University Of Tartu
Institute of Computer Science
Informatics

Essay by Alan Durnev
“Research of CVE-2015-8370 vulnerability”

Supervisor: Meelis Roos
Tartu 2016
**Introduction**

GNU Grub is one of the most widely used bootloaders on Unix-like systems and it is the one of the element in chain of launching the Kernel of any OS.

The vulnerability that is being currently research came with the fault in the code of Grub2 in the commit of 2009 year.
This 0-day vulnerability allows the intruder to get access to the **Grub Rescue Shell** (skipping authentication) and as a result completing a successful attack on gaining root privileges and access to all of the locally stored data. In addition, the attacker is able to perform different manipulations, such as installing rootkit.

There are 2 almost equal functions `grub_username_get()` and `grub_password_get()` that are vulnerable, as a result only one of them will be reviewed (the bold one). In a few words: there is a missing check expressions that allows the intruder to press **Backspace** button **28** times which leads to the execution of the `grub_rescue_run()` function (the one that launches shell).

**Exploiting the vulnerability**

```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);
        } else
        {
            grub_menset (buf + cur_len, 0, buf_size - cur_len);
        }

        grub_xputs ("\n");
        grub_refresh ();
        return (key != '\e');
    }
```
The function `grub_memset()` tries to zero all unused bytes of the username buffer and takes 3 parameters: 1. parameter takes the address of the first unused byte and 2. 0 is passed as an argument by default in order to replace any value with it (0) 3. parameter takes the number of bytes that have to be zeroes – `grub_memset(buffer + current_length, 0, buffer_size – current_length)`. The username buffer (array) is 1024 bytes long.

```c
void *
grub_memset (void *start, int c, int len)
{
    char *p = start;
    if (memcheck ((int) start, len))
    {
        while (len -- > 0)
            *p ++ = c;
    }
    return errnum ? NULL : start;
}
```

So, we have an unsigned integer to represent the pointer to the currently stored character in the buffer. The C language has this policy for passing negative integer value to the unsigned integer variable: it will convert the value by repeatedly adding or subtracting (maximum_value + 1) until the value is in the range of the right type. (ie
[unsigned int x = -1; // x == UINT_MAX -> true
]
)

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

    And we have this place in the code which is the fault one, where we can make cur_len = cur_len – 1 (abstractly [in the matter of assignment] new cur_len = -1 if cur_len was 0 [Backspace button {‘\b’} was pressed immediately]):

```c
if (key == '\b')
{
    cur_len--;
    grub_putchar("\b");
    continue;
}
```

No check for the undeflow of the unsigned int
This is how **out of bound overwrite** occurs, producing high value which is later used in calculation of the starting address.

Then we get the **second overflow** by adding this high value to the starting address of the buffer (array) the result value will need more than 32bit variable to held.

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

So, basically, with 1 Backspace pressed the cur_len will be equal to 0xFFFFFFFF (= 2^{31-1} - 2^{31} = -1 for signed int, because of C policy) and when we will add this value to the starting address of the buffer we will get **decremented by one value** (buf + (-1)) and as a result set the whole buffer (1024 bytes) and (1024 – (-1)) byte (**first byte under the buffer**) to zero.

And now when we can overwrite any number of bytes below the username buffer to zero, we have to find the special and most interesting one. By taking look at the picture, we can see that the return address of grub_memset() function is 16byte long from the initial buffer and by pressing Backspace button 17 times (17 bytes shifted in total) we will overwrite the highest byte of the return address.

Addresses 0x00EB53E8, 0x0053E8, 0x0000E8, 0x000000 are the addresses that when they are jumped into it causes the reboot.

At the lowest addresses (usually), starting from 0x0 each processor has an **Interrupt Vector Table**.

Function that has the Grub2 Rescue Shell functionality is named `grub_rescue_run()`. 
There is a while loop inside grub_username_get() that ends with the press of ‘\n’, ‘\r’ or ‘\e’ – newline character, carriage return character (both are connected with the Enter button) and Escape button respectively.

```c
while (1)
{
    key = grub_getkey();
    if (key == '\n' || key == '\r')
        break;
    if (key == '\e')
    {
        cur_len = 0;
        break;
    }
}
```

Now when we have 17-20 times Backspace button pressed and the 0x00…… instruction executed (as a return statement) after Enter button pressed (reboot), we will have the CPU register IP/PC (Instruction Pointer/Program Counter) set to 0x00000000 (0x0) and it will be executed as the next instruction. The EBX CPU register will hold the value of last typed key (after pressing 28 times the Backspace button, we will press Enter button in order to exit from the loop and the ASCII code of the Enter button is ‘0xD’ -> the EBX register will have value ‘0xD’). The ESI CPU register will hold the value of cur_len variable (will hold value **-28**, 0xFFFFFFFE4 in hex).

The movsl instruction is an opcode for memory copy “function” and movsl %ds:(%esi),%es:(%edi) means that we copy everything from the memory referenced by the ESI register (from address 0xFFFFFFFFFE4, in other words top_memory_address - 28) to 0x0
It is stated that if the CPU registers are in this position, then for the third time (after shifted code in the memory) we will get retw instruction that after executing will jump us to the calling procedure, which is pointed by the value in the ESP (stack pointer [top of the stack]), which is at the moment of the 3rd iteration is 0xE00C, which is the address of grub_rescue_run() function.

After that Grub2 Rescue Shell is launched and because of little modifications only inside the first interrupt vector, it will work correctly and every malware will be possible to deploy through this shell.

How to avoid this vulnerability?

First of all, should be mentioned that there are many bootloaders and many of them have as strong as weak sides. GNU Grub2 is just the most used one and in theory it can be replaced by the one that does not suffer from such zero-day serious vulnerability.
Secondly, all problem came from the problem of the unsigned integer variable underflow and the fix is the easiest one that could be imagined:

Slightly changing the if condition by adding additional expression.

```c
if (key == '\b' && cur_len)
{
    //...
```

By pressing Backspace button this condition will evaluate to be true only if the `cur_len` > 0.

**References**

2. http://git.savannah.gnu.org/cgit/grub.git/commit/?id=b391bdb2f2c5ccf29da66cecd6fb7566656a704d