Formal Methods & Security?

Erik Poll

Digital Security

Radboud University Nijmegen

FM: A killer application for security?

Maybe security warrants the extra effort & costs of FM?

"Looks like another killer app."

- Highest levels of security certification using Common Criteria *require* the use of FM
  - Common Criteria certifications are not widely used, and when they are, only at lower levels that do not require FM

Specifying security?

Specifying security is hard!

Hacking = exploiting unwanted & unexpected functionality

'There are unknown unknowns' – Donald Rumsfeld

Security specs can degenerate into incomplete lists of 'negative' properties that should not be possible

- **eg '*Cross-Site Scripting XSS should be impossible*'**

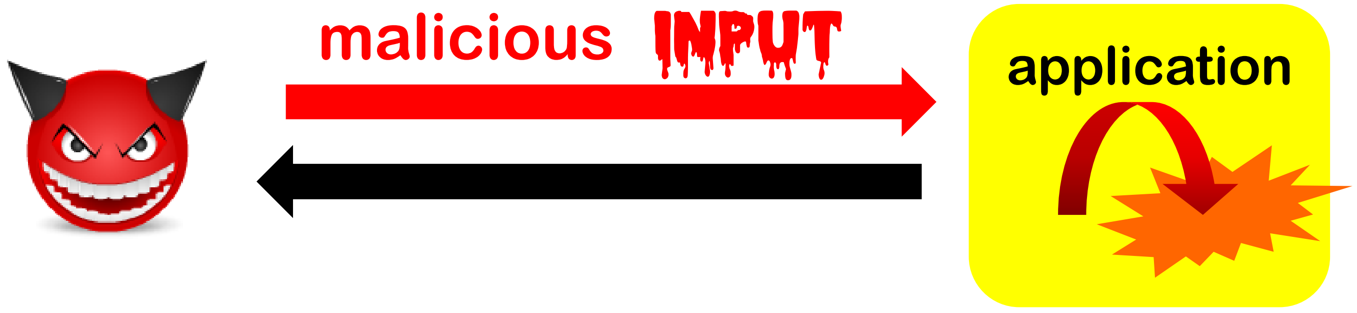
security software ≠ software security

- Obvious target for applying FM: security software
  - ie. software implementing security controls / functionality, such as authentication protocols, security protocols, access control mechanisms, cryptography
  - Some nice results, eg formally verified C implementation of TLS
- However, *ALL* software needs to be secure, not just the security software
  - eg bugs in PDF viewers, image processing software, …

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

- *Woody Allen*

The I/O attacker model ('hacking')



- Garbage In, Garbage Out
  becomes *Malicious* Garbage In, *Security Incident* Out
- *Attacker goal:* DoS, remote code execution, or anything in between
- *How?* Abusing any buggy functionality & weird behaviour of the application
  - Buffer overflow, integer overflow, mis-processed NULL character, XSS, SQL injection, path traversal, deserialization attacks, ...

Root cause analysis: INPUT handling

- LangSec (language-theoretic security) points to the central role of input languages in causing security flaws
  - ie file formats & protocols such as TCP/IP, TLS, Bluetooth, GSM/UMTS/LTE, HTTP(S), HTML5, URL, XML, S/MIME, Flash, JPG, PDF, Word, Excel, URLs, file names, SMB …
- Many security flaws come down to bad parsing of malformed input
- Root causes:
  - Many & overly complex input languages, stacked & nested
  - Poorly – INFORMALLY – specified input languages
  - Lots of buggy, handwritten parser code then results in lots of weird behaviour for I/O attacker to have fun with

Way forward?

- *Why are people still writing long prose specs of protocols & languages?*
- *Why are people still hand-writing parser code?*

- **Regular expressions, grammars & parser generation are basic FMs that have been around for decades…**

- **DARPA Safe Documents (SafeDocs), https://www.darpa.mil/program/safe-documents, Aug 2018**

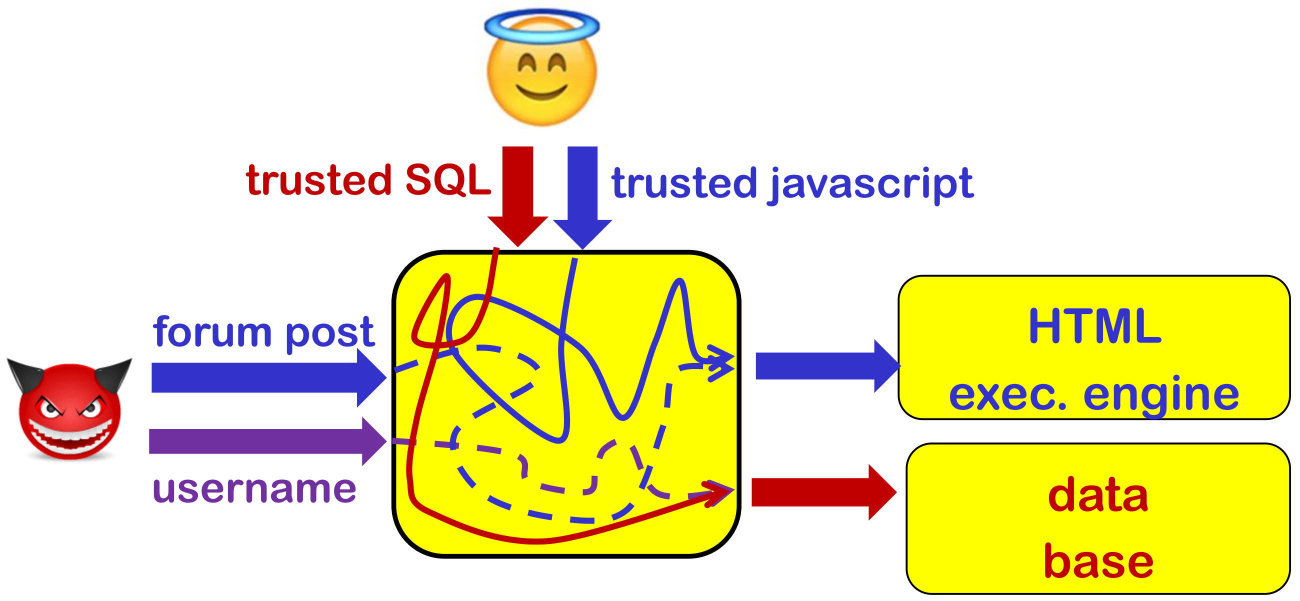
Unintended vs buggy parsing

- In addition to buggy parsing, security problems can also be caused by unintended parsing,
  - eg interpreting a user name as SQL statement, resulting in SQL injection, or as HTML/javascript, resulting in XSS, or choking on a NULL terminator in a user name.
- Root cause
  - application handles many input languages & inputs from many trusted & untrusted sources, and fails to separate these

Way forward?

Can't we use type systems, domain-specific languages, information flow types, … to disentangle

- *different languages? (eg SQL vs HTML vs user names vs ..)*
- *different trust levels? (eg user inputs vs compile-time constants)*



STRINGS
CONSIDERED
HARMFUL
ART
HIST