How Conficker makes use of MS08-067 ?

By Bui Quang Minh & Hoang Xuan Minh

Conficker emerges as a hot topic recently. This is the most widespread virus since Code Red. The news has been talking so much about it that I decided to write an article too.

This article concerns the spreading technique used by this virus, particularly the way it exploits the MS08-067 security vulnerability in the Server Service of Windows.

MS08-067 Technical Details
I have actually had an article discussing about this flaw [1] when it was published but I will do it over again in more details now.

RPC protocol in Server Service supports a remote procedure converting any path (for instance, \\C\Program Files\..\Windows) to Canonicalization path (\\C\Windows). But Windows does not handle well overly long path, resulting in buffer overrun.

To concretize, Windows (svchost process) uses NetpwPathCanonicalize() function of netapi32.dll library to perform the above mentioned operation. The pseudo-code comes following:

```
func _NetpwPathCanonicalize(wchar_t* Path)
{
          // check Path length
          if( !_function_check_length(Path) )
                    return;
          …
          _CanonicalizePathName(Path);
          …
        return;
}

func _CanonicalizePathName(wchar_t* Path)
{
          // protect stack with cookie - /GS
          _save_security_cookie();
          …
          wchar _wcsBuffer[420h];
          …
          // this is the function causing the overrun
          wcscat(wcsBuffer,Path);
          …
          // converting function
          _ConvertPathMacros(wcsBuffer);
          …
          return;
}
```

As we can see from the pseudo-code, NetpwPathCanonicalize() checks the length of the path before passing it into CanonicalizePathName() function. However, CanonicalizePathName() uses wcscat() to copy the path into a local variable (wcsBuffer). The consequence is that the function wouldn't create a buffer-overflow in the first run but it would in the subsequents. For example, the contents of wcsBuffer after each call to this function would be:

- Call 1 : wcsBuffer = "\\a\aaaaa\aaaa\..\..\a"
- Call 2 : wcsBuffer = "\\a\aaaaa\aaaa\..\..\a\\a\aaaaa\aaaa\..\..\a"
- Call 3 : wcsBuffer = "\\a\aaaaa\aaaa\..\..\a\\a\aaaaa\aaaa\..\..\a\\a\aaaaa\aaaa\..\..\a"
- …

So we can definitely overflow Server Service with several calls to NetpwPathCanonicalize() function remotely providing appropriate path length. Up to this point, it seems as if the road had been cleared out.

But two other obstacles appear:

- **Cookie**: The CanonicalizePathName() function was built with /GS option, which protects it with a cookie put before the return address. Whenever the return address is overwritten, so is the cookie and the system therefore knows that a buffer overflow has been encountered.
- **DEP**: the process of Server Service (svchost.exe) is protected with DEP by default. As a result, if Shellcode is put on stack, DEP won't allow code execution.

What exploiting techniques were used by Conficker?

Now let's draw our attention to a function used in CanonicalizePathName(), which is called ConvertPathMacros() by Microsoft. This function does not perform any check against the cookie and hence was taken advantage by Conficker to redirect control to Shellcode.

The article of Microsoft [2] also mentioned the ConvertPathMacros() function but did not describe its role in the exploitation correctly. More precisely, Microsoft pointed out that this function used a local variable to store the buffer and the exploitation would overflow it in order to overwritten the return address of ConvertPathMacros().

But in actuality, ConvertPathMacros() does not have any portion of code that directly copies and overflows such local buffer. It is made possible to overwrite the return address of this function owing to a weakness in its string processing algorithm. As a consequence, wcscpy() function, which is called within ConvertPathMacros(), has its return address overwritten.

For DEP bypassing, Conficker makes use of ZwSetInformationProcess() function to disable DEP in runtime mode. After that, Conficker redirects control to Shellcode on stack.

Conficker uses instructions available in AcGenral.dll library, which is loaded by svchost, to overcome both previous protection mechanisms.

So with this method of exploiting, Conficker just needs to call NetpwPathCanonicalize() one time to successfully attack.

Spreading module of Conficker

Using above exploiting techniques, Conficker can exploit many different Windows versions (XP SP2/SP3, English, Italian,…). With a particular IP address, Conficker will try attacking with a lot of malicious code, each for one version of Windows. This increases the rate of success. Here comes the pseudo-code:

```
func __Thread_Attack (IpAddress)
{
          …
          // Create an Url for shellcode to download virus.
          url = Make_Url_Download();
          …
          While(1)
          {
                    // If connection fails, abort.
                    if( ! IsConnect(IpAddress)) return;
                    …
                    // Create attacking buffer, each time call Make_Buffer(),
                    // a buffer for a particular Windows version will be created.
                    buffer  = Make_Buffer(url, buffer);
                    …
                    // Attack
                    Attack(IpAddress, buffer);

                    // Wait 1 second, if successfully exploit, break from the loop.
                    // if not, try the next exploiting buffer.
                    if( WaitForSingleObject(1000) != WAIT_TIMEOUT ) break;
          }
}
```

Conficker Shellcode activity
- Decode (Xor with 0xC4).
- Get the addresses of necessary API functions: LoadLibrary(), ExitThread().
- Load urlmon.dll library into the process.
- Get the address of URLDownloadToFileA() function in urlmon.dll.
- Download virus from the attacking computer using http protocol.
- Source address used for download: http://xxxxxx:port/xxxxx
- Downloaded virus is saved under the name x.
- Kill the thread (ExitThread).

Which OS are susceptible to Conficker attack
After reversing Conficker, I found 51 Windows versions that could be attacked by Conficker (SP1, SP2 SP3 and languages are considered different versions). One interesting thing is that the addresses of the exploiting module for different versions of Windows used by Conficker are the same as those of metasploit exploit code [3]. This shows high possibility that the virus creator take these addresses from metasploit. The following a list of operating system susceptible to Conficker.

1	Windows 2000.
2	Windows XP SP2/SP3 English.
3	Windows XP SP2/SP3 Arabic.
4	Windows XP SP2/SP3 Taiwan.
5	Windows XP SP2/SP3 Chinese.
6	Windows XP SP2/SP3 Czech.
7	Windows XP SP2/SP3 Danish.
8	Windows XP SP2/SP3 German.
9	Windows XP SP2/SP3 Greek.
10	Windows XP SP2/SP3 Spanish.
11	Windows XP SP2/SP3 Finnish.
12	Windows XP SP2/SP3 French.
13	Windows XP SP2/SP3 Hebrew.
14	Windows XP SP2/SP3 Hungarian.
15	Windows XP SP2/SP3 Italian.
16	Windows XP SP2/SP3 Japanese.
17	Windows XP SP2/SP3 Korean.
18	Windows XP SP2/SP3 Dutch.
19	Windows XP SP2/SP3 Norwegian.
20	Windows XP SP2/SP3 Polish.
21	Windows XP SP2/SP3 Brazilian.
22	Windows XP SP2/SP3 Portuguese.
23	Windows XP SP2/SP3 Russian.
24	Windows XP SP2/SP3 Swedish.
25	Windows XP SP2/SP3 Turkish.
26	Windows 2003 SP1/SP2 English.

Reference
[0] http://security.bkis.vn/
[1] http://bkav.com.vn/tinh_hinh_an_ninh_mang/27/10/2008/6/1896/ (Vietnamese)
[2] http://blogs.technet.com/srd/archive/2009/03/16/gs-cookie-protection...
[3] http://trac.metasploit.com/browser/framework3/trunk/modules/exploits/windows/...