EDS: Exploitation Detection System

By Amr Thabet
Q-CERT

About The Author

- **Amr Thabet (@Amr_Thabet)**
- **Malware Researcher at Q-CERT**
- **The Author of:**
  - Security Research and Development Framework (SRDF)
  - Pokas x86 Emulator
- **Wrote a Malware Analysis Paper for Stuxnet**

Introduction

- Now the APT Attack become the major threat
- Bypasses all defenses
- Standards and Policies doesn’t work
- Bypasses IDS, IPS, Firewalls .. etc

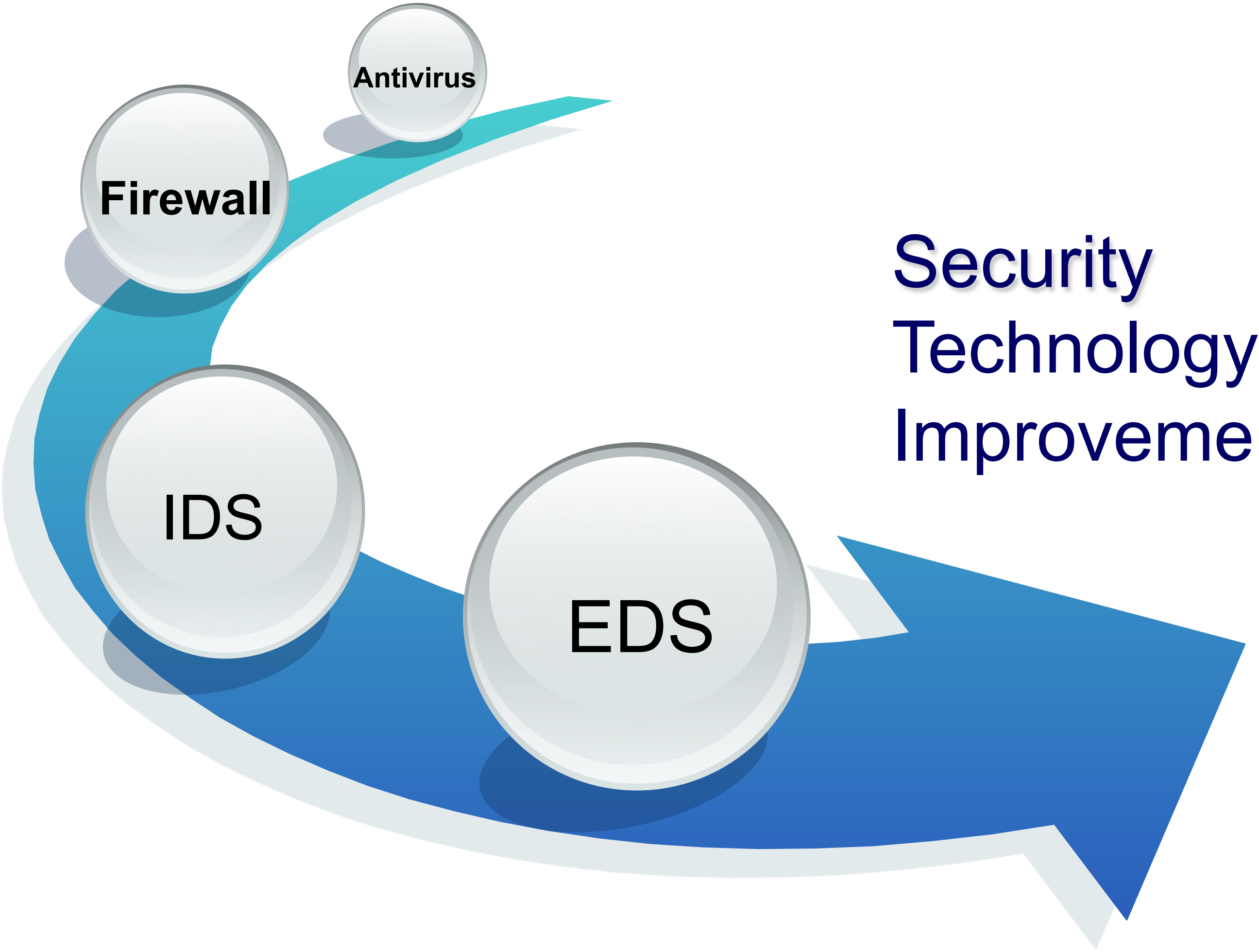
Introduction

- **The Attacker uses:**
  - Client-side attacks and exploits
  - Spear-phishing attacks
- **Uses undetectable malwares**
- **Uses HTTP and HTTPs**
- **Attack the servers from the infected clients**

Introduction

- The Next Security Technology is the : “Exploitation Detection Systems”
- EDS is only way to stop Attacks from behind
- Stop Attacks from Client-Side
- Stop successful exploitation for a 0-day

Improvements in Defense



Security Technology Improvements

Introduction

- **The Talk today is about:**
  - EDS as a concept and next technology
  - EDS: the new tool that I created
  - The Development of EDS
  - SRDF Framework (adv ☺)
- **I will try to explain everything for who don’t know about Exploits … etc**

Contents

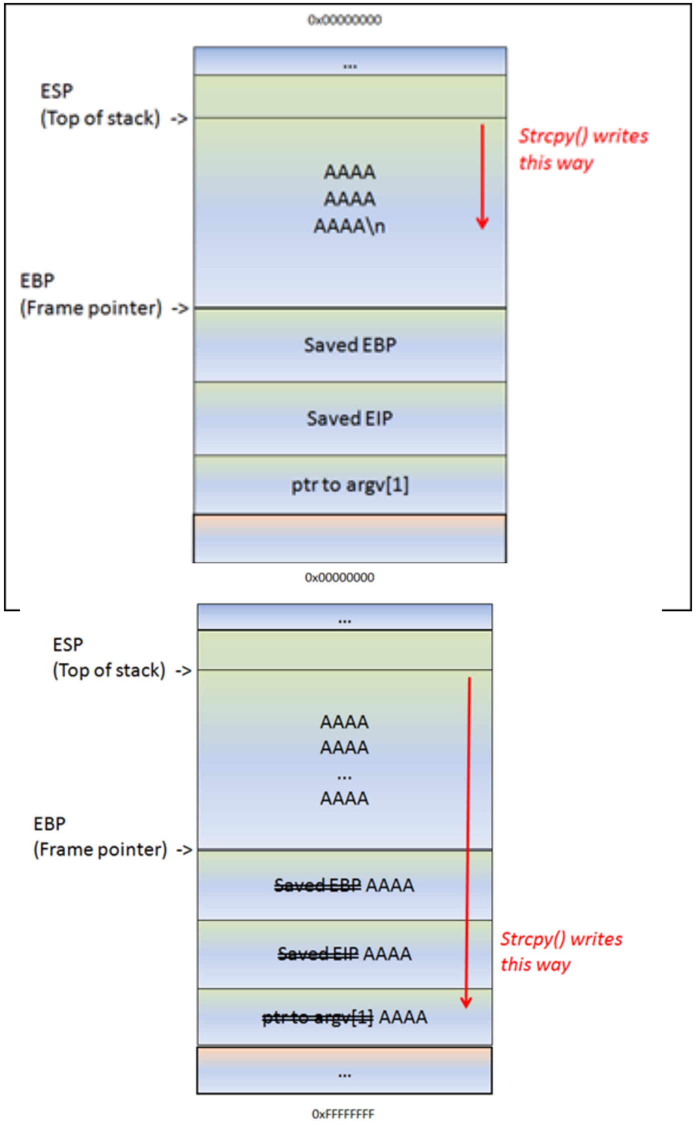
- Motivation and Goals
- The Design of EDS
- Mitigations in Depth
- Monitoring System
- Development and Future work

Goals

- Stop Exploitation for new 0-days
- Works with Memory Corruption Exploits
- Detect Compromised Processes
- Prevent and/or Alert of Exploited Processes

Memory Corruption Vulnerabilities

- **Simply write data in places you are not intended to write on it**
- **Like:**
  - Pointers
  - Return addresses
- **Change how the application behave**
- **Check: www.corelan.be**



Antivirus vs EDS

- EDS is not signature based
- EDS doesn’t detect malware
- EDS main goal to stop exploitation
- EDS is memory based
- EDS searches for evidence of Memory corruption and indication of compromise

Previous Work

Compile-Time Solutions:

- Takes Long time to affect
- Always there’s exceptions

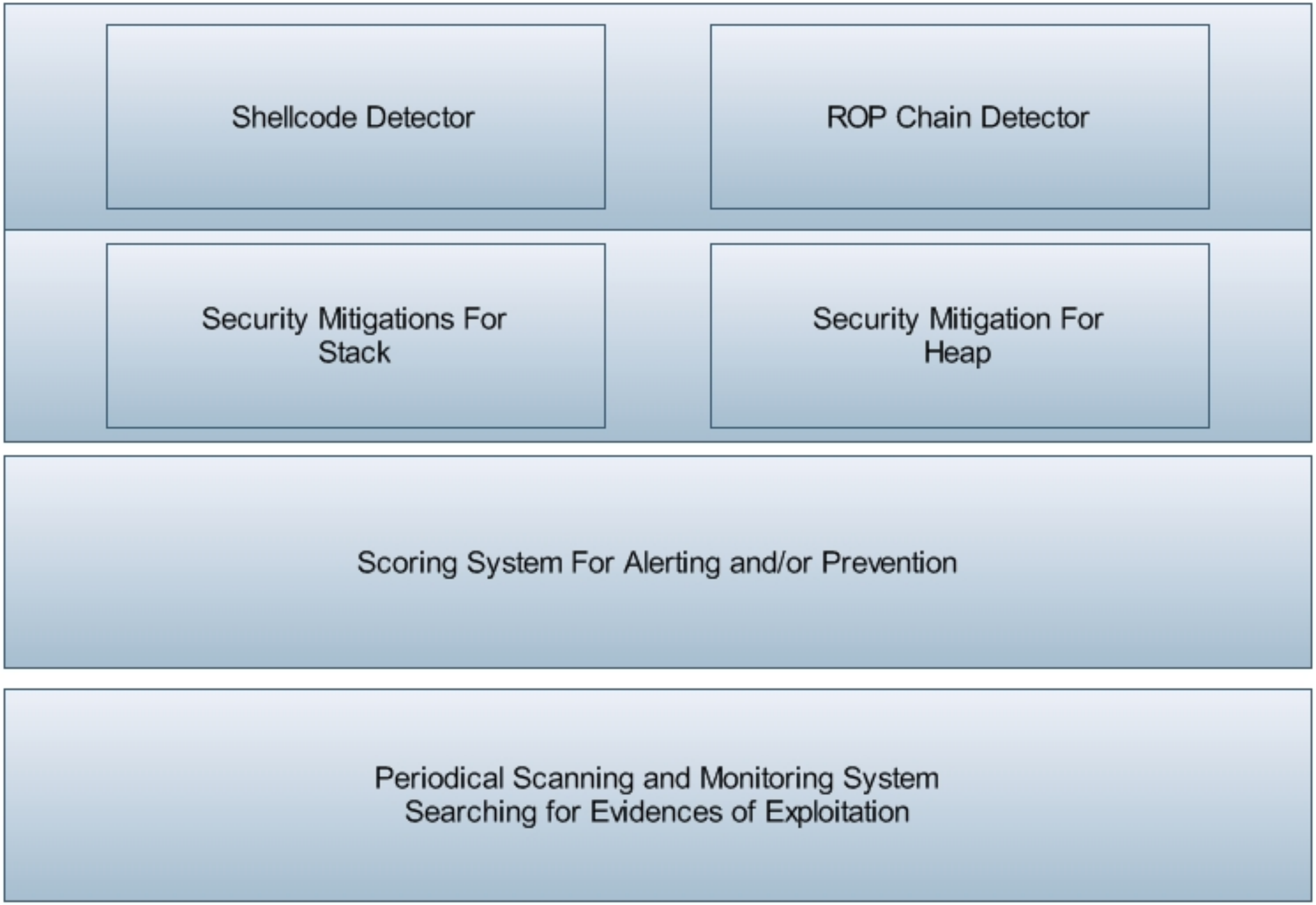
Current Run-time Solutions:

- Only One Layer of Defense
- On-Off Mitigations
- No detection of this layer was bypassed or not
- A fight between false positives and false negatives

What’s New?

- Co-operative Mitigations
- Based on Scoring System
- Prevention and Alerting Infected processes
- Additional layer with Monitoring System

Design of EDS



Shellcode Detector

ROP Chain Detector

Security Mitigations For Stack

Security Mitigation For Heap

Scoring System For Alerting and/or Prevention

Periodical Scanning and Monitoring System Searching for Evidences of Exploitation

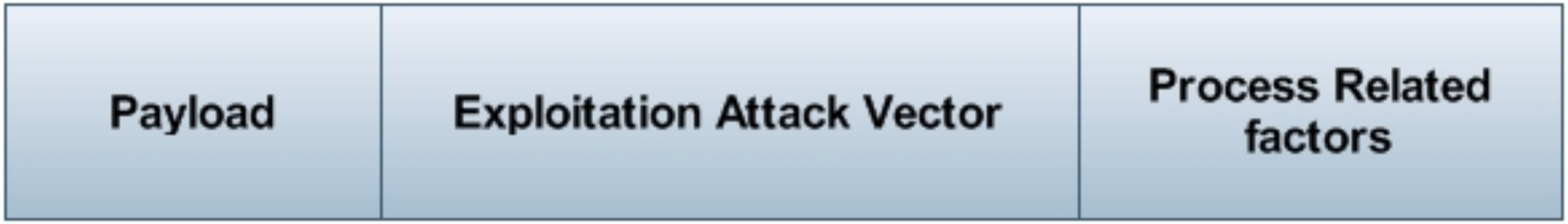
Design of EDS

- **Payload Detection:**
  - Shellcode Detection
  - ROP Chain Detection
- **Security Mitigations For Stack:**
  - ROP Detection
- **Security Mitigation For Heap:**
  - Heap Overflow
  - Heap Spray
  - Use After Free

Design of EDS

Scoring System:

- Based On Payload Detection and Security Mitigations
- Scoring Based on Payload, Attack Vector and The Process abnormal behavior



Payload

Exploitation Attack Vector

Process Related factors

Design of EDS

Monitoring System:

- Searches for Evidence of Exploitation
- Detect bypassed Mitigations
- Alert the Administrators to Take Action
- Looking at the previous EDS reports for this process

Mitigation In Depth: Payload

- Increase the score of suspiciously
- Detect suspicious inputs and tries for exploitation.
- Divided Into:
  - Shellcode Detection
  - ROP Chain Detection

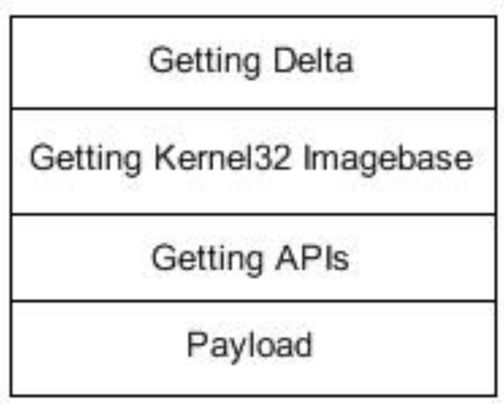
What’s Shellcode?

- It is simply a portable native code
- Sent as a bunch of bytes in a user input
- Do a specific action when the processor executes it
- The attacker modify the return address to point to it.

What’s Shellcode?

- It gets its place in memory
- Then it gets the kernel32 DLL place in memory
- Get windows functions (APIs) from it
- And then … **ATTACK**
- Check:
- http://www.codeproject.com/Articles/325776/The-Art-of-Win32-Shellcoding

Shellcode Skeleton



Getting Delta

Getting Kernel32 Imagebase

Getting APIs

Payload

What’s Shellcode

- Some shellcodes shouldn’t have null bytes (sent as string)
- Some are encrypted
- There’s a loop to decrypt it
- Some are in ascii
- Some doesn’t include loop but many pushes (to be in ascii)

Shellcode Detection

Goals:

- Very fast shellcode detector
- Very hard to bypass … min false negative
- Low false positive

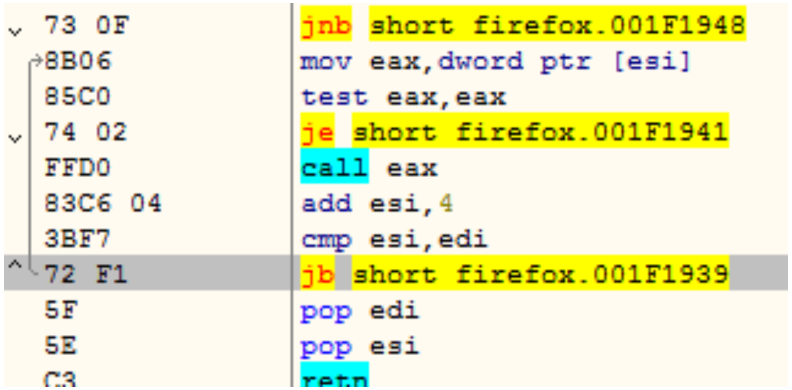
Shellcode Detector

- **Static Shellcode Detector**
- **Divided into 3 phases:**
  - Indication of Possible Shellcode (GetPC … etc)
  - Filter by invalid or privileged Instructions
  - Filter by Flow Analysis

Indication of Possible Shellcode

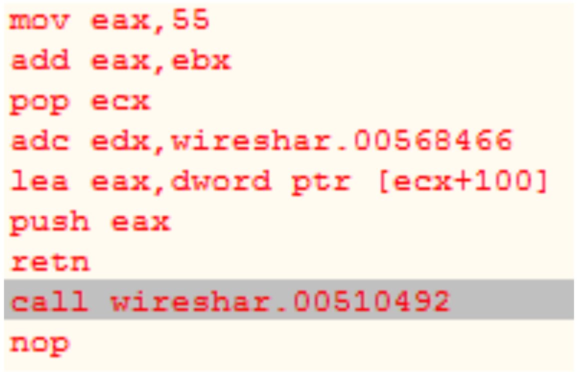
Search for Loops

- Jump to previous



```
73 0F        jnb short firefox.001F1948
8B06         mov eax,dword ptr [esi]
85C0         test eax,eax
74 02        je short firefox.001F1941
FFD0         call eax
83C6 04      add esi,4
3BF7         cmp esi,edi
72 F1        jb short firefox.001F1939
5F           pop edi
5E           pop esi
C3           retn
```

- Call to previous (Call Delta)

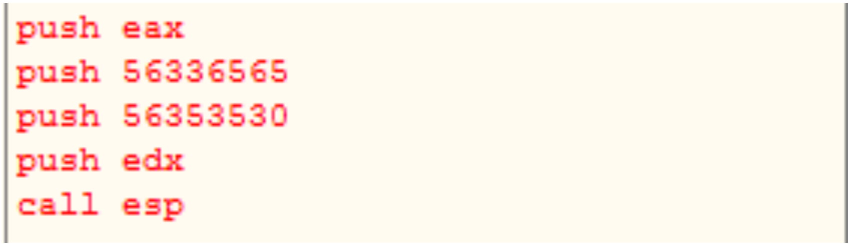


```
mov eax,55
add eax,ebx
pop ecx
adc edx,wireshar.00568466
lea eax,dword ptr [ecx+100]
push eax
retn
call wireshar.00510492
nop
```

- Loop Instruction

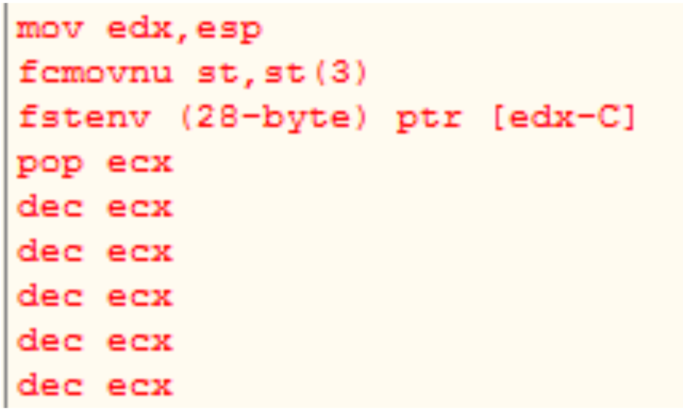
Indication of Possible Shellcode

- High rate of pushes end with flow redirection



```
push eax
push 56336565
push 56353530
push edx
call esp
```

- Search for fstenv followed with at least 5 valid instructions after it



```
mov edx,esp
fcmovnu st,st(3)
fstenv (28-byte) ptr [edx-C]
pop ecx
dec ecx
dec ecx
dec ecx
dec ecx
dec ecx
```

Skip Invalid Instructions

- We skip all invalid instructions.
- We skip all privileged instructions like:

  IN, OUT, INT, INTO, IRETD, WAIT, LOCK, HLT … etc

- Skip Instructions with unknown Behavior like:

  JP, AAM, AAD, AAA, DAA, SALC, XLAT, SAHF, LAHF, LES, DES,

Flow Analysis

- **Check on ESP Modifications through loops**
  - If there's many pushes with no pops in loops
- **Check on Compares and Jccs in th code**
  - Search for Jcc without compare or similar before it.
- **Check on % of Nulls and Null-Free**

Shellcode Statistics

File Type	Total No of Pages	Infected Pages	Presentage
Pcap	381	40	2%
Pcap	11120	543	4%
Wmv	104444	4463	4%

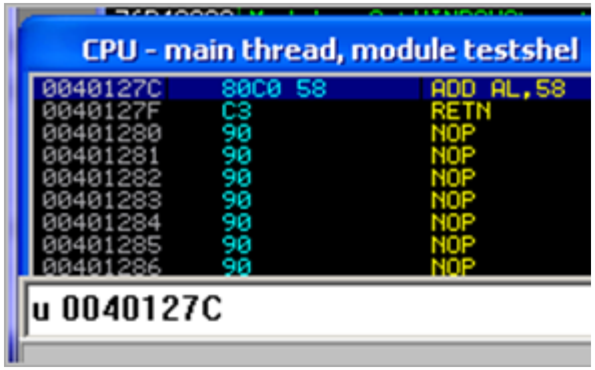
- Scan per page
- False Positives in range 4% Infected Pages
- All of these samples are legitimate

Shellcode Statistics

- It detects all Metasploit Shellcodes
- Detects all working shellcodes in Shellstorm (win32 – ASLR Bypass)
- Detected Encoded Shellcodes by metasploit Encoders
- Manual Evasion is possible

What’s ROP Chain

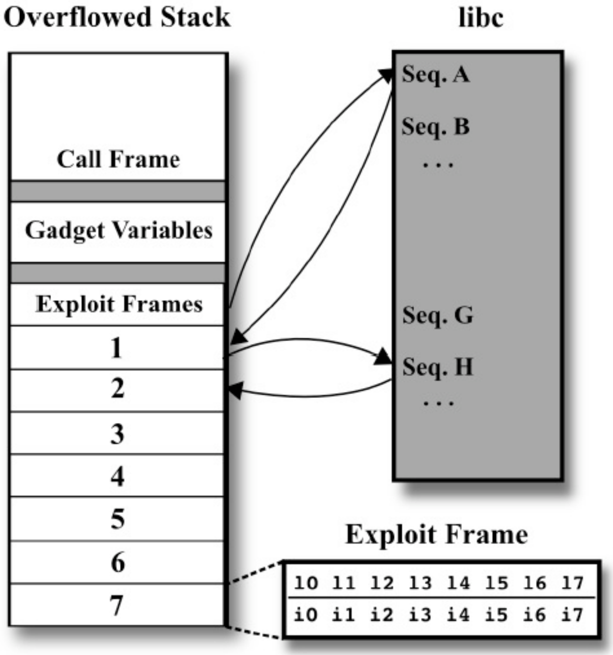
- Very small code in a legitimate dll
- End with “ret” instruction
- Attackers uses a series of it
- All of them together = a working shellcode
- Used to bypass DEP



CPU - main thread, module testshel

```
0040127C   80C0 58    ADD AL,58
0040127F   C3         RETN
00401280   90         NOP
00401281   90         NOP
00401282   90         NOP
00401283   90         NOP
00401284   90         NOP
00401285   90         NOP
00401286   90         NOP
```

```
u 0040127C
```



ROP Chain Detection

- **It’s a very simple ROP Detection**
- **Search for Return with these criteria:**
  - the address is inside an executable page in a module
  - the return address not following a call
  - Followed by ret or equivalent instructions in the next 16 bytes
  - Not Following series of (0xCC)

Stack Mitigations

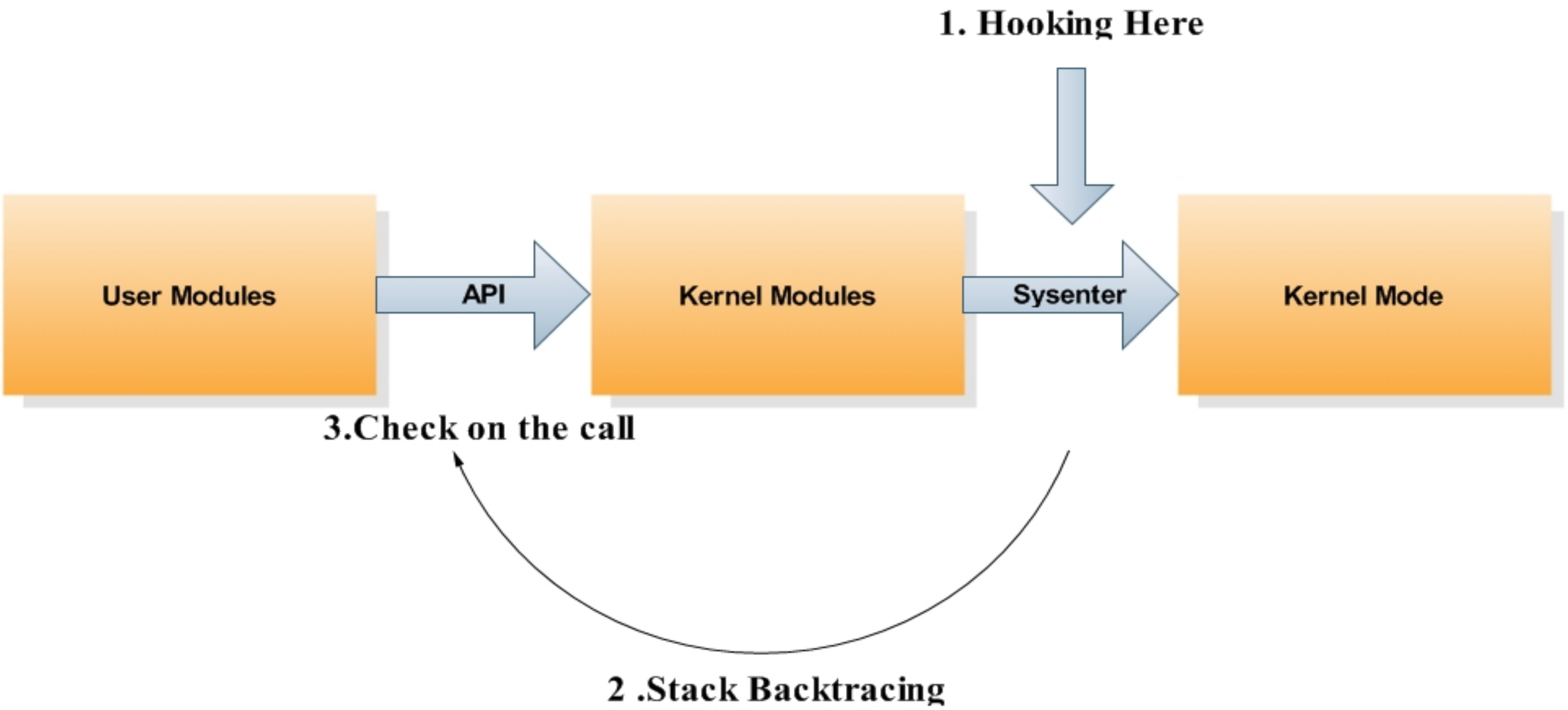
- We detect ROP Attacks
- The Mitigation is named “Wrong Module Switching”
- We detect SEH Overwrite
- We scan for Leaked ROP chains (which not overwritten)

ROP Attack Vector

- ROP are used to bypass DEP
- They mostly ret to VirtualProtect API
- Make the shellcode's memory executable
- Or calls to another windows APIs

Wrong Module Switching

- Detect ROP Attacks
- Based on Stack Back-tracing



Wrong Module Switching

- Hooks in Kernel-Mode on win32
- Uses SSDT Hooking
- Hooking on WOW64 for win64
- Hook Specific APIs
- Hooks:
  - VirtualProtect and similar functions
  - CreateProcess and similar
  - Network and Socket APIs
  - And more

Wrong Module Switching

- **Using Stack Backtracing to Return to The API Caller**
- **Checks the API Call are:**
  - Check The Call to this API or not
  - Check The Parameters
  - Check the next Call Stack if it calls to the function that calls to the API
  - Check The SEH if it’s in the same module
  - Check if there’s null parameters
  - Near return address after the call
  - And more
- **Gives a score to API call**

Wrong Module Switching

- Check on Different Calls like:

  -
    ```
    Call dword ptr [<kernel32.API>]
    ```

  -
    ```
    Lea eax, <kernel32.API>
    call eax
    ```

  -
    ```
    Call API
    API:Jmp dword ptr [<kernel32.API>]
    ```

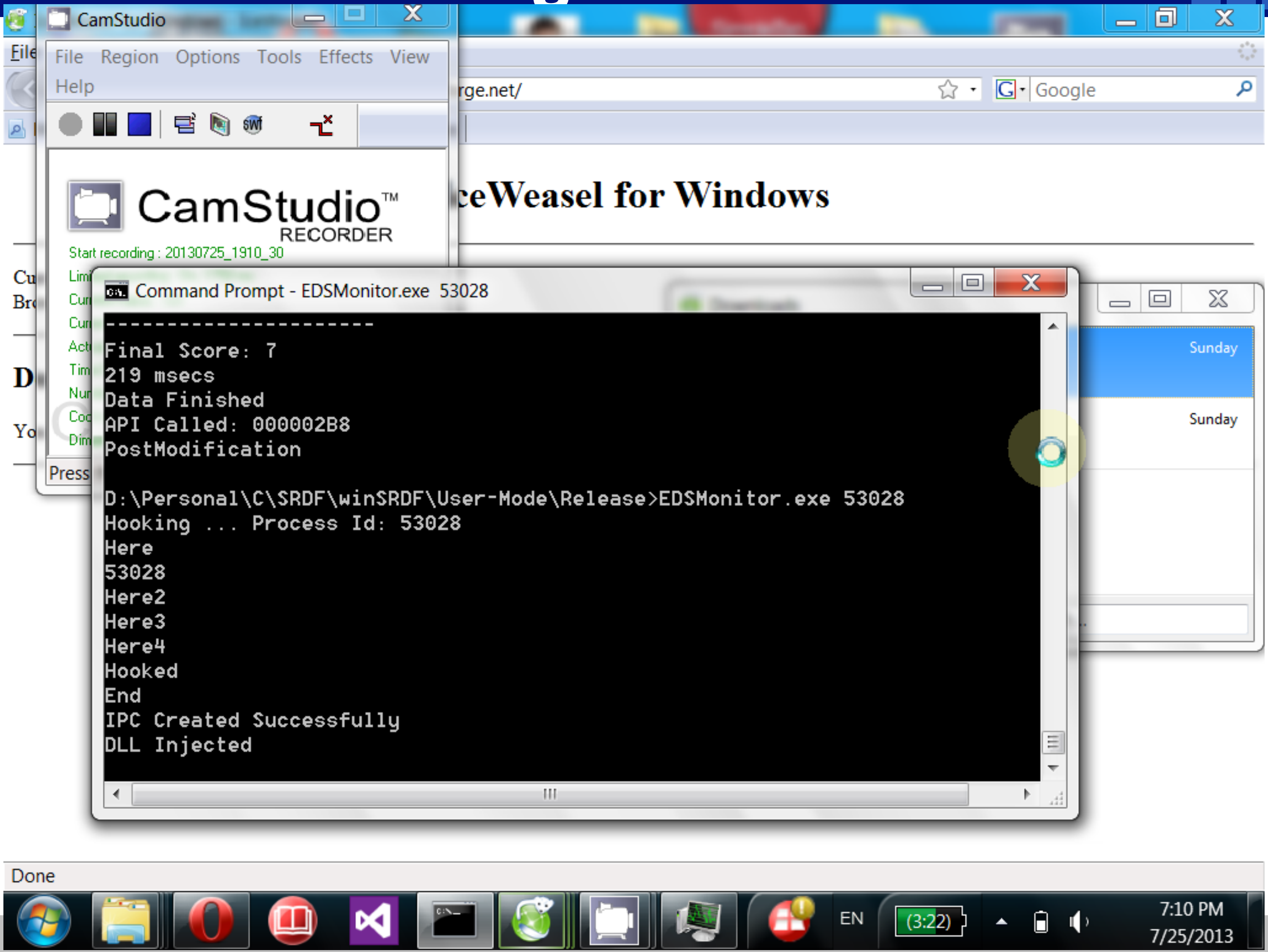
Wrong Module Switching

- Category Parameters based on:
  - **CONST:** push xxxxxxxxh
    OR lea eax, [xxxxxxxxh]
    push eax
  - **STACK:** lea eax, [ebp +/- xxxxh]
    push eax
  - **REGISTER:** push exx
  - **UNKNOWN:** push any

Wrong Module Switching

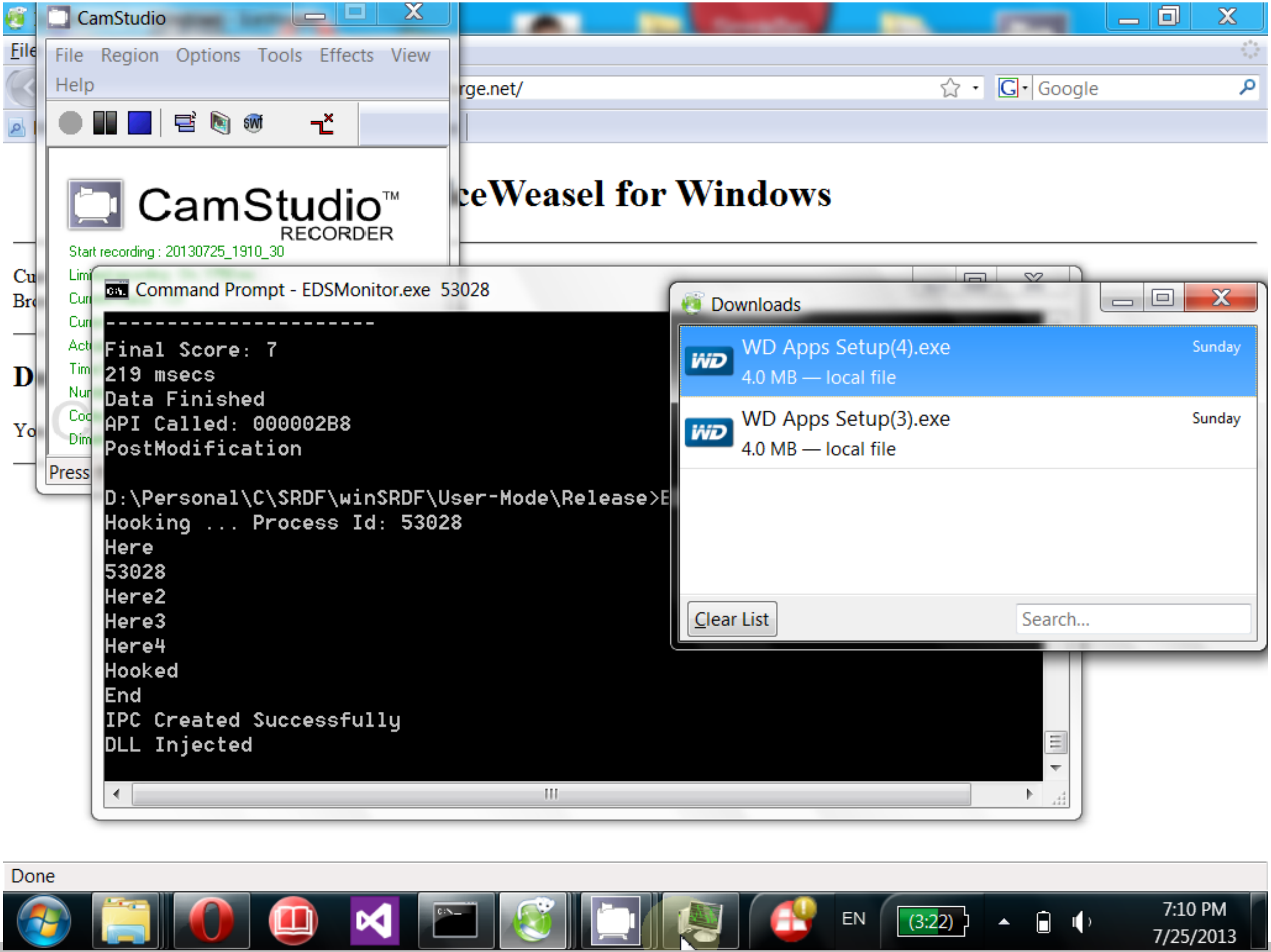
Demo on ShellExecute

Demo: Hooking Firefox with EDS



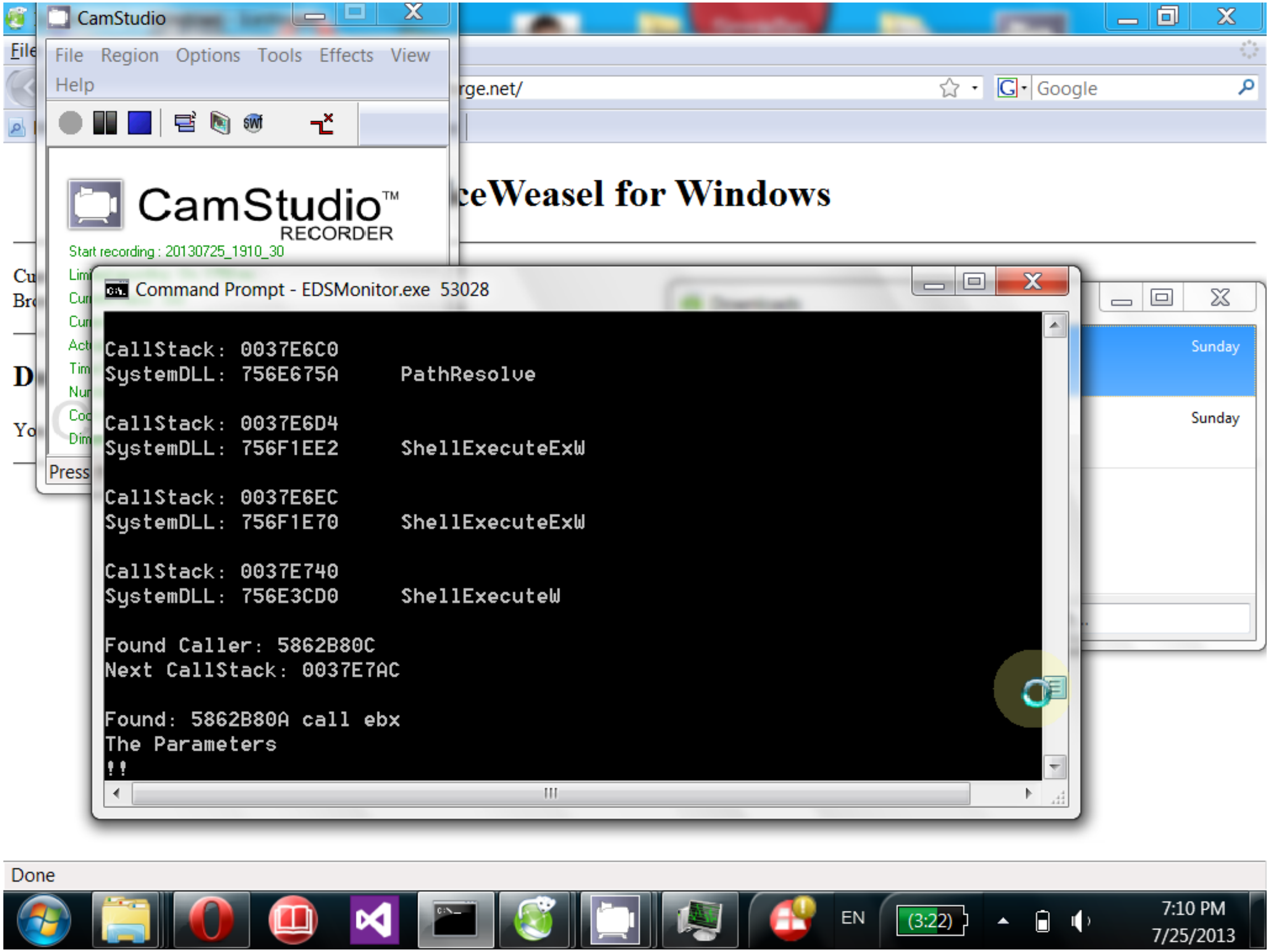
```
----------------------
Final Score: 7
219 msecs
Data Finished
API Called: 000002B8
PostModification

D:\Personal\C\SRDF\winSRDF\User-Mode\Release>EDSMonitor.exe 53028
Hooking ... Process Id: 53028
Here
53028
Here2
Here3
Here4
Hooked
End
IPC Created Successfully
DLL Injected
```

Demo: Force Firefox to create Process



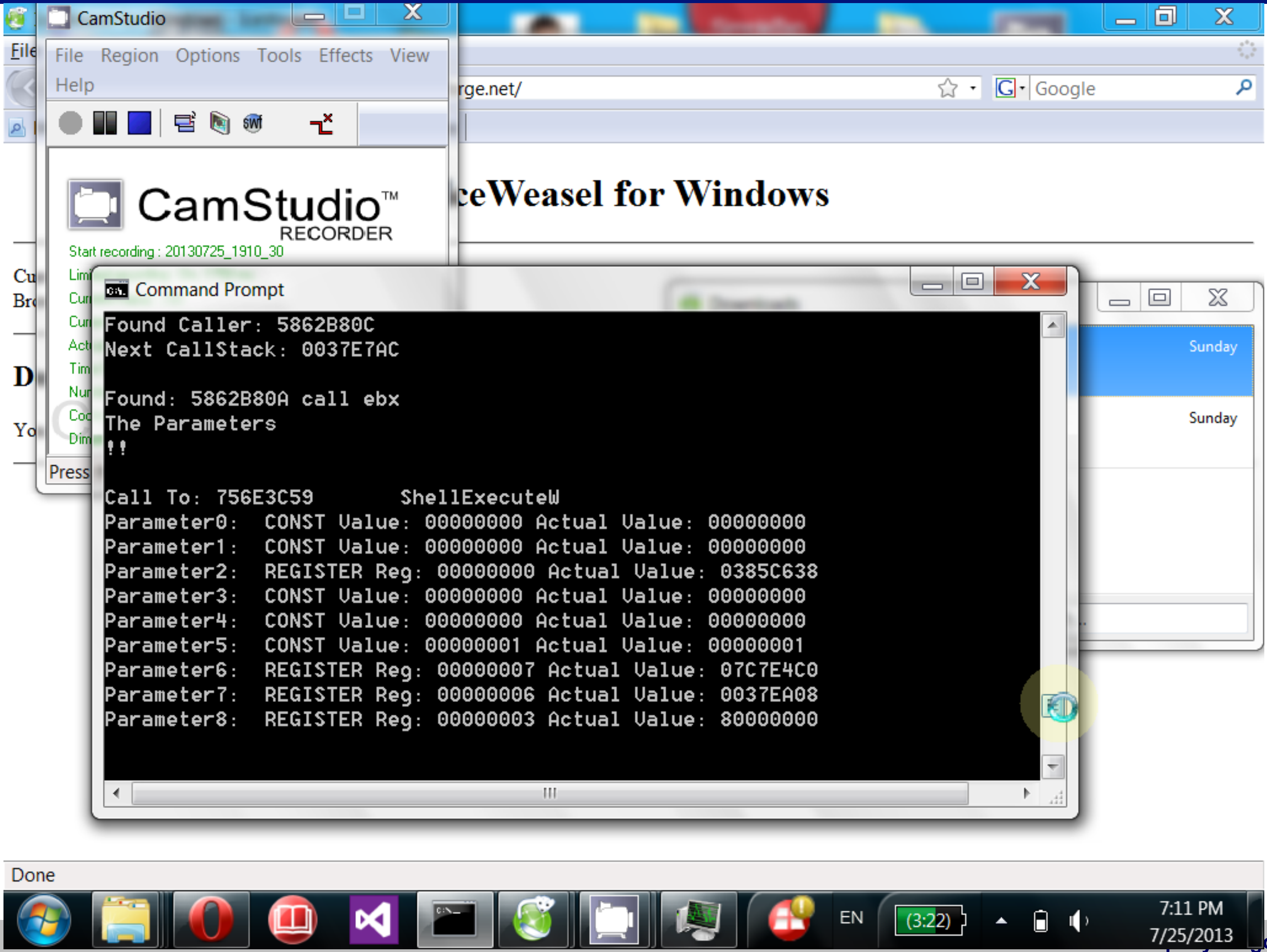
```
-----------------------
Final Score: 7
219 msecs
Data Finished
API Called: 000002B8
PostModification

D:\Personal\C\SRDF\winSRDF\User-Mode\Release>E
Hooking ... Process Id: 53028
Here
53028
Here2
Here3
Here4
Hooked
End
IPC Created Successfully
DLL Injected
```

Demo: The call stack to ShellExecute



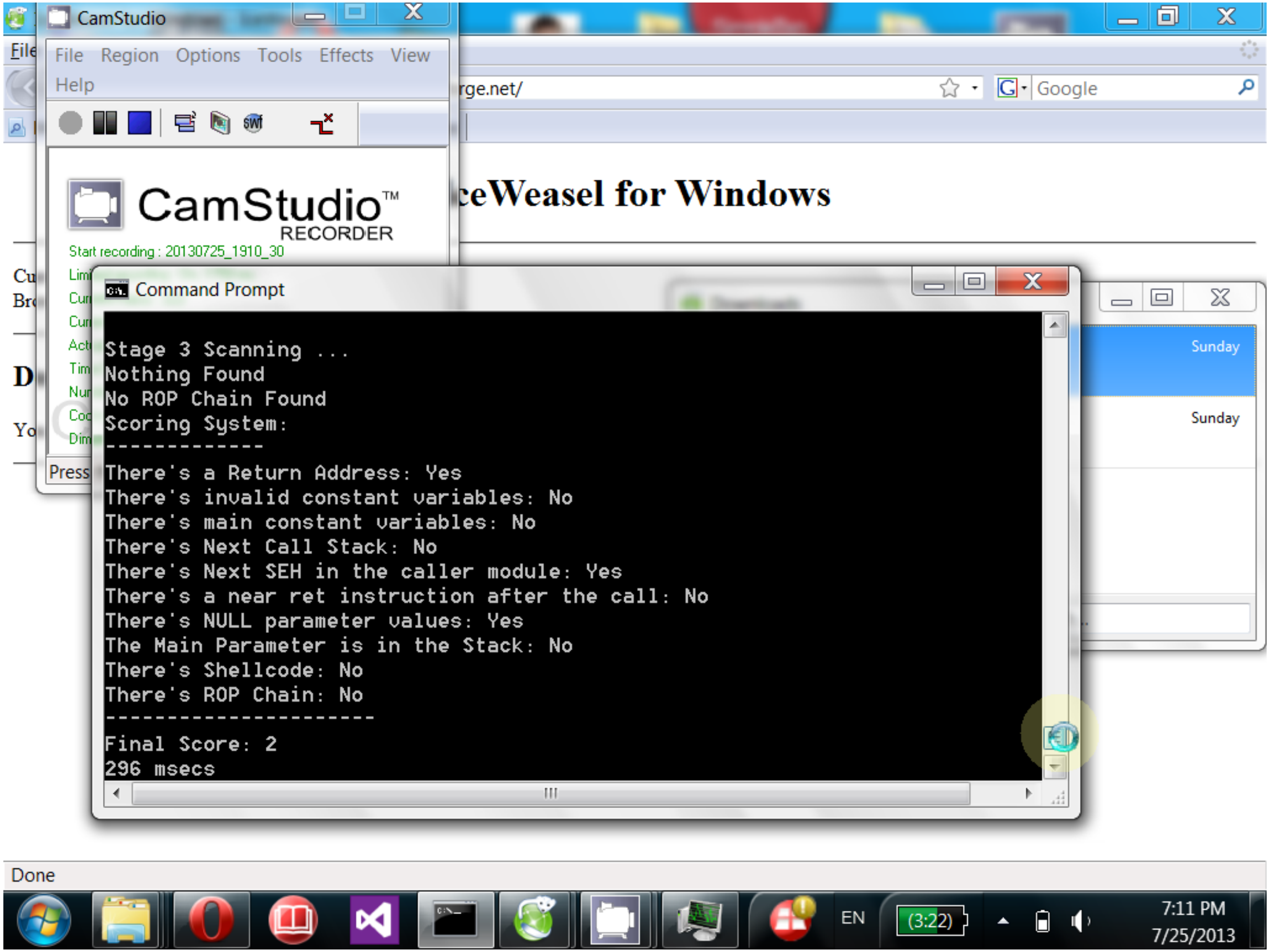
```
CallStack: 0037E6C0
SystemDLL: 756E675A     PathResolve

CallStack: 0037E6D4
SystemDLL: 756F1EE2     ShellExecuteExW

CallStack: 0037E6EC
SystemDLL: 756F1E70     ShellExecuteExW

CallStack: 0037E740
SystemDLL: 756E3CD0     ShellExecuteW

Found Caller: 5862B80C
Next CallStack: 0037E7AC

Found: 5862B80A call ebx
The Parameters
!!
```

Demo: The ShellExecute Params



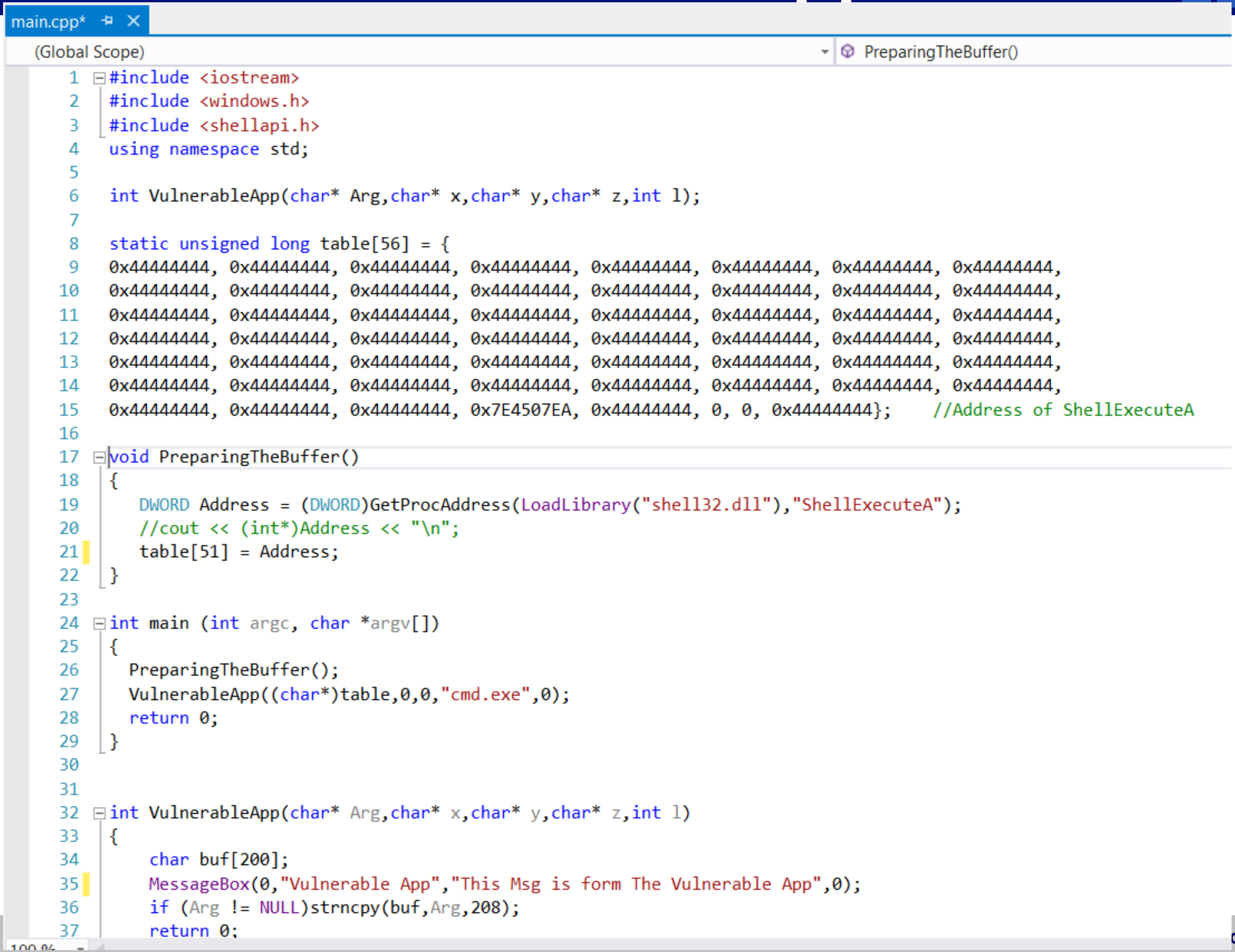
```
Found Caller: 5862B80C
Next CallStack: 0037E7AC

Found: 5862B80A call ebx
The Parameters
!!

Call To: 756E3C59       ShellExecuteW
Parameter0:  CONST Value: 00000000 Actual Value: 00000000
Parameter1:  CONST Value: 00000000 Actual Value: 00000000
Parameter2:  REGISTER Reg: 00000000 Actual Value: 0385C638
Parameter3:  CONST Value: 00000000 Actual Value: 00000000
Parameter4:  CONST Value: 00000000 Actual Value: 00000000
Parameter5:  CONST Value: 00000001 Actual Value: 00000001
Parameter6:  REGISTER Reg: 00000007 Actual Value: 07C7E4C0
Parameter7:  REGISTER Reg: 00000006 Actual Value: 0037EA08
Parameter8:  REGISTER Reg: 00000003 Actual Value: 80000000
```

Demo: The Action Scoring



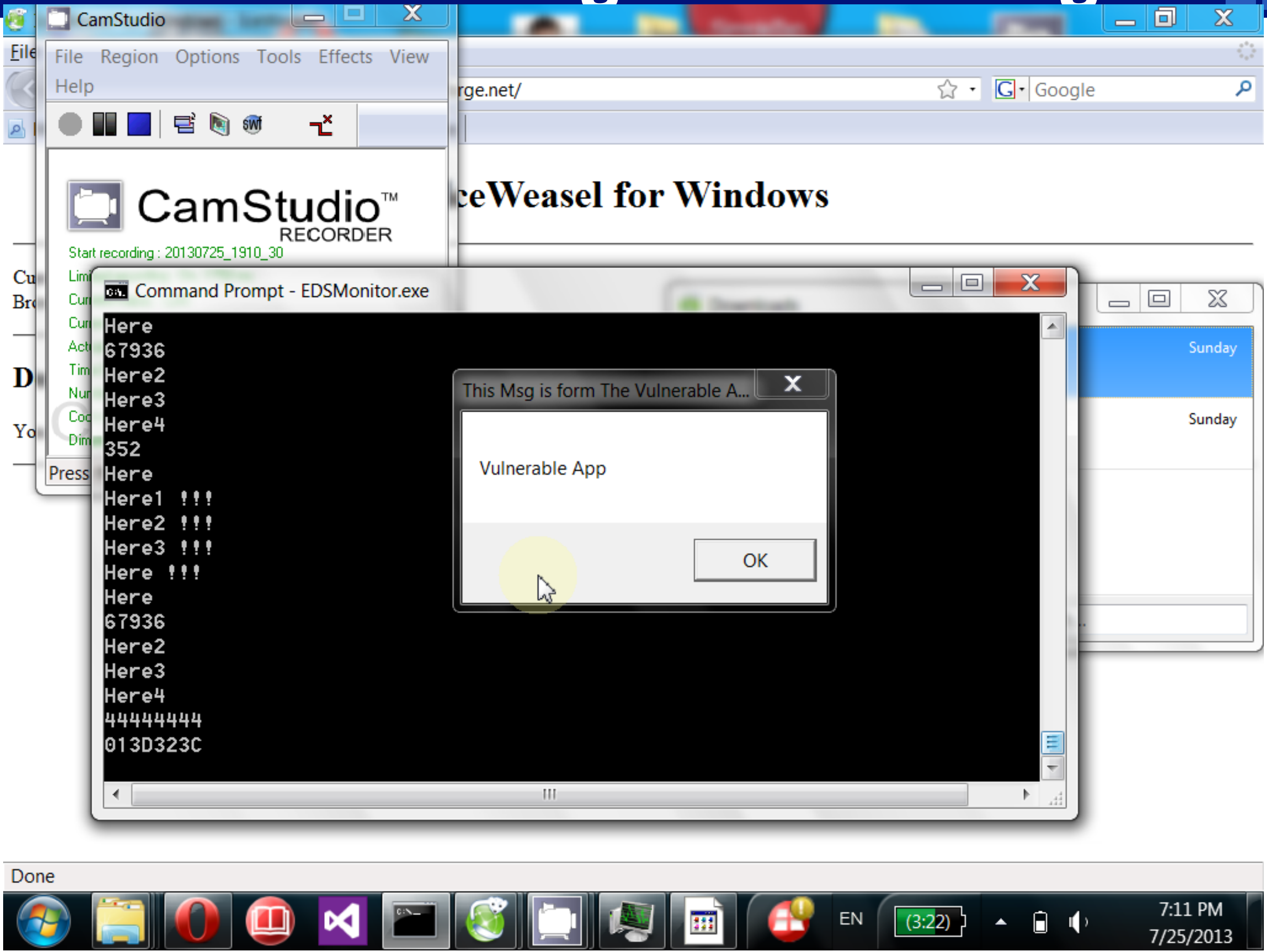
```
Stage 3 Scanning ...
Nothing Found
No ROP Chain Found
Scoring System:
-------------
There's a Return Address: Yes
There's invalid constant variables: No
There's main constant variables: No
There's Next Call Stack: No
There's Next SEH in the caller module: Yes
There's a near ret instruction after the call: No
There's NULL parameter values: Yes
The Main Parameter is in the Stack: No
There's Shellcode: No
There's ROP Chain: No
----------------------
Final Score: 2
296 msecs
```

Demo: a Vulnerable application



```
#include <iostream>
#include <windows.h>
#include <shellapi.h>
using namespace std;

int VulnerableApp(char* Arg,char* x,char* y,char* z,int l);

static unsigned long table[56] = {
0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444,
0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444,
0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444,
0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444,
0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444,
0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444,
0x44444444, 0x44444444, 0x44444444, 0x7E4507EA, 0x44444444, 0, 0, 0x44444444};    //Address of ShellExecuteA

void PreparingTheBuffer()
{
   DWORD Address = (DWORD)GetProcAddress(LoadLibrary("shell32.dll"),"ShellExecuteA");
   //cout << (int*)Address << "\n";
   table[51] = Address;
}

int main (int argc, char *argv[])
{
  PreparingTheBuffer();
  VulnerableApp((char*)table,0,0,"cmd.exe",0);
  return 0;
}


int VulnerableApp(char* Arg,char* x,char* y,char* z,int l)
{
    char buf[200];
    MessageBox(0,"Vulnerable App","This Msg is form The Vulnerable App",0);
    if (Arg != NULL)strncpy(buf,Arg,208);
    return 0;
```

Demo: Running and Hooking it



CamStudio
File Region Options Tools Effects View
Help
CamStudio™ RECORDER
Start recording : 20130725_1910_30
Press
eWeasel for Windows
Command Prompt - EDSMonitor.exe
Sunday
Sunday
Done
EN
(3:22)
7:11 PM
7/25/2013

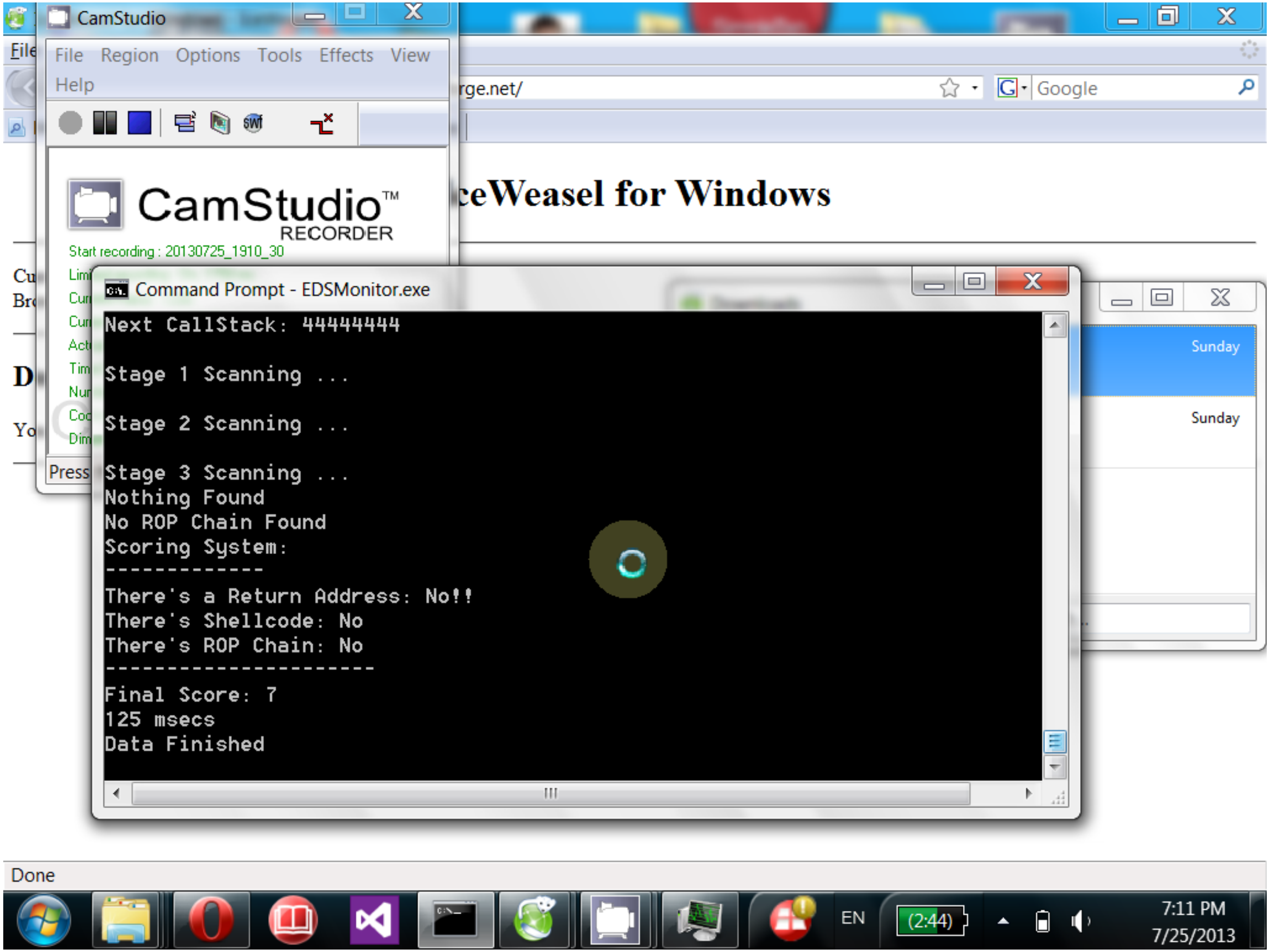
```
Here
67936
Here2
Here3
Here4
352
Here
Here1 !!!
Here2 !!!
Here3 !!!
Here !!!
Here
67936
Here2
Here3
Here4
44444444
013D323C
```

This Msg is form The Vulnerable A...

Vulnerable App

OK

Demo: The Action Scoring and Detection



Command Prompt - EDSMonitor.exe

```
Next CallStack: 44444444

Stage 1 Scanning ...

Stage 2 Scanning ...

Stage 3 Scanning ...
Nothing Found
No ROP Chain Found
Scoring System:
-------------
There's a Return Address: No!!
There's Shellcode: No
There's ROP Chain: No
----------------------
Final Score: 7
125 msecs
Data Finished
```

SEH Mitigation

- SEH is a linked list of pointers to functions handle an error

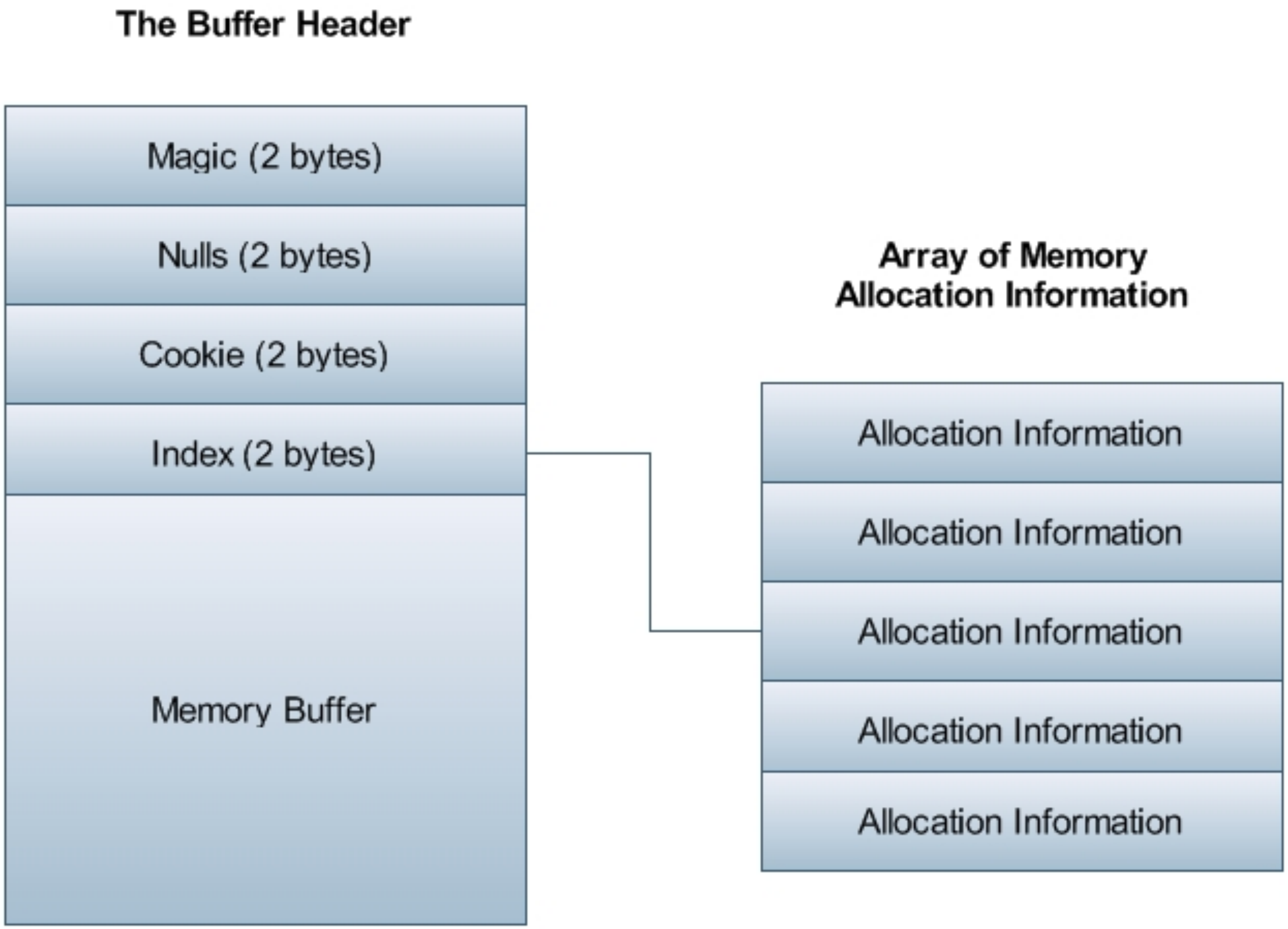
- Very basic Mitigation
- Saves the SEH Linked List
- Check if it ends differently

Mitigations For Heap

- We mitigate these attack vectors:
  - Heap Overflow
  - Heap Spray
  - Heap Use After Free
- Hooks GlobalAlloc and jemalloc
- Create a new Header for memory allocations

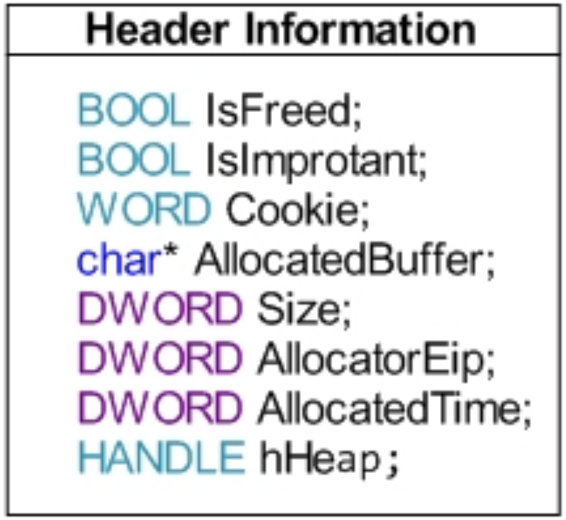
New Header Design

❖It’s Divided Into 2 Headers



Design of Buffer Header

- This is a Header in a separate Buffer
- It points to the buffer
- It get the Caller Module and the allocation Time
- It checks for vtable inside the buffer and Mark it as Important
- It reset everything in ~ 2 secs



Header Information

```
BOOL IsFreed;
BOOL IsImprotant;
WORD Cookie;
char* AllocatedBuffer;
DWORD Size;
DWORD AllocatorEip;
DWORD AllocatedTime;
HANDLE hHeap;
```

Overflow Mitigation

- It checks for:
  - **Nulls**: to stop the string overwrite
  - **Cookie**: to stop managed overwrite
- It’s used mainly against jemalloc

HeapSpray Mitigation

- **It searches for Allocations:**
  - Many Allocations from the same Module
  - Large Memory Usage
  - In very small time
- **Take 2 random buffers**
- **Scan for shellcode and ROP chains**

Use-After-Free Mitigation

- Scans for vtable inside buffers
- Delay the free for these buffers
- Wipe them with 0xBB
- Free them at the end of the slot ~ 2 secs
- Detect Attacks when access 0xBB in Heap

Put All together

- It does 2 type of scanning:
  - **Critical Scanning**: when calls to an API to check ROP Attack or detect HeapSpray .. etc
  - **Periodical Scanning**: That's the monitoring system

Scoring System

- It’s based on the Mitigation
- It stop the known Attacks and terminate the Process
- Alert for suspicious Inputs
- Take Dump of the Process

Monitoring System

- **It scans Periodically**
- **Checks for possible Attacks**
- **Like:**
  - Check Executable Places in Stack
  - Check Executable Places in Memory Mapped Files
  - Search for ROP Chains and Shellcode in Stack and Heap
  - Check Threads running in place outside memory
  - And many more

Future Work

- We are planning to create a central Server
- Receives Alerts and warning
- Monitoring Exploitations on client machine
- With a graphical Dashboard

Future Work: Dashboard

- The Dashboard includes Suspicious Processes in all Machines
- Includes the files loaded inside the suspicious processes (PDF, DOC … etc)
- Includes IPs of these processes connect to (after review the Privacy policy)

Future Work: Dashboard

- EDS will become your Memory and Exploitation Monitor.
- Will correlate with your network tools
- Will be your defense inside the client
- More Intelligent than Antivirus
- Better Response

Dashboard: What you can Detect

- **Using this Dashboard you can detect:**
  - Suspicious PDF or Word File many people opened it:
    it could be an email sent to many people in the company

Dashboard: What you can Detect

- **Using this Dashboard you can detect:**
  - In small time … IE for many employees become suspicious with similar shellcode:

    could be a suspicious URL visited by a phishing mail

Dashboard: What you can Detect

- **Using this Dashboard you can detect:**
  - You can detect suspicious IPs did a scanning over your network and now suspicious processes connect to it

Development

- The EDS is based on SRDF
- "Security Research and Development Framework"
- Created by Amr Thabet
- Includes 3 main contributors

SRDF

- development framework
- Support writing security tools
- Anti-Malware and Network Tools
- Mainly in windows and C++
- Now creating linux SRDF and implementation on python

SRDF Features

Parsers:

- PE and ELF Parsers
- PDF Parser
- Andoid (APK or/and DEX) Parser

Static Analysis:

- Include wildcard like YARA
- x86 Assembler and Disassembler
- Android Delivk Java Disassembler

SRDF Features

- **Dynamic Analysis:**
  - Full Process Analyzer
  - Win32 Debugger
  - x86 Emulator for apps and shellcodes
- **Behavior Analysis:**
  - API Hooker
  - SSDT Hooker (for win32)
  - And others

SRDF Features

Network Analysis

- Packet Capturing using WinPcap
- Pcap File Analyzer
- Flow Analysis and Session Separation
- Protocol Analysis: tcp, udp, icmp and arp
- App Layer Analysis: http and dns
- Very Object Oriented design
- Very scalable

SRDF

- Very growing community

- I will present it in vb 2013 BERLIN 2 - 4 October 2013

- Become a part of this growing community

SRDF

- **Reach it at:**
  - Website: www.security-framework.com
  - Source: https://github.com/AmrThabet/winSRDF
  - Twitter: @winSRDF

- **Join us**

What we reach in EDS

- We developed the Mitigations separately
- We tested the Shellcode Scanner on real shellcodes
- Still testing on real world scenarios
- Join us and help us.

Reach Us

- Still there’s no website for EDS
- You can reach us at SRDF Website: www.security-framework.com
- And my Twitter: @Amr_Thabet
- Just mail me if you have any feedback
  - Amr.thabet[@#!*^]owasp.org

Conclusion

- EDS is the new security tool for this Era
- The Last line to defend against APT Attacks
- Still we are in the middle of the Development
- SRDF is the main backbone for it
- Join Us

Big Thanks to

- Jonas Lekyygaurd
- Anwar Mohamed
- Corlan Team
- All Defcon Team
- Big thanks for YOU

Thank You !