Lecture 40 - Software Security Defenses

CprE 308

April 16, 2015

Software Security Defenses

Topics

This lecture:

- Review Buffer Overflow Concepts
- Buffer Overflow (Practical Example)
  - Reversing and remotely exploiting a Windows web server
- Operating system defenses
  - Stack canaries, DEP, ASLR
- Quiz
- Feedback Forms

Review

The Vulnerable Source Code

Why is this vulnerable?

```
#include <stdio.h>
int main(int argc, char **argv) {
  char buf[64];
  strcpy(buf, argv[1]);
}
```

Why is this Vulnerable?

- Program is soliciting input from the user through the program arguments
- Input is stored to memory (buf)
- Input bounds are not checked and data in memory can be overwritten
- The main function has a return address that can be overwritten to point to data in the buffer

Exploit

Exploit = [Payload] + JMP logic]

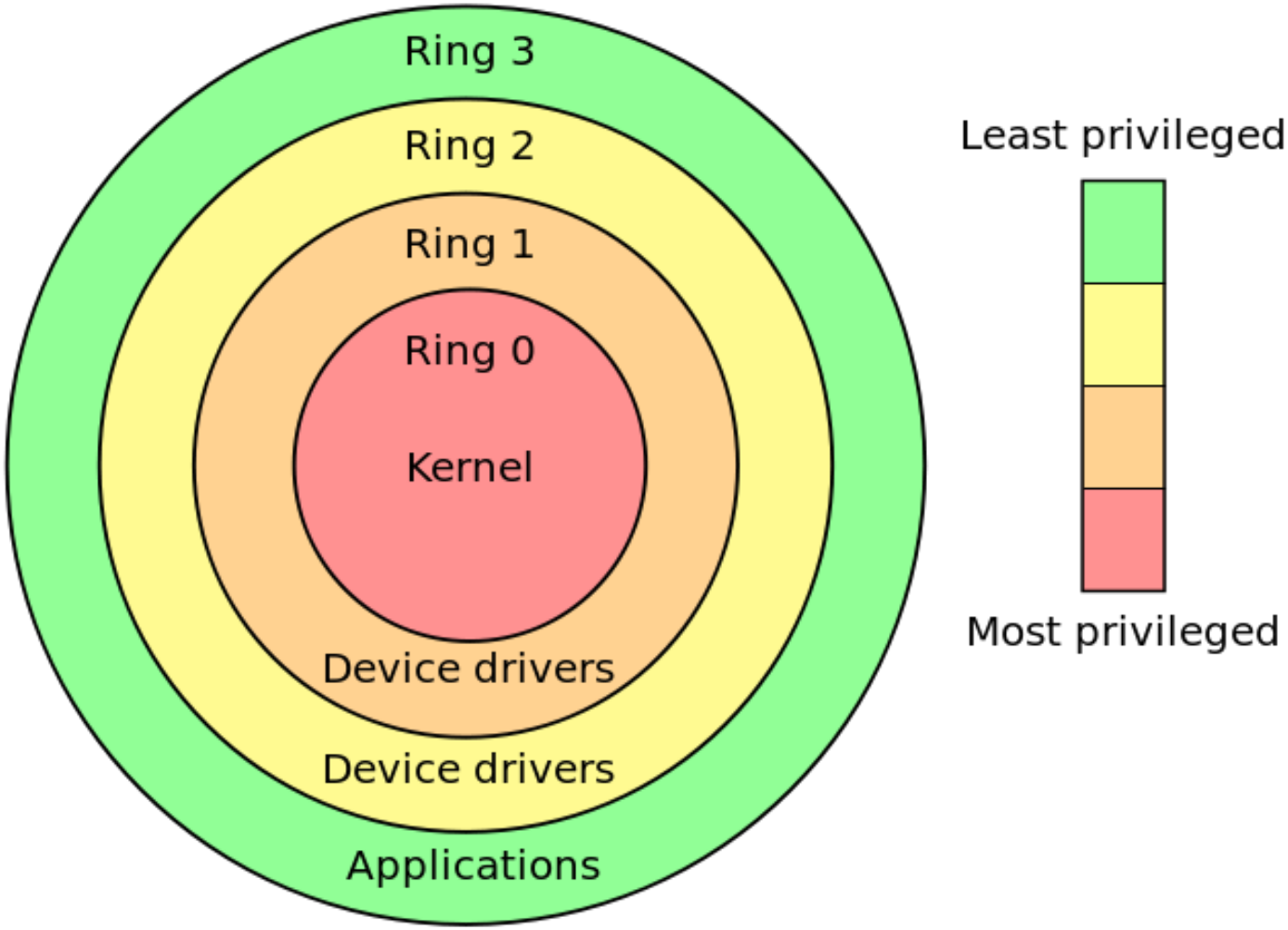
Payload = [NOP Sled + Shellcode]

- Payload is the length of the buffer
  - sizeof(NOP Sled) = sizeof(buffer) minus sizeof(shellcode)
  - CPU executes NOP's until it hits shellcode
- JMP logic is what extends past the bounds of the buffer and overwrites the stack pointer (to point back into the buffer containing shellcode)
  - May be able to overwrite/control other registers besides the stack pointer to take control

Exploit Privilege

- Shellcode runs at the permission level of the user that ran the program
  - What if the vulnerable code was in the Kernel?

Exploit Privilege



Exploit Privilege

- Could there be a Ring -1?
  - Bios, hardware, virtual machine host

Typical Payloads (Post Exploitation)

- Stager code that downloads and runs more malicious code
  - Usually needed if the buffer is too small to hold all the exploit code
- Add a user account
- Patch vulnerability + install a protected backdoor
- Keylogger, network monitor, etc.

Secure Coding

Practice defensive coding and check your inputs!

```
#include <stdio.h>
int main(int argc, char **argv)
{
    char buf[64];
    // LEN - 1 so that we don't write a null byte past
    // the bounds of buf if n = sizeof(buf)
    strncpy(buf, argv[1], 64-1);
}
```

Secure Coding (Alternative Functions)

Source: http://developer.apple.com

Don't use these functions	Use these instead
strcat	strlcat
strcpy	strlcpy
strncat	strlcat
strncpy	strlcpy
sprintf	snprintf
vsprintf	vsnprintf
gets	fgets

OS Defense: Stack Canaries

A canary is a tripwire placed before the return pointer that when overwritten by a buffer overflow changes the value of the canary and signals to the OS that the program state has been compromised.

- Terminator canaries - Canaries with randomly selected values placed at the ends of strings or buffers
- Random canaries - Randomly placed canaries that are hard to predict to an attacker, usually implemented as an XOR of the original control data

Practical Example

Important Resources

- Link: Iowa State University Software Center
  - Windows Operating System ISO images
  - VMWare Workstation or VMWare Fusion (Mac)
- Link: Kali Linux
  - VMWare Images
- Link: Online walkthrough of attack

Network Setup

Victim

- Windows XP Professional Service Pack 3
  - MiniShare 1.4.1
  - OllyDbg 1.10

Attacker

- Kali Linux
  - Python
  - Metasploit
  - Netcat

Common Vulnerabilities and Exposures

- What is a CVE?
  - Common Vulnerabilities and Exposures (CVE)
  - Assigns an identifier to the description of the problem
- Giant databases of known vulnerabilities (and exploits)
  - `http://cve.mitre.org/`
  - `http://nvd.nist.gov/`
  - `http://www.rapid7.com/db/`

MiniShare 1.4.1 CVE-2004-2271

The vulnerability is caused due to a boundary error in the handling of HTTP "GET" requests. This can be exploited to cause a buffer overflow by sending a specially crafted overly long request with a pathname larger than 1787 bytes.

Successful exploitation can lead to execution of arbitrary code.

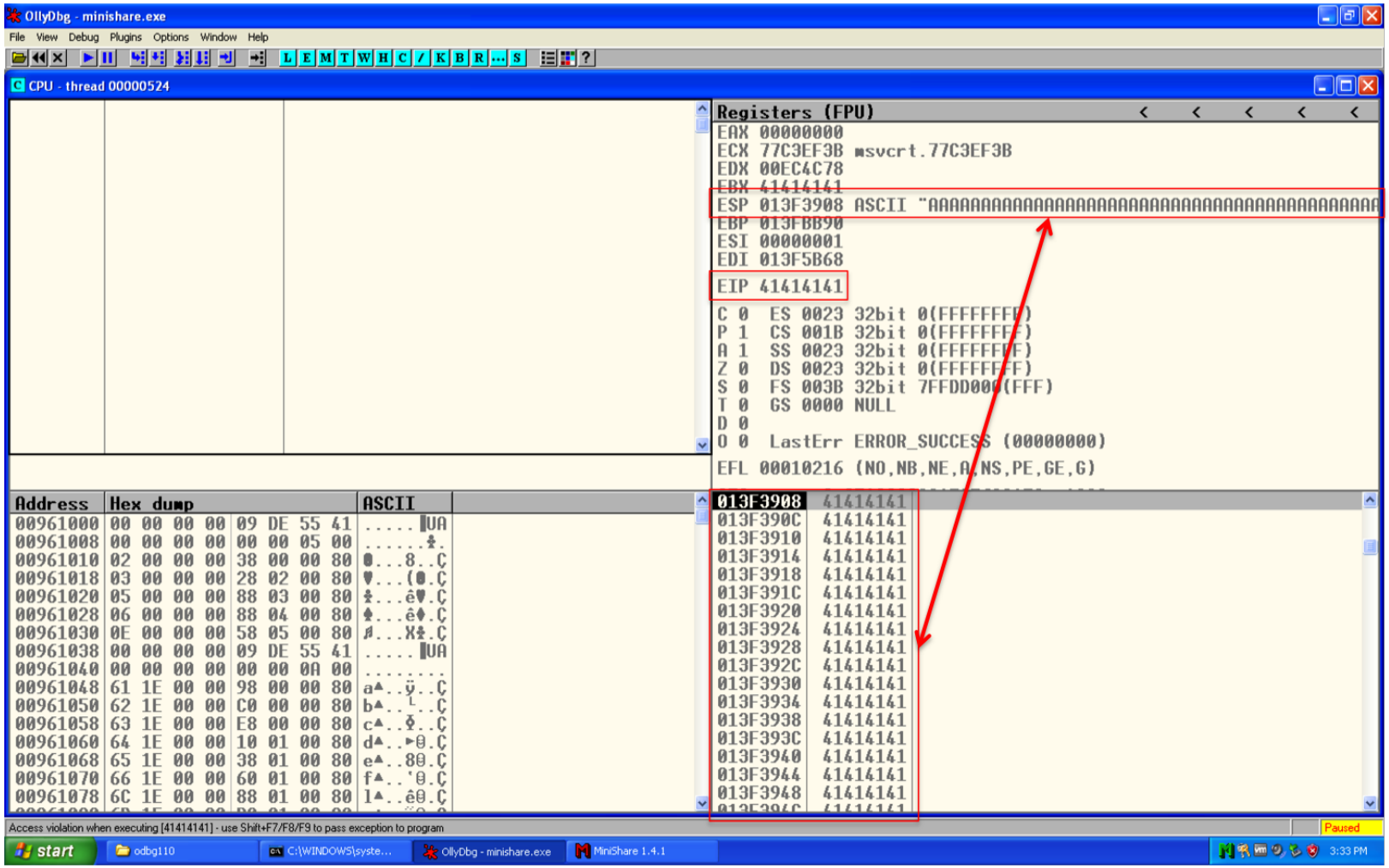
Reference:
https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2271

Demo

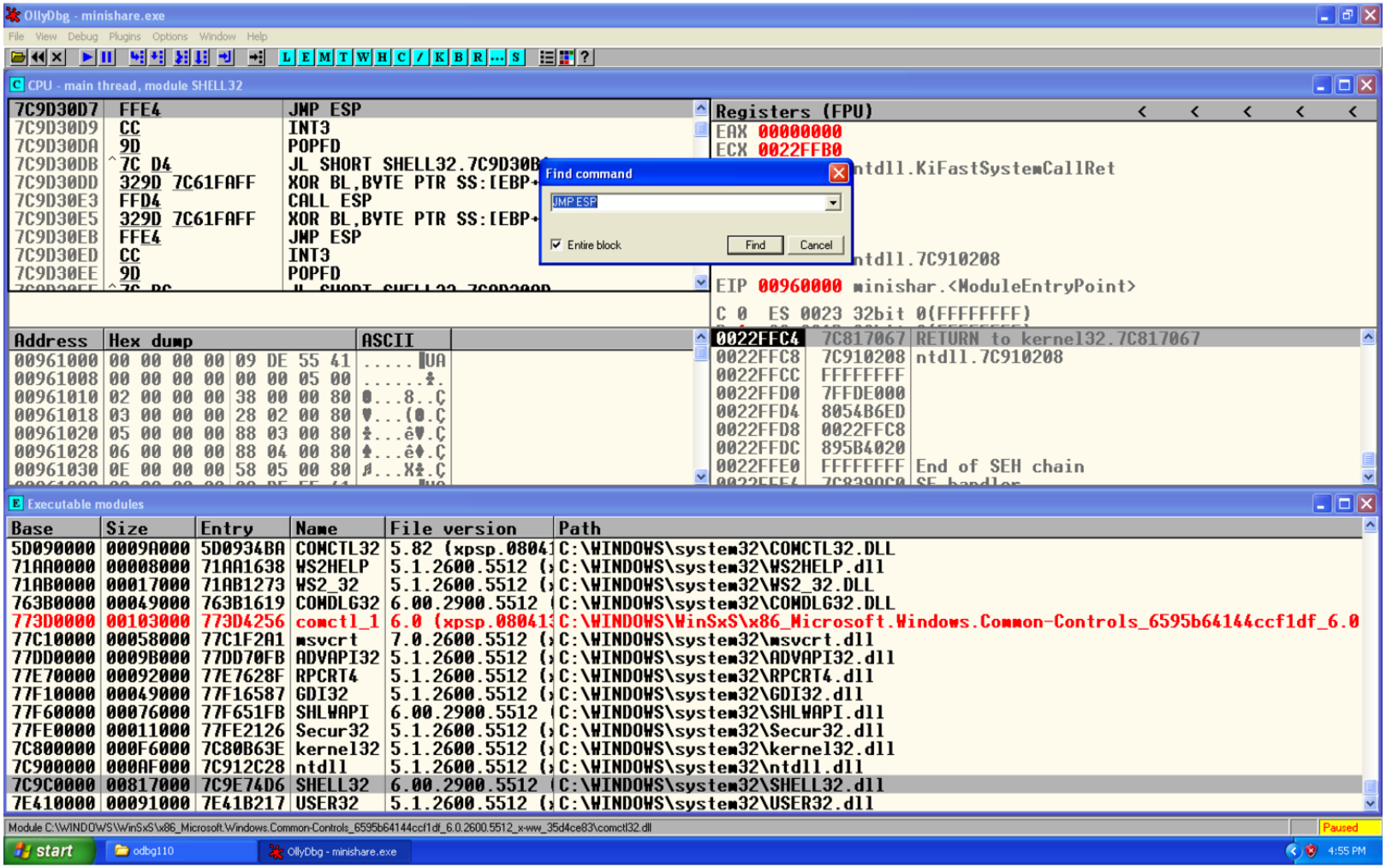
Demo building exploit for MiniShare 1.4.1

Note: See the markdown source of this presentation for an extended discussion in the comments.

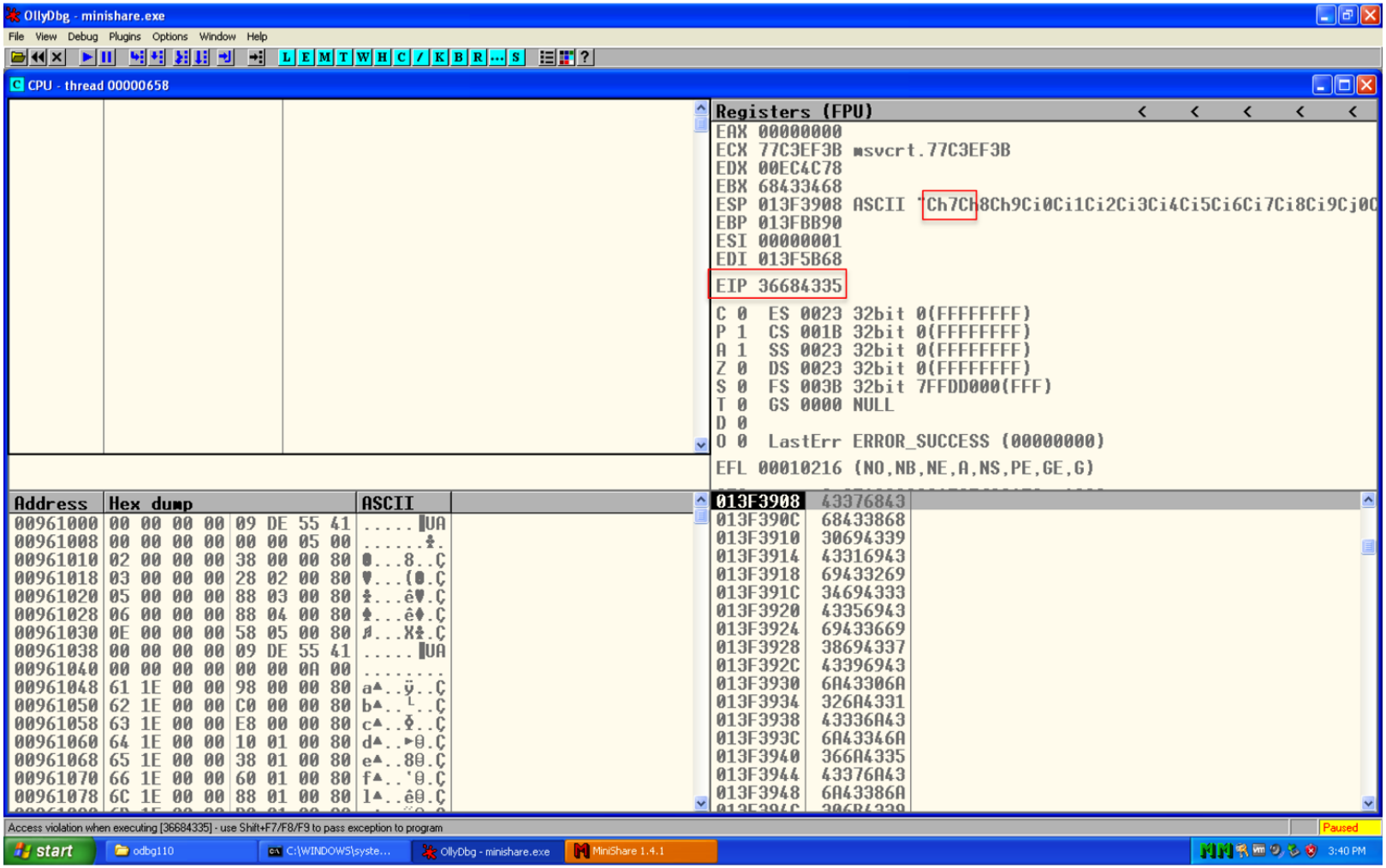
Reproduce Error



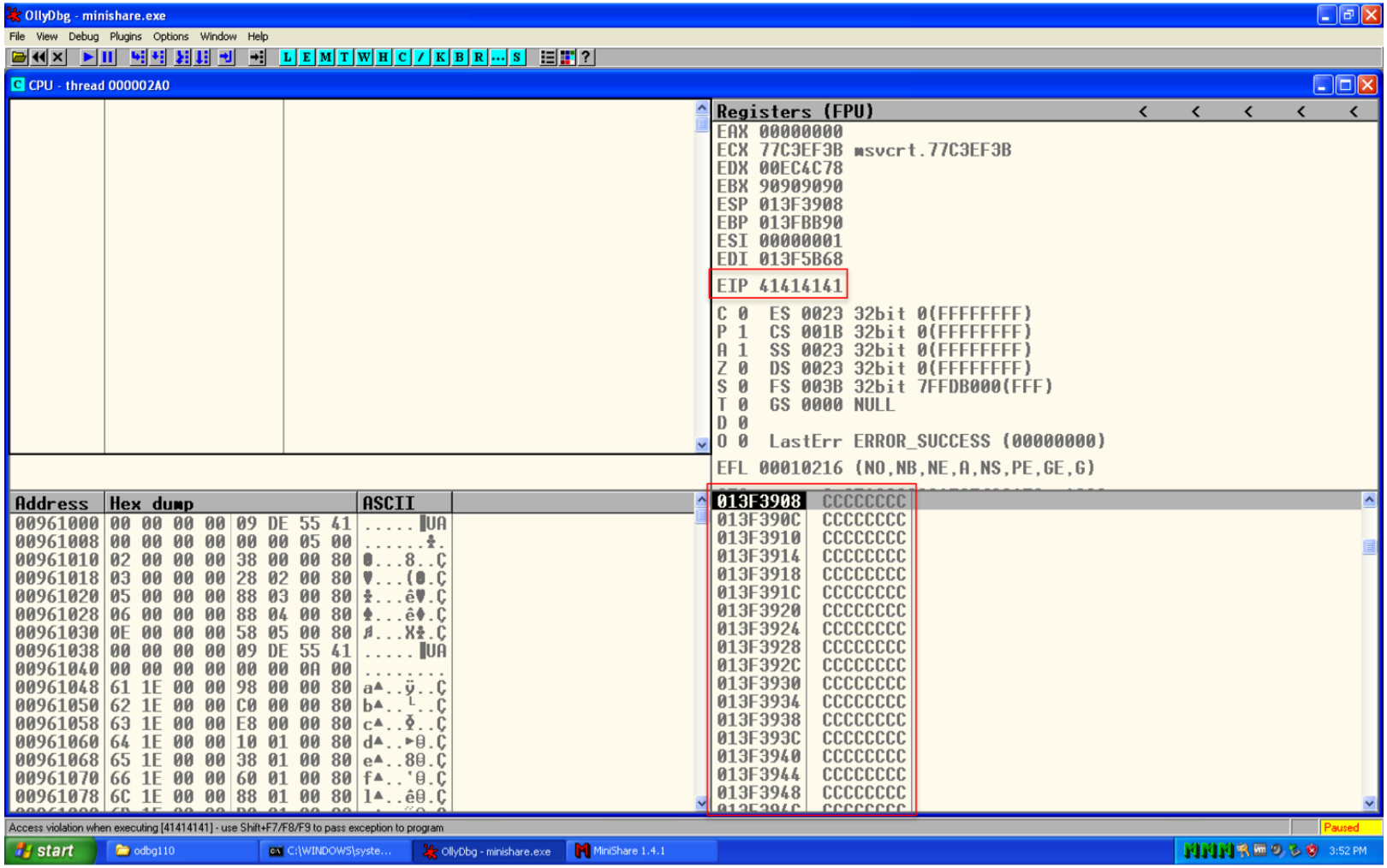
Find a JMP ESP Instruction



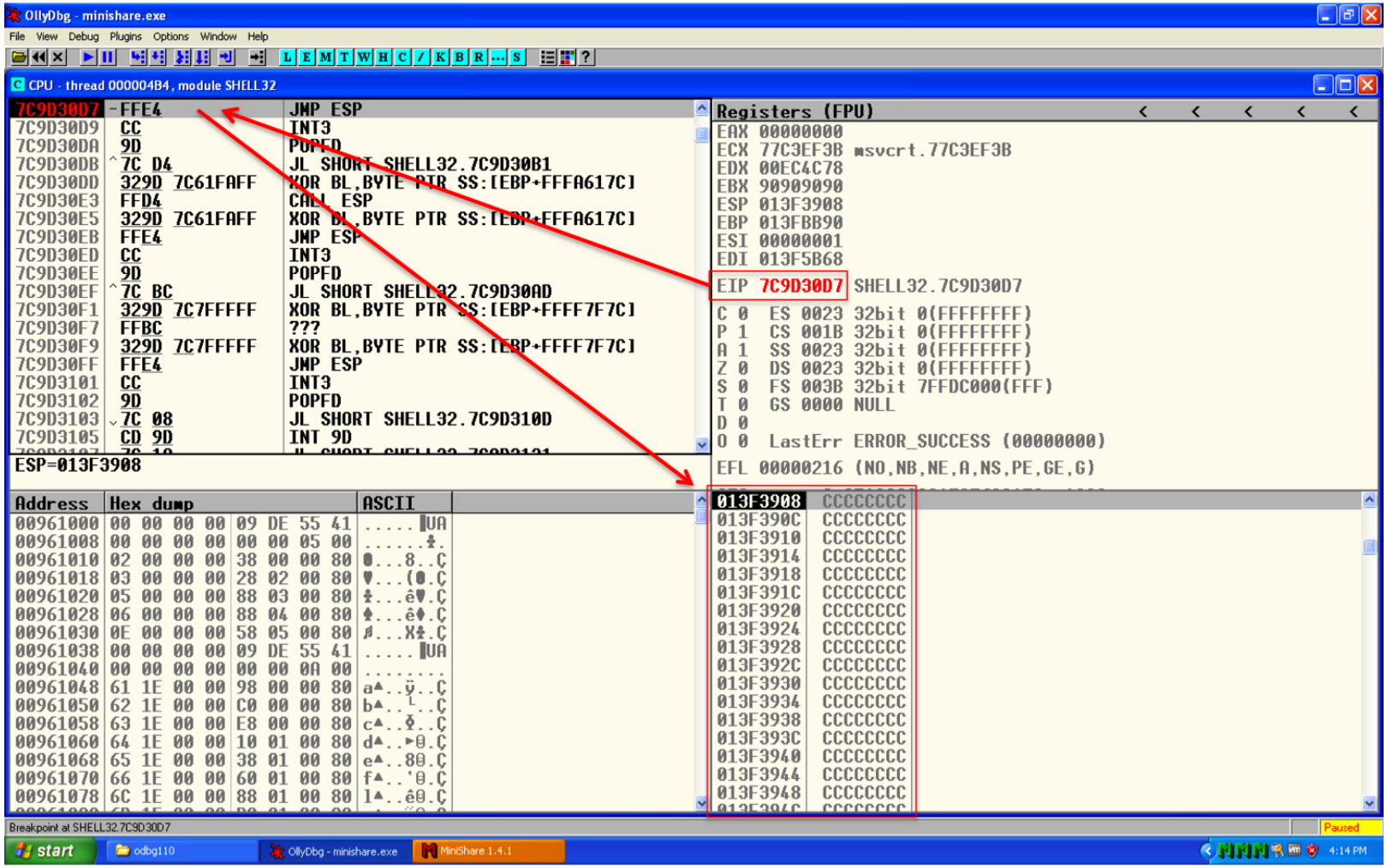
Calculate Overwrite Offsets



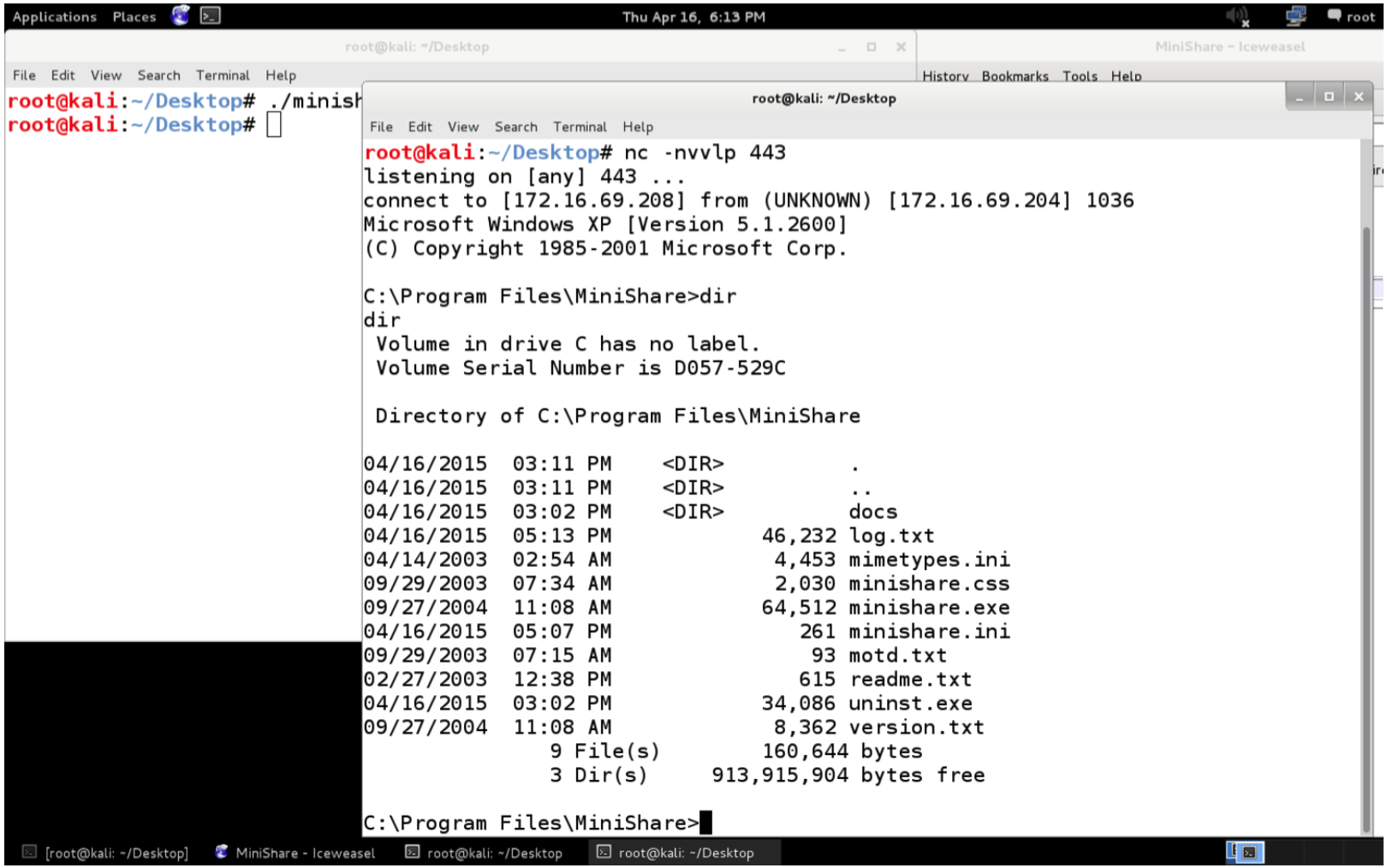
Test Offsets



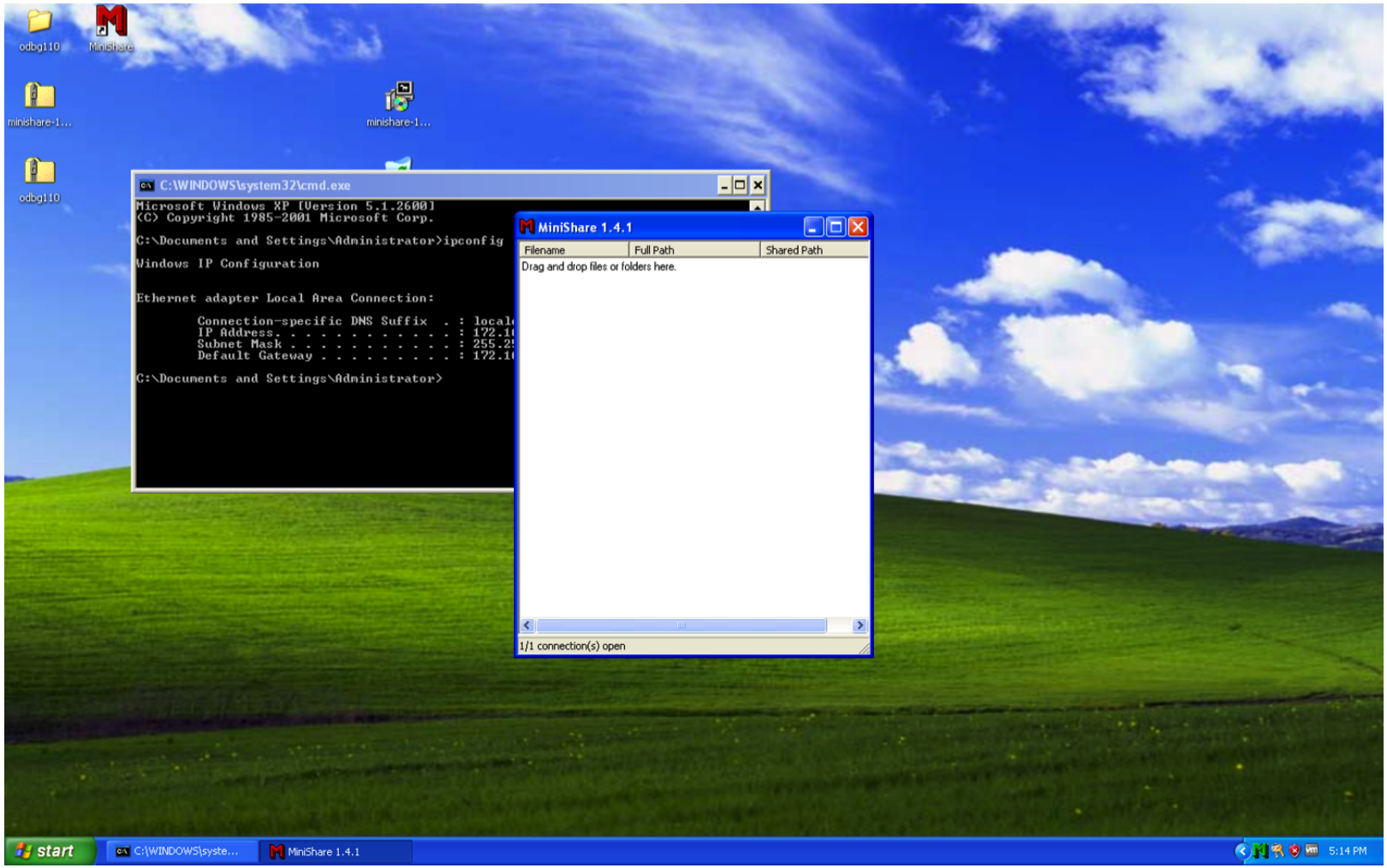
Jump to Shellcode



Execute Reverse TCP Shell



Silent Attack



Exploit

- len(buffer) $<$ 1787
- NOP filler could be any valid bytes here (just not 0x00, 0x0A, 0x0D), since the filler is not going to be executed
- A small NOP sled is added before the shellcode to give our exploit some stability
- Futher experimentation reveals we only have 2220 bytes to play with, leaving only 429 bytes for the payload

```
buffer[0]          EIP        ESP
| <-NOP Filler-> | JMP ESP | [(NOP * 16) Shellcode]-> |
0                  1787       1791                      2220
```

Metasploit Modules

- Create modularized exploit code for automatic exploitation of known vulnerabilities
  - `http://www.rapid7.com/db/modules/exploit/windows/http/minishare_get_overflow`
  - `https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/http/minishare_get_overflow.rb`

Writing exploits requires some skill. Running exploits requires very little knowledge (script kiddies).

OS Defenses

OS Defense: Stack Canaries

A canary is a tripwire place before the return pointer that when overwritten by a buffer overflow changes the value of the canary and signals to the OS that the program state has been compromised.

- Terminator canaries - Canaries with randomly selected values placed at the ends of strings or buffers
- Random canaries - Randomly placed canaries that are hard for an attacker to predict, usually implemented as an XOR of the original control data

OS Defense: Data Execution Prevention (DEP)

Uses segementation to mark areas of memory as either executable or non-executable. Executing memory marked as non-executable causes a segmentation fault.

- Prevents an attacker from executing code in an area of memory that was intended soley for data
  - Idea: If we know its data, don't treat it like code!
- Attackers can combat this technique by using what already exists in memory
  - Return-to-libc just sets up parameters and then sets the return pointer to a function in the standard C library (i.e. calls a C function)

OS Defense: Address Space Layout Randomization (ASLR)

Randomly rearranges the address locations in memory of key data areas of a process, including the base of the executable and the positions of the stack, heap and libraries.

- Prevents an attacker from reliably jumping to a position in memory
  - Idea: It doesn't matter if you can control the instruction pointer if you don't know where to jump to!
- Attackers combat this technique by trying to reduce entropy
  - Heap spraying loads several copies of the attack into memory to increase success rates, by hoping the jump hits an exploit
  - Leaky/danging pointers (pointers that don't point where they are suppose to be pointing)

Quiz

Questions

Any questions before the quiz?