Scientific Research vs. Bug Hunting

Dr. Ismael Ripoll
Universitat Politècnica de València
Departamento de Sistemas y Computadores
iripoll@disca.upv.es
http://cybersecurity.upv.es
About me

- **What papers say:**
  - PhD in 1996 in Computer Science.
  - Assistant professor in the Universidad Politècnica de València.
  - Expert in: Real-time theory, real-time OS, hypervisors, cybersecurity.

- **What I did:**
  - Contributed to RTLinux, back in 1998.
  - Several Projects with European Space Agency to develop the XtratuM hypervisor (and related technologies) for the Next Generation space crafts.
  - Contribution to the Linux kernel.
  - Multiple CVE’s: Linux, glibc, grub, etc.
About me

1) Started researching in real-time scheduling.
   - A lot of math and theorems, nice theorems and solutions, Earliest deadline first, aperiodic servers, hyperperiod reduction etc.

2) Then moved to real time operating systems.
   - RTLinux-GPL and working solutions.

3) A small shift to hypervisors.
   - The logical step, from OS to hypervisor.
   - Real solutions that solve real problems.

4) A big shift to cybersecurity.
   - I has been a researcher, a designer and a programmer.
   - Now I’m a security researcher.

From academia to hacking.
Outline

- What is cybersecurity.
- Offensive security.
- The need for offensive security.
- An example of offensive security (bug hunting).
- Takeaways.
What is cybersecurity?

- Cybersecurity as defined by the European Digital Single Market:
  “Cybersecurity relies on a triad of people, processes and technology (hardware and software). This brings to cybersecurity a combination of technical, socio-economic, ethical and jurisdictional challenges. Furthermore, different broad cybersecurity discourses can be identified within academic and stakeholder communities depending on the primary preoccupation – e.g. privacy and the protection of fundamental rights; criminality and law enforcement; or defence, and national security – though with no clear-cut boundary between them. The Opinion had to draw on these different domains and could not address science and technology separately from the rest.”

- It is not a simple definition.

- In fact, it is an “ill defined problem”.

Ismael Ripoll
What is cybersecurity?

- Malware analysis.
- Reverse engineering.
- Risk and security assessment.
- Incident response.
- Intrusion detection.
- Social engineering.
- Security awareness motivation.
- Exploit development.
- Development of defensive tools.

- Cryptoanalysis.
- Development of defensive tools.
- Law development and enforcement.
- Fault tolerance.
- Design of protocols.
- Forensics.
- Cryptography.
- Defensive programming.
- Data bases.
- Networking.

And many more
What is cybersecurity?

- What is medicine?
- What is physics?
- What is computer science?
- What is cybersecurity?

Maybe it is the wrong question.

- What is a “cybersecurity expert”?
- Is there anything like a “physic expert”? 
What is cybersecurity?

- All those areas are cybersecurity.

  BUT

- Cybersecurity exists because there are hackers.
What is cybersecurity?

- In other research fields, the “root of the problem” is
  - The physical laws
  - The human body
  - The chemical reactions
  - Earth dynamics
  - Logic abstractions (math)
  - Human behaviour
  - Etc.

- In cybersecurity the “root of the problem” is
  - The will to use our electronic devices against us.

Without malicious hackers, there would be only bugs, but not attacks!
What is cybersecurity?

- It depends on what the attackers know.
- It is a changing environment.
- It is a technological escalation.
Offensive security

- The art of war, Sun Tzu:
  
  “If you **know the enemy** and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat. If you know neither the enemy nor yourself, you will succumb in every battle”

- John Lambert, twitter:
  
  “If you shame attack research, you misjudge its contribution. Offense and defense aren’t peers. **Defense is offense’s child**.”
Offensive security

- Don’t wait for the hit.
- Know the tools (technology) of the attackers “before” they can use it.

- If offensive technology is not considered a valid research topic then

→ we are condemned to be the victims.
Offensive security

- Find vulnerabilities (before the attackers do).
- Discover new kind of weaknesses.
- Take actions before having an incident.
- Develop mitigation techniques before they are actually needed.
- Learn to think as attackers do.
Offensive security

What kind of task is “vulnerability hunting”?
A) “Development task” (i.e. apply already known solutions) ?
B) “Engineering task” (i.e. design solutions using patterns) ?
C) “Research task” (i.e. it is a creative process)?

Some bugs are so naive that can be found using a simple source code scanner.

But real bugs are:
– Hard to find and exploit.

Bug hunting is even harder than standard research.

It is very challenging and demanding task: it is research.
Offensive security

• More problems for the researchers in offensive security:
  – **Responsive disclosure** → Don’t publish the results until they have been fixed.
  – **Legal responsibilities** → like body dissection in the middle ages. You could get burned because you want to know.
  – **The knowledge gap** → Cybersecurity is always opening new attack techniques which sometimes they are not “appreciated” by the community (until it is too late).

Spectre y Meltdwon were “suggested” many years before, But no one cared... until a PoC was developed.
Offensive security

- Offensive security is not appealing for academic researches:
  - Hard to find nice results.
  - Hard to publish.
  - Easy to get sued.

- But it is interesting for bad hackers:
  - Lot of money to earn.
  - Bugs are there for a long period of time, waiting to be discovered.
The actual status

Cybersecurity research

Firewalls
Patches

Hackers

Offensive security
How to incentive white research?

ZERODIUM Payouts for Desktops/Servers*

Windows, macOS, Linux, Any OS

RCE: Remote Code Execution
LPE: Local Privilege Escalation
SBX: Sandbox Escape or Bypass
VME: Virtual Machine Escape

https://zerodium.com/program.html
How to incentivize white research?

ZERODIUM Payouts for Mobiles

- Remote Jailbreak with Persistence (FB, Signal, Telegram, WhatsApp, iMessage)
- RCE: Remote Code Execution (FB, Signal, Telegram, WhatsApp, iMessage)
- LPE: Local Privilege Escalation (FB, Signal, Telegram, WhatsApp, iMessage)
- SBX: Sandbox Escape or Bypass

- Up to $1,500,000
- Up to $1,000,000
- Up to $600,000
- Up to $500,000
- Up to $300,000
- Up to $150,000
- Up to $100,000
- Up to $50,000
- Up to $25,000
- Up to $15,000

- iOS/Android

- Code Signing Bypass (iOS/Android)
- Secure Boot (Android)
- LPE to Root (iOS/Android)
- LPE to System (Android)
- Verifed Boot (Android)
- ASLR Bypass (iOS/Android)
- Touch ID Bypass (Android)
- Smartcard Bypass (Android)
- PIN Bypass (Android)

Ismael Ripoll
What is vulnerability hunting?

- Steps in vulnerability hunting:
  1) Find a fault or error: design, implementation, deployment, interaction, etc..
  2) Show that the error can be abused to cause a failure.
  3) Show that the failure has an impact on security.
  4) Develop a PoC to show that it is not an abstract issue but something real.
  5) Propose workarounds and fixes.
An example of bug discovery + exploitation

• We started the research with the following question:

  Q: What is the first user interaction with a Linux system?

  A: GRUB takes the control before the Linux kernel.

  GRUB: Grand Unified Bootloader.

• GRUB management is protected with user/password.
An example of bug discovery + hacking

- GNU/Linux ecosystem is great for hackers.
- It is trivial to get the source code of any application:
  
  ```
  $ apt source grub2
  ```
- Analyze the source code.
- Good identifier names are of great help
  
  ```
  $ grep -r "Enter username" | grep '.c:'
  ```

  ```
  grub-core/normal/auth.c:
  grub_puts_ (N_("Enter username: "));
  ```
- We use QEMU to debug the boot sequence.
Hacking GRUB2: sources

./grub-core/normal/auth.c:grub_auth_check_authentication()

```c
grub_err_t
grub_auth_check_authentication (const char *userlist)
{
    char login[1024];
    struct grub_auth_user *cur = NULL;
    static unsigned long punishment_delay = 1;
    char entered[GRUB_AUTH_MAX_PASSLEN];
    struct grub_auth_user *user;

    grub_memset (login, 0, sizeof (login));

    if (is_authenticated (userlist))
    {
        punishment_delay = 1;
        return GRUB_ERR_NONE;
    }

    grub_puts_ (N("Enter username: "));

    if (!grub_username_get (login, sizeof (login) - 1))
        goto access_denied;
```
Hacking GRUB2: sources

./grub-core/normal/auth.c:grub_username_get()

```c
static int grub_username_get (char buf[], unsigned buf_size) {
    unsigned cur_len = 0;
    int key;

    grub_err_t grub_err_check_authentication (const char *userlist) {
        char login[1024];
        struct grub_auth_user *cur = NULL;
        static unsigned long punishment_delay = 1;
        char entered[GRUB_AUTH_MAX_PASSLEN];
        struct grub_auth_user *user;

        grub_memset (login, 0, sizeof (login));

        if (is_authenticated (userlist))
            {
                punishment_delay = 1;
                return GRUB_ERR_NONE;
            }

        grub_puts_ (N("Enter username: "));

        if (!grub_username_get (login, sizeof (login) - 1))
            goto access_denied;

        grub_memset (buf + cur_len, 0, buf_size - cur_len);

        grub_puts ("\n");
        grub_refresh ();
        return (key != '\e');
    }
```

access_denied:

```c
```

Hacking GRUB2: sources

./grub-core/normal/auth.c:grub_username_get()

```c
static int grub_username_get (char buf[], unsigned buf_size) {
    unsigned cur_len = 0;
    int key;

    while (1) {
        key = grub_getkey ();
        if (key == 'n' || key == 'r')
            break;
        if (key == 'e') {
            cur_len = 0;
            break;
        }
        if (key == 'b') {
            cur_len--;
            grub_printf ("\b");
            continue;
        }
        if (!grub_isprint (key))
            continue;
        if (cur_len + 2 < buf_size) {
            buf[cur_len++] = key;
            grub_printf ("%c", key);
        }
    }
    grub_memset (buf + cur_len, 0, buf_size - cur_len);
    grub_xprintf ("\n");
    grub_refresh ();
    return (key != 'e');
}
```
Hacking GRUB2: sources

./grub-core/normal/auth.c:grub_username_get()

```c
static int grub_username_get (char buf[], unsigned buf_size) {
    unsigned cur_len = 0;
    int key;
    
    while (1){
        key = grub_getkey ();
        if (key == '\n' || key == '\r')
            break;
        if (key == '\e') {
            cur_len = 0;
            break;
        }
        if (key == '\b') {
            cur_len--;
            grub_putchar ("\b");
            continue;
        }
        if (!grub_isprint (key))
            continue;
        if (cur_len + 2 < buf_size) {
            buf[cur_len+1] = key;
            grub_putchar ("%c", key);
        }
    }
    grub_memset (buf + cur_len, 0, buf_size - cur_len);
    grub_puts ("\n");
    grub_refresh ();
    return (key != '\e');
}
```

- `cur_len` is always decremented.
- Integer underflow when decrementing `cur_len`.
- BACKSPACE (`\b`) to decrement `cur_len`.
- Buffer underflow by 2 when `cur_len + 2 < buf_size`.
- Defensive programming.
Hacking GRUB: memory

<table>
<thead>
<tr>
<th>Segment Offset</th>
<th>Segment Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0000</td>
<td>0x0000 0400</td>
</tr>
<tr>
<td>0x0000 0800</td>
<td>0x0000 0820</td>
</tr>
<tr>
<td>0x0000 1000</td>
<td>0x0000 8000</td>
</tr>
<tr>
<td>0x0007 1000</td>
<td>0x0007 ffff</td>
</tr>
<tr>
<td>0x007b 5339</td>
<td>0x07f8 f351</td>
</tr>
<tr>
<td>0xFFff FFff</td>
<td>ROM</td>
</tr>
</tbody>
</table>

For debugging:
- Grub: kernel.img
- Grub: normal.img

Grub: kernel.img
- Code
- Stack
- Scratch mem

IVT: Interrupt Vector Table

grub_username_get()
{
  ...
}
grub_autentication()
{
  ...
}
grub_memset()
{
  ...
}
grub_rescue_run()
{
  ...
}
Hacking GRUB: stack smashing

Username buffer

grub_memset() return address

This is an underflow,
But only which writes zeros

grub_memset (buf + cur_len, 0, buf_size - cur_len);

Ismael Ripoll
Hacking GRUB: stack smashing

We only can control the length of the overflow: 17, 18, 19, 20, ...

Username buffer

16 bytes

grub_memset() return address

Overwrites part of the return address.

0x07eb53e8

20 or more backspace strokes -> jmp *0x0

Ismael Ripoll
Hacking GRUB: now what?

- Under normal circumstances, a jump to 0x0 means:
  - Page not mapped,
  - Code not executable,
  - No control about the contents
- In other words:

GAME OVER
Hacking GRUB: jump to zero

BUT:
Processor is in REAL-MODE → no memory protection
Then it can be executed!

Hexdump of 0x0

Disassembled of 0x0

0x0000: push %ebx
0x0001: incl (%eax)
0x0003: push %ebx
0x0005: incl (%eax)
0x0007: ret
0x0009: loop 0xb
0x000b: push %ebx
0x000d: incl (%eax)
0x000f: push %ebx
0x0011: incl (%eax)
0x0013: push %ebx
0x0015: incl (%eax)
0x0017: push %ebx
0x0019: incl (%eax)
0x001b: push %ebx
0x001d: incl (%eax)
0x001f: movsl %ds:(%esi),%es:(%edi)
0x0021: incb (%eax)
0x0023: xchg %ebp,%ecx
0x0025: add %dh,%al
0x0028: jge 0x0
Ismael Ripoll

Hacking GRUB: self-modifying code

The %esi register has the value of: cur_len

The attacker can control only %esi

Not that much

If %esi == 28 then MAGIC happens

BUT

Copy from %esi to 0x0

movsl %ds:(%esi),%es:(%edi)
Hacking GRUB: self-modifying code

CPU REGS

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eax</td>
<td>0x7f658</td>
<td></td>
</tr>
<tr>
<td>ecx</td>
<td>0x0</td>
<td></td>
</tr>
<tr>
<td>edx</td>
<td>0x0</td>
<td></td>
</tr>
<tr>
<td>ebx</td>
<td>0x0d</td>
<td>(13)</td>
</tr>
<tr>
<td>esp</td>
<td>0x7f664</td>
<td></td>
</tr>
<tr>
<td>ebp</td>
<td>0x0</td>
<td></td>
</tr>
<tr>
<td>esi</td>
<td>0xffff ffe4</td>
<td>(-24)</td>
</tr>
<tr>
<td>edi</td>
<td>0x0</td>
<td>(0)</td>
</tr>
<tr>
<td>eip</td>
<td>0x0</td>
<td>(0)</td>
</tr>
<tr>
<td>esp</td>
<td>0x7f664</td>
<td></td>
</tr>
<tr>
<td>eflags</td>
<td>0x200046 [PF ZF ID]</td>
<td></td>
</tr>
</tbody>
</table>

Ismael Ripoll
Hacking GRUB: self-modifying code

0x0000: push %ebx
0x0001: incl (%eax)
0x0003: lock push %ebx
0x0005: incl (%eax)
0x0007: ret
0x0009: loop 0xb
0x000b: push %ebx
0x000d: incl (%eax)
0x000f: push %ebx
0x0011: incl (%eax)
0x0013: push %ebx
0x0015: incl (%eax)
0x0017: push %ebx
0x0019: incl (%eax)
0x001b: push %ebx
0x001d: incl (%eax)
0x001f: movsl %ds:(%esi),%es:(%edi)
0x0021: incb (%eax)
0x0023: xchg %ebp,%ecx
0x0026: add %dh,%al
0x0028: jge 0x0

0x0000: enter $0x5b66,$0x66
0x0004: push %ebx
0x0005: incl (%eax)
0x0007: ret
0x0009: loop 0xb
0x000b: push %ebx
0x000d: incl (%eax)
0x000f: push %ebx
0x0011: incl (%eax)
0x0013: push %ebx
0x0015: incl (%eax)
0x0017: push %ebx
0x0019: incl (%eax)
0x001b: push %ebx
0x001d: incl (%eax)
0x001f: movsl %ds:(%esi),%es:(%edi)
0x0021: incb (%eax)
0x0023: xchg %ebp,%ecx
0x0026: add %dh,%al
0x0028: jge 0x0

CPU REGS

eax 0x7f658
ecx 0x0 (?)
edx 0x0 (?)
ebx 0xd (13)
esp 0x7f664
ebp 0x0 (?)
esi 0xffff ffe4 (-20)
edi 0x0 (0)
ebp 0x0 (0)
esp 0x79ace
eflags 0x200046[ PF ZF ID ]
Hacking GRUB: self-modifying code

Esp: 0x73f3c

Now GOTO 0xe00c

What it at 0xe00c??
Hacking GRUB: the shell

```c
void __attribute__((noreturn))
grub_rescue_run (void)
{
    #ifdef QUIET_BOOT
        grub_printf ("Entering rescue mode...\n");
    #endif

    while (1)
    {
        char *line;

        /* Print an error, if any. */
        grub_print_error ();
        grub_errno = GRUB_ERR_NONE;

        grub_rescue_read_line (&line, 0, NULL);
        if (! line || line[0] == '\0')
            continue;

        grub_rescue_parse_line(line, grub_rescue_read_line, NULL);
        grub_free (line);
    }
}
```

---

Ismael Ripoll
Hacking GRUB: the shell

Ismael Ripoll
Hacking GRUB: the shell

- What can be done from here:
  - Powerful CLI (Command Line Interface)
    - Change memory, disk, etc.
  - Modify the system, for example install a backdoor.
- The attacker has full control of the system.
  - Note: the exploitation depends on the contents of the ROM. We used the QEMU ROM.

A reddit user

.... Finding overflow bugs is hard, but it's possible and I can imagine how it's done. Up until the part where they jump to 0x0, it was "just" impressive.

But figuring out you can jump out from 0x0 to a critical function by exactly 28 backspace key presses, and waiting 3 iterations of self-modifying code to run? - That's some black magic shit.
Hacking GRUB: summary

- A simple and innocent bug.
  - But in a critical path of the code.
- Got control flow from the keyboard.
- We explorer hundreds of alternatives
  - Stack analysis.
  - Partial address overwrite.
  - Jump to ZERO and
  - Partially controllable self-modifying code.
- Until we got what we where looking for.
  - A ROOT SHELL.
Hacking GRUB: takeaways

- There are bugs in ALL types of software.
  - This bug was in the code from 2009 to 2015.
- This fault was “simple” but was hard to exploit.
- Do not discard local or physical attacks vectors.
- If there is a bug in the boot sequence, then → “secure booting” not that secure.
Back to research
Talk takeaways

• Vulnerability hunting (offensive security) requires:
  – A deep knowledge of a wide range of technologies, a lot of effort/time and good luck

  Offensive security == scientific research

• How far ahead are the attackers today?

• Offensive security must be key research topic.