Intermediate to Zeek

NSF Cybersecurity Summit Training

Oct 23th, 2023

Who Are We?

Fatema Bannat Wala

- Security Engineer @ ESnet/LBNL
- 8 yrs of Network defense
- Zeek LT Member / Zeek Training Subgroup Lead

Christian Kreibich

- Zeek Technical Lead and LT member
- Principal Engineer at Corelight
- 20+ years of network security & research

Agenda

- What is a Zeek Cluster? - Fatema (15 mins)
- Become familiar with Cluster Configs - Fatema (45 mins)
- Zeek's Filesystem Layout - Fatema (30 mins)

  <break 30 > 10:30am - 11am
- Running Zeek in a Cluster - Fatema (30mins)
- Customizing Zeek - Fatema (30mins)
- Zeek Package Manager (zkg) - Fatema (30mins)

  <Lunch 1hr> 1pm - 2pm
- Logging Framework - Christian (30 mins)
- Notice Framework - Christian (30 mins)
- Telemetry Framework - Christian (30 mins)

  <break 30> 3:30pm - 4pm
- New Zeek Management Framework - Christian (45 mins)
- Adjacent Technologies - Christian (15 mins)

Housekeeping

- Zeek's Official Documentation:
  - https://docs.zeek.org/en/master/
- Join Zeek Slack channel #training

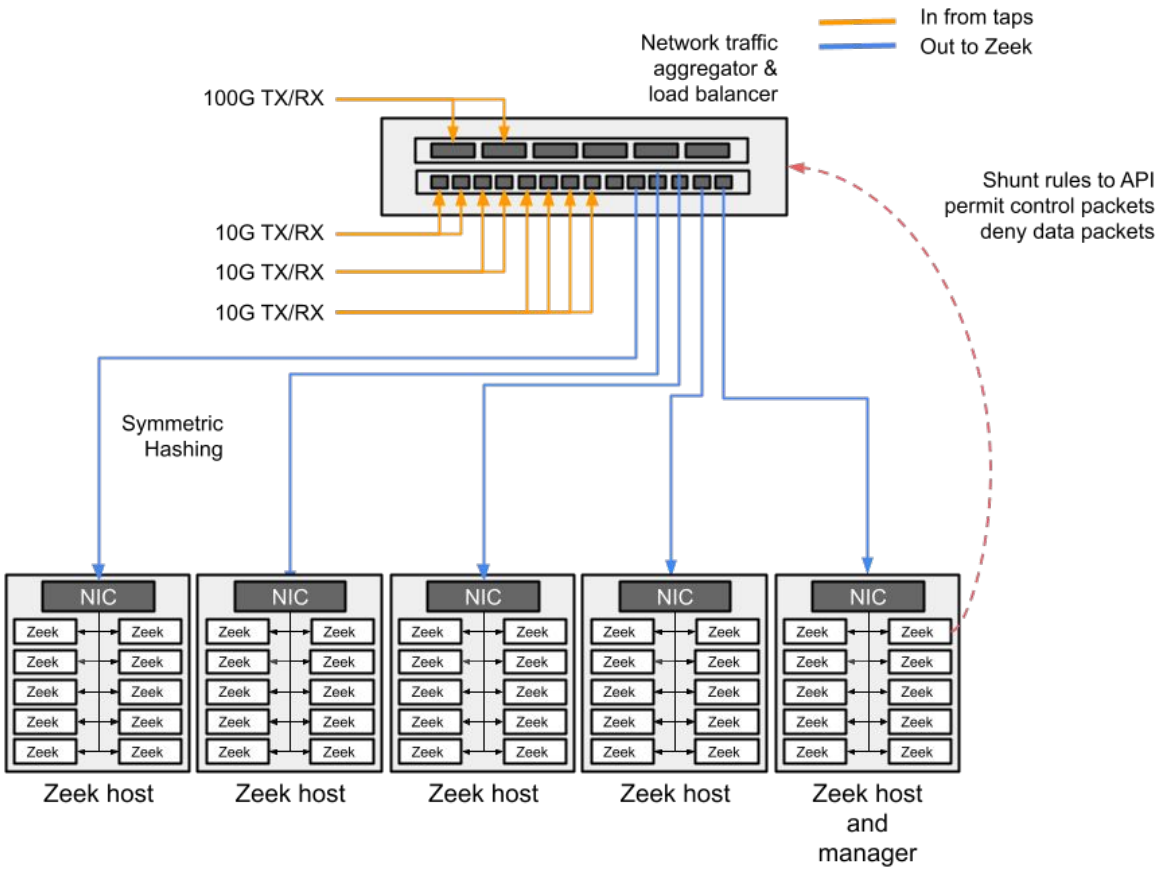
Docker / Zeek Usage

- Docker How to doc
  - https://github.com/zeek/zeek-training/tree/master/Intro-to-Zeek22
- Docker image
  - Zeek v6.0.1 installed on the image- https://hub.docker.com/repository/docker/zeek/training
    - # docker pull zeek/training:intro23
    - # docker run -it --privileged zeek/training:intro23
  - Path inside the container to training resources used during this training: /zeek/training-res/
- Verify that Zeek runs correctly:
  # zeek –h

What is a Zeek Cluster?

What / Why

- Zeek processes started with scripts that tune their purpose
  - As opposed to running a single Zeek process, reading from a pcap, for instance
  - Can run on one or more pieces of hardware
- Scales quite nicely
- Supporting tools handle tasks like monitoring processes and log rotation
- If you need 24x7 network monitoring, clustering is your answer



Source : Zeek Project. Creative Commons

Sniffing Packets

- SPAN/MIRROR ports
  - Configured on the router/switch
  - Pros :
    - Flexible (could mirror only certain network segments)
    - Might keep your network people happy (they can control the setup)
  - Cons :
    - Consumes router resources (CPU/physical ports)
    - Vulnerable/Visible to attackers (modifying/viewing configs)
    - Mistakes in managing network gear can impact flows

Sniffing Packets

- Taps
  - No moving parts, no config, invisible to attackers
  - Each link consumes two ports
  - Passive
    - Split light at fixed ratios (50/50 or 70/30 are common)
    - Light budget requires coordination with networking team based on distances
  - Active
    - Electrically powered device
    - Useful for converting signal types (such as 10 Gb SR to 10 Gb LR)
    - Useful for copper infrastructure

Tap Aggregation

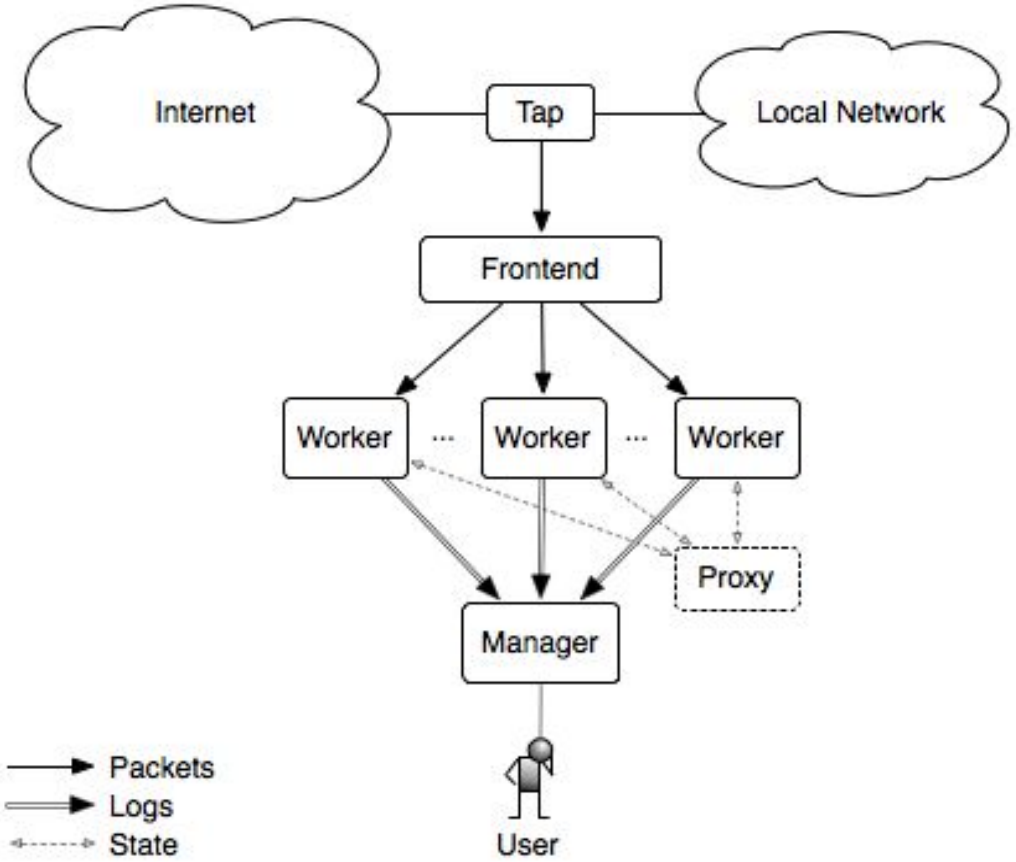
- Merges multiple tap sources
- Can output to groups of tools
- Arista / Gigamon / Ixia, for example
- Advanced features
  - Packet deduplication
  - Packet Slicing
  - Timestamping
  - Tagging

Worker Load Balancing

- Commercial Solutions
  - Examples : Endace / Napatech / Myricom
  - Expensive solution for very high performance
  - I’m most familiar with Myricom
    - Provides in-memory ring buffers with symmetric hashing
    - Very easy to support
    - Proprietary driver, updates can lag behind kernel versions

Worker Load Balancing

- Open Source
  - AF_PACKET
    - Built into Linux kernel
    - Can be challenging to setup and tune
    - Interrupt-based
  - PF_Ring
    - External modules, recompile kernel for upgrades
    - Compile zeek with the correct pf_ring libs
  - DPDK
    - Zeek package handles configuration
    - Also check out Vlad's talk at ZeekWeek
    - Comes with ~ 50 NIC drivers
    - Polling-based
    - Does not currently support bonded interfaces



Source : Zeek Project. Creative Commons

Cluster Components

- Manager
- Worker
- Proxy
- Logger

Cluster Components

- Manager
  - Writing logs from workers (when not using separate logger nodes)
  - Useful for smaller volumes or limited hardware
  - Can also run scripts when you want to conduct analysis across multiple workers.

Cluster Components

- Manager
- Worker
  - Analyze network packets to create (most) logs
  - Run scripts to alert on interesting traffic

Cluster Components

- Manager
- Worker
- Proxy
  - Data storage (can be distributed with multiple proxies)
  - State sharing among cluster workers
  - Built on Broker (topic-based pub/sub communication using C++ Actor Framework)
  - Start with one, then add more if load is an issue

Cluster Components

- Manager
- Worker
- Proxy
- Logger
  - Write logs
  - Easy support to load balance among multiple logger nodes
  - Ideal way to offload log I/O in high volume environments
  - Defaults to TSV format, but easy to switch to JSON

Cluster Architecture

- Single Physical Box
  - Inexpensive, easy management
  - Useful for a single feed

Cluster Architecture

- Hardware Cluster
  - Separate boxes for workers
    - 1U Pizza boxes
    - Fast CPU, Lots of RAM, small disk (low I/O requirements)
    - Allows horizontal scaling
    - Not good candidates for VM environments
  - Manager/Proxy/Logger on a single host
    - Moderate CPU/RAM
    - Fast Disk, ideal for logger node(s)
    - Good components to consider running in a VM environment

Cluster Architecture

- Networking
  - 1Gb interfaces more than adequate for inter-process comms
  - Broker used to share state, make queries, send logs to manager/logger
  - SSH used by Zeekctl (more on that later!)

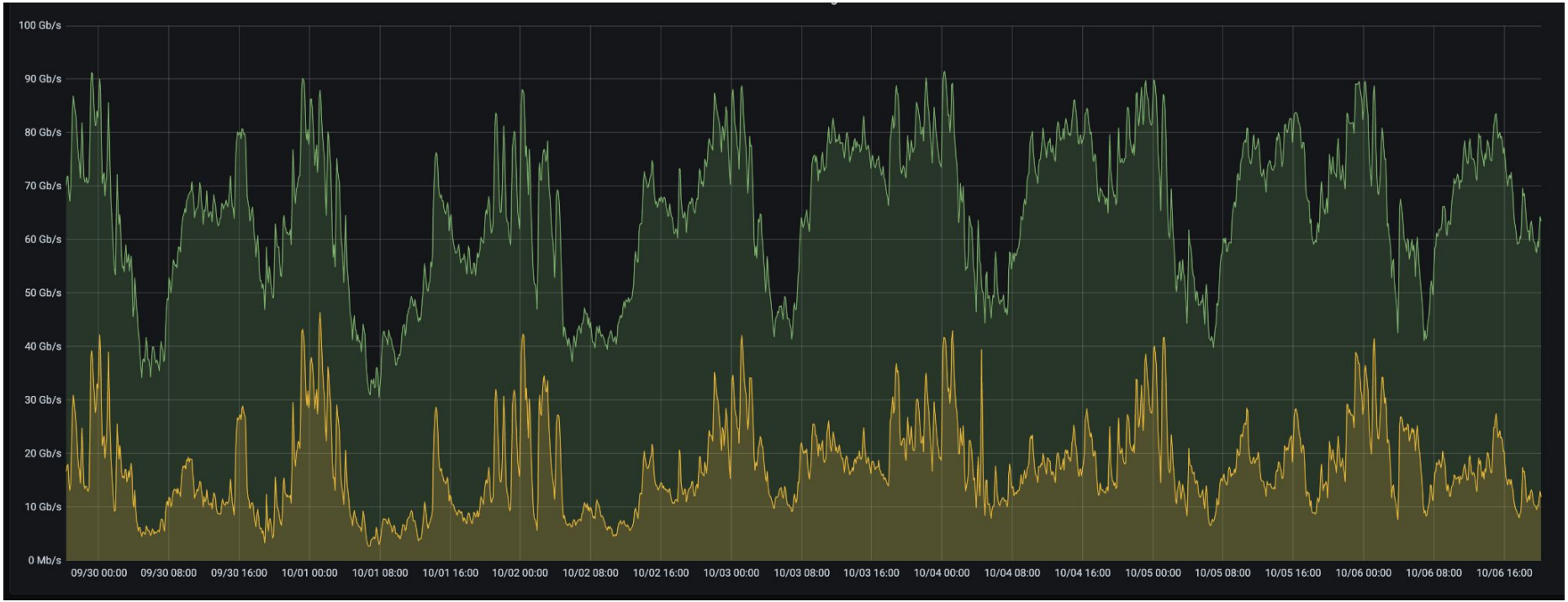
Performance

- How big?
  - That depends. A lot.
  - ~ 6 GB per worker process
  - Faster CPUs are always better
- LBL 100G Paper
  - Good architecture overviews
- SEPTun-Mark-II
  - Suricata focused, but deep dive on NUMA, OS tuning, etc.
- CPU Pinning
  - Avoid using hyperthreaded cores for workers
- Fine tuning ethtool - tro/tso/rx/tx/ etc..
- Metrics/Monitoring
  - Prometheus - Zeek Exporter from ESNET
  - Telemetry FW
  - Collectd

Shunting

- Reduce load by blocking elephant flows
- Streams/large file transfers over TLS, for example
- Ideally, you block the data packets, but let TCP control packets through
  - Zeek can tell you conn size, packet count, duration
- Dumbno (for Arista)

Shunting

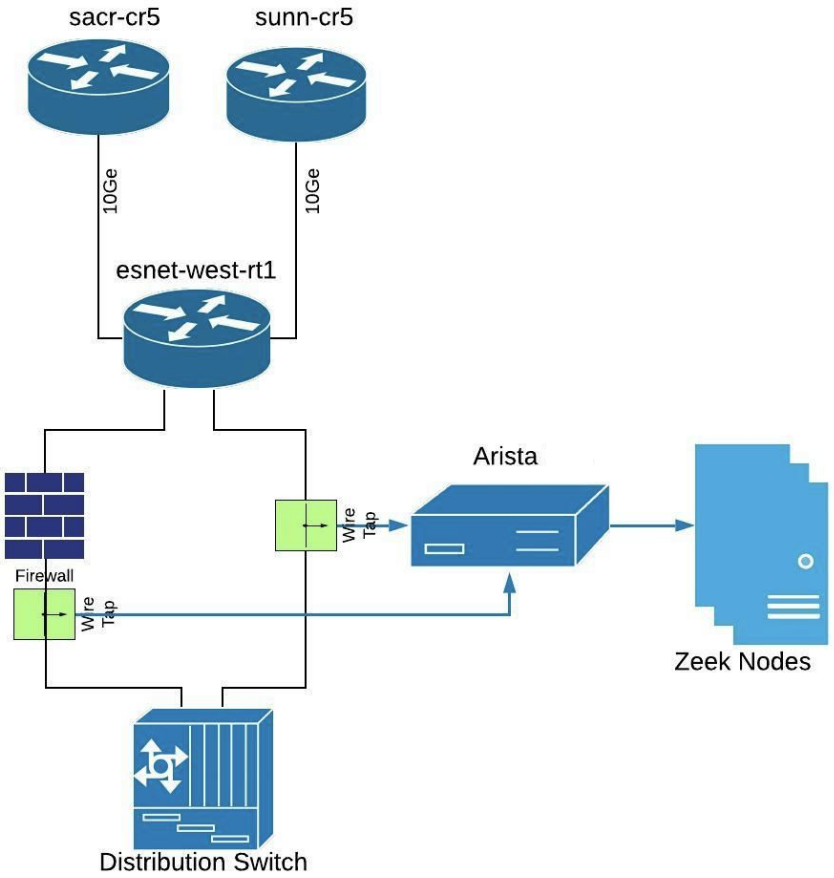

100 Gb/s
90 Gb/s
80 Gb/s
70 Gb/s
60 Gb/s
50 Gb/s
40 Gb/s
30 Gb/s
20 Gb/s
10 Gb/s
0 Mb/s
09/30 00:00
09/30 08:00
09/30 16:00
10/01 00:00
10/01 08:00
10/01 16:00
10/02 00:00
10/02 08:00
10/02 16:00
10/03 00:00
10/03 08:00
10/03 16:00
10/04 00:00
10/04 08:00
10/04 16:00
10/05 00:00
10/05 08:00
10/05 16:00
10/06 00:00
10/06 08:00
10/06 16:00

Real World Deployments

- Indiana University
- University of Delaware
- ESnet

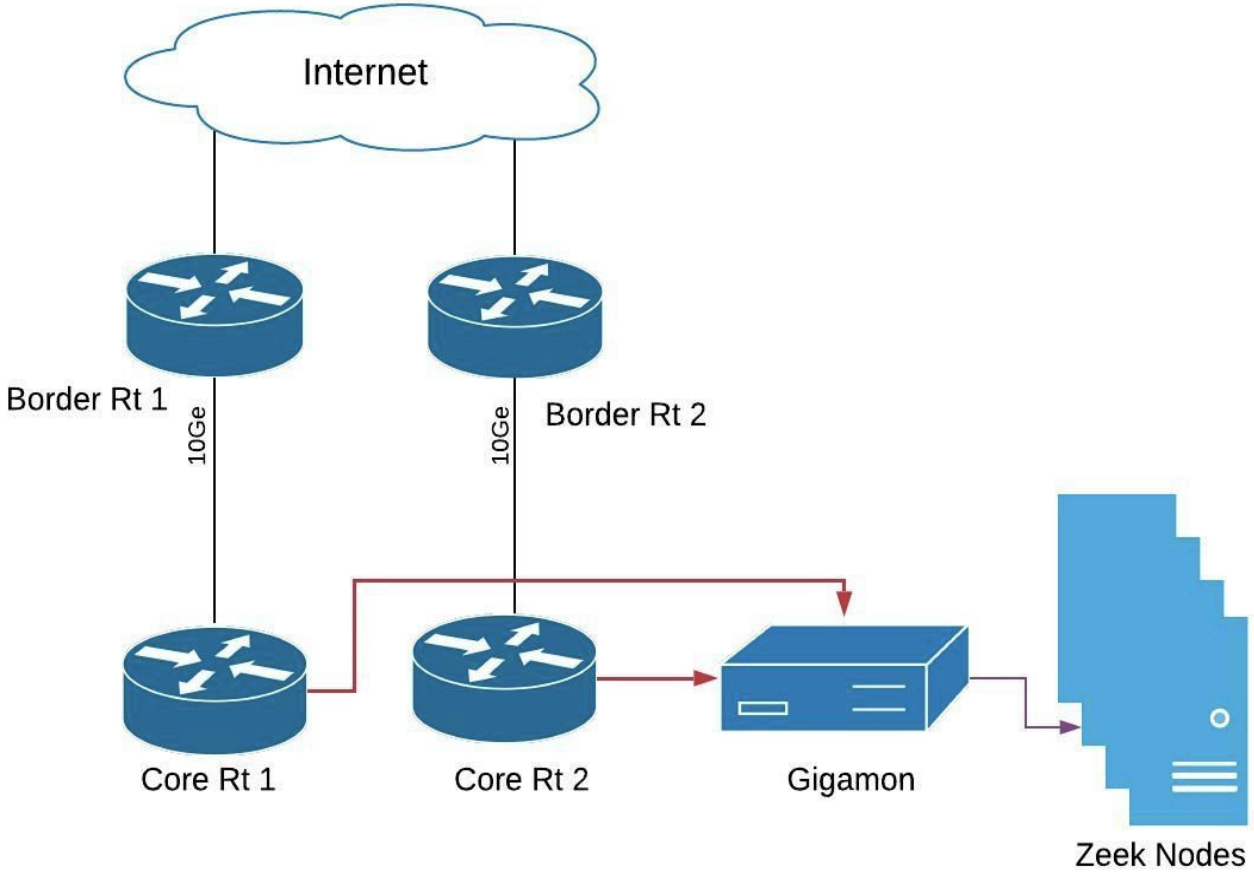
N/w Architecture for Zeek deployment # 1

At ESnet

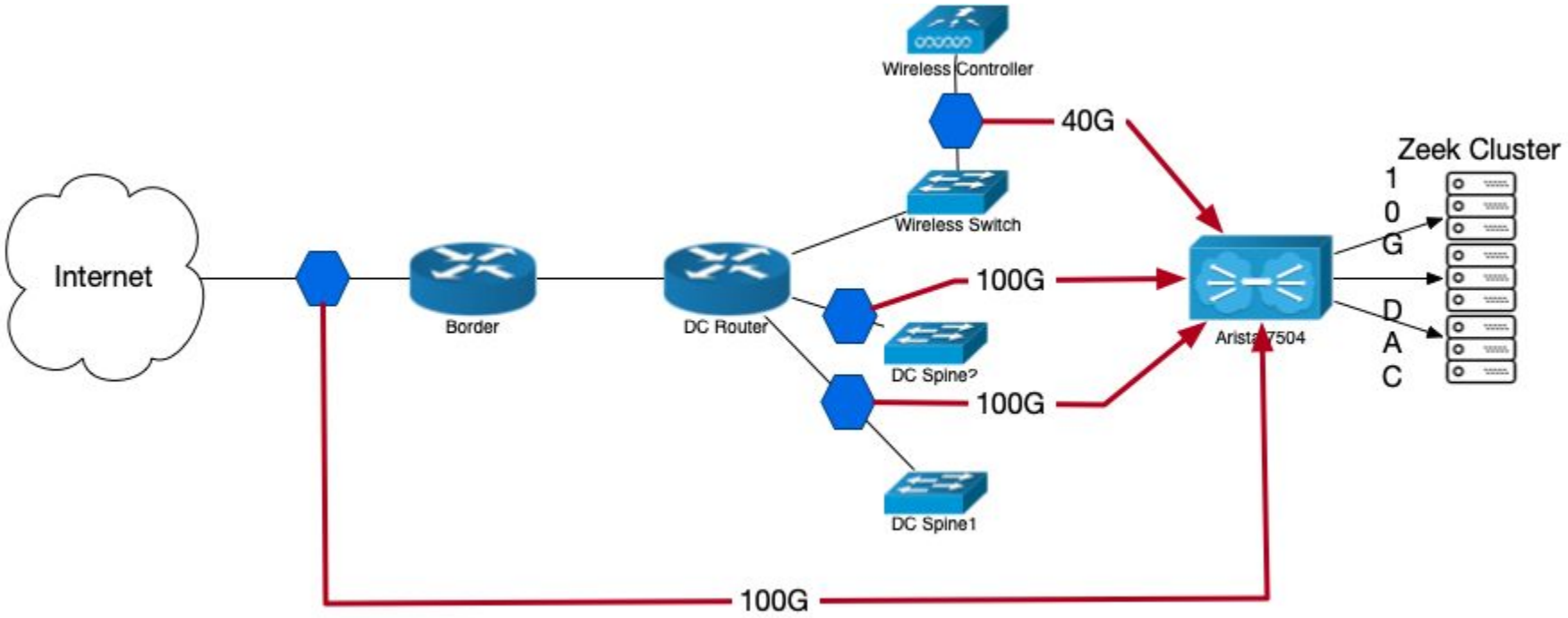


N/w Architecture for Zeek deployment #2

At UD



Indiana University (One Campus Subset)



Filesystem layout

Filesystem Layout

- Not an exhaustive tour!
- Default Path : /opt/zeek/ or /usr/local/zeek
  - bin/
  - etc/
  - include/ - ignoring today
  - lib/ var/lib - ignoring today
  - logs/
  - share/
  - spool/

Filesystem Layout

- bin/
  - zeek-cut
    - Extract columns from zeek logs (non-JSON)
    - Convert Unix epoch
  - zeek
    - Zeek executable with various options to run
  - zkg
    - Zeek package manager executable
  - zeekctl
    - Cluster management tool
    - Python script, more about it later

Filesystem Layout

- etc/
  - network.cfg
    - Define your local networks
  - node.cfg
    - Configure a cluster
    - Extra features (lb_procs, pin_cpu, lb_method)
  - zeekctl.cfg
    - Configure cluster management tool
    - Set shorter rotation time

Filesystem Layout

- logs/
  - current
    - Default Log Path (cluster-mode)
  - current directory
    - Default Log Path (CLI mode, logs overwritten each run)
  - YYYY-MM-DD
    - Logs archived to this dir,rotated hourly by default

Filesystem Layout

- spool/
  - state.db
    - cluster worker state
  - zeekctl-config.sh
    - Cluster management variables
  - worker-1
    - .pid
    - .cmdline
    - .env_vars
  - logger
    - logs (current is just a symlink)

Filesystem Layout

- share/
  - zeek/base/
    - Base scripts
  - zeek/policy/
    - Additional scripts
  - zeek/site/
    - Site specific scripts
    - <Zkg packages dir>

Docker / Zeek Usage

Docker Setup

- Github Link- Docker How to doc
  - https://github.com/zeek/zeek-training/tree/master/Intro-to-Zeek22
- Docker image
  - Zeek v6.0.1 installed on the image- https://hub.docker.com/repository/docker/zeek/training
    - # docker pull zeek/training:intro23
    - # docker run -it --privileged zeek/training:intro23 /bin/bash
  - Path inside the container to training resources used during this training: /zeek/training-res/

Verifying if Zeek installed correctly:

/zeek/bin/zeek –h

export PATH=/zeek/bin:$PATH

MORNING BREAK [30 mins]

Exercise#1

Fire up a cluster

- Edit node.cfg - comment out the standalone section, and uncomment the logger, proxy, manager and worker-1 sections.
    - # vi /zeek/etc/node.cfg

```
//You have to compile Zeek with pf_ring
[worker-1]
type=worker
host=10.0.0.50
interface=eth0
lb_method=pf_ring
lb_procs=10
pin_cpus=2,3,4,5,6,7,8,9,10,11
```

```
//Supported out-of-the-box
[worker-1]
type=worker
host=localhost
interface=af_packet::eth0
lb_method=custom
lb_procs=8
pin_cpus=0,1,2,3,4,5,6,7
```

- Set shorter rotation time
    - # vi /zeek/etc/zeekctl.cfg
      LogRotationInterval = 180 //(3mins)
- Set your local network
      # vi /zeek/etc/network.cfg

- turn off the checksum with Zeek redef

  # vi /zeek/share/zeek/site/local.zeek

  redef ignore_checksums = T;
- Then run:
  - # zeekctl deploy
- Make sure cluster is running:
  - # zeekctl status
- Create some traffic for the cluster to sniff
  - # wget google.com
  - # ls /zeek/logs/current/
- Send few pcaps to the sniffing interface:
  - # tcpreplay -i eth0 --mbps=500 /zeek/training-res/*.pcap
  - # ls /zeek/logs/current
- Stop the cluster
  - # zeekctl stop

Customizing Zeek

- /zeek/share/zeek/site/local.zeek
  - local customizations
    - Can be done inline or separate script
  - enable or disable scripts
  - change variables

Customizing Zeek

Loading additional scripts:

nano /zeek/share/zeek/site/local.zeek
<Changes to local.zeek require restart in cluster mode>

@load policy/protocols/http/header-names

zeek –h
$ZEEKPATH

Zeek file search path order :
. (current directory)
/usr/local/zeek/share/zeek
/usr/local/zeek/share/zeek/policy
/usr/local/zeek/share/zeek/site

Customizing Zeek

- Zeek scripts have an export section.
  - Redefinable variables
  - main.zeek is best place to start looking

```
# less /zeek/share/zeek/base/protocols/http/main.zeek

export {

    # This setting changes if passwords used in
    # Basic-Auth are captured or not.
    option default_capture_password = F;
}
```

Customizing Zeek

Value

```
redef SSH::password_guesses_limit = 15;
```

Table

```
redef SSH::ignore_guessers += {
   [192.168.1.2/32] = 192.168.1.20/32 };
```

Boolean

```
redef HTTP::log_server_header_names = T;
```

Config Framework

Change values on the fly

```
module Test;

export {
    option enable_feature = F;
}

@redef Test::enable_feature = T;
```

Define config option file

```
---
redef Config::config_files += { “/zeek/share/zeek/site/config.dat” };
```

Format

```
—
[option name][tab/space][new value]
Test::enable_feature     T
```

Customizing Zeek (10 min)

vi /zeek/share/zeek/site/local.zeek

Log server header names in HTTP (in local.zeek)
@load policy/protocols/http/header-names

Start with : /zeek/training-res/http-auth.pcap
tcpreplay -i eth0 --mbps=500 /zeek/training-res/http-basic-auth.pcap

Log http usernames and passwords
redef HTTP::default_capture_password=T;

tcpreplay -i eth0 --mbps=500 /zeek/training-res/http-auth.pcap

Package Manager (zkg)

- Package Manager
  - zkg
  - https://packages.zeek.org

- Can add new scripts or analyzers
  - Analyzer examples :
    - QUIC
    - PerfSonar
  - Scripts :
    - BZAR
    - DHCP OUI data

Let's Use zkg!

- List all zeek packages and install a few
  ```
  # zkg refresh
  # zkg list all
  # zkg install cve-2020-0601
  ```

- Check if they got installed
  ```
  # zkg list installed
  # ls /zeek/share/zeek/site
  ```

- Let's load the packages
  uncomment *@load packages* in site/local.zeek

Let’s Use zkg!

- zkg load/unload and info
  ```
  # zkg unload cve-2020-0601
  # zkg info cve-2020-0601
  ```
- Remove and uninstall all the packages
  ```
  # zkg purge
  # zkg list installed
  ```
- Other useful zkg commands
  remove, pin, unpin, upgrade, refresh

Zkg Exercise

10 minutes..

Running Zeek with zkg installed packages, specifically with JA3.

```
# zkg install ja3
# zeekctl deploy
# tcpreplay -i eth0 --mbps=500 /zeek/training-res/extract.pcap
# less /usr/local/zeek/logs/current/ssl.log
# cat ssl.log | zeek-cut -d ja3 ja3s
```

Zkg Exercise

Extract a file from a file transfer connection using file-extract zkg package

```
# zkg install file-extraction
```

Add all file types extraction to the config of the package:

```
# vi /zeek/share/zeek/site/packages/file-extraction/config.zeek
redef path = "/extracted_files/";
@load ./plugins/extract-all-files.zeek


# mkdir /extracted_files
# zeekctl deploy
# tcpreplay -i eth0 --mbps=500 /zeek/training-res/extract.pcap
# ls /extracted_files/
# less files.log
```

LUNCH Break

[Be back by 2pm]

HAND OFF TO CHRISTIAN

The Logging Framework

Packets → Zeek → ?

Packets → Zeek → Logs!

Logging Framework: Basic Purpose

- Manages all aspects of log writes in Zeek
- Provides APIs to Zeek scripts to
  - Define logs and their structure
  - Configure output technologies and their properties
  - Write individual log entries
  - Extend a log's structure
  - Adapt and filter log writes
- Zeek's default scripts provide ~60 logs
- Main reason people start using Zeek!

Logging Framework: Main Concepts

- Streams — what to log
  - Corresponds to a single log (for example, conn.log)
  - Defines fields with name and type
- Filters — how & whether to log
  - Determines what gets written and where
  - Each stream has a default filter that just writes everything
- Writers — where to log to
  - Defines output format
  - Zeek ships with ASCII writer (tab-separated or JSON) and SQLite writer
  - Other available via Zeek packages

Logging Framework: Defining a Log

```
# Create ID for our new stream. This identifies the log in a module.
redef enum Log::ID += { LOG };

# Define (or redef, in order to extend) the record type to log.
# This defines all log columns and data types:
type Info: record {
    ts: time        &log;
    id: conn_id     &log;
    service: string &log &optional;
    missed_bytes: count &log &default=0;
};

event zeek_init() {
    # Create the stream. This adds a default filter that writes ASCII logs.
    Log::create_stream(Foo::LOG, [$columns=Info,
                                  $path="foo"]);
}
```

Logging Framework: Writing a Log Entry

```
# Identify a suitable place to write the log entry.
# This is usually in an event handler; often when
# Zeek is done with a connection:
event connection_state_remove(c: connection) {
    local rec: Foo::Info = [$ts=network_time(), $id=c$id];
    Log::write(Foo::LOG, rec);
}
```

Logging Framework: Behold foo.log!

```
#separator \x09
#set_separator  ,
#empty_field (empty)
#unset_field -
#path   foo
#open   2019-04-07-00-27-05
#fields ts   id.orig_h      id.orig_p      id.resp_h      id.resp_p      service missed_bytes
#types  time addr port addr port string  count
1052146262.950001      203.241.248.20  3051     80.4.124.41    80   -    0
#close  2019-04-07-00-27-05
```

Logging Framework: Behold foo.log (in JSON)

```
{"ts":1052146262.950001,
 "id.orig_h":"203.241.248.20",
 "id.orig_p":3051,
 "id.resp_h":"80.4.124.41",
 "id.resp_p":80,
 "missed_bytes":0}
```

- Enabled by an ASCII writer configuration setting:

  ```
  redef LogAscii::use_json = T;
  ```

- Also see the `add-json` Zeek package

Logging Framework: Add a Field to conn.log

```
redef record Conn::Info += {
    is_private: bool &default=F &log;
};

event connection_state_remove(c: connection) {
    if ( c$id$resp_h in Site::private_address_space )
        c$conn$is_private = T;
}
```

- Note how this just adds state, nothing changes about the existing log write(s)
- Available in training material: add the following to local.zeek and redeploy.
  @load /zeek/training-res/log-private.zeek
- Replay some traffic to trigger log writes:
  # tcpreplay -i eth0 --mbps=500 /zeek/training-res/mal.pcap

Logging Framework: Disable a Whole Log

```
event zeek_init() {
    Log::disable_stream(Syslog::LOG);
}
```

Logging Framework: Steering Output

```
function myfunc(id: Log::ID, path: string, rec: HTTP::Info) : string {
    local r = Site::is_local_addr(rec$id$resp_h) ? "local" : "remote";
    return fmt("%s-%s", path, r);
}

event zeek_init() {
    Log::remove_filter(HTTP::Log, "default");
    local filter: Log::Filter = [$name="http-split",
$path_func=myfunc];
    Log::add_filter(HTTP::LOG, filter);
}
```

- Available in training material: add the following to local.zeek and redeploy.
  @load /zeek/training-res/split-path.zeek
- Replay some traffic to trigger log writes:
  # tcpreplay -i eth0 --mbps=500 /zeek/training-res/*.pcap

Logging Framework: Filtering Log Writes

- Built on the concept of hooks in the Zeek language
  - Hooks are like events but synchronous: think callbacks.
  - One hook can have multiple handlers, any of which can shortcut further processing.
- `Log::log_stream_policy(rec: any, id: ID);`
  - Global policy hook, invoked for every log write
  - Great way to interpose on any log write in the system
- `Log::PolicyHook: hook(rec: any, id: ID, filter: Filter);`
  - Each log can also define its own policy hook, for filtering specific to this log.
- Issuing a "`break;`" in a hook will veto the log write.

Logging Framework: Further Reading

- Read the docs: https://docs.zeek.org/en/master/frameworks/logging.html
- Watch Justin's talk at ZeekWeek'22.
- Real-world examples of log additions/tweaks:
  - This adds a Community ID field to conn.log
    `@load protocols/conn/community-id-logging`
  - An interesting example of adding columns
    https://github.com/corelight/log-add-vlan-everywhere
  - Packages that add parsers usually add logs too
    https://github.com/cisagov/icsnpp-bacnet

The Notice Framework

Notice Framework

- Notices convey interesting activity that Zeek encounters. They can be alerts, depending on your site's policy. Many places in Zeek trigger notices, and you can add more.
- Notices have actions:
  - ACTION_LOG, ACTION_ALARM, ACTION_EMAIL, …
- Hooks allow applying actions and modification of notices before they're sent on to the action plugins:
  - hook Notice::policy(n: Notice:Info)
  - Hooks can have priorities (default is 0, higher means earlier)
  - Hooks can abort later hook bodies with the break keyword.

Notice Framework

Raising Notices

- NOTICE function is called to raise a notice for any occurrence that a user may want to be notified about or take action on:

```
NOTICE([$note=Password_Guessing,
        $msg=fmt("%s appears to be guessing SSH passwords (seen in %d conns).",
        key$host, r$num),
        $src=key$host,
        $identifier=cat(key$host)]);
```

Notice Framework

Automated Suppression

- Notices have built-in suppression (to avoid repeated events)
  - Generally controlled/defined by the script writer
  - “intrinsically duplicate” notices suppression is implemented with an optional field in Notice::Info records named $identifier which is a simple string.
  - $suppress_for optional field can be used to configure custom interval.

Notice Exercise

- Install scan detection:
  # zkg install bro-simple-scan
- Uncomment @load packages in local.zeek
- Run against mal.pcap
  # zeekctl deploy
  # tcpreplay -i eth0 --mbps=500 /zeek/training-res/*.pcap
- Look at the generated notice.log file

The Telemetry Framework

Telemetry Framework

- This framework provides metrics about Zeek's internal state, as well as characteristics of the monitored traffic.
- It's fairly Prometheus-inspired and supports scraping.
- Zeek ships with some metrics by default. You can add others by scripting.
- These metrics are not a Zeek log data export mechanism.

Telemetry Framework

- Supports the same metric types as most Prometheus client libraries, with the exception of the Summary type.
- Metrics types:
  - Counter - Continuously increasing, resets on process restart.
  - Gauge - Gauge metric can increase and decrease
  - Histogram - Pre-configured buckets of observations.

Telemetry Framework

In a cluster setting:

- Metrics can be collected on manager node.
  - No aggregation of metrics happen at the collecting node
  - Each node has its own metric registry independent of the other nodes.
  - The centralized metrics receive an additional “endpoint” label that can be used to identify the originating node.

Telemetry Framework

Logging and Publishing of metrics:

- By default telemetry.log is written for the metrics of each node in a cluster.
- To allow scraping, the manager process can be configured to run a HTTP server on port 9911/tcp for Prometheus exposition:

  ```
  @load frameworks/telemetry/prometheus
  ```

- To just scrape available metrics without a cluster setup, try:

  ```
  $ zeek exit_only_after_terminate=T Broker::metrics_port=9911/tcp
  ```

Telemetry Framework - Exercise

- Load the following script in local.zeek

  # vi /zeek/share/zeek/site/local.zeek

  @load frameworks/telemetry/prometheus

- Redeploy the zeek cluster

  # zeekctl stop

  # zeekctl deploy

- Scrape the metrics exposed on port 9911

  # curl -s localhost:9911/metrics | grep version

Telemetry Framework - Exercise cont.

- Scrape the metrics exposed on port 9911

  # curl -s localhost:9911/metrics | grep manager

  # curl -s localhost:9911/metrics | grep log_writes

- Look at the default telemetry.log file

  # less /zeek/logs/current/telemetry.log

  # less /zeek/logs/current/telemetry.log | grep manager

Telemetry Framework - Custom Metrics Example

- Look at the custom script to count the number of http requests:
  # cat /zeek/training-res/http-counter.zeek

Telemetry Framework - Custom Metrics Example

- Load the following script in local.zeek

  #vi /zeek/share/zeek/site/local.zeek

  @load /zeek/training-res/http-counter.zeek

- Redeploy the zeek cluster

  # zeekctl stop

  # zeekctl deploy

- Scrape the metrics exposed on port 9911

  # curl -s localhost:9911/metrics | grep monitored_http_requests

AFTERNOON BREAK [30 mins]

Meet the

Management Framework

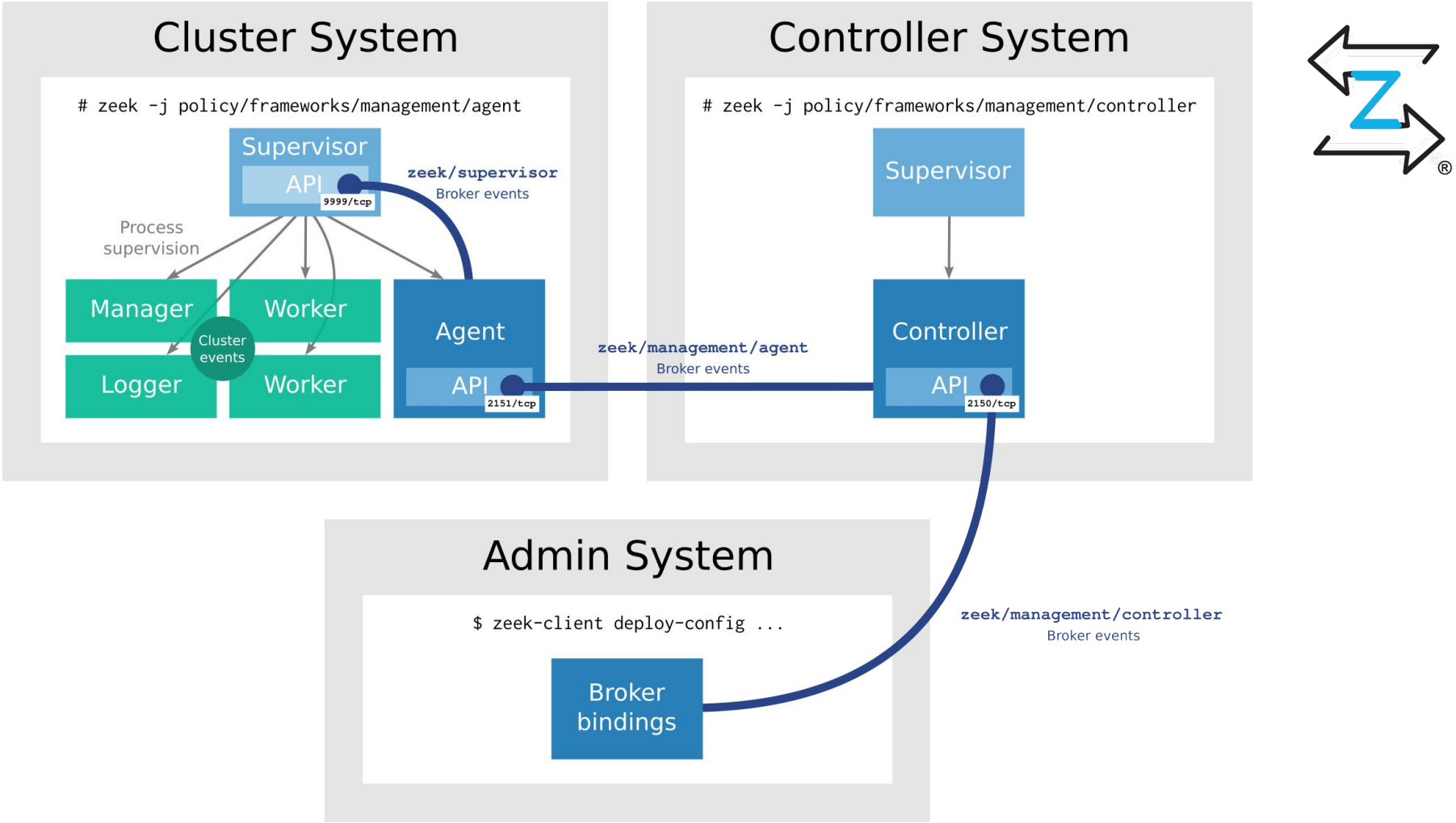
`policy/frameworks/management`

Goals of this effort

- Establish modern cluster management for Zeek
  - zeekctl's model isn't a great fit for today's infrastructure
    - ssh, rsync, and command invocations
  - Lots of Zeek-external activity — but also solid & featureful!
- Zeek-based Service model
  - Zeek as a service manager
  - Events for messaging
  - Build on Zeek's supervisor framework
  - Be container-friendly
- Feature parity with zeekctl is not a goal, but a guideline

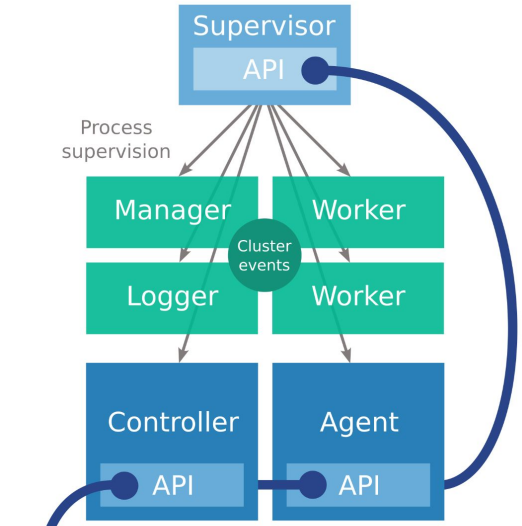
Expectation management

- The Management Framework is an emerging feature
  - Great for single-machine settings & exploring clusterized Zeek
  - Multi-machine settings not fully supported yet
- For production settings, zeekctl remains the right choice
- We plan to deprecate zeekctl in the Zeek 7 timeframe


Cluster System
zeek -j policy/frameworks/management/agent
Supervisor
API
9999/tcp
zeek/supervisor
Broker events
Process supervision
Manager
Worker
Cluster events
Logger
Worker
Agent
API
2151/tcp
zeek/management/agent
Broker events
Controller System
zeek -j policy/frameworks/management/controller
Supervisor
Controller
API
2150/tcp
Admin System
$ zeek-client deploy-config ...
Broker bindings
zeek/management/controller
Broker events

All-in-one System

```
# zeek -j policy/frameworks/management/agent \
    policy/frameworks/management/controller
```




```
$ zeek-client deploy-config ...
```

Check out the documentation:

https://docs.zeek.org/en/master/frameworks/management.html

Let’s get started!

- Prepare two shells in a fresh container

  To launch additional shells in the existing container, try:

  ```
  $ docker exec -it <container> bash
  ```

- Launch an all-in-one system

```
# zeek -j policy/frameworks/management/controller
    policy/frameworks/management/agent
```

Note the “-j”: we’ll use Zeek’s Supervisor, a built-in process manager

- Inspect the result

```
# netstat -tpln
# pstree -pTU 0
```

- Meet zeek-client, the default client CLI

```
# zeek-client --help
# zeek-client show-config
# zeek-client --set <TAB>
```

- Inspect available management instances

```
# zeek-client get-instances
```

zeek-client's output is generally in JSON, suitable for piping into jq etc.

- Write a first cluster config: cluster-minimal.cfg

```
[manager]
role = manager

[logger]
role = logger

[worker-01]
role = worker
interface = lo
```

- Staging and retrieving the configuration

```
# zeek-client stage-config cluster-minimal.cfg
# zeek-client get-config
# zeek-client get-config --as-json
```

- Ctrl-C the Zeek tree, start it up again, retrieve again

```
# zeek-client get-config
```

The controller has persisted the configuration, reloaded it, and is serving it again.

- Let’s run it!

```
# zeek-client deploy
```

- Retrieve current node state

```
# zeek-client get-nodes
```

A view across all instances showing Zeek nodes with their roles, ports, and PIDs.

- Ctrl-C the Zeek tree, start it up again, retrieve nodes again

```
# zeek-client get-nodes
```

The controller has reloaded and re-deployed the configuration.

- Let’s make a mistake

  Edit the configuration: add another worker, give it an invalid interface.
  (See cluster-error.cfg)

- Upload and deploy this, in a single command

```
# zeek-client deploy-config cluster-error.cfg
```

- Let’s tweak logging and run some traffic

  To study logging, we’ll dial down the log rotation interval. Create local.zeek with this line:

  ```
  redef Log::default_rotation_interval = 20 sec;
  ```

  Restart everything with this file included — it propagates through to all nodes:

  ```
  # zeek -j policy/frameworks/management/controller
      policy/frameworks/management/agent local.zeek
  ```

- Replay a pcap

  ```
  # tcpreplay -i eth0 -mbps=500 /zeek/training-res/extract.pcap
  ```

- Let’s take a look at log production now.

- Step 1: logs get written in local output folders

  Take some time to poke around ${ZEEKROOT}/var/lib/nodes/<node>/.

- Step 2: log rotation

  Rotated logs get placed into ${ZEEKROOT}/spool/log-queue/.
  Log rotation interval: `Log::default_rotation_interval`, as usual, default: 1h.

- Step 3: log archival

  Log rotation concludes with log archival into ${ZEEKROOT}/logs/.
  By default, log archival happens once per rotation interval. To adjust, redef `Management::Agent::archive_interval`.

  For details and additional customization knobs, see:
  https://docs.zeek.org/en/master/frameworks/management.html#log-management

- ## Restart nodes

```
# zeek-client restart [<node1> …]
```

Compare affected PIDs in subsequent `get-nodes` outputs.

- ## Inspect cluster node state

You can retrieve global ID values from any set of Zeek nodes. They get rendered in JSON. A simple one:

```
# zeek-client get-id-value ignore_checksums
```

A more complex one:

```
# zeek-client get-id-value Log::active_streams manager
```

Multiple instances

Let's approximate typical future deployment: multiple Zeek hosts, each with their own instance, and a separate controller. You need three shells, one for each of the following:

```
# zeek -j policy/frameworks/management/controller
# zeek -j policy/frameworks/management/agent
    /zeek/training-res/management/local.inst1.zeek
# zeek -C -j policy/frameworks/management/agent
    /zeek/training-res/management/local.inst2.zeek
```

Deploy a cluster across these instances:

```
# zeek-client deploy-config cluster-multi-inst.cfg
```

Now explore via `get-nodes`, check out `pstree`, etc.

Adjacent technologies: JavaScript

- Starting with Zeek 6, you can script Zeek in JavaScript as well.
- This is an experimental extension, not a full substitute for the Zeek language.
- This runs a Node interpreter and can use packages from the Node ecosystem.
- Really good for quick prototyping, API interaction with other systems, etc
- Implemented as a Zeek plugin!
- See https://docs.zeek.org/en/master/scripting/javascript.html for details.

Adjacent technologies: Spicy

- Spicy is a parser generator for structured data
- Domain-specific language to capture syntax and semantics
- Compiles to a safe C++ runtime that is independent of Zeek
  - Prototypes exist e.g. for Wireshark
- Zeek supports Spicy parsers via a glue layer to its event loop
- For an example, see Zeek's QUIC parser sources (new with 6.1)
- See https://docs.zeek.org/projects/spicy/en/latest/

More Zeek Resources

Zeek Documentation:
https://docs.zeek.org/en/master/

Zeek packages:
https://packages.zeek.org/packages

Zeek btest pcaps:
https://github.com/zeek/zeek/tree/master/testing/btest/Traces

Zeek Github source:
https://github.com/zeek/zeek

Zeek online:
https://try.zeek.org/#/?example=hello

Thanks for attending!
Enjoy Zeeking

Stay Connected

Website - https://zeek.org

Slack - https://zeek.org/slack

Discourse - https://community.zeek.org

Find out more ways to connect at: https://zeek.org/community/