Chronicle

WHITEPAPER

YARA-L: A New Detection Language for Modern Threats

SUMMARY

Most enterprises use a SIEM to analyze security data, to detect threats and investigate incidents. However, most leading SIEM products were created more than a decade ago, and were designed for a different world. Today, the threat landscape and IT environment looks quite different: Data generated in petabytes, not terabytes; a mature public cloud infrastructure; new technologies such as EDR that generate useful but massive amounts of telemetry; and threats such as fileless malware that are either ephemeral or silent and persistent.

We believe a new approach is required, and Chronicle offers that with YARA-L for threat detection. It is powerful, easily understood, and designed for threat analysis, not data query.

INTRODUCTION

Today, security operations and detection/response is largely about visibility and data — security telemetry data. Telemetry data allows us to know what is happening in our ever-expanding environments and ultimately enables much-sought-after situational awareness. While many assume that this means security naturally derives from having large amounts of data, telemetry on its own doesn't provide visibility. More important is that data can be used for detection, alert triage, response, threat hunting, and other tasks.

In essence, this means it's not just the quantity of telemetry, but also the quality — consistency, structure, fidelity — of data that creates a clear signal about the attacker's present and past behaviors. In fact, if more data translates into more noise, it is of no help. If more data translates into the user needing to run more searches, it is again of no help. If more data means that you have to pay dramatically more without a clear increase in security insight, it is, again, of no help...

The other critical component, and the one least under a security vendor's control, is visibility. Visibility is composed of the various windows of insight which generate telemetry from within an enterprise network. These may be traditional log sources, network and endpoint sensors and other data. Typically, aspects of visibility are limited to discrete points of view and therefore offer only a partial picture of activity occurring within an enterprise environment (for example, having only firewall logs will give you a skewed picture most of the time). This "pre-selection" is typically a side effect of rate-based or quota-based pricing, combined with infrastructure with limited scale, making the ability to store all the necessary data logs unaffordable or impossible.

But storage is only part of the problem. Success in enterprise defense hinges on accessibility of security data. Without a mechanism to access log data, no amount of storage will make an organization safer if said organization cannot slice and dice or otherwise get at the stored data.

Chronicle

REQUIREMENTS FOR BETTER DETECTION

So how does Chronicle solve the critical challenges of data and visibility? In three ways:

- It offers defenders the opportunity to maximize the visibility data they can access, by not requiring pre-selection of data sources.

- Our predictable pricing model (per employee per year) allows defenders to avoid sacrificing aspects of their visibility due to cost of storage or access.

- Under the hood, Chronicle offers massive amounts of compute power allocated to data manipulation operations and queries.

Indeed, successful detection and investigation tools are first about being able to collect and retain visibility data, without incurring an inordinate cost. Building a platform that can scale to petabytes is not that difficult in this public cloud age. However, creating such a platform that would not cost millions for nearly every organization is dramatically more difficult.

The Chronicle platform has two capabilities that enable superior detection:

1. **Structured data (organized via our Unified Data Model, or UDM)**—this means that both rules and algorithms will run reliably and detect cleanly using any data collected by the platform. This includes logs, EDR records, network traffic metadata (from Zeek and friends) and other telemetry—much more than a traditional SIEM tool.

2. **Automatically-enriched data (connected to DHCP, DNS, threat intelligence and other context sources)** — this means that the detection logic will apply intelligently and not via a blunt text match (e.g. we can match a domain threat indicator to a log that has only an IP address in it...consider how this might be done in a SIEM, if at all).

From many available approaches, it would be tempting to predict that Chronicle's detection would rely on machine learning or AI, given Google known strengths in the area of Artificial Intelligence (AI). While machine learning may help with revealing anomalies and detecting some of the unknowns, in most cases defenders do know something about a threat's nature or its behavior. What is needed is an easy and effective way to channel this knowledge into detections. This means a new approach that utilizes rules and other detection methods is needed.

A good detection engine allows an enterprise to build specific rules for atomic observables (like a "known bad" file hash or a specific registry key change), but should also allow for the creation of more general rules to support the study and identification of potential TTPs. By allowing for this continuum of logic, security teams can create detections for key observables such as IOCs and for attacker tradecraft such as specific activity sequence or process execution patterns. Enabling both scenarios is critical to the success of both a detection rule engine and the defenders' security posture.

CAN YOUR RULES DO THIS?

While detection rules might not sound particularly novel, Chronicle is adding a new model for detection logic (rules, later models, etc) that has:

- Broad coverage of security telemetry such as logs, endpoint / EDR data, network traffic metadata, and other telemetry
- Petabyte scale for both real-time and historical detections, by running in the cloud on Google infrastructure
- A Unified Data Model (UDM) under the hood, for enabling consistent data quality and accessibility
- Smart matching logic that enables the system to match to enriched data "automagically" and be more effective than text string matching
- Focused detection language that is designed for threat detection, not merely for querying the data
- Capability to mix and match rules and machine learning models at a later time.

This approach will deliver additional benefits such as being easy to understand and test (a challenge that plagues algorithmic and especially ML-based threat detection approaches). The world has enough inscrutable 50-line SQL queries as it is. In fact, to enable collaborative detection across the industry, the detection logic needs to be easy to read, review, package and share. Of course, that does not mean that anybody can detect an advanced threat using this approach without any training — some hard security problems are...hard. It does mean that reading a rule will allow anybody to understand how the detection will work — hence this approach is easier to teach and explain to build security talent at an organization.

The Chronicle detection model will support both simple detections (such as string matching rules for file names or registry keys) and complex logic (such that may look like an actual script or a program). For analysts who require more, the approach is extensible and modular. New components can be written and connected to rules and detections to enable new operations in the future. For example, if you reverse a DGA algorithm, you can then drop the logic into a detection rule.

The approach will be open and public, and mapped to existing frameworks such as MITRE ATT&CK and Sigma.

We call our approach YARA-L because it is inspired by YARA — invented by Google's VirusTotal team, for malware analysis and applied to logs (hence the "L") and other security telemetry inside the Chronicle platform.

What is important is that YARA-L is a language to express detections, not merely to query the data in order to eventually use said data for detection. ***In essence, it is a threat detection language, not a data query language!*** It is designed by and for security analysts, admittedly with some help from malware reverse engineers (who created the original YARA).

Understanding the success of YARA is essential in developing a strategy for nurturing the success of this new detection language, YARA-L, broadly, and within the Chronicle detection engine specifically.

How is YARA-L used with Chronicle (today and in the future)?

1. Define real-time detection rules
2. Define historical detection rules of arbitrary complexity
3. Perform advanced (hunting-style) searches

While the cases 2 and 3 are similar, 2 implies a scheduled rule run leading to an alert while 3 is about interactive exploration of data by a human analyst.

YARA-L SCENARIOS

What are some scenarios where we think this approach works uniquely well?

- You hear about a new piece of malware that uses a particular registry key for persistence
- You analyze a new threat and realize that it uses a Powershell launching with a hidden window parameter
- You have a need to expand coverage of a particular MITRE ATT&CK technique in your environment
- You want to cast a wide net of looking for types of newly popular suspicious activities such as access to browser cookie and credential files
- You need to watch for specific activities that are necessary for the attacker to take in order to steal particularly valuable data

YARA-L shines in all of these scenarios and while other approaches may help with some of them, it is hard to find an approach that works universally. First, a traditional SIEM product with UI-based rule configuration lacks flexibility and hence the coverage of use cases, coupled with typically slower performance. Next, a search-based tool will require a layer of additional data analysis to deliver the detections. Finally, a programming language-based approach (e.g. detection rules in Python or Go, as some elite organizations use) would be massive overkill for non-programming SOC analysts and other security professionals. We believe that YARA-L is the "just right" approach for the threat detection problem today.

Detection needs continue to shift and grow as organizations realize legacy product solutions are not the sole answer to security. As investments are made into human analysts, threat responders, and hunters, we have the opportunity to introduce YARA-L as both a microscope and wide angle lens to defenders' toolkits. We will support an investigative mindset that allows for ready on-boarding and application of threat intelligence, analyst insights, and investigative techniques.

Thus far, the collective industry has failed to develop a unified solution and remains unable — in most cases and/or without extreme reliance on uniquely talented people — to effectively use procedural-based detectors derived from the study of attacker TTPs. YARA-L is an opportunity to add a fresh approach while also providing Chronicle customers a powerful mechanism to detect threats and intuitively pursue and iterate upon security investigations.

Chronicle

SOME YARA-L EXAMPLES

Example 1 - CLI Magic: The example below relies on regex match to a Windows command line, on data from an EDR or sysmon. Today, many Chronicle customers send EDR and sysmon data into the platform. Note that not every EDR has detections for all the threats, hence such post-processing of EDR data with YARA-L does deliver value. A traditional SIEM is not likely to even have a field for a command line arguments, by the way.

```
profile susp_powershell_download_file  {
    meta:
            author = "Chronicle Security"
            description = "Rule to detect PowerShell one-liner to download a file"
            version = "0.01"
            created = "2019-12-16"
            reference = "https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Windows%20-%20
                    Download%20and%20Execute.md"

    condition:
        if re.regex(strings.lower(udm.process.command_line), ".*powershell.*net.webclient.*") then
            outcome.match()
        end   }
```

Example 2 More EDR-ing: This is another simple rule that runs on EDR (or sysmon) data and relies on command line matching. It showcases conditions for grouping and flexible matching to several variables such as command line and process path.

```
profile susp_process_with_variation_of_svchost  {
  meta:
            author = "Chronicle Security"
            description = "Rule to detect process paths or command line execution of files with variations on svchost"
            version = "0.01"
            created = "2019-12-16"

  function:
    func CheckSvchostVariations()

        if (
          re.regex(strings.lower(udm.process.command_line), ".*(svchOst|svhOst|svhost|svchst|svchot|svchostexe)\.exe.*") or
          re.regex(strings.lower(udm.process.path), ".*(svchOst|svhOst|svhost|svchst|svchot|svchostexe)\.exe.*")
          ) then
        return true
      end
      return false
    end

  condition:
      if ( CheckSvchostVariations() )

      then
          outcome.match()
      end   }
```

Example 2 - More EDR: This is another simple rule that runs on EDR (or sysmon) data and relies on command line matching. It showcases conditions for grouping and flexible matching to several variables such as command line and process path.

```
profile susp_process_with_variation_of_svchost {
        meta:
                author = “Chronicle Security”
                description = “Rule to detect process paths or command line execution of files with variations on svchost”
                version = “0.01”
                created = “2019-12-16”
        function:
          func CheckSvchostVariations()
                if (
                        re.regex(strings.lower(udm.process.command_line), “.*(svch0st|svh0st|svhost|svchst|svchot|svchostexe)\.exe.*”) or
                        re.regex(strings.lower(udm.process.path), “.*(svch0st|svh0st|svhost|svchst|svchot|svchostexe)\.exe.*”)
                  ) then
                  return true
                  end
                  return false
                  end
        condition:
                if ( CheckSvchostVariations() )
                then
                outcome.match()
                end }
```

Example 3 - Registry Mess

This rule focuses on registry monitoring. Windows event logs or EDR data are the most likely source for this. The rule mixes event types with specific field values to detect interesting registry operations.

```
profile mitre_T1198_registry_modification_to_trusted_provider_list {
   meta:
      author = “Chronicle Security”
      description = “Detection for registry changes keys associated with Trusted providers”
      reference = “https://attack.mitre.org/techniques/T1198/”
      version = “0.01”
      created = “2019-12-13”

   function:
      func ProviderListRegChange()
         if ( (udm.metadata.event_type == “REGISTRY_MODIFICATION” or udm.metadata.event_type == “REGISTRY_CREATION” ) and
            (udm.target.Registry.registry_key == “HKLM\\SOFTWARE\\Microsoft\\Cryptography\\OID” or
            udm.target.Registry.registry_key == “HKLM\\SOFTWARE\\WOW6432Node\\Microsoft\\Cryptography\\OID” or
            udm.target.Registry.registry_key == “HKLM\\SOFTWARE\\Microsoft\\Cryptography\\Providers\\Trust” or
            udm.target.Registry.registry_key == “HKLM\\SOFTWARE\\WOW6432Node\\Microsoft\\Cryptography\\Providers\\Trust”) )
         then
            return true
         end
         return false
      end

   condition:
      if ( ProviderListRegChange() ) then
            outcome.match()
      end    }
```

Example 4 - Too Late? Better Late Than Never

The rule below looks for ransomware dropping a ransom note, with the goal of detecting before more machines are infected or the ransomware hits open shares on the network. This may come from a wide range of data sources, centered on an endpoint (again, EDR and sysmon, as well as Windows logs - in some cases).

```
profile ransomware_ryuk_ransomnote_created {
   meta:
      author = “blevene”
      description = “Identify when a Ryuk Ransomware ransomnote has been written.”
      version = “0.01”
      created = “2019-12-16”

   condition:
      if (udm.metadata.event_type == “FILE_WRITE”
         and ( strings.to_lower(udm.target.file) = “RyukReadMe.html”)
            or
          strings.to_lower(udm.target.file) = “RyukReadMe.txt” )
      then
         outcome.match()
      end    }
```

Example 5 -Detection Choices

This rule perhaps does not have magic, but it does showcase a few more functions of the language. If you have multiple ways to detect something but want to channel them all into one detection as a result, here is an example.

```
profile malware_powershell_empire
{
   meta:
      author = “Chronicle Security “
      description = “Detection activity related to the OWAAuth malware”
      reference = “https://attack.mitre.org/software/S0072/”
      version = “1.2”
      created = “2019-12-13”
      updated = “2020-01-20”

   function:
      func ScheduledTaskSet()
         if re.regex(strings.to_lower(udm.principal.process.command_line), “.*schtasks .*/tn updater.*”) then
            return true
         end
         return false
      end

   func WritePayload()
            if re.regex(strings.to_lower(udm.principal.process.command_line), “.*sal a new-object;iex\\(a io\\.streamreader\\(\\(a io\\.compression\\.deflat-
estream\\(\\[io.memorystream\\]\\[convert\\]::frombase64string\\(.*\\),\\[io\\.compression\\.compressionmode\\]::decompress\\)\\),\\[text.encod-
ing\\]::ascii.*”)
         then
            return true
         end
         return false
      end

   condition:
      if ScheduledTaskSet() or WritePayload() then
            outcome.match()
      end
}
```

Example 6 - More Malware

This one matches more endpoint logs across multiple fields. Of note, this rule demonstrates the YARA origin of the language here. Note that all of the fields are matched via regexs, which isn't required, but you have that choice.

```
profile malware_win_dropper_sload  {

      meta:
      author = "Google Cloud Security"
      description = "Detection for sLoad dropper marker files"
        reference = "https://www.microsoft.com/security/blog/2019/12/12/multi-stage-downloader-trojan-sload-abuses-bits-almost-exclusively-for-malicious-activities/"
      version = "1.1"
      created = "2019-12-16"
      updated = "2020-01-28"

   function:
            func Marker()
              if udm.metadata.event_type == "FILE_CREATION" and re.regex(strings.to_lower(udm.target.file.full_path), ".*\\_in\\$")
              and re.regex(strings.to_lower(udm.principal.process.command_line), ".*powershell.*")
              then
                        return true
              end
              return false
            end

   condition:
      if Marker()
      then
            outcome.match()
      end
}
```