Software security for no one?

Erik Poll

Digital Security

Radboud University Nijmegen

This talk

- We can't seem to produce secure IT systems

  Over 6189 CVEs recorded in 2018 at cve.mitre.org

- What are root causes behind many security vulnerabilities?
- Can we tackle some of them?

- Much of this talk revolves around parsing
- Most of this should be familiar if you know about LangSec

How come systems can be hacked?

1. Software (“*hacking*”)

   a bug!

   Classical example: the buffer overflow

   Every line of code processing input from outside is a potential security problem.

2. Humans (“*social engineering*”)

   Classic example: send phishing emails to get passwords

3. The combination of software and humans

   a feature?

   Classic example: email Word attachments with malicious macros

Common theme: INPUT

- Software or people mishandling malicious input is *the* common theme in many attacks
  - eg buffer overflow, format string attack, command injection, path traversal, SQL injection, XSS (Cross Site Scripting), Word macros, XML injection, LDAP injection, zip bombs, deserialization attacks, …

- Garbage In, Garbage Out

` leads to

Malicious Garbage In, Security Incident Out

Two types of input problems in software

1. *Buggy* processing

bugs

- Eg buffer overflows

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

2. *Unintended* processing

'features'

- Eg Word macros, SQL injection

This is *intended* behaviour, introduced deliberately, but *exposed by mistake*

This processing can come as a complete surprise:

- systems often involve many more languages (or protocols) than we expect
- these languages may be much more expressive than we expect

Example surprise in processing input

- Windows supports *many notations* for path names
  - classic MS-DOS notation C:\MyData\file.txt
  - file URLs file:///C|/MyData/file.txt
  - UNC (Uniform Naming Convention) \\192.1.1.1\MyData\file.txt

  which can be combined in fun ways, eg file:////192.1.1.1/MyData/file.txt
- Some notations induce *unexpected behaviour* , eg
  - UNC paths to remote servers are handled by the SMB protocol
  - SMB sends your password hash to remote server to authentication
    – aka pass the hash
- This can be exploited by SMB relay attacks on applications handling file names
  - CVE-2000-0834 in Windows telnet,
  - CVE-2008-4037 in Windows XP/Server/Vista, …
  - CVE-2016-5166 in Chromium,
  - CVE-2017-3085 & CVE-2016-4271 in Adobe Flash,
  - ZDI-16-395 in Foxit PDF viewer

[Example thanks to Björn Ruytenberg, https://blog.bjornweb.nl]

Making input problems worse

- *Complex* input languages
  making bugs in parsing likely
  - Eg Adobe Flash = JPG+GIF+PNG+H.264/MPEG4+VP6
    +MP3+AAC+Speex+PCM+ADPCM+Nellymoser+G7.11+..
  - Eg see https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=PDF
- *Many* input languages & formats
  making unintended & unexpected processing likely
- *Very expressive* input languages
  making it easy for attackers to do lots of damage

  Eg Powershell Macros in Word,
  Javascript & DOM in HTML5,
  ActionScript in Flash

What to do about this

- Ideally, we'd like *prevent* input problems by
  - by using small number of well-defined & simple languages
  - by generating parser code to avoid buggy parsing

  (See langsec.org)

- How can we recognise that we may have problems
  - with unintended processing? Strings
  - with buggy parsing? Fuzzing

Strings considered harmful

Danger sign for unintended processing: ⚠

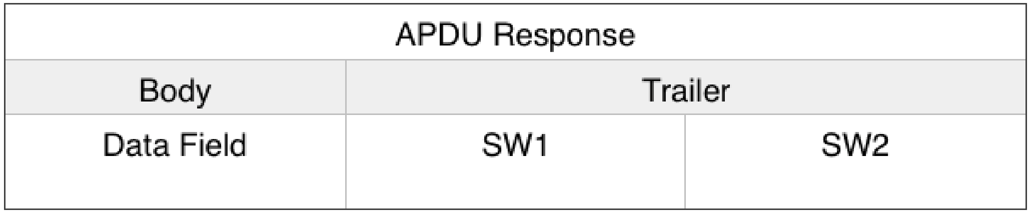
  - Strings and string concatenation
  - API calls that takes a string as argument

- Strings are *useful*, because you can represent all sort of things as strings: eg. file names, URLs, email addresses, shell commands, bits of SQL or HTML,…
- Strings are *dangerous*, because you can represent all sort of things as strings: Hard to know if some API somewhere won't interpret them in way that can do damage
- Proposals to root out DOM-based XSS flaws replace string-based APIs with typed APIs
  - using TrustedHtml, TrustedUrl, TrustedScriptUrl, TrustedJavaScript,…

[Sebastian Lekies, Don't trust the DOM: Bypassing XSS mitigations via script gadgets, OWASP Benelux 2017]

Even processing simple input languages can go wrong

Sending an extended length APDU can crash a contactless payment terminal.



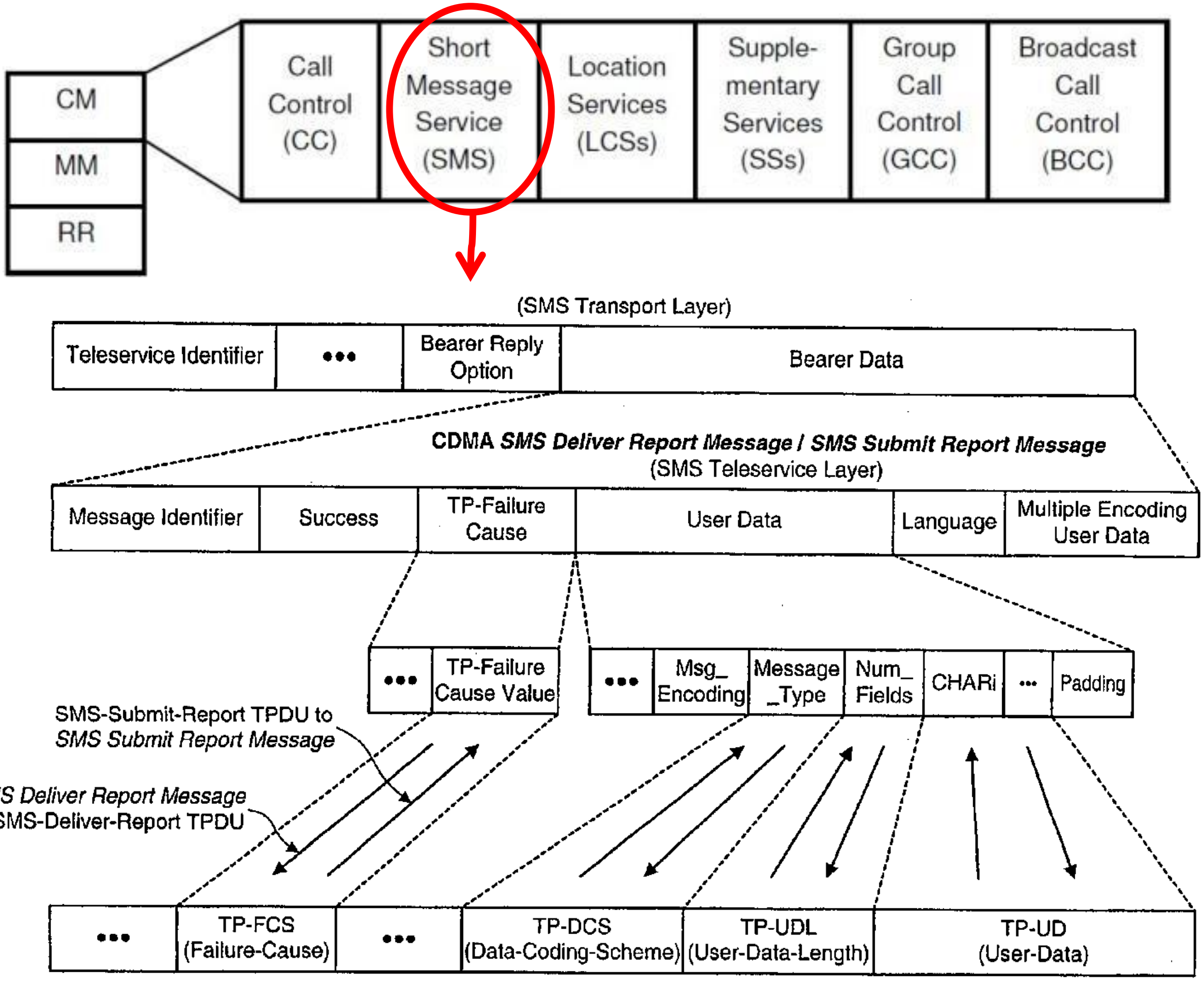
APDU Response		
Body	Trailer	
Data Field	SW1	SW2

[Jordi van den Breekel, A security evaluation and proof-of-concept relay attack on Dutch EMV contactless transactions, MSc thesis, 2014]

Processing complex input languages *will* go wrong

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]

Example: GSM protocol fuzzing

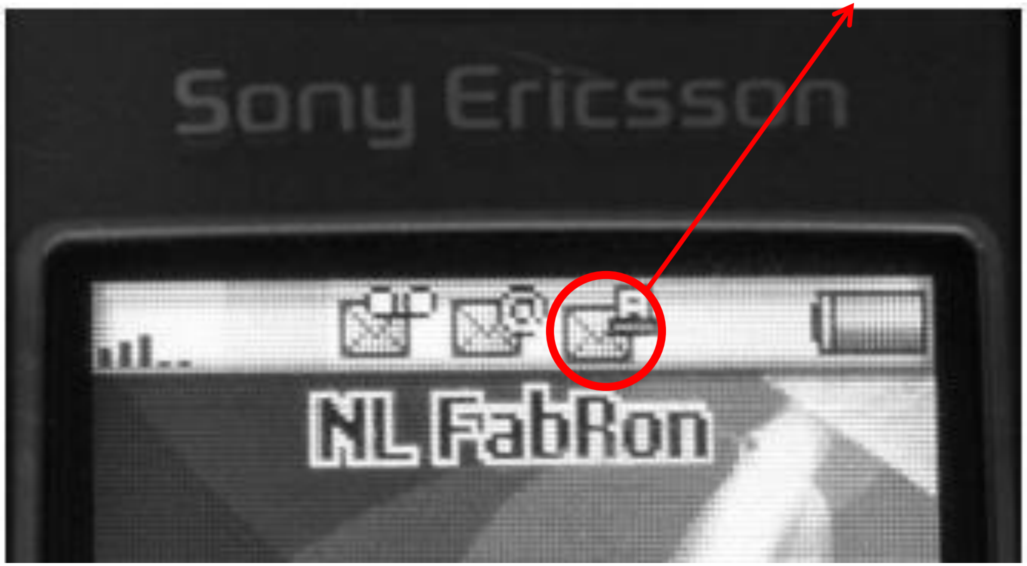
Fuzzing SMS layer of GSM reveals weird functionality in GSM standard and in phones

Example: GSM protocol fuzzing

Fuzzing SMS layer of GSM reveals weird functionality in GSM standard and in phones

- eg warnings about receiving faxes (!?)

you have a fax!



Only way to get rid if this icon; reboot the phone

Example: Fuzzing OCPP [ongoing research by Ivar Derksen]

- OCPP is a protocol for charge points
  - to talk to back-end server
- OCPP can use XML or JSN messages
- Simple classification of messages in

1. malformed JSN/XML
2. well-formed JSN/XML, but not legal OCPP
3. well-formed OCPP

provides an interesting test oracle:

do mal-/well-formed requests trigger mal-/well-formed responses?

This does not involve any understanding of the protocol semantics yet!

Test results with fuzzing an OCPP server

- Mutation fuzzer generates 26,400 variants from 22 example OCPP messages in JSN format
- Problems spotted by our simple test oracle:
  - 945 malformed JSN requests result in malformed JSN response. *Server should never emit malformed JSN!*
  - 75 malformed JSN requests and 40 malformed OCPP requests result in a valid OCPP response that is not an error message. *Server should not process malformed requests!*
- So server violates LangSec principle of no processing before full recognition
- Code is a open-source project touted as ‘premium software’

Conclusions

- Buggy or unintended parsing are root causes of much security trouble
  - As highlighted by the LangSec (langsec.org) approach, though that emphasises buggy parsing over unintended parsing
- Ironically, parsing is one the best-understood techniques in computer science
  - We have regular expressions, context-free grammars, EBNF, ABNF, finite automata, … and tools to generate code from these. Apparently, nobody is using these…?
- Heavy use of strings in code is a warning sign
- Fuzzing is a great way to get a first impression of the quality of code, even without understanding any protocol semantics.

Thanks for your attention

http://langsec.org

Paper deadline for LangSec 2018 @ IEEE S&P: January 31th