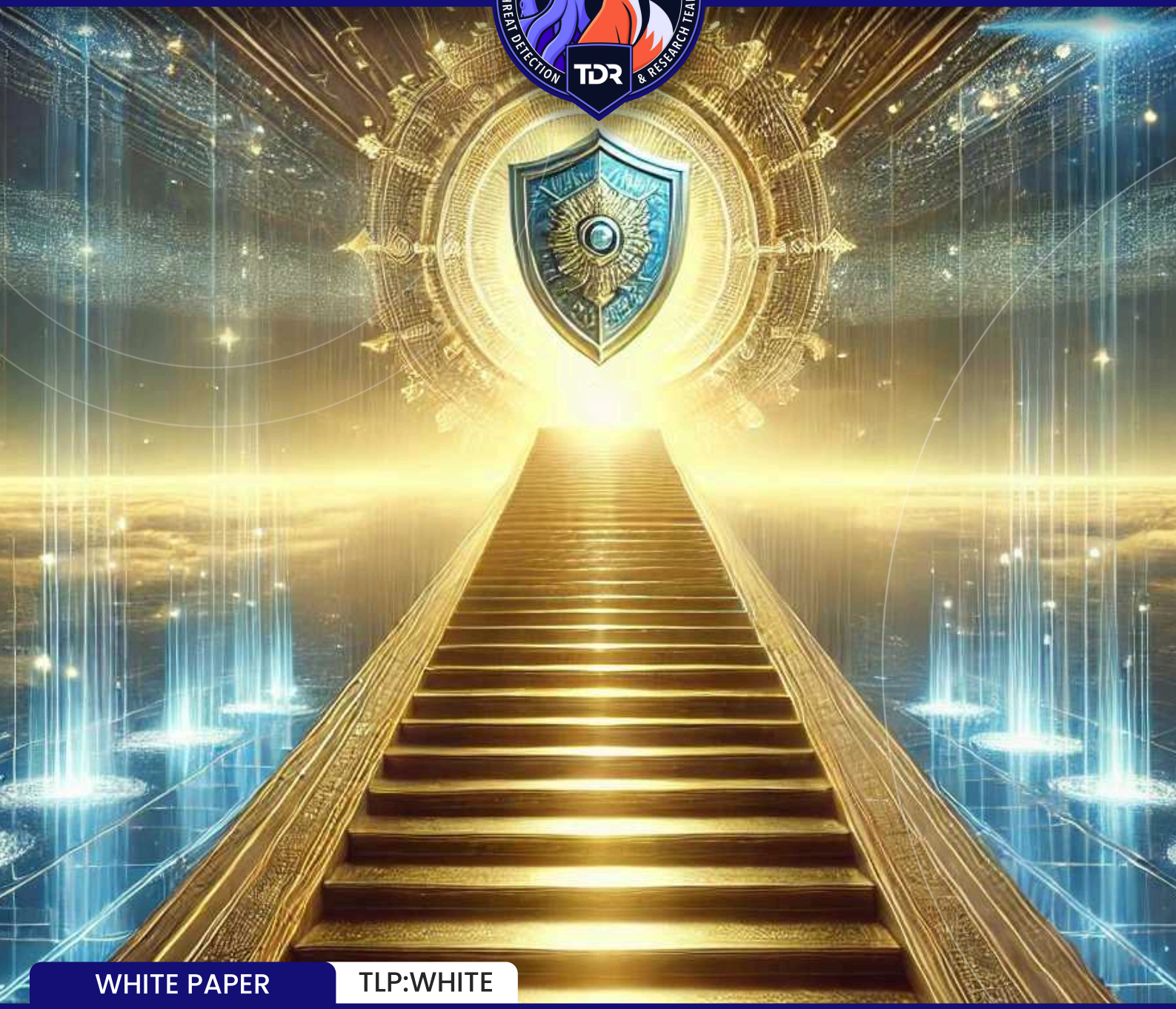



WHITE PAPER TLP:WHITE

# Detection engineering at scale: one step closer

TDR team, February 2025

sekoia

# Table of Contents

# Introduction

Security Operations Center (SOC) and Detection Engineering teams frequently encounter challenges in both creating and maintaining detection rules, along with their associated documentation, over time. These difficulties stem largely from the sheer number of detection rules required to address a wide range of technologies.

In this article Sekoia.io presents an approach designed to address these challenges. It introduces our detection approach and some related problems, outlines the regular and automated actions performed through CI/CD pipelines, and highlights the importance of incorporating ongoing monitoring and review into the detection engineering process, despite these measures.

# A Two-Faced issue

The challenges faced by detection engineers today can be viewed as twofold: on one hand, the number and complexity of attacks continue to increase; on the other hand, enterprise environments are expanding, transitioning to hybrid models, and exposing a larger attack surface.

In the first article of this series, we will discuss the challenges at hand and provide practical examples to illustrate them.

## Attackers on the rise

The cyber threat landscape has evolved significantly over the past decade. This evolution has resulted in a dramatic increase in both the number of attackers and the variety of Techniques, Tactics, and Procedures (TTPs) they employ. Fortunately, not every new attacker introduces a unique TTP. Attackers often reuse existing TTPs, which helps streamline the work of detection engineers.

From a defender perspective, it is essential to follow all of these TTPs, and try to detect the most used ones. As an example, the Sekoia.io rules catalog currently lists almost a thousand rules mapped to the MITRE ATT&CK matrix. Still, this does not fully cover the matrix, which is continually evolving, as demonstrated by regular updates to the MITRE ATT&CK framework. Furthermore, attackers frequently seek to avoid detection, for instance by leveraging legitimate binaries, underscoring the importance of this issue. The LOLBAS (Living Off The Land Binaries, Scripts, and Libraries) project, along with other comparable initiatives, provides valuable insights into the scope of this challenge.

An important point for defenders building detection rules for various customers is also that not every rule can be deployed everywhere for a given TTP. A recent example is the ClickFix social engineering tactic. This tactic involves displaying fake error messages in web browsers to deceive users into copying and executing a given malicious PowerShell code, finally infecting their systems. If detection engineers only rely on the common host and network events, it will result in difficulties to build a generic approach and would need some heavily customised filters. For further in depth details, please check our related blogpost.

## Defense all over the place

On the other side, enterprise environments have also changed, with organisations now operating in hybrid setups and utilising a diverse array of products. While these advancements sometimes enhance security and provide access to a greater volume of logs, they also pose significant issues for detection engineers.

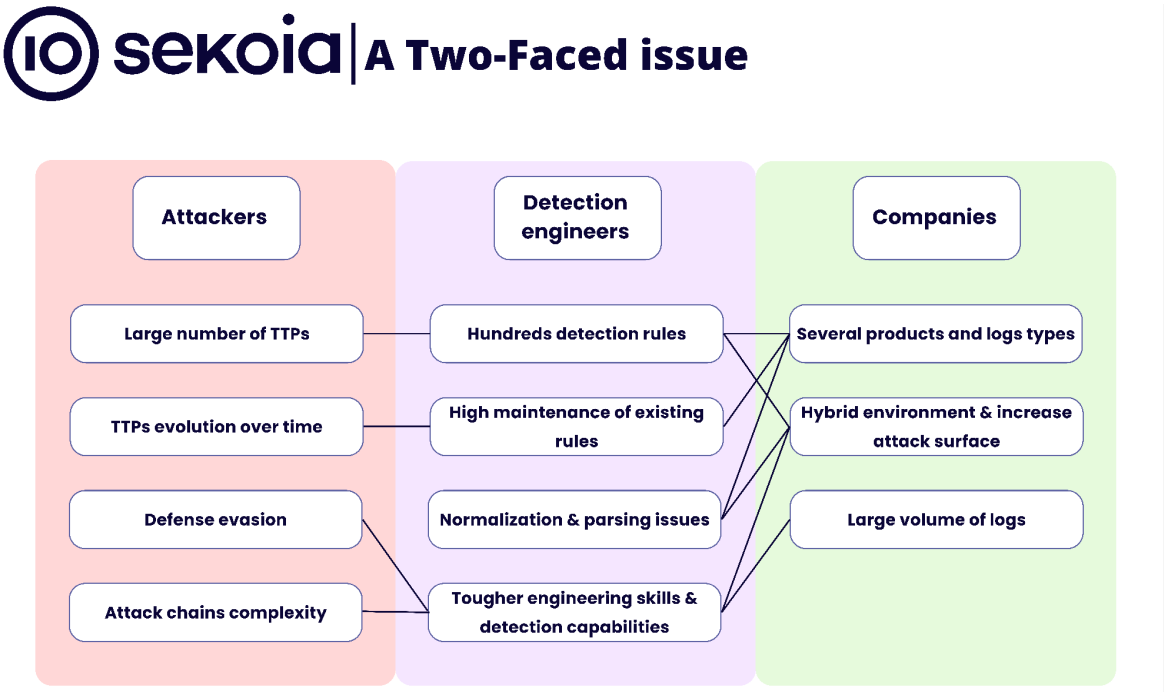
The access to large volumes of logs presents significant challenges in processing data and applying detection rules at scale. Additionally, it becomes increasingly difficult for both detection engineering and integration teams to analyse the logs, parse them, and develop effective detection rules.

The lack of standardisation in the logs generated by hundreds of different products, combined with the sheer volume of data, makes parsing and normalisation particularly difficult. These processes, which have always been critical to detection, have become increasingly complex over time. As a result, detection rules often face issues such as multiple conditions required to handle improperly normalised fields or, more frequently, values.

When it comes to cloud-related detection rules, normalisation often becomes an unattainable goal. Each cloud provider - whether AWS, GCP, Azure, or others - employs unique event structures and fields, making standardisation across providers extremely challenging.

This complexity is one of the primary reasons why many detection engineering teams accumulate hundreds, if not thousands, of detection rules over time. These rules become difficult to manage and nearly impossible to confidently delete, posing maintenance constraints. This situation benefits no one, as it often leads to alert fatigue for SOC analysts, who must contend with frequent false positives triggered by an overabundance of detection rules.

The schema below summarises these two key issues, providing an overview before delving into some concrete examples.




## Practical example

A detection use case we recently faced, that could illustrate our concern, is related to the phishing AiTM (Adversary-in-The-Middle) attack. It is increasingly popular among attackers targeting personal or corporate documents that are stored in a cloud solution.

The typical attack scenario steps, which are also illustrated in this video, are:

- The user receives an email containing a fake Office link to authenticate with its corporate email account.
- The Office link redirects to a typosquatted domain intercepting the access to Microsoft 365 portal, allowing the attacker to catch user login/password and authenticated cookie.
- The attacker re-injects the stolen cookie to access the victim account and exfiltrates its corporate documents.

As outlined in our previous blog post, we chose to use Sigma as our detection language, although this approach can be adapted to other detection languages. Our goal is to create a detection rule that is sufficiently generic to work on various solutions. Starting with Microsoft, we initially developed the following Sigma pattern:

```
detection:
  aitm_source_ip:
    sekoiaio.tags.source.ip: AiTM
    user.email: '*'
  microsoft_365_login:
    event.action: UserLoggedIn
  entra_id_login:
    azuread.category: SignInLogs
  condition: aitm_source_ip and (microsoft_365_login or entra_id_login)
```

This Sigma rule combines a first selection based on our Cyber Threat Intelligence (CTI) enrichment along with two other selections that detect user authentication. As shown in the condition field, the rule has two distinct selections according to the Microsoft product's logs to work with both Microsoft 365 and Entra ID products.

This issue could be resolved by improving normalisation, such as standardising field names and values across both Microsoft 365 and Microsoft Entra ID products but also any other products. By adopting fields based on the Elastic Common Schema (ECS) Reference, we developed the following updated and more generic version:

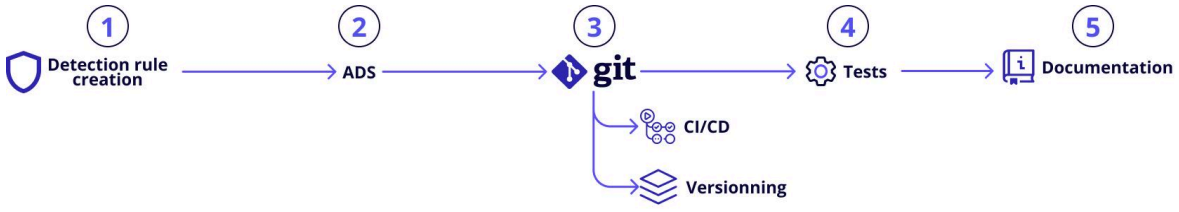
```
detection:
  aitm_source_ip:
    sekoiaio.tags.source.ip: AiTM
    user.email: '*'
    event.category: authentication
  condition: aitm_source_ip
```

This shows that building rules that are "vendor-agnostic" is a constant challenge, especially since sometimes the normalisation is not obvious at first sight for both detection engineers and the team building the logs parsers.

# The catalyst: an approach to detection engineering at scale

Our approach serves as a catalyst to help you scale efficiently with minimal challenges; however, it is not a universal solution to all problems. It demands careful attention and expertise from detection engineers and is designed to align closely with practices commonly employed by developers and DevOps teams.

This methodology is structured around five key steps, as outlined below:




We will now take a closer look at each step.

## Detection rule creation

This may seem like an obvious step, but the details involved are often less straightforward. A detection rule, irrespective of the language used, comprises two key components: the metadata and the detection pattern.

At Sekoia.io, we use Sigma as our detection language, meaning the files are written in YAML. Each file is a detection rule, composed of selections that are evaluated based on a condition defined at the end. While this approach is effective, it poses a challenge: if a detection engineer wants to add a filter to exclude a false positive, the modification will affect the entire detection pattern. This limitation is common across most detection languages, where filters are not separated from the core detection logic. This is a crucial consideration for the "Continuous Tests" section of this blog, as it necessitates testing the syntax and the logic of the detection rule after every single change.

It is important to emphasise that we have deliberately chosen to avoid creating certain types of detection rules that are excessively complex to manage over time and offer relatively low usefulness. Our focus is exclusively on detecting Technics, Tactics and Procedures (TTPs), as we strongly believe these will remain relevant regardless of the specific circumstances. For instance, when a new vulnerability emerges, a detection rule is created only if it is deemed highly critical, such as the case of ZeroLogon in the past.

Similarly, we refrain from creating what we refer to as "IoC detection rules", where a specific file name is flagged for a particular intrusion set. This decision is based on two key considerations. First,

our CTI feed is capable of handling such detections, and we believe it is more effective to manage them in the STIX format. STIX allows us to add validity dates to an indicator, furthermore our CTI feed can be consumed separately, allowing us to protect more customers. Second, from a long-term perspective, these rules tend to be cumbersome to maintain and manage, making them less practical in our approach to scalable detection engineering.

The metadata associated with detection rules are equally as important as the detection patterns themselves, particularly for automation purposes. In the process of creating detection rules, it is highly recommended to carefully consider the metadata based on your specific automation requirements. In essence, ask yourself: what metadata will be necessary for newly created detection rules? Additionally, what information should be maintained or updated with each modification of an existing detection rule?

All this metadata is then used in CI/CD pipelines to automatically generate documentation and testing frameworks, which is described later in this article.

Once a detection rule is created - or more often, concurrently - the detection engineer documents their research in an Alerting and Detection Strategy file.

## Alerting and Detection Strategy Framework (ADS)

This framework, developed by Palantir and slightly adapted to suit our needs, is designed to encourage detection engineers to carefully consider various aspects when creating detection rules and to document their thought process. We will not go into detail about this framework, as it is thoroughly documented here. This approach benefits rule reviewers and provides valuable context for future detection engineers who may need to work with rules that are several years old.

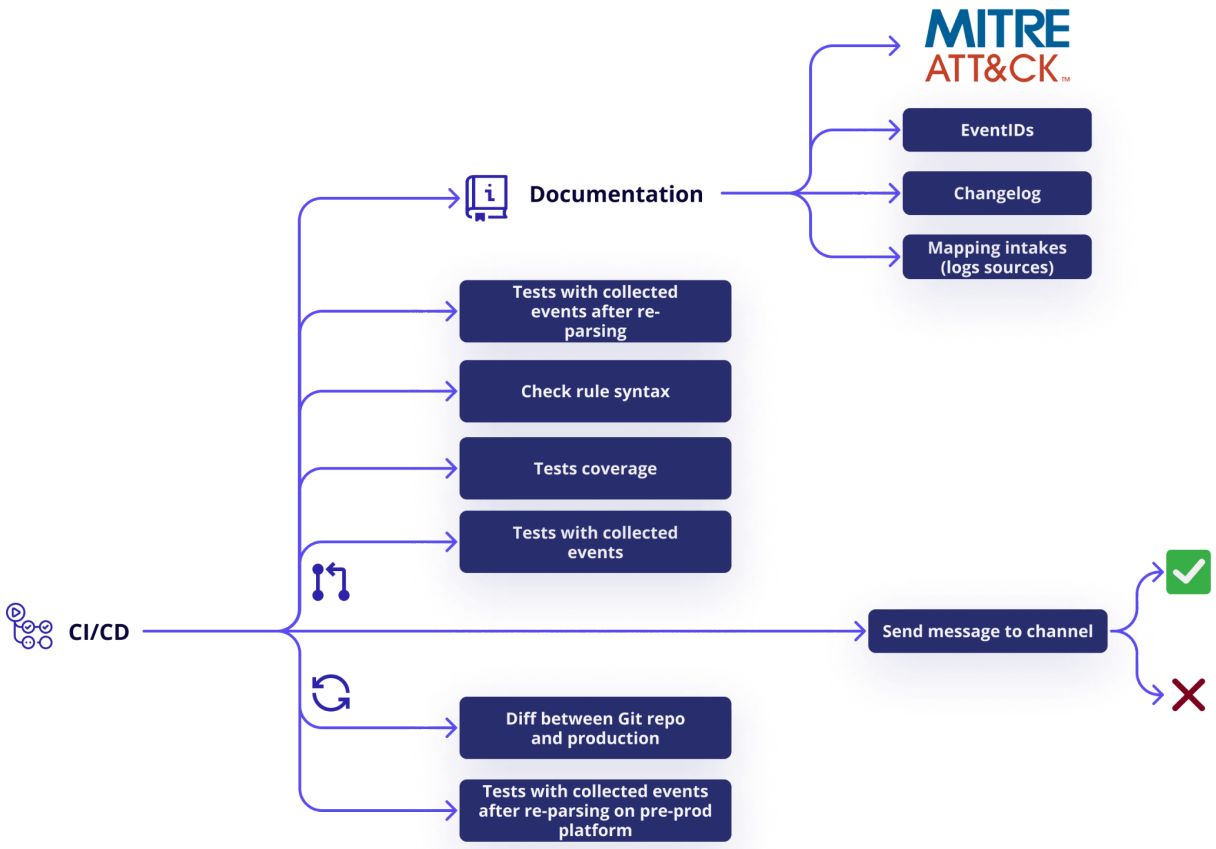
Every aspect of this framework is important, but the Validation and False Positives steps are particularly critical for us.

Detection engineers must thoroughly document how their detection rule was validated. This documentation is essential not only for confirming the rule's functionality but also for enabling others to replicate the validation process. In some instances, particularly with certain products, conducting a lab test may not be feasible. In such cases, it is crucial to specify whether the validation was performed in a reproducible test environment or achieved otherwise.

Documenting observed false positives, including their frequency and the query used to identify them, is equally important. This information is critical for thorough rule reviews prior to moving them into production and remains valuable in the long term. A rule that performs well in year *n* may degrade significantly by year *n+1*. This degradation can result from various factors, such as changes in your customer base and their environments or the introduction of new products in the market that execute previously unobserved actions.

## CI/CD and versioning

To enable detection engineers to scale effectively and sustainably over the long term, CI/CD and versioning are essential. At Sekoia.io, we leverage GitHub alongside GitHub Actions to automate the generation of documentation and conduct continuous testing. The diagram below provides an overview of our CI/CD process:




As shown in the diagram, several tests and the generation of documentation, represented in the upper part of the diagram, are executed only when a new pull request is created, while two specific tests are performed on a daily basis. Finally, we utilise instant messaging to send notifications. This approach aligns with our goal of adopting best practices from development and DevOps.

With this overview in mind, let's delve deeper to understand the reasoning behind these practices.

## Continuous tests

As shown above, several tests are performed during our CI/CD process. To understand the purpose of each one, we think it is better to proceed with some use cases.

## Use case 1: adding a filter to a detection rule

This scenario is a common occurrence among detection engineering teams: a false positive has been identified and needs to be excluded.

Whether the exclusion involves a simple or complex filter, the impact on the detection rule is fundamentally the same. As previously mentioned, any change affects the entire rule pattern, making the first and most obvious step the verification of the rule's syntax to ensure it remains valid on a grammatical level.

Once the syntax has been validated, the next step is to test the logic of the rule. However, before proceeding, it is essential to establish whether the appropriate tests are in place. Much like in software development, ensuring comprehensive test coverage is crucial. Every logical condition of the rule must be tested to confirm that it functions as intended and aligns with the detection engineer's objectives.

For detection engineers using Sigma for instance, simply verifying that the "filter" selection works correctly after adding an exclusion is insufficient. Changes to the condition could potentially break the rule, and inadvertent modifications or omissions could introduce regressions. Thorough testing of both the syntax and logic is therefore imperative.

Once test coverage is confirmed, we proceed to testing the detection rule's logic. This begins by retrieving events—either from our lab, where the detection engineer creates and tests the rule to ensure it triggers an alert, or, if not feasible, to retrieve an event provided by the editor.

However, one significant challenge remains: the test events are captured at the time the detection rule is created, meaning the tests are static and do not adapt over time. This raises a critical question: what happens to our rules if a log parser is modified? Yes, such changes could potentially break the rules.

This leads us to use case number 2.

## Use case 2: log parser update

Fortunately, the original log is preserved within these test events. To address this issue, the original log is replayed against the parsers daily to ensure the rule continues to function as expected.

Ideally, your organisation has a pre-production environment where test events can be replayed against parsers before any changes impact detection rules in production. This is the approach we follow, in addition to replaying these events in the production environment as a precautionary measure.

Here is a summary of all the tests continuously performed through the CI/CD process:

1. Rule Syntax Checking: Ensuring the detection rule is grammatically correct.

2. Test/Code Coverage: Verifying comprehensive test coverage for the rule's logic.
3. Execution of Tests with Real Events: Running tests using real events created by the detection engineer.
4. Replaying Original Logs in Pre-Production: Using the original logs (unparsed message) from the test events to validate them against log parsers in a pre-production environment.
5. Replaying Original Logs in Production: Similar to step 4, but executed in the production environment to ensure continued functionality.

This approach provides a robust and scalable testing framework for your detection rules, seamlessly integrated into the CI/CD pipeline. The trade-off, however, is that detection engineers must exercise meticulous attention to detail, which may require additional time when creating a detection rule.

It leaves one final crucial step in the process: the automatic generation of documentation.

## Automatically built documentation

Most of the documentation needs were enumerated earlier in this article:

- Identifying which detection rules are compatible with which log source.
- Providing a comprehensive MITRE ATT&CK matrix that includes all rules, along with individual matrices for each log source (such as this one for ProofPoint TAP for instance), as some detection rules apply only to certain log sources.
- Listing the EventIDs used by our detection rules, enabling customers to prioritize their configuration efforts accordingly. *These eventIDs are retrieved from the detection rule pattern as well as the "test events".*
- Maintaining a changelog for each rule to keep customers informed of updates. We opted not to rely on Git commits to allow for greater flexibility.

All of this is performed automatically, thanks to the rigorous initial process, which ensures seamless integration with the CI/CD pipeline.

# Do not leave out all the rest

The approach outlined in this article has helped us mitigate the impact of common issues, such as detection rules failing unexpectedly after a rule modification or a parser change. However, some challenges persist, and it remains essential to monitor Key Performance Indicators (KPIs) regularly.

In detection engineering, the most critical KPIs are often the false positive and false negative rates of detection rules. While identifying false negatives is nearly impossible, tracking false positives, although challenging, is feasible and provides valuable insights into rule effectiveness. This challenge is particularly pronounced for companies like ours, operating exclusively as an editor without offering managed services. As a result, the only metrics we receive come from our partners and customers. Precisely, the SOC analyst has the choice between two alerts status, closed or rejected, that could be interpreted respectively as true positive or false positive:


In such cases, several questions arise. For example, how should be categorized a “blocked outgoing HTTP request to a malicious domain”? Is it a false positive, a true positive, or a true positive but benign alert? This scenario highlights the importance of carefully defining alert statuses. However, even with a well-thought-out classification system, human interpretation remains a significant factor.

From our perspective, this situation is always a true positive—it cannot be considered benign, even if the request is blocked, because the request itself indicates an active process within the network attempting to access the malicious domain. However, others might classify it as benign or even as a false positive for many valid reasons. A similar debate can occur in cases where an antivirus quarantines a threat. These differing interpretations underscore the complexity of KPI assessment in detection engineering.

In the end, while our approach can contribute to building a more robust and scalable detection engineering pipeline, continuously monitoring detection rule KPIs and conducting daily reviews of detection rules remain essential practices that should not be left out.

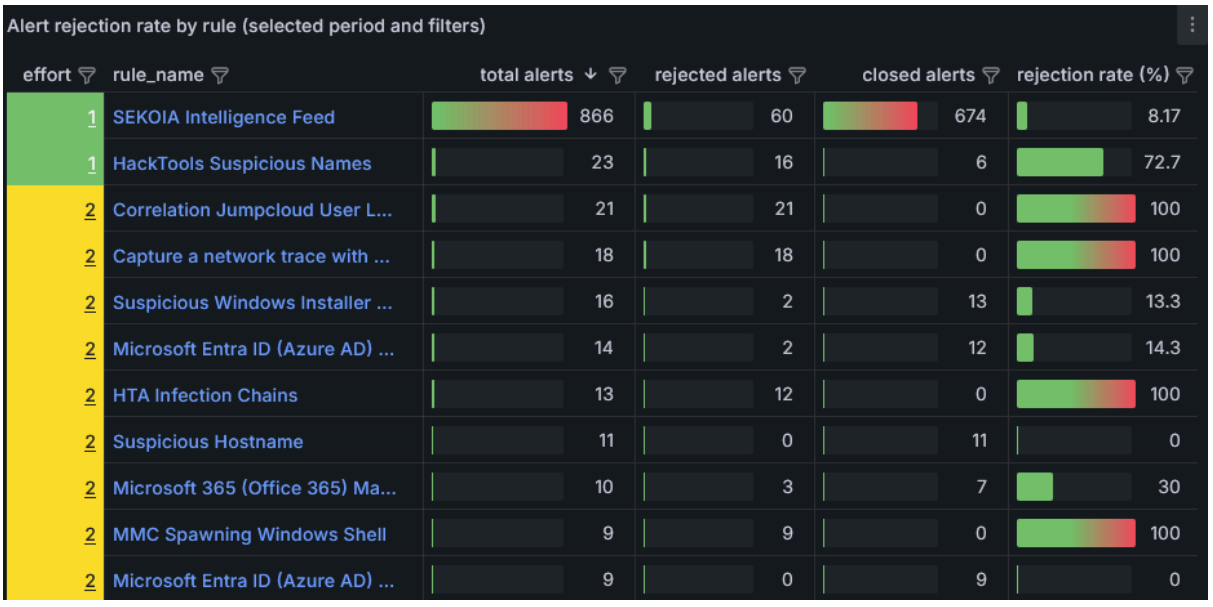
# Monitoring detection rules

Detection engineers always need more telemetry to improve their detection capabilities. In the lifecycle of a detection rule many changes are required to avoid false positives and extend the detection. Having the data is a first step but you need to process it efficiently. Our approach there was to give them both a **proactive** and a **reactive** approach to analyse that data.

For the **proactive approach** we are using the open source tool Grafana, and have built dashboards where the analyst will find:

- alert total / by status / by rate of status for each rule
- alert ratio comparison between the last 7 days and the range time between 7 and 14 days before

The "*alert rejection rate by rule*" dashboard gives some results that need to be interpreted cautiously considering the SOC analyst bias we have written about in the beginning of this article. Nevertheless it could help the detection engineer to easily pivot on the detailed results for each rule and review quickly, in depth, the related alerts and events. Of course some of these rejection rates need to be pondered, as a 100% rejection rate for a small number of alerts may not be representative.




| effort | rule_name | total alerts | rejected alerts | closed alerts | rejection rate (%) |
|---|---|---|---|---|---|
| 1 | SEKOIA Intelligence Feed | 866 | 60 | 674 | 8.17 |
| 1 | HackTools Suspicious Names | 23 | 16 | 6 | 72.7 |
| 2 | Correlation Jumpcloud User L... | 21 | 21 | 0 | 100 |
| 2 | Capture a network trace with ... | 18 | 18 | 0 | 100 |
| 2 | Suspicious Windows Installer ... | 16 | 2 | 13 | 13.3 |
| 2 | Microsoft Entra ID (Azure AD) ... | 14 | 2 | 12 | 14.3 |
| 2 | HTA Infection Chains | 13 | 12 | 0 | 100 |
| 2 | Suspicious Hostname | 11 | 0 | 11 | 0 |
| 2 | Microsoft 365 (Office 365) Ma... | 10 | 3 | 7 | 30 |
| 2 | MMC Spawning Windows Shell | 9 | 9 | 0 | 100 |
| 2 | Microsoft Entra ID (Azure AD) ... | 9 | 0 | 9 | 0 |

The "*temporal comparison*" dashboard provides valuable insights into trends following modifications of a specific rule and, more broadly, facilitates the monitoring of observed threats and their evolution over time. This can help identify cases where further research may be necessary for a given TTP. Additionally, it is important to detect any sudden decrease in the number of alerts generated by a rule, as this may be a weak signal of potential detection failures related to issues such as formatting, data collection, or parsing.

| rule_name | alert_nb_ratio | diff_n_min_7d_and_n7_min_14d | nb_alert_n7_min_14d | nb_alert_n_min_7d |
|---|---|---|---|---|
| Telegram Bot API Request | 95 | -94 | 95 | 1 |
| SentinelOne EDR Threat Mitigation Report Kill Success | 11 | -10 | 11 | 1 |
| Palo Alto Cortex XDR (EDR) Alert Not Blocked (Medium Severity) | 10.9 | -69 | 76 | 7 |
| Suspicious Kerberos Ticket | 9 | -9 | 9 | 0 |
| Fortinet FortiGate Firewall Login In Failure | 6.50 | -198 | 234 | 36 |
| Microsoft Defender for Office 365 Medium Severity AIR Alert | 6 | -6 | 6 | 0 |
| Successful Overpass The Hash Attempt | 5.50 | -9 | 11 | 2 |
| SentinelOne EDR Threat Mitigation Report Quarantine Success | 5.33 | -13 | 16 | 3 |
| Google Workspace User Deletion | 5 | -4 | 5 | 1 |

For the **reactive approach**, we chose to provide a push notification mechanism (through our internal instant messaging app), allowing detection engineers to subscribe to a given rule and receive a daily digest of the rule' sightings. Since Grafana dashboards already provide a great overview, we do not need to follow every rule as this will just result in spamming us and therefore not looking at these rules. However, subscribing to some rules to receive notifications only for them is important because it allows a detection engineer to easily check a rule's quality when the rule is created or updated. And as for the previously mentioned Grafana's dashboards, the results come with the appropriate link to pivot to the alerts, then the events: the analysis could start immediately and help decide and prioritise if any change is needed or not.

## Alerts found for the threat DCSync Attack.

Followers: ["Guillaume Couchard"].

| Pivot Links | Threat Name | Alert numbers | Rejected alerts | Closed alerts |
|---|---|---|---|---|
| Alerts - | DCSync Attack | 44 | 24 | 12 |

## Alerts found for the threat HackTools Suspicious Names.

Followers: ["Guillaume Couchard"].

| Pivot Links | Threat Name | Alert numbers | Rejected alerts | Closed alerts |
|---|---|---|---|---|
| Alerts - | HackTools Suspicious Names | 9 | 6 | 0 |

Similar to the Grafana dashboards, we also provide push notifications on a daily basis for the top 10 alerts ratio & diff in the alerts numbers as shown in the following screenshot:

**Rules alerts ratio between today-24H and -24H-48H**

| Pivot Links | Rule Name | Alert ratio 24h | Diff Nb Alerts 24H |
|---|---|---|---|
| Alerts - | Advanced IP Scanner | 43.83 | -514 |
| Alerts - | WAF Block Rule | 29.33 | -7337 |
| Alerts - | AD Privileged Users Or Groups Reconnaissance | 27.29 | -368 |
| Alerts - | FreeRADIUS Failed Authentication | 22.00 | -22 |
| Alerts - | Darktrace Threat Visualizer Model Breach Suspicious Activity | 19.00 | -36 |
| Alerts - | Internet Scanner Target | 0.02 | 86 |
| Alerts - | Suspicious DLL Loading By Ordinal | 0.07 | 25 |
| Alerts - | TOR Usage Generic Rule | 0.08 | 134 |
| Alerts - | Python Exfiltration Tools | 0.14 | 6 |
| Alerts - | Potential Azure AD Phishing Page (Adversary-in-the-Middle) | 0.14 | 12 |

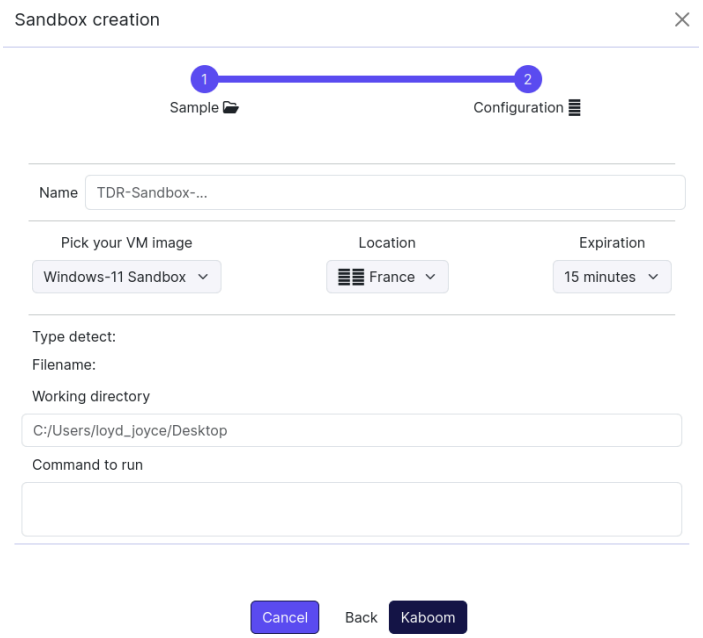
This is just like the Grafana dashboard but only for the most obvious detection rules, which allows us to focus on the main issues and not be spammed.

Focusing on these issues often means trying to find out what went wrong with the rule and, often, this means testing the rule again in a detection lab by replaying the attack / TTP / tool.

## Detection labs

Going back to the beginning of this series, a detection engineer has to impersonate the attacker in order to catch the real attack. As explained in the first article, we need to cover a very large technology environment, meaning that those must be available as quickly and simply as possible for each analyst. The amount of time and energy to build and maintain a detection lab could be an obstacle to work on a thousand rules!

Several solutions are possible, but on our side we decided to provide to all detection engineers an on demand lab for Windows or Linux environments with Endpoint and Network events collections automatically set up. This provides, in a couple of minutes, a way to either directly interact with the host and run an attack, or upload and execute a provided file. Furthermore for other labs that require several hosts, we tend to automate their creation using common devops best practices (terraform and ansible).




For cloud environments and SaaS products, we are mainly relying on our integration partners to provide the test tenant or when not possible the test events / apis allowing us to reproduce the attack behavior.

Our focus is on minimizing friction in the creation and update process for detection rules, and let our detection engineer focus on the detection techniques. The next step is to prioritize the detection rules changes that need to be made.

## Promising results

Following this methodology for a few months, we could observe some global results (with all the previously mentioned cornerstones), giving us motivation to continue and move forward in the continuous improvement of it.

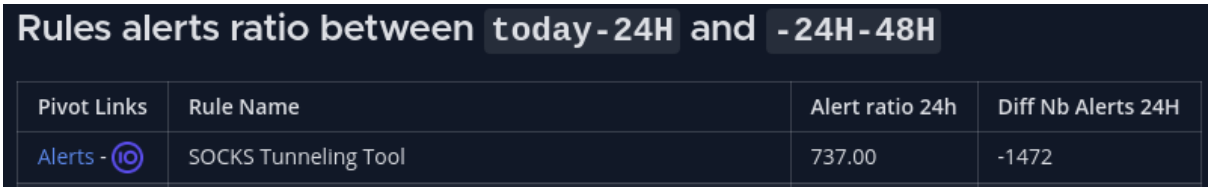
For instance while creating 25 and updating 83 detection rules, our measure rejection rate decreases from 58% to 38% on a given customer perimeter. Of course this is quite difficult to appreciate in one period of time, but we were able to have the same result in different time slots.

For the detection rule “PowerShell Malicious Nishang PowerShell Commandlets”, a daily result pinging an analyst was:

| Pivot Links | Threat Name | Alert numbers | Rejected alerts | Closed alerts |
|---|---|---|---|---|
| Alerts - | PowerShell Malicious Nishang PowerShell Commandlets | 11 | 7 | 1 |

Hunting into the related events, some matching patterns were obvious false positives, allowing to identify the filter to add and some pattern keyword to remove.

Another simple example where a surge of attacker tool usage (Chisel) on one customer is detected through an alert peak on our “SOCKS Tunneling Tool” rule:



Rules alerts ratio between `today-24H` and `-24H-48H`

| Pivot Links | Rule Name | Alert ratio 24h | Diff Nb Alerts 24H |
|---|---|---|---|
| Alerts - | SOCKS Tunneling Tool | 737.00 | -1472 |

After the analysis, the detection engineer confirmed this was a true positive, and also that the similarity strategy (which is how we group events into a same alert), should be fixed to help SOC analysts having relevant events in one alert group by the same hostname and username (in this specific case).

# Conclusion

Overall this article, we share our ideas and applied operational methodology allowing us to handle our detection rules at scale.

In the first part, we described how detection engineers have to face challenges from managing a high volume and diversity of logs to leveraging normalisation for detection rules. In the second part, we mentioned how implementing detection engineering at scale while adopting best practices from software development serves as a cornerstone for enhancing the efficiency and reliability of detection efforts. Finally, in this last part, we present how we enhance the process from the second part to be efficient and realistic for detection engineering teams. We discuss how, while a process may appear robust in theory, often has some limitations.These can be mitigated by a proactive and reactive approach at looking at KPIs by the detection engineers.

Without being the perfect solution, this could be from our point of view completely usable by other detection engineering teams, or at least trigger some discussion around it.

If you have any questions and feedback, please feel free to contact us.


## About Sekoia.io TDR team

TDR is the Sekoia Threat Detection & Research team. Created in 2020, TDR provides exclusive Threat Intelligence, including fresh and contextualised IOCs and threat reports for the Sekoia SOC Platform. TDR is also responsible for producing detection materials through a built-in Sigma, Sigma Correlation and Anomaly rules catalogue.

TDR is a team of multidisciplinary and passionate cybersecurity experts, including security researchers, detection engineers, reverse engineers, and technical and strategic threat intelligence analysts.

Threat Intelligence analysts and researchers are looking at state-sponsored & cybercrime threats from a strategic to a technical perspective to track, hunt and detect adversaries. Detection engineers focus on creating and maintaining high-quality detection rules to detect the TTPs most widely exploited by adversaries.

## About Sekoia.io

Sekoia.io is the European cybertech, leading provider of Extended Detection and Response (XDR) solutions based on Cyber Threat Intelligence (CTI). Its mission is to provide businesses and public organizations with the best protection technologies against cyber attacks.

By combining threat anticipation through knowledge of attackers (Sekoia Intelligence) with automation of detection and response, the Sekoia SOC platform (Sekoia Defend – XDR) provides security teams a unified view and total control over their information systems. Its interoperability with third-party solutions and compliance with international technical standards enable organizations to take full advantage of their existing technologies.

Sekoia.io gives its customers the means to focus their human resources on high value-added missions, optimize their cyber-defense strategy and regain the advantage against advanced cyber threats.

Find more publications on blog.sekoia.io

Please don't hesitate to provide your feedback on our publications by clicking here.
You can also contact us at tdr@sekoia.io for further discussions.