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
Now the APT Attack become the major threat

Bypasses all defenses

Standards and Policies doesn’t work

Bypasses IDS, IPS, Firewalls .. etc
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
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

- Firewall
- IDS
- EDS
- Antivirus
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
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
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

- 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
  - Call to previous (Call Delta)
  - Loop Instruction

```
jnb short firefox.001F1948
mov eax, dword ptr [esi]
test eax, eax
je short firefox.001F1941
call eax
add esi, 4
cmp esi, edi
jb short firefox.001F1939
pop edi
pop esi
ret
```
```
mov eax, 55
add eax, ebx
pop ecx
adc edx, wireshar.00568466
lea eax, DWORD PTR [ecx+100]
push eax
ret
```
```
call wireshar.00510492
nop
```
Indication of Possible Shellcode

- High rate of pushes end with flow redirection

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

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

mov edx,esp
fcmovnu st,st(3)
fstenv (28-byte) ptr [edx-C]
pop ecx
dec ecx
dec ecx
dec ecx
dec ecx
```
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 the code
  - Search for Jcc without compare or similar before it.

- Check on % of Nulls and Null-Free
### Shellcode Statistics

<table>
<thead>
<tr>
<th>File Type</th>
<th>Total No of Pages</th>
<th>Infected Pages</th>
<th>Presentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pcap</td>
<td>381</td>
<td>40</td>
<td>2%</td>
</tr>
<tr>
<td>Pcap</td>
<td>11120</td>
<td>543</td>
<td>4%</td>
</tr>
<tr>
<td>Wmv</td>
<td>104444</td>
<td>4463</td>
<td>4%</td>
</tr>
</tbody>
</table>

- 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 use a series of it
- All of them together = a working shellcode
- Used to bypass DEP
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

1. Hooking Here

2. Stack Backtracing

3. Check on the call
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, [xxxxxxxh]
  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

Command Prompt - EDSMonitor.exe 53028

Final Score: 7
219 msecs
Data Finished
API Called: 00000028
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

Company Logo
Demo: The call stack to ShellExecute
Demo: The ShellExecute Params

Command Prompt:

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

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: REGISTER Reg: 00000000 Actual Value: 00000001
- Parameter6: REGISTER Reg: 00000000 Actual Value: 007C6400
- Parameter7: REGISTER Reg: 00000006 Actual Value: 0037EA80
- Parameter8: REGISTER Reg: 00000003 Actual Value: 80000000
Demo: The Action Scoring
```cpp
#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, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444,
    0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444,
    0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444, 0x44444444,

    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 from The Vulnerable App",0);
        if (Arg != NULL) strncpy(buf,Arg,208);
        return 0;
    }
```
Demo: Running and Hooking it
Demo: The Action Scoring and Detection

CamStudio Recorder

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

**The Buffer Header**

- Magic (2 bytes)
- Nulls (2 bytes)
- Cookie (2 bytes)
- Index (2 bytes)

**Array of Memory Allocation Information**

- Allocation Information
- Allocation Information
- Allocation Information
- Allocation Information
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
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
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
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
Using this Dashboard you can detect:

- You can detect suspicious IPs did a scanning over your network and now suspicious processes connect to it
The EDS is based on SRDF
“Security Research and Development Framework”
Created by Amr Thabet
Includes 3 main contributors
- 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
   - Android (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
Very growing community

I will present it in ver 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.
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 Lekyygaard
- Anwar Mohamed
- Corlan Team
- All Defcon Team

Big thanks for YOU
Thank You!