

Security Testing of Stateful Systems

Erik Poll

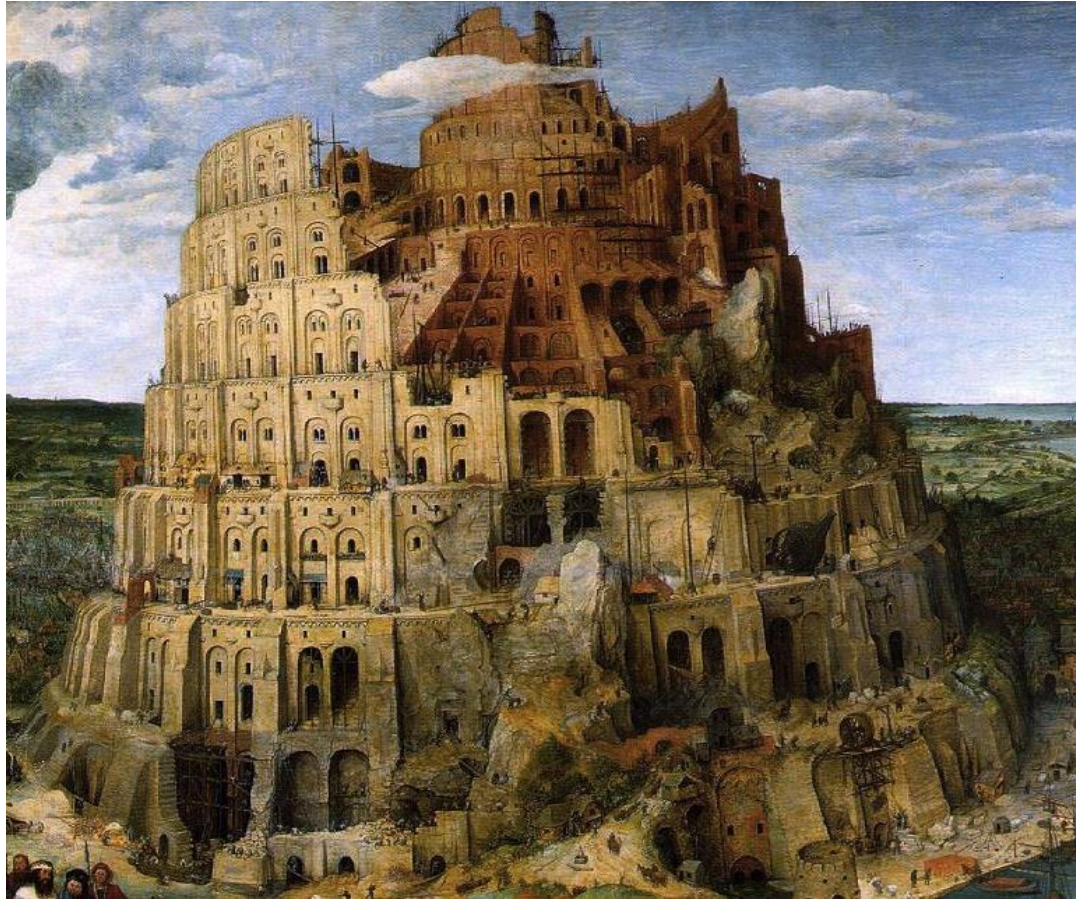
joint work with several PhD & MSc students

Digital Security group

Radboud University, Nijmegen, the Netherlands

Overview

1. Wider context: **LangSec**
2. **State machine learning** as form of security testing
3. More general forms of **fuzzing** for stateful systems



1. LangSec (Language-Theoretic Security)

LangSec

Important root cause of security problems:

*overly complex, expressive, poorly specified, ambiguous, input languages
(aka formats, protocols,...)*

Eg PDF, JPEG, Word, Bluetooth, TCP/IP, TLS, 5G,



Sergey Bratus & Meredith Patterson
'The science of insecurity' CCC 2012

<http://www.youtube.com/watch?v=3kEfedtQVOY>



Typical bug categories

OWASP Top 10 [2017]

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging & Monitoring

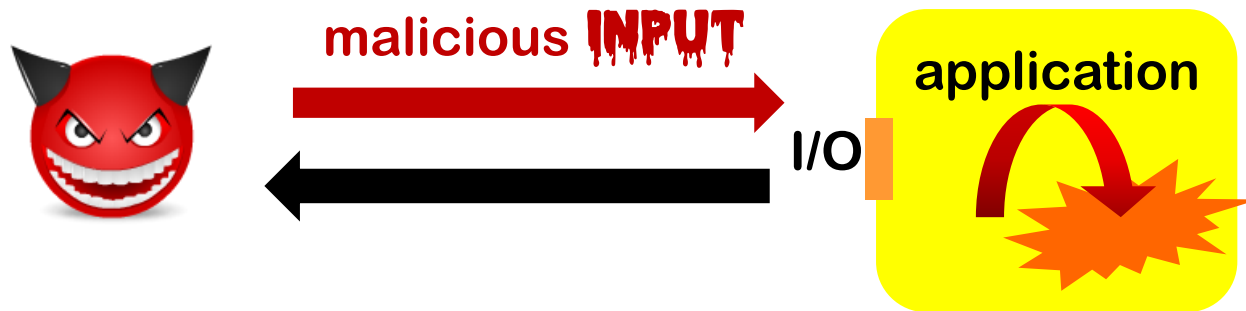
CWE TOP 25 [2022]

- 1 Out-of-bounds Write
- 2 Cross-site Scripting
- 3 SQL Injection
- 4 Improper Input Validation
- 5 Out-of-bounds Read
- 6 OS Command Injection
- 7 Use After Free
- 8 Path Traversal
- 9 Cross-Site Request Forgery (CSRF)
- 10 Unrestricted Upload of File with Dangerous Type
- 11 NULL Pointer Dereference
- 12 Deserialization of Untrusted Data
- 13 Integer Overflow or Wraparound
- 14 Improper Authentication
- 15 Use of Hard-coded Credentials
- 16 Missing Authorization
- 17 Command Injection
- 18 Missing Authentication for Critical Function
- 19 Improper Restriction of Bounds of Memory Buffer
- 20 Incorrect Default Permissions
- 21 Server-Side Request Forgery (SSRF)
- 22 Race Condition
- 23 Uncontrolled Resource Consumption
- 24 Improper Restriction of XML External Entity Reference
- 25 Code Injection

CWE TOP 1000

```
CWE-1: Absence of Authentication
CWE-2: Improper Validation of User Input
CWE-3: Missing Authorization
CWE-4: Improper Input Validation
CWE-5: Out-of-bounds Read
CWE-6: OS Command Injection
CWE-7: Use After Free
CWE-8: Path Traversal
CWE-9: Cross-Site Request Forgery (CSRF)
CWE-10: Unrestricted Upload of File with Dangerous Type
CWE-11: NULL Pointer Dereference
CWE-12: Deserialization of Untrusted Data
CWE-13: Integer Overflow or Wraparound
CWE-14: Improper Authentication
CWE-15: Use of Hard-coded Credentials
CWE-16: Missing Authorization
CWE-17: Command Injection
CWE-18: Missing Authentication for Critical Function
CWE-19: Improper Restriction of Bounds of Memory Buffer
CWE-20: Incorrect Default Permissions
CWE-21: Server-Side Request Forgery (SSRF)
CWE-22: Race Condition
CWE-23: Uncontrolled Resource Consumption
CWE-24: Improper Restriction of XML External Entity Reference
CWE-25: Code Injection
CWE-26: Improper Restriction of XML External Entity Reference
CWE-27: Improper Restriction of XML External Entity Reference
CWE-28: Improper Restriction of XML External Entity Reference
CWE-29: Improper Restriction of XML External Entity Reference
CWE-30: Improper Restriction of XML External Entity Reference
CWE-31: Improper Restriction of XML External Entity Reference
CWE-32: Improper Restriction of XML External Entity Reference
CWE-33: Improper Restriction of XML External Entity Reference
CWE-34: Improper Restriction of XML External Entity Reference
CWE-35: Improper Restriction of XML External Entity Reference
CWE-36: Improper Restriction of XML External Entity Reference
CWE-37: Improper Restriction of XML External Entity Reference
CWE-38: Improper Restriction of XML External Entity Reference
CWE-39: Improper Restriction of XML External Entity Reference
CWE-40: Improper Restriction of XML External Entity Reference
CWE-41: Improper Restriction of XML External Entity Reference
CWE-42: Improper Restriction of XML External Entity Reference
CWE-43: Improper Restriction of XML External Entity Reference
CWE-44: Improper Restriction of XML External Entity Reference
CWE-45: Improper Restriction of XML External Entity Reference
CWE-46: Improper Restriction of XML External Entity Reference
CWE-47: Improper Restriction of XML External Entity Reference
CWE-48: Improper Restriction of XML External Entity Reference
CWE-49: Improper Restriction of XML External Entity Reference
CWE-50: Improper Restriction of XML External Entity Reference
CWE-51: Improper Restriction of XML External Entity Reference
CWE-52: Improper Restriction of XML External Entity Reference
CWE-53: Improper Restriction of XML External Entity Reference
CWE-54: Improper Restriction of XML External Entity Reference
CWE-55: Improper Restriction of XML External Entity Reference
CWE-56: Improper Restriction of XML External Entity Reference
CWE-57: Improper Restriction of XML External Entity Reference
CWE-58: Improper Restriction of XML External Entity Reference
CWE-59: Improper Restriction of XML External Entity Reference
CWE-60: Improper Restriction of XML External Entity Reference
CWE-61: Improper Restriction of XML External Entity Reference
CWE-62: Improper Restriction of XML External Entity Reference
CWE-63: Improper Restriction of XML External Entity Reference
CWE-64: Improper Restriction of XML External Entity Reference
CWE-65: Improper Restriction of XML External Entity Reference
CWE-66: Improper Restriction of XML External Entity Reference
CWE-67: Improper Restriction of XML External Entity Reference
CWE-68: Improper Restriction of XML External Entity Reference
CWE-69: Improper Restriction of XML External Entity Reference
CWE-70: Improper Restriction of XML External Entity Reference
CWE-71: Improper Restriction of XML External Entity Reference
CWE-72: Improper Restriction of XML External Entity Reference
CWE-73: Improper Restriction of XML External Entity Reference
CWE-74: Improper Restriction of XML External Entity Reference
CWE-75: Improper Restriction of XML External Entity Reference
CWE-76: Improper Restriction of XML External Entity Reference
CWE-77: Improper Restriction of XML External Entity Reference
CWE-78: Improper Restriction of XML External Entity Reference
CWE-79: Improper Restriction of XML External Entity Reference
CWE-80: Improper Restriction of XML External Entity Reference
CWE-81: Improper Restriction of XML External Entity Reference
CWE-82: Improper Restriction of XML External Entity Reference
CWE-83: Improper Restriction of XML External Entity Reference
CWE-84: Improper Restriction of XML External Entity Reference
CWE-85: Improper Restriction of XML External Entity Reference
CWE-86: Improper Restriction of XML External Entity Reference
CWE-87: Improper Restriction of XML External Entity Reference
CWE-88: Improper Restriction of XML External Entity Reference
CWE-89: Improper Restriction of XML External Entity Reference
CWE-90: Improper Restriction of XML External Entity Reference
CWE-91: Improper Restriction of XML External Entity Reference
CWE-92: Improper Restriction of XML External Entity Reference
CWE-93: Improper Restriction of XML External Entity Reference
CWE-94: Improper Restriction of XML External Entity Reference
CWE-95: Improper Restriction of XML External Entity Reference
CWE-96: Improper Restriction of XML External Entity Reference
CWE-97: Improper Restriction of XML External Entity Reference
CWE-98: Improper Restriction of XML External Entity Reference
CWE-99: Improper Restriction of XML External Entity Reference
CWE-100: Improper Restriction of XML External Entity Reference
```

ONE main bug category: **INPUT** handling



More in particular: **PARSING** of input is dangerous

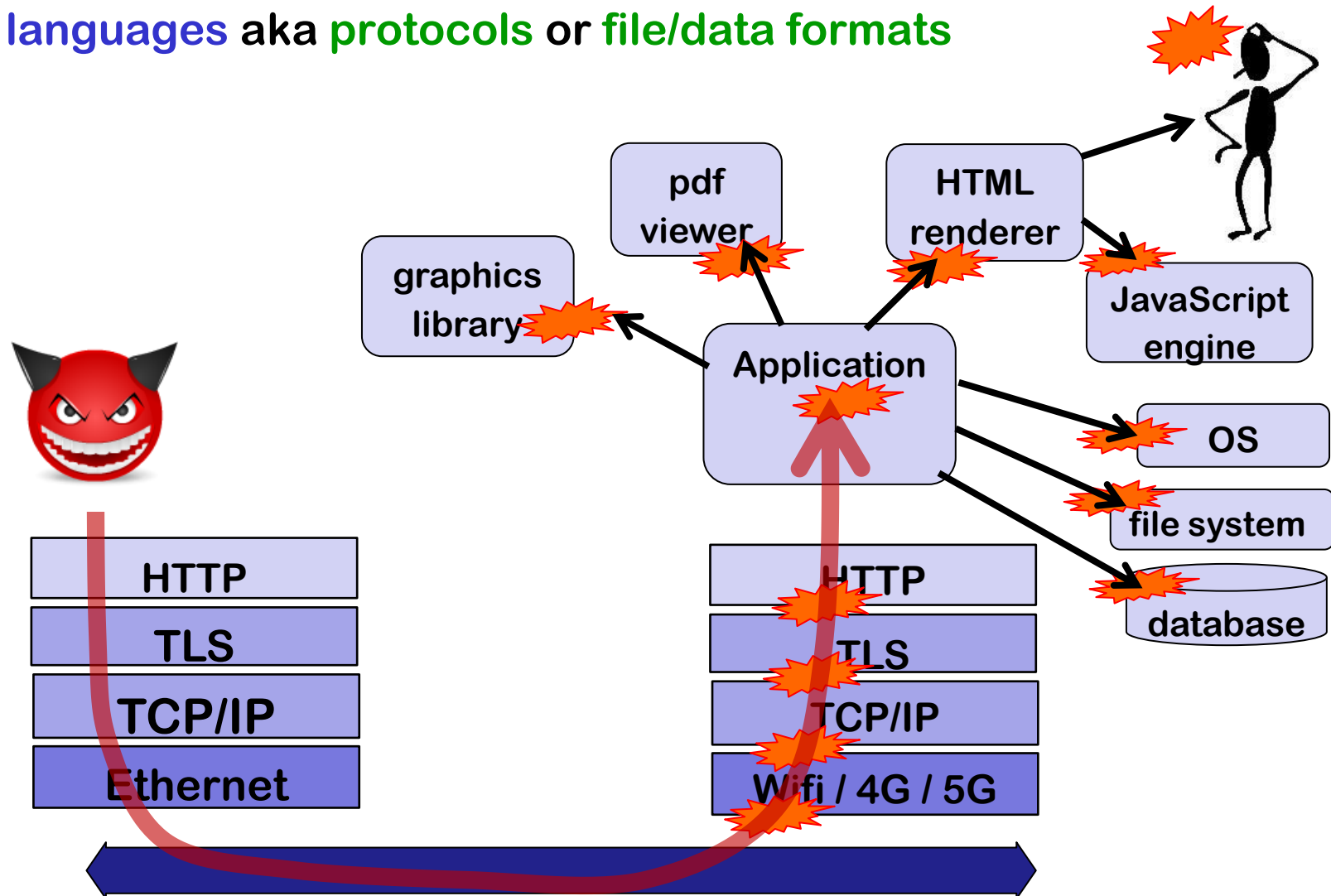
Garbage In, Garbage Out

becomes *Malicious Garbage In, Security Incident Out*

or *Garbage In, Evil Out*

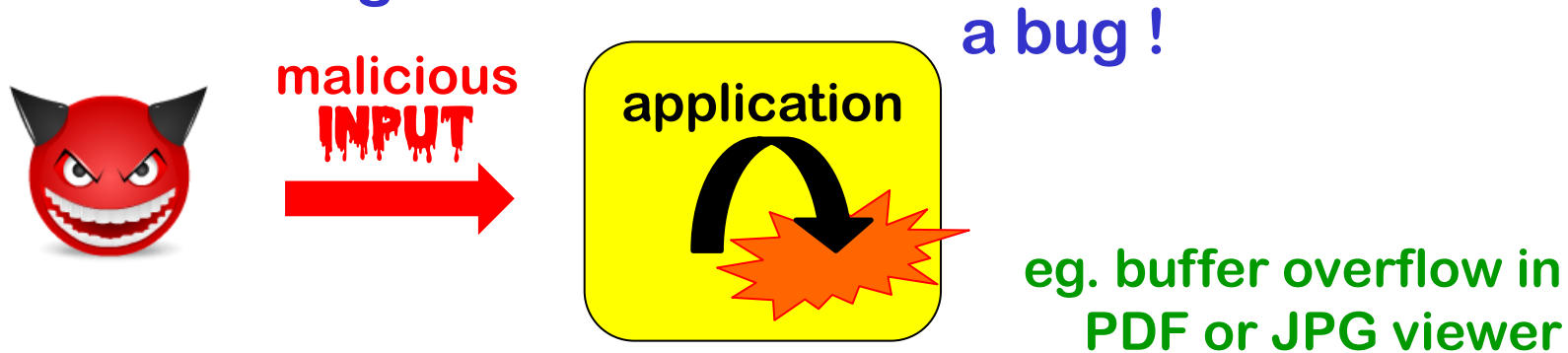
Parsing of many languages in many places

Parsing aka **decoding**, **interpreting** or **processing**
Input languages aka **protocols** or **file/data formats**

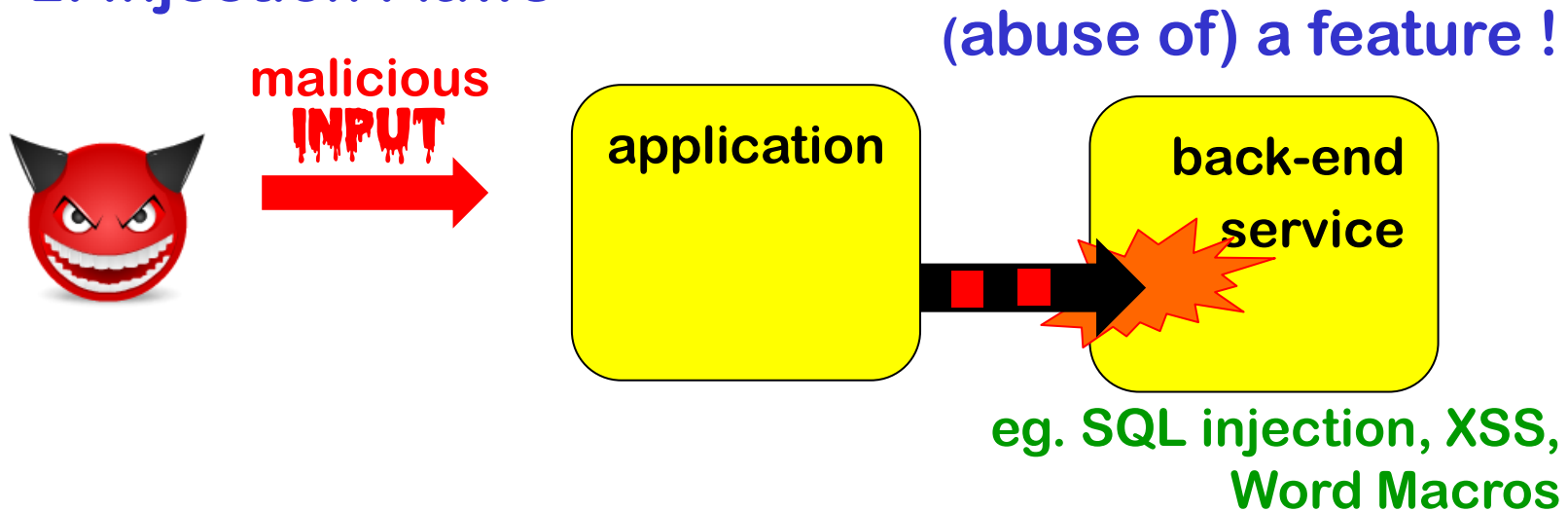


Two sub-categories: 1) bugs & 2) features

1. Processing Flaws



2. Injection Flaws

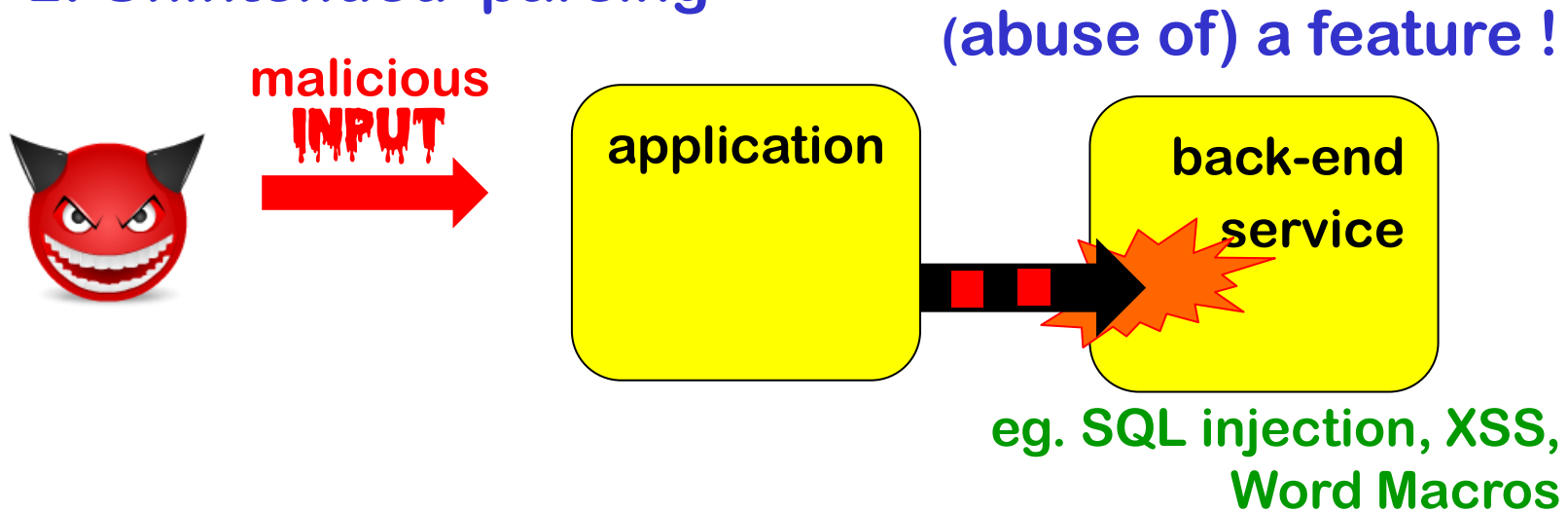


Or 1) insecure parsing and 2) unintended parsing

1. *Insecure, buggy* parsing



2. *Unintended* parsing



LangSec: tackling buggy parsing

Adding **input validation** is not the (best) solution,

- as we are only adding another parser

More structural 'LangSec' solutions to address root causes

1. **Provide clear, formal spec of input language**

ideally as regular expression or (E)BNF grammar

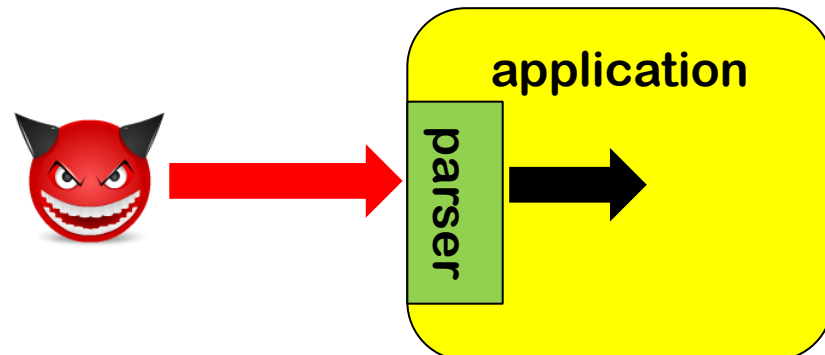
2. **Generate parser code**

using a parser generator tool

3. **Separate parsing** (of **raw bytes and strings** into some **structured datatype**)

from subsequent processing (of that **structured data**)

See langsec.org



Tackling unintended parsing

- Using more specific data types instead of **STRINGS**
 - different types to distinguish different **formats**
 - eg **URL** vs **file name**
 - to distinguish different **trust levels**
 - eg **compile-time constants** and **escaped input** vs **raw user input**

Exemplified by
Google's Trusted Type API

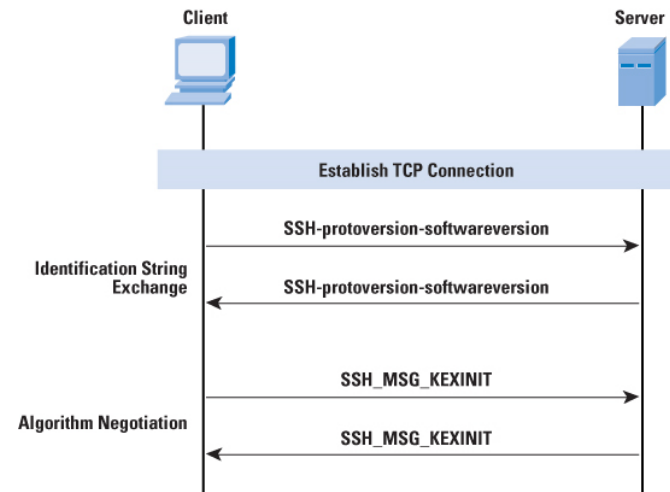
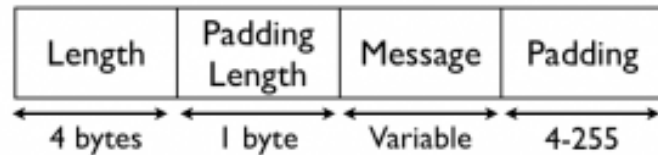


[Erik Poll, *Strings considered harmful*, USENIX :login; 2019]

2. State machine learning for security

Sessions, i.e. sequences of inputs

Many protocols not only involves a language of **input messages** but also a notion of **session**, ie. *sequence of messages*

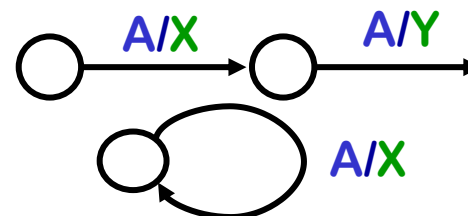
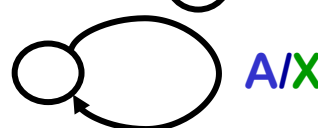


- Most specs only describe the **happy flow**...
- For security, getting **unhappy flows** correct is crucial
- Fortunately, we can extract state machines from implementations using **state machine inference** aka **active learning** using just black box testing

Active Learning aka State Machine Inference

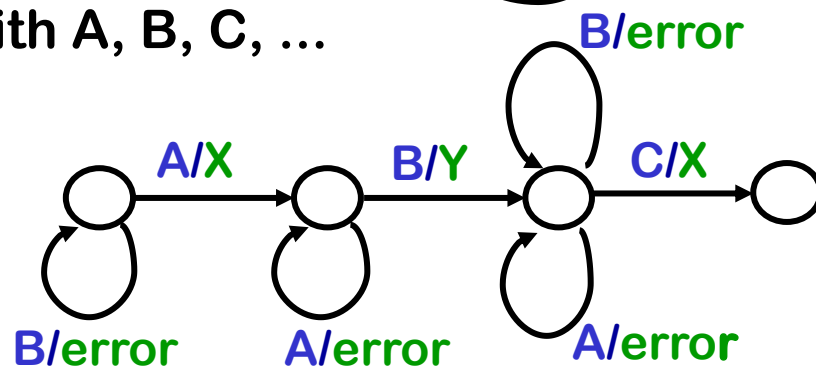
Just try out many sequences of **inputs**, and observe **outputs**

Eg. suppose input **A** results in output **X** 

- If second input **A** results in *different* output **Y** 
- If second input **A** results in the *same* output **X** 

Now try more sequences of inputs with A, B, C, ...

to e.g. infer



The inferred Mealy machine is an **under-approximation** of real system

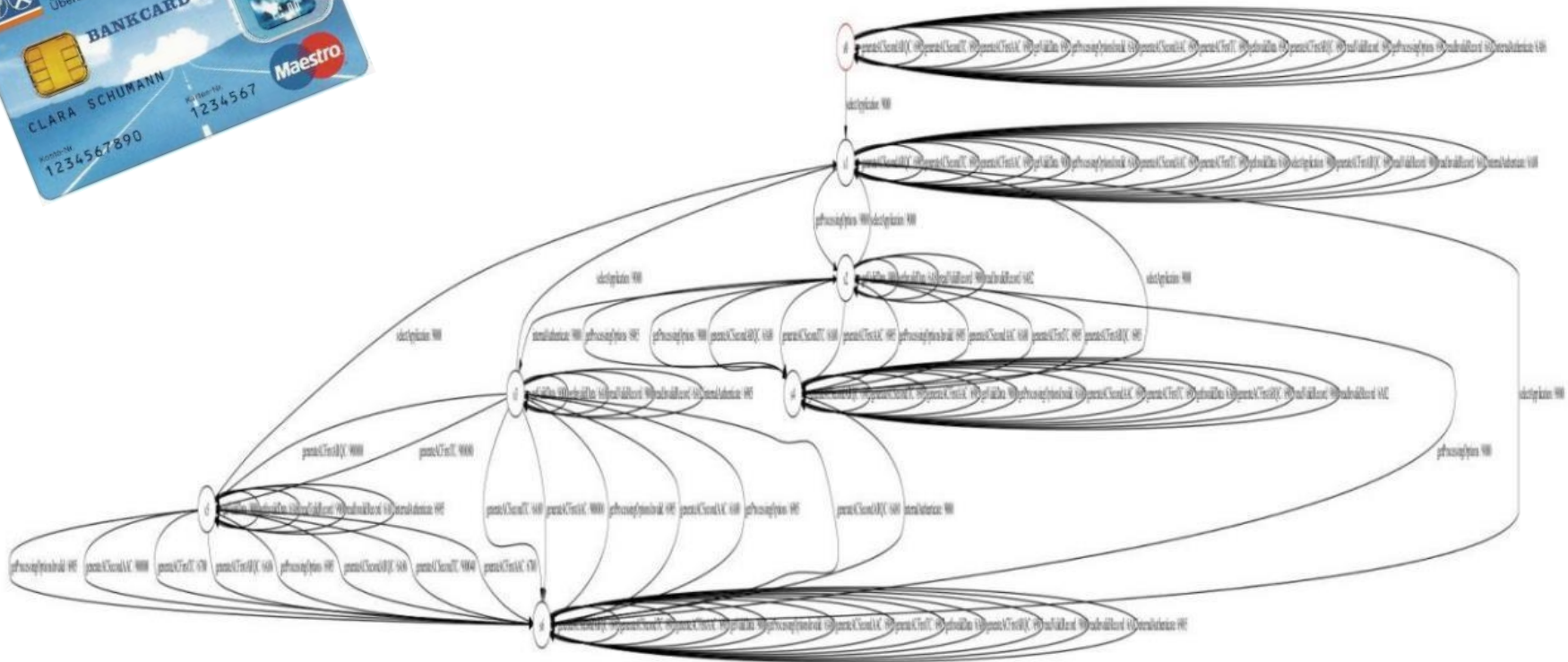
L* algorithm [Angluin 1987], implemented (in improved form) in e.g. **LearnLib**

Case study: EMV

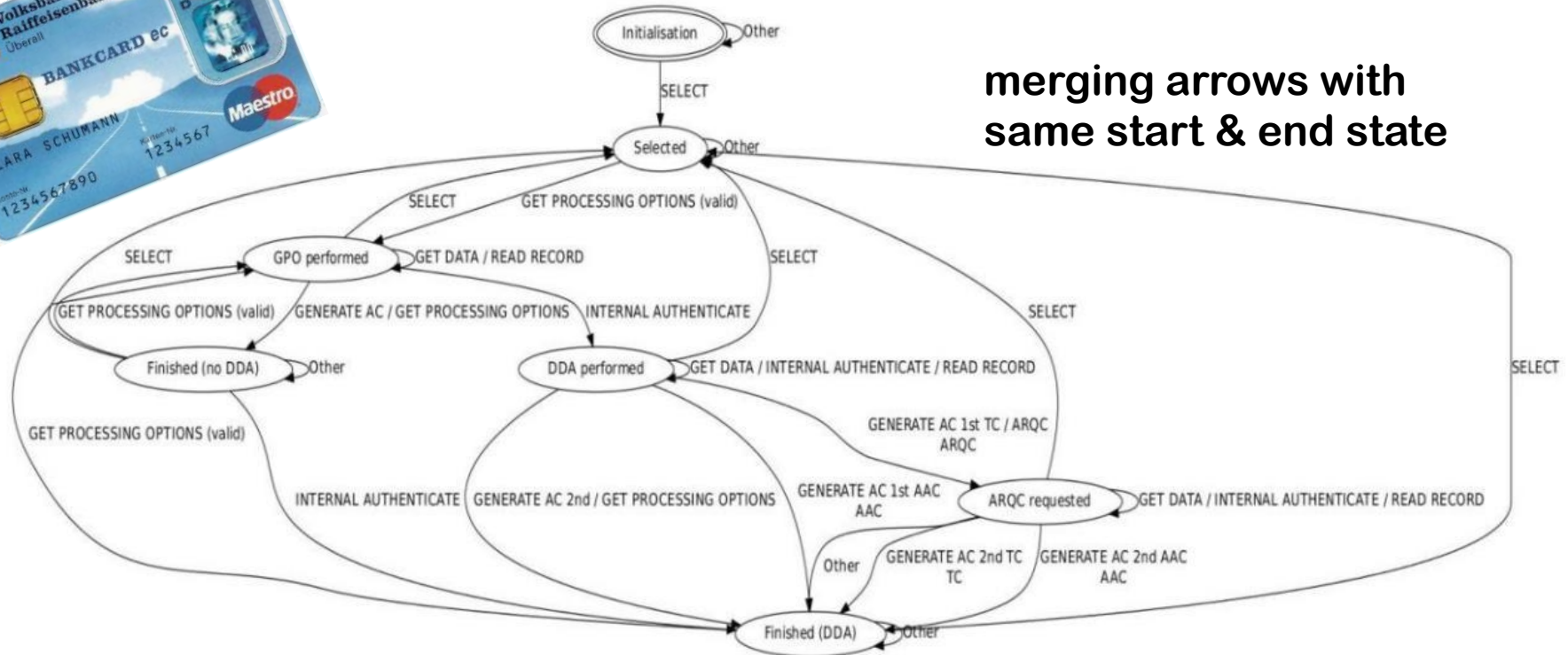
- Most banking smartcards implement a variant of EMV
 - EMV = Europay-Mastercard-Visa
- Specification in 4 books totalling > 700 pages
- Contactless payments: another 7 books with > 2000 pages



State machine inference of card



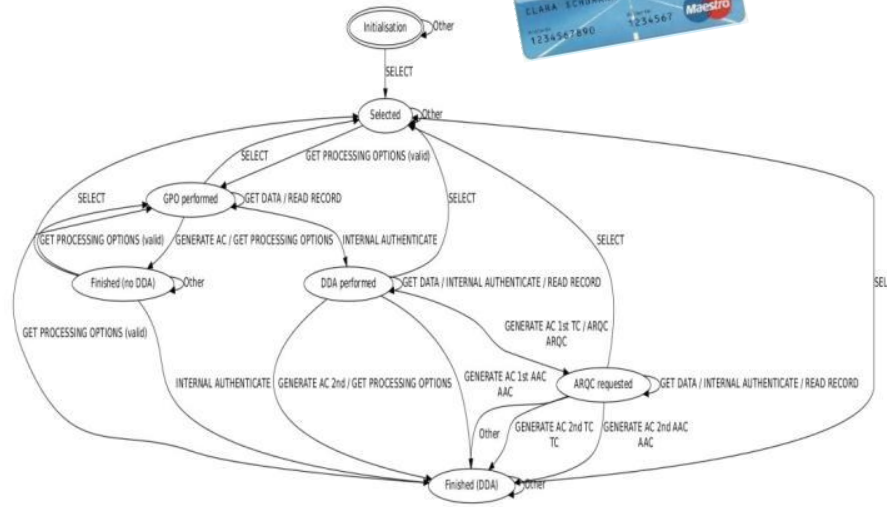
State machine inference of card



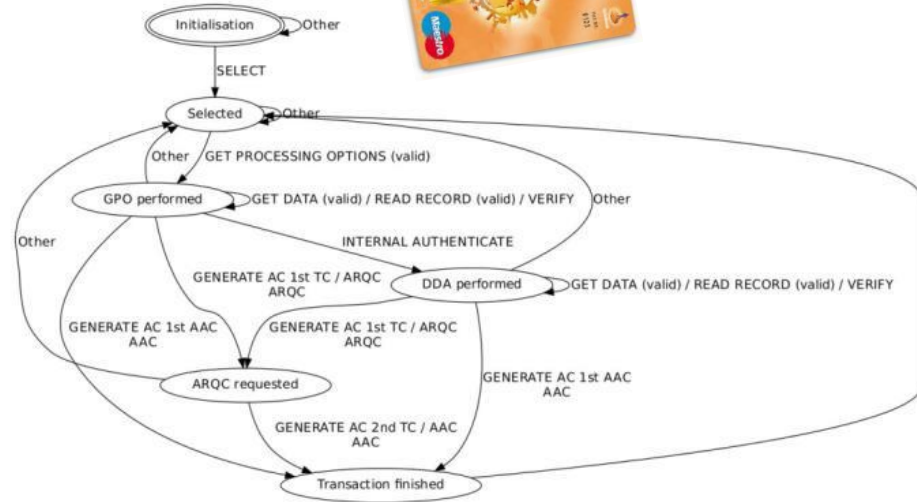
We found no bugs, but lots of variety between cards.

[Fides Aarts et al., *Formal models of bank cards for free*, SECTEST 2013]

Using state machines for comparison



Volksbank Maestro implementation

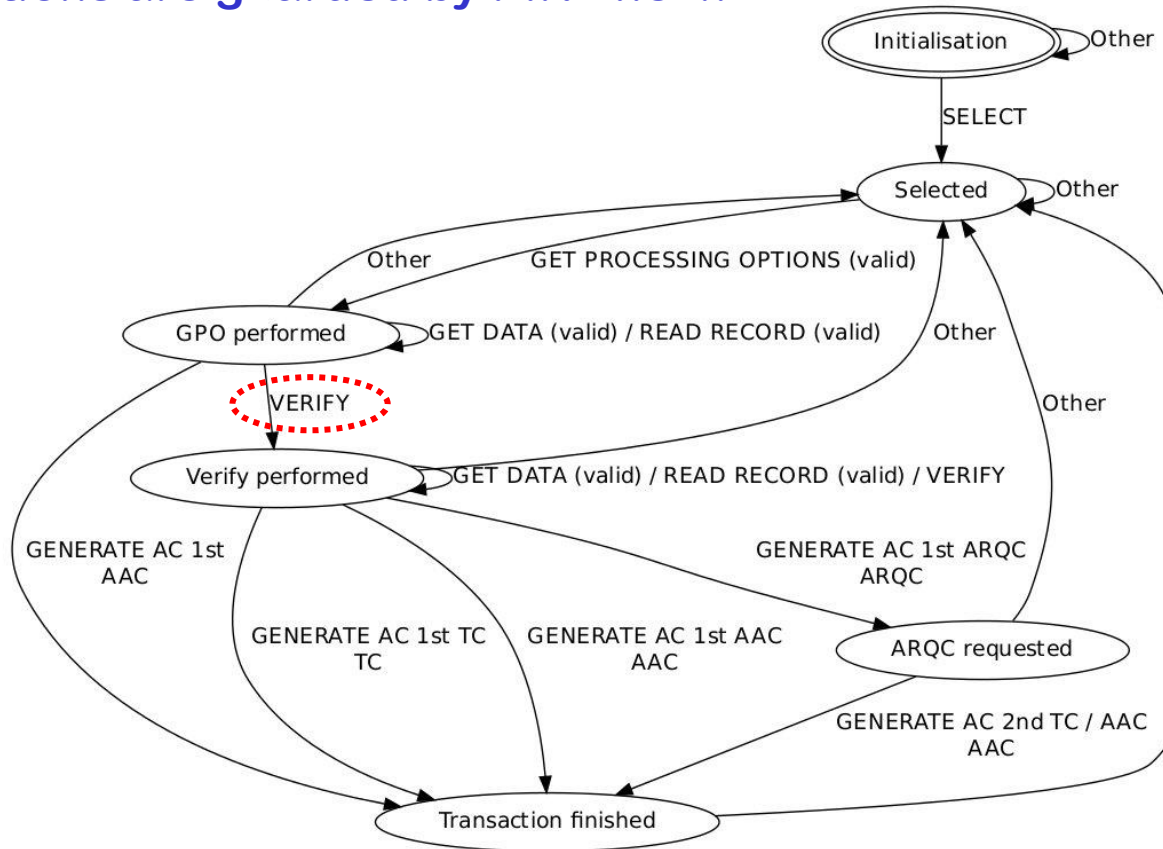


Rabobank Maestro implementation

Are both implementations correct & secure? Or compatible?

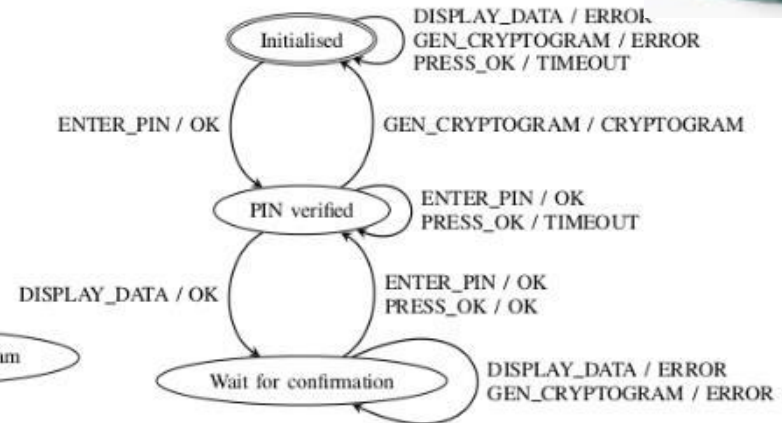
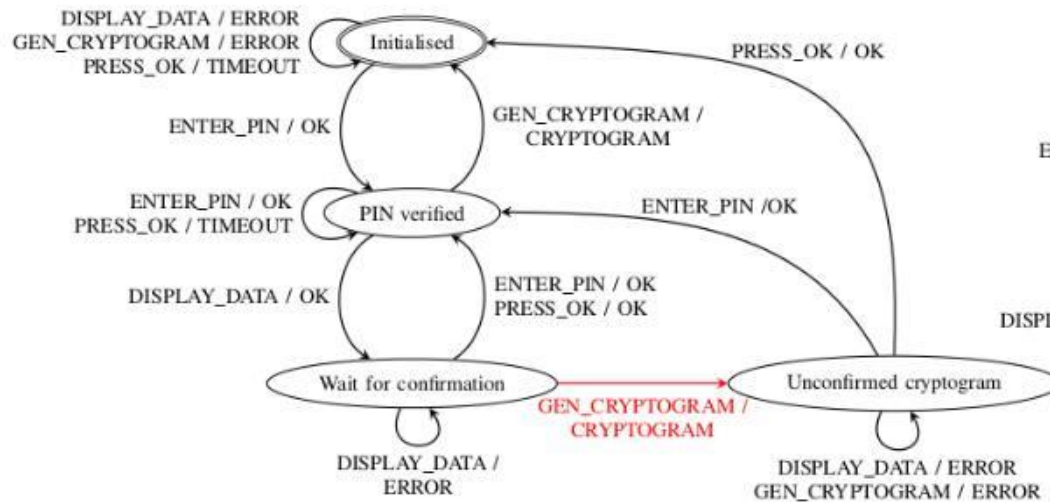
Using state machine for security analysis

Which actions are guarded by PIN check?



State machine inference of banking tokens

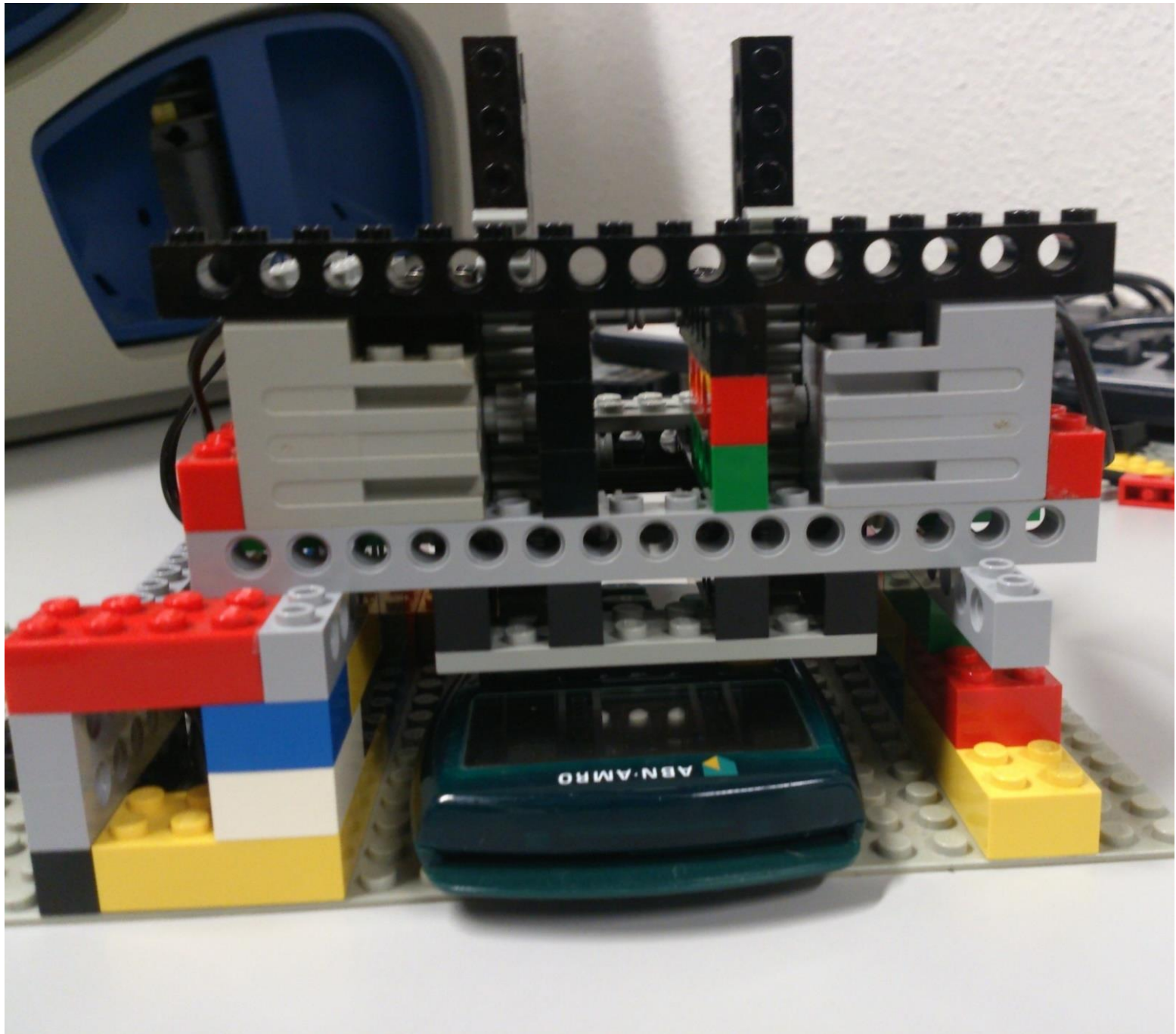
State machines inferred for flawed & patched device

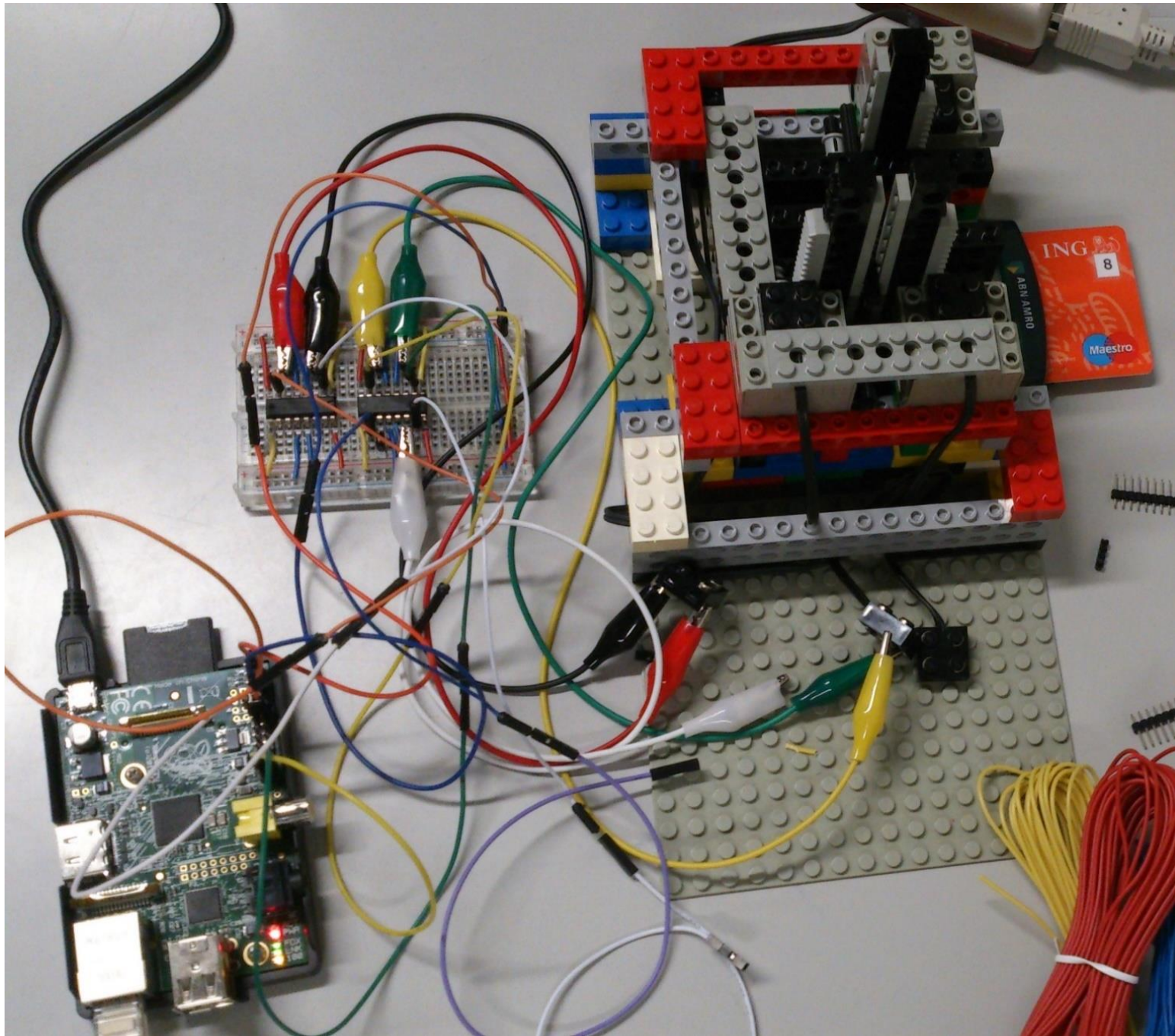


[Georg Chalupar, Stefan Peherstorfer, Erik Poll, Joeri de Ruyter, *Automated reverse engineering using Lego*, WOOT 2014]

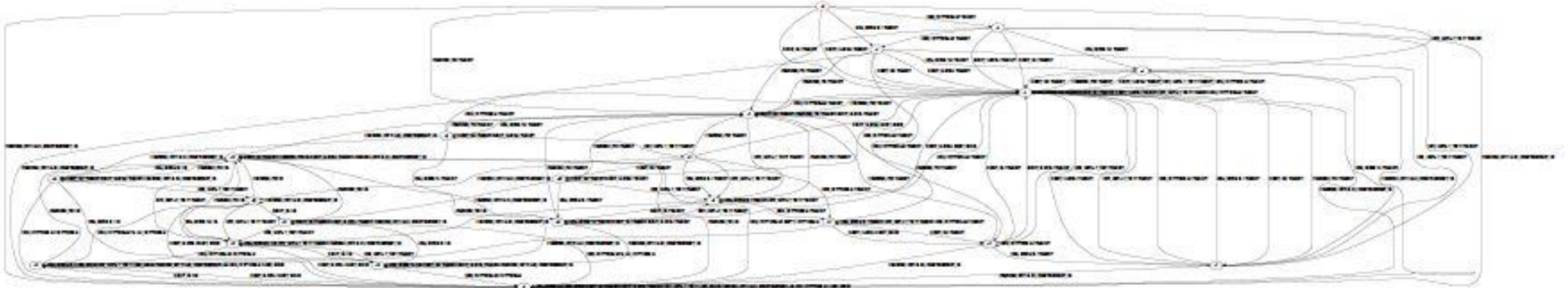
Movie at <http://tinyurl.com/legolearn>







State machine of Gemalto internet banking device



Complete inferred state machine

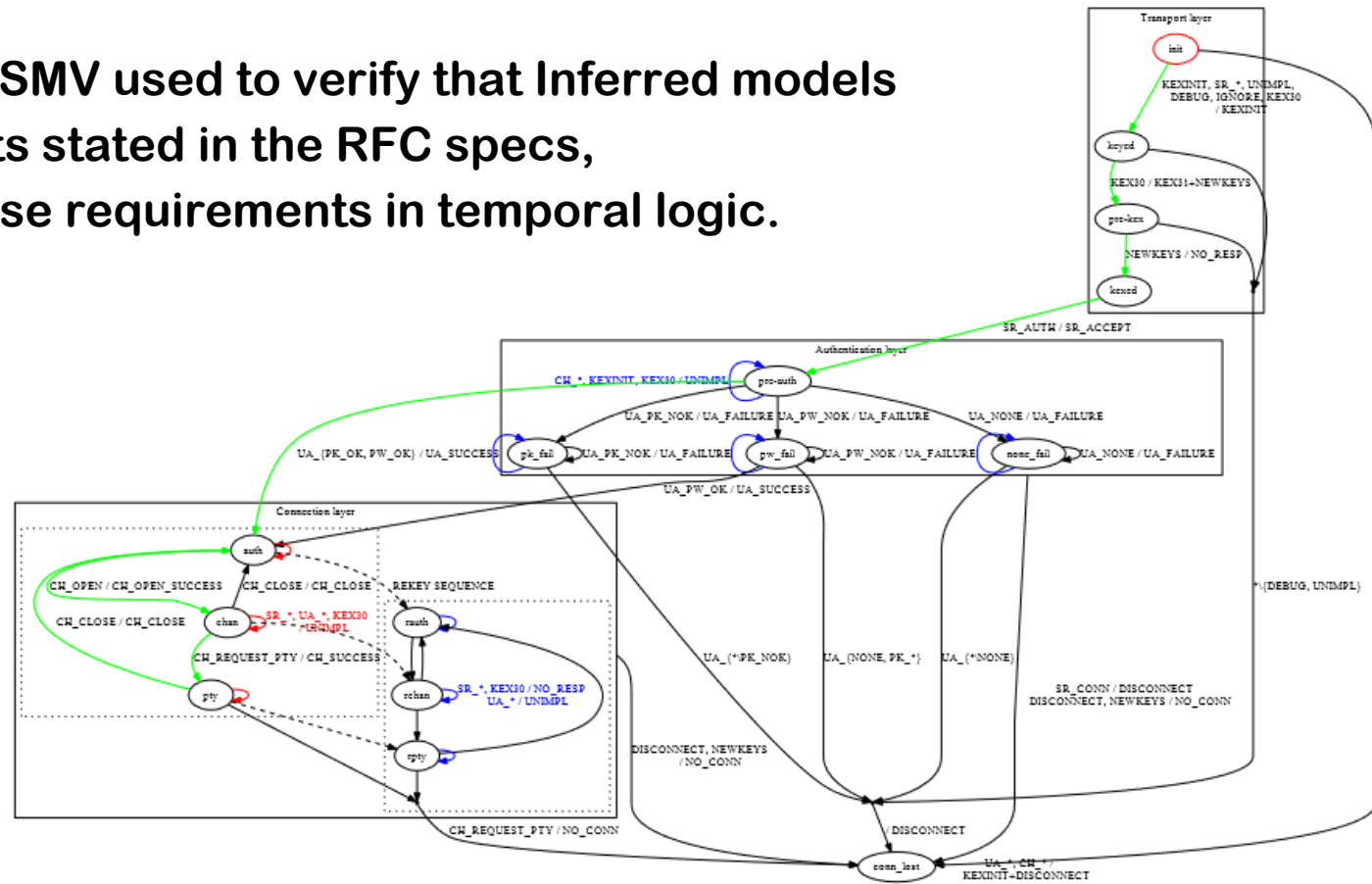
Would you trust this to be secure?

Did designers of Gemalto SWYS (Sign What You See) really intend all this complexity?



State machine inference of SSH implementations

Modelchecker NuSMV used to verify that Inferred models meet requirements stated in the RFC specs, by expressing these requirements in temporal logic.



[Paul Fiterau-Brosteau, Toon Lenaerts, Erik Poll, Joeri de Ruiter, Frits Vaandrager and Patrick Verleg, *Model Learning and Model Checking of SSH Implementations*, SPIN 2017]

Typical prose specifications: SSH ☹️

Quote from the RFCs defining SSH:

“Once a party has sent a SSH_MSG_KEXINIT message for key exchange or re-exchange, until it has sent a SSH_MSG_NEWKEYS message, it MUST NOT send any messages other than:

- *Transport layer generic messages (1 to 19) (but SSH_MSG_SERVICE_REQUEST and SSH_MSG_SERVICE_ACCEPT MUST NOT be sent);*
- *Algorithm negotiation messages (20 to 29) (but further SSH_MSG_KEXINIT messages MUST NOT be sent);*
- *Specific key exchange method messages (30 to 49).*

The provisions of Section 11 apply to unrecognised messages”

Understanding state machines from prose is hard!

This is another instance of the issues raised by LangSec: poor, informal specification of input formats

[Erik Poll, Joeri de Ruyter and Aleksy Schubert, *Protocol state machines and session languages*, LangSec 2015]



3. Fuzzing of stateful systems

(work in progress)

Stateful fuzzing

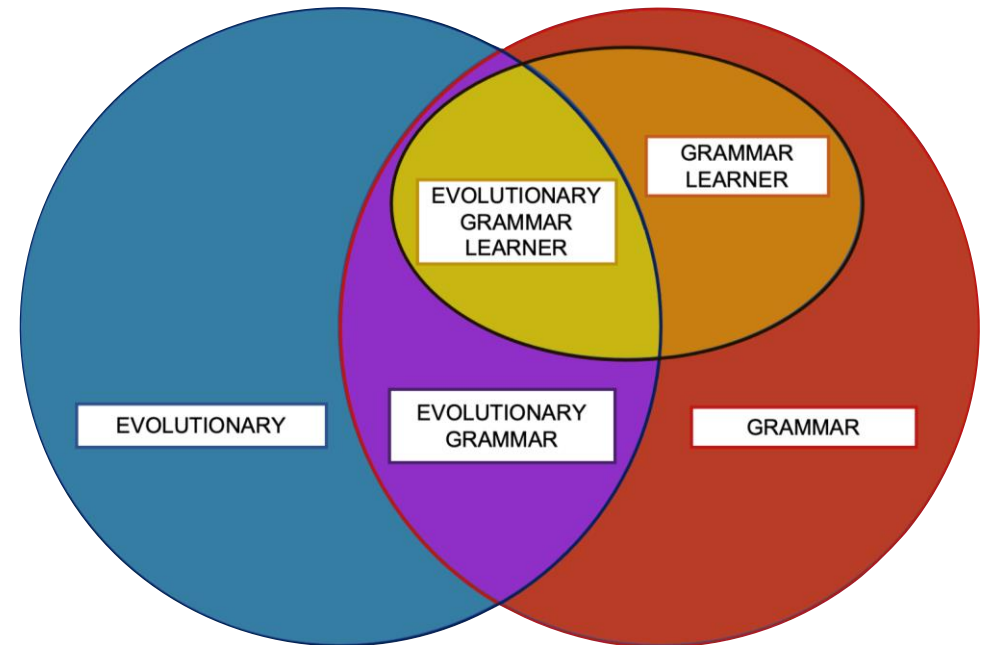
- **State machine inference** is a limited form of **fuzzing**:
 - it does not use **strange/malformed input messages**,
 - but only **strange *sequences of normal* input messages**
- Given the success of fuzzing, how can we combine this with fuzzing for fuzzing stateful systems?
 - Note: fuzzing is successful because of the problems signalled by LangSec
- Not only for cryptographic protocols!
 - In fact, fuzzing cryptographic protocols is hard, as it requires a custom test harness that is labour-intensive to make

Survey of fuzzers for stateful systems [ArXiv:2301.02490]

There are many fuzzers around, but not that many for stateful systems

7 categories of stateful fuzzers

- Using different combinations of white box, grey box, black box
- Some infer a state machine, using active or passive learning



Open questions:

- *What are the optimal combinations?*
- *Should implementations be made “fuzzer-friendly”?*

[Cristian Daniele, Seyed Andarzian, Erik Poll, *Fuzzers for stateful systems: Survey and Research Directions*, ArXiv:2301.02490, 2023]

Conclusions

- Most security flaws are **INPUT** processing flaws
- These flaws arise in **PARSING** of many **input languages / formats**
 - 1) **buggy/insecure parsing** or 2) **unintended parsing**
- LangSec identifies root causes and points to structural solutions
- Particular case of input format: *sequence of messages in a protocol*
- We can **automatically extract state machine** of such behaviour using **state machine learning** (aka **active learning**)
- State machine learning is limited form of fuzzing.
How can we best fuzz stateful systems?

Should protocols be implemented in a fuzzer-friendly way?

*Also use passive learning using **FlexFringe**? [Verwer & Hammerschidt, 2022 arXiv2203.16331]*