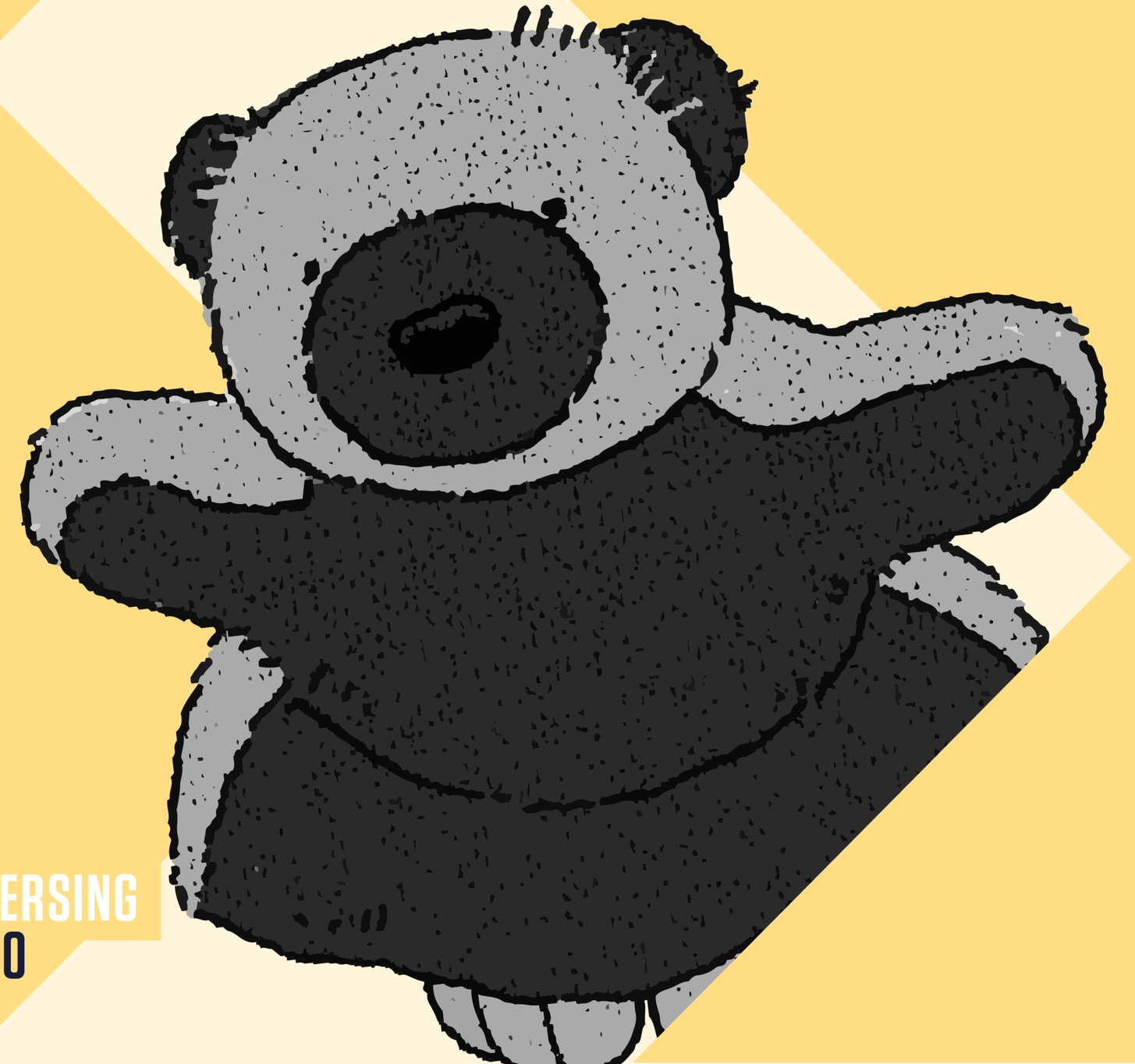

REVERSING 2020

SESSION

HILKO BENGEN

IT Security Expert,
Transportation & Logistics

RULES AS CODE: A LOOK AT THE YARA COMPILER'S OUTPUT

This talk

- Given a compiled YARA ruleset, how easy is it to reconstruct the individual rules?
- Strings/regular expressions seem easy...
  ...but what about the condition syntax?
- Possible improvements to YARA itself
- Warning: Some C and assembly included

Rule Compiler

```
yarac -d filename="XXX" -d filepath="XXX" [...] \
    apt_aa19_024a.yar apt_agent_btz.yar apt_alienspy_rat.yar [...] \
    compiled.yac
```

What happens inside the `yr_compiler_add_*` functions?

- Single–pass compiler, driven directly from code in `libyara/grammar.y`
- Builds a large data structure in memory that hangs off a `YR_RULES` struct.
- Aho-Corasick automaton variant for string/regex/hex–pattern matching.
- Rule names, meta information, tags, string names, external variables are stored as–is.
- Conditions are compiled into a single bytecode program.

Arena

- Custom memory allocator, used to build up and store `YR_RULES`
- Pointer relocation for data structures and some operands in the bytecode program
- Used to load/save compiled from/to disk...
  ...or any streaming reader/writer implementation.

Scanning

```
yara -d filename=cmd.exe [...] -C compiled.yac /path/to/cmd.exe
```

What happens inside the `yr_rules_scan_*` or `yr_scanner_scan_*` functions?

- Execute multi-pattern matcher, record matches, offsets
- Interpret and run bytecode program.
  The bytecode program collects pattern match results and is responsible for marking rule matches. Without it, none of the string matches matter.
- Read file or process memory via `YR_ITERATOR`
  - Possibly multiple times—once for pattern matching and on-demand by the bytecode program
  - May slow down process memory scanning considerably
  - Streaming operation (stdin or network streams) not possible without storing the stream

Bytecode engine

- Stack-based machine
  - 1 byte opcode
  - Operand: 0/4/8 bytes, depending on instruction
- Distinct memory areas

  stack RW, holds operands, results for basic operations and function calls
  - run-time configurable: `--stack-size`
  - default: 16k * 64bit

  `mem[]` RW scratch memory
  - compile-time configurable, related to loop implementation
  - default: 20 * 64bit

  Arena RO code + static data

  input file RO
  - accessed via `intXX` and `uintXX` functions
  - backed by `YR_ITERATOR`
- Giant switch statement, see `libyara/exec.c:yr_execute_code`

Instruction set (1)

- Arithmetic/logical operations for 64bit int, float values
- String compare
- Data conversions
  - Integer $\rightarrow$ Float
  - String $\rightarrow$ Boolean
- Conditional jumps, relative addresses
- Stack, `mem[]` access
- Lookup of individual string matches, offsets, count
- Counting, grouping string matches: `OF`, e.g.
  - `4 of $str_*`
  - `all of them`

Instruction set (2)

- Module import, initialization
- Iterators
  - Used in `for...in` expressions
  - Setup for array, dict, integer-range, integer-list access
  - Generic `ITER_NEXT` operation
- Direct input file access
- Objects: `YR_OBJECT`
  - Access external variables
  - Access code and data from modules
- filesize
- Mark rule matches
- Check result from results
- Halt instruction

Bytecode engine: Examples

Simple “truthy” rule

```
rule t { condition: true }
```

compiles to:

```
00000000:    INIT_RULE 0x000000000000001b ; rule#0 <t>; next = 0000001b (+27)
00000009:         PUSH 0x0000000000000001
00000012:   MATCH_RULE 0x0000000000000000 ; rule#0 <t>
0000001b:         HALT
```

Simple “falsy” rule

```
rule f { condition: false }
```

compiles to:

```
00000000:    INIT_RULE 0x000000000000001b ; rule#0 <f>; next = 0000001b (+27)
00000009:         PUSH 0x0000000000000000
00000012:   MATCH_RULE 0x0000000000000000 ; rule#0 <f>
0000001b:         HALT
```

Bytecode engine: Examples

String match: “any of”

```
rule s_any {
  strings:
    $a = "foo"
    $b = /(bar|baz|quux)/
  condition: any of them
}
```

compiles to:

```
00000000:    INIT_RULE 0x0000000000000037 ; rule#0 <s_any>; next = 00000037 (+55)
00000009:         PUSH 0x0000000000000001
00000012:         PUSH 0xfffabadafabadaff ; undefined
0000001b:         PUSH 0x00007f53c2218010 ; string <rule#0 s_any>.$a
00000024:         PUSH 0x00007f53c2218048 ; string <rule#0 s_any>.$b
0000002d:           OF
0000002e:   MATCH_RULE 0x0000000000000000 ; rule#0 <s_any>
00000037:         HALT
```

Bytecode engine: Examples

String match: “all of”

```
rule s_all {
  strings:
    $a = "foo"
    $b = /(bar|baz|quux)/
  condition: all of them
}
```

compiles to:

```
00000000:    INIT_RULE 0x0000000000000037 ; rule#0 <s_all>; next = 00000037 (+55)
00000009:         PUSH 0xfffabadafabadaff ; undefined
00000012:         PUSH 0xfffabadafabadaff ; undefined
0000001b:         PUSH 0x00007fad1f3a7010 ; string <rule#0 s_all>.$a
00000024:         PUSH 0x00007fad1f3a7048 ; string <rule#0 s_all>.$b
0000002d:           OF
0000002e:   MATCH_RULE 0x0000000000000000 ; rule#0 <s_all>
00000037:         HALT
```

Bytecode engine: Examples

Modules

```
import "tests"
rule tc {
  condition:
    tests.constants.one < tests.constants.two
}
```

compiles to:

```
00000000:       IMPORT 0x00007fc895815011 ; "tests"
00000009:    INIT_RULE 0x000000000000004b ; rule#0 <tc>; next = 00000054 (+75)
00000012:     OBJ_LOAD 0x00007fc89581501a ; "tests"
0000001b:    OBJ_FIELD 0x00007fc895815020 ; "constants"
00000024:    OBJ_FIELD 0x00007fc89581502a ; "one"
0000002d:    OBJ_VALUE
0000002e:     OBJ_LOAD 0x00007fc89581502e ; "tests"
00000037:    OBJ_FIELD 0x00007fc895815034 ; "constants"
00000040:    OBJ_FIELD 0x00007fc89581503e ; "two"
00000049:    OBJ_VALUE
0000004a:       INT_LT
0000004b:   MATCH_RULE 0x0000000000000000 ; rule#0 <tc>
00000054:         HALT
```

Bytecode engine: Examples

External variables:

```
rule fn {
      condition: filename == "explorer.exe" or filename == "cmd.exe"
}
```

```
00000000:    INIT_RULE 0x0000000000000040 ; rule#0 <fn>; next = 00000040 (+64)
00000009:     OBJ_LOAD 0x00007f695b227025 ; "filename"
00000012:    OBJ_VALUE
00000013:         PUSH 0x00007f695b22702e
0000001c:       STR_EQ
0000001d:        JTRUE 0x0000001a         ; -> 00000037 (+26)
00000022:     OBJ_LOAD 0x00007f695b227046 ; "filename"
0000002b:    OBJ_VALUE
0000002c:         PUSH 0x00007f695b22704f
00000035:       STR_EQ
00000036:           OR
00000037:   MATCH_RULE 0x0000000000000000 ; rule#0 <fn>
00000040:         HALT
```

Bytecode engine: Examples

```
00000000:    INIT_RULE 0x000000000000001b ; rule#0 <t>; next = 0000001b (+27)
00000009:         PUSH 0x0000000000000001
00000012:   MATCH_RULE 0x0000000000000000 ; rule#0 <t>
0000001b:    INIT_RULE 0x000000010000001b ; rule#1 <f>; next = 00000036 (+27)
00000024:         PUSH 0x0000000000000000
0000002d:   MATCH_RULE 0x0000000000000001 ; rule#1 <f>
00000036:       IMPORT 0x00007f81a215f015 ; "tests"
0000003f:    INIT_RULE 0x000000020000004b ; rule#2 <tc>; next = 0000008a (+75)
00000048:     OBJ_LOAD 0x00007f81a215f01e ; "tests"
00000051:    OBJ_FIELD 0x00007f81a215f024 ; "constants"
0000005a:    OBJ_FIELD 0x00007f81a215f02e ; "one"
00000063:    OBJ_VALUE
00000064:     OBJ_LOAD 0x00007f81a215f032 ; "tests"
0000006d:    OBJ_FIELD 0x00007f81a215f038 ; "constants"
00000076:    OBJ_FIELD 0x00007f81a215f042 ; "two"
0000007f:    OBJ_VALUE
00000080:       INT_LT
00000081:   MATCH_RULE 0x0000000000000002 ; rule#2 <tc>
0000008a:    INIT_RULE 0x0000000300000037 ; rule#3 <s_any>; next = 000000c1 (+55)
00000093:         PUSH 0x0000000000000001
0000009c:         PUSH 0xfffabadafabadaff ; undefined
000000a5:         PUSH 0x00007f81a1e5c010 ; string <rule#3 s_any>.$a
000000ae:         PUSH 0x00007f81a1e5c048 ; string <rule#3 s_any>.$b
000000b7:           OF
000000b8:   MATCH_RULE 0x0000000000000003 ; rule#3 <s_any>
000000c1:         HALT
```

Real-world rulesets

- `signature-base` ruleset curated by Florian Roth
- Combined YARA ruleset
  - 3422 individual rules
  - 17241 “strings” patterns
  - 67730 instructions, 429,053 bytes
  - file size: 7,537,707 bytes
- Possible optimizations
  - Deduplicate strings
  - Optionally strip meta information
  - Optionally strip string pattern names
  - Introduce instruction variants with smaller operand sizes

General computation?

- No indirect addressing mode
- No primitives to build proper arrays or strings
- No CALL/RETURN . . . but we could build something similar based on jump-tables in code
- Input/Output
  - We can use the file that is given to YARA as input.
  - Output is harder: Signal via `OP_MATCH_RULE` for one bit per rule. It is practical to output some numbers at best

Output

A putchar function:

```
import "tests"
rule output {
  condition:
    tests.putchar(0x41) and tests.putchar(0x42) and
    tests.putchar(0x43) and tests.putchar(0x44) and
    tests.putchar(0x0a) and true // <- Signal match
}
```

```
$ ./yara output.yar /dev/null
ABCD
output /dev/null
```

```
IMPORT    str_tests
OBJ_LOAD  str_tests
OBJ_FIELD str_putchar
SET_M     0             ; save address
PUSH      0x41          ; "A"
CALL      str_i
POP                     ; ignore return value
PUSH_M    0
PUSH      0x42
CALL      str_i
POP
PUSH_M    0
PUSH      0x43
CALL      str_i
POP
PUSH_M    0
PUSH      0x44
CALL      str_i
POP
PUSH_M    0
PUSH      0x0a
CALL      str_i
POP
HALT
str_tests:
        DATA "tests"
str_putchar:
        DATA "putchar"
str_i:
        DATA "i"
```

Porting C code?

```
main(k){float i,j,r,x,y=-16;while(puts(""),y++<15)for(x
=0;x++<84;putchar(" .:-;!/>)|&IH%*#"[k&15]))for(i=k=r=0;
j=r*r-i*i-2+x/25,i=2*r*i+y/10,j*j+i*i<11&&k++<111;r=j);}
```

Porting C code?

```
main(k) {
  float i, j, r, x, y = -16;
  while(puts(""), y++<15)
    for(x=0;
        x++<84;
        putchar(" .:-;!/>)|&IH%*#"[k&15]))
      for(i=k=r=0;
          j=r*r-i*i-2+x/25,
            i=2*r*i+y/10,
            j*j+i*i<11&&k++<111;
          r=j);
}
```

Acknoledgements

- `@bnbdr`, for some nice security research and exploitation writeups for previous YARA versions:
  - https://bnbdr.github.io/posts/swisscheese/
  - https://bnbdr.github.io/posts/extracheese/
- Florian Roth, for the `signature-base` ruleset: https://github.com/Neo23x0/signature-base
- Ken Perlin for the Mandelbrot program: https://mrl.nyu.edu/~perlin/

Contact information

- E–Mail: bengen@hilluzination.de
- Twitter: `@_hillu`
- Github: https://github.com/hillu
- Tools will be released at https://github.com/hillu/yara-rules-re

Speaker

REVERSING 2020

Q/A

HILKO BENGEN

IT Security Expert,
Transportation & Logistics

RULES AS CODE: A LOOK AT THE YARA COMPILER'S OUTPUT

NEXT SPEAKER

Jo Johnson
Principal
Software Engineer,
Specter Ops