1986–2016
ESnet
30 YEARS OF NETWORKING
AT THE SPEED OF SCIENCE

Notice Correlation and Covert-Timing Channels

Michael Dopheide & Ross Gegan
ESnet
Lawrence Berkeley National Laboratory

BroCon
Austin, TX
Sept, 13, 2016

U.S. DEPARTMENT OF ENERGY
Office of Science

BERKELEY LAB

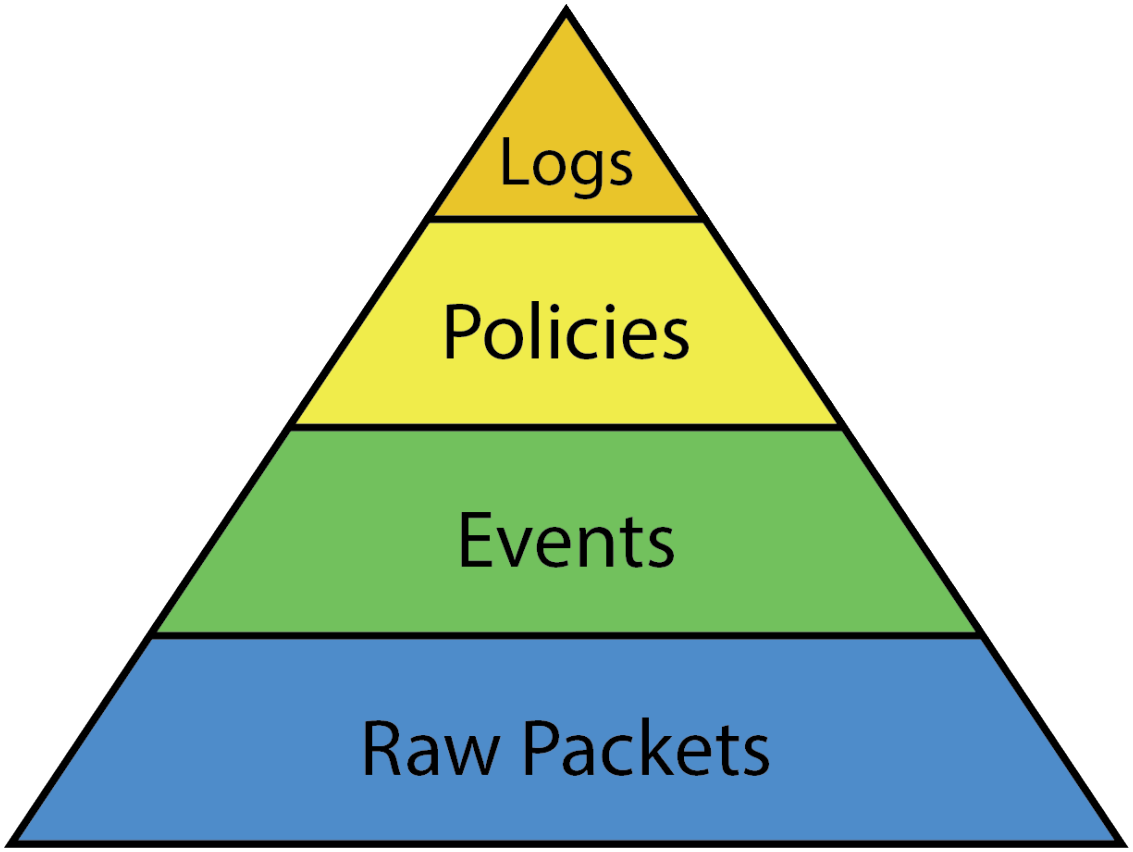
Table of Contents

Dop Introduction

- Almost 10 Years at NCSA
  - Started in systems engineering and transitioned to operational security
- 3.5 years doing penetration testing for a major bank
  - Interesting for a little while…
- Joined ESnet in February 2015.

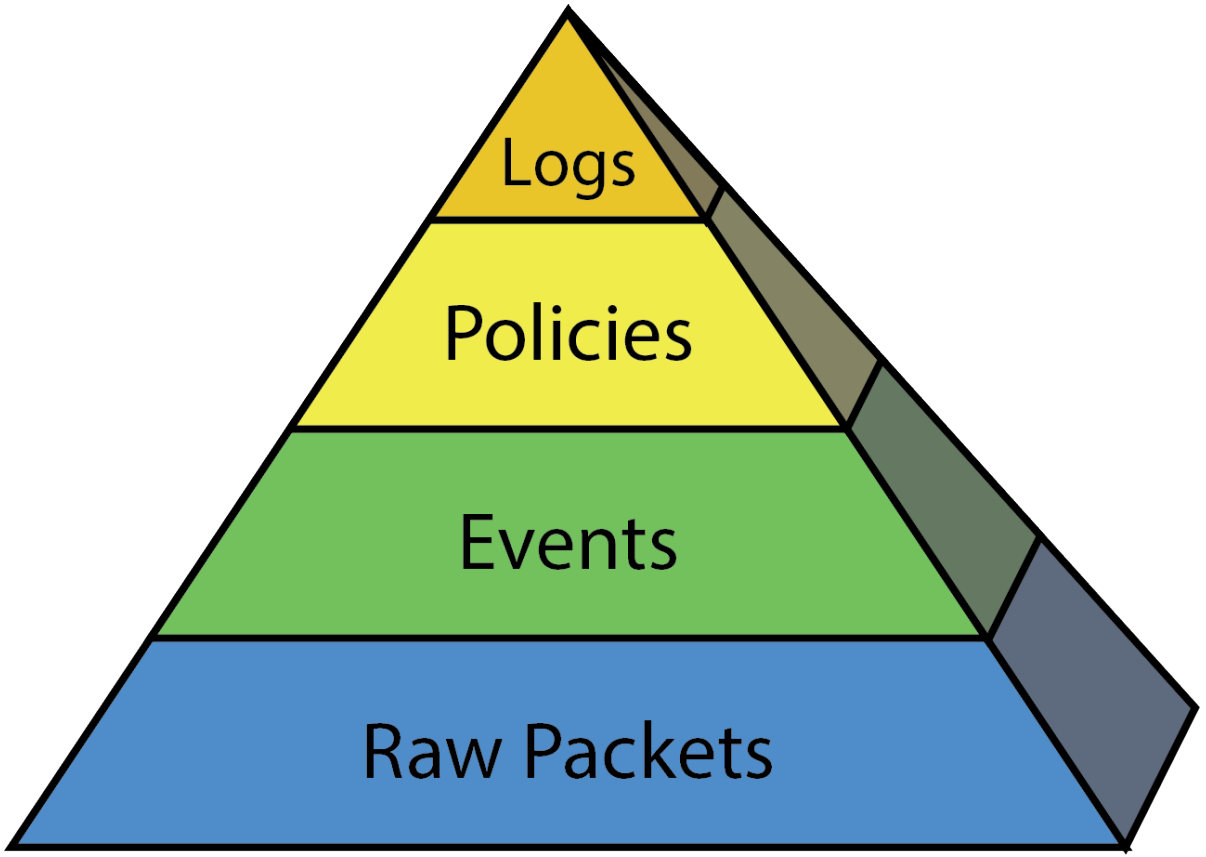
Illustration by Nick Buraglio

Pyramids*

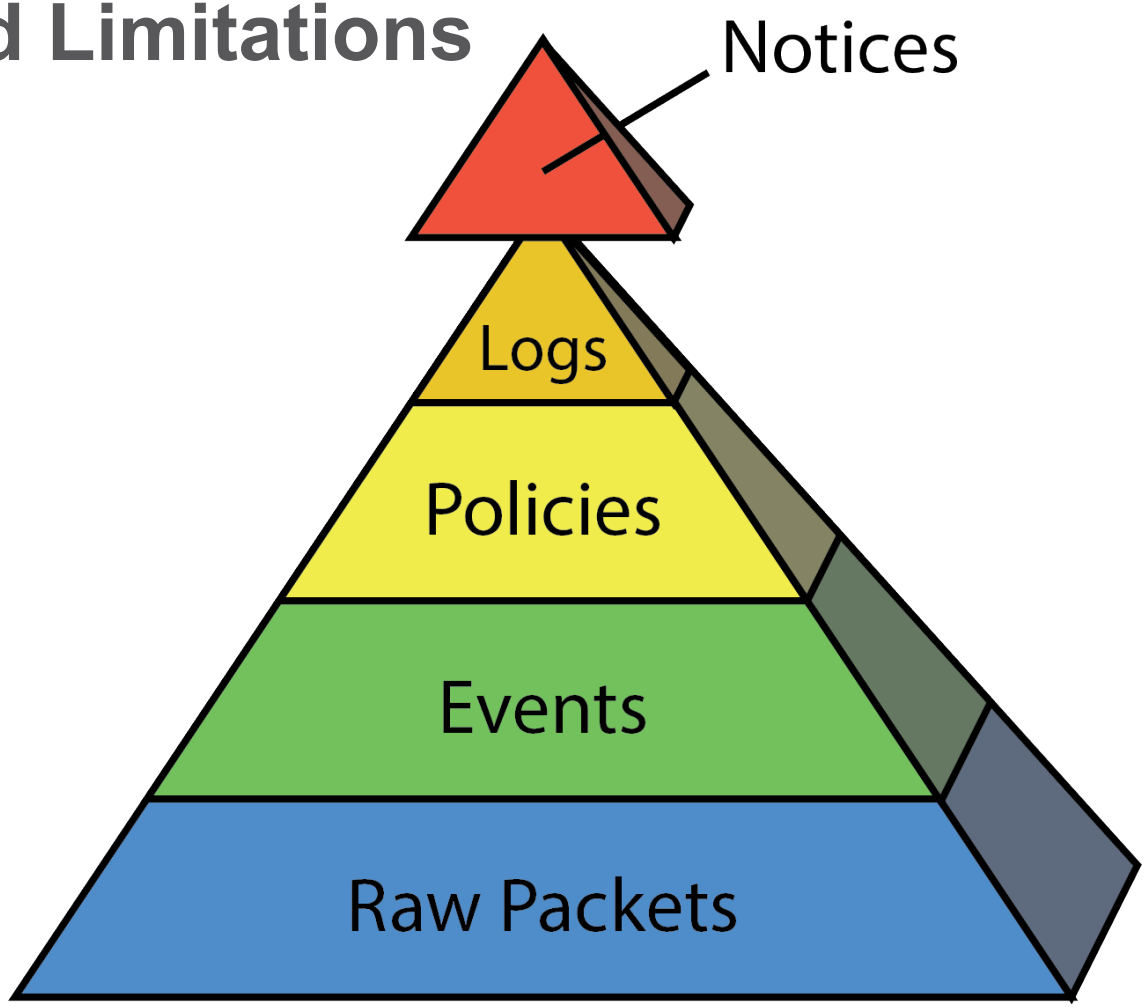


*Triangles

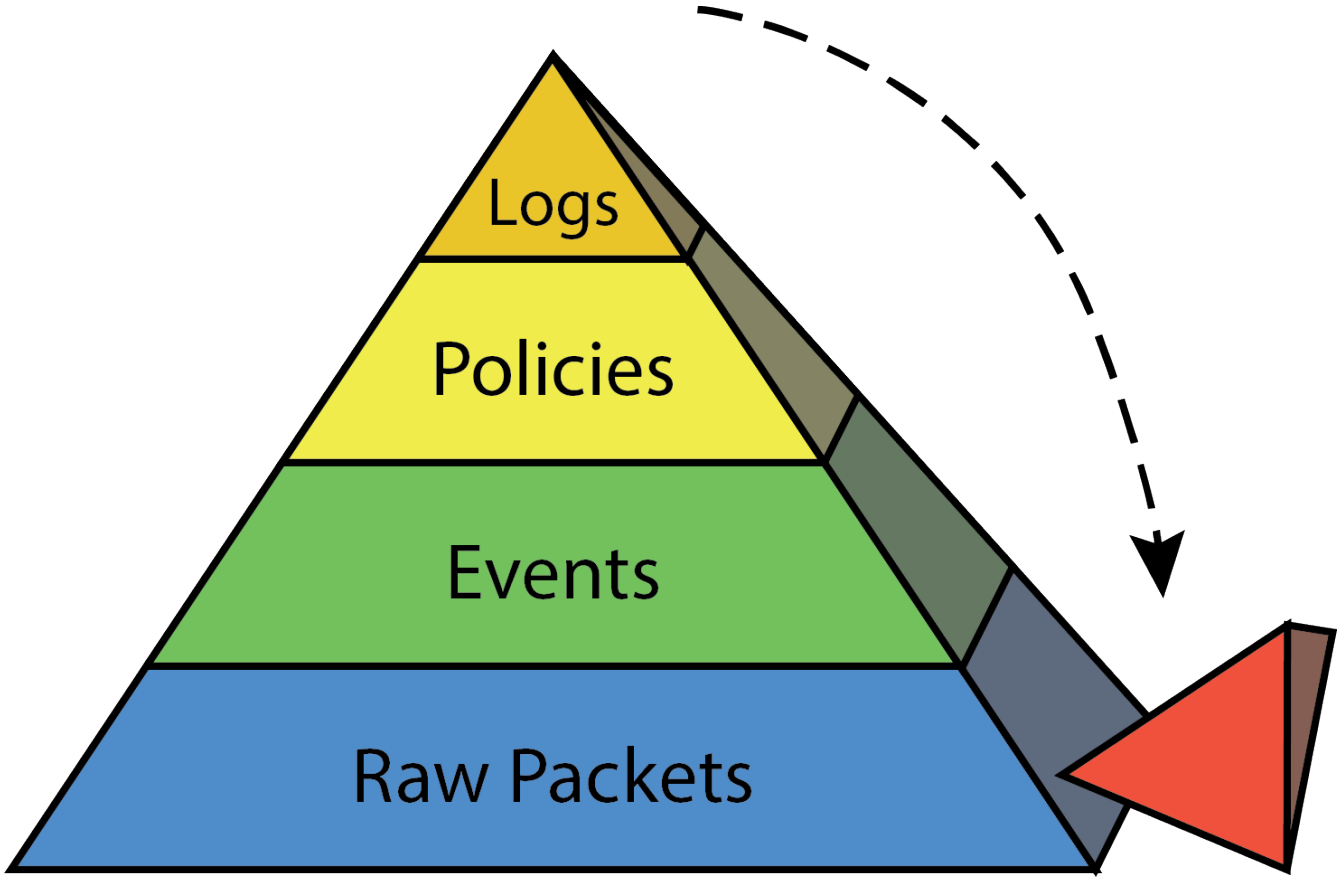
Actual Pyramids



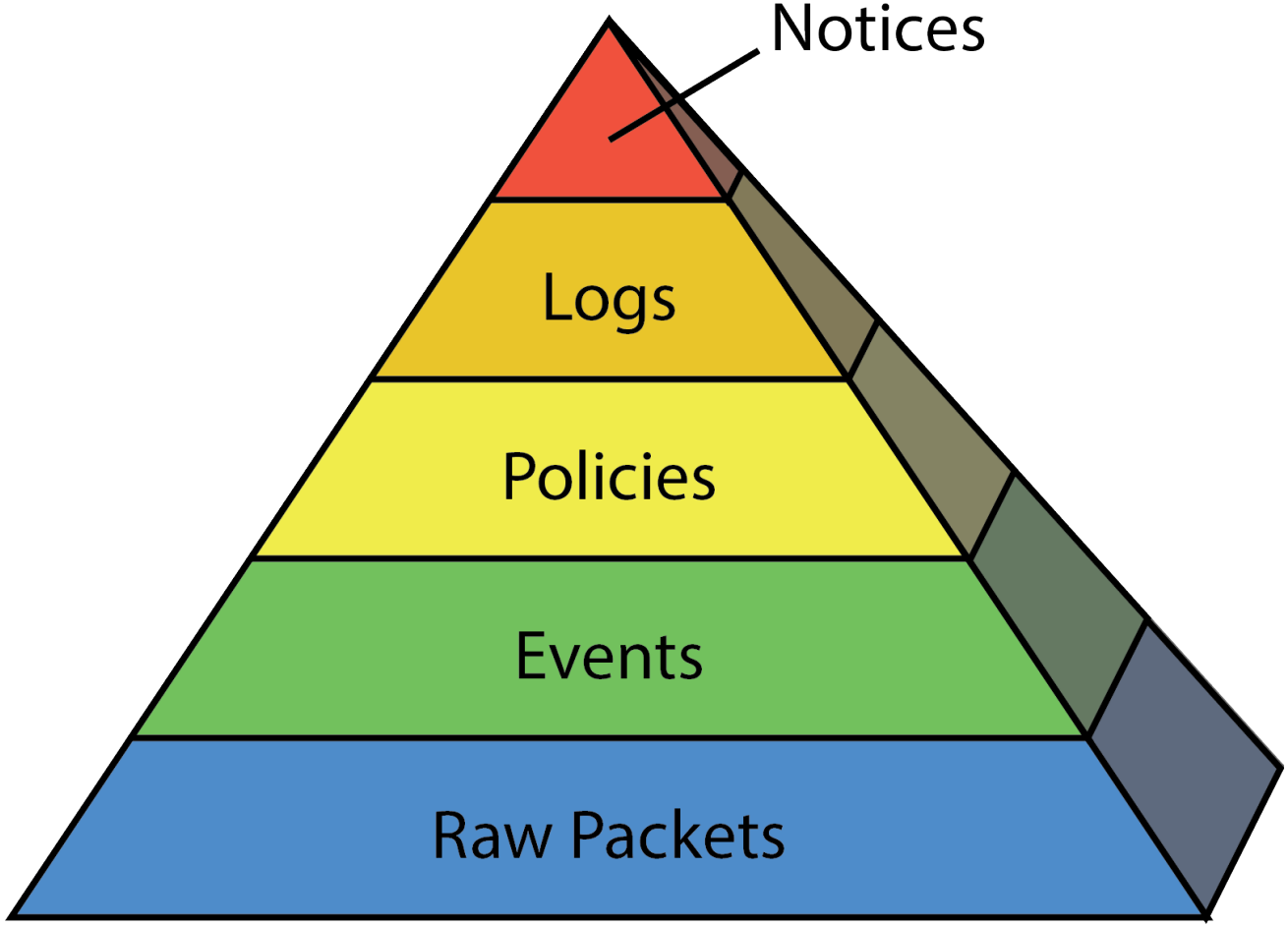
Pyramid Limitations



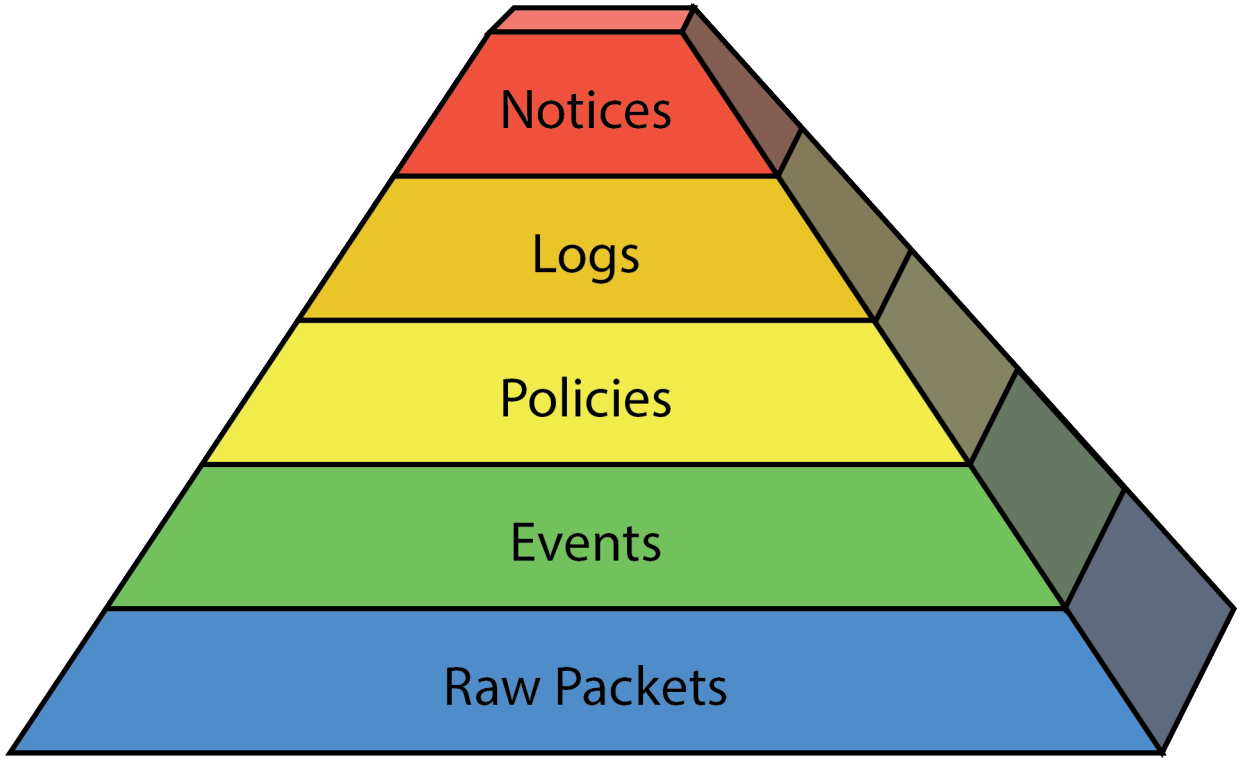
Pyramid Fail



Pyramids are hard to rebuild

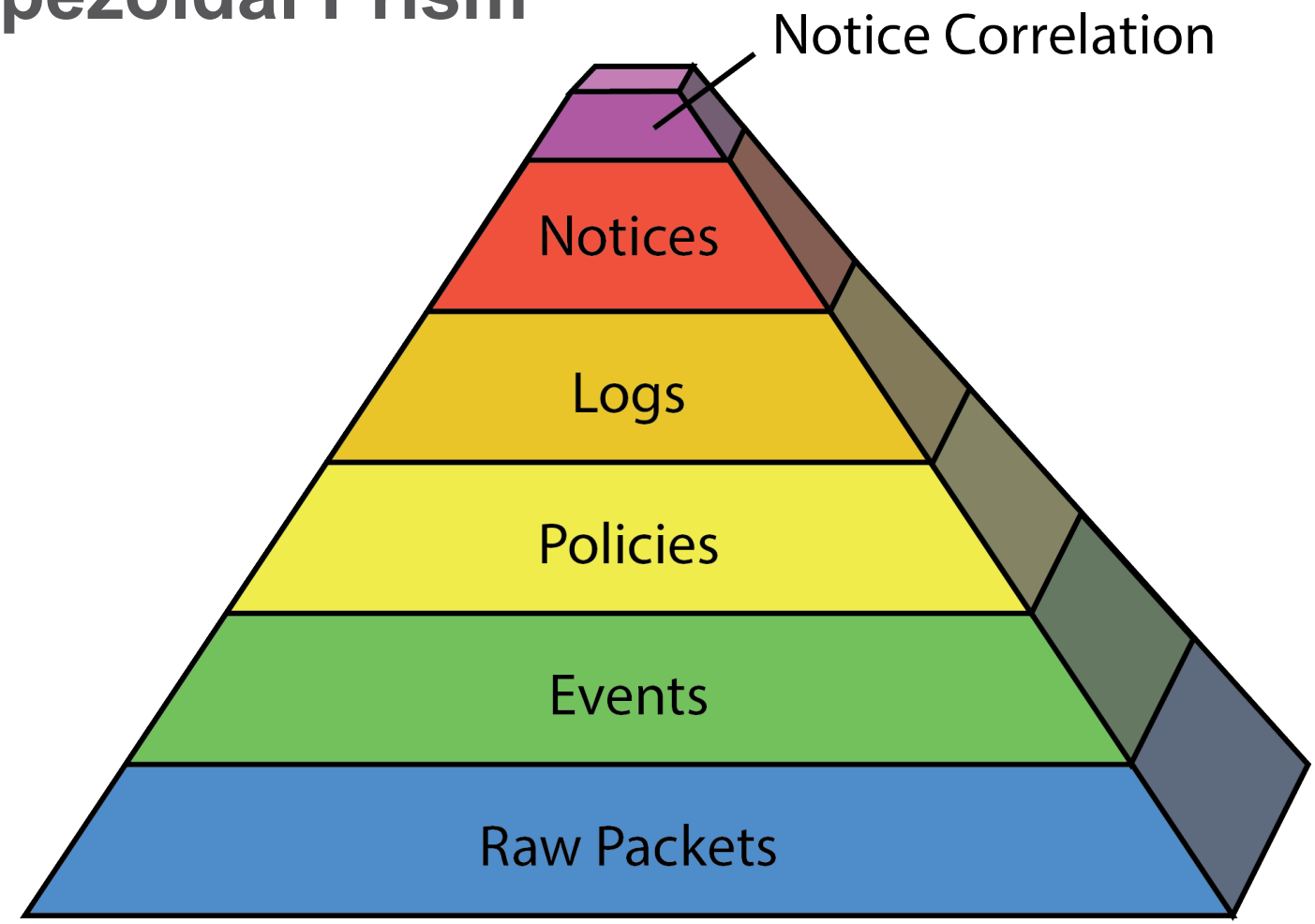


Trapezoidal Prism*

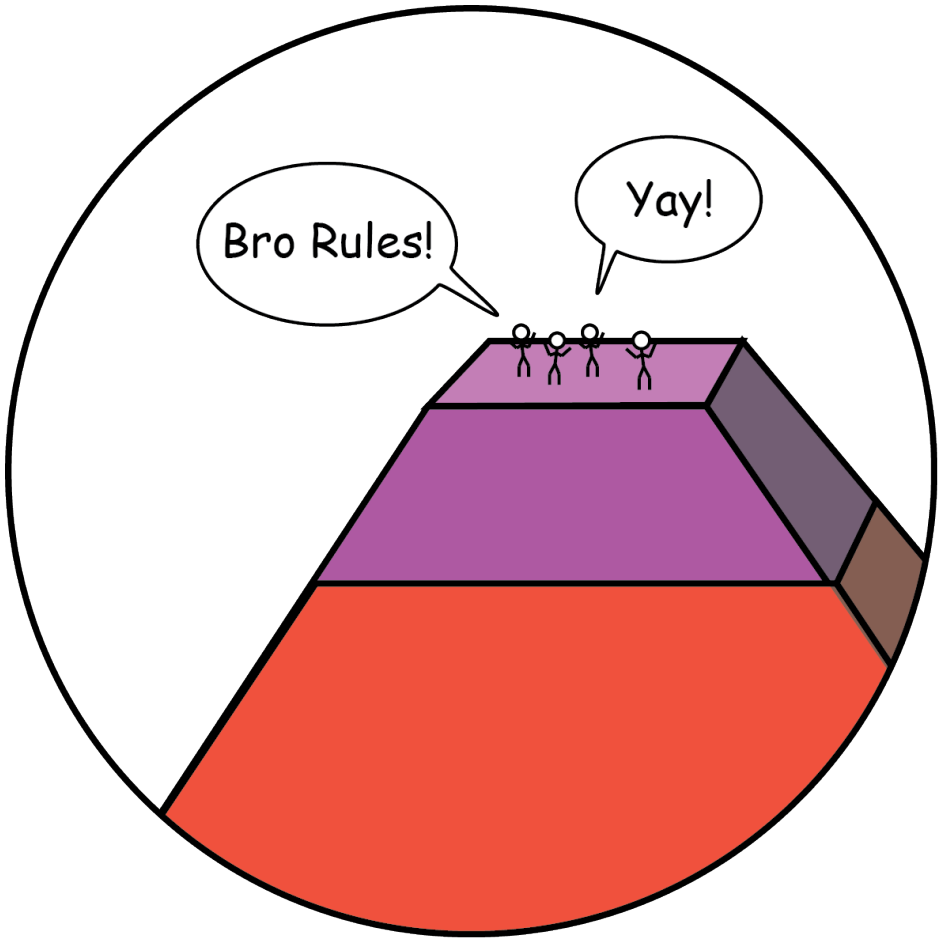


*Mayan Pyramid

Trapezoidal Prism



Added Benefits



Part 1: Multi-Notice Correlation

Given

- Bro ships with a lot of common policies
- Many more available from the community
- Policies should (must) be tuned to the specifics of your network

Problem Statement

With a new job and no knowledge of what *normal* looks like, how can I have a quick impact on ability to detect and block bad actors? How can I speed up the tuning of built-in and custom policies?

Start with Password Guessing and Intel

SSH::Password_Guessing

I couldn't simply turn on blocking because I didn't know what our user community habits were.

Intel::Notice

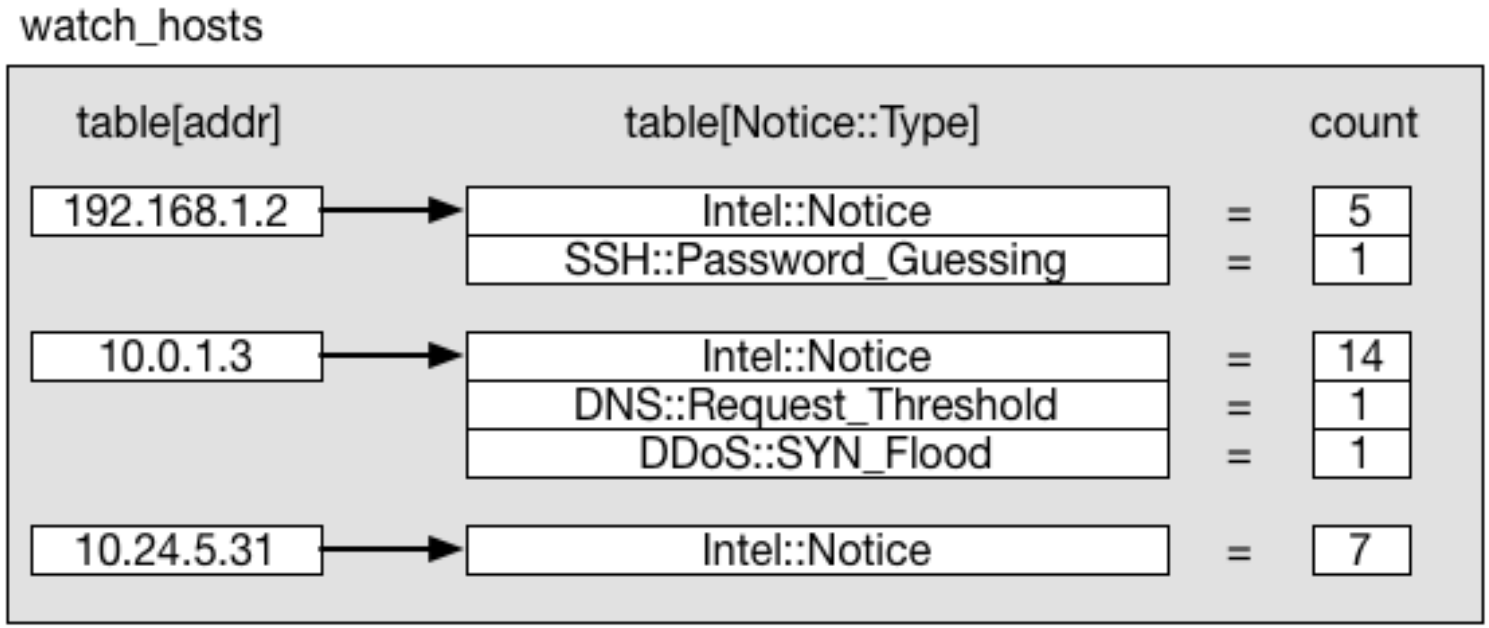
Intel feeds come with varying levels of confidence – can't block an IP just because it's in Intel.

- Using primarily both CriticalStack and REN-ISAC feeds
- ~100,000 indicators

However, if we can keep track of source IPs that flag both, that's something we can block!

The Basic Flow

```
global watch_hosts: table[addr] of table[Notice::Type] of
count &write_expire = 120 min &synchronized;
```



The Basic Flow

Define new notice types and define which types you want to block or alert on:

```
redef enum Notice::Type += {
  Multi::Multi_Notice,
  Multi::Multi_Notice_AutoBlock,
  Multi::Multi_Notice_AutoBlockAlarm,
  Multi::Single_Notice_Threshold,
  Multi::Single_Notice_Threshold_Block
};

global multi_notice_types: set[Notice::Type] = {
              SSH::Password_Guessing,
              Bash::HTTP_Header_Attack
} &redef;
```

The Basic Flow

```
hook Notice::policy(n: Notice::Info)
{
        if( n$note in multi_notice_types ){
                if(n?$conn){
                        watch_host(n$conn$id$orig_h,n);
                }else{
                        watch_host(n$src,n);
                }
        }
}

event Intel::log_intel(rec: Intel::Info){
     # any Intel hit, add to watch list.
     local wn = Notice::Info($note=Intel::Notice);
     watch_host(rec$id$orig_h,wn);
}
```

Notice Log Entry

```
1471667754.084883   -   -   -   -   -   -   -   --   Multi::Multi_Notice_AutoBlock
    Host triggered multi-notice correlation Intel::Notice:24__SSH::Password_Guessing:1
    11.22.33.44   -   -   -lbl-worker-1-4
    Notice::ACTION_LOG,BHR::ACTION_BHR,Notice::ACTION_ALARM 3600.000000   F   -
    -   -   -   -   -   -
```

`Intel::Notice:24__SSH::Password_Guessing:1`

- We saw this host via Intel 24 times and when the first password guessing notice hit we blocked it.
- The higher Intel count is just a result of the password guessing thresholds.

SSH::Password_Guessing

For sources not already blocked, in July 2016:

41 Unique IPs found SSH password guessing
12 of those in Intel

Immediate Lessons:

- Allows us to block some bad actors while getting comfortable
- Intel feeds only go so far (at least ours)
- Perhaps we can adjust our thresholds

This led to…

SSH::Foreign_Threshold_Block

Modified SSH::Password_Guessing to be more aggressive for non-U.S. sources.

July 2016:
139 Non-U.S. IPs found and auto-blocked
    60 in Intel

The reason we see many more IPs than the original 41 is because of lower thresholds.

DNS examples

- DNS::Request_Threshold
  - ESnet's DNS resolvers were getting hammered
  - Set thresholds to throw a notice
  - We can never really auto-block on just this notice as there are lots of reasons to legitimately make DNS requests at the thresholds we have set.

- DNS::Possible_Weird_CVE_2015_7547_Attack
  - Rough policy to detect DNS DoS that results in a lot of false positives.

DNS examples

When combined with Intel, DNS::Request_Threshold blocked 13 unique hosts in June 2016.

More interesting however, was the following:

```
1465974656.496484   Multi::Multi_Notice_AutoBlock
   DNS::Possible_Weird_CVE_2015_7547_Attack:1__DNS::Reque
st_Threshold:1
```

No Intel involved… this is a great example of two non-perfect policies combining to confidently block some potentially bad activity. The offending host in this case was in the Netherlands.

Notice Correlation without Intel: DDoS

ESnet was the target of some minor SYN flooding DDoS attacks. The result was two policies with different thresholds, the second allowing for more SYNs, but over a longer span of time.

```
19 : DDoS::SYN_DDoS_Attempt only
 4 : DDoS::SYN2_DDoS_Attempt only
 4 : Tripped both
```

The four that tripped both thresholds were sending SYNs so fast that they hit the higher threshold in the smaller time window.

To answer the question, “Are there hosts hitting both policies?”

- Could have done this correlation by hand
- Instead, added both notice types to multi_notice_types
  - In fact, we did this before the question was asked.

Single Notice Threshold

```
1465303777.308319   -   -   -   -   -   -   -   --
   Multi::Single_Notice_Threshold_Block    Crossed block
threshold of 10 for HTTP::HTTP_SensitiveURI   -
   80.98.206.222-   -   -  lbl-worker-1-6
   Notice::ACTION_LOG,Notice::ACTION_ALARM,BHR::ACTION_BH
R   3600.000000  F   -   -   -   -   -   -   -
```

This gives us a way to track repeat offenders before blocking.

Not as well tested feature…

- Support for correlation with Notice Types that won't block automatically
  - Unless the number of unique notice types is over the threshold, then block.

```
global multi_non_block_thres: count = 3 &redef;
global multi_notice_non_block_types: set[Notice::Type] = {
            SSH::Success
        } &redef;
```

For example:
`Will NOT Block:  Intel::Notice and SSH::Success`

But with threshold 3:
`Will Block:  Intel::Notice, SSH::Success, and DNS::Request_Threshold`

For a whitelisted scanner…

```
1469048610.980754    CQ4hxs4dNbZQnufXWe   11.22.33.44  56666
    55.66.77.88  80   -    -    -    tcp Multi::Multi_Notice_AutoBlockAlarm
    Host triggered multi-notice correlation
    DDoS::HTTP_DDoS_HEAD_Attempt:1__DDoS::HTTP_DDoS_Attempt:1__HTTP::HTT
PSensitivePOST:822__Bash::HTTP_Header_Attack:3770 11.22.33.44
    55.66.77.88  80   -    lbl-worker-1-12
    Notice::ACTION_ALARM,Notice::ACTION_LOG,BHR::ACTION_BHR
    3600.000000  F    -    -    -    -    -    -    -
```

```
DDoS::HTTP_DDoS_HEAD_Attempt:1
DDoS::HTTP_DDoS_Attempt:1
HTTP::HTTPSensitivePOST:822
Bash::HTTP_Header_Attack:3770
```

Part 1 : Notice Correlation Wrap-up

- Easy win for your new job
- Great for testing out new, not-so-perfect policies

Code:
https://github.com/dopheide/bro_notice_correlation

Blog Post:
http://blog.samoehlert.com/correlating-bro-notices

Part 2: Convert Timing Channels

1) Introduction
2) Covert Timing Channels (CTCs)
3) Detection techniques
4) Bro Policies
5) Detection Implementation
6) Conclusions and Future Work

Ross Introduction

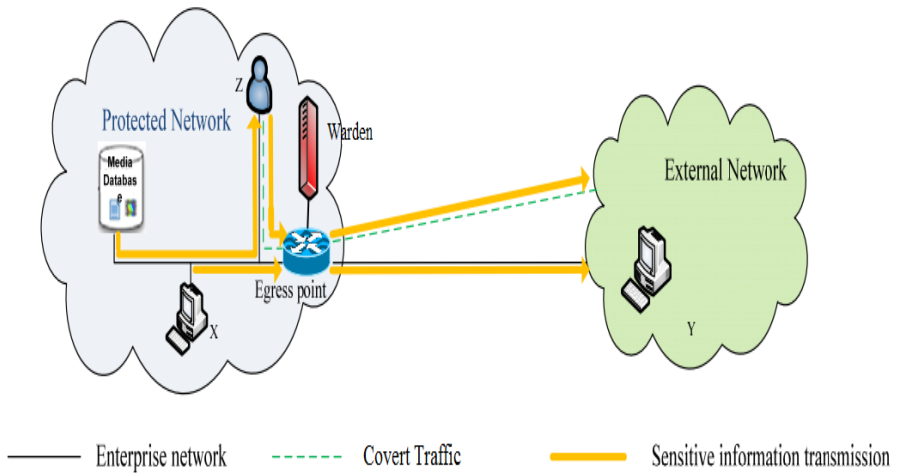
- UC Davis graduate student.
- Interning at ESnet.
  - Project: Detecting covert timing channels using Bro.

1986–2016
ESnet
30 YEARS OF NETWORKING
AT THE SPEED OF SCIENCE

What are Covert Timing Channels?

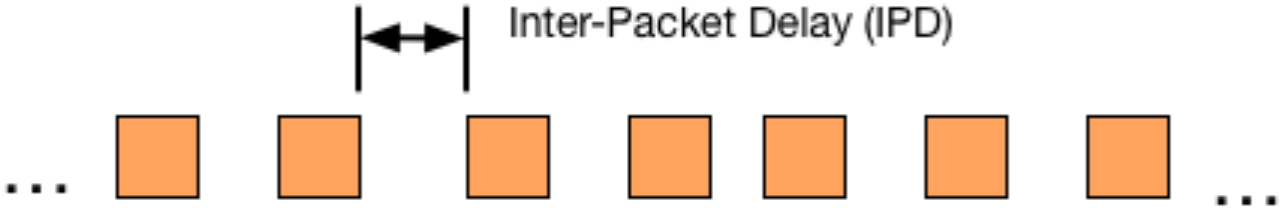
Covert Timing Channels

- Network Covert Timing Channels encode data in the inter-packet delays (IPDs)
- Allows hidden communication using authorized channels
- Can be used for malicious purposes

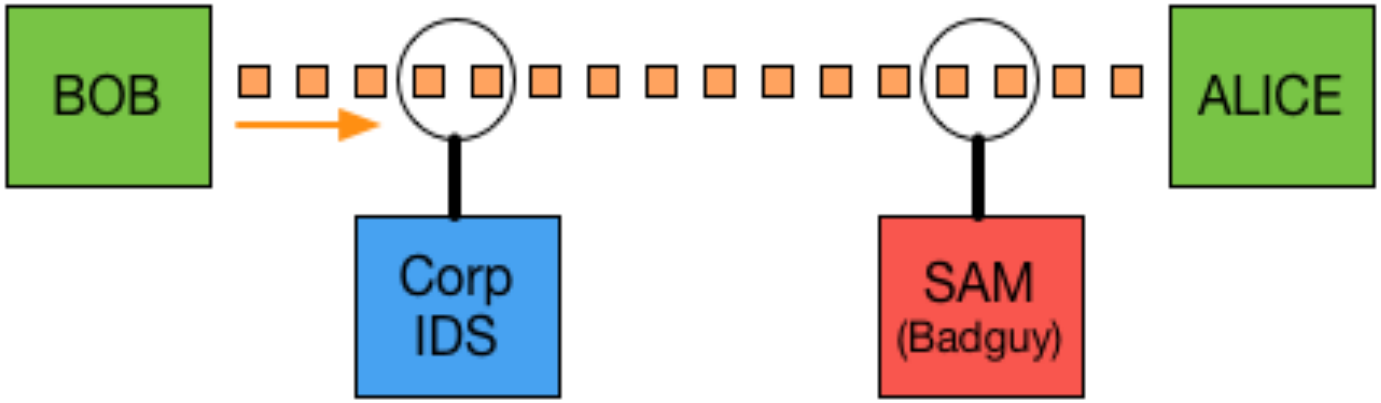


Covert Timing Channels

All traffic is going to have some randomness in the delays between each packet

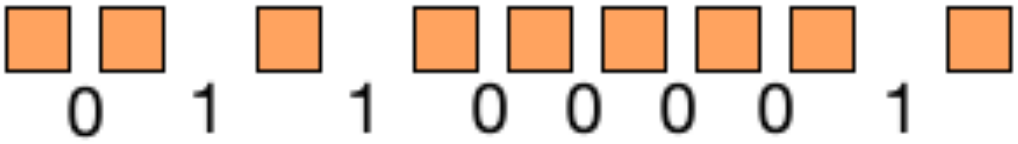


In this example, Bob is sending standard business traffic to Alice. Nothing out of the ordinary.



Covert Timing Channels

However, if Bob (or an attacker with appropriate access) is able to manipulate the IPDs beyond normal randomness….



The IPDs can be used to send data along with the normal traffic. Now an outside accomplice, anywhere on the network path, can received the covert data. The corporate IDS likely won't notice any difference.

Types of Covert Timing Channels [5]

Active Channels

- IPCTC
- Model-Based CTC (MBCTC)
- Time-Replay CTC (TRCTC)

Passive Channels

- Jitterbug

Fig. 1. IPCTC/TRCTC/MBCTC scenario

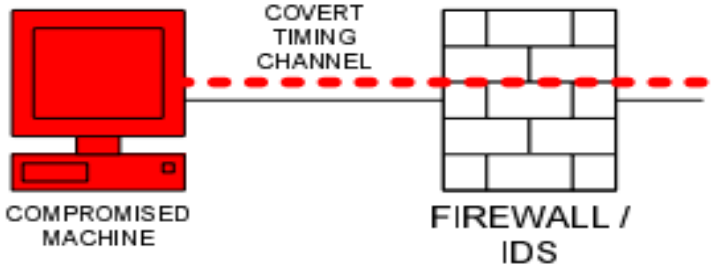


Fig. 2. JitterBug scenario

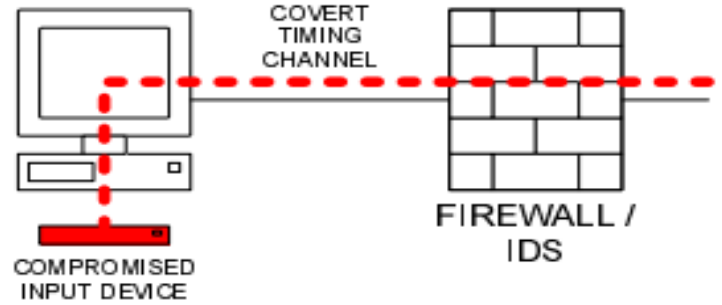


Image: S. Gianvecchio and H. Wang. [5]

1986–2016
ESnet
30 YEARS OF NETWORKING
AT THE SPEED OF SCIENCE

Covert Timing Channel Mitigation

Disrupting Covert Timing Channels [7][8]

- Goal: Eliminate the covert channel or reduce channel bandwidth.
- Add noise to a process's timing information. (Ex: fuzzy time technique)
- Can hurt legitimate traffic performance, especially for applications such as VoIP.

Image: Network Pump [8]

Detecting Covert Timing Channels

- Use Bro to identify potential CTC flows, then report and selectively disrupt.
  - Focus as much as possible on Bro to maintain portability of code with low barrier to entry for other organizations
- Monitor the incoming traffic's inter-packet delays (IPDs).
- Compare the IPD distribution with expected values for legitimate traffic.

Types of Detection Tests

- Shape Tests
  - Measures the first-order statistics
    - Ex: Shannon Entropy
- Regularity Tests
  - Measures second-order and higher statistics
    - Ex: Corrected-Conditional Entropy

Common Detection Tests [1][5]

- Shannon Entropy (EN)
- Corrected-Conditional Entropy (CCE)
- Kullback-Liebler Divergence (KLD)
- Kolmogorov-Smirnov Test (KST)

	IPCTC	TRCTC	Jitterbug	MBC TC
EN	Good	Poor	Good	Fair
CCE	Good	Good	Poor	Good
KLD	Good	Poor	Poor	Poor
KST	Good	Poor	Poor	Poor

Source: model-based detection [5]

1986–2016
ESnet
30 YEARS OF NETWORKING
AT THE SPEED OF SCIENCE

Detection Test Implementation

Training Data

- Record legitimate IPDs using Bro
- Ex: 200,000 HTTP IPDs, 75,000 SSH IPDs
- Used to create a 5-bin histogram representing the expected traffic distribution.
- IPDs are sorted, then divided into 5 equally sized group.
- Cutoff values determining bin ranges.
- Use different bins for each application for best results.

Source: model-based detection

Bro Policy Script

1. Check if a flow's size is large enough to test.
2. If so, add it to a table of flows.
3. For each new packet in those flows, get the IPD and assign a bin value between 1 and 5.
4. Once a flow has 2000 IPDs, perform the detection tests using the bin distribution.
5. Record the results in a log file.

Creating Sample CTCs

- IPCTC, TRCTC, and Jitterbug channels were created using Expect scripts to automate SSH keystrokes.
- For MBCTC, actual traffic traces injected with CTCs were used.
- Additional CTCs planned – iTRCTC, Liquid, Mimic.

Fig. 1. IPCTC/TRCTC/MBCTC scenario

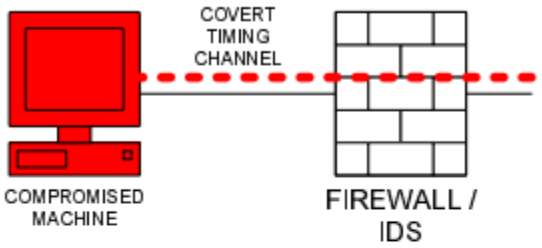
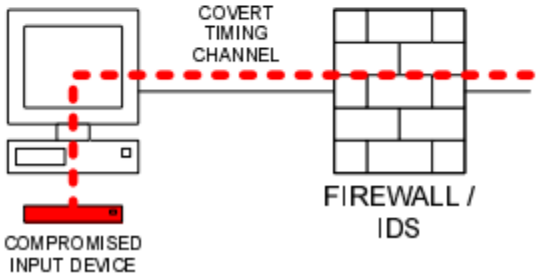


Fig. 2. JitterBug scenario



IPCTC Script

- IPCTC = Basic ON/OFF channel
- Script connects to receiver using SSH, and sends a sequence of bits.
- To send 1-bit, keystroke is generated during a 100ms interval.
- To send 0-bit, no keystroke is generated during the interval.

Fig. 1. IPCTC/TRCTC/MBCTC scenario

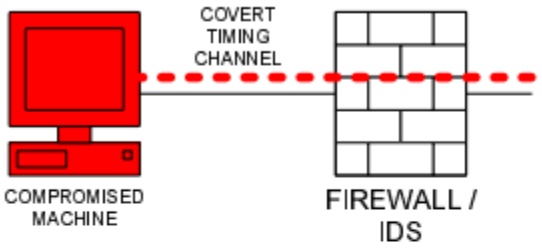
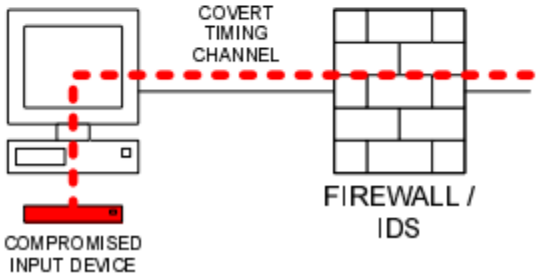


Fig. 2. JitterBug scenario



TRCTC Script

- TRCTC mimics legitimate traffic by replaying from two recorded IPD sets.
- Recorded SSH IPDs on development system using a Bro script.
- To send a 0-bit, replay from the set of IPDs **below** a given time threshold.
- To send a 1-bit, replay from the set of IPDs **above** a given time threshold.

Fig. 1. IPCTC/TRCTC/MBCTC scenario

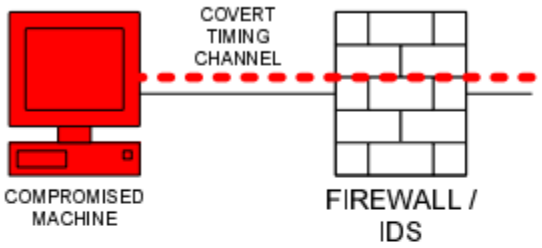
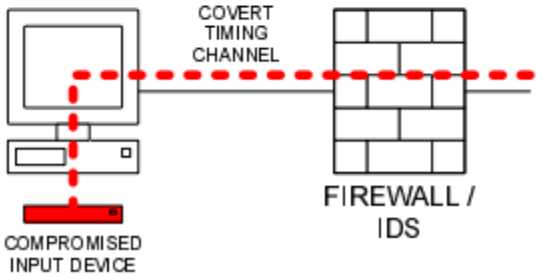


Fig. 2. JitterBug scenario



Jitterbug Script

- Jitterbug adds small delays to each keystroke to embed CTCs.
- To send a 1-bit, IPD modulo **W != 0**.
- To send a 0-bit, IPD modulo **W = 0**.
- Expect script uses **W = 20ms**, sends keystrokes with IPDs either multiples of 10 or 20.

Fig. 1. IPCTC/TRCTC/MBCTC scenario

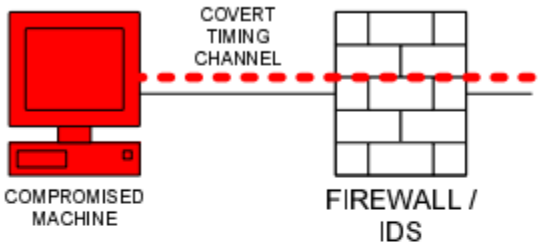
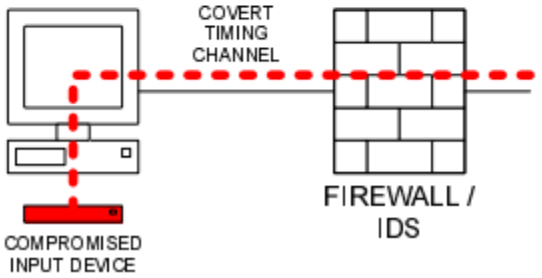


Fig. 2. JitterBug scenario



MBCTC Samples

- Traffic recorded on OC-192 link in San Jose. [12]
- 10% of large flows were replaced with MBCTCs flows.
- Channel embedded using exponential or pareto distributions.
- Replay the capture using tcpreplay. [11]

Fig. 1. IPCTC/TRCTC/MBCTC scenario

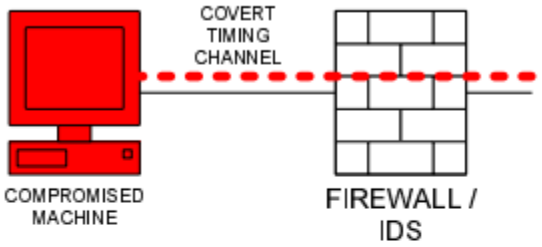
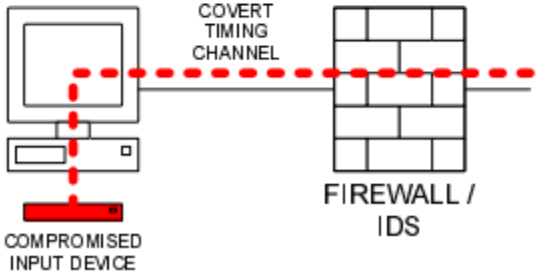


Fig. 2. JitterBug scenario



Detection Results

- CTCs were created using SSH and HTTP.
- Average entropy scores for CTC flows are lower than legitimate flows.
- Larger distances from expected distributions => Larger KST and KLD scores.

Average Scores	EN	CCE	KST	KLD
Legit	0.42	0.37	0.33	0.49
IPCTC	**0.071**	**0.062**	**0.77**	**1.45**
TRCTC	0.40	**0.13**	0.41	0.69
Jitterbug	**0.18**	**0.17**	0.66	1.20

Table: SSH channel scores, bold indicates successful detection

Performance

- CCE test is most reliable, but also most expensive.
- Worker packet loss more than doubles using CCE test.
- No significant increase in packet loss without CCE test.

Conclusion

- CTC detection can be efficiently implemented in Bro.
- Detection tests performed mostly as expected.
- Different thresholds are required for different applications for best results.
- CCE Test is most effective, but increases packet loss.

Future Work

- More detection tests.
- More types of CTCs.
- Reduce packet loss rate.
- Improve scaling by copying or ‘shunting’ long flows to a designated Bro box to collect needed IPDs
- Combine with GPU to perform expensive tests (CCE).

- We need your help!
  - ESnet doesn’t have much to covertly transmit.

References

1. R. Archibald and D. Ghosal. A comparative analysis of detection metrics for covert timing channels. Computers & Security, 2014.

2. S. Cabuk. Network covert channels: Design, analysis, detection, and elimination. Ph.D. dissertation, Purdue University, West Lafayette, IN., USA, 2006.

3. S. Cabuk, C. Brodley, and C. Shields. Ip covert timing channels: Design and detection. Proceedings of the 2004 ACM Conference on Computer and Communications Security, 2004.

4. M. Caetano, P. Vieira, J. Bordim, and P. Barreto. International journal of computer science and network security. International Journal of Computer Science and Network Security, vol. 10, pp. 13-20, August, 2010

5. S. Gianvecchio and H. Wang. An entropy-based approach to detecting covert channels. In IEEE Transactions on Dependable and Secure Computing, Vol. 8, No. 6, 2011.

6. M. Guinther. Multi-core network acceleration: Breakthrough networking performance with optimized software for multi-core processors. Wind River Systems, Inc., 2010. 12 Anonymous.

References

7. W.-M. Hu. Reducing timing channels with fuzzy time. Journal of Computer Security 1.3, 1992: 233-254, 1992.

8. M. H. Kang, I. Moskowitz, and S. Chincheck. The pump: A decade of covert fun. Computer Security Applications Conference, 21st Annual, IEEE, 2005.

9. K. Kothari. Mimic: An active covert channel that evades regularity-based detection. Comput. Netw., vol. 57, no. 3, 2013.

10. K. S. Lee, H. Wang, and H. Weatherspoon. Phy covert channel: Can you see the idles. USENIX symposium on Networked Systems Design and Implementation (NSDI), April, 2014.

11. tcpreplay developers. tcpreplay website. http: // tcpreplay. synfin. net/ trac/ wiki/ tcpreplay. , 2014.

12. CAIDA. CAIDA Data. http: // www. caida. org/ data/ overview/ , 2014