Van bit tot bug:
Doorheen de lagen van de systeemstack

Jo Van Bulck

DistriNet, KU Leuven, Belgium jo.vanbulck@kuleuven.be

https://vanbulck.net

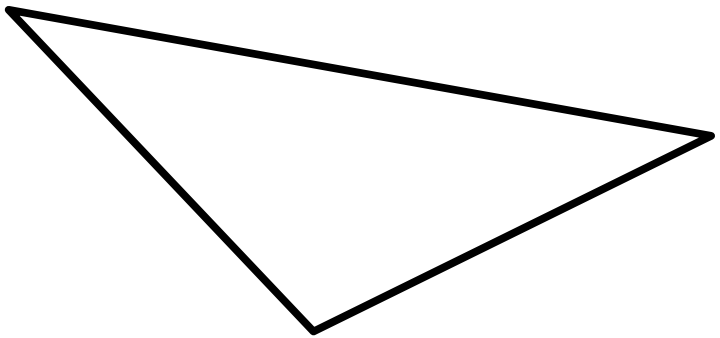
KU Leuven Startdagen, 16/09/2026

DistriNet

KU LEUVEN

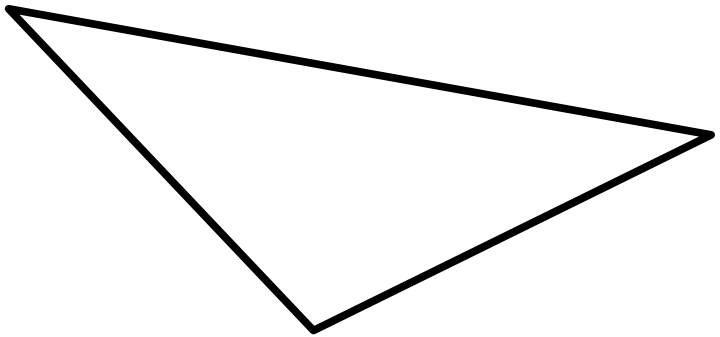
Welkom aan de Universiteit!

Educatie



Welkom aan de Universiteit!

Educatie



Onderzoek

Maatschappelijke dienstverlening

$ whoami

https://vanbulck.net/

Onderzoek: ik breek computers om ze veiliger te maken

- Onderzoeksgroep DistriNet
- HW/SW interface
  - Microarchitectural attacks and defenses
  - Trusted execution environments

Onderwijs: waar kom je me tegen?

- Besturingsystemen (2e ba)
- Wetenschapscommunicatie (2e ba)
- Wetenschappelijke vorming (3e ba)
- Development of Secure Software (ma)

Wetenschappelijke vooruitgang is het ontdekken van een meer en meer begrijpelijke eenvoud

~ Georges Lemaître

MAP OF COMPUTER SCIENCE

THEORETICAL COMPUTER SCIENCE

COMPUTATIONAL COMPLEXITY
DOES PPNP ?
PSPACE
NP COMPLETE
NP
BNP
P

INFORMATION THEORY
0100110110011001
ALGORITHMS
CODING THEORY
PARITY THEORING

CRYPTOGRAPHY
PUBLIC KEY
PRIVATE KEY

COMPUTABILITY THEORY

TURING MACHINE
ALAN TURING
LAMBDA CALCULUS

ALGORITHMS
BUBBLE SORT
MERGE SORT
ANALYSIS OF ALGORITHMS
ALGORITHMS

AND MORE
GRAPH THEORY
PARALLEL PROGRAMMING

OPTIMISATION

BOOLEAN SATISFIABILITY
(SAT)

MACHINE LEARNING
REINFORCEMENT

ARTIFICIAL INTELLIGENCE
A.I.

ROBOTICS

SUPER COMPUTING

COMPUTER VISION

IMAGE PROCESSING

NATURAL LANGUAGE PROCESSING
CHATBOTS

KNOWLEDGE REPRESENTATION

APPLICATIONS

TELEPRESCENCE

AUGMENTED REALITY
YOUTUBE
DOMAIN DE SAIENCE
MAP OF COMPUTER SCIENCE

VIRTUAL REALITY

SIMULATION

HRMBN COMPUTER INTERACTION

INTERNET OF THINGS

COMPUTER ENGINEERING

SCHEDULING

COMPUTER ARCHITECTURE
CPU
FPGA

HARDWARE
MONITOR
SOUND
SSD
SPU
POWER
CPU
MOTHERBOARD

SOFTWARE AND PROGRAMMING LANGUAGES
PYTHON
JAVASCRIPT
WEB APPS
PHP
SWIFT
PSRL
BROWSER
SQL
APPLICATIONS
SAVA
C#
C++
C
OPERATING SYSTEM
ASSEMBLY
BIOS
SYSTEM
MACHINE CODE
COMPILER

COMPILERS

SOFTWARE ENGINEERING
DESIGN METHOD
UNIT TESTING
VERSION CONTROL
OBJECT ORIENTED DESIGN

OPERATING SYSTEMS
iOS
macOS

NETWORKING

DATA MANAGEMENT
DATABASES
PERFORMANCE

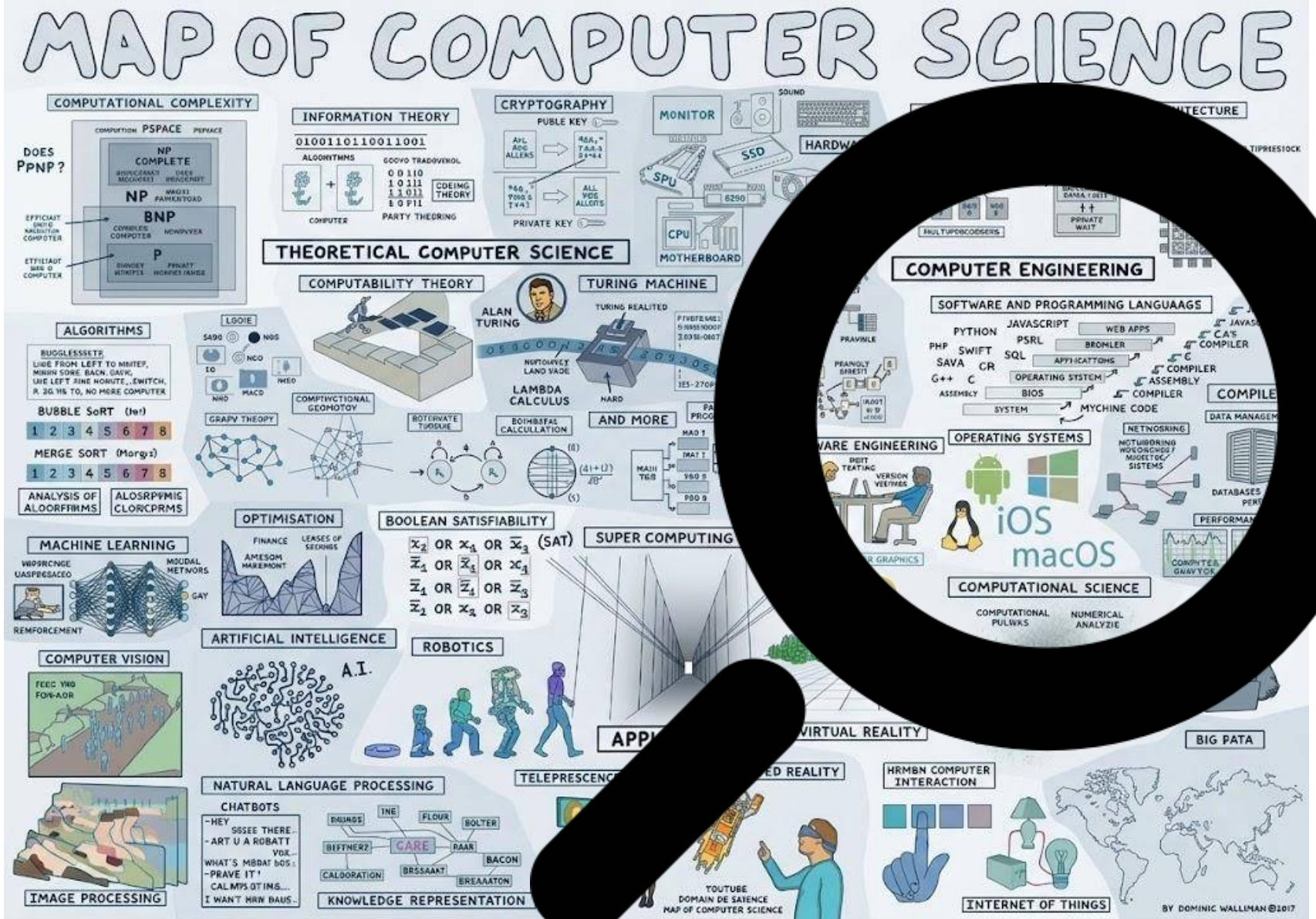
COMPUTER GRAPHICS

COMPUTATIONAL SCIENCE
COMPUTATIONAL PROBLEMS
NUMERICAL ANALYSIS
BIOINFORMATICS
COMPUTATIONAL CHEMISTRY

HACKING

BIG DATA

BY DOMINIC WALLIMAN ©2017


MAP OF COMPUTER SCIENCE
COMPUTATIONAL COMPLEXITY
DOES PPNP ?
PSPACE
NP COMPLETE
NP
BNP
P
INFORMATION THEORY
0100110110011001
ALGORITHMS
COMPUTER
PARITY THEORING
CRYPTOGRAPHY
PUBLE KEY
PRIVATE KEY
MONITOR
SOUND
SSD
SPU
CPU
MOTHERBOARD
HARDWA
THEORETICAL COMPUTER SCIENCE
COMPUTABILITY THEORY
ALAN TURING
TURING MACHINE
TURING REALITED
LAMBDA CALCULUS
AND MORE
ALGORITHMS
BUBBLE SORT
MERGE SORT
ANALYSIS OF ALOORFIMMS
ALOSRPFMIC CLORCPRMS
GRAPH THEORY
OPTIMISATION
BOOLEAN SATISFIABILITY
(SAT)
SUPER COMPUTING
MACHINE LEARNING
REMFORCEMENT
ARTIFICIAL INTELLIGENCE
A.I.
ROBOTICS
COMPUTER VISION
IMAGE PROCESSING
NATURAL LANGUAGE PROCESSING
CHATBOTS
KNOWLEDGE REPRESENTATION
TELEPRESCENCE
VIRTUAL REALITY
HRMBN COMPUTER INTERACTION
INTERNET OF THINGS
BIG PATA
TOUTUBE
DOMAIN DE SAIENCE
MAP OF COMPUTER SCIENCE
BY DOMINIC WALLIMAN ©2017
COMPUTER ENGINEERING
SOFTWARE AND PROGRAMMING LANGUAAGS
PYTHON
JAVASCRIPT
WEB APPS
PHP
SWIFT
PSRL
SQL
SAVA
C
C++
ASSEMBLY
BIOS
SYSTEM
MYCHINE CODE
COMPILER
OPERATING SYSTEMS
NETWORKING
DATA MANAGEM
DATABASES
iOS
macOS
COMPUTATIONAL SCIENCE
COMPUTATIONAL PULWKS
NUMERICAL ANALYZIE

Begrijpelijke eenvoud: “1e wet van computerwetenschappen”

“Any problem in Computer Science can be solved with another layer of indirection”

– Butler W. Lampson

https://en.wikipedia.org/wiki/Fundamental_theorem_of_software_engineering

Begrijpelijke eenvoud: “1e wet van computerwetenschappen”

“Any problem in Computer Science can be solved with another layer of indirection”

– Butler W. Lampson

… except for the problem of too many levels of indirection ;-)

https://en.wikipedia.org/wiki/Fundamental_theorem_of_software_engineering

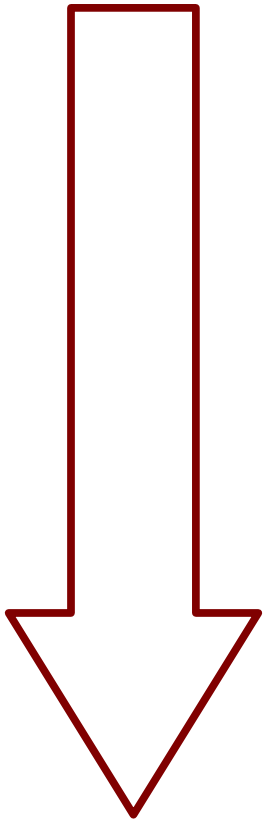
De kracht van abstracties: De systeemstack

https://xkcd.com/676/

- Toren/stack van **abstracties**
- Interface verbergt de **complexiteit** van de implementatie
- Elke laag bouwt verder op **functionaliteit** eronder

Doorheen de lagen van de systeemstack...

Doorheen de lagen van de systeemstack...



Doorheen de lagen van de systeemstack...

Language runtimes

Containers

Web browsers

Databases

Compilers and programming languages

Operating systems

Hardware

Doorheen de lagen van de systeemstack: Studieprogramma

Beginselen van programmeren; modulair programmeren
(1e ba)

top-down

Comparitive Study of Programming languages
(2e ma)

Compilerconstructie
(1e/2e ma)

Doorheen de lagen van de systeemstack: Studieprogramma

Beginselen van programmeren; modulair programmeren (1e ba)

top-down

Comparitive Study of Programming languages (2e ma)

Compilerconstructie (1e/2e ma)

Computer networks (3e ba)

Distributed systems (1e ma)

Gegevensbanken (1e ba)

bottom-up

Besturingssystemen (2e ba)

Computerarchitectuur (2e ba)

Overzicht: Van bit tot bug…

1. SQL injection attacks
(mens → programmeertaal)

2. Memory-safety attacks
(C → machine code)

3. Microarchitectural attacks
(machine code → hardware)

Overzicht: Van bit tot bug…

1. SQL injection attacks
(mens → programmeertaal)

2. Memory-safety attacks
(C → machine code)

3. Microarchitectural attacks
(machine code → hardware)

Gegevensbanken 101: Tabel layout

wachtwoord	naam	email	grades
pass123	Jo	jo@school.edu	A+
secret88	Anna	anna@school.edu	B
qwerty99	Mark	mark@school.edu	A-
stud2026	Joana	joana@school.edu	B+

Gegevensbanken 101: SQL queries

```
SELECT * FROM studenten WHERE naam = 'Jo';
```

SELECT *

The asterisk (*) requests **all columns** (wachtwoord, naam, email, grades) for the matching rows.

FROM studenten

Specifies the target table name in the database where the records are stored.

WHERE naam = 'Jo'

Applies a logical condition. Only rows where the field `naam` is exactly equal to `'Jo'` will be returned.

Gegevensbanken 101: SQL queries

```
SELECT * FROM studenten WHERE naam = 'Jo';
```

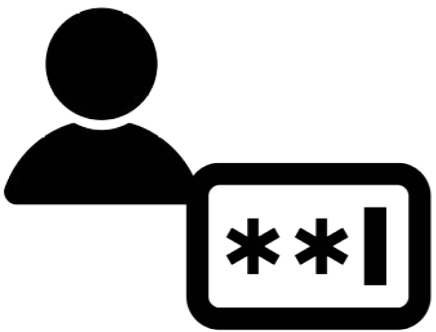
wachtwoord	naam	email	grades
pass123	Jo MATCH	jo@school.edu	A+
secret88	Anna	anna@school.edu	B
qwerty99	Mark	mark@school.edu	A-
stud2026	Joana	joana@school.edu	B+

SQL injectie: Inputinterpretatie over abstractielagen

```
/* naam = wat de gebruiker intypt */
query = "SELECT * FROM studenten "
        + "WHERE naam = '" + naam + "'"
        + "AND wachtwoord = '" + pwd + "'";
```

Wat je verwacht:

```
SELECT * FROM studenten
WHERE naam = 'Jo' AND wachtwoord = 'pass123'
```



SQL injectie: Inputinterpretatie over abstractielagen

```
/* naam = wat de gebruiker intypt */
query = "SELECT * FROM studenten "
        + "WHERE naam = '" + naam + "'"
        + "AND wachtwoord = '" + pwd + "'";
```

Wat een aanvaller intypt

```
SELECT * FROM studenten
WHERE naam = 'Jo' AND wachtwoord = '' OR '1'
= '1' <- altijd waar!
```

=> de databank geeft ALLE studenten terug

SQL injectie: Inputinterpretatie over abstractielagen

```
/* naam = wat de gebruiker intypt */
query = "SELECT * FROM studenten "
        + "WHERE naam = '" + naam + "'"
        + "AND wachtwoord = '" + pwd + "'";
```

Wat een aanvaller intypt

```
SELECT * FROM studenten
WHERE naam = 'Jo' AND wachtwoord = '' OR '1'
= '1' <- altijd waar!
```

=> de databank geeft ALLE studenten terug

Waarom lukt dit?

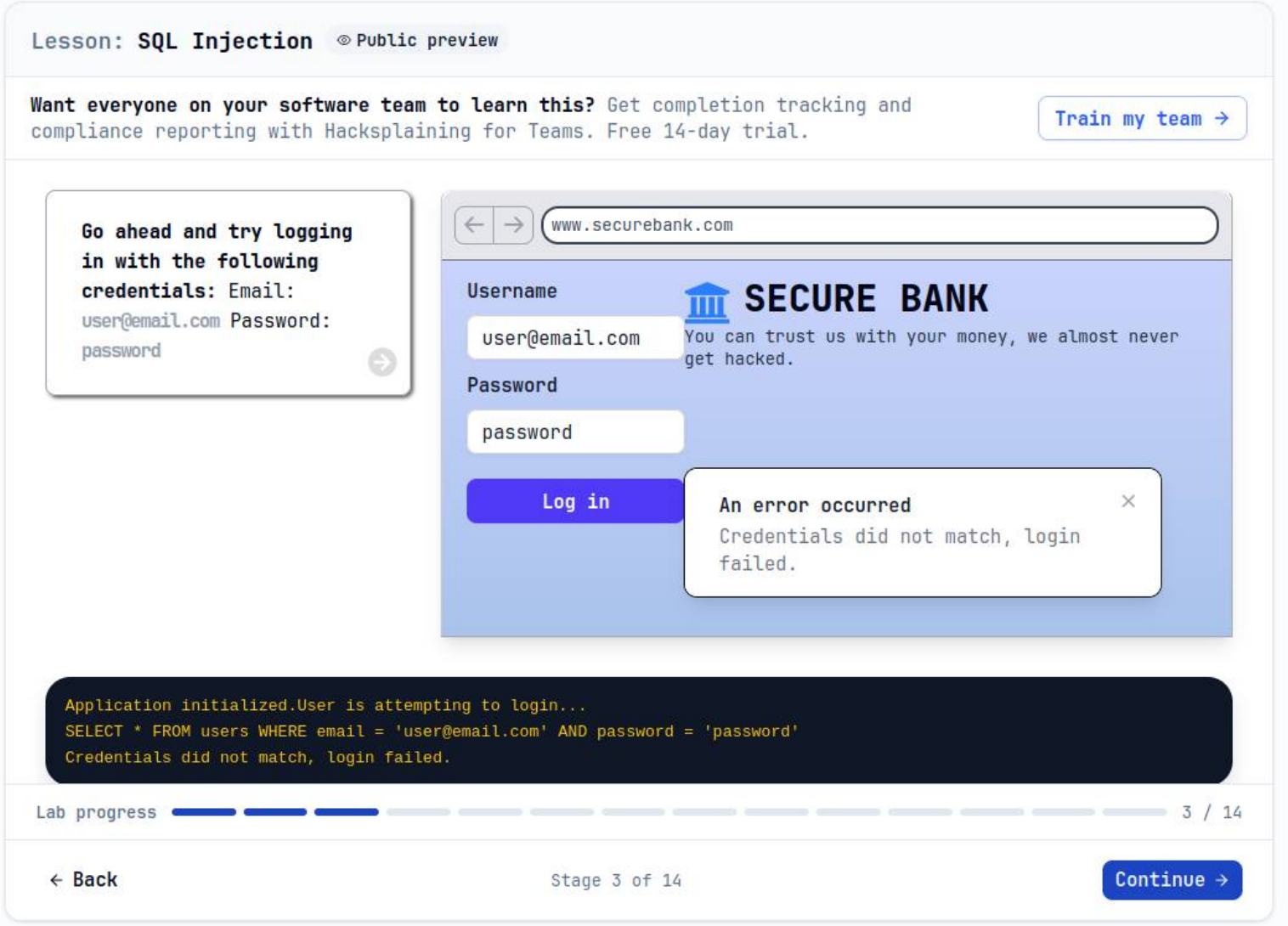
De databank krijgt één lange string:

- Programmeur: *trusted input*
- Gebruiker: *untrusted input*

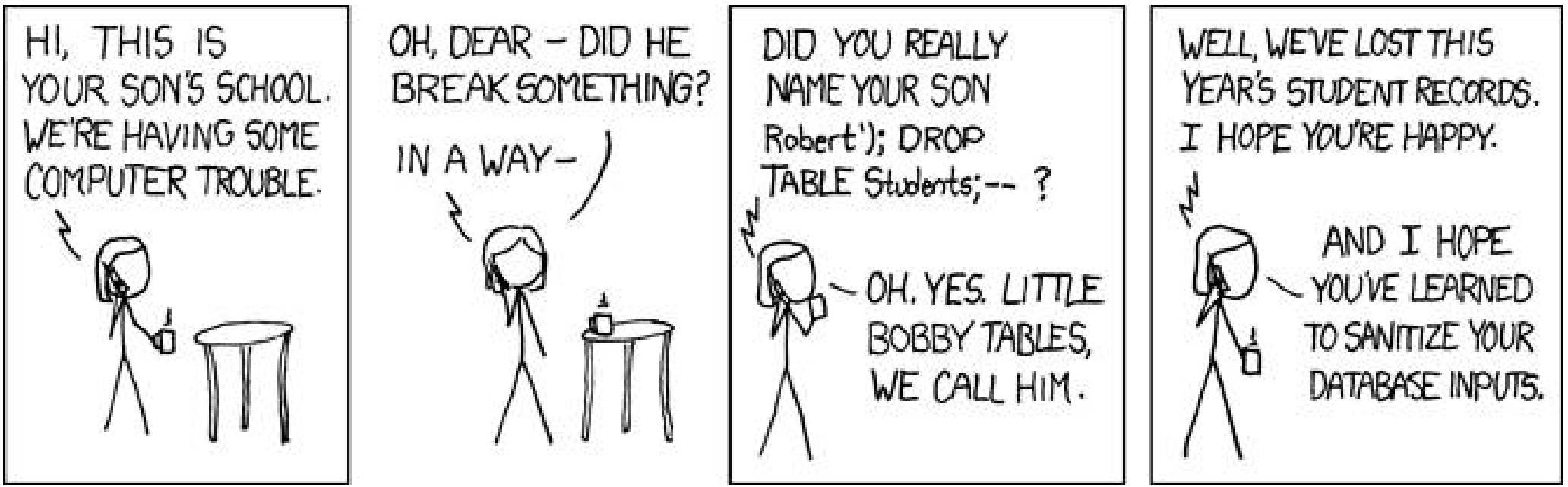
→ Code en data door elkaar(!)

Zelfde bug, andere laag:

- command injection (shell)
- XSS (browser)



https://www.hacksplaining.com/lessons/sql-injection/



https://xkcd.com/327/

Overzicht: Van bit tot bug…

1. SQL injection attacks
(mens → programmeertaal)

2. Memory-safety attacks
(C → machine code)

3. Microarchitectural attacks
(machine code → hardware)

Illusie: Wat we zien in een programmeertaal (C)

```c
#include <stdio.h>

int main(void)
{
    int isAdmin = 0;
    char name[8];

    printf(format: "Enter name: ");
    gets(name);

    if (isAdmin != 0) {
        printf(format: "hello %s: access granted!\n", name);
    } else {
        printf(format: "hello %s: access denied!\n", name);
    }
    return 0;
}
```

Illusie: Wat we zien in een programmeertaal (C)

```c
#include <stdio.h>

int main(void)
{
    int isAdmin = 0;
    char name[8];

    printf(format: "Enter name: ");
    gets(name);

    if (isAdmin != 0) {
        printf(format: "hello %s: access granted!\n", name);
    } else {
        printf(format: "hello %s: access denied!\n", name);
    }
    return 0;
}
```

Toegewezen variabelen:

- `isAdmin` is ingesteld op 0 (standaard geweigerd)
- `name` array bevat maximaal 8 tekens

Illusie: Wat we zien in een programmeertaal (C)

```c
#include <stdio.h>

int main(void)
{
    int isAdmin = 0;
    char name[8];

    printf(format: "Enter name: ");
    gets(name);

    if (isAdmin != 0) {
        printf(format: "hello %s: access granted!\n", name);
    } else {
        printf(format: "hello %s: access denied!\n", name);
    }
    return 0;
}
```

Toegewezen variabelen:

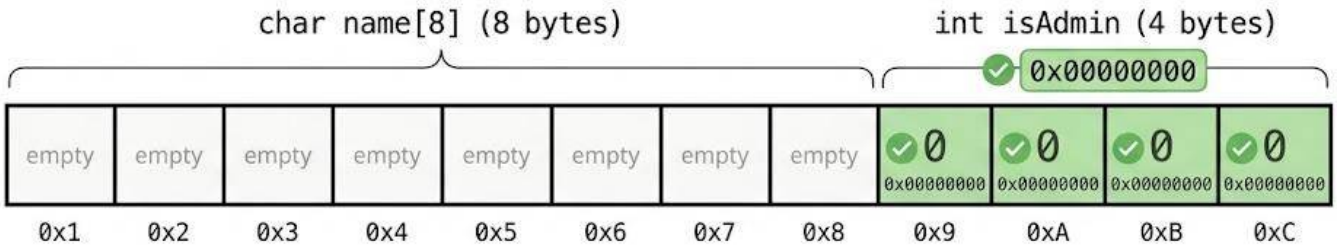
- `isAdmin` is ingesteld op 0 (standaard geweigerd)
- `name` array bevat maximaal 8 tekens

Verwachte Uitvoering:

`isAdmin` = 0 → access denied

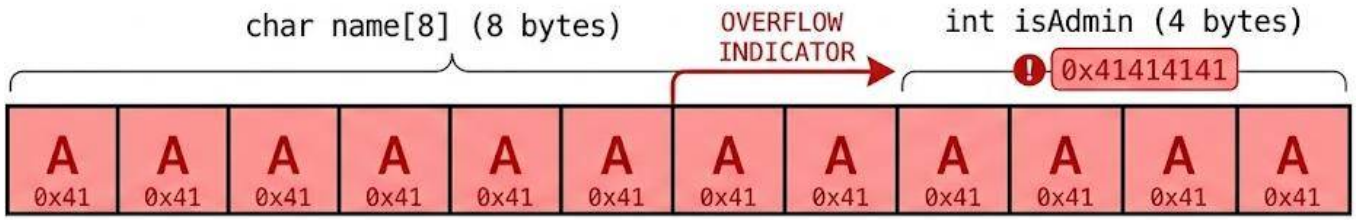
Buffer overflow: onder de abstractie zitten bytes

```c
#include <stdio.h>

int main(void)
{
    int isAdmin = 0;
    char name[8];

    printf(format: "Enter name: ");
    gets(name);

    if (isAdmin != 0) {
        printf(format: "hello %s: access granted!\n", name);
    } else {
        printf(format: "hello %s: access denied!\n", name);
    }
    return 0;
}
```



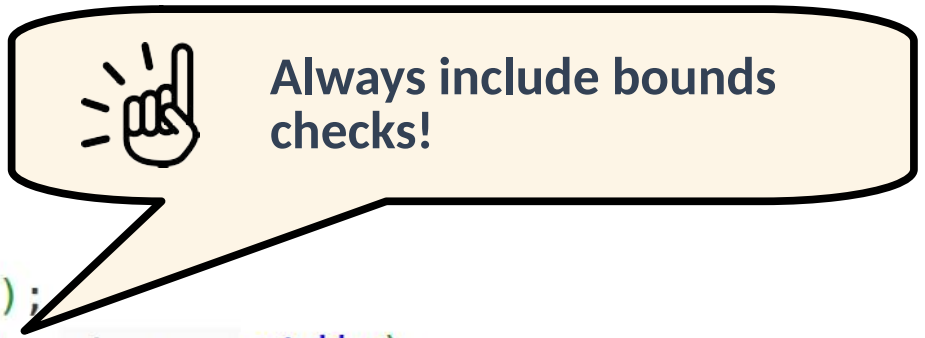
Buffer overflow: onder de abstractie zitten bytes

```c
#include <stdio.h>

int main(void)
{
    int isAdmin = 0;
    char name[8];

    printf(format: "Enter name: ");
    gets(name);

    if (isAdmin != 0) {
        printf(format: "hello %s: access granted!\n", name);
    } else {
        printf(format: "hello %s: access denied!\n", name);
    }
    return 0;
}
```

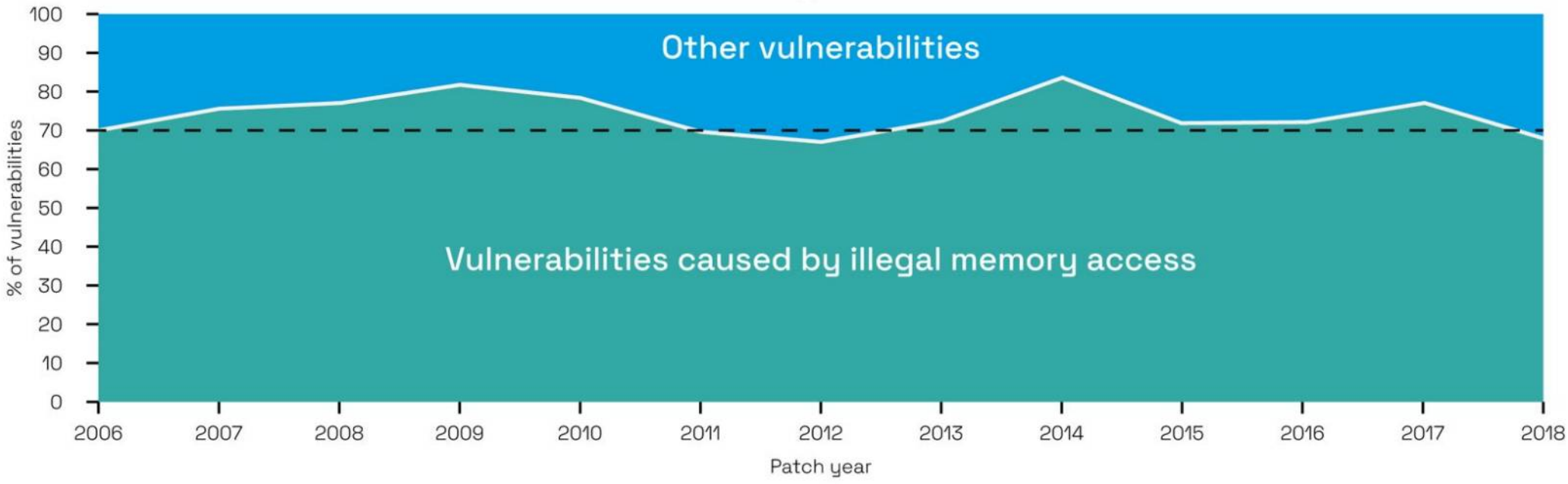


Buffer overflow: onder de abstractie zitten bytes

```c
#include <stdio.h>

int main(void)
{
    int isAdmin = 0;
    char name[8];

    printf(format: "Enter name: ");
    fgets(s: name, n: sizeof(name), stream: stdin);

    if (isAdmin != 0) {
        printf(format: "hello %s: access granted!\n", name);
    } else {
        printf(format: "hello %s: access denied!\n", name);
    }
    return 0;
}
```



Relevantie van buffer overflows “in the wild”



Source: "Trends, challenges and shifts in vulnerability mitigation", Matt Miller (MSRC), BlueHat IL 2019.

Overzicht: Van bit tot bug…

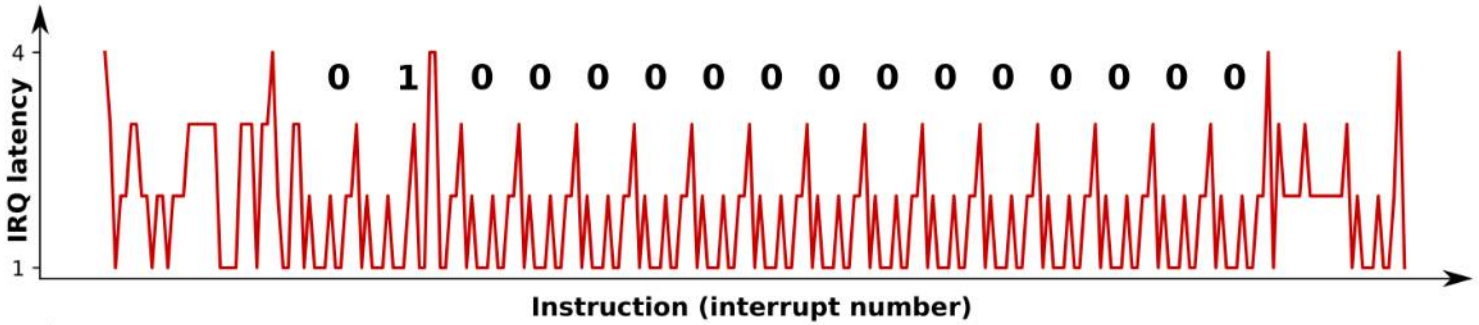
1. SQL injection attacks
(human → programmeertaal)

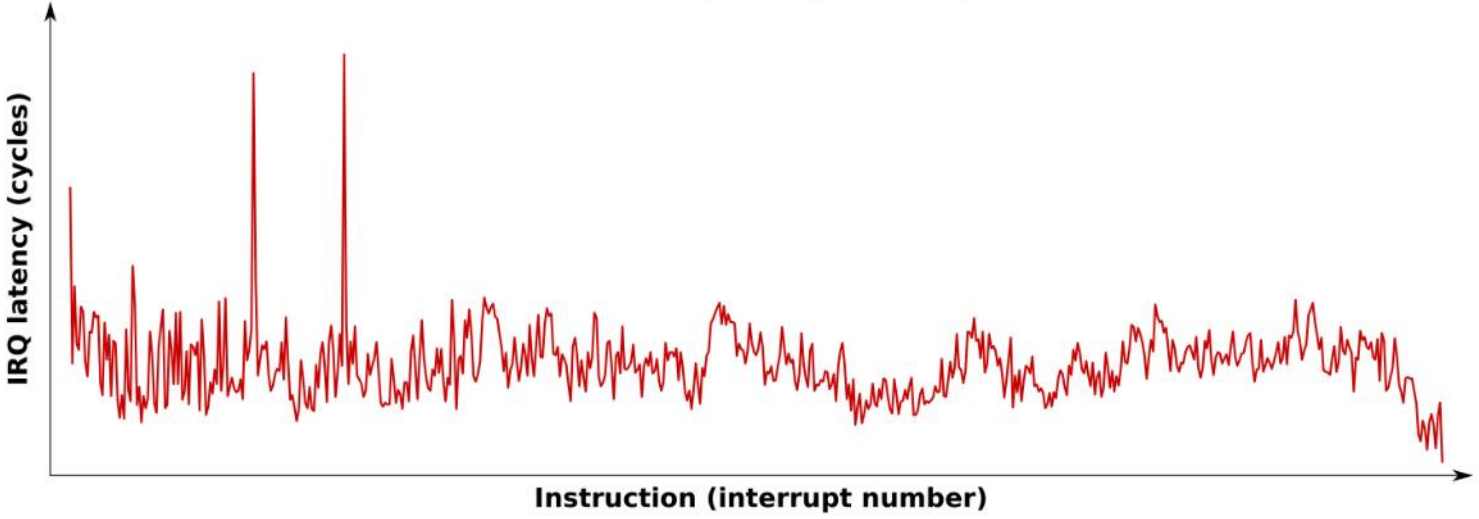
2. Memory-safety attacks
(C → machine code)

3. Microarchitectural attacks
(machine code → hardware)


VAULT DOOR
WEIGHT: 22 1/2 Tons
THICKNESS: 22 Inches
STEEL: 11 Layers of Special Cutting and Drill Resistant
LOCKS: 4 Hamilton Watch Movements for Time Locks

Microarchitectural Timing Leaks in Practice





TEXAS INSTRUMENTS

intel

Van Bulck et al. "Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic", CCS 2018

Example: CPU Cache Timing Side Channel

Cache principle: CPU speed $\gg$ DRAM $\rightarrow$ *cache code/data*

```
while true do
    maccess(&a);
endwh
```

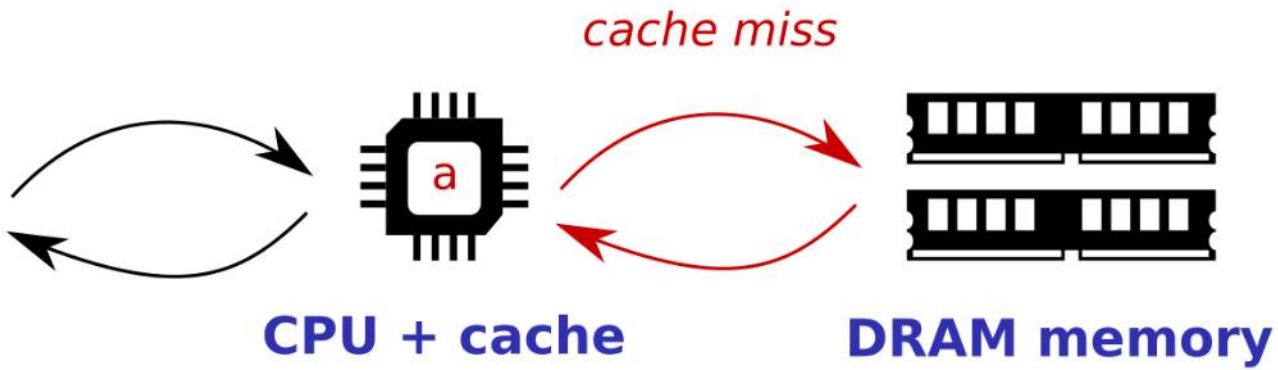
CPU + cache

DRAM memory

Example: CPU Cache Timing Side Channel

Cache miss: Request data from (slow) DRAM upon first use

```
while true do
    maccess(&a);
endwh
```



Example: CPU Cache Timing Side Channel

Cache hit: No DRAM access required for subsequent uses

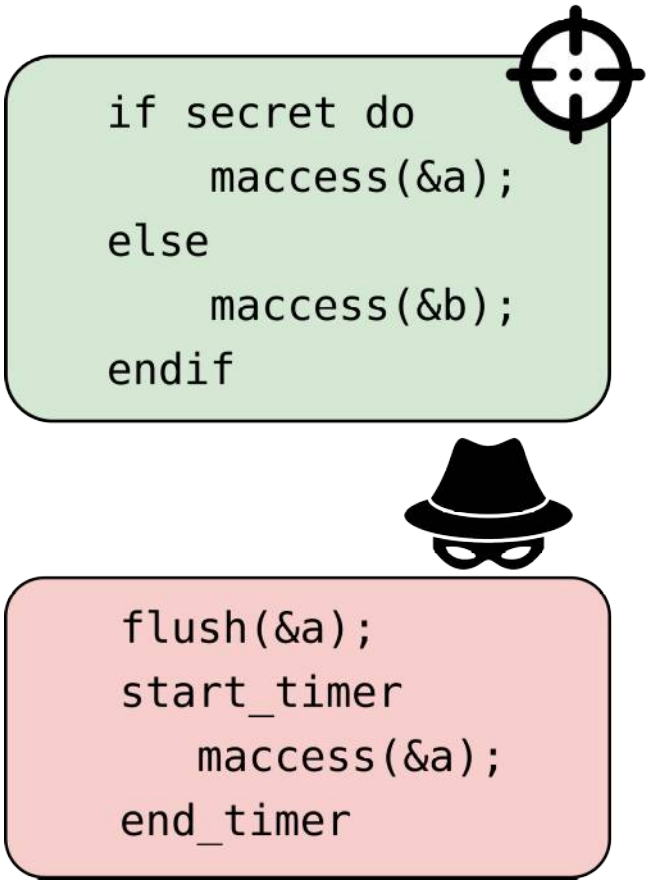
```
while true do
    maccess(&a);
endwh
```

cache hit

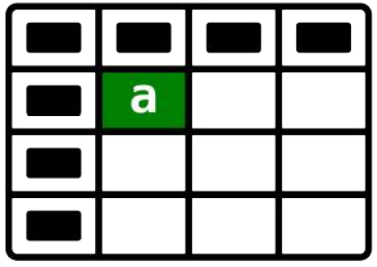
CPU + cache

DRAM memory

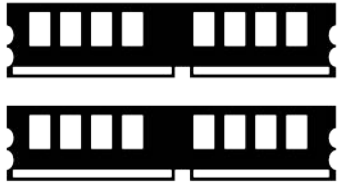
Cache Timing Attacks in Practice: Flush+Reload



'a' is accessible to attacker

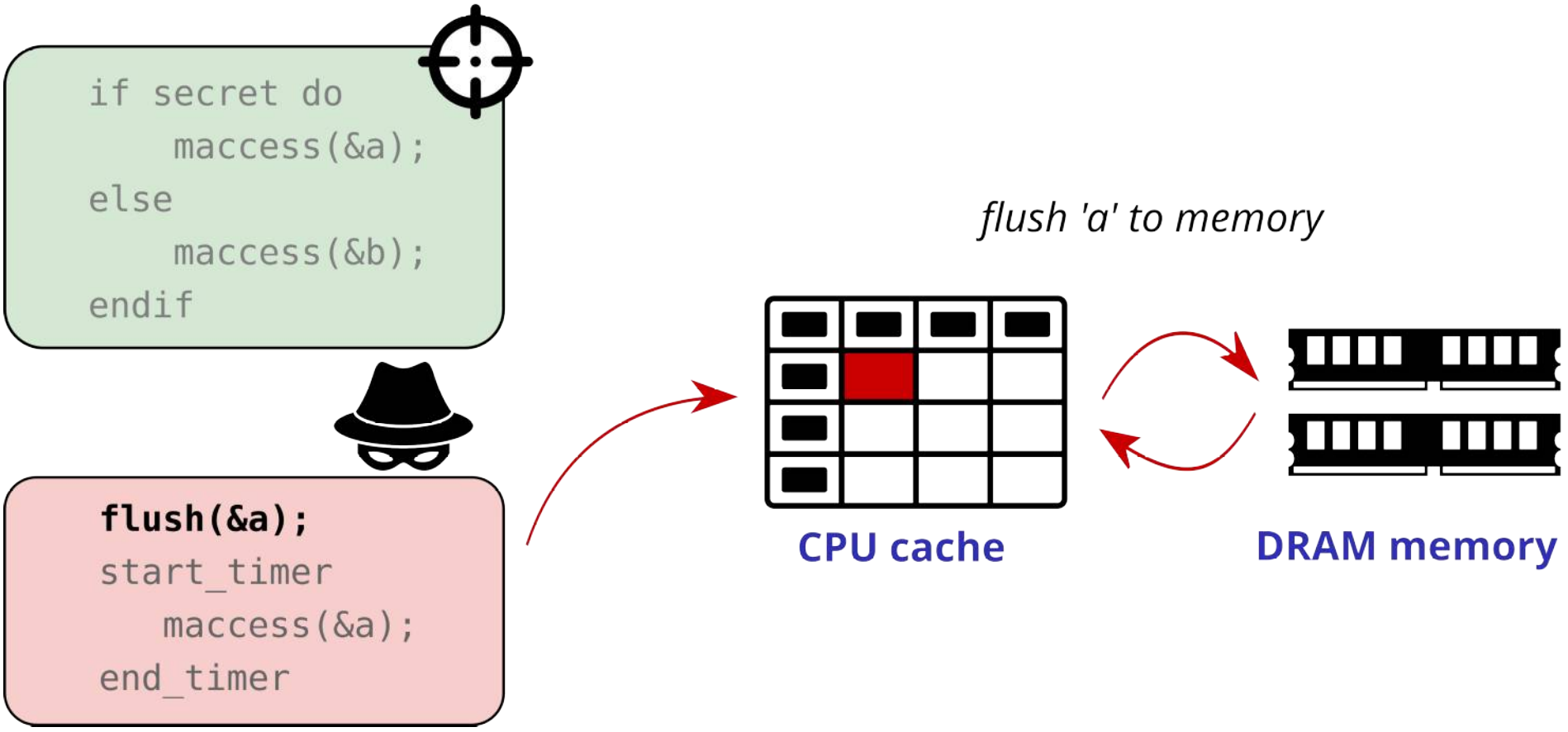


CPU cache

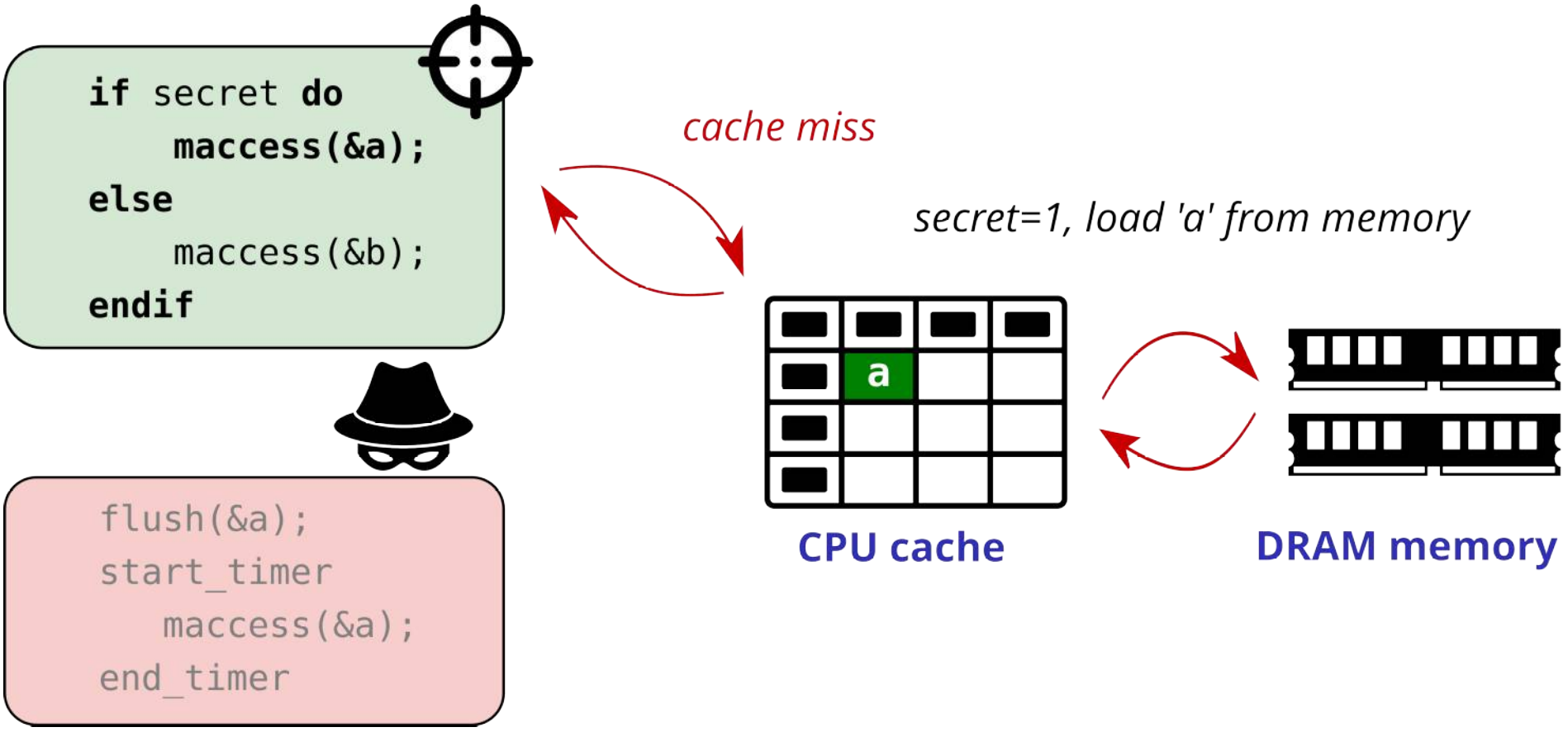


DRAM memory

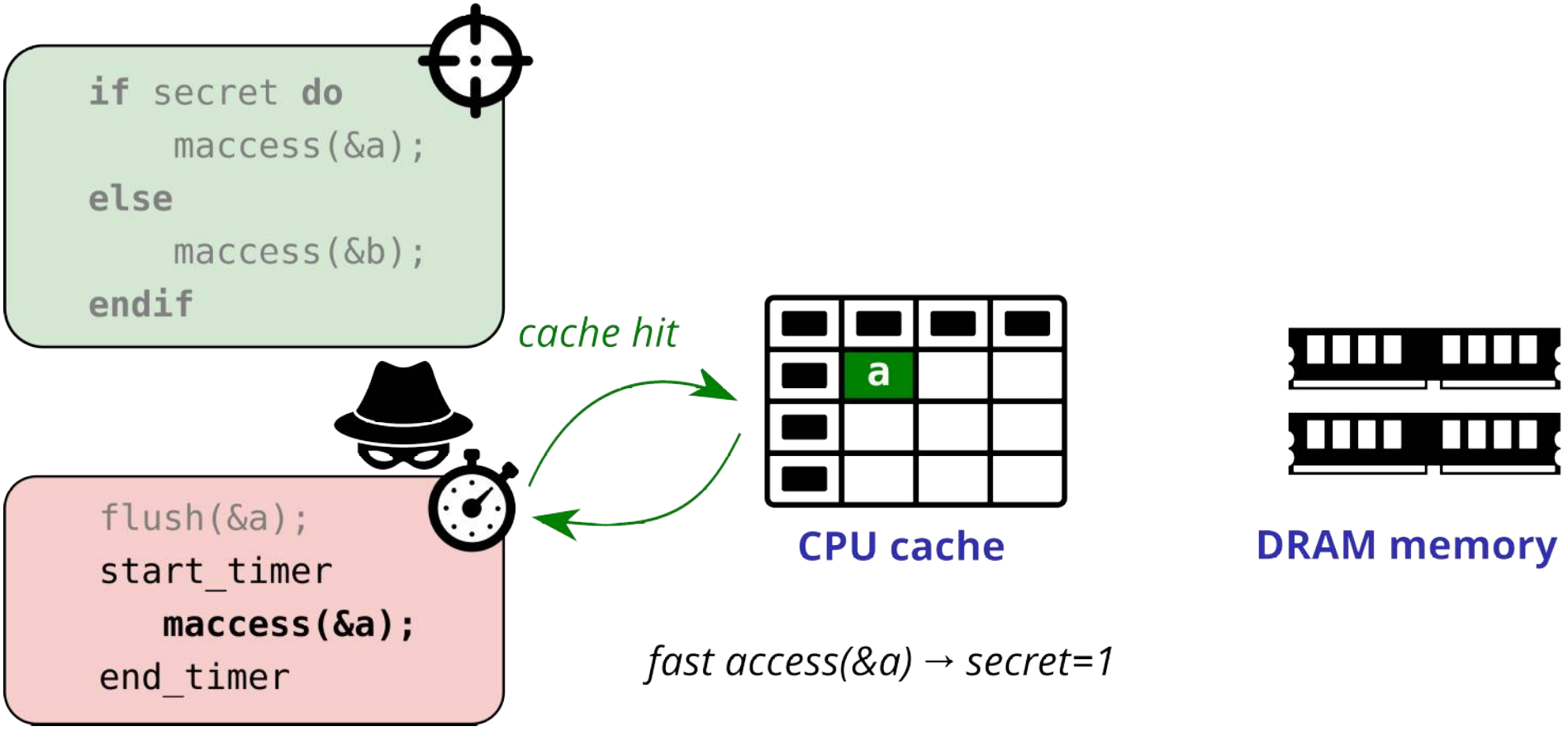
Cache Timing Attacks in Practice: Flush+Reload



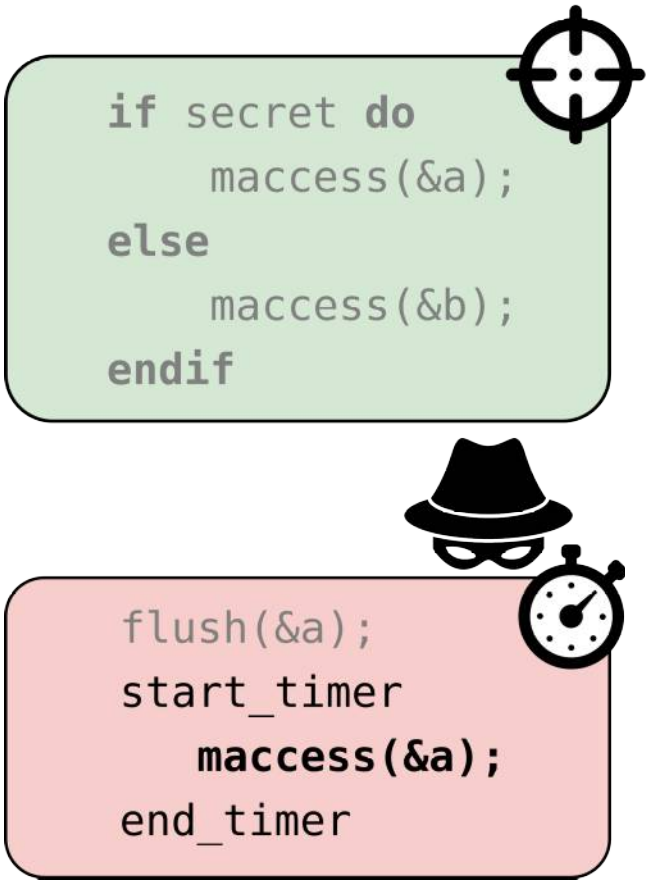
Cache Timing Attacks in Practice: Flush+Reload

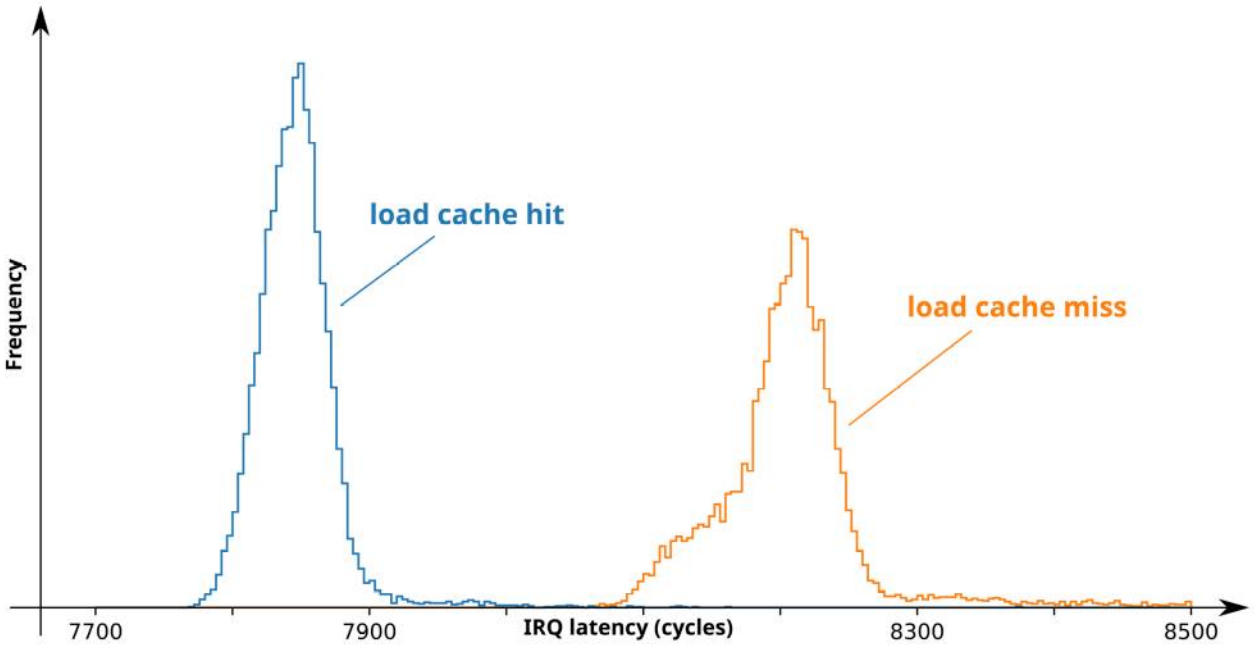


Cache Timing Attacks in Practice: Flush+Reload



Cache Timing Attacks in Practice: Flush+Reload





Demo: Spying on Keystrokes with Flush+Reload

```
25336620182821: Cache Hit (218 cycles) after a pause of 3 cycles
25336620297955: Cache Hit (216 cycles) after a pause of 43 cycles
25336620341217: Cache Hit (216 cycles) after a pause of 15 cycles
25336620363985: Cache Hit (212 cycles) after a pause of 4 cycles
25336620483903: Cache Hit (218 cycles) after a pause of 42 cycles
25336620499835: Cache Hit (216 cycles) after a pause of 3 cycles
25336620552419: Cache Hit (216 cycles) after a pause of 19 cycles
25336621476911: Cache Hit (218 cycles) after a pause of 300 cycles
25336974127733: Cache Hit (220 cycles) after a pause of 104704 cycles
25337739302241: Cache Hit (214 cycles) after a pause of 263629 cycles
25337739686069: Cache Hit (218 cycles) after a pause of 116 cycles
25337739773947: Cache Hit (218 cycles) after a pause of 27 cycles
25337739997613: Cache Hit (228 cycles) after a pause of 84 cycles
25338346337023: Cache Hit (228 cycles) after a pause of 211810 cycles
25338346617849: Cache Hit (224 cycles) after a pause of 81 cycles
25338346627851: Cache Hit (228 cycles) after a pause of 2 cycles
25338346634917: Cache Hit (228 cycles) after a pause of 1 cycles
25338346653587: Cache Hit (222 cycles) after a pause of 5 cycles
25338346811743: Cache Hit (220 cycles) after a pause of 58 cycles
25338346899541: Cache Hit (222 cycles) after a pause of 35 cycles
25338346911083: Cache Hit (222 cycles) after a pause of 3 cycles
25339081895869: Cache Hit (204 cycles) after a pause of 268339 cycles
25339081934737: Cache Hit (228 cycles) after a pause of 3 cycles
25339082052305: Cache Hit (226 cycles) after a pause of 34 cycles
25339082092569: Cache Hit (228 cycles) after a pause of 8 cycles
25339082116253: Cache Hit (224 cycles) after a pause of 3 cycles
25339082273651: Cache Hit (202 cycles) after a pause of 53 cycles
25339815487639: Cache Hit (226 cycles) after a pause of 232157 cycles
```

https://github.com/isec-tugraz/cache_template_attacks

```
3 super secret keystroke timings
4
5 F
```

Plain Text Tab Width: 8 Ln 5, Col 2 INS

Van bit tot bug: wat onthoud je?

1. Kracht van abstracties

→ Elke laag verbergt de complexiteit van de lagen eronder

2. Maar geen enkele abstractie is perfect

→ Aanvallers breken door abstractielagen

3. Wie de lagen kent, ziet de bugs

→ Belang van de *top-down* en *bottom-up* exploratie van de systeemstack in de komende jaren :-)

Happy hacking!

Vragen of nieuwsgierig? jo.vanbulck@kuleuven.be