Software Protection Research

ISSISP 2017—Introduction

Christian Collberg

Department of Computer Science
University of Arizona

http://collberg.cs.arizona.edu

collberg@gmail.com

Supported by NSF grants 1525820 and 1318955 and
by the private foundation that shall not be named

Man-At-The-End Scenarios

Tools vs. Counter Tools

What is Obfuscation?

What is Tamperproofing?

Exercises

Questions

July 17 | July 18 | July 19 | July 20 | July 21

09:00 — 10:30	**Basic obfuscation**	Christian Collberg
10:30 — 11:00	Coffee Break	
11:00 — 12:30	**Basic obfuscation [hands-on]**	Christian Collberg
12:30 — 14:00	Lunch	
14:00 — 16:00	**Exploitation & basic defenses [hands-on]**	Josselin Feist
16:00 — 16:30	Coffee Break	

09:00 — 10:30	**Watermarking & Fingerprinting**	Christian Collberg
10:30 — 11:00	Coffee Break	
11:00 — 12:30	**Anti-deobfuscation Techniques**	Bjorn De Sutter
12:30 — 14:00	Lunch	
14:00 — 15:30	**Protection Evaluation**	Bjorn De Sutter
15:30 — 16:00	Coffee Break	
16:00 — 17:30	**Malware Analysis**	Arun Lakhotia
18:00 — 19:30	Poster Session	

- Hands-on during the lectures
- Install the Tigress obfuscator:

  http://tigress.cs.arizona.edu/#download

- Get the test program:

  http://tigress.cs.arizona.edu/fib.c

- Tigress runs on Linux and MacOS

July 17 | July 18 | July 19 | July 20 | July 21

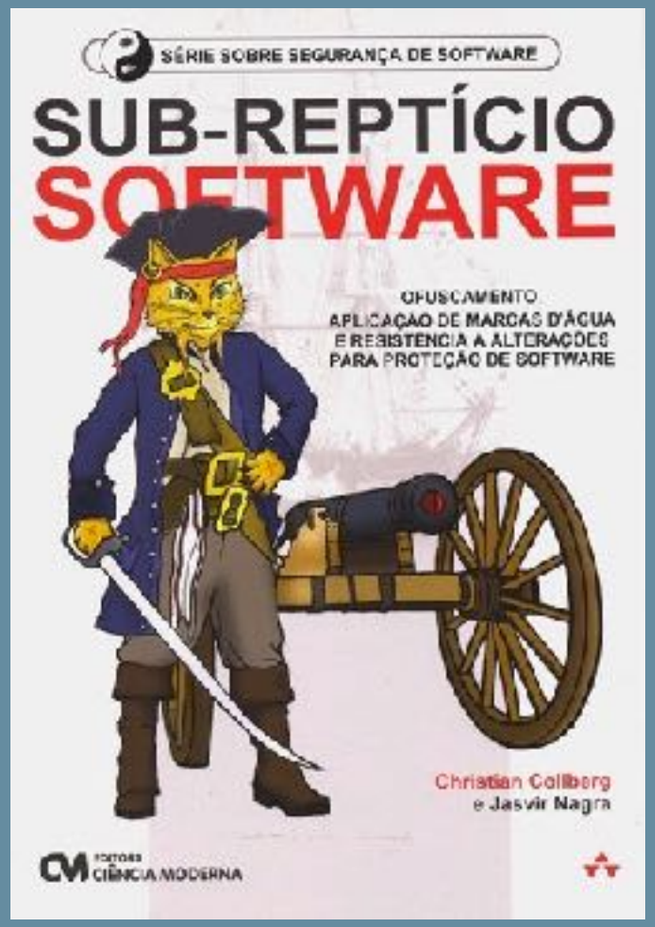
09:00 — 10:30	**Foundations of obfuscation**	Roberto Giacobazzi
10:30 — 11:00	Coffee Break	
11:00 — 12:30	**Obfuscation & static semantic analyses**	Roberto Giacobazzi
12:30 — 14:00	Lunch	
14:00 — 15:30	**Practical Formal Methods**	Sébastien Bardin
15:30 — 16:00	Coffee Break	
16:00 — 17:30	**Malware Analysis [hands-on]**	Arun Lakhotia

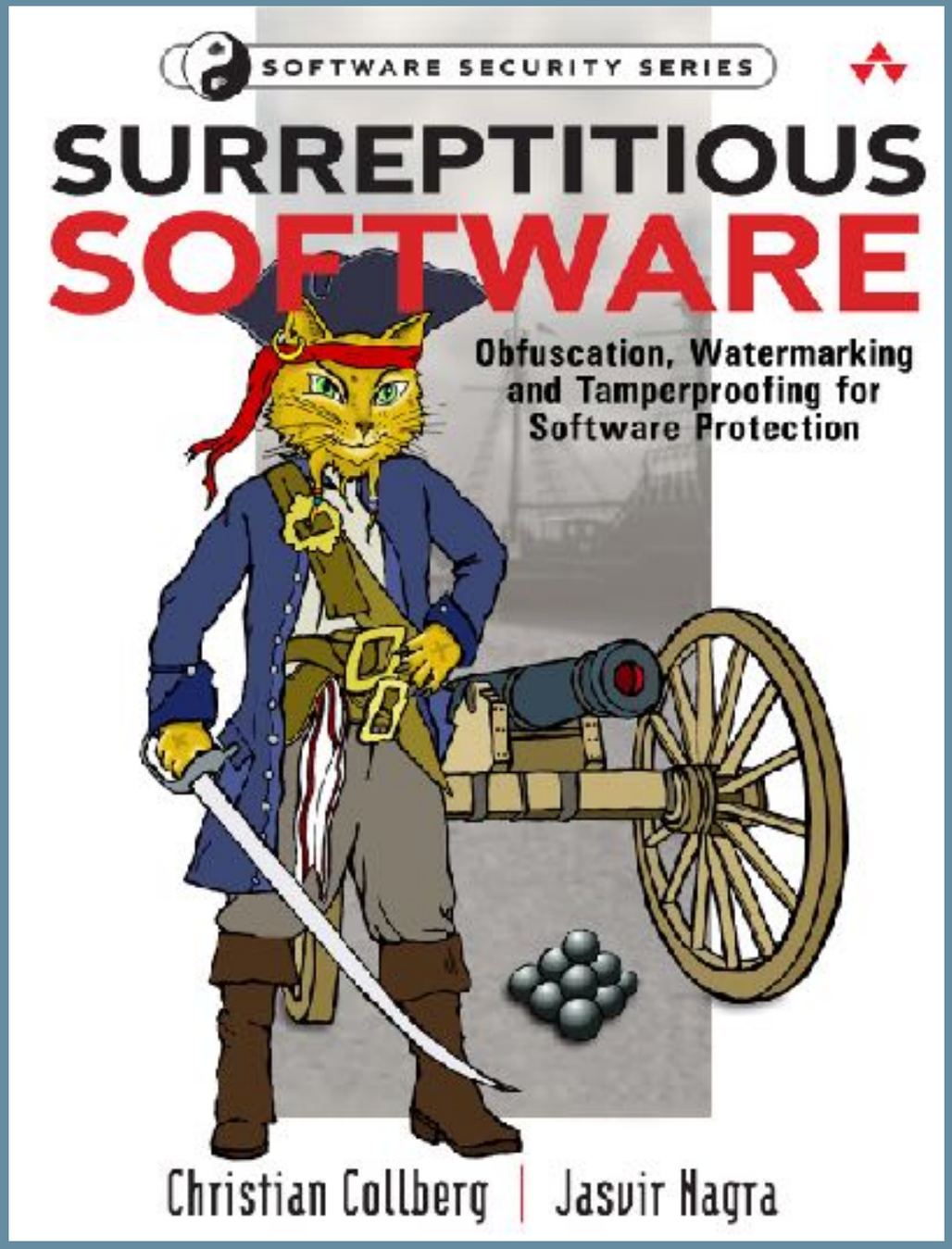
July 17 | July 18 | July 19 | July 20 | July 21

09:00 — 10:30	**Advanced Software Protection**	Jack Davidson
10:30 — 11:00	Coffee Break	
11:00 — 12:30	**Advanced Software Protection**	Jack Davidson
12:30 — 14:00	Lunch	
14:00 — 23:00	Social Event	

July 17 | July 18 | July 19 | July 20 | July 21

09:00 — 10:30	**Side-channel Attacks**	Damien Couroussé
10:30 — 11:00	Coffee Break	
11:00 — 12:30	**Fault Injection**	Ronan Lashermes
12:30 — 14:00	Lunch	
14:00 — 16:00	**The Industrial Challenge in Software and Information Protection**	Yuan Xiang Gu
16:00 — 16:30	Coffee Break	
16:30 — 17:30	**Panel: Industrial Challenges**	Yuan Xiang Gu

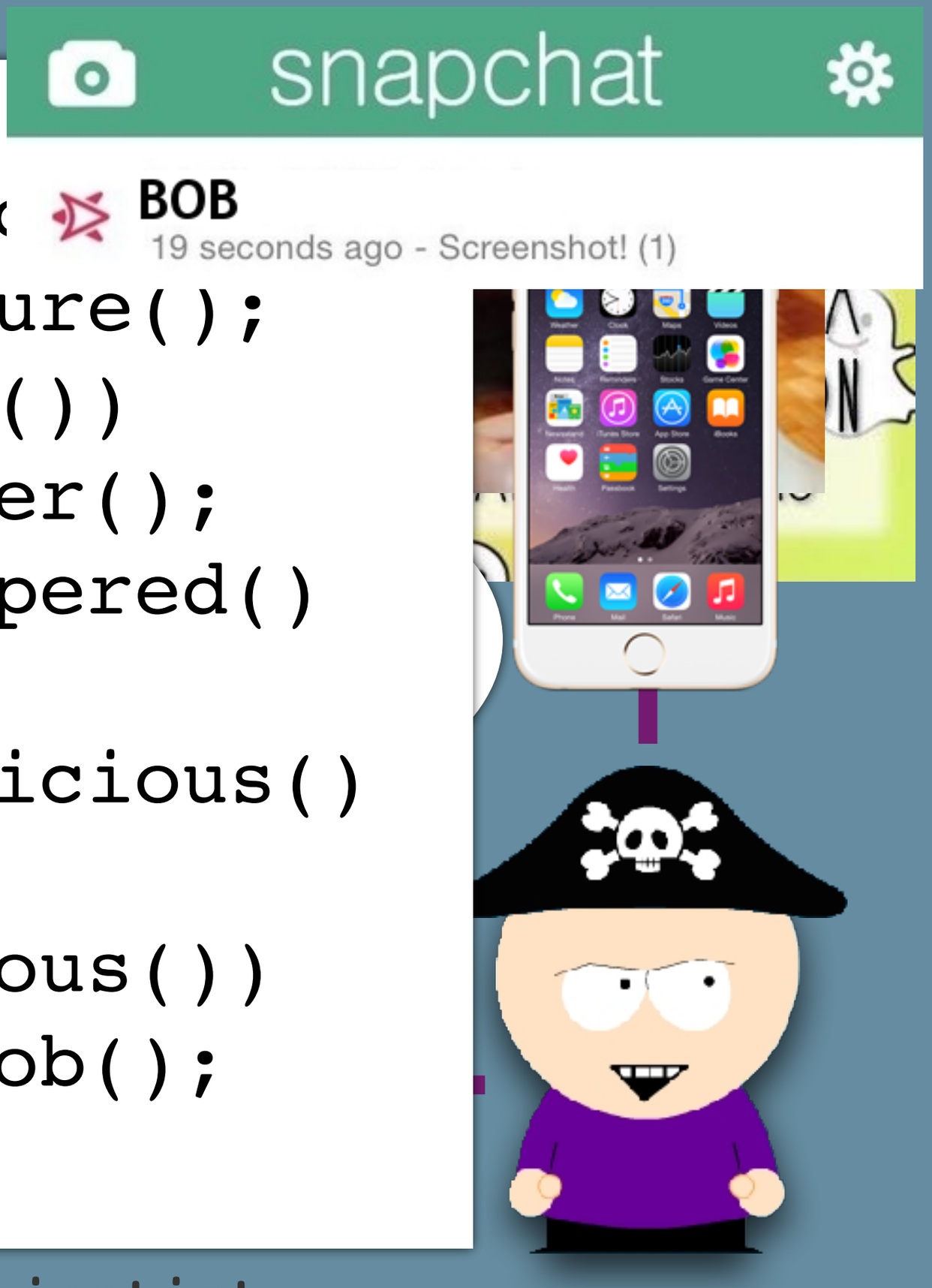

SÉRIE SOBRE SEGURANÇA DE SOFTWARE
SUB-REPTÍCIO
SOFTWARE
OFUSCAMENTO
APLICAÇÃO DE MARCAS D'ÁGUA
E RESISTÊNCIA A ALTERAÇÕES
PARA PROTEÇÃO DE SOFTWARE
Christian Collberg
e Jasvir Nagra
CIÊNCIA MODERNA


SOFTWARE SECURITY SERIES
SURREPTITIOUS
SOFTWARE
Obfuscation, Watermarking
and Tamperproofing for
Software Protection
Christian Collberg | Jasvir Nagra


软件加密与解密

Man-at-the-End Scenarios

```
snapchat() {
    after (8 secon
        remove_picture();
    if (screenshot())
        notify_sender();
    if (app_is_tampered()
        ||
       env_is_suspicious()
        ||
       bob_is_curious())
          punish_bob();
}
```



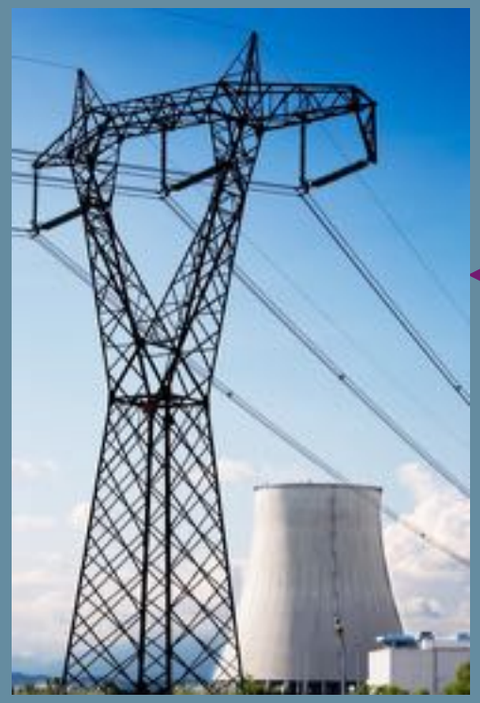
Security and Privacy Scientist

Man-At-The-End

MATE attacks occur in any setting where an adversary has physical access to a device and compromises it by inspecting, reverse engineering, or tampering with its hardware or software.

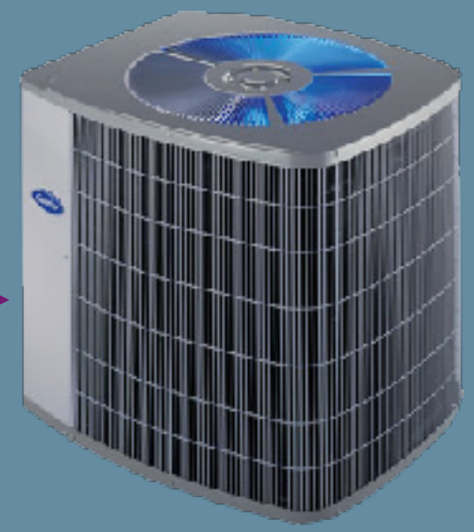
```
set_top_box() {
    if (bob_paid("ESPN"))
        allow_access();

    if (hw_is_tampered()
        ||
        sw_is_tampered()
        ||
        bob_is_curious()
        ||…)
            punish_bob();

}
```

Code & Content



kWh

On/Off



Off!

Cleemput, Mustafa, Preneel, *High Assurance Smart Metering*

```
int main () {
    if (false){
       printf(“License expired!”);
       abort;
    }
}
```

Hack!

- Extract Code!
- Discover Algorithms!
- Find Design!
- Find Keys!
- Modify Code!

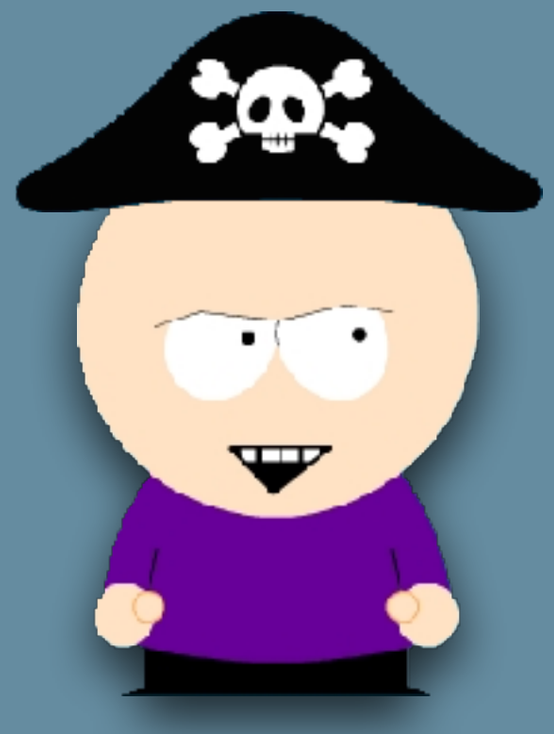
Hack!

```
int foo() {
  … … … …
}
```

Man-At-The-End

```
int main () {
    … … …
    trade_secret()
    … … …
}
```

“Code Lifting”



```
int DigitalRightsMgmt () {
    album=download();
    key=0x47…;
    song=decrypt(key,album);
    play(song);
}
```

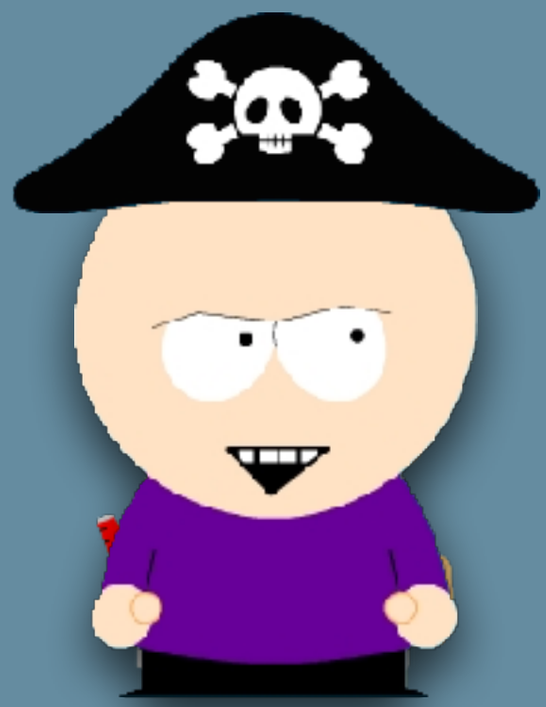
encrypt(

)

Cached secret data

The Washington Post

Saturday, June 28, 2003

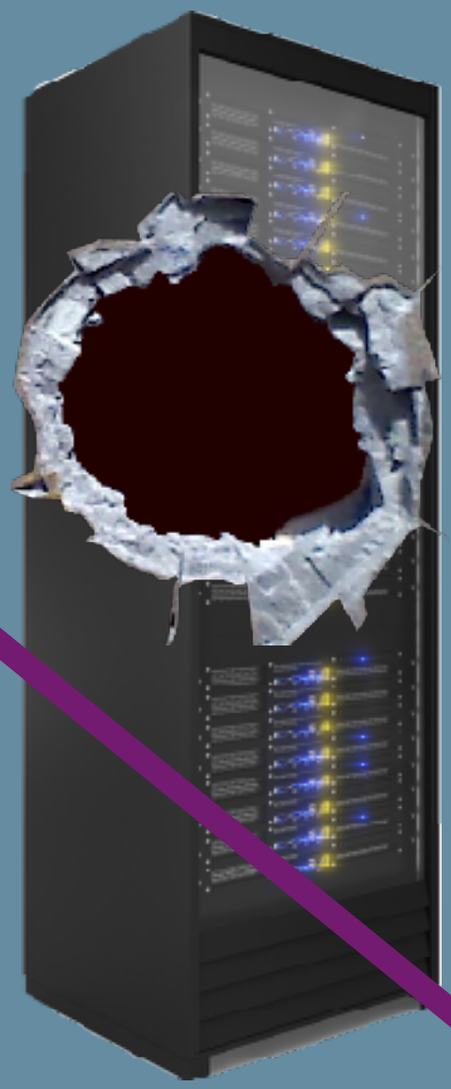


Malicious insider!

Secret!

- Evade discovery!
- Hide intent!
- Destroy data!
- Exfiltrate secrets!

Malware!



Secret!

- Exploit vulnerability!
- Evade discovery!
- Survive reboot!
- Hide intent!
- Destroy data!
- Exfiltrate secrets!

Exercises

Discuss with your friends!!!

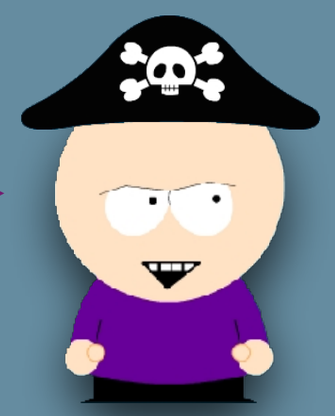
Man-At-The-End

MATE attacks occur in any setting where an adversary has physical access to a device and compromises it by inspecting, reverse engineering, or tampering with its hardware or software.

Can you think of other situations where a MATE (Man-At-The-End) attack could occur?

Discuss with your friends!!!

- Extract Code!
- Discover Algorithms!
- Find Design!
- Find Keys!
- Modify Code!

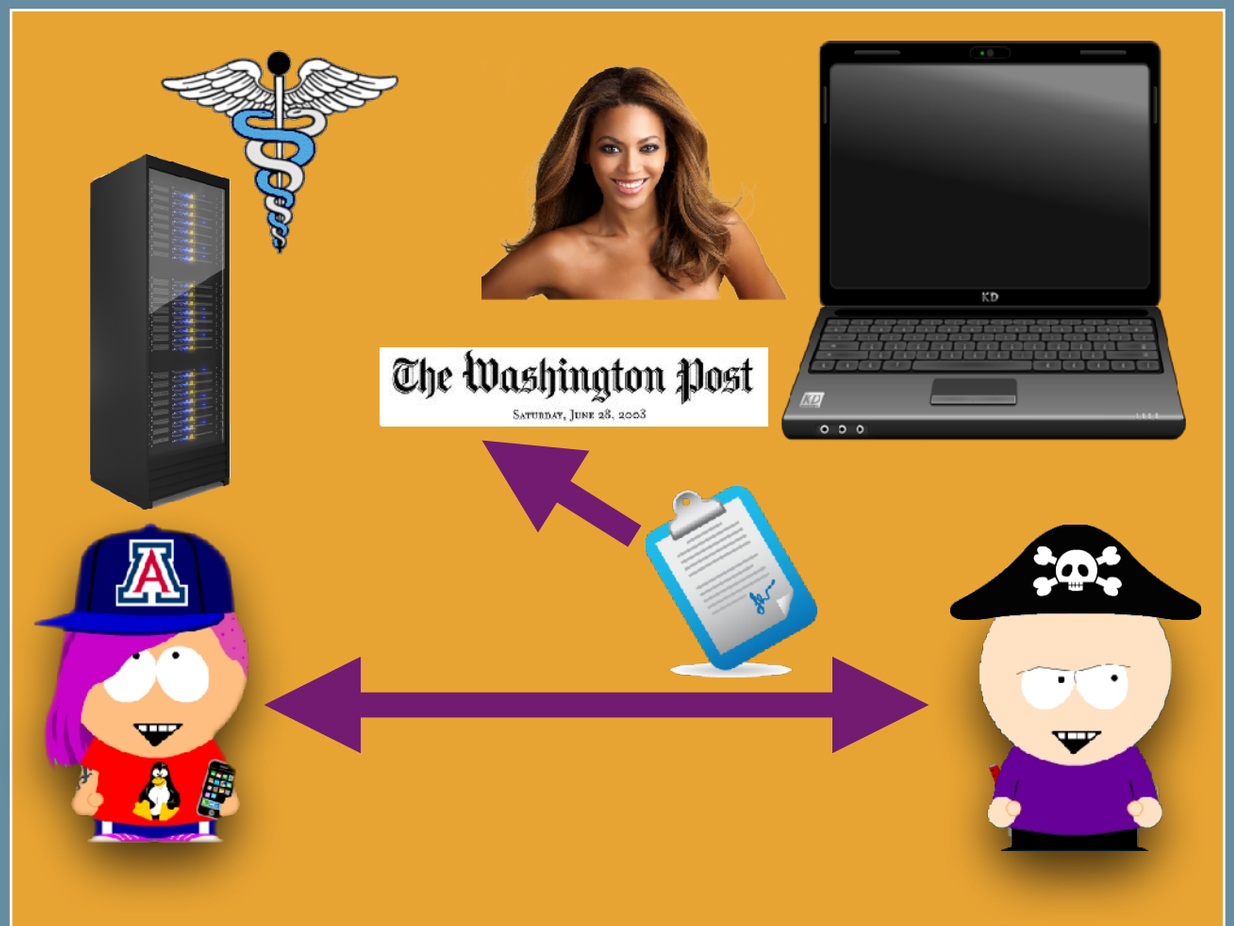


```
int DigitalRightsMgmt () {
    album=download();
    key=0x47…;
    song=decrypt(key,album);
    play(song);
}
```

Consider these two MATE scenarios!

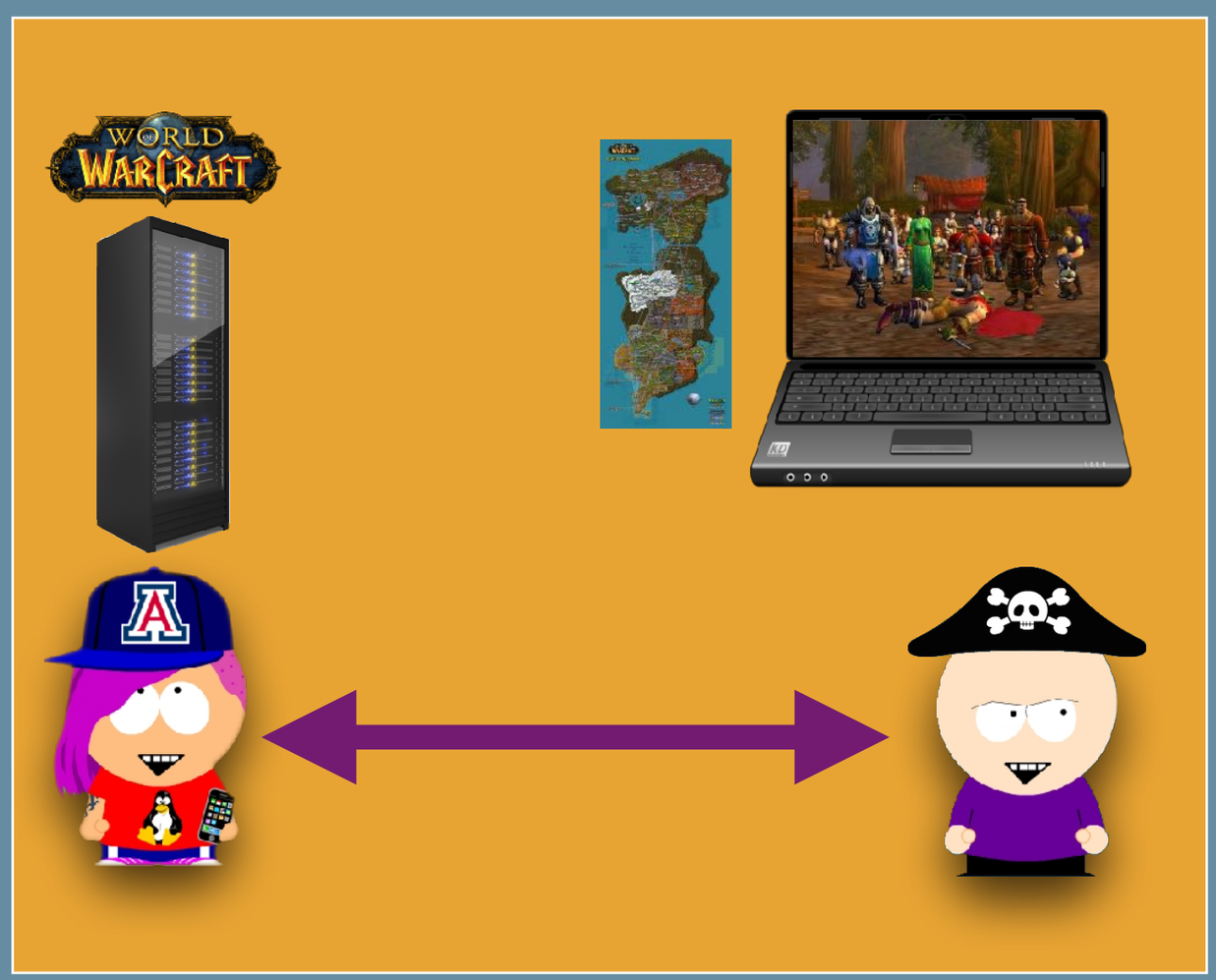
How are they similar? Different?

```
int main () {
    if (false){
      printf("License expired!");
      abort;
    }
}
```



Consider these two MATE scenarios!

How are they similar? Different?



Remote Man-At-The-End

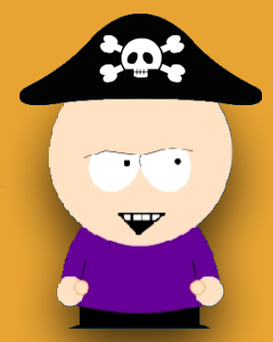
R-MATE attacks occur in distributed systems where untrusted clients are in frequent communication with trusted servers over a network, and where a malicious user can get an advantage by compromising an untrusted device.

```
int main () {
    if (false){
      printf("License expired!");
      abort;
    }
}
```

Hack!

Consider these two MATE scenarios!

How are they similar? Different?



Tools

vs.

Counter Tools

Code Transformations

Obfuscation

Tamperproofing

Remote Attestation

Whitebox Cryptography

Environment Checking

Watermarking

```
Prog() {
}
```

Assets

- Source
- Algorithms
- Keys
- Media

Overhead?

Protection?

Aspire

ARXAN

NAGRA KUDELSKI GROUP

code Virtualizer

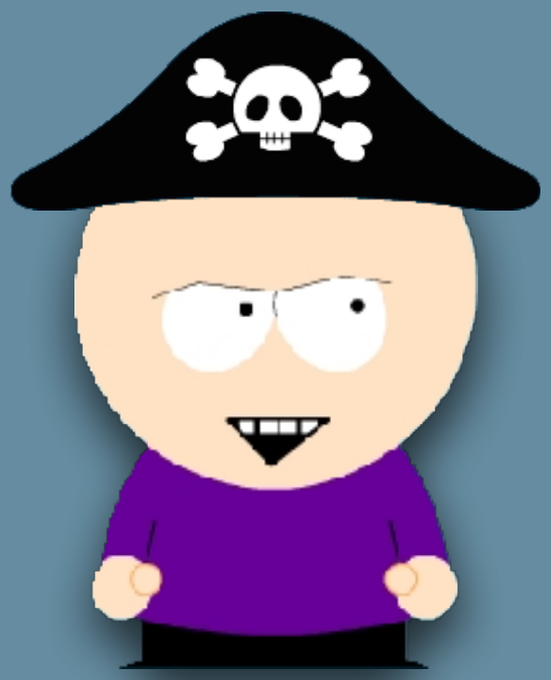
Tool

irdeto

Tigress

VMProtect software

Obfuscator-LLVM



Code Analyses

Static analysis

Dynamic analysis

Concolic analysis

Disassembly

Decompilation

Slicing

Debugging

Emulation

T

Precision?

Prog'

Hex-Rays
state-of-the-art code analysis

- Source
- Algs
- Keys
- ta

TRITON
Dynamic Binary Analysis

angr

S²E

What Matters?

Performance

Time-to-Crack

S²E

angr

TRITON
Dynamic Binary Analysis

KLEE

Hex-Rays
state-of-the-art code analysis

Stealth

AVG

KASPERSKY lab

NOD32 antivirus system

AntiVir

McAfee SECURITY

avast! antivirus

ZONE LABS

bitdefender
secure your every bit

symantec.

Performance Matters?

Metric	Program	Slowdown
absolute time	application	<1s
relative	application	1.5x
relative	security kernel	100x-1000x

Code virtualizer	ExeCryptor	VMProtect	Themida
100x	700x	500x	1200x

Liem, Gu, Johnson: A compiler-based infrastructure for software-protection, PLAS'08

Indistinguishability Obf.

Program	Generate	Run
2-bit multiplier	1027 years	10^8 years
16-bit point function	7 hours, 25G	4 hours (later, 20 minutes)

Bernstein et al., Bad Directions in Cryptographic Hash Functions, IS&P'15

Apon, et al., Impl. Cryptographic Program Obfuscation, CRYPTO'14

Banescu, et al, Benchmarking Indistinguishability Obf. – A candidate impl.

Time-to-Crack Matters

Program	Adversary	Time
hw+sw		many years
well protected	highly skilled, motivated	4-6 weeks
≈VMProtect	experienced reverse engineer	≈12 months
mass market malware		minutes-hours

What is Obfuscation?

```
public class C {
static Object get0(Object[] I) {
   Integer I7, I6, I4, I3; int t9, t8;
   I7=new Integer (9);
   for (;;) {
     if (((Integer)I[0]).intValue()%
        ((Integer)I[1]).intValue()==0)
       {t9=1; t8=0;}
     else
       {t9=0; t8=0;}
     I4=new Integer(t8);
     I6=new Integer(t9);
     if ((I4.intValue ()^I6.intValue ())!=0)
       return new Integer(((Integer)I[1]).intValue());
      else {
       if ((((I7.intValue()+
              I7.intValue()*I7.intValue())%2!=0)?0:1)!=1)
          return new Integer (0);
       I3=new Integer(((Integer)I[0]).intValue()%
         ((Integer)I[1]).intValue ());
       I[0]=new Integer(((Integer)I[1]).intValue());
       I[1]=new Integer(I3.intValue());
      }
   }
}
```

```
public class C {
    static int gcd(int x, int y) {
        int t;
        while (true) {
            boolean b = x % y == 0;
            if (b) return y;
            t = x % y;
            x = y;
            y = t;
        }
    }
}
```

```
int main() {
  … … … …
}
```

Abstraction Transformation

Destroy module structure, classes, functions, etc.!

Control Transformation

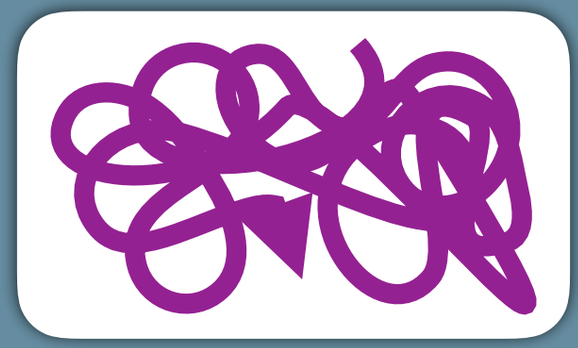
Replace data structures with new representations!

Data Transformation

Destroy if-, while-, repeat-, etc.!

Make the program change at runtime!

Dynamic Transformation



```
int main() {
    int y = 6;
    y = foo(y);
    bar(y,42);
}

int foo(int x)
    return x*7;
}

void bar(int x, int z) {
    if (x==z)
        printf("%i\n",x);
}
```

```
int main() {
    int y = 6;
    y = foobar(y,99,1);
    foobar(y,42,2);
}
```

Abstraction
Transformation

```
int foobar(int x, int z, int s) {
    if (s==1)
        return x*7;
    else if (s==2)
        if (x==z)
            printf("%i \n”,x);
}
```

```
int main () {
    int y = 12;
    y = foobar(y,99,1);
    foobar(y,36,2);
}
int foobar(int x, int z, int s) {
    if (s==1)
        return (x*37)%51;
    else if (x==z) {
        int x2=x*x%51,x3=x2*x%51;
        int x4=x2*x2%51,x8=x4*x4%51;
        int x11=x8*x3%51;
        printf("%i\n",x11);
    }
}
```

Data Transformation

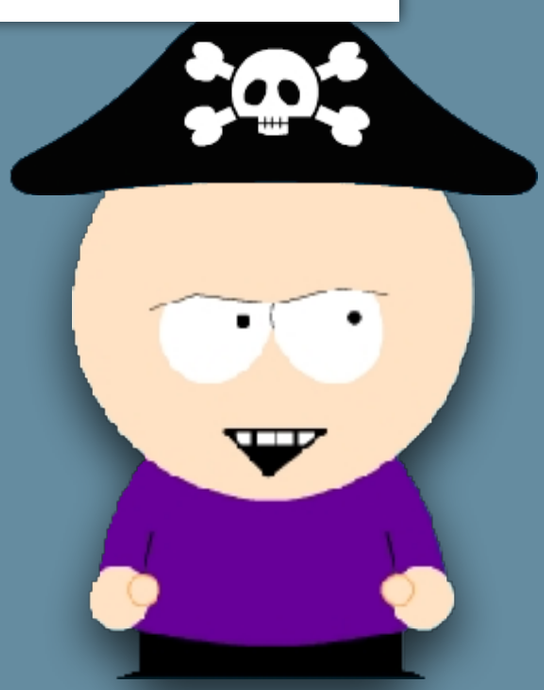
```
int foobar(int x, int z, int s){
    char* next=&&cell0;
    int retVal = 0;
    cell0: {next=(s==1)?&&cell1:&&cell2;
            goto *next;}
    cell1: {retVal=(x*37)%51; goto end;}
    cell2: {next=(s==2)?&&cell3:&&end;
            goto *next;}
    cell3: {next=(x==z)?&&cell4:&&end;
            goto *next;}
    cell4: {
        int x2=x*x%51,x3=x2*x%51;
        int x4=x2*x2%51,x8=x4*x4%51;
        int x11=x8*x3 % 51;
        printf("%i \n",x11); goto end;
    }
    end: return retVal;
}
```

Control Transformation

What is Tamperproofing?

```
int foo () {
    if (today > “Aug 17,2016”){
      printf(“License expired!”);
      abort;
    }
}
```

```
check(){
 if (hash(foo)!=42)
   abort()
}
```



```
int hash (addr_t addr,int words){
   int h = *addr;
   for(int i=1; i<words; i++) {
        addr++;
        h ^= *addr;
    }
    return h;
}
```

```
int foo() {
  … … … …
}
```

Detect tampering

```
int main () {
    if (hash(foo,1000) != 0x4C49F346)
        foo();
}
```

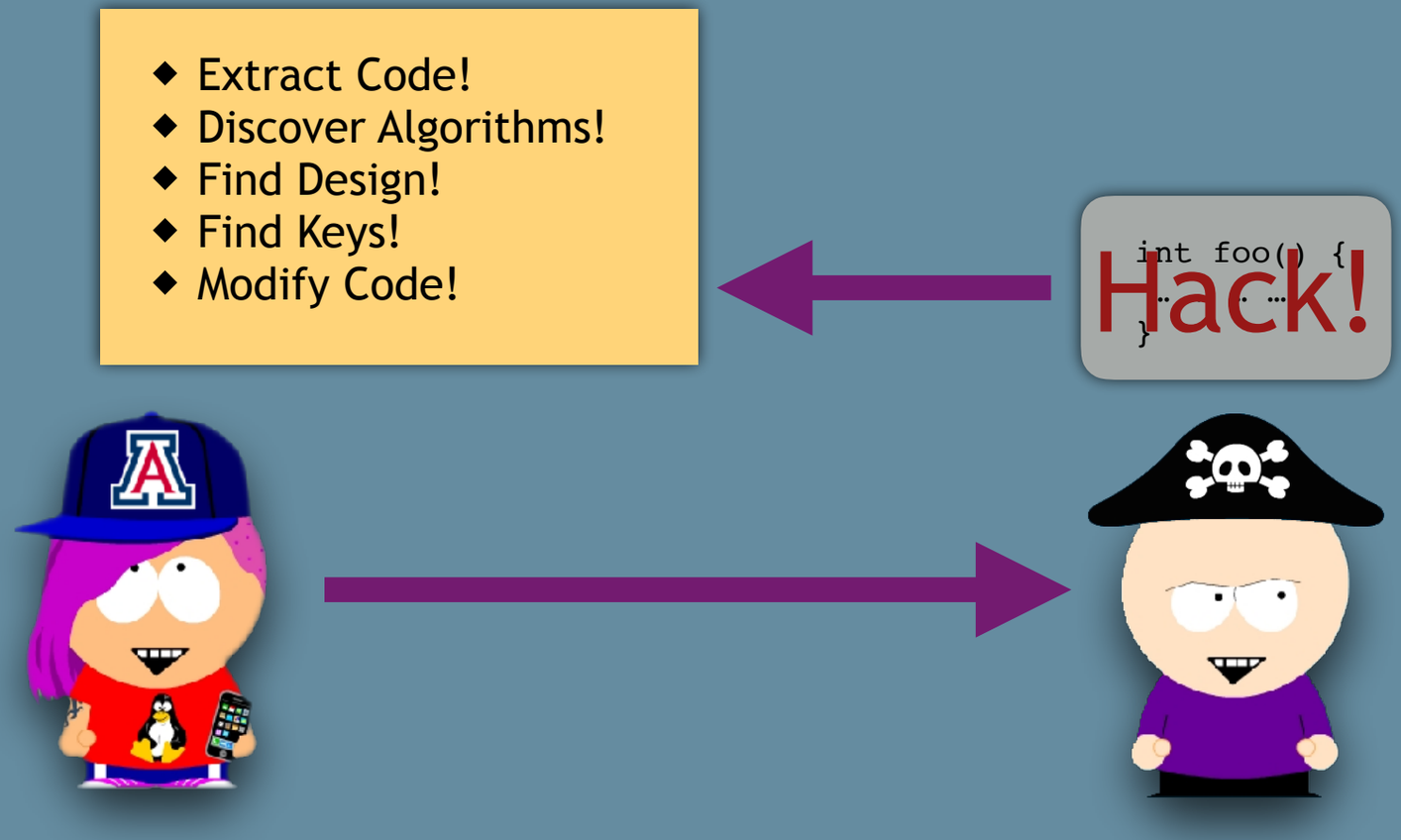
- ***crash the program***
- ***phone home***
- ***refuse to run***
- ***run slower***
- ***make wrong results***

Respond to tampering

Exercises

Discuss with your friends!!!

- *Why do we obfuscate?*
- *Why do we tamperproof?*



- *Can obfuscation be used to tamperproof a program?*

- *Should you both obfuscate and tamperproof a program?*

If so, why?

Questions?