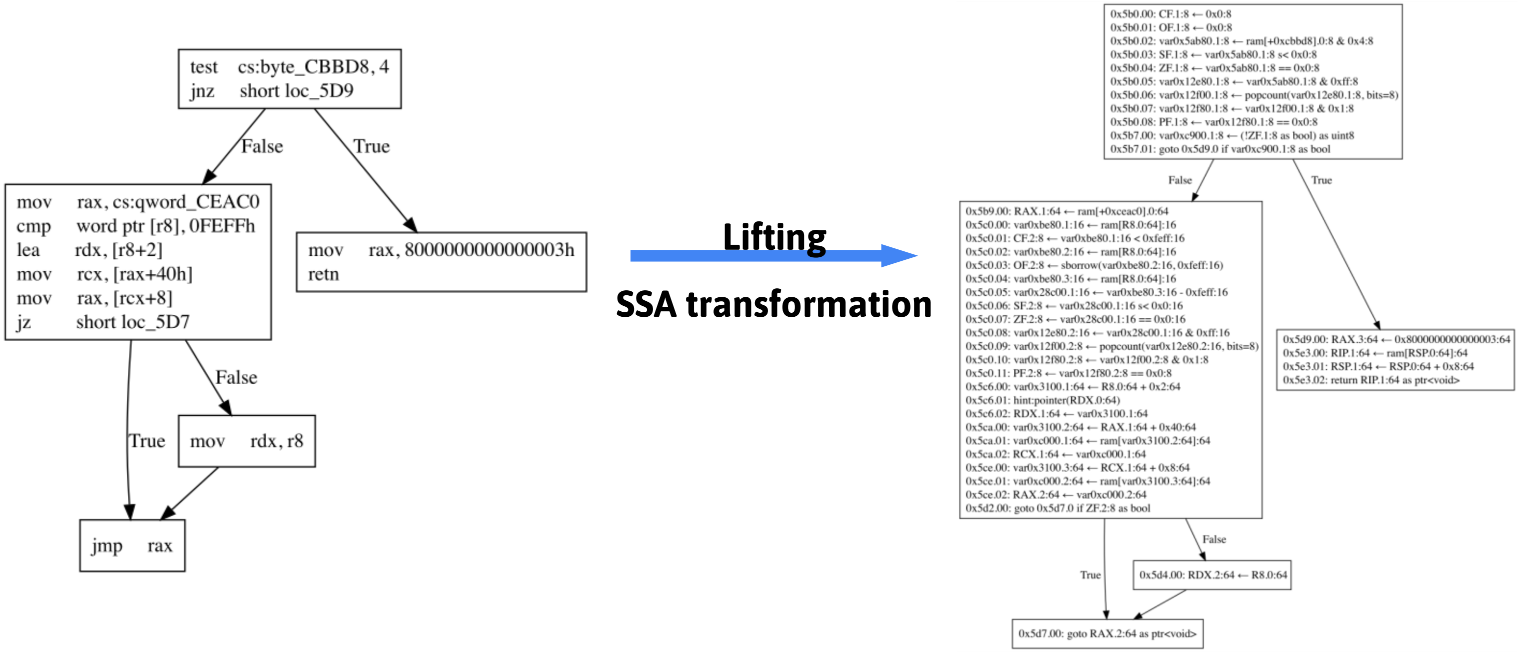
<u>Typically takes 4-6s per firmware image (100s of modules)</u>

Inspired by: "Sys: A Static/Symbolic Tool for Finding Good Bugs in Good (Browser) Code" (Brown et al., USENIX Security 2020)

```
(base)
┌─[sam@binarly]─
└─[~/Projects/binarly-symbolic]─ ./target/release/smiscan -v -d data ./SmmSmbiosElog-8e61fd6b-7a8b-404f-b83f-aa90a47cabdf.smm
```

IR lifting

- Extract uniform SSA from IR representation for 32-bit and 64-bit modules
- IR explicitly encodes instruction side-effects



Binarly Semantic annotations

```
binarly::efi::services] service call to InstallPpi: EFI_PEI_INSTALL_PPI
binarly::efi::services] resolved type: ptr<fn(PeiServices: ptr<PEFI_PEI_SERVICES>, PpiList: ptr<EFI_PEI_PPI_DESCRIPTOR>) -> EFI_STATUS>
binarly::efi::services] - PeiServices: ptr<PEFI_PEI_SERVICES> = 0xfadefada:32
binarly::efi::services] - PpiList[0]: struct<EFI_PEI_PPI_DESCRIPTOR>
binarly::efi::services]   - Flags: 0x10:32
binarly::efi::services]   - Guid:  EFI_PEI_RESET_PPI_GUID
binarly::efi::services]   - Ppi:   0xffac4a3c
binarly::efi::services] - PpiList[1]: struct<EFI_PEI_PPI_DESCRIPTOR>
binarly::efi::services]   - Flags: 0x80000010:32
binarly::efi::services]   - Guid:  AMI_PEI_SBINIT_POLICY_PPI_GUID
binarly::efi::services]   - Ppi:   0xffac4a38
```

- Annotate IR with types and service information (similar to efiXplorer[3] and FwHunt[4])
- Identify analysis entry-points based on module type, e.g.:
  - SMI handlers (DXE/SMM modules)
  - PEI notification callbacks (PEI modules)

3: https://github.com/binarly-io/efiXplorer
4: https://github.com/binarly-io/fwhunt-scan

Binarly Static checkers

- Checkers based on lightweight static analysis defined using an eDSL:

```
let mut matcher_builder = MatcherBuilder::new();

let s1 = matcher_builder.add_rule(ServiceCall::new(&project, "GetVariable"));
let s2 = matcher_builder.add_rule(ServiceCallChain::new(&project, "GetVariable"));

matcher_builder.add_transition(s1, s2)?;
matcher_builder.add_terminal(s2);
```

- Control-flow properties (reachability)
- Data-flow properties (data-dependence)
- Inferred call-site properties (e.g., arguments passed, type information)
- Domain-specific annotations:
    - Service-specific (e.g., GetVariable variants in PEI and DXE phases)
    - Common APIs (e.g., CopyMem, ZeroMem, etc.)

Under-constrained Symbolic Execution

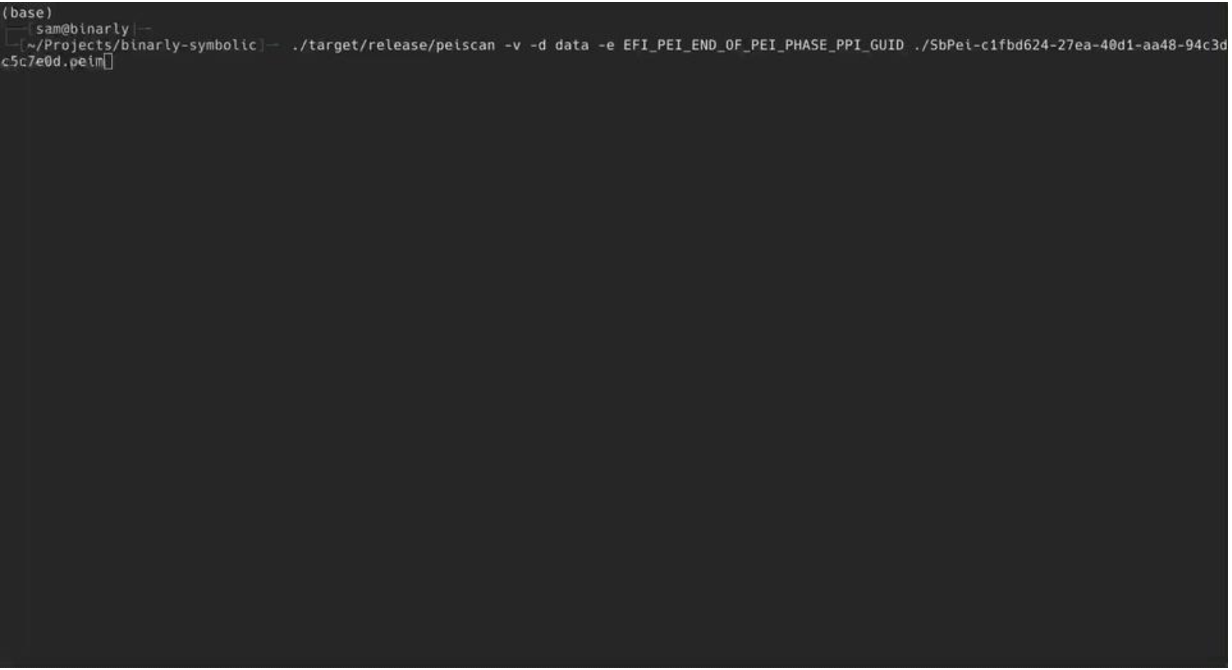
- Similar to past research:
  “Finding BIOS Vulnerabilities with Symbolic Execution and Virtual Platforms”[5]

Binarly team approach:

- Instrument anything (IR operation granularity)
- Simulate execution from anywhere
- Reason about hardware interactions and partial state using symbolic variables injected during simulation
- Identify violations of model assumptions (e.g., input to API should not be user-controlled)
- <u>No source-code required!</u>

5: https://www.intel.com/content/www/us/en/developer/articles/technical/finding-bios-vulnerabilities-with-symbolic-execution-and-virtual-platforms.html

PEI-phase vulnerabilities



(BRLY-2022-014/CVE-2022-32579)

GetVariable leading to arbitrary write

PEI-phase vulnerabilities

```
(base)
┌─[sam@binarly]─
└─[~/Projects/binarly-symbolic]─  ./target/release/peiscan -v -d data PlatformInitAdvancedPreMem-56bbc314-b442-4d5a-ba5c-d842dafdbb24.peim
```

(BRLY-2022-027/CVE-2022-28858)

GetVariable without DataSize check

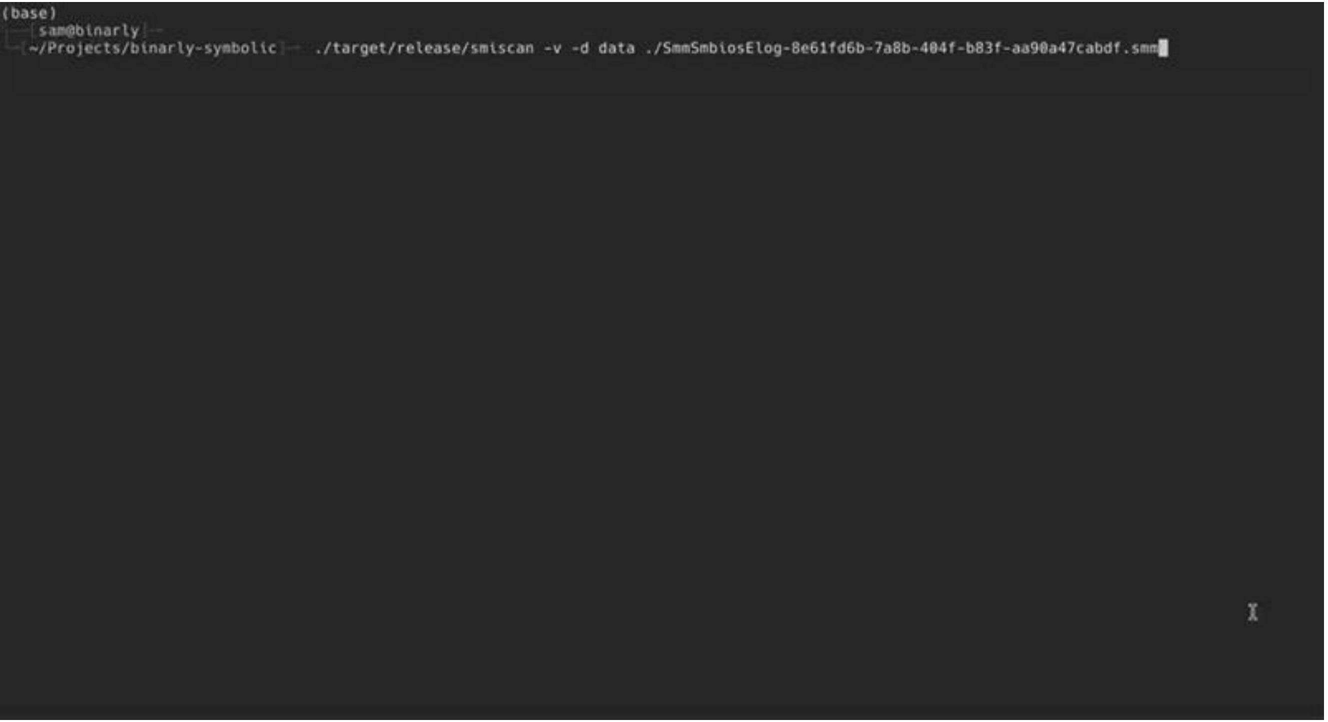
&

False Positive detection

```
binarly_checkers::types] setting label for rule 0 on entity 54 at 0xffae8894
binarly_checkers::types] setting label for rule 0 on entity 157 at 0xffae8871
binarly_checkers::types] setting label for rule 1 on entity 54 at 0xffae8894
binarly_checkers::types] setting label for rule 1 on entity 157 at 0xffae8871
binarly_checkers::types] stepping the searcher
binarly_checkers::types] no current checker
binarly_checkers::types] new checker has length 2
binarly_checkers::types] rule state 0 matches entity 54
binarly_checkers::types] rule state 0 accepts entity 54
binarly_checkers::types] continue with next transition
binarly_checkers::types] rule state 1 matches entity 54
binarly_checkers::types] rule state 1 does not accept transition to entity 54
binarly_checkers::types] rule state 1 matches entity 157
binarly_checkers::types] rule state 1 does not accept transition to entity 157
binarly_checkers::types] removing last transition set
binarly_checkers::types] rule state 0 matches entity 157
binarly_checkers::types] rule state 0 accepts entity 157
binarly_checkers::types] continue with next transition
binarly_checkers::types] rule state 1 matches entity 54
binarly_checkers::types] rule state 1 accepts entity 54
binarly_checkers::types] reached terminal for this path
```

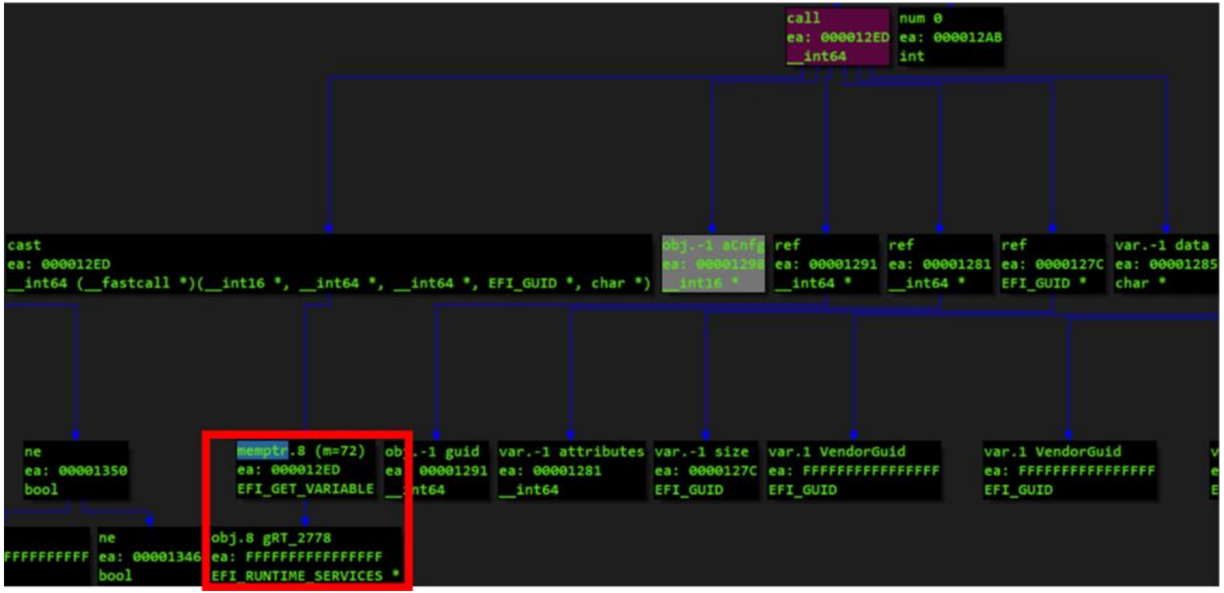
DXE/SMM vulnerabilities

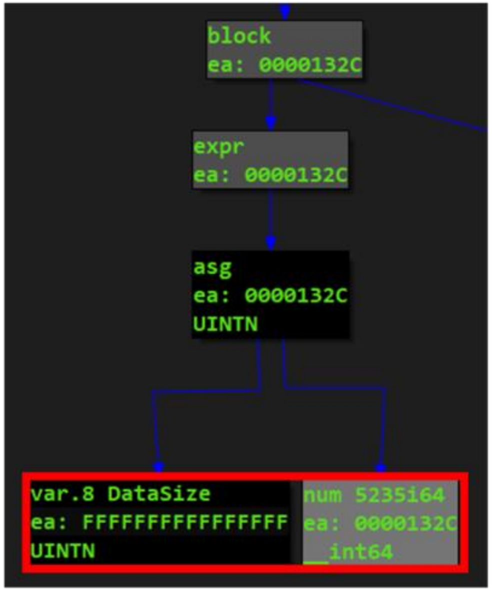
(BRLY-2022-016/CVE-2022-33209)

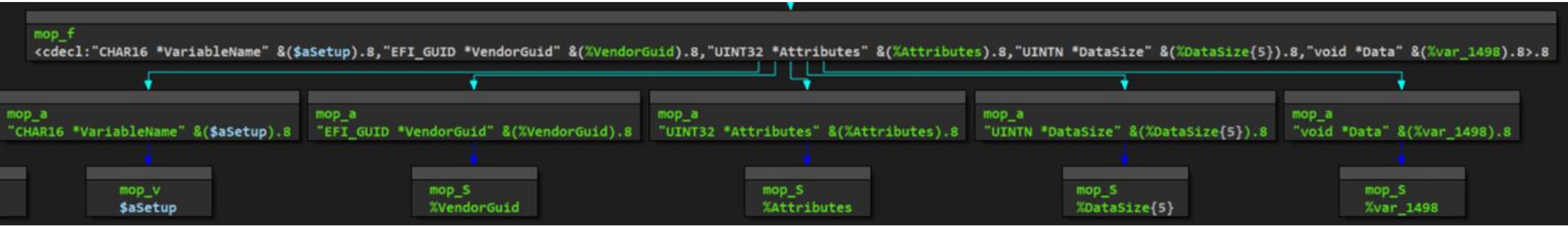


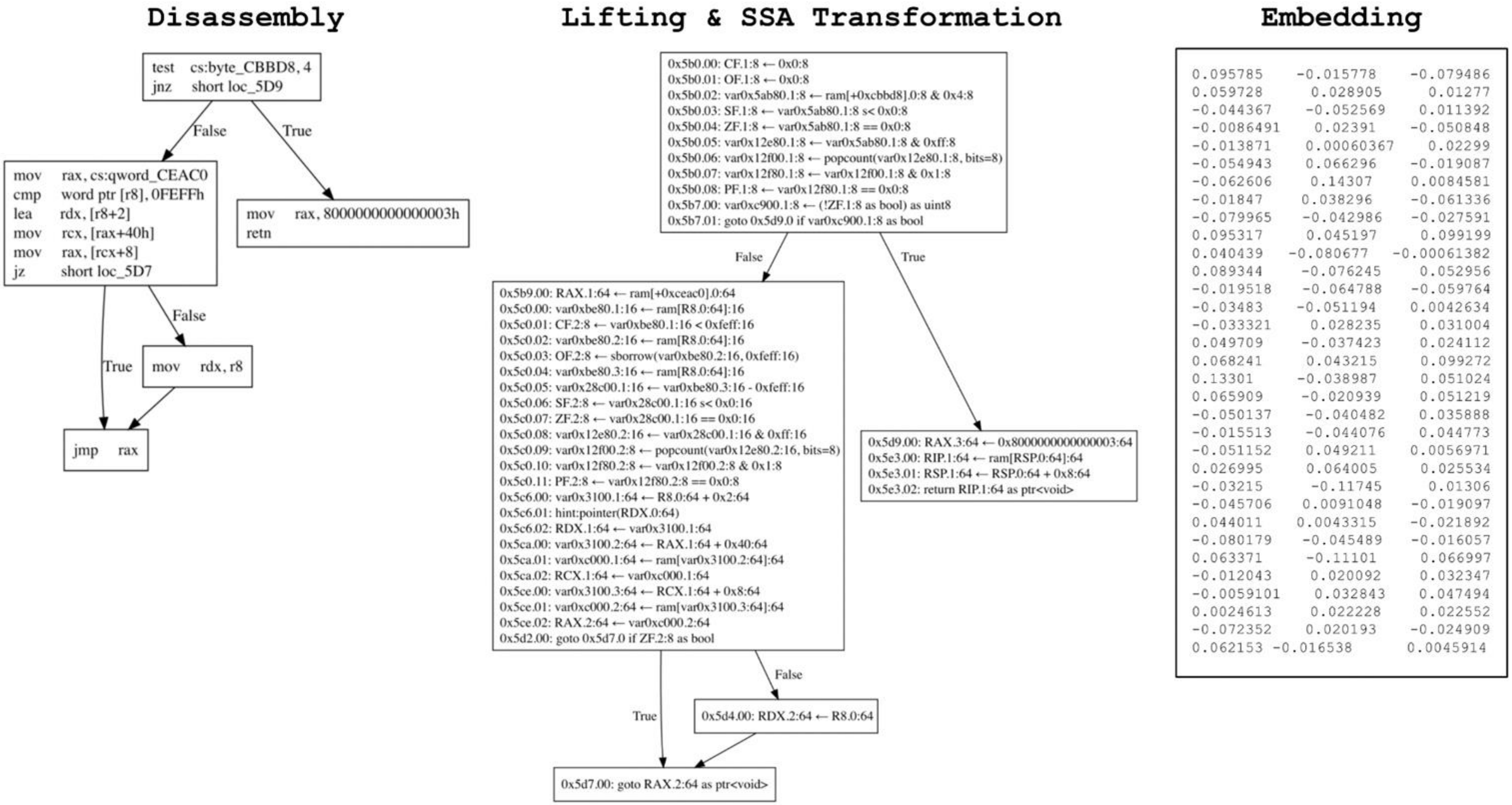
Buffer overflow discovery
&
CommBuffer reconstruction


(gRT_2778->GetVariable)(aCnfg, &guid, &attributes, &size, data)

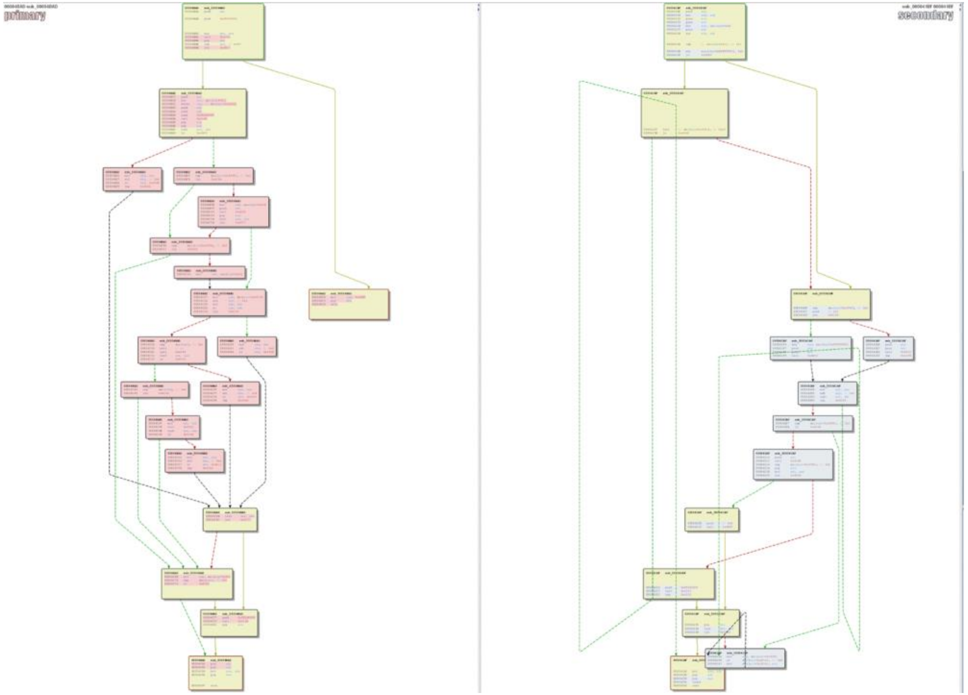

call
ea: 000012ED
__int64
num 0
ea: 000012AB
int
cast
ea: 000012ED
__int64 (__fastcall *)(__int16 *, __int64 *, __int64 *, EFI_GUID *, char *)
obj.-1 aCnfg
ea: 00001298
__int16 *
ref
ea: 00001291
__int64 *
ref
ea: 00001281
__int64 *
ref
ea: 0000127C
EFI_GUID *
var.-1 data
ea: 00001285
char *
ne
ea: 00001350
bool
memptr.8 (m=72)
ea: 000012ED
EFI_GET_VARIABLE
var.-1 attributes
ea: 00001281
__int64
var.-1 size
ea: 0000127C
EFI_GUID
var.1 VendorGuid
ea: FFFFFFFFFFFFFFFF
EFI_GUID
var.1 VendorGuid
ea: FFFFFFFFFFFFFFFF
EFI_GUID
ne
ea: 00001346
bool
obj.8 gRT_2778
ea: FFFFFFFFFFFFFFFF
EFI_RUNTIME_SERVICES *


block
ea: 0000132C
expr
ea: 0000132C
asg
ea: 0000132C
UINTN
var.8 DataSize
ea: FFFFFFFFFFFFFFFF
UINTN
num 5235i64
ea: 0000132C
__int64


mop_f
<cdecl:"CHAR16 *VariableName" &($aSetup).8,"EFI_GUID *VendorGuid" &(%VendorGuid).8,"UINT32 *Attributes" &(%Attributes).8,"UINTN *DataSize" &(%DataSize{5}).8,"void *Data" &(%var_1498).8>.8
mop_a
"CHAR16 *VariableName" &($aSetup).8
mop_a
"EFI_GUID *VendorGuid" &(%VendorGuid).8
mop_a
"UINT32 *Attributes" &(%Attributes).8
mop_a
"UINTN *DataSize" &(%DataSize{5}).8
mop_a
"void *Data" &(%var_1498).8
mop_v
$aSetup
mop_S
%VendorGuid
mop_S
%Attributes
mop_S
%DataSize{5}
mop_S
%var_1498


Disassembly
test cs:byte_CBBD8, 4
jnz short loc_5D9
False
True
mov rax, cs:qword_CEAC0
cmp word ptr [r8], 0FEFFh
lea rdx, [r8+2]
mov rcx, [rax+40h]
mov rax, [rcx+8]
jz short loc_5D7
mov rax, 8000000000000003h
retn
False
True
mov rdx, r8
jmp rax
Lifting & SSA Transformation
0x5b0.00: CF.1:8 ← 0x0:8
0x5b0.01: OF.1:8 ← 0x0:8
0x5b0.02: var0x5ab80.1:8 ← ram[+0xcbbd8].0:8 & 0x4:8
0x5b0.03: SF.1:8 ← var0x5ab80.1:8 s< 0x0:8
0x5b0.04: ZF.1:8 ← var0x5ab80.1:8 == 0x0:8
0x5b0.05: var0x12e80.1:8 ← var0x5ab80.1:8 & 0xff:8
0x5b0.06: var0x12f00.1:8 ← popcount(var0x12e80.1:8, bits=8)
0x5b0.07: var0x12f80.1:8 ← var0x12f00.1:8 & 0x1:8
0x5b0.08: PF.1:8 ← var0x12f80.1:8 == 0x0:8
0x5b7.00: var0xc900.1:8 ← (!ZF.1:8 as bool) as uint8
0x5b7.01: goto 0x5d9.0 if var0xc900.1:8 as bool
False
True
0x5b9.00: RAX.1:64 ← ram[+0xceac0].0:64
0x5c0.00: var0xbe80.1:16 ← ram[R8.0:64]:16
0x5c0.01: CF.2:8 ← var0xbe80.1:16 < 0xfeff:16
0x5c0.02: var0xbe80.2:16 ← ram[R8.0:64]:16
0x5c0.03: OF.2:8 ← sborrow(var0xbe80.2:16, 0xfeff:16)
0x5c0.04: var0xbe80.3:16 ← ram[R8.0:64]:16
0x5c0.05: var0x28c00.1:16 ← var0xbe80.3:16 - 0xfeff:16
0x5c0.06: SF.2:8 ← var0x28c00.1:16 s< 0x0:16
0x5c0.07: ZF.2:8 ← var0x28c00.1:16 == 0x0:16
0x5c0.08: var0x12e80.2:16 ← var0x28c00.1:16 & 0xff:16
0x5c0.09: var0x12f00.2:8 ← popcount(var0x12e80.2:16, bits=8)
0x5c0.10: var0x12f80.2:8 ← var0x12f00.2:8 & 0x1:8
0x5c0.11: PF.2:8 ← var0x12f80.2:8 == 0x0:8
0x5c6.00: var0x3100.1:64 ← R8.0:64 + 0x2:64
0x5c6.01: hint:pointer(RDX.0:64)
0x5c6.02: RDX.1:64 ← var0x3100.1:64
0x5ca.00: var0x3100.2:64 ← RAX.1:64 + 0x40:64
0x5ca.01: var0xc000.1:64 ← ram[var0x3100.2:64]:64
0x5ca.02: RCX.1:64 ← var0xc000.1:64
0x5ce.00: var0x3100.3:64 ← RCX.1:64 + 0x8:64
0x5ce.01: var0xc000.2:64 ← ram[var0x3100.3:64]:64
0x5ce.02: RAX.2:64 ← var0xc000.2:64
0x5d2.00: goto 0x5d7.0 if ZF.2:8 as bool
0x5d9.00: RAX.3:64 ← 0x8000000000000003:64
0x5e3.00: RIP.1:64 ← ram[RSP.0:64]:64
0x5e3.01: RSP.1:64 ← RSP.0:64 + 0x8:64
0x5e3.02: return RIP.1:64 as ptr<void>
True
False
0x5d4.00: RDX.2:64 ← R8.0:64
0x5d7.00: goto RAX.2:64 as ptr<void>
Embedding
0.095785 -0.015778 -0.079486
0.059728 0.028905 0.01277
-0.044367 -0.052569 0.011392
-0.0086491 0.02391 -0.050848
-0.013871 0.00060367 0.02299
-0.054943 0.066296 -0.019087
-0.062606 0.14307 0.0084581
-0.01847 0.038296 -0.061336
-0.079965 -0.042986 -0.027591
0.095317 0.045197 0.099199
0.040439 -0.080677 -0.00061382
0.089344 -0.076245 0.052956
-0.019518 -0.064788 -0.059764
-0.03483 -0.051194 0.0042634
-0.033321 0.028235 0.031004
0.049709 -0.037423 0.024112
0.068241 0.043215 0.099272
0.13301 -0.038987 0.051024
0.065909 -0.020939 0.051219
-0.050137 -0.040482 0.035888
-0.015513 -0.044076 0.044773
-0.051152 0.049211 0.0056971
0.026995 0.064005 0.025534
-0.03215 -0.11745 0.01306
-0.045706 0.0091048 -0.019097
0.044011 0.0043315 -0.021892
-0.080179 -0.045489 -0.016057
0.063371 -0.11101 0.066997
-0.012043 0.020092 0.032347
-0.0059101 0.032843 0.047494
0.0024613 0.022228 0.022552
-0.072352 0.020193 -0.024909
0.062153 -0.016538 0.0045914

Binary Diffing == BinDiff



"Graph-based comparison of Executable Objects" - 2005, SSTIC
https://actes.sstic.org/SSTIC05/Analyse_differentielle_de_binaires/SSTIC05-article-Flake-Graph_based_comparison_of_Executable_Objects.pdf

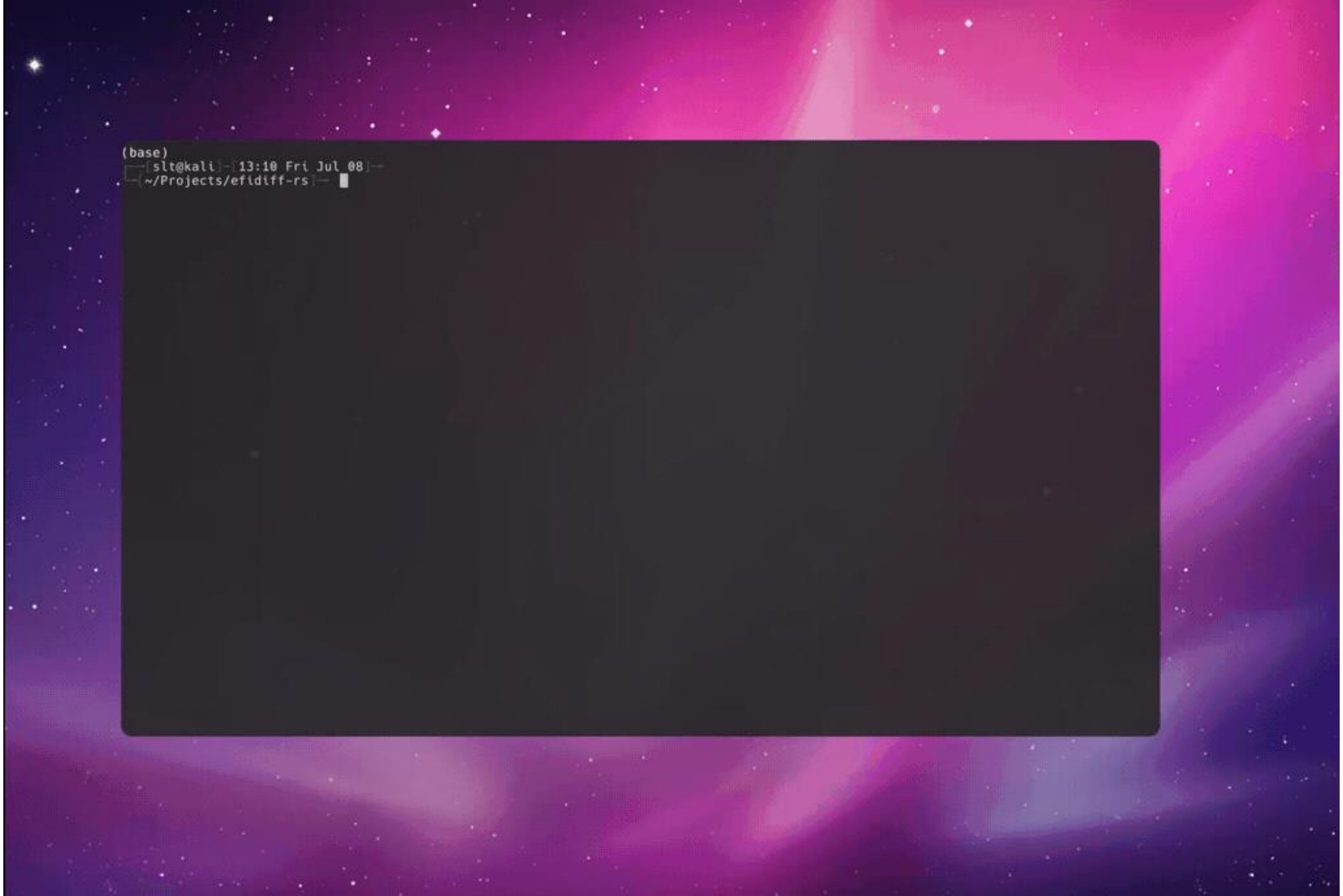
Binary Diffing == BinDiff

Matched Functions

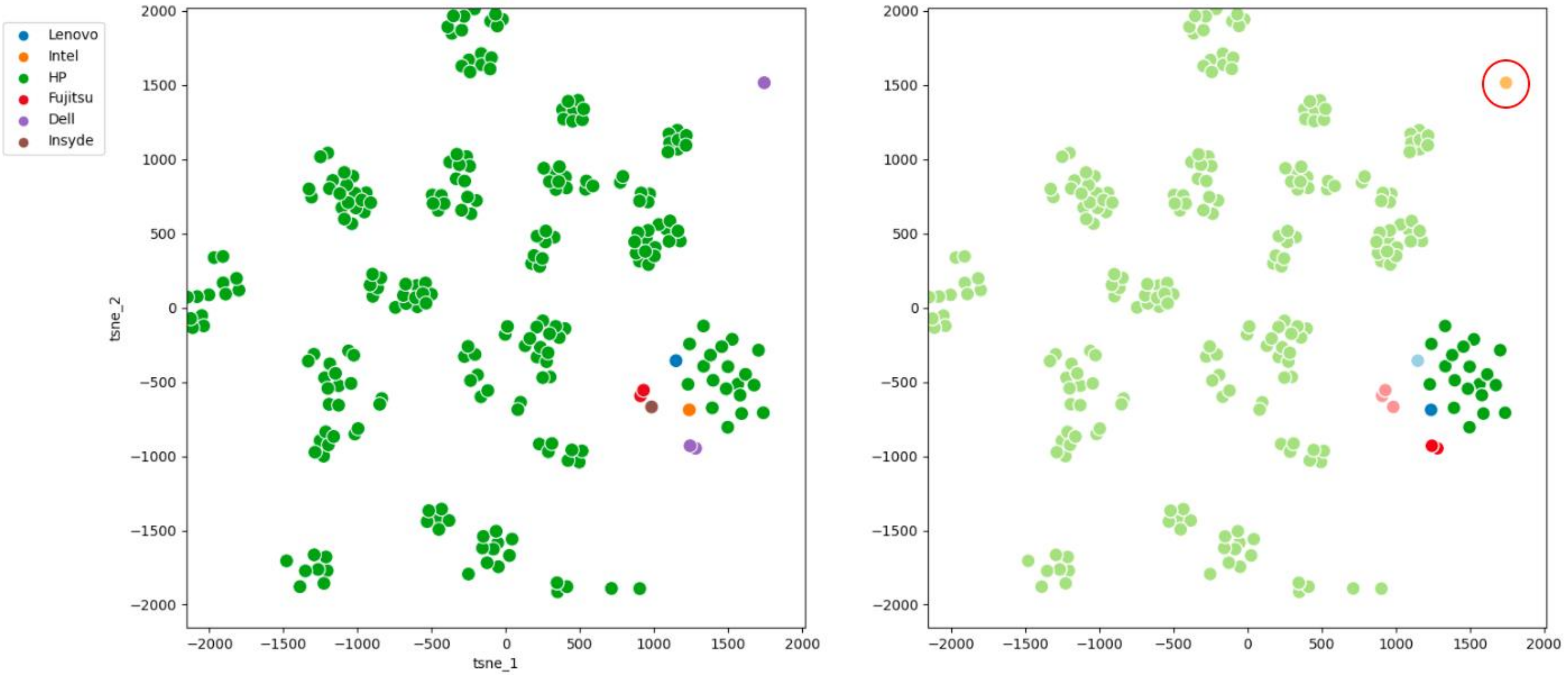
Similarity	Confid	Change	EA Primary	Name Primary	EA Secondary	Name Secondary
0.95	0.99	GI--…	000000018001…	CoreCreateEventInternal	000000018001F10C	sub_000000018001F10C
0.99	0.99	-I--…	000000018000…	CoreExitBootServices	0000000180009F4C	sub_0000000180009F4C
0.99	0.99	-I--…	000000018002…	InternalAllocatePool	00000001800233B0	sub_00000001800233B0
1.00	0.99	----…	000000018000…	sub_18000030C	000000018000030C	sub_000000018000030C
1.00	0.99	----…	000000018000…	sub_180000358	0000000180000358	sub_0000000180000358
1.00	0.99	----…	000000018000…	sub_180000420	0000000180000420	sub_0000000180000420
1.00	0.99	----…	000000018000…	sub_18000045C	000000018000045C	sub_000000018000045C
1.00	0.99	----…	000000018000…	sub_1800004EC	00000001800004EC	sub_00000001800004EC
1.00	0.99	----…	000000018000…	sub_1800005D0	00000001800005D0	sub_00000001800005D0
1.00	0.99	----…	000000018000…	sub_180000624	0000000180000624	sub_0000000180000624
1.00	0.99	----…	000000018000…	sub_1800006D8	00000001800006D8	sub_00000001800006D8
1.00	0.99	----…	000000018000…	sub_180000730	0000000180000730	sub_0000000180000730
1.00	0.99	----…	000000018000…	sub_1800007A4	00000001800007A4	sub_00000001800007A4
1.00	0.99	----…	000000018000…	sub_1800007E4	00000001800007E4	sub_00000001800007E4
1.00	0.99	----…	000000018000…	sub_1800007FC	00000001800007FC	sub_00000001800007FC
1.00	0.99	----…	000000018000…	sub_180000910	0000000180000910	sub_0000000180000910
1.00	0.99	----…	000000018000…	sub_1800009E0	00000001800009E0	sub_00000001800009E0
1.00	0.99	----…	000000018000…	sub_180000A4C	0000000180000A4C	sub_0000000180000A4C
1.00	0.99	----…	000000018000…	sub_180000AD4	0000000180000AD4	sub_0000000180000AD4

https://binarly.io/posts/A_deeper_UEFI_dive_into_MoonBounce

```
{
  "name": "FunctionSimilarity",
  "meta": {
    "description": "Check how similar the module's functions are to the same module (by GUID) in a previous firmware version.",
    "extra_info": {
      "modules": [
        {
          "guid": "d6a2cb7f-6a18-4e2f-b43b-9920a733700a",
          "hash": "9df301ebb3d4035ff173a0f66c17f1fa8c01241b7a472b9fce5927b1019c9eed",
          "name": "DxeCore",
          "similarity": "Very dissimilar (less than 20% similarity)"
        }
      ]
    },
    "severity": 2
  },
  "status": 1
},
{
  "name": "AddedModuleVariables",
  "meta": {
    "description": "Check if this module references any variables that were not referenced by the same module (by GUID) in a previous firmware version.",
    "extra_info": {
      "modules": [
        {
          "guid": "d6a2cb7f-6a18-4e2f-b43b-9920a733700a",
          "hash": "9df301ebb3d4035ff173a0f66c17f1fa8c01241b7a472b9fce5927b1019c9eed",
          "name": "DxeCore",
          "new_variables": [
            "AmiCapUp",
            "BbsPopupCalled",
            "BootCurrent",
            "BootFlow",
            "BootNext",
            "BootOrder",
            "DefaultBootOrder",
            "DefaultLegacyDevOrder",
            "DefaultUefiDevOrder",
            "ErrOut",
            "ErrOutDev",
            "FBODual_priority",
            "FBOUefi_priority",
            "FPDT_Variable",
            "FastBootOption",
            "IsaDmaMask",
            "IsaIrqMask",
            "LastBoot",
            "LastBootFailed",
            "LegacyDevOrder",
            "MemoryTypeInformation",
            "OldBootOrder",
            "OldLegacyDevOrder",
            "OldUefiDevOrder",
            "PreviousMemoryTypeInformation",
            "RSCInfoAddresss",
            "SIO_DEV_STATUS_VAR",
            "Setup",
            "UefiDevOrder"
          ]
        }
      ]
    },
```


(base)
[slt@kali]-[13:10 Fri Jul 08]
[~/Projects/efidiff-rs]

T-distributed Stochastic Neighbor Embedding (TSNE)



Function to Find SEARCH

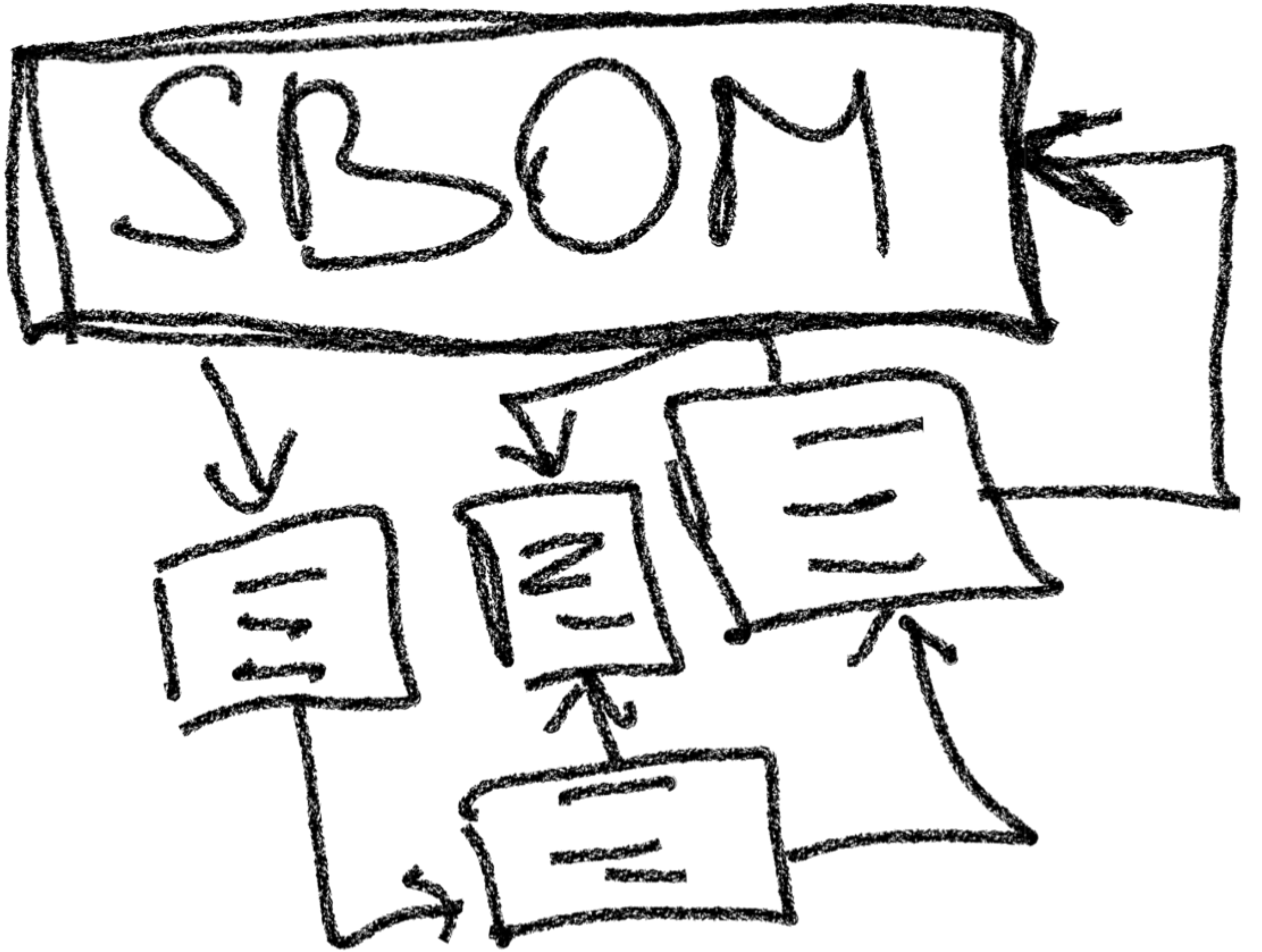
64e0d850110de977c78f56e7ae99154c_Q85_011500.bin_0D24

A: X86:LE:64; F: 11; B: 171

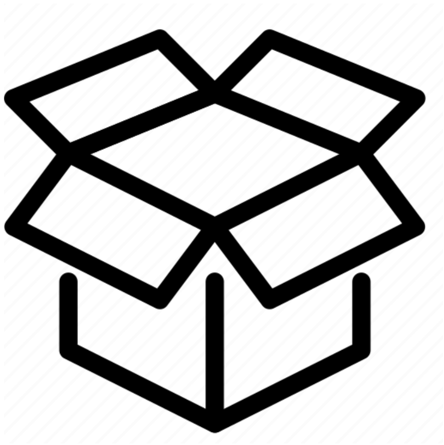
METADATA [-]

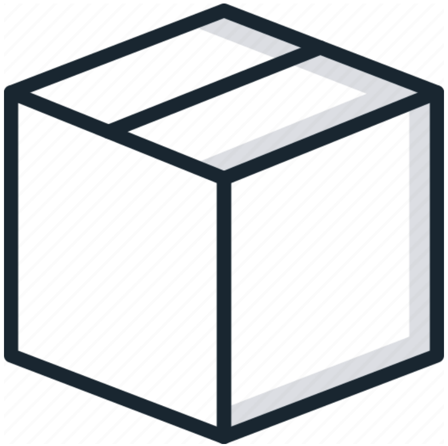
0x12a8

Property	Value
location	0x12a8
operations	0x12dd.1 RCX.0:64 ← 0x2050:64 0x12e4.0 CF.0:8 ← 0x0:8 0x12e4.1 OF.0:8 ← 0x0:8 0x12e4.2 EDX.0:32 ← 0x0:32 0x12e4.3 RDX.0:64 ← EDX.0:32 as uint64 0x12e4.4 SF.0:8 ← EDX.0:32 s< 0x0:32 0x12e4.5 ZF.0:8 ← EDX.0:32 == 0x0:32 0x12e4.6 var0x12e80.0:32 ← EDX.0:32 & 0xff:32 0x12e4.7 var0x12f00.0:8 ← popcount(var0x12e80.0:32, bits= 0x12e4.8 var0x12f80.0:8 ← var0x12f00.0:8 & 0x1:8 0x12e4.9 PF.0:8 ← var0x12f80.0:8 == 0x0:8 0x12e6.0 var0x3200.0:64 ← RAX.0:64 + 0x140:64 0x12e6.1 var0xc000.0:64 ← ram[var0x3200.0:64]:64 0x12e6.2 var0x27700.0:64 ← var0xc000.0:64 0x12e6.3 RSP.0:64 ← RSP.0:64 - 0x8:64 0x12e6.4 ram[RSP.0:64]:64 ← 0x12ec:64 0x12e6.5 call var0x27700.0:64
types	SF.0:8 = 0x0:8 / bool RDX.0:64 = 0x0:64 / ptr<void> RSP.0:64 = sp-0x78:64 / uint64 **Stack** sp-80 = 0x12ec:64 / uint64 sp-28 = ? / uint64 sp-20 = ? / uint64 sp-18 = ? / uint64 sp-10 = ? / uint64 sp-8 = ? / uint64 sp+10 = 0x0:64 / ptr<void> sp+18 = 0xff:64 / uint64 sp+20 = T / ptr<void> **Globals** 0x2050 = T / EFI_GUID **Branch/call target** *(*0x23a0:64+0x140:64)+0x0:64 / EFI_LOCATE_PROTOCOL

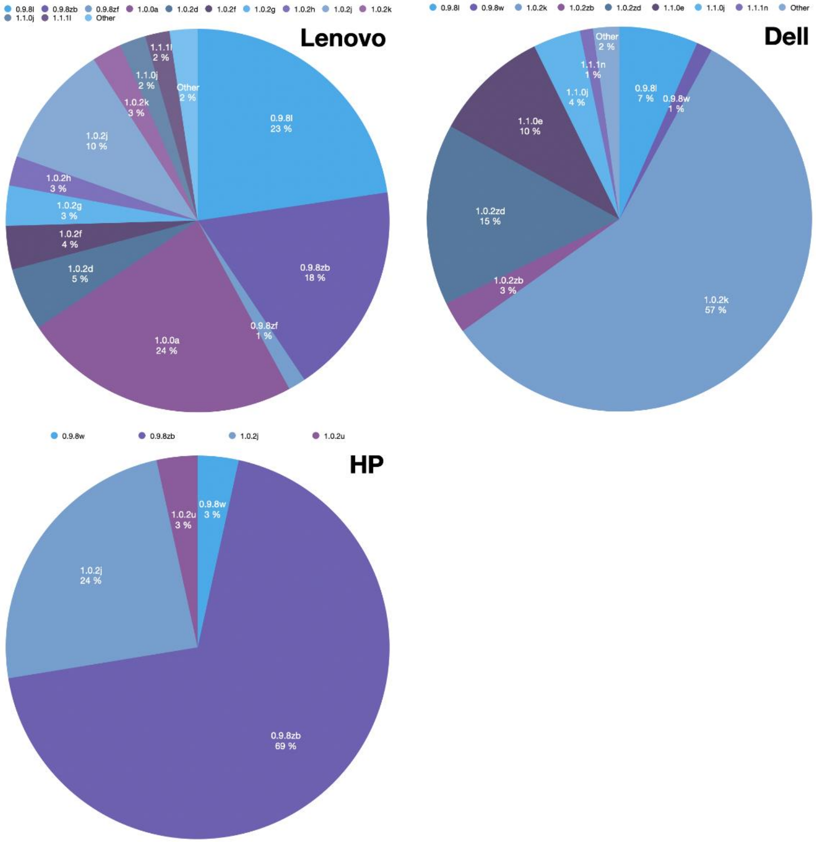

SBOM

SBOM => Open/Closed Source Challenges





SBOM == Policy != Technology



https://binarly.io/posts/OpenSSL_Usage_in_UEFI_Firmware_Exposes_Weakness_in_SBOMs

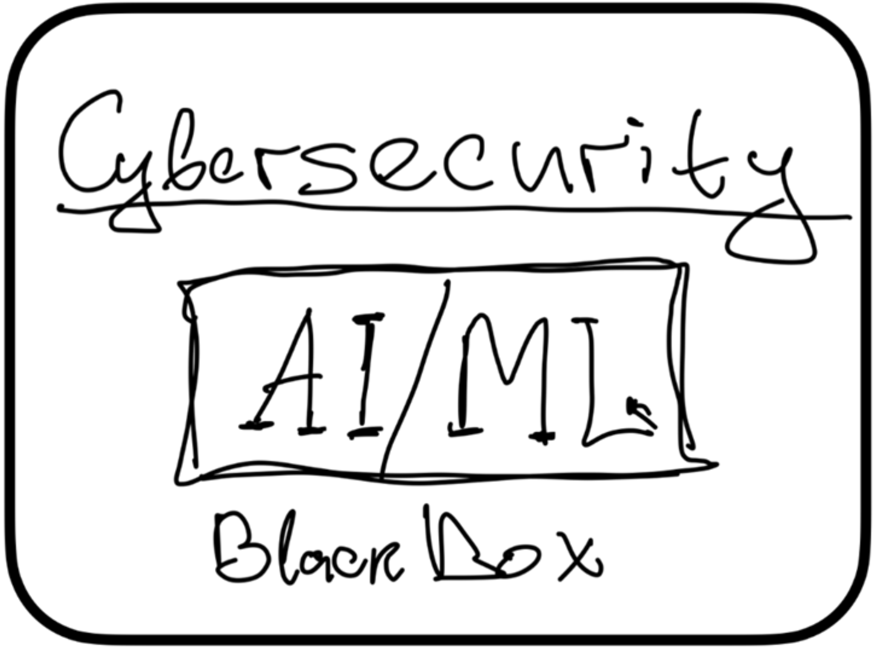
Next directions for REsearch

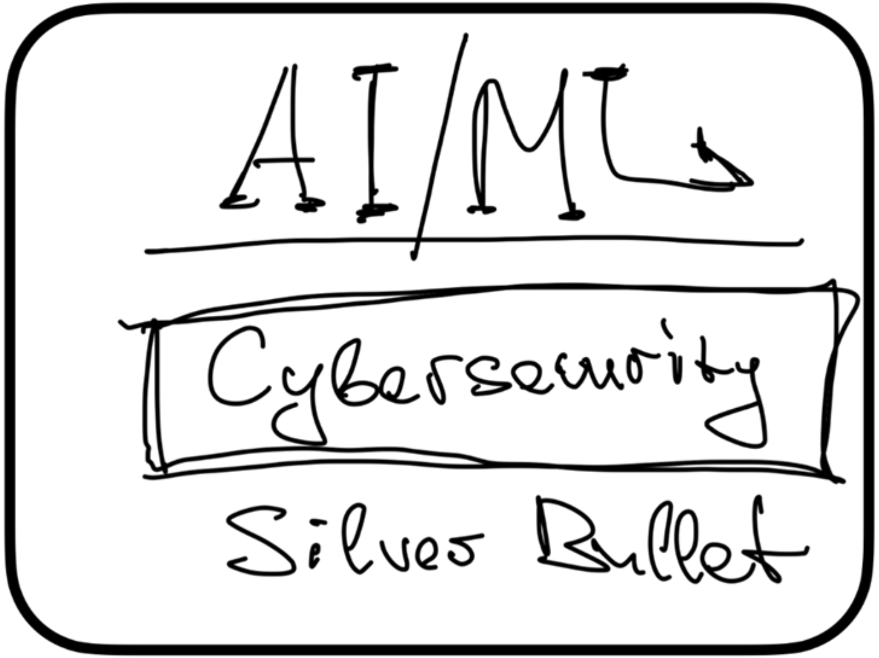
- Detecting known vulnerabilities is different from finding known unknowns. When automating vulnerability research, it is extremely important to scope the search area correctly.

- We find more problems than we can automatically explain and triage. Automating the process of explaining exploitability of the findings is one of the most important challenges facing the industry.

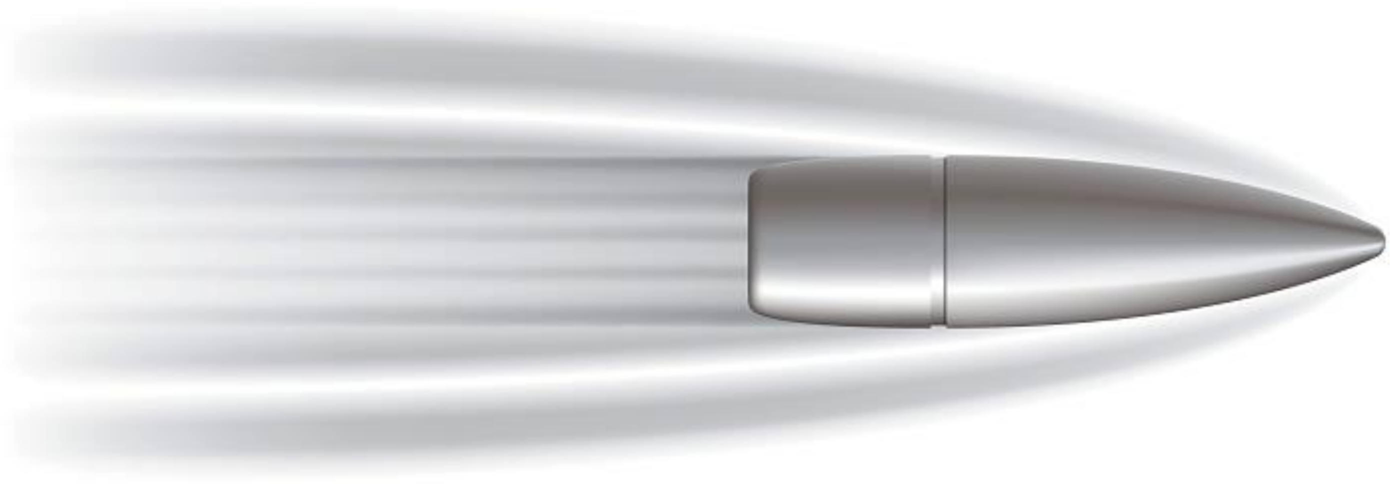
- ML models guided by code semantics can automate the search for well documented security problems.

The new old challenges of machine learning





AI/ML doesn't solve all problems magically



Thank You!