An Incremental Learner for Language-Based Anomaly Detection in XML

Harald Lampesberger

Department of Secure Information Systems
University of Applied Sciences Upper Austria
harald.lampesberger@fh-hagenberg.at

LangSec Workshop, 26. May 2016

Motivation

Extensible Markup Language (XML)

- Data serialization format for many protocols
- SOAP/WS-*, XMPP, SAML, XHTML, RSS, Atom, ...

Schema validation is a first-line defense

- A schema specifies types of elements and production rules
- Validation rejects unacceptable inputs

Two language-theoretic flaws

1. XML Schema (XSD) extension points are wildcards
2. References raise expressiveness beyond context free

XSD Extension Points

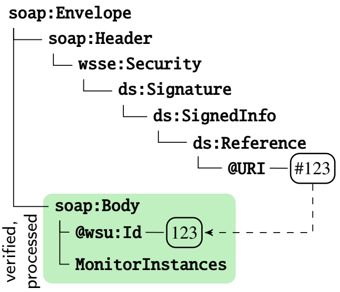
From http://schemas.xmlsoap.org/soap/envelope/

```
...
<xs:element name="Header" type="tns:Header"/>
<xs:complexType name="Header">
  <xs:sequence>
    <xs:any namespace="##other" minOccurs="0"
     maxOccurs="unbounded" processContents="lax"/>
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
...
```

Signature Wrapping Attack

Digitally signed part ≠ processed part

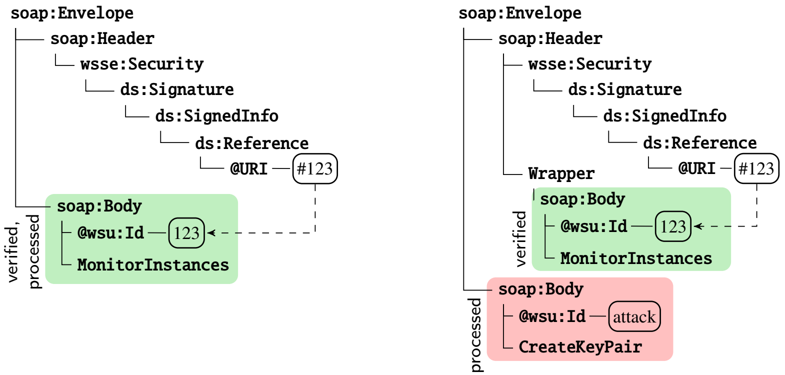
- Used in WS-Security and SAML single sign-on
- Somorovsky et al. (2012): 11/14 SAML implementations vulnerable



Signature Wrapping Attack

Digitally signed part $\neq$ processed part

- Used in WS-Security and SAML single sign-on
- Somorovsky et al. (2012): 11/14 SAML implementations vulnerable



Jensen et al. (2011): removing extension points is hard

Language-Based Anomaly Detection

Approach: learn the acceptable language

Language-Based Anomaly Detection

Approach: learn the acceptable language

1. Datatyped XML Visibly Pushdown Automaton (dXVPA)
    - Mixed-content XML streaming
    - Datatypes generalize character data
    - Character-data XVPA (cXVPA) for stream validation

Language-Based Anomaly Detection

Approach: learn the acceptable language

1. Datatyped XML Visibly Pushdown Automaton (dXVPA)
   - Mixed-content XML streaming
   - Datatypes generalize character data
   - Character-data XVPA (cXVPA) for stream validation
2. Incremental learner for grammatical inference
   - Constructs a dXVPA from examples
   - Unlearning and sanitization against poisoning attacks

Language-Based Anomaly Detection

Approach: learn the acceptable language

1. Datatyped XML Visibly Pushdown Automaton (dXVPA)
   - Mixed-content XML streaming
   - Datatypes generalize character data
   - Character-data XVPA (cXVPA) for stream validation
2. Incremental learner for grammatical inference
   - Constructs a dXVPA from examples
   - Unlearning and sanitization against poisoning attacks
3. Experiments
   - Train and test
   - Two synthetic scenarios from ToXgene
   - Two realistic scenarios from Axis2 web service

dXVPAs

Event stream alphabets

- Σ_{call} ... startElement
- Σ_{ret} ... endElement
- Σ_{int} ... datatypes

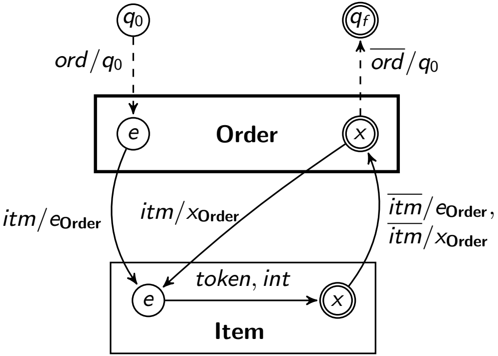
Stack alphabet = states

States partitioned into modules (schema types)

Transitions in and between modules

cXVPA representation

- Unified text checks
- Fast validation



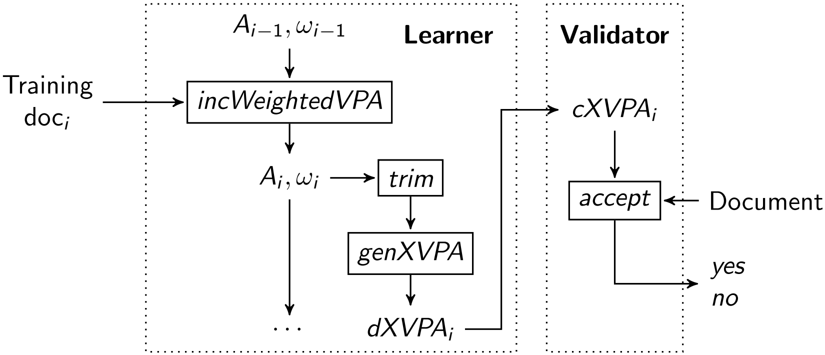

```
<ord>
  <itm>Product A</itm>
  <itm>8877955335</itm>
</ord>
```

Incremental Learning Step

Learner computes an updated dXVPA

- Datatyped event stream
- $A_i \ldots$ incrementally updateable automaton
- $\omega_i \ldots$ frequencies of states and transitions

Validator checks acceptance

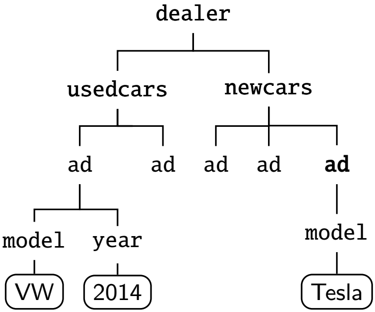


How Learning Works

Every event stream prefix gets a unique state

- A named state is a pair (u, v)
- $u \dots$ typing-context string
- $v \dots$ left-sibling string

Merge two states if they are k-l-locally the same

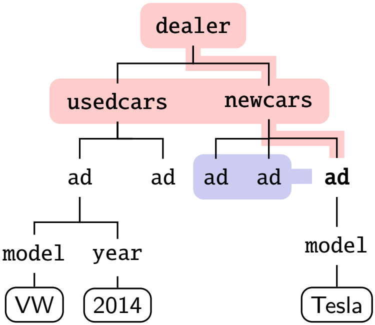


How Learning Works

Every event stream prefix gets a unique state

- A named state is a pair (u, v)
- $u \dots$ typing-context string
- $v \dots$ left-sibling string

Merge two states if they are k-l-locally the same

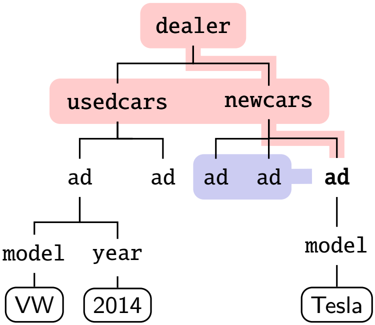


How Learning Works

Every event stream prefix gets a unique state

- A named state is a pair (u, v)
- $u \ldots$ typing-context string
- $v \ldots$ left-sibling string

Merge two states if they are k-l-locally the same



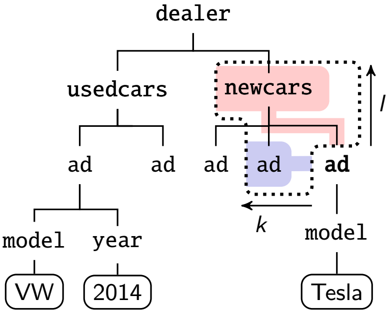
$(dealer\#usedcars \cdot newcars, ad \cdot ad)$

How Learning Works

Every event stream prefix gets a unique state

- A named state is a pair (u, v)
- $u \ldots$ typing-context string
- $v \ldots$ left-sibling string

Merge two states if they are k-l-locally the same



$(dealer\#usedcars \cdot newcars, ad \cdot ad)$

↓ 1-1 local

$(newcars, ad)$

Poisoning Attacks

$\omega_i \ldots$ frequencies of states and transitions from learning

Unlearning

- An already learned attack is later identified
- Remove specific knowledge by decrementing ω_i
- Trim zero-weight states and transitions

Sanitization

- Hidden poisoning attacks
- Assumption: only few of those
- Decrement ω_i and trim zero-weight states and transitions

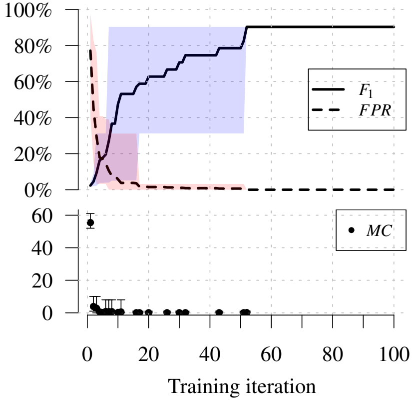
Experiments

Two synthetic and two realistic datasets

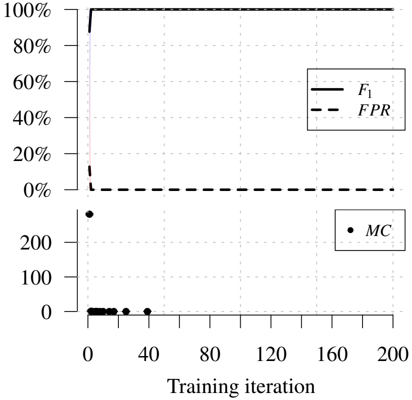
Learning progress

- Train and test, binary classification, mind changes (MC)

Catalog, $k = 1, l = 2$



VulnShopAuthOrder, $k = 1, l = 2$



Conclusions

Learner outperformed schema validation

- All signature wrapping attacks were detected (schema validation: 0)
- No false positives
- False negatives resulted from coarse XSD datatypes
- Fast convergence

Contributions in the paper

- dXVPA and cXVPA language representations
- Lexical datatype system for datatype inference from text
- Algorithms for the incremental learner
- Details on experiments

Use cases

- Security mechanism for any XML-based interaction
- Especially for systems using composed schemas
- XML firewall

Appendix

Lexical Datatype System

Learner needs a datatyped event stream

- Lexically distinct XSD datatypes instead of character data

1. Lexical subsumption
   - Minimally required datatypes
   - Strict subsumption is ambiguous
   - `false` $\mapsto \{$*lang.*, *bool.*, *NCName*$\}$
2. Preference heuristic
   - Datatypes partitioned into kinds
   - "Preferred" relation
   - e.g. boolean $<$ language
   - `false` $\mapsto \{$*boolean*$\}$

Example

- $\{$`1, 0, true, 33`$\} \mapsto \{$*boolean*, *unsignedByte*$\}$