Harvesting Artifacts

Improving Useful Data Extraction from Cybersecurity Incident Reports

Matt Sisk, Robin Ruefle, Sam Perl

June 16, 2017

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This material is based upon work funded and supported by the Department of Homeland Security under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

DM17-0209

Harvesting Artifacts

- Background & Data
- Approach
- Challenges & Solutions
- Testing & Measurement
- Summary of Results & Future Work

Harvesting Artifacts

Background & Data

Background

- US-CERT receives incident reports from a diverse constituency.
- Each ticket is an observation of problematic activity by a particular reporter.
- Tickets vary in content, context, and in the types of data it contains.
  - Sometimes a ticket describes what actions were taken and a chain of communication over a period of time.
- Our project attempts to find usable data that is ‘locked’ inside of the workflow system.
  - Many tickets contain indicators and the context of how they were found, what threats tried to do, etc.

US-CERT “Information Discovery” Project Tasking goals include:

- Trending, Exploration, Automation, Data Mining

Project Description and Goals

We are working on solutions to analyze this data set at scale. In some samples we observed technical data inside of forms, or narrative descriptions, or cut/paste snippets from other places.

We were extracting information from the reports using regular expressions, but could we improve our results?

Regex Project Goals

- Make existing extraction methods (regex) more readable and manageable
- Possibly identify more useful information in the reports to extract
  - Ideally actionable or usable in situational awareness
- Have an ability to measure our improvement (if any)
- In summary: extract current types more accurately and begin to extract new types

Harvesting Artifacts

Approach

Approach

Ground Truth Testing

- Sampled 50 reports that are ‘rich’ with observables we want to automatically extract.

Selection

- Started by searching for ‘typical’ records
  - Artifact dense, both “normal” and edge cases
  - Arduous process!
- Manual parsing allowed for testing for true positives, false positives, and false negatives as the regex library was expanded and refined

Approach

Solution for Edge Cases – Bulk Query Tool

- Queries against the entire corpus
- Random order search for experimentation and finding edge cases
- Allowed us to write ‘looser’ regular expressions for exploratory purposes.
  - Queries returned many more records including many false positives
  - BUT also lots of valuable edge cases!
- "Interesting" reports added to Ground Truth testing suite

Harvesting Artifacts

Challenges & Our Solutions

Challenge 1: New Category Types!

- In our sample, we found many new categories that we might be able to extract with regular expressions
  - TLDs, Malware Names, Attack Categories and Behaviors
  - Countries and Adjectivals
  - ISPs, ASNs, CVEs

- TLD: .pineapple .biz .etc
- Malware: Trojan.Win32.VBKrypt.ovip, Win32.Fareit.A/Zeus
- Attacks: C2, CnC, C&C, command and control
- Country: Switzerland, Swiss

Solution for Category Types

- Built a regex tool that autogenerates optimized expressions from a list of raw tokens (as opposed to naive '|' constructs)
- Based on a 'trie' (prefix tree) data structure
- Autogeneration yielded far greater performance metrics vs traditional baseline regex construction
- 40% more efficient

Autogeneration Summary

- Start with a list of tokens, for example:

```
dog
dingo
cat
doggo
```

- Naive regex:

```
(dingo|cat|doggo|dog)
```

- Autogenerated and optimized:

```
(cat|(d(og(go)?|ingo)))
```

- Can be done manually if you're good with regexes
- But not so easily when the list comprises hundreds of tokens

Autogeneration Example

```
primitives["tld"] = """
(?:(?:a(?:c(?:c(?:ountants?|enture)|t(?:ive|or)|ademy|o)?|l(?:i(?:baba|pay)|l(?:finanz|y)|sace)?|b(?
:b(?:ott|vie)?|udhabi|ogado)|u(?:t(?:hor|os?)|ction|dio?)?|n(?:alytics|droid|quan)|r(?:amco|chi|my|
pa|te)?|i(?:r(?:force|tel)|g)?|p(?:artments|p(?:le)?)|m(?:sterdam|ica)?|s(?:sociates|ia)?|g(?:akhan|
ency)?|d(?:ult|ac|s)?|q(?:uarelle)?|t(?:torney)?|e(?:ro|g)?|a(?:rp|a)|z(?:ure)?|vianca|fl?|kdn|ws?|x
a?|o))|(?:b(?:a(?:r(?:c(?:lay(?:card|s)|elona)|efoot|gains)?|n(?:d|k)|uhaus|yern|idu|by)?|o(?:s(?:tik|
ch)|o(?:ts|k)?|ehringer|utique|ats|nd|m|t)?|r(?:o(?:adway|ther|ker)|idgestone|adesco|ussels)?|u(?:
ild(?:ers)?|dapest|siness|gatti|zz|y)|l(?:ack(?:friday)?|oomberg|ue)|e(?:ntley|rlin|ats|er|st|t)?|i(?:ng
o?|ble|ke|d|o|z)?|n(?:pparibas|l)?|b(?:va|c)?|h(?:arti)?|m(?:s|w)?|c(?:g|n)|zh?|d|f|g|j|s|t|v|w|y))|(?:c
(?:o(?:m(?:p(?:a(?:ny|re)|uter)|m(?:unity|bank)|sec)?|n(?:s(?:truction|ulting)|t(?:ractors|act)|dos)|u
(?:pons?|ntry|rses)|l(?:lege|ogne)|o(?:king|l|p)|rsica|ffee|ach|des)?|a(?:r(?:e(?:ers?)?|avan|tier|ds
|s)?|n(?:cerresearch|on)|p(?:etown|ital)|s(?:ino|a|h)|t(?:ering)?|m(?:era|p)|ll?|fe)?|l(?:i(?:ni(?:que|c
)|ck)|o(?:thing|ud)|ub(?:med)?|eaning|aims)?|h(?:a(?:n(?:nel|el)|se|t)|r(?:istmas|ome)|urch|eap|lo
e)?|r(?:edit(?:union|card)?|icket|uises|own|s)?|i(?:t(?:y(?:eats)?|ic)|priani|rcle|sco)?|e(?:nter|rn|b|
o)|u(?:isinella)?|y(?:mru|ou)?|f(?:a|d)?|b(?:a|n)|sc|c|d|g|k|m|n|v|w|x|z))|(?:d(?:e(?:l(?:ivery|oitte|ta|l
)|nt(?:ist|al)|al(?:er|s)|si(?:gn)?|mocrat|gree|v)?|i(?:rect(?:ory)?|amonds|scount|gital|et)|a(?:t(?:ing
|sun|e)|bur|nce|d|y)|o(?:wnload|mains|cs|ha|g)?|u(?:rban|bai)|rive|clk|vag|ds|np|j|k|m|z))|(?:e(?:x(
?:p(?:osed|ress|ert)|traspace|change)|n(?:gineer(?:ing)?|terprises|ergy)|d(?:u(?:cation)?|eka)|u(?
:rovision|s)?|ve(?:rbank|nts)|m(?:erck|ail)|s(?:tate|q)?|a(?:rth|t)|quipment|r(?:ni)?|pson|c|e|g|t))|(?:
f(?:i(?:na(?:nc(?:ial|e)|l)|r(?:estone|mdale)|sh(?:ing)?|t(?:ness)?|lm)?|a(?:i(?:rwinds|th|l)|s(?:hion|t
)|mily|ns?|ge|rm)|o(?:r(?:sale|ex|um|d)|o(?:tball)?|undation|x)?|l(?:i(?:ghts|ckr|r)|o(?:rist|wers)|smi
dth|y)|r(?:o(?:ntier|gans)|esenius|l)?|u(?:rniture|tbol|nd)|e(?:edback|rrero)|tr|yi|j|k|m))|(?:g(?:o(?:l(
?:d(?:point)?|f)|o(?:g(?:le)?)?|p|t|v)|r(?:a(?:inger|phics|tis)|een|ipe|oup)?|u(?:i(?:tars|de)|ardian|cci
|ge|ru)?|a(?:l(?:l(?:ery|up|o))?.....
```

Challenge 2: Defanging!

- Analysts deliberately obfuscate potentially harmful data to prevent it from being routable or executable
- Some approaches are more common than others but there is no standard method
- Typical types of data: IP addresses, FQDN, email, file extensions
- Examples:
  - www dot google dot com (www.google.com)
  - www[.]google[.]com
  - www[.google[.com
  - www{.}google{.}com
  - incidents at cert dot org (incidents@cert.org)
- While I was writing this slide Powerpoint helpfully turned the urls and addresses above into a link… (but not the defanged ones)

Solution for Defanging

- We realized modularity is important
  - Modularity makes regex much easier to read
  - Modular regex are easier to reuse and maintain by others

- **Not modular**:

```
ipv4 = r"""
      (?:25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9][0-9]|[0-9]
      (?:[\[\(<{](?:\.|dot|DOT)[\]\)>}]?|[\[\(<{]?(?:\.|dot|DOT)[\]\)>}]|[\
[\(<{][dD][\]\)>}]|\s(?:\.|dot|DOT)\s|(?:\.|dot|DOT))
      (?:25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9][0-9]|[0-9])
      (?:[\[\(<{](?:\.|dot|DOT)[\]\)>}]?|[\[\(<{]?(?:\.|dot|DOT)[\]\)>}]|[\
[\(<{][dD][\]\)>}]|\s(?:\.|dot|DOT)\s|(?:\.|dot|DOT))
      (?:25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9][0-9]|[0-9])
      (?:[\[\(<{](?:\.|dot|DOT)[\]\)>}]?|[\[\(<{]?(?:\.|dot|DOT)[\]\)>}]|[\
[\(<{][dD][\]\)>}]|\s(?:\.|dot|DOT)\s|(?:\.|dot|DOT)
      (?:25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9][0-9]|[0-9])
"""
```

- **Modular**:

```
ipv4 = r"""
      %(quad)s %(dot)s %(quad)s %(dot)s %(quad)s %(dot)s %(quad)s
""" % primitives
```

Sample Regex Construction

- Putting it all together

```
primitives["dot"] = r"""
     (?:
          # asymmetric brackets ok
          [\[\(<{]  (?: \. | dot | DOT ) [\]\)>}]?
        | [\[\(<{]? (?: \. | dot | DOT ) [\]\)>}]
        | [\[\(<{] [dD] [\]\)>}]
          # spaces must both be present
        | \s (?: \. | dot | DOT ) \s
          # plain dot has to come last
        | (?: \. | dot | DOT )
     )
"""

primitives["tld"] = ... # you've seen this one

fqdn = r"""
     (?: (?: [a-zA-Z0-9][a-zA-Z0-9\-_]* # subdomains
             %(dot)s )+                 # dots
         (?: %(tld)s ))                 # top-level domain
""" % primitives
```

Harvesting Artifacts

Testing & Measuring

Ground Truth Testing Framework

- How we measured success
- Runs on a set of pre-selected and manually parsed reports
- Measures false positives and false negatives between runs
- Good for catching regressions and improvements in regex iterations

Bulk Testing Framework

- Runs on entire corpus
- Measures raw extraction counts for each category for comparison between runs
- Saves raw extractions for each category for quick comparison with prior runs

Harvesting Artifacts

Results and Impact

Summary of Results and Impact

- We went from extracting **10** common incident data types to **24**
  - Found and now successfully extract 14 new types
- Tools for better interrogation of our corpus (of incident reports) and have methods for identifying more new types in the future
- Our Observable extraction rates went from **380,000** to **1,800,000** on 3 years of reports
- We now recognize and extract many cases of defanging in our data
- Our Regex tools are now much more modular and readable. They are easier to maintain and add to in the future.

24 Types of incident data we extract

- IPv4
- IPv4 CIDR
- IPv4 range
- IPv6
- IPv6 CIDR
- IPv6 range
- MD5
- SHA1
- SHA256
- ssdeep
- FQDN
- email address
- URL
- user agent
- filename
- filepath
- registry key
- ASN
- CVE
- country
- ISP
- ASN owner
- malware
- attack type

Tools we created to support Regex Project

- Ground Truth testing suite
  - Tests regex iterations over selected known reports and illuminates false positives, false negatives, and true positives and their differences between runs, along with statistics

- Autogen
  - Generates optimized regexes from a provided list of tokens and primitives in order to increase overall performance

- Bulk query
  - Tests regexes over a random selection of the corpus for exploration and testing purposes
  - Runs the suite over the entire corpus generating statistics and saving results for comparison across runs, allowing for performance measurements

Future Plans

- In discussions about open source
  - Bulk query
  - Autogeneration from tokens
  - Smoke testing suite
  - Dataset of defanged examples
- Regex enhancements
  - Extract more types
  - Expand observational catalog of tokens/constructs for categories such as malware and attack types
- Multi-stage Regex
- Automated parsing of reports into semantic sections first
- Adapt and improve other data mining methods on our corpus
  - See “Acing the IOC Game: Toward Automatic Discovery and Analysis of Open-Source Cyber Threat Intelligence”

Contact Information

Presenter / Point of Contact

Matt Sisk

Senior Member of the Technical Staff

Network Engineer/Researcher

Telephone: +1 412.268.9214

Email: sisk@cert.org

Project POCs

Robin Ruefle

Senior Member of the Technical Staff

Team Lead

Email : rmr@cert.org

Samuel Perl

Senior Member of the Technical Staff

Task Technical Lead

Email: sjperl@cert.org