Detecting Unauthorized Behavior From Legitimate Accounts

GIAC (GCDA) Gold Certification

Author: Rodney Caudle, caudlerodney@gmail.com

Advisor: Rick Wanner

Accepted: April 19, 2022

Abstract

Incident Responders face an almost insurmountable amount of log events, and the move to the Cloud has only intensified this dependency on log data for intrusion analysis. Attackers are shifting techniques to utilize compromised accounts to hide their activity among the volume of legitimate business activity. Security teams struggle to detect and alert to this scenario as the ratio between time-to-discovery and time-to-compromise continues to exhibit a sizeable gap. A new approach is needed to tackle complex alerting using tools available to security teams. This paper defines a new methodology to detect unauthorized access from legitimate accounts. The paper also uses open-source tools to implement this methodology, providing a cost-effective solution available to all security teams.

1. Introduction

Detecting anomalous behavior is a goal of most security monitoring programs. This concept, while seemingly simple, is a moderately complex goal to accomplish. Introducing three concepts encountered by security monitoring programs will help define the complexity and provide structure for addressing the challenges. These concepts include baselining normal activity, data alignment, and missing data elements.

Detecting anomalous behavior has been used to sell security products as a solution to understanding the vast amounts of data collected from operating environments. CapGemini defined, in 2014, four steps needed to detect anomalous behavior using machine learning algorithms. The first is to "determine the difference between normal and suspicious behavior" (CapGemini, 2022). CapGemini® uses User Behavior Analytics (UBA) to detect anomalous activity. Microsoft® also mentions the ability to track "user and entity behavioral analytics" (Microsoft Corporation, 2022). Both companies use a similar approach requiring an extensive data set of activity and using machine learning algorithms to process the data for anomalous activity. Wikipedia describes UBA as looking "at patterns of human behavior and then analyzes them to detect anomalies that indicate potential threats" (Wikipedia, 2021).

The approaches mentioned above describe a similar process with a first step to define normal behavior so that you can alert to deviations (anomalies). However, this is not a new concept, and the term "Know Normal" is used by security teams, such as New York State's Counter-Terrorism Cyber Incident Response Team (NY State DHS, 2021). The concept of what is normal is used in operational monitoring solutions to detect conditions requiring attention, including slow application response time, disk space consumption, and several other operational factors monitored continuously. To generate an alert on a slow application response time, a definition of the acceptable range must be determined. This same concept applies to security monitoring, referred to as establishing a baseline of normal activity. To generate an alert on anomalous activity, a baseline of normal activity must be established. Creating a baseline of normal activity is the first challenge to anomalous behavior detection.

The second challenge to anomalous detection is aligning the data used as a basis for baseline activity and detection processes. In mathematics, this concept establishes a canonical normal, or standard form, of the data collected. The Open Web Application Security Project (OWASP) defines canonicalization as the process of converting something from one representation to the simplest form (OWASP, 2013). A practical example is the inconsistent use of field names found in log events. Vendors will often represent the same piece of information using different field names. For example, a field containing an IP address of a host requesting a web page might be clientIP, originIP, source_ip, sourceIP, src, or srcIp. Aligning all variations of this information into a single field name, "src.ipaddr" would simplify the construction and maintenance of any logic built from this data set. CyberWarDog described this problem as a data quality issue in a blog post from 2017 (CyberWarDog, 2017). This blog post captures a scenario of data quality as "Data fields from different data sources do not have the same name." Inconsistent field naming conventions accurately characterize the problem and are one of the reasons to engage in data quality improvement before committing to analysis, correlation, or hunting scenarios. A failure to address data quality can result in large amounts of false positives, wasting security analyst time and reducing confidence in the monitoring program. In summary, aligning the data fields before beginning analysis will result in more accurate analysis and easier maintenance.

The third and final concept to introduce is also a part of CyberWarDog's blog post. The scenario is defined as "Data sources missing data" in the post but should, more accurately, be described as data needed for analysis is missing when data is collected (CyberWarDog, 2017). For example, consider an event representing the start of a process. The event will contain a process id, process name, date/time of occurrence, and some additional information about the status of the event. If the application producing the event does not record host information as part of the event data, consolidating the event can lead to difficulties tying back to the source. This is just one example of information that is missing from event data.

One reason for these missing data elements is described in a paper published by Computer Associates titled Common Event Grammar (Computer Associates, 2011). This document describes the perspective of an event which provides a hypothesis when data appears to be missing from the events once collected. CEG describes the perspective of an event as the number of entities involved in the expression of said event. According to Common Event Grammar, a single event could contain information from up to four different entities embedded in a single event record. Single entity events describe activities occurring on a single host. In simplistic terms, the source and destination of an event are the same. An event with two entities is the most common type of event found. Access logs for a web server are an example of this perspective of events. The web server application creating the events has the perspective describing actions happening to itself. Rephrasing the events as a statement can help understand the perspective.

Perspectives involving more than two entities, such as an Intrusion Prevention Server (IPS), may switch perspectives for certain log events. An IPS typically describes information between two other entities; "I observed this IP attempting to connect to that IP on port 443." However, for maintenance events or events about itself, the perspective may shift only to include two entities or even single entity perspectives. Understanding an event source can shift perspectives as needed and will help with data alignment in addition to filling in the missing data.

The three concepts discussed in this introduction will provide a foundation for the methodology presented in this paper. Addressing the need for baseline activity, aligning the data collected, and realizing there may be missing data fields are essential to accomplishing the goal of detecting unauthorized access from legitimate accounts. The following section will continue this discussion by taking a deeper examination of data alignment and establishing a canonical set of fields. These fields will provide a basis for business logic that identifies baseline activity and ultimately alerts to deviations.

2. Baselining and Canonical Fields

The first step in establishing a security monitoring program is to define the canonical set of fields. Mapping the vendor-specific events to a single, unified set of fields will allow normalization of the event sources and efficient correlation after collection. This paper will use fields defined by the Common Event Grammar (CEG) (Computer Associates, 2011). The CEG provides a common structure for events and a finite set of fields for mapping collected events. The following is a set of fields used in this paper to map fields from collected events.

Area	**Field Name**	**Description**
src	ipaddr	IP Address for the source of the event
src	fqdn	A fully qualified domain name derived from the IP address
src	hostname	The hostname for the source of the event
src	username	Username for the source of the event
src	domainname	Domain name for the source of the event
src	userid	User id for the source of the event
src	port	Network port for the source of the event
dst	ipaddr	IP Address of the destination or target
dst	fqdn	A fully qualified domain name derived from the IP address
dst	hostname	The hostname for the destination of the event
dst	username	Username for the destination of the event
dst	domain name	Domain name for the destination of the event
dst	userid	User id for the destination of the event
dst	port	Network port for the destination of the event
event	datetime	Time of the event occurrence (UTC)
event	result	Success or Failure
event	id	Unique ID for the individual event
event	category	Category of event

event	class	Class of event

Table 1: Normalized Field Listing (chosen from available CEG fields)

To illustrate how these fields will be used in this paper, consider the following example SSH login event.

```
     Jan 26 20:51:40 HOME-LAB sshd[79487]: Accepted
password for user1 from 10.1.1.1 port 60480 ssh2
```

Parsing this example event into the canonical fields above would result in the following set of field-value pairs. The parsing can be accomplished with Logstash or another client to collect the log events.

Field	Value
src.ipaddr	10.1.1.1
src.port	60480
dst.username	user1
dst.hostname	HOME-LAB
event.datetime	Jan 26 20:51:40
event.result*	Success
event.timezone*	-0500
event.category*	System Access
event.class*	Login Attempt

Table 2: Sample SSH Event

Some field values were extracted from the raw event data, and others were hardcoded at collection time based on identifying the event perspective. The fields notated by an asterisk (*) were not included in the event but filled in at collection time. Refer to the missing data and event perspective previously discussed. The perspective of this SSH login event is about two entities: server and client. Understanding the perspective and the entity creating the event allows the information gaps to be addressed during the collection of the event. Still, other fields will be added after collecting during an enrichment phase. Regardless of the event source (Linux, Unix, Windows, etc.), the resultant mapping would look like this example if a login event occurred. However, each vendor will provide differing amounts of information and potentially different perspectives.

3. Methodology

The canonical field list will be the basis for all analysis in this paper. The rules and analysis defined in later sections assume a normalized event repository is available for processing. A consistent methodology will be presented from the normalized repository to accomplish the detection scenarios identified. The methodology implemented is based on real-world analysis and has been implemented with success many times. The steps of this methodology are:

Step 1:	Filter	Define a filter for an event feed
Step 2:	Aggregate	Query the filtered event feed into sample events
Step 3:	Enrich	Enhance sample events with historical information
Step 4:	Alert	Alert on conditions handling suppression to prevent overwhelming analysts

These steps will be connected using elasticsearch indexes acting as simple queues. However, alternative implementations could pass the events using other queueing mechanisms like Kafka or Mulesoft. Visualizing the information flow this paper will follow to generate actionable alerts is captured in Figure one (1) below.

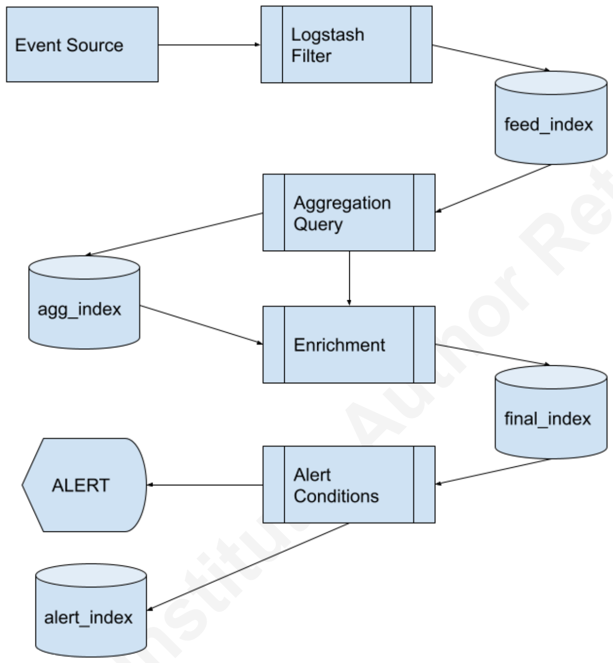

Figure 1: Information Flow Diagram

Step 3 (Enrichment) is displayed in the diagram as retrieving information from the aggregation index, although this may not happen during implementation. For example, enrichment could be constructed to retrieve from external data sources, internal data sources, or any repository needed. Additionally, a scenario requiring multiple enrichments may be executed serially or parallel. This paper will define the enrichments for the scenario and provide more information and example scripts.

Following these steps will provide a consistent framework for documenting and growing a correlation mesh of detection routines. This framework aims to produce consistently actionable anomalous events for security analysts to invest time in examining. However, real-world conditions exist, which may lead to false positives for a scenario. IP address rotation or mobile network behavior (when viewed at the IP level) are examples of conditions captured for each scenario. Conditions known for each scenario will be documented as part of Step 4 (Alert Conditions) within each scenario. The output from Step 4: Alert Conditions will be actionable and are expected to be delivered to an alert tool for escalation following defined policies. The alerts will be captured to an elasticsearch index and displayed in a Kibana dashboard to complete each scenario.

3.1. MITRE ATT&CK Framework T1078

The MITRE ATT&CK framework provides a standard framework for describing attack scenarios commonly observed during intrusion analysis. This paper focuses on the ATT&CK technique T1078 Valid Accounts and sub-techniques as used by attackers to target victims (MITRE ATT&CK, 2021). ATT&CK version 5 includes detections for T1078, including Logon Session Creation, Logon Session Metadata, and User Account Authentication. This section will describe these detections and how to use the detections to alert anomalous behavior by legitimate accounts.

MITRE expands on the Valid Account technique with four sub-techniques focused on desirable types of accounts. Microsoft defines these accounts as attractive for credential theft (Microsoft, 2021). MITRE captured this classification as sub-techniques T1078.001 Default Accounts, T1078.002 Domain Accounts, T1078.003 Local Accounts, and T1078.004 Cloud Accounts. Focusing implementations of the techniques discussed in this paper on specific account types will help produce actionable alerts more quickly than tackling all possible kinds of accounts at an enterprise level.

MITRE provides documentation of example procedures where adversaries have used this technique in executing attacks. MITRE documents fourteen examples where technique T1078 was used in an intrusion. Additional examples are provided for each of the sub-techniques as well. This information is valuable for building the business case to pursue protective or detective controls such as the detection mesh discussed in this paper.

In recent versions of the MITRE ATT&CK framework, MITRE has started documenting Mitigations and Detections. These sections focus on preventing the technique or detecting the technique's use. For T1078, two detections are identified as DS0028 Logon Session and DS0002 User Account. For each detection data source and data, source components are provided. This information has been factored into the normalized Common Event Grammar section described above.

Specifically, MITRE recommends tracking three pieces of information for Valid Accounts (MITRE ATT&CK, 2021): Logon Session Creation, Logon Session Metadata, and User Account Authentication (MITRE ATT&CK, 2021). The user account creation is not included by MITRE in this technique but is captured as a different technique (T1585 Establish Accounts) in the ATT&CK framework. This paper incorporates data sources providing events matching DS00028 Logon Session Creation. Logon events, as discussed above, are examples of this detection data source.

While this paper focuses on logon events as a primary example of the techniques described, the same techniques would work when applied to other data sources such as authentication data sets or resource access data sets. The process for detecting and alerting to anomalous behavior is valid for any data set. This paper will capture examples utilizing data sources beyond what is currently envisioned by MITRE ATT&CK framework. However, ATT&CK is continuing to evolve and may incorporate aspects of additional data sources in future versions. The current version is still a powerful source of data for intrusion analysts and provides a good starting point for analysis. However, additional guidance is needed to produce actionable alerts from the generalized advice provided by MITRE ATT&CK. This paper will start from the MITRE ATT&CK framework and demonstrate the additional guidance needed to produce actionable alerts for each scenario discussed in later sections.

3.2. New Source IPs

3.2.1. Description

Tracking where an account originates is essential to defining anomalous activity. Once this baseline is established, alerting to variations or deviations from the baseline is possible. This scenario will build alerts specifically for detecting activity for a user account from a previously unknown IP address. For example, if an application user account that usually originates from a few locations in New York suddenly originates from the United Kingdom, an alert should be generated and investigated. These scenarios have become more important for most enterprises because the pandemic shifted a large portion of the workforce to home offices. Monitoring these locations can prevent an enterprise from becoming a victim of a compromised home network.

This scenario will establish a series of logstash pipelines to process data in a structured manner to produce the desired outcome. For this scenario, the objective is to alert on the first time a source IP address. As discussed in section 2, this information is held in the src.ipaddr field of the canonical dataset.

3.2.2. Step 1 - Filter

This scenario will utilize events that match the following criteria regardless of which event source the events originated from.

```
event.class = "Login Attempt"
```

A minimum set of fields will be preserved, if available, in the normalized event retrieved for aggregation for each event. Specifically preserving src.ipaddr, src.fqdn, src.hostname, src.username, dst.hostname, dst.username, dst.domainname, dst.ipaddr, dst.hostname, event.result and event.datetime.

All events are stored in the feed_index—the feed_index acts as potential input to correlations and aggregations in the next step. The following logstash filter can be used to filter events for this scenario. The events can be picked up from any source available to Logstash. (Elastic, 2022) The example filter below will pick up the events from a flat file, and any new events will flow into the pipeline. These events will be processed, and all relevant events extracted, normalized, and forwarded to the feed_index destination.

```
input {
    file {
        path => "/data/events/scenario1/step1/*.log"
    }
}

filter {
    date {
        match => [ "event.datetime" , "MMM  d HH:mm:ss", "MMM dd HH:mm:ss" ]
    }
}

output {
    if [event.class] == "Login Attempt" {
        elasticsearch {
            index => "%{[some_field][sub_field]}-%{+YYYY.MM.dd}"
        }
    } else {
```

```
                stdout { codec => rubydebug }
            }
        }
```

Figure 2: Step 1 Logstash Filter

This filter ingests events from a series of files stored in a directory, assigns the appropriate format for the date/time field, checks for the minimal set of fields needed for aggregation, and stores the appropriate files into feed_index for the next step in the workflow. The input format assumes json formatted events for these scenarios. There are many formats available, but json is easily ingestible for processing.

3.2.3. Step 2 - Aggregate

Aggregations are executed periodically to extract calculations based on one or more feeds. The aggregation should be designed to preserve the critical pieces of information from the raw event feed. Aggregations are used to save on storage space and retention. For example, for a login attempt, the critical information is in the fields src.ipaddr, dst.ipaddr, dst.username, and event.result. The exact set of fields needed can be customized to each environment. The goal is to preserve the fields required to help an analyst drill into and filter down to raw events during an investigation.

Raw events occur with a granularity equivalent to or less than a second. However, for the purposes of this analysis, 15 minute windows, or even hourly aggregation windows can be used depending on how quickly the desired alert for this scenario. This paper will use a five-minute window of aggregation for this scenario.

```
        input {
            elasticsearch {
                hosts => "127.0.0.1"
                index => "scenario1-feed_index*"
                scroll => "5m"
                schedule => "*/5 * * * *"
                query => '{ "query": { "bool": { "must": [], "filter": { "range": 
{"event.datetime": {"gte": "now-5m"} } }, "should": [], "must_not": [] } } }'
            }
        }

        filter {
            aggregate {
                task_id => 
"%{src.ipaddr}_%{dst.ipaddr}_%{dst.username}_%{event.year}_%{event.month}_%{event.day}"
                code => "
                    map['src.ipaddr'] = event.get('src.ipaddr');
                    map['dst.ipaddr'] = event.get('dst.ipaddr');
                    map['dst.username'] = event.get('dst.username');
                    map['event.result'] = event.get('event.result');
                    map['event.datetime'] ||= event.get('event.datetime');
                    map['event.count'] ||= 0;
                    map['event.count'] += 1;
                    map['event.aggregation'] = true;
                "
                push_map_as_event_on_timeout => true
                timeout => 30
                timeout_task_id_field => "event.aggregation_id"
            }

            if (![event.aggregation]) {
                drop {}
            }
        }

        output {
            elasticsearch {
                index => "scenario1-agg_index-%{+YYYY.MM.dd}"
            }
        }
```

Figure 3: Aggregate Logstash Filter

The aggregate filter plugin from elasticsearch provides the capability to roll up events, preserving specific fields and discarding others. Aggregation allows the creation of alert-specific minimum fields to be preserved, which are ideal for the scenarios discussed in this paper. In the configuration above, the query executed uses relative time information to gather events from the preceding five minutes. This timeframe will need to be synchronized with the aggregation window desired. Continuing the example in this scenario, the output from this aggregate plugin looks like Figure four (4) below.

```
    {
           "event.aggregation" => true,
                "dst.username" => "user1",
                  "src.ipaddr" => "10.1.1.1",
                "event.result" => "Success",
                 "event.count" => 80,
        "event.aggregation_id" => "10.1.1.1_10.1.1.200_user1_2022_Mar_11",
              "event.datetime" => "2022-03-12T02:51:40.000Z",
                  "dst.ipaddr" => "10.1.1.200",
                    "@version" => "1",
                  "@timestamp" => 2022-03-11T21:23:38.550Z
    }
```

Figure 4: Aggregated Event Sample

When launching the aggregation pipeline, the number of logstash workers needs to be reduced to one (1), so the aggregation will work properly. With aggregated events being created for this scenario, the next step is adding enrichments to augment the information in preparation for alerting.

3.2.4. Step 3 - Enrich

Enrichments are added at this phase to provide context for deciding to alert in the next step. For this scenario, there will be several enrichments added, but the most critical is a determination of whether this source IP has been observed in the past. Capturing this information will require the creation of a second query to reflect on the aggregation index and determine if the source IP from the current event has been observed previously. This input filter should resemble Figure five (5) below, picking up new aggregated events using a query similar to the previous step.

```
        input {
            elasticsearch {
                hosts => "127.0.0.1"
                index => "scenario1-agg_index*"
                scroll => "5m"
                schedule => "*/5 * * * *"
                query => '{ "query": { "bool": { "must": [], "filter": { "range":
{"event.datetime": {"gte": "now-5m"} } }, "should": [], "must_not": [] } } }'
            }
        }
```

Figure 5: Enrichment Input

This input queries for all aggregate events received in the last 5 minutes and is scheduled to periodic execute every 5 minutes. For every event received, the filter plugin will look for occurrences of the src.ipaddr field in the past 30 days. However, the filter should exclude the last five minutes, or the enrichment will always be "true." The ranges used for a boundary on date and time should be aligned with the ingest window for this step as well as the desired reflection period. Figure six (6) describes the filter plugin used to create this reflection setting the value event.first_seen flag to “true” if an event exists with the same src.ipaddr field and “false” if none are found.

```
        filter {
            elasticsearch {
                hosts => "127.0.0.1"
                index => "scenario1-agg_index*"
                query => 'src.ipaddr: "%{src.ipaddr}" AND event.datetime <= "now-5m" AND
event.datetime >= "now-30d"'
                fields => { "event.first_seen" => "src.ipaddr" }
                result_size => 1
                enable_sort => false
            }
            if [event.first_seen] {
                mutate {
                    replace => { "event.first_seen" => "false" }
                }
            } else {
                mutate {
                    replace => { "event.first_seen" => "true" }
                }
            }
        }
```

Figure 6: Enrichment Filter Plugin

The enriched event will add a field called “event.first_seen” with a Boolean value of either true or false. Figure seven (7) shows an example of the enriched event below.

```
        {
                    "event.result" => "Success",
                        "@version" => "1",
                    "dst.username" => "user1",
                      "dst.ipaddr" => "10.1.1.200",
            "event.aggregation_id" => "10.1.1.1_10.1.1.200_user1_2022_Mar_14",
                     "event.count" => 5,
                      "src.ipaddr" => "10.1.1.1",
               "event.aggregation" => true,
                "event.first_seen" => "true",
                      "@timestamp" => 2022-03-15T00:15:49.351Z,
                  "event.datetime" => "2022-03-15T01:51:40.000Z"
        }
```

Figure 7: Enriched Event

Finally, the enriched events are saved to an elasticsearch index called "final_index" for alerting. The full logstash filter file is provided in the appendix as an attachment.

3.2.5. Step 4 - Alert

With the fully enriched event residing in the final_index elasticsearch index, alerting is a simple process of setting the criteria and filtering the events periodically. The input for the alerting step will utilize a similar elasticsearch plugin with criteria set to only include events where the field event.first_seen is set to "false". These events are forwarded to an output of another elasticsearch index and can be sent to an alerting output plugin to generate an email alert, sms, or other options.

```
        input {
            elasticsearch {
                hosts => "127.0.0.1"
                index => "scenario1-final_index*"
                scroll => "5m"
                schedule => "*/5 * * * *"
                query => '{ "query": { "bool": { "must": [{"match_phrase":
{"event.first_seen": "false" }}], "filter": { "range": {"event.datetime": {"gte": "now-
5m"} } }, "should": [], "must_not": [] } } }'
            }
        }

        filter {
        }

        output {
            stdout { codec => rubydebug }
            elasticsearch {
                index => "scenario1-alert_index-%{+YYYY.MM.dd}"
            }
        }
```

Figure 8: Alert Logstash Filter

3.2.6. First Seen Scenario Visualization

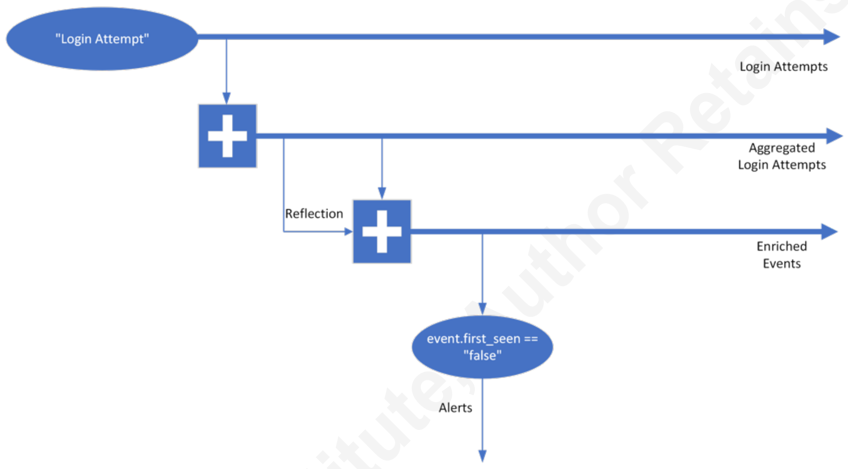

Figure 9: First Seen Scenario Visualization

3.2.7. Expansion and Alternatives

In this scenario, the objective is to alert on first seen occurrences based on the src.ipaddr field. However, a slight modification to look at dst.ipaddr would generate an alert on new communication pathways. This additional alert could potentially alert to lateral movement, exfiltration, or command and control channels. To handle this dst.ipaddr alternative, simply repeat the first two steps or create a new enrichment filter based on the aggregated events created from Step two (2) above. The output from this enrichment can either flow into the same elasticsearch index or a second index depending on the preferred approach. Figure 10 below contains the enrichment filter from Step 3 modified to handle the dst.ipaddr field.

```
        input {
            elasticsearch {
                hosts => "127.0.0.1"
                index => "scenario1-agg_index*"
                scroll => "5m"
                schedule => "*/5 * * * *"
                query => '{ "query": { "bool": { "must": [], "filter": { "range":
{"event.datetime": {"gte": "now-5m"} } }, "should": [], "must_not": [] } } }'
            }
        }

        filter {
            elasticsearch {
                hosts => "127.0.0.1"
                index => "scenario1-agg_index*"
                query => 'dst.ipaddr: "%{dst.ipaddr}" AND event.datetime <= "now-5m" AND
event.datetime >= "now-30d"'
                fields => { "event.first_seen" => "dst.ipaddr" }
                result_size => 1
                enable_sort => false
            }
            if [event.first_seen] {
                mutate {
                    replace => { "event.first_seen" => "false" }
                }
            } else {
                mutate {
                    replace => { "event.first_seen" => "true" }
                }
            }
        }

        output {
            elasticsearch {
                index => "scenario1-final_index-%{+YYYY.MM.dd}"
            }
        }
```

Figure 10: dst.ipaddr Enrichment Filter

This example did not attempt to add additional enrichment dimensions for consideration during alert generation. However, the enrichment step can be extended to provide additional enrichments for geographic location, IP reputation, the organizational role of the IP, and others. Additional dimensions added can help refine alerts and tune the alerts to your environment.

3.2.8. Errors and Troubleshooting

Latency is a real concern with event processing pipelines such as the ones defined in this paper. Each enrichment will add latency to the processing pipeline, increasing the amount of time needed to reach an actionable alert. This paper defined a scenario requiring 15 minutes from step one (1) (filtering) through step four (4) (alerting). If the dst.ipaddr enrichment is added, this could increase the time required to 20 minutes. Careful maintenance of the pipeline is needed to maintain a pipeline that provides sufficient assurance in the alerts generated but minimizes the time required to reach an actionable alert.

Reducing latency is possible by creating an additional event stream for each enrichment and alerting combination. In this scenario, both the src.ipaddr enrichment and the dst.ipaddr would be occurring in parallel, not requiring additional latency. The additional event streams can feed on the same aggregation events but having the enrichment and alerting steps happen simultaneously saves on latency.

Finally, consider the cost of each additional enrichment against the value added to reaching an actionable alert. For example, the pipeline could be constructed to enrich all IP addresses (both src and dst) with geographic location as part of this scenario. While convenient to access, this information is not crucial to reaching the determination of whether this IP is considered “first_seen”. The security analyst researching alerts could complete the addition of geographic information later, or this information could be enriched as part of a geofencing scenario in another part of the correlation mesh after alert generation.

4. Conclusions

This paper discussed the importance of baselining activity, a canonical set of fields, and filling in the missing information present in security events. Additionally, the concept of event perspective was briefly discussed as a reason for the missing data. An example was provided showing how the perspective of an event could be used to complete the missing information at collection time. Accomplishing the three important activities discussed in the introduction leads to a solid foundation for building more complex, higher assurance alert scenarios for investigations.

The use of a four-step methodology (Filter – Aggregate – Enrich – Alert) demonstrated how to define complex scenario-based event processing pipelines alerting to unusual activity from legitimate accounts. The first two steps (Filter – Aggregate) create a baseline of event activity that can be recursively queried to enrich the events for actionable alerts. Examples for each step were provided based on a standard elasticsearch, logstash, and kibana (ELK) stack.

Finally, a discussion on alternative enrichments, latency, and the balance between performance and latency was discussed. Optimization of event processing pipelines is an important design aspect of a correlation mesh. Examples were given for handing the alternative to alert on first seen dst.ipaddr fields. This alternative could provide an alert to lateral movement as an adversary begins to explore laterally from a compromised asset.

The methodology and techniques discussed in this paper will provide a more well-defined approach to generating actionable alerts. This structured approach allows companies without large budgets for toolsets to generate an effective correlation mesh. The use of reflective enrichments is a powerful technique for increasing the assurance of alerts generated from security event consolidation.

References

CapGemini. (2022). *Anomalous Behavior Detection.* Retrieved from CapGemini: https://www.capgemini.com/service/digital-services/insights-data/using-analytics-to-protect-organizations-from-fraud/anomalous-behavior-detection/

Computer Associates. (2011). *What is CEG?* Retrieved from ftpdocs.broadcom.com: https://ftpdocs.broadcom.com/cadocs/0/CA%20Enterprise%20Log%20Manager%20r12%205%2001-ENU/Bookshelf_Files/HTML/675145.html

CyberWarDog. (2017, 12 15). *Ready to Hunt? Show me your data.* Retrieved from cyberwardog.blogspot.com: https://cyberwardog.blogspot.com/2017/12/ready-to-hunt-first-show-me-your-data.html

Microsoft. (2021, 07 29). *Attractive Accounts for Credential Theft.* Retrieved from docs.microsoft.com: https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/plan/security-best-practices/attractive-accounts-for-credential-theft

Microsoft Corporation. (2022, 01 26). *Get behavior analytics and anomaly detection.* Retrieved from docs.microsoft.com: https://docs.microsoft.com/en-us/defender-cloud-apps/anomaly-detection-policy

MITRE ATT&CK. (2021, 10 20). *Logon Session.* Retrieved from attack.mitre.org: https://attack.mitre.org/versions/v10/datasources/DS0028/

MITRE ATT&CK. (2021, 10 20). *User Account.* Retrieved from attack.mitre.org: https://attack.mitre.org/versions/v10/datasources/DS0002/

MITRE ATT&CK. (2021, 10 19). *Valid Accounts.* Retrieved from attack.mitre.org: https://attack.mitre.org/versions/v10/techniques/T1078/

NY State DHS. (2021, 09). *New York State DHS.* Retrieved from www.dhses.ny.gov: https://www.dhses.ny.gov/system/files/documents/2021/09/know-normal-activity.pdf

OWASP. (2013, 05 12). *Canonicalization, locale, and Unicode.* Retrieved from wiki.owasp.org: https://wiki.owasp.org/index.php/Canonicalization,_locale_and_Unicode

Wikipedia. (2021, 12 7). *User Behavior Analytics.* Retrieved from Wikipedia: https://en.wikipedia.org/w/index.php?title=User_behavior_analytics&oldid=1059098948

Wikipedia. (2022, 02 10). *Canonical Form.* Retrieved from Wikipedia.com: https://en.wikipedia.org/w/index.php?title=Canonical_form&oldid=1071046658