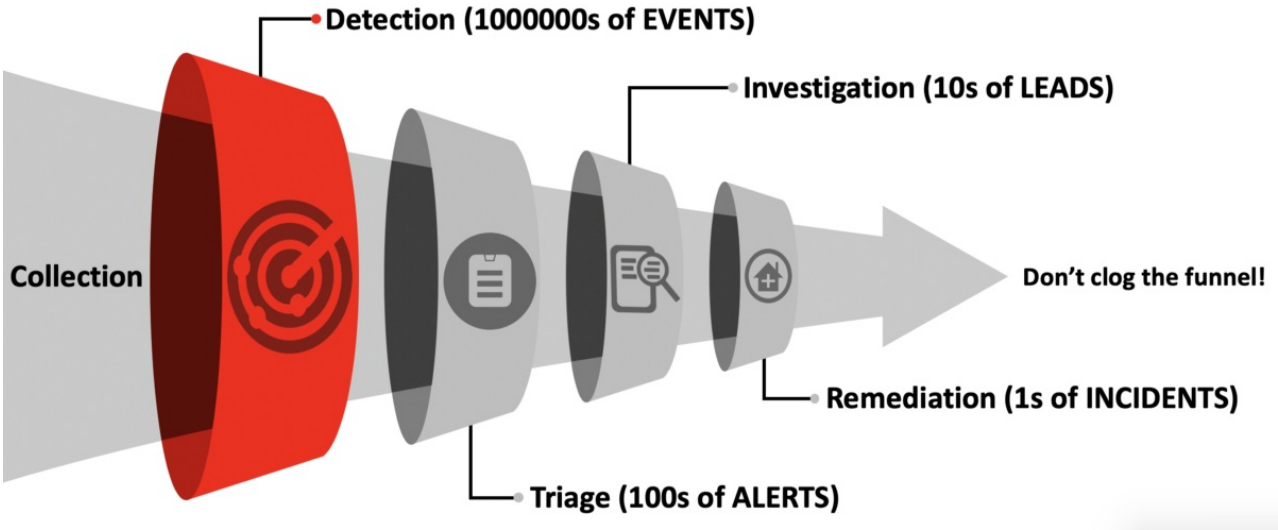
Posts By SpecterOps Team Members

posts.specterops.io/detection-spectrum-198a0bfb9302

February 21, 2020

Detection Spectrum

The is a model that SpecterOps uses to describe the phases involved in the detection and response process. We like to associate each blog post with the phase(s) that the post's topics relate to. This specific post is focused on a strategy that can be applied to the Detection phase of the funnel.



Detection Engineering with Kerberoasting Blog Series:

-
-

This is the second of a multipart detection engineering blog series by the SpecterOps detection team. The goal of this series is to introduce and discuss foundational detection engineering concepts. To make these concepts as consumable as possible, we are focusing the entire series around Kerberoasting. Focusing on this technique allows readers to focus on the strategies presented in each article instead of worrying about the details of the technique itself. The first post of the series discussed the concept of capability abstraction which is how tools abstract the complexities of the attack techniques they enable. Below is the "abstraction map" for the Kerberoasting technique that was produced in the previous post.

<table>
<tr><th colspan="5">T1208 - Kerberoasting</th></tr>
<tr><td>Tool</td><td>PowerShell
Invoke-Kerberoast</td><td>Rubeus
kerberoast</td><td>Mimikatz
kerberos::ask</td><td>Rubeus
asktgs</td></tr>
<tr><td>Managed Code</td><td colspan="2">.NET
KerberosRequestorSecurityToken</td><td></td><td rowspan="3"></td></tr>
<tr><td>Windows API Function</td><td colspan="2">InitializeSecurityContext</td><td>LsaCallAuthenticationPackage</td></tr>
<tr><td>RPC</td><td colspan="3">4f32adc8-6052-4a04-8701-293ccf2096f0
C:\WINDOWS\SYSTEM32\SspiSrv.dll</td></tr>
<tr><td>Network Protocol</td><td colspan="4">Kerberos TGS-REQ/REP</td></tr>
</table>

Introduction

We've all seen or heard it. Whether it stems from a detection engineer sharing detection logic publicly or a red teamer finding a trivial bypass to an EDR product's prevention capability, it is common to see someone label detections as being "brittle". The term brittle is usually used in a pejorative manner and is meant to imply that the detection is easy to bypass. Typically, this is because the detection logic is focused on one of the more superficial layers of the abstraction map, or is built using indicators that are trivial for an attacker to change such as hashes, IP addresses, domains, specific strings, or even command-line arguments. In a reality that includes commodity attacks, it would seem silly to ignore these simple straightforward detections purely because they are "easy" to bypass. At the very least, an organization should be in a position to detect unchanged known tools. With this in mind, such a detection doesn't seem to be inherently bad because it has a place in an overall detection strategy.

Instead of using loaded terms like brittle, simple, or bad to describe detection logic, this post proposes that detections fall on a "detection spectrum" where one end represents "precise" logic while the other represents "broad" logic.

Detection Spectrum

So what do the terms precise and broad tell you about your detection logic? The precise end of the spectrum represents detection logic that values the reduction of potential false positives over the reduction of potential false negatives. On the other end of the spectrum, broad detections value the reduction of potential false negatives over the reduction of potential false positives. The canonical example of a precise detection is looking for a malicious tool based on its default cryptographic hash. By definition, this approach is unlikely to produce false positives (a legitimate event that collides with the malicious binaries hash), but false negatives (a malicious event that does not share this hash value) are very likely. Alternatively, there is value in identifying the chokepoints that are most central to an attack technique to reduce the possibility of false negatives. For instance, the capability map defined in the previous post shows that a central feature of Kerberoasting is the Kerberos TGS-REQ network request, so detections built around this layer are inherently more broad than tool focused detections.

Typically when someone refers to detection logic as “brittle” it is because that detection falls too far on the precise end of the spectrum, but the term brittle implies judgment. Practically, it is difficult to judge detection logic without knowing the detection engineer’s intention. If the goal of the detection is to detect a specific default version of a tool with no false positives, then a precise detection is the perfect solution. If, on the other hand, the goal is to detect the use of an entire attack technique, then obviously it would be beneficial for the detection to migrate toward the broad pole on the spectrum. Of course, broad detections are not silver bullets. They typically create a more significant triage burden than precise detections. This is something that an organization must consider when they are weighing their desire for coverage against their resourcing. The point is that there is a place for precise detections just like there is a place for broad detections.

Think about the detections your organization has. Do they tend to lean to the precise (low false positive) or broad (low false negative) end of the spectrum? In reality, the diversity of most organizations generates a varied result. Factors such as staff size, telemetry collection, technical expertise, security automation, data pipeline, and even breadth of analyst responsibilities will impact this answer. For instance, an organization with a two-person IT team responsible for security and operations likely doesn’t have the bandwidth, experience, or time to create detections that will produce a significant number of false positives. However, an organization with a thirty person security team that is solely responsible for detecting and responding to security incidents can likely error on the broad side of the spectrum.

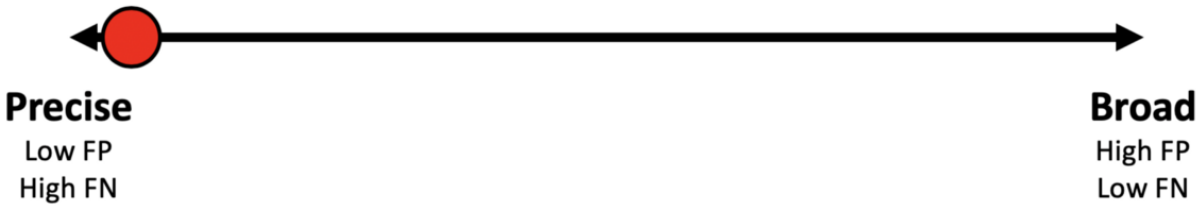
As detection engineers, we are responsible for understanding the breadth of our target attack technique and the level of our organization’s triage and investigation capabilities. By understanding these factors, we can build detections that the organization can consume, while still being aware of the unaddressed threat. We can use this information to drive the growth of the detection and response program in a meaningful way.

Case Study

The rest of this post builds on the Kerberoasting use case from the first post in this series. The remainder of the post explores three detections that demonstrate how important it is to understand the goal the detection is trying to achieve.

Example 1

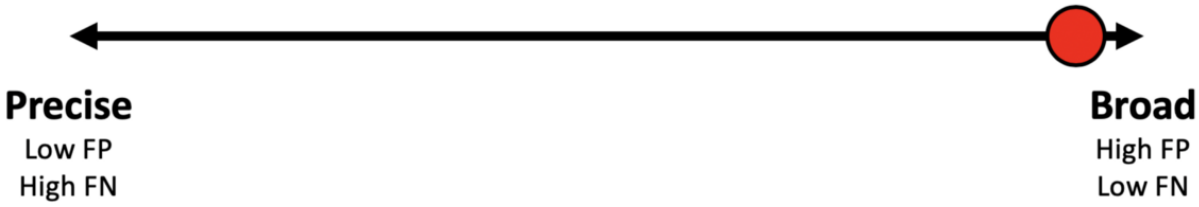
Consider a situation in which the detection engineer writes a detection that leverages PowerShell logging to alert when the “Invoke-Kerberoast” function is called. This approach detects the unaltered version of Invoke-Kerberoast, but the tool is relatively easy for an attacker to change to bypass this detection. For instance, an alert will not be generated if the attacker chooses to use any Kerberoasting tool other than Invoke-Kerberoast. If this detection was plotted on the detection spectrum it’d likely be near the “precise” pole. What does it mean for a detection to be “precise”? Precise indicates that alerts produced by this detection will exhibit an extremely low propensity of false positives. After all, who would arbitrarily name a legitimate PowerShell function Invoke-Kerberoast? However, precise detections have an extremely high propensity for false negatives. For instance, any Kerberoasting attempt that doesn’t use Invoke-Kerberoast will bypass this detection. The blind spots or limitations are easy to see in this approach which is why these “precise” detections are often mislabeled as “brittle”. If this detection only fires on true positives is that r bad thing? Generally, this type of detection is considered high fidelity and is appreciated by analysts responsible for triaging the resulting alerts because the precise nature reduces the triage burden. This detection is conceptually plotted on the detection spectrum below:



Example 2

A second example to consider will focus on the opposite end of the spectrum; specifically on the broad end. In part one of this series, Jared explained that regardless of implementation, a Kerberoasting attempt must make a TGS-REQ request to the Kerberos Key Distribution Center on the Domain Controller. With this in mind, an enthusiastic detection engineer may want to create a detection that focuses on how it can comprehensively account for every possible implementation of Kerberoasting. As a result, the detection engineer may write a detection that alerts anytime a connection is made where the destination host is the DC and the destination port is 88. For those that referred to the last detection as being too brittle, this particular detection surely answers that problem. Although, most readers probably see an equivalent number of flaws in this approach. This detection is a hyperbolic example of what the “broad” end of the

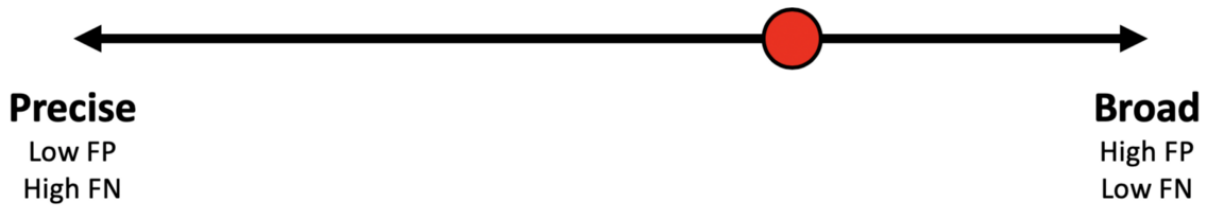
spectrum looks like, where the detection alerts on approximately every Kerberos request and potentially some non-Kerberos traffic that happens to occur over TCP port 88. Broad detections are characterized as detections that have a low likelihood of a false negative, but will typically exhibit a high rate of false positives. For anyone that has ever worked in a SOC, this type of detection can be a nightmare that results in an effective denial of service on the alert triage process. Like the previous example, this detection is conceptually plotted on the detection spectrum below:



Example 3

The following example shows an attempt to convert the broad detection discussed earlier to a bit more balanced detection. As stated in the previous post, asktgs is a command that allows Rubeus to request a Ticket Granting Service ticket outside of the normal request flow by handcrafting the TGS-REQ network request and hand parsing the TGS-REP response. While alerting on every TCP port 88 connection to a domain controller provides fairly broad coverage of all Kerberoasting attempts, the number of false positives (legitimate Kerberos requests) that will inevitably be produced renders this alert unusable in production. How can a detection engineer reduce false positives while maintaining a broadly focused scope? Expectation management! It is extremely important to have a clear goal for the detection logic and sometimes “detect all instances of Kerberoasting” is not a realistic goal for a single detection. To address the class of Kerberoasting implementations that are represented by “asktgs”, a detection engineer might define their goal as “detect instances of Kerberoasting where the Kerberos TGS-REQ is made outside of the expected Windows flow”. The first question that must be answered is “what is the expected Windows flow” for Kerberos TGS-REQ network requests? Maybe the detection engineer decides to look at commonalities among the majority of TCP port 88 requests and notices that the majority of these events originate from the lsass.exe process. Armed with this knowledge they can execute Rubeus’s asktgs command and find that the request originates from rubeus.exe (assuming they made no changes). The detection engineer’s first instinct may be to create a detection that looks for TCP port 88 requests made by rubeus.exe, but what happens if the attacker makes a trivial change like changing the name of the Rubeus binary? It turns out that maybe looking for rubeus.exe making TCP port 88 requests moved too far to the precise side of the spectrum to meet the stated goal as shown below:

Instead, the detection engineer might choose to create a detection that looks for any process that is not lsass.exe that makes a TCP port 88 connection to a domain controller. This represents a detection with a narrower scope than what they started with, but also stays broad enough to consider implementations that fall outside of stock Rubeus behavior. Depending on the environment, there may be additional need to narrow in further, but this gives the general idea of how detections can move along the spectrum. Below is an approximation of where this detection might fall on the spectrum.



Conclusion

Detection Engineering is difficult. We are responsible for developing a deep technical understanding of attack techniques and aligning the resulting risk within the inherent limitations of the organization's triage and investigation capabilities. While capability abstraction is a great tool for explicitly documenting the technical details of an attack technique, the detection spectrum is meant to facilitate discussions about the intention of certain detection logic. Is this detection focused on reducing false positives or false negatives? Most organizations can currently have the capability to consume alerts from precise detections, but generally, there is a fear of resulting false negatives. Review your detection logic. Are most detections focused on the precise end of the spectrum? If so, ask yourself why this bias exists and how you can enable the organization to consume broader (more false-positive prone) detection logic. Maybe the issue is lacking context in the generated alert, maybe analysts could benefit from triage playbooks for certain alert types, maybe some portion of the triage process can be automated to remove that first layer of false positives. Ultimately, organizations benefit from reducing the likelihood of false negatives, but it must be done in a consumable way for the analysts that receive the alerts.

This article discussed individual detection logic in a vacuum. The reality is that you are not limited to one detection per attack technique. Instead, a solid detection strategy combines numerous detections that address different layers of the capability abstraction and fall at different points of the detection spectrum. The next post in this series will explain our methodology of utilizing numerous detection approaches to build a comprehensive detection strategy while also considering the impact on alert triage.