INTRUSION DETECTION SYSTEM

Zeek Cluster

Author
Dana Abushwesh
Leen Falah

1 Abstract

This report talking about Zeek clusters' power. Dive deeper into specific aspects to discover how this symphony of capabilities, features, and benefits can safeguard your system.

2 Introduction

Zeek is a hybrid of both anomaly and rule based techniques. Zeek engine structure is indeed very efficient and useful as it creates incoming traffic into a series of events which can be either treated or left as is. An event could be anything like a a new connection , a new ssh connection or anything. The real power of Zeek comes with the Policy Script interpreter. This engine has its own scripting language (Zeek scripts eg anomaly.zeek) which comes with its own custom made data types and data structures tailored for the use of dealing with network data.

3 what is zeek?

Zeek, at its core, is a network security monitoring tool with a focus on protocol analysis and traffic inspection. While it can perform both **anomaly-based detection** and **signature-based detection**, it is important to understand the nature of these detection mechanisms within the context of Zeek.

Anomaly-Based Detection:
Zeek is well-suited for anomaly-based detection. Anomaly detection in Zeek involves establishing baselines of normal network behavior and identifying deviations from these patterns. Zeek achieves this through its scripting language, which allows users to define custom detection logic based on deviations from expected behavior. This approach is particularly effective for identifying unknown or novel threats that may not have known signatures.

Signature-Based Detection:
Zeek also supports signature-based detection, allowing the identification of known threats based on predefined patterns or signatures. Signature-based detection is effective for recognizing well-known malware, attack patterns, and vulnerabilities. Users can create custom signatures or leverage existing ones to identify specific indicators of compromise (IoCs) in network traffic.

Protocol Analysis: A significant strength of Zeek is its deep protocol analysis capabilities. Zeek dissects network protocols, scrutinizing packet payloads and behavior to uncover anomalous activities, protocol violations, and potential security risks. This protocol-centric approach allows Zeek to identify deviations from expected behavior within the context of specific network protocols.

The choice between anomaly-based and signature-based detection in a Zeek cluster often depends on the specific security requirements and the nature of the threats organizations are aiming to detect. Some deployments may emphasize one approach over the other, while others may use a combination of both for a more comprehensive detection strategy. The flexibility of Zeek's scripting language allows security teams to tailor the intrusion detection mechanisms to suit their unique needs.

4 Architecture of Zeek Cluster

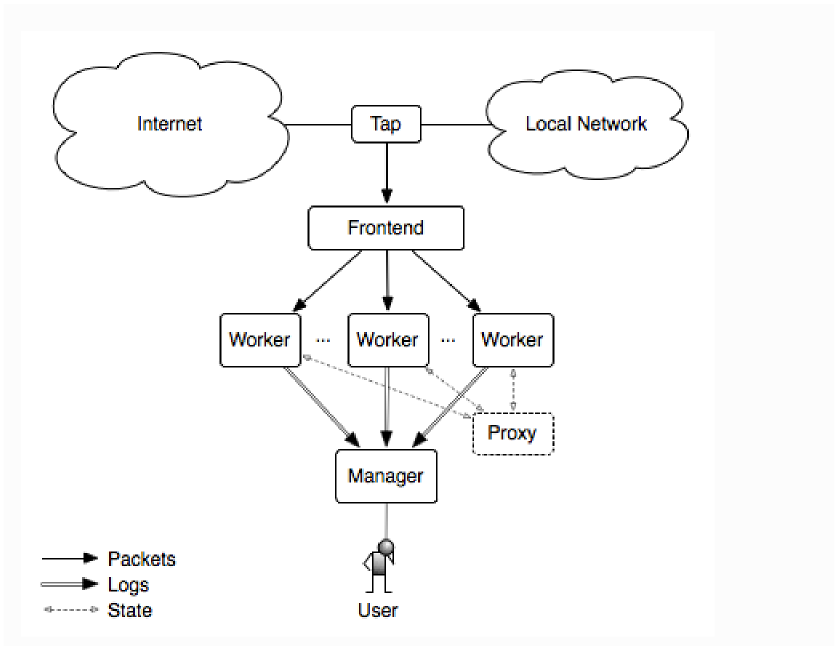


Figure 1: Zeek cluster Architecture

The **tap**is a mechanism that splits the packet stream to make a copy available for inspection. Examples include the monitoring port on a switch and an optical splitter on fiber networks.

4.1 Zeek's Cluster Components

By distributing network traffic across hosts and processes, overall traffic finally reaches a volume that can be effectively analyzed by a single worker process. Zeek then acts as a distributed network security monitor to perform analysis across many dozens or hundreds of workers, all acting on a small fraction of the overall traffic volume. The analysis of the worker process is further facilitated by nodes such as managers and proxies, ultimately logging the alerts and or relevant network logs. A Zeek cluster therefore consists of four main components: a manager, workers, proxies, and a logger.

Manager
The manager is a Zeek process that has two primary jobs. It normally receives log messages and notices from the rest of the nodes in the cluster using the Zeek communications protocol. It combines the individual logs that each worker produces so that the result is a set of joint logs instead of many discrete logs that you would have to combine in some manner with post-processing. (Note that if you use a separate logger node, then the logger receives all logs instead of the manager.) The manager also supports other functionality and analysis which requires a centralized, global view of events or data.

Worker

The worker is the Zeek process that sniffs network traffic and does protocol analysis on the reassembled traffic streams. Most of the work of an active cluster takes place on the workers. Workers typically represent the bulk of the Zeek processes that are running in a cluster. The fastest memory and CPU core speed you can afford is recommended since all of the protocol parsing and most analysis will take place here. There are no particular requirements for the disks in workers since almost all logging is done remotely to the manager (or dedicated logger). Normally, very little is written to disk.

Proxy

A proxy is a Zeek process that may be used to offload data storage or any arbitrary workload. A cluster may contain multiple proxy nodes. The default scripts that come with Zeek make only minimal use of proxies, so a single one will usually be sufficient. But custom scripts may make more use of it to partition data or workloads, providing greater cluster scalability potential than just doing similar tasks on a single, centralized manager node.
Zeek processes acting as proxies don't tend to be extremely hard on CPU or memory, and users frequently run proxy processes on the same physical host as the manager.

Logger

A logger is an optional Zeek process that receives log messages from the rest of the nodes in the cluster using the Zeek communications protocol. The purpose of having a logger to receive logs instead of the manager is to reduce the load on the manager. If no logger is needed, then the manager will receive logs instead.

4.2 Broker Communication Framework

Zeek uses the Broker Library to exchange information with other Zeek processes. Broker uses CAF (C++ Actor Framework) internally to connect nodes and exchange arbitrary data over networks. Broker then introduces, on top of CAF, a topic-based publish/subscribe communication pattern using a data model compatible with Zeek's. Broker itself can be utilized outside the context of Zeek, with Zeek itself making use of only a few predefined Broker message formats that represent Zeek events, log entries, etc.
The Broker library enables applications to communicate in Zeek's type-rich data model via publish/subscribe messaging.

4.3 Zeek deployment architecture

Zeek deployment architecture, including a policy script interpreter, events engine, network, and logs and notification components.

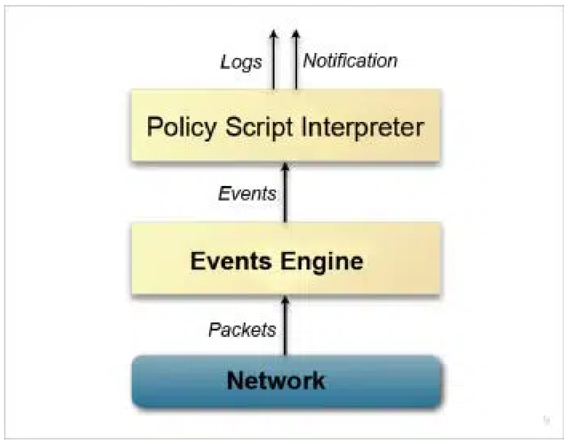


Figure 2: Zeek script interpreter

the main elements:

Policy Script Interpreter: This component reads and interprets the Zeek policy scripts, which define the rules and actions for analyzing network traffic. The policy script interpreter determines what data to collect from the network and how to analyze it.

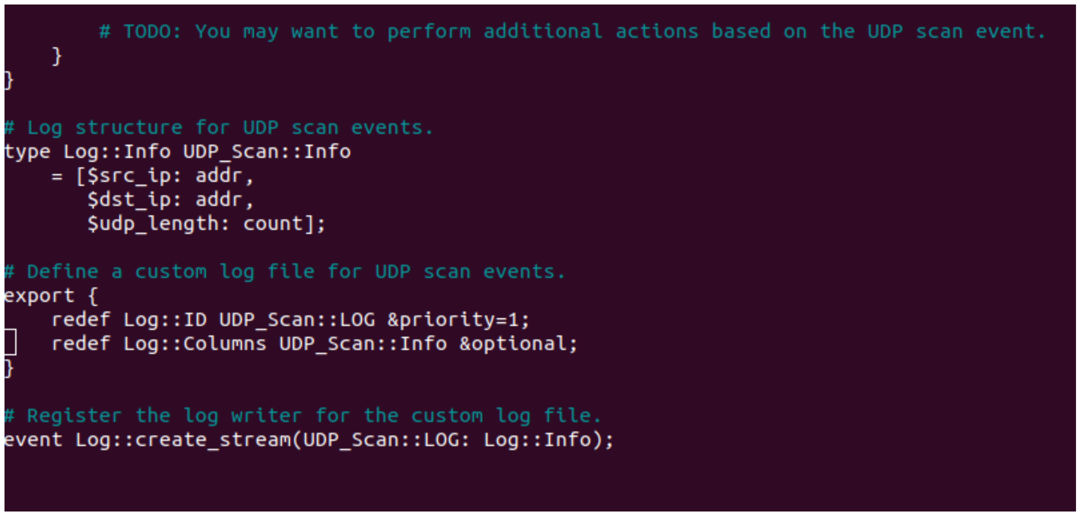
Events Engine: This component processes the data collected from the network according to the rules defined in the policy scripts. It generates events based on the detected activity and takes any necessary actions, such as logging or sending notifications.

Network: The diagram shows packets flowing through the network, representing the data that Zeek analyzes. Zeek can be deployed on various network segments to monitor specific traffic flows.

Logs and Notification: Zeek can generate logs of network activity and send notifications about suspicious events. This information can be used for security monitoring, incident response, and network forensics.

4.4 Zeek scripting Language

The Zeek scripting language is a domain-specific language (DSL) designed specifically for network traffic analysis and monitoring. Formerly known as the Bro scripting language, it was developed to provide a flexible and powerful way to define the behavior of the Zeek network analysis framework. Zeek scripts are used to express various aspects of network analysis, such as protocol parsing, event handling, and log generation.
As shown in Figures(3),(4).



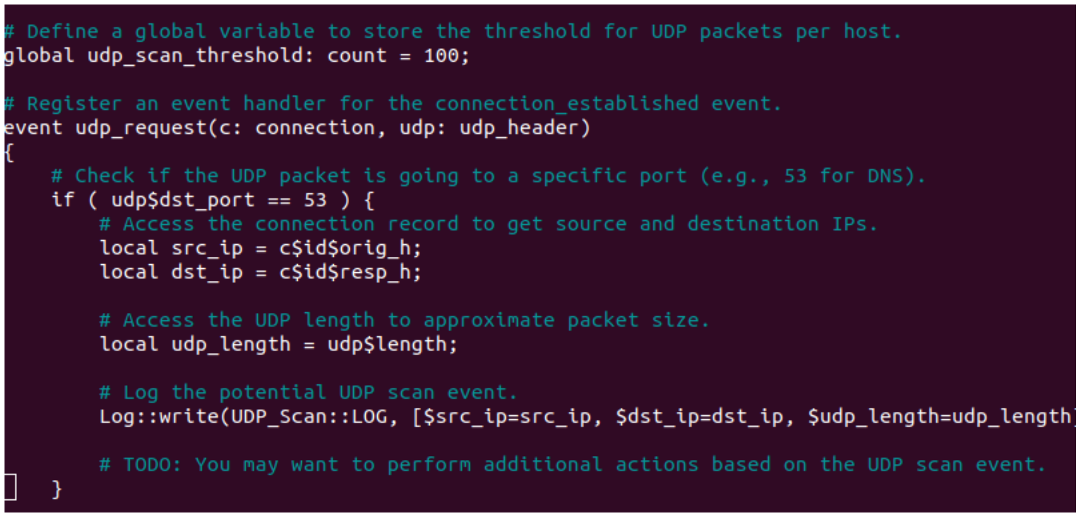
```
        # TODO: You may want to perform additional actions based on the UDP scan event.
    }
}

# Log structure for UDP scan events.
type Log::Info UDP_Scan::Info
    = [$src_ip: addr,
       $dst_ip: addr,
       $udp_length: count];

# Define a custom log file for UDP scan events.
export {
    redef Log::ID UDP_Scan::LOG &priority=1;
    redef Log::Columns UDP_Scan::Info &optional;
}

# Register the log writer for the custom log file.
event Log::create_stream(UDP_Scan::LOG: Log::Info);
```

Figure 3: Example of zeek scripts



```
# Define a global variable to store the threshold for UDP packets per host.
global udp_scan_threshold: count = 100;

# Register an event handler for the connection_established event.
event udp_request(c: connection, udp: udp_header)
{
    # Check if the UDP packet is going to a specific port (e.g., 53 for DNS).
    if ( udp$dst_port == 53 ) {
        # Access the connection record to get source and destination IPs.
        local src_ip = c$id$orig_h;
        local dst_ip = c$id$resp_h;

        # Access the UDP length to approximate packet size.
        local udp_length = udp$length;

        # Log the potential UDP scan event.
        Log::write(UDP_Scan::LOG, [$src_ip=src_ip, $dst_ip=dst_ip, $udp_length=udp_length

        # TODO: You may want to perform additional actions based on the UDP scan event.
    }
```

Figure 4: Example of zeek scripts

Features of the Zeek scripting language

Event-driven: Zeek operates in an event-driven fashion, responding to events triggered by the arrival of network traffic or other conditions. Zeek scripts define event handlers that specify how the system should react to specific events, such as the initiation of a connection, the detection of a protocol violation, or the expiration of a timer.

Protocol Analysis: Zeek scripts are commonly used to define how different network protocols should be parsed and analyzed. Protocol analyzers in Zeek scripts enable the extraction of meaningful information from network traffic, such as identifying hosts, services, and protocols.

Logging: Zeek scripts generate logs that capture information about network activity. These logs can include connection logs, HTTP logs, DNS logs, and more. The flexibility of the scripting language allows users to customize log formats and contents based on their specific monitoring needs.

Customization: Users can write custom scripts to extend and customize Zeek's functionality. This allows security professionals and network administrators to tailor Zeek to their specific requirements, adding new protocol analyzers, creating custom detection logic, and responding to unique events.

Modularity: Zeek scripts are organized into modules, and each module typically addresses a specific aspect of network analysis. This modular structure makes it easier to manage and organize complex analysis tasks.

4.4.1 Using script language for different purpose

In Zeek, signature-based detection and anomaly-based detection are implemented using custom scripts written in the Zeek scripting language. The Zeek scripting language provides a flexible and expressive environment for defining detection logic and capturing specific patterns or deviations in network traffic.

Signature-Based Detection:

For signature-based detection in Zeek, the primary method involves creating custom signatures or leveraging existing ones to identify known patterns associated with malicious activity. This is often implemented using regular expressions or other pattern-matching techniques. Here's a simplified example:

```
# Example Zeek script for signature-based detection

signature /malicious_pattern/ {
    ip-proto == TCP
    payload /malicious_string/
    event "MaliciousActivity"
}
```

In this example:

signature is a keyword used to define a signature. /malicious_pattern/ is a regular expression specifying a pattern to match in the payload. ip-proto == TCP ensures that the signature is applied to TCP traffic. payload /malicious_string/ specifies the payload content to match. event "MaliciousActivity" triggers an event when the signature is matched, allowing for a custom response or logging.

Security analysts can define multiple signatures to target different types of malicious activities or known threats.

Anomaly-Based Detection:

Anomaly-based detection in Zeek involves creating custom scripts to define baselines of normal behavior and identify deviations from these baselines. The scripting language allows analysts to specify conditions under which certain behaviors are considered anomalous. Here's a simplified example:

```
# Example Zeek script for anomaly-based detection

event connection_state_change(c: connection, old_state: connection_state, new_state:
connection_state) {
    if (new_state == S3) {
        if (c$duration > 3600) {
            print fmt("Anomalous long-duration connection detected: %s", c$id$orig_h);
            event "LongDurationConnection";
        }
    }
}
```

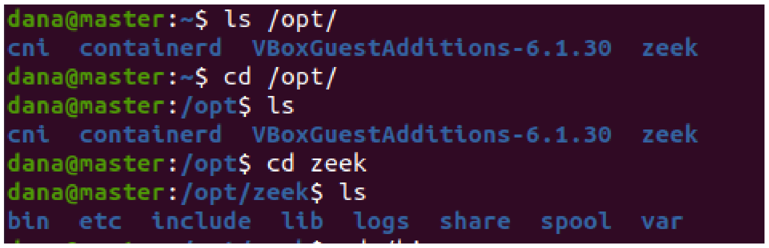
In this example:

event connection_state_change is triggered when there is a change in the connection state. The script checks if the new state is S3 (indicating an established connection). If the connection duration exceeds a specified threshold (3600 seconds), it is considered anomalous. A custom event is triggered, and a message is printed to indicate the detection of a long-duration connection.

Security analysts can create similar scripts to capture deviations in various network parameters, such as traffic volume, connection duration, or protocol-specific anomalies.

It's important to note that these are simplified examples, and actual Zeek scripts for detection can be more complex and tailored to specific use cases. Security professionals often customize and extend these scripts to match the unique characteristics of their network environments and the threats they are aiming to detect.

4.4.2 Directories under /opt/zeek



```
dana@master:~$ ls /opt/
cni  containerd  VBoxGuestAdditions-6.1.30  zeek
dana@master:~$ cd /opt/
dana@master:/opt$ ls
cni  containerd  VBoxGuestAdditions-6.1.30  zeek
dana@master:/opt$ cd zeek
dana@master:/opt/zeek$ ls
bin  etc  include  lib  logs  share  spool  var
```

Figure 5: Zeek directories

1. /opt/zeek/bin: This directory typically contains the binary executable files for Zeek. These binaries include the main Zeek executable, as well as various utility programs and scripts.

2./opt/zeek/etc: Configuration files for Zeek are stored in this directory. It includes the main configuration file **(”node.cfg”)** and other configuration files for specific components and modules. The **”networks.cfg”** file, which defines the local network topology, is often found here.

3. /opt/zeek/share: The ”share” directory contains shared data files, scripts, and other resources that are used by Zeek. This includes the default policy scripts, signatures, and other supporting files.

4./opt/zeek/spool: Zeek may use this directory to store temporary or spool files during its operation. For example, Zeek may write temporary files related to log data or other processing tasks here.

5./opt/zeek/logs: This directory is commonly used to store Zeek log files. The logs can include network traffic logs, notice logs, and other output generated by Zeek during its analysis.
The most important directory is **/opt/zeek/logs/current** directory is typically used to store the most recent log files generated by Zeek during its analysis of network traffic.
contains various log files The specific log files depend on the scripts and configuration you have enabled in your Zeek deployment. Common log files include conn.log (connection log), dns.log (DNS log), http.log (HTTP log), and more.

6./opt/zeek/lib: The ”lib” directory contains shared libraries and plugins used by Zeek. It may include additional scripting or plugin modules that extend Zeek’s functionality.

4.5 Logs

In a Zeek cluster, the distribution of logs across different nodes depends on the role of each node and its responsibilities within the cluster.

Worker Nodes:

Role: Worker nodes are responsible for analyzing portions of the network traffic and generating logs based on their analysis.

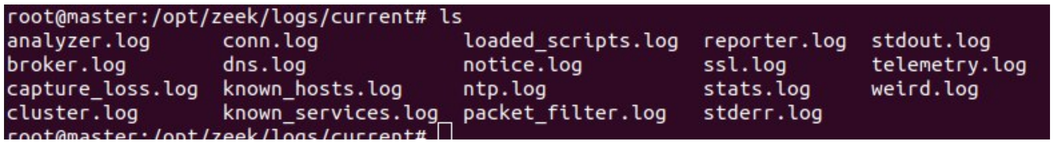
Logs: Worker nodes typically produce logs that contain information about specific aspects of network traffic. Examples of log files generated by worker nodes include conn.log (connection log), dns.log (DNS log), http.log (HTTP log), and more. Each worker node independently generates its own set of logs based on its analysis tasks.

Logger Nodes:

Role: Logger nodes in a Zeek cluster are dedicated to centralizing and storing logs generated by worker nodes.

Logs: Logger nodes do not perform packet analysis but are responsible for aggregating and storing logs from worker nodes.As shown on Figure(6). Logs on logger nodes include consolidated information from all worker nodes in the cluster. Logger Common log files on logger nodes include consensus.log (cluster consensus log) and transfer.log (log transfer log), among others.

in summery all the logs are created by different nodes are stored in **Logger** node.



```
root@master:/opt/zeek/logs/current# ls
analyzer.log      conn.log            loaded_scripts.log  reporter.log  stdout.log
broker.log        dns.log             notice.log          ssl.log       telemetry.log
capture_loss.log  known_hosts.log     ntp.log             stats.log     weird.log
cluster.log       known_services.log  packet_filter.log   stderr.log
root@master:/opt/zeek/logs/current#
```

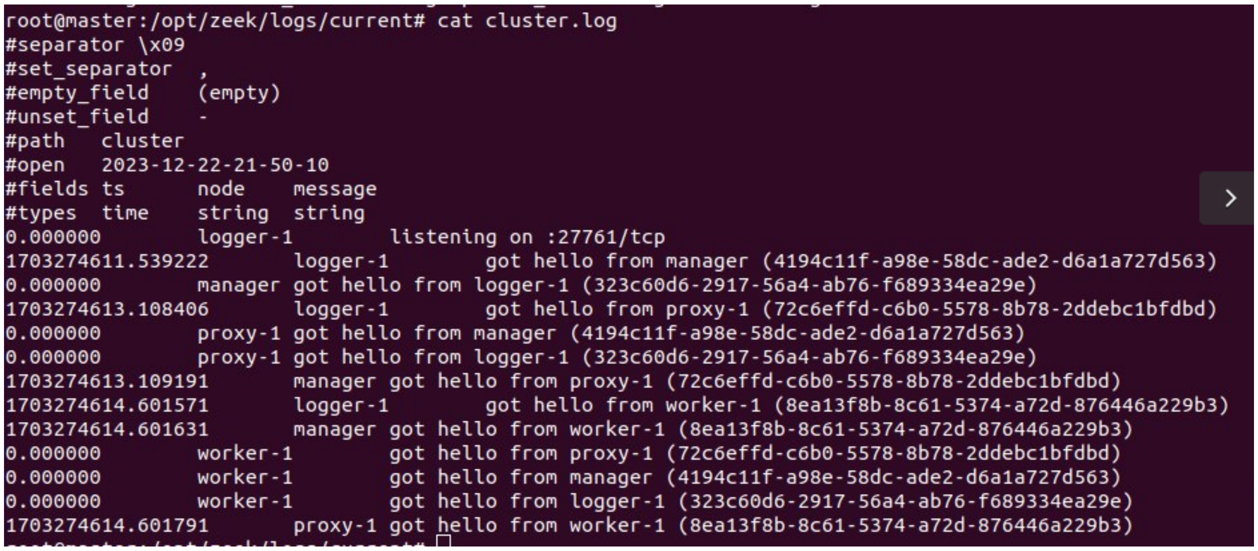
Figure 6: Logger node logs

Manager Node:

Role: The manager node coordinates the activities of the worker nodes, manages script deployment, and consolidates results from worker nodes. *Logs*: Logs on the manager node include information about the overall status of the Zeek cluster, communication between nodes, cluster-wide events, and administrative activities. The manager node does not generate detailed logs related to network traffic analysis but may log information about script loading and execution across the cluster.

4.5.1 The difference between manager and worker about logs

The **cluster.log file** in a Zeek cluster represents a log file that provides information about the overall coordination and communication within the Zeek cluster. This log is typically generated on the manager node and provides insights into the activities and status of the entire cluster, including interactions between the manager and worker nodes.
The difference between the Manager node and the Worker node is **cluster. log**.



```
root@master:/opt/zeek/logs/current# cat cluster.log
#separator \x09
#set_separator  ,
#empty_field    (empty)
#unset_field    -
#path   cluster
#open   2023-12-22-21-50-10
#fields ts      node    message
#types  time    string  string
0.000000        logger-1        listening on :27761/tcp
1703274611.539222       logger-1        got hello from manager (4194c11f-a98e-58dc-ade2-d6a1a727d563)
0.000000        manager got hello from logger-1 (323c60d6-2917-56a4-ab76-f689334ea29e)
1703274613.108406       logger-1        got hello from proxy-1 (72c6effd-c6b0-5578-8b78-2ddebc1bfdbd)
0.000000        proxy-1 got hello from manager (4194c11f-a98e-58dc-ade2-d6a1a727d563)
0.000000        proxy-1 got hello from logger-1 (323c60d6-2917-56a4-ab76-f689334ea29e)
1703274613.109191       manager got hello from proxy-1 (72c6effd-c6b0-5578-8b78-2ddebc1bfdbd)
1703274614.601571       logger-1        got hello from worker-1 (8ea13f8b-8c61-5374-a72d-876446a229b3)
1703274614.601631       manager got hello from worker-1 (8ea13f8b-8c61-5374-a72d-876446a229b3)
0.000000        worker-1        got hello from proxy-1 (72c6effd-c6b0-5578-8b78-2ddebc1bfdbd)
0.000000        worker-1        got hello from manager (4194c11f-a98e-58dc-ade2-d6a1a727d563)
0.000000        worker-1        got hello from logger-1 (323c60d6-2917-56a4-ab76-f689334ea29e)
1703274614.601791       proxy-1 got hello from worker-1 (8ea13f8b-8c61-5374-a72d-876446a229b3)
```

Figure 7: Cluster.log file

The manager node logs events related to the coordination of the entire Zeek cluster. This includes activities such as distributing work to worker nodes, managing script deployment, and achieving consensus on configuration changes.

Worker nodes perform protocol analysis on portions of the network traffic assigned to them. As a result, their logs contain detailed information about the analyzed traffic, including connection details, protocol-specific events, and anomalies.

Proxy Nodes: *Role:* Proxy nodes(optional), if used, are responsible for distributing traffic among worker nodes in a **load-balanced** manner.
Logs: Proxy nodes may produce logs related to load balancing and traffic distribution. These logs can include information about connections being redirected to specific worker nodes. The specifics of proxy node logs depend on the configuration and the type of proxy used.

5 Layout / Topology

In a Zeek cluster setup, every Zeek process is assigned a cluster role. Such a process is then called a Zeek node, a cluster node, or just named after the role of the process (the manager, the loggers, . . .). A basic Zeek cluster uses four different node types, enumerated in the script-level variable.
.Manager
. Logger
. Worker
. Proxy
In small Zeek deployments, all nodes may run on a single host. In large Zeek deployments, nodes may be distributed across multiple physical systems for scaling.
Currently, a single Manager node in a Zeek cluster exists. Further, connectivity between nodes is determined statically based on their type:

Every node connects to all loggers and the manager.
Each worker connects to all proxies.

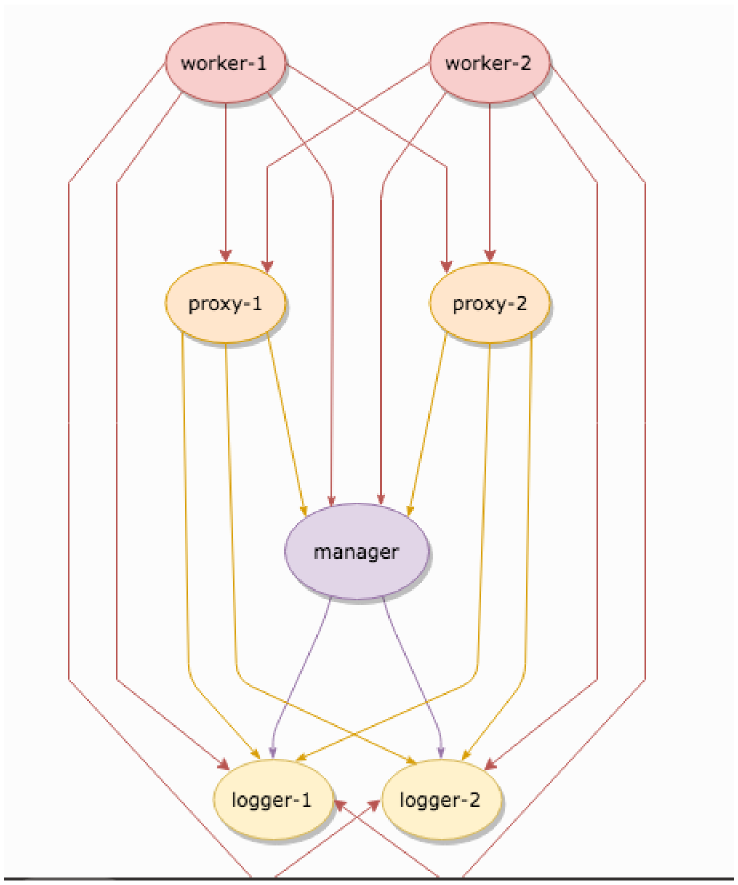


Figure 8: Zeek topology

Utilization of each node type:

Workers: are a good first choice for doing the brunt of any work you need to be done. They should be spending a lot of time performing the actual job of parsing/analyzing incoming data from packets, so you might choose to look at them as doing a "first pass" analysis and then deciding how the results should be shared with other nodes in the cluster.

Proxies: serve as intermediaries for data storage and work/calculation offloading. Good for helping offload work or data in a scalable and distributed way. Since any given worker is connected to all proxies and can agree on an "arbitrary key -¿ proxy node" mapping (more on that later), you can partition work or data amongst them in a uniform manner. e.g. you might choose to use proxies as a method of sharing non-persistent state or as a "second pass" analysis for any work that you don't want interfering with the workers' capacity to keep up with capturing and parsing packets. Note that the default scripts that come with Zeek make minimal use of proxies, so if you are coming from a previous ZeekControl deployment, you may want to try reducing it down to a single proxy node. If you come to have custom/community scripts that utilize proxies, that would be the time to start considering scaling up the number of proxies to meet demands.

Manager: This node will be good at performing decisions that require a global view of things since it is in a centralized location, connected to everything. However, that also makes it easy to overload, so try to use it sparingly and only for tasks that must be done in a centralized or authoritative location. Optionally, for some deployments, the Manager can also serve as the sole Logger.

Loggers: These nodes should simply be spending their time writing out logs to disk and not used for much else. In the default cluster configuration, logs get distributed among available loggers in a round-robin fashion, providing failover capability should any given logger temporarily go offline.

6 capabilities of a Zeek cluster

Scalability: Zeek clusters can scale horizontally by adding multiple worker nodes to distribute the analysis workload. This allows the cluster to handle increased network traffic and achieve higher throughput.

Parallel Processing: Worker nodes in a Zeek cluster operate independently, processing different portions of the network traffic in parallel. This parallel processing capability enhances the overall efficiency of network analysis.

Flexible Deployment: Zeek clusters can be deployed in various configurations, including standalone deployments, multi-node clusters, and clusters with dedicated logger nodes. This flexibility allows users to tailor the deployment to their specific needs and resource availability.

Decentralized Analysis: Each worker node in a Zeek cluster performs an independent analysis of its assigned portion of network traffic. This decentralized approach allows the cluster to efficiently analyze diverse and distributed network environments.

Customizable Scripting: Zeek's scripting language allows users to customize and extend the functionality of the network analysis. Users can write custom scripts to define specific behaviors, analyze custom protocols, and detect unique security events.

Centralized Log Storage: Logger nodes in a Zeek cluster centralize and store logs generated by worker nodes. This centralized log storage simplifies log analysis and provides a unified view of network activity across the entire cluster.

Load Balancing *(Optional)*: In some Zeek cluster configurations, proxy nodes may be used to distribute traffic among worker nodes, providing load-balancing capabilities. This ensures that the analysis workload is evenly distributed across the cluster.

Real-Time Analysis: Zeek clusters can provide real-time analysis of network traffic, allowing for the prompt detection of security incidents, protocol anomalies, and other events.
Network Protocol Support: Zeek supports a wide range of network protocols out of the box, and users can extend its protocol analysis capabilities by writing custom scripts.

Integration with External Systems: Zeek clusters can be integrated with external systems and tools, allowing for the export of logs and results to other security information and event management (SIEM) systems, databases, or custom analysis tools.

Security Monitoring: Zeek clusters are widely used for security monitoring, allowing organizations to detect and respond to network security threats effectively.

7 Conclusion

A Zeek cluster is a distributed network security monitoring architecture that leverages multiple nodes working collaboratively to analyze network traffic comprehensively. Consisting of manager, worker, and optional proxy nodes, a Zeek cluster enables efficient handling of large volumes of network data. The manager node orchestrates the cluster, distributing specific analysis tasks to worker nodes, each tasked with monitoring distinct aspects of network activity. Worker nodes generate protocol-specific logs through in-depth traffic analysis. While these logs are produced on the worker nodes, they are often centralized on a logger node for streamlined storage and management. Zeek clusters enhance scalability, enabling organizations to monitor and analyze network traffic effectively, identify security threats, and gain valuable insights into network behavior for improved cybersecurity. The versatility of Zeek clusters makes them valuable tools for network administrators, security analysts, and organizations seeking robust network security solutions. so the cluster as intrusion detection capabilities encompass both anomaly-based and signature-based detection mechanisms. It provides a flexible environment where security professionals can leverage custom scripts to define their own detection logic, whether it involves identifying anomalies in network behavior or detecting specific signatures associated with known threats.

8 References

https://docs.zeek.org/en/master/cluster-setup.html
https://docs.zeek.org/en/master/frameworks/cluster.html
https://www.opensourceforu.com/2019/05/a-sneak-peek-at-zeek-the-flexible-network-security-monitor/
https://docs.zeek.org/en/master/frameworks/broker.htmlbroker-framework