HOW NOT TO SHOOT YOURSELF IN THE FOOT WITH PATTERNS

JO JOHNSON – PRINCIPAL SOFTWARE ENGINEER

SPECTEROPS

BIO

Mostly a software engineer, occasionally a security researcher

4 years working on IDS and NSM tools at Gigamon

~10 years working on anti-virus at Microsoft

@Pyr0joe

AGENDA

- Yara vs other security tools that rely on pattern matching
- Motivations for better Yara patterns
- How Yara matches patterns
- Performance and pitfalls for string patterns
- Other performance tweaks

COMPARISON OF SECURITY TOOLS THAT DO PATTERN MATCHING

YARA

Usually run out of band or after the fact against interesting files and memory dumps. (VT retrohunt, quarantined mail attachments, etc.)

Loads the full file and runs all patterns against all the data.

Significantly more accessible for infosec practitioners than proprietary AV signatures or esoteric IDS\IPS rules

ANTIVIRUS

Run in **real time** on client endpoints with blocking, quarantine and removal of malicious files and processes

Use a mixture of file hash, pattern matching, emulation, unpacking, behavioral analysis and cloud lookups to classify files and processes.

Aggressively stop analyzing files and processes when they are not known bad via caching and limiting initial analysis to subsets of total data.

IDS\IPS

Run network traffic against a series of patterns in near real time (or real time for IPS) looking for malicious or suspicious traffic.

Typically look at most of the data coming to them, with some provisions for stopping monitoring very large network flows

Can limit patterns to specific protocols\networks

Newer software has some file extraction and analysis capabilities that come with additional overhead.

WHAT ABOUT EDR\NEXT-GEN AV?

- Capabilities vary widely
- Some are closer to full AV engines
- Others are similar to Yara or even connect to Yara

WHY WORRY ABOUT YARA PERFORMANCE

- Reduce the impact on endpoints if scanning a large environment
- Virustotal Live Hunt or an in-house equivalent running in near real time
- Less waiting time running new rules against an existing corpus
- Do more with limited hardware budgets

WHY WORRY ABOUT YARA PERFORMANCE

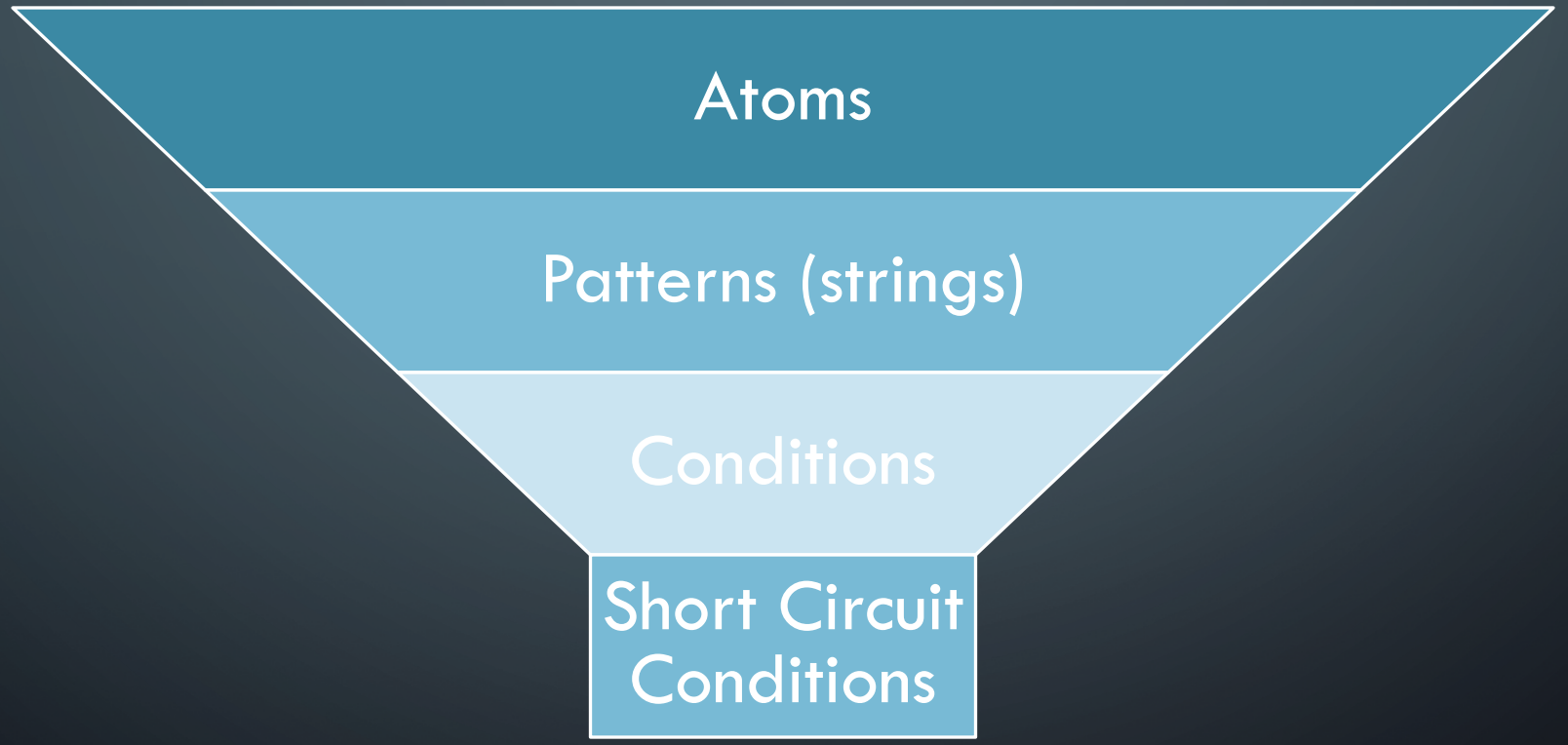
NEAR REAL TIME AND REAL TIME

- Run rules through an AV that supports Yara like ClamAV
  - Realtime blocking, unpacking, and memory scanning against emulation
  - Increased attack surface on deployed machines
- Run limited real time scanning with ProcFilter

  https://github.com/godaddy/procfilter
- Near real time scanning of files extracted by IDS\NSM like Suricata and Zeek

HOW DOES YARA MATCH PATTERNS?

HIERARCHY



HOW DOES YARA MATCH PATTERNS?

ATOMS

- Atoms are substrings of a pattern with a maximum length defined at compile time by YR_MAX_ATOM_LENGTH with a default value of 4
- For each pattern, Yara finds one or more atoms in the pattern that are necessary but not sufficient to match that pattern.
- Yara uses a scoring system to choose atoms that are most likely to be unique

HOW DOES YARA MATCH PATTERNS?

AHO-CORASICK

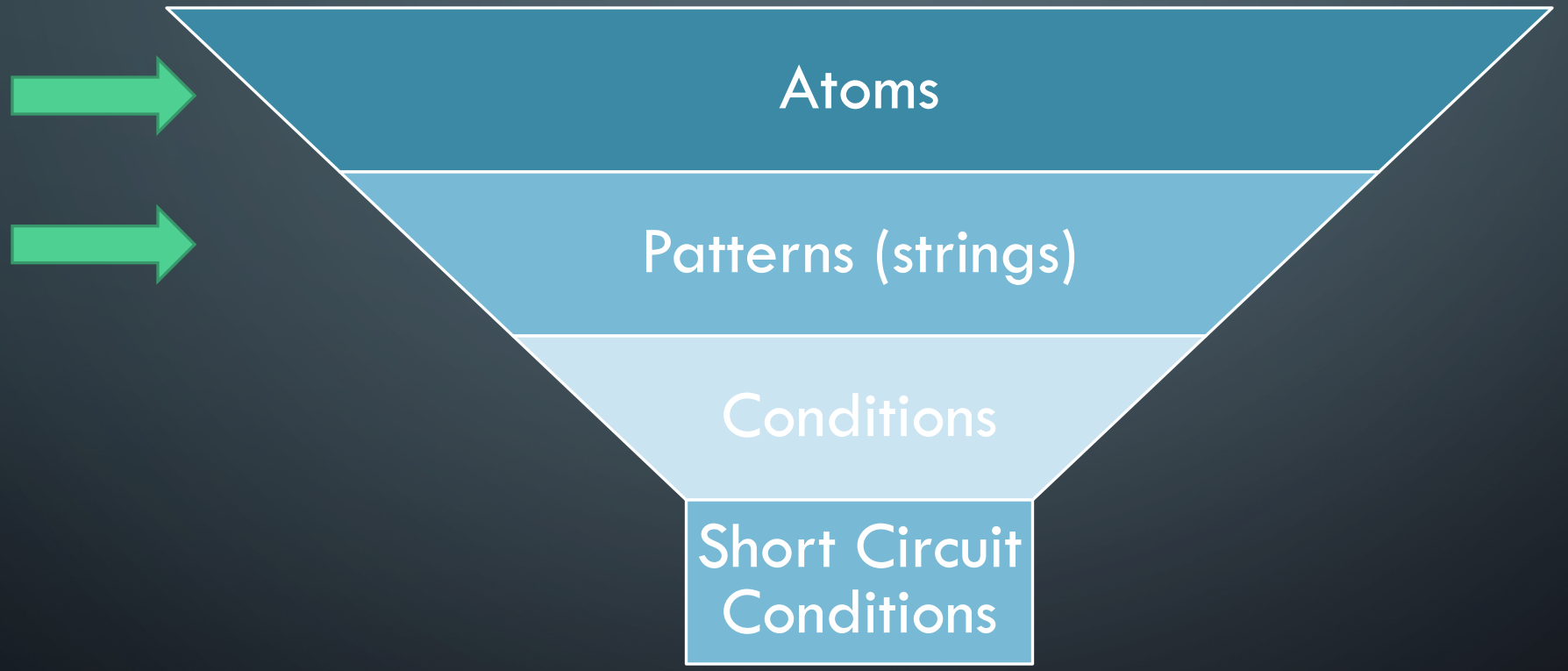
- String matching algorithm for finding matching substrings in a string for a set of smaller strings
- Time complexity is linear in size of the searched string, total length of the set of strings to search for, and the number of matched strings.
- Yara uses this plus the list of selected atoms for the initial scan of data, before attempting to match full patterns

HOW DOES YARA MATCH PATTERNS?

CONDITIONS AND MODULES

- After running all the string patterns, Yara applies the conditions in each rule
- Conditions can be as simple as matching one string, or complicated logic with Boolean operators, loops and functions from imported modules
- Conditions can also do limited matching of content at fixed offsets within a file
- Unlike atoms, not all conditions are executed if a previous statement short circuits and the result of a condition no longer impacts the overall result

PATTERNS


Atoms
Patterns (strings)
Conditions
Short Circuit Conditions

PATTERNS

COMMON PITFALLS

- Regex
- Way too common atom { 00 00 00 00 } or { FF FF FF FF }
  - Yara *WILL* yell at you and VT Live Hunt will kick you out
- Too common atom e.g. { 8B FF 55 8B EC } // win32 dll export preamble
- Strings shorter than 4 characters

PATTERNS

STRING MODIFIERS

- fullword – Does not increase overhead significantly and gives a tighter signature
- Base64 – Will generate 3 atoms per atom for the base string
- wide – reduces the effective value of YR_MAX_ATOM_LENGTH by half. So for 4 byte atoms, wide strings only have 64k possibilities instead of 4 billion
- nocase – Fans out the number of generated atoms by up to 2^YR_MAX_ATOM_LENGTH of 16 at the default

PATTERNS

STRING MODIFIERS - WIDE VS NOCASE TRADEOFF

- For a ruleset with a significant number of wide rules it might make sense to recompile with a value of 8 for YR_MAX_ATOM_LENGTH to get more entropy
- The caveat here is that nocase rules will result in up to 256 atoms for each nocase string
- This might also incur performance penalties on 32-bit processors.

PATTERNS

STRING MODIFIERS - XOR

- The xor modifier will generate 256 atoms for every atom a string would normally generate.
- Xor can inadvertently result in bad atoms
  - 55AA > 00FF and FF00.

```
// Find xor encrypted MBR
strings:
    $error1 = "Invalid partition table" xor
    $cryptmbr = "\x55\x55\xAA\xAA"
xor
condition:
    $error1 and $cryptmbr
```

PATTERNS

WILDCARDS AND JUMPS

- Hex strings offer limited regex like abilities via wildcards and jumps
- Wildcards can work at the nibble level up, so { 4D ?? 9? 00 } is valid
- Wildcards will split apart patterns for what is eligible to be an atom, an atom cannot contain a wildcard.
- Other than the impact on atom selection, wildcards are very fast

PATTERNS

WILDCARDS AND JUMPS

- Jumps are effectively variable length wildcards, but only at the byte level.
  - E.g. { 4D 5A [0-256] 50 45 } // MZ and PE header within 256 bytes can match 4D 5A 50 45 or 4D 5A ??…?? 50 45 as long as there are at most 256 bytes between them
- If an atom matches, jump will need to do one comparison for each byte in the length of the atom, so longer jumps can be expensive if they don't have strong atoms.
- For jumps longer than 200 bytes, including infinite ones, Yara generates an atom on either side of the jump if possible.

PATTERNS

IF YOU REALLY NEED REGEX

- Try to get it working with wildcards and jumps on hex strings first
- Make sure the regex has at least one decent atom and uses a limited length instead of *, like /{"msg": ".{0,1000}"/
- Try to use limited character classes instead of just . if it makes sense like [0-9a-zA-Z]{1,200}

PATTERNS

UNDERSTANDING ATOMS AND MATCHING

- Run Yara with “-sS” to log stats on both atoms and full pattern matches.
- “number of strings” is the atom count
- “number of AC matches” is the total count of atom matches

```
size of AC transition table       : 512
average length of AC matches lists :
1.000000
number of rules                   : 2
number of strings                 : 3
number of AC matches              : 3
number of AC matches in root node  : 0
```

PATTERNS

UNDERSTANDING ATOMS AND MATCHING

- The number of atoms is useful when working with wildcards, jumps and regex to understand what the potential performance overhead will be
- "-sS" also logs what strings matched a given file
- This output is useful for debugging what string is not hitting for a given rule

PATTERNS

LIMITING BY FILETYPE

REGEX (BAD)

```
strings:
    $swf = /^[FCZ]WS/
condition:
    $swf and …
```

STRINGS (OK)

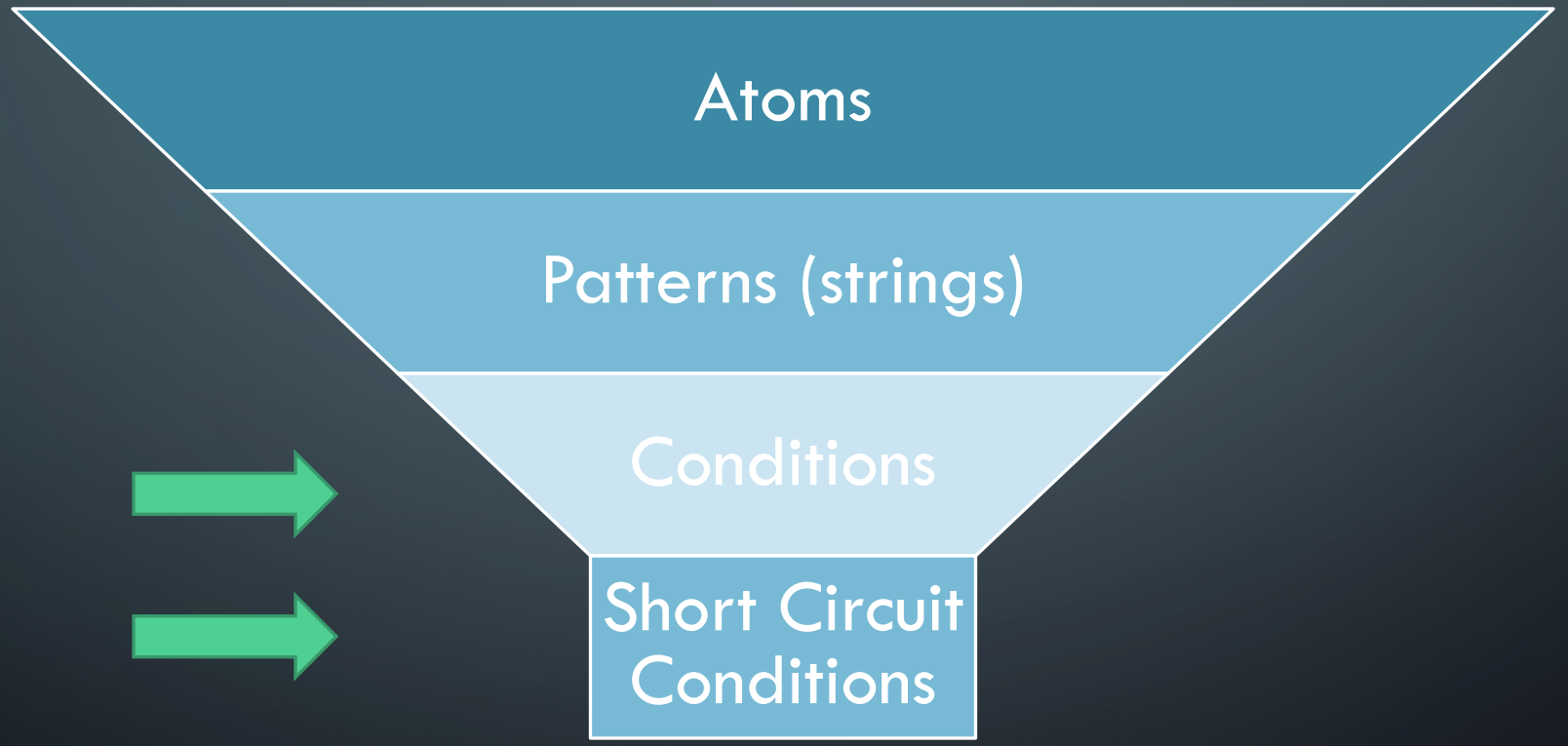
```
strings:
  $swf = “FWS”
  $swc = “CWS”
  $swz = “ZWS”
condition:
  ($swf or $swc or $swz) and …
```

CONDITION (IDEAL)

```
condition:
      /* 'CWS' = '43 57 53' */
      ((uint16be(0x0) == 0x4357 and
uint8(0x2) == 0x53) or
       /* 'FWS' = '46 57 53' */
       (uint16be(0x0) == 0x4657 and
uint8(0x2) == 0x53)  or
       /* 'ZWS' = '5a 57 53' */
      (uint16be(0x0) == 0x5a57 and
uint8(0x2) == 0x53)) and …
```

EXAMPLE FROM HTTPS://INQUEST.NET/BLOG/2018/09/30/YARA-PERFORMANCE

CONDITIONS


Atoms
Patterns (strings)
Conditions
Short Circuit Conditions

CONDITIONS

WHAT IS SHORT CIRCUITING?

- Short circuiting is a term used for Boolean operators that will return once a result is guaranteed, even if they haven't run all expressions.
  - false and dosomething() will return false and never run dosomething because the false expression guarantees that the and operator can never return true
  - true or dosomething() likewise will return true without calling dosomething.
- Short circuiting allows avoiding expensive calls like full file hashes unless they are needed.

CONDITIONS

SHORT CIRCUITING TRICKS – STATIC FILE HASH

- Lots of examples gate a full file hash on just a filetype, but that's expensive
- Can we only hash full files if they are very likely to be what we're looking for?
- What if we hash the first 1k after the pe entry point for every file first?

CONDITIONS

SHORT CIRCUITING TRICKS – STATIC FILE HASH

```
condition:
        (hash.md5(pe.entry_point, pe.entry_point+1024) == "feba6c919e3797e777
8e8f2e85fa033d" and
         hash.md5(0, filesize) == "275876e34cf609db118f3d84b799a790")
        or
        ...
        or
        (hash.md5(pe.entry_point, pe.entry_point+1024) == "9e107d9d372bb6826b
d81d3542a419d6" and
         hash.md5(0, filesize) == "e4d909c290d0fb1ca068ffaddf22cbd0")
```

CONDITIONS

FILE TYPE RECOGNITION

- Conditions are a better place to do file type recognition if the indicator is a small number of bytes at a fixed location.
- Moving the file type recognition to conditions avoids creating and running atoms for each file type you care about.
- That said, the magic library is not on Windows and slow, so it's better to use patterns than call that library.

KEY TAKEAWAYS

- Keep an eye on how many atoms rules are creating and try to limit it when possible, you can't limit which atoms run when
- Watch out for atoms and patterns that hit a large proportion of files
- Use regex and magic sparingly or not at all
- Move simple fixed location byte patterns to conditions
- Order conditions from least to most expensive with short circuiting operators

THANKS!