

# Bro @ KIT

Jan Grashöfer, Christian Titze, Matthias Grundmann

DECENTRALIZED SYSTEMS AND NETWORK SERVICES RESEARCH GROUP (DSN)  
INSTITUTE OF TELEMATICS, FACULTY OF INFORMATICS

```
[ 65%] Building CXX object src/analyzer/protocol/teredo/CMakeFiles/plugin-Bro-Teredo.dir/
[ 65%] Linking CXX static library libplugin-Bro-Teredo.a
make[3]: Leaving directory '/home/jgras/devel/bro/build'
[ 65%] Built target plugin-Bro-Teredo
make[3]: Entering directory '/home/jgras/devel/bro/build'
Scanning dependencies of target plugin-Bro-UDP
make[3]: Leaving directory '/home/jgras/devel/bro/build'
make[3]: Entering directory '/home/jgras/devel/bro/build'
[ 66%] Building CXX object src/analyzer/protocol/udp/CMakeFiles/plugin-Bro-UDP.dir/UDP
[ 66%] Building CXX object src/analyzer/protocol/udp/CMakeFiles/plugin-Bro-UDP.dir/Plu
[ 66%] Building CXX object src/analyzer/protocol/udp/CMakeFiles/plugin-Bro-UDP.dir/ever
[ 66%] Building CXX object src/analyzer/protocol/udp/CMakeFiles/plugin-Bro-UDP.dir/ever
[ 66%] Linking CXX static library libplugin-Bro-UDP.a
make[3]: Leaving directory '/home/jgras/devel/bro/build'
[ 66%] Built target plugin-Bro-UDP
make[3]: Entering directory '/home/jgras/devel/bro/build'
Scanning dependencies of target plugin-Bro-XMPP
make[3]: Leaving directory '/home/jgras/devel/bro/build'
```

```
36 hook extend_match(info: Info, s: Seen, items: set[Item])
37 {
38     local matches = |items|;
39     for ( item in items )
40     {
41         local meta = item$meta;
42         if ( meta$expire > 0 sec &&
43             meta$last_match + meta$expire < network_time() &&
44             ! hook single_item_expired(item) )
45         {
46             # Item already expired
47             --matches;
48             remove(item, F);
49             next;
50         }
51     }
52
53     # Update last match
54     item$meta$last_match = network_time();
55     insert(item);
56 }
57 if ( matches < 1 )
58     break;
```

1. Improving Threat Intelligence matching  
*Jan*
2. Security-oriented Performance Analysis  
*Christian*
3. Ransomware detection in academic environments  
*Matthias*

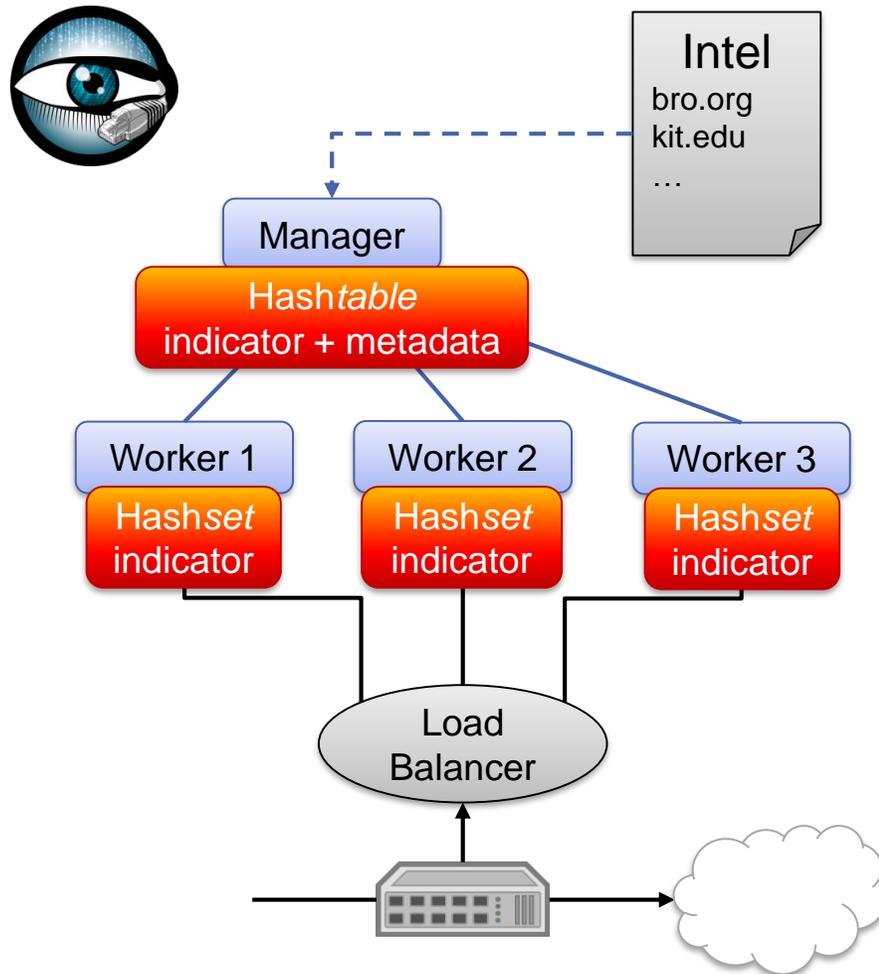
Jan Grashöfer

# **IMPROVING THREAT INTELLIGENCE MATCHING**

# The Intelligence Framework

Indicator Type	Example
ADDR	192.0.2.42, 2001:db8::23
SUBNET	192.0.2.0/24, 2001:db8::/32
URL	<del>http://</del> example.com/test/
SOFTWARE	Mozilla/5.0...
EMAIL	malicious@example.com
DOMAIN	www.example.com
USER_NAME	not used
CERT_HASH	38762cf...bb7f0a
PUBKEY_HASH	ee4aa5...0a750c
FILE_HASH	5bd9d8...39b8d1
FILE_NAME	infected.pdf

# Improve Intel Matching

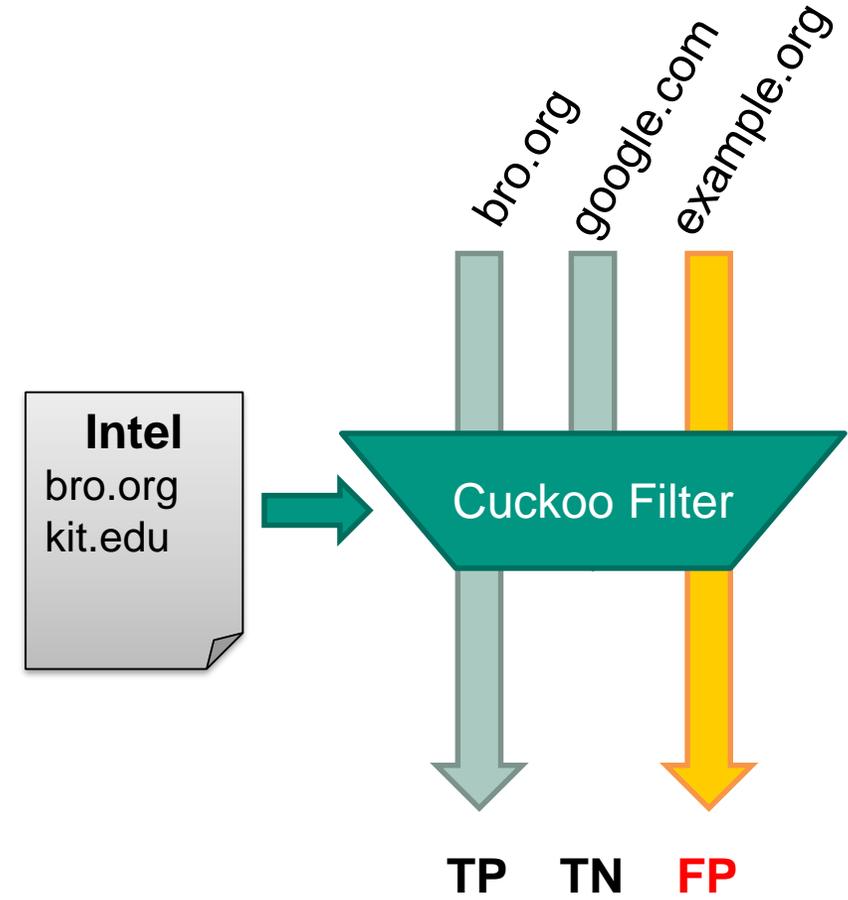
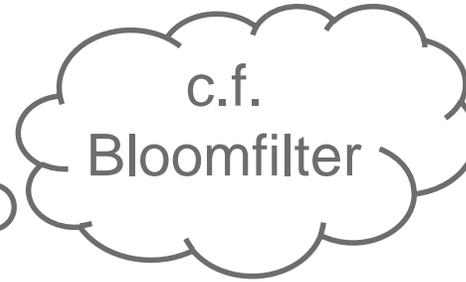


- Bro supports clustering to scale
  - Manager → coordination
  - Worker → traffic processing
- Intelligence Framework
  - Manages indicator + metadata
  - Distributes indicators for matching
- Requirements:
  - Efficient matching
  - Support removal
- Idea: Use Cuckoo Filter?

© Matthias Vallentin

# Cuckoo Filter\*

- Probabilistic data structure
  - High space efficiency
  - False Positives (FP) but **never** False Negatives!
  - FP-Probability tunable → space tradeoff
  
- Support removal



\* B. Fan, D. G. Andersen, M. Kaminsky, und M. D. Mitzenmacher, „Cuckoo Filter: Practically Better Than Bloom“, 2014, S. 75–88.

# Cuckoo Filter – Exploration

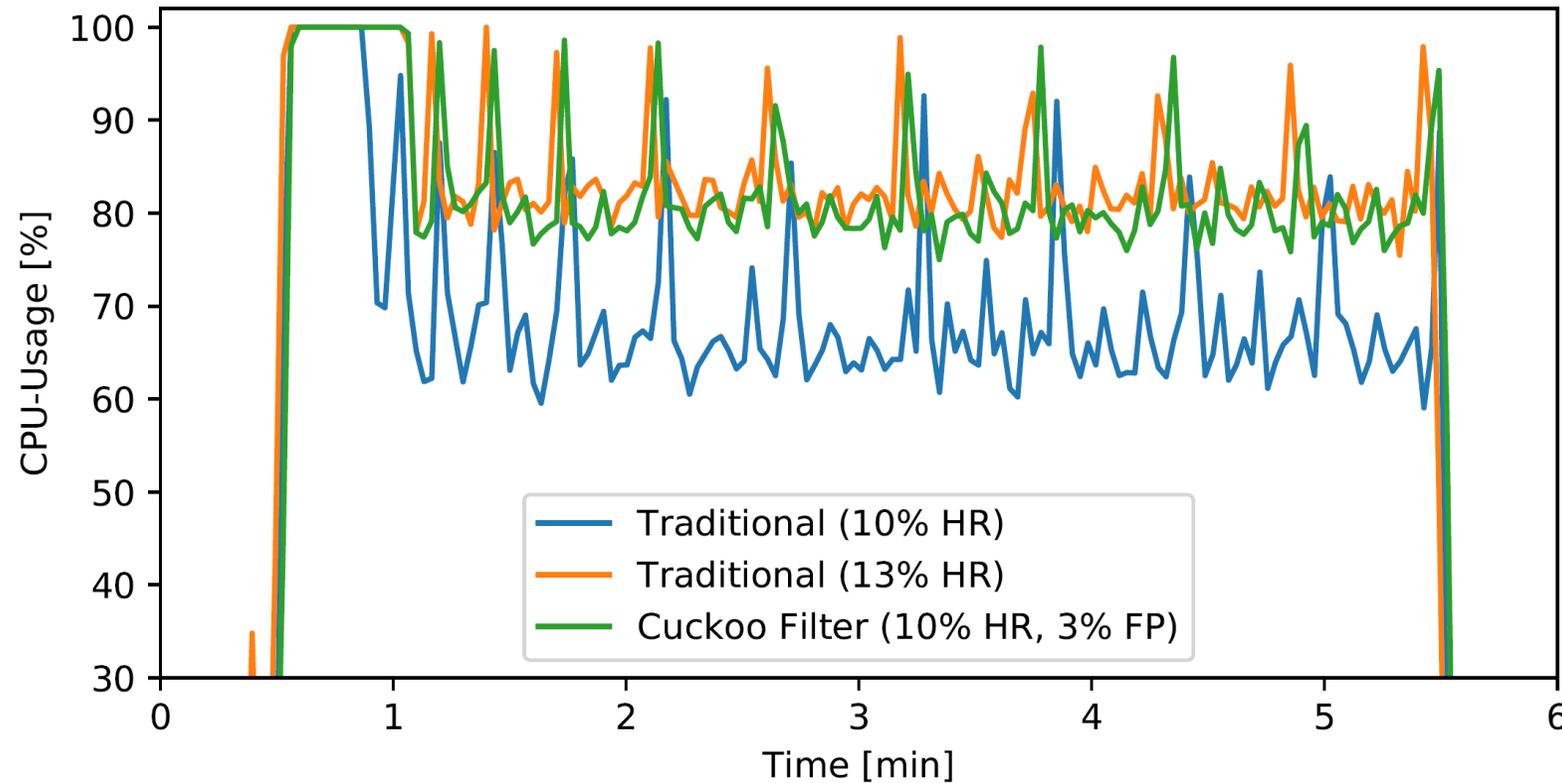
- New implementation **cuculiform** (C++)
  - allows runtime configuration → script-land interface
  - discovered implementation and configuration pitfalls
  
- Comparison of different Implementations

Implementation	Structure Size [KiByte]	FP-Rate [%]	Lookup-Time [ $\mu$ s]
Bro Hashtable	211 202.8	-	1.7589
Bro Hashset	145 666.8	-	1.9281
Reference	1 024.0	2.9794	0.9637
Rust	1 024.0	2.9789	1.0847
Cuculiform	1 024.0	2.9787	1.3746

50 workers →  
save ~10 GB

Fingerprint size 8 bit, 4 elements per bucket, 1 Mi elements capacity,  
mean of 1 000 runs (confidence intervals negligible)

# Cuckoo Filter – Integration into Bro



- CPU-Usage on worker nodes varying data structure and Hit Rate (HR)

► Overhead occurs on the worker!

# Future Work

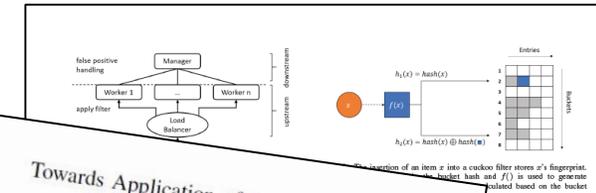
## ■ Estimate typical Intel Framework workloads

- Number of Indicators
- Connections per Hour

## ■ FP-Rate ≠ FP-Probability

- Assume google.com yields a false positive  
→ FP-Rate degrades
- Solution: Adaptive Cuckoo Filter\*

## ▶ “Real-life” benchmarks



**Towards Application of Cuckoo Filters in Network Security Monitoring**

Jan Grabhöfer, Florian Jacob, Hannes Hartenstein  
Karlsruhe Institute of Technology  
jan.grabhoefer@kit.edu, florian.jacob@student.kit.edu, hannes.hartenstein@kit.edu

**Abstract**—In this paper, we study the feasibility of applying the recently proposed cuckoo filters to improve space efficiency for set membership testing in Network Security Monitoring. We present conceptual insights for the practical application of cuckoo filters and provide a cuckoo filter implementation allowing routine configuration. To evaluate the practical applicability of cuckoo filters, we integrate our implementation into the Bro Network Security Monitor, compare it to traditional data structures and conduct a brief operational evaluation. We find that cuckoo filters allow remarkable memory savings, while potential performance trade-offs, caused by introducing false positives, have to be carefully evaluated on a case-by-case basis.

**I. INTRODUCTION**

Set membership testing is a common operation in Network Security Monitoring (NSM). Determining whether a given item is known to be of special relevance or has been seen before are prevalent use cases.

A prominent example of set membership testing is intrusion anomaly detection, intrusion detection in machine learning based anomaly detection, intrusion detection in multi-purpose network indicators of compromise (IOCs) like hashes of transferred files. The effectiveness of this approach depends on the quality of the input data. Hence, the term *Threat Intelligence* is used to emphasize that proper misuse detection requires refined, high quality data. While the effectiveness of misuse detection relies on data quality, the effectiveness of misuse detection is key for practical applications. Consequently, data structures that maximize lookup performance and minimize space costs are of particular interest in this domain.

The use of probabilistic data structures, like the well-known Bloom filter [1], is a common approach to reduce space costs and improve performance of set membership tests. This is achieved by relaxing the task of set membership testing to so called *approximate set membership testing*, allowing a bearable number of false positives. That means, the data structure might consider an element part of the set that was never added. Unfortunately, traditional probabilistic data structures do not support deletion of elements without introducing significant overhead, leaving them impractical for many real-world scenarios. In 2014, Fan et al. introduced cuckoo filters [2], which promise to overcome these shortcomings.

Given the huge and still growing scales of today's computer networks, the efficient matching of threat intelligence is a key challenge in network security. In this paper, we study the feasibility of cuckoo filters to improve space efficiency for threat intelligence matching, focusing on the example of threat intelligence monitoring.

**II. RELATED WORK & FUNDAMENTALS**

In this section, we will discuss related work (II-A) and explain the basic concepts behind cuckoo filters (II-B).

**A. Related Work**

Probabilistic data structures have a comprehensive history regarding applications in networking [3]. A recurring pattern in their use is to establish a filtering step prior to a complex processing task. For example, Dharmaparkari et al. make use of Bloom filters for matching byte signatures in network traffic [4]. False positives are mitigated by a downstream analyzer which applies a computational complex deterministic algorithm to verify matches. Upstream filtering to reduce the pressure on downstream processing is a common approach in the context of the wide-spread manager-worker pattern as shown in Figure 1. But, in case of Threat Intelligence matching, today's IOC sets are rapidly evolving. Hence, appropriate probabilistic data structures are required to support manipulating operations, i.e. insertions and deletions, without impeding continuous operation.

In 2014, Fan et al. introduced cuckoo filters [2], a probabilistic data structure for approximate set membership testing. Contrary to Bloom filters, cuckoo filters allow item removal without rebuilding the whole data structure or introducing false negatives. Interestingly, for many common applications cuckoo filters perform even better than Bloom filters, showing removal-supporting alternatives in terms of space cost and lookup performance [2]. Consequently, cuckoo filters have been brought up in the networking context [5]. In 2016, Valentin already suggested to investigate the applicability of cuckoo filters [6] for Threat Intelligence matching using the Bro Network Security Monitor [7], which is the basis for our work.

\*Please note that we use the terms upstream & downstream to refer to the order of processing steps.

higher the load factor, the collisions<sup>2</sup>. The achievable probability are conflicting. We focus on cuckoo filters for entries per bucket, as terms of space efficiency seen 0.002 and 0.00001. In this configuration as well and false-positive rates.

**II. INSIGHTS**

reference implementation of to implement cuckoo filters. The configuration of the false-positive reference implementation and a C++-template parameter. Although a cuckoo filter is same item multiple times [2], is stored only once. Given that the pressure on downstream data lack of element counts if needed, called *cuckoo-fm*<sup>3</sup>, is available. In this section we will discuss and during implementation. As the number of buckets varies instances, the larger target set of  $b_1$  to the actual number of buckets, corresponding residue class ( $mod\ m$  buckets) is not generally applicable, and therefore no involution for all

$$[a \oplus b]_m \neq [a]_m \oplus [b]_m$$

explained by the fact that  $x \otimes y$  operates  $b > m$ , the modulo operation might be-pattern in different ways on both sides. If  $m = 2^k$ , the modulo operation wise and of  $2^k - 1$ , which keeps the

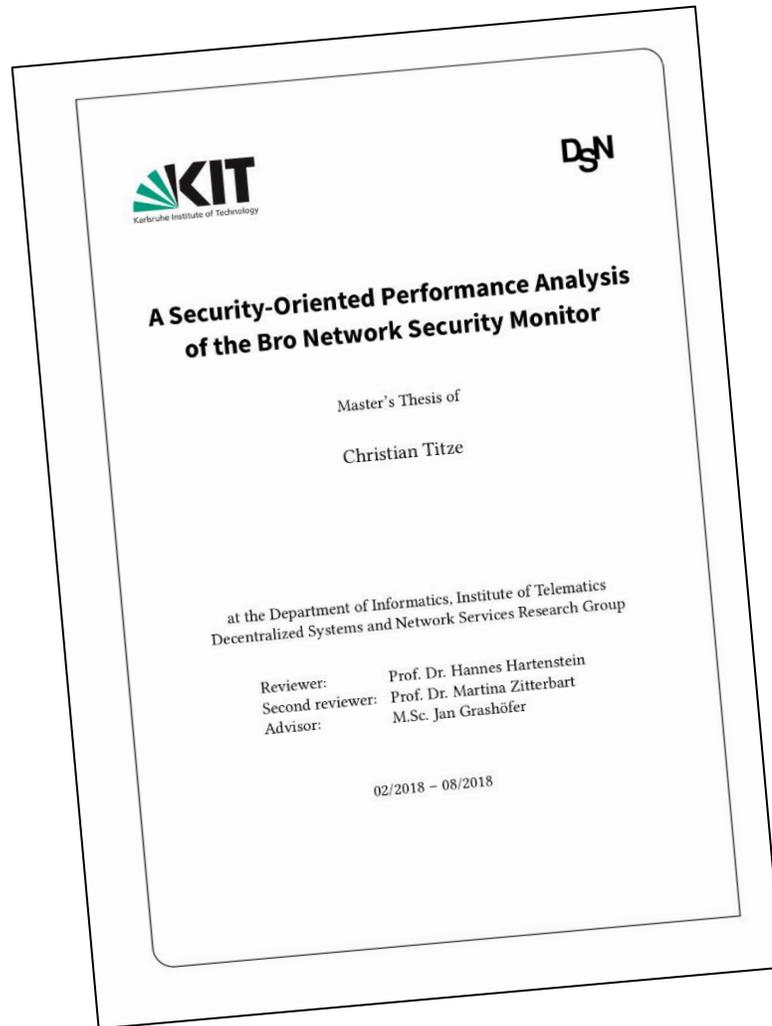
of entries per bucket also influences the achievable only freely available at: <https://github.com/kit-dsn/>

\* M. Mitzenmacher, S. Pontarelli, und P. Reviriego, „Adaptive Cuckoo Filters“, in *2018 Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX)*, S. 36–47.

Christian Titze

# **SECURITY-ORIENTED PERFORMANCE ANALYSIS**

# Research Question & Possible Answer



How can network traffic be leveraged to impact the performance of Bro?



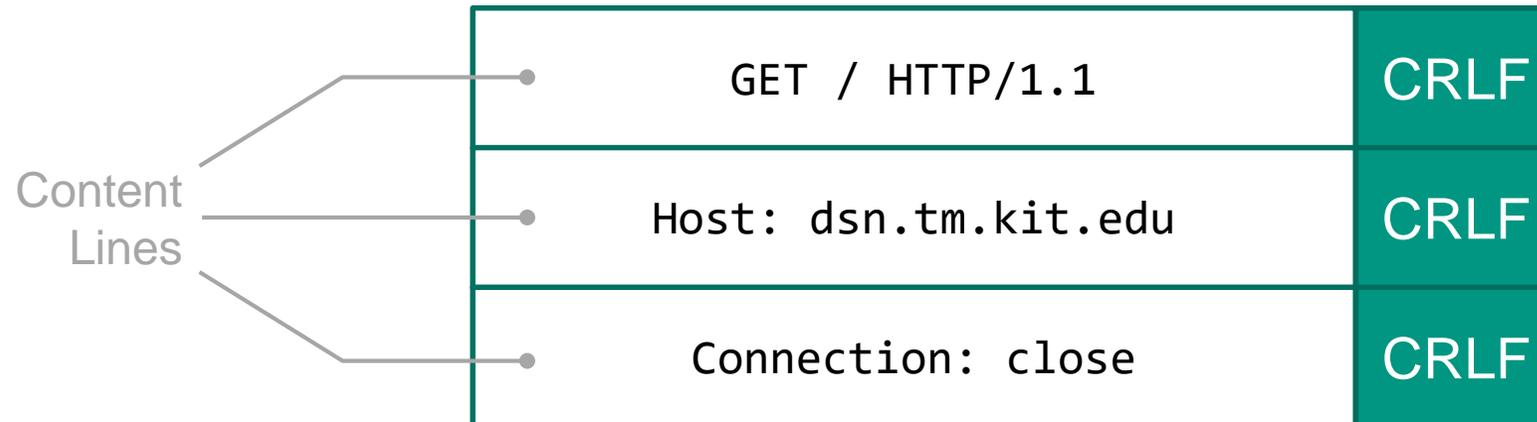
By exploiting the structure of text-based application-layer protocols to excessively generate events.

# Structure of Text-Based App-Layer Protocols

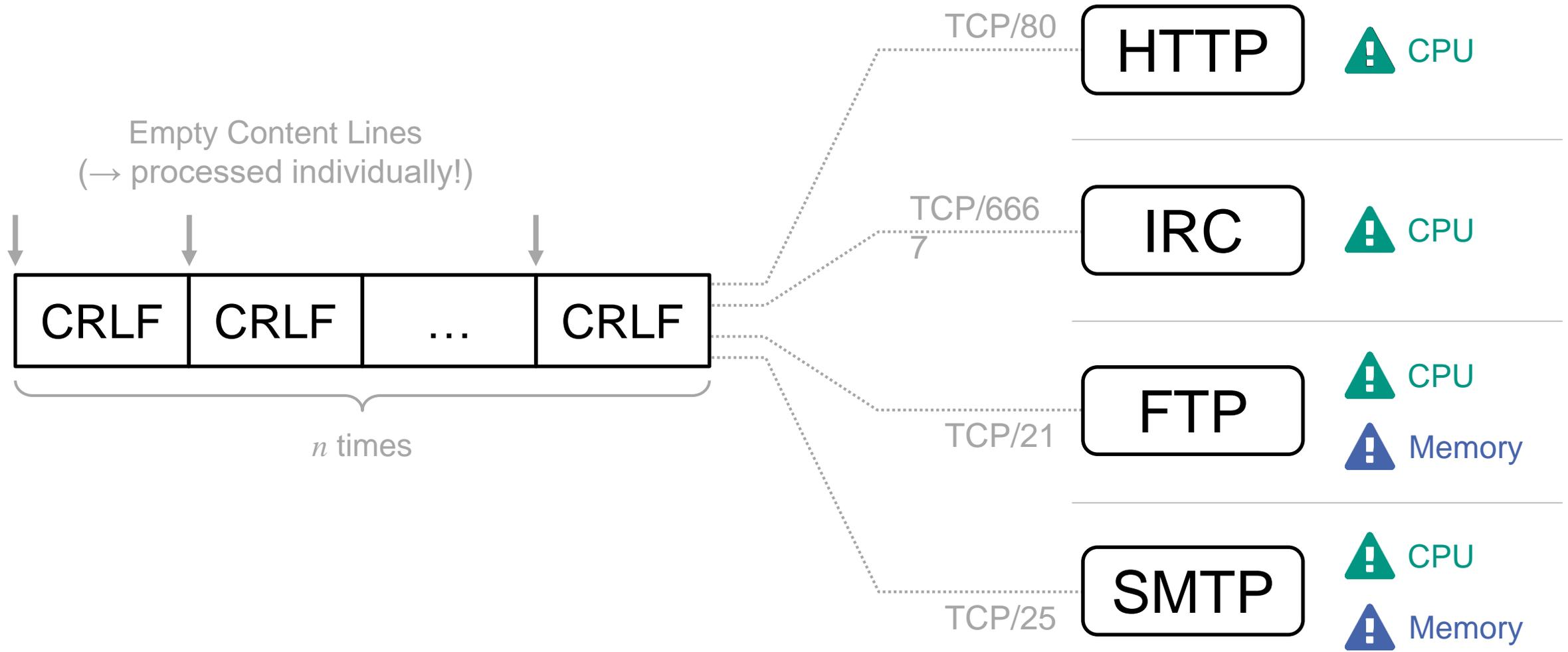
- Structure of virtually all text-based application-layer protocols <sup>[1]</sup>:



- For example, HTTP:



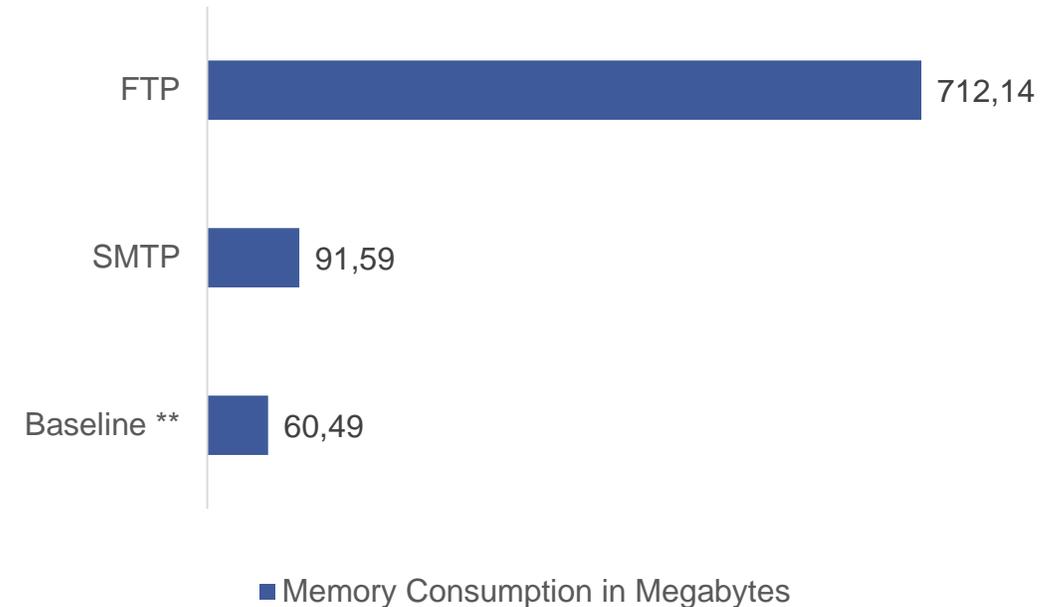
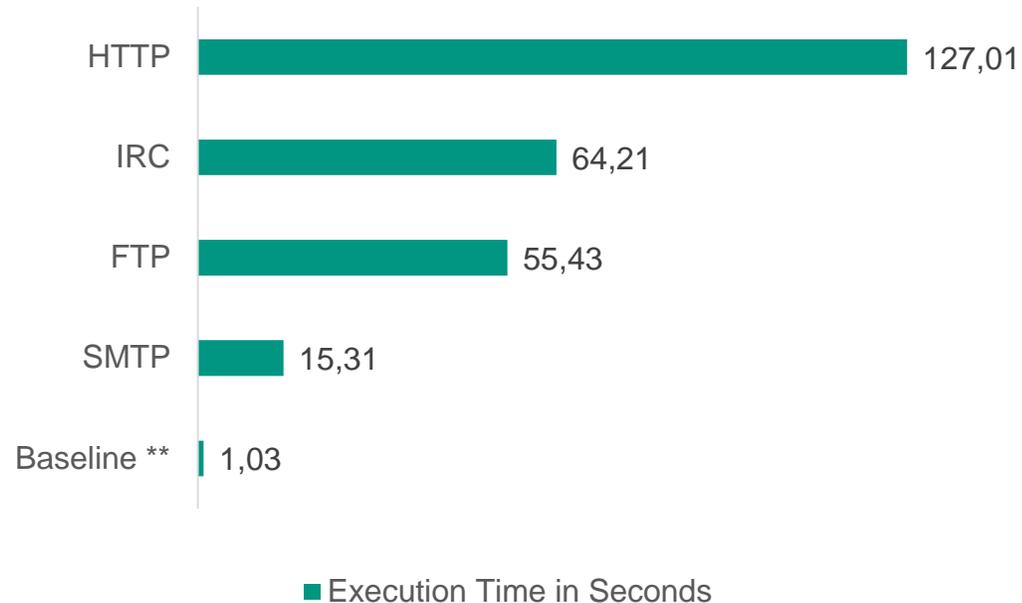
# Attack Traffic



# Performance Impact \*

## CPU

## Memory



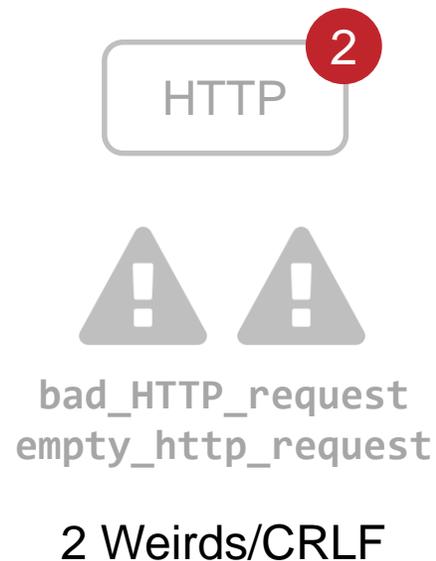
**2.1 MB**  
Attack Traffic

**1500**  
Packets

\* Numbers are for 1 Million CRLFs. Increase is linear with number of CRLFs.  
 \*\* With DPD's port detection off or when sent to non-vulnerable Analyzer.

# CPU Impact

- Caused by **excessive number of handled events**
- Normally: Only low-level events occur in excessive numbers
- Here: Empty content lines trigger other events excessively



# Memory Impact

- Every empty content line is treated as a new “request”
- State is kept about each “request”
- Each “request” is added to a queue of pending commands

~ 30 Bytes  
Allocated per CRLF

SMTP

~ 640 Bytes  
Allocated per CRLF

FTP

# Thank You!

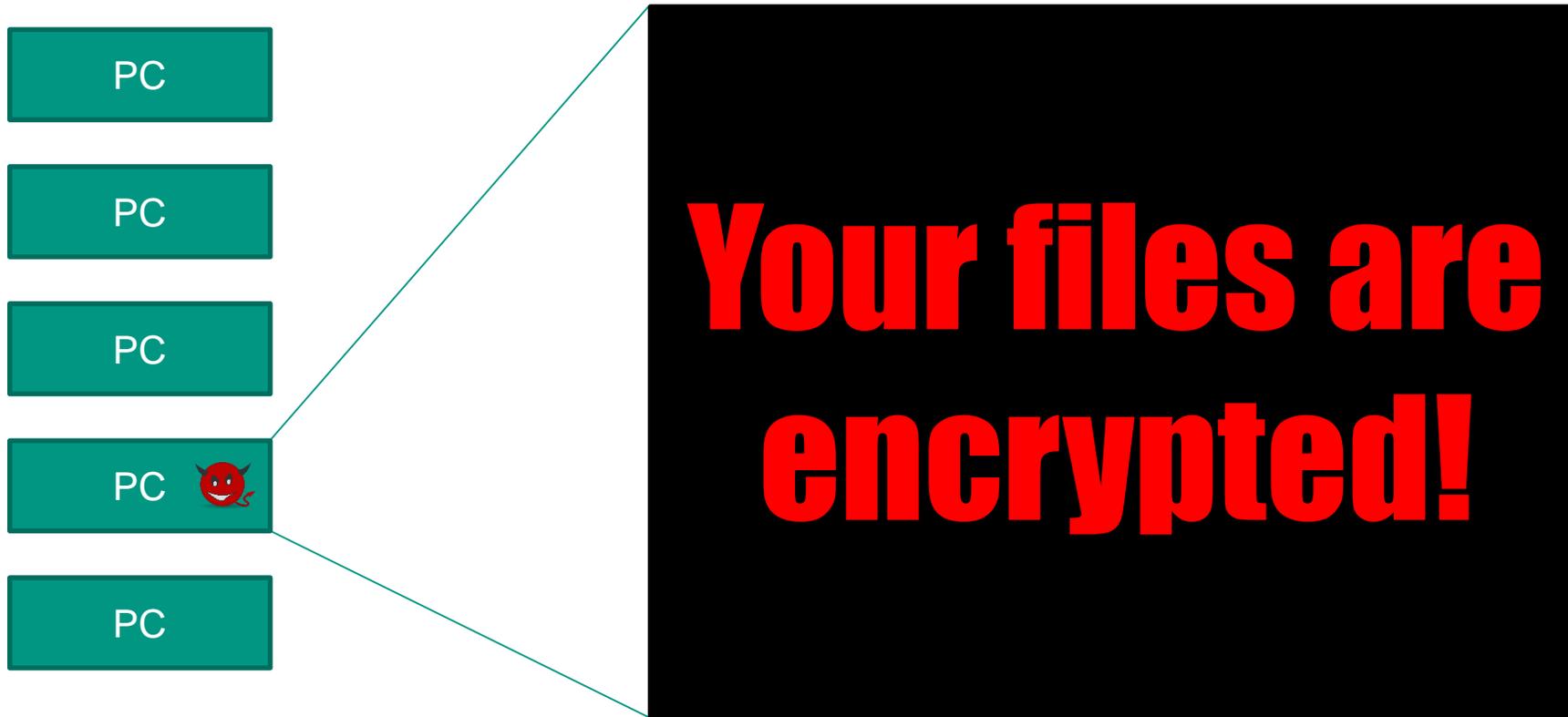
## “The monitor will be attacked.”

— Vern Paxson, *Bro: A System for Detecting Network Intruders in Real-Time*

Matthias Grundmann

# **RANSOMWARE MONITORING IN ACADEMIC ENVIRONMENTS**

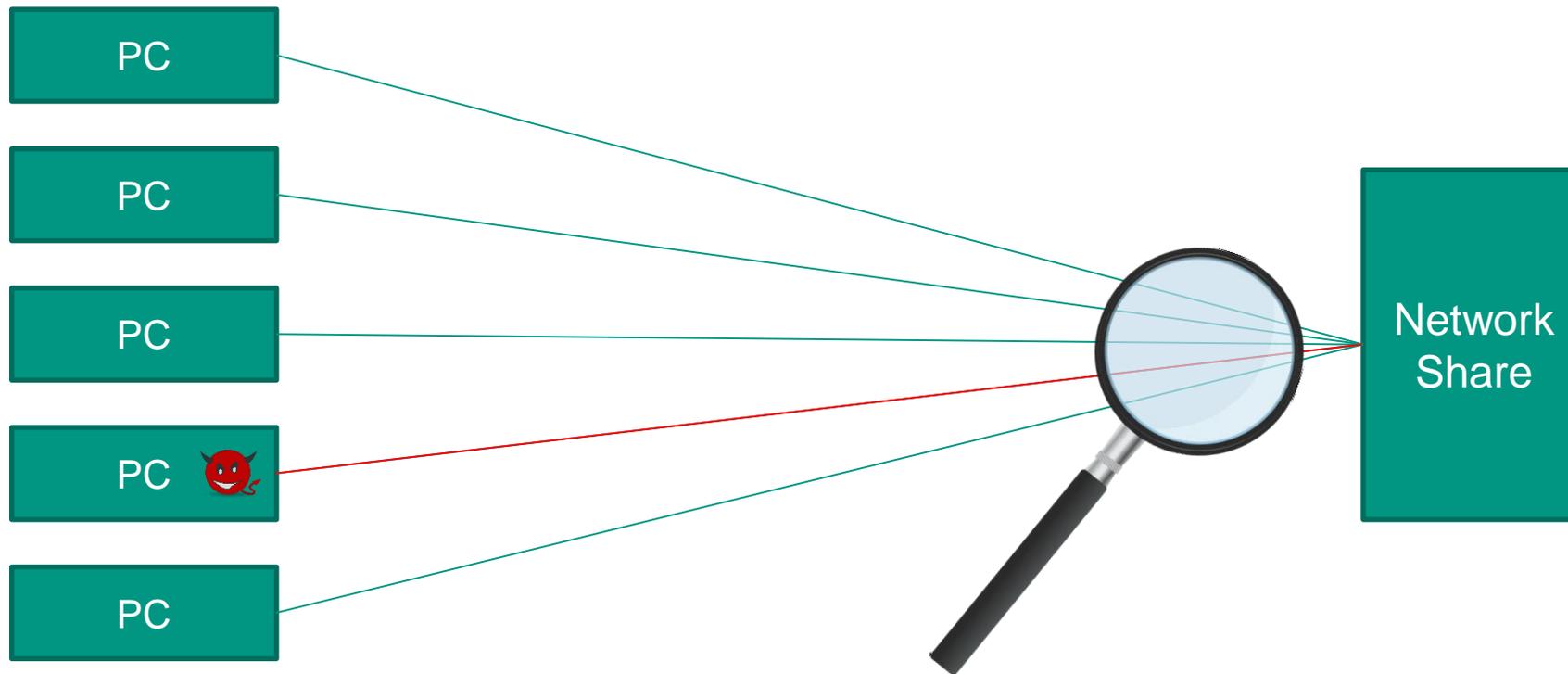
# The Problem



- We want to detect active ransomware as fast as possible

# Our Approach

- Monitor SMB traffic to network shares
- Can we detect encryption of files?



# Comparing Two Files...

```
00000350: 5500 0000 5600 0000 5700 0000 5800 0000 U...V...W...X...
00000360: 5900 0000 5a00 0000 5b00 0000 5c00 0000 Y...Z...[...\...
00000370: 5d00 0000 5e00 0000 5f00 0000 6000 0000 ]...^..._...`...
00000380: 6100 0000 6200 0000 6300 0000 6400 0000 a...b...c...d...
00000390: 6500 0000 6600 0000 6700 0000 6800 0000 e...f...g...h...
000003a0: 6900 0000 6a00 0000 6b00 0000 6c00 0000 i...j...k...l...
000003b0: 6d00 0000 6e00 0000 6f00 0000 7000 0000 m...n...o...p...
000003c0: 7100 0000 7200 0000 7300 0000 7400 0000 q...r...s...t...
000003d0: 7500 0000 7600 0000 7700 0000 7800 0000 u...v...w...x...
000003e0: 7900 0000 7a00 0000 7b00 0000 7c00 0000 y...z...{...\...
000003f0: 7d00 0000 7e00 0000 7f00 0000 8000 0000 }...~..._...`...
00000400: 5200 6f00 6f00 7400 2000 4500 6e00 7400 R.o.o.t. .E.n.t.
00000410: 7200 7900 0000 0000 0000 0000 0000 0000 r.y.....
00000420: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000430: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000440: 1600 0500 ffff ffff ffff ffff 0100 0000 .....
00000450: 108d 1094 9b4f cf11 86ea 00aa 00b9 29e8 .....d.O.....)
00000460: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000470: 3775 c014 0000 0000 8020 0000 0000 0000 7u...O.....
00000480: 5000 0100 7700 0500 7200 5000 6000 6900 P.o.w.e.r.P.o.i.
00000490: 6e00 7400 2000 4400 6f00 6300 7500 6d00 n.t. .D.o.c.u.m.
000004a0: 6500 6e00 7400 0000 0000 0000 0000 0000 e.n.t.....
000004b0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000004c0: 2800 0201 0200 0000 0300 0000 0000 ffff (.
000004d0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000004e0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000004f0: 0000 0000 0a00 0000 bd96 0000 0000 0000 .....
00000500: 0500 5300 7500 6d00 6d00 6100 7200 7900 .S.u.m.m.a.r.y.
00000510: 4900 6e00 6600 6f00 7200 6d00 6100 7400 I.n.f.o.r.m.a.t.
00000520: 6900 6f00 6e00 0000 0000 0000 0000 0000 i.o.n.....
00000530: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000540: 2800 0201 0400 0000 ffff ffff ffff ffff (.
00000550: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000560: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000570: 0000 0000 5100 0000 d8e2 0000 0000 0000 ...Q.....
00000580: 0500 4400 6f00 6300 7500 6d00 6500 6e00 .D.o.c.u.m.e.n.
00000590: 7400 5300 7500 6d00 6d00 6100 7200 7900 t.S.u.m.m.a.r.y.
000005a0: 4900 6e00 6600 6f00 7200 6d00 6100 7400 I.n.f.o.r.m.a.t.
000005b0: 6900 6f00 6e00 0000 0000 0000 0000 0000 i.o.n.....
000005c0: 3800 0201 ffff ffff ffff ffff ffff ffff 8.....
000005d0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000005e0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000005f0: 0000 0000 0000 0000 e001 0000 0000 0000 .....
```

Entropy:  
2.48

```
00000350: fae9 9d29 4b85 2d4b 5c2d f338 6ebe b4ad ...)K.-K\-.8n...
00000360: 5aad 1ca8 9e71 1880 cc2c cd66 e277 4334 Z....q....f.wC4
00000370: d328 d646 2663 f731 8e69 39e6 f8bf 4443 (.F&c.1.i9...DC
00000380: e124 99f1 ea8f d4f3 0a27 f8a8 873b b685 .$.....'!;..
00000390: 47db c243 dc89 dfd6 cbbd 5d39 ac91 8342 G..C.....]9...B
000003a0: aeb4 19ac 2b80 4d1d aaaf 917a 152b 5c25 ....+M....z.+%
000003b0: 61e0 05f7 1454 6e42 4cf6 7f1e fdf8 c4da a....TnBL.....
000003c0: 44d4 b570 6caf 69a0 cf6d 8fea c103 aadd D..pl.i.m.....
000003d0: fd56 e8ec 2bf0 4f40 0e03 510f 002e 5497 .V...+O@..Q...T.
000003e0: fe91 05bb 4efe e126 c8b4 3454 29f1 2cda ....N..&..4T) ,.
000003f0: a012 39d1 8320 1575 ca72 e0ca 5cc0 729c ..9.. .u.r...\r.
00000400: 9df4 b1fe b3b3 f136 24c6 982f 672d dd95 .....6$/g..
00000410: a84a 49b8 16e9 4b6f 5b01 5e2b 06d8 15ea .JI...Ko[.^.+...
00000420: 8595 6c92 a5a4 ad74 1769 3f3f c98d 2aff .l....t.i??..*.
00000430: ab66 0ffe 3e57 8be5 3406 fdbc 667f 3f64 .f.>W..4...f.?d
00000440: 3e82 bd43 d616 41f3 0263 e016 5eac 8f6d >..C..A..c..^..m
00000450: 06c5 42c0 b8df 3e57 29ab a88e b71d 71e3 ...z&..>T).....q.
00000460: 70f7 0000 ac0a 0000 050b 0e0d 0b30 .....R.....k..
00000470: c6f4 f342 2955 aa08 89d0 835e c800 5000 (...4).....s5....
00000480: 43b4 a3ab 8110 aa90 7cae d4db b009 c3ac C.....|.....Y..
00000490: 80c1 8ef0 084c 6c1d 6fb2 2b02 c013 a30c .....Ll.o.+...\.
000004a0: 0a0b 516c 0b13 6d26 71c9 9881 9d22 0859 ..Ql.m&q.....".Y
000004b0: 4f3f 8717 e523 0000 a3f4 0000 0000 f551 O?..#).....Q
000004c0: 1883 23ed bf07 6500 a683 0001 2500 858b ..#...e.....S..
000004d0: 52b9 df8d b249 d272 2590 2a00 8011 cd7d R....I."%.*....}
000004e0: a64f 51a6 6df5 d25d b409 22e7 c7d0 a5d0 .OQ.m...=/...Gr.
000004f0: 01db f212 e417 78ed c0ed c047 72cd 8023 .....x....Gr..#
00000500: 2961 5b55 6b9c 4cc0 5516 4471 9ddf f370 )a[Uk.L.U.Dq...p
00000510: 3ce0 26b2 1a03 0533 ebf5 bc0d 2767 128d <.&....3....'g..
00000520: 875b f7e9 ce19 aa87 9f05 bf6e 88fe 8509 .[.....n....]
00000530: a76a 66e2 d1bd 3253 f8dc 2858 4541 c55d .jff...2S..(XEA.)
00000540: 581b 309d ef72 85e2 1163 567b 23ca 493e X.O...r...eV{#.#.I>
00000550: 18f2 f871 5895 7794 d379 40eb d646 7dec ...qX.w..y@..F).
00000560: c7d7 e171 4ade 7125 567e de4b 282b e517 ...qJ.q%V~.K(+..
00000570: 4d9a 40ae b8e5 ca4a 824f d250 8f50 e9da M.@....J.O.P.P..
00000580: 4510 ae1a b835 29d3 ca97 7eab 7607 f398 E....5)....~.v...
00000590: 0aec c816 635f 0c5e c4d9 2540 6f05 899e .....c..^...%o...
000005a0: 8721 4283 daf5 9aec f1a2 1079 1e82 703f .!B.....y.p?
000005b0: df1a fd6c 74cb dce4 b4e3 f96a 9d69 95ff ...lt.....j.i..
000005c0: 844c 5046 fd35 698a 5fa4 5ac0 cb68 4d7c .LPF.5i.._Z..hM|
000005d0: 8acb a729 f002 3473 134b 7d80 6cea c188 (...).4s.K}.l...
000005e0: 0880 8f4e 0b6e 71be 6d6e 2703 e516 e651 ...N.nq.mn'!...Q
000005f0: 501a 5d17 f7af 2ea8 97af 2145 3410 e222 P.].....!E4.."
```

Entropy:  
7.69

# Detecting Encryption of Files

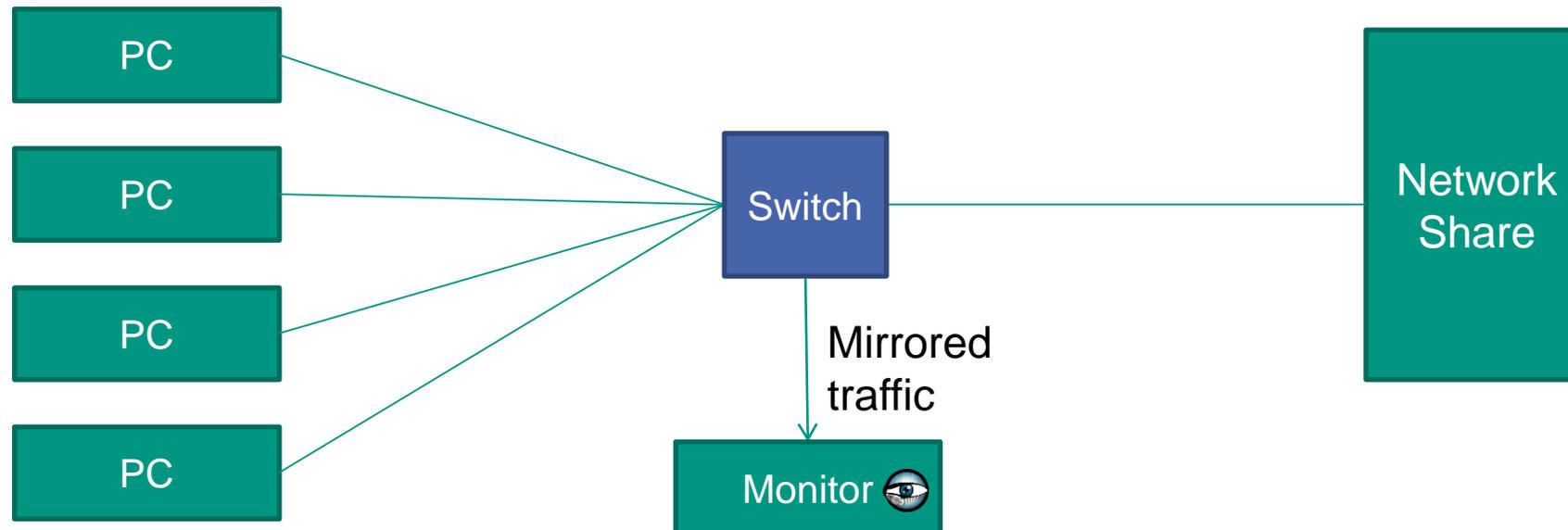
- Idea: Use entropy to detect encryption
- Compare entropy of SMB\_READ and SMB\_WRITE requests
- Implementation
  - Bro script
  - For every SMB\_READ
    - Calculate entropy of first 2 kB
    - Store entropy in cache for 10 min
  - For every SMB\_WRITE
    - Check if entry for read in cache
    - Calculate entropy of first 2 kB
    - Compare entropy
    - Alert if entropy increased by more than threshold

# Related Work

- Mike Stokkel, Fox-IT (Delft)
  - Presentation at BroCon '16
  - Detect writing of high-entropy files
    - Alert when more files encrypted than threshold
- D.A.C. Mülders, TU Eindhoven
  - Master thesis „Network based Ransomware Detection on the Samba Protocol”
  - Offline analysis of combination of read and write requests
    - Calculate difference of entropy and file size
    - Detect encryption and compression
- Eduard Steinmiller, Andreas Baumeister
  - KASTEL-Lab „Security“ in winter term 2017/18
  - Online analysis by observation of difference of moving average of entropy of read and write requests

# Setup for Evaluation

- Evaluation needs
  - Real user traffic
  - Real ransomware traffic
- Use KIT's university network for evaluation
  - Setup monitors SMB traffic to shared folders

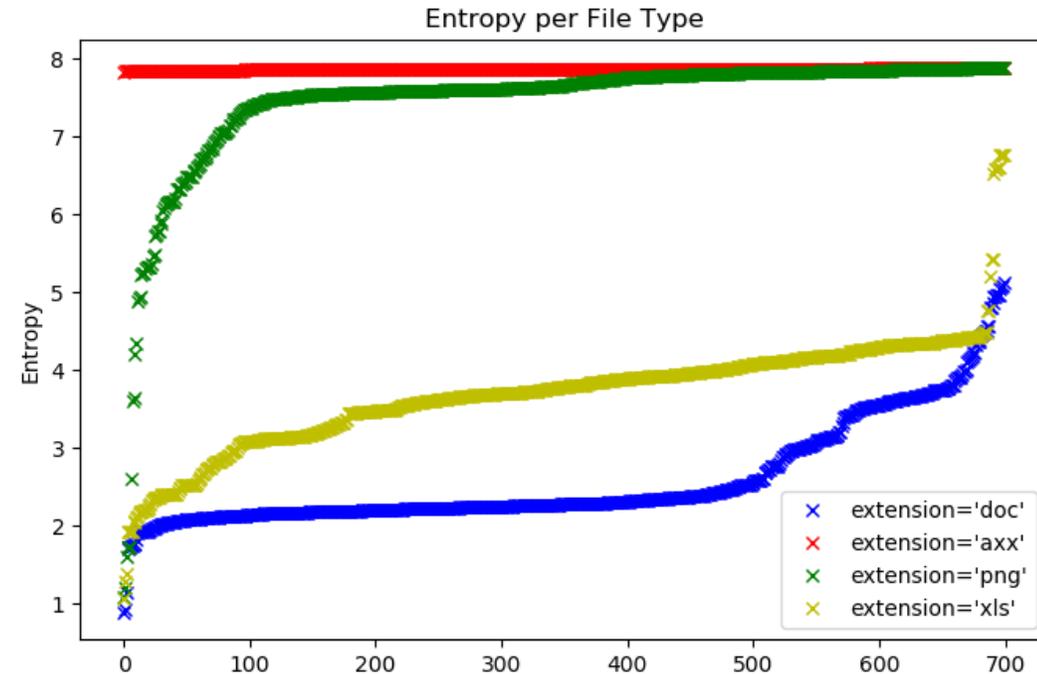


# Challenges

- Link SMB\_READ and SMB\_WRITE requests to same file
  - SMB file id not unique, so use tuple of SMB file id and connection uid
- Keeping state
  - Store entropy of read file, compare when same file written
- Process traffic of up to 8 Gbit/s
  - Package drops during traffic peaks
- Handling of false positives
  - Single encrypted file is no reason for alarm
  - Use SumStats framework to collect encryption incidents
  - Only alert if 5 encryptions in 30 seconds
- Privacy
  - Exclude shares with home directories

# Current State and Open Questions

- Detection quality
  - Encrypting multiple files in folder successfully detected
  - Few false positives
- Working on secure setup for tests with real ransomware



- Is entropy the best measure to detect encryption of files?
  - Maybe use randomness tests?
  - Analysis of file extensions and mime types?
  - Analysis of access patterns?

# Summary: Bro @ KIT

1. Improving Threat Intelligence matching with Bro
2. Security-oriented Performance Analysis
3. Ransomware detection in academic environments