ASAP: Automatic Semantics-Aware Analysis of Network Payloads

Tammo Krueger
Nicole Krämer
Konrad Rieck

24.9.2010 @ ECML/PKDD Workshop on Privacy and Security Issues in Data Mining and Machine Learning

Outline

1 Introduction

2 ASAP Framework
- Alphabet Generation
- Matrix Factorization
- Template Generation

3 Experiments
- Toy Data
- Malware Communication
- Intrusion Detection

4 Conclusion & Future Work

Outline

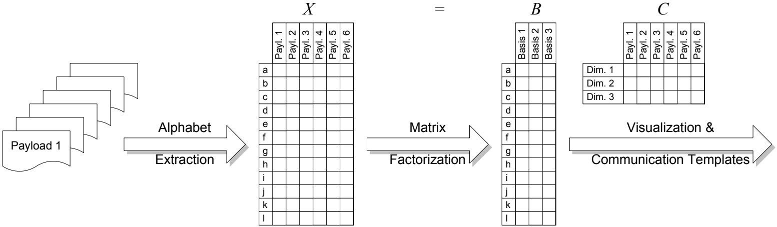
1 Introduction

2 ASAP Framework
- Alphabet Generation
- Matrix Factorization
- Template Generation

3 Experiments
- Toy Data
- Malware Communication
- Intrusion Detection

4 Conclusion & Future Work

Introduction – Idea of ASAP



- ASAP = Automatic Semantics-aware Analysis of network Payloads
- Processing framework:
  1. Generate meaningful alphabet from collected data
  2. Calculate lower-dimensional representation via matrix factorization
  3. Analyze data by inspecting both the basis B and the coordinates C
- Useful for intrusion detection, fuzz testing and forensic analysis (honeypot data, malware communication)

Outline

1 Introduction

2 ASAP Framework
- Alphabet Generation
- Matrix Factorization
- Template Generation

3 Experiments
- Toy Data
- Malware Communication
- Intrusion Detection

4 Conclusion & Future Work

ASAP Framework – Alphabet generation

- Possible features: n-gram data or tokens
- Focus the analysis by splitting the feature set:

$$F_{all} = F_{protocol} \cup F_{alphabet} \cup F_{volatile}$$

- Calculate frequency f of each feature and test via t-test:

$$\begin{aligned} p_{protocol} &= t.test(H_0 : f = 1.0) \\ p_{volatile} &= t.test(H_0 : f = 0.0) \end{aligned}$$

- Adjust p-values for multiple testing (Holm)
- Keep features which are not part of the protocol ($p_{protocol} < 0.05$) **and** not volatile ($p_{volatile} < 0.05$)
- Group features which co-occur (correlation ≈ 1.0) to a "letter" of $F_{alphabet}$

ASAP Framework – Matrix Factorization

- Factorization of alphabet matrix $A \in \mathbb{R}^{k,N}$ with $B \in \mathbb{R}^{k,\ell}, C \in \mathbb{R}^{\ell,N}, b_i \in \mathbb{R}^{k,1}, c_i \in \mathbb{R}^{\ell,1}, \underline{\ell \ll k}$:

$$A \approx BC = \overbrace{\begin{bmatrix} b_1 & \dots & b_\ell \end{bmatrix}}^{\text{basis}} \underbrace{\begin{bmatrix} c_1 & \dots & c_N \end{bmatrix}}_{\text{coordinates}}$$

- Two factorizations considered:
  - **PCA**: maximize the described variance of the data

$$\begin{aligned} b_i &= \underset{\|b\|=1}{\arg\max} \operatorname{var}\left(X^\top b\right) \\ & \quad \text{s.t. } b \perp b_j,\ j < i. \end{aligned}$$

  - **NMF**: minimize error with positive basis and coordinates

$$\begin{aligned} (B, C) &= \underset{B,C}{\arg\min} \|X - BC\| \\ & \quad \text{s.t. } b_{ij} \geq 0,\ c_{jn} \geq 0\,. \end{aligned}$$

ASAP Framework – Template Generation

```
function genTemplate(alphabet, weights, maxGap)
    representation := []                                  ▷ empty list
    L := tokens ordered by weight
    while length(L) $\geq$ 2 do
        curToken := L.pop()                ▷ token with highest weight
        for (gap := 0; gap $\leq$ maxGap; maxGap++) do
            for (i := 1; i $\leq$ length(L); i++) do
                token := L[i]
                if overlap(curToken, token, gap) then
                    curToken := merge(curToken, token)
                    L.remove(token)         ▷ remove merged token
                    i := 0              ▷ restart matching process
        representation.push(curToken)       ▷ no more overlaps
    representation.extend(L)           ▷ add any remaining token
    return representation
```

Outline

1 Introduction

2 ASAP Framework
- Alphabet Generation
- Matrix Factorization
- Template Generation

3 Experiments
- Toy Data
- Malware Communication
- Intrusion Detection

4 Conclusion & Future Work

Experiments – Toy Data

```
GET static/3lpAN6C2.html HTTP/1.1
Host: www.foobar.com
Accept: */*
```

Request for static content

```
GET cgi/search.php?s=Eh0YKj3r3wD2I HTTP/1.1
Host: www.foobar.com
Accept: */*
```

Search query

```
GET cgi/admin.php?action=rename&par=dBJh7hS0r5 HTT
Host: www.foobar.com
Accept: */*
```

Administrative request

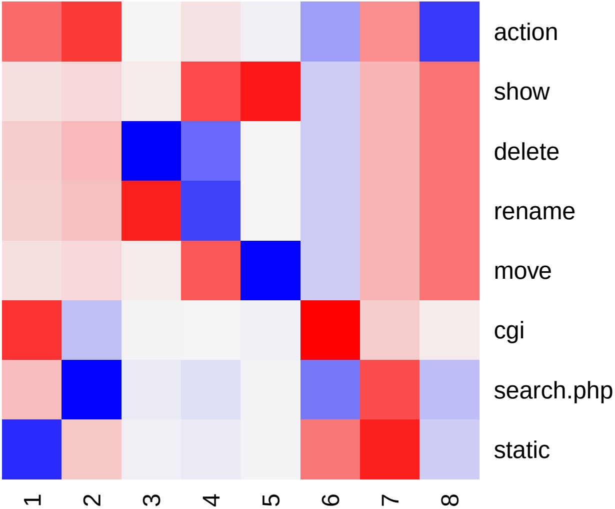
action $\in$ { show, delete, rename, move }

Experiments – Toy Data Alphabet

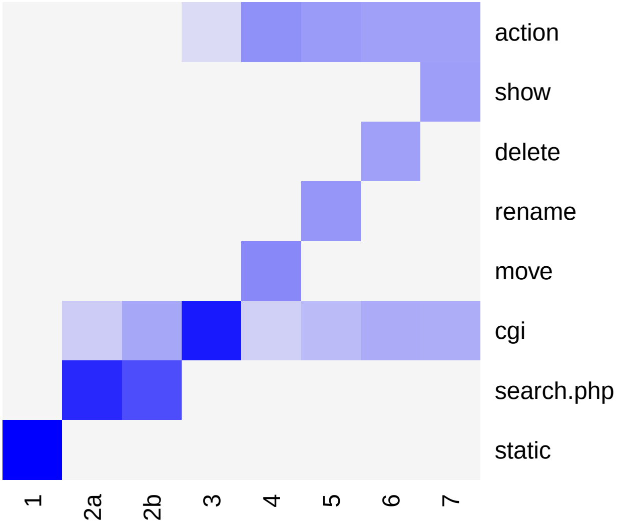
- Tokens of requests based on the standard HTTP delimiter set ({`()<>@,:;\"/[]?={}&\n\r \t`})
- Resulting alphabet:
  1. `static`
  2. `cgi`
  3. (`search.php` ∧ `s`)
  4. (`action` ∧ `admin.php` ∧ `par`)
  5. `rename`
  6. `move`
  7. `delete`
  8. `show`

No protocol (like `GET`) or volatile (like `dBJh7hS0r5`) tokens in the alphabet!

Experiments – Toy Data Matrix Factorization PCA



Experiments – Toy Data Matrix Factorization NMF



Experiments – Malware Communication

- Repetitive execution of malware in sandbox environment
- 4-grams as basic strings and NMF as factorization
- Resulting textual representation of base vectors:

	IRC sessions of Vanbot
1)	`MODE #las6↩USER b ∧ JOIN #las6 ∧ 041- Running TFTP wormride...`
2)	`c↩MODE #ns↩USER ∧ c +xi↩JOIN #ns ∧ ub.28465.com↩PONG :hub.2`
	HTTP requests for updates of Vanbot
3)	`GET /lal222.exe HTTP/1.0↩Host:  zonetech.info ∧ /lb3.ex ∧ /las1.ex...`
	IRC session of Virut file infector
4)	`x↩USER e020501 .  .  .  ∧ JOIN &virtu3 ∧ NICK bb ∧ CK gv ∧ R h020`

Experiments – Intrusion Detection

- Task: detect anomalous packets in a network stream
- Compare two centroid-based anomaly detectors:

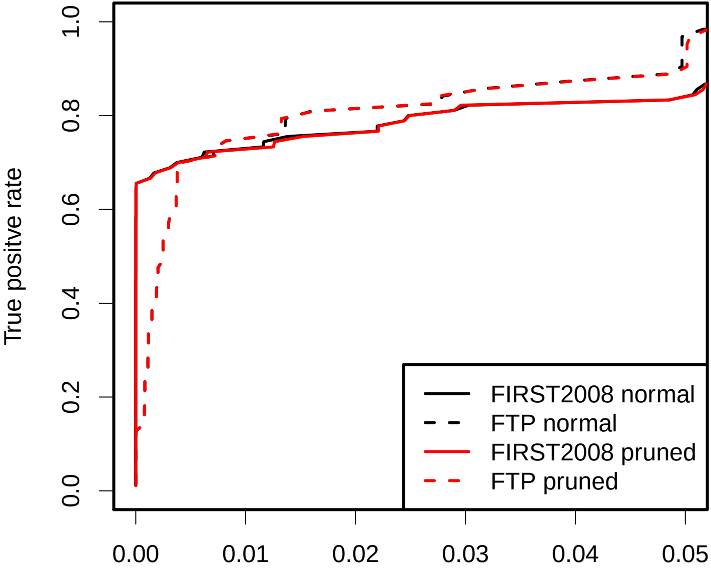
$$\mu_{full} = \frac{1}{N}\sum_{i=1}^{N}\phi(p_i) \qquad \mu_{pruned} = B(\frac{1}{N}\sum_{i=1}^{N}c_i)$$

- For each new data point calculate distance to centroid and decide based on fp-tuned threshold, whether it is anomalous or not
- Compare receiver operating characteristic (ROC) of both detectors and measure run-time performance
- Two datasets:

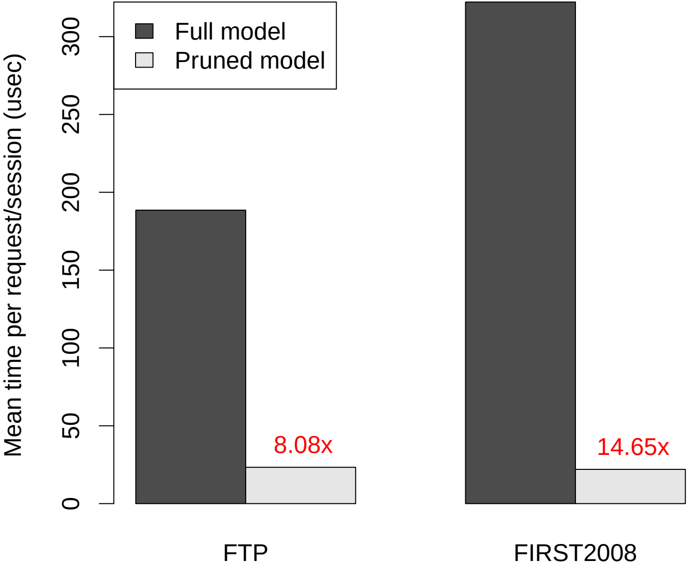
FIRST2008 60 days of HTTP requests (static and CMS)

FTP03 10 days of FTP sessions

Experiments – Intrusion Detection ROC



Experiments – Intrusion Detection run-time



Outline

1 Introduction

2 ASAP Framework
- Alphabet Generation
- Matrix Factorization
- Template Generation

3 Experiments
- Toy Data
- Malware Communication
- Intrusion Detection

4 Conclusion & Future Work

Conclusion & Future Work

- Given a set of network payloads the ASAP framework. . .
  1. generates a meaningful alphabet based on statistical tests
  2. extracts semantics-aware base vectors (inspection) and corresponding coordinates for each payload (visualization)
  3. gives a concise textual representation of the bases
- Future Work:
  - Evaluate sparse methods (both PCA and NMF)
  - Adjust objective function of NMF to incorporate domain knowledge
  - Extend to other domains (pure text data, bioinformatics)
  - Improve template generation process (alignment)

Questions? Remarks?
Thanks for your attention!