



TRUST DELIVERED

YARA Rules 101

Learn to Write & Use High-Quality Rules for Threat Hunting & Detection

November 6, 2024

Today's Speakers



Paul Roberts
Director, Editorial &
Content, RL



Danil Panache
Solutions Architect,
RL



Laura Dabelic
Threat Analyst,
RL

RL At-A-Glance

40B+



Searchable Threat Repository

8X



Larger Than Nearest Competitor

60+



Cybersecurity Companies Trust RL

20M



Files Analyzed Daily

FASTEST



Software/File Deconstruction

3M



Malware Identified Daily

300



Employees Globally

CRN



5 Star Rated Partner Program

Gartner



Recognized for SSCS Solution

Verizon Business
DBIR



Report Contributor

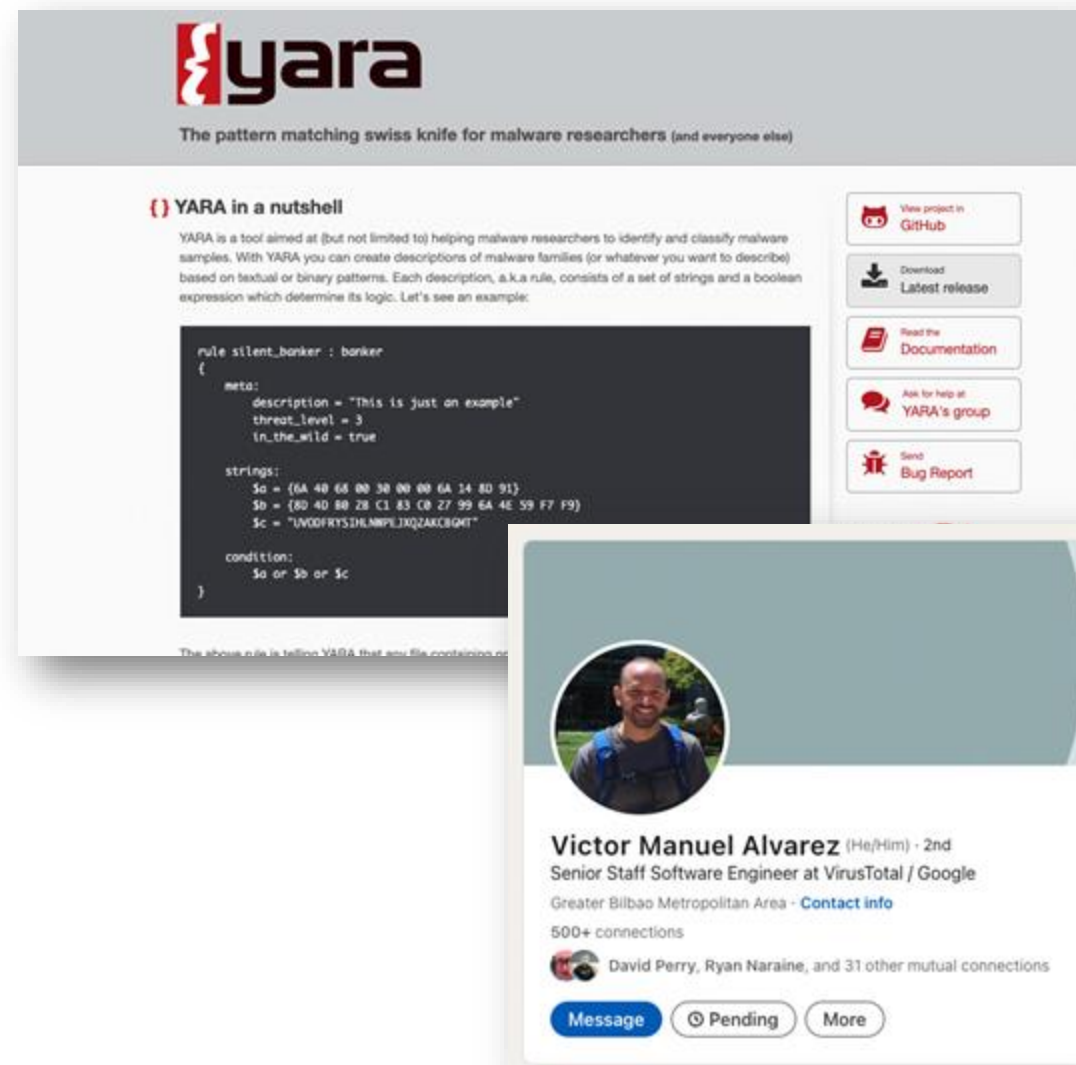
Agenda

- **A brief history of YARA**
 - What YARA rules are (and aren't) good for
- **Applications of YARA Rules**
 - Malware Hunting & Retro Hunting
 - Email Analysis
 - Brand Protection
 - Compliance and DLP
- **What makes a good YARA?**
 - Sources of reliable YARA rules
 - How to evaluate YARA quality
- **How to write and tune your own YARA rules**

A Brief History of YARA

A Brief History of YARA

- **YARA = “Yet Another Recursive Acronym”**
- Developed by Victor Alvarez (@plusvic) at VirusTotal
- Released on GitHub in 2013
 - [virustotal.github.io/yara](https://github.com/VirusTotal/yara)
 - A tool for simplifying the use of textual or binary patterns to detect malware or other threats lurking within IT environments
 - Multi-platform (Windows, Linux, Mac OS X) accessible via CLI or Python scripts (using yara-python extension)
- Scores of security firms, public agencies and individuals/nonprofits generate YARA rules



Applications of YARA Rules

Malware Hunting

YARA rules for Malware Hunting are used in a forward-looking perspective. When files flow through an organization, having the ability to efficiently scan them for potential malicious elements at scale is imperative.

Retro Hunting

YARA rules for Retro Hunting look in the other direction - where have similar files shown up before? This can start with a single sample, sometimes pulled as part of incident response procedure, and is fine-tuned to reduce false positive hits while still encompassing the unique identifiers of potentially malicious elements.

Ability	Example
Check file types with header magic	<code>uint16(0) == 0x5A4D</code> matches PE files
Hunt for process injection APIs	<code>\$sus = "VirtualAlloc" or \$sus2 = "WriteProcessMemory"</code>
Match hex patterns unique to malware	<code>\$hex = { 4D 5A ?? ?? 45 ?? 78 }</code> for specific variants
Use wildcards and ranges for flexibility	<code>{ 4D 5A [4-8] 45 }</code> matches patterns with 4-8 bytes between

Malware Hunting Example

```
rule sample_malware_loader {  
  meta:  
    description = "Basic detection patterns for potential malware"  
    author = "Security Research Example"  
    reference = "For educational and research purposes"  
    date = "2024-11"  
  
  strings:  
    // Common malware string decryption routine  
    $decrypt = { 8B 45 ?? 83 C0 ?? 89 45 ?? 8B 45 ?? 0F BE 00 }  
  
    // Unique API hashing pattern  
    $api_hash = { 33 D2 6A ?? 68 ?? ?? ?? ?? 8B F9 E8 }  
  
    // Common C2 communication patterns  
    $c2_pattern1 = { 68 ?? ?? ?? ?? 68 ?? ?? ?? ?? 68 ?? ?? ?? ?? E8 }  
    $c2_pattern2 = { 8B 4D ?? 83 C1 ?? 51 68 ?? ?? ?? ?? E8 }  
  
    // Known malware export names  
    $export1 = "ServiceMain" fullword  
    $export2 = "DllRegisterServer" fullword  
  
  condition:  
    uint16(0) == 0x5A4D and  
    $decrypt and  
    $api_hash and  
    1 of ($c2_pattern*) and  
    1 of ($export*)  
}
```

Metadata - documents who wrote it, when, what it detects, and why it exists

Assembly code patterns, aligning to how the malware performs decryption, API hashing to resolve Windows API calls, and command/control communication

Function names that malware uses to pose as legitimate

Conditions that need to be met for the rule to trigger

Email Analysis

As ubiquitous as emails are, it's important to scan email content and attachments to detect phishing attempts, malicious documents, and social engineering patterns. Similar to a spam filter but with deeper and more granular inspection capabilities, these can help identify a large array of potential threat vectors.

Ability	Example
Find suspicious sender patterns	<code>\$domain = /microsoft-support.com microsoft.com/</code>
Match urgent language indicators	<code>\$urgent = /urgent immediate action account.*suspended/</code>
Detect malicious attachment types	<code>uint32(0) == 0x464C457F matches ELF files</code>
Look for phishing markers	<code>\$sus = "kindly verify your {username password account}"</code>
Spot credential harvesting forms	<code>\$form = /<input.type=[""]password[""].>/</code>

Email Analysis Example (strings and condition)

strings:

```
// Suspicious sender patterns
$fake_domain1 = //g[0o]{2}g[1]e[- ]?(?:docs|security|verify|auth)/ nocase ascii wide
$fake_domain2 = /paypal-{0,1}secure\.[a-zA-Z]{2,3}/ nocase
$fake_domain3 = /app[1]e[- ]?(?:id|verify|secure|support)/ nocase
```

```
// Urgent language patterns
$urgent1 = "immediate action required" nocase
$urgent2 = "account suspended" nocase
$urgent3 = "unusual activity" nocase
$urgent4 = "security alert" nocase
$urgent5 = /login attempt from .{1,20} location/ nocase
```

```
// Common phishing phrases
$phish1 = "verify your identity" nocase
$phish2 = "confirm your account" nocase
$phish3 = "update your payment" nocase
$phish4 = /click\s+here\s+to\s+(verify|confirm|update)/ nocase
```

```
// Suspicious form elements
$form1 = "<input" nocase
$form2 = "type=\"password\"" nocase
$form3 = "name=\"card" nocase
$form4 = "cvv" nocase
```

```
// Social engineering keywords
$social1 = "gift card" nocase
$social2 = "winning" nocase
$social3 = "inheritance" nocase
$social4 = "invoice" nocase
```

Commonly spoofed domains
to appear legitimate

Common language patterns
to denote urgency, frequently
a tactic in email attacks

condition:

```
// Main logic to detect suspicious emails
(
  // Suspicious sender with urgent language
  (any of ($fake_domain*) and 2 of ($urgent*))
  or
  // Phishing attempts
  (2 of ($phish*) and 2 of ($form*))
  or
  // Social engineering attempts
  (2 of ($social*) and any of ($urgent*))
)
and
// Size constraint to avoid false positives on tiny files
filesize > 200
```

filesize limitations to
avoid potential false
positives

Brand Protection

Much like the e-mail use case, preventing bad actors from impersonating your brand and reputation for their own machinations can be solved via YARA rules as well. This is often most powerful when combined with email rules as well, as emails are a very common avenue for unauthorized use of company assets (such as logos, products, and company name).

Ability	Example
Detect logo color patterns	<code>\$brand_blue = { 00 84 E9 } for specific RGB values</code>
Find company name variations	<code>\$name = /(acm[e3] @cme acc?me)./ for "acme"</code>
Match product identifiers	<code>\$product = /Phone\s*(1[2-4] 1[2-4]\s*Pro)/ for Phone models</code>
Spot counterfeit markers	<code>\$sus = /authentic.*[5-9][0-9]%.*discount/ for suspicious pricing</code>
Look for domain squatting	<code>\$domain = /acme-(support service store)./ for fake Acme domains</code>

Brand Protection Example (strings and condition)

strings:

```
// Company name variations (example using "Acme")
$brand1 = "Acme" nocase
$brand2 = "4cme" nocase // replacing the "a" with "4"
$brand3 = /[@A]cm[e3]?/ nocase // Catches Acme,@cm,acm3, etc.

// Domain squatting patterns
$domain1 = /acme[-.]?(store|shop|outlet)\./ nocase
$domain2 = /acme[-.]?auth(orized|entic)\./ nocase
$domain3 = /official[-.]?acme/ nocase

// Product identifiers
$product1 = "RoadRunner Pro" nocase
$product2 = "Coyote Traps" nocase
$product3 = /Acme\s*Rockets/ nocase

// Counterfeit indicators
$counterfeit1 = /authentic.*acme.*([5-9][0-9]|100)%.*off/ nocase
$counterfeit2 = /wholesale[^.]{1,30}acme/ nocase
$counterfeit3 = "factory direct" nocase
$counterfeit4 = "replica" nocase

// Logo detection - specific RGB values
$logo_color1 = { 00 00 00 } // ACME Black
$logo_shape1 = {89 50 4E 47 0D 0A 1A 0A [12-40] FF FF FF [2-8] 00 00 00 [4-12] 00 00 00 [8-24] 00 00 00 [2-8] FF FF FF
```

condition:

```
// Main detection logic
(
  // Domain squatting detection
  (any of ($brand*) and any of ($domain*))
  or
  // Counterfeit product detection
  (
    any of ($product*) and
    2 of ($counterfeit*)
  )
  or
  // Logo abuse detection
  (
    $logo_shape1 and
    $logo_color1 and
    @logo_color1 < @logo_shape1 + 1000
  )
)
and
// Avoid tiny files
filesize > 500
```

Compliance/DLP

DLP is a logical use case for YARA rules, although other methods and techniques exist that can technologically supersede YARA for detection. These rules can look for sensitive information patterns such as credit card numbers, SSNs, API keys, etc.

Ability	Example
Match PII patterns	<code>\$ssn = /[0-9]{3}-[0-9]{2}-[0-9]{4}/</code> for SSN format
Find API and access keys	<code>\$aws_key = /AKIA[0-9A-Z]{16}/</code> for AWS keys
Detect credit card numbers	<code>\$cc = /[0-9]{12}(?:[0-9]{3})?/</code> for Visa cards
Spot healthcare data	<code>\$medical = ^bICD-(?:9 10)b.{0,20}[A-Z][0-9]{2}.?[0-9]?/</code> for ICD codes
Identify source code leaks	<code>\$code_secret = /(password secret key)s*=\s*["']["'^"]{8,}["']/</code>

Compliance/DLP Example (strings and condition)

strings:

```
// PII Detection
$ssn = /[0-9]{3}[-\s]?[0-9]{2}[-\s]?[0-9]{4}/
$email = /[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,6}/
$phone = /(\+|d{1,2}\s)?(\d{3})?\d{3}[\s.-]\d{4}/

// Financial Information
$credit_card1 = /4[0-9]{12}([0-9]{3})/ // Visa
$credit_card2 = /5[1-5][0-9]{14}/ // Mastercard
$credit_card3 = /3[47][0-9]{13}/ // American Express
$bank_account = \b[0-9]{8,12}\b/

// Healthcare Information
$medical_record = \bMRN\s*[:=#]\s*[0-9]{6,12}\b/
$icd_code = \bICD-(9|10)\b.{0,20}[A-Z][0-9]{2}\.?[0-9]?/
$hipaa_terms = /(PHI|Protected Health Information|HIPAA)/

// API Keys and Credentials
$saws_key = /AKIA[0-9A-Z]{16}/
$api_key = /(api[_-]key|api[_-]secret|access[_-]token)\s*[:]["']?[0-9a-zA-Z]{32,}["']/
$private_key =
/(BEGIN|END)\s+(RSA|DSA|EC|OPENSSH)\s+PRIVATE\s+KEY/

// Internal Document Markers
$confidential = /(CONFIDENTIAL|INTERNAL USE ONLY|DO NOT SHARE)/
$project_code = /Project[_\s]+(Name|ID):\s*[A-Za-z0-9_]{3,}/

// Source Code Secrets
$hardcoded_secret = /(password|secret|key)\s*=\s*["']?["']?[8,}["']/
$config_file = /(config\.json|\.env|\.ini|\.cfg)$/
```

condition:

```
// Trigger on combinations of sensitive data
(
  // PII Combination
  (any of ($ssn, $email, $phone) and $confidential)
  or
  // Financial Data
  (any of ($credit_card*) and $bank_account)
  or
  // Healthcare Data
  (any of ($medical_record, $icd_code) and $hipaa_terms)
  or
  // Code and Credentials
  (any of ($saws_key, $api_key, $private_key) and $config_file)
  or
  // Multiple PII in same file
  (2 of ($ssn, $email, $phone, $credit_card1, $credit_card2, $credit_card3))
)
and
// File must be big enough to be a document
filesize > 50 and
```

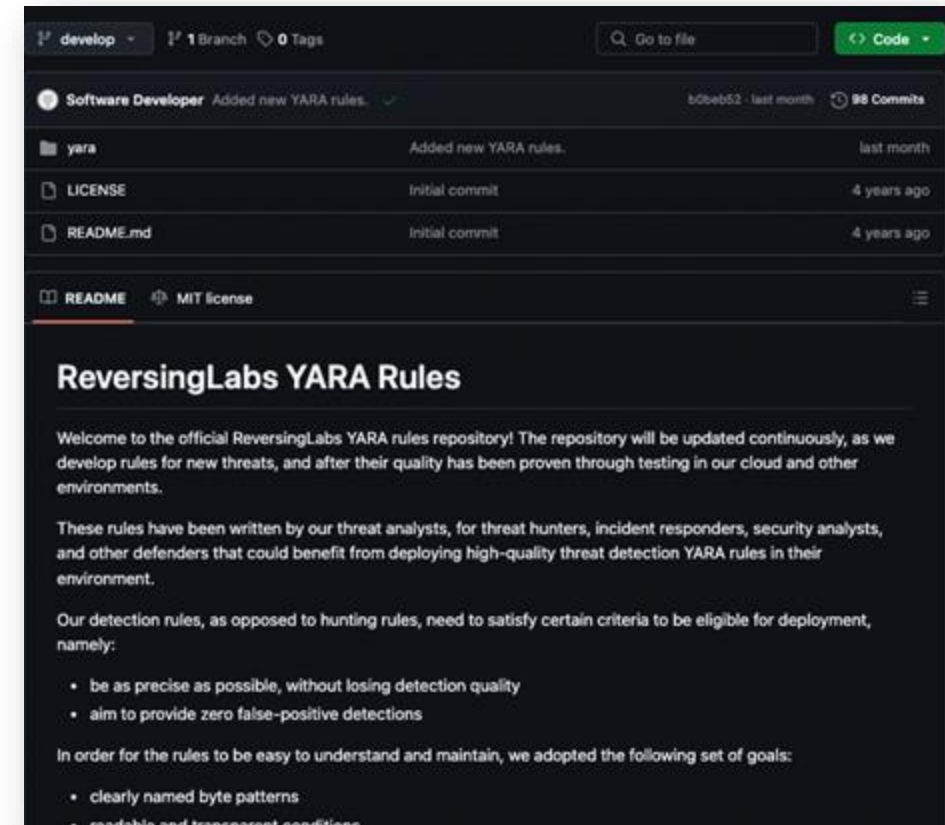
What Makes a Good YARA Rule?

Evaluating YARA Quality

- YARA rules are used for static analysis, but malware is malleable
- Work better on threats with consistent structure
- Susceptible to polymorphic and dynamic threats
 - Use of GO and RUST languages complicate YARA detection
- Need a talented YARA author
 - Simply copying and pasting strings (section headers from .py files) or byte code isn't enough
 - Brittle YARA rules will break with any code change
- YARA “noise” (false positives) also a big concern
- Creating your own YARA rules takes practice, dedication
 - Big dividends in terms of threat protection and readiness

ReversingLabs YARA Rules

- ✓ Written by RL threat analysts, threat hunters, etc.
- ✓ Detection rules focused on precision (zero false positives)
- ✓ Common goals:
 - Clearly named, extensive byte patterns (20+ lines)
 - Readable, transparent conditions
 - Match up to unique malware functionality
 - Code byte patterns preferred over strings
- ✓ Work with YARA version 3.2.0 or greater



Other Reliable Sources of YARA Rules

ReversingLabs also recommends YARA rules from the following sources:

→ Cybersecurity and Infrastructure Security Agency (CISA)

US-CERT

→ <https://yaraify.abuse.ch/yarahub/>

→ <https://github.com/InQuest/awesome-yara#rules>

→ <https://valhalla.nexttron-systems.com/>

→ <https://github.com/Neo23x0/signature-base>

Writing (and Tuning) Your Own YARA Rules

QUESTIONS

A large, vibrant red 3D question mark is the central focus, standing prominently on a dense field of black 3D dollar signs. The dollar signs are scattered across the entire background, creating a textured, almost chaotic surface. The lighting is dramatic, with strong highlights and deep shadows, emphasizing the three-dimensional nature of the symbols.

YARA Resources



Open Source YARA Rules

LEARN MORE



Find 100+ YARA Rules on GitHub

CHECK THEM OUT



How to Write YARA Rules for Malware Detection

GET A VIRTUAL TOUR

<https://www.reversinglabs.com/open-source-yara-rules>

<https://github.com/reversinglabs/reversinglabs-yara-rules>

<https://www.reversinglabs.com/blog/writing-detailed-yara-rules-for-malware-detection>

Upcoming RL Virtual Events



RL REVERSINGLABS WEBINAR

Exposing Software Supply Chain Weaknesses

Featuring Guest Speaker
IDC's Katie Norton

LIVE: NOV 20, 12PM-1PM ET

Katie Norton
RESEARCH MANAGER,
DEVSECOPS & SSCS, IDC
IDC



RL REVERSINGLABS VIRTUAL EVENT
NOV 21, 9-10:30AM ET | 2-3:30 UK

Global Perspectives on Software Supply Chain Security

Exploring current and emerging regulations in the US, UK & EU

FEATURING INSIGHTS FROM THE

National Cyber Security Centre

pwc

<https://www.reversinglabs.com/webinar/webinar-line-up>



Thank You