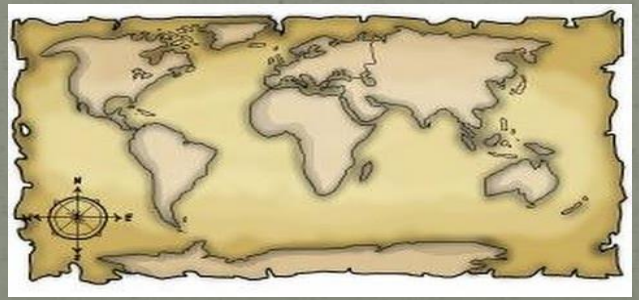
Check Point
SOFTWARE TECHNOLOGIES LTD.

cp<r>
CHECK POINT RESEARCH

Karta

Source code assisted Geographic-based binary matching



@EyalItkin

https://github.com/CheckPointSW/Karta

Who Am I

- **Eyal Itkin**
- Vulnerability Researcher
- cp<r> Check Point Research
- Focus embedded devices & network protocols
- @EyalItkin

What The FAX?!

- In 2018 we presented our research on the FAX protocol
  - https://research.checkpoint.com/sending-fax-back-to-the-dark-ages/
- Used HP OfficeJet firmware as our test case
- Our goal was to find a RCE over the protocol
- Demonstrated a RCE over the telephone line ☺

What The FAX?!

- In 2018 we
  - https://
-dark-ages/
- Used HP C
- Our goal w
- Demonstra

Motivation

- Identifying known (open source) code in binaries
  - Helping researchers in the RE process
  - Locating 1-Days in a given firmware
  - Automatically identifying used libc functions
  - Tired of reverse engineering Net-snmp again and again

Motivation

- Dealing with Large (> 50,000 functions) binaries
  - HP OfficeJet firmware: ~65,000 functions
  - Cisco router firmware: ~200,000 functions
  - Even TeamViewer is huge: ~143,000 functions

Motivation

- The complexity of most tools depends on N >> K
  - N – Firmware Size (> 50,000 functions)
  - K – Library size (50 – 3000 functions)
- Meaning that current diffing tools fail on large binaries:
  - BinDiff (Zynamics)
  - Diaphora (Joxean Koret - @matalaz)

Sneak Peek - FAX

- During our FAX research we needed a debugger
- When looking for a useful 1-Day in the firmware:
  - Identify the used open sources
  - Search for the vulnerable functions
- Eventually we exploited “Devil’s Ivy” from gSOAP

Sneak Peek - FAX

- Karta's identifier output:

```
Identified Open Sources:
------------------------
libpng: 1.2.29
zlib: 1.2.3
OpenSSL: 1.0.1j
gSOAP: 2.7
mDNSResponder: unknown
```

- A short Google search:

Vulnerability Details : CVE-2017-9765

Integer overflow in the soap_get function in Genivia gSOAP 2.7.x and 2.8.x before 2.8.48, as used on Axis cam
denial of service (stack-based buffer overflow and application crash) via a large XML document, aka Devil's Ivy.
configurations on general-purpose computers.

- Would Karta match the vulnerable function?

Sneak Peek - FAX

- Karta's ide
- A short G
- Would Ka

```
int __fastcall gSOAP_soap_get_pi(int a1)
{
  int v1; // r5
  int *v2; // r6
  int v3; // r7
  int v4; // r4
  unsigned int v5; // r0
  int v6; // r6
  int v8; // [sp+0h] [bp-54h]

  v1 = a1;
  v2 = &v8;
  v3 = 64;
  while ( 1 )
  {
    v5 = gSOAP_soap_getchar(v1);
    v4 = v5;
    if ( v5 == -1 || v5 == '?' )
      break;
    if ( v3 > 1 )
    {
      if ( v5 < 0x21 )
        LOBYTE(v4) = ' ';
      *(_BYTE *)v2 = v4;
      v2 = (int *)((char *)v2 + 1);
      --v3;
    }
  }
```

en Sources:

9
1j
: unknown

Vulnerabi
Integer ove
denial of se
configurati

sed on Axis cam
aka Devil's Ivy.

Sneak Peek - TeamViewer

- Project Zero had a series of blog post on fuzzing WebRTC
  - https://googleprojectzero.blogspot.com/2018/12/adventures-in-video-conferencing-part-1.html
- One of the vulnerabilities was CVE-2018-6155
  - Use-After-Free in libvpx – a popular open source
- "… it has the potential to affect software … other than WebRTC."

Sneak Peek - TeamViewer

- Karta's identifier output:

```
Identified Open Sources:
------------------------
zlib: 1.2.5
mDNSResponder: unknown
libjpeg: 8b
libvpx: 1.6.1
```

- A look on the vulnerable function:

```
void vp8_deblock(VP8_COMMON *cm, YV12_BUFFER_CONFIG *source,
                 YV12_BUFFER_CONFIG *post, int q, int low_var_thresh,
                 int flag) {
  double level = 6.0e-05 * q * q * q - .0067 * q * q + .306 * q + .0065;
  int ppl = (int)(level + .5);

  // EI [CVE]: This is the code line responsible for CVE-2018-6155
  const MODE_INFO *mode_info_context = cm->show_frame_mi;
  int mbr, mbc;
```

- Would Karta match the vulnerable function?

Sneak Peek - TeamViewer

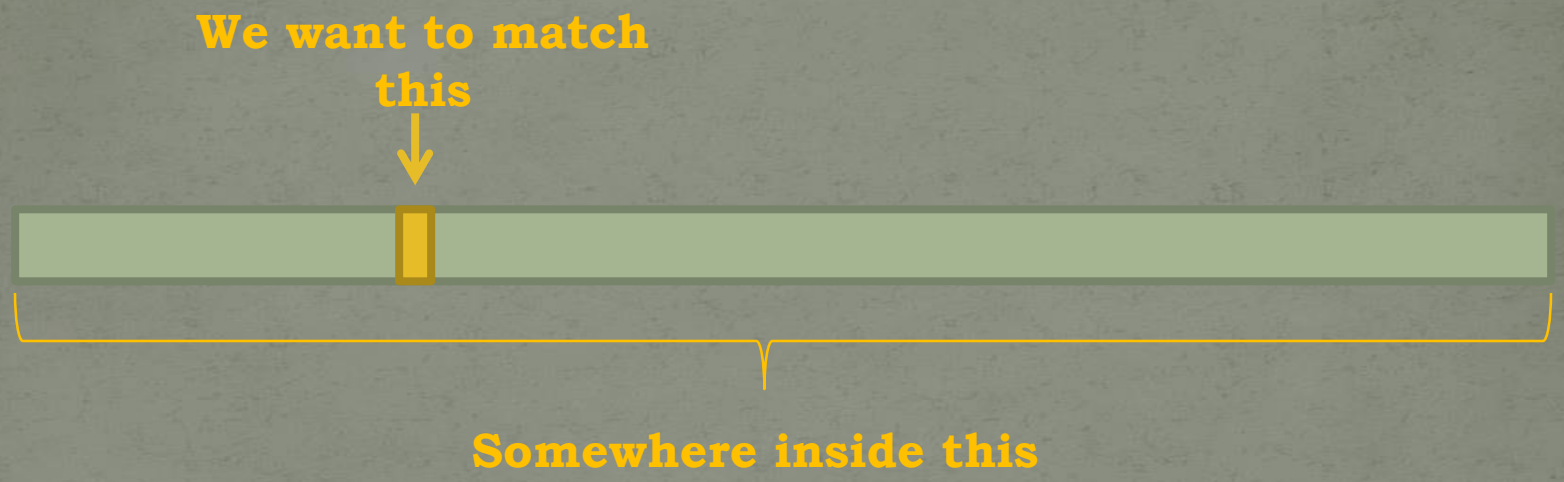
- Karta'
- A lool
- Would Karta match the vuln

```
void vp8_debloc

  double level
  int ppl = (in

  // EI [CVE]:
  const MODE_IN
  int mbr, mbc;
```

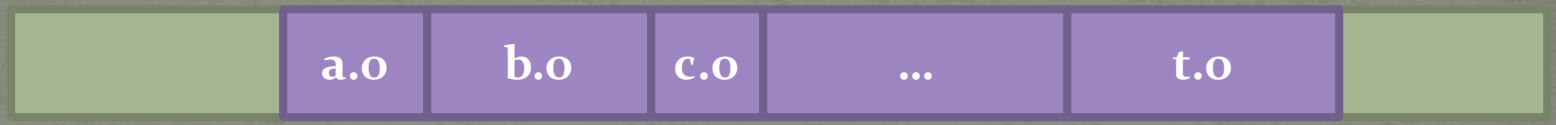

```
.text:0230DCF0                  libvpx_vp8_deblock proc near
.text:0230DCF0 55               push    ebp
                                v       ebp, esp
                                b       esp, 14h
                                ld      dword ptr [ebp+14h]
                                sh      ebx
                                sh      edi
                                tp      qword ptr [ebp-14h]
                                d       qword ptr [ebp-14h]
                                d       ds:dbl_2831A38  ; 6.0e-05
                                ul      st, st(1)
                                ul      st, st(1)
                                ul      st, st(1)
                                d       st(1)
                                ul      ds:dbl_2831A48  ; 0.0067
                                ul      st, st(2)
                                ubp     st(1), st
                                ch      st(1)
                                ul      ds:dbl_2831A50  ; 0.306
                                ddp     st(1), st
                                dd      ds:dbl_2831A40  ; 0.065
                                dd      ds:dbl_2A96DD8  ; 0.5
                                ll      sub_244D700
                                v       ecx, [ebp+8]
                                v       [ebp-8], eax
                                v       edi, [ecx+0A40h]
                                v       edx, [ecx+0A24h]
                                v       ebx, [ecx+0A80h]
.text:0230DD4C C1 E7 04         shl     edi, 4
.text:0230DD4F 03 FA            add     edi, edx
.text:0230DD51 89 55 F4         mov     [ebp-0Ch], edx
.text:0230DD54 89 7D F0         mov     [ebp-10h], edi
.text:0230DD57 85 C0            test    eax, eax
.text:0230DD59 0F 8E 49 01 00 00 jle    loc_230DEA8
```

Mapping the Binary

Mapping the binary



Mapping the binary



Mapping the binary

A compiled library is merely a collection of

compiled files,

attached to one another by a linker

- Instead of matching **functions**, match **files**

Karta

- Karta := Russian for “Map”

https://github.com/CheckPointSW/Karta

- Source code assisted binary matching
- Geographic-based matching
  - Locates the library inside the binary blob
  - Generates a map of the files inside the binary
- Architecture agnostic (works on canonical representation)

Fingerprinting

- Base Phase – Use an identifier
  - Can we find the library?
  - What version is it?
  - Load the matching (pre-compiled) config for it
- Currently relies on basic string search

```
"Copyright (c) 1995-1996 Guy Eric Schalnat, Group 42, Inc."
```

- Could (should) be improved in the future

```
aAsn1_lib_c_1    DCB "asn1_lib.c",0       ; DATA XREF: sub_203F65E4+6↑o
                 DCB    0
aAsn_1PartOfOpenssl1_0_1j15Oct201 DCB "ASN.1 part of OpenSSL 1.0.1j 15 Oct 2014",0
```

Zooming-In

- Phase I – Locate “Anchor” functions

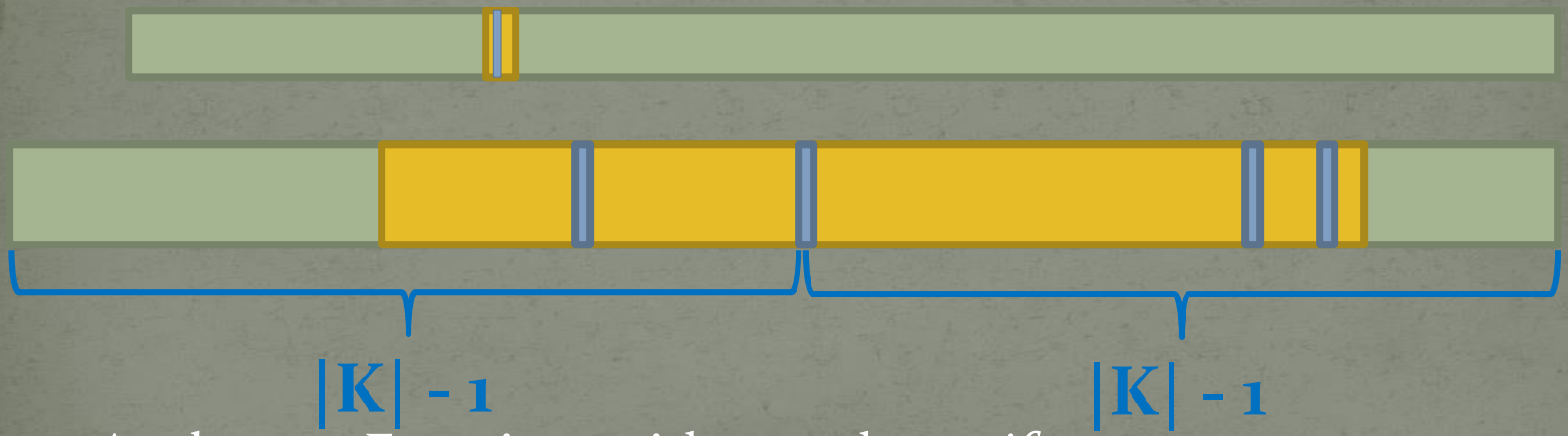
```
R0, #0xC1059ED8
R2, #0x367CD507
R3, #0x3070DD17
R12, #0xF70E5939
LR, #0xFFC00B31
```

```
+aLibpngVersion1 DCB 0xA                    ; DATA XREF: libpng_png_get_copyright↑o
+                DCB " libpng version 1.2.29 - May 8, 2008",0xA
+                DCB "   Copyright (c) 1998-2008 Glenn Randers-Pehrson",0xA
+                DCB "   Copyright (c) 1996-1997 Andreas Dilger",0xA
+                DCB "   Copyright (c) 1995-1996 Guy Eric Schalnat, Group 42, Inc.",0xA
+                DCB 0
```

- Anchors := Functions with complex artifacts

Zooming-In

- Phase I – Locate “Anchor” functions



- Anchors := Functions with complex artifacts

Where are the files?

- Phase II – Draw basic file boundaries

- Some files will remain “floating”

Where are the files?

- Phase III – Build a canonical representation

- We only need to process O(K) functions
- Each context will store useful features

Where are the files?

- F
- V
- E

```
{
    Function Name: png_zalloc,
    Is Static: False,
    Stack Frame Size: 32,
    Instruction Count: 45,
    Numeric Constants: [0, 1048576, 8, 16, 24, 4294967295],
    Strings: [Potential overflow in png_zalloc()],
    Calls: [png_malloc, png_warning],
    Hash: 4f22adfbedd81edc133c0188e19b7396,
    Call Order: {
                    png_malloc : [[]],
                    png_warning: [[]]
                }
}
```

Where are the files?

- Phase IV – Use file “hints”

- File Hint := Function with a string of it’s file name

Basic Matching

- Phase V – Locate “agent” functions

- Agents := Functions with file-unique artifacts

Matching Rounds

- Phase VI – Starting the main matching rounds
- “Regular” binary matching
  - Scoring similarities
  - Using Control Flow Graph (CFG) analysis
- Special attention to geographic location

Geographic Matching

- Rule I – Candidates <u>must</u> be in **same file**
  - Major reduction to the search space
- Rule II - Compilers tend to preserve function order
  - Adaptively boost the score of neighbours
  - Use the neighbours to “Discover” new candidates

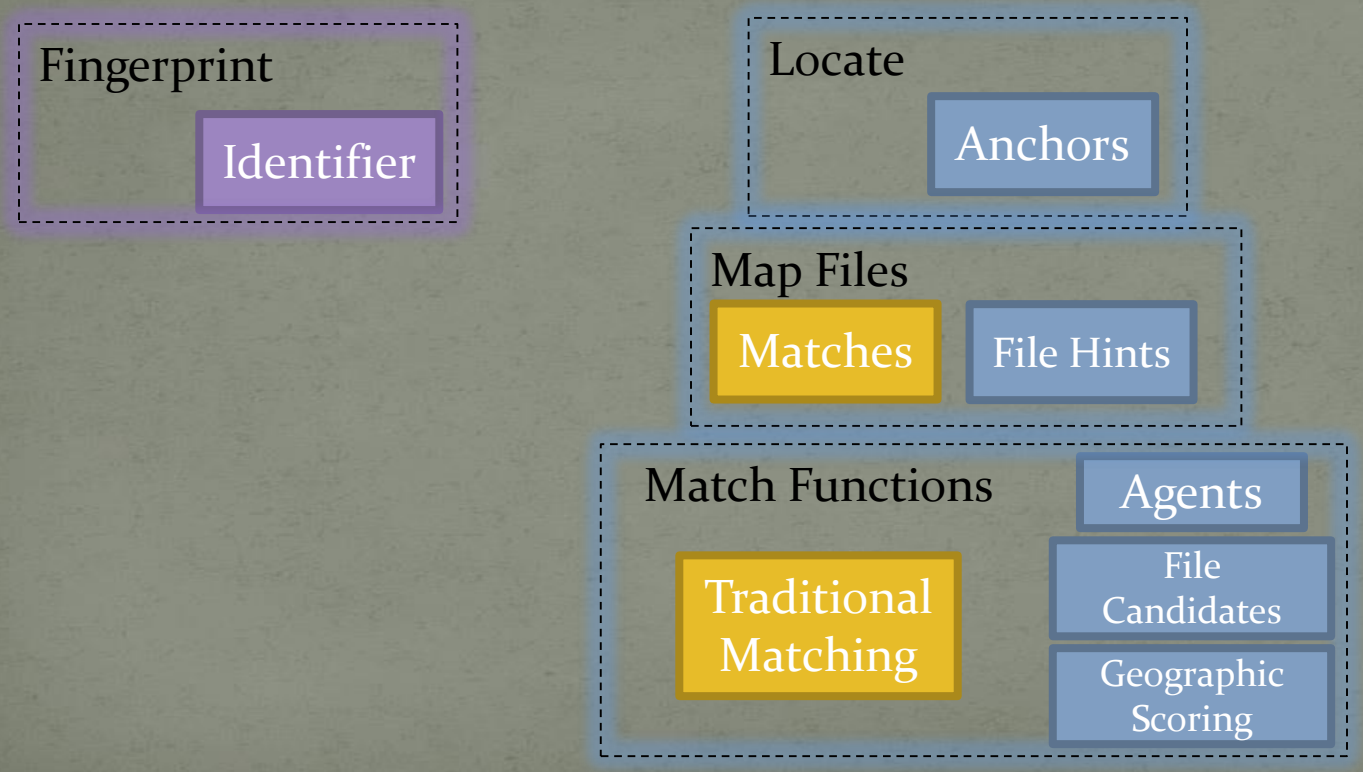
Using Karta

How To Use Karta?

- Compiling a configuration:

- Matching a binary against the configurations:

Architecture



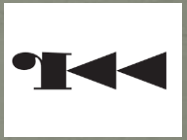
Modularity

- Karta introduces geographic based matching
  - Zooming in on the approximate whereabouts of the library
  - Only comparing candidates from the same file
  - Boosting the score of neighbours
- The rest of the modules can be replaced
  - Better CFG matching heuristics
  - Better scoring algorithm (maybe Machine Learning?)

Modularity

- First developed as an IDA Plugin
  - Code written in Python

- I took @megabeets_'s advice and the disassembler is now an API – can support other disassemblers



- Thanks to @megabeets_ radare2 support is almost ready
- No support for GHIDRA (yet)

Compilation Notes

- Building canonical configurations require some attention
  - Only basic compilation – no optimizations
  - Disable all inline heuristics
  - Different configs for Unix-based/Windows-based binaries
- (Simple) Guidelines are added for each compiled open source
- In the end, the *.json config is architecture agnostic ☺

Linker Optimizations

- Hardly seen in embedded environments
- Integrated into Visual Studio (Windows binaries)
- Key reason to the distinction between Windows configs and non-Windows configs

Linker Optimizations

- Matching libtiff inside Acrobat Reader's 2d.x3d:
  - Started with: 176 / 500 functions ☹
- The linker optimizations break Karta's assumptions:
  - There are no distinct boundaries between files
  - There are less functions than expected

Linker Optimizations

- Merge functions that appears twice (different names):

```c
static int
_TIFFNoFixupTags(TIFF* tif)
{
    (void) tif;
    return (1);
}
```

```c
static int
Fax3FixupTags(TIFF* tif)
{
    (void) tif;
    return (1);
}
```

```
mov     eax, offset libtiff_Fax3Encode
mov     [esi+218h], eax
mov     [esi+220h], eax
mov     [esi+228h], eax
xor     eax, eax
mov     dword ptr [esi+1F8h], offset libtiff__TIFFNoFixupTags
inc     eax
mov     [esi+1FCh], ecx
mov     dword ptr [esi+200h], offset libtiff_Fax3VGetField
mov     [esi+204h], ecx
mov     dword ptr [esi+20Ch], offset libtiff__TIFFFax3fillruns
mov     dword ptr [esi+210h], offset libtiff_Fax3PostEncode
mov     dword ptr [esi+22Ch], offset libtiff_Fax3Close
mov     dword ptr [esi+234h], offset libtiff_Fax3Cleanup
pop     edi
```

Linker Optimizations

- These optimizations also mess up the CFG:
  - Merging together two vertices in the graph
- Additional checks showed other diffing tools fail to match any linker-optimized function in this binary (2d.x3d)
- We need to somehow detect this optimization, and restore back the CFG

Linker Optimizations

- **Solution:** generate a hash-based Linker IDs
  - (Src) Functions with the same ID are linker-identical
  - Creates groups of “collision candidates”
- When a binary function has two collision candidates in it’s options list, it declares a collision
  - The candidate from the same file is taken as “the” match
  - The other candidates are marked as “collision-matched”

Linker Optimizations

```
                        _TIFFClampDoubleToFloat proc near       ; CODE XREF: _TIF

                        arg_0           = qword ptr  4

F2 0F 10 4C 24 04                       movsd   xmm1, [esp+arg_0]
66 0F 2F 0D 40 43 00 00                 comisd  xmm1, ds:__real@47effffe0000000
76 07                                   jbe     short loc_4187
D9 05 4C 43 00 00                       fld     ds:__real@7f7fffff
C3                                      retn
                        ; ---------------------------------------------------------

                        loc_4187:                               ; CODE XREF: _TIF
F2 0F 10 05 74 43 00 00                 movsd   xmm0, ds:__real@c7effffe0000000
66 0F 2F C1                             comisd  xmm0, xmm1
76 07                                   jbe     short loc_4
D9 05 7C 43 00 00                       fld     ds:__real@f
C3                                      retn
                        ; ---------------------------------

                        loc_419C:
DD 44 24 04                             fld     [esp+arg_0]
D9 5C 24 04                             fstp    dword ptr [
D9 44 24 04                             fld     dword ptr [
C3                                      retn
                        _TIFFClampDoubleToFloat endp
```

cal

```
F2 0F 10 4C 24 04
66 0F 2F 0D 1C 2F 00 00
76 07
D9 05 24 2F 00 00
C3
```

```
_TIFFClampDoubleToFloat proc near       ; CODE XREF: __TI
                                        ; __TIFFVSetField

arg_0           = qword ptr  4

                movsd   xmm1, [esp+arg_0]
                comisd  xmm1, ds:__real@47effffe0000000
                jbe     short loc_12FB
                fld     ds:__real@7f7fffff
                retn
; ---------------------------------------------------------

loc_12FB:                               ; CODE XREF: _TIF
                movsd   xmm0, ds:__real@c7effffe0000000
                comisd  xmm0, xmm1
                jbe     short loc_1310
                fld     ds:__real@ff7fffff
                retn
; ---------------------------------------------------------

loc_1310:                               ; CODE XREF: _TIF
                fld     [esp+arg_0]
                fstp    dword ptr [esp+arg_0]
                fld     dword ptr [esp+arg_0]
                retn
_TIFFClampDoubleToFloat endp
```

```
F2 0F 10 05 28 2F 00 00
66 0F 2F C1
76 07
D9 05 30 2F 00 00
C3
```

```
DD 44 24 04
D9 5C 24 04
D9 44 24 04
C3
```

- The candidate from the same file i
- The other candidates are marked a

Linker Optimizations

- After adding this heuristic the results improved drastically
  - Before: 176 / 500 functions (Acrobat Reader – 2d.x3d)
  - After: **248** / 500 functions (Acrobat Reader – 2d.x3d)

File Name	Source Function Name	Binary Address	Binary Function Name	Matching Rule \ Information
tif_compress	_TIFFNoFixupTags	0x1002DEDB	libtiff__TIFFNoFixupTags	Merge - Linker optimization merged source functions
tif_compress	_TIFFtrue	0x1002DEDB	libtiff__TIFFNoFixupTags	Merge - Linker optimization merged source functions
tif_dumpmode	DumpFixupTags	0x1002DEDB	libtiff__TIFFNoFixupTags	Merge - Linker optimization merged source functions
tif_lzw	LZWFixupTags	0x1002DEDB	libtiff__TIFFNoFixupTags	Merge - Linker optimization merged source functions
tif_fax3	Fax3FixupTags	0x1002DEDB	libtiff__TIFFNoFixupTags	Merge - Linker optimization merged source functions
tif_luv	LogLuvFixupTags	0x1002DEDB	libtiff__TIFFNoFixupTags	Merge - Linker optimization merged source functions
tif_dir	tif_dir_TIFFClampDoubleToFloat	0x10034FAE	libtiff_tif_dirwrite_TIFFClampDoubleToFloat	Merge - Linker optimization merged source functions
tif_dirread	_TIFFFillStrilesInternal	0x1002DEDB	libtiff__TIFFNoFixupTags	Merge - Linker optimization merged source functions

Matching Results

- HP OfficeJet test case (65,000 functions):

	libpng	zlib	OpenSSL*
# functions	300	75	3514
# matched functions (TP)	277	68	2857
# unmatched referenced functions	0	3	454
# mismatched functions (FP)	0	0	8
Percentage matched	92% (100%)	91% (96%)	82% (85%)
Running time	26 **seconds**	23 **seconds**	19 minutes

  - Executed inside a Workstation VM on my PC
  - *Didn't manually analyze all of OpenSSL (it is Huge)

Why Karta?

Why Karta and not X?

- It is hard to differentiate between all the bin-diffing tools:
  - BinDiff, Diaphora, Pigaios, FunctionSimSearch
- Each tool has a different goal, and it shows up in it's features
- Here is a brief comparison between the different tools each with the it's goal as was described by their authors

Why Karta and not X?

- **BinDiff**
  - https://www.zynamics.com/bindiff.html
- A comparison tools for binary files
- Assists researchers to quickly find **differences** and similarities in disassembled code
- With BinDiff you can identify and isolate fixes for-supplied patches

Why Karta and not X?

- Diaphora
  - https://github.com/joxeankoret/diaphora
- διαφορά, Greek for 'difference'
- Program **diffing** plugin for IDA, similar to BinDiff

Why Karta and not X?

- Pigaios
  - https://github.com/joxeankoret/pigaios
- 'πηγαίος', Greek for 'source' as in 'source code'
- Tools for diffing / matching source code directly against binaries
- The idea is to point a tool to a code base (regardless of it being compilable)

Why Karta and not X?

- FunctionSimSearch*
  - https://github.com/googleprojectzero/functionsimsearch
- A set of tools to efficiently preform a fuzzy search into a relatively large space of possible functions (the binary)
- The goal is to match known (possibly vulnerable) functions in order to identify statically linked software libraries

Why Karta

- **Identifies** and **matches** open source libraries in a binary
- Designed to support huge binaries with minimal performance

impact	BinDiff	Diaphora	Pigaios	FunctionSimSearch	Karta
Open source	No	Yes	Yes	Yes	Yes
Architecture Agnostic	Yes	Yes	Yes	No	Yes
Supports Large Binaries	No	No	No	Yes	Yes
Source Code Assisted	No	No	Yes	No	Yes
Identifies Versions	No	No	No	Yes	Yes

Demo Time

Thank You

@EyalItkin

eyalit@checkpoint.com

https://github.com/CheckPointSW/Karta