Inline Detection of Copy-Paste Botnet C&C

Jordan Garzon, Asaf Nadler

Akamai Technologies, Enterprise Security Research

BotConf 2020

Akamai

Experience the Edge

About Us

Jordan Garzon

Data Scientist
Akamai Enterprise Security Research

jgarzon@akamai.com

Asaf Nadler

Principal Researcher Lead
Akamai Enterprise Security Research

anadler@akamai.com

 |

Agenda

1 Introduction

What are copy-paste botnets?
How to shut down bot communication to these botnets using inline URL filtering?

2 System Overview

A system that processes repetitive patterns of copy-paste botnet URLs to identify new botnets

3 Analysis & Takeaways

Introduction

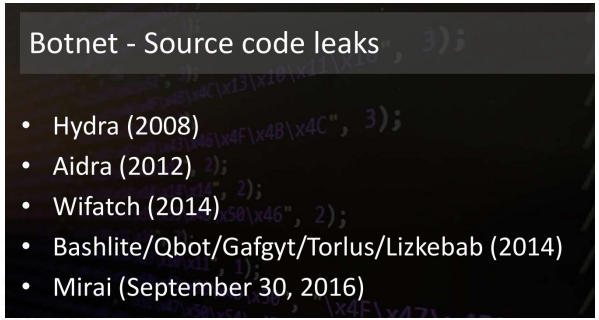
- Bots often communicate with their command and control (C&C) servers over HTTP/S

- A common defensive approach to block bot C&C communication over HTTP/S is by using **URL blacklists.** However, blacklists commonly suffers from a high false negative rate (i.e., “misses”)

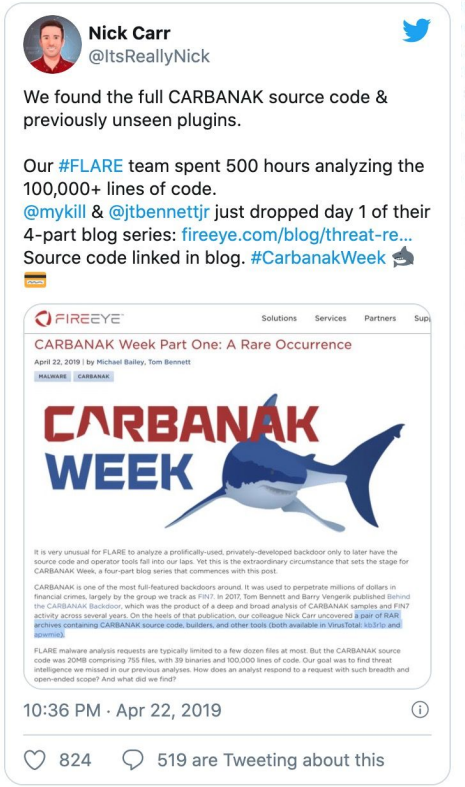
- We’re interested in detecting bot C&C communication over HTTP/S inline despite not being listed in URL blacklists

Background: Copy-Paste Bots

- The source code of bots and their C&C servers is often leaked and published online
- Bot owners often choose to copy and reuse leaked source code to save development time and effort



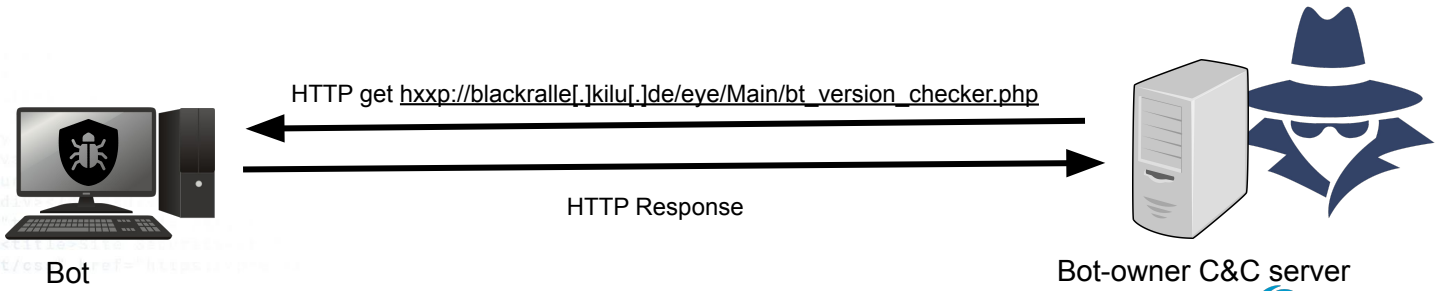
Taken from *"Mirai - Beyond the Aftermath", Joven et al., BotConf 2018*



 |

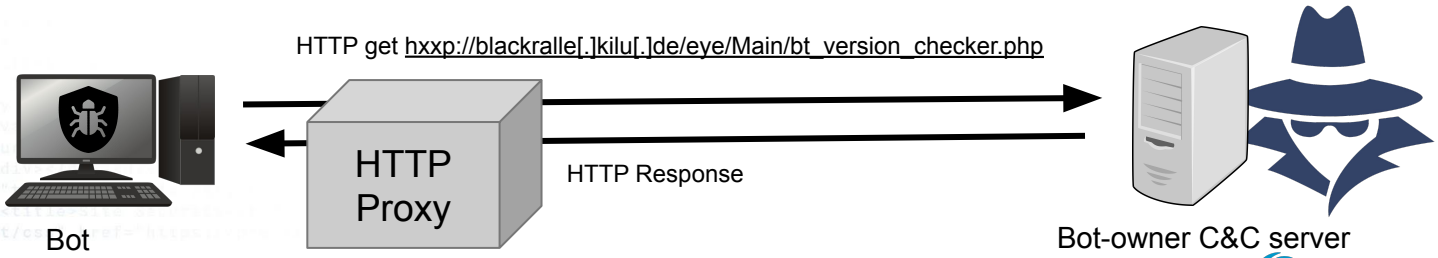
Background: Copy-Paste Bots

- Copying source code results in inherent similarities
- For example: see recent SpyEye C&C URLs
  - hxxp://pr0ghz[.]dsdo-services[.]cz[.]cc/Main/bt_version_checker.php
  - hxxp://cobra07[.]webuda[.]com/pcy/Main/bt_version_checker.php
  - hxxp://dfasd[.]22web[.]net/tom/Main/bt_version_checker.php
  - hxxp://cms[.]micadel[.]fr/logs/Main/bt_version_checker.php
  - …
- Can we infer that a newly seen URL of the form hxxp://*/Main/bt_version_checker.php is a SpyEye C&C URL?



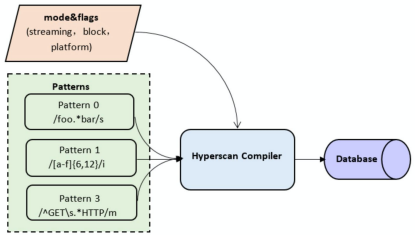
Background: Inline URL Filtering

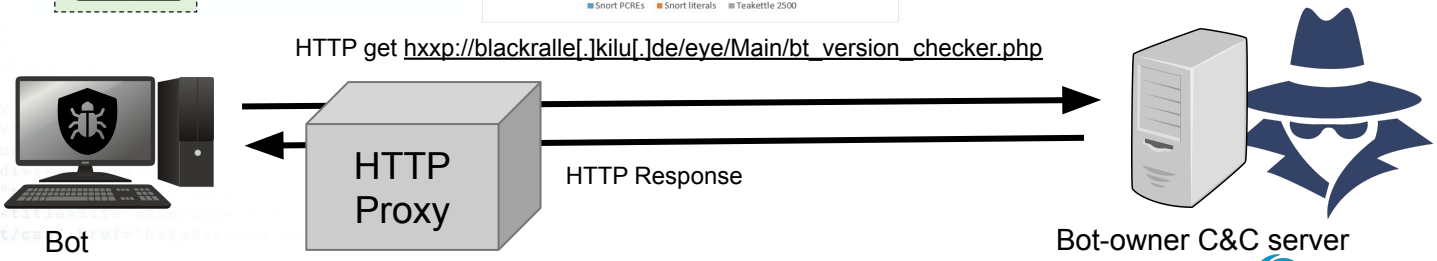
- Malicious URL detection is a well-studied problem
  - Batch learning, online learning, representation learning, etc. [1]
- Various model provide good results (e.g., neural networks) but deploying them on high-performing HTTP proxy for blocking inline HTTP requests requires extensive computation



Background: Inline URL Filtering

- We therefore examine a simpler solution, typically used by HTTP proxies -- **inline regular expression scanning**
- For instance: through compiling regex in advance using the open-source Intel Hyperscan engine [7], and applying it in real-time

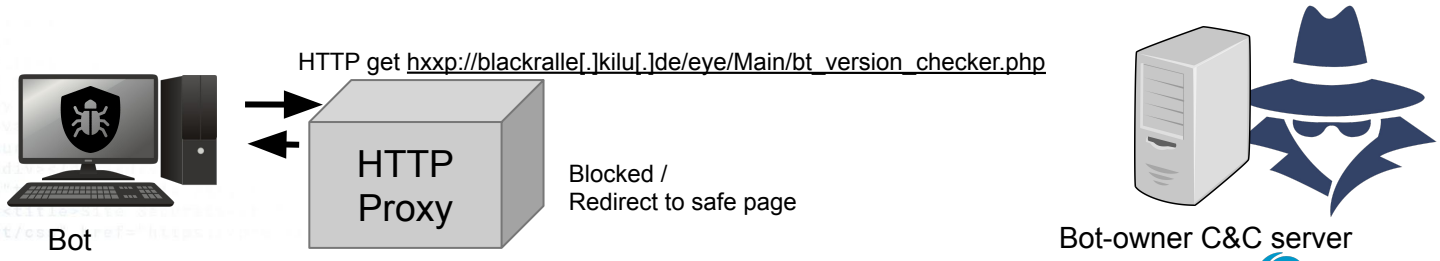




Motivation

- Design a system to:
  - Identify known and recurring bot C&C URL patterns
  - Characterize the patterns in efficient regular expressions to be deployable on a high performance HTTP proxy

- Eventually, the regular expressions will be used to block new bot C&C communication whose URL match the regular expressions



Challenges

- Scalability of producing regular expressions:
  - Our collected datasets has >150k C&C URLs that are updated hourly
  - Manually producing regular expressions of URLs is unreasonable

- Sensitivity vs. specificity:
  - Specificity: Too accurate regex will never generalize to new C&C URLs
  - Sensitivity: Incorrect regex might block legitimate URLs

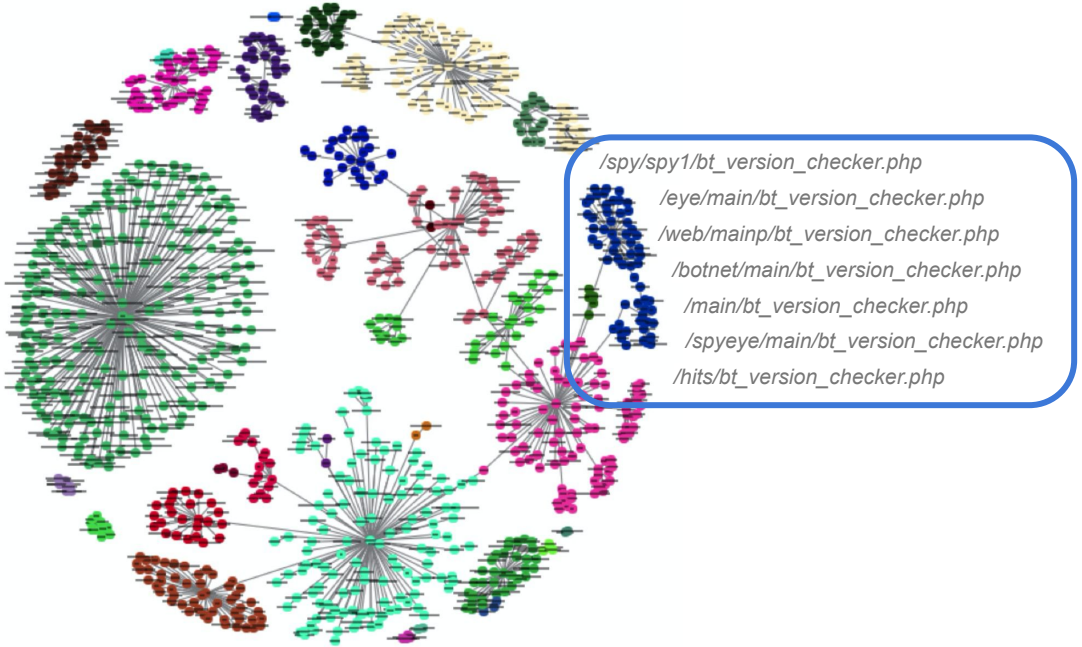
- Performance:
  - Ideally, we wish to block C&C URLs inline on the HTTP proxy
  - Pre-compiled regex (using Hyperscan) are acceptable, but more complex solutions are not (e.g., neural networks)

System Overview

Akamai

A C&C URL Language Model

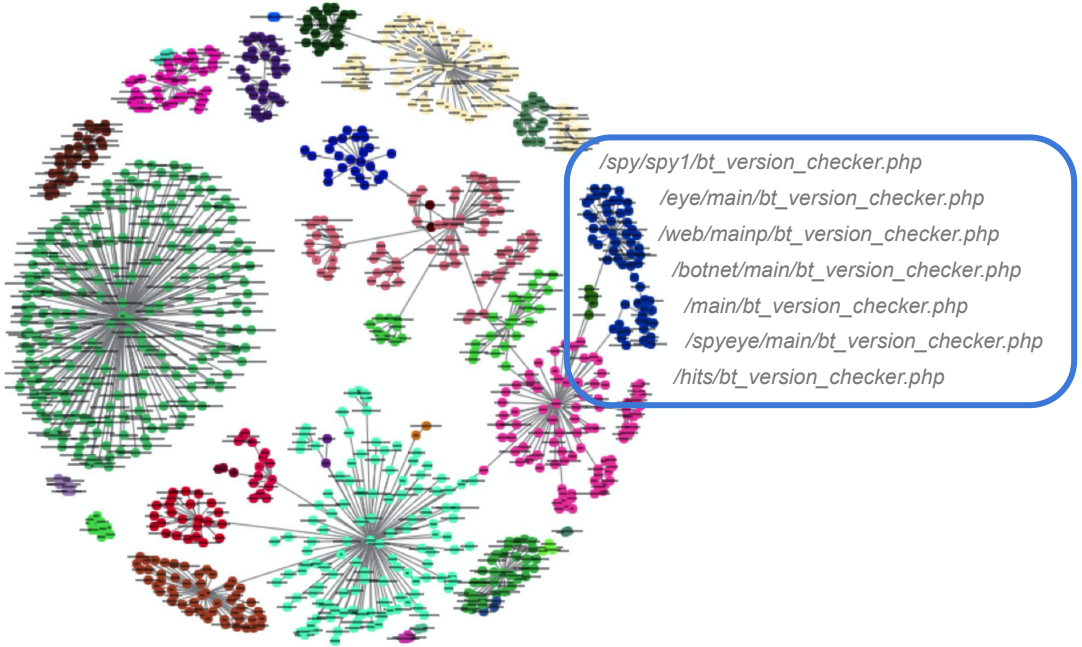
- Construct a language model of bot C&C URL paths
- Cluster similar paths based on their proximity within the model



 |

=> Naive Regular Expressions

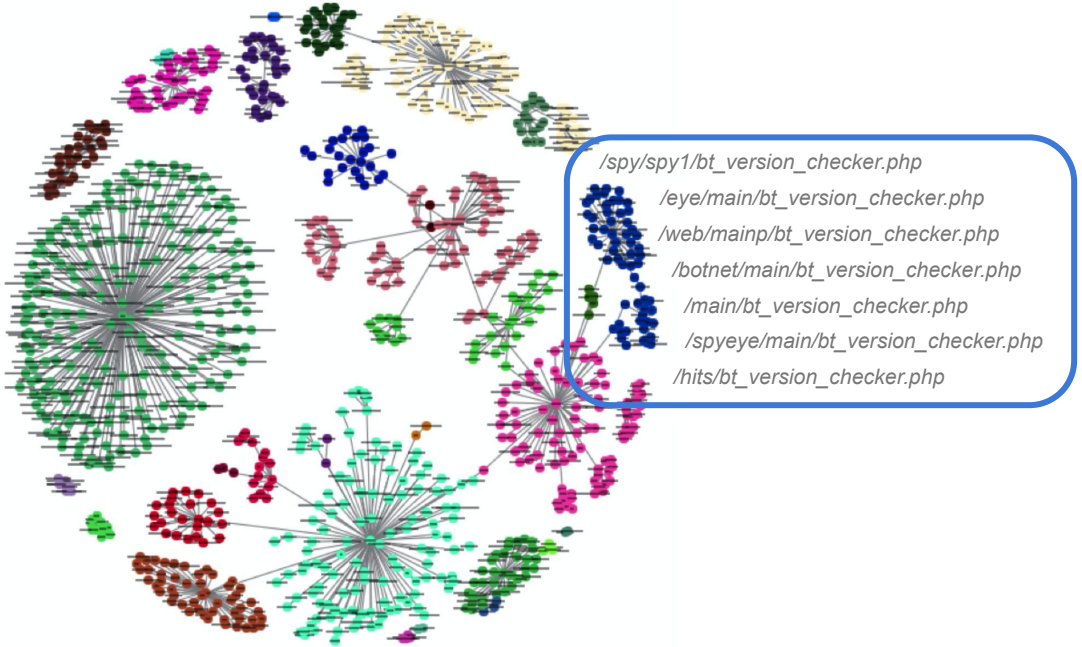
- For every cluster, output a naive regex (i.e., OR cond.)
- Observation: the naive regex is too specific



(“/spy/spy1/bt_version_checker.php”|
“/eye/main/bt_version_checker.php”|
“/web/mainp/bt_version_checker.php”|
“/botnet/main/bt_version_checker.php”|
“/main/bt_version_checker.php”|
“/spyeye/main/bt_version_checker.php”|
“/hits/bt_version_checker.php”)

=> Increasing Regex Sensitivity

- Use Bartoli et al. [3-6] genetic algorithm to shorten the regex while still matching all of the input C2 URL paths



/spy/spy1/bt_version_checker.php
/eye/main/bt_version_checker.php
/web/mainp/bt_version_checker.php
/botnet/main/bt_version_checker.php
/main/bt_version_checker.php
/spyeye/main/bt_version_checker.php
/hits/bt_version_checker.php

("/spy/spy1/bt_version_checker.php"|
"/eye/main/bt_version_checker.php"|
"/web/mainp/bt_version_checker.php"|
"/botnet/main/bt_version_checker.php"|
"/main/bt_version_checker.php"|
"/spyeye/main/bt_version_checker.php"|
"/hits/bt_version_checker.php")

(?=(([^_](?=(\w+))\3)+))\1

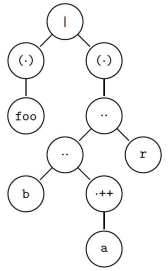
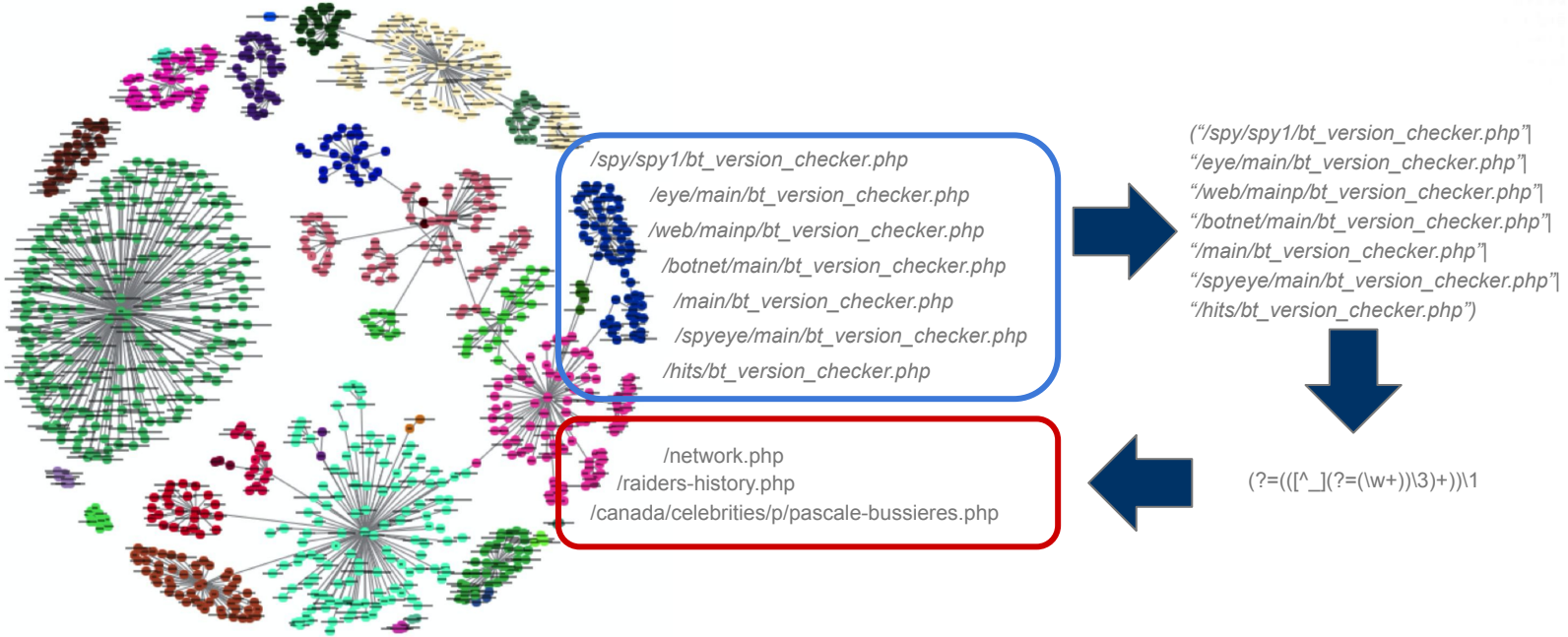


Figure 1: Tree representation of the regular expression (foo)|(ba++r).

 |

=> Increase Regex Specificity

- Penalize for matching on benign URLs
- Solve iteratively, until there are no benign URL path matches



 |

Analysis & Takeaways

Akamai

Analysis & Takeaways

- **Distance between URL paths**
- **Clustering**
- **Automatic Regex generation**
- **Experimental Setup**
- **Results**

Constructing a bot C&C language model

- Key idea: propose **a pairwise distance measure** between C&C URLs paths (strings) to allow grouping of similar URLs

hxxp://pr0ghz[.]dsdo-services[.]cz[.]cc/Main/bt_version_checker.php ← *Path*

	Path 1	...	Path N
Path 1	0	...	...
...	...	0	...
Path N	....	...	0

- Proposal: the Smith-Waterman algorithm [2] (see next slide)

Smith-Waterman vs. Edit Distance (Levenshtein)

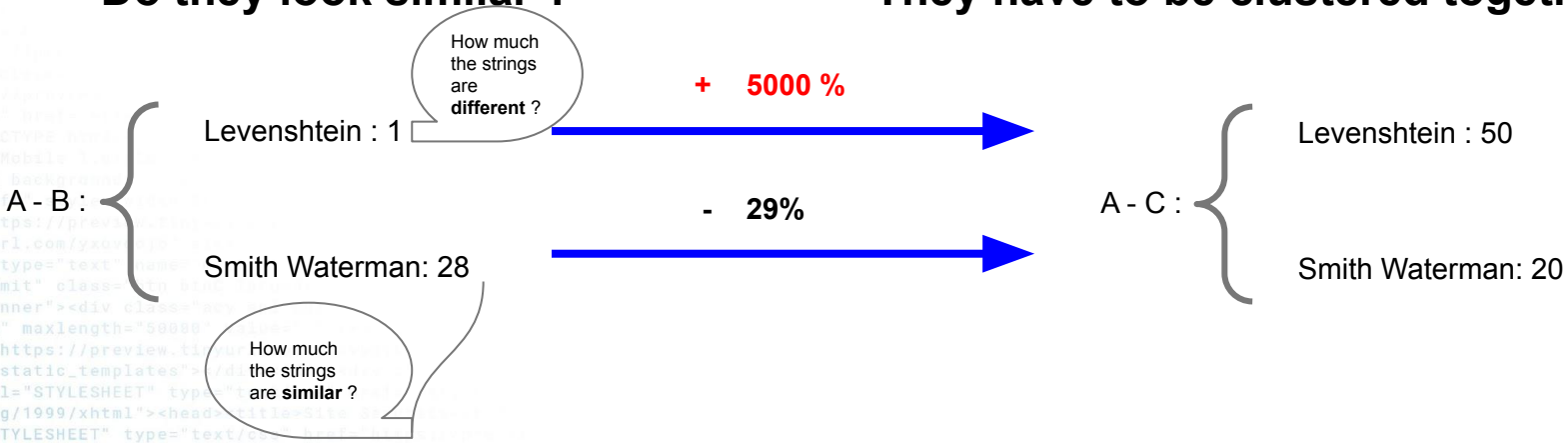
/malware/givemeyourmoney.php **(A)**

/Malware/givemeyourmoney.php **(B)**

/malware/nowijustcreatedafolderuselessinmywebsiteforthefun/givemeyourmoney.php **(C)**

Do they look similar ?

They have to be clustered together



 |

The Smith-Waterman Distance Measure

Look for the best alignment between 2 sequences

Originates in computational biology to find common patterns between 2 nucleotides sequences.

```
A C T T G T C T T
A C T _ G _ _ T T
```

Match	1
Unmatch	-1
Gap penalty	-1

/malware/nowijustcreatedafolderuselessinmywebsiteforthefun/givemeyourmoney.php

/malware----------/givemeyourmoney.php

match len = 8 *gap len = 50* *match len = 20*

Edit distance = 50

SW score = 20

Normalization and transformation to edit distance

Issues

- SW can return any number between 0 and N with N the max length of the path in our dataset
- Big strings can get bigger scores
- Clustering algorithm works with edit distance not with similarity distance

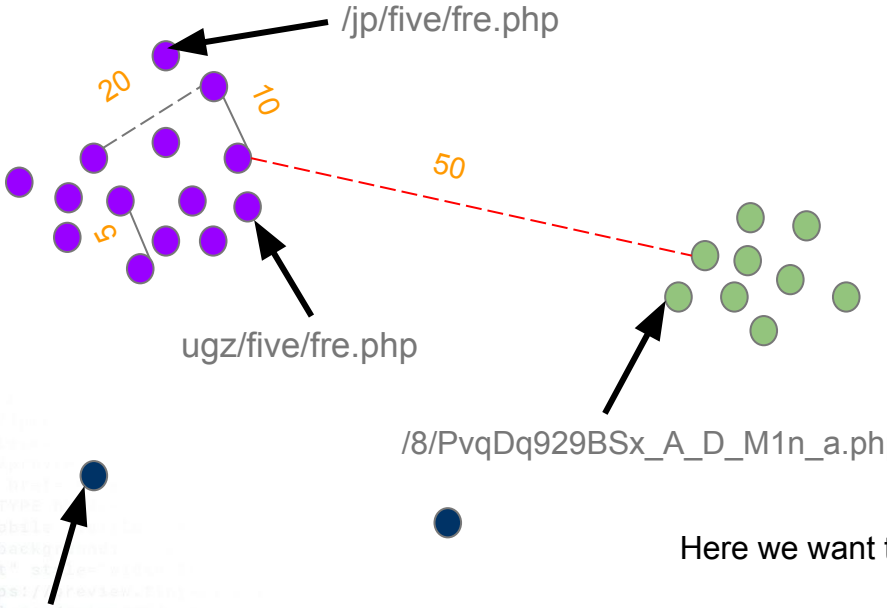
Normalization

$$similarityScore(path_i, path_j) = 100 * \frac{sw_{ij}}{max(len(path_i), len(path_j))}$$

Similarity to edit distance

$$editScore = |100 - similarityScore|$$

Clustering with DBScan



Epsilon	Minimal distance between two points to be in the same cluster
MinPoints	Minimal number of points to create a cluster
Metric	‘precomputed’ = SW !

Why DBSCAN ?

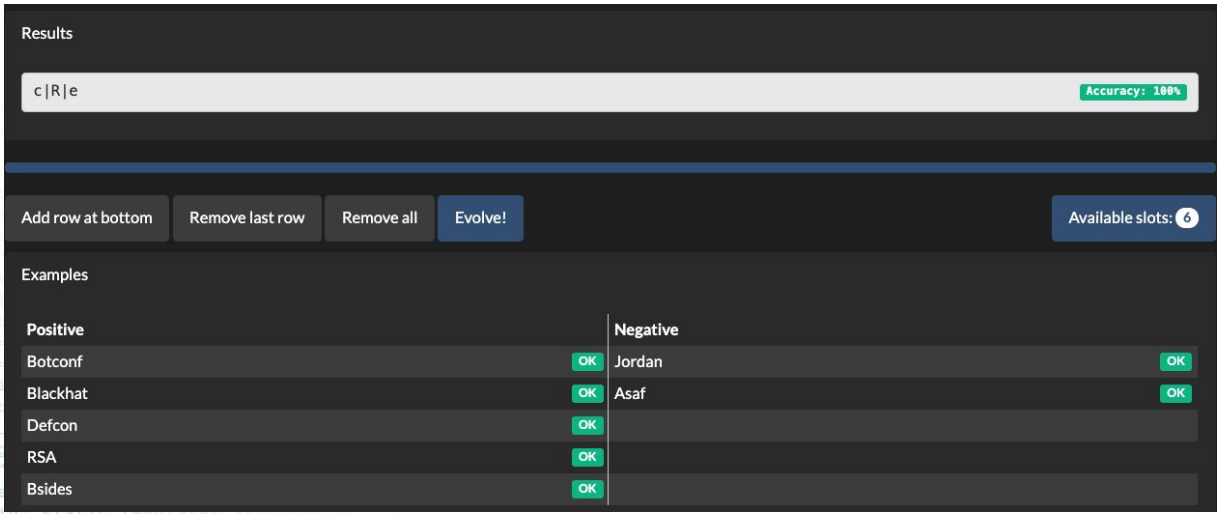
Here we want to choose Epsilon and MinPoints because we understand them !

How to choose the parameters ?

Since we really understand the parameters, we can choose them without an algorithm.
Epsilon 20 = Group paths that are 80% similar !

Now we have cluster of paths…,

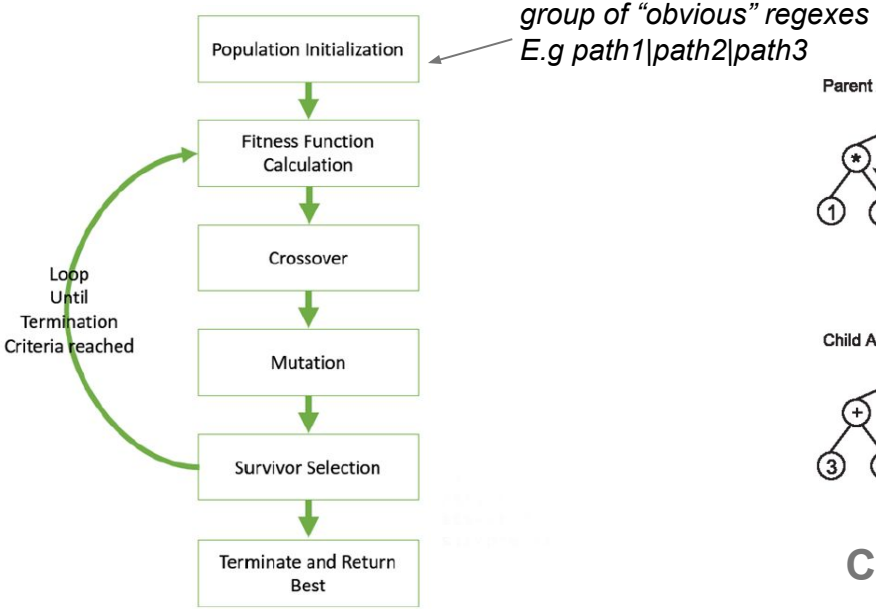
let’s play Regex Golf !



http://regex.inginf.units.it/golf/

From clusters to regular expressions [3-6]

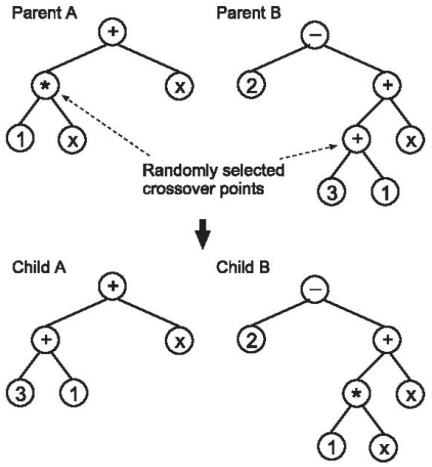
- Genetic Programming is a branch of evolutionary algorithm
- Based on the idea of evolution of a population



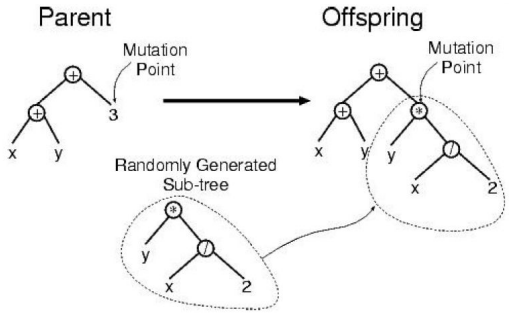
group of "obvious" regexes
E.g path1|path2|path3

- **Goal**: Find an optimal solution to a function
- A difference with a classic reinforcement learning?

  It is a reinforcement learning method but it **is not** using gradient descent for example



Crossover



Mutation

From clusters to regular expressions [3-6]

AST representation

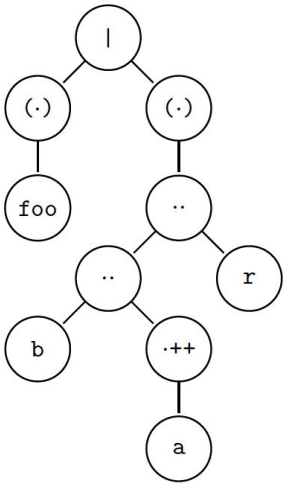


Figure 1: Tree representation of the regular expression `(foo)|(ba++r)`.

Leaf in {alphabets, range, \w, \d, .)

Branch in { possessive quantifiers, non capturing group, concatenator…}

2 fitness functions :

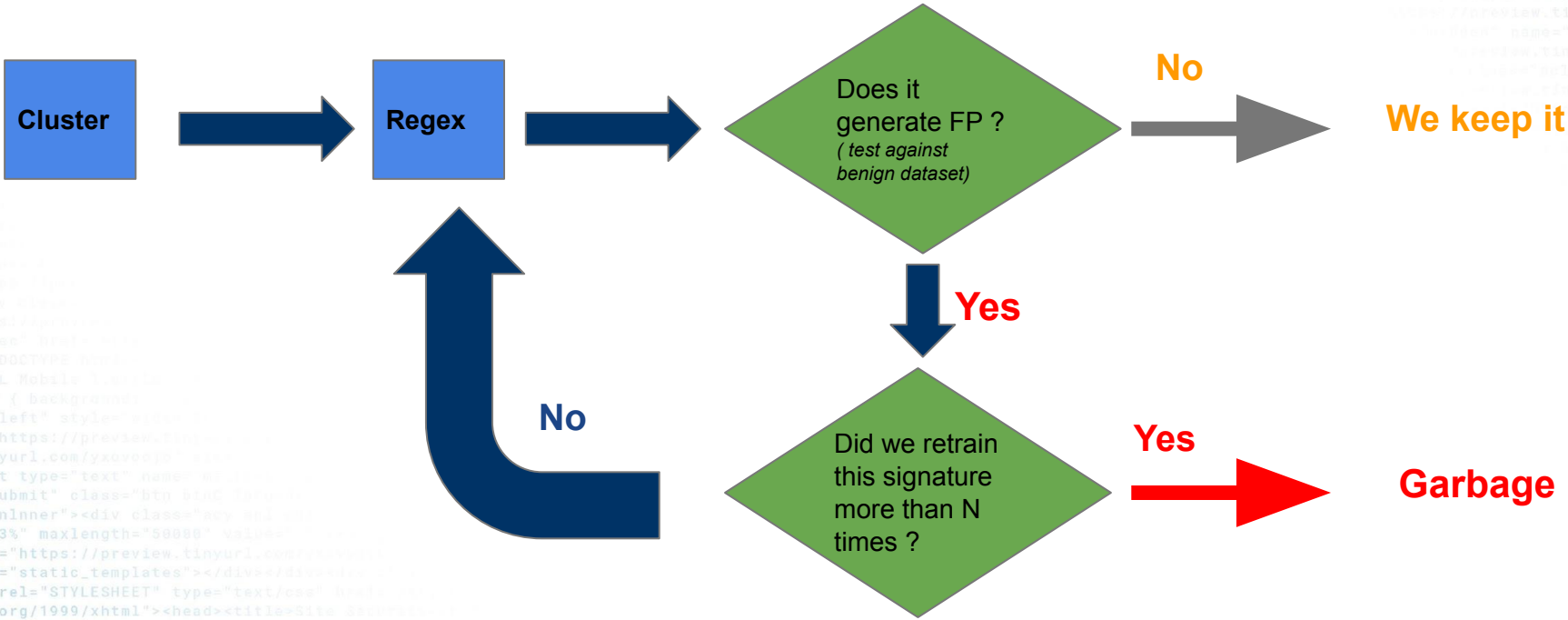
1. Sum of levenshtein distance between detected string and desired string

2. Len of regex

Default value

Parameters	**Value**
Number of tree by string	4
Number of iterations	1000

From clusters to regular expressions



 |

Experimental Setup

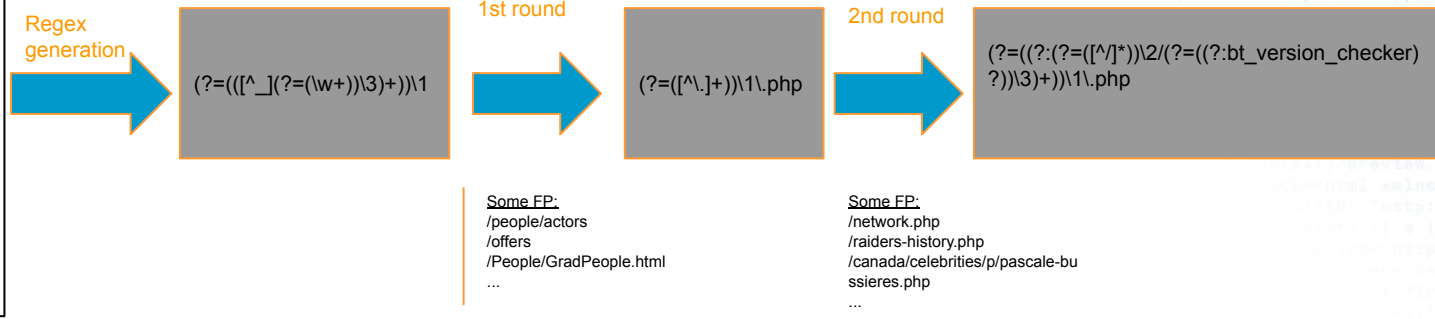
Experimental Setup

- AWS R5.24xl instance: 96 vCPU, 768 GiB memory
- Datasets: 1.4M benign and bot C&C URLs (see below)

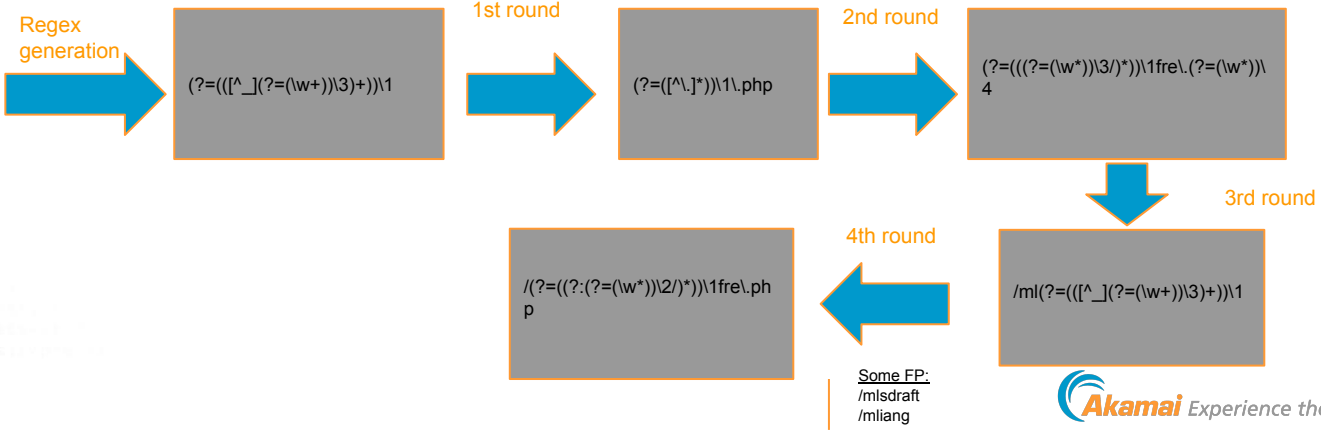
C&C / Benign	Dataset	# Urls	# Domain	# Paths
C&C	A collection of 3rd party threat lists	162k	69k	**106k**
Benign	Teseract URL Dataset	296k	69k	257k
Benign	ICSX Dataset, Uni. of New Brunswick	35k	0.3k	30k
Benign	Alexa Top 200k URLs	N/A	N/A	59
Benign	Akamai HTTP traffic (cleaned*)	N/A	N/A	996
-	**Total**	**>331k**	**>69.3k**	**1.4 M**

Compiling Regexes - Example !

/spy/spy1/bt_version_checker.php
/spyeye/main/bt_version_checker.php
/main/bt_version_checker.php
/botnet/main/bt_version_checker.php
/eye/main/bt_version_checker.php
/men/main/main/bt_version_checker.php
/hits/bt_version_checker.php
/root/main/bt_version_checker.php
/web/mainp/bt_version_checker.php
/panel/bt_version_checker.php
…
53 samples

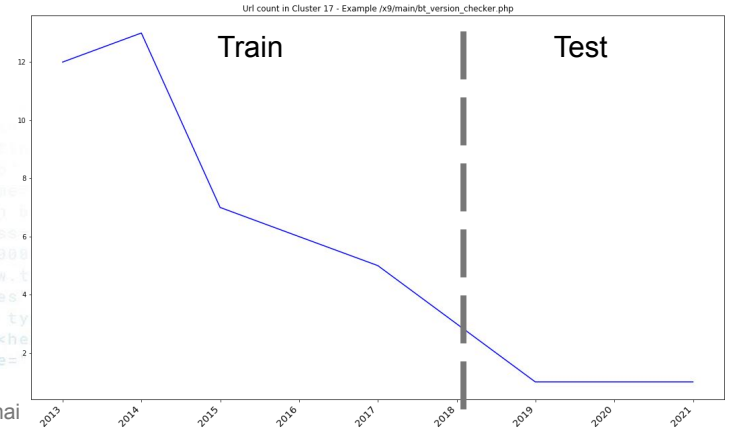


ml/vrs/peta/3/lok/panel/fre.php
/ml/vrs/slyn3/lky/panel/fre.php
/vrs/ptc/lok/panel/fre.php
/ml/vrs/sly/n5/lok/panel/fre.php
/ml/vrs/ptb/lok/panel/fre.php
/ml/vrs/sly/5/lok/panel/fre.php
/ml/vrs/peta/lok/panel/fre.php
/ml/vrs/sly/lok/panel/fre.php
/ml/vrs/sly/n1/lok/panel/fre.php
/ml/vrs/sly/n2/lok/panel/fre.php
/ml/vrs/pn2/lok/panel/fre.php
…
39 samples



Validation and results

- 34 regular expressions, 1.3k newly detected C&C URLs
  - >10.2k VT URL detections
  - >12 URL patterns results in 96.3% accuracy
- Time-based cross validation
  - Train: C&C URLs inserted before 2018
  - Test: C&C URLs inserted since 2018



Regex ID	# URLs (Train)	# URLs (Test)	% Caught
2	74	245	245 (100%)
8	45	59	59 (100%)
10	32	119	119 (100%)
17	47	2	2 (100%)
14	3	36	28 (77.78 %)
3	2	81	0 (0%)
18	12	27	26 (96.3 %)
0	28	4	4 (100%)

Cross-validation results by cutting in 2018

Conclusions & Future Work

- Using a genetic algorithm and clustering (Smith Waterman on DBScan) , we were able to construct **34** regexes that identify **1.3k** new C2 URLs

- For every cluster, training on 12 URLs with a similar pattern was sufficient to identify other URLs with 96.3% accuracy

Future work

- Cluster regexes from the same C2 families together

- Release Python - Java* source code

** Regex generation source code is in Java*
https://github.com/MaLeLabTs/RegexGenerator

References

[1] Sahoo, Doyen, Chenghao Liu, and Steven CH Hoi. "Malicious URL detection using machine learning: A survey." *arXiv preprint arXiv:1701.07179* (2017).

[2] Smith, Temple F., and Michael S. Waterman. "Identification of common molecular subsequences." *Journal of molecular biology* 147.1 (1981): 195-197.

[3] Bartoli, Alberto, et al. "Playing regex golf with genetic programming." *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. 2014.

[4] Bartoli, Alberto, et al. "Automatic generation of regular expressions from examples with genetic programming." *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*. 2012.

[5] Bartoli, Alberto, et al. "Regex-based entity extraction with active learning and genetic programming." *ACM SIGAPP Applied Computing Review* 16.2 (2016): 7-15.

[6] Bartoli, Alberto, et al. "Learning text patterns using separate-and-conquer genetic programming." *European Conference on Genetic Programming*. Springer, Cham, 2015.

[7] Wang, Xiang, et al. "Hyperscan: a fast multi-pattern regex matcher for modern cpus." *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*. 2019.

Thank you

Q&A