Step aside! A fuzzy trip down side-channel lane

Side-channel assisted fuzzing in embedded systems

Master thesis, Master Computing Science
a.k.a. TRU/e Master in Cyber Security (`true-security.nl`)

Gerdriaan Mulder, `gmulder@ghcm.nl`

Supervisor: dr. ir. Erik Poll
Second reader: prof. dr. Lejla Batina

10 December 2020

iCIS | Digital Security
Radboud University

"Zoom etiquette"

- During the presentation ($\pm$ 30 minutes):
  - Disable audio transmission.
  - Question? Announce it in chat.
  - Briefly enable audio transmission when given a turn.

"Zoom etiquette"

- During the presentation ($\pm$ 30 minutes):
  - Disable audio transmission.
  - Question? Announce it in chat.
  - Briefly enable audio transmission when given a turn.
- Question round after the presentation:
  - Keep audio transmission *disabled*.
  - Question or comment? Announce it in chat.
  - Briefly enable audio transmission when given a turn.

Outline

Context

Assessing the security of embedded systems:

- Typically no source code available
- Hardware may not be designed to protect secrets
- Introspection or debugging is difficult

Context

Assessing the security of embedded systems:

- Typically no source code available
- Hardware may not be designed to protect secrets
- Introspection or debugging is difficult

Finding flaws in embedded systems:

- Observe side-channels of the embedded system
- *Fuzzing* the embedded system

Context

Assessing the security of embedded systems:

- Typically no source code available
- Hardware may not be designed to protect secrets
- Introspection or debugging is difficult

Finding flaws in embedded systems:

- Observe side-channels of the embedded system
- *Fuzzing* the embedded system

This thesis combines three topics:

- Fuzzers
- Side-channels
- *Embedded system:* Smartcards

Preliminaries — Fuzzers

- Testing many different *inputs* against a *fuzzing target*

Preliminaries — Fuzzers

- Testing many different *inputs* against a *fuzzing target*
- *Goal:* Finding many different *execution paths*

Preliminaries — Fuzzers

- Testing many different *inputs* against a *fuzzing target*
- *Goal:* Finding many different *execution paths*
- *Level of insight:* White-box, gray-box, black-box

Preliminaries — Fuzzers

- Testing many different *inputs* against a *fuzzing target*
- *Goal:* Finding many different *execution paths*
- *Level of insight:* White-box, gray-box, black-box
- *Types:* Generational, mutational, evoluationary, differential

Preliminaries — Fuzzers

- Testing many different *inputs* against a *fuzzing target*
- *Goal:* Finding many different *execution paths*
- *Level of insight:* White-box, gray-box, black-box
- *Types:* Generational, mutational, evoluationary, differential

Two fuzzers used in this thesis

Preliminaries — Fuzzers

- Testing many different *inputs* against a *fuzzing target*
- *Goal:* Finding many different *execution paths*
- *Level of insight:* White-box, gray-box, black-box
- *Types:* Generational, mutational, evoluationary, differential

Two fuzzers used in this thesis

- AFL, *American Fuzzy Lop*[1] (mutational, evolutionary)
- DifFuzz[3][1] (differential, based on AFL)

[1] Not part of this presentation

Preliminaries — Side-channels

- Observable effects of the execution of a program

Preliminaries — Side-channels

- Observable effects of the execution of a program
- Execution time, or *timing side-channel*

Preliminaries — Side-channels

- Observable effects of the execution of a program
- Execution time, or *timing side-channel*
- Power usage, or *power side-channel*

Preliminaries — Side-channels

- Observable effects of the execution of a program
- Execution time, or *timing side-channel*

Example: PIN[2] checker

[2] Personal Identification Number

Preliminaries — Side-channels

- Observable effects of the execution of a program
- Execution time, or *timing side-channel*

Example: PIN[2] checker

1. Report correct/incorrect PIN after the first wrong digit

[2] Personal Identification Number

Preliminaries — Side-channels

- Observable effects of the execution of a program
- Execution time, or *timing side-channel*

Example: PIN[2] checker

1. Report correct/incorrect PIN after the first wrong digit
2. Report correct/incorrect PIN after all digits have been checked

[2] Personal Identification Number

Preliminaries — Side-channels

- Observable effects of the execution of a program
- Execution time, or *timing side-channel*

Example: PIN[2] checker

1. Report correct/incorrect PIN after the first wrong digit
2. Report correct/incorrect PIN after all digits have been checked

1. Timing side-channel!

[2] Personal Identification Number

Preliminaries — Side-channels

- Observable effects of the execution of a program
- Execution time, or *timing side-channel*

Example: PIN[2] checker

1. Report correct/incorrect PIN after the first wrong digit
2. Report correct/incorrect PIN after all digits have been checked

1. Timing side-channel!
2. *Constant-time*

[2] Personal Identification Number

Preliminaries — Side-channels

- Observable effects of the execution of a program
- Execution time, or *timing side-channel*

Example: PIN[2] checker

1. Report correct/incorrect PIN after the first wrong digit
2. Report correct/incorrect PIN after all digits have been checked

1. Timing side-channel!
2. *Constant-time* (under certain conditions)

[2] Personal Identification Number

Preliminaries — Side-channels (applications)

1. Uncovering a secret
   - PIN, password, secret key, etc.
   - "Squeezing a key through a carry bit", Filippo Valsorda (34C3)

Preliminaries — Side-channels (applications)

1. Uncovering a secret
   - PIN, password, secret key, etc.
   - "Squeezing a key through a carry bit", Filippo Valsorda (34C3)
2. Guiding a fuzzer
   a. find two inputs that result in minimum and maximum resource consumption

   b. find enough inputs that result in the biggest code coverage

Preliminaries — Side-channels (applications)

1. Uncovering a secret
   - PIN, password, secret key, etc.
   - "Squeezing a key through a carry bit", Filippo Valsorda (34C3)
2. Guiding a fuzzer
   a. find two inputs that result in minimum and maximum resource consumption
      - Low resource consumption: *invalid input*
      - High resource consumption: *valid input*
   b. find enough inputs that result in the biggest code coverage

Preliminaries — Side-channels (applications)

1. Uncovering a secret
   - PIN, password, secret key, etc.
   - "Squeezing a key through a carry bit", Filippo Valsorda (34C3)
2. Guiding a fuzzer
   a. find two inputs that result in minimum and maximum resource consumption
      - Low resource consumption: *invalid input*
      - High resource consumption: *valid input*
   b. find enough inputs that result in the biggest code coverage
      - information on branches taken
      - exit codes indicating valid/invalid input, length, etc.

Preliminaries — Smartcards

Example of a smartcard (banking card, public domain)

Preliminaries — Smartcards

- Embedded systems providing (security-related) applications
  - Banking card, telephony (SIM) card, *OV-Chipkaart*
  - *Contact* vs. *contactless* cards
  - Personalization or provisioning
  - Smartcard *terminal*

Preliminaries — Smartcards

- Embedded systems providing (security-related) applications
  - Banking card, telephony (SIM) card, *OV-Chipkaart*
  - *Contact* vs. *contactless* cards
  - Personalization or provisioning
  - Smartcard *terminal*
- Standardized through ISO7816[2]
  - *Application Protocol Data Unit* (APDU), "message"
  - Status words

Preliminaries — Smartcards

- Embedded systems providing (security-related) applications
  - Banking card, telephony (SIM) card, *OV-Chipkaart*
  - *Contact* vs. *contactless* cards
  - Personalization or provisioning
  - Smartcard *terminal*
- Standardized through ISO7816[2]
  - *Application Protocol Data Unit* (APDU), "message"
  - Status words
- Java Card
  - Programming language: subset of Java
  - Not limited to one hardware platform
  - Implements ISO7816

Research questions

1. *What fuzzers are already using side-channels?*
2. *How can we safely interface a Java Card smartcard with the fuzzer AFL?*
3. *How can we provide the fuzzer AFL with side-channel information?*

Case studies

Case study I: *PasswordEq*

- Custom Java Card applet
- Fixed password of 10 bytes
- Two password compare instructions:
  1. *Unsafe* with regards to timing
  2. *Safe* with regards to timing
- Status words report *Incorrect length*, *Correct/incorrect password*

Case studies

Case study I: *PasswordEq*

- Custom Java Card applet
- Fixed password of 10 bytes
- Two password compare instructions:
  1. *Unsafe* with regards to timing
  2. *Safe* with regards to timing
- Status words report *Incorrect length*, *Correct/incorrect password*

Case study II: Unprovisioned SIM card

- Instruction set discovery using status words

Experiments

Highlighting two of the six experiments:

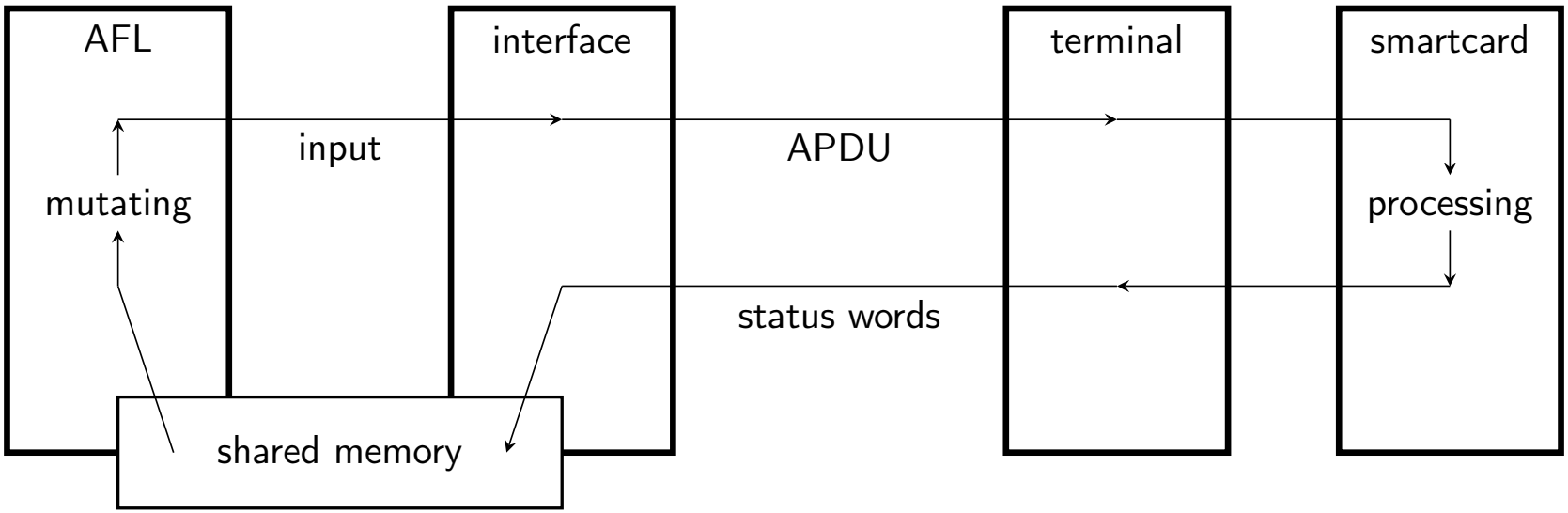
1. Smartcard instruction set discovery using status words
2. `PasswordEq` password extraction using status words and timing

Materials used for these experiments:

- JavaCOS A40 Java Card smartcard / unprovisioned SIM card
- OmniKey CardMan 5121 smartcard terminal
- Laptop[3] running a Linux distribution
- Vanilla AFL
- Custom interface program
  - Translation layer between AFL and Java Card
  - Filtering of unwanted inputs
  - Logging

[3] i5-3320M @ 2.60GHz CPU, 16GB memory

Experiments — Setup



Schematic overview of the interaction between smartcard and AFL

Experiment I: smartcard instruction set discovery

- Unprovisioned / non-personalized SIM card
- Black-box approach
- *Pseudo* side-channel status words
- Looking for status words that indicate "an implemented instruction"
- Ideally faster than brute-force approach, due to feedback loop
- Fuzzing time: 30 minutes

Experiment I: smartcard instruction set discovery

- Unprovisioned / non-personalized SIM card
- Black-box approach
- *Pseudo* side-channel status words
- Looking for status words that indicate "an implemented instruction"
- Ideally faster than brute-force approach, due to feedback loop
- Fuzzing time: 30 minutes

Results:

- Throughput: 17 APDUs per second
- 109 instructions found

Experiment I: smartcard instruction set discovery

- Unprovisioned / non-personalized SIM card
- Black-box approach
- *Pseudo* side-channel status words
- Looking for status words that indicate "an implemented instruction"
- Ideally faster than brute-force approach, due to feedback loop
- Fuzzing time: 30 minutes

Results:

- Throughput: 17 APDUs per second
- 109 instructions found
- Estimated brute-force time: 14 minutes

Experiment I: smartcard instruction set discovery

- Unprovisioned / non-personalized SIM card
- Black-box approach
- *Pseudo* side-channel status words
- Looking for status words that indicate "an implemented instruction"
- Ideally faster than brute-force approach, due to feedback loop
- Fuzzing time: 30 minutes

Results:

- Throughput: 17 APDUs per second
- 109 instructions found
- Estimated brute-force time: 14 minutes

Conclusion:

- Fuzzing for finding a smartcard's instruction set works
- Using a fuzzer is *less* time-efficient than a brute-force approach

Experiment II: `PasswordEq` password extraction

Recap of the `PasswordEq` Java Card applet:

- Fixed password of 10 bytes
- Two password checking instructions:
  1. *Unsafe* with regards to timing
  2. *Safe* with regards to timing
- Status words report *Incorrect length*, *Correct/incorrect password*

Experiment II: `PasswordEq` password extraction

Approach:

1. Let AFL generate APDUs
2. Send APDUs to the Java Card smartcard
3. Record status words in AFL's shared memory
4. Record execution time in AFL's shared memory
5. Repeat

Experiment II: `PasswordEq` password extraction

Approach:

1. Let AFL generate APDUs
2. Send APDUs to the Java Card smartcard
3. Record status words in AFL's shared memory
4. Record execution time in AFL's shared memory
5. Repeat

- The timing side-channel reveals how many characters were correct
  - This reduces the search space from 256^{10} to $256 \cdot 10$, or linear search time rather than exponential
- Ideally, AFL zooms in on the *unsafe* password checking instruction
- Ideally, AFL finds the correct length, and the correct password, due to timing information

Experiment II: `PasswordEq` password extraction

Results, after fuzzing 1 hour and 40 minutes:

- AFL found *safe* and *unsafe* instructions
- AFL found the correct length *parameter*
- AFL *did not* find the correct password

Experiment II: `PasswordEq` password extraction

Results, after fuzzing 1 hour and 40 minutes:

- AFL found *safe* and *unsafe* instructions
- AFL found the correct length *parameter*
- AFL *did not* find the correct password

Conclusion

- Adding timing information did not result in finding the password
- Complexity of the stored password
- Discrepancy between length parameter and length of the supplied password
  - *Example:* Length parameter: 10, actual password length: 5
  - Applet: length correct, password incorrect

Future work

- Fully automated instruction set discovery using side-channels

Future work

- Fully automated instruction set discovery using side-channels
- Feeding a power side-channel to AFL

Future work

- Fully automated instruction set discovery using side-channels
- Feeding a power side-channel to AFL
- White-box instrumentation of Java Card applets

Wrapping up

- Research questions
  1. *What fuzzers are already using side-channels?*
  2. *How can we safely interface a Java Card smartcard with the fuzzer AFL?*
  3. *How can we provide the fuzzer AFL with side-channel information?*

Wrapping up

- Research questions
  1. *What fuzzers are already using side-channels?*
  2. *How can we safely interface a Java Card smartcard with the fuzzer AFL?*
  3. *How can we provide the fuzzer AFL with side-channel information?*

1. DifFuzz

Wrapping up

- Research questions
  1. *What fuzzers are already using side-channels?*
  2. *How can we safely interface a Java Card smartcard with the fuzzer AFL?*
  3. *How can we provide the fuzzer AFL with side-channel information?*

1. DifFuzz
2. Custom interface program

Wrapping up

- Research questions
  1. *What fuzzers are already using side-channels?*
  2. *How can we safely interface a Java Card smartcard with the fuzzer AFL?*
  3. *How can we provide the fuzzer AFL with side-channel information?*

1. DifFuzz
2. Custom interface program
3. Store it in AFL's shared memory

Thanks for your attention!

Stay safe!

“Zoom etiquette”

- Question round after the presentation:
  - Keep audio transmission *disabled*.
  - Question or comment? Announce it in chat.
  - Briefly enable audio transmission when given a turn.

References I

american fuzzy lop.
http://lcamtuf.coredump.cx/afl/.
Fuzzer used in this thesis.

ISO Central Secretary.
Identification cards — Integrated circuit cards — Part 4: Organization, security and commands for interchange.
Standard ISO/IEC 7816-4:2013, International Organization for Standardization, Geneva, CH, 2013.

Shirin Nilizadeh, Yannic Noller, and Corina S. Păsăreanu.
Diffuzz: Differential fuzzing for side-channel analysis.
In *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, pages 176–187, Piscataway, NJ, USA, 2019. IEEE Press.

List of experiments

1. DifFuzz paper
2. Exposing a timing side-channel in a Java Card applet
3. Using DifFuzz to fuzz a Java Card applet on a smartcard
4. Status words as pseudo side-channel information for AFL
5. Smartcard instruction set discovery using status words
6. Status words, timing, AFL, `PasswordEq`