BERKELEY LAB

Lawrence Berkeley National Laboratory

U.S. DEPARTMENT OF ENERGY

UNIVERSITY OF CALIFORNIA

BERKELEY LAB

LAWRENCE BERKELEY NATIONAL LABORATORY

U.S. DEPARTMENT OF ENERGY

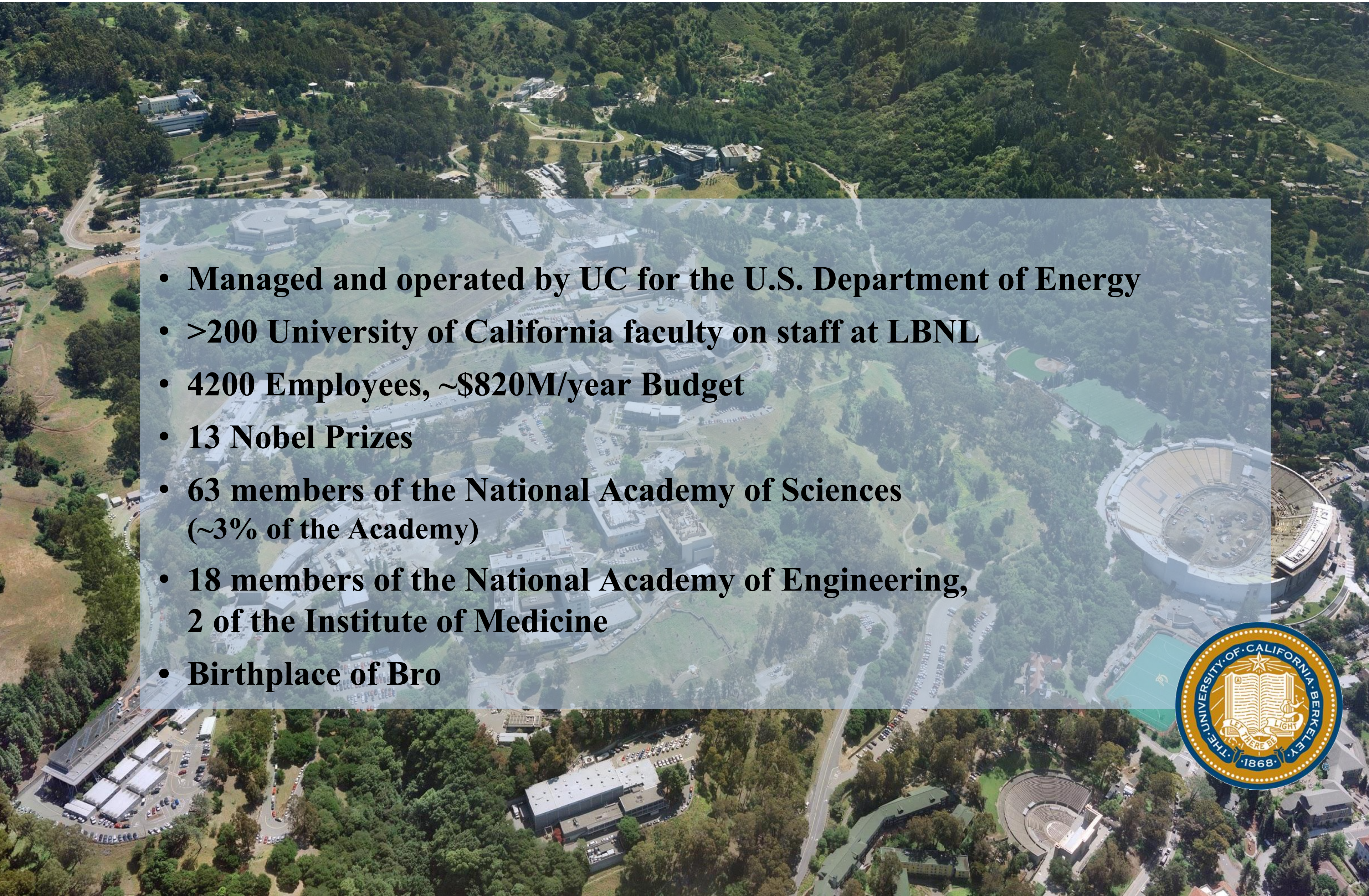
Scan-detection Internals: clusterization and netcontrol for active-response

Aashish Sharma

14th September 2016

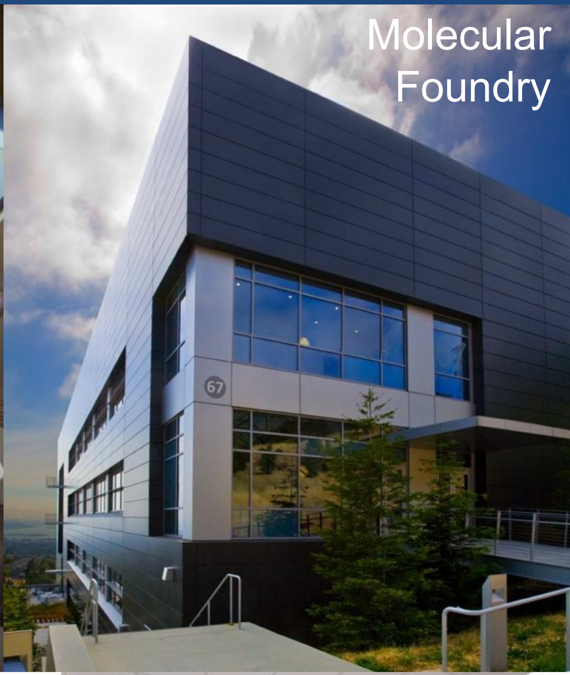
#BroCon2016

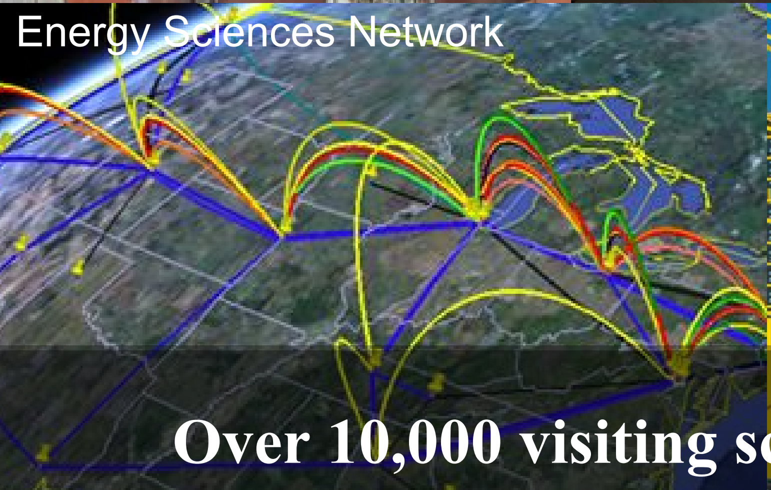
80 Years of World-Leading Team Science at Lawrence Berkeley National Laboratory

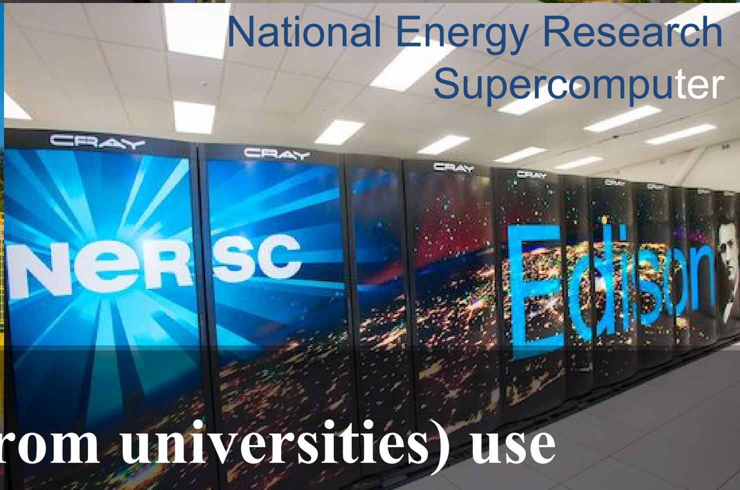


- **Managed and operated by UC for the U.S. Department of Energy**
- **>200 University of California faculty on staff at LBNL**
- **4200 Employees, ~$820M/year Budget**
- **13 Nobel Prizes**
- **63 members of the National Academy of Sciences**
  (~3% of the Academy)
- **18 members of the National Academy of Engineering,**
  2 of the Institute of Medicine
- **Birthplace of Bro**

World-Class User Facilities
Serving the Nation and the World







Over 10,000 visiting scientists (~2/3 from universities) use Berkeley Lab research facilities each year

Overview

- A case for scan-detection
- Internals of scan-detection
  - what is a scan
- Clusterization and its problems
- Scan-NG features and how are those implemented
- What's in for the future

Philosophically a scan is an attribution or an intentionality problem but operationally we want to make it a measurement problem.

- Partha Banerjee, LBL

Recon

- We want to know if scans are coordinated, distributed*
- What is the scale of a recon ?
- what is intention of a recon ?
- No clear success criteria of a recon
- Don't even know what attackers found out, although the traffic went through your network

*M. Javed and V. Paxson. Detecting stealthy, distributed SSH brute-forcing. In Proc. ACM SIGSAC conference on Computer & communications security, pages 85–96, 2013

TABLE IV. ATTACK PHASES

Attack Phase	Description	Incident Count
Scan Phase	Attackers try to identify vulnerable hosts and gather information about the target, e.g., services that are running.	1/1
Breach Phase	Attackers gain access to the system (e.g., using stolen or guessed credentials or by exploiting system misconfiguration (e.g., world writable files on an open share).	30/39
Penetration	Attackers exploit vulnerability (e.g., buffer overflow vulnerability) to obtain unauthorized access to the system.	9/10
Control	Attackers set up the compromised host to accept remote commands and provide reusable access (e.g., connect to command and control channel or install a backdoor).	21/23
Embedding	Attackers hide their malware and tracks by embedding the malware in the system, e.g., installing a rootkit, deleting system logs, adding ssh keys to authorized_key file, changing configuration files.	8/9
Data extraction/ modification	Attackers change or modify data in the system, e.g., deface web pages, copy database content, or steal information.	7/7
Attack-relay/ misuse	Attackers start misusing the system for personal gain, e.g., spam, DDoS using a bot, password harvesting, distributing warez, spreading virus, and phishing.	48/61

Q. How many incidents are detected at Scan Phase?

Ans: We might not even have an incident yet (at the scan phase)

Q. Of all the incidents we detect, for how many can we go back to and find the scan-phase that might have caused it ?

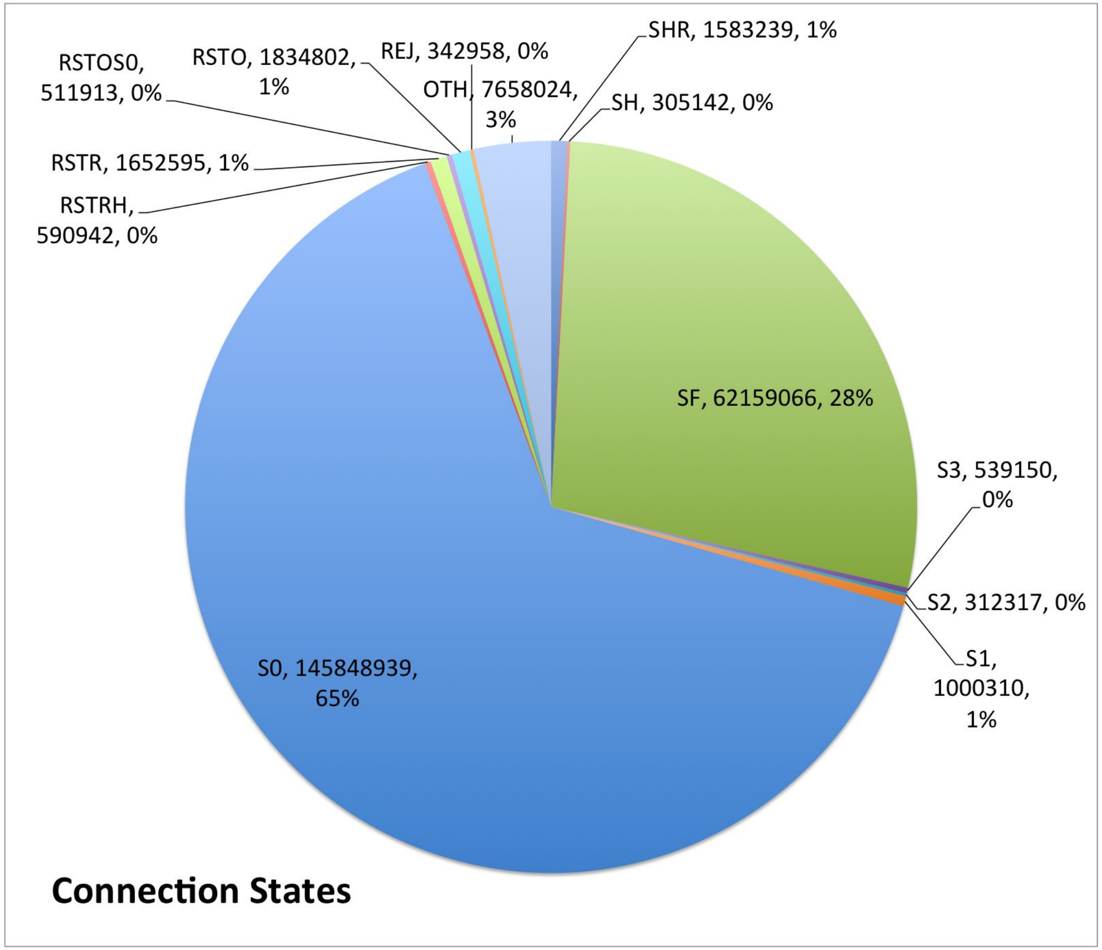
Q. How many incidents happen without any scan-phase/recon ?

Sharma, A., Kalbarczyk, Z., Barlow, J., and Iyer, R. Analysis of security data from a large computing organization. In Dependable Systems & Networks (DSN) (2011), IEEE.

Why scan-detection ?

- Important to know about malicious activity early and quickly
- Attention to recon is as important as any other defense mechanism

Characteristics of network traffic

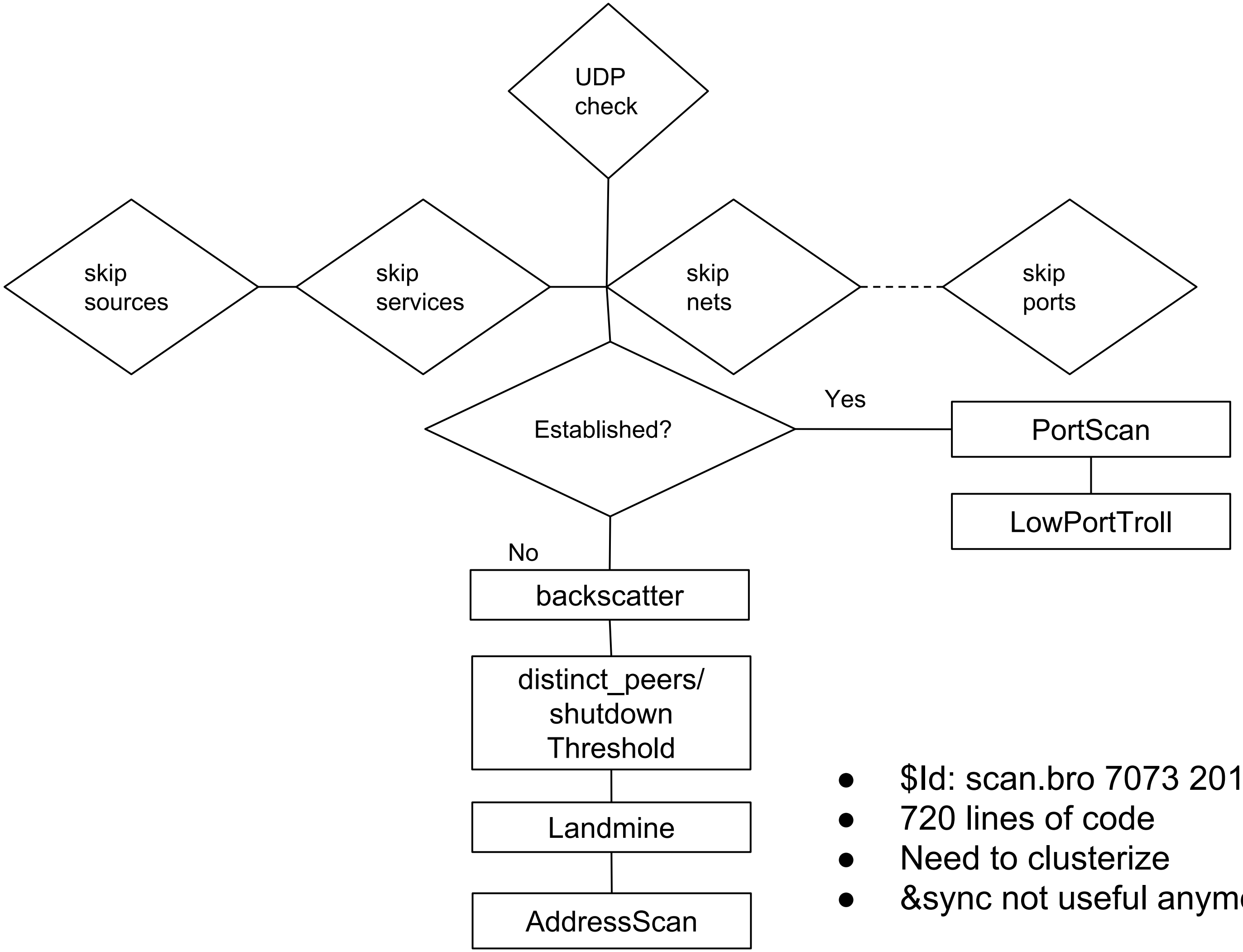


S0 Connection attempt seen, no reply.
S1 Connection established, not terminated
SF Normal establishment and termination.
REJ Connection attempt rejected.
S2 Connection established and close attempt by originator seen (but no reply from responder).
S3 Connection established and close attempt by responder seen (but no reply from originator).
RSTO Connection established, originator aborted (sent a RST).
RSTR Established, responder aborted.
RSTOS0 Originator sent a SYN followed by a RST, we never saw a SYN-ACK from the responder.
RSTRH Responder sent a SYN ACK followed by a RST, we never saw a SYN from the (purported) originator.
SH Originator sent a SYN followed by a FIN, we never saw a SYN ACK from the responder (hence the connection was "half" open).
SHR Responder sent a SYN ACK followed by a FIN, we never saw a SYN from the originator.
OTH No SYN seen, just midstream traffic (a "partial connection" that was not later closed).

Strategies for scan-detection

- Summary statistics
  - “N” IP or port in “t” time
- Signature Based
  - eg. Metasploit signature
- Behavior Based
  - Nmap scans start with 80/tcp, 443/tcp + icmp
- Probabilistic methods
  - Threshold Random Walk
- know_your_network_approach
  - Knockknock and Landmine

Overly simplified OldScan-1.5.3



- $Id: scan.bro 7073 2010-09-13 00:45:02Z vern $
- 720 lines of code
- Need to clusterize
- &sync not useful anymore

scan.bro - One pill to cure all?

Scan detection needs to be broken into many sub-parts

- TCP
- UDP
- ICMP
- IPv4
- IPv6
- external
- internal scanners

Scan-detection: Underlying Reasoning...

- WE KNOW WHAT THEY DON'T KNOW
- WE DON'T KNOW WHAT THEY FOUND OUT
- WE WANT TO KNOW WHAT IS THEY WANT TO KNOW (hopefully before they find it out)

Heuristics

KnockKnock	LandMine
• Incoming remote IP connection and checks it against **table of known-services** for the LBNL IP and accesses if that's a good or bad connection. • Policy is adaptive based on popularity of ports	• Policy - ingests the list of allocated subnets from a text-file using input-framework • Any connection not in the above list is a Darknet Connection • "N" such connections lead to a conclusion that this is a scanner • Block the IP.
AddressScan & LowPortTrolling • "Bro treats connections differently depending on application protocol. • Bro only performs bookkeeping if the connection attempt failed (was either unanswered, or elicited a TCP RST response). • For others, it considers all connections, whether or not they failed. It then tallies the number of distinct destination addresses to which such connections (attempts) were made. • If the number reaches a configurable parameter N, then Bro flags the source address as a scanner. By default, Bro sets N = 100"	**Backscatter** • Generally Victims of DoS attacks • result of address spoofing • Not really scanners

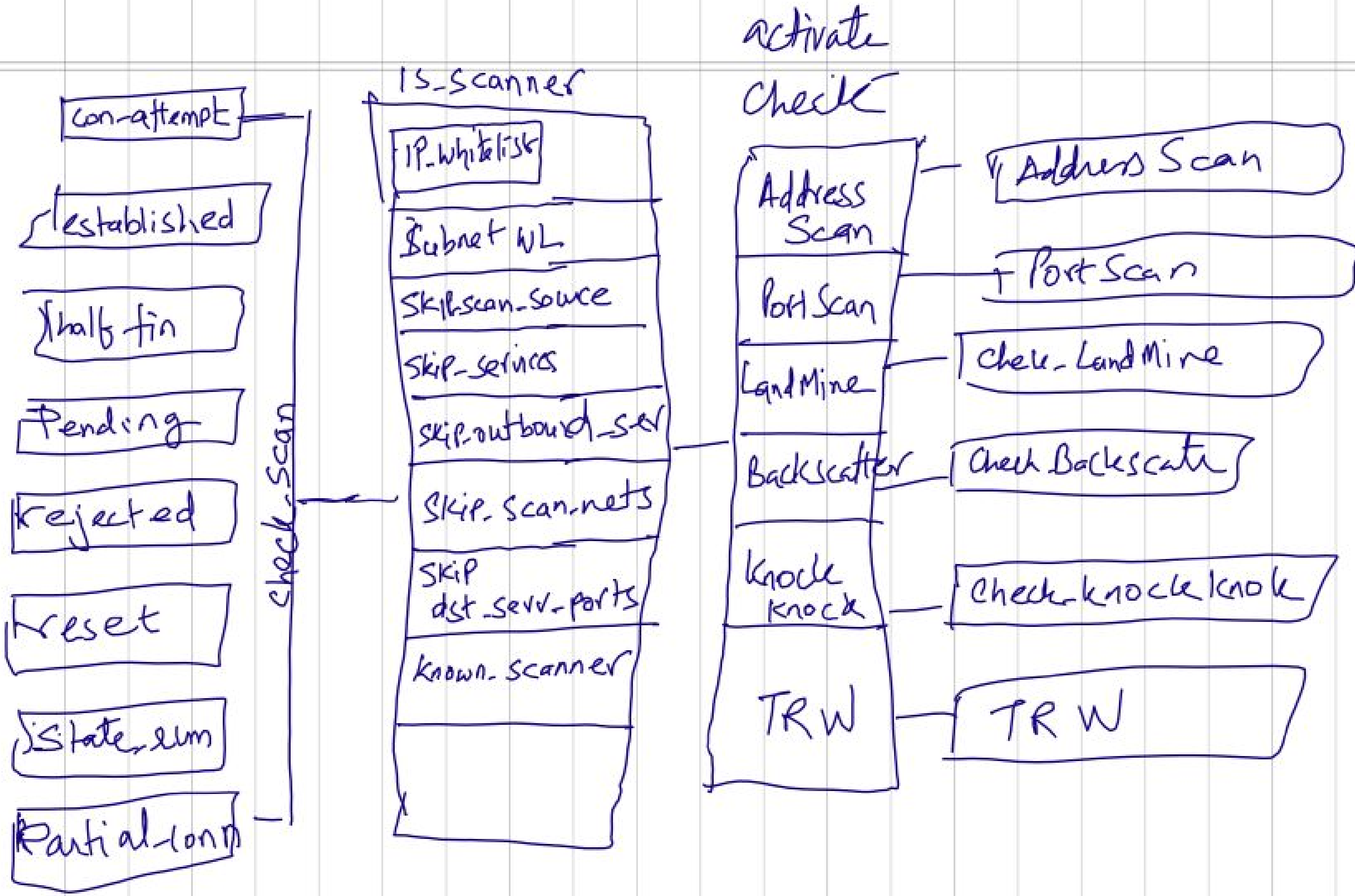
Potential issues with clusterization of scan-detection

- Communication overhead - Scan detection is kind of the worst-case for distributed analysis: one needs to count across *all* connections.
- In a cluster we split things up via load-balancing, but for scan detection we need to essentially revert that through communication.
- Timely state synchronization across the workers
- Scans are unpredictable rates so cannot employ epochs
  - need to detect fast and slow scanners both
- How to implement dynamic thresholds
- Detection needs to run in both cluster and standalone setup

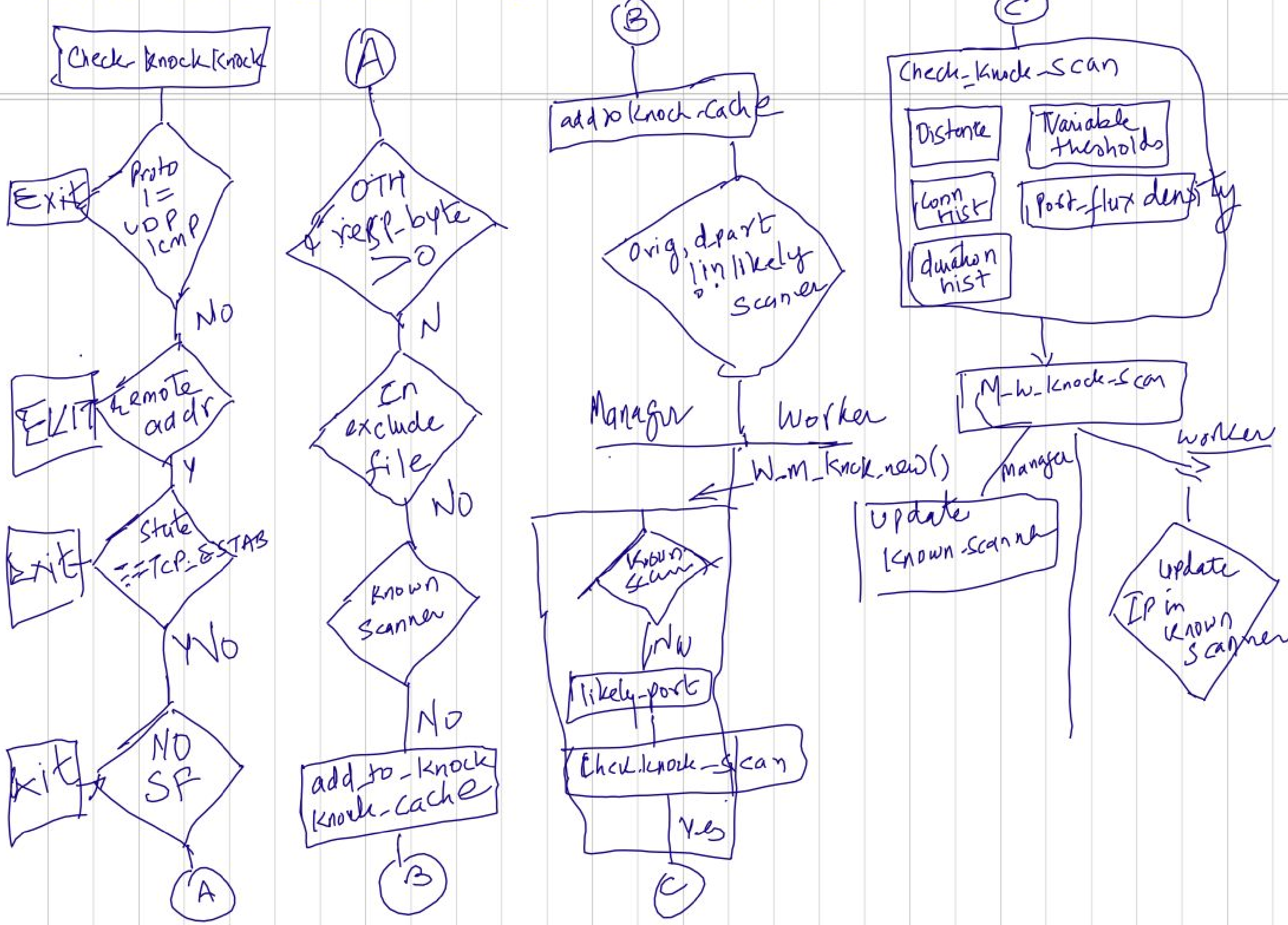
Events Table

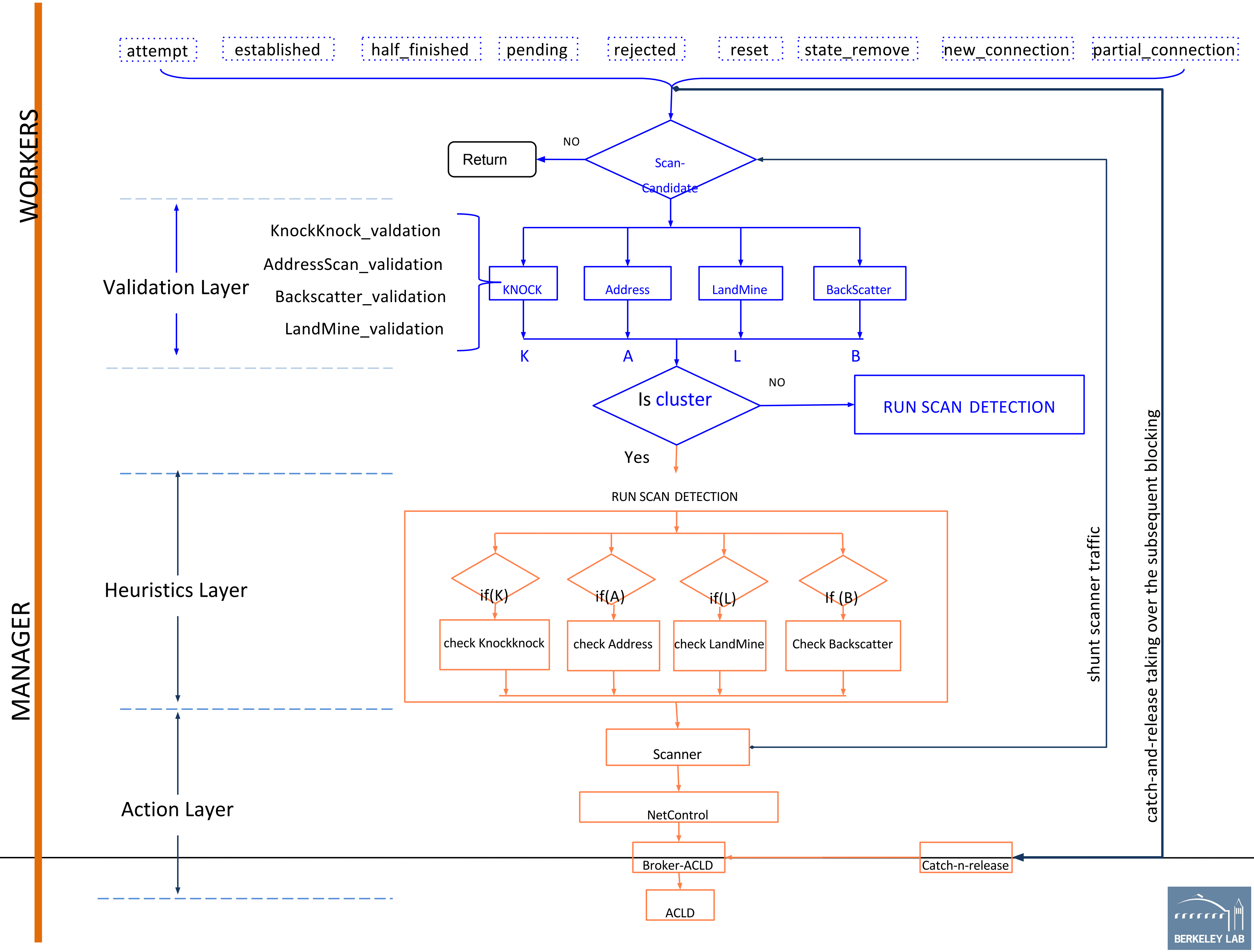
Event	Description
connection_attempt	This event is raised when an originator unsuccessfully attempted to establish a connection. “Unsuccessful” is defined as at least **tcp_attempt_delay** seconds having elapsed since the originator first sent a connection establishment packet to the destination without seeing a reply
connection_established	Generated when seeing a SYN-ACK packet from the responder in a TCP handshake. An associated SYN packet was not seen from the originator side if its state is not set to TCP_ESTABLISHED. The final ACK of the handshake in response to SYN-ACK may or may not occur later, one way to tell is to check the history field of connection to see if the originator sent an ACK, indicated by ‘A’ in the history string.
connection_half_finished	Generated when one endpoint of a TCP connection attempted to gracefully close the connection, but the other endpoint is in the TCP_INACTIVE state. This can happen due to split routing, in which Bro only sees one side of a connection.
connection_pending	Generated for each still-open TCP connection when Bro terminates.
connection_rejected	Generated for a rejected TCP connection. This event is raised when an originator attempted to setup a TCP connection but the responder replied with a RST packet denying it.
connection_reset	Generated when an endpoint aborted a TCP connection. The event is raised when one endpoint of an established TCP connection aborted by sending a RST packet.
connection_state_remove	Generated when a connection’s internal state is about to be removed from memory. Bro generates this event reliably once for every connection when it is about to delete the internal state. As such, the event is well-suited for script-level cleanup that needs to be performed for every connection. This event is generated not only for TCP sessions but also for UDP and ICMP flows.
new_connection	Generated for every new connection. This event is raised with the first packet of a previously unknown connection. Bro uses a flow-based definition of “connection” here that includes not only TCP sessions but also UDP and ICMP flows
partial_connection	Generated for a new active TCP connection if Bro did not see the initial handshake. This event is raised when Bro (observed traffic from each endpoint, but the activity did not begin with the usual connection establishment

ARCHITECTURE DIAGRAM - 1



ARCHITECTURE - DIAGRAM - 2 - knockknockScan





WORKERS
MANAGER
attempt
established
half_finished
pending
rejected
reset
state_remove
new_connection
partial_connection
Return
NO
Scan-Candidate
Validation Layer
KnockKnock_valdation
AddressScan_validation
Backscatter_validation
LandMine_validation
KNOCK
Address
LandMine
BackScatter
K
A
L
B
Is cluster
NO
RUN SCAN DETECTION
Yes
Heuristics Layer
RUN SCAN DETECTION
if(K)
if(A)
if(L)
If (B)
check Knockknock
check Address
check LandMine
Check Backscatter
shunt scanner traffic
catch-and-release taking over the subsequent blocking
Action Layer
Scanner
NetControl
Broker-ACLD
Catch-n-release
ACLD
BERKELEY LAB

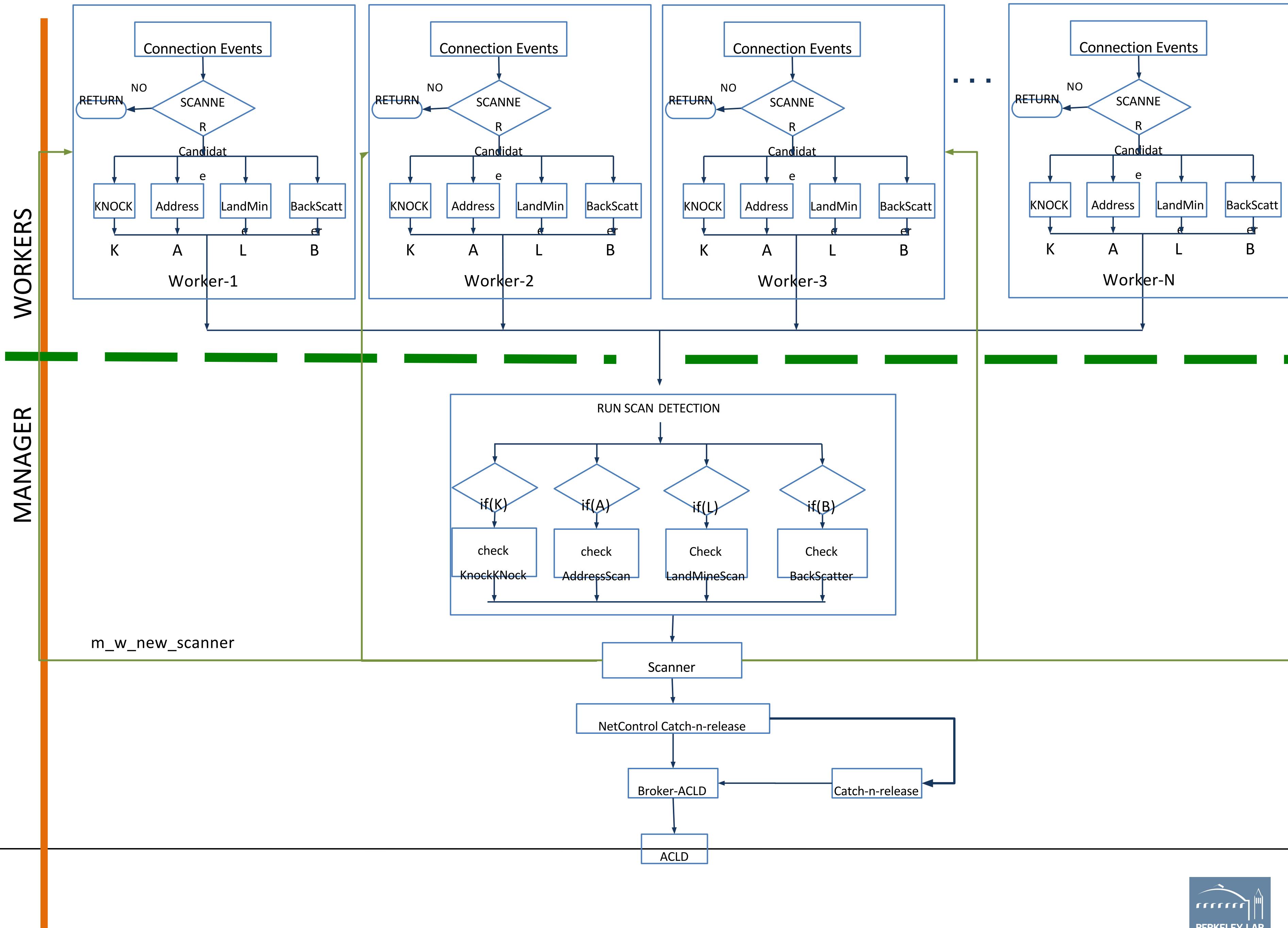
Filtration – what qualifies (or not qualifies) as a potential scan candidate

<table>
<tr><th>Desc</th><th>KnockKnock</th><th>LandMine</th><th>BackScatter</th><th>AddressScan</th></tr>
<tr><td>c$proto == TCP</td><td>Only TCP connections</td><td>Only TCP and ICMP</td><td>Only TCP connections</td><td>TCP and ICMP
(UDP disabled by default)</td></tr>
<tr><td>Internal Scanners</td><td>Internal scanners handled separately</td><td>- NA -</td><td>Internal host scanning handled separately</td><td>Internal host scanning handled separately</td></tr>
<tr><td>DARKNET</td><td>Fast-track Darknet</td><td>Fast-track Darknet</td><td>-</td><td>we ignore all darknet connections since LandMine will take care of it</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>(c$resp$state == TCP_ESTABLISHED)
OR
if (/SF/ in c$conn$conn_state)</td><td>full established conns not interesting</td><td>full established conns not interesting</td><td>Full established conns not interesting</td><td>Full established conns not interesting
if (established) return "" ;</td></tr>
<tr><td>Min_Subnet_check</td><td></td><td>if (|Site::subnet_table| < MIN_SUBNET_CHECK)
return F ;</td><td></td><td></td></tr>
<tr><td>(state == "OTH" && resp_bytes >0)</td><td># mid stream traffic - ignore</td><td></td><td># mid stream traffic - ignore</td><td></td></tr>
<tr><td>Pass/fail criteria</td><td>ignore traffic to host/port this is primarily whitelisting</td><td>if ((is_failed(c) || is_reverse_failed(c)))</td><td>(c$orig$state == TCP_SYN_ACK_SENT && c$resp$state == TCP_INACTIVE)
OR
(c$orig$state == TCP_SYN_SENT && c$resp$state == TCP_INACTIVE)
OR
(c$history == "F" || c$history == "R")
OR
(c$history == "H" && /s|a/ !in c$history))</td><td>Ignore if :
1) outbound && service in skip_outbound_services
2) local_address
3) orig in skin_scan_sources
4) orig in skip_scan_nets
5) outbound and [resp, service] in skip_dst_server_ports</td></tr>
</table>

Simple clusterization

```
module Clus;

export {

	global m_w_add: event (ip: addr);
	global w_m_new: event (ip: addr);
	global add_to_cache: function(ip: addr);

	global intermediate_cache: table [addr] of string &redef;

}



@if ( Cluster::is_enabled() )
@load base/frameworks/cluster
redef Cluster::manager2worker_events += /Clus::m_w_add/;
redef Cluster::worker2manager_events += /Clus::w_m_new/;
@endif


function log_reporter(msg: string)
{
	event reporter_info(current_time(), msg,
peer_description);
}


event new_connection(c: connection)
{
	local ip = c$id$orig_h;

	if (ip !in intermediate_cache)
	{
		add_to_cache(ip) ;
	}
}
```

```
function add_to_cache(ip: addr)
	{
	log_reporter(fmt ("add_to_cache %s", ip));
	intermediate_cache[ip] = fmt("%s",peer_description);
@if ( Cluster::is_enabled() )
	event Clus::w_m_new(ip);
@endif
	}


@if ( Cluster::is_enabled() && Cluster::local_node_type() == Cluster::MANAGER )
event Clus::w_m_new(ip: addr)
	{
	log_reporter(fmt ("w_m_new: %s", ip));
	if ( ip in intermediate_cache )
		return;

	intermediate_cache[ip] = fmt("%s",peer_description);
	event Clus::m_w_add(ip);
	}
@endif



@if ( Cluster::is_enabled() && Cluster::local_node_type() != Cluster::MANAGER )
event Clus::m_w_add(ip: addr)
	{
	log_reporter(fmt ("m_w_add: %s", ip));
	intermediate_cache[ip] = fmt("%s",peer_description);
	}
@endif
```



BERKELEY LAB

Old vs New

Heuristic	OldScan	scan-NG
LandMine	Limited: Manual define Landmine addresses *const landmine_address: set[addr] &redef;*	Extensive - derives allocated vs unallocated subnets *if (resp in Site::local_nets && resp !in Site::subnet_table)* **Extended feature**
AddressScan	Same global distinct_peers: table[addr] of set[addr]	No Change **Consistent**
Shutdown Threshold	Same > N failures	No change
Backscatter	Limited to a few ports *const backscatter_ports = {* *80/tcp, 53/tcp, 53/udp, 179/tcp, 6666/tcp, 6667/tcp,* *} &redef;*	Port Agnostic **Extended feature** Relies on a new logic to infer reflection attacks and static src ports *if (|distinct_backscatter_peers[orig][orig_p]| < 2)*
Knockknock	Did not exist	Maintains list of valid services in the network Tracks failed connections to non-existing services Uses really low and dynamic thresholds **New**
clusterized	No	Yes **New**
false +ve	Plenty due to directionality problems due to content_gaps	Very few overall - still testing **Improvement**
Memory	tables and sets	use hyperloglog (opaque of cardinality) resulting in 80% less memory usage

Performance and features

- Memory mgmt
- Speed detection
- Accuracy
- dynamic thresholds
- Realtime whitelists
- FP identification

Performance: Stats.bro

```
event new_connection(c: connection)
{

        # for new connections we just want C to the darknet spaces
        # to speed up reaction time and to avoid tcp_expire_delays of 5.0 sec

        if (gather_statistics)
        {
                s_counters$event_peer = fmt ("%s", peer_description);
                s_counters$new_conn_counter += 1;
        }
```

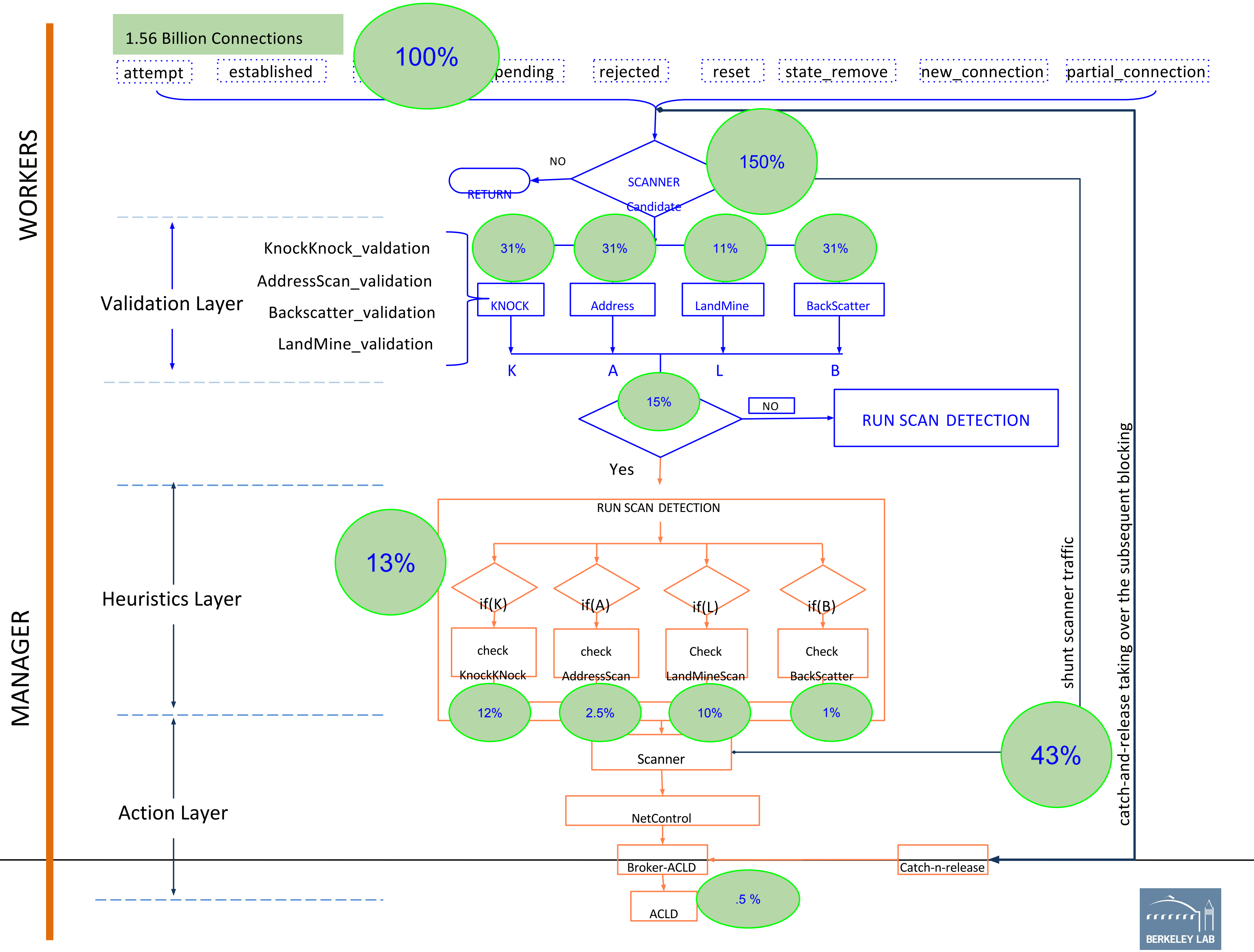
```
function is_catch_release_active(cid: conn_id): bool
{
        if (gather_statistics)
                s_counters$is_catch_release_active += 1;
```

```
function check_scan(c: connection, established: bool, reverse: bool)
{

        local orig=c$id$orig_h ;

        ### already a known_scanner
        if (orig in Scan::known_scanners && Scan::known_scanners[orig]$status)
        {
                if (gather_statistics)
                        s_counters$already_scanner_counter += 1;

                return ;
        }

        if (not_scanner(c$id))
        {
                if (gather_statistics)
                        s_counters$not_scanner += 1;
                return ;
        }
```

```
Sep  6 09:56:44 Reporter::INFO  STATISTICS: [new_conn_counter=1319180748, is_catch_release_active=2012865010, known_scanners_counter=0, not_scanner=1521025761, darknet_counter=93913909, not_darknet_counter=267966286, already_scanner_counter=370319299, filteration_
ntry=0, filteration_success=157923637, c_knock_filterate=319187619, c_knock_checkscan=142152605, c_knock_core=141512559, c_land_filterate=31860412, c_land_checkscan=32029192, c_land_core=0, c_backscat_filterate=319187619, c_backscat_checkscan=112479244, c_backscat
core=112379773, c_addressscan_filterate=319187619, c_addressscan_checkscan=129392135, c_addressscan_core=129392135, check_scan_counter=0, worker_to_manager_counter=166156888, run_scan_detection=162623097, check_scan_cache=157923637, event_peer=<uninitialized>]
anager  -
Sep  6 10:56:44 Reporter::INFO  STATISTICS: [new_conn_counter=1400859856, is_catch_release_active=2136028909, known_scanners_counter=0, not_scanner=1613624608, darknet_counter=98765405, not_darknet_counter=284805133, already_scanner_counter=392225200, filteration_
ntry=0, filteration_success=167449700, c_knock_filterate=339574570, c_knock_checkscan=143378350, c_knock_core=142738304, c_land_filterate=33506555, c_land_checkscan=32029192, c_land_core=0, c_backscat_filterate=339574570, c_backscat_checkscan=113721971, c_backscat
core=113620789, c_addressscan_filterate=339574570, c_addressscan_checkscan=130740532, c_addressscan_core=130740532, check_scan_counter=0, worker_to_manager_counter=167517758, run_scan_detection=163976115, check_scan_cache=167449700, event_peer=<uninitialized>]
anager  -
Sep  6 11:56:48 Reporter::INFO  STATISTICS: [new_conn_counter=1484818202, is_catch_release_active=2264004893, known_scanners_counter=0, not_scanner=1711407179, darknet_counter=103616901, not_darknet_counter=301121088, already_scanner_counter=414096304, filteration
entry=0, filteration_success=176718174, c_knock_filterate=359856781, c_knock_checkscan=144152905, c_knock_core=143512859, c_land_filterate=35152698, c_land_checkscan=32029192, c_land_core=0, c_backscat_filterate=359856781, c_backscat_checkscan=114498025, c_backsca
_core=114395821, c_addressscan_filterate=359856781, c_addressscan_checkscan=131615464, c_addressscan_core=131615464, check_scan_counter=0, worker_to_manager_counter=168418384, run_scan_detection=164860189, check_scan_cache=176718174, event_peer=<uninitialized>]
```

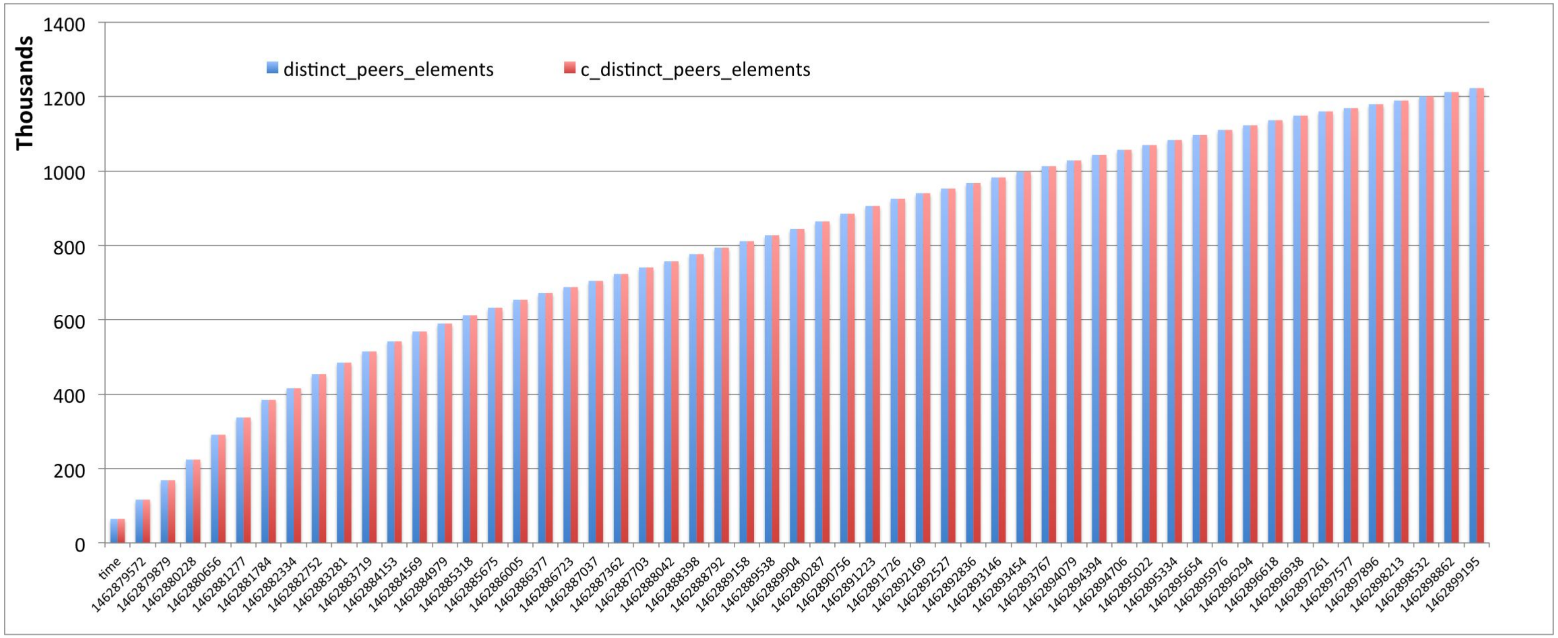
Counter Name	Counters ~ 1 day	Counters ~ 7 days
new_conn_counter	184,772,975	1,569,935,400 (100%)
is_catch_release_active	273578054 (148%)	2,382,883,254 (151.78%)
not_scanner	170877124 (92.47%)	797,378,521 (50.79%)
darknet_counter not_darknet_counter	62747298 (33.95%) 13601622 (7.36%)	103,620,129 (6.60%) 320,578,718 (20.41%)
already_scanner_counter	79308450 (42.92%)	435,007,325 (27.70%)
filter_entry filter_success	58024703 (31.40%) 27135590 (14.68%)	384,651,055 (24.5%) 185,705,196 (11.82%)
c_knock_filter c_knock_checkscan	58024703 (31.40%) 21936393 (11.87%)	384,651,055 (24.5%) 151,338,638 (9.63%)
c_land_filter c_land_checkscan	21392978 (11.57%) 19848677 (10.74%)	384,651,055 (24.5%) 32,029,192 (2.04%)
c_backscat_filter c_backscat_checkscan	58024703 (31.40%) 2005200 (1.08%)	384,651,055 (24.5%) 121,802,144 (7.75%)
c_addressscan_filter c_addressscan_checkscan	58024703 (31.40%) 4510730 (2.44%)	384,651,055 (24.5%) 139,784,051 (8.9%)
worker_to_manager_counter	27133670 (14.68%)	176,982,937 (11.27%)
run_scan_detection	24965156 (13.51%)	173,071,224 (11.02%)

BERKELEY LAB



1.56 Billion Connections
100%
attempt
established
pending
rejected
reset
state_remove
new_connection
partial_connection
WORKERS
NO
RETURN
SCANNER
Candidate
150%
Validation Layer
KnockKnock_valdation
AddressScan_validation
Backscatter_validation
LandMine_validation
31%
31%
11%
31%
KNOCK
Address
LandMine
BackScatter
K
A
L
B
15%
NO
RUN SCAN DETECTION
Yes
MANAGER
Heuristics Layer
13%
RUN SCAN DETECTION
if(K)
if(A)
if(L)
if(B)
check KnockKNock
check AddressScan
Check LandMineScan
Check BackScatter
12%
2.5%
10%
1%
Action Layer
Scanner
NetControl
Broker-ACLD
ACLD
.5 %
Catch-n-release
43%
shunt scanner traffic
catch-and-release taking over the subsequent blocking
BERKELEY LAB

Hyperloglog and state table memory

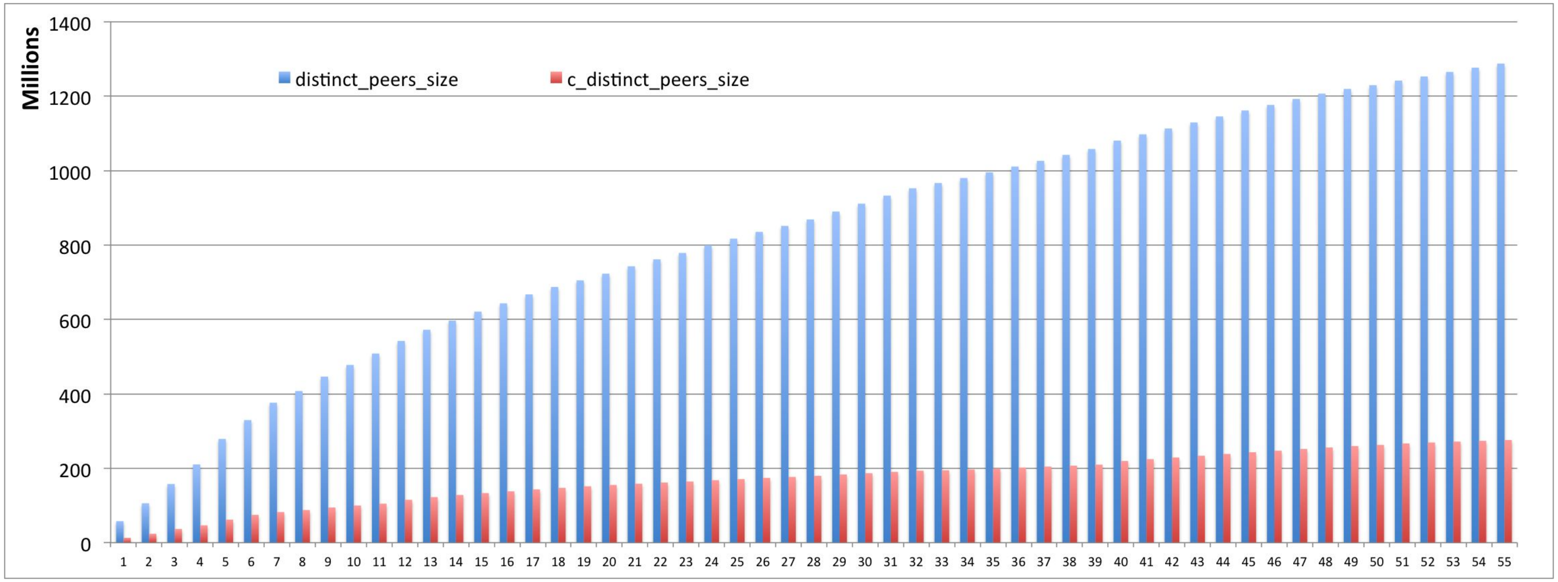



```
if (enable_big_tables)
{
        if ( orig !in distinct_peers )
                distinct_peers[orig] = set() &mergeable;

        if ( resp !in distinct_peers[orig] )
                add distinct_peers[orig][resp];

        local n = |distinct_peers[orig]|;

        local address_scan_result = check_address_scan_thresholds(orig, resp, outbound, n);
```

```
if (orig !in c_distinct_peers)
{
        local cp: opaque of cardinality = hll_cardinality_init(0.1, 0.99);
        c_distinct_peers[orig]=cp ;
}

hll_cardinality_add(c_distinct_peers[orig], resp);

 local d_val = double_to_count(hll_cardinality_estimate(c_distinct_peers[orig])) ;
```

BERKELEY LAB

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;
```

Detection Latency

1461742286.580579	45.121.9.123	KnockKnockScan	1461742074.132371	1461742074.887315	5.000498	597	256	0.754944	0.002949	
1461742407.061258	45.121.9.123	KnockKnockScan	1461742074.132371	1461742074.887315	5.000498	854	256	0.754944	0.002949	
1461742537.534465	1.174.156.155	KnockKnockScan	1461742477.573477	1461742478.552456	6.009176	2	2	0.978979	0.489489	
1461742638.132537	150.70.188.182	KnockKnockScan	1461742577.691097	1461742577.866958	5.278427	2	2	0.175861	0.08793	JP
1461742688.905063	1.174.156.155	KnockKnockScan	1461742477.573477	1461742478.552456	6.009176	5	3	0.978979	0.326326	
1461742708.918003	213.6.124.22	KnockKnockScan	1461742644.899279	1461742644.914503	5.033051	2	2	0.015224	0.007612	
1461742789.415944	150.70.188.182	KnockKnockScan	1461742577.691097	1461742577.866958	5.278427	4	2	0.175861	0.08793	JP
1461742809.472211	1.174.156.155	KnockKnockScan	1461742477.573477	1461742478.552456	6.009176	8	3	0.978979	0.326326	
1461742849.790529	213.6.124.22	KnockKnockScan	1461742644.899279	1461742644.914503	5.033051	4	2	0.015224	0.007612	
1461742880.108170	176.232.229.231	KnockKnockScan	1461742817.789336	1461742817.790107	5.021672	2	2	0.000771	0.000386	
1461742900.334705	189.166.155.92	KnockKnockScan	1461742841.646816	1461742842.460690	5.832686	2	2	0.813874	0.406937	
1461742911.402014	150.70.188.182	KnockKnockScan	1461742577.691097	1461742577.866958	5.278427	6	2	0.175861	0.08793	JP
1461742941.485399	62.248.25.6	KnockKnockScan	1461742885.790328	1461742885.889194	6.101757	2	2	0.098866	0.049433	
1461742982.103003	213.6.124.22	KnockKnockScan	1461742644.899279	1461742644.914503	5.033051	6	2	0.015224	0.007612	
1461743032.217585	176.232.229.231	KnockKnockScan	1461742817.789336	1461742817.790107	5.021672	4	2	0.000771	0.000386	
1461743042.222850	122.116.211.59	KnockKnockScan	1461742986.881469	1461742987.842975	5.962774	3	2	0.961506	0.480753	
1461743052.895237	189.166.155.92	KnockKnockScan	1461742841.646816	1461742842.460690	5.832686	4	2	0.813874	0.406937	
1461743072.906497	114.198.172.22	KnockKnockScan	1461743013.805050	1461743013.817437	5.000501	3	2	0.012387	0.006194	
1461743083.004051	114.33.233.155	KnockKnockScan	1461743023.020995	1461743023.508899	5.000528	3	2	0.487904	0.243952	
1461743104.143831	62.248.25.6	KnockKnockScan	1461742885.790328	1461742885.889194	6.101757	4	2	0.098866	0.049433	
1461743154.634689	176.232.229.231	KnockKnockScan	1461742817.789336	1461742817.790107	5.021672	6	2	0.000771	0.000386	
1461743174.644987	189.166.155.92	KnockKnockScan	1461742841.646816	1461742842.460690	5.832686	6	2	0.813874	0.406937	
1461743225.865889	114.198.172.22	KnockKnockScan	1461743013.805050	1461743013.817437	5.000501	5	2	0.012387	0.006194	
1461743225.865889	62.248.25.6	KnockKnockScan	1461742885.790328	1461742885.889194	6.101757	6	2	0.098866	0.049433	
1461743246.154756	114.33.233.155	KnockKnockScan	1461743023.020995	1461743023.508899	5.000528	5	2	0.487904	0.243952	
1461743297.142623	200.158.92.183	KnockKnockScan	1461743236.630197	1461743237.562772	6.980282	2	2	0.932575	0.466287	
1461743297.142623	75.99.152.163	KnockKnockScan	1461743232.632175	1461743233.383366	5.000261	3	2	0.751191	0.375596	

Detection Time

Avg. time between connections

Increasing detection speed

- Problem
  - all events use conn expiration timers as in the table

conn_expiration_timer	Interval	Description
tcp_SYN_timeout	5.0 secs	Check up on the result of an initial SYN after this much time.
tcp_attempt_delay	5.0 secs	Wait this long upon seeing an initial SYN before timing out the connection attempt.
tcp_close_delay	5.0 secs	Upon seeing a normal connection close, flush state after this much time.
tcp_connection_linger	5.0 secs	When checking a closed connection for further activity, consider it inactive if there (n't been any for this long. Complain if the connection is reused before this much time (elapsed.

- This basically means that all events trigger after 5.0 secs of actual activity on the wire

Solution: speed up detection

- Not changing expiration_timers : haven't studied the effect – could be drastic

```
### speed up landmine and knockknock for darknet space
event new_connection(c: connection)
{

        # we just want to supply c to check only for darknet spaces
        # to speed up reaction time and to avoid tcp_expire_delays of 5.0 sec issue

        local tp = get_port_transport_proto(c$id$resp_p);

        if (tp == tcp && c$id$orig_h !in Site::local_nets && is_darknet(c$id$resp_h) )
        {
                Scan::check_scan(c, F, F);
        }
}
```

- Leverage on "insider-information" - We know our darknet/unallocated spaces
- Use new_connection event and fast-track the connections going to darknet to scan-detection module instead of waiting for other events to kick in post timer-expirations

Faster detection

60.251.100.116	Scan::DETECT	KnockKnockScan	1464854810.526877	1464854810.544769	1464854810.544769	0.017892	3	3	0.017892	0.005964
116.252.170.147	Scan::DETECT	KnockKnockScan	1464856027.246515	1464856027.325838	1464856027.325838	0.079323	3	3	0.079323	0.026441
116.1.214.122	Scan::DETECT	KnockKnockScan	1464856027.077345	1464856027.325838	1464856027.325838	0.248493	3	3	0.248493	0.082831
123.56.132.202	Scan::DETECT	KnockKnockScan	1464858493.742455	1464858494.130484	1464858494.130484	0.388029	3	3	0.388029	0.129343
120.33.204.254	Scan::DETECT	KnockKnockScan	1464858509.865612	1464858510.295089	1464858510.295089	0.429477	3	3	0.429477	0.143159
218.7.204.149	Scan::DETECT	KnockKnockScan	1464859892.686279	1464859892.767807	1464859892.767807	0.081528	3	3	0.081528	0.027176
101.201.243.213	Scan::DETECT	BackscatterSeen	1464859912.388075	1464859912.866777	1464859912.866777	0.478702	10	10	0.478702	0.04787 CN
106.91.201.133	Scan::DETECT	KnockKnockScan	1464861111.009664	1464861111.347880	1464861111.347880	0.338216	3	3	0.338216	0.112739
171.88.164.182	Scan::DETECT	KnockKnockScan	1464861968.598142	1464861969.148787	1464861969.148787	0.550645	3	3	0.550645	0.183548
95.9.138.198	Scan::DETECT	KnockKnockScan	1464862390.287287	1464862390.868363	1464862390.868363	0.581076	15	15	0.581076	0.038738
49.68.65.177	Scan::DETECT	KnockKnockScan	1464863185.554896	1464863185.694704	1464863185.694704	0.139808	3	3	0.139808	0.046603

Detection Time

Avg. time between connections

ts scanner state detection start_ts detect_ts detect_latency total_conn total_hosts_scanned duration scan_rate

1464900975.699798 203.193.173.41 Scan::DETECT KnockKnockScan 1464900842.468613 1464900975.699798 1464900975.699798 133.231185 3 3 133.231185 44.410395 IN - 20.0 77.0 0.0 manager

CONN.LOG

1464900842.468613	CUXzwp1vSPtEpsxzE5	203.193.173.41	2631	128.3.86.149	25	tcp	S0	F	T	0	S	1	48	0	0	worker-2
1464900933.227438	CCuvnu2NMu6MXe70i8	203.193.173.41	3242	128.3.5.165	25	tcp	S0	F	T	0	S	1	48	0	0	worker-15
1464900970.708817	CcWrUM1SaCoDPquwo5	203.193.173.41	3757	131.243.46.136	25	tcp	S0	F	T	0	S	1	48	0	0	worker-7

Whitelist Mgmt

- IP and Subnet based whitelist
- Clusterized
- Self-cleaning
  - when IP or subnet is added to the whitelist bro purges it from the scan tables *and*
  - removes the nullzero blocks using netcontrol/acld
- Saves restarts
  - saves problem of many IPs from a subnet being blocked and we removed only one (facebook example)

Whitelist in action

1473416025.833145 Scan::KnockKnockScan **108.61.123.72 scanned a total of 12 hosts**: [80/tcp] (port-flux-density: 6) (origin: FR distance: 5528.29 miles) - 108.61.123.72 -- manager Notice::ACTION_DROP,Notice::ACTION_LOG 60.000000 F -

Block is removed due to catch-n-release timer expiration kicking in ...

1473419748.634896 **Scan::WebCrawler** **108.61.123.72 crawler is seen**: yacybot (/global; amd64 FreeBSD 10.3-RELEASE-p7; java 1.8.0_92; GMT/en) http://yacy.net/bot.html - 108.61.123.72 worker-11 Notice::ACTION_LOG

1473419748.634896 **Scan::PurgeOnWhitelist 108.61.123.72 is removed from known_scanners after whitelist**: [scanner=108.61.123.72, status=T, detection=KnockKnockScan, detect_ts=1473416025.886353, event_peer=worker-11, expire=F] 108.61.123.72 worker-11 Notice::ACTION_LOG 3600.000000 F

Sep 9 03:13:45	776032	108.61.123.72	NetControl::DROP	3600.000000	36000.000000	1	-
Sep 9 03:13:45	776032	108.61.123.72	NetControl::DROPPED	3600.000000	36000.000000	1	-
Sep 9 03:15:18	776032	108.61.123.72	NetControl::INFO	3600.000000	36000.000000	1	Already blocked using catch-and-release - ignoring duplicate
Sep 9 04:13:46	776032	108.61.123.72	NetControl::UNBLOCK	3600.000000	36000.000000	1	-
Sep 9 04:15:48	780727	108.61.123.72	NetControl::SEEN_AGAIN	36000.000000	86400.000000	2	-
Sep 9 04:15:48	780727	108.61.123.72	NetControl::UNBLOCK	36000.000000	86400.000000	2	108.61.123.72 is removed from known_scanners after whitelist:

- removes from known_scanners
- removes from catch-n-release hell
- removes ACLD blocks on the router, if any

BERKELEY LAB

HotSubnets

- Often scanners can origin from the same subnet - ie identify bad neighborhoods
- Subnet-escalation advice and capabilities
    - Scan::HotSubnet 41.67.117.0/24 (10 scanners originating)

```
1473148542.091330 Scan::KnockKnockScan 41.67.117.20 scanned a total of 3 hosts: [2323/tcp] (port-flux-density: 6) (origin: EG distance: 7389.16 miles)
1473157363.407512 Scan::KnockKnockScan 41.67.117.42 scanned a total of 3 hosts: [2323/tcp] (port-flux-density: 6) (origin: EG distance: 7389.16 miles)
1473157406.837344 Scan::KnockKnockScan 41.67.117.171 scanned a total of 3 hosts: [2323/tcp] (port-flux-density: 6) (origin: EG distance: 7379.80 miles)
1473157406.837344 Scan::HotSubnet 41.67.117.0/24 has 3 scanners originating from it
1473157789.703106 Scan::KnockKnockScan 41.67.117.52 scanned a total of 3 hosts: [2323/tcp] (port-flux-density: 6) (origin: EG distance: 7387.13 miles)
1473169192.007087 Scan::KnockKnockScan 41.67.117.15 scanned a total of 3 hosts: [2323/tcp] (port-flux-density: 6) (origin: EG distance: 7389.16 miles)
1473183220.677926 Scan::KnockKnockScan 41.67.117.248 scanned a total of 3 hosts: [2323/tcp] (port-flux-density: 6) (origin: EG distance: 7389.16 miles)
1473196818.669538 Scan::KnockKnockScan 41.67.117.130 scanned a total of 3 hosts: [2323/tcp] (port-flux-density: 6) (origin: EG distance: 7392.02 miles)
1473202572.027443 Scan::KnockKnockScan 41.67.117.93 scanned a total of 3 hosts: [2323/tcp] (port-flux-density: 6) (origin: EG distance: 7389.16 miles)
1473204092.503435 Scan::KnockKnockScan 41.67.117.143 scanned a total of 3 hosts: [2323/tcp] (port-flux-density: 6) (origin: EG distance: 7389.16 miles)
1473206860.829055 Scan::KnockKnockScan 41.67.117.57 scanned a total of 3 hosts: [2323/tcp] (port-flux-density: 6) (origin: EG distance: 7393.71 miles)
1473206860.829055 Scan::HotSubnet 41.67.117.0/24 has 10 scanners originating from it
```

notice.log: Scan::HotSubnets

S.no	Uniq scanners in /24	How many such /24
1	3	55634
2	10	4141
3	25	913
4	100	53
5	200	8

```
1473750000.009613   Scan::HotSubnet 178.175.114.0/24   ( 3 scanners originating )    178.175.114.79    manager Notice::ACTION_LOG
1473750313.190990   Scan::HotSubnet 195.34.28.0/24     ( 10 scanners originating )   195.34.28.26      manager Notice::ACTION_LOG
1473750328.685138   Scan::HotSubnet 195.43.67.0/24     ( 10 scanners originating )   195.43.67.32      manager Notice::ACTION_LOG
1473750405.154414   Scan::HotSubnet 109.185.63.0/24    ( 3 scanners originating )    109.185.63.181    manager Notice::ACTION_LOG
1473750526.342934   Scan::HotSubnet 36.37.136.0/24     ( 3 scanners originating )    36.37.136.17      manager Notice::ACTION_LOG
1473750530.967447   Scan::HotSubnet 183.129.235.0/24   ( 3 scanners originating )    183.129.235.181   manager Notice::ACTION_LOG
1473750568.814431   Scan::HotSubnet 117.202.192.0/24   ( 3 scanners originating )    117.202.192.243   manager Notice::ACTION_LOG
1473750666.635887   Scan::HotSubnet 49.32.72.0/24      ( 10 scanners originating )   49.32.72.36       manager Notice::ACTION_LOG
1473750932.757024   Scan::HotSubnet 42.117.114.0/24    ( 3 scanners originating )    42.117.114.36     manager Notice::ACTION_LOG
1473751070.188362   Scan::HotSubnet 93.116.84.0/24     ( 3 scanners originating )    93.116.84.207     manager Notice::ACTION_LOG
1473751299.735644   Scan::HotSubnet 177.105.121.0/24   ( 10 scanners originating )   177.105.121.141   manager Notice::ACTION_LOG
1473751362.930388   Scan::HotSubnet 31.173.240.0/24    ( 10 scanners originating )   31.173.240.35     manager Notice::ACTION_LOG
1473751426.632496   Scan::HotSubnet 177.137.125.0/24   ( 10 scanners originating )   177.137.125.173   manager Notice::ACTION_LOG
1473751455.651451   Scan::HotSubnet 37.237.212.0/24    ( 3 scanners originating )    37.237.212.26     manager Notice::ACTION_LOG
1473751469.627464   Scan::HotSubnet 178.249.209.0/24   ( 3 scanners originating )    178.249.209.197   manager Notice::ACTION_LOG
1473751604.053208   Scan::HotSubnet 41.252.61.0/24     ( 3 scanners originating )    41.252.61.218     manager Notice::ACTION_LOG
1473751723.227480   Scan::HotSubnet 117.248.197.0/24   ( 3 scanners originating )    117.248.197.119   manager Notice::ACTION_LOG
1473751724.706433   Scan::HotSubnet 188.113.198.0/24   ( 3 scanners originating )    188.113.198.8     manager Notice::ACTION_LOG
1473751976.211508   Scan::HotSubnet 85.115.243.0/24    ( 10 scanners originating )   85.115.243.101    manager Notice::ACTION_LOG
1473752182.718434   Scan::HotSubnet 178.175.6.0/24     ( 3 scanners originating )    178.175.6.238     manager Notice::ACTION_LOG
```

HotSubnet /24 with > 200 scanners

S.no	ASN	Subnet	Owner
1	262355	177.125.216.0 /24	VESX Networks, BR
2	262355	177.125.217.0 /24	VESX Networks, BR
3	262355	177.125.218.0 /24	VESX Networks, BR
4	262355	177.125.219.0/24	VESX Networks, BR
5	50676	91.236.204.0/24	TELCOMNET , RU
6	6461	64.125.239.0/24	ZAYO-6461 - Zayo Bandwidth Inc, US
7	9808	112.5.236.0/24	CMNET-GD Guangdong Mobile Communication Co.Ltd., CN
8	42570	185.35.62.0/24	KS-ASN1 This ASN is used for Internet security research. Internet-scale port scanning activities are launched from it. Don_t hesitate to contact portscan@nagra.com would you have any question., CH

SF_to_Scanner

May 8 08:08:35 Scan::KnockKnockScan 112.74.135.36 scanned a total of 3 hosts: [21/tcp] (port-flux-density: 6) (origin: CN distance: 0.00 miles) on 128.3.28.64 128.3.20.30 128.3.28.110 112.74.135.36 manager Notice::ACTION_LOG,Notice::ACTION_DROP 3600.000000 F

May 8 08:08:35 History::SF_to_Scanner outgoing SF to scanner 112.74.135.36 112.74.135.36 Notice::ACTION_LOG

Conn.log :

May 8 03:49:46	112.74.135.36	61291	128.3.28.110	21	tcp	3.059543	0	0	S0
May 8 03:49:55	112.74.135.36	61291	128.3.28.110	21	tcp	S0			
May 8 03:49:46	128.3.28.110	3	112.74.135.36	10	icmp	9.073815	152	0	OTH
May 8 03:51:23	131.243.2.64	20	112.74.135.36	56755	tcp	0.789239	520	0	SF
May 8 03:51:26	131.243.2.64	20	112.74.135.36	57266	tcp	0.656309	0	0	SF
May 8 03:51:29	131.243.2.64	20	112.74.135.36	57735	tcp	0.672116	0	0	SF
May 8 03:51:31	131.243.2.64	20	112.74.135.36	58196	tcp	0.381356	0	0	SF
May 8 03:51:34	131.243.2.64	20	112.74.135.36	58595	tcp	0.722489	0	0	SF
May 8 03:51:37	131.243.2.64	20	112.74.135.36	59047	tcp	0.378877	0	0	SF
May 8 03:51:40	131.243.2.64	20	112.74.135.36	59431	tcp	0.543354	0	0	SF
May 8 03:51:46	131.243.2.64	20	112.74.135.36	60295	tcp	0.569139	0	0	SF
May 8 03:51:48	131.243.2.64	20	112.74.135.36	60692	tcp	0.783772	0	0	SF

Implementation

```
event connection_established(c: connection) &priority=-5
{
        local src = c$id$orig_h;
        local dst = c$id$resp_h;

        # ignore remote originating connections
        if (src !in Site::local_nets)
                return ;

        if (c$resp$state == TCP_ESTABLISHED)
        {
                add_to_bloom(dst) ;
        }

}
```

global tcp_outgoing_SF : opaque of bloomfilter ;
global tcp_conn_duration_bloom : opaque of bloomfilter ;

```
event connection_state_remove(c: connection) &priority=-5
{
        local src = c$id$orig_h;
        local dst = c$id$resp_h;

        # ignore remote originating connections
        if (src !in Site::local_nets)
                return ;


        # only worry about TCP connections
        # we deal with udp and icmp scanners differently
        if (c$conn$proto == udp || c$conn$proto == icmp )
                return ;

        if (c$duration > 60 secs)
        {
                bloomfilter_add(tcp_conn_duration_bloom, src);
        }

}
```

```
function check_conn_history(ip: addr): bool
{
        local result = F ;

        local seen = bloomfilter_lookup(History::tcp_outgoing_SF, ip);

        if (seen == 1)
        {
                NOTICE([$note=History::SF_to_Scanner, $src=ip,
                        $msg=fmt("outgoing SF to scanner %s", ip)
```

Identifying Legitimate Scanners

- Web crawlers, spiders, search engine indexers
- Yes, we'd like to be top hit on google
- Automatically identify web-crawlers and not flag them as scanners

Dynamic Thresholds

- High and medium threshold ports
- port flux density - basically a popularity function of a given port - less popular == higher threshold

Sep 5 23:13:10   Scan::KnockKnockScan   131.117.245.15 scanned a total of **12 hosts**: [4028/tcp] **(port-flux-density: 2)** (origin: IQ distance: 7482.10 miles)

Sep 5 23:13:15   Scan::KnockKnockScan   124.106.53.200 scanned a total of **5 hosts:** [4028/tcp] **(port-flux-density: 3)** (origin: PH distance: 6999.04 miles)

Sep 5 23:48:19   Scan::KnockKnockScan   125.26.23.65 scanned a total of **3 hosts:** [4028/tcp] **(port-flux-density: 6)** (origin: TH distance: 7855.57 miles)

identify-search-engines

Tap into http_request and http_header events

```
event http_request(c: connection, method: string, original_URI: string, unescaped_URI: string, version: string) &priority=3
{
        if (ok_robots in original_URI)
        {
                local orig=c$id$orig_h ;

                if (orig !in Scan::whitelist_ip_table)
                {
                        local _msg = fmt("web-spider seeking %s", original_URI) ;
                        NOTICE([$note=WebCrawler, $src=orig, $msg=fmt("%s", _msg)]);

                        event Scan::m_w_add_ip(orig, _msg);
                }
        }
}
```

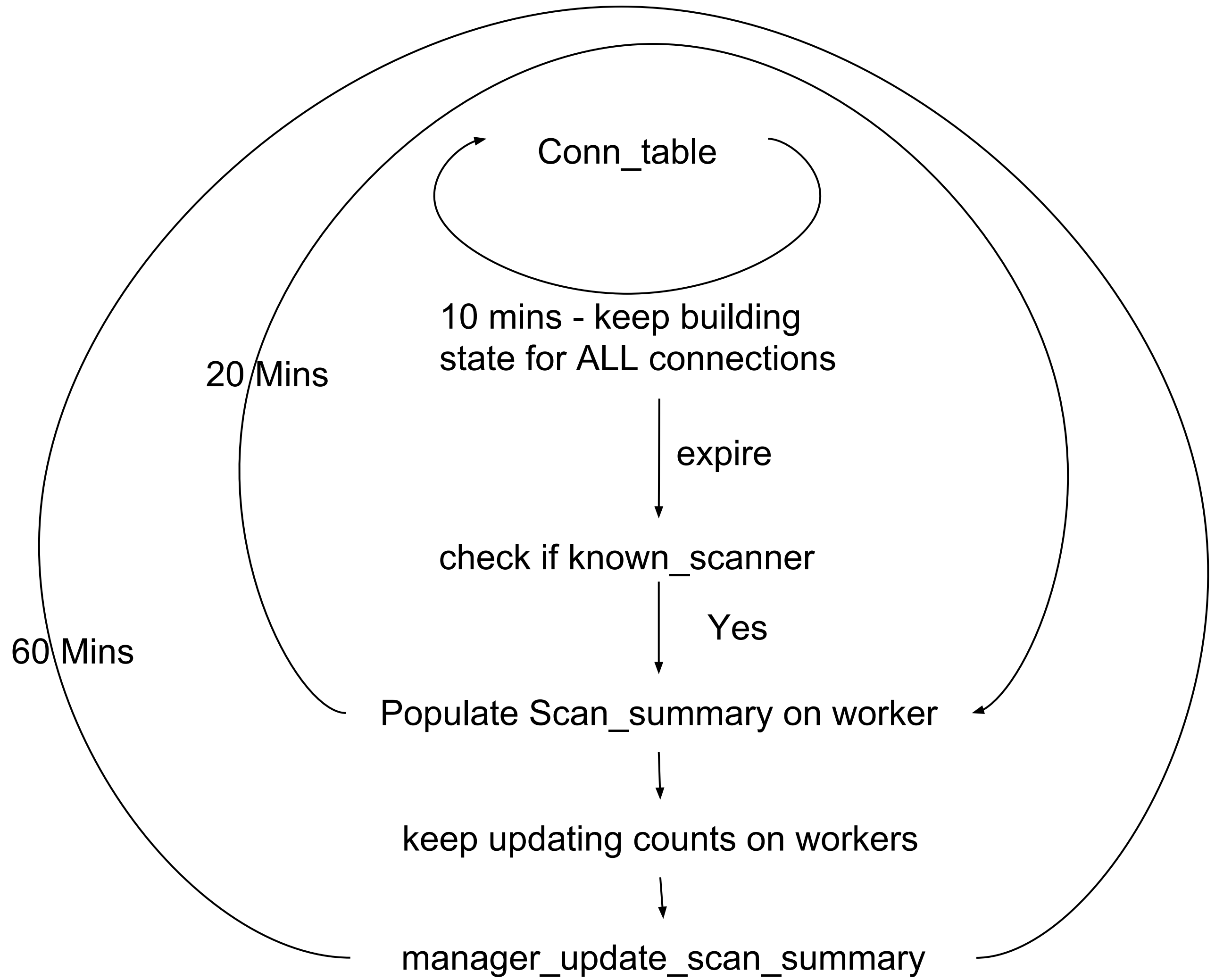
```
event http_header(c: connection, is_orig: bool, name: string, value: string) &priority=2
{
                if ( name == "USER-AGENT" && ok_web_bots  in value )
                {
                        local orig=c$id$orig_h ;
                        if (orig !in Scan::whitelist_ip_table)
                        {
                                local _msg = fmt ("%s crawler is seen: %s", orig, value);
                                NOTICE([$note=WebCrawler, $src=orig, $msg=fmt("%s", _msg)]);
                                event Scan::m_w_add_ip(orig, _msg) ;
                        }
                }
}
```

Scan-Summary

#fields ts	scanner	state	detection	start_ts	end_ts	detect_ts	detect_latency	total_conn	total_hosts_scanned	duration	scan_rate	country_code	region	city	distance
1473359945.139915	166.154.222.141	Scan::DETECT	KnockKnockScan	1473358239.225943	1473359945.139915	1473359945.139915	1705.913972	3	3	1705.913972	568.637991	US	(emp		
1473363593.103833	166.154.222.141	Scan::UPDATE	KnockKnockScan	1473358867.823400	1473363390.585065	1473359945.139915	1077.316515	23	23	4522.761665	196.641812	US	(emp		
1473367203.264704	166.154.222.141	Scan::UPDATE	KnockKnockScan	1473358867.823400	1473363390.585065	1473359945.139915	1077.316515	23	23	4522.761665	196.641812	US	(emp		
1473370810.158499	166.154.222.141	Scan::UPDATE	KnockKnockScan	1473358867.823400	1473368822.682992	1473359945.139915	1077.316515	48	44	9954.859592	226.246809	US	(emp		
1473374478.860150	166.154.222.141	Scan::UPDATE	KnockKnockScan	1473358867.823400	1473373753.229130	1473359945.139915	1077.316515	76	69	14885.405730	215.730518	US	(emp		
1473378082.058692	166.154.222.141	Scan::UPDATE	KnockKnockScan	1473358867.823400	1473376018.250225	1473359945.139915	1077.316515	86	79	17150.426825	217.09401	US	(emp		
1473381682.034921	166.154.222.141	Scan::UPDATE	KnockKnockScan	1473358867.823400	1473380475.182069	1473359945.139915	1077.316515	108	96	21607.358669	225.076653	US	(emp		
1473385286.614574	166.154.222.141	Scan::UPDATE	KnockKnockScan	1473358867.823400	1473384956.095426	1473359945.139915	1077.316515	133	117	26088.272026	222.976684	US	(emp		
1473388954.013977	166.154.222.141	Scan::UPDATE	KnockKnockScan	1473358867.823400	1473386313.906947	1473359945.139915	1077.316515	144	128	27446.083547	214.422528	US	(emp		
1473392561.887521	166.154.222.141	Scan::UPDATE	KnockKnockScan	1473358867.823400	1473390439.869610	1473359945.139915	1077.316515	172	154	31572.046210	205.013287	US	(emp		
1473396163.027289	166.154.222.141	Scan::UPDATE	KnockKnockScan	1473358867.823400	1473394830.983372	1473359945.139915	1077.316515	192	170	35963.159972	211.548	US	(empty)	(emp	

- Provides summary of
  - when scan started,
  - when it ended,
  - when was it detected
  - how many connections were made by the scanner
  - how many uniq hosts did it touch
  - latency of detection
  - total duration of the scan
- Clusterized
- Memory efficient - relies on opaque of cardinality
- Incremental scan-summary for the lifetime of the scanner

Scan-Summary Architecture



Blocking speed

bro network_time()

/usr/local/bro/logs/current/notice.log

1473400423.893674 Sc Detection Time .143.125 scanned a total of 3 hosts: [2323/tcp] (port-flux-density: 6) (origin: VN distance: 7665.18 -- manager Notice::ACTION_DROP,Notice::ACTION_LOG 60.000000 F -

/usr/local/bro/logs/broker-acld-logs/broker.log

1473400424.110827:brokerlisten:INFO:Got event NetControl::acld_add_rule. id=0, arule: {'comment': ' no=Scan::KnockKnockScan msg=1. Broker: Netcontrol acld_add_rule] (port-flux-density: 6) (origin: VN distance: 7665.18 miles) ', 'cookie': 760472L,

1473400424.111010:brokerlisten:INFO:**Sending to ACLD**: nullzero 760472 1.54.143.125/32 -, no=Scan::KnockKnockScan msg system_time() nned a total of 3 hosts: [2323/tcp] (port-flux-density: 6) (origin: VN distance: 76 **800 µs**

1473400424.111826:brokerliste Broker: **Sending to ACLD**: ::acld_rule_added id=0, arule={'comment': ' no=Scan::KnockKnockScan msg=1.54.143.125 scanned a total of 3 hosts: [2323/tcp] (port-flux-density: 6) (origin: VN distance: 7665.18 miles) ', 'cookie': 760472L, 'command': 'nullzero', 'arg': '1.54.143.125/32'}, msg=

/syslog/acld.log

Sep 8 22:53:44 **acld: NETS** stat Broker: **Sending to BRO**: in=1.54.143.125 replicant=yes **ats=1473400424.109729 cts=1473400424.109791** cmt= 54.143.125 scanned a total of 3 hosts: [2323/tcp] (port-flux-density: 6) (origin: VN distance: 7665.18 miles)

ACLD: Arrival timestamp

ACLD: Completion timestamp

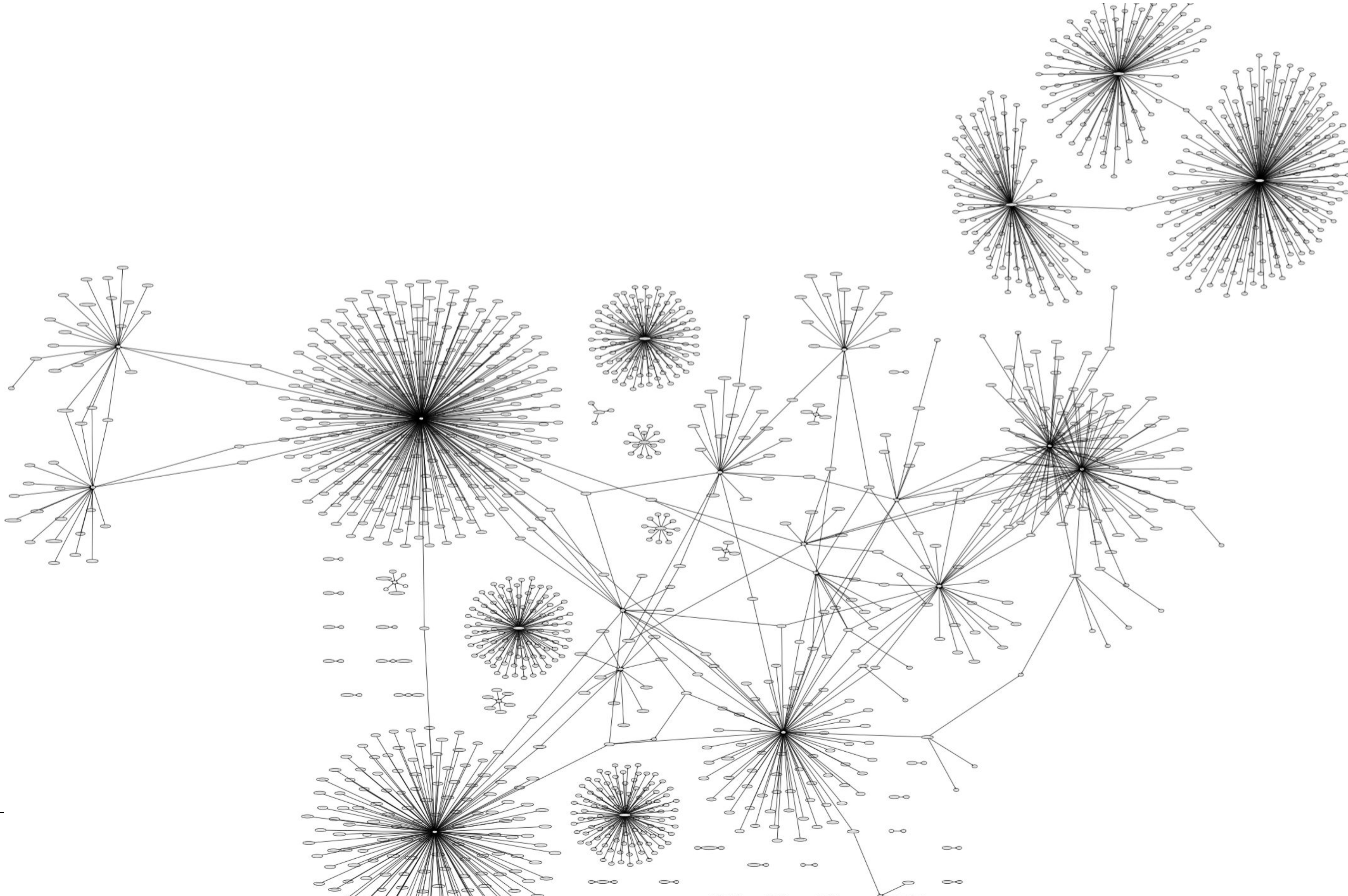
system_time()

BERKELEY LAB

drop times

timestamp	Action	Delta (t_n - t_{n-1})	Source
1473663871.195220	Scan::KnockKnockScan	t=0s	notice.log
1473663871.195220	NetControl::REQUESTED	t=0s	netcontrol.log
1473663871.226191	Brokerlisten: Got event	30.9 ms	broker.log
1473663871.226378	brokerlisten:INFO:Sending to ACLD	187 μs	broker.log
1473663871.226359	ACLD Arrival timestamp	-0.19 μs	acld.log
1473663871.226420	ACLD Complete Timestamp	61 μs	acld.log
1473663871.227030	brokerlisten:INFO:Received from ACLD	610 μs	broker.log

Table of Known Services



Usability

- Plug-n-play
- Works with netcontrol-framework
- All configuration knobs moved to one single file
- Accompanying whitelist allows for addressing false-positives in real-time
- No need to restart Bro
- Dynamic thresholds and post-detection vetting reduces false positives significantly
- GeoIP inclusion in blocking threshold heuristics

files and description

File	Description
check-scan.bro	first file which taps into events and calls function check-scan
check-scan-impl	functions which enables clusterization
scan-config	ALL user configuration settings are redef variables centrally located here. No need to go into any other policy to tweak
scan-base	important core functions – I can actually move a bunch of functions from check-scan and check-scan-impl here but will wait
scan-summary	add-on code which generates scan_summary.log (pretty good log actually)
check-*	heuristics for knockknock, landmine, addressScan, BackScatter etc. All files are basically – 2 or 3 functions – validate_* , check-* , check-thresholds (this name varies)
scan-*	additional supporting scripts for input data, whitelists, host-profiling data, subnet-info for landmine etc etc

Reliability

- What if subnets file is Empty or incomplete
  - accuracy of functions like - is_darknet or is_scanner or validation_func for heuristics
- Typose in the whitelist entry
  - catch reporter_error for all input files
- co-ordination with netcontrol
  - Any Bro shall not unblock what it did not block
- Memory and CPU on Manager

Pass Fail Criteria

- Not miss anything existing infrastructure flags
  - More accurate than existing policy
- Find more badness
- Speed
- Practical False +ve rate
- Pass peer review
- Bro runs stable for > 1 month
- Key to success is to be able to count failures correctly
- We should know what they know

users/developers/bro people

Users	Developers	Bro People
notice.log and scan-summary.log	access to known_scanners table	how to make table persistent
memory efficient	use of hyperloglog and bloom-filters	hard to find data-structure sizes/usage
whitelist capability/ Dynamic darknets and configs on fly	input-framework + tap into reporter_error event	dealing failures in input-files due to lame typos
stable code	extendable and modular	Manager CPU is mystery
plug & play	clusterization insights	ability to account of w2m and m2w events
speed & accuracy	you can fix scan.bro	Should scan-detection be in C++ land instead of policy land ?

Must and Should Requirement checklist - or feature list of scan detection

- Accuracy
- Reaction
  - Must block really fast scanners
  - Must block really slow scanners
- Smart ACL mgmt - keep scanners blocked only until active - no unnecessary acl consumption
- State management
- Block sooner if they came back (catch-n-release)
- Very very long state management (bloom filters)
- Variety - Should be able to Block based on different events (AddressScan, PortScan, deep block, vuln-signature - ntp monlist or data feeds such as tor)
- Should be able to Handle redundancy in infrastructure - ie avoid race conditions in blocking and unblocking independently
  - Atomicity in blocking and unblocking
  - Accountability in blocking and unblocking
- Whitelisting mechanism
- Outsmart attackers over attackers so that they cannot easily guess/defeat block thresholds (Dynamic thresholds)

Must and Should Requirement checklist - aka feature list of scan detection

- Ability to add new heuristics very quickly
- Identify and Remove false positives quickly and suppress them in future
  - .gov, US .edu or foreign .edu etc
- Optimize ACLs, don't block what's already blocked somewhere else
  - eg. icmp timestamp query is blocked on border router so no need to block those offending IP's any more or port 135, 137, 445 scanners
- Do not block what's blocked by the border router
- Watchdog processes to account for functionality
  - alert if too many failures on blocking
  - alert if too many success on blocking
  - alert if rate of blocking changes etc etc
- Verification capabilities
  - are blocking working as expected. Router ACLs are functional - violations of policies should alert (hey I am seeing SF on 445)
- Prioritize a list of ports/IPs/nets to be aggressively blocked
- Careful and slow in blocking a certain set
- Mechanisms to handle established connection scanners/bruteforcers (RDP, SSH)

what does it not do

- Smart Defenses against spoofing udp
- Persistence - restarts should not matter
- Dynamic responses based on situation - eg. Change from acld to nullzero on thresholds
- Expire blocks based on priorities (icmp sooner than ssh for example)
- If possible figure out intentions why this scan specifically
- Who responded and why and what they sent?
- Highlight new trends
- Big limitation - this is on[illegible]CP

Availability

https://github.com/initconf/scan-NG/

Or

use bro-pkg install initconf/scan-NG

Alternatively, try Justin Azoff's unified-scan policy which is significant improvement over stock misc/scan.bro

Questions and comments

security@lbl.gov

asharma@lbl.gov