Analysis of patterns and statistical findings of a SSH log

Arttu Ylä-Sahra

16. toukokuuta 2018

Contents

1. Introduction
   - Motivation, data, domain
   - Technical setup
2. General statistics
   - Key properties
   - Username-password pairs
3. Association rules
4. Clusterization
   - Introduction and techniques
   - Results
5. Conclusions

Motivation

- I manage a personal home server...
- ... which experiences *constant* break-in attempts
  - There is a high number of bots scanning the nowadays "small" IPv4 address space
- Particularly SSH attacks are evident, partially due to the fact of it being the only common open port
- Something I could do to measure it..?

Data domain, and general properties

- Baseline information for login attempts is collected
  - A.K.A: Time, attempted username and password, IP address
  - All connection attempts will fail authentication; attacker behavior analysis is out of scope, since it is quite difficult to do safely, and has larger risks in it
  - The baseline information is essentially all useful data we can quickly and easily derive from attempted logins
- Dataset is entirely self-collected, and therefore no copyright questions arise
- It is slightly unclear if IPs should be considered "personal information"in this situation, in scope of GDPR and other applying legislation - IP addresses not shown in this presentation

Technical setup

- A Raspberry Pi...
- ... equipped with specialized honeypot software.
  - Unprivileged SSH connection logging script made in Python. No actual, SUID-capable SSH daemons were placed in harm's way!
- Runs on a separate IoT VLAN network, has its own public IPv4 address - isolated from other household networks!
- Regularly audited with `rkhunter` and by checking public lists of abusive IPs

Technical setup

- Logs are saved in a simple text file. Heartbeats *(H)* are also included, to calculate approximate attack *(C)* density. IP addresses are masked for privacy.

Example

```
H|1523742300
C|**.***.***.**|VN|YWRtaW4=|YWRtaW4=|1523742318
H|1523742320
C|***.**.***.*|DO|YWRtaW4=|cGFzc3dvcmQ=|1523742328
C|***.***.***.**|BR|YWRtaW4=|ZGVmYXVsdA==|1523742336
H|1523742340
```

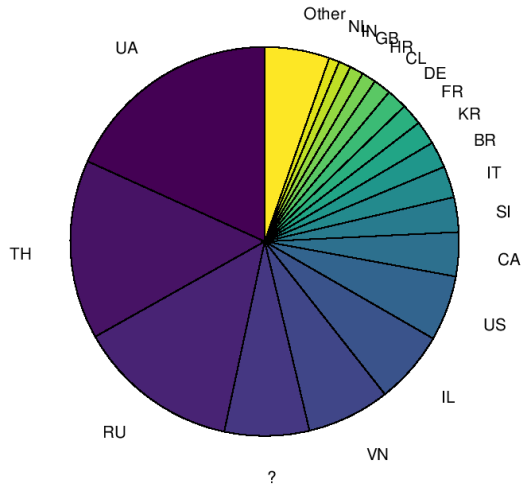
Technical setup

- Logs are then preprocessed using a Ruby program specifically made for this express purpose
  - e.g IP address to country
  - The logs are output in a format which is easy for a human reader
  - Some statistical analysis is also done by this program; more on that on later slides
- Some further postprocessing is made with Octave, mostly graphical functionality
- Finally, some of the data is instrumented automatically to this LaTeXpresentation!

General statistics

- A total of 97746 login attempts were recorded
  - These login attempts were split into 2487 *sequences*, in which one sequence is a set of continuous (close in time) attacks from same origin; roughly 39.303 attacks per sequence were recorded by average.
- There were roughly 171.648 attacks per hour during the times when the SSH honeypot was running
- Most popular username was `root (80109 hits / ~81.96%)` (!!), and password `123456 (1388 hits / ~1.42%)`
- Most attacks arrived from `CN (75774 hits / ~77.52%)`, with a total of 93 different countries found (including unknowns as one country). When countries are sorted by count, median comes at 12 attacks.

Countries



- China has been omitted from the chart (would otherwise take a roughly 3/4 share). Ukraine has slightly over 4k hits
- Leading countries typically poor and/or corrupt, but industrial countries are not exempt!
  - Ukraine, Thailand, Russia, Vietnam.. Israel, United States, Canada.. Slovenia, Italy, Brazil, South Korea..

Username-password pairs

- There were a total of 27783 unique combinations recorded, of which 41.558% were seen only once, and 69.172% were seen maximum of 3 times
- Most common ones tend to be weak, default passwords; no doubt an attractive target, due to the availability of such badly configured, publicly accessible systems
- Rarer ones are either unique combinations not commonly seen, or variants of the more common combinations (more about this in the clustering section)

Username-password pairs

- Sample of the most common combinations, taken from the log

Example

```
# Username, Password, Count
admin,default,139
ubnt,ubnt,140
pi,raspberry,145
admin,1234,149
root,password,157
admin,password,167
root,admin,175
support,support,220
admin,admin,243
root,root,245
```

Basic properties of association analysis

- We want to find common patterns between sequences
- If an attacker tries X, does he/she/it also try Y?
- Let's try the Apriori algorithm, as presented in the demonstrations!
- One sequence is treated as one transaction, with each unique username-password pair as one 'item'. Cutoff was at 12 attempts for each unique pair

Results for association analysis

- No universally applying results; strongest association has 8% support
- But most rules have fair confidence; all recorded rules have at least 50%, many of them more
- Weak combinations tend to associate with other weak combinations
  - Intepretation: systemic fishing for easy targets? Poor configuration tends to correlate with poor maintenance...
  - Seems to match casual observations from log files and expected behavior from bots

Sample from the list of found association rules

- Sample of the strongest rules

Example

```
(admin:password) -> (admin:admin), (support 94 transactions / 8.0411%, confidence 60.6452%)
(admin:1234) -> (admin:admin), (support 92 transactions / 7.87%, confidence 71.875%)
(admin:1234) -> (admin:password), (support 83 transactions / 7.1001%, confidence 64.8438%)
(admin:password) -> (admin:1234), (support 83 transactions / 7.1001%, confidence 53.5484%)
(root:admin) -> (admin:admin), (support 81 transactions / 6.929%, confidence 51.9231%)
(admin:1234) -> (root:root), (support 78 transactions / 6.6724%, confidence 60.9375%)
(admin:1234) -> (user:user), (support 72 transactions / 6.1591%, confidence 56.25%)
(user:user) -> (admin:1234), (support 72 transactions / 6.1591%, confidence 67.2897%)
(admin:1234) -> (admin:admin123), (support 69 transactions / 5.9025%, confidence 53.9062%)
(admin:admin123) -> (admin:1234), (support 69 transactions / 5.9025%, confidence 71.875%)
```

Introduction to clusterization methods

- There seemed to be distinct groups of username-password-combinations
  - Often similar variants of various weak combinations
  - Any other relations perhaps?
- Two clustering methods were experimented with: hierachical and partitioning
- For both methods, single item is an username-password pair
- Distance between two items was measured using Levenshtein string distance; basically addition, replacement and removal of characters

Hierachical clustering

- UP-pairs and clusters form a *tree*, with closer nodes being more similar
  1. Initial state: all nodes are single username-password pairs, graph only shows those
  2. Until only one node remains, do:
     - Select a random node from the remaining nodes
     - Find the nearest (by average of pairs if many, least distance) node from other nodes, remove them from the list. and merge them into one node. Return this combined node to the list
     - Add this new node to the graph, indicating this new merged mode, and draw arrows from its 'ancestors' to the new node
  3. End: only one node remaining, nodes form a binary tree

Partitioning clustering

- UP-pairs form clusters of *limited maximum distance*
- Somewhat similar to hierachical clustering before
  1. Initial state: all pairs remain, no clusters exist
  2. Until no pairs remain, do:
     - Select next pair from the stack
     - Measure the average distance of the pair to each element of a cluster; a combination must be adequately close to *at least one element of a cluster*, but must not be *too distant from any element of a cluster*
     - Add the pair to the nearest cluster - or if none match, create a new 1-element cluster for the pair
  3. End: all pairs are in clusters of 1 to N elements

Results

- Tendency to form groups exists; e.g weak passwords come in many variants. Partitioning did find a substantial amount of 1-combination groups though, so not universal
- Various types of simple key patterns of changes - think ASD or 12345
- However, not quite conclusive yet - not all data has been analyzed!
- Rarer variants could have more interesting findings, but current clusterization method rather slow and unoptimized.

What to take away from this presentation?

- If you own a server...
  - If you haven't already done so, disable SSH root logins immediately; this already kneecaps over 80% of attempts!
  - Utilize 2FA and Fail2Ban. Enforced 2FA blocks effectively blocks all non-targeted SSH attacks, since a simple username-password guess won't work. Fail2Ban blocks persistent offenders at firewall level as well.
  - Consider geoblocking according to expected origins of good traffic; e.g: Finland only originated 0.007% of attacks!

Sources and tools used

- Personal SSH Honeypot (`otamatone.arttuys.fi`) for raw data
- Custom-written Ruby, Octave scripts for curation and processing of data
- Lecture Apriori example for association analysis
- MaxMindDB IP - Country database