Gen™

GenRex Demonstration:

Level Up Your Regex Game

Nice, 14 April 2024
Dominika Regéciová
@regeciovad

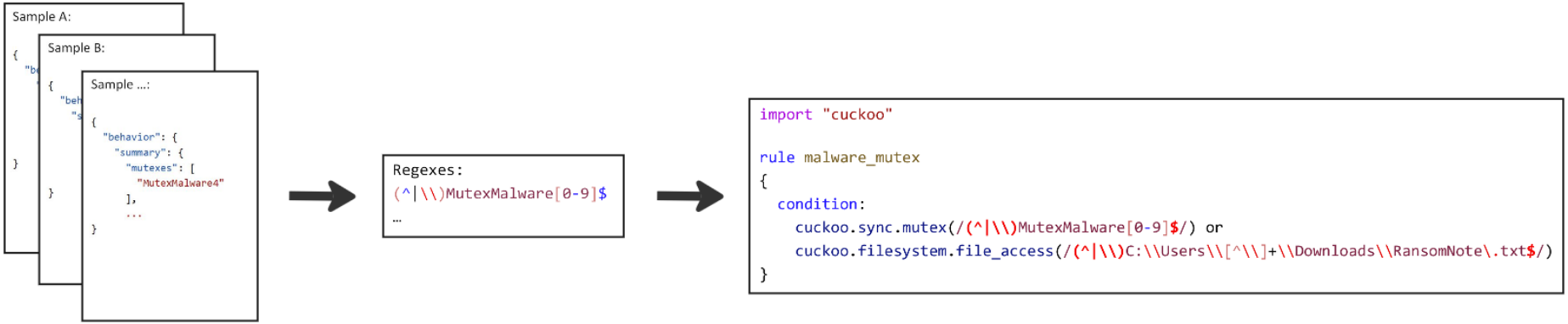
norton Avast LifeLock by norton Avira AVG Reputation Defender by norton CCleaner

A Few Notes About Me

- Pronouns: she/her/hers
- Senior Researcher at Gen
- Projects with ESA and Czech Police
- My research:
  - Formal models and languages in security
  - Pattern matching
  - Blockchain technology

Motivation

- A tool for generalization of named objects (such as mutexes) and much more
- GenRex was presented at IEEE TrustCom 2023
- [GenRex: Leveraging Regular Expressions for Dynamic Malware Detection](#) (postprint)



GenRex as an Open-Source Project

- We published GenRex as an open-source project in February this year
- [Know Your YARA Rules Series: #6 We Present GenRex – A Generator of Regular Expressions]

Dynamic analysis

- Detection rules based on artifacts
  - Named objects (mutexes, semaphores, etc.)
  - Events (like creating a file, deleting a register, etc.)
- The naming of the artifacts is often based on algorithms
- Regular expressions for precise detection of current and future variants of the malware family

Variability of Named Objects

Sample A:

```
{
  "behavior": {
    "summary": {
      "mutexes": [
        "MutexMalware3"
      ],
      ...
}
```

Sample B:

```
{
  "behavior": {
    "summary": {
      "mutexes": [
        "MutexMalware9"
      ],
      ...
}
```

Sample C:

```
{
  "behavior": {
    "summary": {
      "mutexes": [
        "MutexMalware4"
      ],
      ...
}
```

YARA Rules

- [YARA Cuckoo module documentation]
- **A rule can describe both static and dynamic characteristics**

```
import "cuckoo"

rule malware_mutex
{
    condition:
        cuckoo.sync.mutex(/(^|\\)MutexMalware[0-9]$/)
}
```

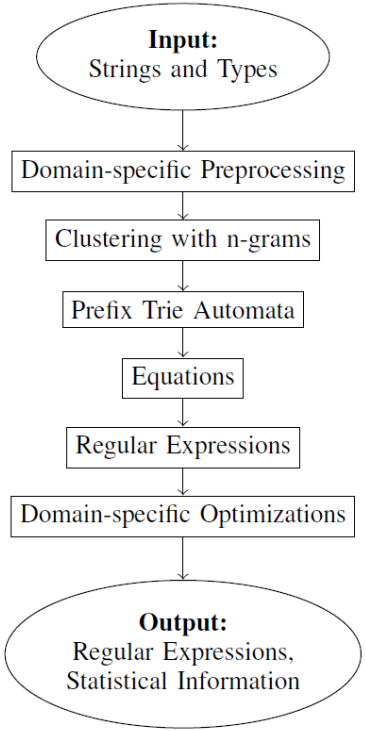
```
$ ./yara -x cuckoo=behavior_report.json my_rules.yar sample_file
```

YARA Rules

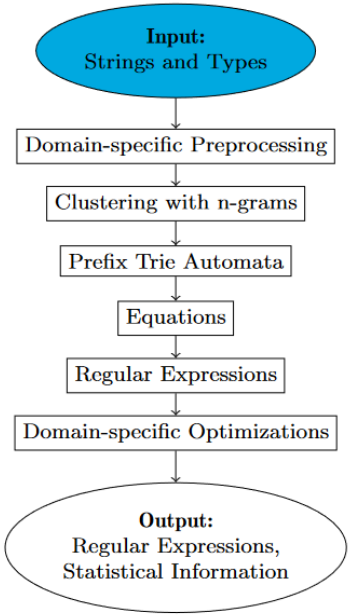
- [My extension to the Cuckoo module]
- Added functions and number of matched strings

```
import "cuckoo"

rule test_cuckoo
{
    condition:
    cuckoo.genrex.api_call(/WNetGetProviderNameW/) >= 3 or
    cuckoo.genrex.atom(/rOBDoI/) >= 3 or
    cuckoo.filesystem.file_access(/(^|\\)C:\\Users\\[^\\]+/) >= 12 or
    cuckoo.registry.key_access(/(^|\\)Software\\Downloader/) or
    cuckoo.sync.mutex(/kzyyjqyi/) >= 1 or
    cuckoo.genrex.resolved_api(/iertutil.dll!#16/) >= 3 or
    cuckoo.genrex.semaphore(/LJpExtC8rffiNYPa94/) >= 2
}
```

Model Architecture for GenRex



GenRex: Input and Types

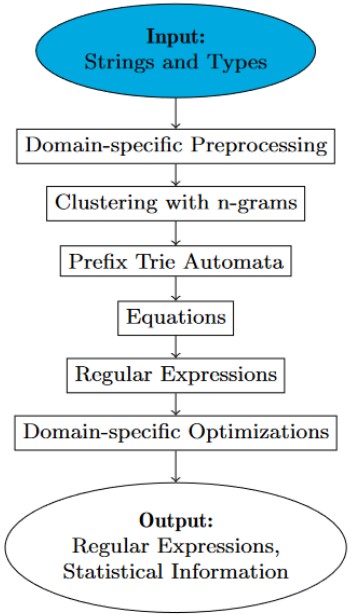


The input is defined as:

- Lists of strings and their source
- Input type (mutexes, files, registers,…)

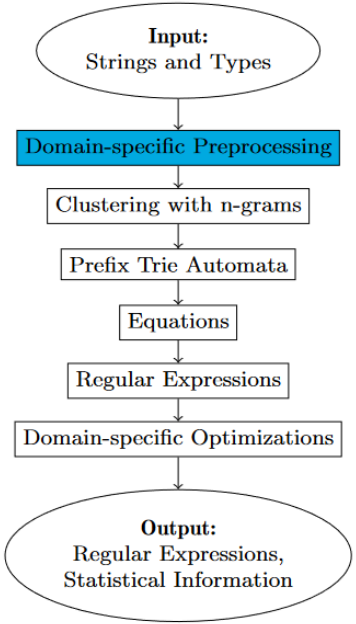
```
{
  "source": {
    “hash1": ["aabcmalware7992", "adeemalware3022", "aefdmalware1896"],
    “hash2": ["bfbcmalware5996", "bbcamalware4508"]
  },
  "input_type": "mutex"
}
```

GenRex: Input and Types



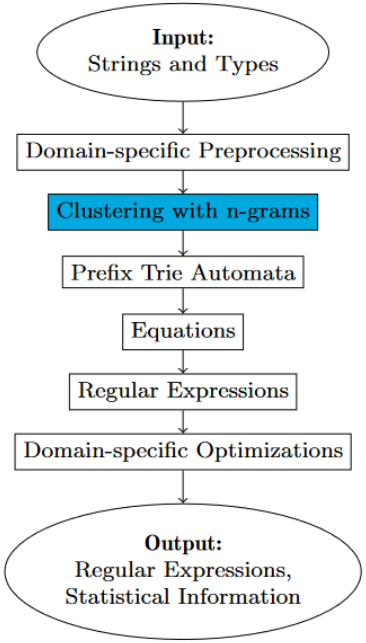

```
import genrex

results = genrex.generate(
  input_type=genrex.InputType.MUTEX,
  source={
    “hash1": ["aabcmalware7992", "adeemalware3022","aefdmalware1896"],
    “hash2": ["bfbcmalware5996", "bbcamalware4508"],
  })

print("Results:")
for result in results:
  print(result)
```


GenRex: Domain-specific Preprocessing



- **Domain-specific Preprocessing:**
  - Replacing a hexadecimal format \xHH as one character
  - Removing strings containing only GUID
  - Removing prefixes such as HKEY_LOCAL_MACHINE, Global\, C:\Program Files, etc.
  - Replacing user names (as Susan) for [^\\]+

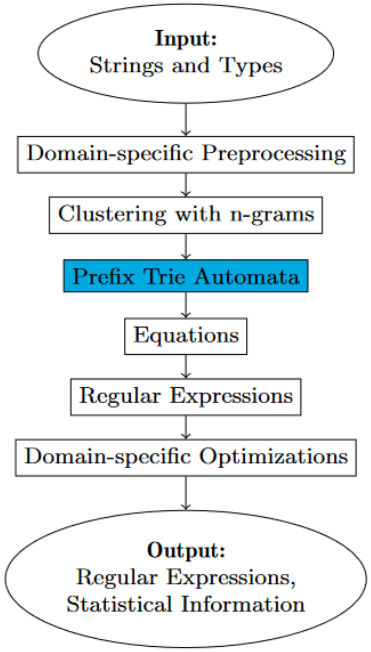
GenRex: Clustering Phase



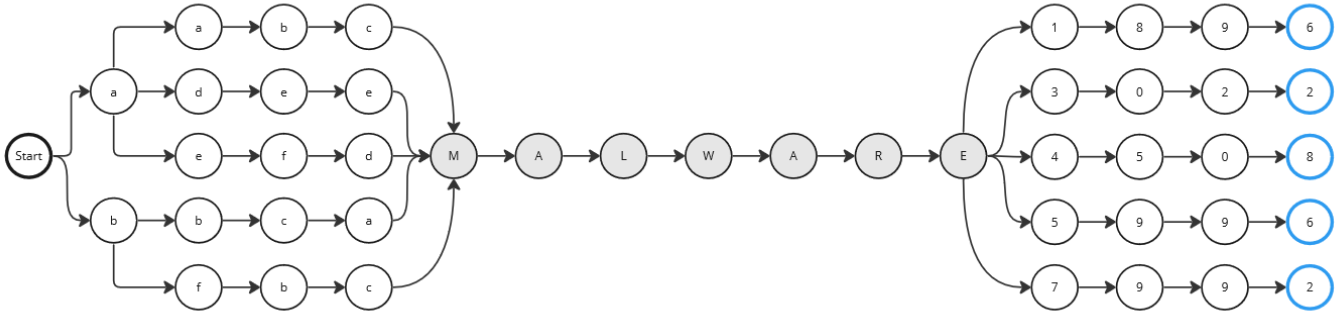
- The clustering based on n-grams
- The length of n-gram is calculated based on input strings and type
- Statistical information

```
"ngram": "malware",
"unique": 5,
"min": 2,
"max": 3,
"average": 2.5,
"resources": ["aabcmalware7992", "adeemalware3022", "aefdmalware1896",
              "bbcamalware4508", "bfbcmalware5996"],
```

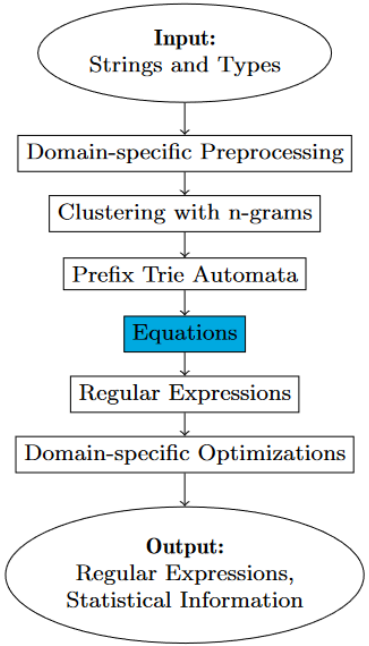

GenRex: Prefix Trie Automata



- **Updated algorithm for prefix trie creation:**
  - A small prefix trie just with one n-gram
  - The rest of the strings are added while detecting the n-gram as a common part
  - The trie is then minimized



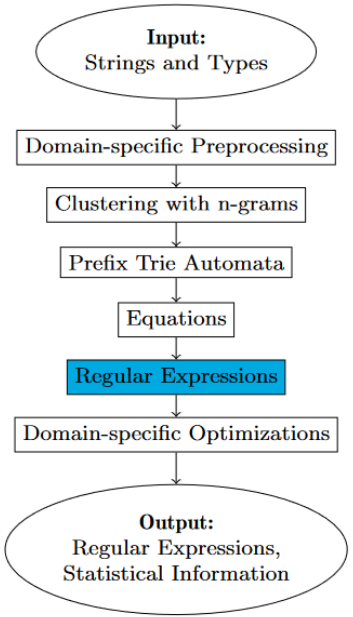
GenRex: Equations



- **Two matrices:**
  - Matrix A for all states in the trie
  - Matrix B for the final states
- **The simplified Brzozowski algebraic method**

```
for n = number_of_states decreasing to 1:
  for i = 1 to n:
    B[i] += A[i,n] . B[n]
    for j = 1 to n:
      A[i,j] += A[i,n] . A[n,j]
```

GenRex: Creation of Regular Expressions



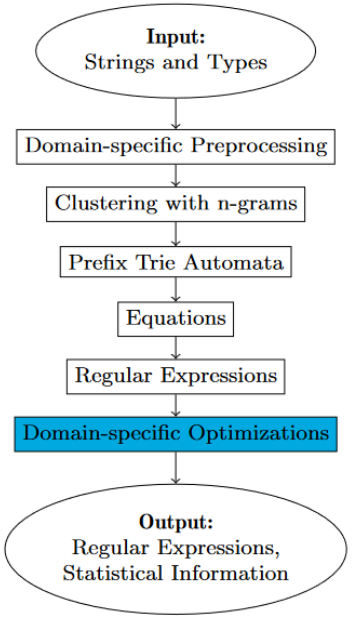
- **By solving the set of equations, we are creating one regular expression for each cluster**

```
["m"] . ["alware"] => "malware"
["aabc", "adee"] + ["aefd"] => ["aabc", "adee", "aefd"]
```

- **As a result, we have an internal representation of the regular expressions**

```
(aabc|adee|...)malware(1896|3022|...)
```


GenRex: Domain-specific Optimizations



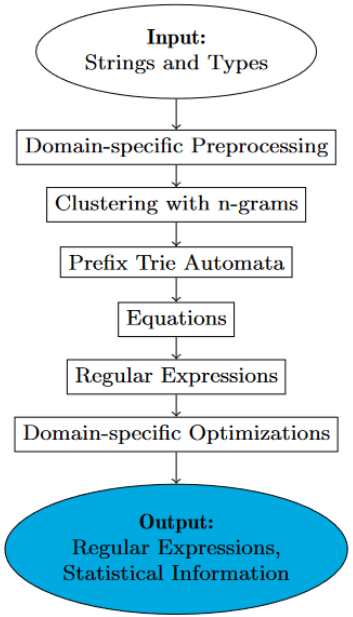
- **We aim to produce results that would:**
  - be readable
  - efficient for pattern matching
  - strict enough not to detect false positives but also generalized to match possible future variants

```
["aabc", "adee", "aefd", "bbca", "bfbc"] => "[0-9a-f]{4}"
["1896", "3022", "4508", "5996", "7992"] => "[0-9]{4}“
```

- **This step also adds the prefix (^|\\) and suffix $ to each regular expression representing named objects**

```
"[0-9a-f]{4}malware[0-9]{4}" => "(^|\\)[0-9a-f]{4}malware[0-9]{4}$"
```


GenRex: Output




```
{ "results": [
    {
      "regex": "(^|\\)[0-9a-f]{4}malware[0-9]{4}$",
      "ngram": "malware",
      "unique": 5,
      "min": 2,
      "max": 3,
      "average": 2.5,
      "resources": ["aabcmalware7992", "adeemalware3022",
        "aefdmalware1896","bbcamalware4508", "bfbcmalware5996"],
      "input_type": "mutex",
      "hashes": [“hash1", “hash2"]
    }
  ]
}
```

Experiments: Dataset

- **Goal: create a YARA rules and evaluate their precision**
- **The dataset contains 54,045 behavioral reports from the CAPEv2 sandbox**
  - 48,976 reports from 10 classified malware families
  - 5,078 reports from cleanware samples
- **Examples of trojans, worms, spyware, and bots**
- **Used artifacts: atoms, files, keys, mutexes, semaphores, API calls, and resolved APIs**
- **The dataset was split into two parts**
  - Pre-known dataset for YARA rules generation
  - Testing dataset

Experiments: How to Create YARA Rules

- **YARA in version 4.3.2 with additional changes to the code**
- **Artmin (Artifacts Minimizer)**
  - Filtration of clean strings, created by the sandbox/emulator
  - The most straightforward implementation – run malware and clean samples and select strings that are in every report
- **The preparation**
  - Select the first 100 samples with SSDeep hash similarity < 50
  - Create an empty YARA rule

Experiments: How to Create YARA Rules

- **Repeat until we cover 100% of the pre-known samples, or we cannot add anything more to the YARA rule**
  - Filter the clean strings
  - Use GenRex for each named object to create regular expressions
  - Select RE with the highest coverage and not false positives
  - Test the current rule on the pre-known dataset
  - Use the uncovered samples for next loop

Experiments: Example of Qakbot YARA Rule

```
import "cuckoo"

rule Qakbot
{
  condition:
    cuckoo.filesystem.file_access(/(^|\\)C:\\Users\\[^\\]+\\(AppData\\Local\\Temp\\~)?onayjeqh\.(tmp|wpl)$/) >= 1 or
    cuckoo.filesystem.file_access(/(^|\\)C:\\INTERNAL\\__empty$/) >= 1 or
    cuckoo.filesystem.file_access(/(^|\\)C:\\ProgramData\\FilesystemMonitor(\\fsmonitor\.(dll|ini))?$/) >= 3 or
    cuckoo.filesystem.file_access(/(^|\\)C:\\ProgramData\\FilesystemMonitor\\fsmonitor\.(dll|ini)$/) >= 2 or
    cuckoo.filesystem.file_access(/(^|\\)C:\\ProgramData\\FilesystemMonitor\\fsmonitor\.dll\.(12[34]\.M|2\.M|m)anifest$/) >= 4 or
    cuckoo.registry.key_access(/(^|\\)SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\ProfileList\\S-1-5-21-962146886-2531488156-1783242634-50[01]$/) >= 2 or
    cuckoo.registry.key_access(/(^|\\)Volatile Environment\\USER(DOMAIN|NAME)$/) >= 2 or
    cuckoo.sync.mutex(/(^|\\)oouoma$/) >= 1 or
    cuckoo.genrex.resolved_api(/msvcrt\.dll!_HUGE/) >= 1
}
```

The resulting YARA rule for the Qakbot family.

Experiments: Evaluation

- **To evaluate each YARA rule, we estimated:**
  - True positives – a number of correctly categorized samples
  - True positive rate – a percentage within the family of correctly categorized samples
  - False positives – a number of falsely matched samples
  - False positives rate - a percentage outside the family
- **The overall true positive rate is 92.34% and the false positive rate is 0.01%**

Family	TP	TPR (%)	FP	FPR (%)
Adload	665	94.33	0	0
Emotet	13,939	96.63	0	0
HarHar	655	100	0	0
Lokibot	3,700	88.28	2	0.004
njRAT	2,460	73.04	2	0.003
Qakbot	4,857	99.00	0	0
Swisyn	12,571	99.83	1	0.002
Trickbot	3,166	75.33	1	0.002
Ursnif	767	57.41	0	0
Zeus	2,444	94.22	1	0.001
All families	**45,224**	**92.34**	**7**	**0.01**

Demo Time!

- Examples will be published in the paper

Resources

- Public GenRex GitHub page
- Blogpost about GenRex
- GenRex demo GitHub page
- GenRex: Leveraging Regular Expressions for Dynamic Malware Detection

THANK YOU FOR YOUR ATTENTION!

