Formal Methods for Security Functionality and for Secure Functionality

Erik Poll

Digital Security

Radboud University Nijmegen

Topics

- Some of my own research on formal methods & security
  - Security seemed like a killer application for formal methods…
  - Incl. work on state machine learning
- Some of other people's research
  - esp. LangSec (Language-theoretic Security) [http://langsec.org]
- Some directions for the future
  - Not only in research but also in teaching:

    Some lightweight formal methods could prevent a lot of security misery

Security is a *software engineering* problem

Fundamental fact of life:

Things can be hacked because there is **SOFTWARE** inside them

Useful technologies that *help but that do not solve* this problem include

- Crypto
  - offers some prevention against very specific security problems
- Network intrusion detection, SOCs (Security Operation Centre), …
  - to detect when things were hacked & then respond

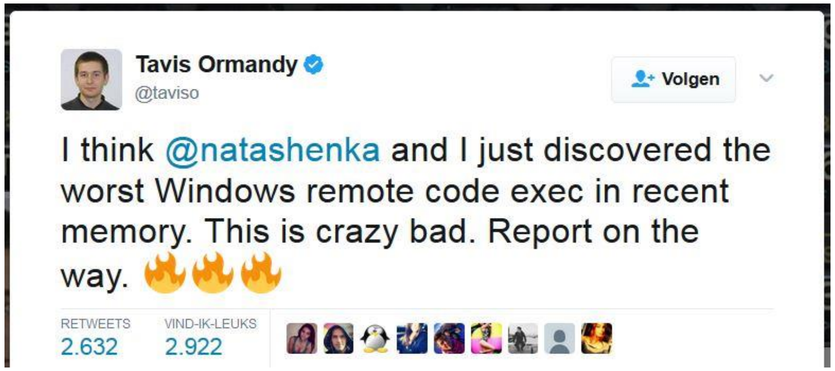
These and other security technologies introduce **YET MORE SOFTWARE**

'Achilles only had an Achilles heel, I have an entire Achilles body'

- *Woody Allen*

Example: Windows Defender bug [CVE 2017-0290]

- Remotely exploitable type confusion in the JavaScript engine inside the MsMpEng malware protection engine
  - Enabled by default on Windows 8, 8.1, 10 and Windows Server 2008 and 2012
  - Runs as `NT AUTHORITY\SYSTEM` without sandboxing
  - Remotely accessible via web browsers, Exchange, and Outlook


Tavis Ormandy
@taviso

I think @natashenka and I just discovered the worst Windows remote code exec in recent memory. This is crazy bad. Report on the way. 🔥🔥🔥

RETWEETS 2.632 VIND-IK-LEUKS 2.922

Vulnerability discovered by Natalie Silvanovich and Tavis Ormandy

https://bugs.chromium.org/p/project-zero/issues/detail?id=1252

One approach to use formal methods for security

1. Take some security-critical system
   - E.g. an OS kernel/hypervisor, security protocol, smartcard program
2. Formalise & verify the security guarantees this system provides
   - These security guarantees can be tricky formalise, but it is (sort of) clear what they are

Great success stories:

- L4.verified: verification of seL4 microkernel
- TLS: analysis & verified implementations of new TLS 1.3

This is 'classic' FM: formally specify & then verify

Security Functionality vs Secure Functionality

- TLS and OS kernels provide security functionality.

  Therefore, they are natural targets for applying FM:

  1. They are obviously security-critical
  2. There are security properties to specify & verify

- *But that about all the other software?*
  This software should also be 'secure', but what does that even mean?
  - What does it mean for a PDF viewer to be secure?
  - Worrying quote: *"Adobe has enhanced JavaScript so that you can easily integrate this level of interactivity into your PDF documents."*

Formal approaches to specify some security

- Temporal logic or security automata
  - E.g. action X only possible after entering PIN code
- Information flow properties
  - Showing that confidential information does not leak
  - Showing that untrusted (tainted) information does not end up in places where it can do damage
- Precondition TRUE in contracts for public interfaces
  - Not just `{P} S {Q}`

    but also `{not P} S {nothing 'bad' happened}`

    or only `{true} S {no WeirdRuntimeException}`

Formal Methods for Secure Functionality

- *What to verify for, say, a PDF viewer?*

- One generic property we would want verify

  This application cannot be hacked

  Hard to specify, let alone formally

- Fortunately, people keep making the same security mistakes!

  So we can approximate this property with

  This application does not contain well-known security flaw X

Standard security problems

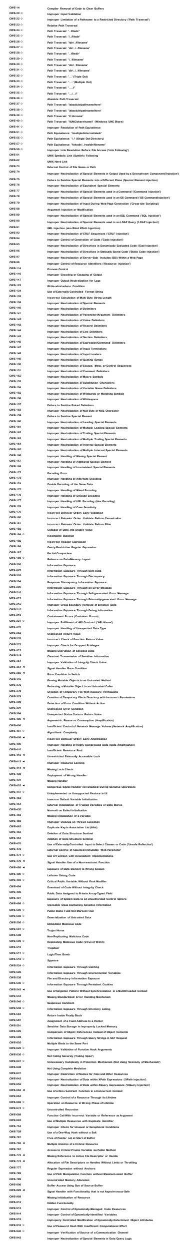
OWASP Top 10 [2017]

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

SANS/CWE TOP 25 [2019]

1. Improper Restriction of Operations within the Bounds of a Memory Buffer
2. Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
3. Improper Input Validation
4. Information exposure
5. Buffer overread
6. SQL Injection
7. Use After Free
8. Integer Overflow
9. CSRF
10. Path Traversal
11. OS Command Injection
12. Out-of-bounds Write
13. Improper Authentication
14. NULL Pointer Dereference
15. Incorrect Permission Assignment
16. Unrestricted Upload of File with Dangerous Type
17. Improper Restriction of XML External Entity
18. Code Injection
19. Use of Hard-coded Credentials
20. Uncontrolled Resource Consumption
21. Missing Release of Resource
22. Untrusted Search Path
23. Deserialization of Untrusted Data
24. Improper Privilege Management
25. Improper Certificate Validation

CWE TOP 668

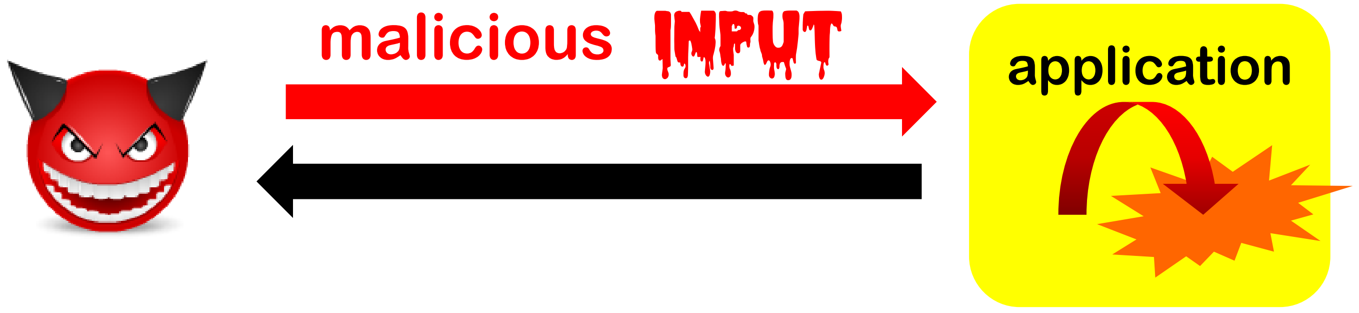


CWE-14	Compiler Removal of Code to Clear Buffers
CWE-20 ○	Improper Input Validation
CWE-22 ○	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
CWE-23 ○	Relative Path Traversal
CWE-24 ○	Path Traversal: '../filedir'
CWE-25 ○	Path Traversal: '/../filedir'
CWE-26 ○	Path Traversal: '/dir/../filename'
CWE-27 ○	Path Traversal: 'dir/../../filename'
CWE-28 ○	Path Traversal: '..\filedir'
CWE-29 ○	Path Traversal: '\..\filename'
CWE-30 ○	Path Traversal: '\dir\..\filename'
CWE-31 ○	Path Traversal: 'dir\..\..\filename'
CWE-32 ○	Path Traversal: '...' (Triple Dot)
CWE-33 ○	Path Traversal: '....' (Multiple Dot)
CWE-34 ○	Path Traversal: '....//'
CWE-35 ○	Path Traversal: '.../...//'
CWE-36 ○	Absolute Path Traversal
CWE-37 ○	Path Traversal: '/absolute/pathname/here'
CWE-38 ○	Path Traversal: '\absolute\pathname\here'
CWE-39 ○	Path Traversal: 'C:dirname'
CWE-40 ○	Path Traversal: '\\UNC\share\name\' (Windows UNC Share)
CWE-41 ○	Improper Resolution of Path Equivalence
CWE-51 ○	Path Equivalence: '/multiple//internal/slash'
CWE-55 ○	Path Equivalence: '/./' (Single Dot Directory)
CWE-57 ○	Path Equivalence: 'fakedir/../realdir/filename'
CWE-59 ○	Improper Link Resolution Before File Access ('Link Following')
CWE-61	UNIX Symbolic Link (Symlink) Following
CWE-62	UNIX Hard Link
CWE-73	External Control of File Name or Path
CWE-74	Improper Neutralization of Special Elements in Output Used by a Downstream Component('Injection')
CWE-75	Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)
CWE-76	Improper Neutralization of Equivalent Special Elements
CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')
CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS CommandInjection')
CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
CWE-88	Argument Injection or Modification
CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
CWE-90	Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')
CWE-91	XML Injection (aka Blind XPath Injection)
CWE-93	Improper Neutralization of CRLF Sequences ('CRLF Injection')
CWE-94	Improper Control of Generation of Code ('Code Injection')
CWE-95	Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')
CWE-96	Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')
CWE-97	Improper Neutralization of Server-Side Includes (SSI) Within a Web Page
CWE-99	Improper Control of Resource Identifiers ('Resource Injection')
CWE-114	Process Control
CWE-116	Improper Encoding or Escaping of Output
CWE-117	Improper Output Neutralization for Logs
CWE-123	Write-what-where Condition
CWE-134	Use of Externally-Controlled Format String
CWE-135	Incorrect Calculation of Multi-Byte String Length
CWE-138	Improper Neutralization of Special Elements
CWE-140	Improper Neutralization of Delimiters
CWE-141	Improper Neutralization of Parameter/Argument Delimiters
CWE-142	Improper Neutralization of Value Delimiters
CWE-143	Improper Neutralization of Record Delimiters
CWE-144	Improper Neutralization of Line Delimiters
CWE-145	Improper Neutralization of Section Delimiters
CWE-146	Improper Neutralization of Expression/Command Delimiters
CWE-147	Improper Neutralization of Input Terminators
CWE-148	Improper Neutralization of Input Leaders
CWE-149	Improper Neutralization of Quoting Syntax
CWE-150	Improper Neutralization of Escape, Meta, or Control Sequences
CWE-151	Improper Neutralization of Comment Delimiters
CWE-152	Improper Neutralization of Macro Symbols
CWE-153	Improper Neutralization of Substitution Characters
CWE-154	Improper Neutralization of Variable Name Delimiters
CWE-155	Improper Neutralization of Wildcards or Matching Symbols
CWE-156	Improper Neutralization of Whitespace
CWE-157	Failure to Sanitize Paired Delimiters
CWE-158	Improper Neutralization of Null Byte or NUL Character
CWE-159	Failure to Sanitize Special Element
CWE-160	Improper Neutralization of Leading Special Elements
CWE-161	Improper Neutralization of Multiple Leading Special Elements
CWE-162	Improper Neutralization of Trailing Special Elements
CWE-163	Improper Neutralization of Multiple Trailing Special Elements
CWE-164	Improper Neutralization of Internal Special Elements
CWE-165	Improper Neutralization of Multiple Internal Special Elements
CWE-166	Improper Handling of Missing Special Element
CWE-167	Improper Handling of Additional Special Element
CWE-168	Improper Handling of Inconsistent Special Elements
CWE-172	Encoding Error
CWE-173	Improper Handling of Alternate Encoding
CWE-174	Double Decoding of the Same Data
CWE-175	Improper Handling of Mixed Encoding
CWE-176	Improper Handling of Unicode Encoding
CWE-177	Improper Handling of URL Encoding (Hex Encoding)
CWE-178	Improper Handling of Case Sensitivity
CWE-179	Incorrect Behavior Order: Early Validation
CWE-180	Incorrect Behavior Order: Validate Before Canonicalize
CWE-181	Incorrect Behavior Order: Validate Before Filter
CWE-182	Collapse of Data into Unsafe Value
CWE-184 ○	Incomplete Blacklist
CWE-185	Incorrect Regular Expression
CWE-186	Overly Restrictive Regular Expression
CWE-187	Partial Comparison
CWE-188 ○	Reliance on Data/Memory Layout
CWE-200	Information Exposure
CWE-201	Information Exposure Through Sent Data
CWE-203	Information Exposure Through Discrepancy
CWE-204	Response Discrepancy Information Exposure
CWE-209	Information Exposure Through an Error Message
CWE-210	Information Exposure Through Self-generated Error Message
CWE-211	Information Exposure Through Externally-generated Error Message
CWE-212	Improper Cross-boundary Removal of Sensitive Data
CWE-215	Information Exposure Through Debug Information
CWE-216	Containment Errors (Container Errors)
CWE-227 ○	Improper Fulfillment of API Contract ('API Abuse')
CWE-241	Improper Handling of Unexpected Data Type
CWE-252	Unchecked Return Value
CWE-253	Incorrect Check of Function Return Value
CWE-273	Improper Check for Dropped Privileges
CWE-311	Missing Encryption of Sensitive Data
CWE-319	Cleartext Transmission of Sensitive Information
CWE-354	Improper Validation of Integrity Check Value
CWE-364 ◄	Signal Handler Race Condition
CWE-365 ◄	Race Condition in Switch
CWE-374	Passing Mutable Objects to an Untrusted Method
CWE-375	Returning a Mutable Object to an Untrusted Caller
CWE-378	Creation of Temporary File With Insecure Permissions
CWE-379	Creation of Temporary File in Directory with Incorrect Permissions
CWE-390	Detection of Error Condition Without Action
CWE-391	Unchecked Error Condition
CWE-394	Unexpected Status Code or Return Value
CWE-405 ◄	Asymmetric Resource Consumption (Amplification)
CWE-406	Insufficient Control of Network Message Volume (Network Amplification)
CWE-407 ○	Algorithmic Complexity
CWE-408 ◄	Incorrect Behavior Order: Early Amplification
CWE-409	Improper Handling of Highly Compressed Data (Data Amplification)
CWE-410	Insufficient Resource Pool
CWE-412 ◄	Unrestricted Externally Accessible Lock
CWE-413 ◄	Improper Resource Locking
CWE-414 ◄	Missing Lock Check
CWE-430	Deployment of Wrong Handler
CWE-431	Missing Handler
CWE-432 ◄	Dangerous Signal Handler not Disabled During Sensitive Operations
CWE-447 ○	Unimplemented or Unsupported Feature in UI
CWE-453	Insecure Default Variable Initialization
CWE-454	External Initialization of Trusted Variables or Data Stores
CWE-455	Non-exit on Failed Initialization
CWE-456	Missing Initialization of a Variable
CWE-460	Improper Cleanup on Thrown Exception
CWE-462	Duplicate Key in Associative List (Alist)
CWE-463	Deletion of Data Structure Sentinel
CWE-464	Addition of Data Structure Sentinel
CWE-470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')
CWE-472	External Control of Assumed-Immutable Web Parameter
CWE-474 ○	Use of Function with Inconsistent Implementations
CWE-479 ◄	Signal Handler Use of a Non-reentrant Function
CWE-488 ◄	Exposure of Data Element to Wrong Session
CWE-489 ○	Leftover Debug Code
CWE-493 ○	Critical Public Variable Without Final Modifier
CWE-494	Download of Code Without Integrity Check
CWE-496	Public Data Assigned to Private Array-Typed Field
CWE-497	Exposure of System Data to an Unauthorized Control Sphere
CWE-498 ○	Cloneable Class Containing Sensitive Information
CWE-500 ○	Public Static Field Not Marked Final
CWE-502 ○	Deserialization of Untrusted Data
CWE-506 ○	Embedded Malicious Code
CWE-507 ○	Trojan Horse
CWE-508	Non-Replicating Malicious Code
CWE-509 ○	Replicating Malicious Code (Virus or Worm)
CWE-510	Trapdoor
CWE-511 ○	Logic/Time Bomb
CWE-512 ○	Spyware
CWE-524 ○	Information Exposure Through Caching
CWE-526	Information Exposure Through Environmental Variables
CWE-538	File and Directory Information Exposure
CWE-539 ○	Information Exposure Through Persistent Cookies
CWE-543 ◄	Use of Singleton Pattern Without Synchronization in a Multithreaded Context
CWE-544	Missing Standardized Error Handling Mechanism
CWE-546 ○	Suspicious Comment
CWE-548 ○	Information Exposure Through Directory Listing
CWE-584	Return Inside Finally Block
CWE-587	Assignment of a Fixed Address to a Pointer
CWE-591	Sensitive Data Storage in Improperly Locked Memory
CWE-595	Comparison of Object References Instead of Object Contents
CWE-598	Information Exposure Through Query Strings in GET Request
CWE-605	Multiple Binds to the Same Port
CWE-622 ○	Improper Validation of Function Hook Arguments
CWE-636 ○	Not Failing Securely ('Failing Open')
CWE-637 ○	Unnecessary Complexity in Protection Mechanism (Not Using 'Economy of Mechanism')
CWE-638	Not Using Complete Mediation
CWE-641	Improper Restriction of Names for Files and Other Resources
CWE-643	Improper Neutralization of Data within XPath Expressions ('XPath Injection')
CWE-652	Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')
CWE-663 ◄	Use of a Non-reentrant Function in a Concurrent Context
CWE-664	Improper Control of a Resource Through its Lifetime
CWE-666 ○	Operation on Resource in Wrong Phase of Lifetime
CWE-674 ○	Uncontrolled Recursion
CWE-688	Function Call With Incorrect Variable or Reference as Argument
CWE-694	Use of Multiple Resources with Duplicate Identifier
CWE-754	Improper Check for Unusual or Exceptional Conditions
CWE-759	Use of a One-Way Hash without a Salt
CWE-761	Free of Pointer not at Start of Buffer
CWE-765 ◄	Multiple Unlocks of a Critical Resource
CWE-767	Access to Critical Private Variable via Public Method
CWE-773 ◄	Missing Reference to Active File Descriptor or Handle
CWE-774 ◄	Allocation of File Descriptors or Handles Without Limits or Throttling
CWE-777	Regular Expression without Anchors
CWE-785	Use of Path Manipulation Function without Maximum-sized Buffer
CWE-789	Uncontrolled Memory Allocation
CWE-806	Buffer Access Using Size of Source Buffer
CWE-828 ◄	Signal Handler with Functionality that is not Asynchronous-Safe
CWE-909	Missing Initialization of Resource
CWE-912	Hidden Functionality
CWE-913	Improper Control of Dynamically-Managed Code Resources
CWE-914	Improper Control of Dynamically-Identified Variables
CWE-915	Improperly Controlled Modification of Dynamically-Determined Object Attributes
CWE-916 ○	Use of Password Hash With Insufficient Computational Effort
CWE-940 ○	Improper Verification of Source of a Communication Channel
CWE-943	Improper Neutralization of Special Elements in Data Query Logic

Top 1 security vulnerability: INPUT handling

Mishandling malicious input is the common theme in *most* attacks

eg buffer overflow, format string attack, command injection, path traversal, SQL injection, XSS, CSRF, Word macros, XML/XPath injection, LDAP injection, CRLF injection, deserialization attacks, zip bombs, ...



- Garbage In, Garbage Out

  leads to

  Malicious Garbage In, *Security Incident* Out

LangSec (Language-theoretic Security)

- Interesting look at root causes of INPUT problems
- Useful suggestions for dos and don'ts

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

- The language in Language-theoretic Security refers to input languages, not modelling or programming languages.

Fallacy of classic input validation?

Classical remedy to input problems: input validation / input sanitisation

remove or encode harmful characters (eg `;` `'` `"`)
before processing inputs

But...

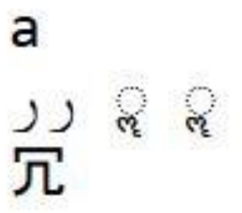
- Which characters are harmful depends on the language/format, and a typical application handles *many* languages.

  Eg `'` problematic for SQL database, < > for web app, `&` for LDAP server

- Instead of validating input before feeding it to crappy software that processes it, maybe that software should be more robust?
  - esp. the parsing it performs as part of any processing

Example: SMS of Death

Text message that used to crash iPhones:



- Should telecom operators or phones do input validation to remove these dangerous Unicode combinations from SMS text messages?
- Or should software be robust in dealing with arbitrary combinations of Unicode?

So, is input validation always the right way to prevent input problems?

https://www.reddit.com/r/iphone/comments/37eaxs/um_can_someone_explain_this_phenomenon/

LangSec: root causes

- Input languages (aka protocols, file formats, encodings) play the central role causing security flaws
- *Any* language *anywhere* in the protocol stack, incl.

  TCP/IP v4 or v6,

  Ethernet, WiFi, GSM, 3G, 4G, 5G, Bluetooth, USB,

  TLS, SSH, OpenVPN,

  HTTP(S), X.509, HTML5 (incl. JavaScript), XML, JSON,

  URLs, email addresses, S/MIME,

  JPG, MP3, MPEG, Flash,

  docx (incl macros), PDF (incl. JavaScript), xls (incl. macros) , …
- This provides a **huge** attack surface for the attacker

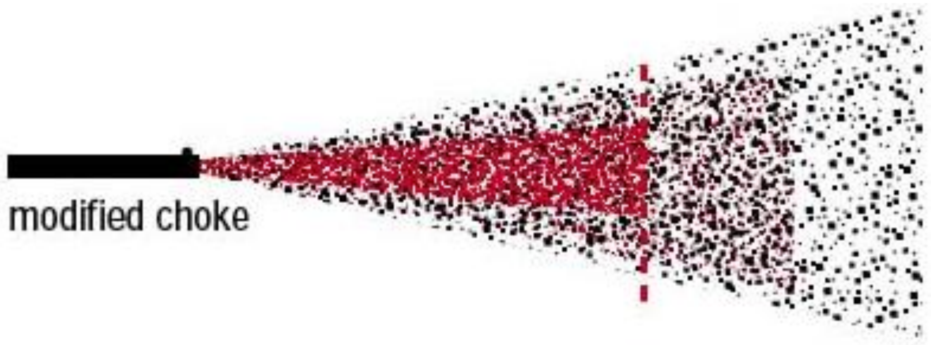
LangSec: root causes of security problems

- *Ad-hoc, imprecise, or complex notion of input validity*

  Eg, have you looked at how complex the PDF file format is? Or HTML5? Or X.509 certificates?

- *Mixing input recognition & processing*

  esp. in shotgun parsers, handwritten code that incrementally parses & interprets input, in a piece-meal fashion

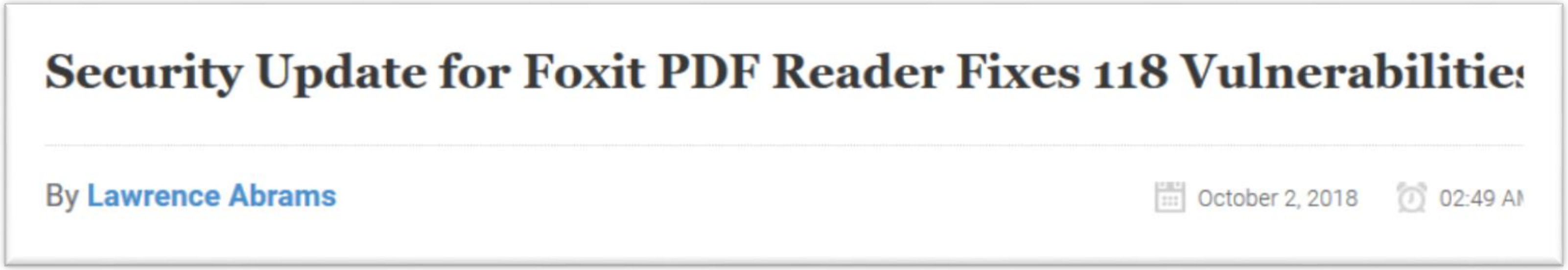


Buggy parsing & processing then results in weird behaviour
- a weird machine - for attackers to have fun with

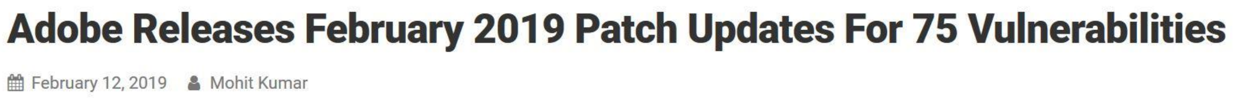
LangSec principles to prevent input problems

1. ***Precisely defined*** input languages
   - eg with regular expression or EBNF grammar
2. ***Generated*** parser code
3. ***Complete*** parsing ***before*** processing
   - Also don't substitute strings & then parse,
     but first parse & then substitute in parse tree
     - cf. parameterised queries instead of dynamic SQL
4. *Keep the input language* ***simple & clear***
   - So that there is minimal chance of bugs or ambiguities
   - So that you give minimal processing power given to attackers

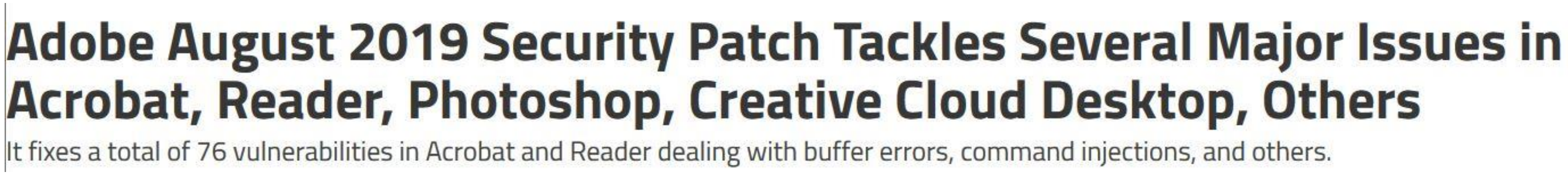
Example INPUT problem: PDF


Security Update for Foxit PDF Reader Fixes 118 Vulnerabilitie

By Lawrence Abrams October 2, 2018 02:49 AI


Adobe Releases February 2019 Patch Updates For 75 Vulnerabilities

February 12, 2019 Mohit Kumar


Adobe August 2019 Security Patch Tackles Several Major Issues in Acrobat, Reader, Photoshop, Creative Cloud Desktop, Others

It fixes a total of 76 vulnerabilities in Acrobat and Reader dealing with buffer errors, command injections, and others.

Root cause: PDF spec is horrendously complex

All PDF viewers suffer from such problems, see

https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=PDF

Example INPUT problem: X.509 certificates

X.509 spec is horrendously complex. Example attacks:

- Multiple comma-separated names in a certificate Common Name

  ```
  paypal.com,mafia.org
  ```

  Different browsers and CAs interpret this in different ways

- ANS.1 attacks in X.509 certificates

  Null terminator in ANS.1 BER-encoded string in a certificate Common Name

  ```
  paypal.com\00mafia.org
  ```

- PKCS#10-tunneled SQL injection

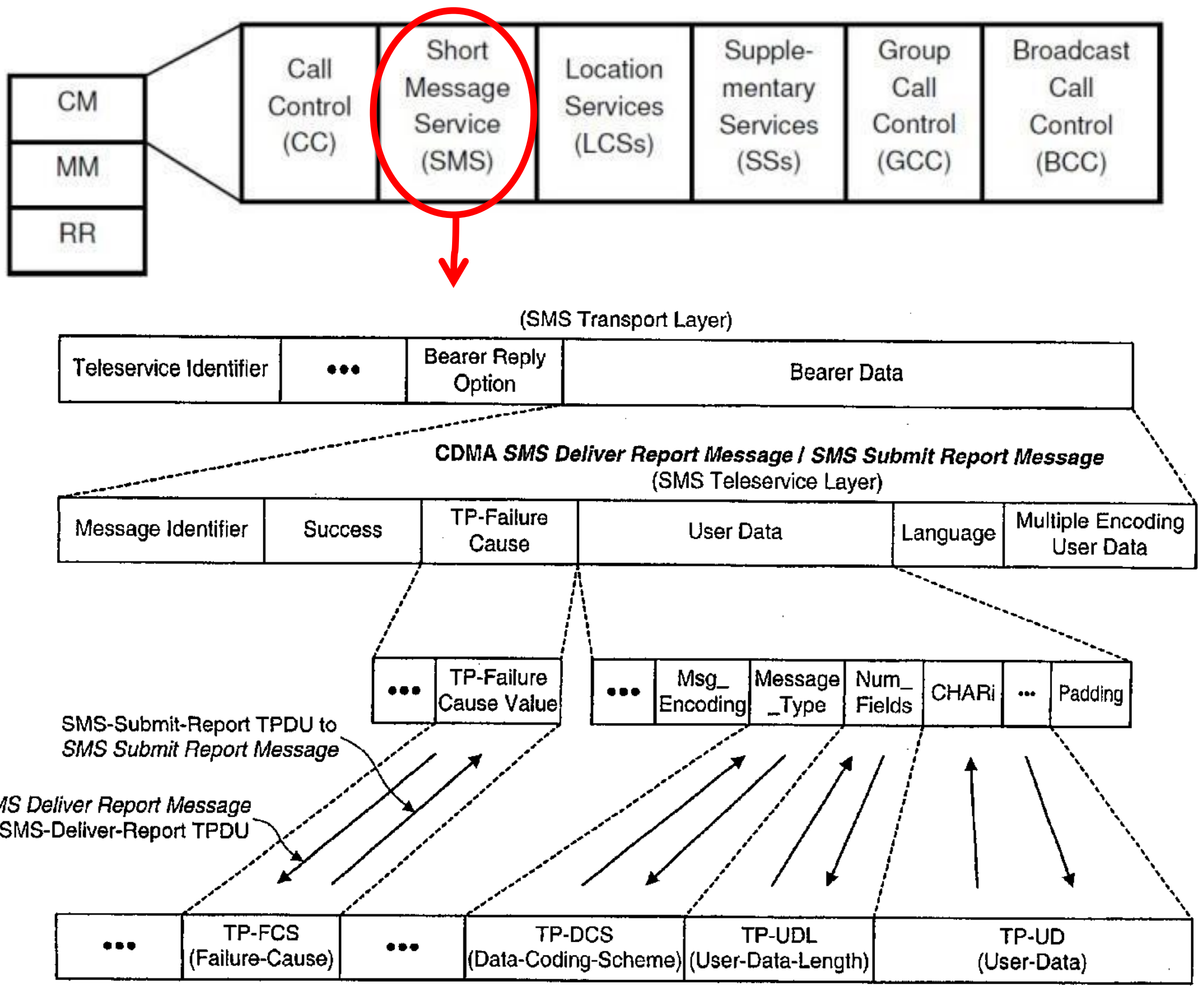
  SQL command inside a BMPString, UTF8String or UniversalString used as PKCS#10 Subject Name ?

[Dan Kaminsky, Meredith Patterson, and Len Sassaman, *PKI Layer Cake: New Collision Attacks against the Global X.509 Infrastructure*, Financial Crypto 2010]

Example INPUT problem: *GSM*

Eg GSM specs

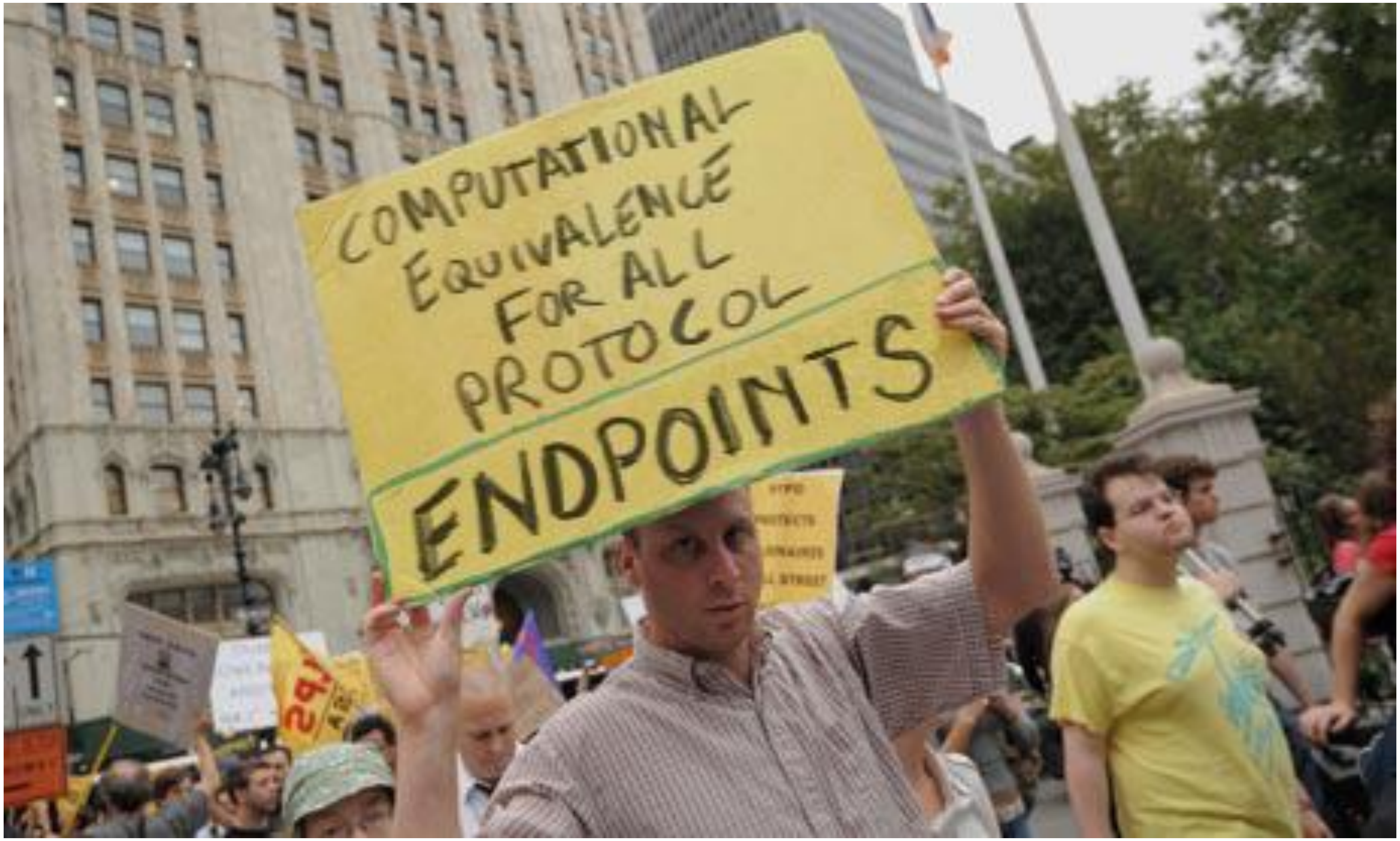
for SMS text messages



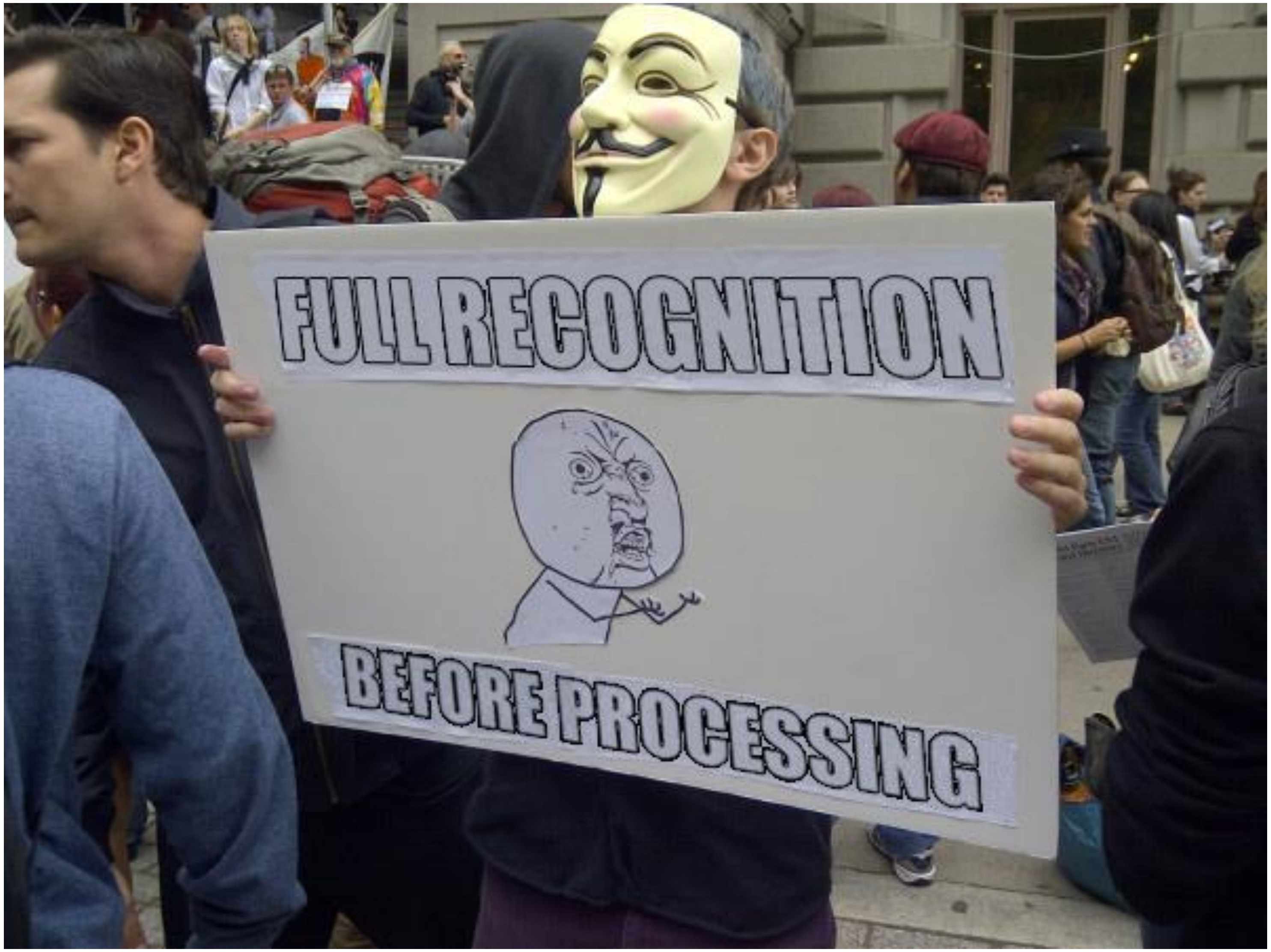
Unsurprisingly, malformed GSM traffic will trigger lots of problems

[Fabian van den Broek, Brinio Hond and Arturo Cedillo Torres, *Security Testing of GSM Implementations*, ESSOS 2014]

LangSec in slogans



[Image by Kythera of Anevern, see http://langsec.org/occupy]



[Image by Kythera of Anevern, see http://langsec.org/occupy]

[Image by Kythera of Anevern, see http://langsec.org/occupy]

LangSec continued:
Protocol state machines

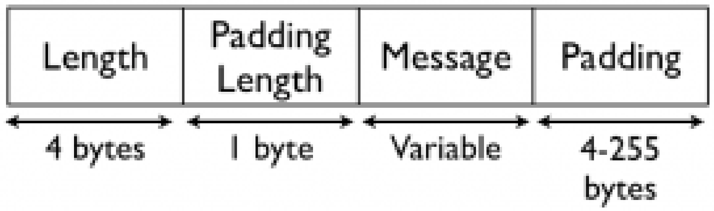
or

Formal Methods for Free!

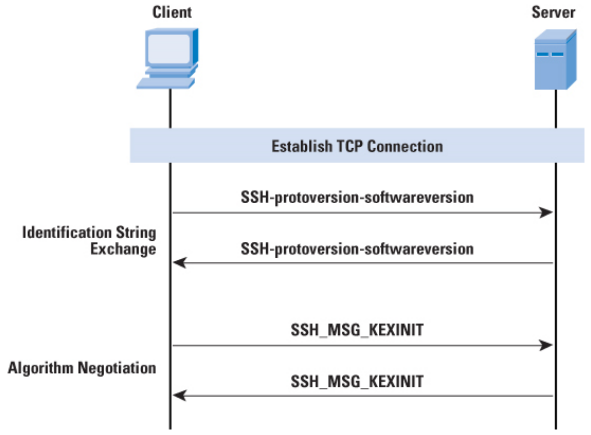
[LangSec 2015]

Sequences of inputs

- Many protocols not only involves a language of input messages



but only a notion of session, ie. *sequence* of messages

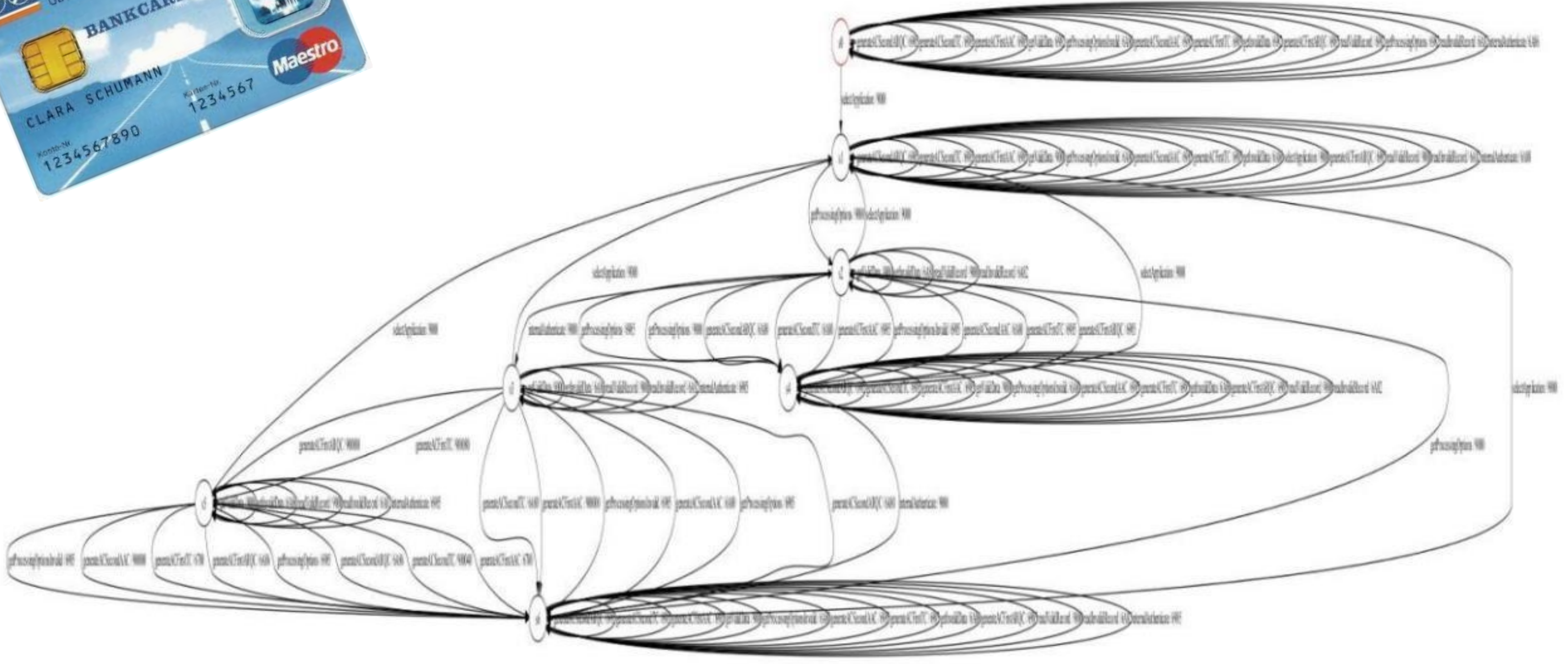

Client
Server
Establish TCP Connection
SSH-protoversion-softwareversion
Identification String Exchange
SSH-protoversion-softwareversion
SSH_MSG_KEXINIT
Algorithm Negotiation
SSH_MSG_KEXINIT

- Most specs only describe the happy flow…
- For security protocols, getting unhappy flows correct is crucial
- Fortunately, we can extract these state machines from code - or hardware - using active learning

Case study: EMV

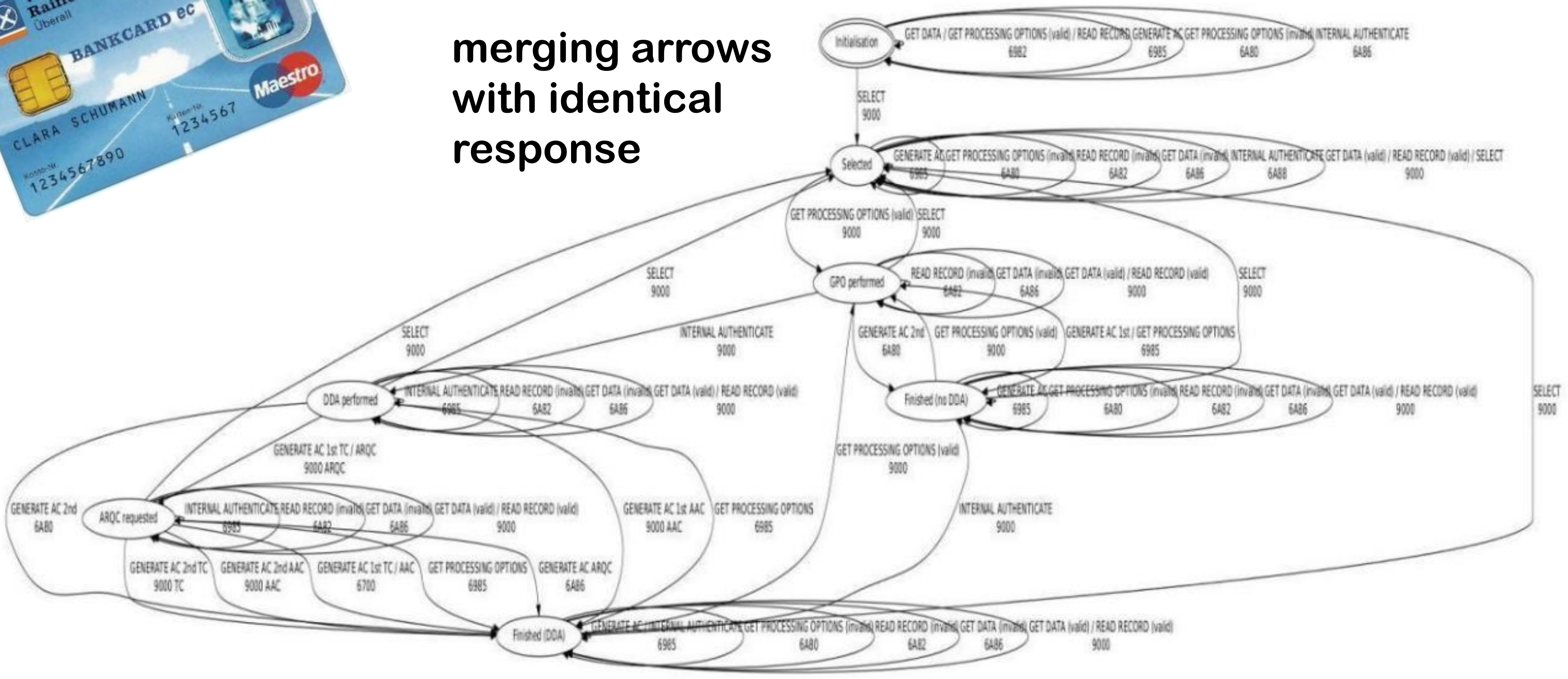
- Most banking smartcards implement a variant of EMV
    - EMV = Europay-Mastercard-Visa
- Specification in 4 books totalling > 700 pages
- Contactless payments: another 7 books with > 2000 pages

State machine learning of Maestro card (using LearnLib)

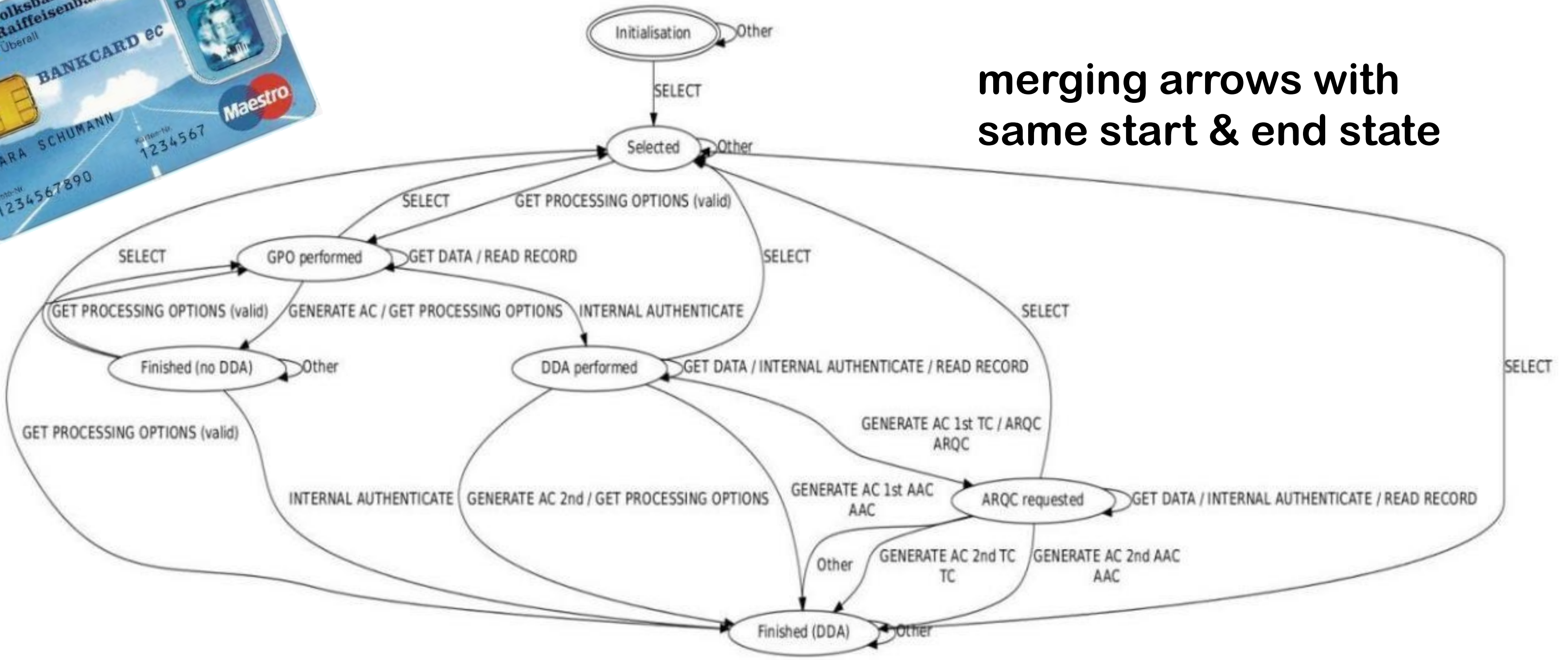


State machine learning of Maestro card (using LearnLib)

merging arrows with identical response



State machine learning of Maestro card (using LearnLib)

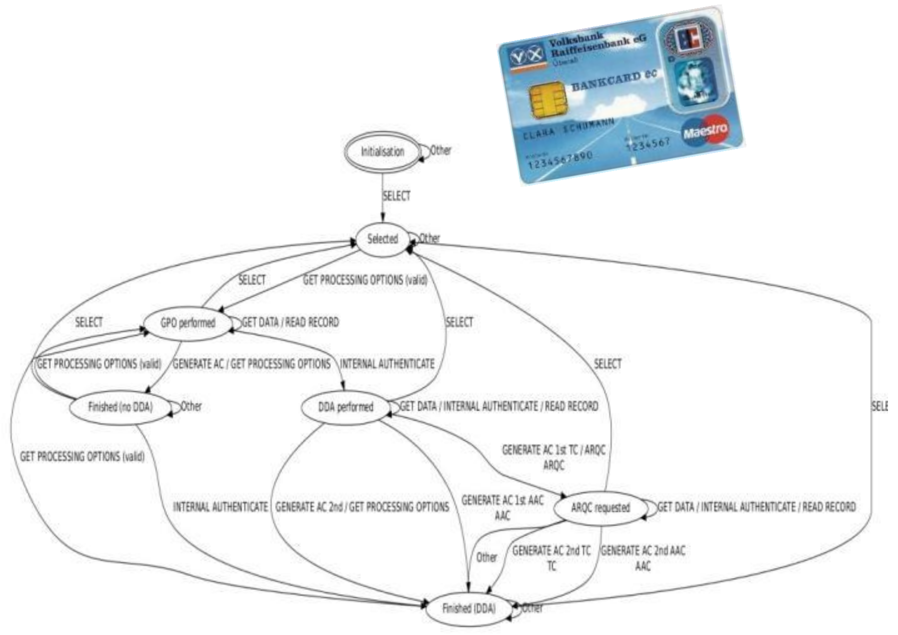


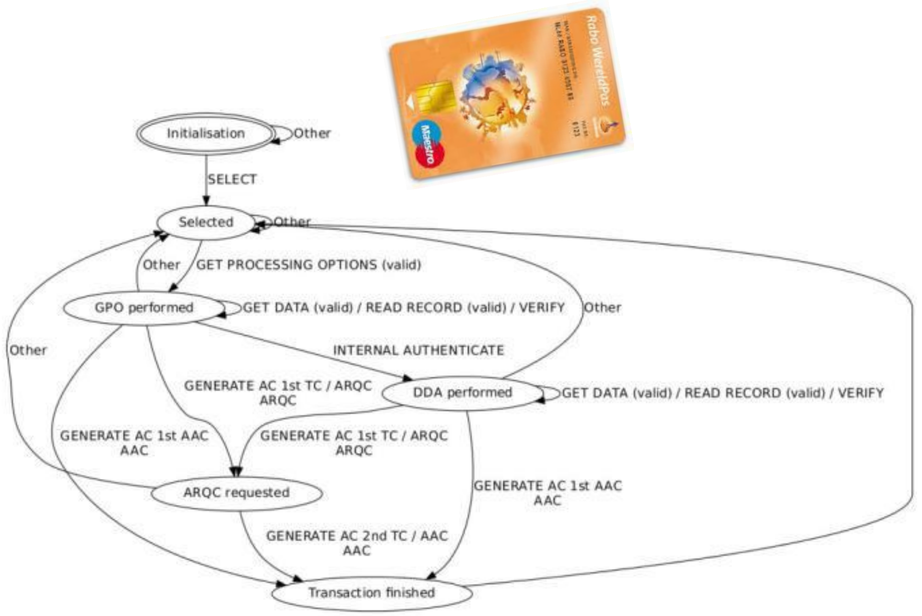
merging arrows with same start & end state

We did not find 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





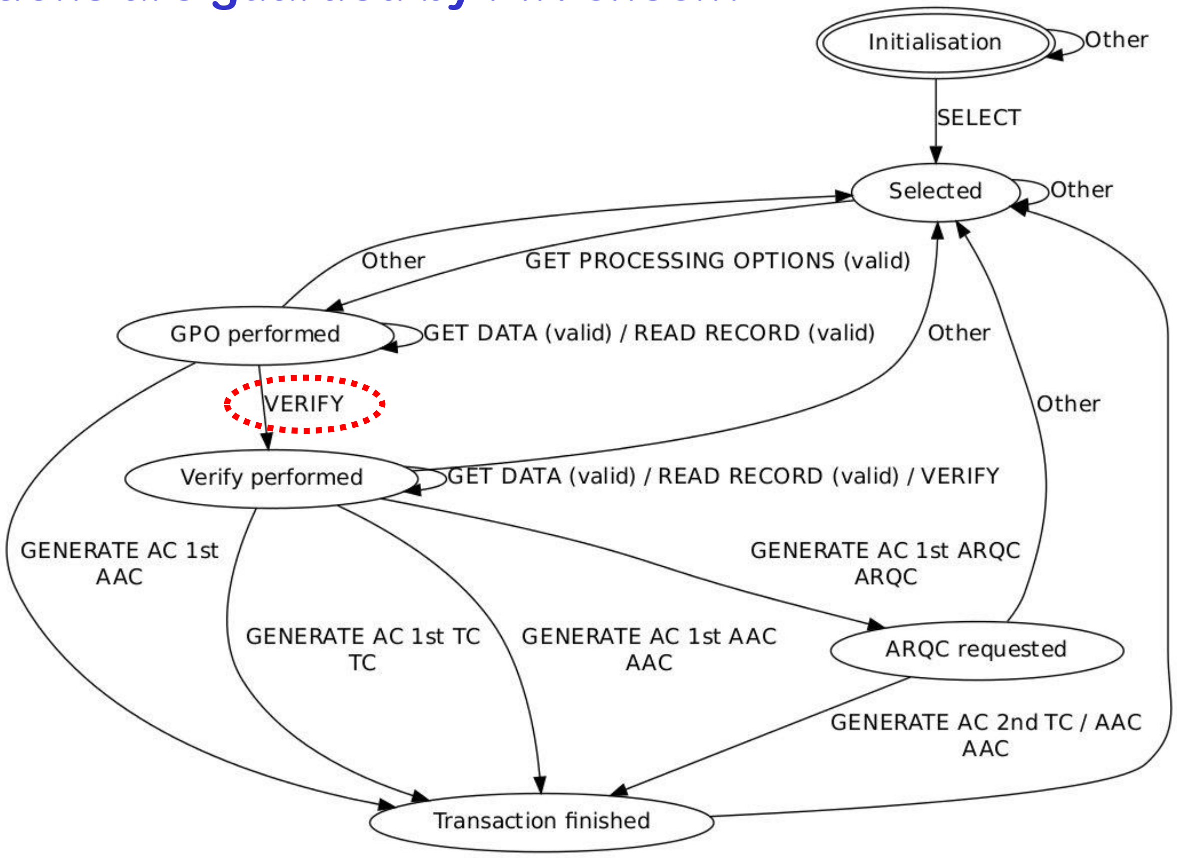
Volksbank Maestro implementation

Rabobank Maestro implementation

Are both implementations correct & secure? Or compatible?

Using state machine for security analysis

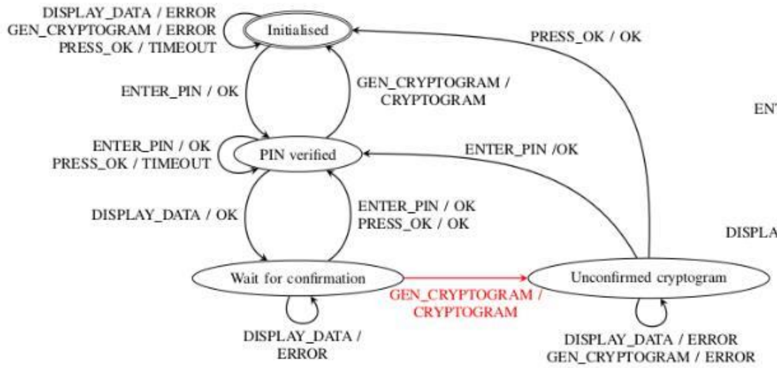
Which actions are guarded by PIN check?

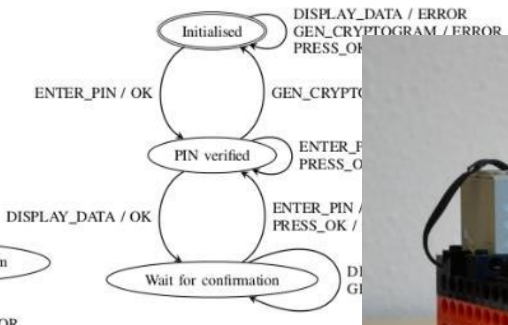


State machine learning of internet banking token

Security flaw in USB-connected internet banking token

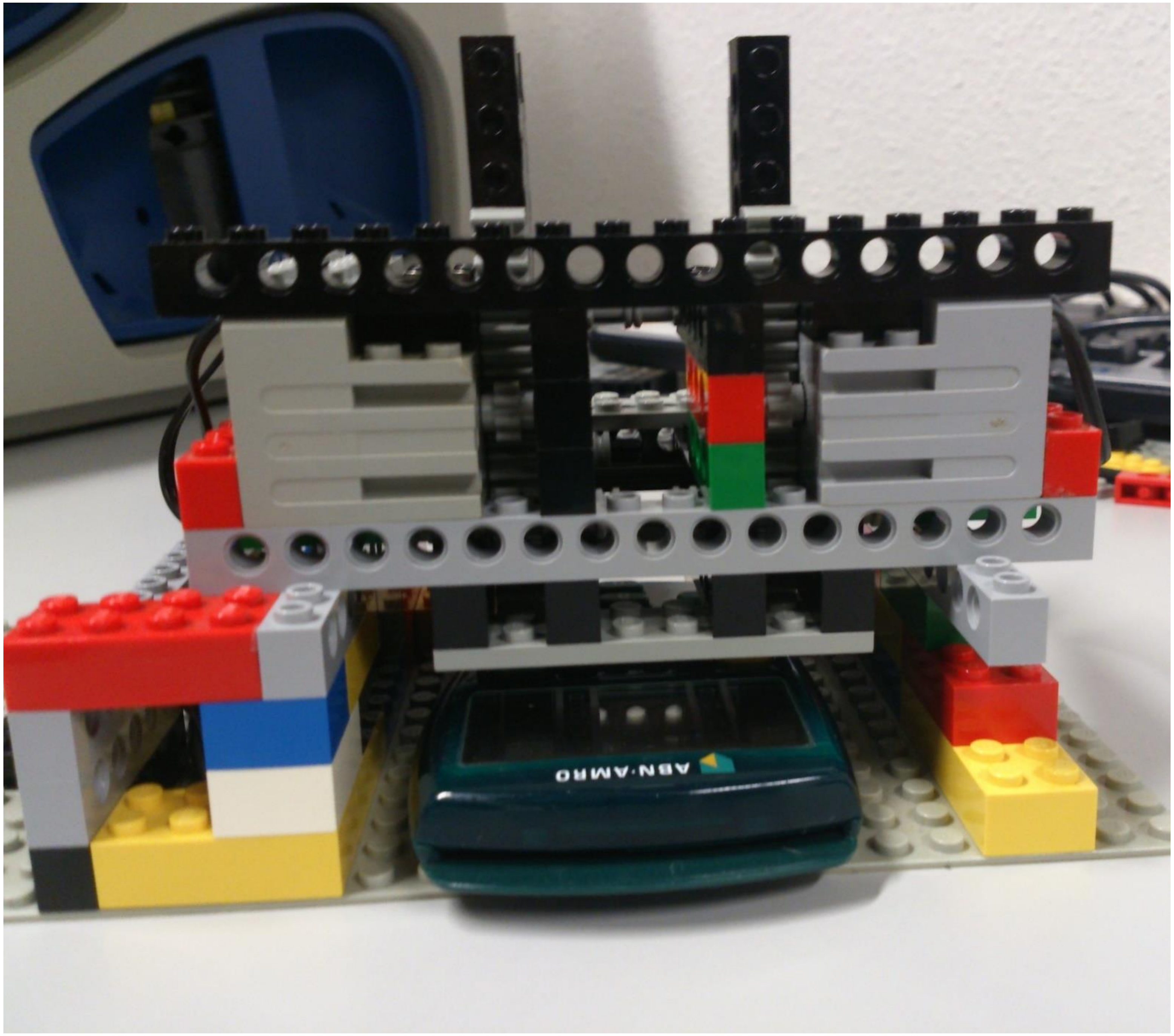
- User has to press OK to confirm transactions
- But … a strange sequence of USB instructions can by-pass this check

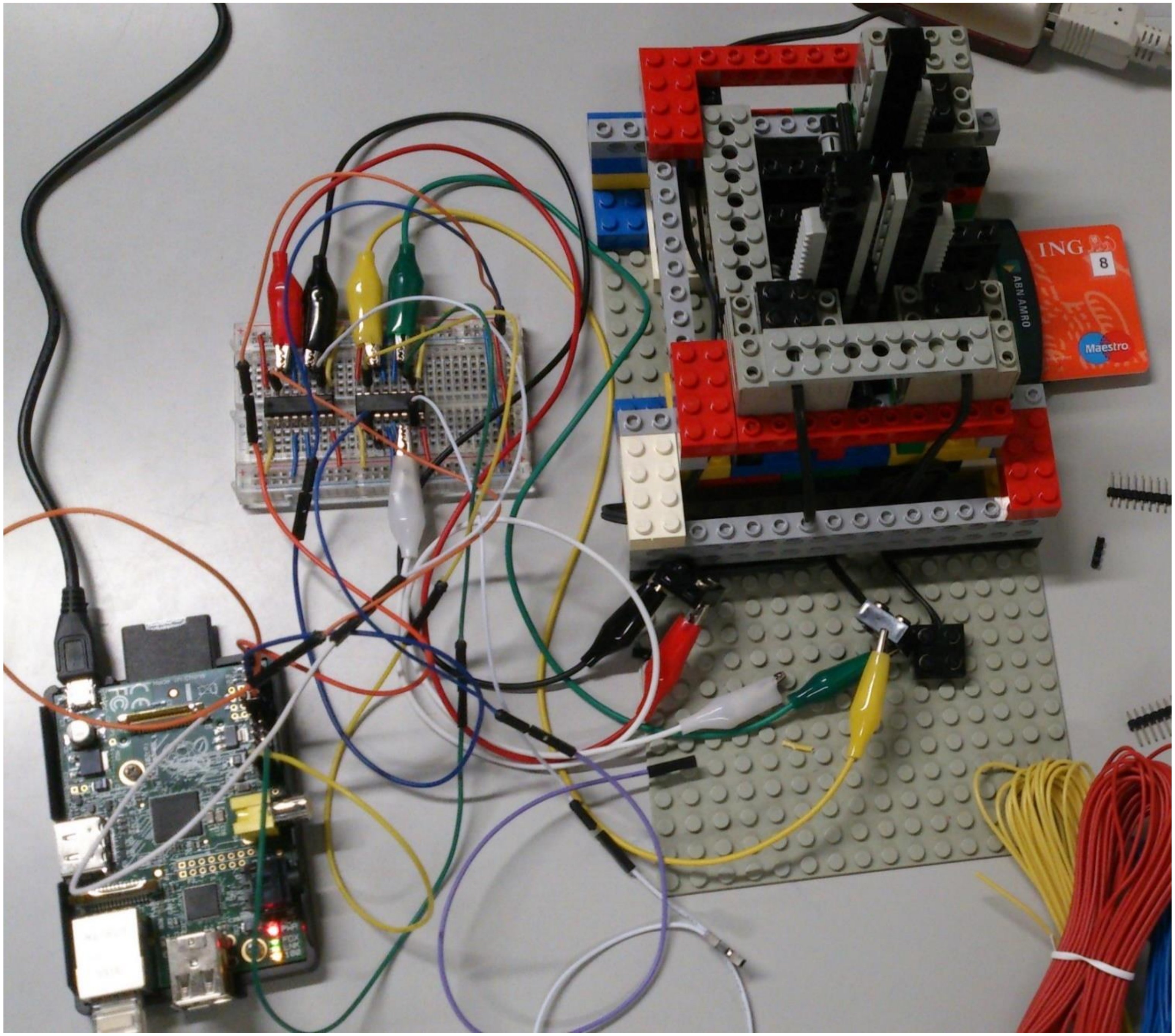




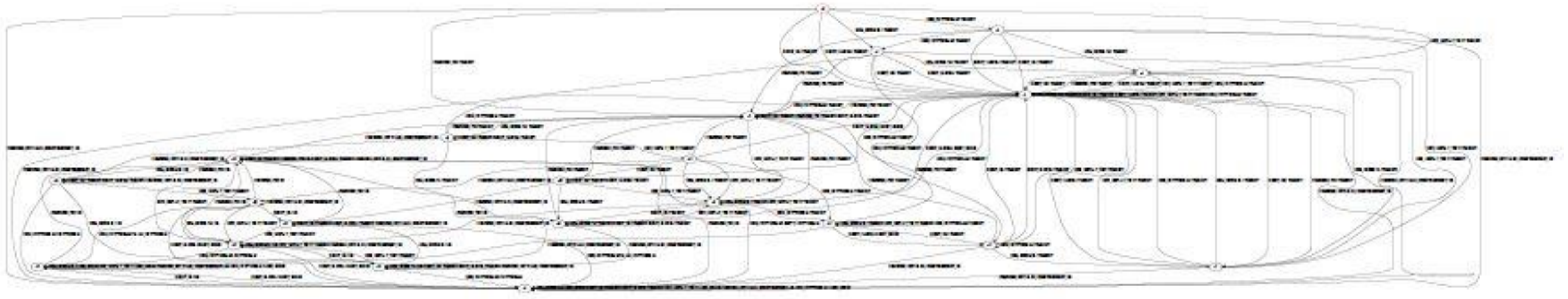
State machine of flawed device & patched device

[Georg Chalupar et al., *Automated reverse engineering using Lego,* WOOT 2014]
Movie at http://tinyurl/legolearn


ABN·AMRO


ING
8
ABN AMRO
Maestro

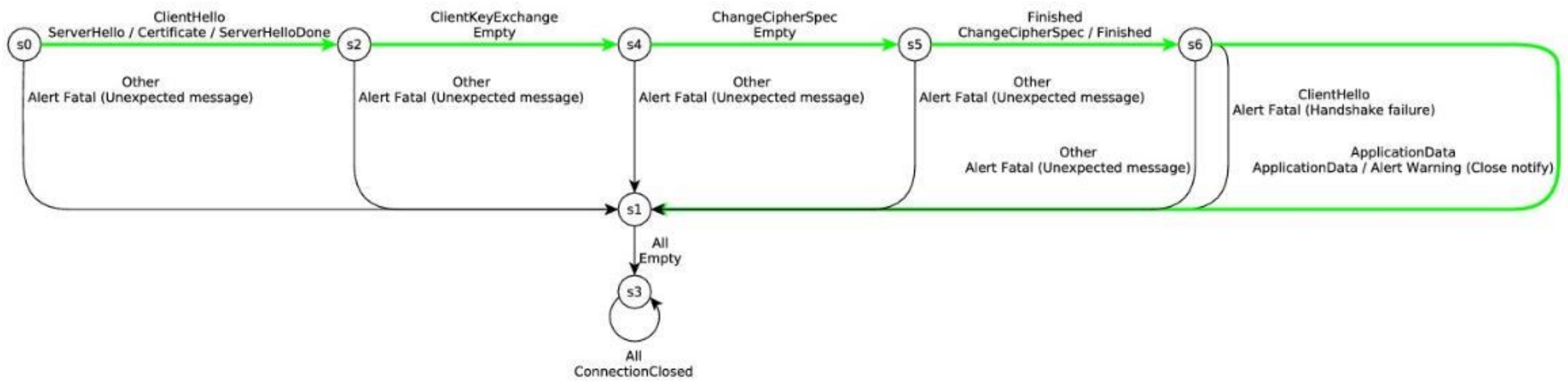
State machine learning of internet banking token



Complete state machine

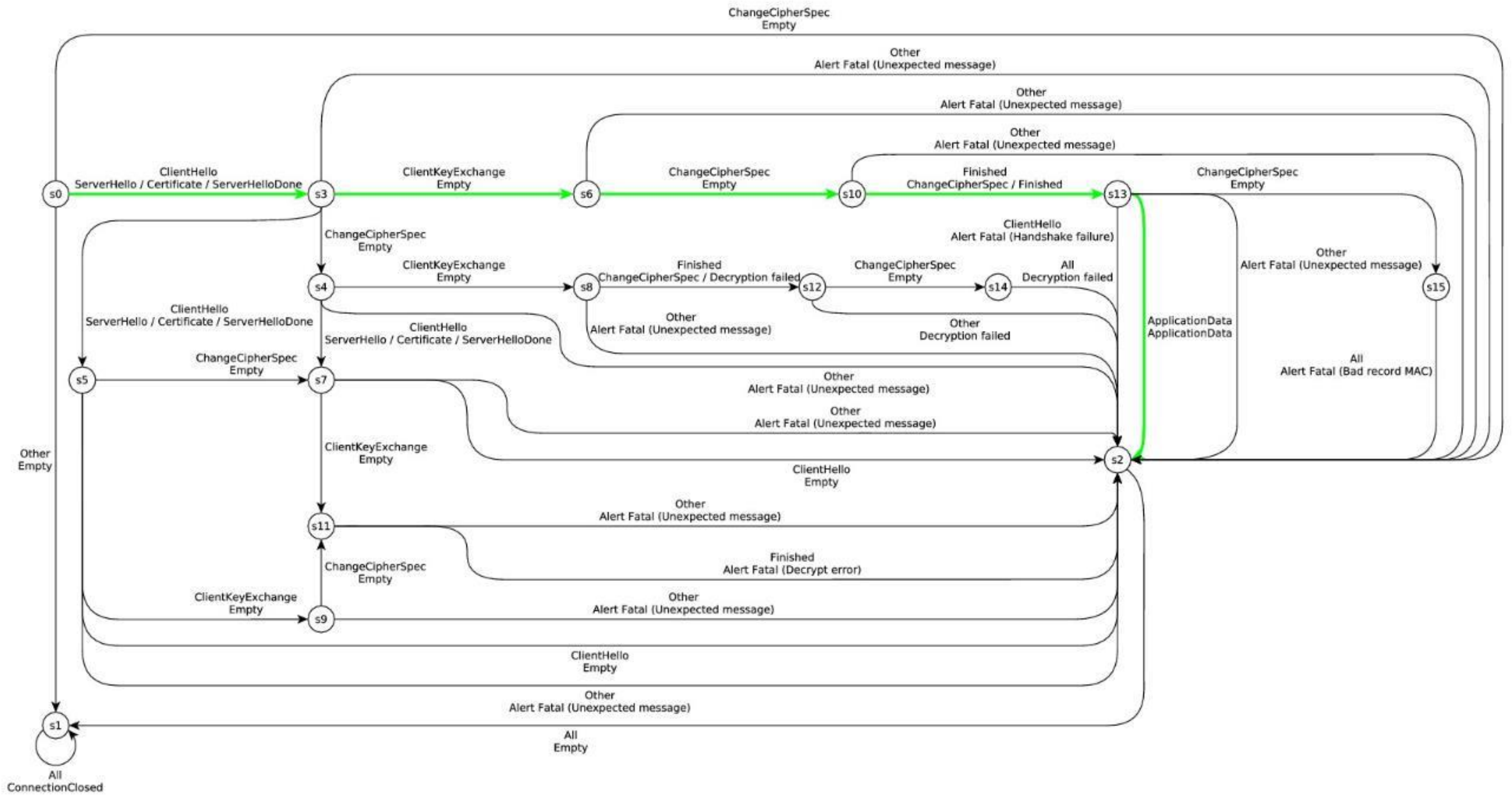
Would you trust this to be secure?

State machine learning for TLS


s0
ClientHello
ServerHello / Certificate / ServerHelloDone
s2
ClientKeyExchange
Empty
s4
ChangeCipherSpec
Empty
s5
Finished
ChangeCipherSpec / Finished
s6
Other
Alert Fatal (Unexpected message)
Other
Alert Fatal (Unexpected message)
Other
Alert Fatal (Unexpected message)
Other
Alert Fatal (Unexpected message)
ClientHello
Alert Fatal (Handshake failure)
Other
Alert Fatal (Unexpected message)
ApplicationData
ApplicationData / Alert Warning (Close notify)
s1
All
Empty
s3
All
ConnectionClosed

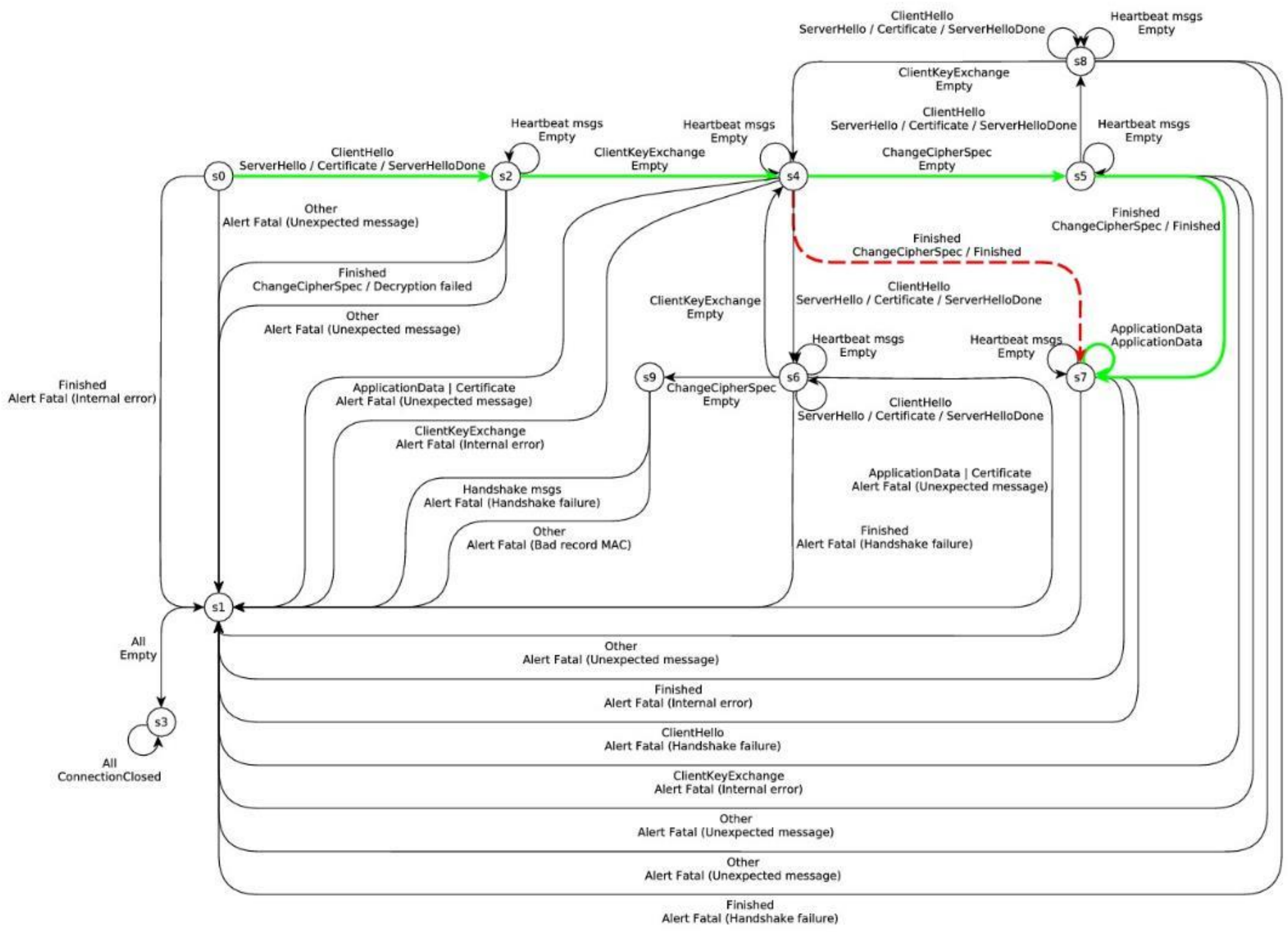
Protocol state machine of the NSS TLS implementation

State machine of OpenSSL



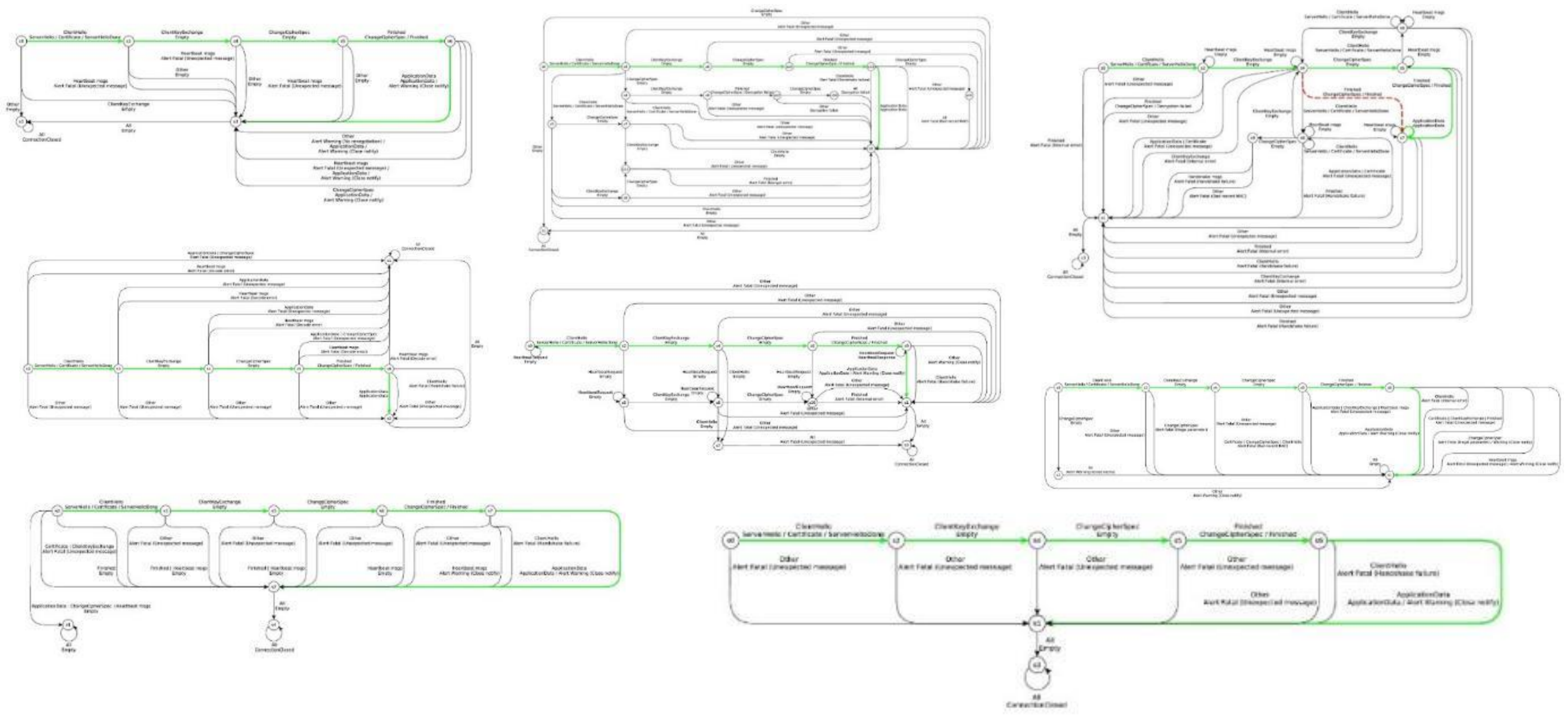
s0
s1
s2
s3
s4
s5
s6
s7
s8
s9
s10
s11
s12
s13
s14
s15
ChangeCipherSpec / Empty
Other / Alert Fatal (Unexpected message)
ClientHello / ServerHello / Certificate / ServerHelloDone
ClientKeyExchange / Empty
ChangeCipherSpec / Empty
Finished / ChangeCipherSpec / Finished
ClientHello / Alert Fatal (Handshake failure)
Finished / ChangeCipherSpec / Decryption failed
All / Decryption failed
Other / Decryption failed
ApplicationData / ApplicationData
All / Alert Fatal (Bad record MAC)
ClientHello / Empty
Finished / Alert Fatal (Decrypt error)
Other / Empty
All / Empty
All / ConnectionClosed

State machine of Java Secure Socket Exchange



ClientHello
ServerHello / Certificate / ServerHelloDone
Heartbeat msgs
Empty
s8
ClientKeyExchange
Empty
ClientHello
ServerHello / Certificate / ServerHelloDone
Heartbeat msgs
Empty
Heartbeat msgs
Empty
Heartbeat msgs
Empty
ClientHello
ServerHello / Certificate / ServerHelloDone
ClientKeyExchange
Empty
ChangeCipherSpec
Empty
s0
s2
s4
s5
Other
Alert Fatal (Unexpected message)
Finished
ChangeCipherSpec / Finished
Finished
ChangeCipherSpec / Finished
Finished
ChangeCipherSpec / Decryption failed
ClientHello
ServerHello / Certificate / ServerHelloDone
ClientKeyExchange
Empty
Other
Alert Fatal (Unexpected message)
ApplicationData
ApplicationData
Heartbeat msgs
Empty
Heartbeat msgs
Empty
Finished
Alert Fatal (Internal error)
s9
s6
s7
ApplicationData | Certificate
Alert Fatal (Unexpected message)
ChangeCipherSpec
Empty
ClientHello
ServerHello / Certificate / ServerHelloDone
ClientKeyExchange
Alert Fatal (Internal error)
ApplicationData | Certificate
Alert Fatal (Unexpected message)
Handshake msgs
Alert Fatal (Handshake failure)
Other
Alert Fatal (Bad record MAC)
Finished
Alert Fatal (Handshake failure)
s1
All
Empty
Other
Alert Fatal (Unexpected message)
Finished
Alert Fatal (Internal error)
s3
ClientHello
Alert Fatal (Handshake failure)
All
ConnectionClosed
ClientKeyExchange
Alert Fatal (Internal error)
Other
Alert Fatal (Unexpected message)
Other
Alert Fatal (Unexpected message)
Finished
Alert Fatal (Handshake failure)

State machine learning for TLS



All implementations we analysed were different! Two had security flaws

Why doesn’t the TLS spec include a state machine?

[Joeri de Ruiter et al., *Protocol state fuzzing of TLS implementations*, USENIX Security 2015]

NO MORE PROSE
SPECIFICATIONS
OF PROTOCOL
STATE MACHINES
USA

Forwarding flaws

[LangSec 2018]

[Strings considered harmful, USENIX ;login:, 2018]

(At least) two types of INPUT problems

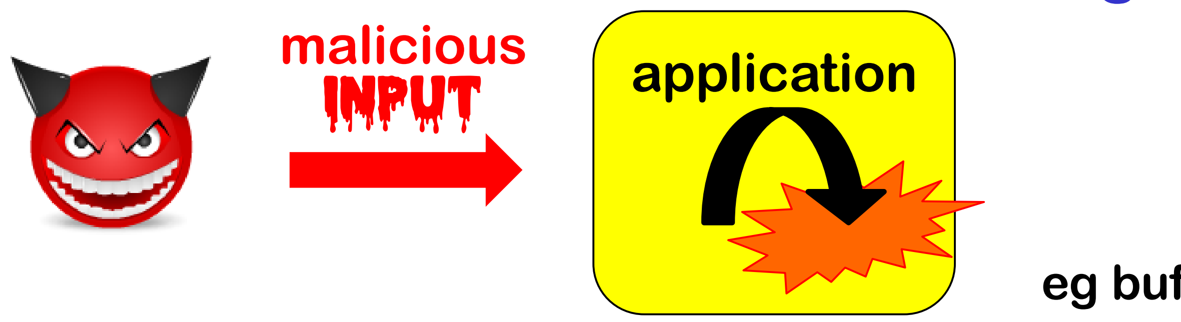
1. Buggy processing & parsing
   - Bug in processing input causes application to go of the rails
   - Classic example: buffer overflow in a PDF viewer, leading to remote code execution

   This is *unintended* behaviour, introduced by *mistake*

2. Flawed forwarding (aka injection attacks)
   - Input is forwarded to *back-end* service/system/API, to cause damage there
   - Classic examples: SQL injection, XSS, format string attack, Word macros

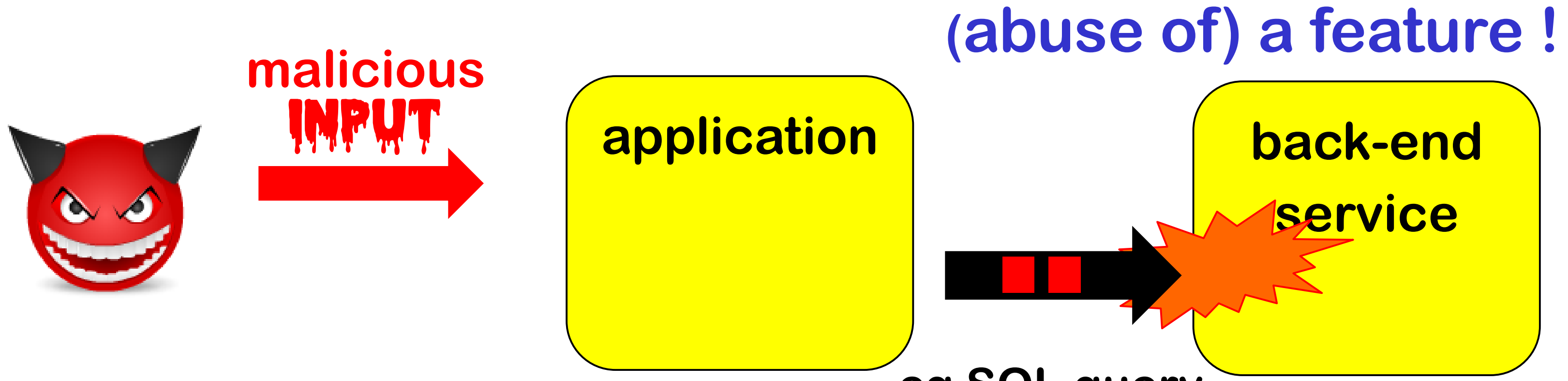
   This is *intended* behaviour of the back-end, introduced *deliberately*, but *exposed by mistake* by the front-end

Processing Flaws



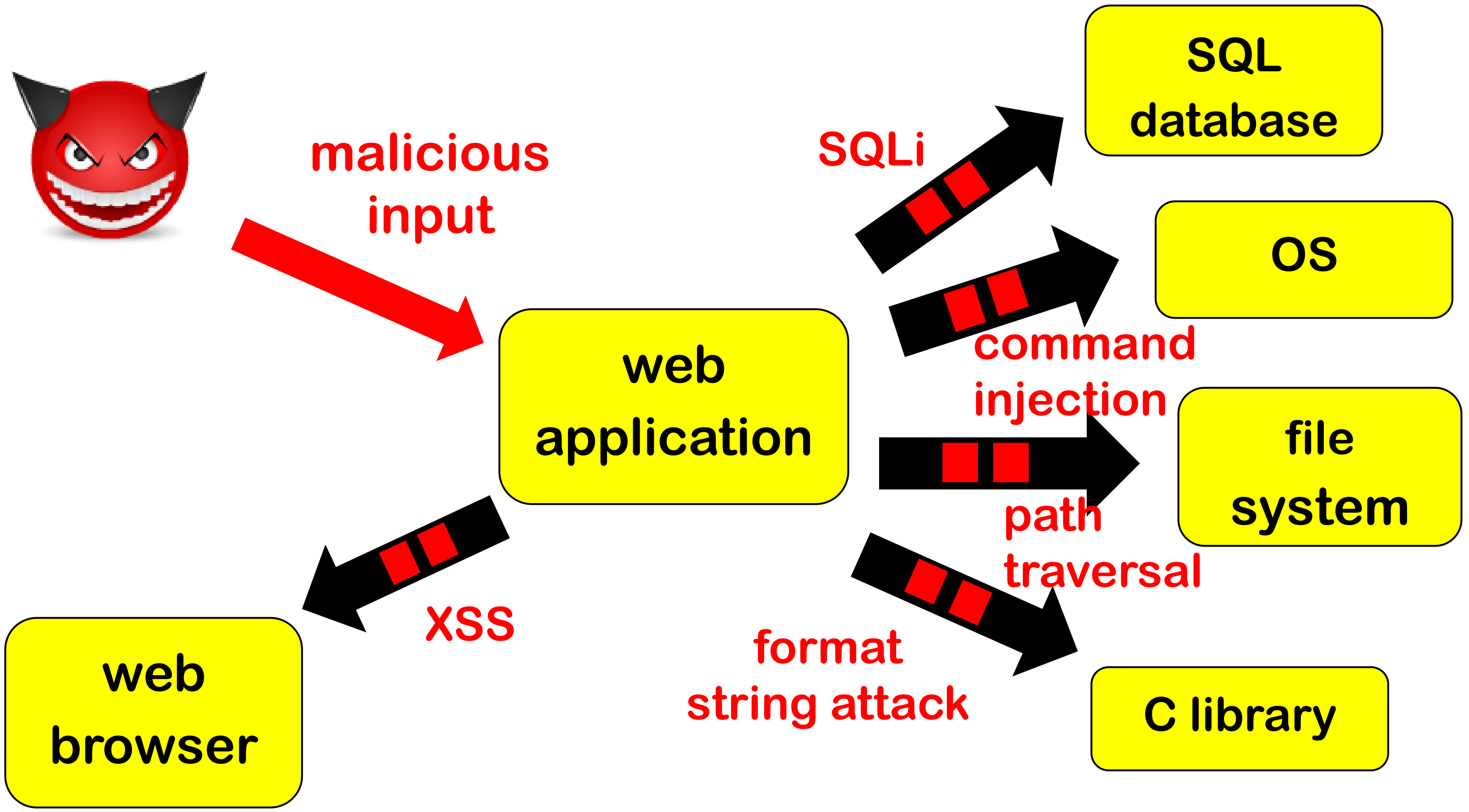
eg buffer overflow in
PDF viewer

Forwarding Flaws



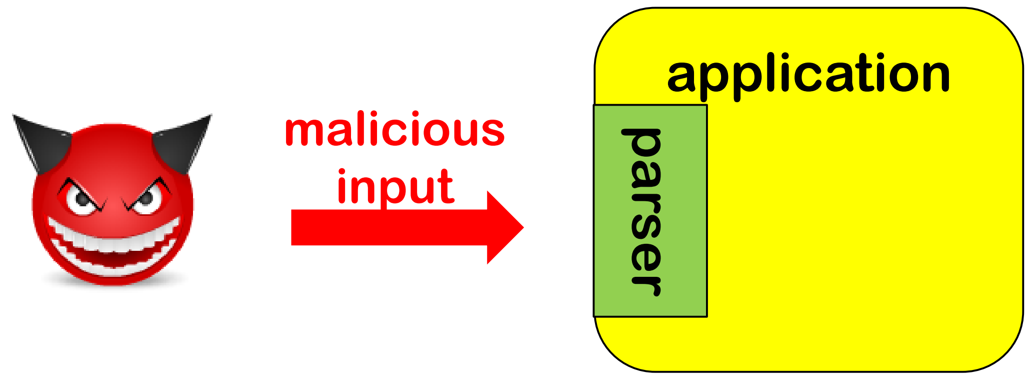
eg SQL query
or Word macro

More back-ends, more languages, more problems



How & where to tackle input problems?

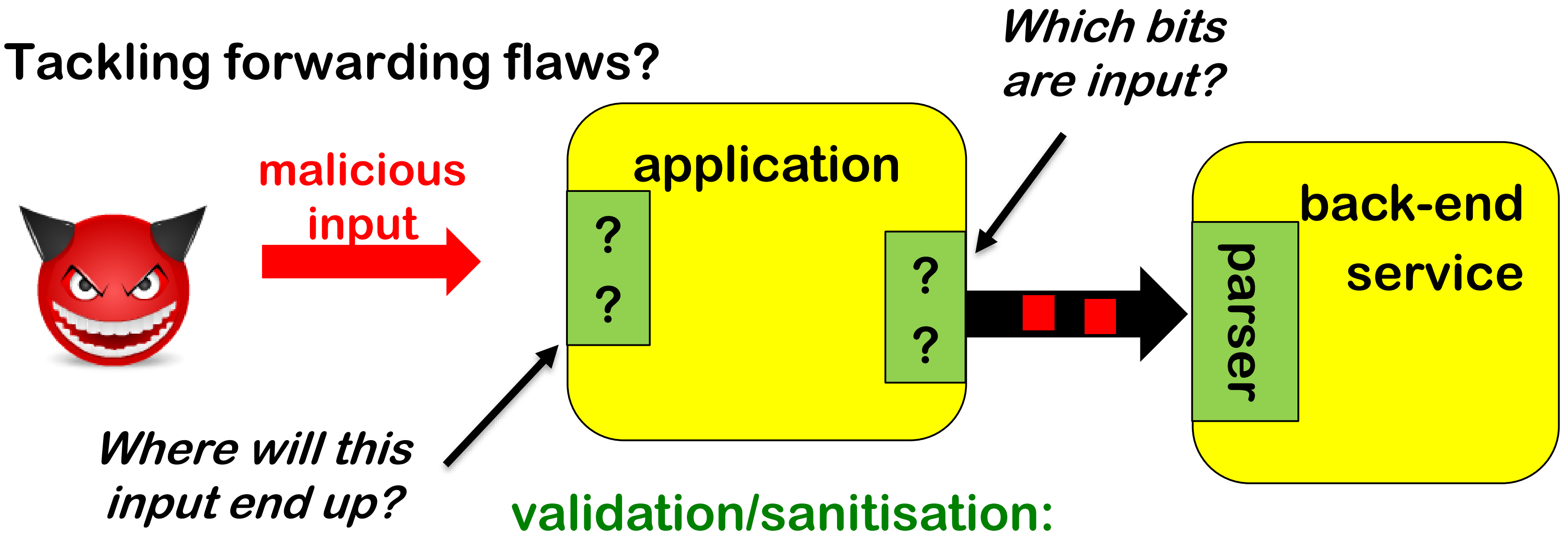
Tackling processing flaws



LangSec approach:

- Simple & clear language spec
- Generated parser code
- Complete parsing before any further processing

Tackling forwarding flaws?



validation/sanitisation:
filtering and/or escaping?

Anti-pattern: STRING CONCATENATION

- Standard recipe for security disaster: concatenating several pieces of data, some of them user input, and passing the result on to some API
  - Classic example: SQL injection
- Note: string concatenation is inverse of parsing

Anti-pattern: STRINGS ⚠

The use of strings in itself is already troublesome

- be it `char*`, `char[]`, `String`, `string`, `StringBuilder`, `...`

- Strings are *useful*, because you use them to represent many things:
  eg. name, file name, email address, URL, shell command, bit of SQL, HTML,...

- This also make strings *dangerous:*

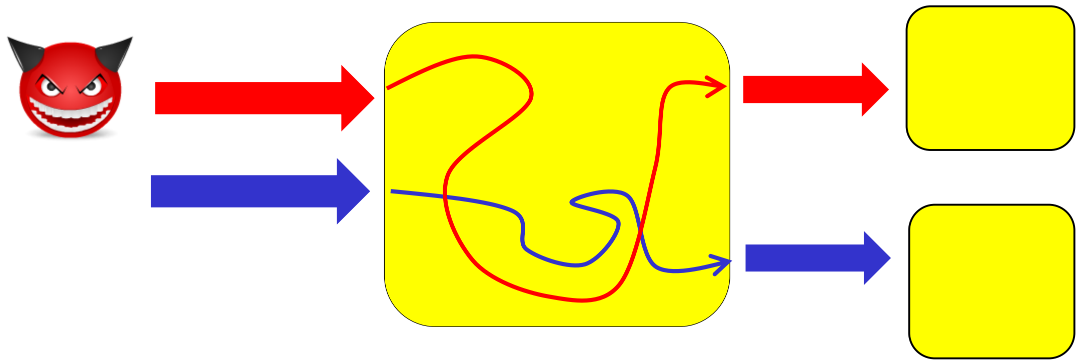
  1. Strings are unstructured & unparsed data, and processing them often involves some interpretation (incl. parsing)
     - If you have a shotgun parser, your code will use strings
  2. The same string may be handled & interpreted in many – possibly unexpected – ways
  3. A string parameter in an API call often hides an expressive & powerful language

Remedy: (1) use types to distinguish *languages*

- **Instead of using strings for everything,**
  use different types to distinguish different kinds of data

  Eg different types for HTML, URLs, file names, user names, paths, …

- **Advantages:**
  - **Types provide structured data**
  - **No ambiguity about the intended use of data**

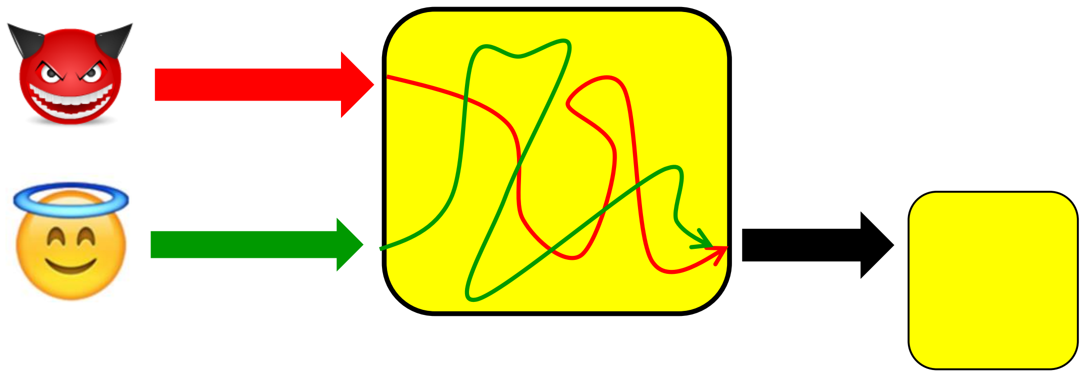


Remedy: (2) use types to distinguish *trust levels*

- Use information flow types to track the origins of data

  and/or to control destinations

  - Eg distinguish untrusted user input vs compile-time constants



The two uses of types, to distinguish (1) languages or (2) trust levels, are orthogonal and can be combined.

Example: Trusted Types for DOM Manipulation

DOM-based XSS flaws are proving difficult to root out

- as latest attacks using script gadgets show

[Lekies et al., *Code-Reuse Attacks for the Web: Breaking Cross-Site Scripting Mitigations via Script Gadgets*, CCS'17]

Google's Trusted Types initiative [https://github.com/WICG/trusted-types] replaces string-based APIs with typed APIs

- using `TrustedHtml, TrustedUrl, TrustedScriptUrl, TrustedJavaScript,`…
- 'safe' APIs for back-ends which auto-escape untrusted inputs

Conclusions

Security is about SOFTWARE & handling INPUT !

- Input handling problems typically come from
  - buggy parsing
  - buggy state machines
  - *unintended* parsing due to forwarding

  Ironically – or embarrassingly –, parsing is a very well-understood area of computer science…

  - We all teach finite state machines, regular expressions, grammars, … to our students, but will they ever use them in practice?
- LangSec provides some constructive remedies to tackle this
  - Have clear, simple & well-specified input languages
  - *Generate* parser code
  - Don't use STRINGS
  - Do use types, to distinguish languages & trust levels

Thanks for your attention!