Analyzing IoT Malware

Emanuele Cozzi[1], Pierre-Antoine Vervier, **Matteo Dell'Amico**[1], Yun Shen[2], Leyla Bilge[2] and Davide Balzarotti[1]

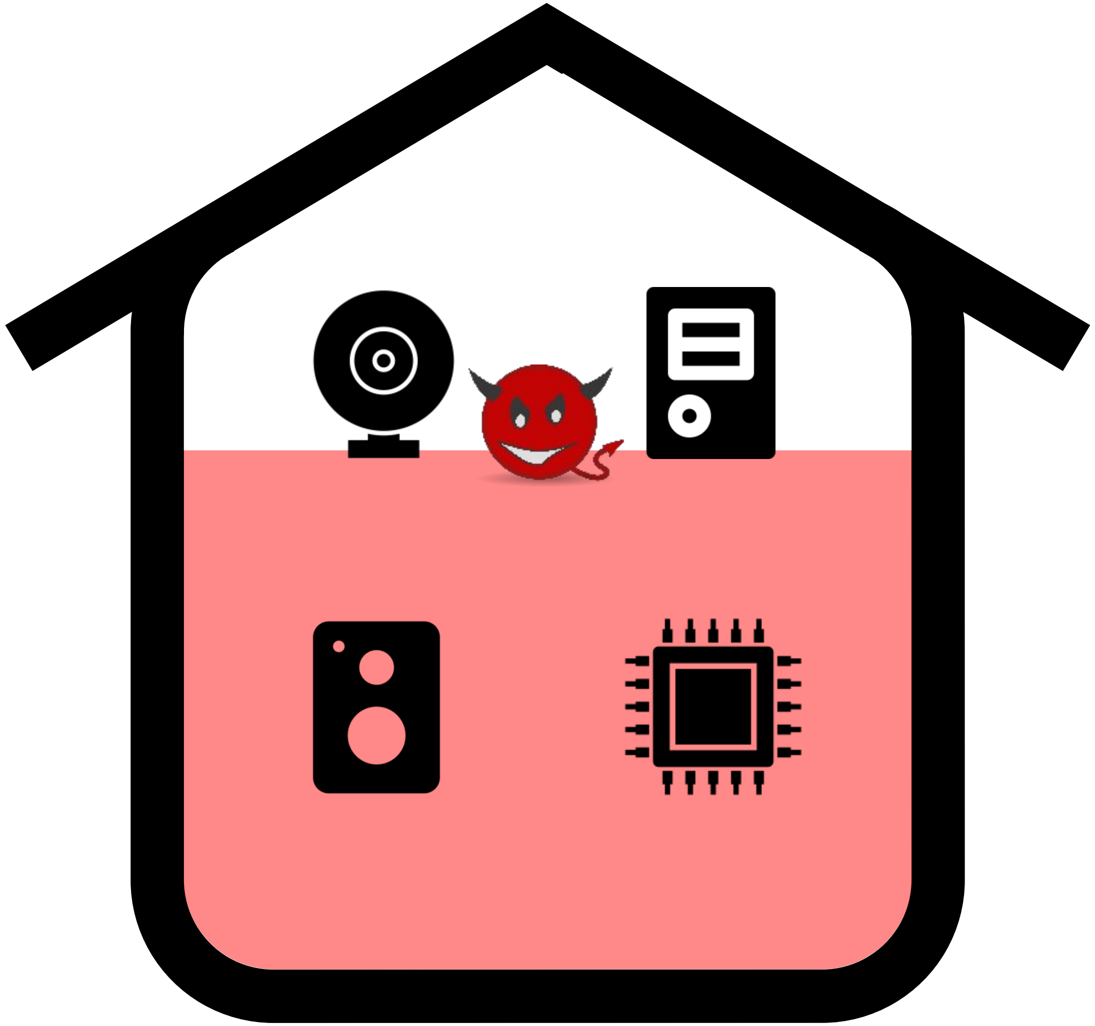
Based on the experimental work in our ACSAC 2020 paper: *The Tangled Genealogy of IoT Malware*

EURECOM

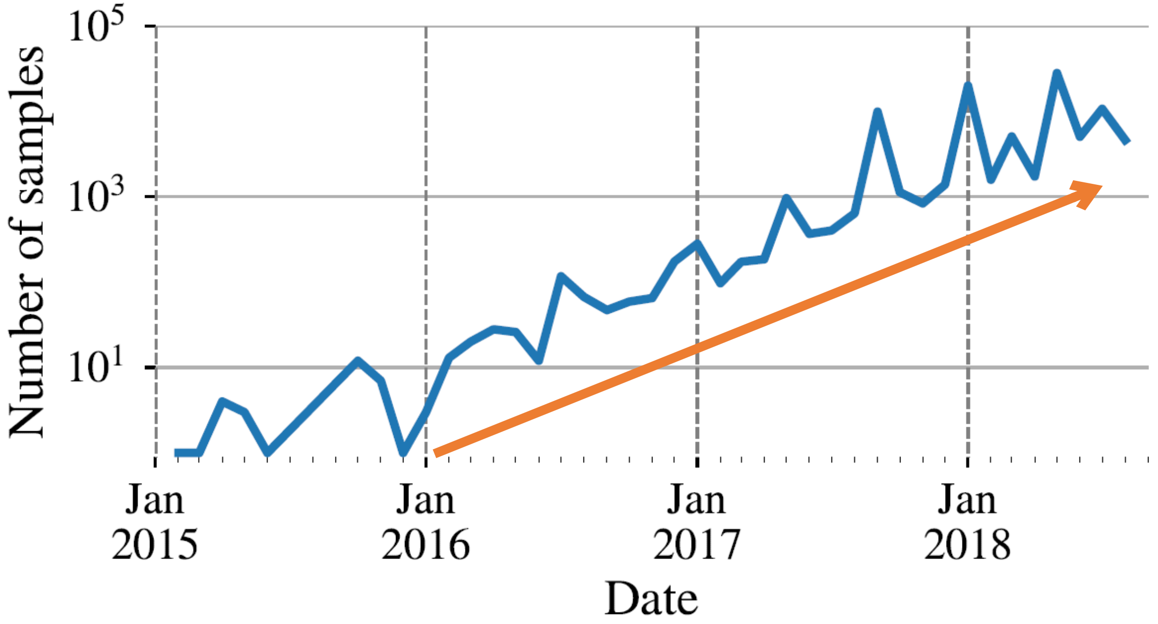
NortonLifeLock

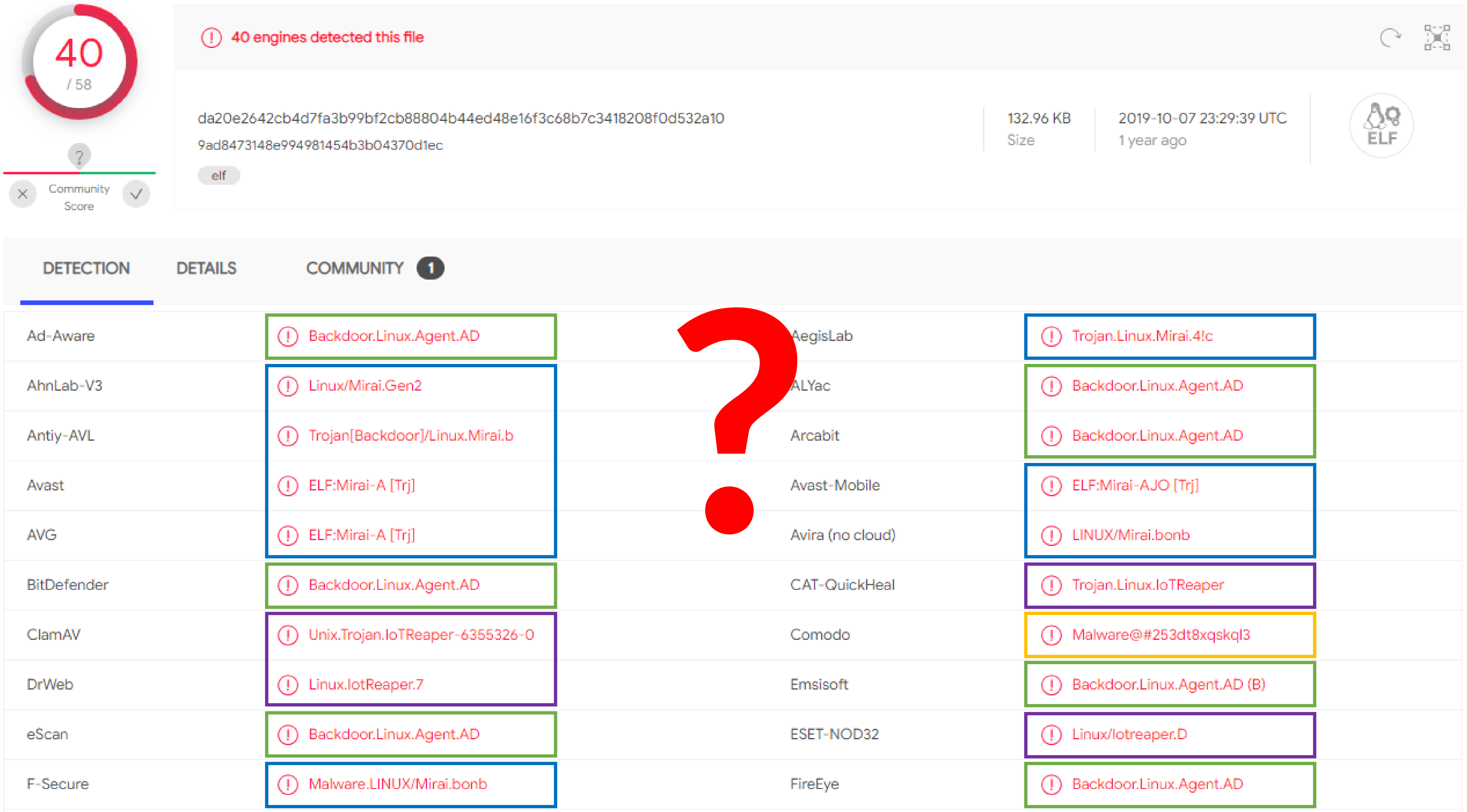
Learning from Authoritative Security Experiment Results (LASER) 2020

IoT devices and malware

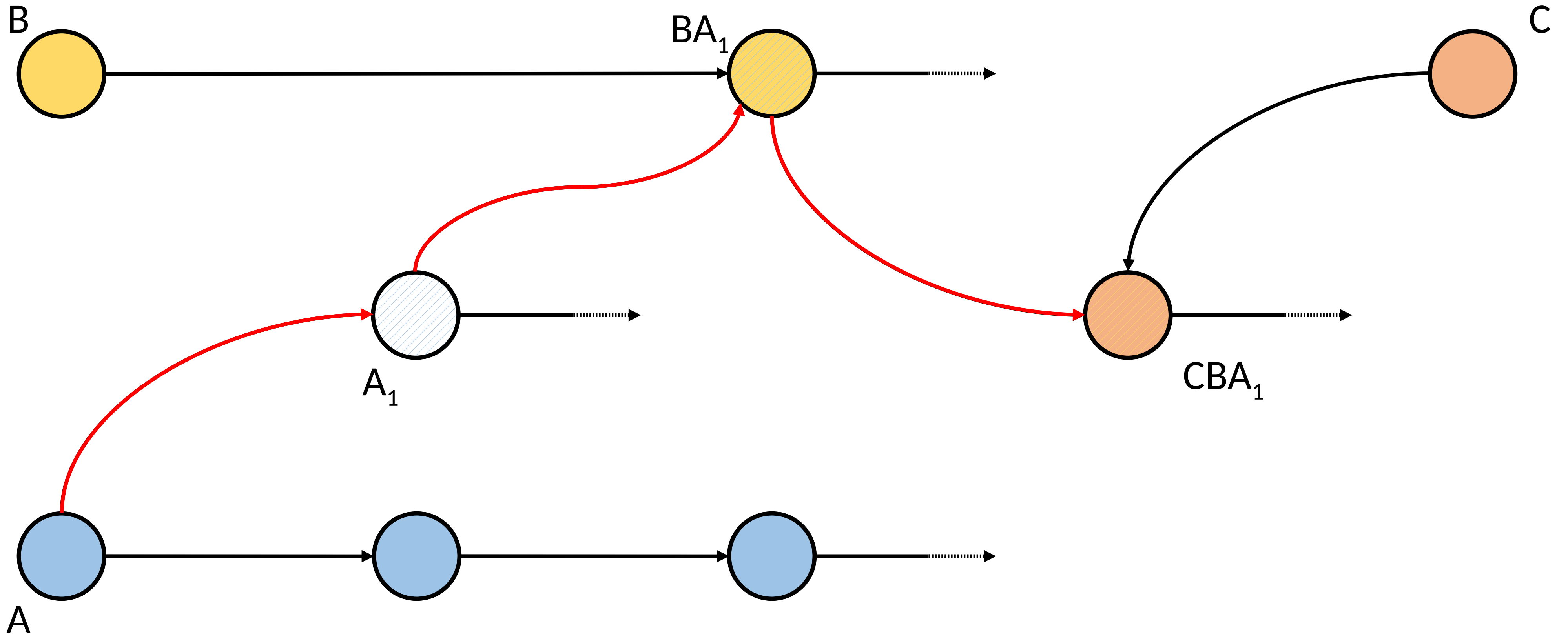


Submissions on VirusTotal

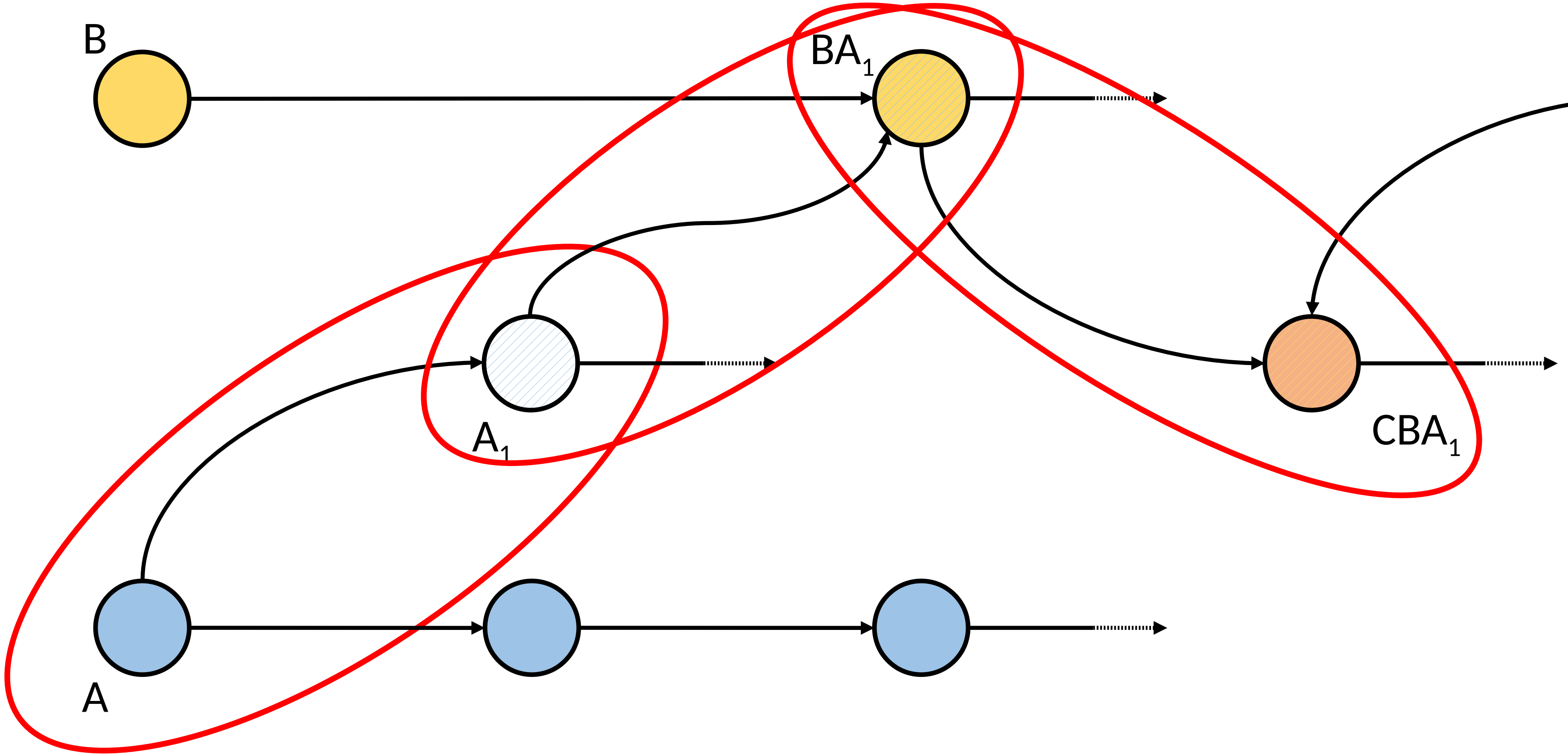



40
/ 58
40 engines detected this file
da20e2642cb4d7fa3b99bf2cb88804b44ed48e16f3c68b7c3418208f0d532a10
9ad8473148e994981454b3b04370d1ec
elf
132.96 KB
Size
2019-10-07 23:29:39 UTC
1 year ago
ELF
Community Score
DETECTION
DETAILS
COMMUNITY 1
Ad-Aware
Backdoor.Linux.Agent.AD
AegisLab
Trojan.Linux.Mirai.4!c
AhnLab-V3
Linux/Mirai.Gen2
ALYac
Backdoor.Linux.Agent.AD
Antiy-AVL
Trojan[Backdoor]/Linux.Mirai.b
Arcabit
Backdoor.Linux.Agent.AD
Avast
ELF:Mirai-A [Trj]
Avast-Mobile
ELF:Mirai-AJO [Trj]
AVG
ELF:Mirai-A [Trj]
Avira (no cloud)
LINUX/Mirai.bonb
BitDefender
Backdoor.Linux.Agent.AD
CAT-QuickHeal
Trojan.Linux.IoTReaper
ClamAV
Unix.Trojan.IoTReaper-6355326-0
Comodo
Malware@#253dt8xqskql3
DrWeb
Linux.IotReaper.7
Emsisoft
Backdoor.Linux.Agent.AD (B)
eScan
Backdoor.Linux.Agent.AD
ESET-NOD32
Linux/Iotreaper.D
F-Secure
Malware.LINUX/Mirai.bonb
FireEye
Backdoor.Linux.Agent.AD

Inter- and intra-family variety



Classification of variants



Our Dataset

- Goal: a **comprehensive view** of IoT malware
- Two conflicting goals
  - Have **as many as possible**
  - Avoid **false positives**
- We got all ELF binaries submitted to VirusTotal (Jan ’15-Aug ’18)
  - We excluded Android
  - We excluded x86/AMD64 binaries (to exclude desktop/servers)
  - Flagged malicious by at least 5 AV engines
- Result: 93.7k samples

First Approach: Feature-Based Clustering

Feature-Based Clustering

- **Unsupervised** method: we don't have a trusted ground truth
- We do have a **pseudo**-ground truth: AV engines' labels
  - Synthetised in AVClass (Sebastian et al. RAID '16)
  - One of our goals is **evaluating** it (and discovering classification mistakes)
- We go for **clustering**

Other ideas, when the "ground truth" isn't really reliable?

Features

- Extracted with Padawan (Cozzi et al., IEEE S&P ‘18)
- 7 categories (143 features in total):
  - **Bytes** (12): entropy, headers, footers, character frequencies
  - **Elf** (54): info obtained parsing the executable (e.g., anomalies, # of sections, stripped, ...)
  - **Strings** (3): IP addresses, paths & URLs found in the binary
  - **Idapro** (16): statistics obtained by disassembling (e.g., # of functions & basic blocks...)
  - **Behavior** (42): data collected from running in the sandbox
    - E.g., read & written files, # of syscalls, ...
  - **Dynamic** (3): errors, stderr, stdout
  - **Nettraffic** (13): network behavior (e.g., # of connections, IPs contacted, DNS activity...)

Processing the Features

- Numeric:
  - $x \rightarrow \log(1 + x)$ to avoid large values dominating
  - Divide by standard deviation (i.e., set stdev=1)
- Categorical (sparse matrixes):
  - One-hot encoding (a categorical feature with n values becomes n boolean features)
  - Tf-idf normalization (i.e., lower the weight of frequent features)
- Multi-sets (e.g., list of domans queried by DNS):
  - Sum of the categorical features
- Paths:
  - Become multisets by taking full path, filename and all parent directories

Clustering Algorithm

- Difficult dataset
  - Very high-dimensional; we need sparse representation (due to one-hot encoding)
  - Missing values (e.g., for cases where disassemble fails or no/trivial behavior)
- We use **FISHDBC**, an algorithm for **arbitrary (dis)similarity** functions
  - Approximates HDBSCAN*, an algorithm of the **density-based** family (DBSCAN and friends)
  - Uses HNSW, a data structure for **approximated nearest neighbors** in non-metric space
    - Scales in complex spaces because we **don't compute all pairwise dissimilarities**
  - Ad-hoc "distance" function for our data:
    - Euclidean, for numeric features
    - Cosine, for categorical ones
    - We ignore "null" columns

Other options to deal with missing/insignificant values?

Validation

- While we don't have a reliable ground truth, we do use AVClass as a **pseudo-ground truth**
  - Our clustering should generally agree with AVClass labels
  - We investigate (some) disagreements manually
  - We check if our clustering can find AVClass misclassifications

"Brute force" analysis, by looking at which group of features makes most sense

- We turn on&off **feature groups** to verify which features are most useful
- We consider wheter samples end up in **pure** (all same AVClass label), **single** (one AVClass label+unknown), **majority** (90%+ one label) or **mixed** clusters
  - We also have **unclustered** samples in density-based approaches

Validation Results

- Binary-specific features (ELF, IDA Pro) are quite precise but they result in very narrow clusters
- Behavior-specific ones are very generic (same observed behavior)
- Through manual evaluation, we couldn't find mislabelings in AV engines

Feature groups					Clusters (# samples)			
ELF	IDA Pro	strings	behaviour	network	*pure*	*single*	*majority*	*mixed*
✓					**44,491**	**4,657**	31,649	14,204
	✓				3,677	45	316	1,082
		✓			18,141	3,120	23,412	50,328
			✓		27,889	1,097	5,726	60,289
✓	✓	✓	✓	✓	34,313	2,337	12,741	45,610
✓	✓	✓	✓		38,825	3,062	24,234	27,531
✓	✓	✓			39,904	2,495	17,667	33,586
✓	✓				42,427	2,587	**34,118**	14,520
		✓	✓	✓	20,822	983	12,964	58,883

Manual Analysis

- We went through a lot of manual analysis to understand in more detail what was happening
- Analyze cluster centroids & their most relevant features
- Get into more details about single samples

```
Cluster -1, 62,698 elements
25,973 gafgyt, 17,016 mirai, 9,487 None, 2,437 tsunami, 813 dofloo, 6972 others
Numeric centroid: entropy 6.05±0.95 [+0.19σ] max_entropy 5.72±2.28 [+0.15σ] min_entropy 4.36±2.38 [+0.21σ]
Top categorical features:
 [0.13] 1.69±0.73 elf.link:static
 [0.08] 0.63±0.93 elf.machine:ARM 32-bit
 [0.06] 0.42±0.81 elf.machine:Intel 80386
 [0.06] 0.54±0.50 elf.e_phnum:3
 [0.06] 0.85±0.36 bytes.unique_bytes==256:True

Cluster 1141, 426 elements
372 asacub, 54 None
Numeric centroid: entropy 6.57±0.03 [+0.70σ] max_entropy 6.85±0.05 [+0.58σ] min_entropy 6.04±0.02 [+0.87σ]
Top categorical features:
 [0.26] 2.00±0.00 elf.interpreter:<none>
 [0.20] 2.00±0.00 elf.link:dynamic
 [0.16] 1.00±0.00 elf.dynfuncs:_ZN7_JNIEnv20CallStaticLongMethodEP7_jclassP10_jmethodIDz
 [0.16] 0.94±0.24 elf.dynfuncs:JNI_OnLoad
 [0.16] 2.00±0.00 elf.machine:ARM 32-bit

Cluster 131, 346 elements
346 None
Numeric centroid: entropy 4.15±0.00 [-1.69σ] max_entropy -1.00±0.00 [-2.36σ] min_entropy -1.00±0.00 [-1.88σ]
Top categorical features:
 [0.46] 2.00±0.00 elf.machine:Intel 80386
 [0.35] 1.00±0.00 elf.entrypoint:0x80486ce
 [0.35] 1.00±0.00 bytes.longest_sequence.length:1849
 [0.34] 1.00±0.00 elf.nsections:7
 [0.34] 1.00±0.00 elf.e_shnum:7

Cluster 3471, 334 elements
235 gafgyt, 98 tsunami, 1 hydra
Numeric centroid: entropy 5.62±0.09 [-0.24σ] max_entropy 6.10±0.16 [+0.30σ] min_entropy 1.30±0.85 [-0.98σ]
Top categorical features:
 [0.43] 1.00±0.00 bytes.longest_sequence.length:16356
 [0.36] 0.99±0.12 bytes.footer:005f5f47495f7864725f73686f727400
 [0.34] 2.00±0.00 elf.machine:ARM 32-bit
 [0.27] 1.00±0.00 elf.nsections:20
 [0.26] 1.00±0.00 elf.entrypoint:0x8190

Cluster 3366, 325 elements
325 gafgyt
Numeric centroid: entropy 6.30±0.00 [+0.44σ] max_entropy 6.41±0.00 [+0.42σ] min_entropy 5.08±0.00 [+0.49σ]
Top categorical features:
 [0.45] 2.00±0.00 elf.machine:Intel 80386
 [0.43] 1.00±0.00 elf.entrypoint:0x8048168
 [0.35] 1.00±0.00 elf.nsections:16
 [0.34] 1.00±0.00 elf.e_shnum:16
 [0.28] 1.00±0.00 bytes.longest_sequence.le
```

The “-1” cluster is “noise”: elements that are actually not clustered with anything else

A First Failure

- Our interpretation was that the features we've been collecting were **simply not powerful enough** for our final goal
- We do find signal, but it's **not the signal we were looking for**
  - We end up clustering by architecture, details of the binaries, …
- We need to restart from scratch with an approach that better reflects commonality in code

Happy Ending: Bindiff-Based Clustering

Code Doesn’t Lie: Using Bindiff

- Current IoT malware is not very sophisticated, and it lends itself well to decompilation in most cases
- We use the Diaphora diffing tool, which takes two binaries and outputs similarity scores between functions
  - Open source & easy to customize for us
- Dissimilarity score: *1 / (# of function pairs with similarity at least 0.5)*
  - We experimented with several approaches, this proved to be the most reliable one
- At first, we only consider dinamically-linked & unstripped binaries
  - Similiarity in libraries could would drive us astray, we remove libraries from unstripped files

Deconstructing the Clustering Algorithm

- The FISHDBC algorithm we used is based on
  - HNSW for search in complex & non-metric spaces
  - A generalized spanning tree based on distances between items
  - A procedure to build clusters on top the spanning tree
- We've found that the spanning tree itself carries most of the information we were looking for
- We just use HNSW and the spanning tree

For us, not treating ML as a "black box" algorithm helped

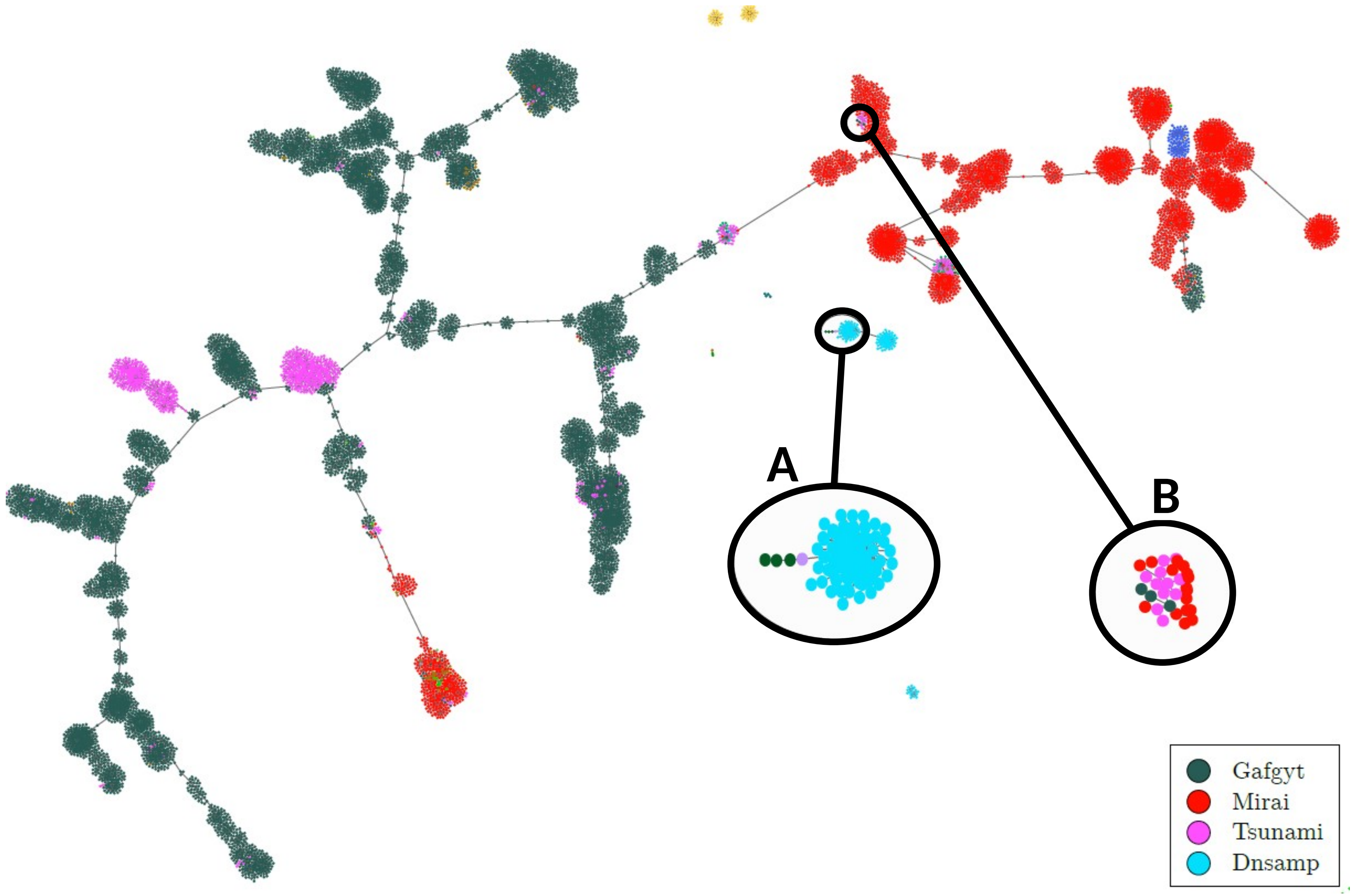
Extracting Library Code

- We want to make our approach work on files that are both statically linked **and** stripped
- We need to detect library code
- We piggyback on binary diffing itself: we use the HNSW to query for similar unstripped files
- When known library functions match others, we mark those (and the following in the file) as library code; we ignore it afterwards

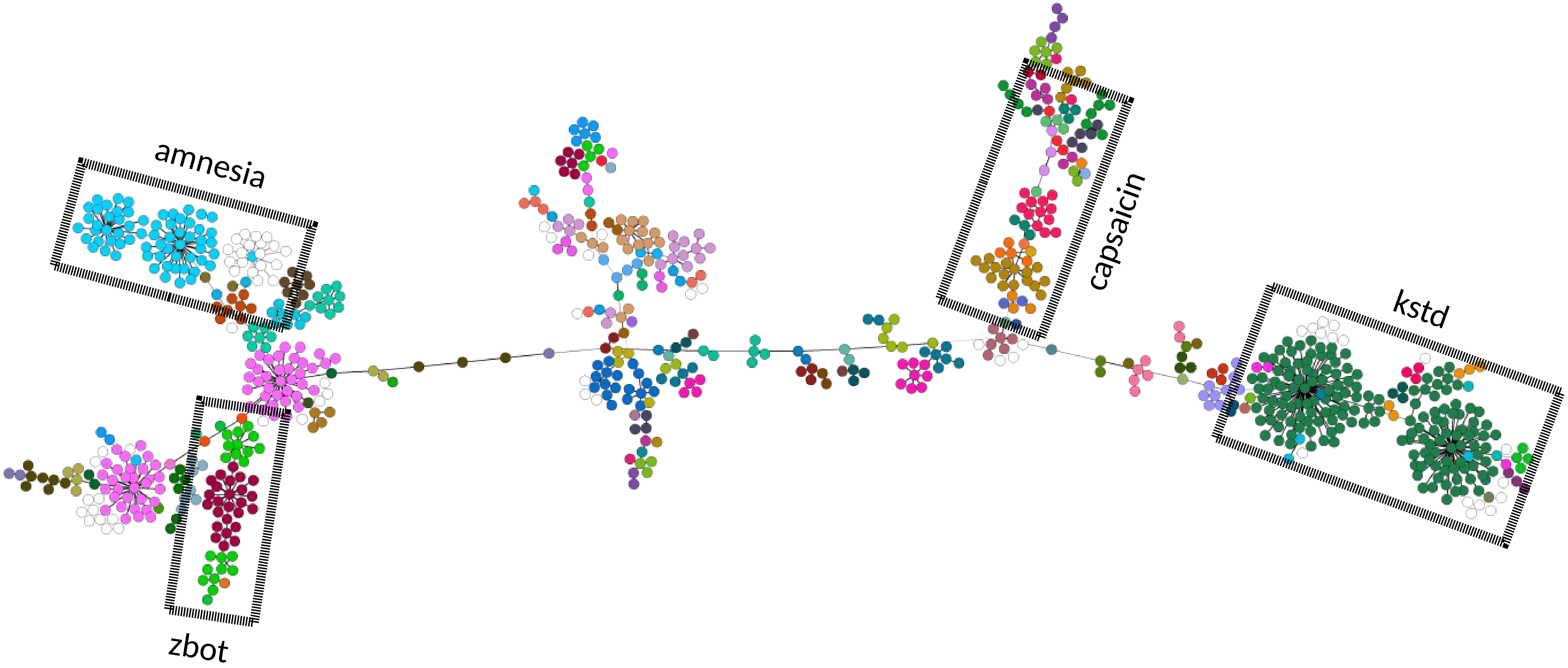
Results

- This new approach, this time, got us results that satisfied us
- Through manual analysis, we were able to **confirm** our spanning tree was a very good representation of the lineage between samples
- We were able to identify errors in AV labels

Code reuse



Variants



Discussion

Outside the Box of Feature Extraction

- We've seen that
  - The standard approach of extracting numeric/categorical features wasn't powerful enough for us
  - In our case, an almost-out-of-the-box similarity function got us the results we were looking for
- This is not a special case
  - For strings: edit distance
  - For files: fuzzy hashes
  - Deep neural networks for binary files: similarity is more precise than embedding (Li et al. ICML'19)

Could Our Work Use a Classical Approach?

- Many of the Diaphora heuristics test for equality of various characteristics
- We can’t exclude that carefully using those features would have worked
- However, that would have
  - Required a lot of work (re-implementing Diaphora’s algorithms)
  - Lost compatibility with future improvements/other approaches (e.g., deep neural networks)
  - Lost agility (e.g., ad-hoc code to handle specific cases)

Conclusions & Open Questions

- The goal of our study is to get a comprehensive panorama of IoT malware
- Current low sophistication enabled a largely automated approach
  - Will this be possible in the future? Will this research question always remain open?
- “Traditional” feature-based approaches didn’t work for us
  - How widespread is this issue?
- Engineering a system based on ad-hoc similarity functions solved the problem
  - We believe it’s an agile approach that we’re finding effective in various areas of security
- We’re putting data on https://github.com/eurecom-s3/tangled_iot