Software Testing and Reverse Engineering
CS4110

Sicco Verwer, Andy Zaidman, Annibale Panichella,
Gaetano Pellegrino, Mark Janssen, Arnd Hartmanns,
Arie van Deursen

TU Delft

By helping us improve the 3TU Cyber Crime Science course, you participate in a raffle to win an **Ipod Shuffle**!

How?

1. By filling in a short questionnaire now and in 5 months
2. Make sure you write down your '**magic word**' so that you can participate in the raffle and claim the prize if you win.

https://docs.google.com/forms/d/1yaAM1nI2nassZZKmQ6ZP_Pd0wgpTIKfzPwoUa_KwKVo/viewform?usp=send_form

TU Delft

Why?

- Software is one of the most **complex** artifacts of mankind
- Errors are easily made and hard to find
- In this course, we study **automated methods** to help find these errors
- Background:
  - Software Engineering
  - Artificial Intelligence
  - Machine Learning
  - Many Smart Tricks…

Setting

- You are given a piece of software, does it work correctly?
- 2 subproblems:
  - What does it do?
    - Reverse engineering
  - What should it do?
    - Testing

Exercise: spot the bugs

```
int balance;

void decrease(int amount)
{
    if (balance <= amount)
    {
        balance = balance - amount;
    }
    else
    {
        printf("Insufficient funds\n");
    }
}

void increase(int amount)
{
    balance = balance + amount;
}
```

Exercise: spot the bugs

```
int balance;

void decrease(int amount)
{
    if (balance <= amount)
    {
        balance = balance - amount;
    }
    else
    {
        printf("Insufficient funds\n");
    }
}

void increase(int amount)
{
    balance = balance + amount;
}
```

should be >=

Exercise: spot the bugs

```
int balance;

void decrease(int amount)
{
    if (balance <= amount)
    {
        balance = balance - amount;
    }
    else
    {
        printf("Insufficient funds\n");
    }
}

void increase(int amount)
{
    balance = balance + amount;
}
```

should be >=

what if amount is negative?

Exercise: spot the bugs

```
int balance;

void decrease(int amount)
{
    if (balance <= amount)
    {
        balance = balance - amount;
    }
    else
    {
        printf("Insufficient funds\n");
    }
}

void increase(int amount)
{
    balance = balance + amount;
}
```

should be >=

what if amount is negative?

what if sum is too large for int?

Exercise: spot the bugs

```
int balance;

void decrease(int amount)
{
    if (balance <= amount)
    {
        balance = balance - amount;
    }
    else
    {
        printf("Insufficient funds\n");
    }
}

void increase(int amount)
{
    balance = balance + amount;
}
```

should be >=

what if amount is negative?

what if sum is too large for int?

How to do this for thousands of lines of code….

Different settings: code and tests

```
int balance;

void decrease(int amount)
{
    if (balance <= amount)
    {
        balance = balance – amount;
    }
    else
    {
        printf("Insufficient funds\n");
    }
}

void increase(int amount)
{
    balance = balance + amount;
}
…
balance = 10; decrease(5);
assert(balance = 5);
increase(5);
assert(balance = 10);
…
```

Different settings: code and tests

```
int balance;

void decrease(int amount)
{
    if (balance <= amount)
    {
        balance = balance - amount;
    }
    else
    {
        printf("Insufficient funds\n");
    }
}

void increase(int amount)
{
    balance = balance + amount;
}
…
balance = 10; decrease(5);
assert(balance = 5);
increase(5);
assert(balance = 10);
…
```

Typical question:

Are the tests sufficient?

Different settings: only code

```
int balance;

void decrease(int amount)
{
    if (balance <= amount)
    {
        balance = balance - amount;
    }
    else
    {
        printf("Insufficient funds\n");
    }
}

void increase(int amount)
{
    balance = balance + amount;
}
```

Different settings: only code

```
int balance;

void decrease(int amount)
{
    if (balance <= amount)
    {
        balance = balance - amount;
    }
    else
    {
        printf("Insufficient funds\n");
    }
}

void increase(int amount)
{
    balance = balance + amount;
}
```

Typical question:

What are good tests?

Different settings: obfuscated code

…

```
if(((((input.equals(inputs[2]) && (((a305 == 9) &&
(((a14.equals("f")) && cf) && a94 <=  23)) && (a185.equals("e"))))
&& a277 <=  199) && ((a371 == a298[0]) && (((a382 && (a287 ==
a215[0])) && (a115.equals("g"))) && a396))) && a47 >= 37)) {
  cf = false;
  a170 = a1;
  a185 = "f";
  a100 = (((((a94 * a94)%14999)%14901) + -15097) / 5) + -2185);
        System.out.println("X");
 }
```

…

Different settings: obfuscated code

```
…
if(((((input.equals(inputs[2]) && (((a305 == 9) &&
(((a14.equals("f")) && cf) && a94 <=  23)) && (a185.equals("e"))))
&& a277 <=  199) && ((a371 == a298[0]) && (((a382 && (a287 ==
a215[0])) && (a115.equals("g"))) && a396))) && a47 >= 37)) {
  cf = false;
  a170 = a1;
  a185 = "f";
  a100 = (((((a94 * a94)%14999)%14901) + -15097) / 5) + -2185);
        System.out.println("X");
 }
…
```

Typical question:

What does it do?

Different settings: binary executable

```
…
push    ebp
mov     ebp, esp
sub     esp, 18h
mov     [ebp-8], ebx
mov     [ebp-4], esi
mov     ebx, [ebp-8]
mov     esi, [ebp-4]
mov     esp, ebp
pop     ebp
retn
…
```

Different settings: binary executable

```
…
push    ebp
mov     ebp, esp
sub     esp, 18h
mov     [ebp-8], ebx
mov     [ebp-4], esi
mov     ebx, [ebp-8]
mov     esi, [ebp-4]
mov     esp, ebp
pop     ebp
retn
…
```

Typical question:

Can it be broken?

What will you learn

- What is testing and reversing research?
- State-of-the-art software testing and reversing tools
- Apply these tools to real software:
  - Own projects
  - Open source software
  - Communication protocols
  - CrackMe and/or Malware

What will you do

- Team up with two fellow students
- Read papers on a chosen topic
- Write a summary of these and at least 3 self-chosen papers
- Present the topic
  - Answer questions
  - Ask questions on at least one other topic
- Have fun in the lab
  - Apply techniques from your topic to real software
- Report and present the results

Program

Week	Lecture, Lecture hall Chip		Lab, Dijkstrazaal, 9th	
1	9 Feb	Today		
2	16 Feb	Automated reversing		
3	23 Feb	Test case generation		
4	1 Mar	Paper presentations		
5	8 Mar	Paper presentations		
6	15 Mar	Binary Analysis	18 Mar	Assistance
7	22 Mar	Modern software testing	25 Mar	Assistance
8	29 Mar	TBA	1 Apr	Assistance
9	5 Apr	Final presentations	2 Apr	Finalizing

Lectures on Tuesday, 13:45 till 15:45
Lab sessions, start March 18th, Friday, 10:45 till 12:45

Lab connection to Twente via Skype/Double

Grading

- Lab: 50%
  - Ability
  - Independence
  - Results
- Report: 30%
  - Easy to understand explanation of complex techniques
  - Detailed presentation of results
  - Justified conclusions
- Presentation: 20%
  - Capture content in slides
  - How to do science in reversing and testing
  - Answering/asking questions

NO EXAM!

Collaboration

- Git:
  - https://github.com/TUDelft-CS4110
- Slack:
  - https://cs4110.slack.com/
- No use of Blackboard!

Topics

Tools for automated testing

1. Mutation analysis
2. Test case generation

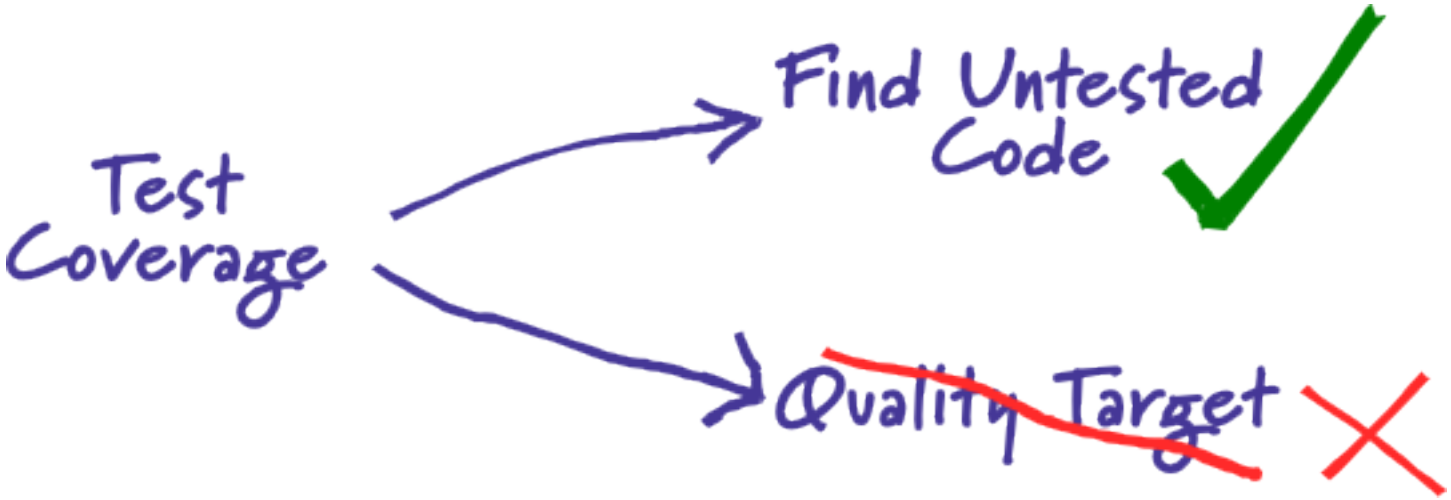
and automated reverse engineering

1. Fuzzing
2. Concolic testing
3. State machine learning
4. Binary analysis

Twente

TU Delft

Mutation testing



- Production code can be covered, yet the tests covering it might still miss a bug (i.e., the tests are not of sufficient quality)
- Is there another way of looking into the quality of tests?

Mutation testing by example

Original

```
if( i >= 0 ) {
    return "foo";
} else {
    return "bar";
}
```

Test

Code is transformed, mutant introduced

Tests remain identical

Mutant

```
if( i < 0 ) {
    return "foo";
} else {
    return "bar";
}
```

Test

Scenario 1

→ Mutant alive

Scenario 2

→ Mutant killed

TU Delft

Some important questions…

What you will learn from reading the papers

- What is mutation testing, what can it do for you?
- Are these mutants representative of actual bugs?

What you will learn from working with Pitest

- Take one of your former well-tested student projects and apply mutation testing to it. What do you find?
- Take 2/3 open source projects of your choice, determine its test coverage and apply Pitest to it

Automated Test Case Generation

Search-Based Software Testing

TU Delft

Traditional Testing

manual design of test cases / scenarios

Class Under Test

```
class Triangle {

 public Triangle(int a, int b, int c) {…}

 public void computeTriangleType() {
    if (a == b) {
        if (b == c)
            type = "EQUILATERAL";
        else
            type = "ISOSCELES";
    } else if (a == c) {
        type = "ISOSCELES";
    } else {
    if (b == c)
        type = "SCALENE";
    else
        checkRightAngle();
     }
    System.out.println(type);
  }
}
```

Traditional Testing

manual design of test cases / scenarios

Class Under Test

```
class Triangle {

 public Triangle(int a, int b, int c) {…}

 public void computeTriangleType() {
    if (a == b) {
        if (b == c)
            type = "EQUILATERAL";
        else
            type = "ISOSCELES";
    } else if (a == c) {
        type = "ISOSCELES";
    } else {
    if (b == c)
        type = "SCALENE";
    else
        checkRightAngle();
     }
    System.out.println(type);
  }
}
```

```
@Test
  public void test(){
    Triangle t = new Triangle(1,2,3)
    t.isTriangle();
    t.computeTriangleType();
  }
}
```

Traditional Testing

manual design of test cases / scenarios

Class Under Test

```
class Triangle {

 public Triangle(int a, int b, int c) {…}

 public void computeTriangleType() {
    if (a == b) {
        if (b == c)
            type = "EQUILATERAL";
        else
            type = "ISOSCELES";
    } else if (a == c) {
        type = "ISOSCELES";
    } else {
    if (b == c)
        type = "SCALENE";
    else
        checkRightAngle();
     }
    System.out.println(type);
  }
}
```

```
@Test
  public void test(){
    Triangle t = new Triangle(1,2,3)
    t.isTriangle();
    t.computeTriangleType();
  }
}
```

```
@Test
  public void test(){
    Triangle t = new Triangle(1,1,1)
    t.isTriangle();
    t.computeTriangleType();
  }
}
```

Traditional Testing

manual design of test cases / scenarios

Class Under Test

```
class Triangle {

 public Triangle(int a, int b, int c) {…}

 public void computeTriangleType() {
    if (a == b) {
        if (b == c)
            type = "EQUILATERAL";
        else
            type = "ISOSCELES";
    } else if (a == c) {
        type = "ISOSCELES";
    } else {
    if (b == c)
        type = "SCALENE";
    else
        checkRightAngle();
     }
    System.out.println(type);
  }
}
```

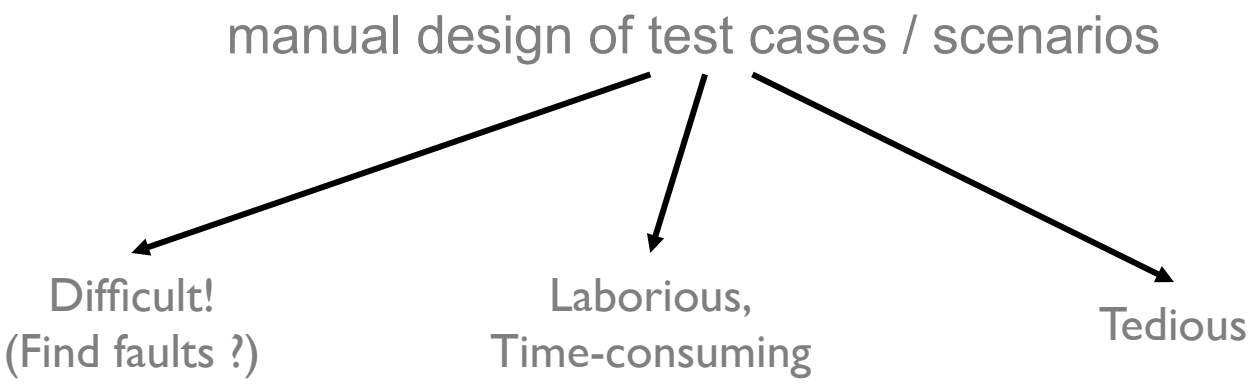
```
@Test
  public void test(){
    Triangle t = new Triangle(1,2,3)
    t.isTriangle();
    t.computeTriangleType();
  }
}
```

```
@Test
  public void test(){
    Triangle t = new Triangle(1,1,1)
    t.isTriangle();
    t.computeTriangleType();
  }
}
```

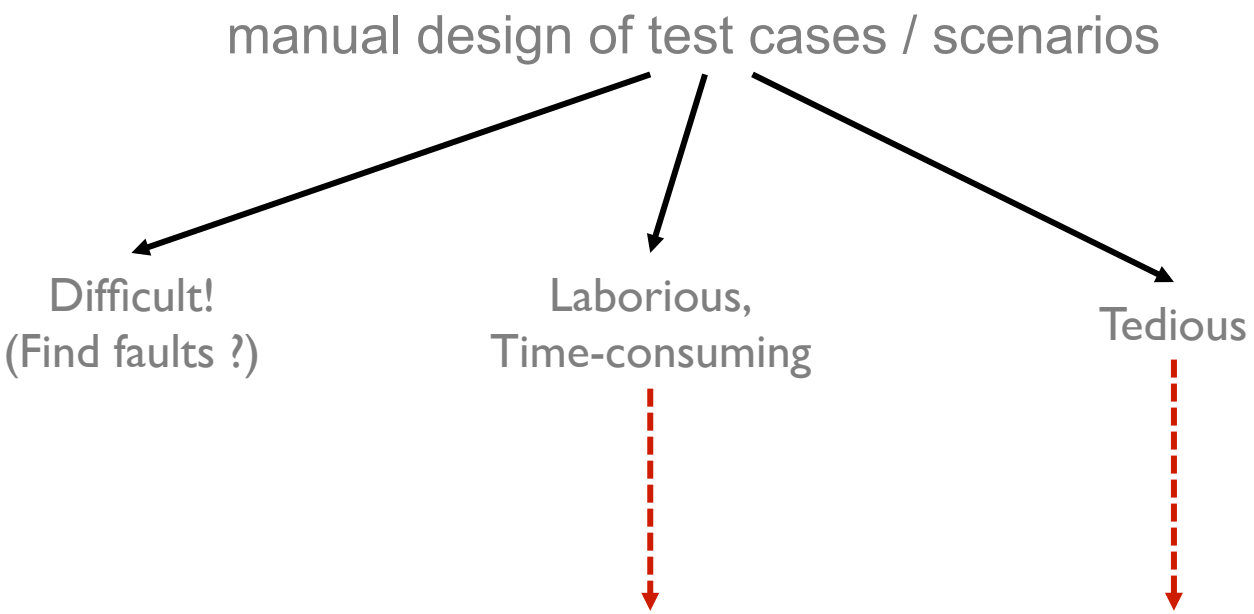
```
@Test
  public void test(){
    Triangle t = new Triangle(1,1,1)
    t.isTriangle();
    t.computeTriangleType();
  }
}
```

TUDelft

Traditional Testing

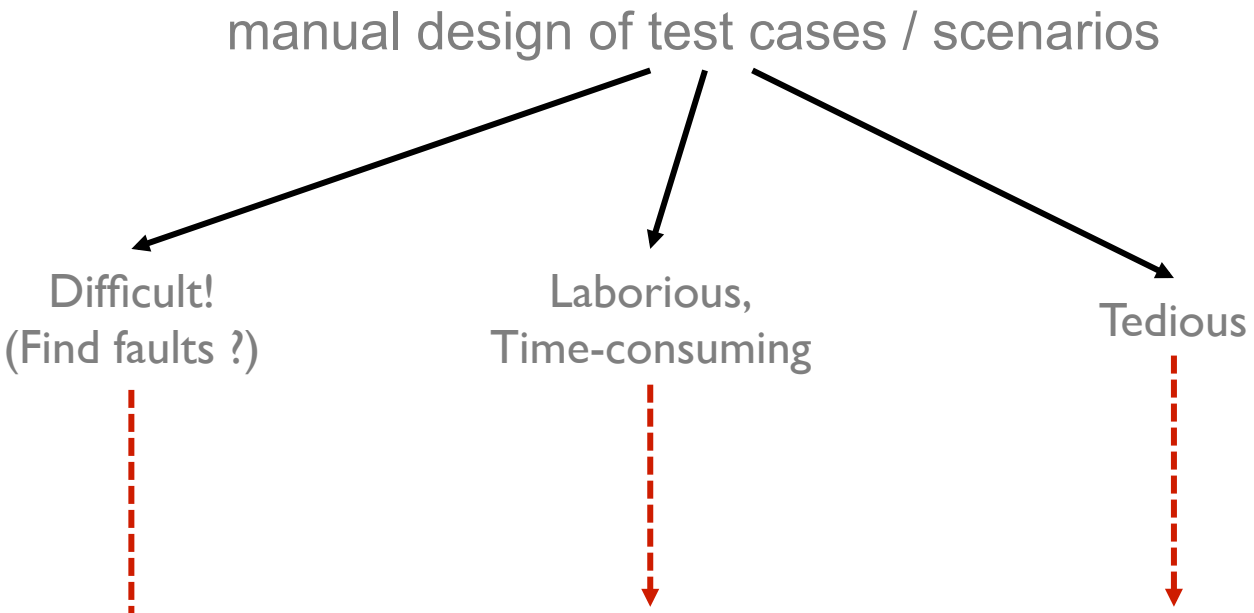


Traditional Testing



Search-Based Software Testing:
Automatic Generation of Test Cases
"It is not a human's time being consumed"

Traditional Testing



Search-Based Software Testing:
Automatic Generation of Test Cases
"It is not a human's time being consumed"

Search-Based Software Testing:
Using good fitness function to
reach/expose the faults

Evolutionary Testing

Evolutionary Testing

```
@Test
  public void test(){


  }
}
```

Evolving Tests

Evolution in Nature

Time

Evolutionary Testing

```
@Test
  public void test(){
  }
}
```

```
@Test
  public void test(){
    Triangle t = new Triangle(   ,   ,   )
  }
}
```

Evolving Tests

Evolution in Nature

Time

Evolutionary Testing

```
@Test
  public void test(){

  }
}
```

```
@Test
  public void test(){
    Tri
  }
}
```

```
@Test
  public void test(){
    Triangle t = new Triangle(  ,  ,  )
    t.isTriangle();

  }
}
```

Evolving Tests

Evolution in Nature

Time

Evolutionary Testing

```
@Test
  public void test(){
  }
}
```

```
@Test
  public void test(){
    Tri
  }
}
```

```
@Test
  public void test(){
    Triangle t = new Triangle(
    t.i
  }
}
```

```
@Test
  public void test(){
    Triangle t = new Triangle(   ,   ,   )
    t.isTriangle();
    t.computeTriangleType();
  }
}
```

Evolving Tests

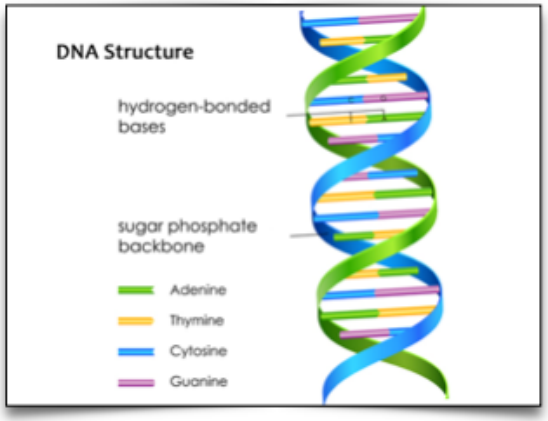
Evolution in Nature

Time

TU Delft

Evolutionary Testing

```
@Test
public void test(){
  Statement 1
  Statement 2
  Statement 3
  . . .
  Assertion 1
  Assertion 2
  . . .
}
```



Basic Elements

```
Statement 1
Statement 2
Statement 3
```

Basic

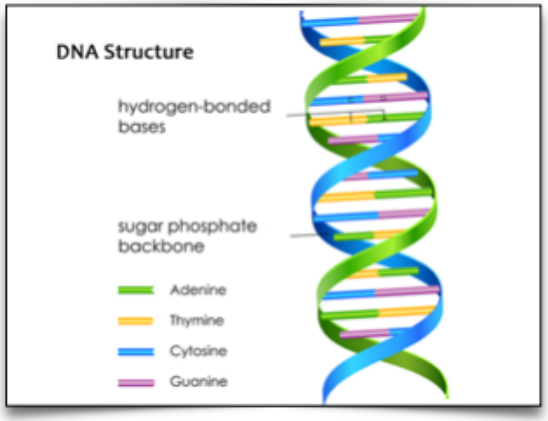
- Adenine
- Thymine
- Cytosine
- Guanine

Evolutionary Testing

Recombination

```
@Test
public void test(){
  Statement 1
  Statement 2
  Statement 3
  . . .
  Assertion 1
  Assertion 2
  . . .
}
```

Recombination




```
@Test
public void test1(){
  Statement 1
  Statement 2
  Statement 3
}
```

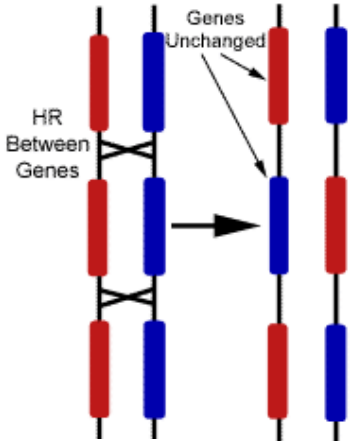
```
@Test
public void test2(){
  Statement 4
  Statement 5
  Statement 6
}
```

Parental Tests

```
@Test
public void test1(){
  Statement 1
  Statement 5
  Statement 3
}
```

```
@Test
public void test2(){
  Statement 1
  Statement 2
  Statement 3
}
```

Recombined Tests



Parental DNA

Recombined DNA

TU Delft

What will you learn?

From reading the papers:

1) What is evolutionary testing?
2) What are "fittest" tests?
3) Coverage?

From working with EvoSuite:

1) Take one of your former well-tested student projects and generate tests.
2) What are the differences between manual/automated tests in terms of coverage?
3) What are the differences between manual/automated tests in terms of "readability"?

Fuzzing

Security/penetration testing

- Normal testing investigates correct behavior for sensible inputs, and inputs on borderline conditions
- Security testing involves looking for the incorrect behavior for really silly inputs
- Try to crash the system!
  - and discover why it crashed!
- In general, this is very hard

Fuzzing

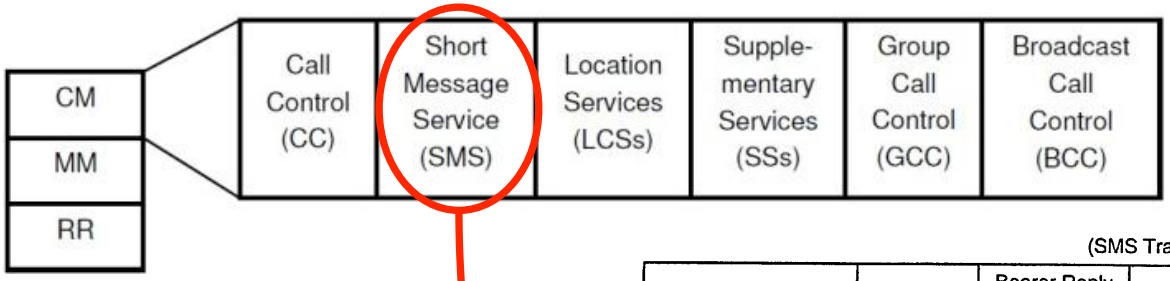
- Test different inputs at random, until the system crashes
- What is the probability of reaching line 11 with random input?

```
 1:int parse(FILE *fp) {
 2:   char cmd[256], *url, buf[5];
 3:   fread(cmd, 1, 256, fp);
 4:   int i, header_ok = 0;
 5:   if (cmd[0] == 'G')
 6:     if (cmd[1] == 'E')
 7:       if (cmd[2] == 'T')
 8:         if (cmd[3] == ' ')
 9:           header_ok = 1;
10:   if (!header_ok) return -1;
11:   url = cmd + 4;
12:   i=0;
13:   while (i<5 && url[i]!='\0' && url[i]!='\n') {
14:     buf[i] = tolower(url[i]);
15:     i++;
16:   }
17:   buf[i] = '\0';
18:   printf("Location is %s\n", buf);
18:   return 0; }
```

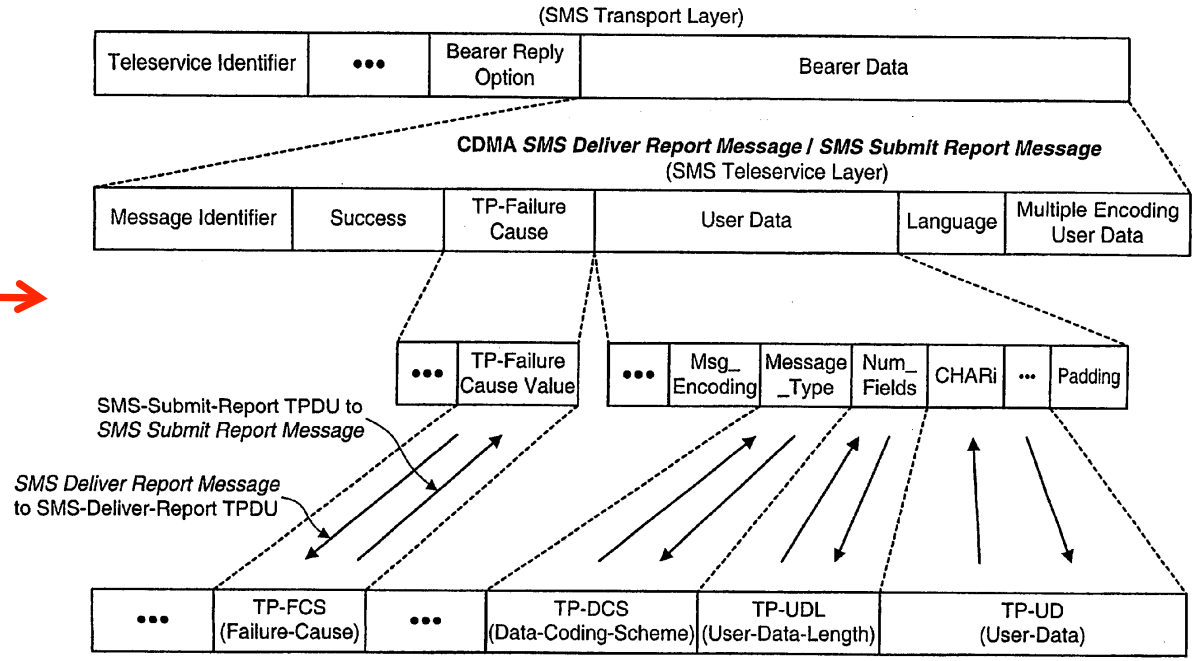
Structured input

- When input has to start with eg. ‘http’, testing all possible strings that start differently is a waste of time
- Fortunately, we often know:
  - Example input files or strings
  - Protocol specifications, or test implementations
- Solutions:
  - Generate random permutations from example files
    - Mutation-based fuzzing
  - Fuzz only values but keep in line with the specification
    - Protocol (generative) fuzzing

Example : GSM protocol fuzzing



GSM is an extremely rich & complicated protocol



Example and slides from Erik Poll

Example : GSM protocol fuzzing

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

eg possibility to send faxes (!?)

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

Example : GSM protocol fuzzing

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

Fuzzing is a lot of fun!

eg possibility to send faxes (!?)

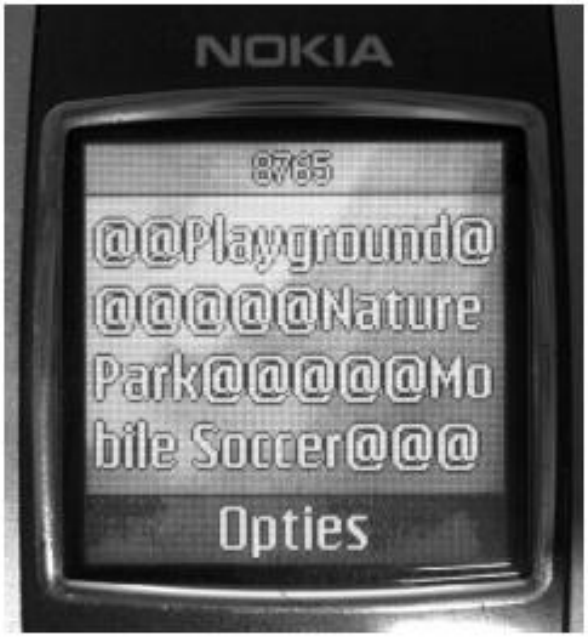
Only way to get rid if this icon: reboot the phone

Example : GSM protocol fuzzing

- More serious: malformed SMS text messages display raw memory content, rather than a text message

(a) Showing garbage

(b) Showing the name of a wallpaper and two games

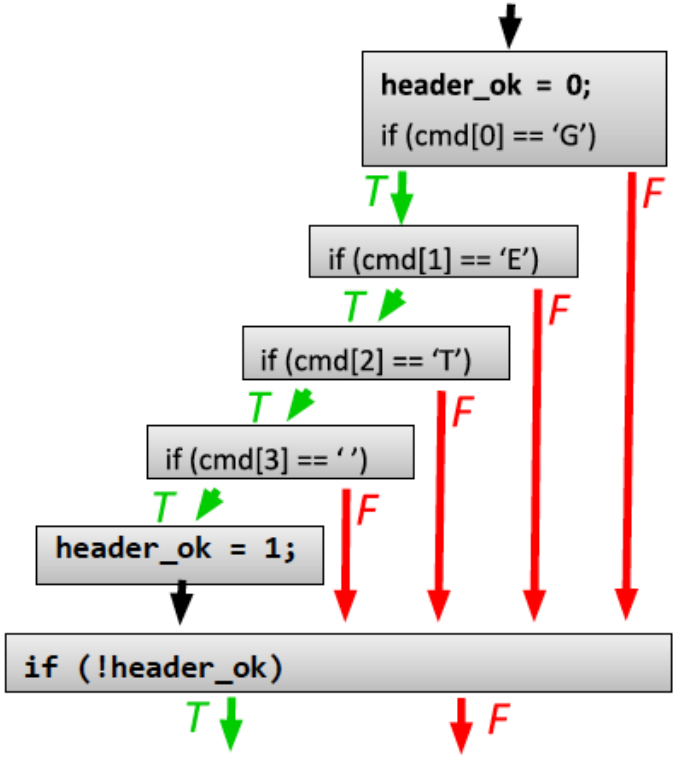


Concolic testing

concrete and symbolic testing

Smarter fuzzing: use system code!

```
 1:int parse(FILE *fp) {
 2:   char cmd[256], *url, buf[5];
 3:   fread(cmd, 1, 256, fp);
 4:   int i, header_ok = 0;
 5:   if (cmd[0] == 'G')
 6:     if (cmd[1] == 'E')
 7:       if (cmd[2] == 'T')
 8:         if (cmd[3] == ' ')
 9:           header_ok = 1;
10:   if (!header_ok) return -1;
11:   url = cmd + 4;
12:   i=0;
13:   while (i<5 && url[i]!='\0' && url[i]!='\n') {
14:     buf[i] = tolower(url[i]);
15:     i++;
16:   }
17:   buf[i] = '\0';
18:   printf("Location is %s\n", buf);
18:   return 0; }
```



- Can we automatically generate interesting input values?

Path exploration

- Try to assignments to all values in cmd that make the program reach line 11:
  - Represent all values as symbolic variables
  - Write down a formula describing all paths through the program that reach line 11

SPECIFY INPUT as symbolic variable:

cmd:	cmd0	cmd1	cmd2	cmd3	cmd4	cmd5	cmd6	cmd7	cmd8	cmd9
example:	'G'	'E'	'T'	' '	'h'	't'	't'	'p'	':'	'/'

(we're considering input of length 10 just for this example)

Path exploration

SPECIFY INPUT:

cmd: | cmd0 | cmd1 | cmd2 | cmd3 | cmd4 | cmd5 | cmd6 | cmd7 | cmd8 | cmd9 |

(we're considering input of length 10 just for this example)

SPECIFY PATH CONSTRAINTS:

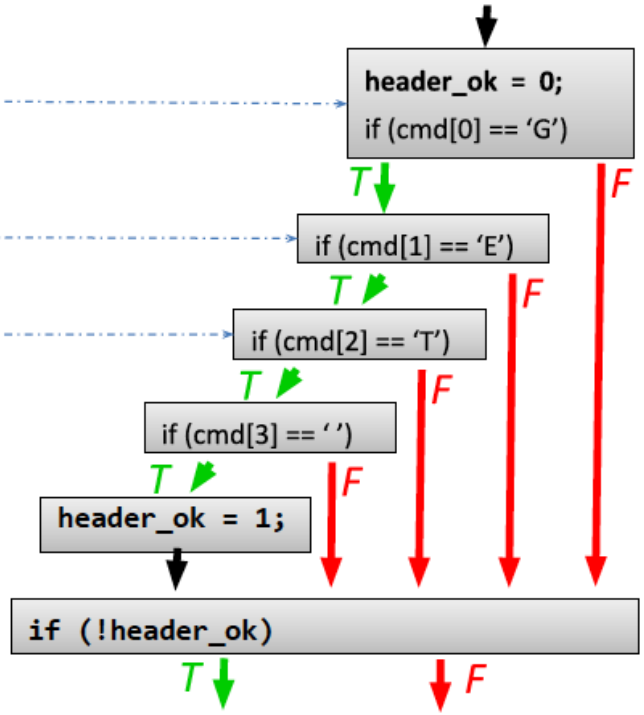
(cmd0 == 'G')

(cmd1 == 'E')

(cmd2 == 'T')

FINAL FORMULA:

(cmd0 == 'G') & (cmd1 == 'E') & (cmd2 == 'T') & (cmd3 == ' ')



Symbolic execution

- Represent all inputs as symbolic values and perform operations symbolically
  - cmd0, cmd1, …
- Path predicate: is there a value for command such that
  (cmd0 == ‘G’) & (cmd1 == ‘E’) & (cmd2 == ‘T’) & (cmd3 == ‘ ’) ?
- Provide all constraints to a **combinatorial solver**, eg. Z3
  - Answer: YES, with cmd0 = ‘G’, cmd1 = ‘E’, …, cmd9 = x
- *Only fuzz inputs that satisfy the provided answer!*

Symbolic execution, example

```
m(int x,y){
    x = x + y;
    y = y - x;
    if (2*y > 8) { ....
                 }
    else if (3*x < 10){ ...
                       }
    }
```

Write down the path predicate needed to reach this line

Symbolic execution, example

```
m(int x,y){
    x = x + y;
    y = y - x;
    if (2*y > 8) {…
                 }
    else if (3*x < 10){…
                      }
    }
```

```
// let x == N and y == M
// x becomes N+M
// y becomes M-(N+M) == -N
// taken if 2*-N > 8, ie N < -4
// taken if N>=-4 and 3(M+N)<10
```

So, (N>=-4) & 3(M+N)<10

Some important questions…

What you will learn from reading the papers

- What are fuzzing and concolic testing methods, what bugs can you find using them?
- When is it beneficial to test concolicly?

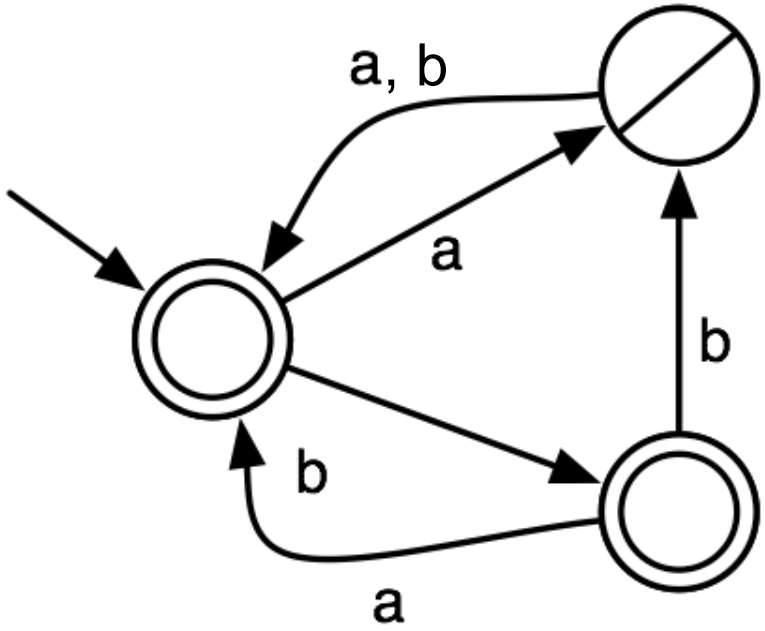
What you will learn from working with AFL/Klee/Angr:

- Take one of your former student projects, fuzz it and apply concolic testing. Can you make it crash? What are the bugs you find?

- Test executable with known bugs, e.g. ImageMagick:
  - https://www.cvedetails.com/vulnerability-list/vendor_id-1749/Imagemagick.html
- What are the bugs you find? Compare with CVE.

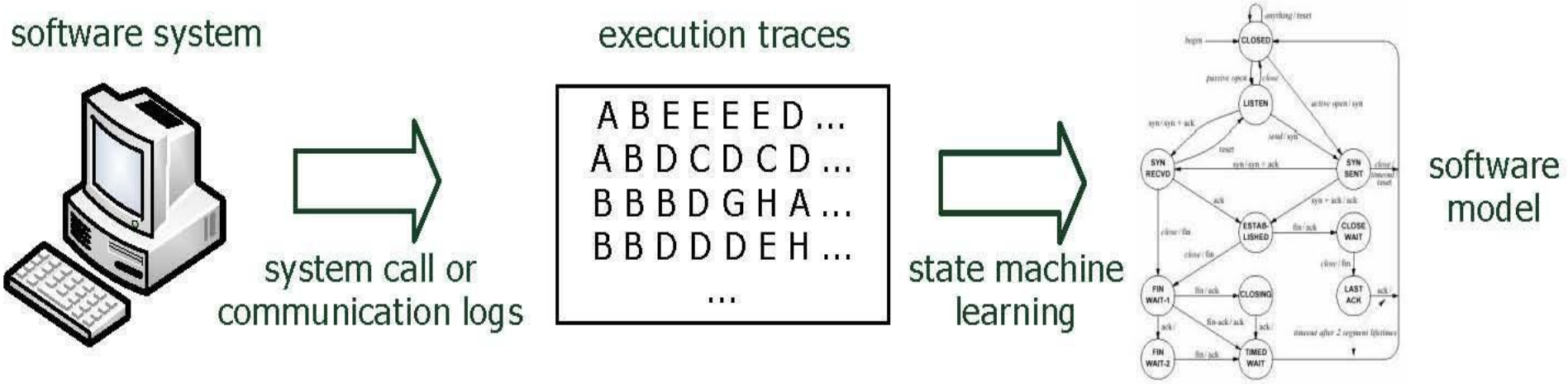
State machine learning

Deterministic Finite State Automata



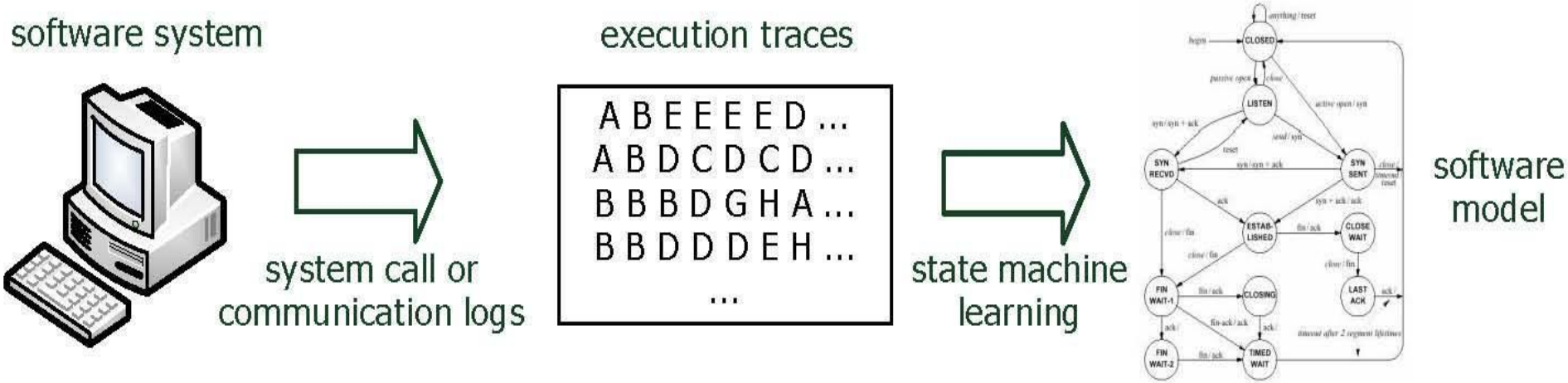
(a,-),(ab,+),(aa,+),(b,+),(bb,-),(bab,+)

State machine learning



Passive learning

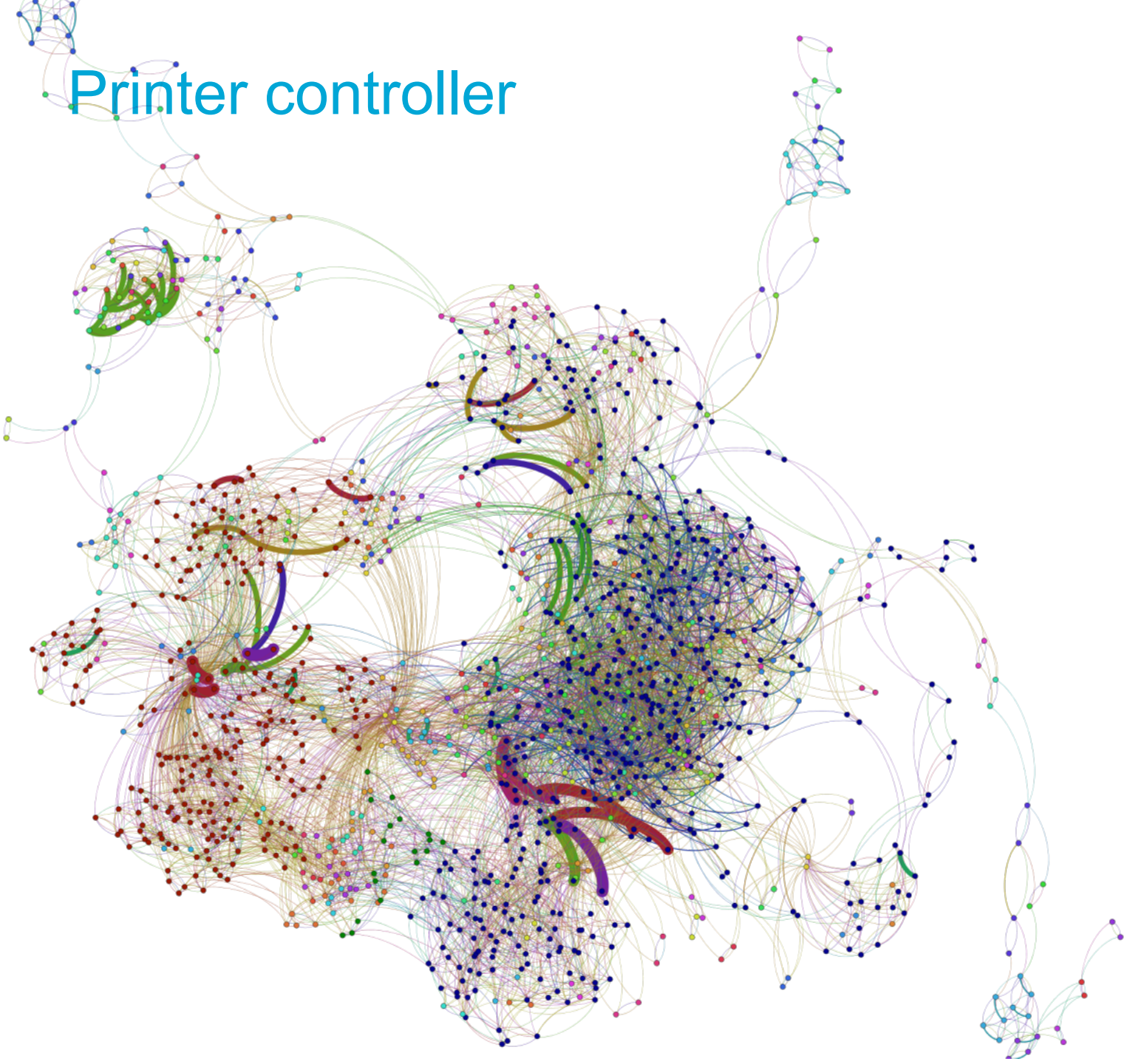
State machine learning



Why learn state machines?

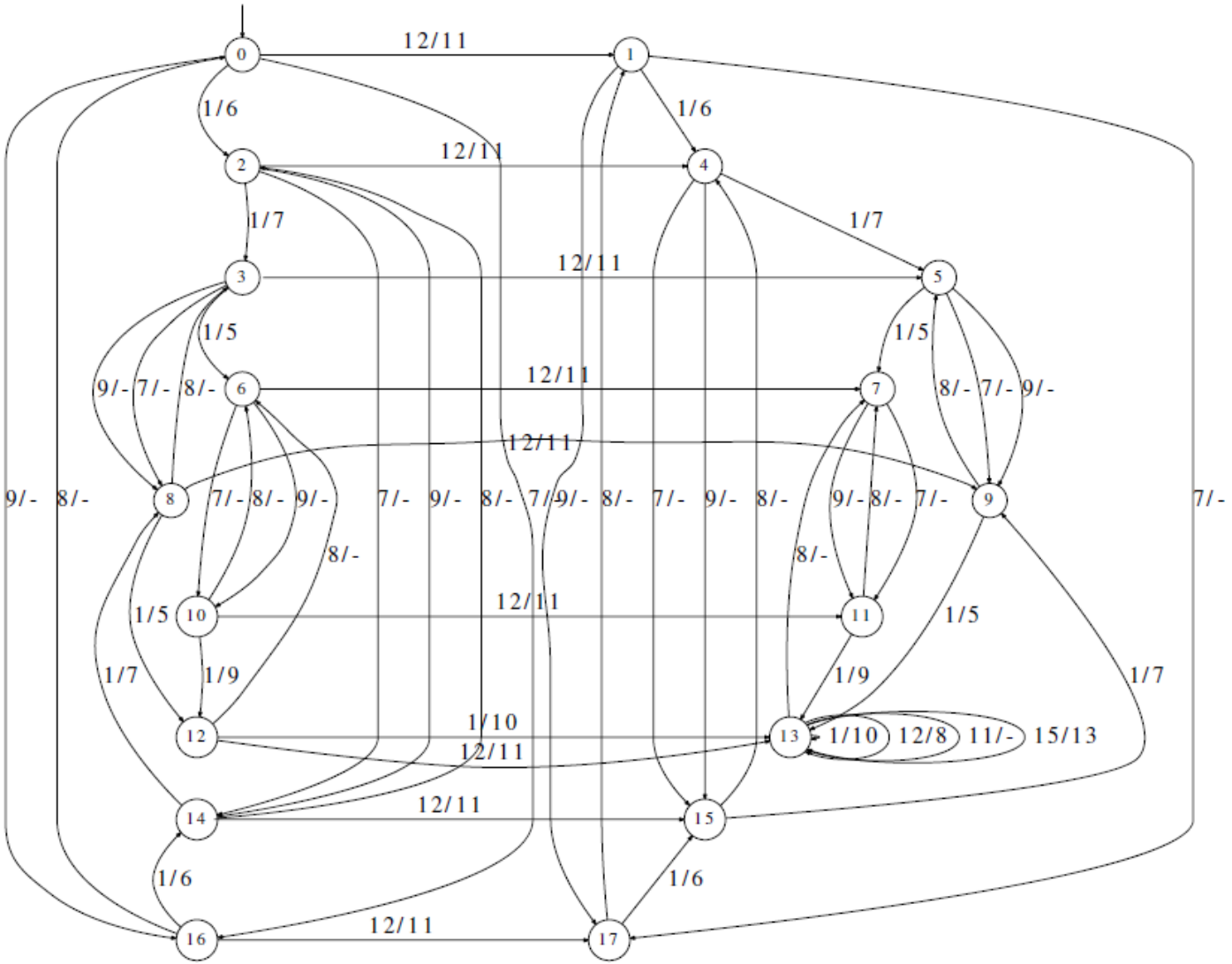
- Powerful analysis methods (e.g., model checkers) are available that can use learned models
- Very intuitive models for software systems
  - Often we don't have models of software components
  - Even when we have models, implementations are often wrong
- Also been used for modeling:
  - Bioinformatics, music, physics, robotics, natural language, …

Printer controller



TU Delft

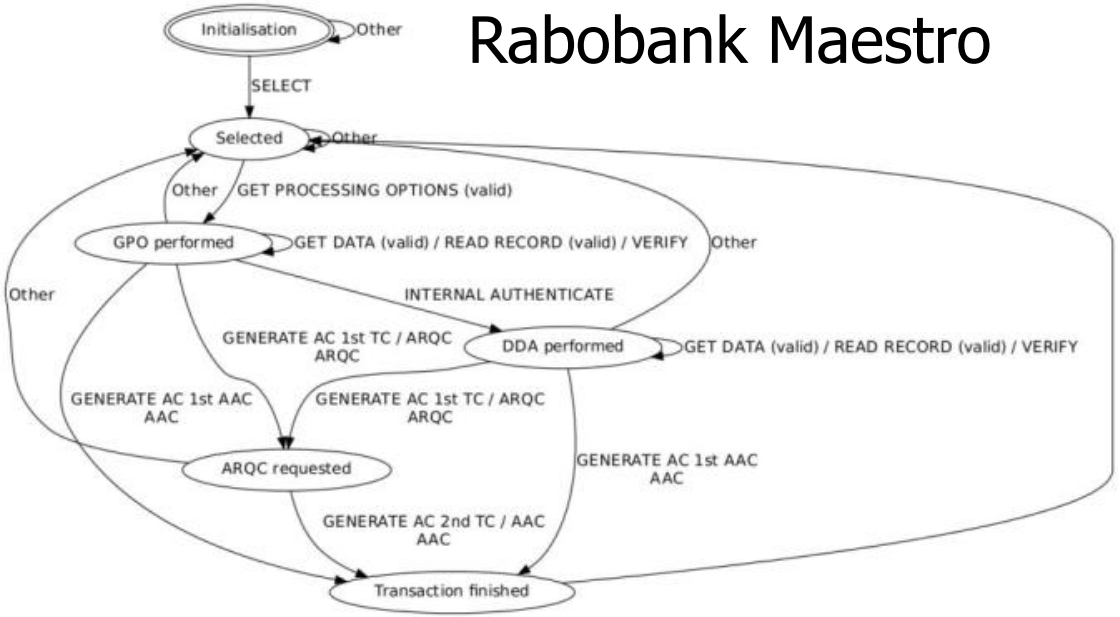
MegaD botnet protocol – Cho et al. 2010



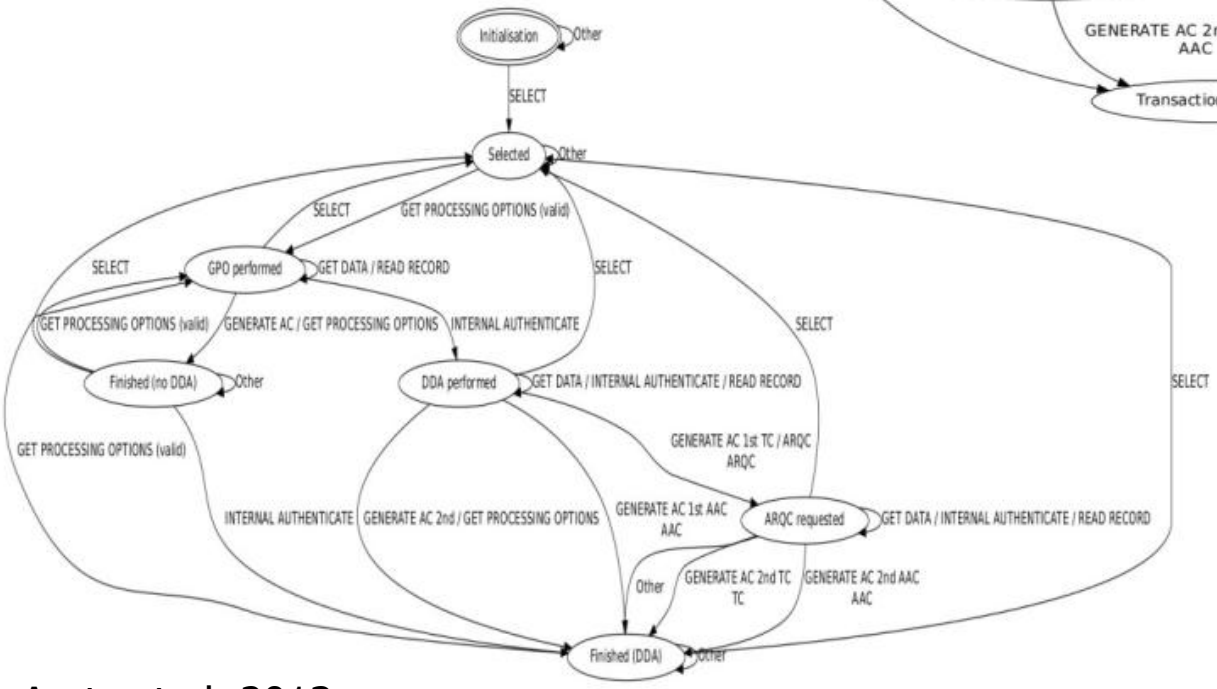
TU Delft

Bank cards

Rabobank Maestro



Volksbank Maestro



Aarts et al. 2013

TU Delft

Both passed a Meastro approved compliance test-suite presumably..

Some important questions…

What you will learn from reading the papers

- Basic algorithms for learning state machines, and how to apply them in practice.

What you will learn from working with DFASAT/QSM/Learnlib:

- If possible, investigate the state space of one of your former communication protocols.
- Look at the protocol used by one/several of:
  - Tribler: http://www.tribler.org
  - A multi-agent environment interface: https://github.com/eishub
  - The transport layer security protocol (TLS)
  - A botnet, e.g., HLUX

- Investigate the inferred state machine, do you see any problems?

Binary reverse engineering

Binary reverse engineering

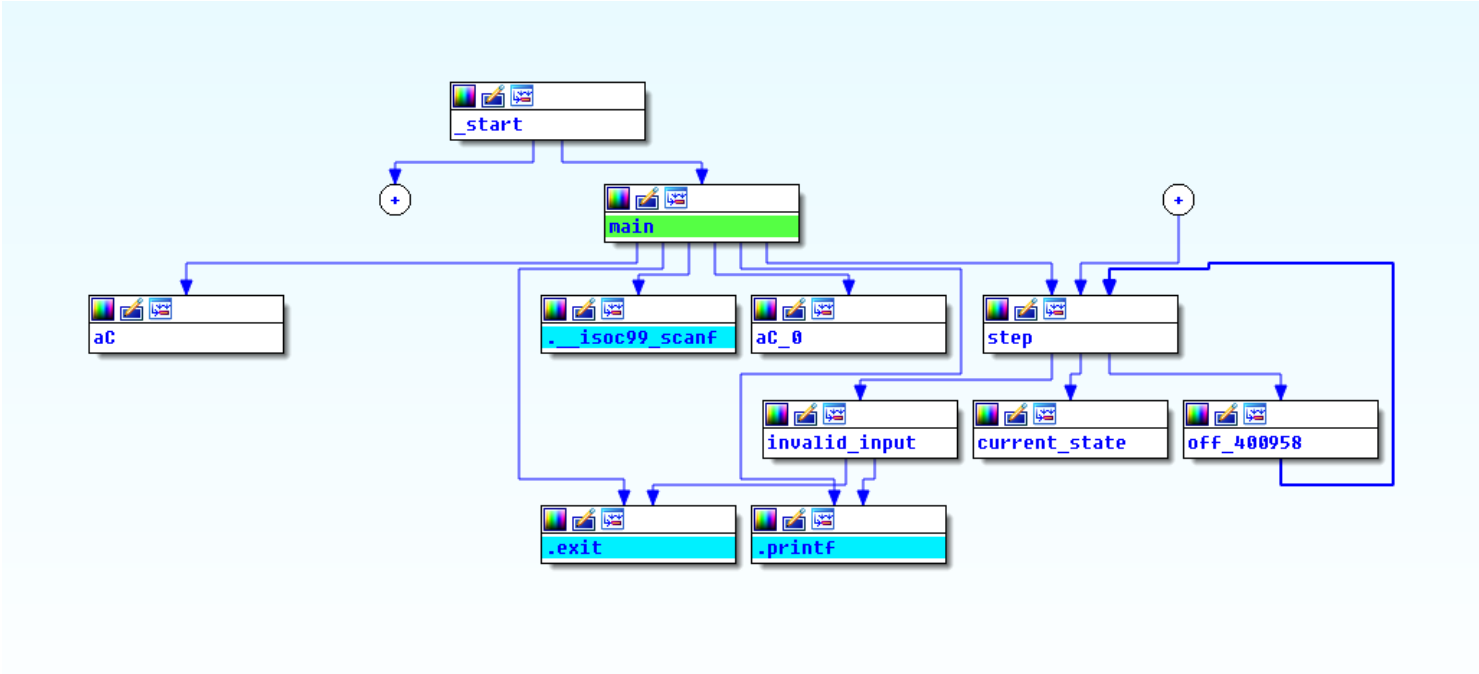
```
int main() {
  // main i/o-loop
  while (1) {
    // read input
    char input = 0;
    int ret = scanf("%c", &input);
    if (ret == EOF)
      exit(0);
    else if (input >= 'A') {
      // operate state machine
      char c = step(input);
      printf("%c\n", c);
    }
  }
}
```

Binary reverse engineering (2)

```
; int __cdecl main(int argc, const char **argv, const char **envp)
                public main
main            proc near               ; DATA XREF: _start+1D↑o

var_14          = dword ptr -14h
var_D           = byte ptr -0Dh
var_C           = dword ptr -0Ch
var_5           = byte ptr -5
var_4           = dword ptr -4

                push    rbp
                mov     rbp, rsp
                sub     rsp, 20h
                mov     [rbp+var_4], 0

loc_40085F:                             ; CODE XREF: main:loc_4008C7↓j
                mov     rdi, offset aC  ; "%c"
                lea     rsi, [rbp+var_5]
                mov     [rbp+var_5], 0
                mov     al, 0
                call    ___isoc99_scanf
                mov     [rbp+var_C], eax
                cmp     [rbp+var_C], 0FFFFFFFFh
                jnz     loc_40088F
                xor     edi, edi        ; status
                call    _exit
; ---------------------------------------------------------------------------

loc_40088F:                             ; CODE XREF: main+32↑j
                movsx   eax, [rbp+var_5]
                cmp     eax, 41h
                jl      loc_4008C2
                movsx   edi, [rbp+var_5]
                call    step
                mov     rdi, offset aC_0 ; "%c\n"
                mov     [rbp+var_D], al
                movsx   esi, [rbp+var_D]
                mov     al, 0
                call    _printf
                mov     [rbp+var_14], eax
```

TU Delft

Binary reverse engineering (3)



Binary reverse engineering (4)

```
1  int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
2  {
3    char v3; // ST13_1@5
4    char v4; // [sp+1Bh] [bp-5h]@2
5    int v5; // [sp+1Ch] [bp-4h]@1
6
7    v5 = 0;
8    while ( 1 )
9    {
10     v4 = 0;
11     if ( __isoc99_scanf("%c", &v4, envp) == -1 )
12       break;
13     if ( v4 >= 65 )
14     {
15       v3 = step();
16       printf("%c\n", (unsigned int)v3);
17     }
18   }
19   exit(0);
20 }
```

Some important questions…

What you will learn from reading the papers

- State-of-the-art tools for understanding binary code, methods to circumvent analysis, and how to avoid those.
- The benefit of visual analysis of code.

What you will learn from using IDA/Radare/Jakstab/GDB:

- Compile one of your former projects with and without optimizations and obfuscation. Understand and label the obtained blocks of assembly.

- Solve a crackme challenge.
- Investigate malicious code.

In conclusion

- Get an overview of state-of-the-art research in testing and reversing
- Use testing and reversing tools in practice
  - *Important for receiving a high grade is to not only apply these tools, but to demonstrate a good understanding of their output*

- We form groups on Github, please register:
  - https://classroom.github.com/group-assignment-invitations/1886dbaeef0c257ff5bd406016bc9eb1
- Slides and papers/topics will be available at:
  - https://github.com/TUDelft-CS4110/syllabus
- Also register on Slack, also for forming groups:
  - https://cs4110.slack.com/

- Email/Slack if you need help forming a group:
  - a.e.zaidman@tudelft.nl