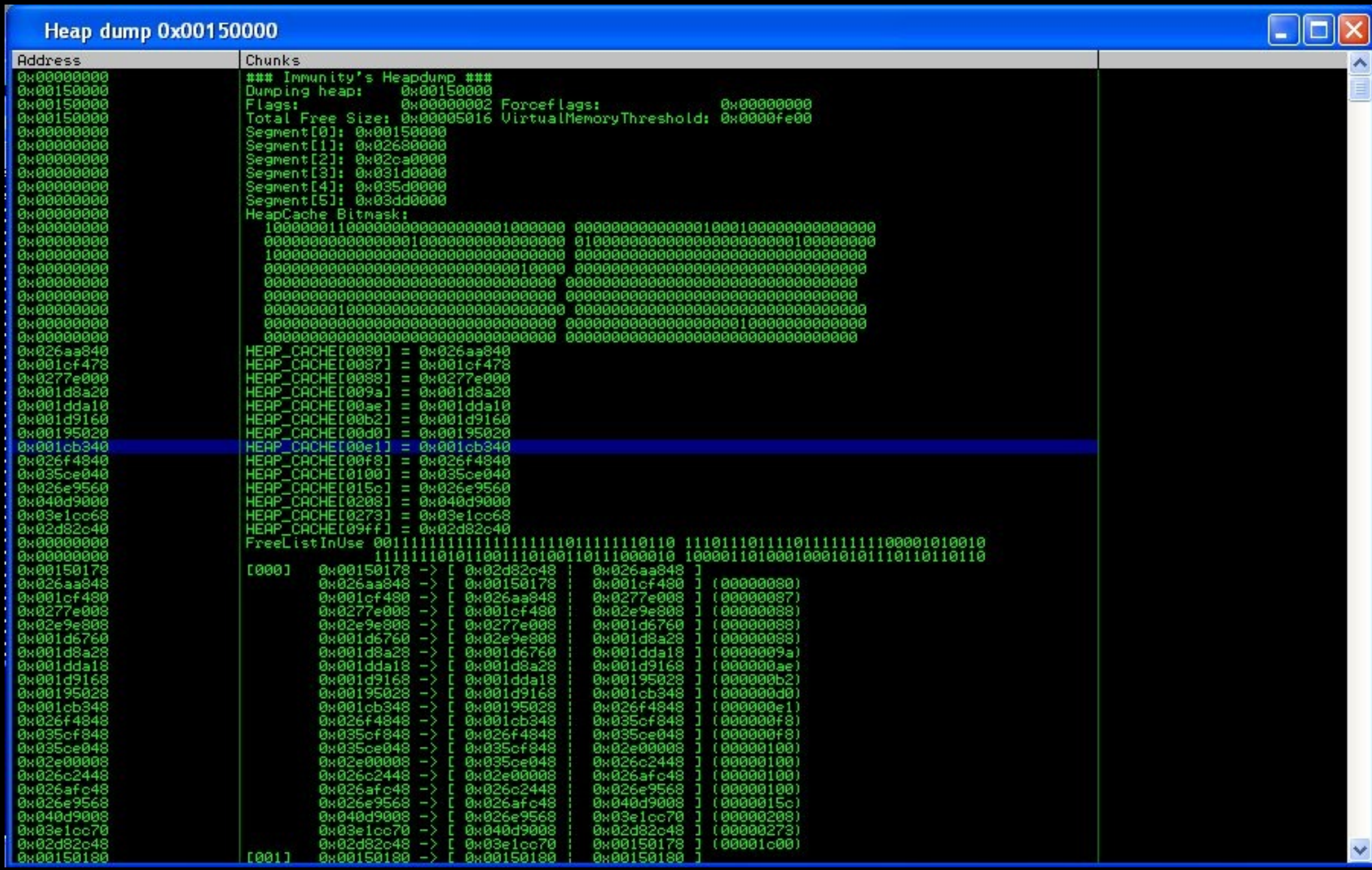
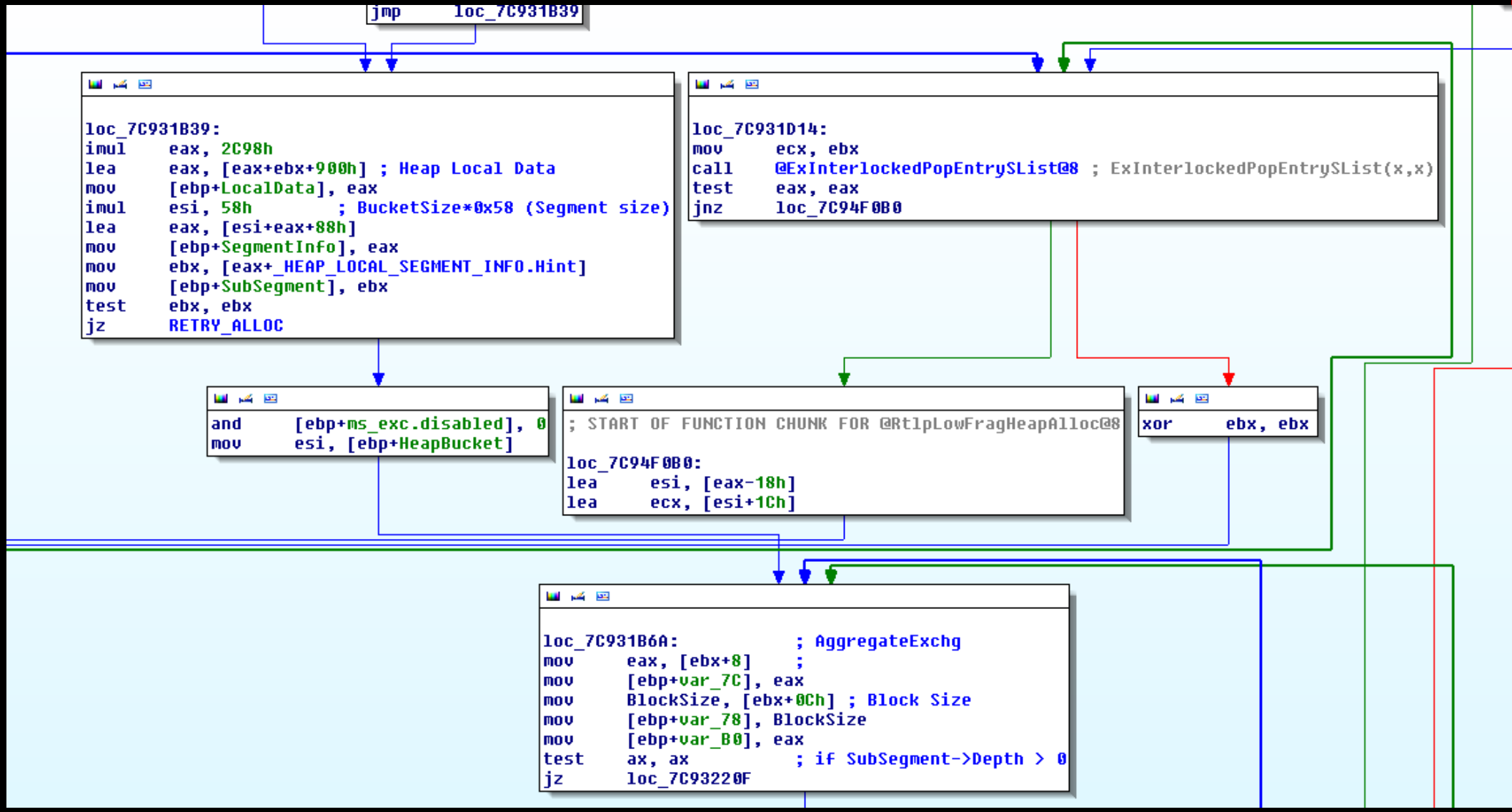
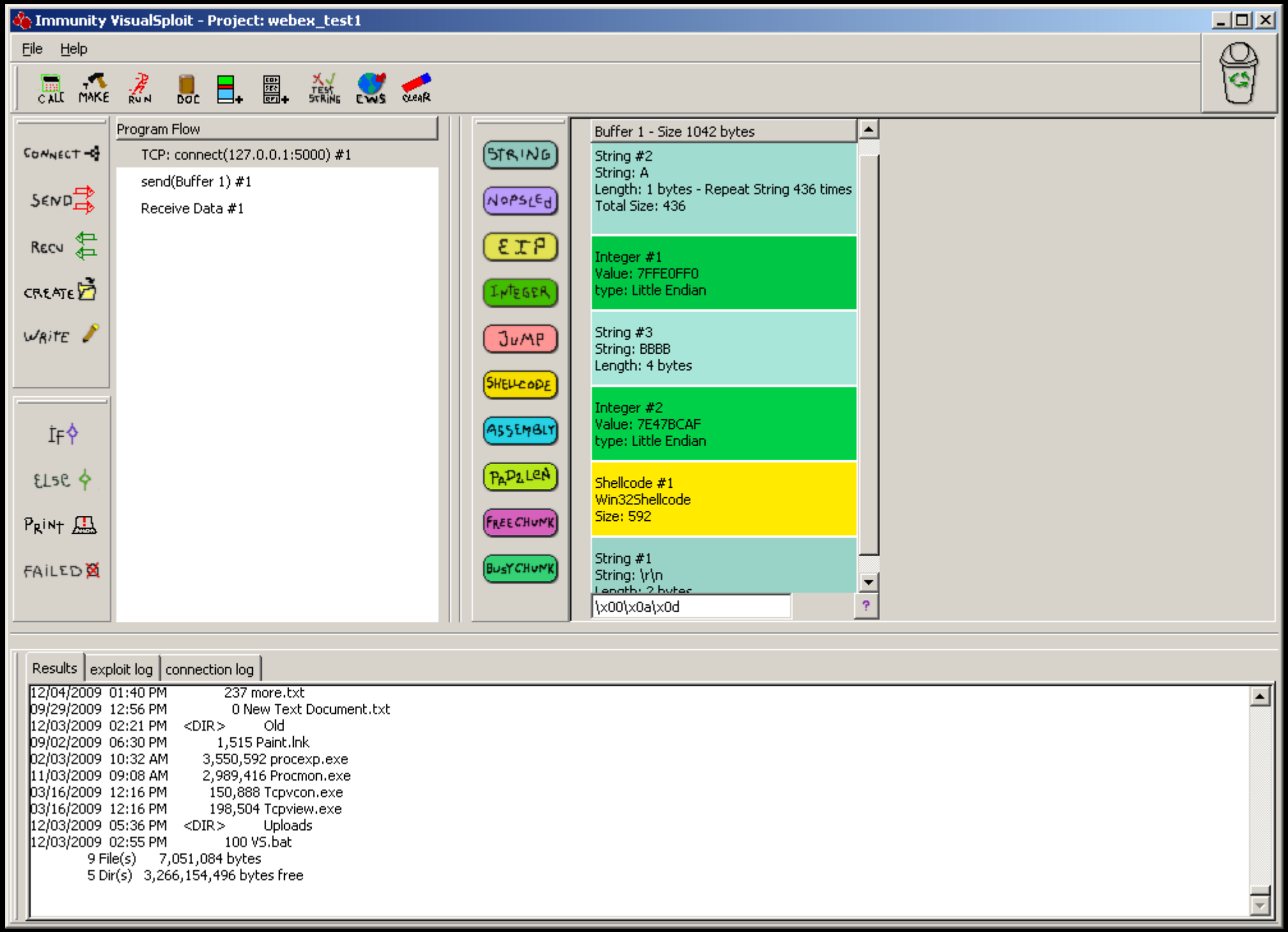
Strategic Surprise

My name is Nico Waisman
and I have an obsession

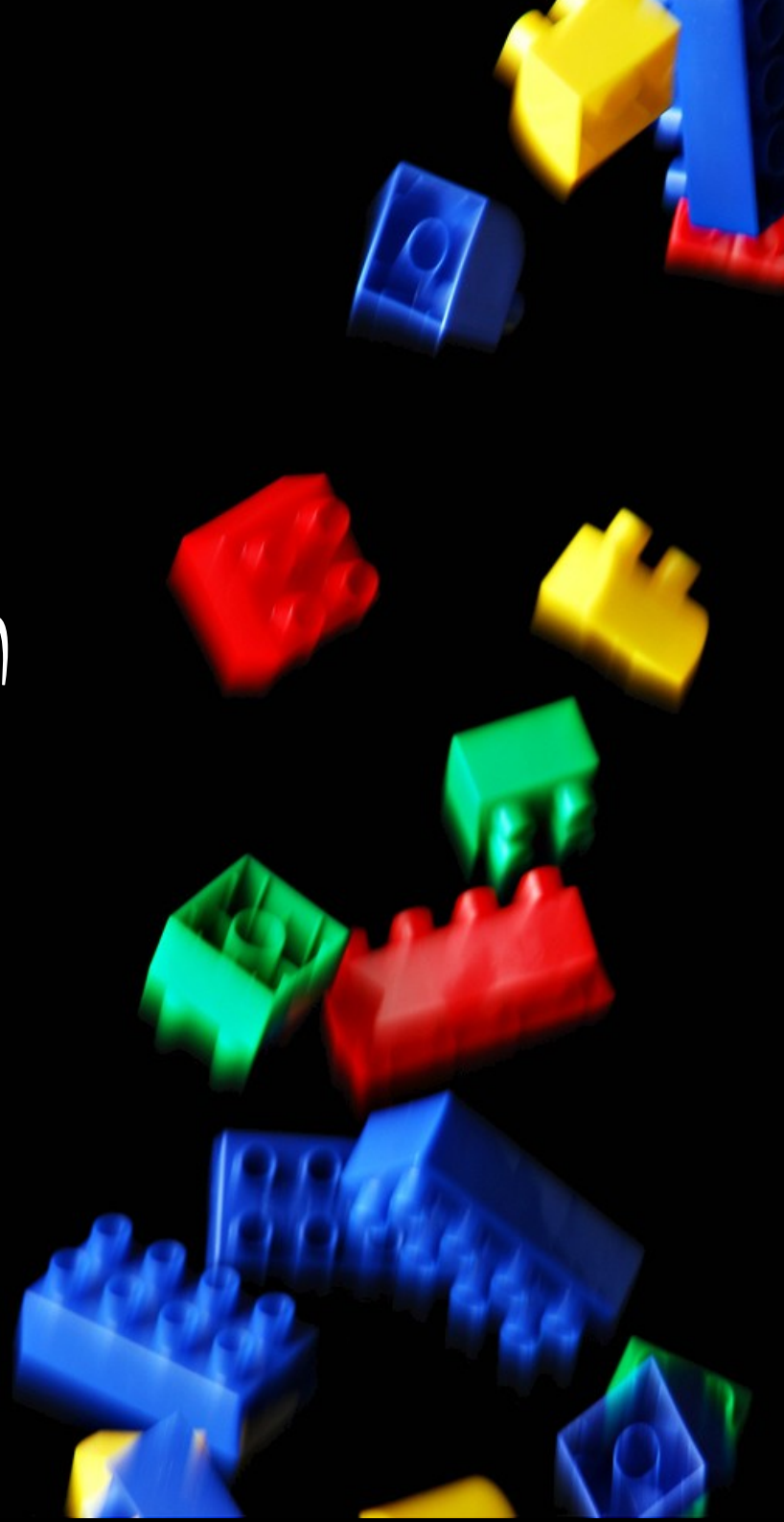
The Past

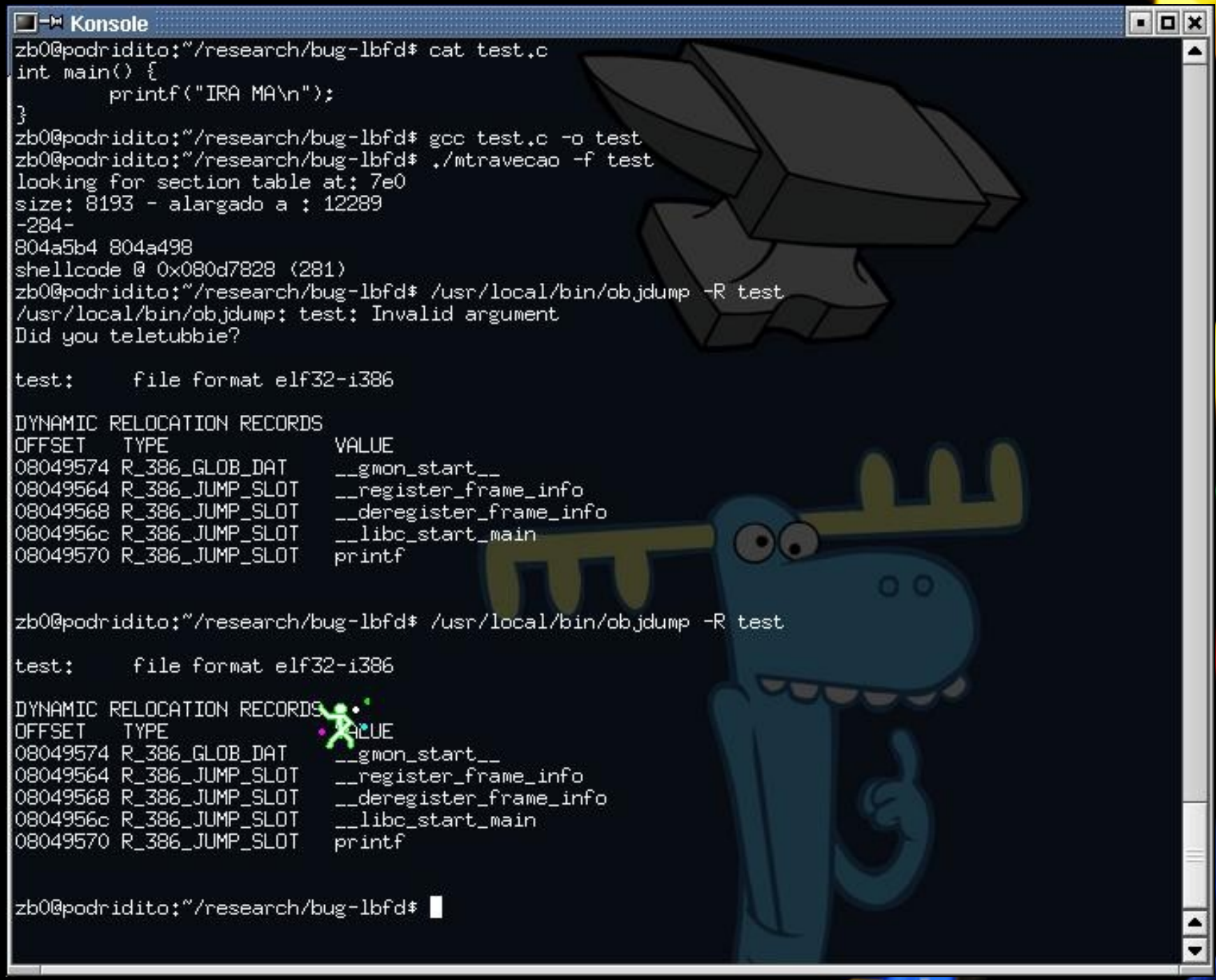

Heap dump 0x00150000
Address
Chunks
0x00000000 ### Immunity's Heapdump ###
0x00150000 Dumping heap: 0x00150000
0x00150000 Flags: 0x00000002 Forceflags: 0x00000000
0x00150000 Total Free Size: 0x00005016 VirtualMemoryThreshold: 0x0000fe00
0x00000000 Segment[0]: 0x00150000
0x00000000 Segment[1]: 0x02680000
0x00000000 Segment[2]: 0x02ca0000
0x00000000 Segment[3]: 0x031d0000
0x00000000 Segment[4]: 0x035d0000
0x00000000 Segment[5]: 0x03dd0000
0x00000000 HeapCache Bitmask:
0x00000000 10000001100000000000000000001000000 00000000000000010001000000000000
0x00000000 00000000000000000100000000000000000 01000000000000000000000000100000000
0x00000000 10000000000000000000000000000000000 00000000000000000000000000000000000
0x00000000 00000000000000000000000000010000000 00000000000000000000000000000000000
0x00000000 00000000000000000000000000000000000 00000000000000000000000000000000000
0x00000000 00000000000000000000000000000000000 00000000000000000000000000000000000
0x00000000 00000000100000000000000000000000000 00000000000000000000000000000000000
0x00000000 00000000000000000000000000000000000 00000000000000000000100000000000000
0x00000000 00000000000000000000000000000000000 00000000000000000000000000000000000
0x026aa840 HEAP_CACHE[0080] = 0x026aa840
0x001cf478 HEAP_CACHE[0087] = 0x001cf478
0x0277e000 HEAP_CACHE[0088] = 0x0277e000
0x001d8a20 HEAP_CACHE[009a] = 0x001d8a20
0x001dda10 HEAP_CACHE[00ae] = 0x001dda10
0x001d9160 HEAP_CACHE[00b2] = 0x001d9160
0x00195020 HEAP_CACHE[00d0] = 0x00195020
0x001cb340 HEAP_CACHE[00e1] = 0x001cb340
0x026f4840 HEAP_CACHE[00f8] = 0x026f4840
0x035ce040 HEAP_CACHE[0100] = 0x035ce040
0x026e9560 HEAP_CACHE[015c] = 0x026e9560
0x040d9000 HEAP_CACHE[0208] = 0x040d9000
0x03e1cc68 HEAP_CACHE[0273] = 0x03e1cc68
0x02d82c40 HEAP_CACHE[09ff] = 0x02d82c40
0x00000000 FreeListInUse 00111111111111111111101111111110110 11101110111101111111110000101010
0x00000000 11111110101100111010011011000010 10000110100010001010111011011011
0x00150178 [000] 0x00150178 -> [0x02d82c48 | 0x026aa848]
0x026aa848 0x026aa848 -> [0x00150178 | 0x001cf480] (00000080)
0x001cf480 0x001cf480 -> [0x026aa848 | 0x0277e008] (00000087)
0x0277e008 0x0277e008 -> [0x001cf480 | 0x02e9e808] (00000088)
0x02e9e808 0x02e9e808 -> [0x0277e008 | 0x001d6760] (00000088)
0x001d6760 0x001d6760 -> [0x02e9e808 | 0x001d8a28] (00000088)
0x001d8a28 0x001d8a28 -> [0x001d6760 | 0x001dda18] (0000009a)
0x001dda18 0x001dda18 -> [0x001d8a28 | 0x001d9168] (000000ae)
0x001d9168 0x001d9168 -> [0x001dda18 | 0x00195028] (000000b2)
0x00195028 0x00195028 -> [0x001d9168 | 0x001cb348] (000000d0)
0x001cb348 0x001cb348 -> [0x00195028 | 0x026f4848] (000000e1)
0x026f4848 0x026f4848 -> [0x001cb348 | 0x035cf848] (000000f8)
0x035cf848 0x035cf848 -> [0x026f4848 | 0x035ce048] (000000f8)
0x035ce048 0x035ce048 -> [0x035cf848 | 0x02e00008] (00000100)
0x02e00008 0x02e00008 -> [0x035ce048 | 0x026c2448] (00000100)
0x026c2448 0x026c2448 -> [0x02e00008 | 0x026afc48] (00000100)
0x026afc48 0x026afc48 -> [0x026c2448 | 0x026e9568] (00000100)
0x026e9568 0x026e9568 -> [0x026afc48 | 0x040d9008] (0000015c)
0x040d9008 0x040d9008 -> [0x026e9568 | 0x03e1cc70] (00000208)
0x03e1cc70 0x03e1cc70 -> [0x040d9008 | 0x02d82c48] (00000273)
0x02d82c48 0x02d82c48 -> [0x03e1cc70 | 0x00150178] (00001c00)
0x00150180 [001] 0x00150180 -> [0x00150180 | 0x00150180]


jmp loc_7C931B39
loc_7C931B39:
imul eax, 2C98h
lea eax, [eax+ebx+900h] ; Heap Local Data
mov [ebp+LocalData], eax
imul esi, 58h ; BucketSize*0x58 (Segment size)
lea eax, [esi+eax+88h]
mov [ebp+SegmentInfo], eax
mov ebx, [eax+_HEAP_LOCAL_SEGMENT_INFO.Hint]
mov [ebp+SubSegment], ebx
test ebx, ebx
jz RETRY_ALLOC
loc_7C931D14:
mov ecx, ebx
call @ExInterlockedPopEntrySList@8 ; ExInterlockedPopEntrySList(x,x)
test eax, eax
jnz loc_7C94F0B0
and [ebp+ms_exc.disabled], 0
mov esi, [ebp+HeapBucket]
; START OF FUNCTION CHUNK FOR @RtlpLowFragHeapAlloc@8
loc_7C94F0B0:
lea esi, [eax-18h]
lea ecx, [esi+1Ch]
xor ebx, ebx
loc_7C931B6A: ; AggregateExchg
mov eax, [ebx+8] ;
mov [ebp+var_7C], eax
mov BlockSize, [ebx+0Ch] ; Block Size
mov [ebp+var_78], BlockSize
mov [ebp+var_B0], eax
test ax, ax ; if SubSegment->Depth > 0
jz loc_7C93220F


Immunity VisualSploit - Project: webex_test1
File
Help
CALC
MAKE
RUN
DOC
TEST STRING
EWS
CLEAR
CONNECT
SEND
RECV
CREATE
WRITE
IF
ELSE
PRINT
FAILED
Program Flow
TCP: connect(127.0.0.1:5000) #1
send(Buffer 1) #1
Receive Data #1
STRING
NOPSLED
EIP
INTEGER
JUMP
SHELLCODE
ASSEMBLY
PADLEN
FREECHUNK
BUSYCHUNK
Buffer 1 - Size 1042 bytes
String #2
String: A
Length: 1 bytes - Repeat String 436 times
Total Size: 436
Integer #1
Value: 7FFE0FF0
type: Little Endian
String #3
String: BBBB
Length: 4 bytes
Integer #2
Value: 7E47BCAF
type: Little Endian
Shellcode #1
Win32Shellcode
Size: 592
String #1
String: \r\n
\x00\x0a\x0d
Results
exploit log
connection log
12/04/2009 01:40 PM 237 more.txt
09/29/2009 12:56 PM 0 New Text Document.txt
12/03/2009 02:21 PM <DIR> Old
09/02/2009 06:30 PM 1,515 Paint.lnk
02/03/2009 10:32 AM 3,550,592 procexp.exe
11/03/2009 09:08 AM 2,989,416 Procmon.exe
03/16/2009 12:16 PM 150,888 Tcpvcon.exe
03/16/2009 12:16 PM 198,504 Tcpview.exe
12/03/2009 05:36 PM <DIR> Uploads
12/03/2009 02:55 PM 100 VS.bat
9 File(s) 7,051,084 bytes
5 Dir(s) 3,266,154,496 bytes free

Once upon a time in Mexico...

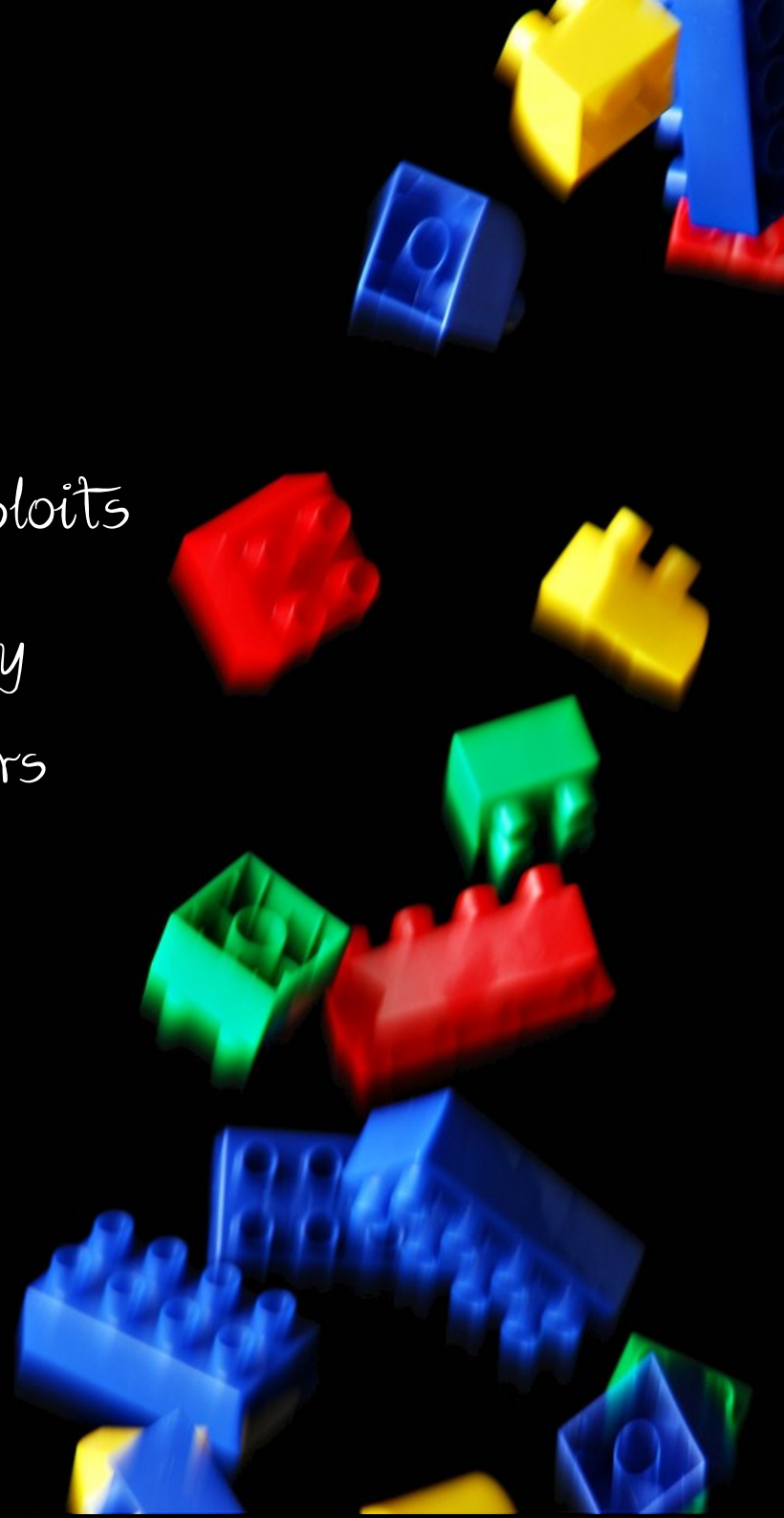



Konsole
zb0@podridito:~/research/bug-lbfd$ cat test.c
int main() {
printf("IRA MA\n");
}
zb0@podridito:~/research/bug-lbfd$ gcc test.c -o test
zb0@podridito:~/research/bug-lbfd$ ./mtravecao -f test
looking for section table at: 7e0
size: 8193 - alargado a : 12289
-284-
804a5b4 804a498
shellcode @ 0x080d7828 (281)
zb0@podridito:~/research/bug-lbfd$ /usr/local/bin/objdump -R test
/usr/local/bin/objdump: test: Invalid argument
Did you teletubbie?
test: file format elf32-i386
DYNAMIC RELOCATION RECORDS
OFFSET TYPE VALUE
08049574 R_386_GLOB_DAT __gmon_start__
08049564 R_386_JUMP_SLOT __register_frame_info
08049568 R_386_JUMP_SLOT __deregister_frame_info
0804956c R_386_JUMP_SLOT __libc_start_main
08049570 R_386_JUMP_SLOT printf
zb0@podridito:~/research/bug-lbfd$ /usr/local/bin/objdump -R test
test: file format elf32-i386
DYNAMIC RELOCATION RECORDS
OFFSET TYPE VALUE
08049574 R_386_GLOB_DAT __gmon_start__
08049564 R_386_JUMP_SLOT __register_frame_info
08049568 R_386_JUMP_SLOT __deregister_frame_info
0804956c R_386_JUMP_SLOT __libc_start_main
08049570 R_386_JUMP_SLOT printf
zb0@podridito:~/research/bug-lbfd$

Advance Dougs Lea's malloc exploits

Vudo - An object superstitiously believed to embody magical powers

Once upon a free()

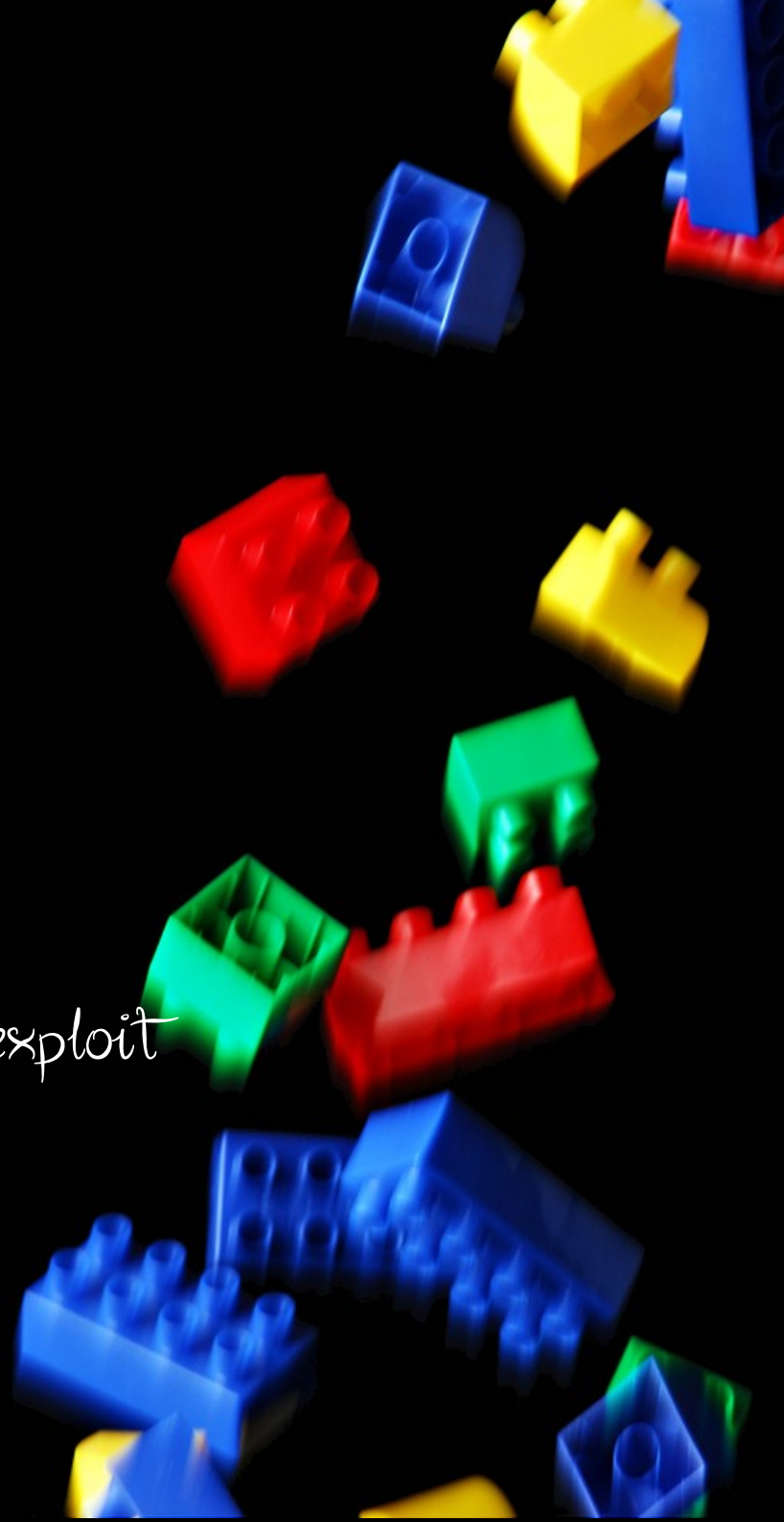


Wuftpd glob/site exec

Gobbles openssh exploit (FUCKYOUTHEO)

Pserverd - 4c1d61tch3z

Solar Designer Netscape JPEG exploit



Understand your exploitation domain

Reverse, Reverse and Reverse a little bit more

Debug, Debug and Debug a little bit more

Five Ws

The Present

00:27 < connection> I've been looking into starting to bindiff Patch Tuesday patches in order to get used to finding sploits just haven't had the time. Any recommendations on windows diffing tools?
00:27 <@jduck> save yourself the heartache and frustration, just kill yourself now
00:27 < connection> haha :P
00:28 < connection> on *nix systems I'm good with diff/patch I'm just ignorant to the windows equiv.
00:28 <@jduck> its nowhere near the same game
00:28 < connection> that drastic of a change eh?
00:29 <@hdm> you spend an hour just getting the patches extracted, another half and hour loading into ida, then umpteenthousandhours going through tens of thousands of changed functions based on the asm alone
00:30 <@hdm> and when you finally win, you only know what changed, not even how to trigger it
00:30 <@hdm> so you spend another umpteen hours trying to trace the code path to that changed bit of code
00:30 <@hdm> only to realize 2 days later it was the wrong change and the one youre looking at is unexploitable
00:30 <@hdm> then you kill yourself
00:30 <@hdm> so like jduck said, skip a step :P

Everyone's Uploads ▾ facepalm SEARCH **Full Text** | Tags Only
Advanced Search

Sort: **Relevant** | Recent | Interesting View: **Small** | Medium | Detail | Slideshow

From *facepalm*

From *facepalm*

From *facepalm*

From *facepalm*

From *facepalm*

From *facepalm*

From kassch


From *facepalm*


From *facepalm*


From *facepalm*

From *facepalm*


From *facepalm*


From *facepalm*


From *facepalm*


From *facepalm*


From *facepalm*


From *facepalm*


From *facepalm*


From *facepalm*

From *facepalm*


From *facepalm*

From *facepalm*


From *facepalm*

From *facepalm*

From *facepalm*

From *facepalm*


From *facepalm*


From mariomaster

Why is HDMoore Sad?

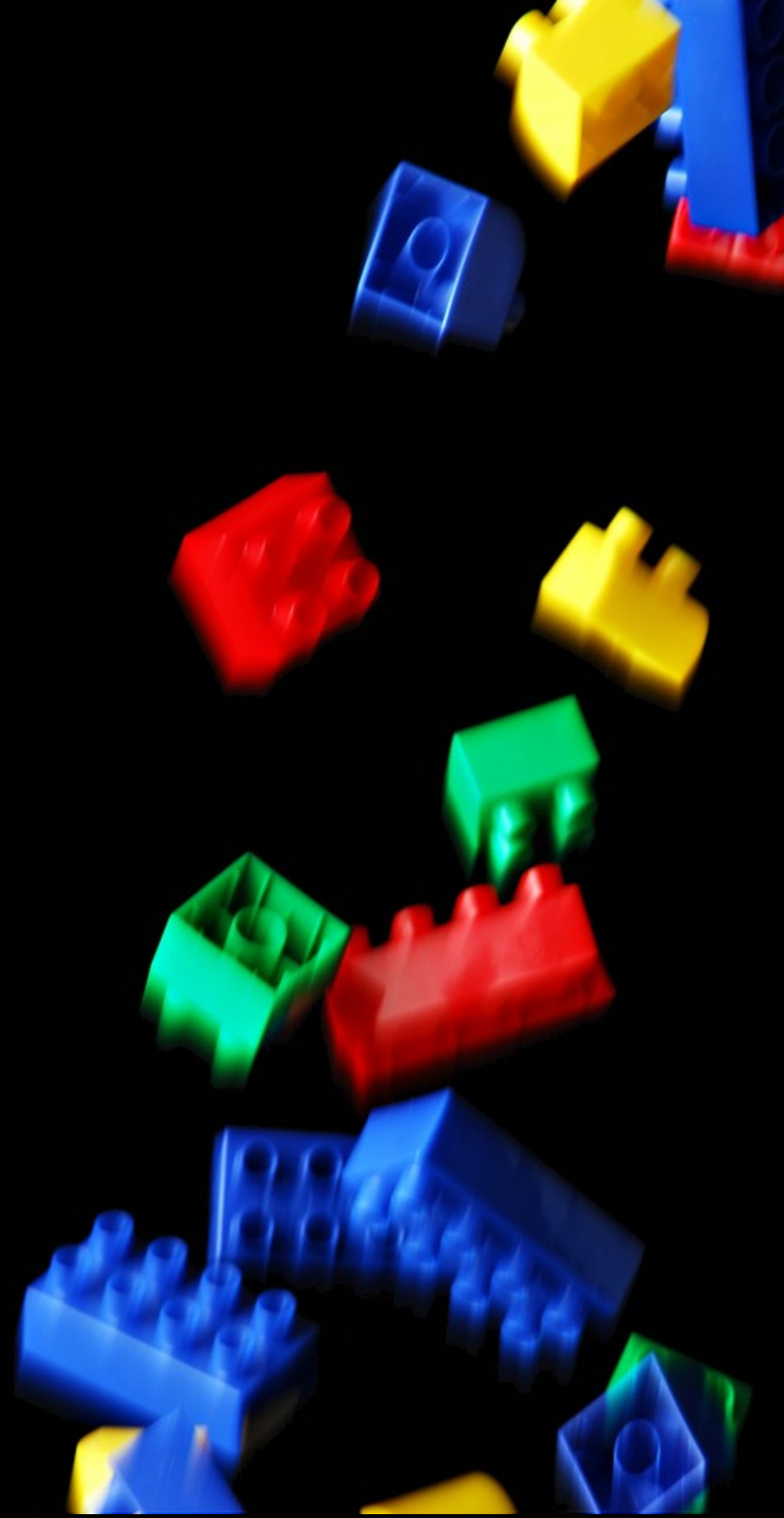
Disclaimer: The imagery used in this Slide may have been altered or modified to some degree from the original image

Exploits are hard...

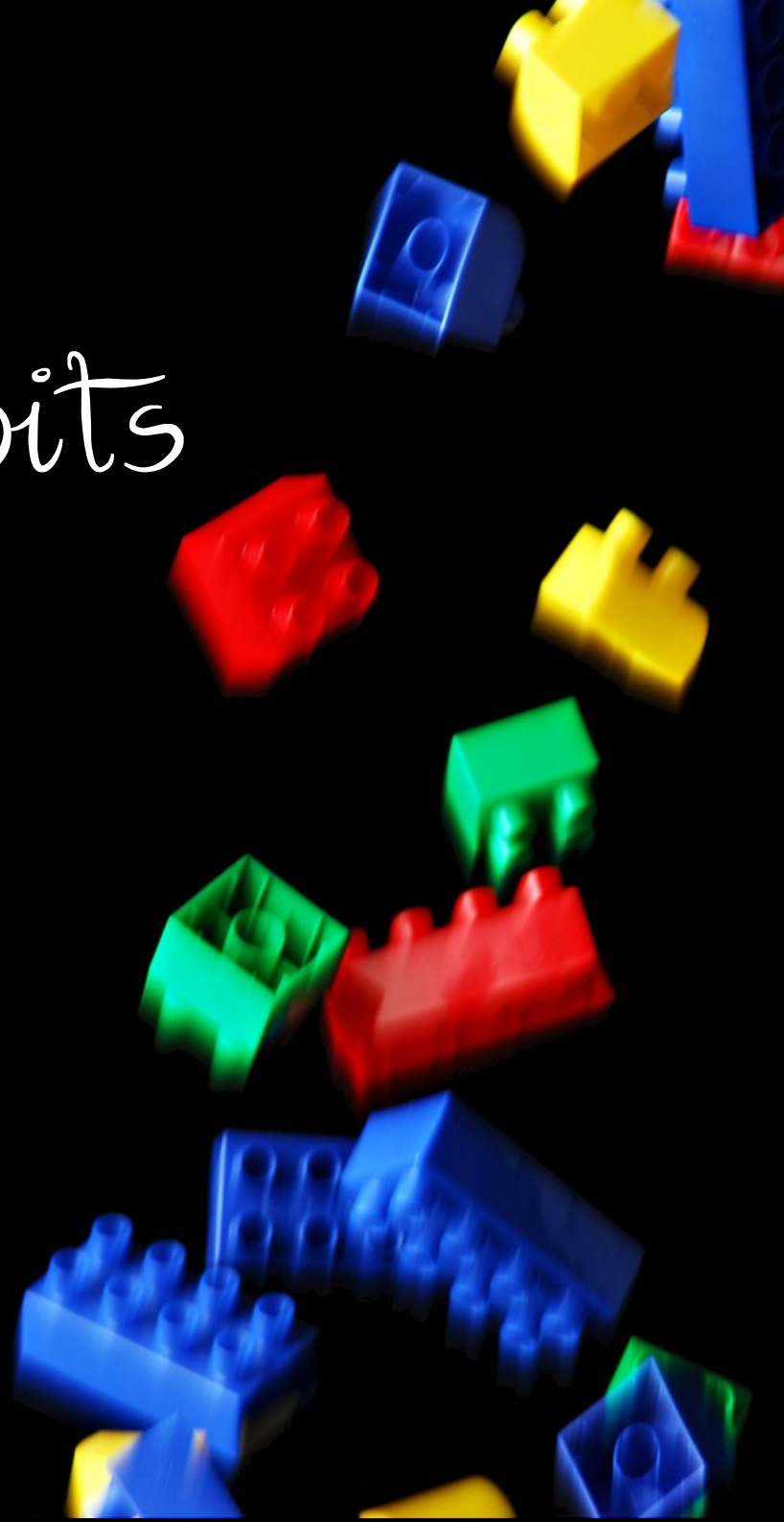
When was the last time you saw a real public exploit?


© AP
Gossip

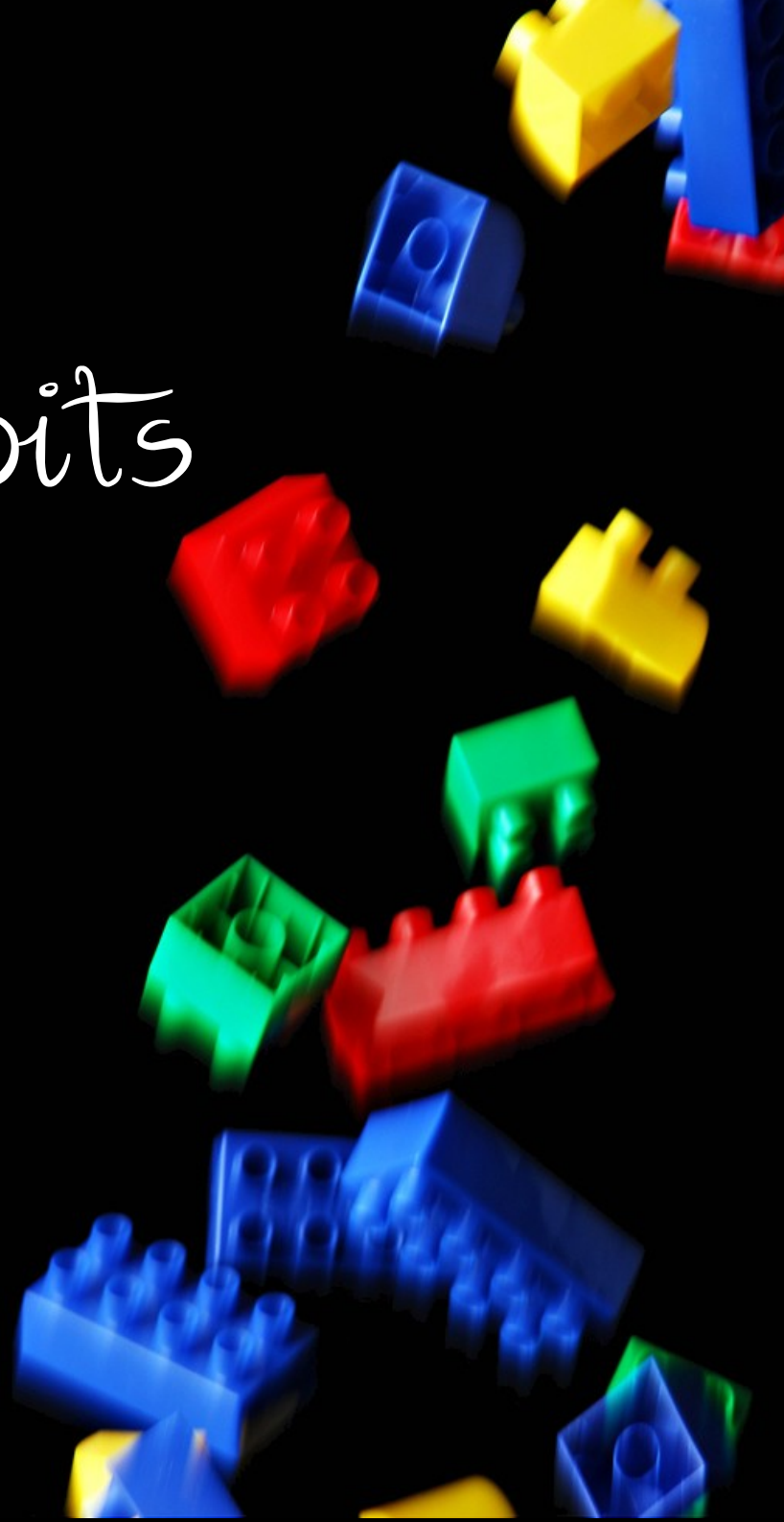
Bindiff Exploits

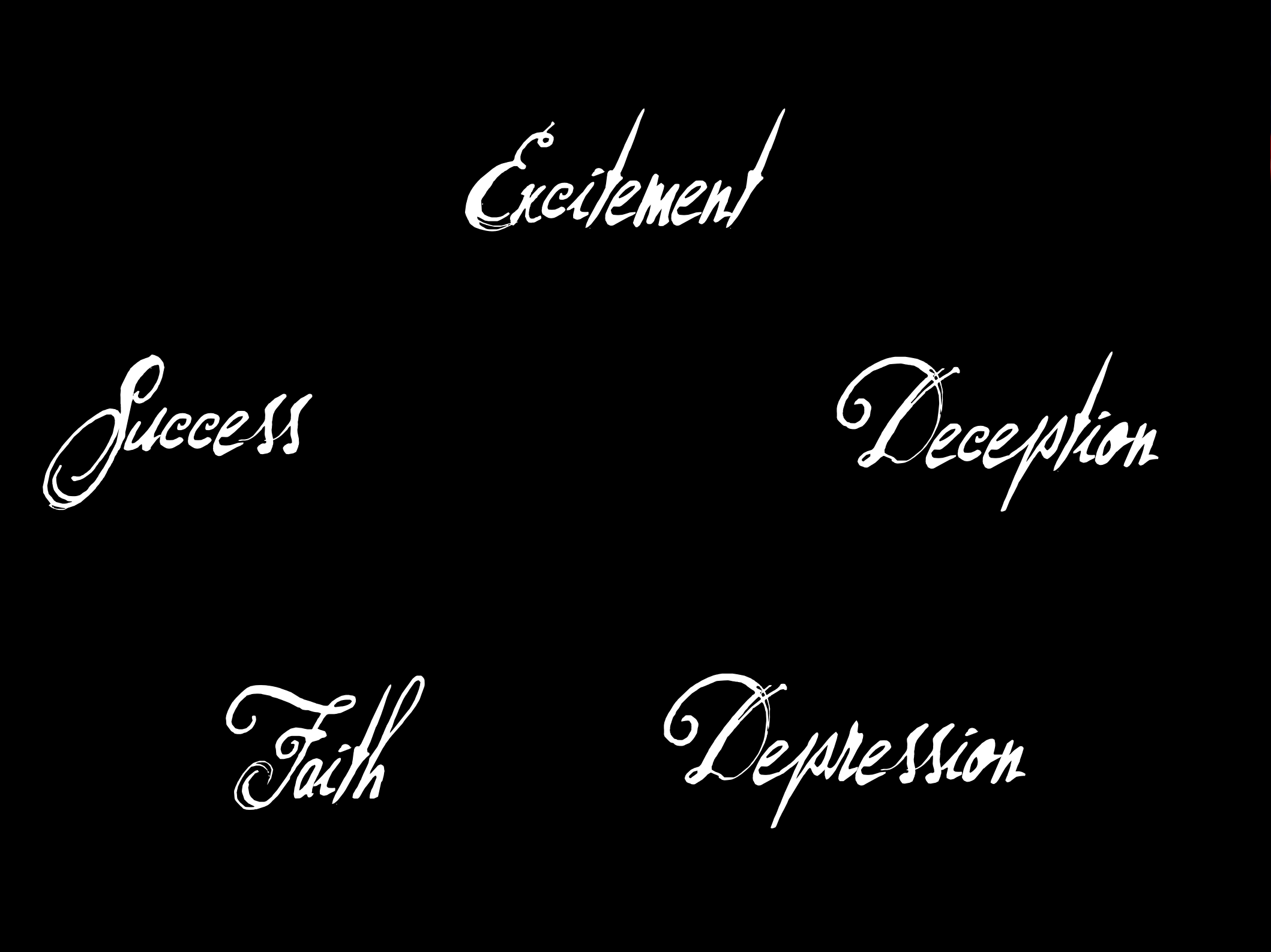


Post Mortem Exploits



Dry Humping Exploits




Excitement
Success
Deception
Faith
Depression

@0xcharlie
Charlie Miller

Reduced from high precision heap manipulation to just "trying stuff" in iPhone exploit. #somebugssuck

7 Mar via Echofon ☆ Favorite ↻ Undo Retweet ↩ Reply

Hope is not a

Business Plan

Exploits are hard...

...but it was always being


DEP
SafeSEH
ASLR
Code Security
Cookies
Metadata encryption

The Element of Surprise

Team vs Individuals

Researchers


You don't need a researcher,
you need a unicorn

Programmer

LAB

Management

Windows 2000

1d: Triggering the bug

2-4d: Understanding the heap layout

2-5d: Finding Soft and Hard Memleaks

5-8d : Finding a reliable Write4

1-2d: Function Pointers and Shellcode

Windows Vista

1 d: Triggering the bug

1-2d: Understanding the heap layout

2-5d: Finding Soft and Hard Memleaks

10-30d : Overwriting a the correct memory

2-5 days: Function pointer and Shellcode

Windows 2000

1d: Triggering the bug

2-4d: Understanding the heap layout

2-5d: Finding Soft and Hard Memleaks

5-8d : Finding a reliable Write4

1-2d: Function Pointers and Shellcode

Windows Vista

1 d: Triggering the bug

1-2d: Understanding the heap layout

2-5d: Finding Soft and Hard Memleaks

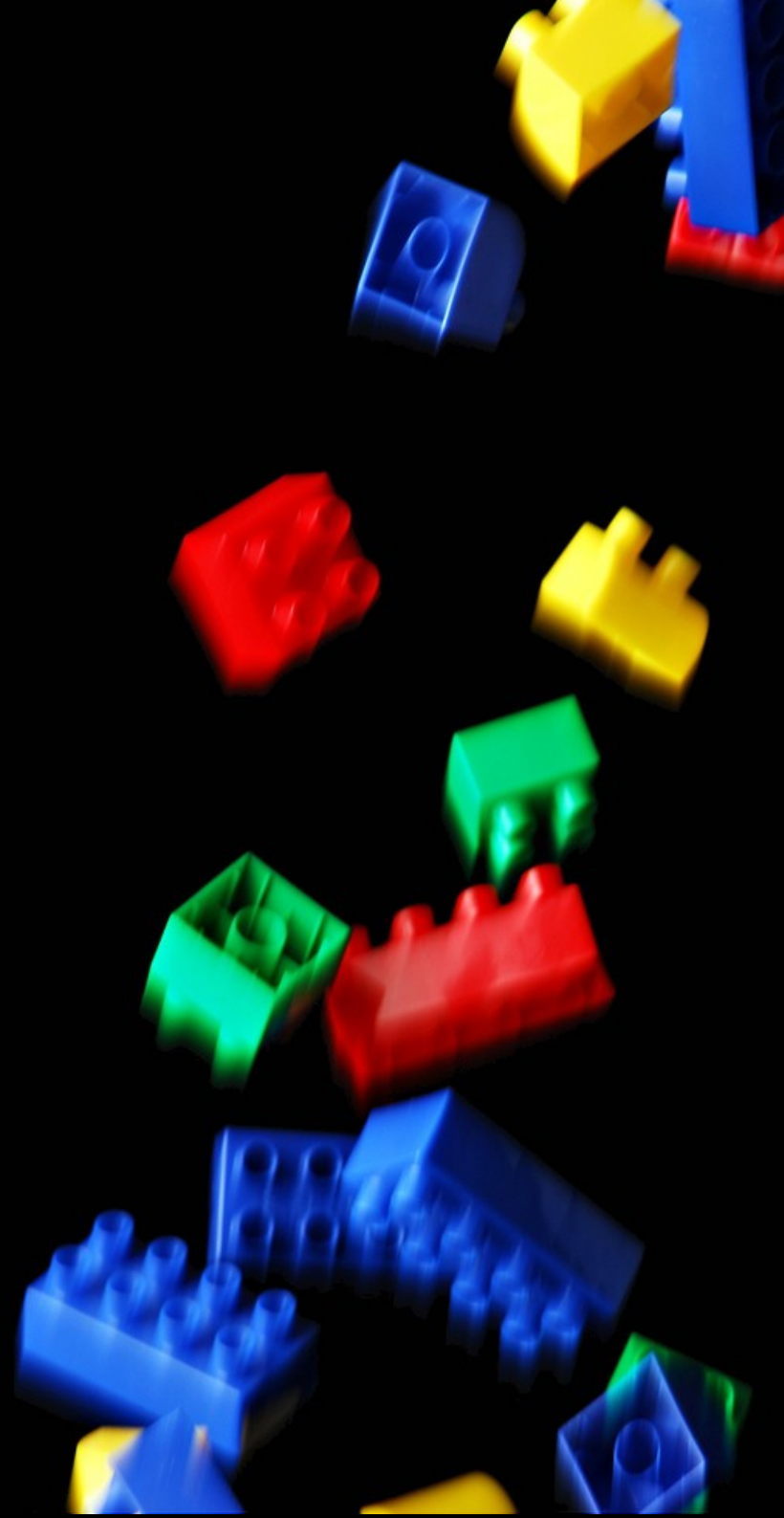
10-30d : Overwriting a the correct memory

2-5 days: Function pointer and Shellcode

Protection never target what we always aim for...

Bug classes die

Primitives dont



Nowadays exploitation techniques are crumbs of the '90 great banquet

There are No Surprises

Questions?

nico@immunityinc.com

@nicowaisman