# Flare-On 5 #12

## A trip into bootkits and esoteric ISAs

dp_1, November 16th 2018

# Who am I

Dario Petrillo, 19 y/o

Studying Computer Engineering @ Sapienza

Capturing flags with TheRomanXpl0it and mHACKeroni

Mainly a reverse engineer

@dario_petrillo

dp1

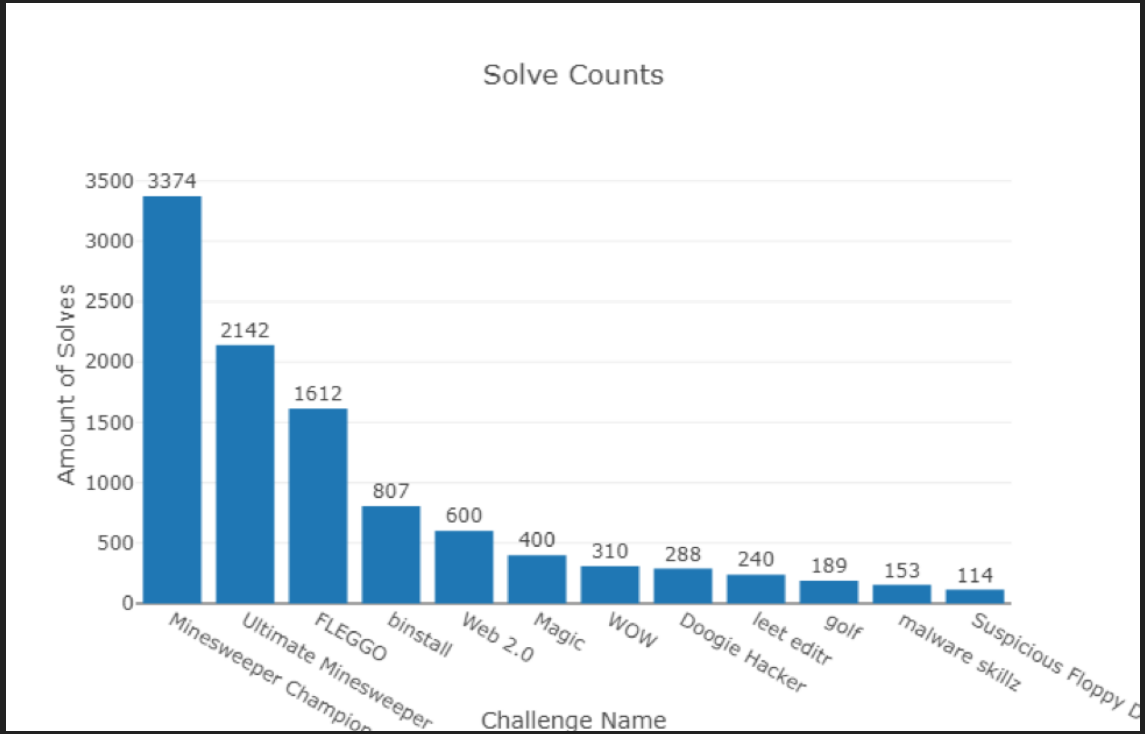
# Flare what?

Annual reverse engineering competition held by Fireye

12 challenges in roughly increasing order of difficulty

Six weeks to solve them all

# Flare what?


Solve Counts
Amount of Solves
3500
3000
2500
2000
1500
1000
500
0
3374
2142
1612
807
600
400
310
288
240
189
153
114
Minesweeper Champion
Ultimate Minesweeper
FLEGGO
binstall
Web 2.0
Magic
WOW
Doogie Hacker
leet editr
golf
malware skillz
Suspicious Floppy D
Challenge Name

# #12: Suspicious floppy disk

“Now for the final test of your focus and dedication. We found a floppy disk that was given to spies to transmit secret messages. The spies were also given the password, we don't have that information, but see if you can figure out the message anyway. You are saving lives.”

# A first look

| Nome | Ultima modifica |
|---|---|
| AUTOEXEC.BAT | 01/08/2018 09:02 |
| COMMAND.COM | 18/04/2005 17:54 |
| CONFIG.SYS | 25/06/2018 11:20 |
| DISPLAY.SYS | 18/04/2005 17:54 |
| EGA.CPI | 18/04/2005 17:54 |
| EGA2.CPI | 18/04/2005 17:54 |
| EGA3.CPI | 18/04/2005 17:54 |
| infohelp.exe | 01/08/2018 08:57 |
| key.dat | 01/08/2018 09:04 |
| KEYB.COM | 18/04/2005 03:04 |
| KEYBOARD.SYS | 18/04/2005 17:54 |
| KEYBRD2.SYS | 18/04/2005 17:54 |
| KEYBRD3.SYS | 18/04/2005 17:54 |
| KEYBRD4.SYS | 18/04/2005 17:54 |
| message.dat | 01/08/2018 09:03 |
| MODE.COM | 18/04/2005 17:54 |
| TMP.DAT | 13/08/2018 23:57 |

# A first look

| Nome | Ultima modifica |
|---|---|
| AUTOEXEC.BAT | 01/08/2018 09:02 |
| COMMAND.COM | 18/04/2005 17:54 |
| CONFIG.SYS | 25/06/2018 11:20 |
| DISPLAY.SYS | 18/04/2005 17:54 |
| EGA.CPI | 18/04/2005 17:54 |
| EGA2.CPI | 18/04/2005 17:54 |
| EGA3.CPI | 18/04/2005 17:54 |
| infohelp.exe | 01/08/2018 08:57 |
| key.dat | 01/08/2018 09:04 |
| KEYB.COM | 18/04/2005 03:04 |
| KEYBOARD.SYS | 18/04/2005 17:54 |
| KEYBRD2.SYS | 18/04/2005 17:54 |
| KEYBRD3.SYS | 18/04/2005 17:54 |
| KEYBRD4.SYS | 18/04/2005 17:54 |
| message.dat | 01/08/2018 09:03 |
| MODE.COM | 18/04/2005 17:54 |
| TMP.DAT | 13/08/2018 23:57 |

# A first look

| Nome | Ultima modifica |
|---|---|
| AUTOEXEC.BAT | 01/08/2018 09:02 |
| COMMAND.COM | 18/04/2005 17:54 |
| CONFIG.SYS | 25/06/2018 11:20 |
| DISPLAY.SYS | 18/04/2005 17:54 |
| EGA.CPI | 18/04/2005 17:54 |
| EGA2.CPI | 18/04/2005 17:54 |
| EGA3.CPI | 18/04/2005 17:54 |
| infohelp.exe | 01/08/2018 08:57 |
| key.dat | 01/08/2018 09:04 |
| KEYB.COM | 18/04/2005 03:04 |
| KEYBOARD.SYS | 18/04/2005 17:54 |
| KEYBRD2.SYS | 18/04/2005 17:54 |
| KEYBRD3.SYS | 18/04/2005 17:54 |
| KEYBRD4.SYS | 18/04/2005 17:54 |
| message.dat | 01/08/2018 09:03 |
| MODE.COM | 18/04/2005 17:54 |
| TMP.DAT | 13/08/2018 23:57 |

```
1  infohelp.exe
2
```

```
1  key goes here
```

```
1  This is not the message you are looking for.
```

# infohelp.exe

Trivially simple program:

> Write password into key.dat

> Print message from message.dat

# infohelp.exe

Trivially simple program:

> Write password into key.dat

> Print message from message.dat

→ Basically does nothing!

# Diving deeper

Something must be intercepting calls to disk

The boot sector is different from an original WinME floppy

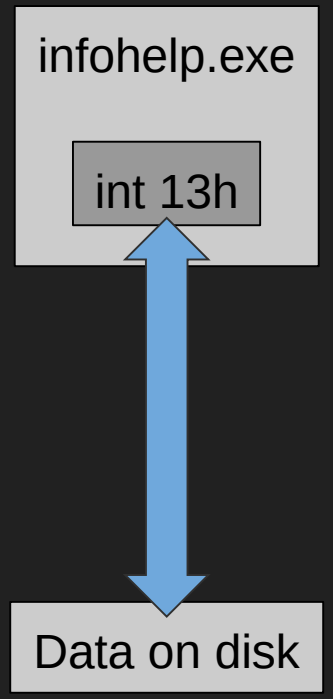
# Diving deeper

Something must be intercepting calls to disk

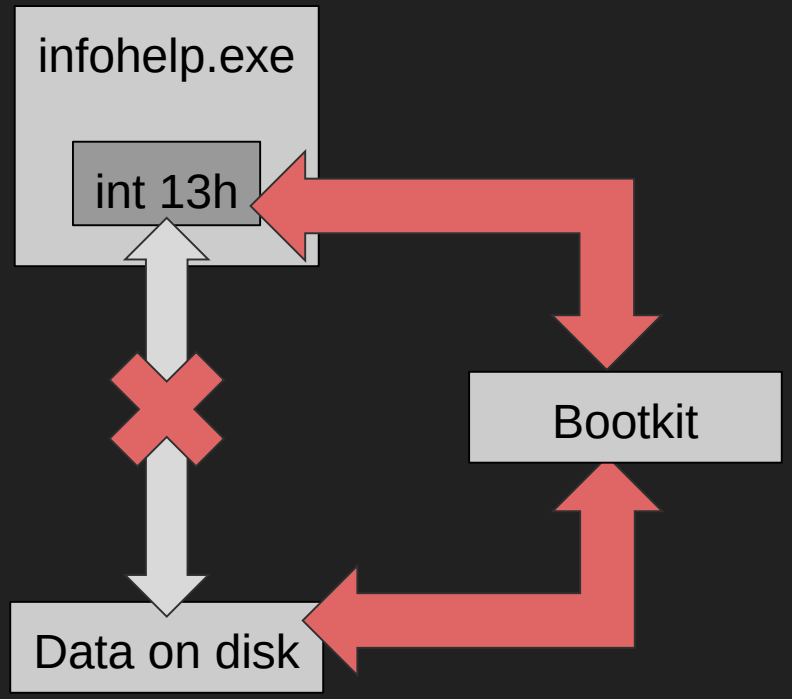
The boot sector is different from an original WinME floppy

Smells like a bootkit!

# Bootkit behaviour



# Bootkit behaviour



# Bootsector analysis

```
loc_801:                                ; DATA XREF: sub_1009ED4+36↓r
                push    es
                xor     ebx, ebx        ; DATA XREF: seg002:100A25C↓r
                mov     bx, ds:413h     ; BIOS data area: size of memory in KB
                sub     bx, 20h ; ' '
                mov     ds:413h, bx
                shl     bx, 6
                mov     ds:66F3h, bx
                mov     es, bx
                mov     si, 843h
                xor     di, di
                mov     cx, 5EB0h
                cld
                rep movsb
                mov     eax, ds:4Ch
                mov     es:5E5Ah, eax
                mov     es:5E6Fh, dl
                mov     ds:200h, eax
                mov     word ptr ds:4Ch, 12h
                mov     word ptr ds:4Eh, es
                pop     es
                popa
                retn
```

# Bootsector analysis

```
loc_801:                                ; DATA XREF: sub_1009ED4+36↓r
                push    es
                xor     ebx, ebx        ; DATA XREF: seg002:100A25C↓r
                mov     bx, ds:413h     ; BIOS data area: size of memory in KB
                sub     bx, 20h ; ' '
                mov     ds:413h, bx
                shl     bx, 6
                mov     ds:66F3h, bx
                mov     es, bx
                mov     si, 843h
                xor     di, di
                mov     cx, 5EB0h
                cld
                rep movsb
                mov     eax, ds:4Ch
                mov     es:5E5Ah, eax
                mov     es:5E6Fh, dl
                mov     ds:200h, eax
                mov     word ptr ds:4Ch, 12h
                mov     word ptr ds:4Eh, es
                pop     es
                popa
                retn
```

> Reserve RAM

# Bootsector analysis

```
loc_801:                                ; DATA XREF: sub_1009ED4+36↓r
        push    es
        xor     ebx, ebx                ; DATA XREF: seg002:100A25C↓r
        mov     bx, ds:413h             ; BIOS data area: size of memory in KB
        sub     bx, 20h ; ' '
        mov     ds:413h, bx
        shl     bx, 6
        mov     ds:66F3h, bx
        mov     es, bx
        mov     si, 843h
        xor     di, di
        mov     cx, 5EB0h
        cld
        rep movsb
        mov     eax, ds:4Ch
        mov     es:5E5Ah, eax
        mov     es:5E6Fh, dl
        mov     ds:200h, eax
        mov     word ptr ds:4Ch, 12h
        mov     word ptr ds:4Eh, es
        pop     es
        popa
        retn
```

> Reserve RAM

> Copy payload

# Bootsector analysis

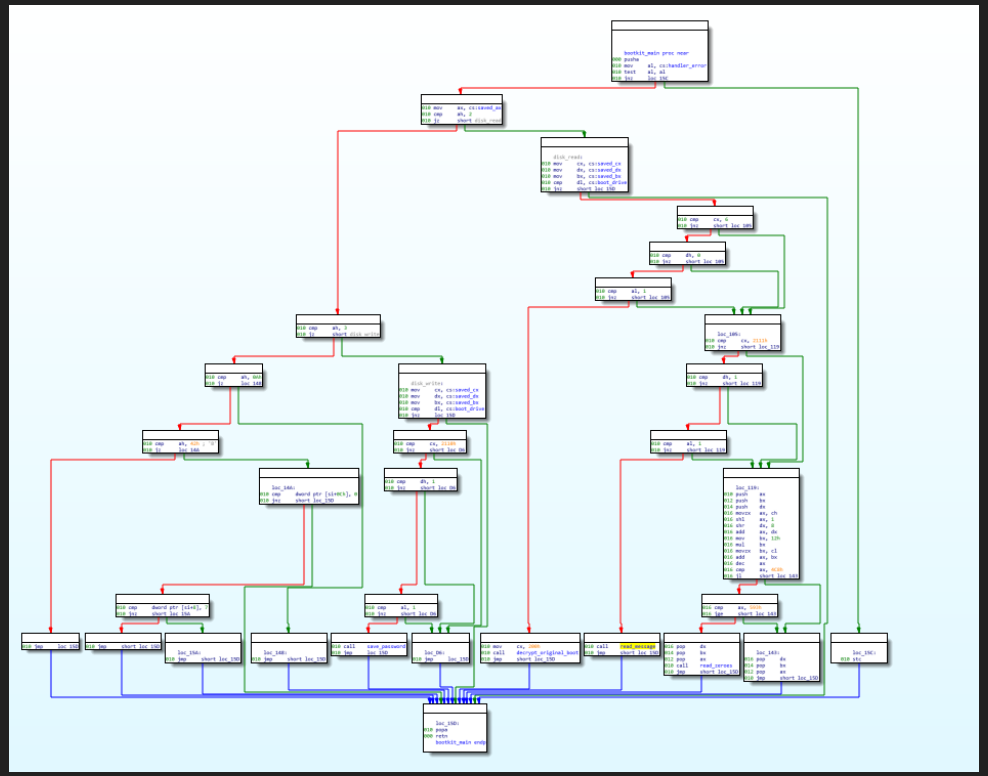
```
loc_801:                                ; DATA XREF: sub_1009ED4+36↓r
        push    es
        xor     ebx, ebx                ; DATA XREF: seg002:100A25C↓r
        mov     bx, ds:413h             ; BIOS data area: size of memory in KB
        sub     bx, 20h ; ' '
        mov     ds:413h, bx
        shl     bx, 6
        mov     ds:66F3h, bx
        mov     es, bx
        mov     si, 843h
        xor     di, di
        mov     cx, 5EB0h
        cld
        rep movsb
        mov     eax, ds:4Ch
        mov     es:5E5Ah, eax
        mov     es:5E6Fh, dl
        mov     ds:200h, eax
        mov     word ptr ds:4Ch, 12h
        mov     word ptr ds:4Eh, es
        pop     es
        popa
        retn
```

> Reserve RAM

> Copy payload

> Install as int 13h handler

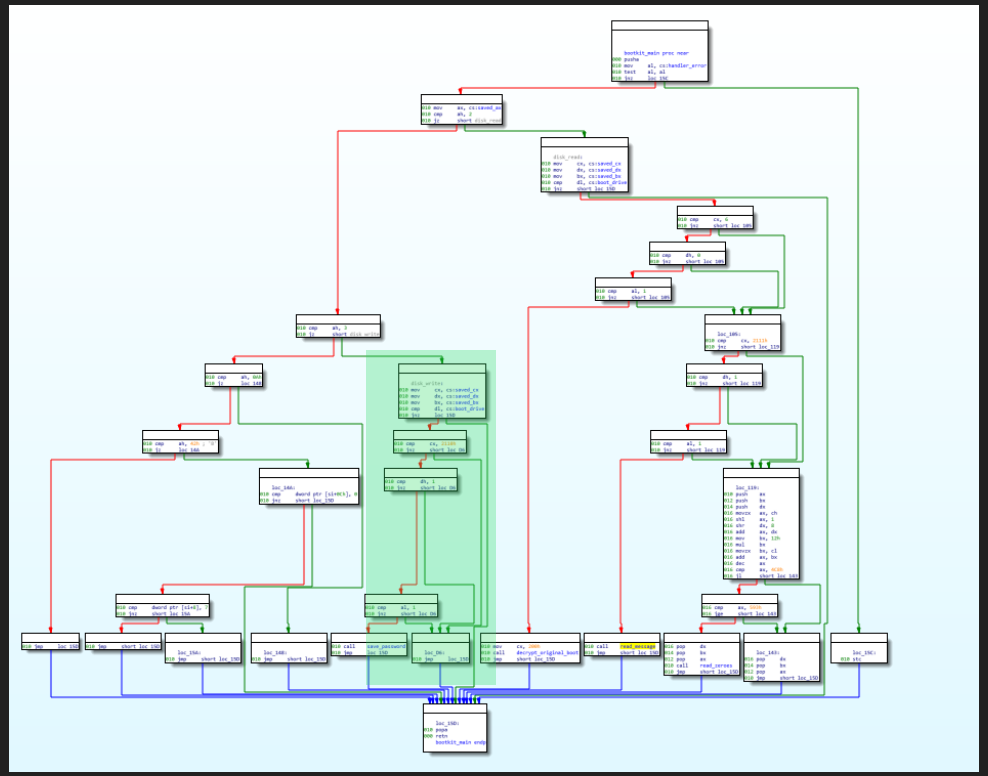
# The interrupt handler



# The interrupt handler

Disk write

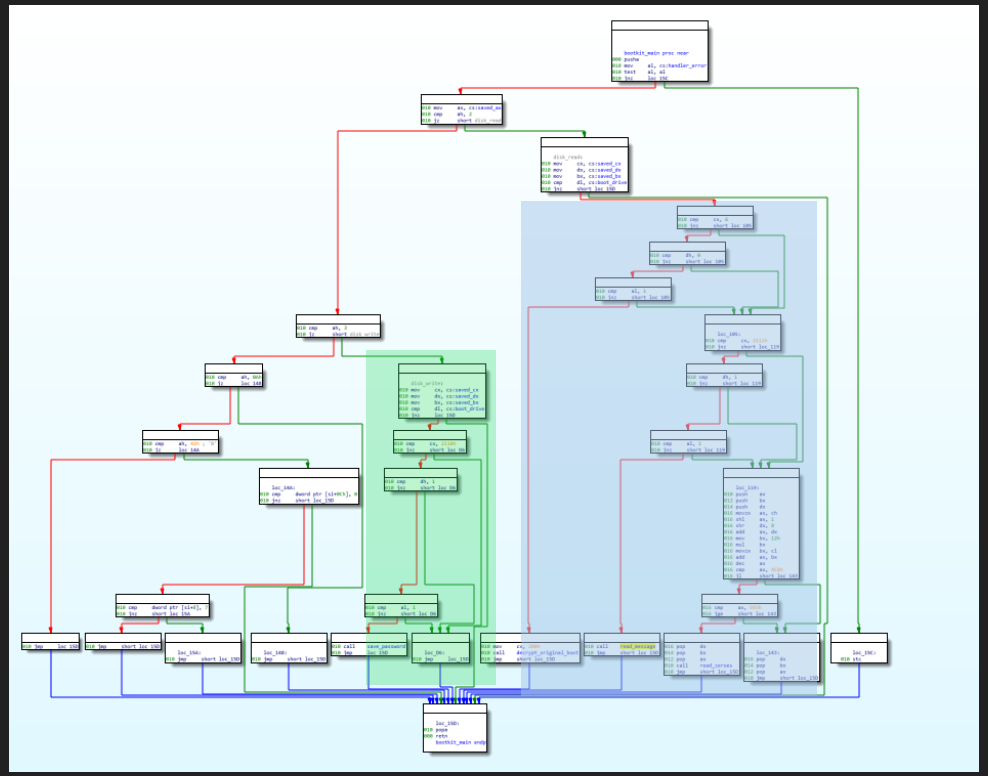
> Save key



# The interrupt handler

Disk write

> Save key

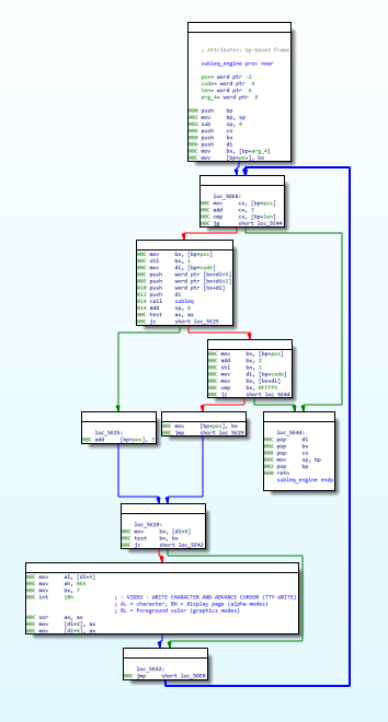


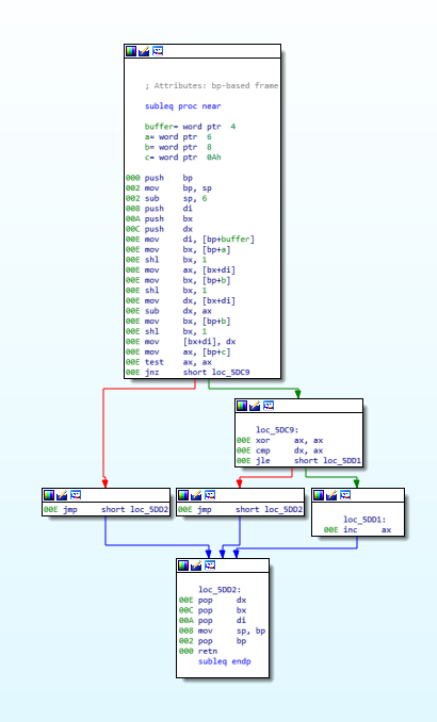
Disk read

> Hide from OS

> Check flag

# Password checker






Two small functions

Looks easy right?

# Subleq

subleq A, B, C

One Instruction Set Computer

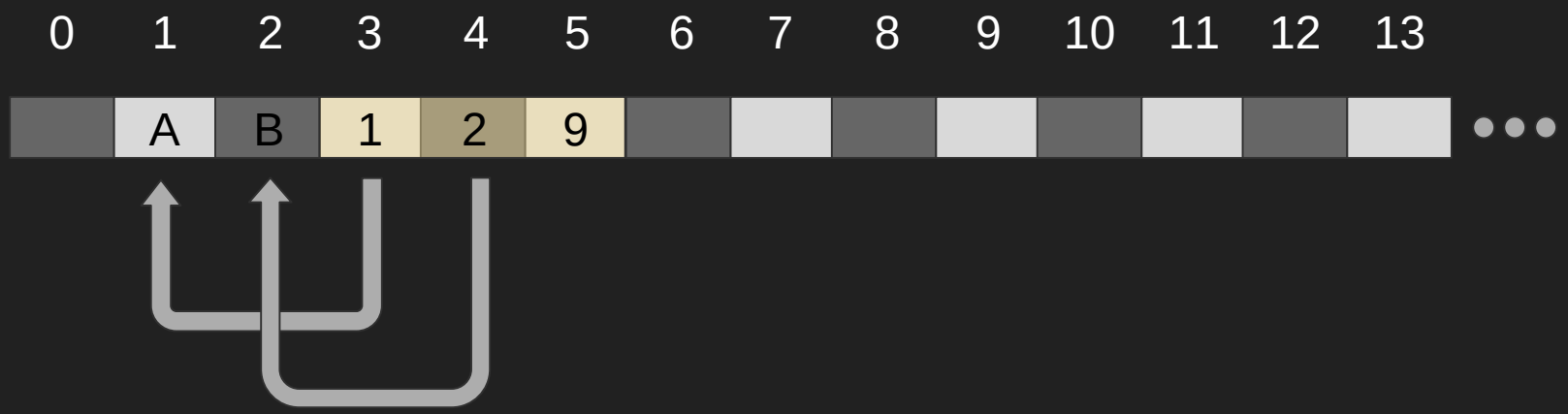
Subtract A from B and jump to C if result is less than or equal to zero

It’s turing complete!

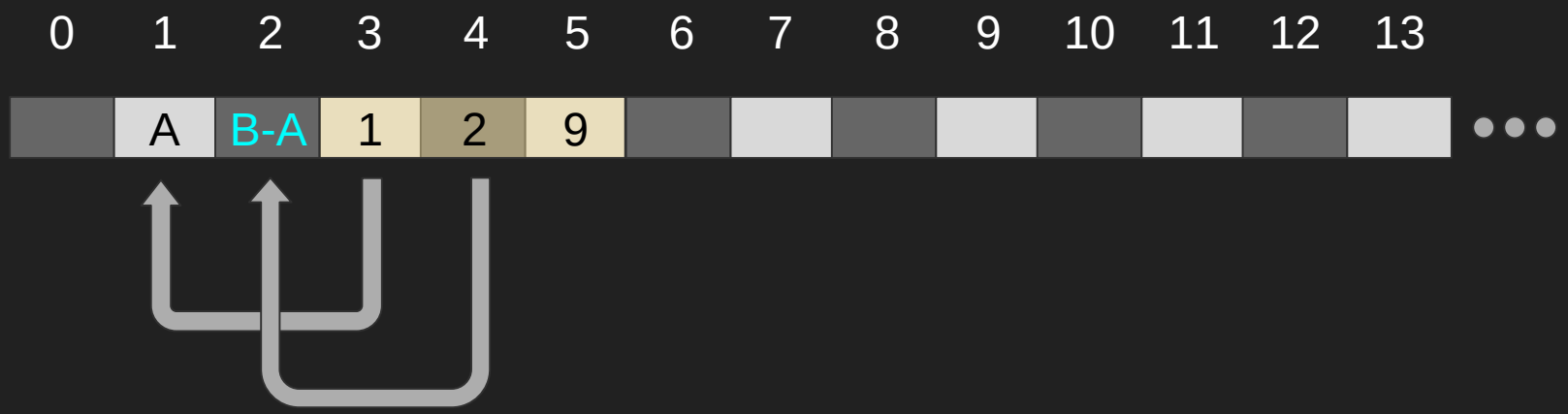
# Subleq

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

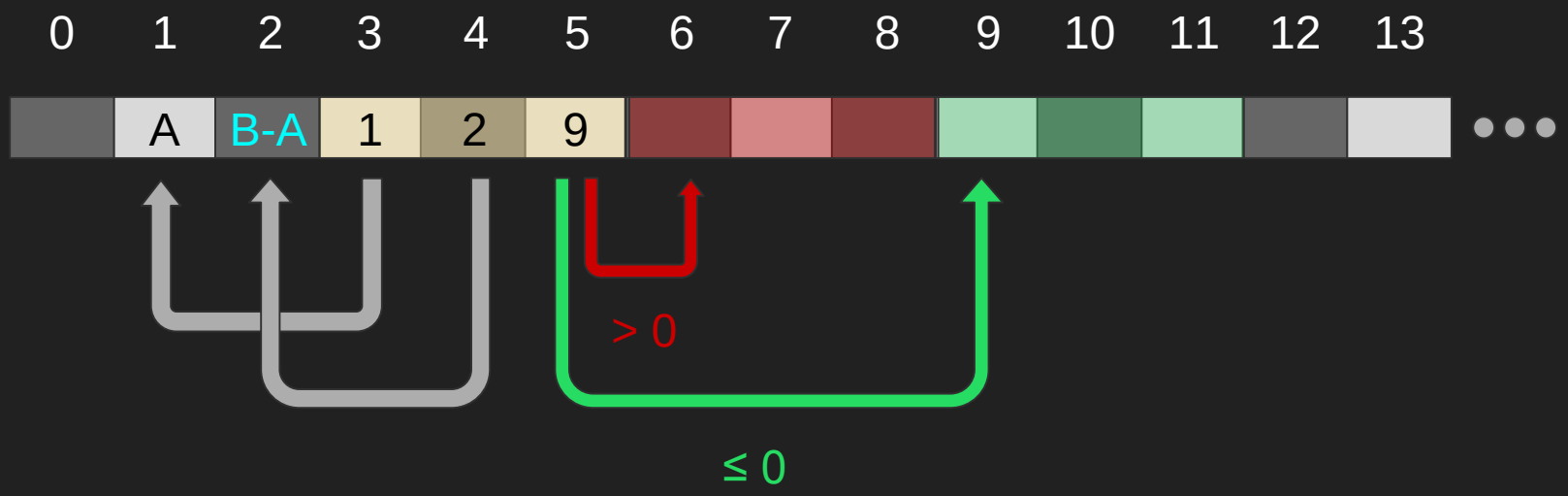
# Subleq



# Subleq



# Subleq



# Subleq: higher level instructions

| | |
|---|---|
| subleq A, A, 0 | memory[A] = 0 |
| subleq 0, 0, ADDR | jump to ADDR |
| subleq A, 0, 0<br>subleq 0, B, 0<br>subleq 0, 0, 0 | memory[B] += memory[A] |
| subleq B, B, 0<br>subleq A, 0, 0<br>subleq 0, B, 0<br>subleq 0, 0, 0 | memory[B] = memory[A] |

# RSSB

rssb A

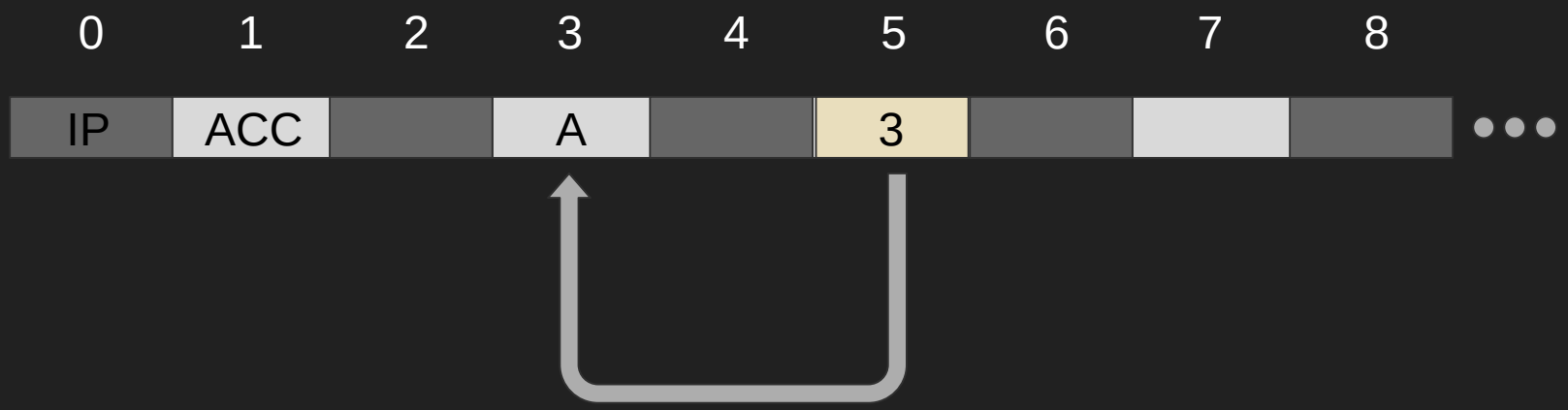
Reverse subtract and skip if borrow

Uses an accumulator register ACC

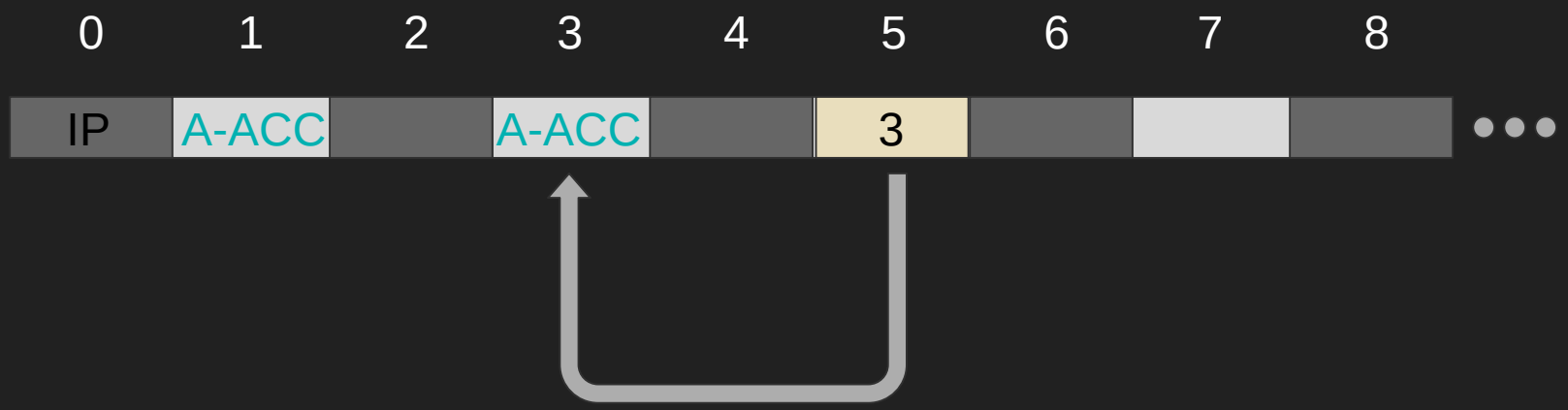
Subtract ACC from A and skip the next instruction if the result is negative

Turing complete, but even more limited

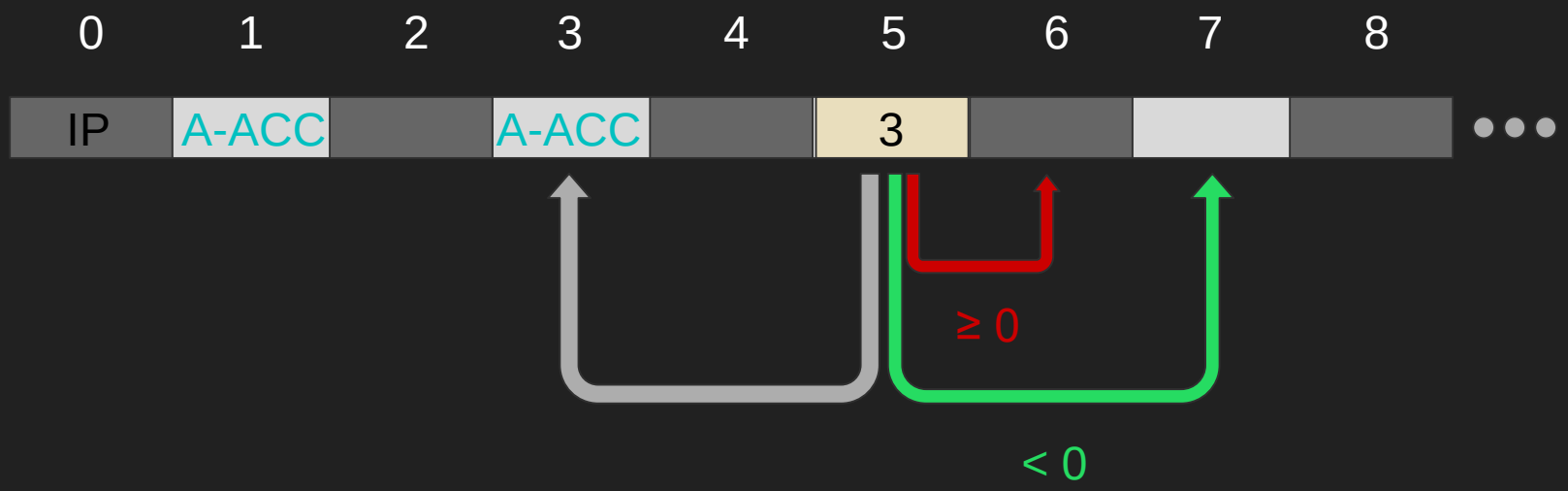
# RSSB



# RSSB



# RSSB



# Making it readable

After a lot of bad code...

```
def parse(self, a):
    if a[0].dest == a[20].ip and a[1].dest == a[18].ip and a[2].op == Vars.acc and a[4].op == Vars.acc:
        if a[5].op == a[20].ip and a[6].src == a[5].op and a[6].dest == a[18].ip and a[7].op == Vars.acc:
            if a[8].src == a[22].ip and a[8].dest == a[23].ip and a[9].op == Vars.acc:
                if a[10].src == a[21].ip and a[10].dest == a[24].ip and a[11].op == Vars.acc:
                    if a[12].src in variables and data[a[12].src] == 1 and a[12].dest == a[18].ip:
                        if a[13].dest == Vars.ip and data[a[13].src] == 11 and a[25].dest == a[19].ip:
                            if a[25].val == 14 and a[25].src in variables and data[a[25].src] == 1:
                                if a[26].dest == Vars.ip and a[26].src == a[25].dest and a[27].dest == a[25].dest:
                                    if a[28].dest == Vars.ip and a[28].src == a[23].ip and a[29].dest == a[25].dest:
                                        if a[30].dest == Vars.ip and a[30].src == a[24].ip:
                                            self.ip = a[0].ip
                                            self.src = a[3].op
                                            self.dest = a[30].ip + a[20].op + 5
                                            self.other = a[28].ip + a[21].op + 19
                                            return True
    return False
```

# Making it readable: subleq

Relatively small, ~600 instructions

Reduced to 70 (~8.5 times less) lines then manually reversed

# Making it readable: subleq

Relatively small, ~600 instructions

Reduced to 70 (~8.5 times less) lines then manually reversed

It’s a RSSB vm

# Making it readable: rssb

~9000 instructions reduced to 230 lines (~39 times less!)

Many instructions are still quite long, time for a bit of manual reversing:

```
5746    RSSB acc
5747    var_2908 = var_2894;
5761    RSSB acc
5762    var_2908 *= 2;
5805    var_2908 *= 2;
5848    var_2908 *= 2;
5891    var_2908 *= 2;
5934    var_2908 *= 2;
5977    var_2908 +=
var_2894;
```

→

```
var_2908 = var_2894 * 33
```

# Making it readable: rssb's xor

```
Inputs: A, B

result = 0
for(i = 0; i < 16; i++)
{
   tmpA = 0, tmpB = 0
   if(A < 0) tmpA = 1
   if(B < 0) tmpB = 1

   tmp = 0
   if(tmpA + tmpB == 1) tmp = 1

   A = A * 2
   B = B * 2
   result = result * 2 + tmp
}
```

→

```
result = A ^ B
```

An XOR takes ~2000 instructions!

# Making it readable: flag checker

```
res = [64639, 62223, ...]

bool check(word *flag, word len)
{
    hash = sum(flag.split('@')[0]) + len(flag) * 3072

    for(int i = 0; i < 15; i++)
    {
                pair = (flag[2 * i + 1] - 32) * 128 + (flag[2 * i] - 32)
                idx = i * 33
                if((idx ^ pair) + hash == res[i]) total -= i
    }

    return total == -105
}
```

# Making it readable: flag checker

Invert the algorithm (bruteforce the hash) and get the flag:

Av0cad0_Love_2018@flare-on.com

# Thank you!

Questions?

Full subleq/rssb analysis code is available at
https://github.com/dp1/flareon5-12