Security Testing of Stateful Systems

Erik Poll

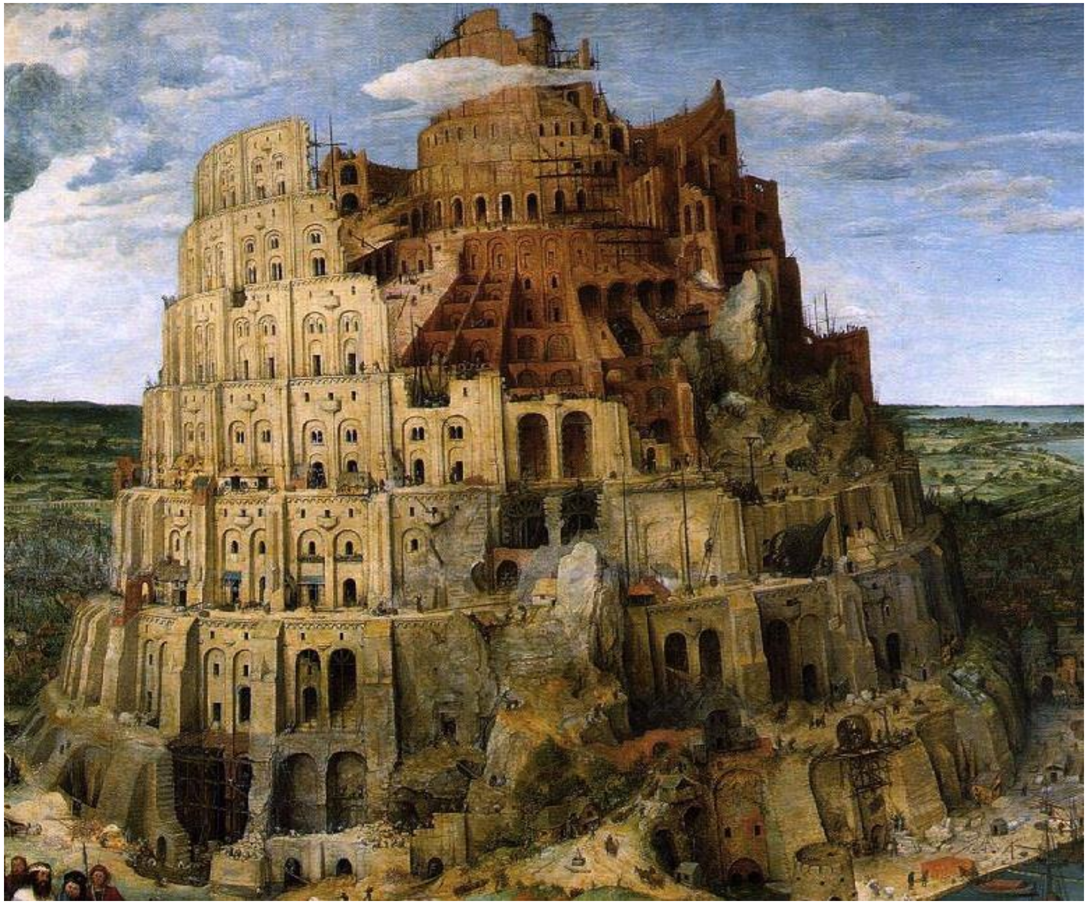
joint work with several PhD & MSc students

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]

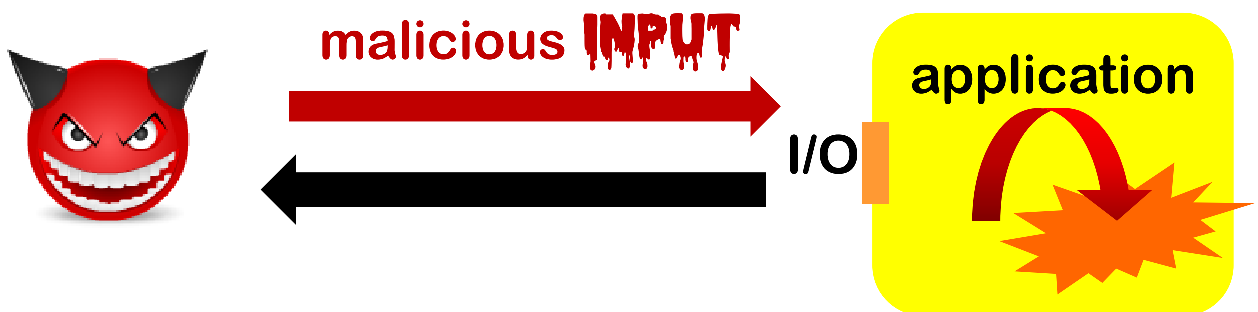
1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging & Monitoring

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

ONE main bug category: INPUT handling



More in particular: PARSING of input is dangerous

Garbage In, Garbage Out

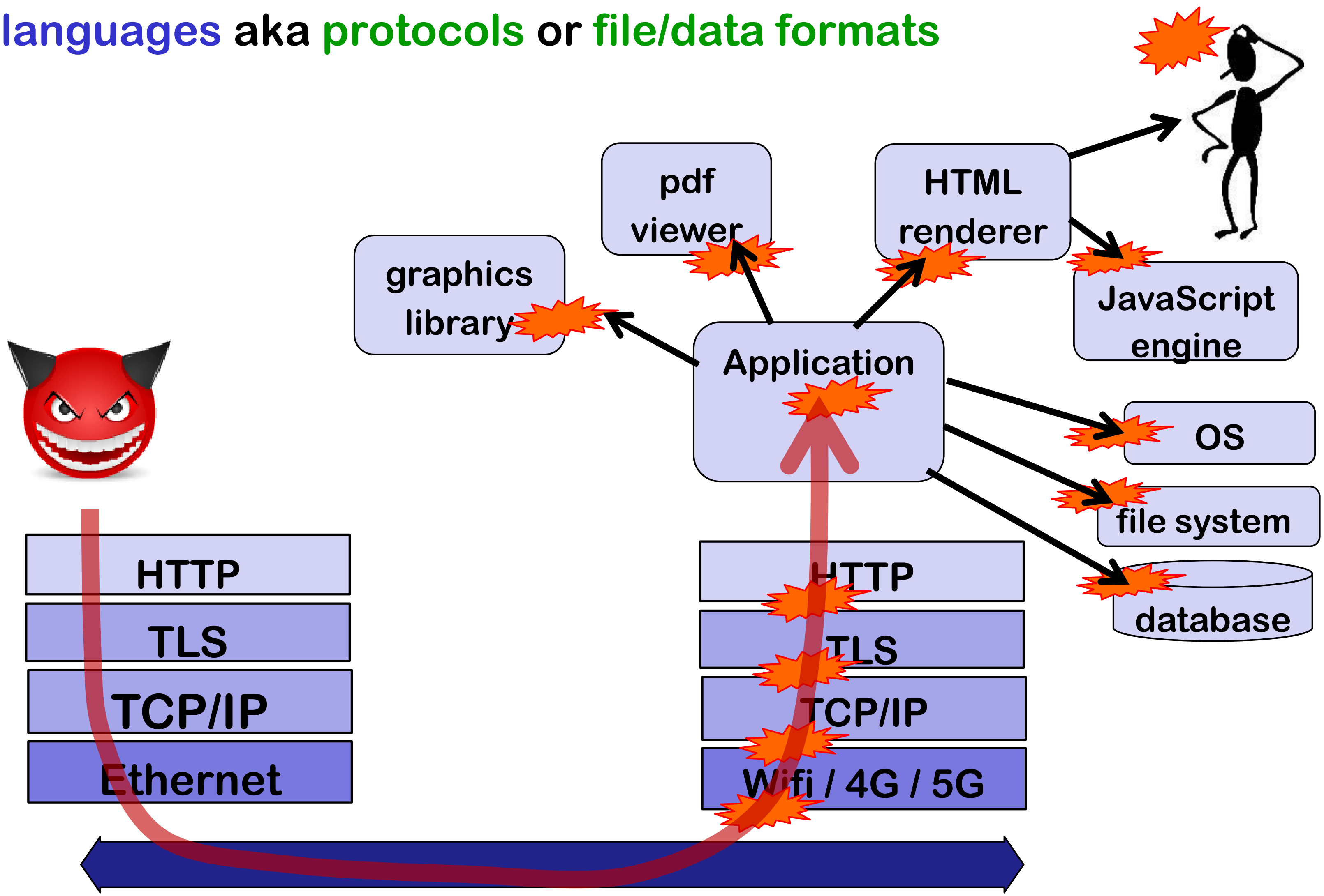
becomes *Malicious* Garbage In, *Security Incident* Out

or Garbage In, Evil Out

Parsing of many languages in many places

Parsing aka decoding, interpreting or processing

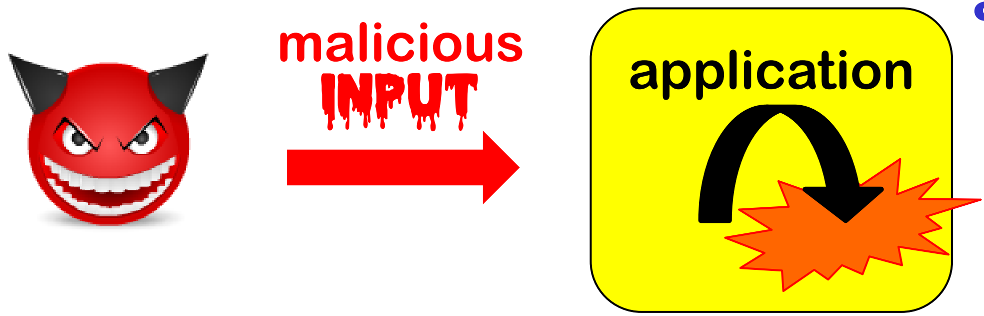
Input languages aka protocols or file/data formats



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

1. Processing Flaws

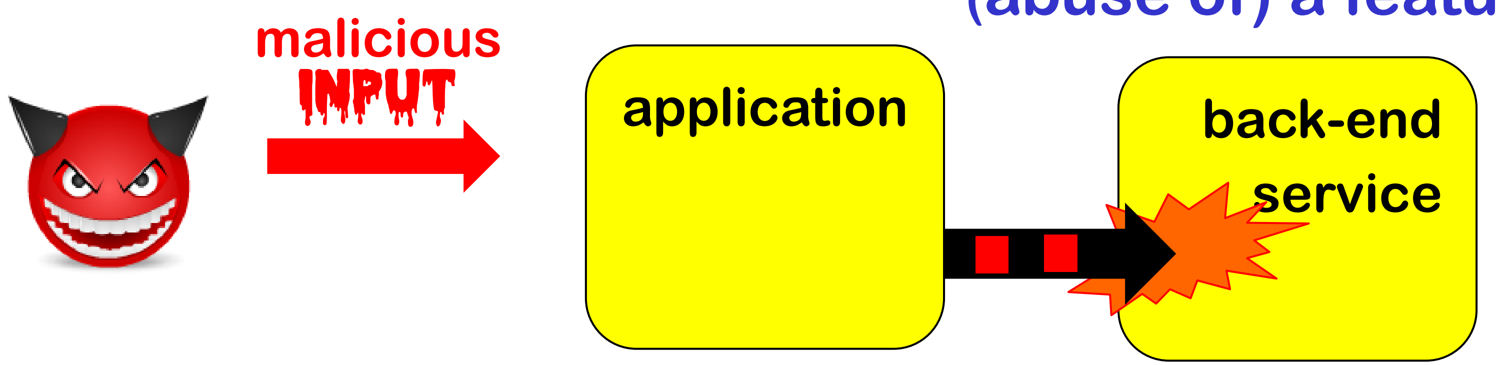
a bug !



eg. buffer overflow in
PDF or JPG viewer

2. Injection Flaws

(abuse of) a feature !

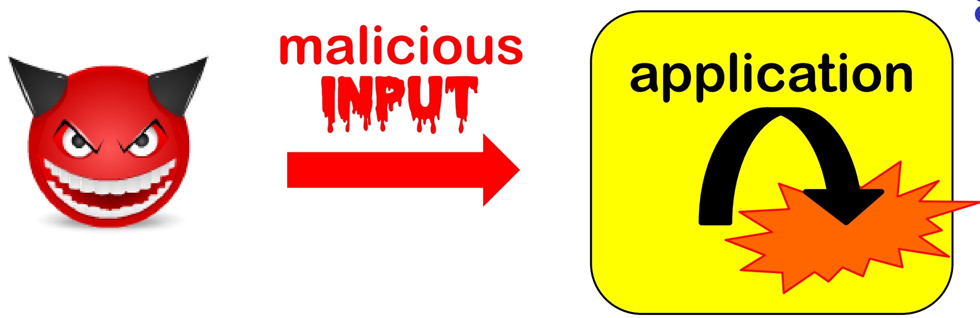


eg. SQL injection, XSS,
Word Macros

Or 1) insecure parsing and 2) unintended parsing

1. *Insecure, buggy* parsing

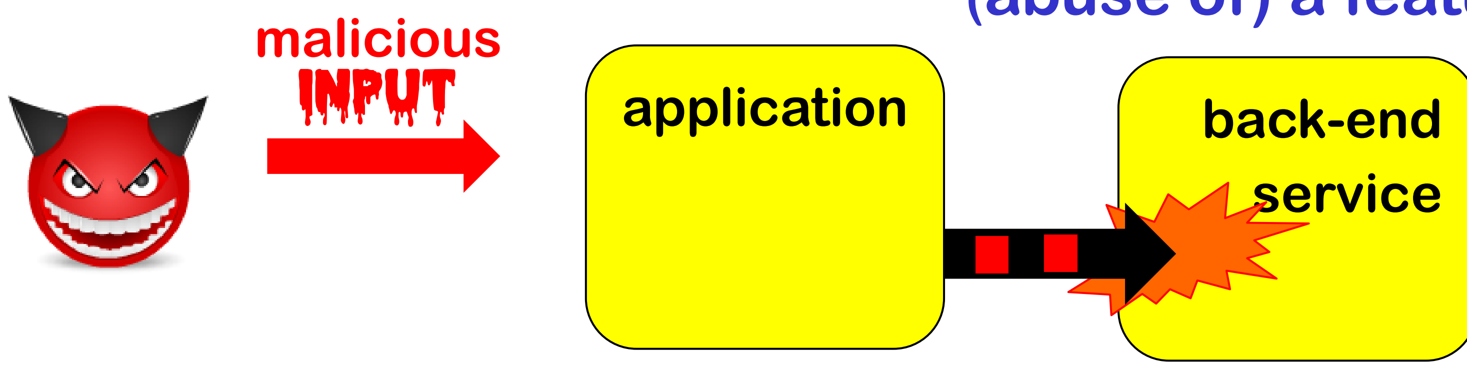
a bug !



eg. buffer overflow in
PDF or JPG viewer

2. *Unintended* parsing

(abuse of) a feature !



eg. SQL injection, XSS,
Word Macros

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**

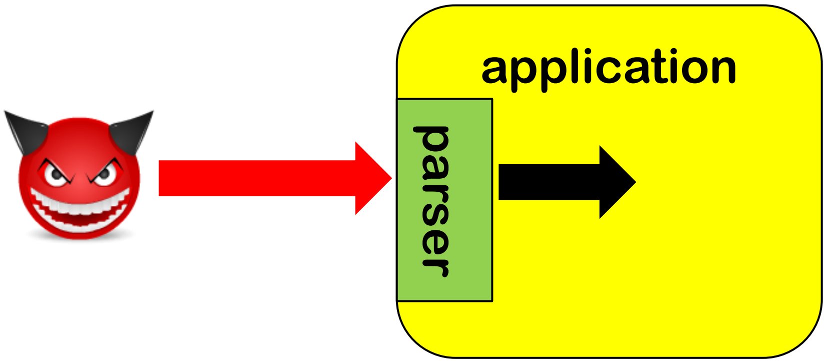
   ideally as regular expression or (E)BNF grammar

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

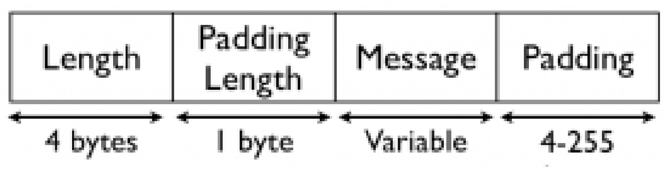
[Erik Poll, *Strings considered harmful*, USENIX :login; 2019]

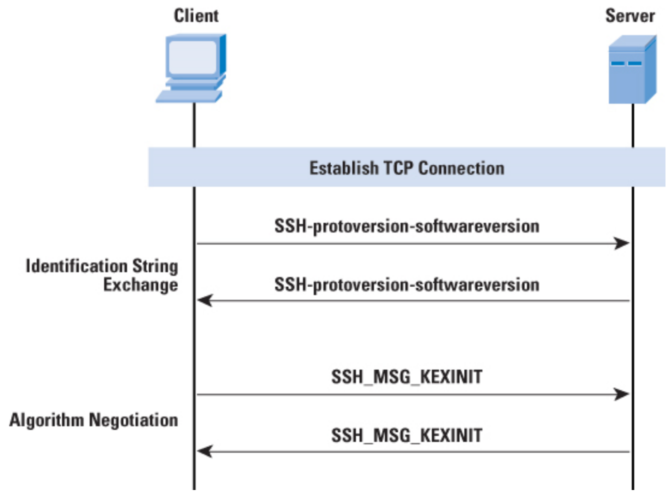
2. State machine learning for security

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



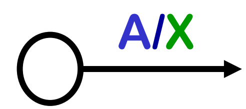


- **Most specs only describe the happy flow…**
- **For security, getting unhappy flows correct is crucial**
- **Fortunately, we can extract state machines from implementations using state machine inference aka active learning using just black box testing**

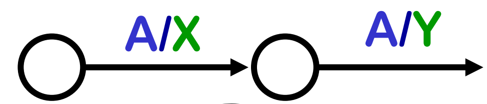
Active Learning aka State Machine Inference

Just try out many sequences of inputs, and observe outputs

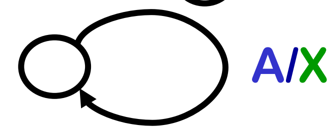
Eg. suppose input A results in output X



- If second input A results in *different* output Y

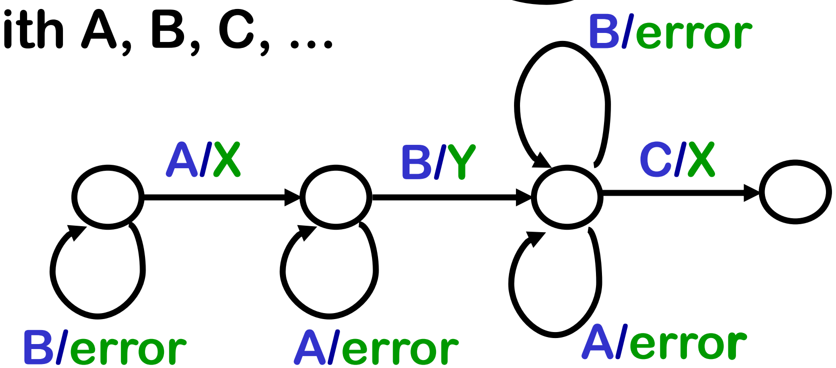


- If second input A results in the *same* output X



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

to e.g. infer



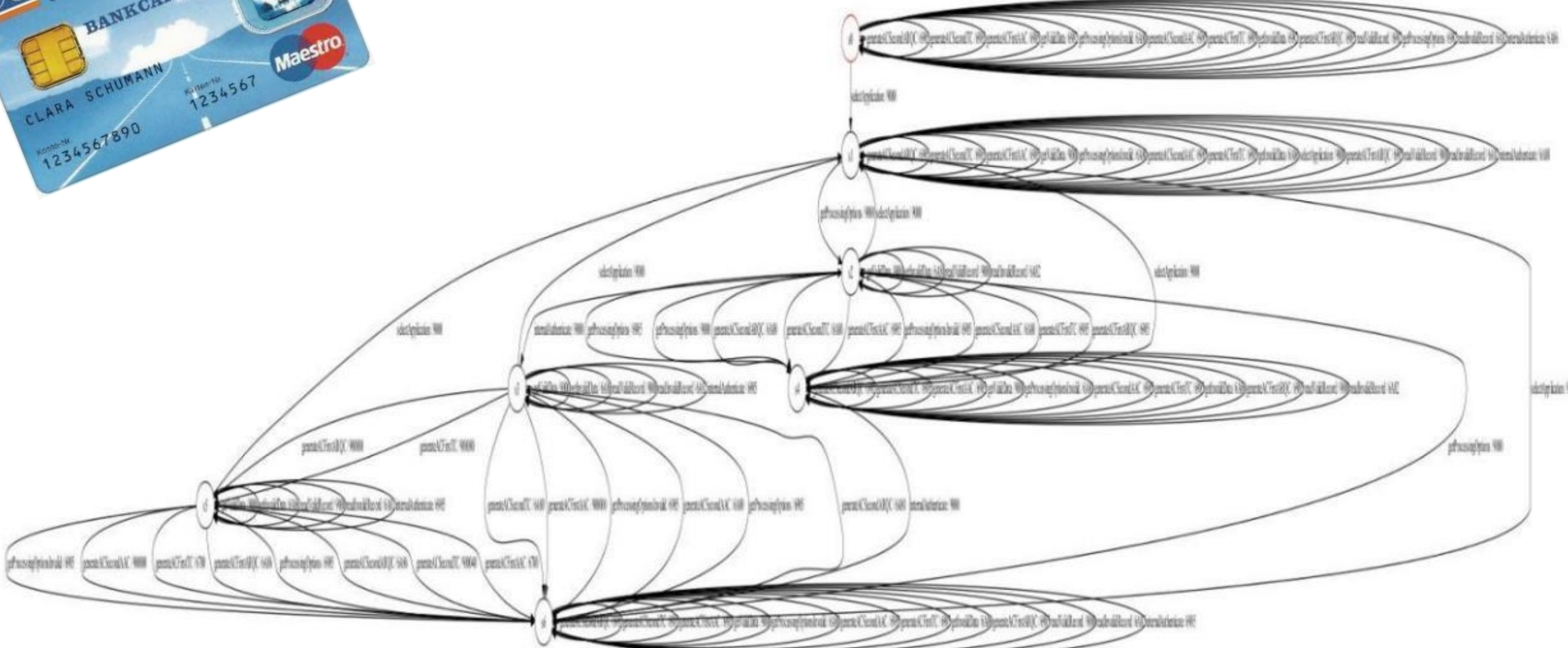
The inferred Mealy machine is an under-approximation of real system

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

Case study: EMV

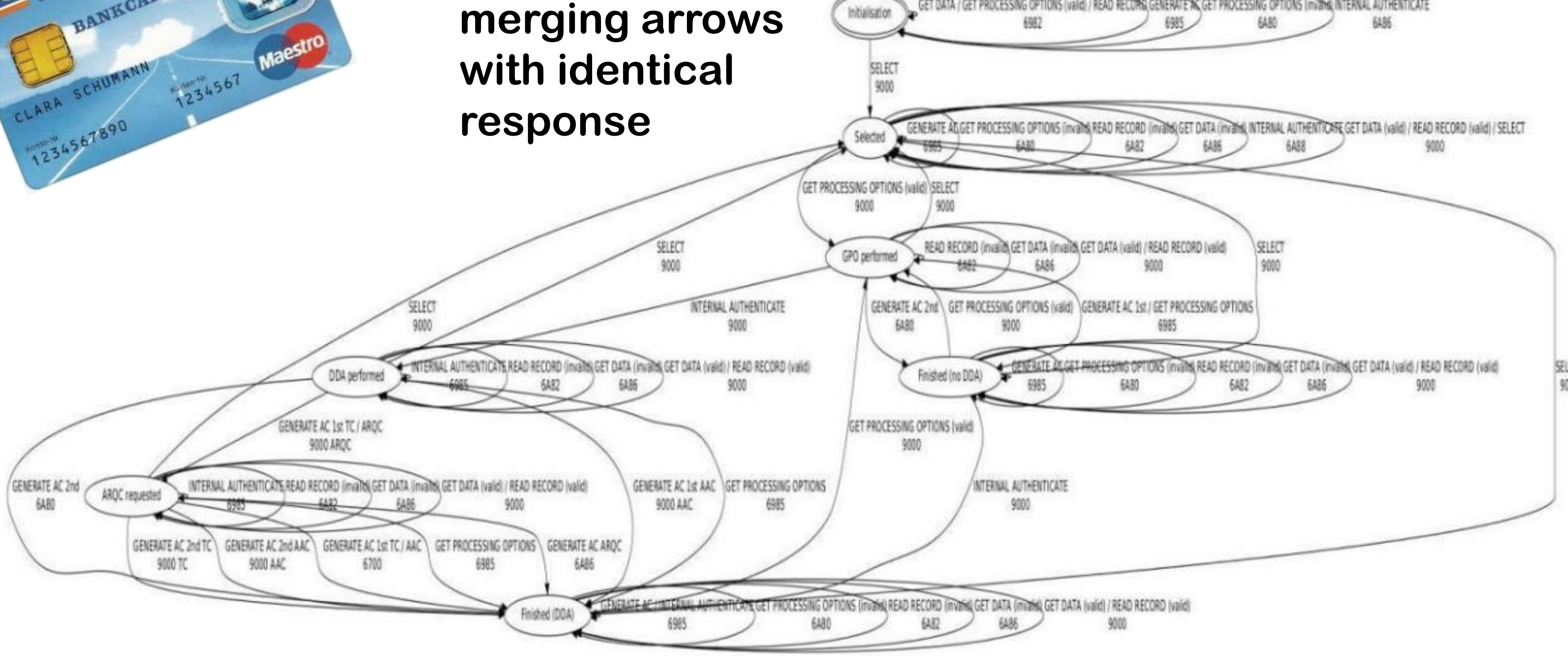
- Most banking smartcards implement a variant of EMV
  - EMV = Europay-Mastercard-Visa
- Specification in 4 books totalling > 700 pages
- Contactless payments: another 7 books with > 2000 pages

State machine inference of Maestro card

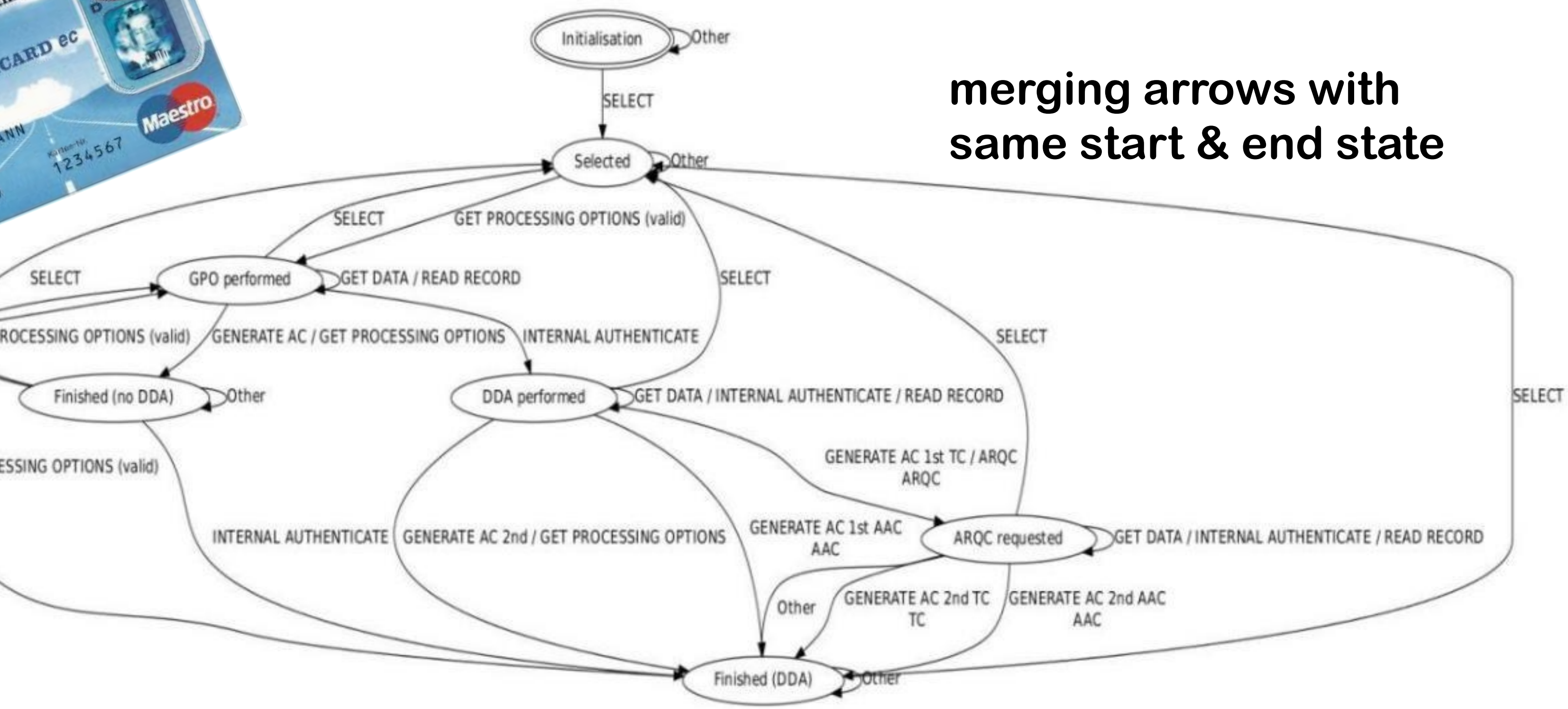


State machine inference of Maestro card

merging arrows with identical response



State machine inference of Maestro card

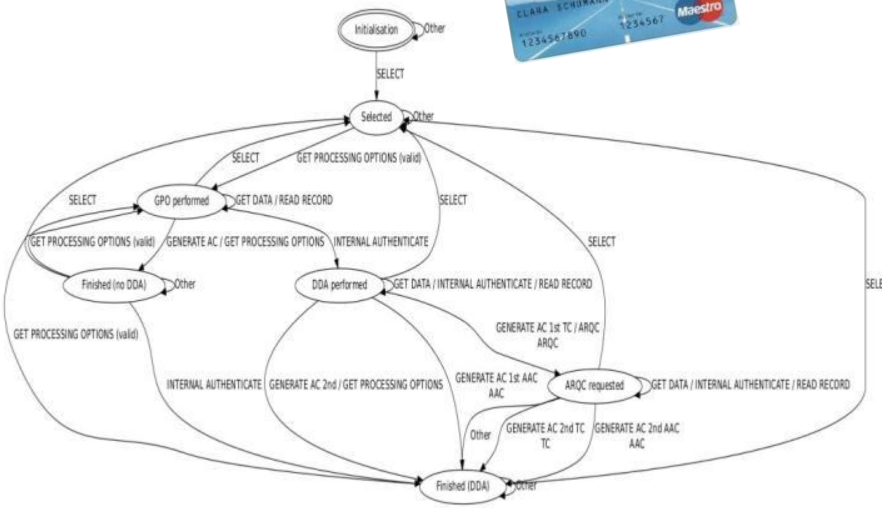


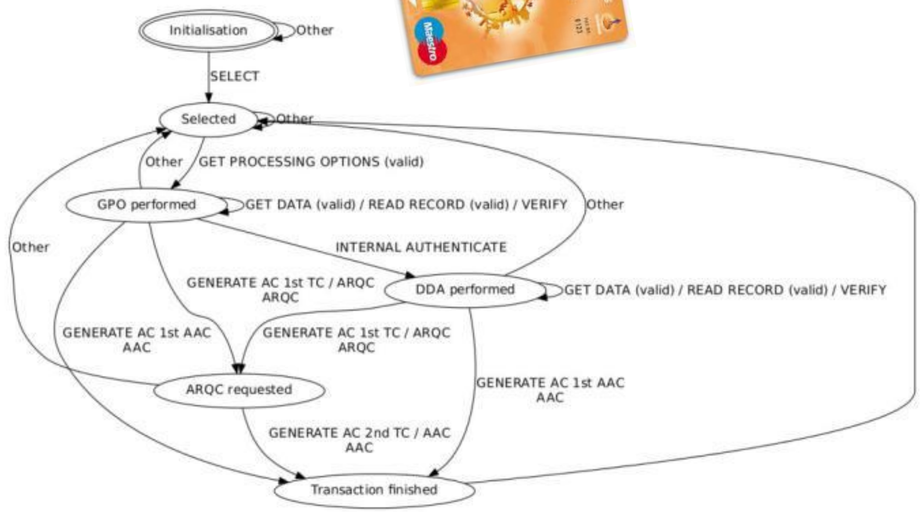
merging arrows with same start & end state

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





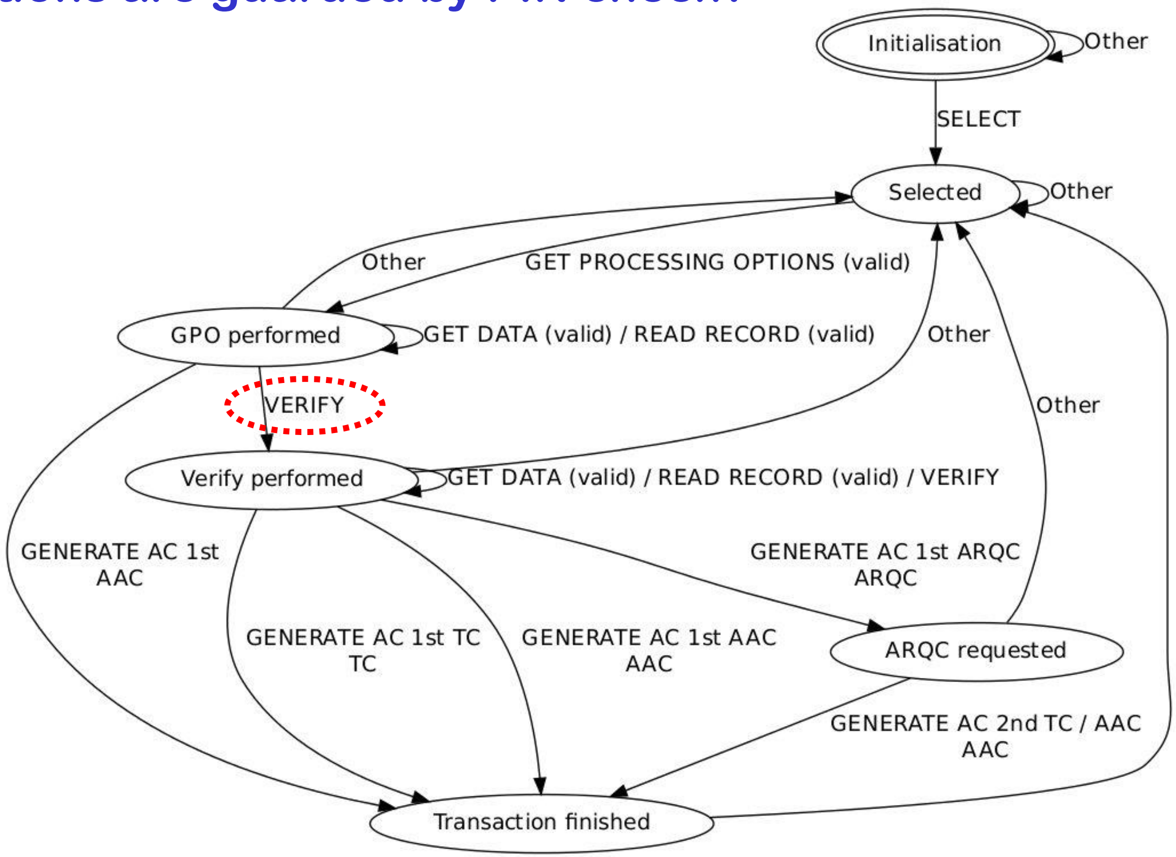
Volksbank Maestro implementation

Rabobank Maestro implementation

Are both implementations correct & secure? Or compatible?

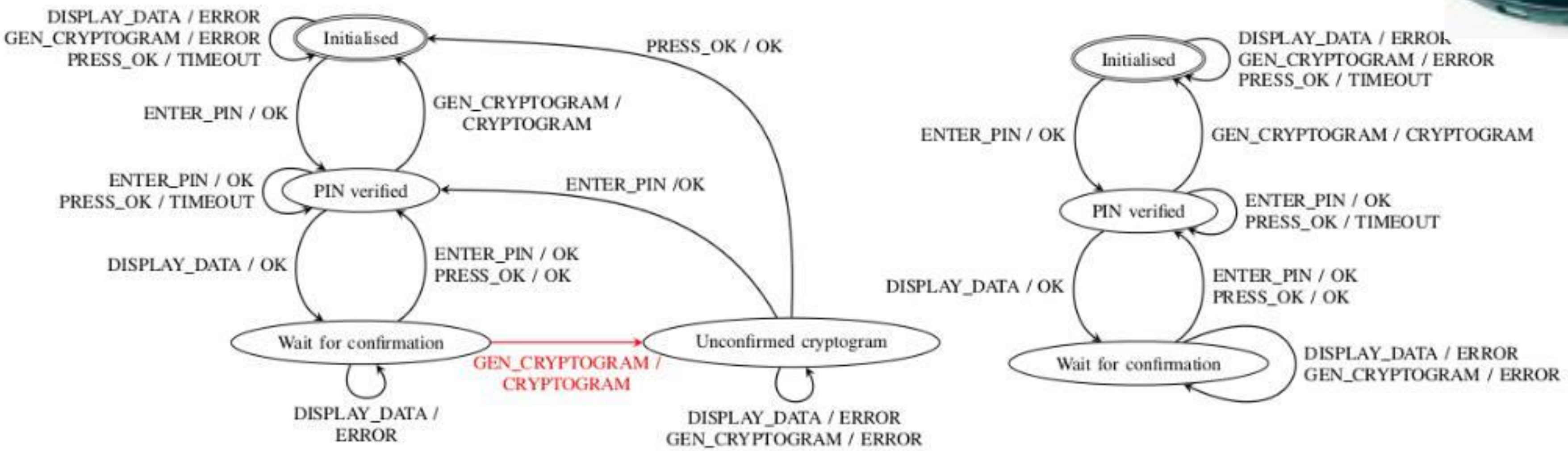
Using state machine for security analysis

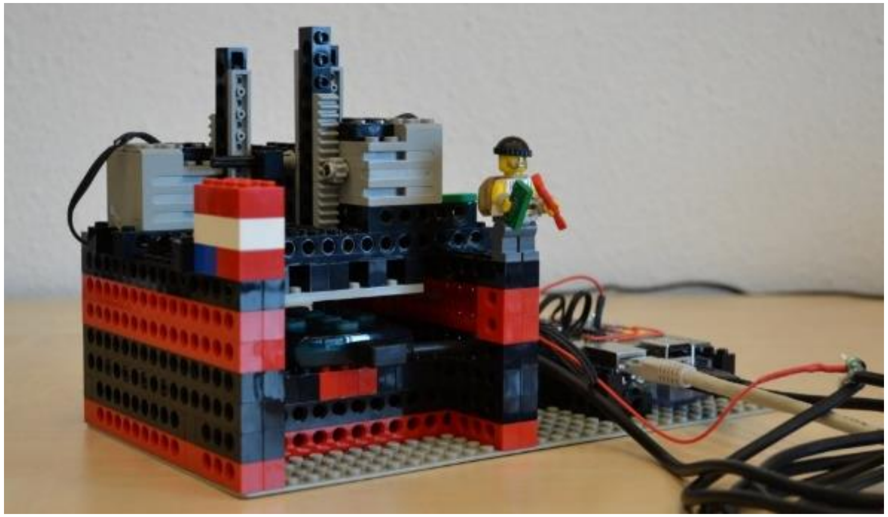
Which actions are guarded by PIN check?



State machine inference of banking tokens

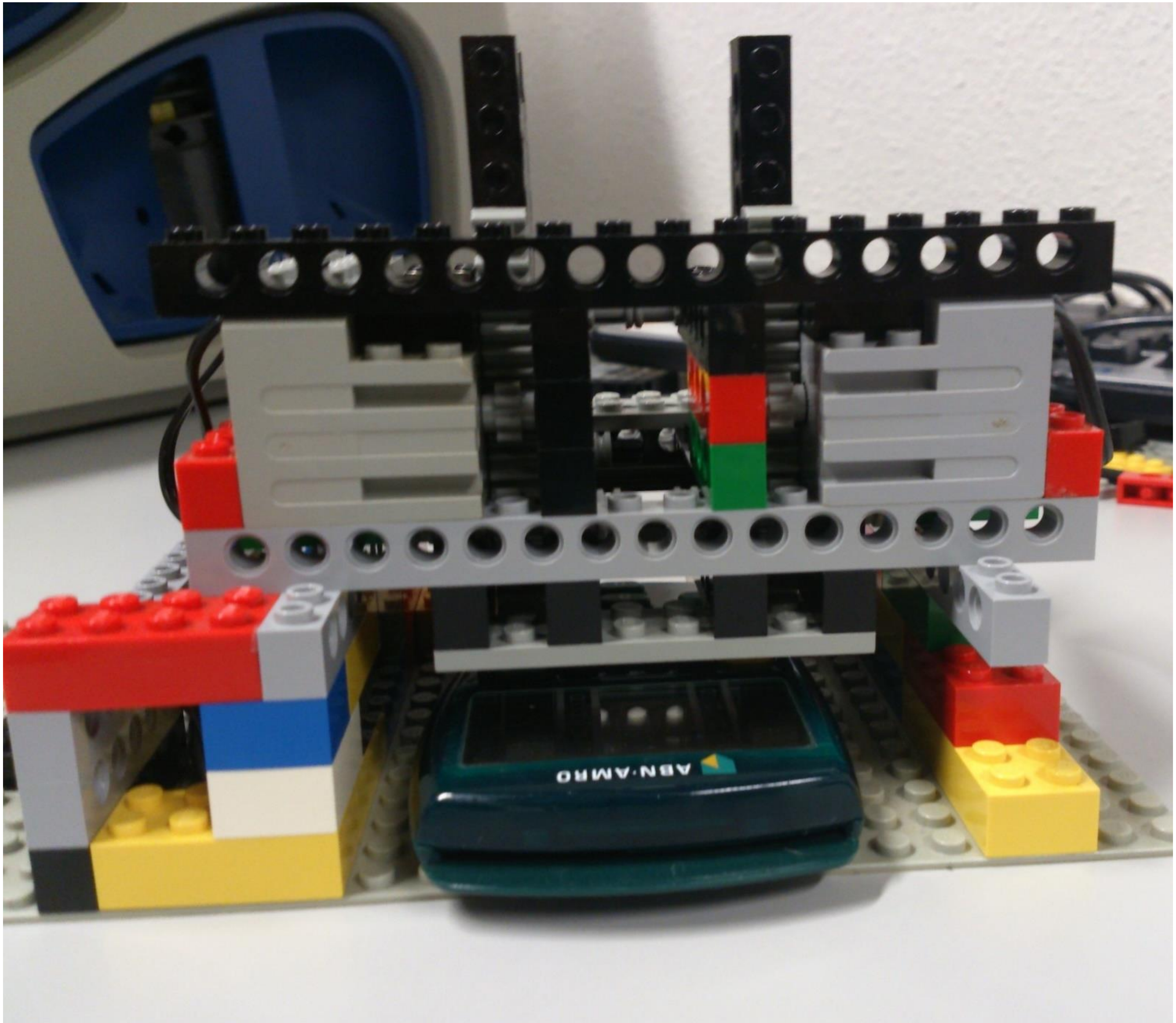
State machines inferred for flawed & patched device

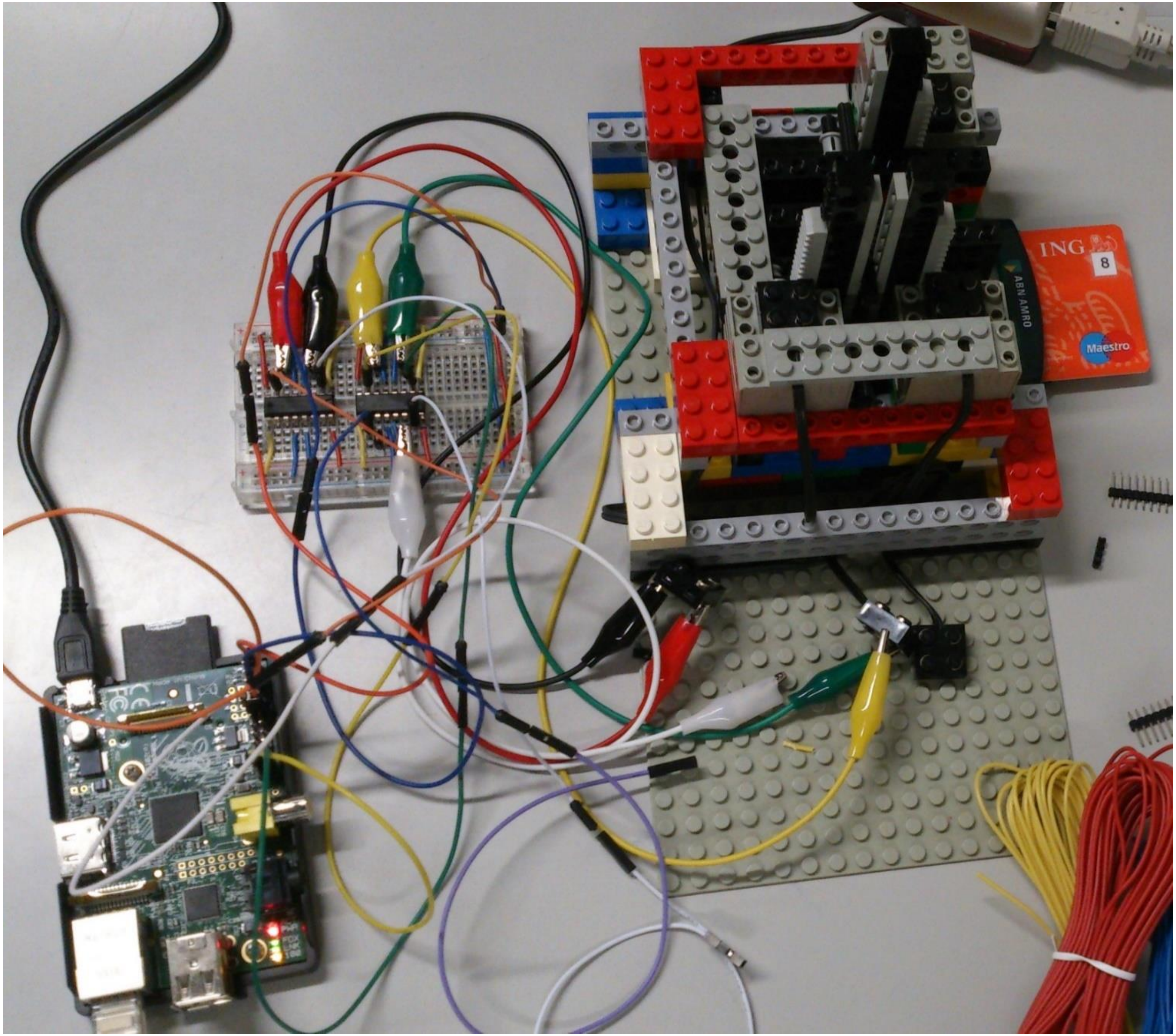




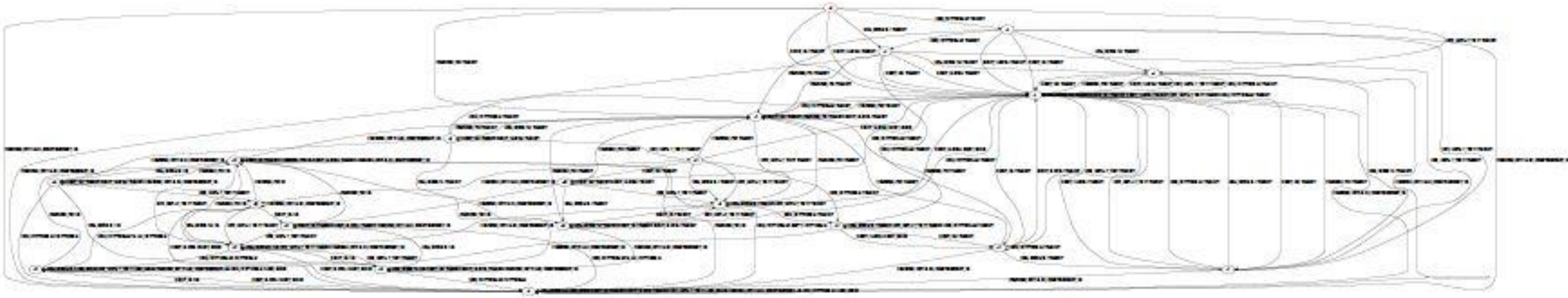
[Georg Chalupar, Stefan Peherstorfer, Erik Poll, Joeri de Ruiter, *Automated reverse engineering using Lego,* WOOT 2014]

Movie at http://tinyurl/legolearn


ABN·AMRO


ING
8
ABN·AMRO
Maestro

State machine of Gemalto internet banking device

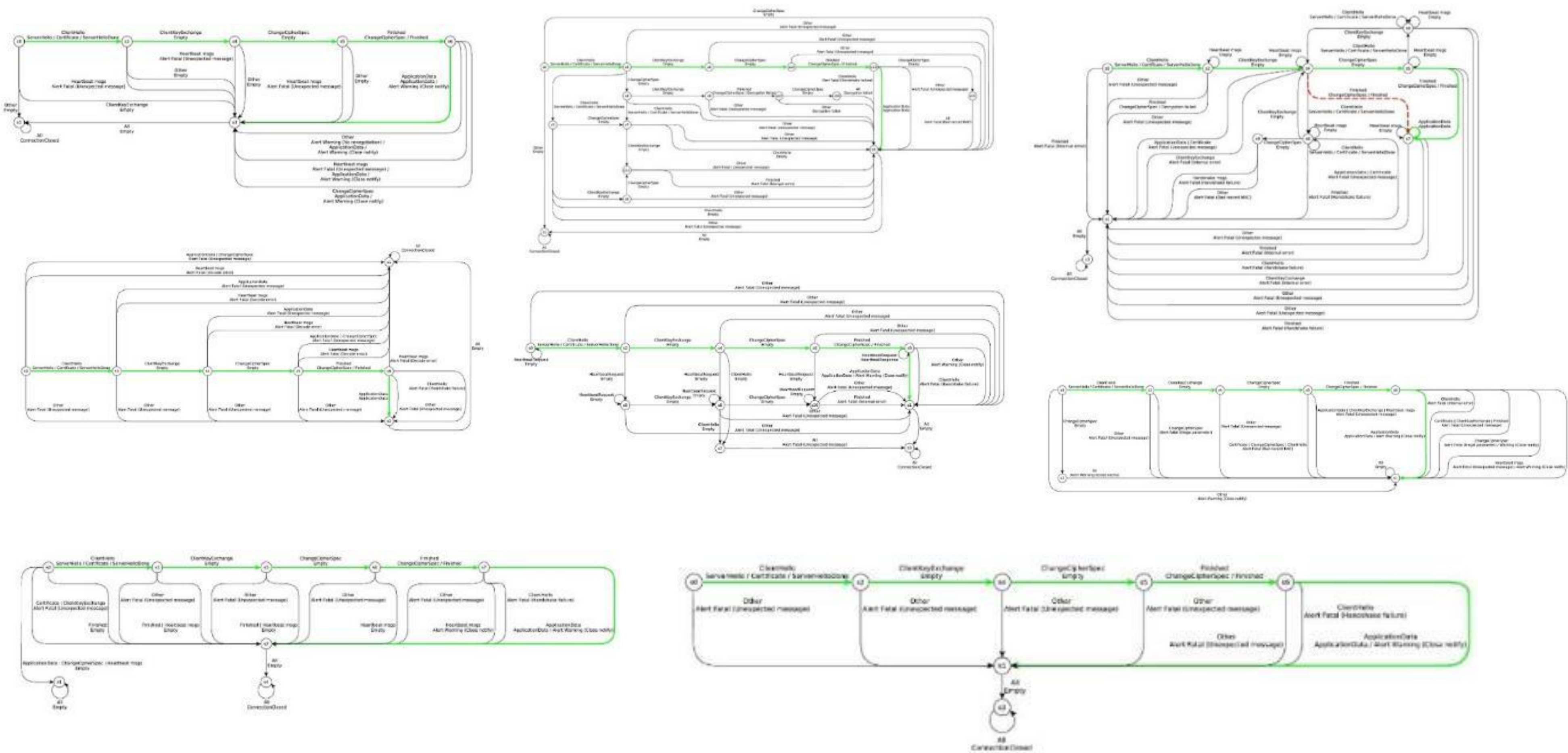


Complete inferred state machine

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 TLS implementation

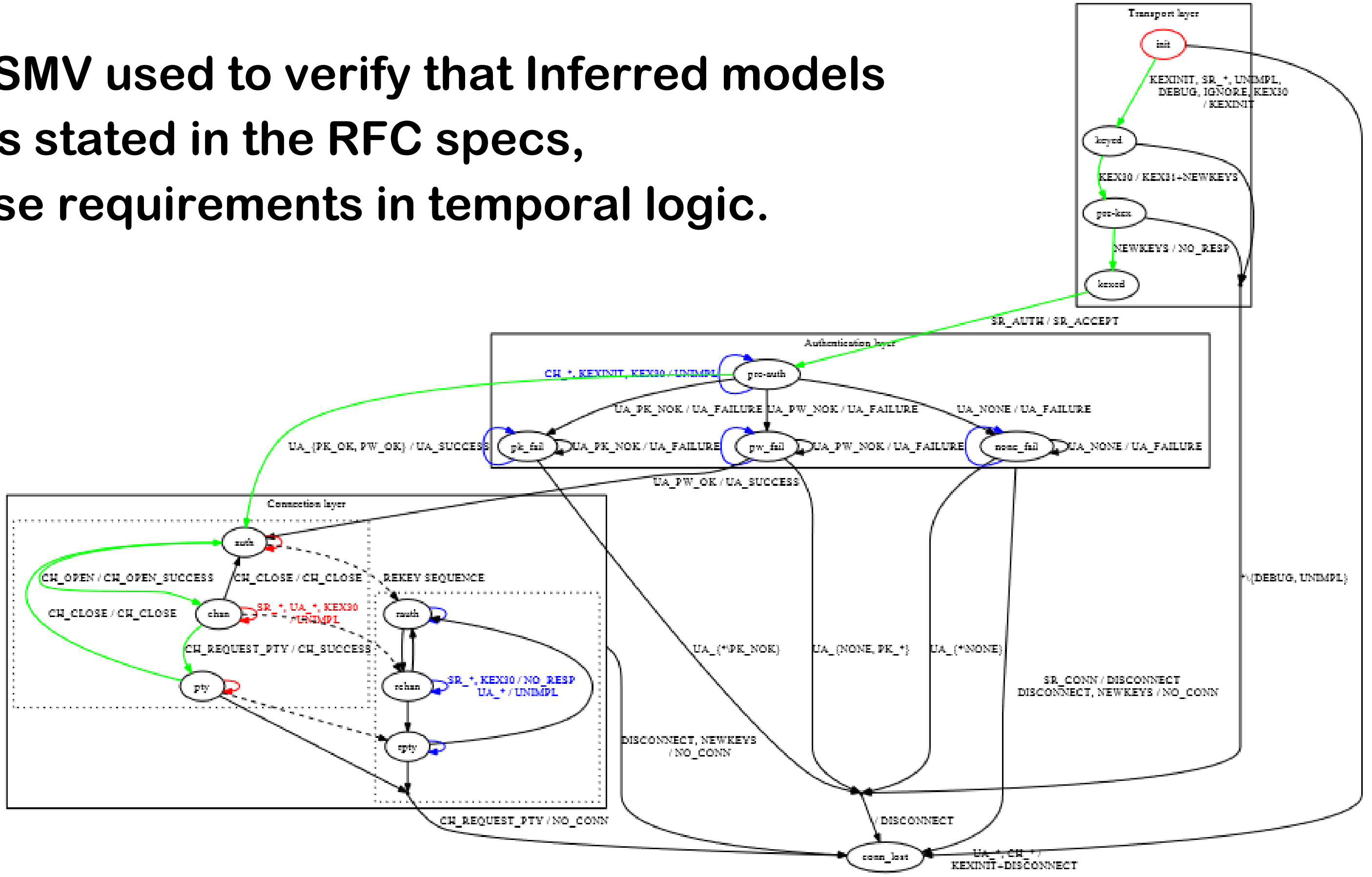


All implementations we analysed are different!
Why doesn't the TLS spec include a state machine?

[Joeri de Ruiter and Erik Poll, *Protocol state fuzzing of TLS implementations*, Usenix Security 2015]

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-Brostean, 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 Ruiter and Aleksy Schubert, *Protocol state machines and session languages*, LangSec 2015]


NO MORE PROSE
SPECIFICATIONS
OF PROTOCOL
STATE MACHINES

There is some progress: TLS 1.3   [RFC 8446]

```
                              START <-----+
               Recv ClientHello |         | Send HelloRetryRequest
                                v         |
                             RECVD_CH ----+
                                | Select parameters
                                v
                             NEGOTIATED
                                | Send ServerHello
                                | K_send = handshake
                                | Send EncryptedExtensions
                                | [Send CertificateRequest]
 Can send                       | [Send Certificate + CertificateVerify]
 app data                       | Send Finished
 after   -->                    | K_send = application
 here                  +--------+--------+
              No 0-RTT |                 | 0-RTT
                       |                 |
   K_recv = handshake  |                 | K_recv = early data
 [Skip decrypt errors] |    +------> WAIT_EOED -+
                       |    |       Recv |      | Recv EndOfEarlyData
                       |    | early data |      | K_recv = handshake
                       |    +------------+      |
                       |                        |
                       +> WAIT_FLIGHT2 <--------+
                                |
                       +--------+--------+
               No auth |                 | Client auth
                       |                 |
                       |                 v
                       |             WAIT_CERT
                       |        Recv |       | Recv Certificate
                       |       empty |       v
                       | Certificate |    WAIT_CV
                       |             |       | Recv
                       |             v       | CertificateVerify
                       +-> WAIT_FINISHED <---+
                                | Recv Finished
                                | K_recv = application
                                v
                            CONNECTED
```

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

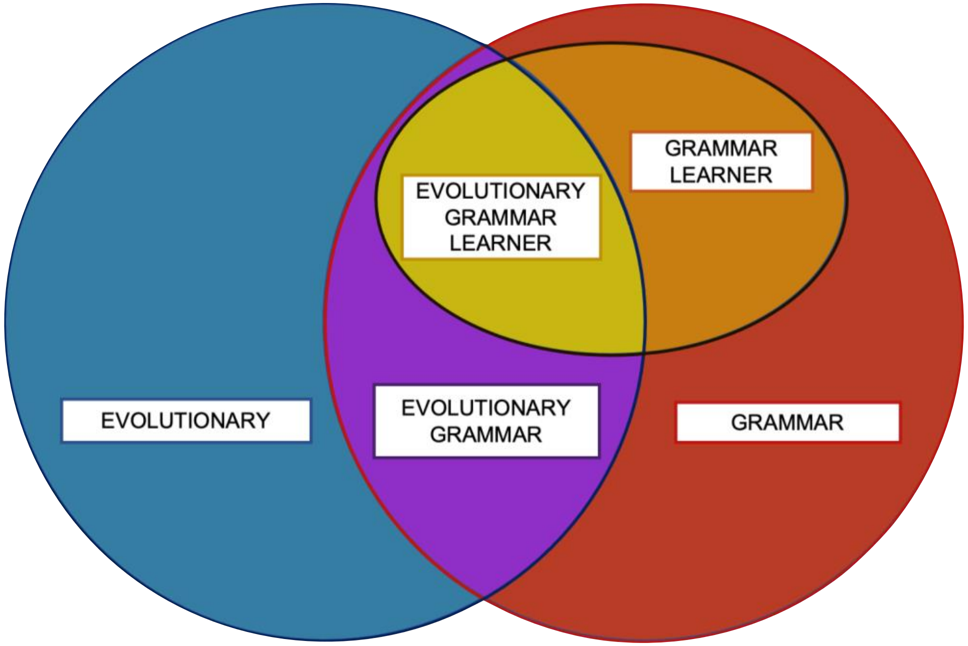
- Not only for cryptographic protocols!
  - In fact, fuzzing cryptographic protocols is hard,
    as it requires a custom test harness that is labour-intensive to make

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]