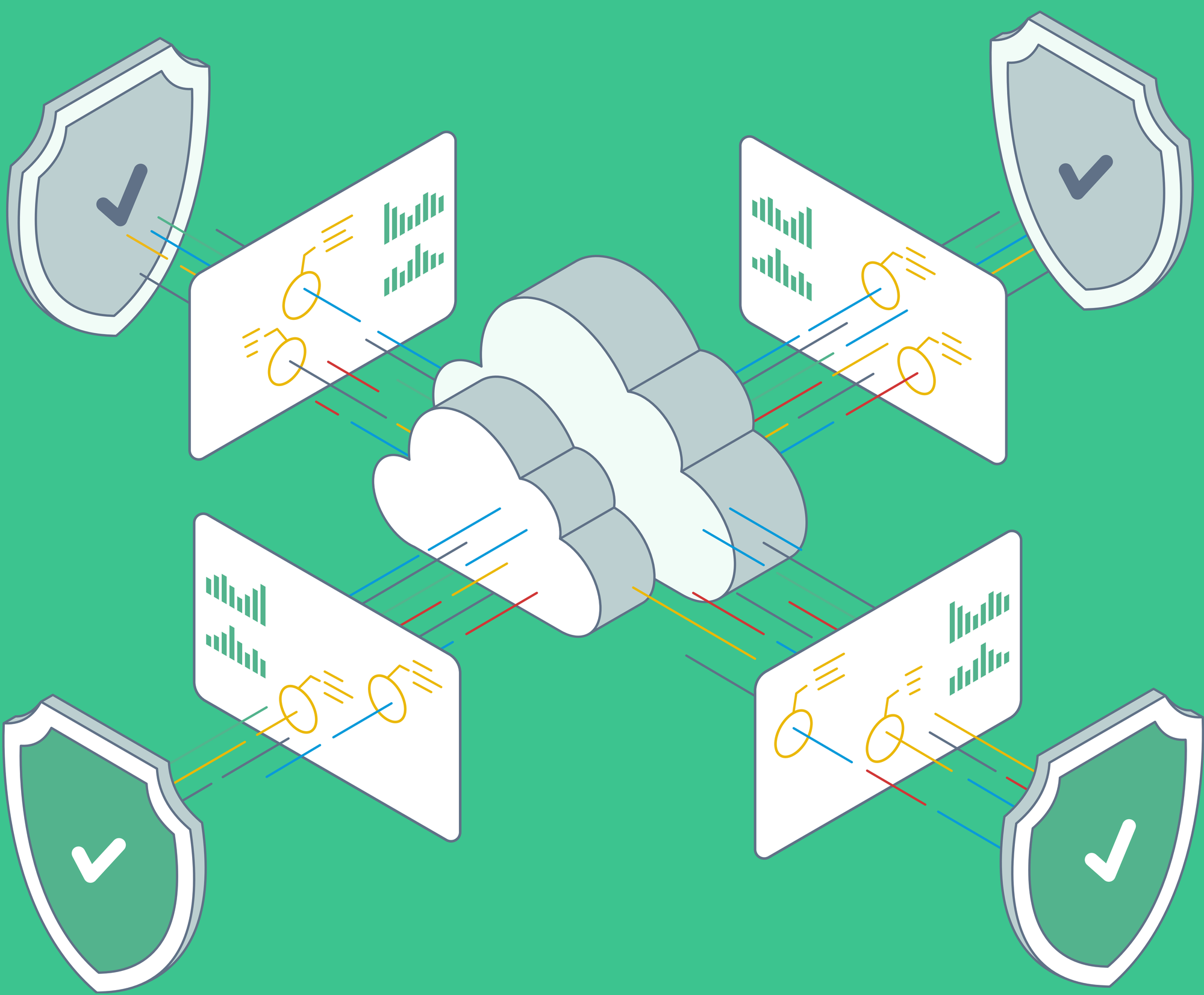
WHITEPAPER

Coralogix Cloud Security

By: Yuval Khalifa, Cyber Solutions Architect at Coralogix



Coralogix

Table of Contents

Introduction

A successful cloud and container-native security posture is the goal that organizations are striving towards but achieving it can seem lofty when faced with mounting complexities and a lack of expert security resources. Setting up a proactive monitoring solution is daunting in and of itself and creating one without causing analyst fatigue and avoiding prohibitive data costs can make it seem nearly impossible.

Security logs like Cloudtrail, VPC Flow logs, GuardDuty, and Auditbeat without network packet data, are not enough to paint the full picture of what's being transmitted, and by whom, allowing attackers to circumvent safeguards, as well as not having the information needed to perform deep enough investigations. However, setting up, processing, and storing packet data can be laborious and cost-prohibitive.

The Coralogix Cloud Security solution brings visibility and threat insights to SOC and DevOps teams within minutes, instead of months. In order to deliver reliable alerts with actionable context, Coralogix uniquely correlates contextual log data and combines it with network packet data.

Here's a full comparison between the STA and all the other methods in AWS:

Feature	Coralogix STA	CloudWatch Metrics	CloudTrail Logs[2]	VPC Flow Logs	GuardDuty	VPC Traffic Mirroring	VPC Firewall
Provides a set of metrics that are calculated based on the traffic	✗ [1]	✓	✗	✗	✗	✗	✗
Provides auditing information about AWS activities	✗	✗	✓	✗	✗	✗	✗
Allows the user to create new metrics and modify existing ones	✓	✗	✗	✗	✗	✗	✓ [6]
Provides Layer 4 Context Data (IPs, Port Numbers)	✓	✗	✗	✓	✗	✓	✗
Provides Layer 7 Context Data (HTTP URI's and methods, SSL Certificates, DNS Queries, FTP commands, files, etc)	✓	✗	✗	✗	✗	✓	✓ [4]
Detects threats or malicious content	✓	✗	✗	✗	✓	✗	✓ [5]
Detects potentially malicious behaviors	✓	✗	✗	✗	✓	✗	✗
Allows the user to understand, modify, disable and create new detection rules	✓	✗	✗	✗	✗	✗	✗ [3]
Allows the user to access and store the captured traffic	✓	✗	✗	✗	✗	✓	✗
Enriches the data (for example by adding domain creation dates to domain names)	✓	✗	✗	✗	✗	✗	✗
Allows integration with OSSEC/Wazuh or similar agents for the purpose of collection of instance specific data such as processes running on each instance	✓	✗	✗	✗	✗	✗	✗
Comes with predefined set of alerts, including ML powered ones	✓	✗	✗	✗	✓	✗	✗
Allows the user to customize the set of alerts and to create new ones	✓	✗	✗	✗	✗	✗	✓
Comes with predefined dashboards for each type of protocol	✓	✗	✗	✗	✗	✗	✗
Can be used to detect lateral movements	✓	✗	✗	✓	✓	✗	✗ [7]
Allows the user to customize the predefined dashboards to his needs	✓	✗	✗	✗	✗	✗	✗

(1) Will be added soon in upcoming versions

(2) Since CloudTrail logs are basically an auditing mechanism for AWS, they provide a different type of context data that is also very valuable for forensic investigations

(3) Allows the user to create new rules

(4) The VPC Firewall contains some protocol analyzers for analyzing the packets passed through it but they are not as extensive and extensible as the Zeek platform embedded in the STA. The Zeek platform contains protocol analyzers for hundreds of different protocols

(5) Can detect threats at the perimeter only and only based on its suricata's signatures list

(6) The offered metrics are very basic like dropped/passed packets. The user can add one dimension to the metrics called "CustomAction". See more here: https://docs.aws.amazon.com/network-firewall/latest/developerguide/monitoring-cloudwatch.html

(7) The VPC firewall can only be installed at the perimeter level, as an inline device, which prevents it from being exposed to the inter-vpc traffic

How It Works

The Coralogix 3-click Cloud Security deployment enables organizations to quickly start improving their security posture, detect threats, and analyze digital forensics without the complexity, long implementation cycles, and high costs of other solutions.

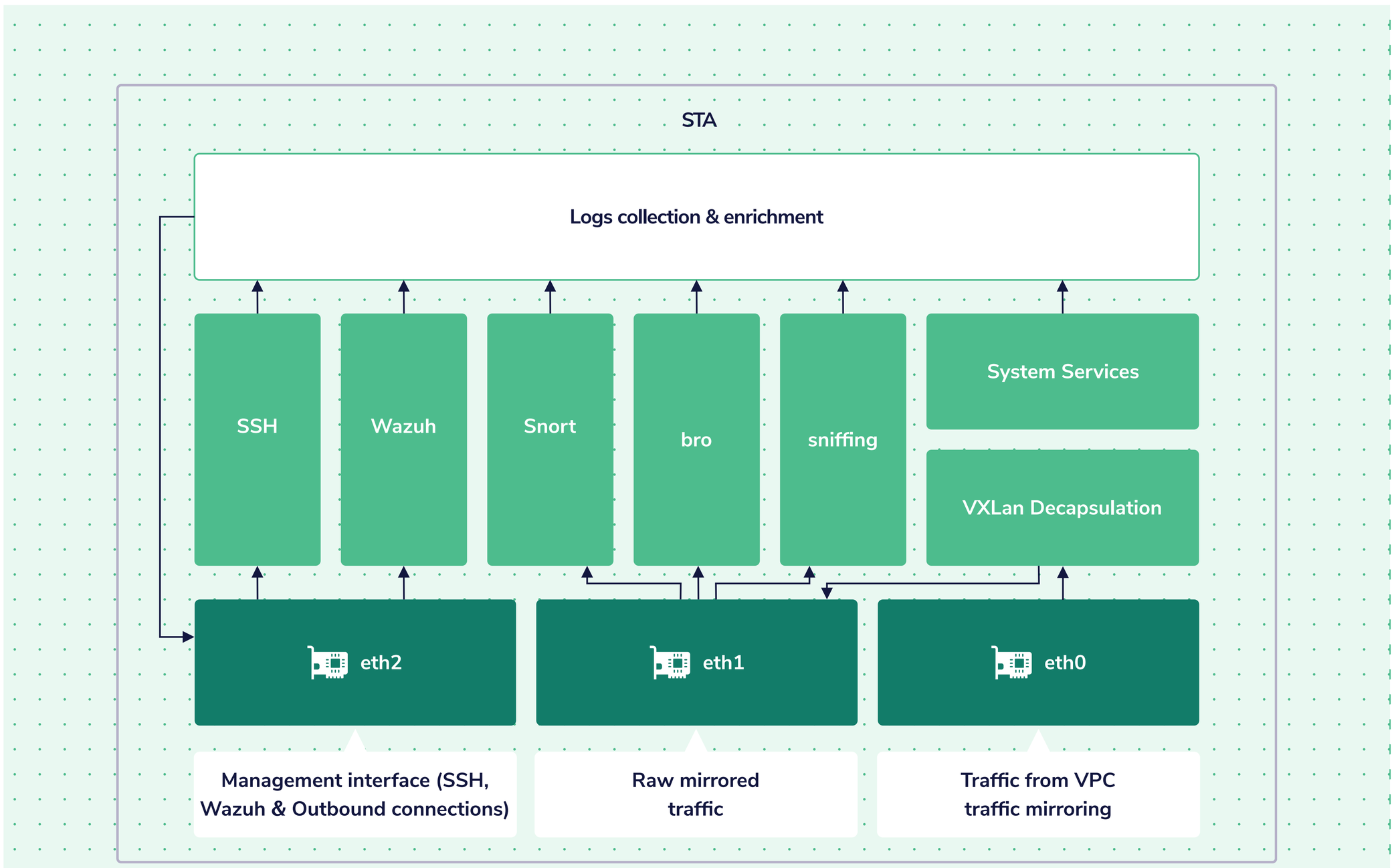
You can mirror your server traffic to Coralogix by using AWS VPC traffic mirroring and Coralogix will automatically capture, analyze, and optionally store the traffic for you while creating meaningful logs, alerts, and dashboards in your Coralogix account.

Coralogix sets up traffic mirroring quickly with a VPC Traffic Mirroring Configuration Automation handler which automatically updates your VPC traffic mirroring configuration based on instance tags and tag values in your AWS account.

The solution provides access to all of your data being transferred between your servers and other cloud-based infrastructure, as well as automated analysis using ML-powered algorithms that alert you to potential threats with the complete ability to tune them, disable them if needed, and easily create any type of new alerts.

In addition, Coralogix automatically enriches the data passing through it such as domain names, certificate names, and much more by using data from several other data sources, allowing you to create more meaningful alerts and reduce false-positives to the minimum while not increasing false-negatives.

The following diagram describes the structure of the STA:



Optimize Costs

With the Coralogix Optimizer, you can reduce up to 70% of storage costs without sacrificing full security coverage and real-time monitoring. This new model enables you to get all of the benefits of an ML-powered logging solution at only a third of the cost and with more real-time analysis and alerting capabilities than before.

Security As Code

Since Coralogix Cloud Security doesn't use any collectors or configurations, it can be deployed as part of your AWS VPC template to assure any new environment set up anywhere will be labeled, observed, and secured.

Attack Prevention vs. Detection

A question we're often asked regarding Coralogix Cloud Security is: Why use Coralogix Cloud Security to **detect** attacks that I can **prevent** by using better firewalls, WAFs, and the like?

The answer is simple: Suppose you would like to install a new burglar alarm at home and you come home one day from work and find out that someone stole several items from your home, without a good set of cameras that document everything going on, how would you better prepare for the next burglar? After all, you can only guess how they circumvented your sophisticated burglar alarm and you will never know for sure which items were stolen.

This is exactly what Coralogix Cloud Security is for, it is not intended as a replacement of your firewalls, security groups, WAFs, and other security components you might have but to complement them by providing a complete high-resolution picture of what actually went on in your network so that even if you will get attacked at some point you'll be able to know exactly what happened and how did it happen and plan better to prevent similar scenarios from happening again.

With that being said, it is worth noting that although Coralogix focuses on detection rather than prevention, it is still possible to achieve both detection and better prevention by integrating Coralogix with an orchestration platform such as Cortex XSOAR. This platform has a production-ready integration with Coralogix that allows you to pull incidents from Coralogix and create automated response playbooks that will use other systems in your organization to achieve better attack prevention as well.

Flexibility & Customizability

Good security is always a tailor-made thing. There are few rules that are right for every organization that seeks a better security posture but the majority of good security rules are organization specific. Every organization has things that make it unique. Many of these can be quite useful when you try to catch malicious activity in your organization – both external and internal. By using this deep internal knowledge to your advantage, you'll essentially convert the problem from a technological one to a pure old-school intelligence problem, forcing attackers to have a much more intimate understanding of your organization in order to be able to hide their tracks effectively. These things can be technological in nature or based on your organization's particular working habits and internal guidelines.

The Coralogix Cloud Security solution, unlike many other solutions on the market today, comes with a predefined set of alerts, dashboards and Snort rules but also with the ability to change all of them and tailor them to suit your organization's needs.

One of the most painful issues that usually deters people from using an IDS solution is that they are notorious for their high false-positive rate, but Coralogix makes it unbelievably easy to solve these kinds of issues.

A default alert becomes too noisy for your organization? Just go into the alerts screen in Coralogix and set a different threshold for that alert. If you are not sure what the correct threshold should be – just set the alert to use ML algorithms to detect and alert you on anomalies in this alert ratio and the system will learn automatically the normal behavior of this alert and alert only when the current behavior deviates from it.

You can learn more about alert types in Coralogix in the following links:

User Defined Alerts
Dynamic Alerts
Ratio Alerts

Even parsing data from your packets for easier filtering and alerting is a matter of seconds. You can create simple rules for extracting important pieces of data from the STA logs into fields, structuring the logs in a different way and much more - you can read more about it here.

A snort rule is too noisy for your organization? Also not a problem – Just update a couple of files on an S3 bucket and the old rule is now disabled and is replaced with your own rule.

And dashboard editing is just a matter of simple 2-3 clicks and you're done.

Default Alerts

Once you install Coralogix Cloud Security, we can update your account with a set of default alerts built specifically for use with the Coralogix. Here's a quick overview of these alerts and their purpose:

STA – NIDS alert detected

This alert will fire every time a security-related issue is detected by Coralogix's Snort engine. You can tune this alert by either modifying the alert's Lucene query or by modifying the disablesid.conf file (to disable the current signature) and then to create a new one in the local.rules file on Coralogix's config S3 bucket. To learn more about how to do that see here.

STA – Unusual reconnaissance activity detected

This alert will fire when Coralogix detects an abnormal rate of Coralogix events indicating that the organization is being scanned from the outside.

STA – Unusual connections rate from blacklisted IPs

This alert will fire when Coralogix detects an abnormal rate of Coralogix events indicating a connection from a potentially malicious IP address.

STA – Request for public IP echo services detected

Many malicious tools will attempt to discover their public IP address. Some will attempt to do that to detect where they are located on the globe, others will use this information for registration with their Command & Control server. This alert will fire when Coralogix, based on Coralogix logs, detects a connection to several sites often used for this purpose by malicious tools.

STA – Trojan activity detected

This alert will fire when the Snort engine of Coralogix detects a Trojan attempt.

STA is not seeing any traffic – MIRRORING is DOWN

This alert will fire when Coralogix detects, based on the logs from Coralogix that it is alive but is not seeing any traffic. This can indicate that there's a problem with your VPC Traffic Mirroring configuration. In the last version of Coralogix, we also published a tool for automating the VPC Traffic Mirroring configuration which can help to fix the problem. You can find more about it here.

STA – Usage of rarely used DNS record types detected

Some DNS record types, like A, AAA, and MX are very commonly used while others like TXT and ISDN are almost never used. Some of those can also be used (or even preferred) by an adversary for a DNS tunneling attack. This alert will fire when an attempt is made to use such a record type. If your organization uses such records for legitimate purposes you can simply remove it from the alert query and possibly create a new alert that will fire only if the rate of DNS requests for that specific record type is abnormal.

STA – Unusual high volume of DNS requests returned NXDOMAIN

This alert would fire when Coralogix detects an abnormal rate of NXDOMAIN responses by DNS servers based on Coralogix logs. Many types of attacks nowadays are some sort of a connection with a Command & Control server. The common way for malware to connect to its Command & Control server today is by using a machine-generated domain name – a.k.a Domain Generation Algorithm (DGA). The way it usually works is that the attacker programs the malware to attempt to generate a domain name based on the current date every day and attempt to reach it and if it fails –

to use the domain that was used until now. That way, if someone would block the access to the Command & Control domain the attacker would simply have to register the domain name that the malware will look for tomorrow and the connection will automatically be restored. Such a strategy would lead to an abnormal rate of DNS requests resulting in NXDOMAIN responses (since the malware will continuously look for domains that are not registered).

STA – DNS activity on TCP detected

DNS most commonly runs on the UDP protocol on port 53. DNS uses TCP in two main scenarios: Domain transfer and for sending large TXT requests. The first one should not be used by unauthorized personnel and definitely not very often and the latter should almost never happen. This alert will fire when such activity (DNS over TCP is detected)

STA – Access to a baby domain was detected

Employees and even more so, servers that are accessing domains that are “young” in the sense that they were registered only very recently are often good indications of malicious activity. This alert fires when access to a domain that was created less than three months has been detected.

STA – BRO Notice Detected

This alert fires when the Bro (a.k.a Zeek) engine in Coralogix has detected anomalous and potentially malicious behavior.

STA – Unusual TOR nodes connectivity

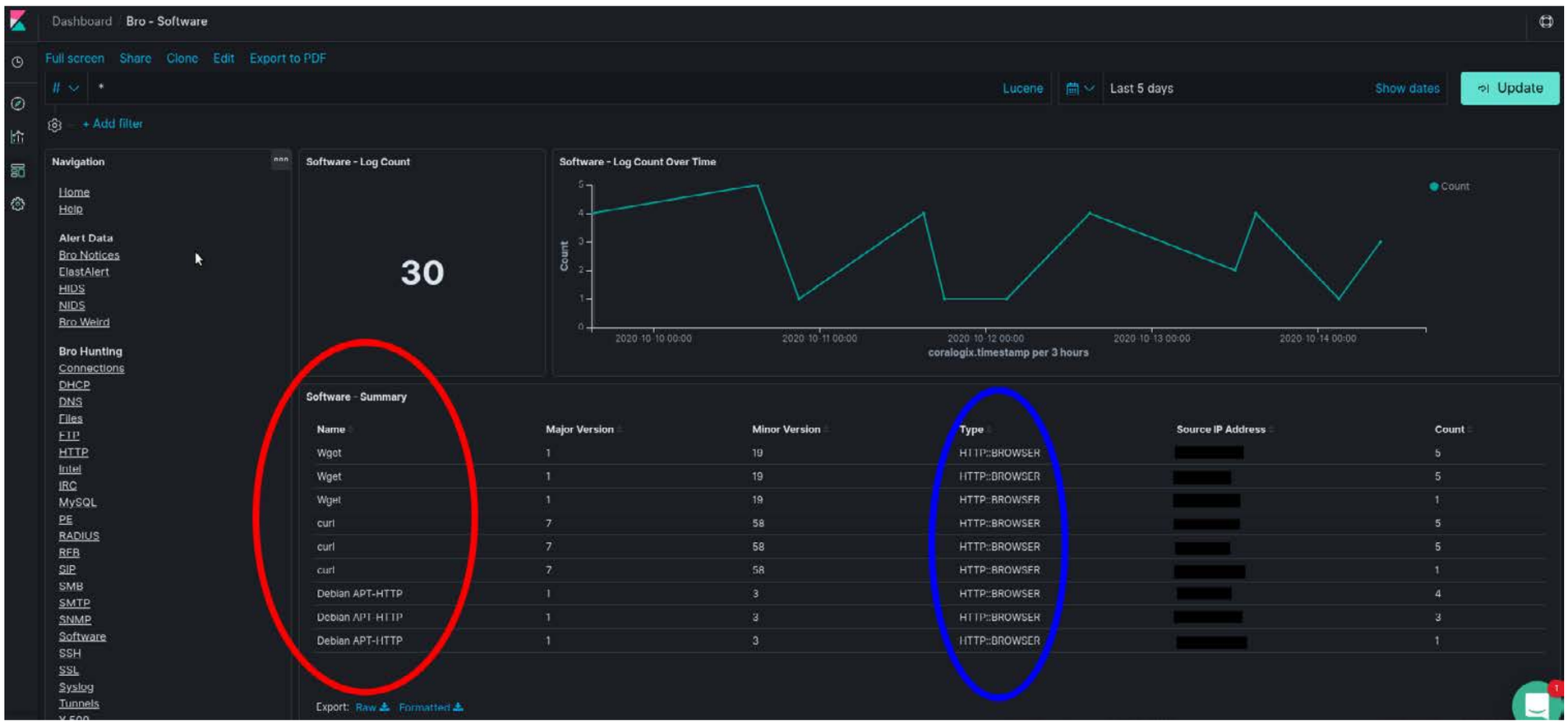
Tor browsers and the Tor network in general are notorious for their malicious usage in hiding adversary actions. This alert would fire when Coralogix detects an anomaly in the rate of connections from Tor nodes (as detected by the Snort engine in Coralogix)

STA is OFFLINE

This alert would fire when no logs are coming from the Coralogix STA

STA – Unrecognized software

The Bro (a.k.a Zeek) engine of Coralogix STA can detect software running on monitored (and unmonitored) servers by deducing them based on the traffic observed by Coralogix. These findings appear on the Software dashboard of Coralogix. This alert is a stub that you can use to whitelist software that you do use (based on what you saw on the software dashboard) and alert on everything new. This is the software dashboard of Coralogix:



STA – Unrecognized software type

See “Coralogix – Unrecognized software” above.

Dashboards

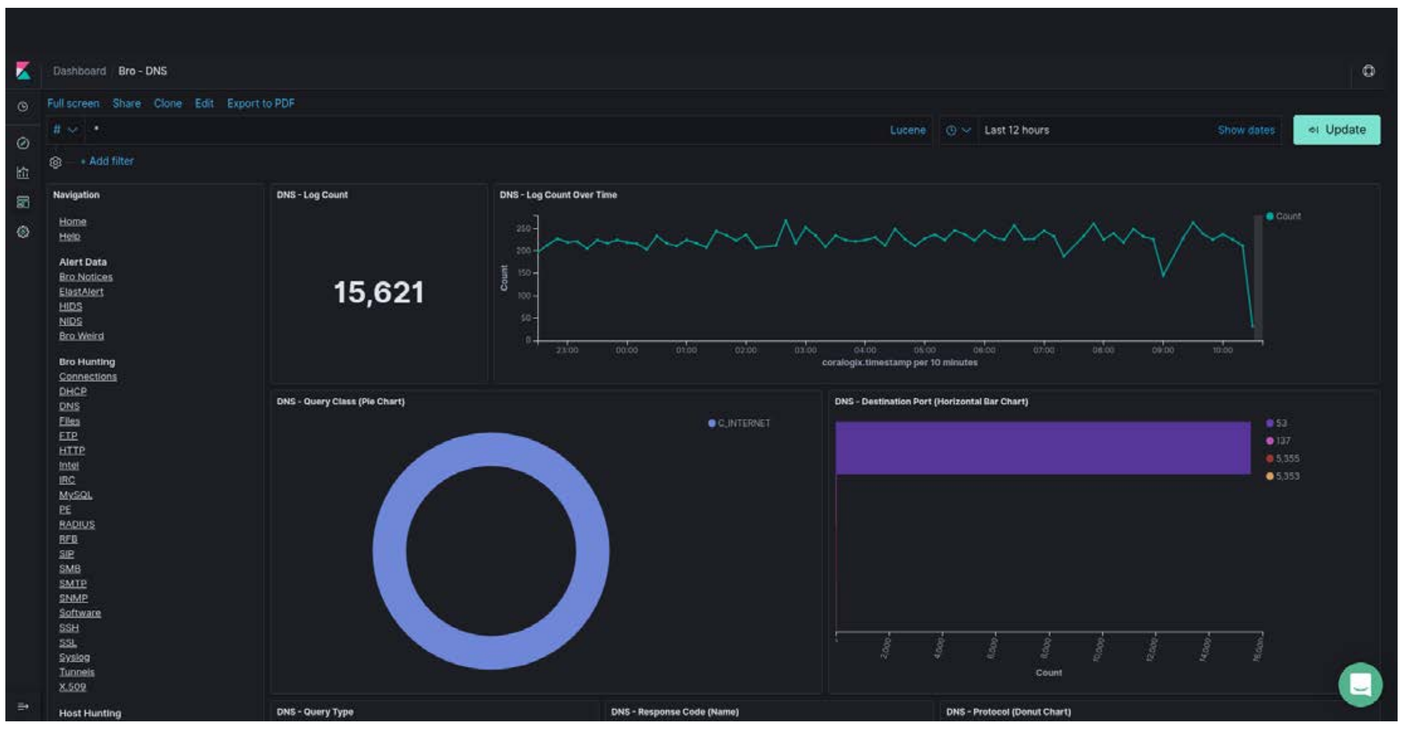
After installing Coralogix Cloud Security for the first time the following dashboards will be automatically added to your account:

Baby Domains

Displays young domains that were created in the last three months that were accessed by the monitored servers.

Bro – Connections

Displays information about connections observed by Coralogix. Includes information such as protocols (e.g. TCP, UDP, ICMP), ports, connection states (e.g. SYN, ACK, PSH, FIN), geographic locations.



Bro – DNS

Displays information about DNS queries and answers detected in the monitored traffic. Includes information such as queries, DNS servers, response codes, geographic locations, protocols, domain names, domains creation date.

Bro – FTP

Displays information about FTP connections detected in the monitored traffic. Includes information such as FTP commands, file paths.

Bro - Files

Displays information about files that were detected in the network traffic in any protocol. Includes information such as file hashes, MIME types, source IPs.

Bro - HTTP

Displays information about HTTP connections, includes information such as hostnames, web methods, URIs, amount of bytes transferred, HTTP referrers, user-agents, MIME types.

Bro - DHCP

Displays information about DHCP connections observed in the traffic.

Bro - IRC

Displays information about IRC connections observed in the traffic.

Bro - MySQL

Displays information about MySQL connections found in the traffic. Includes information such as database names.

Bro - RDP

Displays information about Remote Desktop Protocol (RDP) connections found in the monitored traffic.

Bro - SIP

Displays information about SIP connections (VOIP) found in the monitored traffic.

Bro - SMB

Displays information about SMB connections found in the monitored traffic.

Bro - SMTP

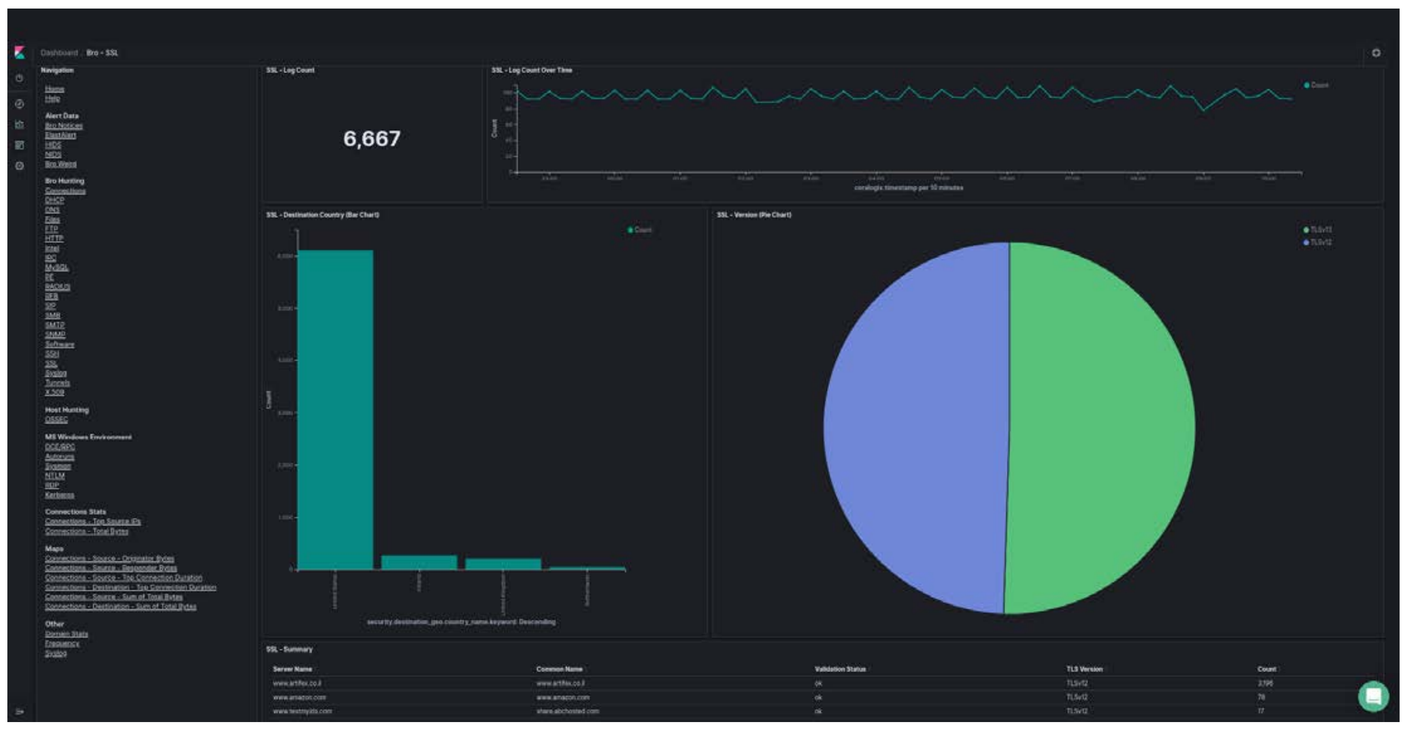
Displays information about Simple Mail Transmission Protocol (SMTP) connections found in the monitored traffic.

Bro - SNMP

Displays information about SNMP connections found in the monitored traffic.

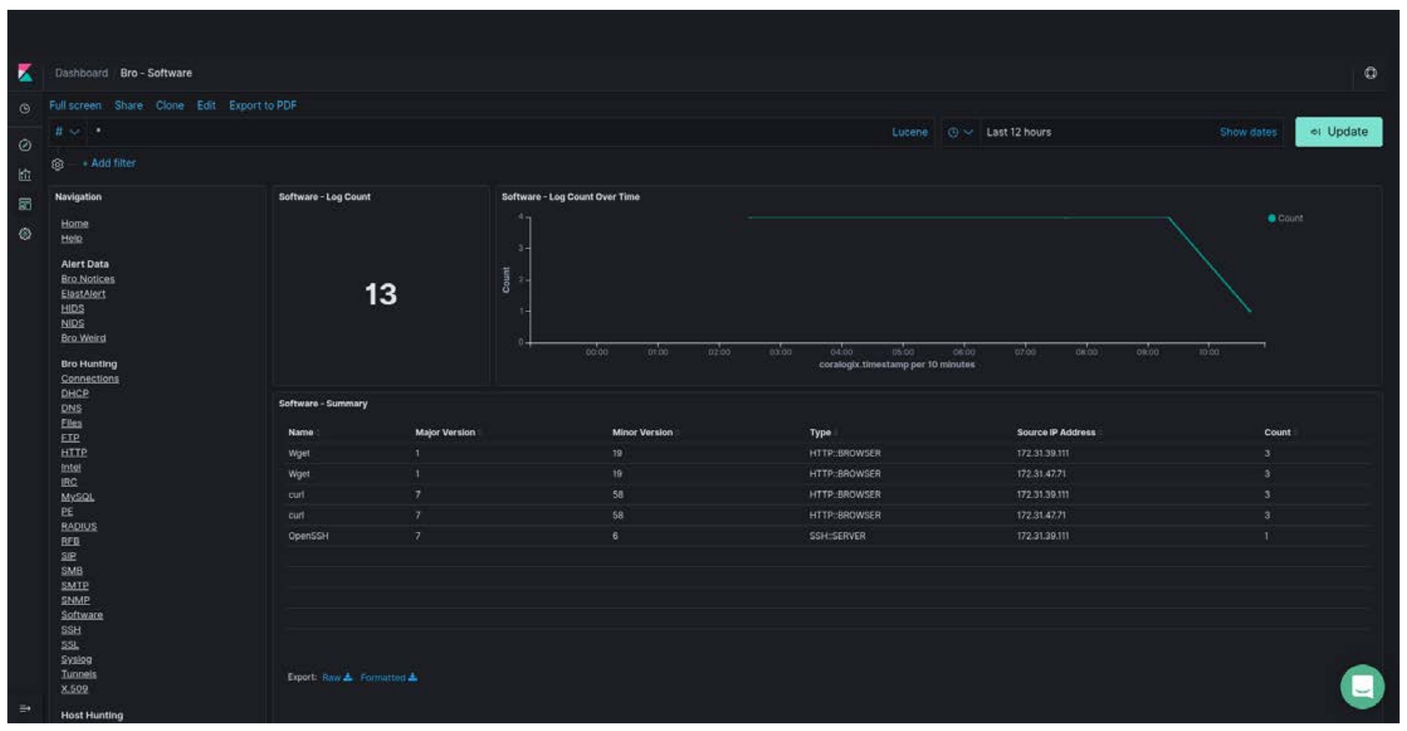
Bro - SSH

Displays information about SSH connections found in the monitored traffic. This data is partially based on packet size analysis.



Bro - SSL

Displays information about SSL/TLS certificates detected in the traffic. Includes information such as certificates details (issuer, CN, validation status)



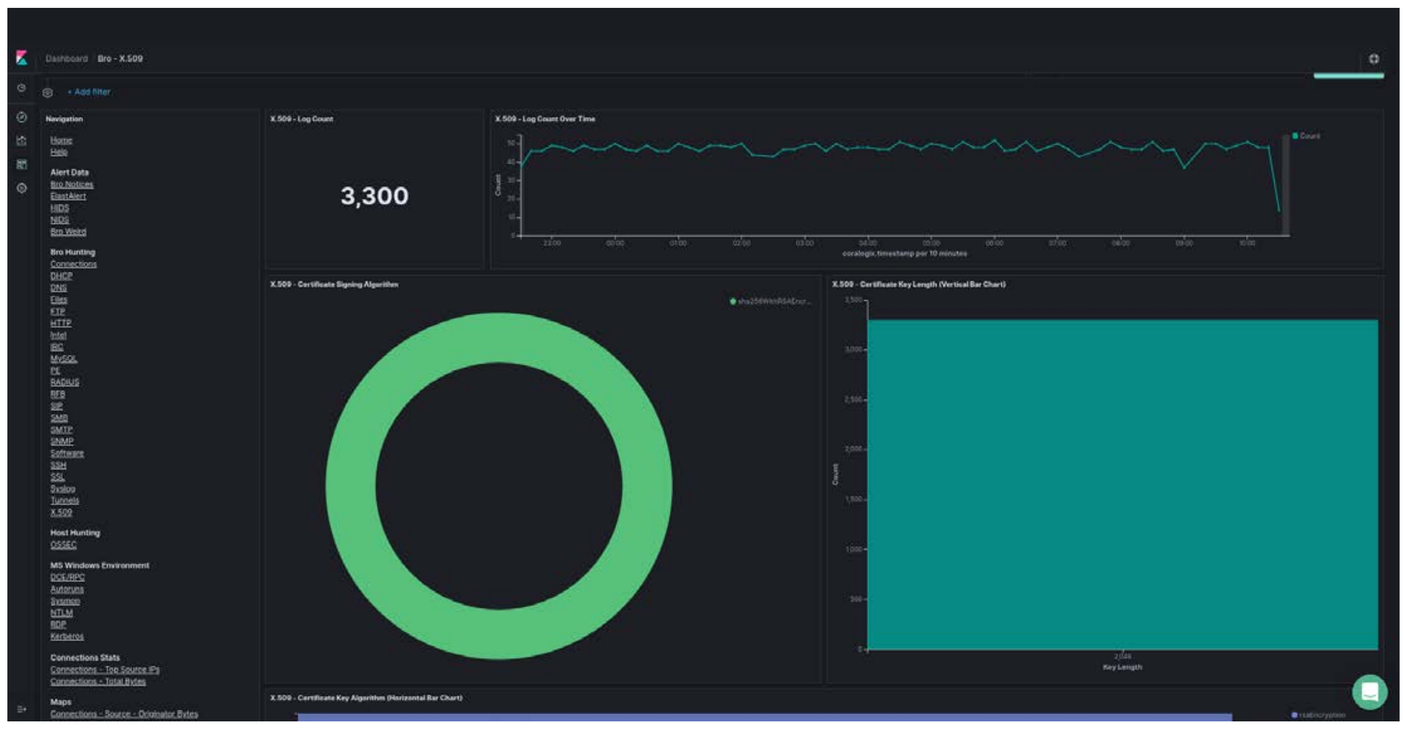
Bro - Software

Displays information about software types of servers and clients as detected in the observed traffic by specific protocol analyzers. Includes

information such as user agents, SSH clients and servers, database servers.

Bro - Tunnels

Displays information about tunnelled connections as detected in the monitored traffic.



Bro - X.509

Displays information about X.509 certificates that have traversed the network.

Connections - Destination - Sum of Total Bytes

Displays a world map with colored dots that their size represents the sum of total bytes transferred to that geographic region.

Connections - Destination - Top Connection Duration

Displays a world map with colored dots that their size represents the top connection duration to that geographic region.

Connections - Source - Originator Bytes

Displays a world map with colored dots that their size represents the sum of originator bytes transferred from that geographic region.

Connections - Source - Responder Bytes

Displays a world map with colored dots that their size represents the sum of responder bytes transferred from that geographic region.

Connections - Source - Sum of Total Bytes

Displays a world map with colored dots that their size represents the sum of total bytes transferred from that geographic region.

Connections - Source - Top Connection Duration

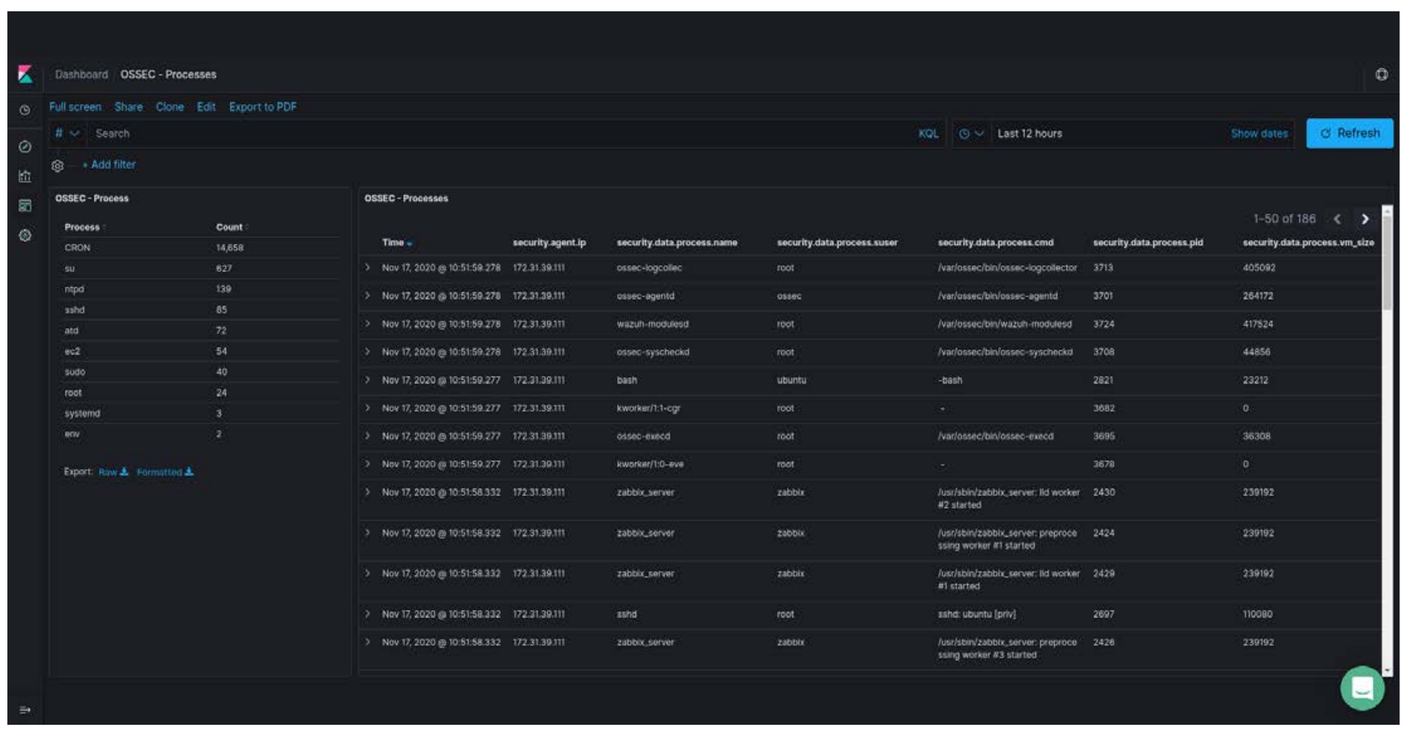
Displays a world map with colored dots that their size represents the top connection duration from that geographic region.

Connections - Top Source IPs

Displays a pie chart of the top source IP addresses with some details.

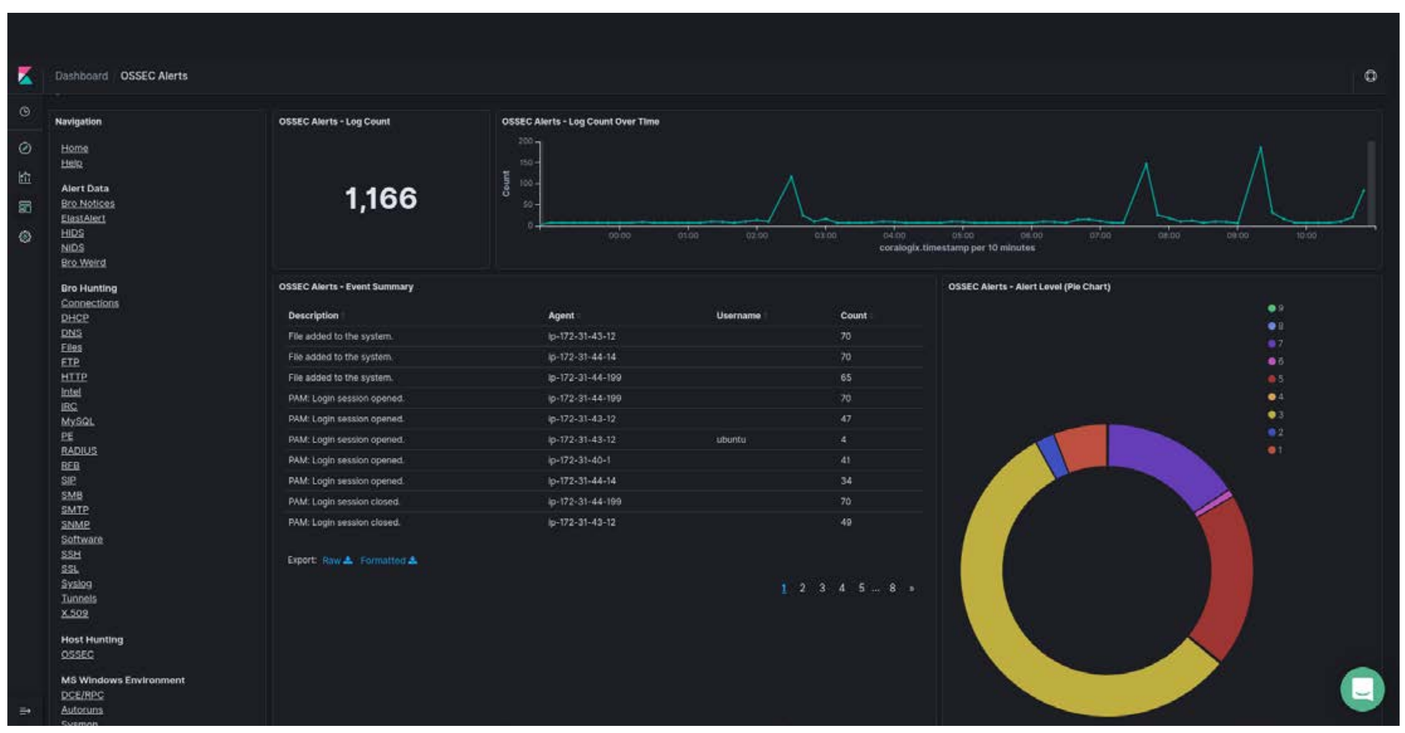
Connections - Total Bytes

Displays total number of bytes transferred by some aggregations such as: by source IP, by destination IP, by Connection, by destination port.



OSSEC - Processes

If Wazuh integration is enabled and configured, displays the processes running on all nodes that run the Wazuh agent and report to Coralogix.



OSSEC Alerts

If Wazuh integration is enabled and configured, displays Wazuh alerts from Wazuhs running on all nodes that run

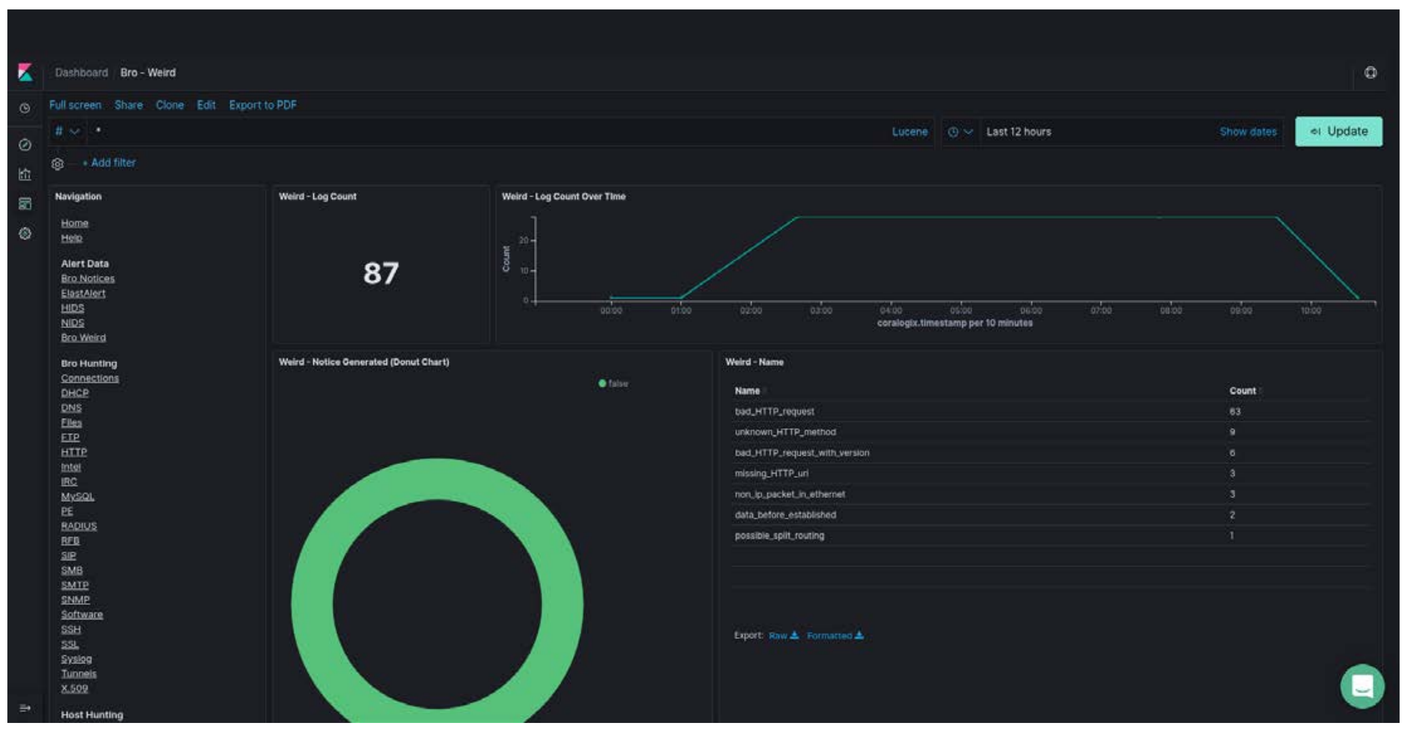
the Wazuh agent and report to Coralogix.

Frequency Analysis

Displays statistics regarding the NLP based score on all domains observed in the traffic.

Bro - Notices

Displays security issues that were detected by Zeek.

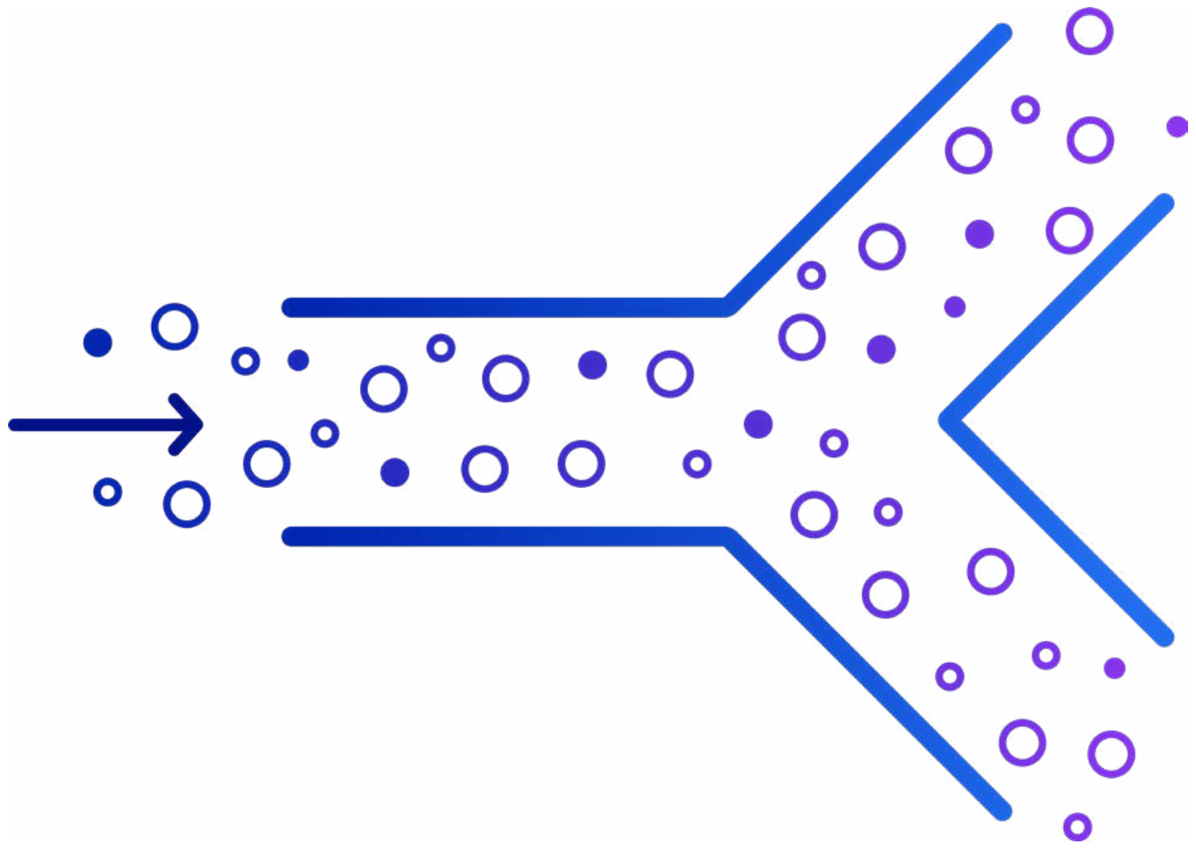


Bro - Weird

Displays possible security issues that were detected by Zeek. Issues presented here not necessarily mean that there's a security issue with your network but it is an indication that something might need to be investigated.

NIDS

Displays security issues that were detected by Snort.



Traffic Mirroring Strategies

In order to be able to detect everything, you have to capture everything and in order to be able to investigate security issues thoroughly, you need to capture every network packet.

More often than not, the data once labeled irrelevant and thrown away is found to be the missing piece in the puzzle when slicing and dicing the logs in an attempt to find a malicious attacker or the source of an information leak.

However, as ideal as this might be, in reality, capturing every packet from every workstation and every server in every branch office is usually impractical and too expensive, especially for larger organizations. Just like in any other field of security, there is no real right or wrong here, it's more a matter of whether or not it is worth the trade-off in particular cases.

There are several strategies that can be taken to minimize the overall cost of the AWS traffic monitoring solution and still get acceptable results. Here are some of the most commonly used strategies:

Mirror by Importance of Resource/Information

Guidelines: After mapping out the most critical assets for the organization from a business perspective, configure the mirroring to mirror only traffic to and from the most critical servers and services to be mirrored and analyzed. For example, a bank will probably include all SWIFT related servers, for a software company it will probably include all traffic from and to their code repository, release location, etc.

Rationale: The rationale behind this strategy is that mirroring the most critical infrastructures will still provide the ability to detect and investigate security issues that can harm the organization the most and will save money by not mirroring the entire infrastructure.

Pros: By following this strategy, you will improve the visibility around the organization's critical assets and should be able to detect issues related to your organization's "crown jewels" (if alerts are properly set) and to investigate such issues.

Cons: Since this strategy won't mirror the traffic from non-crown jewels environments, you will probably fail to pinpoint the exact (or even approximate) path the attacker took in order to attack the organization's "crown jewels".

Tips: If your organization uses a jump-box to connect to the "crown jewels" servers and environments, either configure the logs of that jump-box server to be as verbose as possible and store them on Coralogix with a long retention period or mirror the traffic to the jump-box server.

Mirror by Risk of Resource/Information

Guidelines: After mapping out all the paths and services through which the most critical data of the organization is being transferred or manipulated, configure the mirroring to mirror only traffic to and from those services and routes. The main difference between this strategy and the one mentioned above is that it is focused on sensitive data rather than critical services as defined by the organization.

Rationale: The rationale behind this strategy is that mirroring all the servers and services that may handle critical information will still provide the ability to detect and investigate security issues that can harm the organization the most and will save money by not mirroring the entire infrastructure.

Pros: You will improve the visibility around the critical data across services and environments and you should be able to detect, by configuring the relevant alerts, attempts to modify or otherwise interfere with handling and transferring the organization's sensitive data

Cons: Since this strategy won't mirror traffic from endpoints connecting to the services and paths used for transmission and manipulation of sensitive data, it might be difficult or even impossible to detect the identity of the attacker and the exact or even approximate path taken by the attacker.

Tips: Collecting logs from firewalls and WAFs that control the connections from and to the Internet and sending the logs to Coralogix can help a great deal in creating valuable alerts and by correlating them with the logs from Coralogix can help identify the attacker (to some extent) and his/her chosen MO (Modus Operandi).

Mirror by Junction Points:

Guidelines: Mirror the data that passes through the critical "junction points" such as WAFs, NLBs or services that most of the communication to the organization and its services goes through.

Rationale: The idea behind this strategy is that in many organizations there are several "junction points" such as WAFs, NLBs or services that most of the communication to the organization and its services goes through. Mirroring this traffic can cover large areas of the organization's infrastructure by mirroring just a handful of ENIs.

Pros: You will save money on mirroring sessions and avoid mirroring some of the data while still keeping a lot of the relevant information.

Cons: Since some of the data (e.g. lateral connections between servers and services in the infrastructure) doesn't necessarily traverse the mirrored junction points, it won't be mirrored which will make it harder and sometimes even impossible to get enough information on the attack or even to be able to accurately detect it.

Tips: Currently, AWS cannot mirror an NLB directly but it is possible and easy to mirror the server(s) that are configured as target(s) for that NLB. Also, you can increase the logs' verbosity on the non-monitored environments and services and forward them to Coralogix to compensate for the loss in traffic information.

Mirror by Most Common Access Paths:

Guidelines: Mirror traffic from every server based on the expected and allowed set of network protocols that are most likely to be used to access it.

Rationale: The idea behind this strategy is that servers that expose a certain service are more likely to be attacked by that same service. For example, an HTTP/S server is more likely to be attacked via HTTP/S than via other ports (at least at the beginning of the attack). Therefore, it makes some sense to mirror the traffic from each server based on the expected traffic to it.

Pros: You will be able to save money by mirroring just part of the traffic that arrived or was sent from the organization's servers. You will be able to detect, by configuring the relevant alerts, some of the indications of an attack on your servers.

Cons: Since you mirror only the expected traffic ports, you won't see unexpected traffic that is being sent or received to/from the server which can be of great value for a forensic investigation.

Tips: Depending on your exact infrastructure and the systems and services in use, it might be possible to cover some of the missing information by increasing the services' log verbosity and forwarding them to Coralogix.

Mirror Some of Each:

Guidelines: Randomly select a few instances of each role, region or subnet and mirror their traffic to Coralogix.

Rationale: The idea behind this strategy is that it would be reasonable to assume that the attacker would not know which instances are mirrored and which are not, and also, many tools that are used by hackers are generic and will try to propagate through the network without checking if the instance is mirrored or not, therefore, if the attacker tries to move laterally in the network (manually or automatically), or to scan for vulnerable servers and services, it is very likely that the attacker will hit at least one of the mirrored instances (depending on the percentage of instances you have selected in each network region) and if alerts were properly configured, it will raise an alert.

Pros: A high likelihood of detecting security issues throughout your infrastructure, especially the more generic types of malware and malicious activities.

Cons: Since this strategy will only increase the chances of detecting an issue, it is still possible that you will "run out of luck" and the attacker will penetrate the machines that were not mirrored. Also, when it comes to investigations it might be very difficult or even impossible to create a complete "story" based on the partial data that will be gathered.

Tips: Since this strategy is based on a random selection of instances, increasing the operating system and auditing logs as well as other services logs and forwarding them to Coralogix for monitoring and analysis can sometimes help in completing the picture in such cases.

Recommended Mirroring Baseline

In addition to the specific strategy you'll develop, we also recommend you mirror the following traffic. These will probably cost you near nothing but can be of great value when you'll need to investigate an issue or detect security issues (manually and automatically):

All DNS traffic

It is usually the tiniest traffic in terms of bytes/sec and packets/sec but can compensate for most black spots that will result in such trade-offs.

Mirror traffic that should never happen

Suppose you have a publicly accessible HTTP server that is populated with new content only by scp from another server. In this case, mirroring should be done on the FTP access to the server, since that FTP is one of the most common methods to push new content to HTTP servers, mirroring FTP traffic to this server and defining an alert on such an issue will reveal attempts to replace the HTTP contents even before they have succeeded. This is just one example, there are many possible examples (ssh or nfs to Windows servers, RDP, SMB, NetBIOS and LDAP connections to Linux servers) you probably can come up with more based on your particular environment. The idea here is that since an attacker doesn't have any knowledge of the organization's infrastructure, the attacker will have to first scan hosts to see which operating systems are running and which services they are hosting, for example by trying to connect via SMB (a protocol mostly used by Windows computers) and if there is a response, the attacker would assume that it is Windows. Of course, the same applies to Linux.

Tighter Cloud Access

In cloud infrastructures, instances and even the mirroring configuration are accessible via the Internet and therefore theoretically allows an attacker to find out whether an instance is mirrored and to act accordingly. Because of this, it is even more important to make sure that access to the cloud management console is properly secured and monitored.

Supported Protocols

Coralogix Cloud Security contains several tools that will analyze the traffic that is observed by Coralogix and will generate three types of data streams:

Alerts

Based on the observed traffic and a set of known malicious signatures and known malicious behaviors, the system will generate logs that indicate an alert.

Context Data

In addition to these alerts, the system will provide a very detailed context information based on its protocol analyzers and the data being captured. This type of data is extremely valuable for forensic investigations as well as for detecting anomalous network and user behaviors.

Raw Packet Data

Coralogix supports uploading raw packet data to an S3 bucket for further analysis. Since these are the raw pcap files they contain any protocol data.

These tools support a plethora of protocols including transmission protocols such as TCP and UDP as well as many application protocols such as FTP, HTTP, IRC, POP3, SMTP, SNMP, NTP, DNS, SSH, MYSQL, SIP, IMAP, RDP, SMB and even to the extent of SCADA protocols such as MODBUS and DNP3.

Handling Encrypted Traffic

More websites are encrypting their traffic by using the TLS protocol, servers are no longer managed in clear text telnet sessions but rather in secure SSH sessions, even emails are now being transmitted in SMTPS sessions. This trend is obviously good for keeping our private information private but it also allows attackers to hide their tracks and makes it harder for blue teams to detect threats in the traffic.

You can use many different services for handling SSL termination and then send decrypted traffic to Coralogix for analysis. But even if you don't, Coralogix can detect lots of things about encrypted traffic without having to decrypt it first.

This is just an example of course, the same is true for other encrypted protocols like SMTPS, FTPS and many more. This is even more true since many encryption schemes today are using TLS under the hood.

The list above only relates to the context data but there is a huge list of Snort and Zeek rules that can detect certain types of malicious activity even in encrypted communications based on things like packet size analysis.

Also, since most of the encrypted communications will first query the DNS for the correct IP address, forwarding the DNS traffic to Coralogix can greatly improve its effectiveness by creating dashboards and alerts based on the DNS traffic as well.

The following is a partial list of what Coralogix can detect in encrypted traffic.

Protocol	Detectable from Encrypted Traffic
SSL (HTTPS, FTPS, SMTPS...)	**Trend & Rate** – The amount of connections and bytes that were transferred over time. This can be used to detect anomalies
	TLS/SSL Version – Access to public services that were expected to be secure and are detected as using an old version of TLS can indicate that they were compromised or that an attacker is trying to cause the server and client to use a weaker type of encryption to allow him/her to eavesdrop the traffic. An attack known as SSL Stripping.
	Source & Destination Countries – Based on IP to GeoLocation enrichment we can detect the geographic location of the source and destination servers
	Certificate Details – Since the certificate that is being used by the server is not encrypted, the STA will analyze it and provide details such as Common Name, Issuer Common Name, Issuer Country, Server name, TLS/SSL version, certificate validation status, Certificate chain length, Certificate cipher details
SSH	**Trend & Rate** – The amount of connections and bytes that were transferred over time. This can be used to detect anomalies
	Server and client software types – In the case of outbound SSH connections from your organization to the internet, an attempt that should be considered suspicious, the client software can provide an indication of the malware and even of the attacking group. In the case of connections to your server, the client software type can indicate whether it's a legitimate client or not.
	Source & Destination Countries – Based on IP to GeoLocation enrichment we can detect the geographic location of the source and destination servers
	Cipher Algorithm – The cipher algorithm used in this SSH session. This can indicate that one of your servers is using an unauthorized or non standard cipher and by that becoming vulnerable
	Number of observed authentication attempts – The number of authentication attempts that were observed by the server. Not all of them necessarily indicate failures but if you see a large number here that probably means that someone is trying to guess your SSH password

Writing Effective Snort Rules

Snort is an open-source network intrusion detection system (NIDS) that provides real-time packet analysis and is part of the Coralogix Cloud Security solution.

```
alert tcp $EXTERNAL_NET any -> 10.200.0.0/24 80 (msg:”WEB-IIS CodeRed v2 root.exe access”; flow:to_server,established; uricontent:”/root.exe”; nocase; classtype:web application-attack; reference:url,www.cert.org/advisories/CA-2001 19.html; sid:1255; rev:7;)
```

alert: tells Snort to report this behavior as an alert (it’s mandatory in rules created for Coralogix).

tcp: means that this rule will only apply to traffic in TCP.

$EXTERNAL_NET: this is a variable defined in Snort. By default, the variable HOME_NET is defined as any IP within these ranges: 192.168.0.0/16,10.0.0.0/8,172.16.0.0/12 and EXTERNAL_NET is defined as any IP outside of these ranges. You can specify IP addresses either by specifying a single IP like 10.200.0.0, an IP CIDR range like 192.168.0.0/16 or a list of IPs like [192.168.0.0/16,10.0.0.0/8]. Just note that spaces within the list are not allowed.

any: in this context, it means “from any source port”, then there’s an arrow ‘->’ which means “a connection to” (there isn’t a ‘<-’ operator, but you can simply flip the arguments around the operator. You can use the ‘<>’ operator to indicate that the connection direction is irrelevant for this rule), then an IP range which indicates the destination IP address and then the **port**. You can indicate a port range by using colon like 0:1024 which means 0-1024. In the round parenthesis, there are some directives for setting the alert message, metadata about the rule, as well as additional checks.

msg: is a directive that simply sets the message that will be sent to Coralogix in case a matching traffic will be detected.

flow: is a directive that indicates whether the content we're about to define as our signature needs to appear in the communication to the server ("to_server") or to the client ("to_client"). This can be very useful if, for example, we'd like to detect the server response that indicates that it has been breached.

established: is a directive that will cause Snort to limit its search for packets matching this signature to packets that are part of established connections only. This is useful to minimize the load on Snort.

uricontent: is a directive that instructs Snort to look for a certain text in the normalized HTTP URI content. In this example, we're looking for a url that is exactly the text "/root.exe".

nocase: is a directive that indicates that we'd like Snort to conduct a case insensitive search.

classtype: is a directive that is a metadata attribute indicating which type of activity this rule detects.

reference: is a directive that is a metadata attribute that links to another system for more information. In our example, the value url,<https://....> links to a URL on the Internet.

sid: is a directive that is a metadata attribute that indicates the signature ID. If you are creating your own signature (even if you're just replacing a built-in rule), use a value above 9,000,000 to prevent a collision with another pre-existing rule.

rev: is a directive that indicates the version of the rule.

There's a lot more to learn about Snort rules which supports RegEx parsing, protocol-specific parsing (just like uricontent for HTTP), looking for binary (non-textual) data by using bytes hex values, and much much more. If you'd like to know more you can start here.

Data Enrichment

The raw packet data that is passed through Coralogix is enriched with the following fields, in addition to protocol specific fields, to facilitate the creation of effective and powerful alerting rules that will accurately detect security related threats:

Connection state (security.connection_state, security.connection_state_description)

Allows you to filter, in alerts as well as in searches, only for successful or unsuccessful connections or connections at a certain stage. For example, by using this field it is possible to detect SYN flooding or SYN based reconnaissance.

Whether the connection is established (security.established)

Allows you to filter, in alerts as well as in searches, only for connections that have succeeded or failed. For example, you might want to measure, in a dashboard, the ratio between established and not established connections to your front end servers to detect connectivity issues to them that may result from or be an indication of DDoS attack that is in progress.

Connection duration (security.duration)

Indicates the connection's duration. Each protocol has a typical connection length, for example HTTP connections are relatively short since it's a connectionless protocol while SSH connections can be much longer.
By identifying the correct threshold for your organization for common protocols you can detect attacks such as slow post attacks which are essentially non-volumetric denial of service attacks.

Private/Public Source/Destination IP indications (security.local_orig, security.local_respond)

Makes it easier to create rules that apply only to outbound or inbound connections.

Bytes transferred from each connected party (security.original_ip_bytes, security.respond_ip_bytes)

Makes it easier to detect large data leakage issues, unusual high number of stale connections and many other important issues.

Source & Destination geographic location details (security.*_geo.city_name, security.*_geo.continent_code, security.*_geo.country_code2, security.*_geo.country_code3, security.*_geo.country_name, security.*_geo.location)

Allows you to create whitelist or blacklist based alerts that will fire when a connection from/to an unexpected geographic location is detected.

NLP based score for domain names (security.frequency_scores, security.highest_registered_domain_frequency_score)

Many types of attacks nowadays rely on a technique known as Domain Generation Algorithm, Coralogix automatically calculates a score for each domain, domain part certificate names and much more. This score is based on the frequency of letter combinations in the text and the expected letter combinations in English. By using this score, in alert rules it would be possible to detect DGA usage.

Domain creation date (security.creation_date)

Coralogix will enrich domain names with the date at which they were created. This is since that younger domains are often involved in malicious activity such as DGAs and phishing attempts. By using this field you can easily create an alert that alerts you if a computer is trying to access young domains.

Domain parts (security.highest_registered_domain, security.parent_domain, security.subdomain, security.tld.subdomain, security.top_level_domain)

Allows you to filter for specific types of domains without having to rely on complex regular expressions.

Domain parts lengths (security.parent_domain_length, security.subdomain_length)

Since DGA tend to be quite long, the length of parts of the domain can be used to create alerts that will help in detection of such cases.

Appendix A - Coralogix Cloud Security Installation

System and environment requirements:

A security group that will be used as the security group of the management interface (eth2) of Coralogix. This security group should at least allow the following communications:

Outbound: To Coralogix on port 443/tcp (to send the data)

Outbound: To anywhere on the Internet on port 43/tcp (whois)

Outbound: To s3.amazonaws.com on port 80/tcp

Outbound: To the relevant endpoints and protocols for your AWS region as detailed in this document: https://docs.aws.amazon.com/general/latest/gr/s3.html

Outbound: To download.docker.com on port 443/tcp

Outbound: To APT repositories on ports 80/tcp and 443/tcp

Inbound: At least from your computer on port 22/tcp

A private key that will be used to connect to Coralogix via SSH. Since Coralogix is usually exposed to highly sensitive information, and SSH access to it is almost never required, we strongly recommend that you'll use a dedicated key for Coralogix and store it only once on a safe place and that you won't make any additional copies of it.

At least one available Elastic IP in your AWS account

An S3 bucket for holding Coralogix configuration files (optional but highly recommended)

An S3 bucket for keeping compressed pcap files (optional but recommended)

The name and ID of the VPC you would like Coralogix to be installed in

The name and ID of the subnet you would like Coralogix to be connected to

If you would like to run Coralogix as a spot fleet, the maximum hourly price you or your organization is willing to pay AWS for the instance of Coralogix

The application and subsystem names you would like to assign the data from this Coralogix instance

Included in the installation

During the installation, the following components will be installed in your AWS account depending on the choices you make on the installation UI:

Spot fleet installation:

IAM role and an instance profile with the following permissions (these are the default permissions and they should be limited manually to specific objects after Coralogix is installed):

sts:AssumeRole for services ec2.amazonaws.com and spotfleet.amazonaws.com – This allows the EC2 and SpotFleet services to use this IAM role.

cloudformation:DescribeStackResource on * – This is required by code in the instance to find out whether it was installed properly.

elasticloadbalancing:DescribeInstanceHealth on * – This is required by code in the instance to find out whether it was installed properly.

elasticloadbalancing:RegisterTargets on * – This is required for attaching two additional ENIs to Coralogix upon first boot of the spot.

ec2:* on * – Required by Coralogix fleet manager to launch new spots or terminate spots. Also, this is required for attaching two additional ENIs to Coralogix upon first boot of the spot.

s3:* on * – Required for S3 access to the config & packets buckets

VPC Traffic Mirror Filter – This object is being created as a stub to help you set up a mirroring session later. Without a mirror session (which is NOT created by the CloudFormation template) this object won't do anything – This mirror filter is built with an empty policy, meaning that by using it as-is in a mirroring session you will essentially mirror every traffic from and to the mirrored ENI.

VPC Traffic Mirror Target – This mirror target points at the sniffing NLB. Without a mirror session (which is NOT created by the CloudFormation template) this object won't do anything and will simply facilitate the creation of a VPC Traffic Mirroring Session.

A Security Group for the capturing interfaces (eth0 and eth1) – These security groups allow the mirroring traffic to reach Coralogix for analysis and is attached (on the first boot of the Coralogix instance) to the first two network interfaces which are used by Coralogix for packet capturing (the first one is for VXLAN encapsulated traffic from AWS VPC Mirroring and the second one is for raw traffic). By default, this security group allows all traffic to these network interfaces. Since there are no daemons listening on these network interfaces this doesn't affect the security of Coralogix.

An AWS ElasticIP which will be used as Coralogix's public IP when sending the logs to Coralogix as well as for other outbound connections. This IP will be associated (on the first boot of Coralogix) with the third network interface and will also be used for connecting to Coralogix via SSH.

An NLB, NLB Listener and an NLB target group which will be used for sending the mirrored traffic to Coralogix. The Coralogix instance will be automatically registered as a target in this NLB target group by the CloudFormation process. Although the instance will appear as unhealthy in this target group this is perfectly normal and expected.

If selected that **Wazuh/AWS Inspector** integration is needed the following will be added:

An NLB, NLB Listener and an NLB target group and two NLB listeners (one for Wazuh registrations and another for Wazuh traffic) will be created during the CloudFormation process and will be attached upon first boot of Coralogix.

A spot fleet – This is the spot fleet that will launch the Coralogix spot instance.

The installation process will also set the SSH key pair for the Coralogix instance to the key pair selected on the CloudFormation form.

Upon first boot Coralogix will associate the security group you've selected on the CloudFormation form with the management network interface (eth2)

On-demand installation

IAM role and an instance profile with the following permissions (these are the default permissions and they should be limited manually to specific objects after Coralogix is installed):

sts:AssumeRole for services ec2.amazonaws.com – This allows the EC2 service to use this IAM role.

cloudformation:DescribeStackResource on * – This is required by code in the instance to find out whether it was installed properly.

elasticloadbalancing:DescribeInstanceHealth on * – This is required by code in the instance to find out whether it was installed properly.

elasticloadbalancing:RegisterTargets on * – This is required for attaching two additional ENIs to Coralogix upon first boot of the spot.

ec2:* on * – Required for attaching two additional ENIs to Coralogix upon first boot of the instance.

s3:* on * – Required for S3 access to the config & packets buckets

VPC Traffic Mirror Filter – This object is being created as a stub to help you set up a mirroring session later. Without a mirror session (which is NOT created by the CloudFormation template) this object won't do anything – This mirror filter is built with an empty policy, meaning that by using it as-is in a mirroring session you will essentially mirror every traffic from and to the mirrored ENI.

VPC Traffic Mirror Target – This mirror target points at the sniffing NLB. Without a mirror session (which is NOT created by the CloudFormation template) this object won't do anything and will simply facilitate the

creation of a VPC Traffic Mirroring Session.

A Security Group for the capturing interfaces (eth0 and eth1) – These security groups allow the mirroring traffic to reach Coralogix for analysis and is attached (on the first boot of the Coralogix instance) to the first two network interfaces which are used by Coralogix for packet capturing (the first one is for VXLAN encapsulated traffic from AWS VPC Mirroring and the second one is for raw traffic). By default, this security group allows all traffic to these network interfaces. Since there are no daemons listening on these network interfaces this doesn't affect the security of Coralogix.

An AWS ElasticIP which will be used as Coralogix's public IP when sending the logs to Coralogix as well as for other outbound connections. This IP will be associated (on the first boot of Coralogix) with the third network interface and will also be used for connecting to Coralogix via SSH.

An NLB, NLB Listener and an NLB target group which will be used for sending the mirrored traffic to Coralogix. The Coralogix instance will be automatically registered as a target in this NLB target group by the CloudFormation process. Although the instance will appear as unhealthy in this target group this is perfectly normal and expected.

If selected that **Wazuh/AWS Inspector** integration is needed the following will be added:

An NLB, NLB Listener and an NLB target group and two NLB listeners (one for Wazuh registrations and another for Wazuh traffic) will be created during the CloudFormation process and will be attached upon first boot of Coralogix.

An EC2 instance – For hosting Coralogix.

The installation process of Coralogix Cloud Security:

Contact Coralogix customer support to enable the Cloud Security feature for your account

Login to your Coralogix account and then navigate to Settings => Cloud Security

Set the application and subsystem names you would like to assign the data from this Coralogix instance

Select whether Wazuh/AWS Inspector is needed

Select the size of Coralogix you'll need (small, medium or large)

Select whether the Coralogix instance should run as a spot fleet or as an on-demand instance

Select the AWS region in which you would like Coralogix to be installed in (normally, the best option is in the same region where the instances you'd like to monitor are)

On another tab, make sure you are logged on with the correct AWS account

Click on the "LAUNCH AWS CLOUDFORMATION" button. You'll be redirected to an AWS CloudFormation form

In the AWS CloudFormation form, choose the instance type you would like to use (applicable for on-demand installations only)

Set the name of the S3 bucket you would like Coralogix to use to update/pull configuration to/from (optional but highly recommended)

Set the SSH key name you would like to use for Coralogix

Set the security group that will be used for the management interface

Set the name of the S3 bucket you would like Coralogix to upload compressed pcap files to. (optional but recommended. If you chose to use it just remember to create a lifecycle cleanup hook on this bucket to prevent it from growing too large.

Set the subnet and VPC IDs you would like Coralogix to be connected to

Tick the box below that says “I acknowledge that AWS CloudFormation might create IAM resources.”

Click “Create stack”. While it is creating all the necessary resources in your AWS account, you can go ahead and install the Automatic VPC Traffic Mirroring handler as described here.

Appendix B - Automate VPC Mirroring Configuration for Coralogix

After installing Coralogix Cloud Security and choosing a mirroring strategy suitable for your organization needs the next step would be to set the mirroring configuration in AWS. However, the configuration of VPC Traffic Mirroring in AWS is tedious and cumbersome – it requires you to create a mirror session per network interface of every mirrored instance.

If you, like many others, use auto-scaling groups to automatically scale your services up and down based on the actual need or spot fleets to minimize costs, the situation quickly becomes completely unmanageable.

Luckily for you, we at Coralogix have already prepared a solution for that problem. The tool we've developed can run as a pod in Kubernetes or inside a Docker container. It is written in Go to be as efficient as possible and will require only a minimal set of resources to run properly.

While it is running it will read its configuration from a simple JSON file and will select AWS EC2 instances by tags and then will select network interfaces on those instances and will create VPC Traffic Mirroring sessions for each network interface selected to the configured VPC Mirroring Target using the configured VPC Mirroring Filter.

The configuration used in this document will instruct sta-vpc-mirroring-manager to look for AWS instances that have the tags "sta.coralogix.com/mirror-filter-id" and "sta.coralogix.com/mirror-target-id" (regardless of

the value of those tags), collect the IDs of their first network interfaces (that are connected as eth0) and attempt to create a mirror session for each network interface collected to the mirror target specified by the tag “sta.coralogix.com/mirror-target-id” using the filter ID specified by the tag “sta.coralogix.com/mirror-filter-id” on the instance that network interface is connected to.

To function properly, the instance hosting this pod should have an IAM role attached to it (or the AWS credentials provided to this pod/container should contain a default profile) with the following permissions:

```
ec2:Describe* on *
elasticloadbalancing:Describe* on *
autoscaling:Describe* on *
ec2:ModifyTrafficMirrorSession on *
ec2:DeleteTrafficMirrorSession on *
ec2:CreateTrafficMirrorSession on *
```

Installation

This tool can be installed either as a Kubernetes pod or a Docker container. Here are the detailed instructions for installing it:

Installation as a docker container:

To download the docker image use the following command:

docker pull coralogixrepo/sta-vpc-mirroring-config-manager:latest

On the docker host, create a config file for the tool with the following content (if you would like the tool to report to the log what is about to be done without actually modifying anything set “dry_run” to true)

```
{
  “service_config”: {
    “rules_evaluation_interval”: 10000,
    “metrics_exporter_port”: “:8080”,
    “dry_run”: false
  },
  “rules”: [
    {
      “conditions”: [
        {
          “type”: “tag-exists”,
          “tag_name”: “sta.coralogix.com/mirror-target-id”
        },
        {
          “type”: “tag-exists”,
          “tag_name”: “sta.coralogix.com/mirror-filter-id”
        }
      ],
      “source_nics_matching”: [
        {
          “type”: “by-nic-index”,
          “nic_index”: 0
        }
      ],
      “traffic_filters”: [
        {
          “type”: “by-instance-tag-value”,
          “tag_name”: “sta.coralogix.com/mirror-filter-id”
        }
      ],
      “mirror_target”: {
        “type”: “by-instance-tag-value”,
        “tag_name”: “sta.coralogix.com/mirror-target-id”
      }
    }
  ]
}
```

Use the following command to start the container:

```
docker run -d \
   -p <prometheus_exporter_port>:8080 \
   -v <local_path_to_config_file>:/etc/sta-pmm/Coralogix-pmm.conf \
   -v <local_path_to_aws_profile>/.aws:/root/.aws \
   -e “STA_PM_CONFIG_FILE=/etc/sta-pmm/Coralogix-pmm.conf” \   coralogixrepo/sta-vpc-mirroring-config-
manager:latest
```

Installation as a Kubernetes deployment

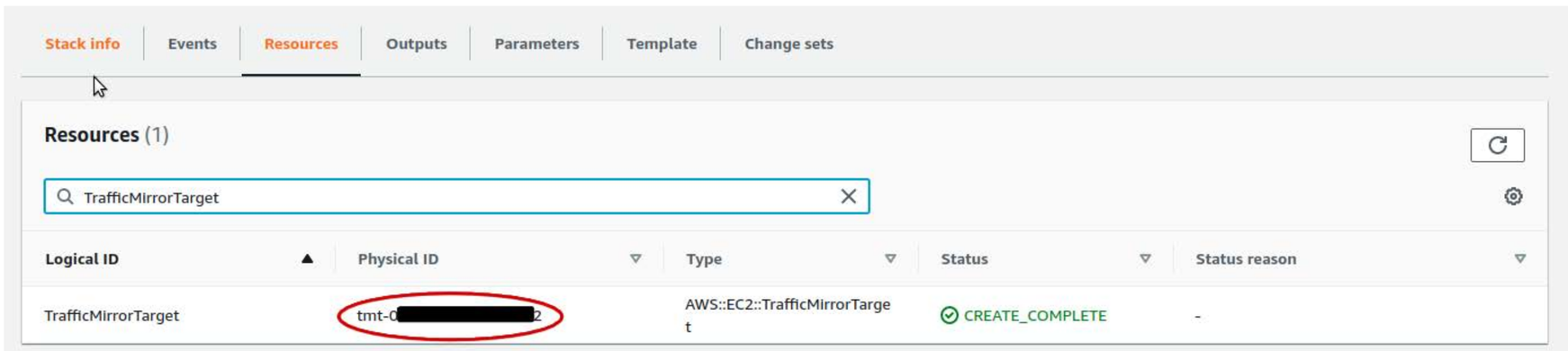
Use the following config map and deployment configurations:

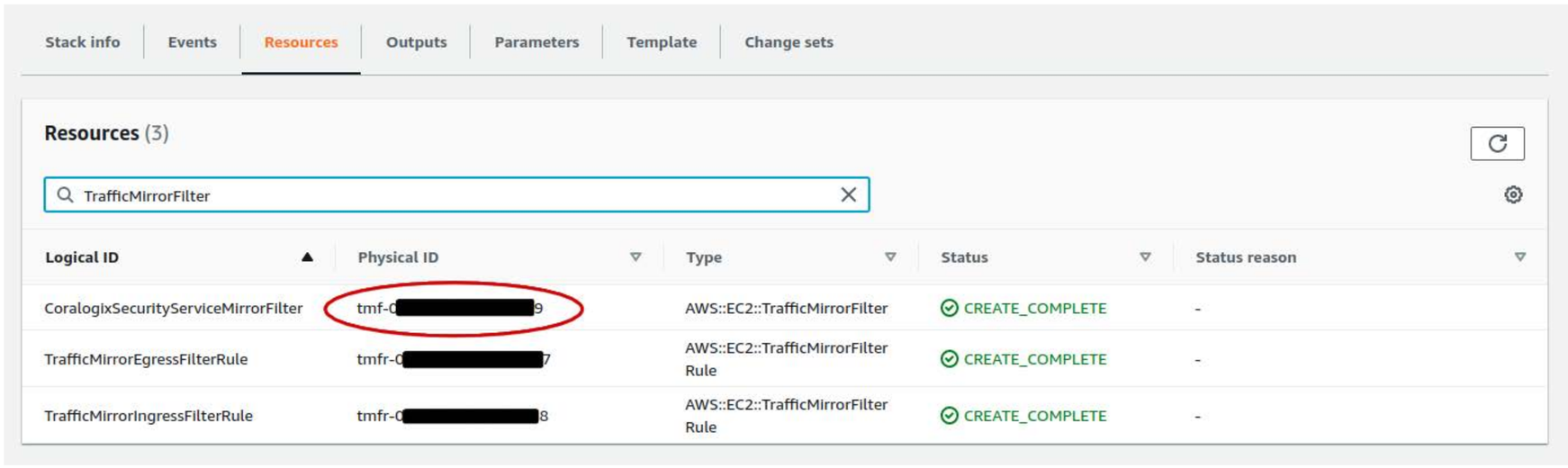
```
apiVersion: v1
kind: ConfigMap
data:
  sta-pmm.conf: |
    {
      “service_config”: {
        “rules_evaluation_interval”: 10000,
        “metrics_exporter_port”: 8080,
        “dry_run”: true
      },
      “rules”: [
        {
          “conditions”: [
            {
              “type”: “tag-exists”,
              “tag_name”: “sta.coralogix.com/mirror-target-id”
            },
            {
              “type”: “tag-exists”,
              “tag_name”: “sta.coralogix.com/mirror-filter-id”
            }
          ],
          “source_nics_matching”: [
            {
              “type”: “by-nic-index”,
              “nic_index”: 0
            }
          ],
          “traffic_filters”: [
            {
              “type”: “by-instance-tag-value”,
              “tag_name”: “sta.coralogix.com/mirror-filter-id”
            }
          ],
          “mirror_target”: {
            “type”: “by-instance-tag-value”,
            “tag_name”: “sta.coralogix.com/mirror-target-id”
          }
        }
      ]
    }
metadata:
  labels:
    app.kubernetes.io/component: sta-pmm
    app.kubernetes.io/name: sta-pmm
    app.kubernetes.io/part-of: coralogix
    app.kubernetes.io/version: ‘1.0.0-2’
  name: sta-pmm
  namespace: coralogix
```

```
---------
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app.kubernetes.io/component: sta-pmm
    app.kubernetes.io/name: sta-pmm
    app.kubernetes.io/part-of: sta
    app.kubernetes.io/version: '1.0.0-2'
  name: sta-pmm
  namespace: sta
spec:
  selector:
    matchLabels:
      app.kubernetes.io/component: sta-pmm
      app.kubernetes.io/name: sta-pmm
      app.kubernetes.io/part-of: sta
  template:
    metadata:
      labels:
        app.kubernetes.io/component: sta-pmm
        app.kubernetes.io/name: sta-pmm
        app.kubernetes.io/part-of: sta
        app.kubernetes.io/version: '1.0.0-2'
      name: sta-pmm
    spec:
      containers:
        - env:
            - name: STA_PM_CONFIG_FILE
              value: /etc/Coralogix-pmm/Coralogix-pmm.conf
            - name: AWS_ACCESS_KEY_ID
              value: XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
            - name: AWS_SECRET_ACCESS_KEY
              value: XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
          image: coralogixrepo/sta-vpc-mirroring-config-manager:latest
          imagePullPolicy: IfNotPresent
          livenessProbe:
            httpGet:
              path: "/"
              port: 8080
            initialDelaySeconds: 5
            timeoutSeconds: 1
          name: sta-pmm
          ports:
            - containerPort: 8080
              name: sta-pmm-prometheus-exporter
              protocol: TCP
          volumeMounts:
            - mountPath: /etc/sta-pmm/Coralogix-pmm.conf
              name: sta-pmm-config
              subPath: sta-pmm.conf
      volumes:
        - configMap:
            name: sta-pmm-config
          name: sta-pmm-config
```

Traffic Mirroring Configuration

To configure instances for mirroring, all you have to do is to make sure that the instances you would like their traffic to be mirrored to your Coralogix Cloud Security, will have the tags “sta.coralogix.com/mirror-filter-id” and “sta.coralogix.com/mirror-target-id” pointing at the correct IDs of the mirror filter and target respectively. To find out the IDs of the mirror target and mirror filter that were created as part of the installation of Coralogix, enter the CloudFormation Stacks page in AWS Console and search for “TrafficMirrorTarget” and for “TrafficMirrorFilter” in the Resources tab:





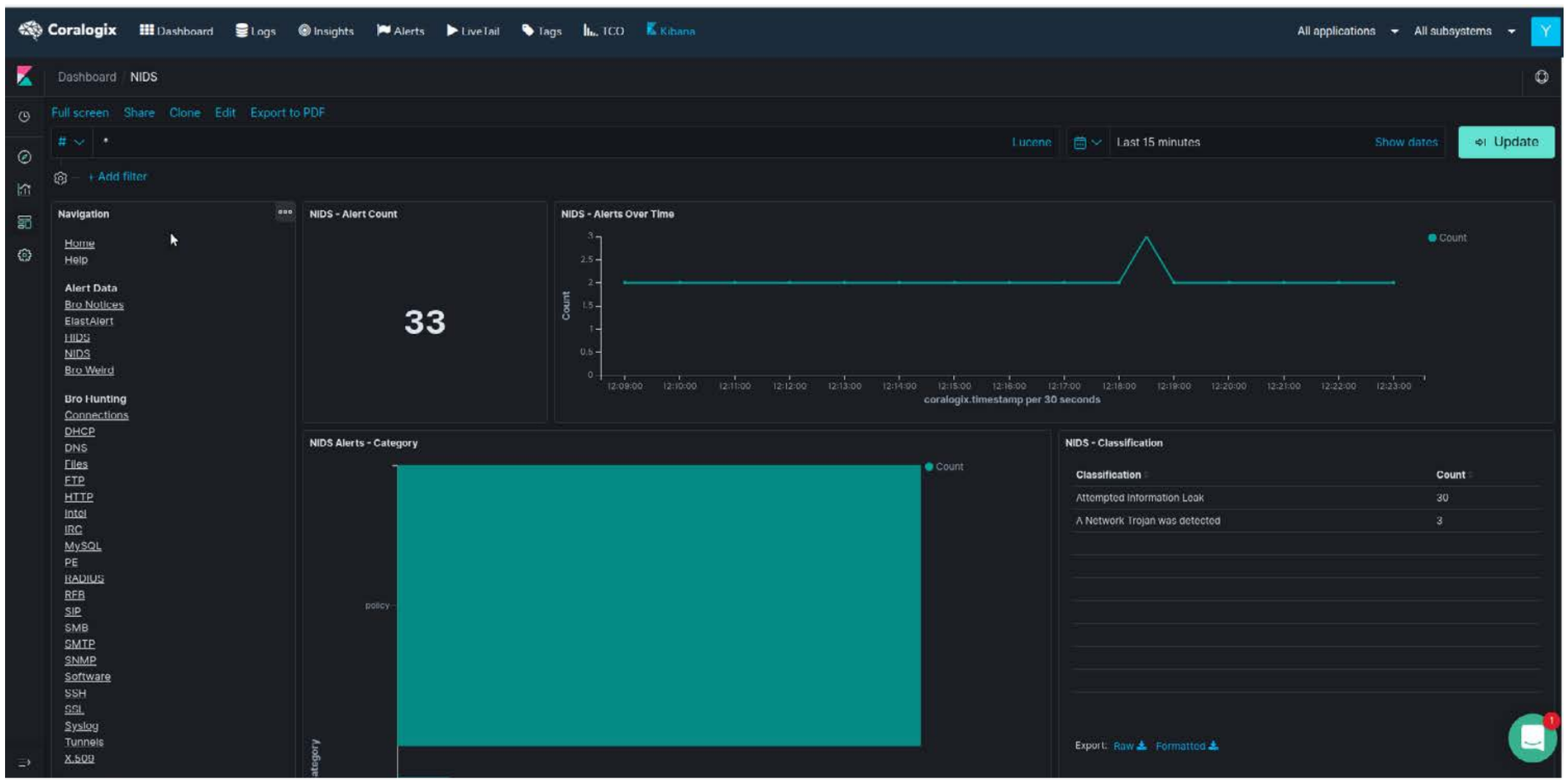
To assign different mirroring policy to different instances, for example to mirror traffic on port 80 from some instances and mirror traffic on port 53 from other instances, simply create a VPC Traffic Mirror Filter manually with the correct filtering rules (just like in a firewall) and assign its ID to the “sta.coralogix.com/mirror-filter-id” tag of the relevant instances.

Pro Tip: You can use AWS “Resource Groups & Tag Editor” to quickly assign tags to multiple instances based on an arbitrary criteria.

Appendix C - Modify Snort Rules

Once you install our Coralogix Cloud Security it will automatically pull a set of snort rules from the Internet and will continue to update them on a daily basis.

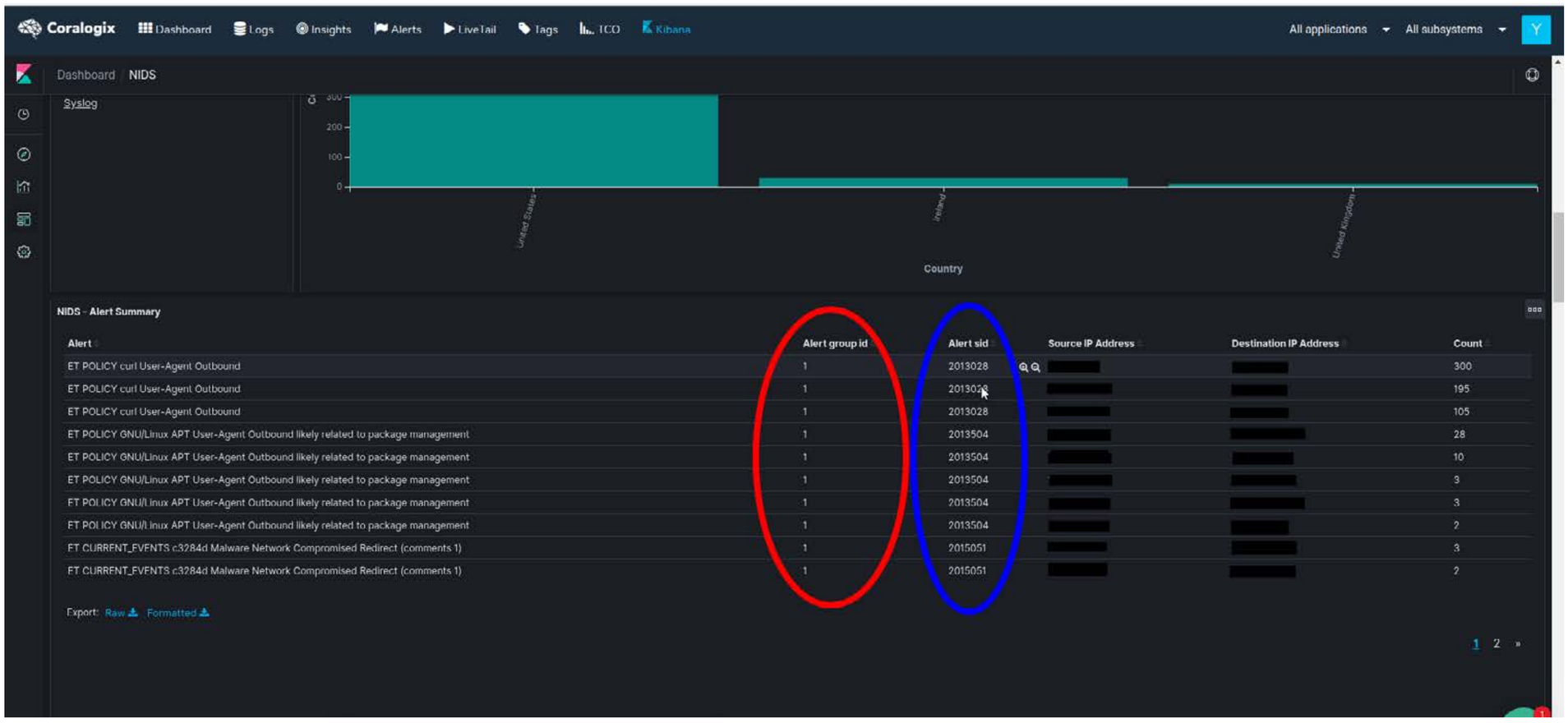
However, in the security business, there's no "one size fits all" solutions. Some of these rules might suit your organization while others might need a few tweaks to get them to work correctly in your environment. In the new version of Coralogix Cloud Security, we added the support for synchronizing its configuration against an S3 bucket. By using this feature, you can easily fine-tune your snort alerts as explained here:



Detect the problematic alert's SID: By navigating to the NIDS dashboard in Kibana you should see a dashboard similar to this one:

If you scroll down a bit, you should be able to see the list of alerts by IP addresses, like the example below. The two columns marked on this

screenshot are the group id (GID) and the snort id (SID) of the alert. We'll need them in the next step to disable the relevant alert and create a new one.



Get the original snort rule: Connect to your Coralogix instance via SSH by using the SSH key you provided during the installation of Coralogix and run the following command (no sudo permissions required):

```
grep '<the snort sid of the alert>' /etc/nsm/rules/downloaded.rules
```

You should get an output similar to this:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:”ET POLICY curl User-Agent Outbound”; flow:established,to_server; content:”User-Agent|3a| curl/”; nocase; http_header; reference:url,www.useragentstring.com/pages/useragentstring.php; classtype:attempted-recon; sid:2013028; rev:2; metadata:created_at 2011_06_14, updated_at 2011_06_14;)
```

Disable the original snort rule: Copy the relevant config files from the S3 bucket you configured for Coralogix to use as its configuration bucket by using the following command:

```
aws --region <your aws region> s3 cp s3://<s3 bucket used for Coralogix configuration>/local.rules /tmp/

aws --region <your aws region> s3 cp s3://<s3 bucket used for Coralogix configuration>/disablesid.conf /tmp/
```

Open the file /tmp/disablesid.conf with your favorite text editor (we highly recommend using either nano or vi/vim) and add lines similar to these at the bottom of the file:

```
# Not relevant for my organization

<alert group id>:<alert snort id>
```

Create a modified version of the original rule (optional): If you only intend to modify an existing rule, copy the rule you found in step 2 above and paste it at the bottom of the local.rules file.

Upload the files back to the S3 bucket: Use the following commands to upload the files to the config S3 bucket (if you only disabled a rule then the second command is not required)

```
aws --region <your aws region> s3 cp /tmp/disablesid.conf s3://<s3 bucket
used for Coralogix configuration>/

aws --region <your aws region> s3 cp /tmp/local.rules s3://<s3 bucket used
for Coralogix configuration>/
```

After you've completed the steps above, your Coralogix Cloud Security will automatically detect the changes and apply them.

Appendix D - Best Practices to Writing Effective Snort Rules

Target the vulnerability, not the exploit - Avoid writing rules for detecting a specific exploit kit because there are countless exploits for the same vulnerability and we can be sure that new ones are being written as you're reading this. For example, many of the early signatures for detecting buffer

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (content:"AAAAAAAAAAAAAA",
msg:"Buffer overrun detected.")
```

The reason for that is of course that to launch a successful buffer overrun attack, the attacker needs to fill the buffer of a certain variable and add his malicious payload at the end so that it would become executable. The characters he chooses to use to fill the buffer are completely insignificant and indeed, after such signatures appeared, many attack toolkits simply used a different letter or letters to fill the buffer and completely evaded this type of signature detection. A much better way would be to attempt to detect these kinds of attacks by detecting incorrect input to fields based on their type and length.

Your peculiarity is your best asset, so use it - Every organization has things that make it unique. Many of these can be quite useful when you try to catch malicious activity in your organization – both external and internal. By using this deep internal knowledge to your advantage, you'll essentially convert the problem from a technological one to a pure old-school intelligence problem, forcing attackers to have a much more intimate understanding of your organization in order to be able to hide their tracks

effectively. These things can be technological in nature or based on your organization's particular working habits and internal guidelines. Here are some examples:

Typical Working Hours: Some organizations I worked at did not allow employees to work from home at all and the majority of employees would have already left the office by 19:00. For similar organizations, it would make sense to set an alert to notify you of connections from the office after a certain hour. An attacker that would install malicious software in your organization would have to know that behavior and tune his malware to communicate with its Command & Control servers at precisely the same time such communications would go unnoticed.

Typical Browser: Suppose your organization has decided to use the Brave browser as its main browser and it gets installed on every new corporate laptop automatically and you have removed the desktop shortcuts to IE/Edge browser from all of your corporate laptops. If this is the case, a connection from the organization, both to an internal as well as external addresses that use a different browser such as IE/Edge should be configured to raise an alert.

IP Ranges based on Roles: If it's possible for you to assign different IP ranges for different servers based on their role, for example, to have all DB servers on 192.168.0.0/24, the web servers on 192.168.1.0/24, etc then it would be possible and even easy to set up clever rules based on the expected behavior of your servers based on their role. For example, database servers usually don't connect to other servers on their own, printers don't try to connect to your domain controllers, etc.

Unusual Connection Attempts: Many organizations use a public share on a file server to help their users share files between them and use network enabled printers to allow their users to print directly from their computers. That means that client computers should not connect to each other, even if you have (wisely) chosen to block such access in a firewall or at the switch, the very attempt to connect from one client to another client computer should raise an alert and be thoroughly investigated.

Uncommon Ports: Some organizations use a special library for communication optimizations between services so that all HTTP communication between servers uses a different port than the common ones (such as 80, 443, 8080, etc). In this case, it's a good idea to create a rule that would be triggered by any communication on these normally common ports.

Honeytokens - In a battlefield like the Internet where everyone can be just about anyone, deception, works well for defenders just as well as it does for the attackers, if not better. Tricks like renaming the built in administrator account to a different, less attractive name and creating a new account named Administrator which you'll never use and create a Snort rule for detecting if this user name, email or password are ever used on the network. It would be next to impossible for attackers to notice that Snort has detected their attempts to use the fake administrator user. Another example is to create fake products, customers, users, and credit card records in the database and then matching Snort rules for detecting them in the network traffic.

Appendix E - Installing and Configuring a Wazuh Agent

If you have installed Coralogix Cloud Security with support for Wazuh, you can improve your security posture and visibility by installing the Wazuh agent on your instances.

Wazuh is a fork of OSSEC, the famous vulnerability assessment tool. It has an agent version for almost every type of operating system (Linux, Windows, macOS, Solaris, AIX, HP-UX). Coralogix Cloud Security includes a Wazuh Manager installation out of the box so you can simply install the agent and configure it to use Coralogix as its manager.

To avoid having to reinstall or reconfigure the Wazuh agent on your instances each time Coralogix is restarted (especially if it is installed as a spot fleet) we configured the installation to create a network load balancer that redirects the traffic to Coralogix's Wazuh server. So the first step of the installation would be to find out the DNS name of your Wazuh network load balancer. To do that follow these steps:

Login to your AWS console and switch to the CloudFormation console

Select the Coralogix Cloud Security stack you created and switch to the Resources tab

Find the "WazuhNLB" on the resources list and copy its ARN string

Switch to the EC2 management console and navigate to the load balancer section

Paste the ARN string you copied into the search box and hit Enter

The DNS name of your Wazuh manager appears in the “DNS name” field of the single object that is now shown.

Now that you have the DNS name of the Wazuh manager, all you have to do is to install the agent on your instances and configure them to connect to the Wazuh manager on Coralogix. We recommend that you would simply paste the installation code into the user data of the instances you would like to connect to Coralogix’s Wazhu manager. This way, if the instance is rebooted and gets a new IP address the connection between Coralogix agent and Coralogix manager will remain intact. This is the installation code for debian based Linux operating systems:

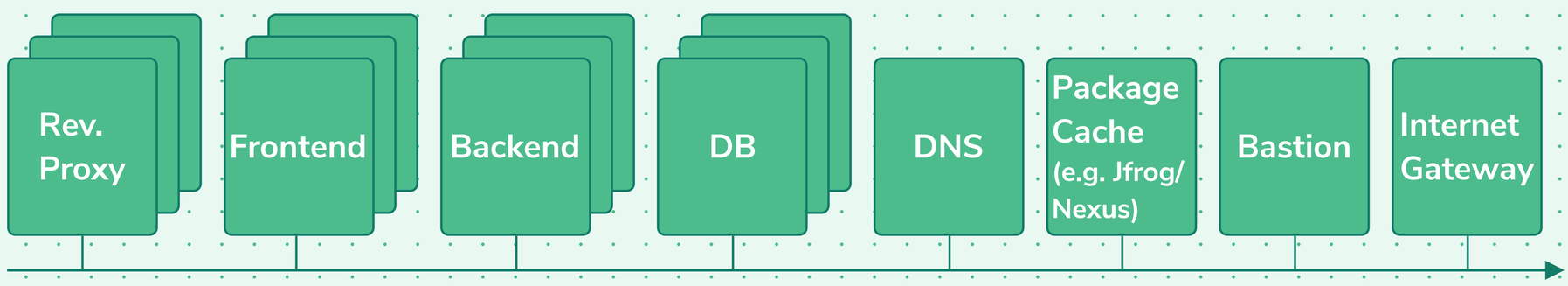
```
#!/bin/bash
logger -s “+Installing WAZUH agent...”
sudo apt-get update
sudo apt-get install curl apt-transport-https lsb-release gnupg2
curl -s https://packages.wazuh.com/key/GPG-KEY-WAZUH | sudo apt-key add -
echo “deb https://packages.wazuh.com/3.x/apt/ stable main” | sudo tee /etc/
apt/sources.list.d/wazuh.list
sudo apt-get update
export WAZUH_MANAGER_IP=”<<<<PUT_WAZUH_MANAGER_DNS_NAME_HERE_>>>>”
sudo -E apt-get install wazuh-agent=3.9.5-1
sudo cat /var/ossec/etc/ossec.conf
```

You can find instructions for installing the Wazuh agent for other types of Linux as well as for other operating systems here. Just remember to set the WAZUH_MANAGER_IP environment variable to your Coralogix’s Wazuh NLB.

Appendix F - Mirroring Strategies - Real Life Examples

Example 1: MyBookStore company:

This example represents a simple books store. The frontend servers hold the code for handling customers requests for searching and purchasing books, all the system's data is stored in the database servers. Currently, the throughput to the website is not extremely high but is expected to grow significantly in the upcoming months due to the new year's sales.



Security groups and public IP configuration:

Server type	Public IP	From	To	Service	Status
Rev. Proxy	EIP	any	any	HTTP/tcp	Allow
		any	any	HTTPS/tcp	Allow
		192.168.1.0/24	Frontend	HTTP/tcp	Allow
		192.168.1.0/24	any	NTP/udp	Allow
		Bastion	any	SSH/tcp	Allow
		192.168.1.0/24	DNS	DNS/udp	Allow
		192.168.1.0/24	Packages Cache	HTTP/tcp, HTTPS/tcp	Allow
Frontend	None	Rev. Proxy	192.168.1.0/24	HTTP/tcp	Allow
		192.168.1.0/24	Backend	Set of custom ports	Allow
		192.168.1.0/24	any	NTP/udp	Allow
		Bastion	192.168.1.0/24	SSH/tcp	Allow
		192.168.1.0/24	DNS	DNS/udp	Allow
		192.168.1.0/24	Packages Cache	HTTP/tcp, HTTPS/tcp	Allow
Backend	None	Frontend	192.168.1.0/24	Set of custom ports	Allow
		192.168.1.0/24	DB	DB Ports	Allow
		192.168.1.0/24	any	NTP/udp	Allow
		Bastion	192.168.1.0/24	SSH/tcp	Allow
		192.168.1.0/24	DNS	DNS/udp	Allow
		192.168.1.0/24	Packages Cache	HTTP/tcp, HTTPS/tcp	Allow
DB	None	Backend	192.168.1.0/24	DB ports/tcp	Allow
		192.168.1.0/24	any	NTP/udp	Allow
		Bastion	192.168.1.0/24	SSH/tcp	Allow
		192.168.1.0/24	DNS	DNS/udp	Allow
		192.168.1.0/24	Packages Cache	HTTP/tcp, HTTPS/tcp	Allow

DNS	None	192.168.1.0/24	DNS	DNS/udp	Allow
		192.168.1.0/24	any	NTP/udp	Allow
		192.168.1.0/24	any	DNS/udp	Allow
		Bastion	192.168.1.0/24	SSH/tcp	Allow
		192.168.1.0/24	Packages Cache	HTTP/tcp, HTTPS/tcp	Allow
Packages Cache	None	192.168.1.0/24	any	HTTP/tcp, HTTPS/tcp	Allow
		192.168.1.0/24	any	NTP/udp	Allow
		192.168.1.0/24	any	DNS/udp	Allow
		Bastion	192.168.1.0/24	SSH/tcp	Allow
Bastion	EIP	Well defined set of IP addresses	192.168.1.0/24	SSH/tcp	Allow
		192.168.1.0/24	any	NTP/udp	Allow
		192.168.1.0/24	DNS	DNS/udp	Allow
		192.168.1.0/24	Packages Cache	HTTP/tcp, HTTPS/tcp	Allow

The network diagram above describes a typical network of a SAAS service, a collection of reverse proxies that handle HTTPS encryption and clients connections that after basic validation such as URIs paths, and sometimes even parameters and HTTP methods and then send the HTTP requests as HTTP to the frontend servers which in turn make requests to their backend servers which process the data and update or query the DB servers and re-turn the requested result.

In this setup the only servers that are exposed to the Internet and are ex-pected to be reachable by other servers on the Internet are the proxy serv-ers and the bastion server. All other servers do not have a public IP address assigned to them.

All DNS queries are expected to be sent to the local DNS server which will resolve them against other DNS servers on the Internet.

Package installations and updates for all the servers on this network is expected to be handled against the local packages caching server. So that all other servers will not need to access the Internet at all except for time synchronization against public NTP servers.

SSH connections to the servers are expected to happen only via the bastion server and not directly from the Internet.

In this scenario, we would recommend using the following mirroring configuration by server type:
Reverse Proxies:
Since these servers are contacted by users from the Internet over a TLS encrypted connection, most of the traffic to it should be on port 443 (HTTPS) and its value to traffic rate ratio is expected to be quite low, we would recommend the following mirror filter configuration:

Outbound:

Rule number	Description	Rule action	Protocol	Source port range	Destination port range	Source CIDR block	Destination CIDR block
100	Monitor everything *	accept	All protocols	-	-	0.0.0.0/0	0.0.0.0/0

* – Although this rule will also mirror the server's encrypted responses to the HTTPS requests we do recommend that you mirror it since it might also contain HTTPS connections initiated by malwares running on the proxy servers to command and control servers and especially since the traffic volume of the server's responses is not expected to be extremely high.

Inbound:

Rule number	Description	Rule action	Protocol	Source port range	Destination port range	Source CIDR block	Destination CIDR block
100	Exclude incoming HTTPS	reject	TCP	-	443	0.0.0.0	<your servers subnet>
200	Monitor everything else	accept	All protocols	-	-	0.0.0.0	0.0.0.0/0

Frontend servers:

Since the frontend servers receive the unencrypted traffic they are essentially the the first place on the network in which we can detect attacks against the system and therefore their traffic data is very valuable from an information security perspective. This is why we believe that you would mirror any traffic from and to these servers. However, since this will of course include the valid traffic by the customers, albeit extremely valuable for creating a good baseline which can later be used to detect anomalies, this can also be rather expensive for the organization. For such cases we would recommend the following mirroring configuration for detecting abnormal behavior of the frontend servers that can indicate a Cyber attack that is currently in progress:

Outbound:

Rule number	Description	Rule action	Protocol	Source port range	Destination port range	Source CIDR block	Destination CIDR block
200	Monitor everything else	accept	All protocols	-	-	0.0.0.0	0.0.0.0/0

* - Although this rule will also mirror the server's responses to the HTTPS requests we do recommend that you mirror it since it might also contain HTTPS connections initiated by malwares running on the proxy servers to command and control servers. Also, this might include unintended responses that will indicate that an attack (such as XSS or SQL Injection) took place.

Inbound:

Rule number	Description	Rule action	Protocol	Source port range	Destination port range	Source CIDR block	Destination CIDR block
100	Exclude incoming HTTPS	reject	TCP	-	80	0.0.0.0	<your servers subnet>
200	Monitor everything else	accept	All protocols	-	-	0.0.0.0	0.0.0.0/0

Backend servers:

Since the frontend servers are contacting these servers with requests that are supposed to be valid after the checks done by the proxy and frontend servers, the traffic volume is expected to be significantly lower than traffic volume to the proxy and frontend servers. Also, since this is the last chance to detect malicious traffic before it reaches the database, we believe that all traffic from these servers should be mirrored. However, if this is still too expensive for your organization, we would recommend that you use the following mirroring configuration to at least be able to tell if a backend server is behaving abnormally which may indicate that it has already been breached:

Outbound:

Rule number	Description	Rule action	Protocol	Source port range	Destination port range	Source CIDR block	Destination CIDR block
100	Monitor everything *	accept	All protocols	-	-	0.0.0.0	0.0.0.0/0

* - Although this rule will also mirror the server's responses to frontend requests we do recommend that you mirror it since it might also contain HTTPS connections initiated by malwares running on the proxy servers to command and control servers. Also, this might include unintended responses that will indicate that an attack (such as XSS or SQL Injection) took place.

Inbound:

Rule number	Description	Rule action	Protocol	Source port range	Destination port range	Source CIDR block	Destination CIDR block
100	Exclude incoming HTTPS	reject	TCP	-	80	0.0.0.0	<your servers subnet>
200	Monitor everything else	accept	All protocols	-	-	0.0.0.0	0.0.0.0/0

Databases:

Since this server is at the heart of the customer's system, we recommend that you would mirror all traffic in and out of this server.

DNS server:

Since this server generates a very minimal amount of traffic that is so extremely valuable for detecting so many types of attacks we recommend that you would mirror all traffic in and out of this server.

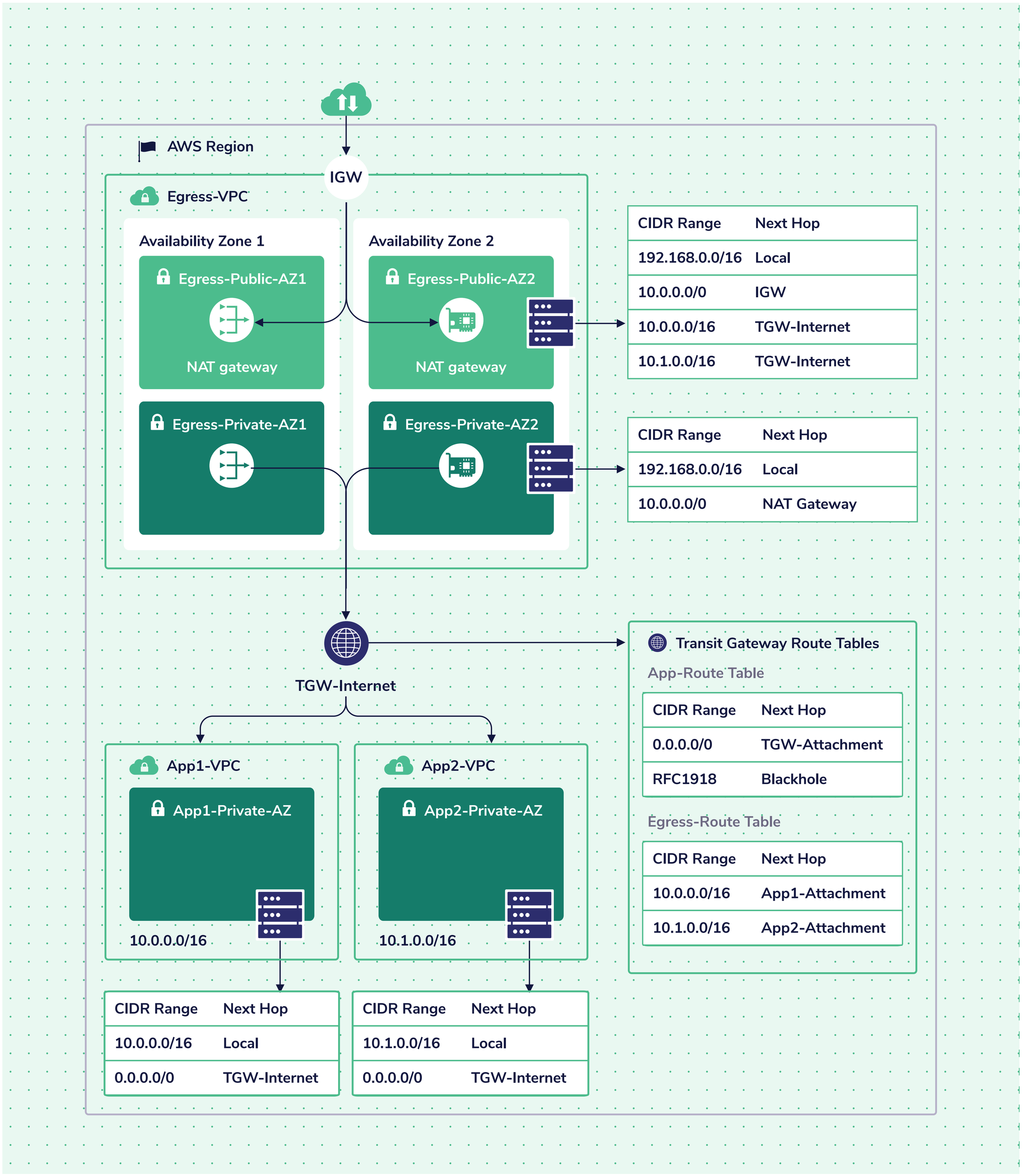
Package cache server:

Since this server is responsible for packages installations on all other servers, the data from and to this server can answer questions like who installed what when, which can be crucial in a forensic investigation and for detecting installation of malicious tools on the server. Also, the traffic volume from and to this server is expected to be quite low. Therefore, we recommend that you would mirror all traffic in and out of this server.

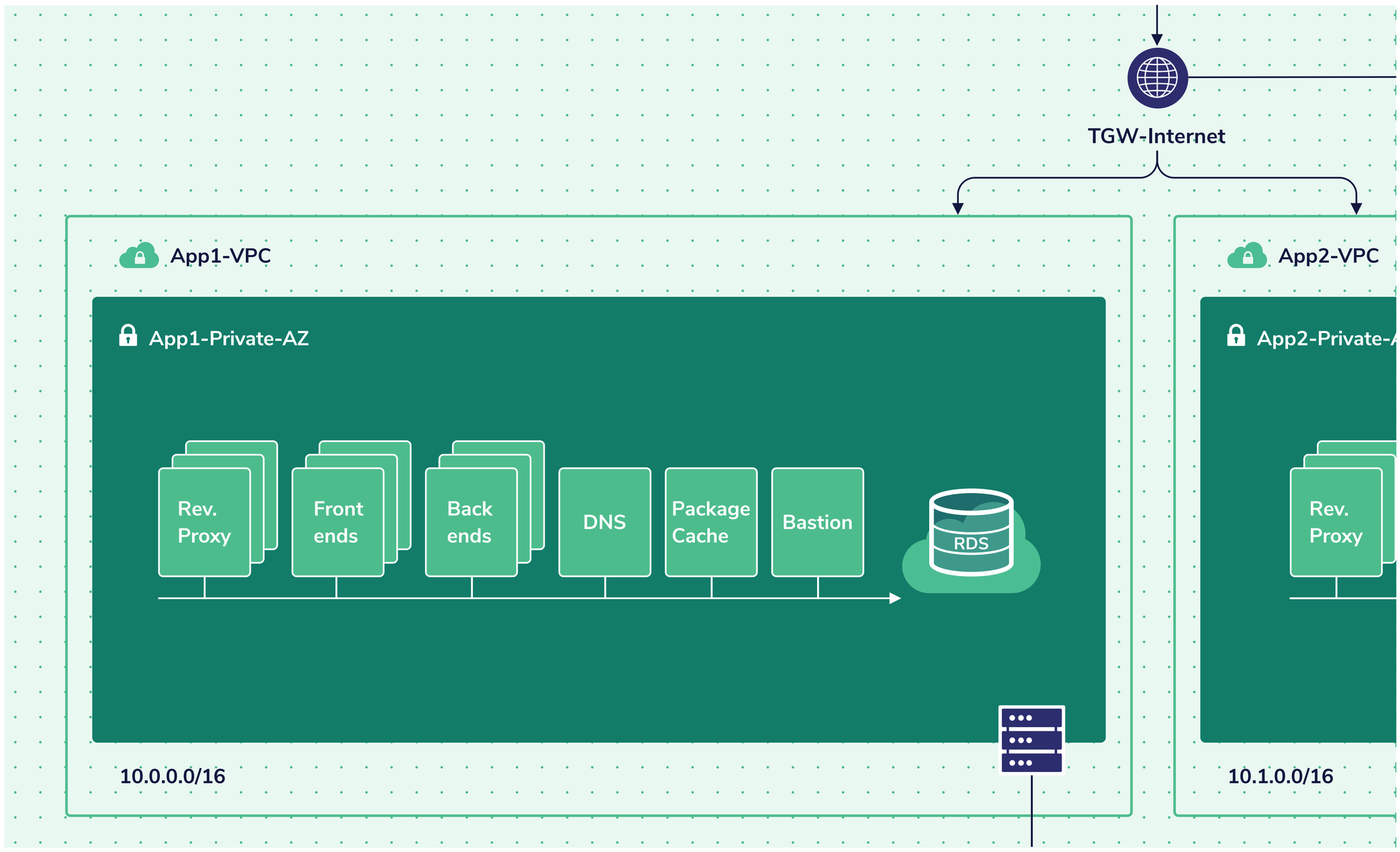
Bastion server:

Since this server is serving as the only way to manage all other servers, provided that it is being used to update the code on the various servers, install required packages, etc., the traffic volume should be relatively low and the value of the data it can provide is extremely high and therefore we recommend that you would mirror all traffic in and out of this server.

Example 2 - GoldenBank:



Zoom-in app 1



Zoom-in app 2

In this scenario, the client is running two different services, in two different private VPCs. The two VPCs cannot communicate with each other. The reverse proxies handle the majority of the traffic and perform basic request validations such as URIs and web methods and possibly also parameters type validation and then forward the request to the frontend servers. Unlike the previous scenario, the connection between the reverse proxy and the frontend servers is also encrypted in TLS.

The frontend servers perform business context validation of the requests and then process them by sending requests to the backend servers.

The backend servers in app1 rely on a storage server, while the backend servers of app2 rely on an AWS RDS service.

Here is the details of the security groups of all instances in both applications' VPCs:

VPC - App1

Server Type	Public IP	From	To	Service	Status
Rev. Proxy	None	any	any	HTTP/tcp	Allow
		any	any	HTTPS/tcp	Allow
		10.0.0.0/16	Frontend	HTTPS/tcp	Allow
		10.0.0.0/16	any	NTP/udp	Allow
		Bastion	any	SSH/tcp	Allow
		10.0.0.0/16	DNS	DNS/udp	Allow
		10.0.0.0/16	Packages Cache	HTTP/tcp, HTTPS/tcp	Allow
Frontend	None	Rev. Proxy	10.0.0.0/16	HTTPS/tcp	Allow
		10.0.0.0/16	Backend	Custom ports set	Allow
		10.0.0.0/16	any	NTP/udp	Allow
		Bastion	any	SSH/tcp	Allow
		10.0.0.0/16	DNS	DNS/udp	Allow
		10.0.0.0/16	Packages Cache	HTTP/tcp, HTTPS/tcp	Allow

Backend	None	Frontend	10.0.0.0/16	Custom ports set	Allow
		10.0.0.0/16	Storage	NFS/udp	Allow
		10.0.0.0/16	Storage	NFS/tcp	Allow
		10.0.0.0/16	any	NTP/udp	Allow
		Bastion	any	SSH/tcp	Allow
		10.0.0.0/16	DNS	DNS/udp	Allow
		10.0.0.0/16	Packages Cache	HTTP/tcp, HTTPS/tcp	Allow
Storage	None	Backend	10.0.0.0/16	NFS/udp	Allow
		Backend	10.0.0.0/16	NFS/tcp	Allow
		10.0.0.0/16	any	NTP/udp	Allow
		Bastion	any	SSH/tcp	Allow
		10.0.0.0/16	DNS	DNS/udp	Allow
		10.0.0.0/16	Packages Cache	HTTP/tcp, HTTPS/tcp	Allow
DNS	None	10.0.0.0/16	any	DNS/udp	Allow
		10.0.0.0/16	any	NTP/udp	Allow
		Bastion	any	SSH/tcp	Allow
		10.0.0.0/16	Packages Cache	HTTP/tcp, HTTPS/tcp	Allow
Package Cache	None	10.0.0.0/16	any	HTTP/tcp, HTTPS/tcp	Allow
		10.0.0.0/16	any	NTP/udp	Allow
		Bastion	any	SSH/tcp	Allow
		10.0.0.0/16	DNS	DNS/udp	Allow

VPC - App2

Server Type	Public IP	From	To	Service	Status
Rev. Proxy	None	any	any	HTTP/tcp	Allow
		any	any	HTTPS/tcp	Allow
		10.1.0.0/16	Frontend	HTTPS/tcp	Allow
		10.1.0.0/16	any	NTP/udp	Allow
		Bastion	any	SSH/tcp	Allow
		10.1.0.0/16	DNS	DNS/udp	Allow
		10.1.0.0/16	Packages Cache	HTTP/tcp, HTTPS/tcp	Allow

Frontend	None	Rev. Proxy	10.1.0.0/16	HTTPS/tcp	Allow
		10.1.0.0/16	Backend	Custom ports set	Allow
		10.1.0.0/16	any	NTP/udp	Allow
		Bastion	any	SSH/tcp	Allow
		10.1.0.0/16	DNS	DNS/udp	Allow
		10.1.0.0/16	Packages Cache	HTTP/tcp, HTTPS/tcp	Allow
Backend	None	Frontend	10.0.0.0/16	Custom ports set	Allow
		10.1.0.0/16	RDS	DB ports	Allow
		10.1.0.0/16	any	NTP/udp	Allow
		Bastion	any	SSH/tcp	Allow
		10.1.0.0/16	DNS	DNS/udp	Allow
		10.1.0.0/16	Packages Cache	HTTP/tcp, HTTPS/tcp	Allow
RDS	None	Backend	10.1.0.0/16	DB ports	Allow
		10.1.0.0/16	any	NTP/udp	Allow
		Bastion	any	SSH/tcp	Allow
		10.1.0.0/16	DNS	DNS/udp	Allow
		10.1.0.0/16	Packages Cache	HTTP/tcp, HTTPS/tcp	Allow
DNS	None	10.1.0.0/16	any	DNS/udp	Allow
		10.1.0.0/16	any	NTP/udp	Allow
		Bastion	any	SSH/tcp	Allow
		10.1.0.0/16	Packages Cache	HTTP/tcp, HTTPS/tcp	Allow
Package Cache	None	10.1.0.0/16	any	HTTP/tcp, HTTPS/tcp	Allow
		10.1.0.0/16	any	NTP/udp	Allow
		Bastion	any	SSH/tcp	Allow
		10.1.0.0/16	DNS	DNS/udp	Allow

This network architecture is quite complex and would require an approach that would combine multiple data sources to provide proper security for it. In addition, it involves traffic that is completely encrypted, storage servers that are contacted via encrypted connections and also AWS services that cannot be mirrored to the STA.

Here are our visibility recommendations for similar networks:

App1

Reverse Proxies and Front-end servers:

Outbound:

Rule number	Description	Rule action	Protocol	Source port range	Destination port range	Source CIDR block	Destination CIDR block
100	Monitor everything *	accept	All protocols	-	-	0.0.0.0	0.0.0.0/0

* - Although this rule will also mirror the server's responses to the HTTPS requests we do recommend that you mirror it since it might also contain HTTPS connections initiated by malwares running on the proxy servers to command and control servers. Also, this might include unintended responses that will indicate that an attack (such as XSS or SQL Injection) took place.

Inbound:

Rule number	Description	Rule action	Protocol	Source port range	Destination port range	Source CIDR block	Destination CIDR block
100	Drop HTTPS*	reject	TCP	-	443	0.0.0.0	10.0.0.0/16
200	Monitor everything else	accept	All protocols	-	-	0.0.0.0	0.0.0.0/0

* Since the incoming traffic on HTTPS is encrypted, the only thing it can indicate is its volume over time which can indicate certain types of DoS or DDoS and also the validity of the certificates used. So if the traffic volume is expected to be high, we would recommend excluding this traffic.

Backend servers:

Since the frontend servers are contacting these servers with requests that are supposed to be valid after the checks done by the proxy and frontend servers, the traffic volume is expected to be significantly lower than traffic volume to the proxy and frontend servers. Also, since this is the last chance to detect malicious traffic before it reaches the database, we believe that all traffic from these servers should be mirrored. However, if this is still too expensive for your organization, we would recommend that you use the following mirroring configuration to at least be able to tell if a backend server is behaving abnormally which may indicate that it has already been breached:

Outbound:

Rule number	Description	Rule action	Protocol	Source port range	Destination port range	Source CIDR block	Destination CIDR block
100	Exclude file service activity **	reject	TCP	-	NFS	10.0.0.0/16	10.0.0.0/16
200	Exclude file service activity **	reject	UDP	-	NFS	10.0.0.0/16	10.0.0.0/16
300	Monitor everything *	accept	All protocols	-	-	0.0.0.0	0.0.0.0/0

* - Although this rule will also mirror the server's responses to frontend requests we do recommend that you mirror it since it might also contain HTTPS connections initiated by malwares running on the proxy servers to command and control servers. Also, this might include unintended responses that will indicate that an attack (such as XSS or SQL Injection) took place.

** - Since the filesystem activity is usually heavy, we would recommend to avoid mirroring this type of traffic unless it is highly valuable in your application or network architecture.

Inbound:

Rule number	Description	Rule action	Protocol	Source port range	Destination port range	Source CIDR block	Destination CIDR block
100	Monitor everything else	accept	All protocols	-	-	0.0.0.0/0	0.0.0.0/0

Storage Servers::

Outbound:

Rule number	Description	Rule action	Protocol	Source port range	Destination port range	Source CIDR block	Destination CIDR block
100	Exclude file service activity *	reject	TCP	-	NFS	10.0.0.0/16	10.0.0.0/16
200	Exclude file service activity *	reject	UDP	-	NFS	10.0.0.0/16	10.0.0.0/16
300	Monitor everything	accept	All protocols	-	-	0.0.0.0	0.0.0.0/0

DNS server:

Since this server generates a very minimal amount of traffic that is so extremely valuable for detecting so many types of attacks we recommend that you would mirror all traffic in and out of this server.

Package cache server:

Since this server is responsible for packages installations on all other servers, the data from and to this server can answer questions like who installed what when, which can be crucial in a forensic investigation and for detecting installation of malicious tools on the server. Also, the traffic volume from and to this server is expected to be quite low. Therefore, we recommend that you would mirror all traffic in and out of this server.

Bastion server:

Since this server is serving as the only way to manage all other servers, provided that it is being used to update the code on the various servers, install required packages, etc., the traffic volume should be relatively low and the value of the data it can provide is extremely high and therefore we recommend that you would mirror all traffic in and out of this server.

App2

Reverse Proxies and Front-end servers:

Outbound:

Rule number	Description	Rule action	Protocol	Source port range	Destination port range	Source CIDR block	Destination CIDR block
100	Monitor everything *	accept	All protocols	-	-	0.0.0.0/0	0.0.0.0/0

* Even though most of the traffic is encrypted since it would include responses to clients requests, the outbound traffic is still extremely valuable. It can indicate the size of the data being sent to the client and assist in detecting large data leaks, it can detect connections originating from the reverse proxy itself and help detect command and control connections, it can also detect connections or connection attempts to other servers other than the frontend servers which can detect lateral movement.

Inbound:

Rule number	Description	Rule action	Protocol	Source port range	Destination port range	Source CIDR block	Destination CIDR block
100	Drop HTTPS*	reject	TCP	-	443	0.0.0.0/0	10.1.0.0/16
200	Monitor everything else	accept	All protocols	-	-	0.0.0.0/0	0.0.0.0/0

* Since the incoming traffic on HTTPS is encrypted, the only thing it can indicate is its volume over time which can indicate certain types of DoS or DDoS and also the validity of the certificates used. So if the traffic volume is expected to be high, we would recommend excluding this traffic.

Backend servers:

Outbound:

Rule number	Description	Rule action	Protocol	Source port range	Destination port range	Source CIDR block	Destination CIDR block
100	Monitor everything *	accept	All protocols	-	-	0.0.0.0/0	0.0.0.0/0

* - Since this server is expected to communicate with the RDS service, we would recommend mirroring all the traffic from this instance as this will allow the detection of database related attacks. In addition, the RDS security group should be configured to allow connections only from the backend server (this can be more strict by configuring the RDS itself to allow connections only if they were identified by a certificate that exists only on the backend server) and an alert should be defined in Coralogix based on CloudTrail logs that would fire if the security group is modified. Also, by forwarding the database logs from the RDS instance, an alert should be defined in Coralogix so it would be triggered if the database detects connections from other IP.

Inbound:

Since the traffic volume here should be quite low due to the validations done by the proxies and front-end servers, and due to the fact that this is the last point we can tap into the data before it is saved into the DB we recommend that you would mirror all incoming traffic from this server

Rule number	Description	Rule action	Protocol	Source port range	Destination port range	Source CIDR block	Destination CIDR block
100	Monitor everything *	accept	All protocols	-	-	0.0.0.0/0	0.0.0.0/0

* Since the incoming traffic on HTTPS is encrypted, the only thing it can indicate is its volume over time which can indicate certain types of DoS or DDoS and also the validity of the certificates used. So if the traffic volume is expected to be high, we would recommend excluding this traffic.

DNS server:

Since this server generates a very minimal amount of traffic that is so extremely valuable for detecting so many types of attacks we recommend that you would mirror all traffic in and out of this server.

Package cache server:

Since this server is responsible for packages installations on all other servers, the data from and to this server can answer questions like who installed what when, which can be crucial in a forensic investigation and for detecting installation of malicious tools on the server. Also, the traffic volume from and to this server is expected to be quite low. Therefore, we recommend that you would mirror all traffic in and out of this server.

Bastion server:

Since this server is serving as the only way to manage all other servers, provided that it is being used to update the code on the various servers, install required packages, etc., the traffic volume should be relatively low and the value of the data it can provide is extremely high and therefore we recommend that you would mirror all traffic in and out of this server.

Improve your cloud security posture today

Have questions? Contact support sales@coralogix.com

SCHEDULE A DEMO