Immunity, Inc. Advisory

Vulnerability

INSTANTANEA: Wins.exe remote vulnerability.

WINS is a Microsoft NetBIOS name server, that basically eliminates the need for broadcast packet to resolve a NetBIOS computer name to an IP address.

WINS has a feature called WINS replication, where one or more WINS servers exchange information with each other about the computers on their respective networks. WINS replication is done on TCP port 42 using a Microsoft proprietary protocol. During this protocol flow, a memory pointer is sent from server to client, and the client uses that to talk with the server. If a special crafted packet is sent to the server, an attacker can control the pointer and can make it point to an attacker-controlled buffer and eventually write 16 bytes at any location.

The packet that we are sending looks like this:

```
----------------------------
|       size of packet     | (excluding 4 bytes of size field)
----------------------------
|       XX XX FF XX        |
----------------------------
|     real addres pointer  |
----------------------------
|       identified long    |
----------------------------
|          ... (etc)       |
----------------------------
```

The size of the packet is passed as argument to HeapAlloc (wins checks that size is less than 0x2F87F8). The second dword is the condition we have to pass to trigger the bug. Finally the address pointer that from now on we call "myself" points to a special structure used by wins to exchange information between servers.

To exploit it, what we do is try to point “myself” to a buffer that we can control, what we do is send a big packet of about 0x40000 bytes so we can guess where it would be. Once we point to something that we control, we need to point to a specific structure that looks like this:

```
---------------------------- 0
|        WHERE -x048       |
----------------------------
|            ...           | ...
----------------------------
|           WHAT           | 0x24
----------------------------
|           WHAT2          |
----------------------------
|           WHAT3          |
----------------------------
|           WHAT4          |
----------------------------
```

Obviously, where is the address that we want to write to, and what* are the 16 bytes that we are writing to where address.

So we have three problems arise:

(a) How to point exactly to my crafty structure
(b) Where to write
(c) What to write

The (a) point is resolved creating a special structure with "where-0x48" * 9 and what * 14, if we repeat this structure, we could brute force the structure and with less than 3 tries we will have our Write16 primitive. (Note: Access Violations are caught by wins.exe).

The (c) point is resolved guessing an approximate address of the 0x40000 bytes malloc.

Now, (b) point is the hardest value to find, and is related to point (a) and c. Because as Oded Horovitz has documented, and common sense says, when a large amount of bytes is freed, it is returned back to the OS, and the consequence are that our function pointer has to be triggered before HeapFree is executed, so we have to discard PEB function pointer. In order not to loose all the advantages that the big buffer gives us, we try to find the return address by brute forcing the stack.

Useful ollydbg breakpoints (SP3)

```
Breakpoints
Address    Module    Active              Disassembly                    Comment
01012EEC   wins      Always               CALL DWORD PTR DS:
[<&KERNEL32.Create
01013404   wins      Log                 MOV EDI,DWORD PTR DS:[<&KERNEL32.lst
01013413   wins      Log                 MOV ESI,DWORD PTR DS:[<&KERNEL32.lst
01015D93   wins      Log                 CALL DWORD PTR DS:[<&KERNEL32.lstrcp
0101811D   wins      Log                 CALL DWORD PTR DS:[<&KERNEL32.lstrcp
0102117C   wins      Always                PUSH ESI
0102122E   wins      Always                MOV ESI,wins.01026520
01021274   wins      Always                ADD EAX,4
01021294   wins      Always                CMP EAX,-1
```

```
010212AE   wins      Always               ADD EDX,4
010212DA   wins      Always               PUSH wins.01026A68
010212E4   wins      Always               CALL wins.01012ACC
01021368   wins      Always               PUSH wins.01003CAC01021397   wins
Always               JMP wins.010212FF
010213E7   wins      Always               CALL wins.01022C8B              recv 240
01021403   wins      Always               CALL wins.010224AA              recv4
01021423   wins      Always               JNB wins.010212FF
0102143E   wins      Always               CALL <JMP.&WS2_32.#151>
01021460   wins      Always               CALL wins.0102185C
010214CF   wins      Always               DEC ECX
010214E9   wins      Always               JMP SHORT wins.010214C9
010214F7   wins      Always               JMP wins.01021416
01021526   wins      Always               CALL DWORD PTR DS:[<&WS2_32.#1>]
01021563   wins      Always               CALL wins.01012806
0102158A   wins      Always               CALL wins.01012DB1
010215B8   wins      Always               JNZ SHORT wins.010215C3
010215C8   wins      Always               CALL wins.01022040
010215D2   wins      Always               XOR EAX,EAX
01021614   wins      Always               CALL DWORD PTR DS:
[<&KERNEL32.Interl
01021622   wins      Always               MOV DWORD PTR SS:[EBP-4FC],ESI
0102165E   wins      Always               CALL wins.01012DB1
01021676   wins      Always               JE wins.010212FF
0102167F   wins      Always               CALL DWORD PTR DS:[<&WS2_32.#14>]
010216BE   wins      Always               CALL wins.01012806
01021790   wins      Always               JMP wins.010216FC
010217EE   wins      Always               MOV EAX,DWORD PTR SS:[EBP-14]
0102197D   wins      Always               PUSH EBP
0102252B   wins      Always               MOV EAX,DWORD PTR SS:[EBP-4]
010225FE   wins      Always               CALL wins.0102240C
```

Discovery Method

This exploit was discovered by tracing through the processes with Ollydbg and manually analyzing the disassembly by Nicolas Waisman.

Affected

All known versions of Wins.exe are affected. Windows 2000 SP2-4 were tested.

History

Research and Exploited by Immunity Researcher Nicolas Waisman, May, 2004.

Released to VSC May, 2004.

Released to public 26 November, 2004

Detection

Immunity Research has provided a working exploit for this problems, on the standard CANVAS distribution.

For questions or comments, please contact Immunity, Inc. at dave@immunitysec.com, or http://www.immunitysec.com