



Introduction to Internet Explorer's Memory Protection

sweetchip@grayhash.com

2015.01.13

문서 정보

제목	Internet Explorer's Memory Protections
날자	2015-01-13
분류	문서
작성자	sweetchip@grayhash.com

Contents

1. Memory Protection
2. Vtguard
 - A. 우회법
3. Protected Free
 - A. 우회법
4. Isolated Heap
 - A. 우회법
5. Conclusion
6. References

1. Memory Protection

IE 는 윈도우 운영체제에 기본으로 설치되어 있는 웹 브라우저이며 현재 사용자 점유율 1 위이다. 하지만 많은 사용자를 보유한 만큼 IE 는 지속적으로 해커들의 Target 이 되고 있다. 실제 IE 는 현재 1 년에 수백 개 이상의 취약점이 발견되고 있다. 하지만 MS 도 그에 맞서 여러 가지의 보호기법을 개발하고 있다. 컴파일러 및 운영체제 자체에서 제공하는 보호기법을 제외하고 지금까지 알려진 보호기법은 Vtguard, Protected Free, Isolated Heap 이다. Vtguard 를 제외한 나머지 2 개의 보호기법은 비교적 최근 (각각 2014 년 6 월, 2014 년 7 월)에 발표되었는데, 두 보호기법 모두 Use-After-Free 취약점에 대항할 수 있도록 만들어진 보호기법이다. 본 문서에서는 IE 의 보호기법들에 대해 간단하게 알아볼 것이다.

2. Vtguard

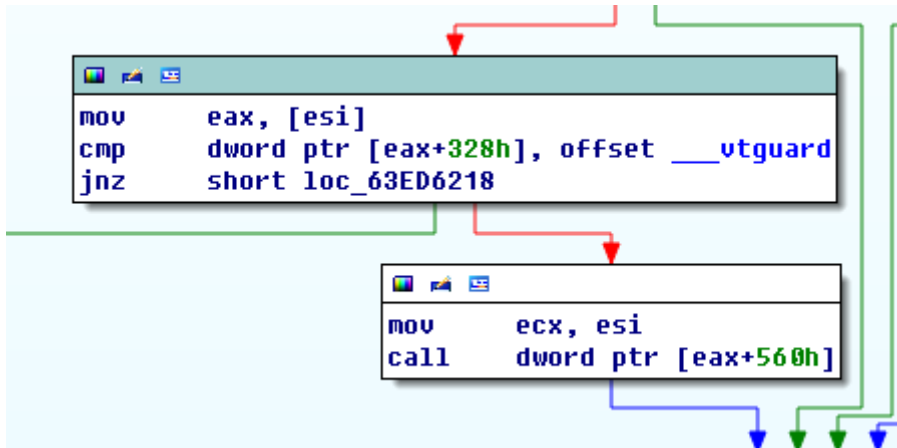
Vtguard 는 Vtable Guard 의 약자이다. 이 보호기법은 Vtable 의 무결성을 검사하기 위한 보호기법이며 현재 IE10, IE11 의 일부 객체에만 적용되어 있다. Use-After-Free 버그와 같이 Vtable 포인터를 손상시킬 수 있는 취약점을 이용하여 Fake Vtable 을 생성한 뒤 참조하도록 만드는 방식의 공격을 방어할 수 있다.

```
.text:695ADAF0 ; const CTableRow::`vftable'
.text:695ADAF0 ??_7CTableRow@6B@ dd offset ?PrivateQueryInterface@CTableRow@UAGJABU_GUID@PAPAX@Z
.text:695ADAF0 ; DATA XREF: CTableRow::~`CTableRow(void)+810
.text:695ADAF0 ; CTableRow::CTableRow(CDoc *)+1210
.text:695ADAF0 ; CTableRow::PrivateQueryInterface(_GUID const &,void * *)
.text:695ADB00 dd offset ?PrivateAddRef@CElement@UAGKXZ ; CElement::PrivateAddRef(void)
.text:695ADB04 dd offset ?PrivateRelease@CElement@UAGKXZ ; CElement::PrivateRelease(void)
.text:695ADB08 dd offset ?PrivateGetTypeInfo@Count@CBBase@UAGJPAI@Z ; CBBase::PrivateGetTypeInfoCount(uint *)
.text:695ADB0C dd offset ?PrivateGetTypeInfo@CBBase@UAGJPAUITypeInfo@UAGKXZ ; CBBase::PrivateGetTypeInfo(uint,ulong,ITypeInfo *)
.text:695ADB10 dd offset ?PrivateGetIDsOfNames@CBBase@UAGJABU_GUID@PAPAGIKPAJ@Z ; CBBase::PrivateGetIDsOfNames(_GUID const &,ushort * *)
.text:695ADB14 dd offset ?PrivateInvoke@CBBase@UAGJABU_GUID@KCPAUTagDISPPARAMS@PAPAUTagVARIANT@PAPAUTagEXCEPTIONINFO@PAPAI@Z ; CBBase::PrivateInvokeEx@CBBase@UAGJJKCPAUTagDISPPARAMS@PAPAUTagVARIANT@PAPAUTagEXCEPTIONINFO@PAPAI@Z ; CBBase::PrivateDeleteMemberByName@CBBase@UAGJJKCPAUTagDISPPARAMS@PAPAUTagVARIANT@PAPAUTagEXCEPTIONINFO@PAPAI@Z ; CBBase::PrivateDeleteMemberByDispID@CBBase@UAGJJKCPAUTagDISPPARAMS@PAPAUTagVARIANT@PAPAUTagEXCEPTIONINFO@PAPAI@Z ; CBBase::PrivateDeleteMemberByDispID(ushort *,ulong,ulong *)
.text:695ADB20 dd offset ?PrivateDeleteMemberByDispID@CBBase@UAGJJKCPAUTagDISPPARAMS@PAPAUTagVARIANT@PAPAUTagEXCEPTIONINFO@PAPAI@Z ; CBBase::PrivateDeleteMemberByDispID(ulong,ulong,ulong *)
.text:695ADB24 dd offset ?PrivateDeleteMemberByDispID@CBBase@UAGJJKCPAUTagDISPPARAMS@PAPAUTagVARIANT@PAPAUTagEXCEPTIONINFO@PAPAI@Z ; CBBase::PrivateDeleteMemberByDispID(ulong,ulong,ulong *)
.text:695ADB28 dd offset ?PrivateGetMemberProperties@CBBase@UAGJJKCPAUTagDISPPARAMS@PAPAUTagVARIANT@PAPAUTagEXCEPTIONINFO@PAPAI@Z ; CBBase::PrivateGetMemberProperties(ulong,ulong,ulong *)
.text:695ADB2C dd offset ?PrivateGetMemberName@CBBase@UAGJJKCPAUTagDISPPARAMS@PAPAUTagVARIANT@PAPAUTagEXCEPTIONINFO@PAPAI@Z ; CBBase::PrivateGetMemberName(ulong,ushort *)
.text:695ADB30 dd offset ?PrivateGetNextDispID@CBBase@UAGJJKCPAUTagDISPPARAMS@PAPAUTagVARIANT@PAPAUTagEXCEPTIONINFO@PAPAI@Z ; CBBase::PrivateGetNextDispID(ulong,ulong,ulong *)
.text:695ADB34 dd offset ?PrivateGetNameSpaceParent@CBBase@UAGJJKCPAUTagDISPPARAMS@PAPAUTagVARIANT@PAPAUTagEXCEPTIONINFO@PAPAI@Z ; CBBase::PrivateGetNameSpaceParent(IUnknown *)
.text:695ADB38 dd offset ?PrivateEnumerateTrackedObjects@CBBase@UAGJJKCPAUTagDISPPARAMS@PAPAUTagVARIANT@PAPAUTagEXCEPTIONINFO@PAPAI@Z ; CBBase::PrivateEnumerateTrackedObjects(ulong *)
.text:695ADB3C dd offset ?PrivateSetTrackingAlias@CBBase@UAGJJKCPAUTagDISPPARAMS@PAPAUTagVARIANT@PAPAUTagEXCEPTIONINFO@PAPAI@Z ; CBBase::PrivateSetTrackingAlias(tagVARIANT *)
.text:695ADB40 dd offset ?PrivateGetTrackingAlias@CBBase@UAGJJKCPAUTagDISPPARAMS@PAPAUTagVARIANT@PAPAUTagEXCEPTIONINFO@PAPAI@Z ; CBBase::PrivateGetTrackingAlias(tagVARIANT *)
.text:695ADB44 ; __vtguard
.text:695ADB48 dd offset ??_ECTableRow@UAGPAPAX@Z ; CTableRow::~`vector deleting destructor'(uint)
.text:695ADB4C dd offset ?GetClassDesc@CWebGLActiveInfo@UBEPBUCLASDESC@CBBase@UAGKXZ ; CWebGLActiveInfo::GetClassDesc(void)
.text:695ADB50 dd offset ?Passivate@CTableRow@UAGKXZ ; CTableRow::Passivate(void)
.text:695ADB54 dd offset ?IsRootObject@CBBase@UAGKXZ ; CBBase::IsRootObject(void)
```

위 사진에서 볼 수 있는 것 처럼 Vtable 안에 __vtguard 함수를 한 개 이상 삽입한다.

```
.text:63D2FD40 ; ===== SUBROUTINE =====
.text:63D2FD40
.text:63D2FD40
.text:63D2FD40 __vtguard proc near ; DATA XREF: .text:6358184010
.text:63D2FD40 ; .text:63581A4010 ...
.text:63D2FD40 mov edi, edi
.text:63D2FD4F int 3 ; Trap to Debugger
.text:63D2FD50 retn
.text:63D2FD50 __vtguard endp
.text:63D2FD50
.text:63D2FD50 ; -----
```

Mshhtml.dll 의 내부를 살펴보면 __vtguard 함수를 찾아볼 수 있으며 내용은 Int 3 이 전부이다.



그 이유는 `__vtguard` 함수의 주소를 쿠키 값으로 사용하기 때문이다. 매번 dll 을 로딩할 때마다 ASLR 로 인해서 Base Address 가 바뀌게 되기 때문에 공격자는 `__vtguard` 함수의 주소 값을 정확히 알 수 없다. 즉, `__vtguard` 는 ASLR Entropy 를 갖는다고 볼 수 있다.

```
.text:643F5AA2 loc_643F5AA2: ; CODE XREF: CDomRange::CBoundaryPoint::GetContainer(CDOMNode * *)+32fj
.text:643F5AA2          push  1
.text:643F5AA4          call  ___report_securityfailure
.text:643F5AA9          int   3 ; Trap to Debugger
.text:643F5AA9 ?GetContainer@CBoundaryPoint@CDomRange@@@AEJPAUCDOMNode@@@Z endp
```

쿠키 값 대조로 Vtable 이 손상된 것을 발견하면 위와 같이 프로그램이 종료된다.

2.A 우회법

- Vtguard 는 ASLR 과 마찬가지로 100% 우회할 수 있는 일반적인 방법은 없다. 대신 특정 상황에서 우회할 수 있는 방법이 몇 가지 존재한다. Vtguard 특성상, **모든 객체에 Vtguard 가 적용되지 않았다는 점에 착안하여 Vtguard 체크 이전에 Vtguard 가 적용되지 않은 다른 객체를 이용하도록 조종할 수 있다는 특정 상황에서** 우회가 가능하다.

이에 대한 내용은 <http://blog.sweetchip.kr/360> 에서 확인할 수 있다. 위 상황에 대해서 간단하게 설명하면, 객체 안에 다른 객체를 가리키는 포인터가 있거나 기타 2 차 Memory Corruption 을 발생시킬 수 있을 만한 값이 있다. 그래서 만약 해커가 Freed Object 를 다시 채울 수 있고 객체 내의 Vtguard 가 적용되지 않은 객체의 포인터를 사용하도록 유도할 수 있다면 Vtguard 를 우회할 수 있다.

- 다른 간단한 방법은 Memory leak 혹은 information leak 이 가능한 버그를 이용하는 방법이다. Vtguard 는 Mshtml.dll 의 `_vtguard` 함수의 주소를 쿠키 값으로 사용하므로 Mshtml.dll 의 Base Address 와 현재 Mshtml 의 `_vtguard` 함수 오프셋을 더하면 쿠키 값으로 사용할 수 있다. 해당 오프셋 값은 mshtml 버전마다 일부 다를 수는 있지만 같은 버전일 경우에는 같다.

최근 환경에서 Use after free 류의 버그를 공략하기 위해서는 ASLR 을 우회하기 위해 Memory leak 을 유도할 수 있는 취약점이 따로 필요하다. 즉, memory leak 은 거의 대부분의 공격에서 필수적이라 볼 수 있으며 이를 통해 자연스럽게 Vtguard 도 우회할 수 있게 된다.

3.Protected Free

2014 년 6 월 MS 에서는 두 번째 메모리 보호기법을 내놓았다. 이 보호기법은 가장 큰 취약점 이슈인 Use-After-Free 취약점을 겨냥하여 만들어진 방법이다. 보호기법 원리 자체는 간단하다. 현재 Free 할 메모리의 크기가 100,000 바이트 이상일 경우에 한번에 Free 시키는 것이다. 그 전까지는 Free 시키지 않는다. 또한 메모리 블록을 zero 로 채운다.

```
.text:63D17CB7 loc_63D17CB7: ; CODE XREF: MemoryProtection::  
.text:63D17CB7          push    ebx          ; Size  
.text:63D17CB8          push    0            ; Val  
.text:63D17CBA          push    edi          ; Dst  
.text:63D17CBB          call   _memset
```

이 뿐만 아니라 Free 하려는 객체 포인터가 여전히 현재 Stack 에 일부분으로 남아있다면 실제로 Free 시키지 않는다. 현재 일부 Use-After-Free 버그는 Free 될 객체가 스택에 남아있는 경우가 있는데 이때 보호기법으로 인해서 실제 Free 함수 호출이 불가능 하므로 다른 방법으로 버그를 트리거 하지 않는 이상 Exploit 이 어렵다. 하지만 이전의 상황과는 달리 Freed Object 의 포인터가 스택에 남아있지 않는 UAF 버그의 경우 여전히 우회가 가능하다.

<http://h30499.www3.hp.com/t5/HP-Security-Research-Blog/Efficacy-of-MemoryProtection-against-use-after-free/ba-p/6556134#VLUWAiusXnM> 에 Protected Free 내용이 있으니 참고하도록 하자.

3.A 우회법

Protected Free 는 Free 할 힙 메모리의 크기가 총 10 만 바이트 이상일 경우 한번에 Free 시키는 방식이다. 그러므로 실제 free() 함수를 호출하기 위해서 10 만 바이트 이상의 메모리를 강제로 할당하고 강제로 Free 시키는 방법을 이용하면 실제 Free 를 유도할 수 있다.

```
function CollectGarbage2()  
{  
    CollectGatbage();  
    var button = document.createElement("button");  
    button.title = new Array(100000).join("0");  
    button.title = null;  
    CollectGarbage();  
}
```

위 코드는 메모리에 10 만 바이트의 문자열을 만들고 강제로 지운 뒤, 가비지 컬렉션을 하는 코드이다. 해당 코드를 강제 Free 가 필요한 지점에 넣으면 강제로 Free 를 호출하는 것을 볼 수 있다. 위 코드는 K33nTeam 이 Protected Free 를 우회하는데 사용한 코드이다. 간단하게 구글링 해보면 이와 비슷한 코드를 많이 볼 수 있을 것이다. 일부 해커는 객체를 수백 개씩 만들고 지우는 등 방법을 이용했다.

간단하게 Free 되는 과정을 다음 예제로 알아보자.

```
<html>
<head>
  <script type="text/javascript">
    function gc() {
      var COptionElement = document.createElement("option");
      COptionElement.title = new Array(100000).join("A");
      COptionElement.title = null;
      COptionElement = null;
      CollectGarbage();
    }
    var arr = new Array();
    alert("start");
    for(var i = 0; i<10; i++)
    {
      arr[i] = document.createElement("button"); // 1
    }
    alert("created");
    for(var i = 0; i<10; i++)
    {
      arr[i] = null;
    }
    arr = null;
    CollectGarbage() // 2
    alert("memset with null");
    gc();
    gc(); // 3
    alert("freed.");
  </script>
</head>
<body>
  hello world :D
</body>
</html>
```

위와 같은 소스코드가 있다고 하자.

우선 windbg 로 CreateElement 함수에 브레이크 포인트를 설정하고 객체 주소를 출력하도록 하는 스크립트를 작성한다. (아래는 MS14-080 패치에 적용된 MSHTML.DLL 이다. 매번 패치 때마다 Offset 이 바뀔 수 있다는 점을 참고하자.)

```
bu MSHTML+004a2104 ".printf W"//cButtonElement : %pW", eax;echo;g"
```

위 코드는 MSHTML+004a2104 지점에 도착했을 경우 eax 값을 형식에 맞게 출력해 주는 코드이다.

주석 1 번에선 CButtonElement 객체를 생성한다.

```
0:018> g
//cButtonElement : 038fe5f0
//cButtonElement : 038fe650
//cButtonElement : 038fe6b0
//cButtonElement : 038fe710
//cButtonElement : 038fe770
//cButtonElement : 038fe7d0
//cButtonElement : 038fe830
//cButtonElement : 038fe890
//cButtonElement : 038fe8f0
//cButtonElement : 038fe950
(428.1b94): Break instruction exception - code 80000003 (first chance)
eax=7ef72000 ebx=00000000 ecx=00000000 edx=7707f8ea esi=00000000 edi=00000000
eip=76ff000c esp=0afef84c ebp=0afef878 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
ntdll!DbgBreakPoint:
76ff000c cc          int     3
0:018> dc 038fe770
038fe770  610816f4 00000001 00000001 00000008  ...a.....
038fe780  0348c1d8 00000000 049fad50 00000000  ..H....P.....
038fe790  00000014 12c00400 00000000 00000000  .....
038fe7a0  034939f0 00000000 00000000 610a9b70  .9I.....p..a
038fe7b0  00000000 00000000 00000000 00000000  .....
038fe7c0  00000000 00000000 43d0ca98 88000000  .....C....
038fe7d0  610816f4 00000001 00000001 00000008  ...a.....
038fe7e0  0348c208 00000000 049fad80 00000000  ..H.....
0:018> !heap -p -a 038fe770
address 038fe770 found in
  _HEAP @ 38d0000
HEAP_ENTRY Size Prev Flags  UserPtr UserSize - state
038fe768 000c 0000 [00] 038fe770 00058 - (busy)
MSHTML!CButton::'vftable'
```

CButton 객체가 생성된 것을 볼 수 있다.

주석 2 에서는 배열에 만든 객체를 전부 삭제한 뒤, CollectGarbage() 를 한번 호출하여 Protected Free 관련 함수를 호출하도록 한다

```

0:018> dc 038fe770
038fe770 00000000 00000000 00000000 00000000 .....
038fe780 00000000 00000000 00000000 00000000 .....
038fe790 00000000 00000000 00000000 00000000 .....
038fe7a0 00000000 00000000 00000000 00000000 .....
038fe7b0 00000000 00000000 00000000 00000000 .....
038fe7c0 00000000 00000000 43d0ca98 88000000 .....C...
038fe7d0 00000000 00000000 00000000 00000000 .....
038fe7e0 00000000 00000000 00000000 00000000 .....

0:018> !heap -p -a 038fe770
address 038fe770 found in
_HEAP @ 38d0000
HEAP_ENTRY Size Prev Flags UserPtr UserSize - state
038fe768 000c 0000 [00] 038fe770 00058 - (busy)

```

아까와 달리 객체의 값들이 전부 null 이 되었다. 하지만 Heap 의 상태를 보니 여전히 Busy 상태 (할당되어 있음)이다.

```

0:018> dc 038fe770
038fe770 0000003e 00000000 00000000 00000000 > .....
038fe780 00000000 00000000 00000000 00000000 .....
038fe790 00000000 00000000 00000000 00000000 .....
038fe7a0 00000000 00000000 00000000 00000000 .....
038fe7b0 00000000 00000000 00000000 00000000 .....
038fe7c0 00000000 00000000 43d0ca98 80000000 .....C...
038fe7d0 0000004a 00000000 00000000 00000000 J.....
038fe7e0 00000000 00000000 00000000 00000000 .....

0:018> !heap -p -a 038fe770
address 038fe770 found in
_HEAP @ 38d0000
HEAP_ENTRY Size Prev Flags UserPtr UserSize - state
038fe768 000c 0000 [00] 038fe770 00058 - (free)

```

마지막 주석 3 의 명령어는 gc() 함수인데 강제로 10 만 바이트의 데이터를 만든 뒤 데이터를 삭제하여 Free 를 유도한다. 그 결과 성공적으로 객체가 Free 된 것을 볼 수 있다. Protected Free 는 exp-sky 와 KeenTeam 에서 C 언어 코드로 복원해둔 코드가 있어서 좀 더 알고 싶으면 다음 링크를 참고하면 된다.

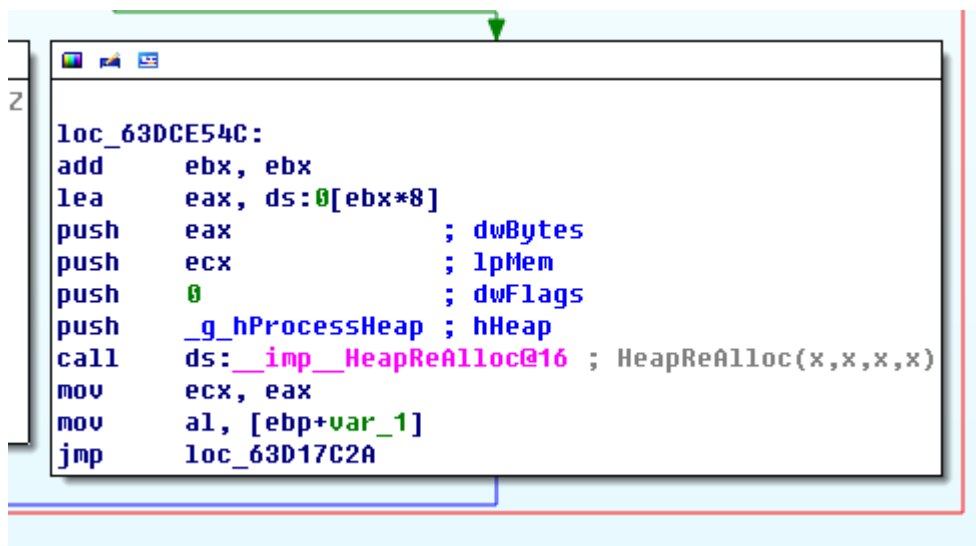
Link1 : <https://github.com/exp-sky/MemoryProtection/blob/master/MemoryProtection.cpp#L362>

Link2: <https://github.com/promised-lu/MemoryProtection/blob/master/MemoryProtection/MemoryProtection.cxx>

4.Isolated Heap

Protected Free 가 적용되고 한달 뒤 Isolated Heap 이 적용되었다. 이 보호기법으로 인해서 한동안 많은 해커들이 상당히 고생했는데, 현재는 KeenTeam 이나 exp-sky 등 해커들이 우회 법을 공개해 둔 상태이다. Isolated Heap 은 Use-After-Free 취약점을 대응하기 위해 만들어진 보호기법이다. 이전에는 Internet Explorer 에서 exploit 이 쉬웠던 편에 속했던 것은 (일반적인 경우) Use-After-Free 였다. 비슷한 크기의 힙을 LFH 라는 윈도우의 힙 매니저의 특성을 이용하여 Free 된 힙 메모리에 다시 원하는 내용을 string 등으로 쉽게 채워 넣어서 EIP 를 조작하는 방법이 유행해서 많은 취약점들이 0-day 취약점으로 악용되었다. 하지만 이 보호기법이 나온 뒤로는 Exploit 이 이전보다 매우 어려워졌고 실제 Use-After-Free 를 이용하여 공격에 사용되는 경우도 크게 줄었다.

패치 이전에 IE 에서 힙에 객체를 할당하는 코드는 다음과 같았다.



```
loc_63DCE54C:
add     ebx, ebx
lea     eax, ds:0[ebx*8]
push   eax           ; dwBytes
push   ecx           ; lpMem
push   0             ; dwFlags
push   _g_hProcessHeap ; hHeap
call   ds:__imp__HeapReAlloc@16 ; HeapReAlloc(x,x,x,x)
mov     ecx, eax
mov     al, [ebp+var_1]
jmp     loc_63D17C2A
```

위처럼 HeapAlloc, HeapReAlloc 등의 함수에 인자로 _g_hProcessheap 이라는 인자를 넘겨줬는데 Isolated Heap 이 적용된 순간부터 다음과 같이 바뀌었다.

```
.text:636CCFAC      mov     esp, esp
.text:636CCFAE      push   5Ch          ; dwBytes
.text:636CCFB0      push   8            ; dwFlags
.text:636CCFB6      push   _g_hIsolatedHeap ; hHeap
.text:636CCFB8      call   _HeapAlloc@12 ; HeapAlloc(x,x,x)
```

DOM 객체나 레이아웃 등 일부 객체에 위 IsolatedHeap 플래그를 넣고 string 으로 이루어진 데이터 같은 일반 값을 메모리에 할당할 때는 Processheap 플래그를 넣는다. 이렇게 힙을 따로 분리함으로써 ProcessHeap 객체와 IsolatedHeap 객체는 서로 힙 영역에 접근이 불가능하게 되었다. 그러므로 모든 힙에 ProcessHeap 이 적용되어 있던 예전처럼 String 값을 이용하여 다시 그 자리에 할당 하는 방법은 더 이상 불가능 하다.

* 과거의 Use-After-Free 취약점을 공격하는 방법은 다음을 참고하면 된다.

4.A 우회법

1. Isolated Heap 이 적용되지 않은 객체의 Use-After-Free 취약점은 여전히 전통적인 방법 (string 으로 채우는 등)으로 exploit 이 가능하다.
2. 버그로 인하여 Free 된 Isolated Heap 이 적용된 객체를 Isolated Heap 이 적용된 객체로 다시 그 자리를 채우는 방법으로 Exploit 이 가능하다.

위 두 가지 방법 이외에도 더 많은 방법이 있을 수 있다. 하지만 확실히 현재 밝혀진 방법은 위 2 가지 이다.

이제 각 방법을 알아보자. 우선 위의 2 가지 상황 모두 사전에 Protected Free 를 먼저 완벽히 우회 해야 하고 Free 된 객체를 완벽히 컨트롤(공격자가 원하는 시점에 Free 를 시키는 등) 할 수 있어야 한다.

1. Isolated Heap 은 대부분 유저가 직접 접근 가능한 DOM 객체 같은 일부 객체에만 적용되어 있기 때문에 Isolated Heap 이 적용되어 있지 않는 객체의 경우 여전히 이전에 String 등으로 채우는 방법으로 가능하다.
2. Isolated Heap 이 적용된 객체는 Isolated Heap 객체로 채울 수 있다. DOM 객체 중 Area 객체가 있다. 이 객체 중 일부 값인 0x4c(Left), 0x50(Top), 0x54(Right), 0x58(Bottom) 총 16 바이트를 공격자가 직접 컨트롤할 수 있다. 그러므로 UAF 버그로 인하여 Free 시킬 객체를 공격자가 정확히 컨트롤할 수 있는 상황이라면 LFH 등을 이용하여 Free 된 객체의 자리에 Area 객체의 0x4c~0x58 지점에 있는 값을 넣음으로 써 충분히 공격이 가능하다.

2 번 방법의 CAreaElement 객체를 살펴보자

```
var area = document.createElement('area');
area.shape = "rect";
area.coords = "1094795585,1094795585,1094795585,1094795585"; // 0x41414141
```

Area 객체를 새로 생성하고 Windbg 로 메모리를 살펴보자.

```

0:018> !heap -p -a 012d37d8
address 012d37d8 found in
_HEAP @ 12b0000 <- g_hIsolatedHeap
HEAP_ENTRY Size Prev Flags UserPtr UserSize - state
012d37d0 000e 0000 [00] 012d37d8 00064 - (busy)
MSHTML!CAreaElement::`vftable'

0:018> dc 012d37d8
012d37d8 611e7118 00000001 00000001 00000008 .q.a.....
012d37e8 033cd308 00000000 031fa2a0 00000000 ..<.....
012d37f8 00000007 00400400 00000000 00000000 .....@.....
012d3808 0340b640 611e70f4 012d380c 00000000 @.@.p.a.8-....
012d3818 00000000 00000000 00000000 41414141 .....AAAA
012d3828 41414141 41414141 41414141 00000000 AAAAAAAAAAAAA...
012d3838 00000001 00000000 3eb063cd 0c003087 .....c.>.0..
012d3848 000004d4 000002b7 012d36dc 012d39b8 .....6-..9-.
0:018> dc
012d3858 012d36f4 012d3584 00000040 00000000 .6-..5-..@.....
012d3868 0341c510 06000001 00000000 00000000 ..A.....
012d3878 30b063c3 0800308e 61272bc4 00000001 .c.0.0..+'a...
012d3888 00000001 00000008 00000000 00000000 .....
012d3898 00000000 012d38c8 00000012 00c40000 ....8-.....
012d38a8 90000e00 00060004 012d3010 00000000 .....0-....
012d38b8 00000000 00000000 34b063c7 08003080 .....c.4.0..
012d38c8 012d3880 012d34b8 70220012 00000851 .8-..4-..."pQ...
0:018> dc g_hIsolatedHeap L1
621332e0 012b0000 ..+.

```

위처럼 메모리에 의도한 16 바이트가 들어간 것을 볼 수 있다. 이를 이용하여 LFH 를 이용하는 등의 방법을 사용하여 Free 된 객체의 메모리를 Area 객체로 다시 덮는 방법을 이용할 수 있다. 해당 문서에는 그 기술은 다루지 않을 것이다. 대신 아래 링크에서 공격하는 컨셉 및 실제 공격하는 과정을 찾아볼 수 있다.

<http://k33nteam.org/blog-4-use-after-free-not-dead-in-internet-explorer-part-1.htm>

http://hitcon.org/2014/downloads/P2_01_Keen%20Team%20-%20New%20Exploit%20Mitigation%20In%20Internet%20Explorer.pdf



5. Conclusion

새로운 보호기법들이 나오면서 비교적 쉽게 Exploit 이 가능했던 취약점들이 이제는 불가능하거나 매우 어려워졌다. 사용자 입장에선 좋은 신호이지만 브라우저의 보안을 연구하는 사람의 입장에선 좀 더 다양한 창의성을 요구하게 되었다. 이처럼 MS 와 같은 벤더는 계속해서 브라우저의 보안을 향상시키고 있으며 Protected Free, Isolated Heap 이 갑자기 등장했던 것처럼 앞으로도 더 많은 보호기법이 등장할 수 있다. 새롭게 출시될 MS 의 Spartan 브라우저에는 어떠한 보호기법이 나타나게 될지 주목할 필요가 있다.



6. Reference

<http://blog.trendmicro.com/trendlabs-security-intelligence/isolated-heap-for-internet-explorer-helps-mitigate-uaf-exploits/>

<http://securityintelligence.com/understanding-ies-new-exploit-mitigations-the-memory-protector-and-the-isolated-heap/>

<https://labs.mwrinfosecurity.com/blog/2014/06/20/isolated-heap-friends---object-allocation-hardening-in-web-browsers/>

<https://kbandla.github.io/posts/2014/Jun/20/Internet%20Explorer%20Isolated%20Heap.html>

<http://h30499.www3.hp.com/t5/HP-Security-Research-Blog/Efficacy-of-MemoryProtection-against-use-after-free/ba-p/6556134#.VLFr58YVFTY>

<http://researchcenter.paloaltonetworks.com/2014/07/beginning-end-use-free-exploitation/>