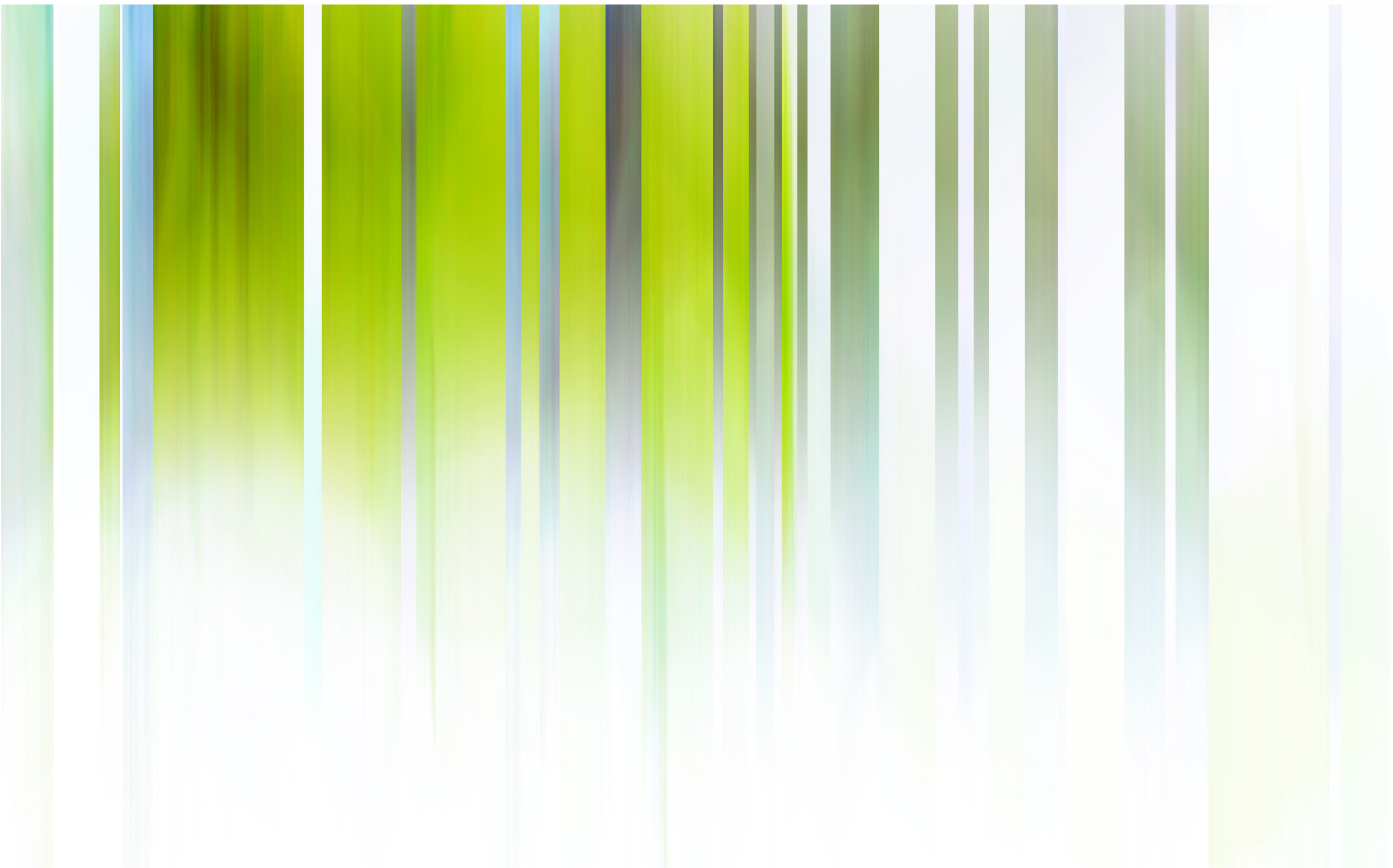
MUNK SCHOOL OF GLOBAL AFFAIRS

UNIVERSITY OF TORONTO

CITIZEN LAB TECHNICAL BRIEF 001

IEXPLORE RAT

BY SETH HARDY | AUGUST 2012



INTRODUCTION

This report describes a remote access trojan (RAT) that three human rights-related organizations taking part in a Citizen Lab study on targeted cyber threats against human rights groups received via email in 2011 and at the end of 2010. Here we refer to it as the IEXPL0RE RAT, after the name of the launcher program. It was first called "Sharky RAT" in Seth Hardy's talk at SecTor 2011. Since then it has also been referred to as c0d0so0 and possibly Backdoor.Briba.

A RAT is a program that allows a remote user full access to a computer. This type of program can be used for legitimate reasons. In these cases, RAT can also stand for remote administration tool. In the case of the IEXPL0RE RAT, the remote user has the ability to record user keystrokes (including passwords), copy and delete files, download and run new programs, and even use the computer's microphone and camera to listen to and watch the user in real-time.

RATs are common in targeted malware attacks against human rights organizations and other NGOs. Targeted attacks with this sort of payload are often referred to as advanced persistent threats (APTs). APTs differ from other traditional computer attacks in that they are designed to be quiet and collect data over time, and act as a starting point for future tracking and compromise of targets. It is not uncommon for an APT infection to persist for months or even years after the malicious program is first run.

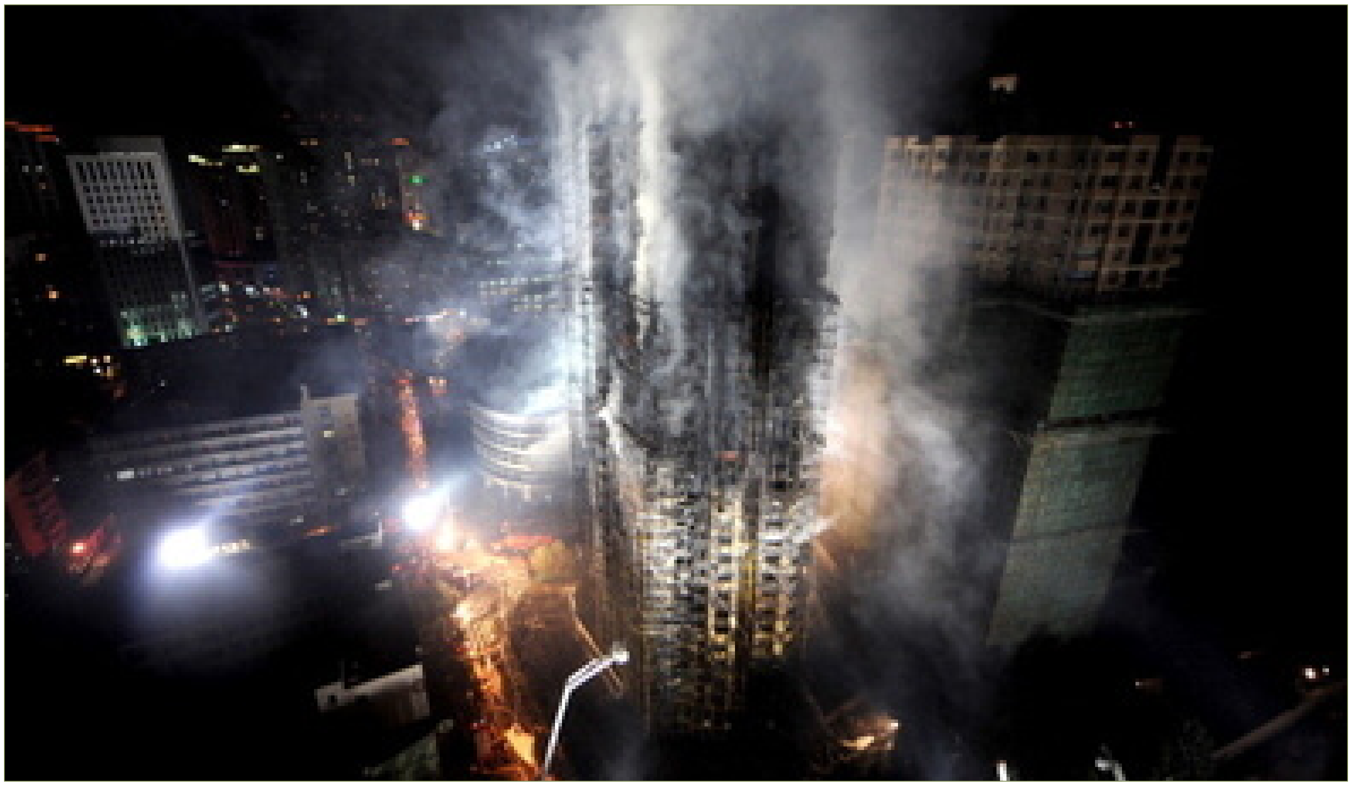
ATTACK VECTOR

Attempted delivery of the malware was via email attachment, employing social engineering techniques. The emails that contained the attached IEXPL0RE RAT were different every time, with a unique email and delivery method used for each attempt, including multiple versions targeted at the same organization. Each email was tailored specifically for the target, both in terms of subject, content, and the way the RAT was attached and hidden.

Organization 1: a human rights NGO received multiple emails with interesting keywords from senders claiming to be from personal friends. These emails included an executable attachment in a password-protected archive, which helps prevent detection by antivirus software. The password was included in the email address.

Organization 2: a news organization operating a website that reports on developments in China, received an email containing a story about a high-rise apartment building fire. Attached to the email were four images and two executable files (.scr extensions) designed to look like images using the Unicode right-to-left override character. When each executable file is run, it will install and launch the malware, drop an image, open the image, and delete itself. The end result is that only an image is left, making the email look more legitimate if the malware is run (figure 1).

FIGURE 1: IMAGE OF A HIGH-RISE FIRE USED TO TRICK RECIPIENTS INTO RUNNING THE MALWARE.

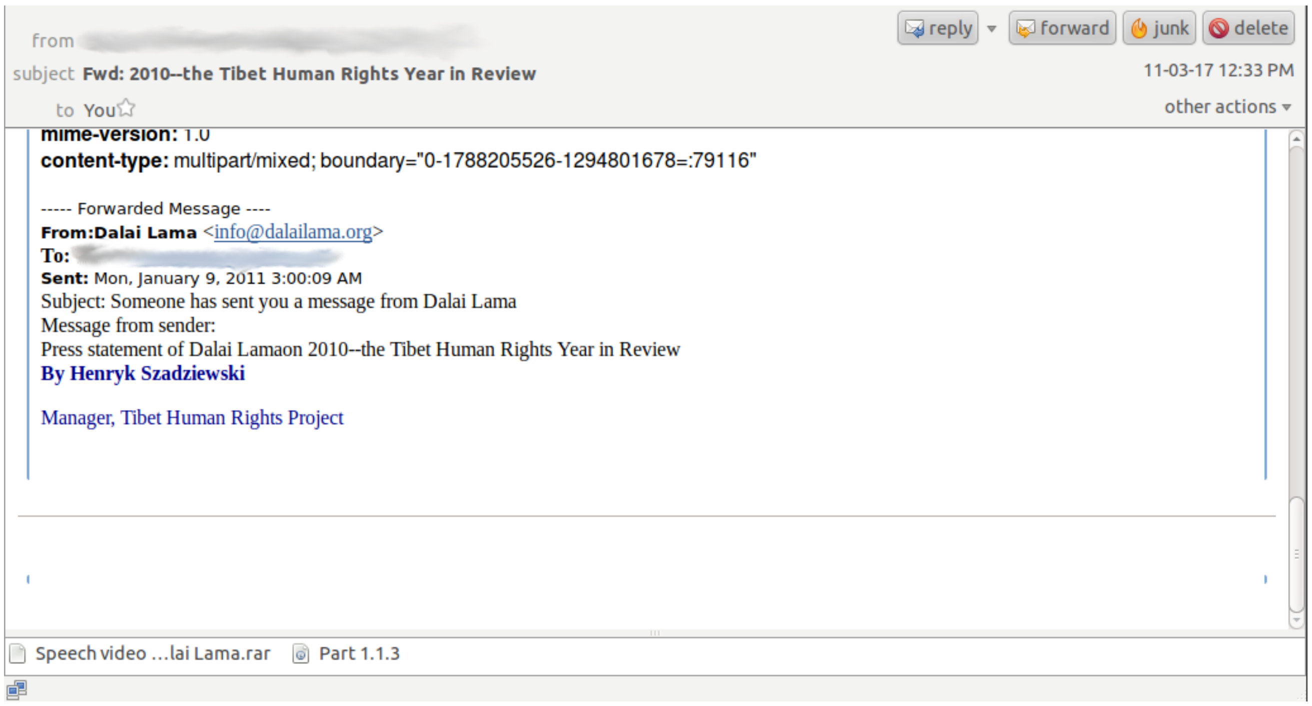


Organization 3: a Tibet-related organization received two emails with different versions of the malware attached. The first file was an executable file designed to appear as a video of a speech by the Dalai Lama, attached to an email about a year review of Tibetan human rights issues (figure 2). The second file was embedded in an Excel spreadsheet attached to an email pretending to be from a conference on climate change.

Emails that contain malicious attachments use a variety of social engineering techniques to appear more legitimate. Methods include using names of real people and organizations, choosing material that is directly related to the target's interests, and including chains of fake forwards to make it appear as if the email has been circulated.

Including the attachments in a RAR file makes them less likely to be discovered by an antivirus (AV) scanner. Putting a password on the archive and including it in the email reduces the chances of AV discovery even further.

FIGURE 2: **EXAMPLE TARGETED EMAIL WITH IEXPL0RE RAT USING SOCIAL ENGINEERING METHODS.**



from
subject **Fwd: 2010--the Tibet Human Rights Year in Review** 11-03-17 12:33 PM
to You

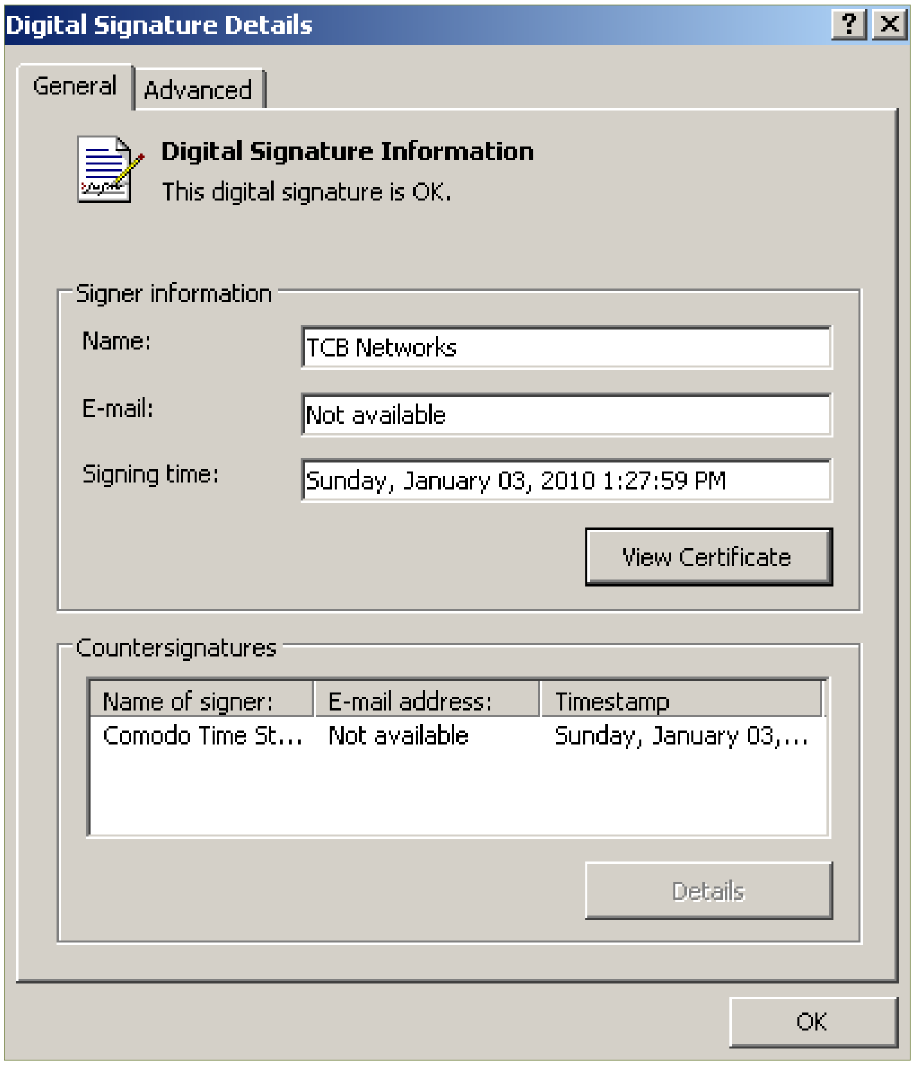
mime-version: 1.0
content-type: multipart/mixed; boundary="0-1788205526-1294801678=:79116"

----- Forwarded Message ----
From:Dalai Lama <info@dalailama.org>
To:
Sent: Mon, January 9, 2011 3:00:09 AM
Subject: Someone has sent you a message from Dalai Lama
Message from sender:
Press statement of Dalai Lamaon 2010--the Tibet Human Rights Year in Review
By Henryk Szadziewski

Manager, Tibet Human Rights Project

A newer version of the RAT payload was later distributed via email in multiple RTF documents to organization 3. The RTF dropped a DLL alongside a legitimate program vulnerable to DLL injection, allowing the program to run without a warning that the program is not digitally signed. StrokeIt, a program for using mouse gestures, uses a file named config.dll without verifying the authenticity of the file. By replacing config.dll with the RAT downloader, the malicious code is run while appearing more legitimate to the operating system (figure 3).

FIGURE 3: **VALID DIGITAL SIGNATURE FOR THE STROKEIT PROGRAM.**


Digital Signature Details

Digital Signature Information
This digital signature is OK.

Signer information

Name:	TCB Networks
E-mail:	Not available
Signing time:	Sunday, January 03, 2010 1:27:59 PM

Countersignatures

Name of signer:	E-mail address:	Timestamp
Comodo Time St...	Not available	Sunday, January 03,...

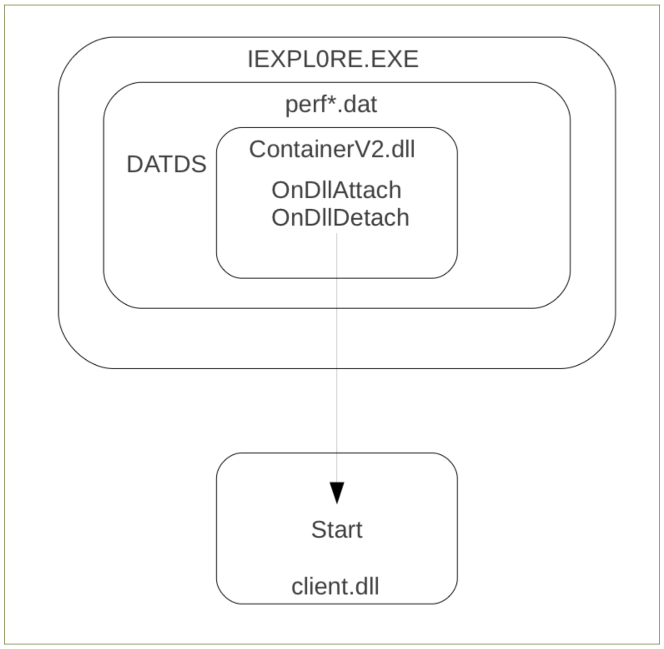
MALWARE

The IEXPL0RE RAT is delivered inside an executable program or document, which is custom-generated for each email. When a user opens the document or runs the program, it installs a launcher program on the computer. Antivirus programs frequently fail to detect the launcher program as malicious, as it is custom built for each specific target: the file contains a configuration file unique to the target, which is different each time it is sent out. This method is used to defeat signature-based antivirus programs, which only scan for files that are known to be malicious. As the launcher program is newly generated every time, it will never end up on a signature list until after it is already been used.

Once installed on a system, the launcher program goes through multiple programs to unpack a contained file (the actual RAT) before it can run. The IEXPL0RE.EXE (or other launcher) program contains multiple programs, layered like an onion, which eventually unpack a DLL (dynamic link library, another form of executable file). The file name varies, but starts with "perf" and has an extension of .dat, and is saved to the %temp% folder (often C:\Documents and Settings\user\Local Settings\Temp).

Once the perf*.dat file is saved to disk, it runs (via injection into svchost.exe, a Windows program) and extracts another DLL into memory. This program is called "ContainerV2," as it is referenced from within the program, although it is never written to the disk. ContainerV2 connects to the Internet and downloads another DLL called "client". The client is also kept in memory and never written to the disk. Once downloaded, ContainerV2 will run the client, which does all of the IEXPL0RE RAT work (figure 4).

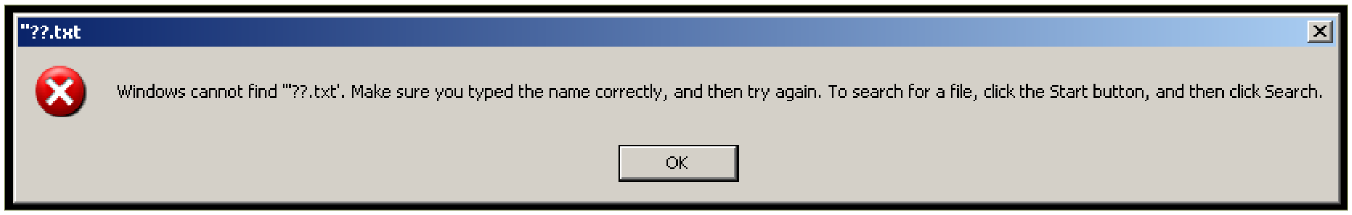
FIGURE 4: **STRUCTURE OF THE RAT LAUNCHER PROGRAM.**



For Organization 1, the executable launch process looks like this:

- 刘毅.exe(attached file, launcher): appears as a text document; when run, displays a fake error message saying the file can't be found (figure 5)
- csv.exe (runs and exits quickly)
- 360tray.exe (runs and exits quickly)
- svchost.exe (with injected perf*.dat code)
- ContainerV2 (injected into svchost.exe)
- client (downloaded and run in memory)

FIGURE 5:
SOCIAL ENGINEERING TECHNIQUE: A FAKE ERROR MESSAGE HIDING THE FACT THAT THIS IS A PROGRAM.


"??.txt

Windows cannot find '"??.txt'. Make sure you typed the name correctly, and then try again. To search for a file, click the Start button, and then click Search.

One advantage of downloading the final stage is that if the attacker wanted to update the RAT software (to add new functionality for example), it can be done very easily. Because code is downloaded every time the malware starts, if the code is changed on the server side, existing compromised machines will automatically update themselves the next time they are restarted. Over the time spent analyzing this malware, the client program did have minor changes, possibly bug fixes.

MD5 hashes, also called message digests, are often used to identify a file based on its content. A hash is a string of hexadecimal characters that identifies a file. Should the file change in any way, the hash will as well. Hashes are designed to be easy to compute from a full file, but it is very difficult to find two files with the same hash.

Use of hashes in the context of the IEXPL0RE RAT is difficult, as the downloaded client may change, and the ContainerV2 program is different for every target. One of the main differences that guarantees that the hash will always be unique is that the configuration file for the RAT (including which command and control servers to connect to) is included in the program.

For reference some MD5 hashes of IEXPL0RE components include:

ORIGINAL ATTACHMENT:

- Organization 1: d7c826ac94522416a0aecf5b7a5d2afe (EXE)
- Organization 2: 66e1aff355c29c6f39b21aedbbed2d5c (SCR)
- Organization 3: 21a1ee58e4b543d7f2fa3b4022506029 (EXE)
- Organization 3: 8d4e42982060d884e2b7bd257727fd7c (XLS)

CONTAINERV2:

- Organization 1: d46d85777062afbcda02de68c063b877
- Organization 2: 85e8c6ddcfa7e289be14324abbb7378d

ORGANIZATION 2 CLIENT (ONLY ACTIVE COMMAND AND CONTROL SERVER):

- November 1, 2011: eb51b384fcbbe468a6877f569021c5d1
- November 29, 2011: 8268297c1b38832c03f1c671e0a54a78 (current as of July 20, 2012)

INFECTION

Once the launcher program is run, it will install the IEXPL0RE binary and a startup link in the Start Menu:

```
C:\Documents and Settings\All Users\Start Menu\Programs\Startup\IEXPL0RE.LNK
C:\Documents and Settings\user\Application Data\Microsoft\Internet Explorer\IEXPL0RE.EXE
```

It also leaves traces of the extracted binary and the link file (the .tmp file below) in %temp%:

```
C:\Documents and Settings\user\Local Settings\Temp\31A.tmp
C:\Documents and Settings\user\Local Settings\Temp\perf6cd2e5e9.dat
```

The RAT also uses a few files for configuration and recording keystrokes:

```
C:\WINDOWS\system\lock.dat
C:\WINDOWS\system\MSMAPI32.SRG
C:\WINDOWS\system32\STREAM.SYS
```

When run, IEXPL0RE will connect to a command and control (C2) server for updates, sending key-logger data, and asking for RAT commands. The C2 server is specified in a configuration file built into the RAT program. Each RAT instance is likely built using a packaging program. The configuration file allows for a primary server and an alternate, and may use either a domain or IP.

Each of the IEXPL0RE samples analyzed uses a different set of C2 servers. One sample uses two domains that point to the same IP. The IP changes every few days to few weeks, but remains in one network block located in China. Other samples use either a single domain name and no backup, or a fixed IP with a localhost address as backup. The localhost address is a way to find and use a proxy, for example, if a computer is using a circumvention system such as Tor. Of the two samples using fixed IPs, both were sent to the same organization, and one appears to be a replacement for the other. Both C2 servers are currently down.

C2 COMMUNICATION

IEXPL0RE has two different methods of communication: HTTP POST and GET. It also has the ability to use a HTTP CONNECT proxy. POST is the preferred method of communication; if it does not work, it will also attempt a GET connection.

All communication from the client to the server is encrypted with a one-byte XOR key 0xCD. (Information in this report shows the data after decryption.) POST commands put the data in the request body, while GET commands put the data in URL parameters. Server responses are all 200 OK messages with data in the body.

The system keeps track of the communication using a sequence number, which is part of the requested URL. The sequence number is nine digits long, starts at 000000001, and generally increments by one for each packet sent. When authenticated, the sequence number jumps to 000001000; if disconnected, the sequence number returns to the next sub-1000 number expected.

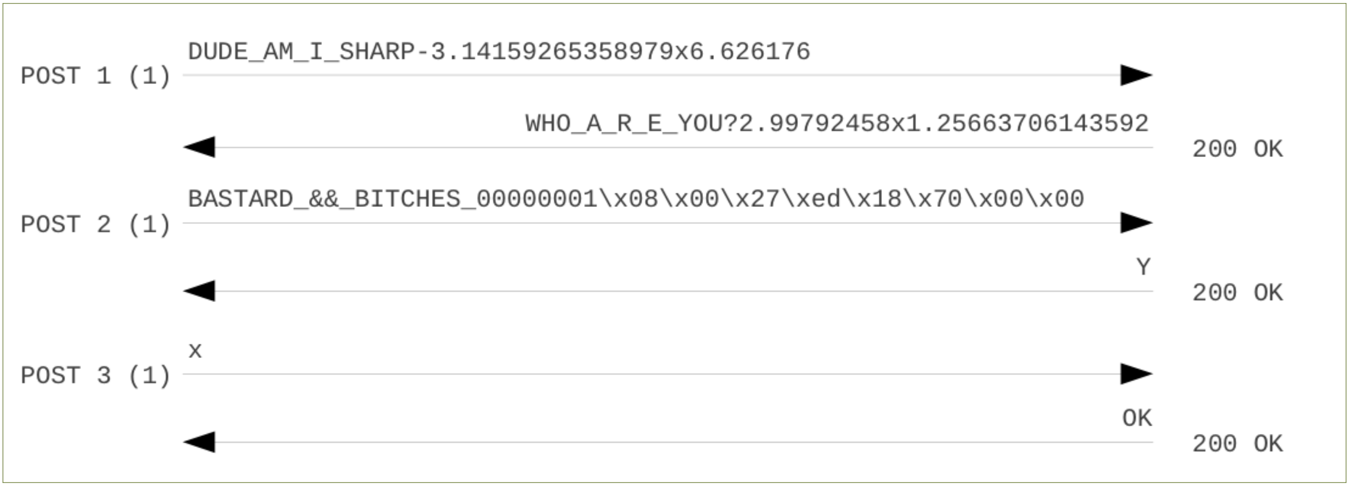
THE HEADERS OF THE REQUEST LOOK LIKE THIS:

```
POST /index000000001.asp HTTP/1.1
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1;)
Host: update.microsoft.com
Connection: Keep-Alive
Content-Type: text/html
Content-Length: 000041
```

The Accept-Language, User-Agent, Connection, and Content-Type headers are always fixed. The Host header is also always fixed as update.microsoft.com; any requests to the C2 server made without this header in place will be rejected, often with a redirect to Microsoft's website.

When run, ContainerV2 communicates with the C2 server, first establishing a socket by a three-way handshake. Below, the text at the start of the arrow indicates the packet type, sequence number, and connection socket. For example, "POST 2 (1)" means that it is using an HTTP POST request, sequence number 2, on the first established connection (figure. 6). The text on the line is the data in the packet after decryption.

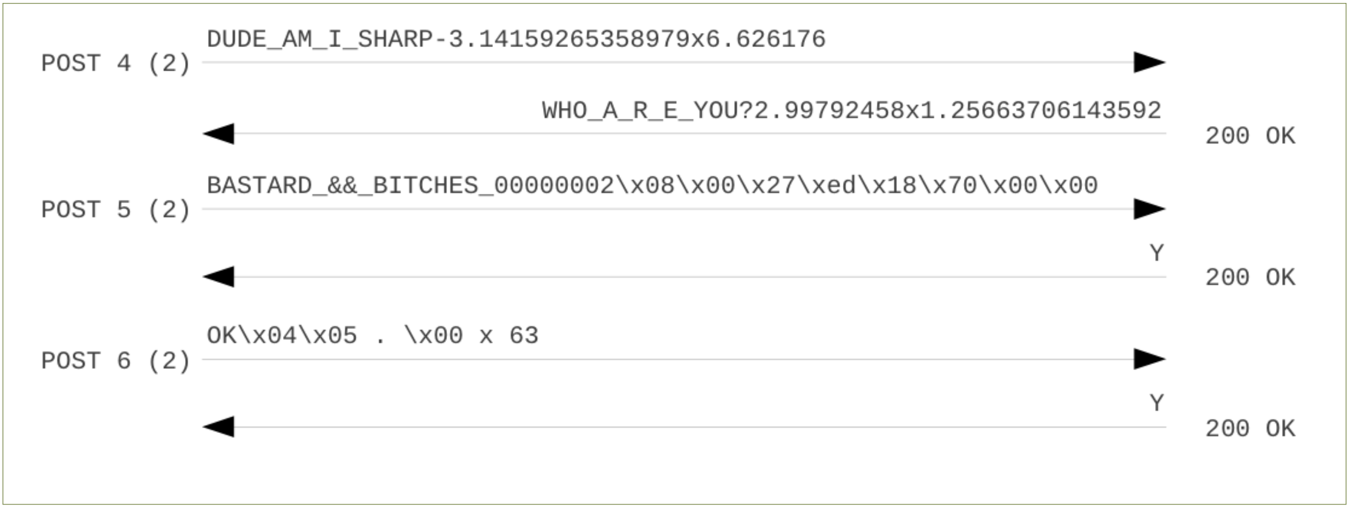
FIGURE 6: FIRST C2 CONNECTION



```
POST 1 (1)  DUDE_AM_I_SHARP-3.14159265358979x6.626176  ->
            <-  WHO_A_R_E_YOU?2.99792458x1.25663706143592   200 OK
POST 2 (1)  BASTARD_&&_BITCHES_00000001\x08\x00\x27\xed\x18\x70\x00\x00  ->
            <-  Y   200 OK
POST 3 (1)  x  ->
            <-  OK   200 OK
```

Once the first connection has been established, a second connection is made using a similar handshake (figure 7).

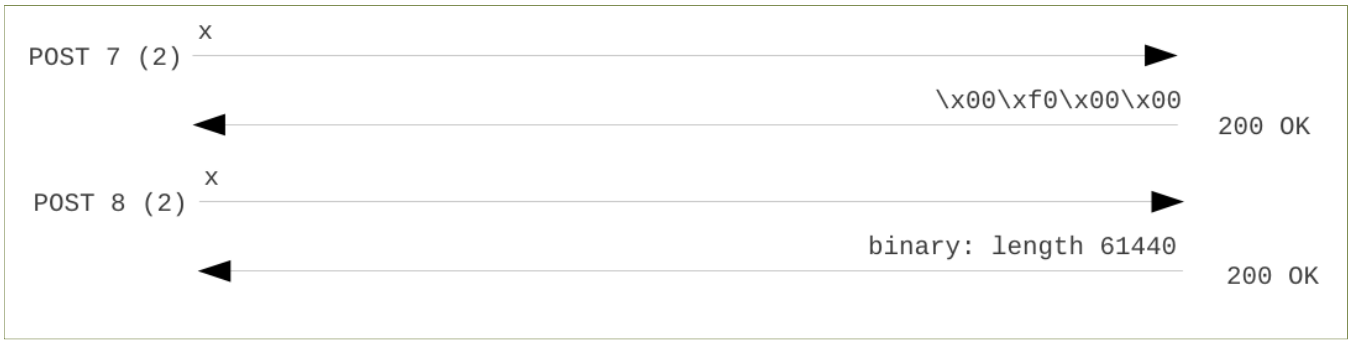
FIGURE 7: SECOND C2 CONNECTION



```
POST 4 (2)  DUDE_AM_I_SHARP-3.14159265358979x6.626176  ->
            <-  WHO_A_R_E_YOU?2.99792458x1.25663706143592   200 OK
POST 5 (2)  BASTARD_&&_BITCHES_00000002\x08\x00\x27\xed\x18\x70\x00\x00  ->
            <-  Y   200 OK
POST 6 (2)  OK\x04\x05 . \x00 x 63  ->
            <-  Y   200 OK
```

When the second connection has been established, the ContainerV2 program uses it to download the client and run it (figure 8).

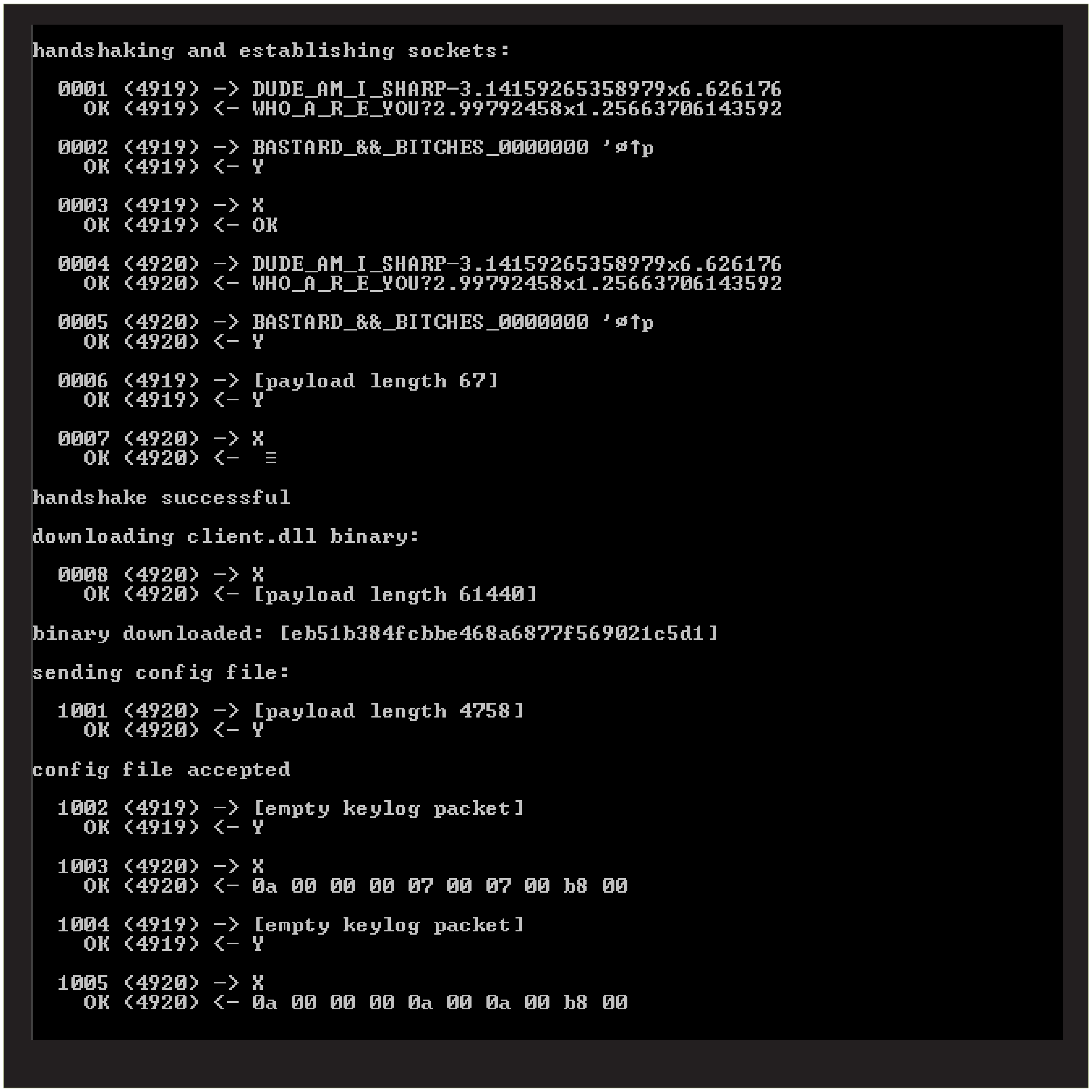
FIGURE 8: **BINARY DOWNLOAD**



Once control has been handed off to the client, one connection is used for sending keylogger data from the client to the server, and the other connection is used to request RAT commands from the server.

With the protocol reversed (see Appendix B for a full listing of commands), it was straightforward to write a program that communicates with the C2 server, downloads the client, and sends back commands as desired. The program maintains the two sockets, sending heartbeat/command request packets at a specified interval, while sending back empty keylogger packets to trick the server into thinking the system is idle (figure 9).

FIGURE 9: AN EXAMPLE OF THE FAKE MALWARE CLIENT COMMUNICATING WITH THE C2 SERVER.



```
handshaking and establishing sockets:

  0001 (4919) -> DUDE_AM_I_SHARP-3.14159265358979x6.626176
    OK (4919) <- WHO_A_R_E_YOU?2.99792458x1.25663706143592

  0002 (4919) -> BASTARD_&&_BITCHES_0000000 'ø↑p
    OK (4919) <- Y

  0003 (4919) -> X
    OK (4919) <- OK

  0004 (4920) -> DUDE_AM_I_SHARP-3.14159265358979x6.626176
    OK (4920) <- WHO_A_R_E_YOU?2.99792458x1.25663706143592

  0005 (4920) -> BASTARD_&&_BITCHES_0000000 'ø↑p
    OK (4920) <- Y

  0006 (4919) -> [payload length 67]
    OK (4919) <- Y

  0007 (4920) -> X
    OK (4920) <-  ≡

handshake successful

downloading client.dll binary:

  0008 (4920) -> X
    OK (4920) <- [payload length 61440]

binary downloaded: [eb51b384fcbbe468a6877f569021c5d1]

sending config file:

  1001 (4920) -> [payload length 4758]
    OK (4920) <- Y

config file accepted

  1002 (4919) -> [empty keylog packet]
    OK (4919) <- Y

  1003 (4920) -> X
    OK (4920) <- 0a 00 00 00 07 00 07 00 b8 00

  1004 (4919) -> [empty keylog packet]
    OK (4919) <- Y

  1005 (4920) -> X
    OK (4920) <- 0a 00 00 00 0a 00 0a 00 b8 00
```

CAPABILITIES

The IEXPL0RE RAT contains over 40 commands that an attacker can use to manipulate the file system and registry, download and run additional programs, and find and exfiltrate data. An infected computer defaults to recording keystrokes and sending this data back to the server at regular intervals. The additional commands are there for interactive control of the system in real-time by an attacker.

This program is likely used in multiple phases. After infection, the keylogger records data including email addresses and passwords. Once an account's credentials have been captured, the attacker can log in and set up a forwarding address or download all of the data stored online. Once a compromised machine has been determined useful by looking at the keylogger data, an attacker can use the RAT functionality to download files and install more specific malware - for example, a Skype plugin that records calls.

While post-infection behaviour from an attacker against a real target has not been observed in this investigation, this is a standard method in targeted attacks.

One particular area of interest with this RAT is that it contains a specific functionality for plugins relating to video and audio capture. Each time the malware connects to the command and control server, it sends a list of all video capture devices present on the computer. This behaviour may indicate that the attacker is specifically interested in seeing who is on the other end of the computer, and is actively collecting data on what the targets look like.

For a full list of the commands supported by IEXPL0RE and a description of what they do, see Appendix B: Command Enumeration.

DETECTION AND MITIGATION

A system infected with the IEXPL0RE RAT can be found by looking for presence of the IEXPL0RE files, or by watching network traffic.

In addition to the IEXPL0RE.EXE file itself, presence of the perf*.dat files and link files in %temp% are an indicator that the system is infected. The timestamps on the files are an indication of how long the system has been infected.

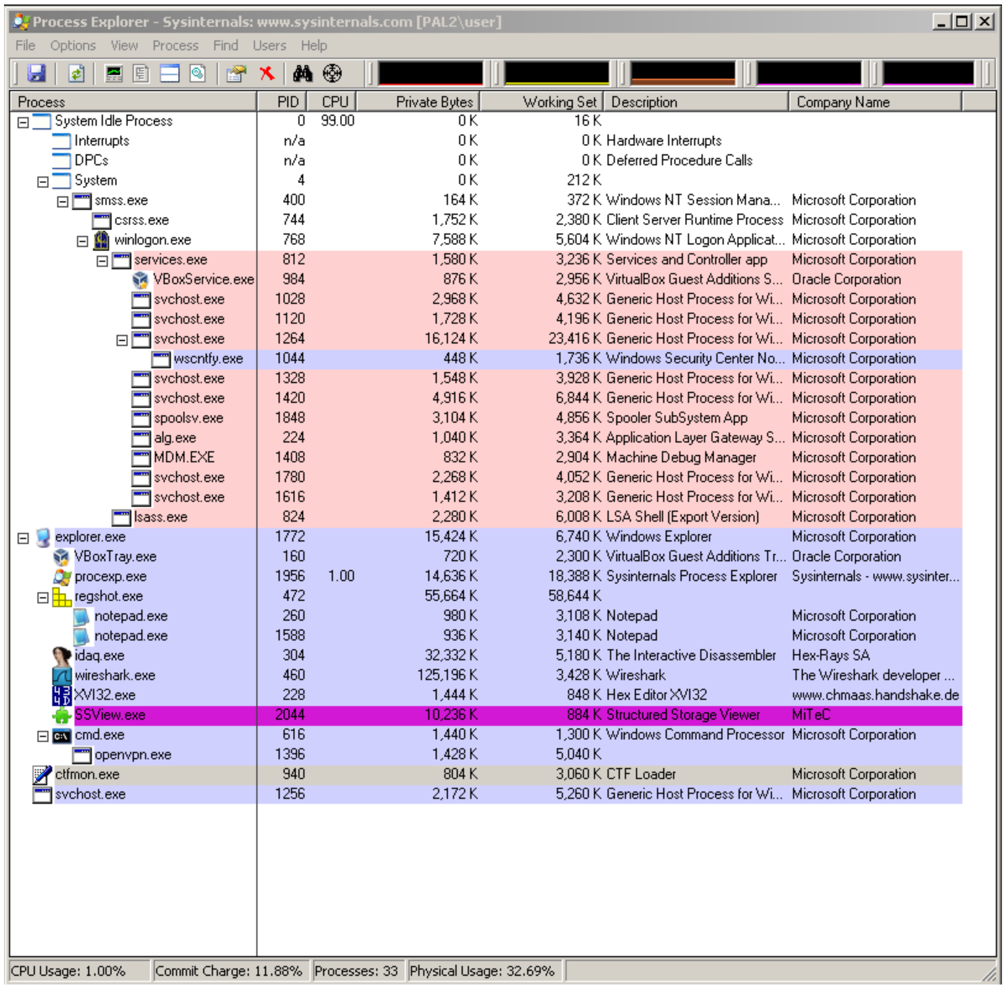
A network intrusion detection system (IDS) can identify infected machines by looking for well-known traffic patterns. The simplest of these is checking for HTTP traffic to /index[0-9]{9}.asp. Blocking this traffic will prevent the infected machine from communicating with the C2 server, receiving new commands, and sending back keylogger data.

The C2 IP or hostname can also be blocked directly once it's found, at the network level or (as a temporary measure) in the infected computer's hosts file.

REMOVAL

A running copy of IEXPL0RE can be stopped by killing the appropriate svchost process. This process is identifiable as it is not in the correct place in the process tree. In figure 10, this is the last process in the list, PID 1256:

FIGURE 10: **PROCESS EXPLORER SHOWING THE INFECTED SVCHOST.EXE PROCESS (1256).**



Process Explorer - Sysinternals: www.sysinternals.com [PAL2\user]

File Options View Process Find Users Help

Process	PID	CPU	Private Bytes	Working Set	Description	Company Name
System Idle Process	0	99.00	0 K	16 K		
Interrupts	n/a		0 K	0 K	Hardware Interrupts	
DPCs	n/a		0 K	0 K	Deferred Procedure Calls	
System	4		0 K	212 K		
smss.exe	400		164 K	372 K	Windows NT Session Mana...	Microsoft Corporation
csrss.exe	744		1,752 K	2,380 K	Client Server Runtime Process	Microsoft Corporation
winlogon.exe	768		7,588 K	5,604 K	Windows NT Logon Applicat...	Microsoft Corporation
services.exe	812		1,580 K	3,236 K	Services and Controller app	Microsoft Corporation
VBoxService.exe	984		876 K	2,956 K	VirtualBox Guest Additions S...	Oracle Corporation
svchost.exe	1028		2,968 K	4,632 K	Generic Host Process for Wi...	Microsoft Corporation
svchost.exe	1120		1,728 K	4,196 K	Generic Host Process for Wi...	Microsoft Corporation
svchost.exe	1264		16,124 K	23,416 K	Generic Host Process for Wi...	Microsoft Corporation
wscntfy.exe	1044		448 K	1,736 K	Windows Security Center No...	Microsoft Corporation
svchost.exe	1328		1,548 K	3,928 K	Generic Host Process for Wi...	Microsoft Corporation
svchost.exe	1420		4,916 K	6,844 K	Generic Host Process for Wi...	Microsoft Corporation
spoolsv.exe	1848		3,104 K	4,856 K	Spooler SubSystem App	Microsoft Corporation
alg.exe	224		1,040 K	3,364 K	Application Layer Gateway S...	Microsoft Corporation
MDM.EXE	1408		832 K	2,904 K	Machine Debug Manager	Microsoft Corporation
svchost.exe	1780		2,268 K	4,052 K	Generic Host Process for Wi...	Microsoft Corporation
svchost.exe	1616		1,412 K	3,208 K	Generic Host Process for Wi...	Microsoft Corporation
lsass.exe	824		2,280 K	6,008 K	LSA Shell (Export Version)	Microsoft Corporation
explorer.exe	1772		15,424 K	6,740 K	Windows Explorer	Microsoft Corporation
VBoxTray.exe	160		720 K	2,300 K	VirtualBox Guest Additions Tr...	Oracle Corporation
procexp.exe	1956	1.00	14,636 K	18,388 K	Sysinternals Process Explorer	Sysinternals - www.sysinter...
regshot.exe	472		55,664 K	58,644 K		
notepad.exe	260		980 K	3,108 K	Notepad	Microsoft Corporation
notepad.exe	1588		936 K	3,140 K	Notepad	Microsoft Corporation
idaq.exe	304		32,332 K	5,180 K	The Interactive Disassembler	Hex-Rays SA
wireshark.exe	460		125,196 K	3,428 K	Wireshark	The Wireshark developer ...
XVI32.exe	228		1,444 K	848 K	Hex Editor XVI32	www.chmaas.handshake.de
SSView.exe	2044		10,236 K	884 K	Structured Storage Viewer	MiTeC
cmd.exe	616		1,440 K	1,300 K	Windows Command Processor	Microsoft Corporation
openvpn.exe	1396		1,428 K	5,040 K		
ctfmon.exe	940		804 K	3,060 K	CTF Loader	Microsoft Corporation
svchost.exe	1256		2,172 K	5,260 K	Generic Host Process for Wi...	Microsoft Corporation

CPU Usage: 1.00% Commit Charge: 11.88% Processes: 33 Physical Usage: 32.69%

The process can be killed with the Process Explorer tool, part of the Sysinternals package (figure 10).

Once the process has been terminated, removal is as simple as deleting the installed files (see the section on Infection above for a list).

CONCLUSIONS

The IEXPL0RE RAT is a good example of the current state of APT attacks, especially those targeting human rights organizations and NGOs. While they are not particularly advanced from a technical standpoint, they are custom designed to appeal to and pique the interest of the recipient.

The attacker uses social engineering to get a foot in the door of an organization. All it takes is for one user to run a malicious program that looks like a legitimate video, spreadsheet, or other document. Once running on a user's machine, the program will silently record passwords and provide the attacker a way of accessing sensitive data.

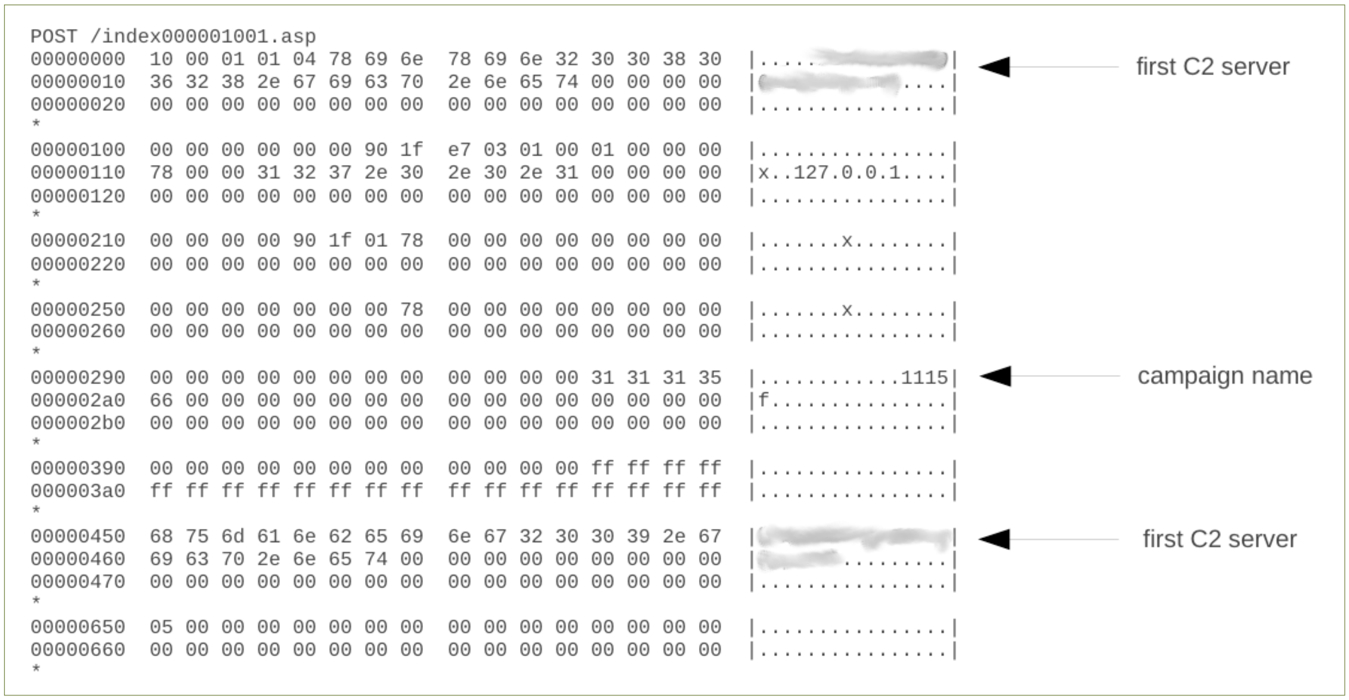
This report describes what is "normal" in this area, by detailing what a common attack looks like at each step of the way, from when an email is first received to when data leaves the network. Many APT campaigns like the one presented in this report exist and continue to steal data every day, but are both avoidable and correctable.

The IEXPL0RE RAT is under active development, as both the client and server components are continuously changing. The server in particular has exhibited different behavior over time, mostly related to blocking unauthorized access from the outside world. For example, the redirect to Microsoft's website when referencing an invalid URL was not present when this investigation began. The presence of development work or upgrades implies that this system is actively used and monitored.

APPENDIX A: CONFIGURATION FILE

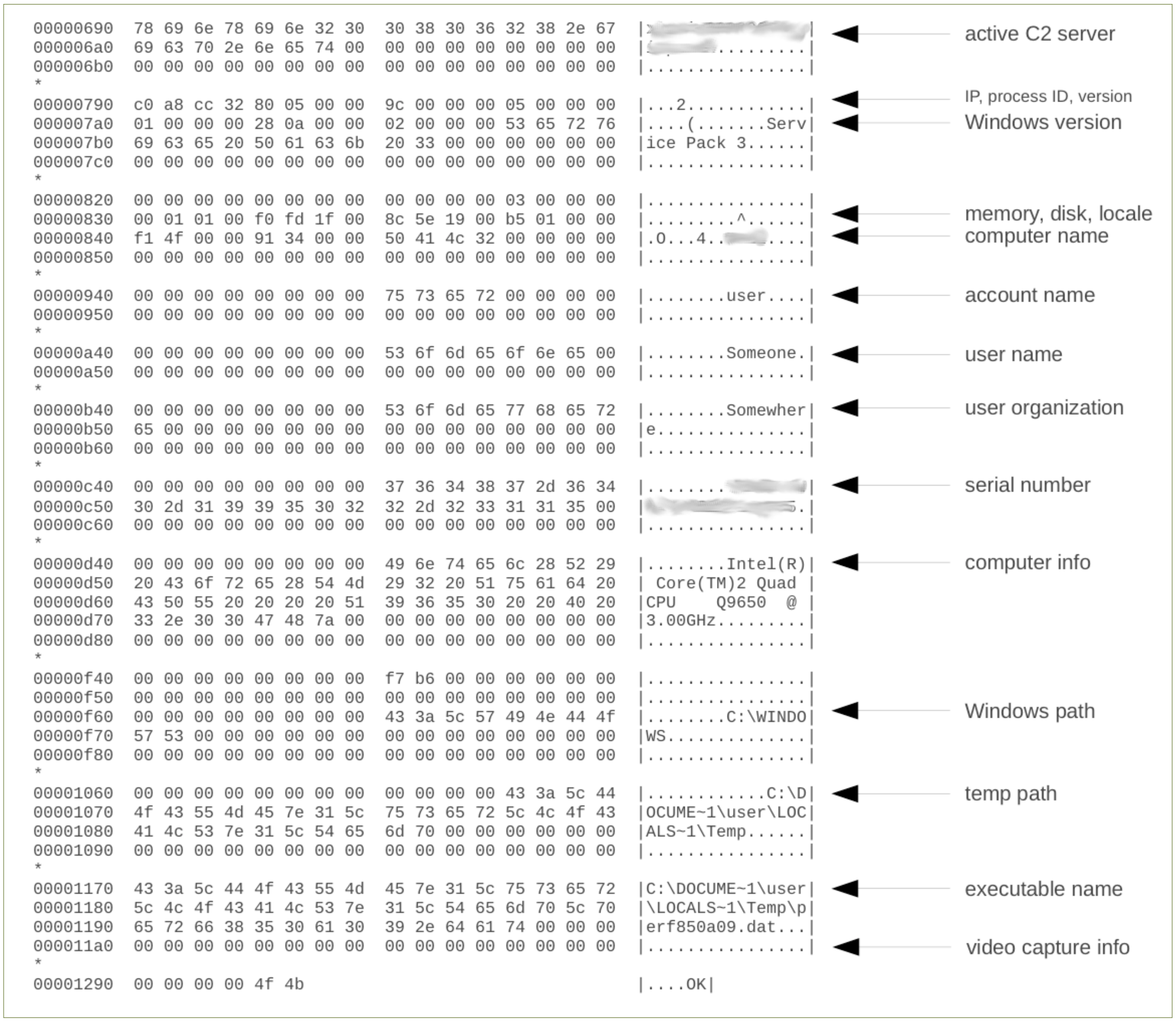
The configuration sent to the C2 server on initial connection has the client configuration at the beginning (figure 11), followed by more information about the infected computer (figure 12).

FIGURE 11: CLIENT CONFIGURATION SENT TO C2



```
POST /index000001001.asp
00000000  10 00 01 01 04 78 69 6e  78 69 6e 32 30 30 38 30  |.....[redacted]|
00000010  36 32 38 2e 67 69 63 70  2e 6e 65 74 00 00 00 00  |[redacted]....|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000100  00 00 00 00 00 00 90 1f  e7 03 01 00 01 00 00 00  |................|
00000110  78 00 00 31 32 37 2e 30  2e 30 2e 31 00 00 00 00  |x..127.0.0.1....|
00000120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000210  00 00 00 00 90 1f 01 78  00 00 00 00 00 00 00 00  |.......x........|
00000220  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000250  00 00 00 00 00 00 00 78  00 00 00 00 00 00 00 00  |.......x........|
00000260  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000290  00 00 00 00 00 00 00 00  00 00 00 00 31 31 31 35  |............1115|
000002a0  66 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |f...............|
000002b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000390  00 00 00 00 00 00 00 00  00 00 00 00 ff ff ff ff  |................|
000003a0  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00000450  68 75 6d 61 6e 62 65 69  6e 67 32 30 30 39 2e 67  |[redacted]|
00000460  69 63 70 2e 6e 65 74 00  00 00 00 00 00 00 00 00  |[redacted].........|
00000470  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000650  05 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000660  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

FIGURE 12: **DETAILED INFORMATION ON INFECTED COMPUTER SENT TO C2**



```
00000690  78 69 6e 78 69 6e 32 30  30 38 30 36 32 38 2e 67  |[redacted]|
000006a0  69 63 70 2e 6e 65 74 00  00 00 00 00 00 00 00 00  |[redacted].........|
000006b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000790  c0 a8 cc 32 80 05 00 00  9c 00 00 00 05 00 00 00  |...2............|
000007a0  01 00 00 00 28 0a 00 00  02 00 00 00 53 65 72 76  |....(.......Serv|
000007b0  69 63 65 20 50 61 63 6b  20 33 00 00 00 00 00 00  |ice Pack 3......|
000007c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000820  00 00 00 00 00 00 00 00  00 00 00 00 03 00 00 00  |................|
00000830  00 01 01 00 f0 fd 1f 00  8c 5e 19 00 b5 01 00 00  |.........^......|
00000840  f1 4f 00 00 91 34 00 00  50 41 4c 32 00 00 00 00  |.O...4..[redacted]....|
00000850  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000940  00 00 00 00 00 00 00 00  75 73 65 72 00 00 00 00  |........user....|
00000950  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000a40  00 00 00 00 00 00 00 00  53 6f 6d 65 6f 6e 65 00  |........Someone.|
00000a50  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000b40  00 00 00 00 00 00 00 00  53 6f 6d 65 77 68 65 72  |........Somewher|
00000b50  65 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |e...............|
00000b60  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000c40  00 00 00 00 00 00 00 00  37 36 34 38 37 2d 36 34  |........[redacted]|
00000c50  30 2d 31 39 39 35 30 32  32 2d 32 33 31 31 35 00  |[redacted]5.|
00000c60  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000d40  00 00 00 00 00 00 00 00  49 6e 74 65 6c 28 52 29  |........Intel(R)|
00000d50  20 43 6f 72 65 28 54 4d  29 32 20 51 75 61 64 20  | Core(TM)2 Quad |
00000d60  43 50 55 20 20 20 20 51  39 36 35 30 20 20 40 20  |CPU    Q9650  @ |
00000d70  33 2e 30 30 47 48 7a 00  00 00 00 00 00 00 00 00  |3.00GHz.........|
00000d80  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000f40  00 00 00 00 00 00 00 00  f7 b6 00 00 00 00 00 00  |................|
00000f50  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000f60  00 00 00 00 00 00 00 00  43 3a 5c 57 49 4e 44 4f  |........C:\WINDO|
00000f70  57 53 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |WS..............|
00000f80  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001060  00 00 00 00 00 00 00 00  00 00 00 00 43 3a 5c 44  |............C:\D|
00001070  4f 43 55 4d 45 7e 31 5c  75 73 65 72 5c 4c 4f 43  |OCUME~1\user\LOC|
00001080  41 4c 53 7e 31 5c 54 65  6d 70 00 00 00 00 00 00  |ALS~1\Temp......|
00001090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001170  43 3a 5c 44 4f 43 55 4d  45 7e 31 5c 75 73 65 72  |C:\DOCUME~1\user|
00001180  5c 4c 4f 43 41 4c 53 7e  31 5c 54 65 6d 70 5c 70  |\LOCALS~1\Temp\p|
00001190  65 72 66 38 35 30 61 30  39 2e 64 61 74 00 00 00  |erf850a09.dat...|
000011a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001290  00 00 00 00 4f 4b                                 |....OK|
```

APPENDIX B: COMMAND ENUMERATION

The following is a list of all commands present in the IEXPL0RE malware, and a detailed description of what data is received or sent over the network for each command.

CODE	COMMAND	SERVER / CLIENT	DESCRIPTION
0x00	Failure	C	Client response for a variety of commands to indicate that the operation did not succeed.
0x01	Success	C	Client response for a variety of commands to indicate that the operation succeeded. Contains variable data related to the command request.
0x01	Reply file does not exist	C	Reply file for plugin does not exist. Packet contains: [4] - Command code (0x01)
0x02	Reply file over 512kB	C	Reply file for plugin is over 512kB. Packet contains: [4] - Command code (0x02)
0x03	Reply file	C	Reply file for plugin. Packet contains: [var] - buffer -- implemented so always 0?
0x03	Shutdown	S	Sends a shutdown + power off + force command to the system. Requires parameters 0/0.
0x04	Reboot	S	Sends a reboot + force command to the system. Requires parameters 0/0.
0x06	Reconnect	S	Disconnects open connections and reconnects.
0x07	Shut off display	S	Sends WM_SYSCOMMAND message SC_MONITORPOWER to shut off the display.
0x0B	Download and install malware	S	Downloads a file, writes it to disk, and possibly executes it. Packet contains: [4] - executable size [var] - executable Depending on configuration and AV software present, writes the file to IEXPL0RE.EXE (in application data folder, Microsoft subfolder), fxsst.dll (in Windows system directory), or SENS64.DLL (in temp path). May run IEXPL0RE.EXE depending on options; may also install configuration file (STREAM.SYS or Cache). Returns failure or success with parameters 0/0.
0x0C	Install dropped files	S	Checks configuration file options and moves the appropriate dropped files to the correct locations (may vary depending on Windows version). Returns failure or success with parameters 0/0.

0x0D	Update configuration file	S	Downloads new configuration parameters and writes the updated information to the configuration file. Packet option 2: [2] · Value1 == 2 [180] · Unused? Packet option 1: update campaign name? [2] · Value1 == 1 [4] · campaign name length [var] · campaign name Packet option 4: update configuration file [2] · Value1 == 4 [2] · port [2] · unknown (offset 264) [2] · unknown (offset 266) [4] · unknown (offset 664) [1] · unknown (offset 274) [1] · unknown (offset 534) [2] · unknown (offset 532) [2] · unknown (offset 270) [2] · unknown (offset 272) [4] · campaign name length [var] · campaign name [4] · C2 name length [var] · C2 name [4] · unknown length [var] · unknown (unused?) [4] · unknown length [var] · unknown (offset 275) [4] · unknown length [var] · unknown (offset 535) [4] · unknown length [var] · unknown (offset 599) Returns failure or success with parameters 0/0.

0x0E	Download and run plugin	S	Opens a new connection in a new thread, downloads a file, then runs it (possibly with Internet Explorer credentials). This looks like a plugin activation for screen captures and audio recording -- references offscreen.dll and offsound.dll. Packet contains: [4] - unknown (field_4) [4] - unknown length [var] - unknown (field_8) [4] - DLL name length [var] - DLL name [4] - DLL arguments length [var] - DLL arguments [4] - Reply filename length [var] - Reply filename [4] - unknown length [var] - unknown (field_620) [4] - unknown length [var] - unknown (field_724) [1] - unknown (field_828) [1] - unknown (field_829) [1] - unknown (field_82A) [1] - Add process ID, socket, verb to DLL arguments? [1] - Create process as IE user? [4] - unknown length [var] - unknown (field_82C) [4] - unknown (field_934) Handshake for the new connection uses connection number -1. If the connection is successful, replies with a failure packet, parameters 0/0, containing: [4] - unknown (field_4) [4] - unknown length [var] - unknown (field_8) [4] - DLL name length [var] - DLL name [4] - Reply filename length [var] - Reply filename [4] - unknown length [var] - unknown (field_620) [4] - unknown length [var] - unknown (field_724) [1] - unknown (field_828) [1] - unknown (field_829) [4] - unknown length [var] - unknown (field_82C) [4] - unknown (field_934) If successful, it will send the C2 an X command. The C2 will reply with a file, which the client writes to disk. The client will send a success or failure packet with parameters 0/0 depending on whether the file was received. If field_828 is non-zero, the client will send the contents of the reply file-name specified in a 0x03 command with parameters 0/0. The way this is implemented, it appears as if it will always send an empty packet. If process creation is unsuccessful, it will send a failure packet with param-eters 0/0 containing the following: [4] - command (0x00)

0x0F	Download and execute file	S	Downloads a file and runs it. Packet contains: [4] · executable size [var] · executable Downloads the file to %temp% and executes it. Returns failure or success with parameters 0/0.
0x10	Unknown	S	Updates a values in the lock.dat file and sets an event. Packet contains: [4] · Value length [var] · Value Sets the DWORD at lock.dat offset 516 to 2, and copies the value from the packet to offset 520. Sets the USERMODECMD event.
0x11	Unknown	S	Reads a value out of %temp%/screenlog.txt. Returns a success or failure command with parameters 0/0 depending on whether the value read equals 1. The command contains: [4] · Value Where value equals: 0 : file does not exist 3 : value read from file equals 1 4 : value read from file equals 0 5 : value read from file equals 2
0x12	Unknown	S	Reads a value out of %temp%/offsoundlog.txt. Returns a success or failure command with parameters 0/0 depending on whether the value read equals 1. The command contains: [4] · Value Where value equals: 0 : file does not exist 3 : value read from file equals 1 4 : value read from file equals 0 5 : value read from file equals 2
0x1E	sub_10004603()	C	If not empty, packet contains: [2] · _WIN32_FIND_DATAA structure length [var] · Data for file name If empty, packet contains: [2] · Set to 0 [1] · 1 if structure size > 40000, 0 if failure Parameters are set to Res1/Res2.

0x20	Move file	S	Moves a file or directory. Packet contains: [4] · Source length [var] · Source [4] · Destination length [var] · Destination Returns failure or success with parameters 1/Res1
0x21	Delete file	S	Deletes a file or directory. Packet contains: [4] · File name length [var] · File name Returns a success or failure command with parameters Res2/Res1.
0x22	Create directory	S	Creates a directory. Packet contains: [4] · Path name length [var] · Path name Returns a success or failure command with parameters Res2/Res1.
0x23	GetSystemInfo request	S	Requests client to send a 0x24 response with the output of GetSystemInfo().
0x24	GetSystemInfo re-sponse	C	Contains a _SYSTEM_INFO struct with the output of GetSystemInfo().
0x26	Get document paths	S	Gets paths for CSIDL special folders PERSONAL (My Documents), DESKTOP-DIRECTORY (Desktop), and HISTORY (Internet history). Returns a success command containing: [4] · My Documents path length [var] · My Documents path [4] · Desktop path length [var] · Desktop path [4] · Internet history path length [var] · Internet history path Parameters are set to 1/Res1.
0x29	Move file or directory	S	Moves a file or directory. Packet contains: [4] · Source length [var] · Source [4] · Destination length [var] · Destination Returns failure or success with parameters 1/Res1.

0x2A	Set file access time and attributes	S	Sets the creation time, last access time, last write time, and file attributes of a file. Packet contains: [8] · CreationTime [8] · LastAccessTime [8] · LastWriteTime [4] · dwFileAttributes [4] · File name length [var] · File name Returns failure or success with parameters Res2/Res1.
0x2B	Unknown	S	Does some file-walking, including across all drives available (A to Z). Replies with a 0x2C command followed by a number of 0x2D commands. Packet contains: [4] · Directory length [var] · Directory [4] · Unknown length [var] · Unknown
0x2C	Unknown start response	C	Response to the 0x2B command. Uses parameters Res2/Res1. Packet contains: [4] · Number of 0x2D packets to follow
0x2D	Unknown response	C	Response to the 0x2B command. Uses parameters Res2/Res1. Packet contains: [4] · Unknown data length [var] · Unknown data, result of sub_10001B73()
0x2F	Owner name, organization, and serial number request	S	Sends the owner name, organization, and serial number. Returns a success response with the following data: [4] · Username length [var] · Username [4] · User organization length [var] · User organization [4] · Serial length [var] · Serial
0x46	Read from file	S	Packet contains: [8] · File offset [2] · Characters to read [4] · Length of field 3 [var] · Field 3 Replies with a 0x47 packet containing data from a file.
0x47	Read from file response	C	Response to 0x46 that contains data from a file. Sent with parameters 2/2. Packet contains: [2] · Size [var] · Data

0x4B	List files	S	Lists files in a given directory along with file size and last write times. Packet contains [4] · Directory length [var] · Directory
0x4C	Start of list files response	C	Start of list response to 0x4B. Sent with parameters 2/2. Packet has no payload.
0x4D	End of list files response	C	End of list response to 0x4B. Sent with parameters 2/2. Packet has no payload.
0x4E	List files response	C	List item for response to 0x4B. Sent with parameters 2/2. Packet contains: [8] · FindFileData.nFileSizeLow, FindFileData.nFileSizeHigh [8] · .ftLastWriteTime.dwLowDateTime, .dwHighDateTime [4] · length of next field [var] · whole string: filename plus size and write time
0x4F	Open file	S	Opens a specified file for use with 0x46 [and friends]. Packet contains: [4] · File name length [var] · File name [4] · File mode length [var] · File mode Returns failure or success with parameters 2/2.
0x50	Close file	S	Closes file opened with 0x4F command. No response sent.
0x5A	Start of running program list	C	Response to 0x5D command that signals the start of a list of running programs. Packet is empty with parameters 3/Res1.
0x5B	End of running program list	C	Response to 0x5D command that signals the end of a list of running programs. Packet is empty with parameters 3/Res1.
0x5C	Running program	C	Response to 0x5D command that contains the first executable module for a single process. One packet is sent per process. Packet contains: [24] · PROCESSENTRY32.th32ProcessID [4] · length of executable module name [var] · length of executable module
0x5D	List running programs	S	Sends a list of executable names for running processes. Replies with a 0x5A response, followed with a 0x5C packet for each executable, and ends with a 0x5B response. Client uses the CreateToolhelp32Snapshot() API function followed by Process32First()/Process32Next() to list all processes. The executable name is the module name returned by Module32First().
0x5E	Kill process	S	Kills a running process. Packet contains: [4] · Process ID Returns failure or success with parameters 3/Res1.

0x5F	Run program	S	Runs a program already present on the client. Packet contains: [4] · Command line length [var] · Command line Returns success or failure with parameters 3/Res1.
0x72	Unknown · open connection B?	S	Creates multiple new threads and a new C2 connection (via full handshake) with connection number 11. Packet contains: [4] · Unknown length [var] · Unknown value (if 0 < length < 80000) Returns success or failure with parameters 11/11. If successful, contains the following payload: [4] · Unknown length [var] · Unknown (v2 + 808) [4] · Unknown value
0x73	Unknown · remove connection B?	S	May relate to uninstalling. Packet contains: [4] · Process ID Kills the process with given process ID, and closes a socket. Returns success or failure with parameters 11/11. If failure, contains the following payload: [4] · Process ID [4] · Unknown length [var] · Unknown (v2 + 808)
0x82	Number of services	C	Response to 0x85 command with the number of services on the system. Packet contains: [4] · Number of services returned by EnumServicesStatusA() Response parameters are 5/5.

0x84	Service information	C	Response to 0x85 command with details on a service. Packet contains: [4] - Service handle [4] - Current state [4] - Start type [4] - Error control [4] - Length of service name [var] - Service name [4] - Length of service display name [var] - Service display name [4] - Length of service binary path [var] - Service binary path [4] - Length of service description [var] - Service description [4] - Length of service start name [var] - Service start name Response parameters are 5/5.
0x85	List services	S	Lists all services on the system. Sends a 0x82 response with the number of services, then 0x84 responses with service details.
0x86	Start service	S	Starts a service on the system. Packet contains: [4] - Length of service name [var] - Service name Returns success with parameters 5/5 if service is started.
0x87	Control service	S	Sends a control message to a service on the system. Packet contains: [4] - Length of service name [var] - Service name [4] - Service control parameter Service control parameters are: 1 - stop 2 - pause 3 - continue Returns success with parameters 5/5 with the payload: [4] - Service current state

0x88	Create service	S	Creates a new service on the system. Packet contains: [4] - Service name length [var] - Service name [4] - Display name length [var] - Display name (set in CreateServiceA()) [4] - Binary path name length [var] - Binary path name [4] - Display name length [var] - Display name (set by ChangeServiceConfig2A()) [4] - Start type Returns success or failure with parameters 5/5. If success, contains the following payload: [4] - Service handle [4] - Current state [4] - Start type [4] - Error control [4] - Length of service name [var] - Service name [4] - Length of service display name [var] - Service display name [4] - Length of service binary path [var] - Service binary path [4] - Length of service description [var] - Service description [4] - Length of service start name [var] - Service start name
0x89	Delete service	S	Deletes a service from the system. Packet contains: [4] - Service name length [var] - Service name Returns success or failure with parameters 5/5.
0x8A	Set service options	S	Changes the display name and start type of a service. Packet contains: [4] - Service name length [var] - Service name [4] - Display name length [var] - Display name (max 256 chars) [4] - Display name length [var] - Display name (max 512 chars) [4] - Service start type Returns success or failure with parameters 5/Res2
0x96	Enumerate registry keys	S	Opens a registry key and enumerates its subkeys. Replies with an 0x97 packet with subkey information. Packet contains: [4] - Registry key name length [var] - Registry key name

0x97	Enumerate registry keys response	C	Contains a list of all the subkey names for a given registry key. Packet contains: [4] - Number of subkeys (N) [var, N times] : [4] - Subkey name length [var] - Subkey name Parameters are set to 6/6.
0x98	Registry key last write time query	S	Requests the last write time on a specified registry key and returns the information in a 0x99 packet. Packet contains: [4] - Registry key name length [var] - Registry key name
0x99	Registry key last write time response	C	Contains the last write time of a registry key. Packet contains: [8] - Last write time (_FILETIME structure) Parameters are set to 6/6.
0x9A	Enumerate registry key values	S	Opens a registry key and enumerates its values. Replies with a 0x9B packet with the number of values and maximum size values. Sends a 0x9C packet for each value, then signifies the end of the list with a 0x9D packet. Packet contains: [4] - Registry key name length [var] - Registry key name
0x9B	Start of registry key value enumeration list	C	Response to the 0x9A command signifying the start of a registry key value enumeration. Packet contains: [4] - Number of values associated with the registry key [4] - Max value name length [4] - Max value length Parameters are set to 6/6.
0x9C	Registry key value enumeration item	C	Response to the 0x9A command signifying a registry key value. Packet contains: [4] - Type [4] - Value name size [var] - Value name [4] - Value data size [var] - Value data Parameters are set to 6/6.

0x9D	End of registry key value enumeration list	C	Response to the 0x9A command signifying the end of a registry key value enumeration. Parameters are set to 6/6.
0x9F	Delete registry key value	S	Deletes a value from a registry key. Packet contains: [4] - Registry key name length [var] - Registry key name [4] - Registry key value length [var] - Registry key value Returns success or failure with parameters set to 6/6.
0xA0	Change registry key value	S	Changes a value of a registry key. Packet contains: [4] - Registry key name length [var] - Registry key name [4] - Registry key old value length [var] - Registry key old value [4] - Registry key new value length [var] = Registry key new value Returns success or failure with parameters set to 6/6.
0xA1	Create empty registry key value	S	Creates a registry key value of a specified type with no value. Packet contains: [4] - Registry key name length [var] - Registry key name [4] - Registry key value name length [var] - Registry key value name [4] - Registry key value type Returns success or failure with parameters set to 6/6.
0xA2	Create registry key	S	Creates a registry key. Can be a subkey. Packet contains: [4] - Registry key name length [var] - Registry key name Returns success or failure with parameters set to 6/6.

0xA3	Set registry key type and value	S	Sets a registry key type and value. Packet always contains: [4] · Registry key name length [var] · Registry key name [4] · Registry value name length [var] · Registry value name [4] · Registry value type The value then can take a different form based on the value type. Value type 0 (empty): No payload. Value type 1 (REG_SZ, null terminated string): [4] · Registry value length [var] · Registry value Value type 3 (REG_BINARY, raw binary data): [4] · Registry value length [var] · Registry value Value type 4 (REG_DWORD, double word): [4] · Registry value Returns success or failure with parameters 6/6.
0xA4	NOP	S	Does nothing. Possibly an unimplemented or deleted function.
0xA5	Delete registry key	S	Deletes a registry key. Can be a subkey. Packet contains: [4] · Registry key name length [var] · Registry key name Returns success or failure with parameters 6/6.
0xB6	Keylogger response	C	Sends keylogger data from the keylogger buffer file. Keylogger data: [4] · Length [1] · All bytes read? 1 or 0 [var] · Keylogger data Parameters are set to the output of function 4 in the class C vtable?
0xB8	Keylogger data request	S	Requests keylogger data in a 0xB6 packet.

0xC8	Unknown - get a screen-shot?	S	Looks suspiciously like taking a screenshot. Packet contains: [2] - Unknown [2] - Unknown [1] - Unknown Replies with failure or success with parameters 8/8. If success, contains the following fields: [2] - Monitor width size (X) in pixels [2] - Monitor height size (Y) in pixels [4] - Unknown [4] - Unknown size (screenshot?) [var] - Unknown (screenshot?)
0xCA	Send keyboard or mouse event	S	Packet contains: [4] - Unknown [4] - Extra information for keybd_event or mouse_event [4] - Flags for keybd_event or mouse_event [4] - Vk for keybd_event [4] - x coordinate for mouse or Vk for keyboard [4] - y coordinate for mouse [4] - Data for mouse event
0xCB	Downloads a file	S	Downloads a file to %temp%\off.dll. Packet contains: [2] - Unknown [2] - Unknown [2] - Unknown [4] - Data length [4] - Unknown length [var] - Unknown [4] - Unknown length [var] - Unknown [4] - File data length [var] - File data Writes log information on the size of the downloaded file to c:\aaa\ccc.txt.