THE ART OF MALWARE C2 SCANNING - HOW TO REVERSE AND EMULATE PROTOCOL OBFUSCATED BY COMPILER

TAKAHIRO HARUYAMA
BINARLY

WHO AM I?

- Takahiro Haruyama (@cci_forensics)
  - Principal Security Researcher at Binarly
  - Previously Staff Threat Researcher at Carbon Black TAU
- Past Research
  - Scalable RE automation (e.g., hunting vulnerable drivers)
  - Anti-Forensics (e.g., firmware acquisition MitM attack)
  - Malware Analysis (e.g., Internet-wide C2 scanning)

AGENDA

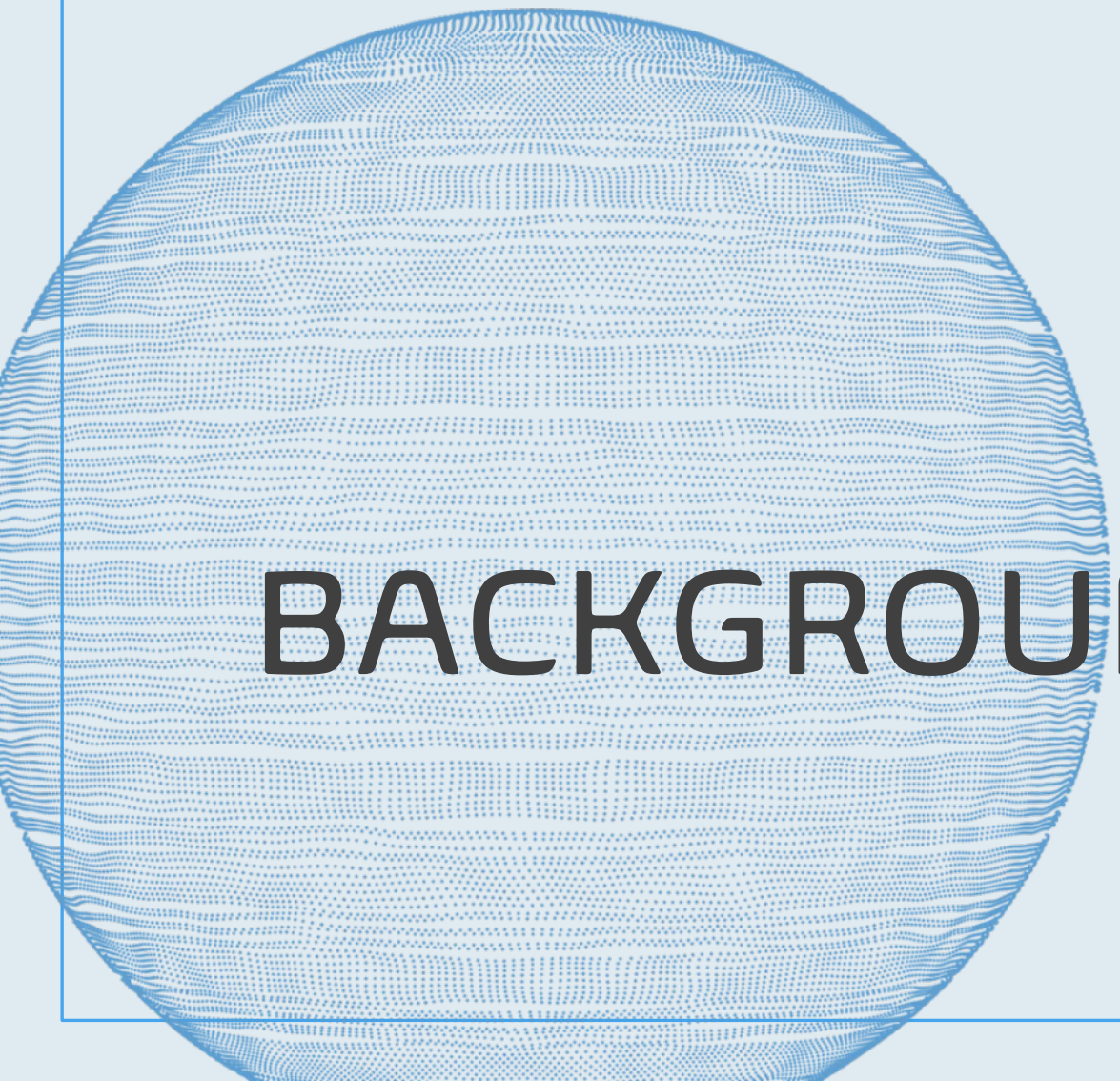
BACKGROUND

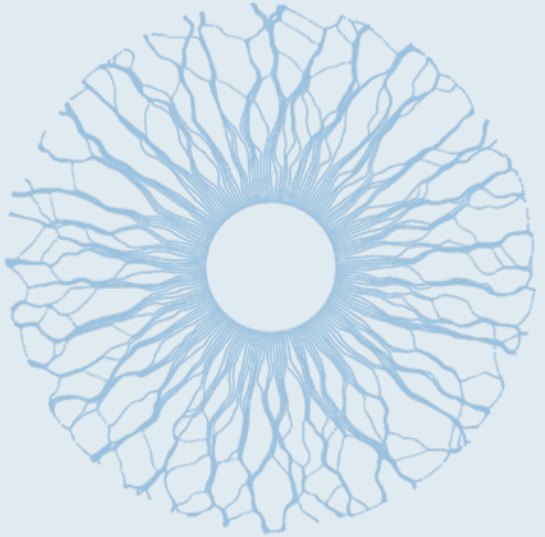
PEELING HODUR: DEFEATING COMPILER-LEVEL OBFUSCATIONS

HODUR PROTOCOL REVERSING

HODUR PROTOCOL EMULATION

WRAP-UP

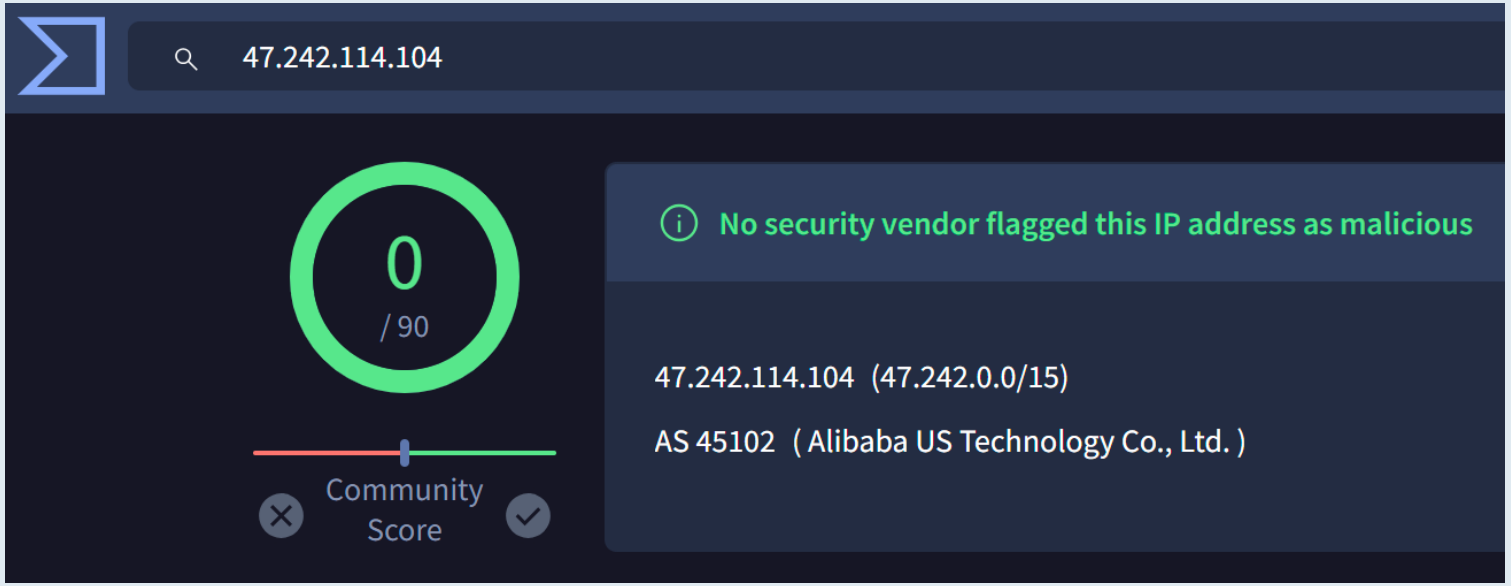




BACKGROUND

WHY MALWARE C2 SCANNING?

- IP reputation is not effective for catching fresh C2s
- Internet-wide C2 scanning is beneficial from both detection and threat intel perspectives



HOW MALWARE C2 SCANNING?

Protocol reversing

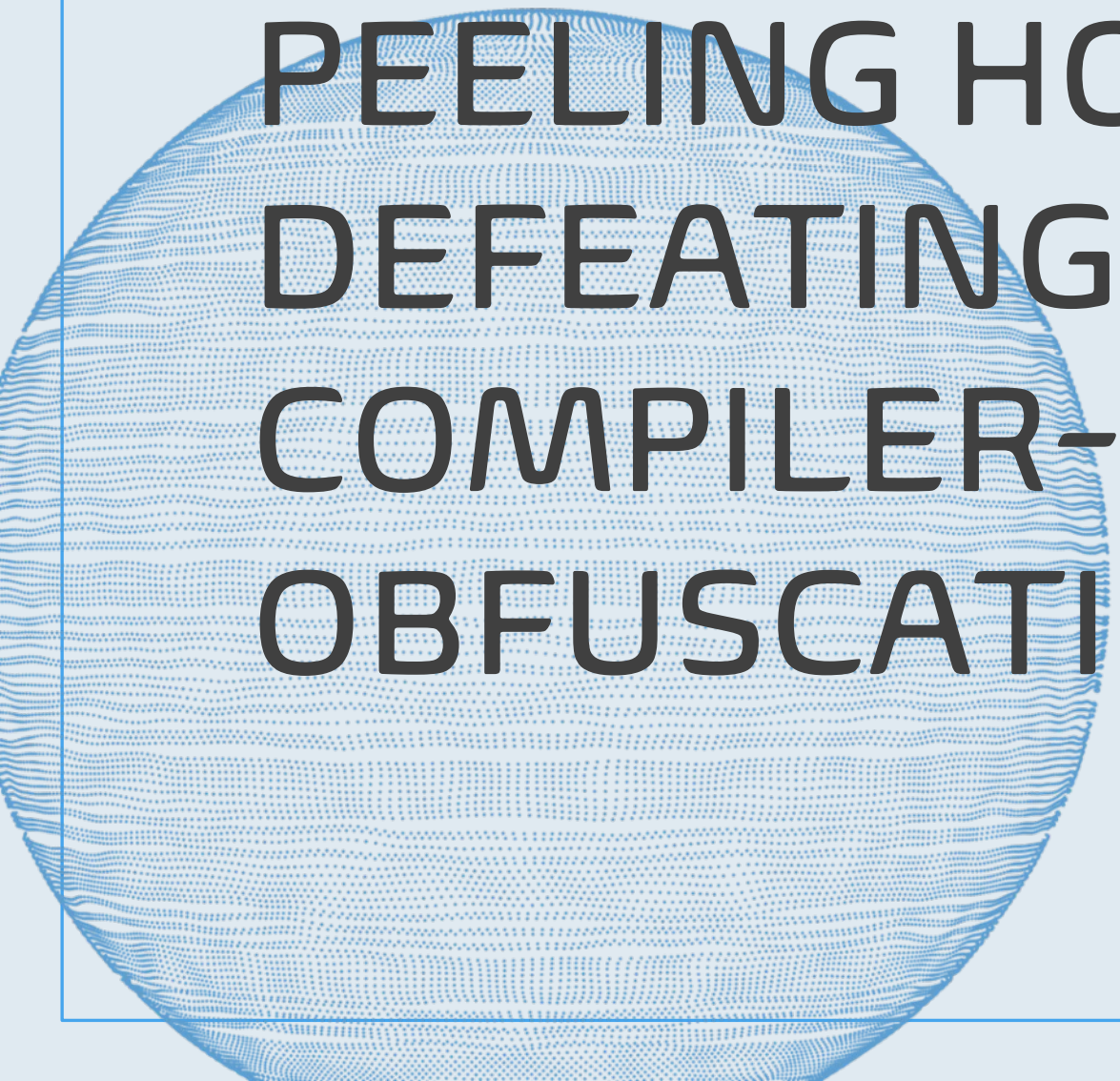
- Identify
  - Data format
  - Encoding/encryption algorithm

Protocol emulation

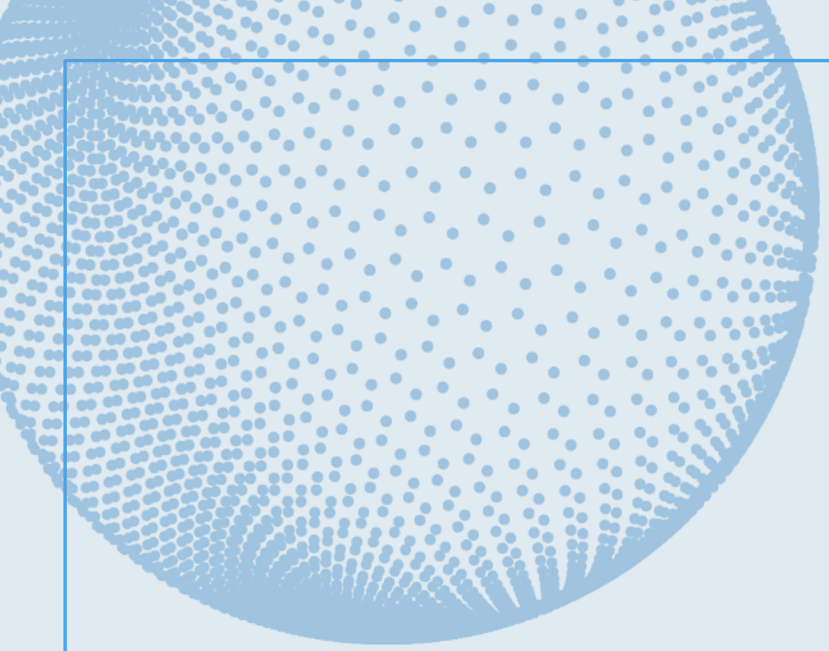
- Develop PoC scanner
- Validate request/response with fake/real C2

CASE: PLUGX

- Long used, but still many variants in the wild
  - Most variants has almost the same C2 protocol except the packet encoding algorithm
- The "Hodur" variants (aka MiniPlug) were obfuscated with multiple methods likely applied at compile time
  - EclecticIQ and Check Point reported the latest variants last year, but no one had described the updated C2 protocol details
- I focus on the Hodur de-obfuscations, then explain the protocol reversing and emulation briefly

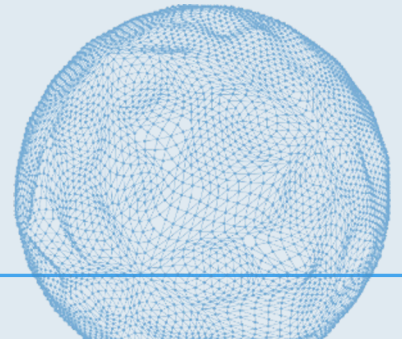


PEELING HODUR: DEFEATING COMPILER-LEVEL OBFUSCATIONS



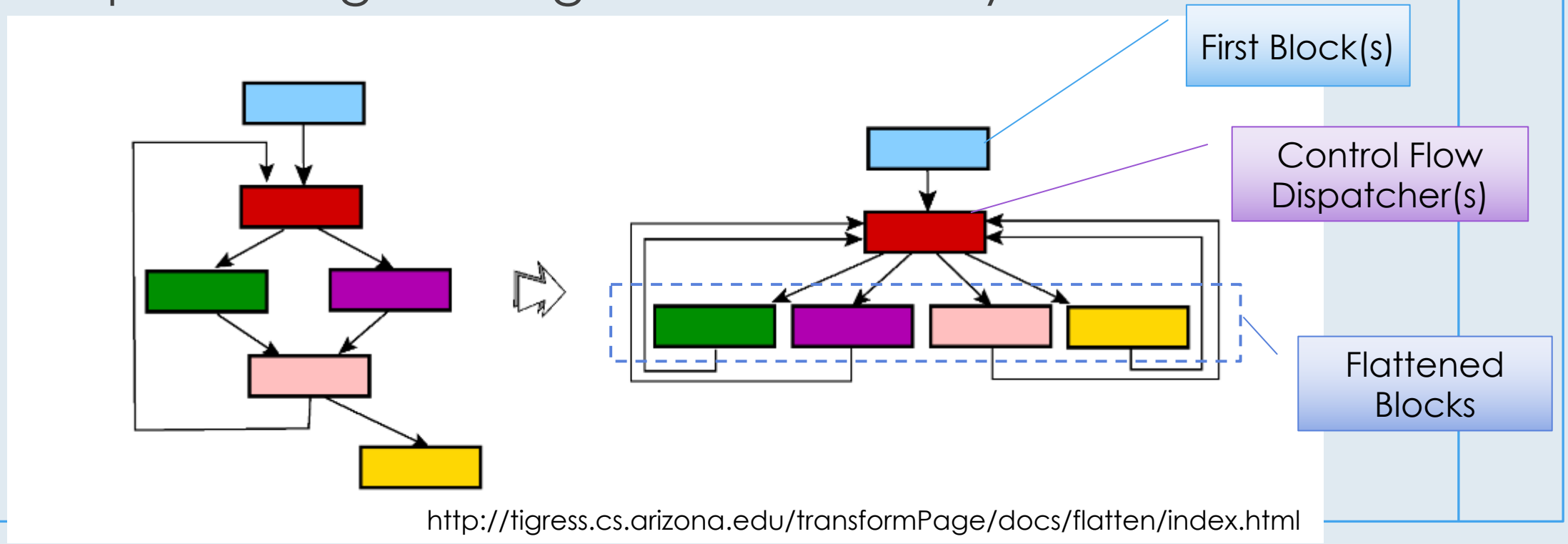
CONTROL FLOW FLATTENING

DEFEATING COMPILER-LEVEL OBFUSCATIONS



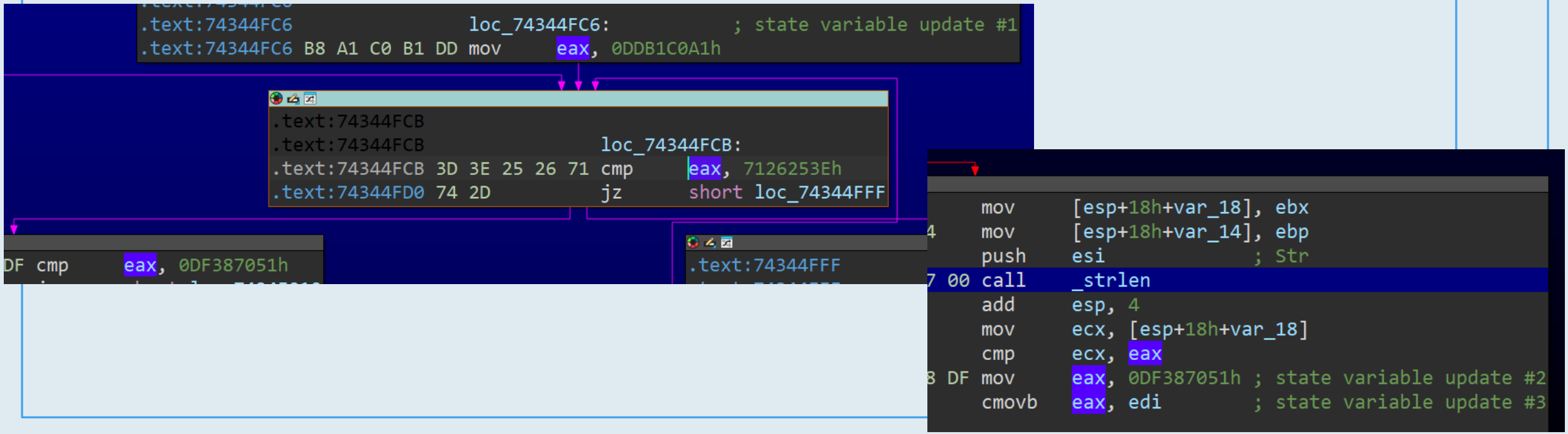
WHAT'S CONTROL FLOW FLATTENING?

- Control flow flattening (CFF) transforms a program's control flow to make it much harder to understand, while preserving the original functionality



HOW CFF WORKS

- Control flow dispatchers decide which block to execute next based on a **state variable**
- The state variable is updated in first/flattened blocks



CONTROL FLOW UNFLATTENING: BASIC STRATEGY

1. Identify control flow dispatchers and state variables
2. Trace back the state variable values from the end of flattened blocks
3. Associate the values with the block IDs
4. Re-order the code flow based on the associations

- I Use IDA Pro microcode for the unflattening task
  - Intermediate representation used by Hex-Rays decompiler
  - We can implement the algorithm in the optblock_t callback

CONTROL FLOW UNFLATTENING: BASIC STRATEGY

1. Iden atchers and state variables
2. Trac able values from the end of flatt
3. Asso the block IDs
4. Re-c ased c
- I Use e
  - tion used by Hex-Rays decompiler
  - algorithm in the optblock_t callback

```
  v1 = 0;
  v2 = 0;
LABEL_2:
  state_var = 0xDDB1C0A1;
  while ( 1 )
  {
    if ( state_var == 0x7126253E )
    {
      v2 = v7 + Str[v6];
      v1 = v6 + 1;
      goto LABEL_2;
    }
    if ( state_var == 0xDF387051 )
      return v7 & (v7 ^ 0xFFFFFF00);
    v6 = v1;
    v7 = v2;
    v4 = v1 < strlen(Str);
    state_var = 0xDF387051;
    if ( v4 )
      state_var = 0x7126253E;
  }
```

```
v1 = 0;
v2 = 0;
while ( v1 < strlen(Str) )
  v2 += Str[v1++];
return v2;
```

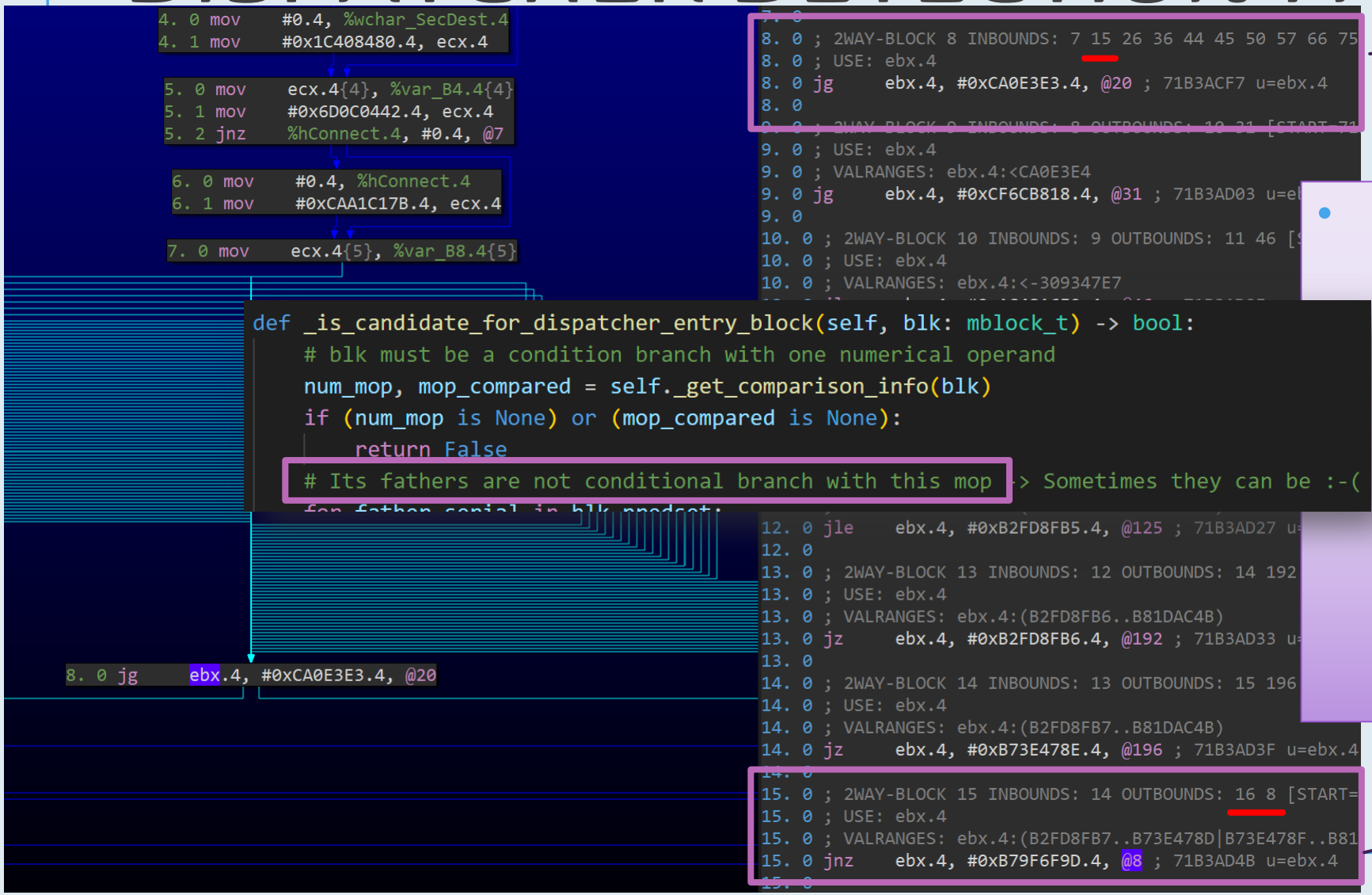
CONTROL FLOW UNFLATTENING: IDA MICROCODE TOOL HISTORY

- HexRaysDeob (2018)
  - The first implementation breaking CFF
  - Ported to IDAPython by Hex-Rays (2019)
  - Tested on only one binary, so some versions implemented
    - APT10 ANEL (2019), Emotet (2022)
- D-810 (2020)
  - Effective for not only OLLVM but also Tigress Flatten
  - Works reliably with different binaries

D-810 ISSUES

- D-810 worked for the most functions of the Hodur samples, but some key functions related to the C2 protocol were still flattened
  - Additional CFF settings?
- Two issues
  1. The control flow dispatcher detections failed
  2. The block state variable tracking failed

ISSUE1: CONTROL FLOW DISPATCHER DETECTION FAILURE



dispatcher

- The dispatcher detection algorithm misses dispatchers whose predecessors are **conditional jumps** by the state variable
  - The genmc plugin was useful for troubleshooting

predecessor

ISSUE1: FIX

- I added another dispatcher detection algorithm
  - The algorithm simply guesses a dispatcher block based on the biggest number of predecessors
  - The dispatcher will be validated based on the entropy value of the state variable (only effective for OLLVM)

```
#if not self._is_candidate_for_dispatcher_entry_block(blk):
if not self._is_candidate_for_dispatcher_entry_block(blk) and blk.serial != self.outmost_dispatch_num:
    return False
```

```
# TODO: I think this can be wrong because we are too permissive in detection of dispatcher blocks
#if len(dispatcher_blk_with_external_father) != 0: # All internal blocks (except the entry block) should not have fathers outside the CFF loop
entropy = self.get_entropy(num_mop.size, blk.serial) # additional check by entropy (only effective for O-LLVM)
if len(dispatcher_blk_with_external_father) != 0 or (entropy < 0.3 or entropy > 0.7): # validate the comparison value's entropy
    unflat_logger.debug(f'mblock {blk.serial} is excluded as a CFF dispatcher ({len(dispatcher_blk_with_external_father)=}, {entropy=})')
    return False
unflat_logger.debug(f'mblock {blk.serial} is detected as a CFF dispatcher entry block')
return True
```

ISSUE1: FIX

- Added another dispatc
- mply gu
- ber of p
- will be v
- e variab

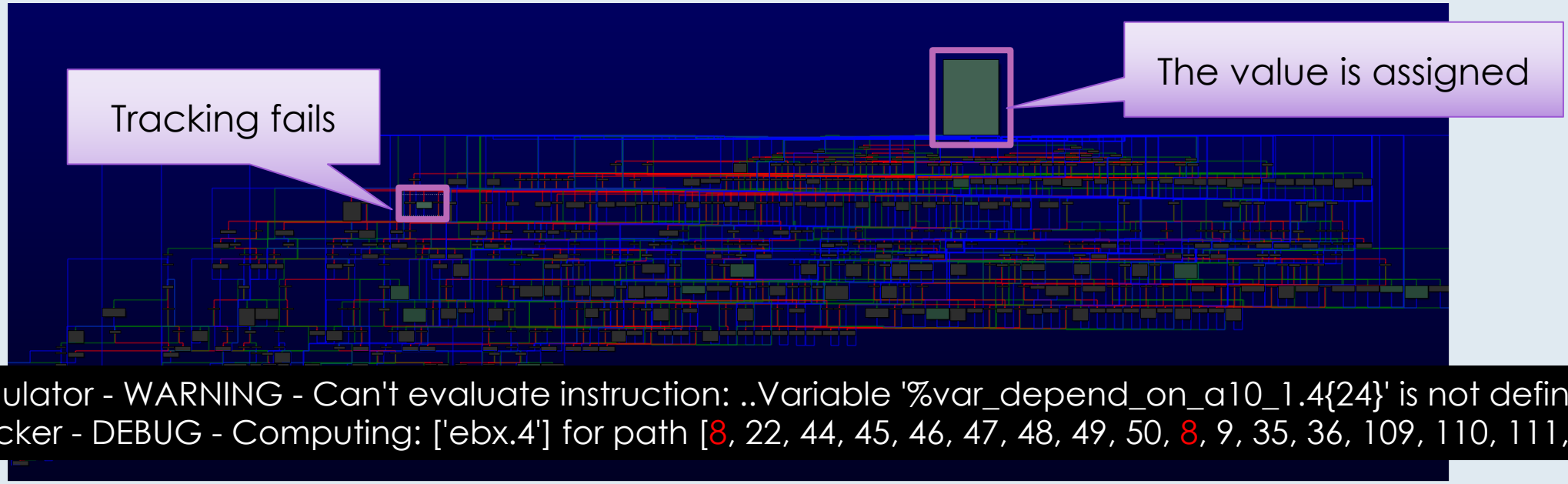
```
        LOBYTE(v137) = WinHttpSendRequest(
                         v243,
                         0,
                         0,
                         post_data,
                         request_length,
                         request_length,// dwT
                         0);
        state_var = -1292005450;
        is_data_remaining = -731028004;
        if ( !v137 )
          state_var = -731028004;
        break;
    }
  }
}
else if ( state_var <= -1292005451 )
{
  if ( state_var == -1404524815 )
    goto LABEL_209;
  if ( state_var == 0xB2D8EADE )
  {
```

```
LOBYTE(v100) = WinHttpSendRequest(
                 v174,
                 0,
                 0,
                 post_data,
                 request_length,
                 request_length,// dwTotalLength (Content-Length) is 0 in GET
                 0);
if ( !v100 )
  goto LABEL_186;

v39 = -230488374;
*(_DWORD *)v171 = (char *)&v177 + 7;
v40 = (char *)&v177 + 6;
wmemcpy(fmt, L"澍黷罰孺汭揿発瑀杣筹繧", 11);
while ( v39 != 1086051502 )
{
  *v40++ = 0;
  v39 = -230488374;
  if ( v40 == *(char **)v171 )
    v39 = 1086051502;
}
for ( i1 = 0; ; i1 = *(_DWORD *)v171 + 1 )
{
  *(_DWORD *)v171 = i1;
  if ( i1 >= 0x16 )
    break;
  *((_BYTE *)fmt + *(_DWORD *)v171) ^= (unsigned __int8)(v171[0] - 29) ^ 0xE3;
}
BYTE6(v177) = 0;
WinHttpReceiveResponse = (BOOL (__stdcall *)(HINTERNET, LPVOID))fn_resolve_API_addr_4((
LOBYTE(v123) = WinHttpReceiveResponse(v174, 0);
```

```
# T                                    o permissive
#if                                    # All intern
ent                                    ) # addition
if                                       (entropy <
    unflat_logger.debug(f'mblock {blk.serial} is excluded as a
    return False
unflat_logger.debug(f'mblock {blk.serial} is detected as a CFF
return True
```

ISSUE2: BLOCK STATE VARIABLE TRACKING FAILURE

- The state variable tracking fails if the value is assigned in the first blocks
  - D-810 only traces in the flattened blocks and doesn't recognize the dispatcher has been reached -> loop ☹




```
D810.emulator - WARNING - Can't evaluate instruction: ..Variable '%var_depend_on_a10_1.4{24}' is not defined
D810.tracker - DEBUG - Computing: ['ebx.4'] for path [8, 22, 44, 45, 46, 47, 48, 49, 50, 8, 9, 35, 36, 109, 110, 111, 112]
```

ISSUE2: FIX

- The added code detects dispatchers in tracking and resumes the tracking from the end of the first blocks
  - The unflattening performance is also improved

```
if self.dispatcher_info and blk_with_multiple_pred.serial == self.dispatcher_info.outmost_dispatch_num:
    logger.debug(f"MopTracker unresolved: reached to the dispatcher {blk_with_multiple_pred.serial}")
    if self.dispatcher_info.last_num_in_first_blks > 0:
        logger.debug(f"Tracking again from the last block {self.dispatcher_info.last_num_in_first_blks} in first blocks before the dispatcher"
        new_tracker = self.get_copy()
        return new_tracker.search_backward(self.mba.get_mblock(self.dispatcher_info.last_num_in_first_blks), None, self.avoid_list, must_use_p
```

```
D810.tracker - DEBUG - MopTracker unresolved: reached to the dispatcher 8
D810.tracker - DEBUG - Tracking again from the last block 7 in first blocks before the dispatcher
D810.tracker - DEBUG - Searching backward (reg): ['%var_depend_on_a10_1.4{24}']
D810.tracker - DEBUG - Searching backward (mem): []
D810.tracker - DEBUG - Searching backward (cst): []
D810.tracker - DEBUG - Updating history with 3. mov     ecx.4{3}, %var_depend_on_a10_1.4{3}
D810.tracker - DEBUG - Removing %var_depend_on_a10_1.4{3} from unresolved mop
D810.tracker - DEBUG - Adding ecx.4{3} in unresolved mop
D810.tracker - DEBUG - Updating history with 1. mov     #0x52B8AAAD.4, ecx.4
D810.tracker - DEBUG - Removing ecx.4 from unresolved mop
D810.tracker - DEBUG - MopTracker is resolved:  [1, 3, 5, 7, 8, 9, 35, 36, 109, 110, 111, 112]
```

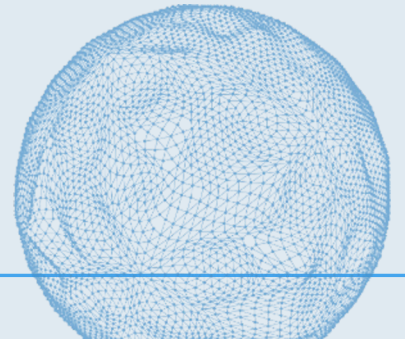
ISSUE2: FIX

```
LABEL_229:
  v23 = v227;
  LastError = 0;
  v183 = 0;
  do
  {
    while ( 1 )
    {
      while ( 1 )
      {
        while ( 1 )
        {
          while ( 1 )
          {
            while ( v23 <= 211870691 )
            {
              if ( v23 > -814958568 )
              {
                if ( v23 <= -82327404 )
                {
                  if ( v23 <= -564835336 )
                  {
                    if ( v23 > -731028005 )
                    {
                      if ( v23 == -731028004 )
                        goto LABEL_186;
                      if ( v23 == -574824336 )
                        goto LABEL_182;
                    }
                    else
                    {
                      if ( v23 == -814958567 )
```

```
LastError = 0;
v179 = 0;
if ( v223 <= 211870691 )
  goto LABEL_130;
is_HTTPS = protocol == 1;
if ( protocol == 1 )
  dwFlags = WINHTTP_FLAG_SECURE;              // TLS
else
  dwFlags = WINHTTP_FLAG_REFRESH;
v59 = 'Km[s';
*(_DWORD *)v165 = (char *)&v171 + 3;
v60 = (char *)&v171 + 2;
wmemcpy(fmt, L"尌魮罰焜楅礶扩臻搗", 9);
while ( v59 != 356431705 )
{
  *v60++ = 0;
  v59 = 'Km[s';
  if ( v60 == *(char **)v165 )
    v59 = 356431705;
}
for ( i = 0; ; i = *(_DWORD *)v165 + 1 )
{
  *(_DWORD *)v165 = i;
  if ( i >= 0x12 )
    break;
  *((_BYTE *)fmt + *(_DWORD *)v165) ^= (unsigned __int8)(v165[0] - 29) ^ 0xE3;
}
BYTE2(v171) = 0;
WinHttpOpenRequest = (HINTERNET (__stdcall *)(HINTERNET, LPCWSTR, LPCWSTR, LPCWSTR
v168 = WinHttpOpenRequest(hConnect, method, 0, 0, 0, (LPCWSTR *)v209, dwFlags);//
```

MIXED BOOLEAN ARITHMETIC EXPRESSIONS

DEFEATING COMPILER-LEVEL OBFUSCATIONS



```
  *((_BYTE *)enc + *(_DWORD *)v165) ^= 0x19 ^ (0x1C - v165[0]) & 0xFA ^ (v165[0] - 0x1D) & 5;
}
LOBYTE(v171) = 0;
WinHttpSetOption = (BOOL (__stdcall *)(HINTERNET, DWORD, LPVOID, DWORD))fn_resolve_API_addr_4((const CHAR *)enc);//
WinHttpSetOption(v168, WINHTTP_OPTION_CONNECT_TIMEOUT, &g_timeout_msec, 4);
v93 = -122112760;
*(_DWORD *)v165 = (char *)&v171 + 1;
v94 = &v171;
*(_OWORD *)enc = g_enc_WinHttpSetOption;
while ( v93 != 599437014 )
{
  *(_BYTE *)v94 = 0;
  v94 = (__int128 *)((char *)v94 + 1);
  v93 = -122112760;
  if ( v94 == *(__int128 **)v165 )
    v93 = 599437014;
}
for ( k = 0; ; k = *(_DWORD *)v165 + 1 )
{
  *(_DWORD *)v165 = k;
  if ( k >= 0x10 )
    break;
  *((_BYTE *)enc + *(_DWORD *)v165) = ~((v165[0] - 0x1D) ^ *((_BYTE *)enc + *(_DWORD *)v165) ^ 0x1C);
}
LOBYTE(v171) = 0;
WinHttpSetOption_1 = (BOOL (__stdcall *)(HINTERNET, DWORD, LPVOID, DWORD))fn_resolve_API_addr_4((const CHAR *)enc);/
WinHttpSetOption_1(v168, WINHTTP_OPTION_RECEIVE_TIMEOUT, &g_timeout_msec, 4);
v151 = -122112760;
*(_DWORD *)v165 = (char *)&v171 + 1;
v152 = &v171;
*(_OWORD *)enc = g_enc_WinHttpSetOption;
while ( v151 != 599437014 )
{
  *(_BYTE *)v152 = 0;
  v152 = (__int128 *)((char *)v152 + 1);
  v151 = -122112760;
  if ( v152 == *(__int128 **)v165 )
    v151 = 599437014;
}
for ( m = 0; ; m = *(_DWORD *)v165 + 1 )
{
  *(_DWORD *)v165 = m;
  if ( m >= 0x10 )
    break;
  *((_BYTE *)enc + *(_DWORD *)v165) ^= (unsigned __int8)(v165[0] - 0x1D) ^ 0xE3;
}
LOBYTE(v171) = 0;
WinHttpSetOption_2 = (BOOL (__stdcall *)(HINTERNET, DWORD, LPVOID, DWORD))fn_resolve_API_addr_4((const CHAR *)enc);/
WinHttpSetOption_2(v168, WINHTTP_OPTION_SEND_TIMEOUT, &g_timeout_msec, 4);
```

- Mixed Boolean Arithmetic (MBA) expressions transform a simple expression into a complex but semantically equivalent form

The same encoded string is decoded in different expressions

SIMPLIFYING MBA EXPRESSIONS

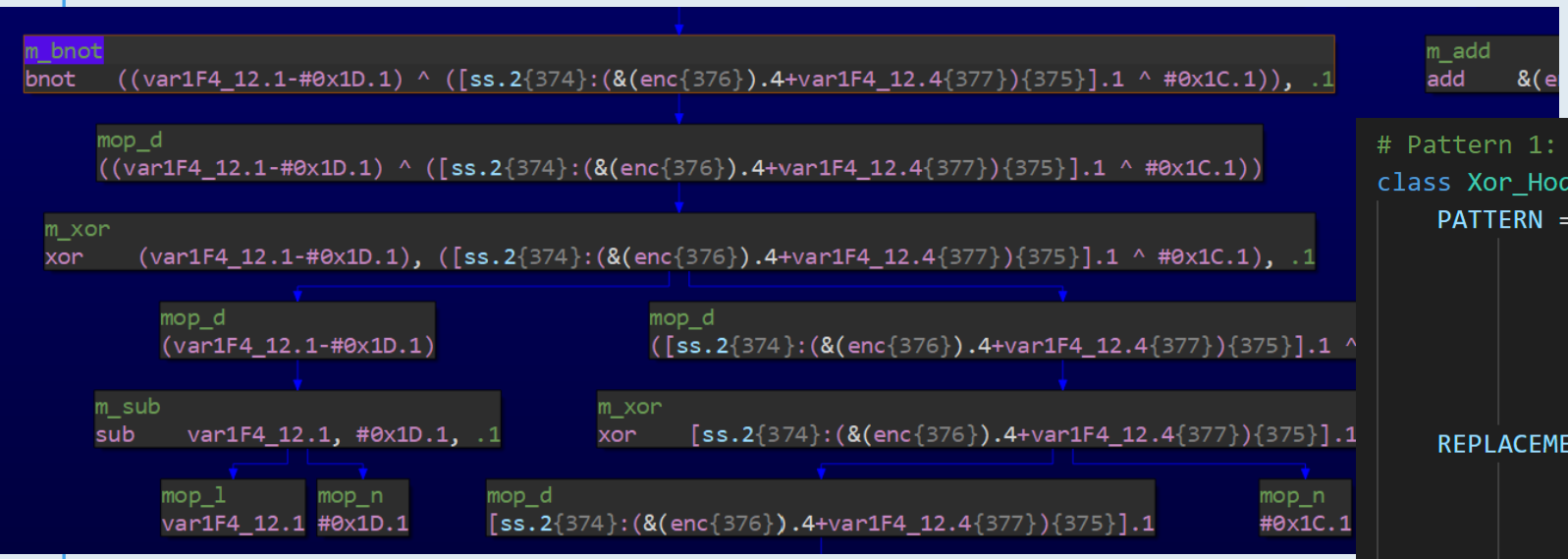
1. Find an obfuscation pattern and hypothesize for simplification
2. Validate the hypothesis by equivalence checking
   - e.g., using Z3 or Arybo
3. Replace the pattern with the simplified one

```
$ ipython

In [1]: import z3

In [2]: x, y = z3.BitVecs("x y", 8)

In [3]: s = z3.SolverFor("QF_BV")

In [4]: s.add((~(x ^ ~y)) != (x ^ y))

In [5]: s.check()
Out[5]: unsat
```

```
$ iarybo 8

In [1]: ~(x ^ ~y) == x ^ y
Out[1]: True
```

SIMPLIFICATION ON IDA + D-810

- D-810 uses a custom AstNode class to represent an (abstract) microcode instruction
  - I could easily define several new replacement patterns
  - genmc is useful to show microcode instruction structures




```
# Pattern 1: ~(enc[i] ^ ((i - 0x1D) ^ 0x1C)) == enc[i] ^ (i - 0x1D) ^ 0xE3
class Xor_Hodur_1(PatternMatchingRule):
    PATTERN = AstNode(m_bnot,
                      AstNode(m_xor,
                              AstLeaf("x_0"),
                              AstNode(m_xor,
                                      AstLeaf("x_1"),
                                      AstLeaf("x_2"))))
    REPLACEMENT_PATTERN = AstNode(m_xor,
                                  AstLeaf("x_0"),
                                  AstNode(m_xor,
                                          AstLeaf("x_1"),
                                          AstNode(m_bnot,
                                                  AstLeaf("x_2"))))
```

SIMPL

- D-810 (abstro
  - I c
  - ge

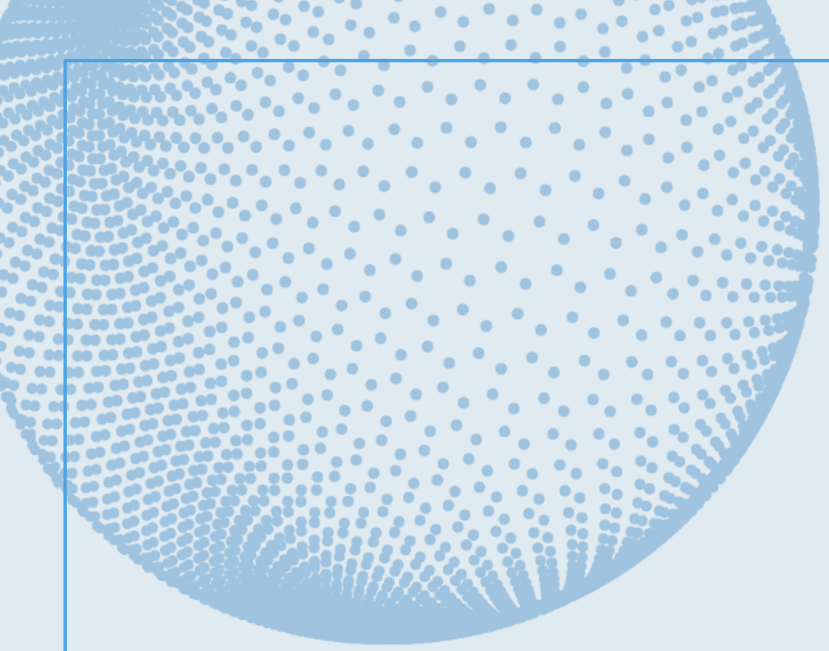
```
  *((_BYTE *)enc + *(_DWORD *)v165) ^= (unsigned __int8)(v165[0] - 0x1D) ^ 0xE3;
}
LOBYTE(v171) = 0;
WinHttpSetOption = (BOOL (__stdcall *)(HINTERNET, DWORD, LPVOID, DWORD))fn_resolve_API_addr_4((const CHAR *)enc);//
WinHttpSetOption(v168, WINHTTP_OPTION_CONNECT_TIMEOUT, &g_timeout_msec, 4);
v93 = -122112760;
*(_DWORD *)v165 = (char *)&v171 + 1;
v94 = &v171;
*(_OWORD *)enc = g_enc_WinHttpSetOption;
while ( v93 != 599437014 )
{
  *(_BYTE *)v94 = 0;
  v94 = (__int128 *)((char *)v94 + 1);
  v93 = -122112760;
  if ( v94 == *(__int128 **)v165 )
    v93 = 599437014;
}
for ( k = 0; ; k = *(_DWORD *)v165 + 1 )
{
  *(_DWORD *)v165 = k;
  if ( k >= 0x10 )
    break;
  *((_BYTE *)enc + *(_DWORD *)v165) ^= (v165[0] - 0x1D) ^ 0xE3;
}
LOBYTE(v171) = 0;
WinHttpSetOption_1 = (BOOL (__stdcall *)(HINTERNET, DWORD, LPVOID, DWORD))fn_resolve_API_addr_4((const CHAR *)enc);
WinHttpSetOption_1(v168, WINHTTP_OPTION_RECEIVE_TIMEOUT, &g_timeout_msec, 4);
v151 = -122112760;
*(_DWORD *)v165 = (char *)&v171 + 1;
v152 = &v171;
*(_OWORD *)enc = g_enc_WinHttpSetOption;
while ( v151 != 599437014 )
{
  *(_BYTE *)v152 = 0;
  v152 = (__int128 *)((char *)v152 + 1);
  v151 = -122112760;
  if ( v152 == *(__int128 **)v165 )
    v151 = 599437014;
}
for ( m = 0; ; m = *(_DWORD *)v165 + 1 )
{
  *(_DWORD *)v165 = m;
  if ( m >= 0x10 )
    break;
  *((_BYTE *)enc + *(_DWORD *)v165) ^= (unsigned __int8)(v165[0] - 0x1D) ^ 0xE3;
}
LOBYTE(v171) = 0;
WinHttpSetOption_2 = (BOOL (__stdcall *)(HINTERNET, DWORD, LPVOID, DWORD))fn_resolve_API_addr_4((const CHAR *)enc);
```

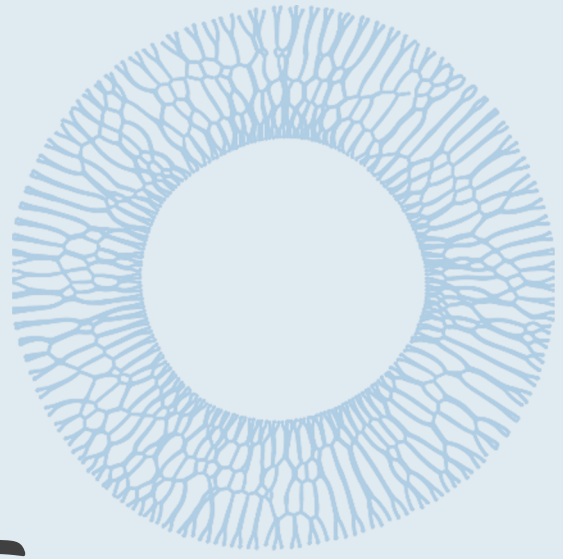
```
m_bnot
bnot    ((var1F4_12.1-#0x1D.1) ^ ([s

mop_d
((var1F4_12.1-#0x1D.1) ^ ([ss

m_xor
xor     (var1F4_12.1-#0x1D.1), ([s

mop_d
(var1F4_12.1-#0x1D.1)

m_sub
sub     var1F4_12.1, #0x1D.1,

mop_l           mop_n
var1F4_12.1     #0x1D.1
```

```
) == enc[i] ^ (i - 0x1D) ^ 0xE3

),
af("x_1"),
af("x_2"))))

_0"),
_xor,
tLeaf("x_1"),
tNode(m_bnot,
    AstLeaf("x_2"))))
```

LIMITATION

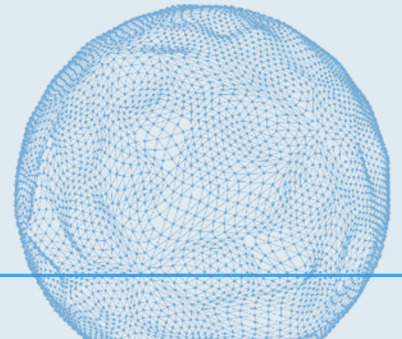
- More functions, more complicated patterns ☹
  - It was difficult to defeat all MBA expressions perfectly
- I only handled interesting patterns, especially related to the string decoding used by the samples





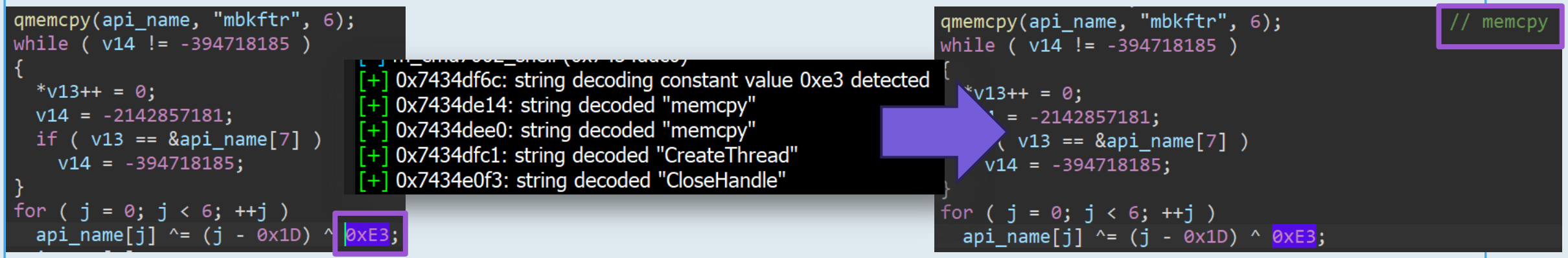
POLYMORPHIC STACK STRINGS

DEFEATING COMPILER-LEVEL OBFUSCATIONS



STACK STRINGS

- All strings are constructed and decoded in the stack area
- After defeating CFF and MBA expressions, the decoding algorithm was identified
  - enc[i] ^= (i + Const) ^ Const
  - The constant value is different per function



COPYING THE ENCODED STRING BYTES INTO STACK

- Sometimes the Hex-Rays decompiler partially recognizes the copy or only shows the assignments
- For static decoding, we need to
  - Construct the bytes from the assigned variables
  - Detect the length and constant value used in the decoding algorithm

```
qmemcpy(api_name, "Tbthmek}", 8);
*(_DWORD *)&api_name[8] = 2137414509;
qmemcpy(&api_name[12], "irv", 3);
v44 = &api_name[15];
```

```
*(_OWORD *)api_name = xmmword_743C72B0;
v20 = -30839;
v21 = -105;
while ( v8 != -927398101 )
{
  *(_BYTE *)v7 = 0;
  v7 = (int *)((char *)v7 + 1);
  v8 = -2095552381;
  if ( v7 == &v23 )
    v8 = -927398101;
}
for ( j = 0; j < 0x13; ++j )
  api_name[j] ^= (unsigned __int8)(j + 0x79) ^ 0x79;
```

Combination of global variable and hard-coded bytes

Length and constant value

VARIOUS ACCESS PATTERNS

```
enc[1] = tmp ^ 0xF;
for ( k = '4}=H'; k != -1054643335; k = -1054643335 )
  tmp = 114;
enc_1[2] = tmp ^ 0xC;
for ( m = '4}=H'; m != -1054643335; m = -1054643335 )
  tmp = 114;
enc_1[3] = tmp ^ 0xD;
for ( n = '4}=H'; n != -1054643335; n = -1054643335 )
  tmp = 111;
enc_1[4] = tmp ^ 0xA;
for ( ii = '4}=H'; ii != -1054643335; ii = -1054643335 )
  tmp = 114;
enc_1[5] = tmp ^ 0xB;
for ( jj = '4}=H'; jj != -1054643335; jj = -1054643335 )
  tmp = 77;
enc_1[6] = tmp ^ 8;
for ( kk = '4}=H'; kk != -1054643335; kk = -1054643335 )
  tmp = 111;
enc_1[7] = tmp ^ 9;
LOBYTE(v308) = 100;
v316 = (LPCWSTR)(enc_1 + 8);
for ( mm = '4}=H'; mm != -1054643335; mm = -1054643335 )
  tmp = (char)v308;
*(_BYTE *)v316 = tmp ^ 0x36;
v49 = '4}=H';
v50 = v316;
```

Additional XORs before decoding

```
i = -17;
enc = xmmword_71BB7220;
LOWORD(v13) = 2050;
BYTE2(v13) = 0;
LOBYTE(enc) = 87;
do
{
  v14[i - 2] ^= (unsigned __int8)(i + 68) ^ 0x32;
  ++i;
}
while ( i );
```

Referencing another variable (enc is decoded)

Defeating MBA expressions is not perfect

```
for ( k = 0; k < 5; ++k )
  api_name[k] = ((~api_name[k] & 0x21 | api_name[k] & 0xCC) ^ 0xC0 | api_name[k] & 0x12 ^ 0x10) ^ (k - 29) ^ 0x12;
```

I decided to take an emulation approach

EMULATION ISSUE IN GENERAL

- Unicorn-based flare-emu library provides users with a flexible interface for scripting emulation tasks on IDA
- The **iterateAllPaths** API emulates all basic block paths in a function
  - Looked to be useful to de-obfuscate stack strings (e.g., ironstrings)
- This API emulates only once per basic block
  - I modified the code to reproduce xor loops detected by CAPA

```
CAPA_CMT = '; CAPA (basic block): encode data using XOR'
```

```
#if address == bbStart and self.enteredBlock is True:
ext_cmt = idc.get_extra_cmt(address, idc.E_PREV)
if address == bbStart and self.enteredBlock is True and ext_cmt != CAPA_CMT:
    if self.blockIdx < len(paths[self.pathIdx]) - 1:
        self.logger.debug("loop re-entering block #%d (%s -> %s), forcing PC to %s" %
```

EMULATION ISSUE IN THIS SAMPLE

- The flare-emu API takes only one path in CFF functions
  - The code simply tracks basic block successors
  - The search ends when revisiting the CFF dispatchers
- Microcode-based solutions
  - Emulate x86 code in an unflattened microcode block order
  - Extend D-810 microcode emulation functionality
- I tried both a little bit, but I realized that they are not straightforward ☹

```
# returns a dictionary where the key is a node in the control flow graph
# and its value is a list of its successor nodes
def _explore(self, start_bb, end_bb=None):
```

SOLUTION

- I utilized another flare-emu API (**emulateRange**) that emulates the code as is, without changing the code flow
  - Some quick hacks added to flare-emu (e.g., LoadLibrary/GetProcAddress hook, infinite loop detection, etc.)
  - The created script worked for **58**% of the tested functions
- I also implemented a script based on the IDA debug hook class (**DBG_Hooks**) to handle the failed functions
- Not elegant, but the combination covers most strings quickly

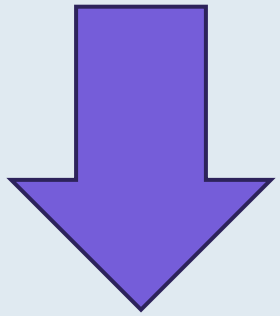
SOLUTION (CONT.)

- Both scripts recover argument strings on call instructions in emulation/debugging
  - The information such as calling convention and argument type is taken through the Hex-Rays decompiler APIs
- The sample dynamically resolves all API addresses except GetProcAddress after decoding the API name strings
  - When an address assignment is detected, the script applies the API function type to the local variable pointer
    - GetTypeSignature() written by Rolf Rolles

```
v8 = fn_resolve_API_addr(v11);
((void (__stdcall *)(int, int *))v8)(v4, &v10);
```

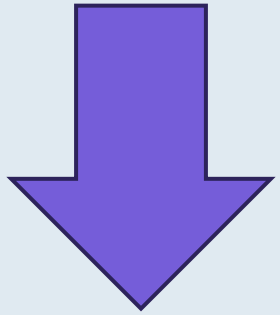
```
CheckRemoteDebuggerPresent = (BOOL (__stdcall *)(HANDLE, PBOOL))fn_resolve_API_addr(v11);// 0x71b351b7 (fn_test_func+0x192): arg0 = CheckRemoteDebuggerPresent
CheckRemoteDebuggerPresent(v4, &v10);
```

```
v170 = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, int))fn_resolve_API_
v171 = v170(
         v173[0],
         v173[1],
         v173[2],
```



Set type to the local variable by ida_hexrays.modify_user_lvars()

```
GetCurrentProcess = (HANDLE (__stdcall *)())fn_resolve_API_addr_0(v181);// 0x74385365 (fn_AdjustTokenPrivileges+0x2623): arg0 = GetCurrentProcess
v171 = (void *)((int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, int))GetCur
                 v173[0],
                 v173[1],
```



Set type to the operand of the call instruction by ida_nalt.set_op_tinfo()

```
GetCurrentProcess = (HANDLE (__stdcall *)())fn_resolve_API_addr_0(v180);// 0x74385365 (fn_AdjustTokenPrivileges+0x2623): arg0 = GetCurrentProcess
v171 = GetCurrentProcess();
```

SOLUTION (CONT.)

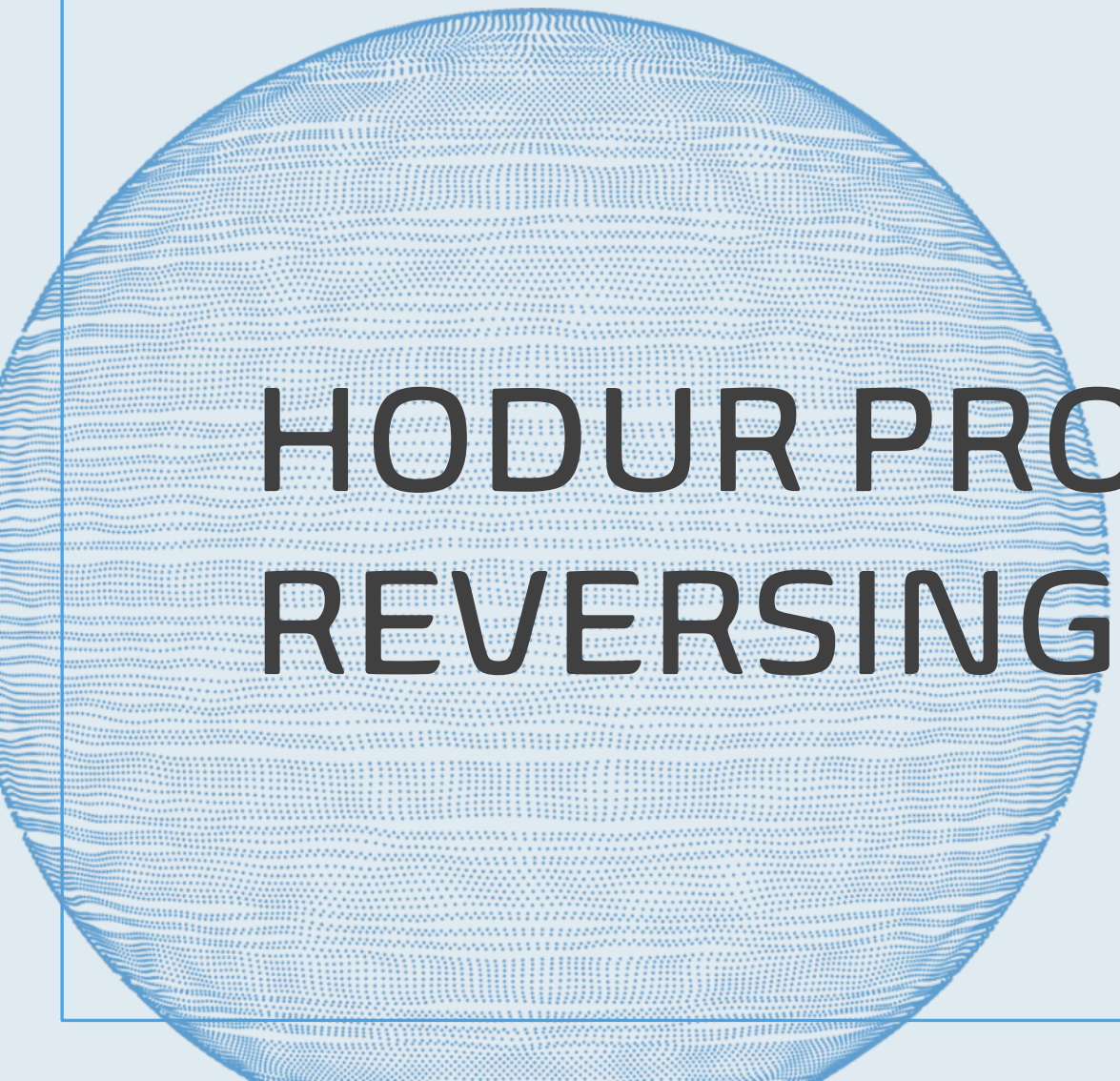
```
mov     eax, dword_743CF45C
movaps  xmm0, ds:xmmword_743C7670

mov     ecx, 0F24306CAh
mov     [esp+94h+var_3C], eax
lea     eax, [esp+94h+var_7C+3]
mov     dword ptr [esp+94h+var_38], ebx
movups  xmmword ptr [esp+94h+api_name], xmm0
mov     [esp+94h+var_80], 'vs}r' ; str = r}sv

mov     dword ptr [esp+94h+var_34], ebp
mov     word ptr [esp+94h+var_7C], 'h``'

mov     [esp+94h+var_94], eax
lea     eax, [esp+94h+var_7C+2]
```

```
[+] 0x74350fa0: string decoding constant value 0xe3 detected
[+] string decoded "WaitForMultipleObjects"
```

- The scripts still don't cover all strings
- A semi-automatic script handles minor cases individually
  - flare-emu **emulateSelection** + static decoding

IDA_CALLSTRINGS SCRIPTS

Used Library and API	Static decoding	Flare-emu iterateAllPaths	Flare-emu emulateRange	Flare-emu emulateSelection	IDA DBG_Hooks
Automated?	Yes	Yes	Yes	No	Yes
Effective for another malware?	No	Yes	Yes	No	Yes
Effective in CFF funcs?	Yes	No	Yes	-	Yes
API func type set?	No	Yes	Yes	No	Yes
Limitation	Strings used by memcpy	Modifications needed to flare-emu and CAPA	All execution paths not covered	Manual selection required	Strings used during debugging



HODUR PROTOCOL REVERSING

PROTOCOL OVERVIEW

- The latest Hodur samples only support HTTP/HTTPS
- Two header values (**Sec-Dest**/**Sec-Site**) used to authenticate clients
- GET request for the initial handshake
  - A **RC4** key returned
- Periodical POST requests to receive C2 commands after the handshake
  - The request/response data are encrypted with the key

AUTHENTICATION HEADERS

- Sec-Dest: %2.2X%ws (e.g., “7BnqmmCg”)
  - A random byte (**0x64-0x99**)
    - 0x64 + 0-0x35 by QueryPerformanceCounter
  - A random 6 characters
    - The checksum depends on the method
    - GET = **99**, POST = **88**

```
In [2]: sum(b for b in b'nqmmCg') & 0xff
Out[2]: 99
```

- Sec-Site: %2.2X%2.2X%ws (e.g., “896B2AC144C9E2E09836”)
  - Two random bytes (**0x64-0x99**)
  - 8-bytes victim ID generated by time-related APIs

INITIAL HANDSHAKE

- GET request with the authentication headers
- A RC4 key is returned if the header values are valid
  - If not valid, no content returned
  - The Hodur sample code checks if the Content-Type is **application/octet-stream**
  - The Content-Length was unknown at static analysis but revealed during the scanner development

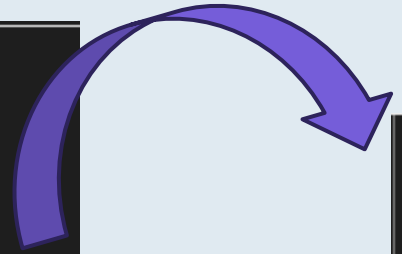
AFTER HANDSHAKE

- The sample receives a C2 command by POST requests
- The POST request and response data are encrypted using RC4
  - The POST data header is the same as the PlugX variants, but the head key is not used
  - The C2 response body also has the same header

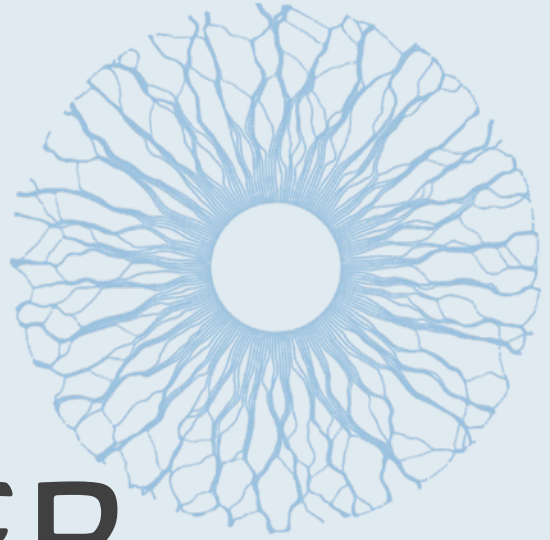
```
struct struc_pkt_header
{
  int key;                                ///< RC4 key nonce? -> not used
  int command_id;                         ///< 2: C2 command request, 0x7002:
remote shell, 0x1007: timeout change, 0x3004: download and run, 0x1005:
uninstall
  int payload_len;
  int errc;
};
```

POST DATA PAYLOAD

```
struct struc_pkt_payload
{
  __int16 imm0;
  __int16 rand_bytes[15];
  struc_victim_info victim_info;
};
```



```
struct struc_victim_info
{
  _DWORD Wow64Process;
  int dwMajorVersion;
  int dwMinorVersion;
  int dwPlatformId;
  int dwBuildNumber;
  __int16 wProductType;
  __int16 wServicePackMajor;
  __int16 wServicePackMinor;
  wchar_t user_name[64] __strl
  wchar_t computer_name[64] __
  wchar_t unk_env_string[64] _
  wchar_t campaign_ID[64] __st
  wchar_t hostname[64] __strli
  char padding[386];
};
```



HODUR SCANNER DEVELOPMENT

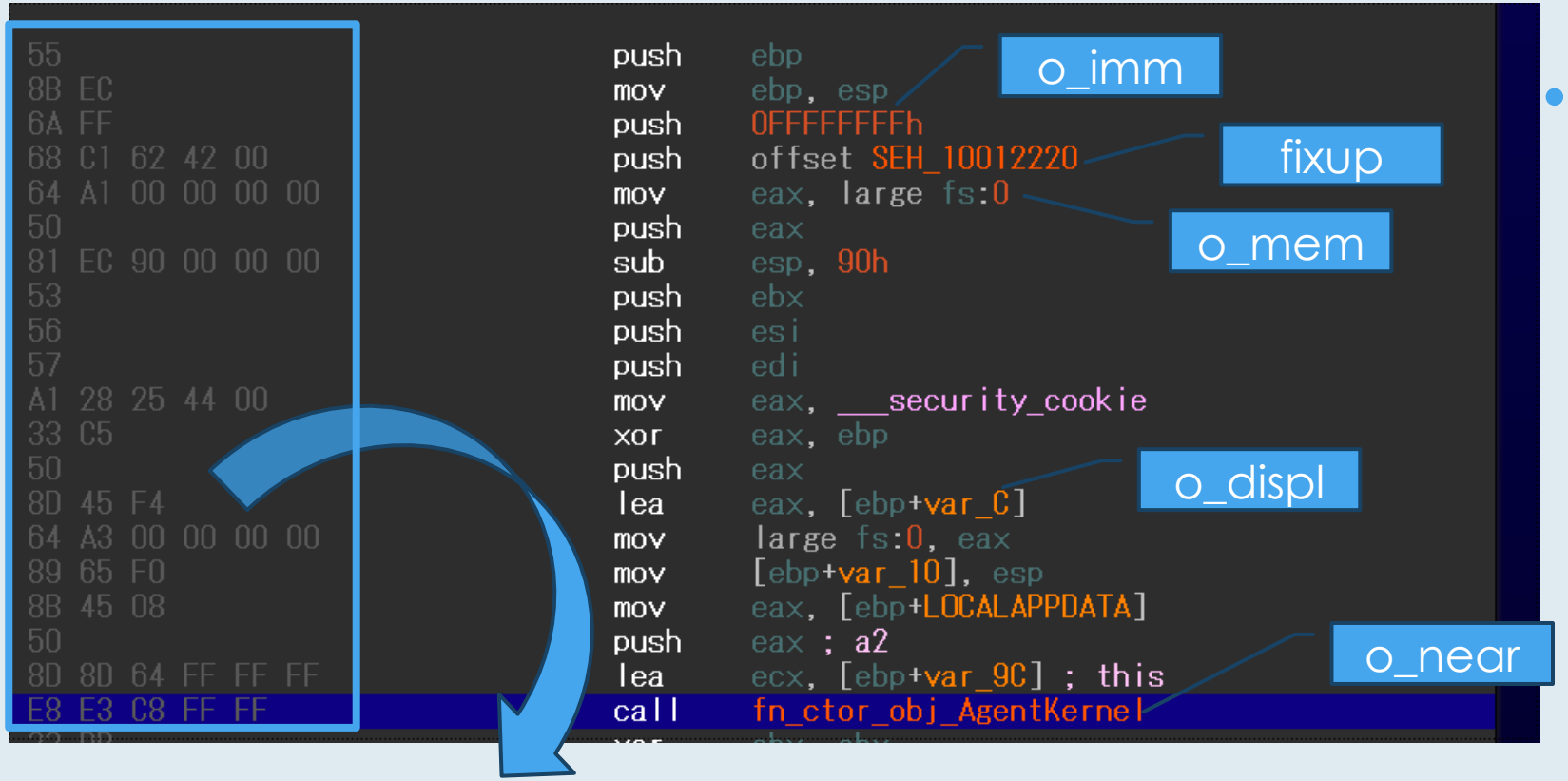
FAKE C2 SERVER FOR VALIDATION

- Developed a fake C2 server to validate the request data of the PoC scanner and other recent samples
  - fakenet (IP diverter) + Python HTTPS server

POST request validation

```
[*] Validating Sec-Dest..
[+] Prefix number 0x95 is valid
[+] The hash of the random bytes b'xbsYpB' matches 88
[*] Validating Sec-Site..
[+] Prefix numbers 0x7f/0x8e is valid
[+] victim_id='F4EB6EF3A8882016'
..
[+] The decrypted POST data is saved as dec_post_data.bin
[*] Responding with PlugX custom header data.. (C2 command = 0x7002)
```

HUNTING RECENT SAMPLES



- VT-retrohunted using [yara_fn]

```
{ 55 8B EC 6A ?? 68 ?? ?? ?? ?? 64 A1 ?? ?? ?? ?? 50 81 EC ?? ?? ?? ??
  53 56 57 A1 ?? ?? ?? ?? 33 C5 50 8D 45 ?? 64 A3 ?? ?? ?? ?? 89 65 ??
  8B 45 ?? 50 8D 8D ?? ?? ?? ?? E8 }
```

HUNTING RECENT SAMPLES (CONT.)

```
GET / HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
Pragma: no-cache
Accept: */*
User-Agent: Mozilla/5.0 (compatible; MSIE 8.0; Wi
Sec-Dest: 94XwswWS
Sec-Site: 9983B2FA880D12F9FCC6
Host: 149.104.12.64

HTTP/1.1 200 OK
Content-Type: application/octet-stream
Content-Length: 32

d148942488930ea29fa07c4c0c88fbedPOST / HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
```

- One of the rules hit the latest sample in Dec last year
  - **CFF was not applied** to the sample
- The C2 included in the sample was active ☺
  - I could check the Content-Length and the format of the GET response

APPROACH BASED ON VALIDATION

- All recent samples had exactly the same C2 protocol encryption and data format
  - Every sample's C2 protocol/port is HTTPS/443
- No need to send the POST request after handshake
  - The C2 likely responded without content until commands are specified by operators
- I started to implement a scanner just checking the difference between GET requests with/without the authentication headers

TLS HANDSHAKE ISSUE

- OpenSSL caused an internal error during the TLS handshake

```
* TLSv1.0 (OUT), TLS header, Certificate Status (22):
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS header, Unknown (21):
* TLSv1.2 (OUT), TLS alert, internal error (592):
* error:0800006A:elliptic curve routines::point at infinity
* Closing connection 0
curl: (35) error:0800006A:elliptic curve routines::point at infinity
```

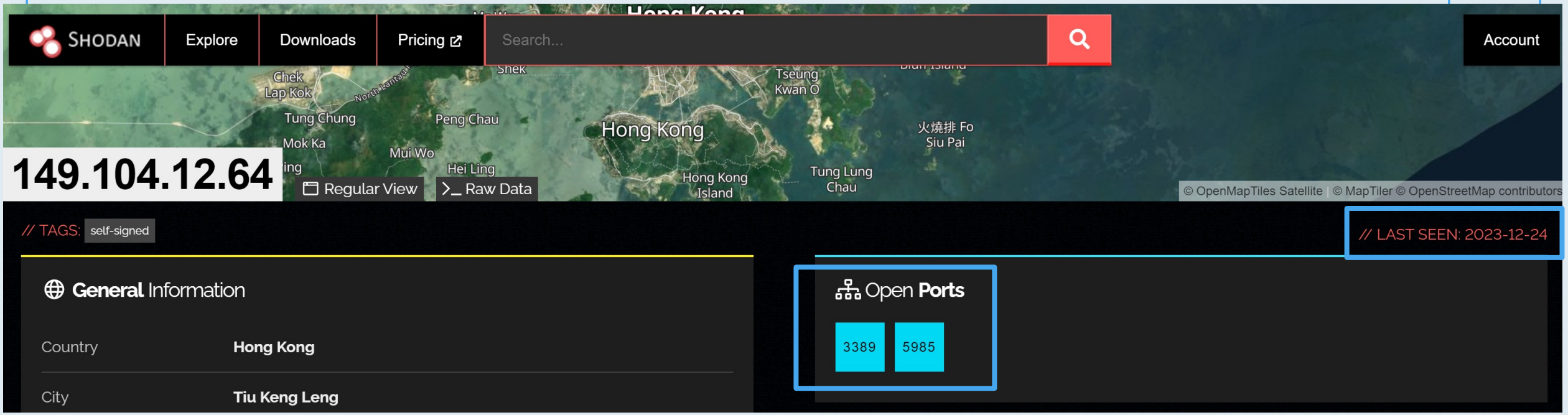
TLS HANDSHAKE ISSUE (CONT.)

- I tested major open source TLS clients
  - Only LibreSSL (pylibtls) worked for the TLS handshake

	OpenSSL	**Mbed TLS (python-mbedtls)**	**wolfSSL (wolfssl-py)**	**LibreSSL (pylibtls)**
Tested version	1.1.1k, 3.0.2, 3.2.0	2.28.6	5.6.0	3.8.2
Worked?	No	No	No	Yes

DETECTION BY THIRD PARTY SCANS

- Shodan haven't been able to recognize the port since at least last Dec
- Censys can detect the port but the protocol is UNKNOWN (not HTTPS)



INTERNET-WIDE SCANNING WORKFLOW

- Automate with Python (Use asynchronous I/O for OpenSSL/JARM scans)
- Exclude as much as possible before the pylibtls scan

- Get the list of hosts open at TCP/443

- Try TLS handshake
- Cause an internal error?

- Match the JARM fingerprint value of the Hodur C2?

- GET request with/without auth headers
- Get a RC4 key-like string only when sending with the headers?

RESULT

- Two C2 servers were found late last December
  - 149[.]104.12.64 and 45[.]83.236.105
- Two months later, Trendmicro referred to the C2s in the blog
- But they are still active

APT & Targeted Attacks

Earth Preta Campaign Uses DOPLUGS to Target Asia

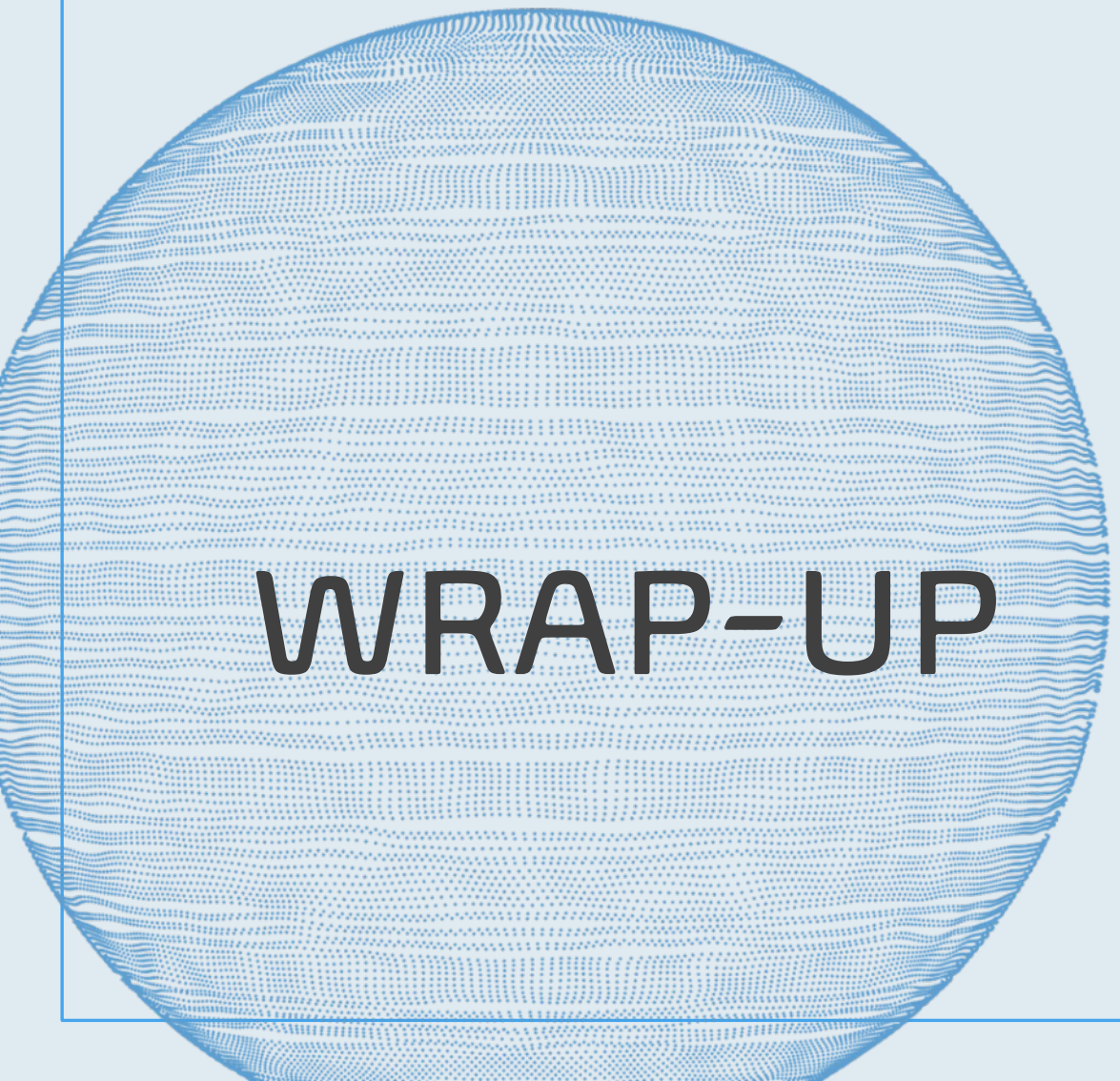
In this blog entry, we focus on Earth Preta's campaign that employed a variant of the DOPLUGS malware to target Asian countries.

By: Sunny Lu, Pierre Lee
February 20, 2024

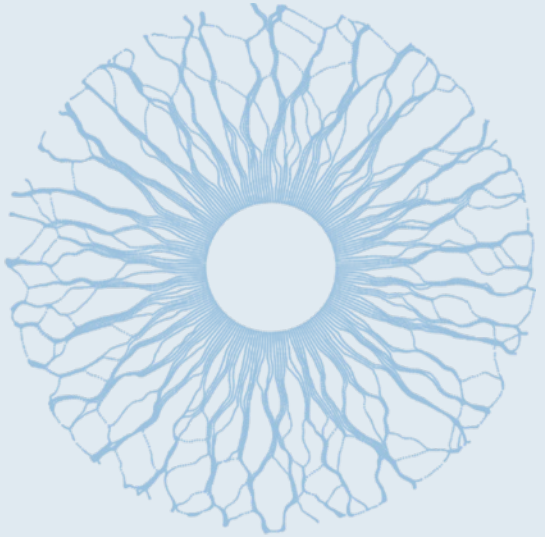
DEMO

```
[*] Authentication headers generated. checksum='ygflkF', victim_id='70FA7450D3323310'
```

```
[D] 45.83.236.105: OpenSSL internal error. Calculating JARM..
[D] 149.104.12.64: OpenSSL internal error. Calculating JARM..
[D] 45.83.236.105: The JARM value matched with Hodur C2
[*] 45.83.236.105:443: No content when sending a query without auth headers
[*] 45.83.236.105:443: RC4 key "a44a9424879f0bd6eaa1094779f889eb" returned when sending a query with auth headers
[+] 45.83.236.105, active, RC4 key = a44a9424879f0bd6eaa1094779f889eb
[D] 149.104.12.64: The JARM value matched with Hodur C2
[*] 149.104.12.64:443: No content when sending a query without auth headers
[*] 149.104.12.64:443: RC4 key "a44a9424879f0bd6eaa1094779f889eb" returned when sending a query with auth headers
[+] 149.104.12.64, active, RC4 key = a44a9424879f0bd6eaa1094779f889eb
[*] new servers found (2 in total)
```



WRAP-UP



WRAP-UP

- Defeating compiler-level obfuscations is easier than before
  - 2-3 months for APT10 ANEL -> 3-4 weeks for Hodur
  - We still need to improve or create tools when RE requires de-obfuscating code precisely
    - Code will be available online after the conference
- The developed scanner keeps tracking the malware C2s on the Internet
  - We can respond proactively using the intel