Extracting ZeroAccess from NTFS Extended Attributes

By Posted by Corey Harrell

Archived: 2026-04-05 16:38:08 UTC

This past week I was reading a paper about the ZeroAccess Trojan when a section about a clever data hiding technique caught my eye. The paper was Sophos's The ZeroAccess Botnet: Mining and Fraud for Massive Financial Gain and I'm referring to the Services.exe section which stated (on page 6):

"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."

I was already fascinated about the Trojan's use of two separate DLLs to maintain persistence but what I read opened my eyes. ZeroAccess leveraged a third DLL for persistence that was hidden inside the NTFS Extended Attributes. A Google search lead me to a third article by Symantec - Trojan.Zeroaccess.C Hidden in NTFS EA. I then remembered a few months back someone from the Yahoo Win4n6 group posted the write-up ZeroAccess – From Rootkit to Nasty Infection – which also discussed ZeroAccess hiding data inside NTFS Extended Attribute. At the time, I read the HitmanPro article and bookmarked it amongst my malware references but I didn't give it too much thought. Now my eyes are open and the use of the NTFS Extended Attribute had my full attention. A few thoughts were running through my head (yes in this order)

1. This is a really cool technique

2. The articles don't mention how to extract the data from the Extended Attribute. I wonder how you can do it.

3. Hiding data in the Extended Attribute is brilliant since antivirus software might not check there. Come to think about it most digital forensic tools don't parse the data inside Extended Attributes either.

4. Wait a second; I never look at the NTFS Extended Attributes during my examinations. I need to come up with a way to do it and add it to my examination process. It could be a check underneath my Malware Searches examination step.

5. Alight, this is definitely cool.

In this post I'm providing answers to all the questions that were racing through my mind. The post is broken up into the sections:

- What Are NTFS Extended Attributes

- My Testing Environment

- Extracting Data from NTFS Extended Attributes

- Extracting ZeroAccess from NTFS Extended Attributes

- Locating Other ZeroAccess Files on System

What Are NTFS Extended Attributes

The NTFS filesystem is the default for the Microsoft Windows server operating systems as well as Windows 2000, XP, Vista, and 7. The Master File Table ($MFT) is the component in NTFS containing information about all the files and folders on the volume. "Every file and directory has at least one entry in the table, and the entries by themselves are very simple." The entry basically consists of a header and attributes. Two attributes are the $EA index and $EA_INFORMATION which are the NTFS Extended Attributes.

The File System Forensic Analysis book shows the $EA_INFORMATION type identifier as 208 and $EA identifier as 224 (credit goes to Harlan for this reference). The book further stated both attributes are "used for backward compatibility with OS/2 applications (HPFS)." In addition, the book Windows Internals, Part 2: Covering Windows Server 2008 R2 and Windows 7 stated "extended attributes are name/value pairs and aren't normally used but are provided for backward compatibility with OS/2 applications". The references provide a little more clarity about what the author of ZeroAccess is doing. They are using hardly used $MFT attributes meant for backward compatibility as a place to hide data. The cool factor with this technique went up a few notches.

Testing Environment

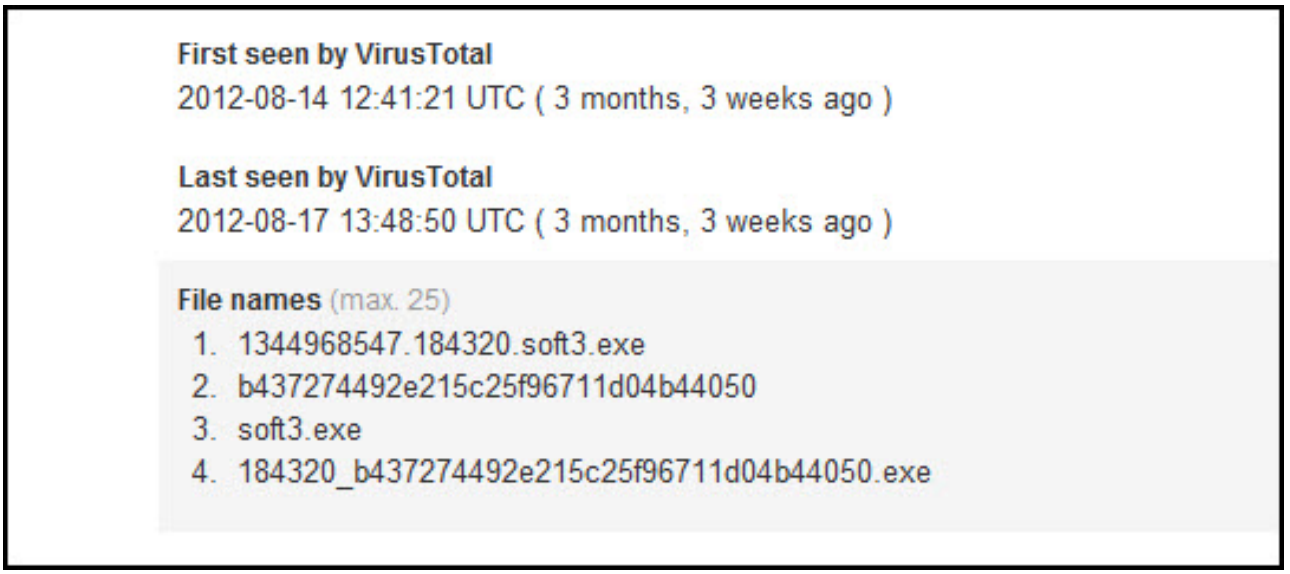
Before diving into the answers to all my questions I wanted to provide information about my testing environment. Knowing the testing environment makes it easier for others to see exactly what I did and how to replicate my testing. For those looking for the "good stuff" skip ahead to the section Extracting Data from NTFS Extended Attributes.

Finding the ZeroAccess Sample

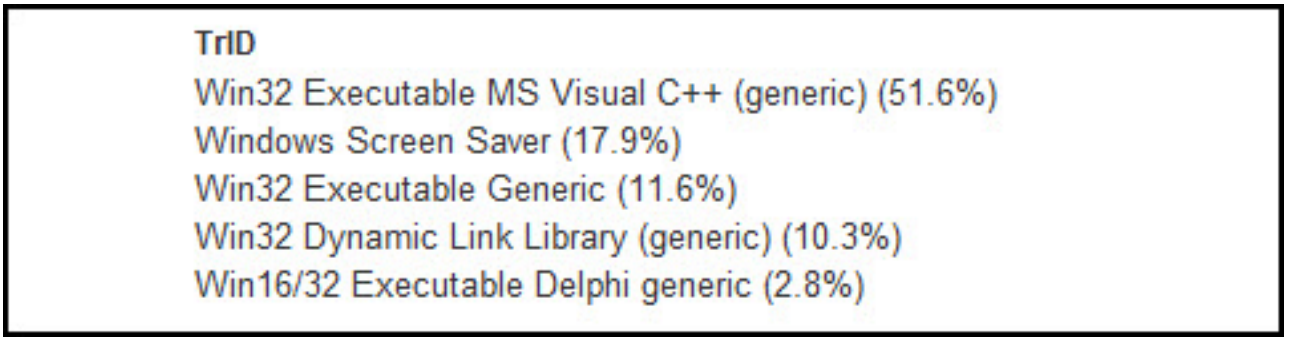
In all the articles about ZeroAccess using the NTFS Extended Attributes I noticed one commonality; this technique was used in an updated version of the Trojan. Check out the commonality: HitmanPro posted their write up on June 25, 2012, Symantec did theirs on August 14, 2012, and Sophos report was released in September 2012. Furthermore, Sophos stated in their opening sentence "since our last paper on ZeroAccess, The ZeroAccess Rootkit, its authors have made significant changes." The point was crystal clear, I needed to find a recent ZeroAccess sample in order to find one leveraging the $EA hiding technique.

I searched for samples in the Virus Share malware repository as you will see why shortly. A generic search for "zeroaccess" wasn't to helpful since it resulted in 6,056 results. I needed a more focused search so I turned to Symantec's detection Trojan.ZeroAccess.C. It reduced my search results to 9 samples.

A nice feature about Virus-Share is it provides links to each sample's VirusTotal's report. I used the VT report to further narrow down my samples. On VirusTotal's Additional Information section, I first looked to see when each sample was first seen.



Remember the commonality with the articles? I only wanted to focus on samples first seen within the past few months. The second check I did on the Additional Information section was to look at the TrID output to determine what samples were executables as opposed to DLLs. Working with an exe makes it a little easier for fast execution.



I narrowed my samples down to five based on the above checks. Not every ZeroAccess Trojan will modify the services.exe and store binary data in the NTFS Extended Attribute. So I started executing one sample after another to see if any modified the services.exe MFT record. I didn't do any dynamic analysis; I pretty much just ran a sample and then looked at the services.exe MFT record to see if the Extended Attributes were added. On the third sample I hit pay dirt. The sample MD5 hash b437274492e215c25f96711d04b44050 (SHA256 658817f5f7722506868d9f717ee19b276fcab0d0ecac071d5d92a4178fdeb5b3) leveraged the NTFS Extended Attributes.

Side note: I mentioned in the Win4n6 group I was looking for ZeroAccess samples that used the NTFS Extended Attributes. Stefan replied with not only the MD5 hash c6e73a75284507a41da8bef0db342400 but links to the article ZeroAccess - new steps in evolution and the ThreatExpert Report. This sample is the next one I'm going to check out.

Configuring my Test System

The Sophos report indicated the services.exe is only patched if ZeroAccess is installed on a system with Windows Vista or higher. I selected for my test system Windows 7 Ultimate 32 bit which I was running inside VMware. I used a fresh install and the only configuration change I made was to disable the User Account Control. I wanted to avoid making ZeroAccess elevate its privileges. I also installed the program Capture.bat as a quick way to see changes made to the file system.

Extracting Data from NTFS Extended Attributes

I had to answer my first question before I could go forward with any testing. How do I extract the binary data from the NTFS Extended Attributes? I reached out to the Win4n6 group for suggestions and Brian Carrier pointed me to the promise land. He suggested I used the Sleuthkit; specifically istat to see what attributes a MFT record has and icat to dump the data in an attribute. The process needed to extract data stored inside NTFS Extended Attributes with TSK is as follows:

Note: the TSK tools can be ran against a forensic image or a mounted drive. My examples throughout the post are against a mounted drive (\\.\PHYSICALDRIVE#) but the path to a forensic image could be used instead.

1. Identify the starting sector of the partition where services.exe is located. Typically, the first sector for Vista and 7 is 2048 (XP is 63). The TSK tool to use is mmls and the command is:

mmls \\.\PHYSICALDRIVE#

2. Identify the inode number for the C:\Windows\System32\services.exe file. The TSK tool to use is ifind and the command is:

ifind -o 2048 -n "Windows/System32/services.exe" \\.\PHYSICALDRIVE#

Command explanation: the –o switch is to specify the sector offset that was discovered with mmls and the –n switch is the file name to get the inode for.

3. Review the attributes associated with the MFT record for C:\Windows\System32\services.exe. The TSK tool to use is istat and the command is:

istat -o 2048 \\.\PHYSICALDRIVE3 12345

Command explanation: the –o switch is the sector offset and the 12345 is the place where the inode number goes. The attributes of interest are $EA_INFORMATION (208) and $EA (224).

4. If present, make note of the NTFS Extended Attributes. In istat's output, next to the attribute name will be a value inside parentheses. i.e. the value in my test appeared as $EA (224-4)

5. Dump the contents from the $EA and $EA_INFORMATION and redirect it to a file. The TSK tool to use is icat and the commands are:

icat -o 2048 \\.\PHYSICALDRIVE# 12345-208-# > EA_INFO.txt
icat -o 2048 \\.\PHYSICALDRIVE# 12345-224-# > EA.txt

Command explanation: the –o switch is the sector offset and the number at the end is the inode number with the attribute value added. The 208-# dumps $EA_INFORMATION and 224-# dumps $EA.

The above process can be used to either read a MFT's record attributes or dump the data stored in those attributes.

Extracting ZeroAccess from NTFS Extended Attributes

To completely answer my first question about how to extract the ZeroAccess Trojan from NTFS Extended Attributes I actually had to do it. This is what I ended up doing:

- Determined what a clean MFT services record looks like
- Located an infected MFT services record
- Extracted ZeroAccess from the MFT services record
- Created the ZeroAccess binary

What Does a Clean MFT Services.exe Record Look like?

It's always good to know what normal looks like so it's easier to see what is abnormal. This was my thinking and why I first looked at a clean MFT services.exe record. Here's what I did and what it looked like. I highlighted in blue values of interest.

1. Confirmed the starting sector for the partition.

mmls \\.\PHYSICALDRIVE2

DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

Slot Start End Length Description
00: Meta 0000000000 0000000000 0000000001 Primary Table (#0)
01: ----- 0000000000 0000002047 0000002048 Unallocated
02: 00:00 0000002048 0020969471 0020967424 NTFS (0x07)
03: ----- 0020969472 0020971519 0000002048 Unallocated

2. Identified the inode number for the C:\Windows\System32\services.exe file.

ifind -o 2048 -n "Windows/System32/services.exe" \\.\PHYSICALDRIVE2

19211

3. Reviewed the attributes associated with the C:\Windows\System32\services.exe MFT record. Notice there are no $EA or $EA_INFORMATION attributes listed under the Attributes section.

istat -o 2048 \\.\PHYSICALDRIVE2 19211

MFT Record Header Values:
Record: 19211 Sequence: 1
$LogFile Sequence Number: 34894400
Allocated File
Links: 2

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 450 ()
Created: 2009-07-13 19:11:26 (Eastern Daylight Time)
File Modified: 2009-07-13 21:14:36 (Eastern Daylight Time)
MFT Modified: 2012-12-06 11:54:31 (Eastern Standard Time)
Accessed: 2009-07-13 19:11:26 (Eastern Daylight Time)

$FILE_NAME Attribute Values:
Flags: Archive
Name: services.exe, services.exe
Parent MFT Record: 7150 Sequence: 1
Allocated Size: 0 Actual Size: 0
Created: 2012-12-06 11:54:31 (Eastern Standard Time)
File Modified: 2012-12-06 11:54:31 (Eastern Standard Time)
MFT Modified: 2012-12-06 11:54:31 (Eastern Standard Time)
Accessed: 2012-12-06 11:54:31 (Eastern Standard Time)

Attributes:
Type: $STANDARD_INFORMATION (16-0) Name: N/A Resident size: 72
Type: $FILE_NAME (48-4) Name: N/A Resident size: 90
Type: $FILE_NAME (48-2) Name: N/A Resident size: 90
Type: $DATA (128-3) Name: N/A Non-Resident size: 259072 init_size: 259072
750372 750373 750374 750375 750376 750377 750378 750379
750380 750381 750382 750383 750384 750385 750386 750387
750388 750389 750390 750391 750392 750393 750394 750395
750396 750397 750398 750399 750400 750401 750402 750403
750404 750405 750406 750407 750408 750409 750410 750411
750412 750413 750414 750415 750416 750417 750418 750419
750420 750421 750422 750423 750424 750425 750426 750427
750428 750429 750430 750431 750432 750433 750434 750435

Located an Infected MFT Services.exe Record

The clean C:\Windows\System32\services.exe file only had four attributes listed. The $STANDARD_INFORMATION (16-0), $FILE_NAME (48-4), $FILE_NAME (48-2), and $DATA (128-3)

attributes. Here's what an infected services.exe MFT record looks like and what I did. I highlighted in blue values of interest and items related to the infection in red.

1. Confirmed the starting sector for the partition.

mmls \\.\PHYSICALDRIVE2

DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

Slot Start End Length Description
00: Meta 0000000000 0000000000 0000000001 Primary Table (#0)
01: ----- 0000000000 0000002047 0000002048 Unallocated
02: 00:00 0000002048 0020969471 0020967424 NTFS (0x07)
03: ----- 0020969472 0020971519 0000002048 Unallocated

2. Identified the inode number for the C:\Windows\System32\services.exe file.

ifind -o 2048 -n "Windows/System32/services.exe" \\.\PHYSICALDRIVE2

42756

3. Reviewed the attributes associated with the C:\Windows\System32\services.exe MFT record.

istat -o 2048 \\.\PHYSICALDRIVE2 42756

MFT Record Header Values:
Record: 42756 Sequence: 1
$LogFile Sequence Number: 98696063
Allocated File
Links: 1

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 483 ()
Last User Journal Update Sequence Number: 17321736
Created: 2009-07-13 19:11:26 (Eastern Daylight Time)
File Modified: 2009-07-13 21:14:36 (Eastern Daylight Time)
MFT Modified: 2012-12-06 17:18:05 (Eastern Standard Time)
Accessed: 2009-07-13 19:11:26 (Eastern Daylight Time)

$FILE_NAME Attribute Values:
Flags: Archive
Name: services.exe
Parent MFT Record: 1802 Sequence: 1

Allocated Size: 0 Actual Size: 0
Created: 2009-07-13 19:11:26 (Eastern Daylight Time)
File Modified: 2012-12-06 17:18:05 (Eastern Standard Time)
MFT Modified: 2012-12-06 17:18:05 (Eastern Standard Time)
Accessed: 2012-12-06 17:18:05 (Eastern Standard Time)

Attributes:
Type: $STANDARD_INFORMATION (16-0) Name: N/A Resident size: 72
Type: $FILE_NAME (48-2) Name: N/A Resident size: 90
Type: $DATA (128-5) Name: N/A Non-Resident size: 259072 init_size: 259072
618 619 620 621 622 623 624 625
626 627 628 629 630 631 632 633
634 635 636 637 638 639 640 641
642 643 644 645 646 647 648 649
650 651 652 653 654 655 656 657
658 659 660 661 662 663 664 665
666 667 668 669 670 671 672 673
674 675 676 677 678 679 680 681
Type: $EA_INFORMATION (208-3) Name: N/A Resident size: 8
Type: $EA (224-4) Name: N/A Non-Resident size: 23404 init_size: 23404
346 347 348 349 350 351

Extract ZeroAccess from MFT Services Record

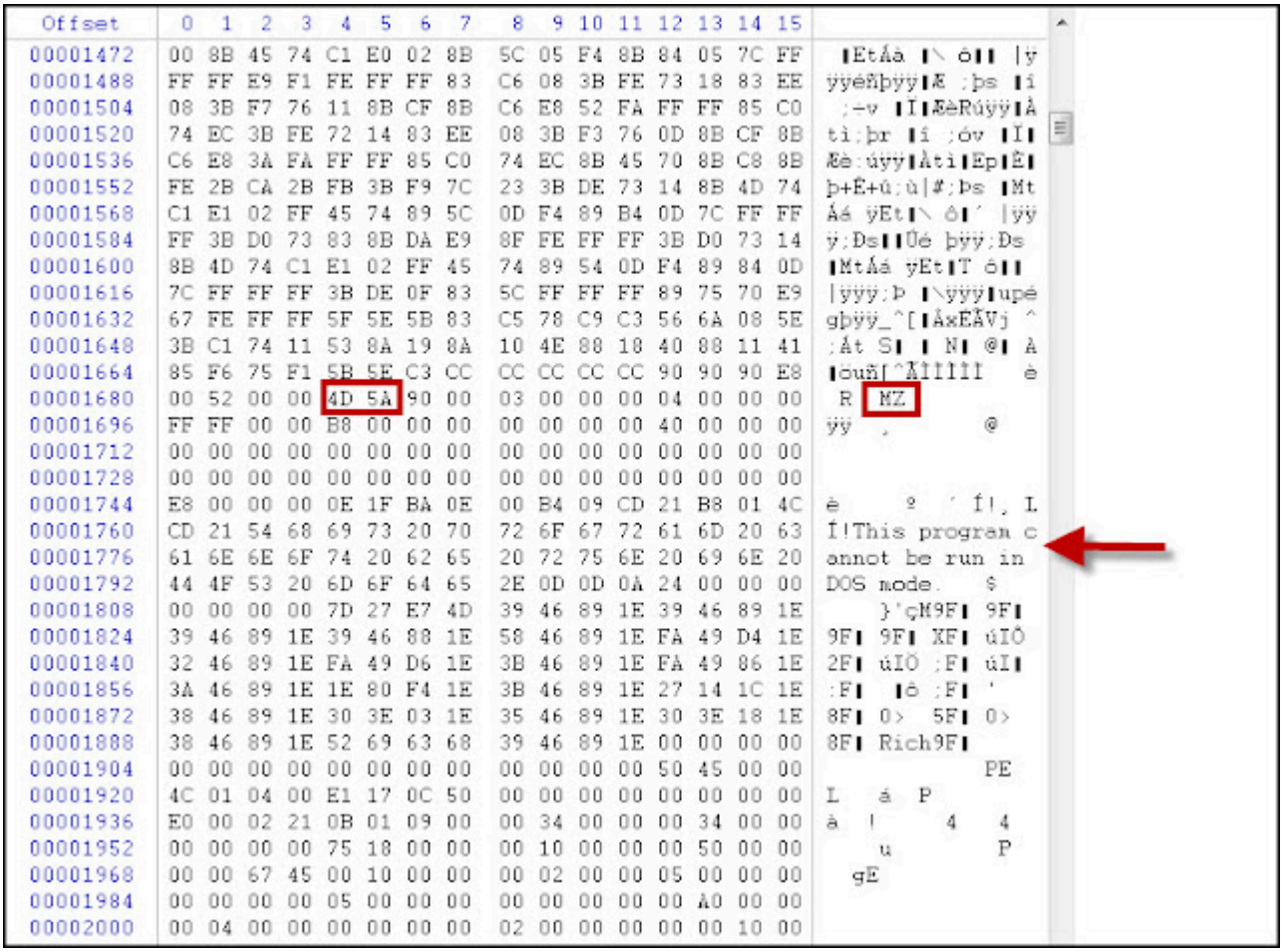
The infected MFT services.exe record had a few different changes. The inode number for C:\Windows\System32\services.exe changed from 19211 to 42756. In addition, the attributes now listed the NTFS Extended Attributes: $EA_INFORMATION (208-3) and $EA (224-4). The ZeroAccess binary data can now be extracted using the attribute values shown with istat. Here's how I extracted the data.

4. Extracted the binary data from the MFT record for C:\Windows\System32\services.exe for both the $EA_INFORMATION and $EA attributes.

icat -o 2048 \\.\PHYSICALDRIVE2 42756-208-3> icat_services_$EA-INFORMATION.bin
icat -o 2048 \\.\PHYSICALDRIVE2 42756-224-4 > icat_services_$EA_binary.bin

Create the ZeroAccess Binary

The Sophos report indicated the ZeroAccess binary data stored in the NTFS Extended Attribute is shellcode and a DLL. This means I had to separate the two in the dumped data; my focus was only on the DLL. There was very little data stored in the $EA_INFORMATION attribute while the majority of the data – including the DLL – was in the $EA attribute. To rebuild the DLL I opened the extracted data in a Hex editor and looked for for a PE file's characteristics. I was looking for the MZ file signature and the MS-DOS message.



Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00001472	00	8B	45	74	C1	E0	02	8B	5C	05	F4	8B	84	05	7C	FF
00001488	FF	FF	E9	F1	FE	FF	FF	83	C6	08	3B	FE	73	18	83	EE
00001504	08	3B	F7	76	11	8B	CF	8B	C6	E8	52	FA	FF	FF	85	C0
00001520	74	EC	3B	FE	72	14	83	EE	08	3B	F3	76	0D	8B	CF	8B
00001536	C6	E8	3A	FA	FF	FF	85	C0	74	EC	8B	45	70	8B	C8	8B
00001552	FE	2B	CA	2B	FB	3B	F9	7C	23	3B	DE	73	14	8B	4D	74
00001568	C1	E1	02	FF	45	74	89	5C	0D	F4	89	B4	0D	7C	FF	FF
00001584	FF	3B	D0	73	83	8B	DA	E9	8F	FE	FF	FF	3B	D0	73	14
00001600	8B	4D	74	C1	E1	02	FF	45	74	89	54	0D	F4	89	84	0D
00001616	7C	FF	FF	FF	3B	DE	0F	83	5C	FF	FF	FF	89	75	70	E9
00001632	67	FE	FF	FF	5F	5E	5B	83	C5	78	C9	C3	56	6A	08	5E
00001648	3B	C1	74	11	53	8A	19	8A	10	4E	88	18	40	88	11	41
00001664	85	F6	75	F1	5B	5E	C3	CC	CC	CC	CC	CC	90	90	90	E8
00001680	00	52	00	00	4D	5A	90	00	03	00	00	00	04	00	00	00
00001696	FF	FF	00	00	B8	00	00	00	00	00	00	00	40	00	00	00
00001712	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001728	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001744	E8	00	00	00	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C
00001760	CD	21	54	68	69	73	20	70	72	6F	67	72	61	6D	20	63
00001776	61	6E	6E	6F	74	20	62	65	20	72	75	6E	20	69	6E	20
00001792	44	4F	53	20	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00
00001808	00	00	00	00	7D	27	E7	4D	39	46	89	1E	39	46	89	1E
00001824	39	46	89	1E	39	46	88	1E	58	46	89	1E	FA	49	D4	1E
00001840	32	46	89	1E	FA	49	D6	1E	3B	46	89	1E	FA	49	86	1E
00001856	3A	46	89	1E	1E	80	F4	1E	3B	46	89	1E	27	14	1C	1E
00001872	38	46	89	1E	30	3E	03	1E	35	46	89	1E	30	3E	18	1E
00001888	38	46	89	1E	52	69	63	68	39	46	89	1E	00	00	00	00
00001904	00	00	00	00	00	00	00	00	00	00	00	00	50	45	00	00
00001920	4C	01	04	00	E1	17	0C	50	00	00	00	00	00	00	00	00
00001936	E0	00	02	21	0B	01	09	00	00	34	00	00	00	34	00	00
00001952	00	00	00	00	75	18	00	00	00	10	00	00	00	50	00	00
00001968	00	00	67	45	00	10	00	00	00	02	00	00	05	00	00	00
00001984	00	00	00	00	05	00	00	00	00	00	00	00	00	A0	00	00
00002000	00	04	00	00	00	00	00	00	02	00	00	00	00	00	10	00

I located the MZ signature at offset 1684 so I copied out the binary data from offset 1684 to the end of the file. The resulting file was SHA256 ee14dcd20b2ee118618e3a28db72cce92ccd9e85dd8410e02479d9d624934b13 which was detected as the ZeroAccess Trojan.

Locating Other ZeroAccess Files on System

Locating the infected services.exe file and extracting the binary data from the NTFS Extended Attributes only addresses one part of the infection. ZeroAccess drops other files on a system and I wanted to find these as well. However, I didn't want to cheat by looking at my Capture.bat log file. I wanted to verify how a system timeline worked by using the services.exe as a starting point to find the other malicious files.

Timelines are my go to tool when I'm trying find files associated with an infection. A timeline can reveal other activity occurring on a system around a certain time of interest. In this case, I was interested in seeing the activity around the time when the services.exe file was last modified as shown below. (the timeline was created with AnalyzeMFT).

	A	B	D	F	J
1	date	time	MACB	sourcetype	short
399	12/6/2012	18:05.9	M...	NTFS $MFT	/Windows/System32/services.exe
400	12/6/2012	18:05.9	..C.	NTFS $MFT	/Windows/System32/services.exe
401	12/6/2012	18:05.1	.A..	NTFS $MFT	/Windows/Installer/{5da39e95-8007-4308-c6cf-bcce61795d0d}/n
402	12/6/2012	18:05.1	M...	NTFS $MFT	/Windows/Installer/{5da39e95-8007-4308-c6cf-bcce61795d0d}/n
403	12/6/2012	18:05.1	..C.	NTFS $MFT	/Windows/Installer/{5da39e95-8007-4308-c6cf-bcce61795d0d}/n
404	12/6/2012	18:05.1	...B	NTFS $MFT	/Windows/Installer/{5da39e95-8007-4308-c6cf-bcce61795d0d}/n
405	12/6/2012	18:04.9	.A..	NTFS $MFT	/Windows/Installer/{5da39e95-8007-4308-c6cf-bcce61795d0d}/@
406	12/6/2012	18:04.9	M...	NTFS $MFT	/Windows/Installer/{5da39e95-8007-4308-c6cf-bcce61795d0d}/@
407	12/6/2012	18:04.9	..C.	NTFS $MFT	/Windows/Installer/{5da39e95-8007-4308-c6cf-bcce61795d0d}/@
408	12/6/2012	18:04.9	...B	NTFS $MFT	/Windows/Installer/{5da39e95-8007-4308-c6cf-bcce61795d0d}/@
409	12/6/2012	18:04.9	.A..	NTFS $MFT	/Windows/Installer/{5da39e95-8007-4308-c6cf-bcce61795d0d}/L
410	12/6/2012	18:04.9	M...	NTFS $MFT	/Windows/Installer/{5da39e95-8007-4308-c6cf-bcce61795d0d}/L
411	12/6/2012	18:04.9	..C.	NTFS $MFT	/Windows/Installer/{5da39e95-8007-4308-c6cf-bcce61795d0d}/L
412	12/6/2012	18:04.9	...B	NTFS $MFT	/Windows/Installer/{5da39e95-8007-4308-c6cf-bcce61795d0d}/L
413	12/6/2012	18:04.9	.A..	NTFS $MFT	/Windows/Installer/{5da39e95-8007-4308-c6cf-bcce61795d0d}
414	12/6/2012	18:04.9	M...	NTFS $MFT	/Windows/Installer/{5da39e95-8007-4308-c6cf-bcce61795d0d}
415	12/6/2012	18:04.9	..C.	NTFS $MFT	/Windows/Installer/{5da39e95-8007-4308-c6cf-bcce61795d0d}
416	12/6/2012	18:04.9	...B	NTFS $MFT	/Windows/Installer/{5da39e95-8007-4308-c6cf-bcce61795d0d}
417	12/6/2012	18:04.9	.A..	NTFS $MFT	/Windows/Installer/{5da39e95-8007-4308-c6cf-bcce61795d0d}/U
418	12/6/2012	18:04.9	M...	NTFS $MFT	/Windows/Installer/{5da39e95-8007-4308-c6cf-bcce61795d0d}/U
419	12/6/2012	18:04.9	..C.	NTFS $MFT	/Windows/Installer/{5da39e95-8007-4308-c6cf-bcce61795d0d}/U
420	12/6/2012	18:04.9	...B	NTFS $MFT	/Windows/Installer/{5da39e95-8007-4308-c6cf-bcce61795d0d}/U

	A	B	D	F	J
1	date	time	MACB	sourcetype	short
420	12/6/2012	18:04.9	...B	NTFS $MFT	/Windows/Installer/{5da39e95-8007-4308-c6cf-bcce61795d0d}/U
421	12/6/2012	18:02.8	.A..	NTFS $MFT	/Windows/Temp/fwtsqmfile00.sqm
422	12/6/2012	18:02.8	M...	NTFS $MFT	/Windows/Temp/fwtsqmfile00.sqm
423	12/6/2012	18:02.8	..C.	NTFS $MFT	/Windows/Temp/fwtsqmfile00.sqm
424	12/6/2012	18:02.8	...B	NTFS $MFT	/Windows/Temp/fwtsqmfile00.sqm
425	12/6/2012	18:00.1	.A..	NTFS $MFT	/Users/lab/AppData/Local/{5da39e95-8007-4308-c6cf-bcce61795d0d}/n
426	12/6/2012	18:00.1	M...	NTFS $MFT	/Users/lab/AppData/Local/{5da39e95-8007-4308-c6cf-bcce61795d0d}/n
427	12/6/2012	18:00.1	..C.	NTFS $MFT	/Users/lab/AppData/Local/{5da39e95-8007-4308-c6cf-bcce61795d0d}/n
428	12/6/2012	18:00.1	...B	NTFS $MFT	/Users/lab/AppData/Local/{5da39e95-8007-4308-c6cf-bcce61795d0d}/n
429	12/6/2012	18:00.1	.A..	NTFS $MFT	/Users/lab/AppData/Local/{5da39e95-8007-4308-c6cf-bcce61795d0d}/L
430	12/6/2012	18:00.1	M...	NTFS $MFT	/Users/lab/AppData/Local/{5da39e95-8007-4308-c6cf-bcce61795d0d}/L
431	12/6/2012	18:00.1	..C.	NTFS $MFT	/Users/lab/AppData/Local/{5da39e95-8007-4308-c6cf-bcce61795d0d}/L
432	12/6/2012	18:00.1	...B	NTFS $MFT	/Users/lab/AppData/Local/{5da39e95-8007-4308-c6cf-bcce61795d0d}/L
433	12/6/2012	18:00.0	.A..	NTFS $MFT	/Users/lab/AppData/Local/{5da39e95-8007-4308-c6cf-bcce61795d0d}
434	12/6/2012	18:00.0	M...	NTFS $MFT	/Users/lab/AppData/Local/{5da39e95-8007-4308-c6cf-bcce61795d0d}
435	12/6/2012	18:00.0	..C.	NTFS $MFT	/Users/lab/AppData/Local/{5da39e95-8007-4308-c6cf-bcce61795d0d}
436	12/6/2012	18:00.0	...B	NTFS $MFT	/Users/lab/AppData/Local/{5da39e95-8007-4308-c6cf-bcce61795d0d}
437	12/6/2012	18:00.0	.A..	NTFS $MFT	/Users/lab/AppData/Local/{5da39e95-8007-4308-c6cf-bcce61795d0d}/U
438	12/6/2012	18:00.0	M...	NTFS $MFT	/Users/lab/AppData/Local/{5da39e95-8007-4308-c6cf-bcce61795d0d}/U
439	12/6/2012	18:00.0	..C.	NTFS $MFT	/Users/lab/AppData/Local/{5da39e95-8007-4308-c6cf-bcce61795d0d}/U
440	12/6/2012	18:00.0	...B	NTFS $MFT	/Users/lab/AppData/Local/{5da39e95-8007-4308-c6cf-bcce61795d0d}/U
441	12/6/2012	18:00.0	.A..	NTFS $MFT	/Users/lab/AppData/Local/{5da39e95-8007-4308-c6cf-bcce61795d0d}/@
442	12/6/2012	18:00.0	M...	NTFS $MFT	/Users/lab/AppData/Local/{5da39e95-8007-4308-c6cf-bcce61795d0d}/@
443	12/6/2012	18:00.0	..C.	NTFS $MFT	/Users/lab/AppData/Local/{5da39e95-8007-4308-c6cf-bcce61795d0d}/@
444	12/6/2012	18:00.0	...B	NTFS $MFT	/Users/lab/AppData/Local/{5da39e95-8007-4308-c6cf-bcce61795d0d}/@

The timeline showed the services.exe file was modified on 12/06/12 at 18:05:09. The activity beforehand included the creation of files and folders in both the C:\Windows\Installer and C:\Users\Lab\AppData\Local folders. The file named n in both locations is the ZeroAcess Trojan.

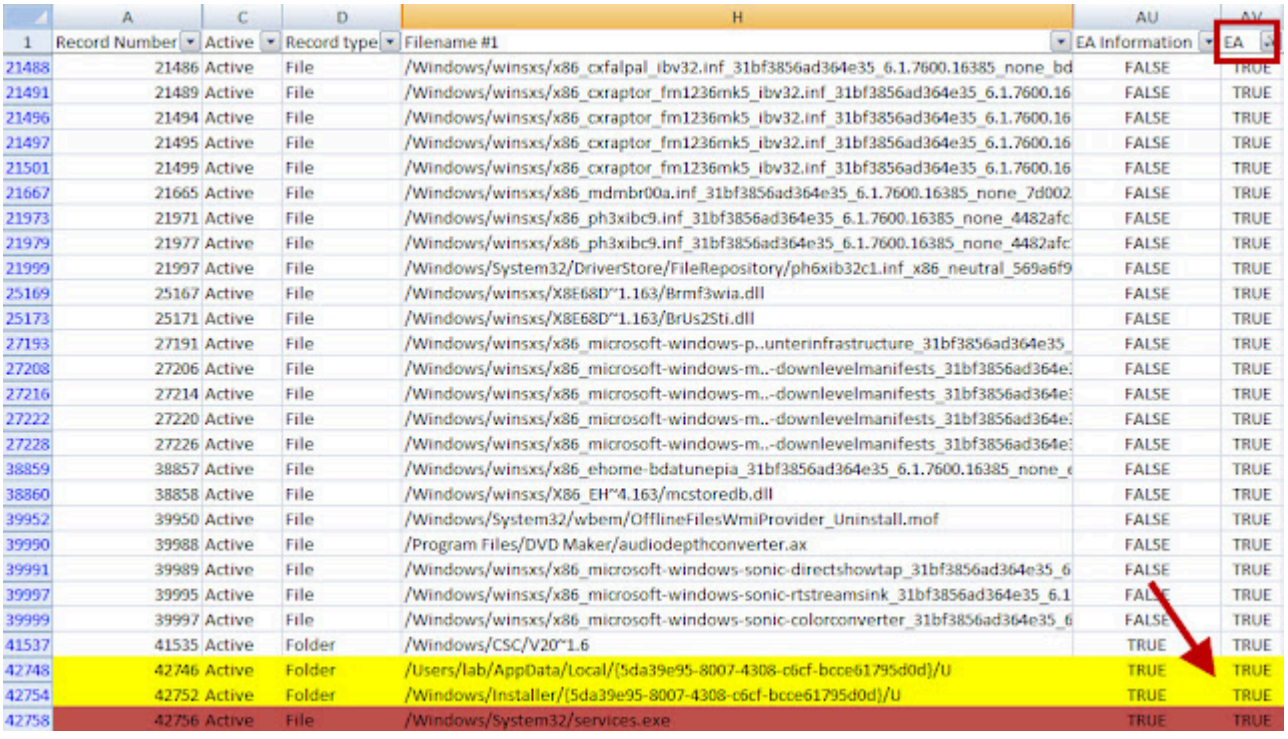
Summary

Storing data inside the NTFS Extended Attributes is a clever way to hide data. Based on the reports I read, ZeroAccess started to use this technique within the last year. So far it appears as if this is not a commonly used technique to hide data among malware. The only reports I could find only talked about ZeroAccess. However, in time I would not be surprised to see other malware families trying to use NTFS Extended Attributes. To see why just think about these two questions: how many tools are checking the NTFS Extended Attributes for every file and how many DFIR practitioners and IT staff are actually aware that malware can be hidden there? If I was a betting man I'd put my money on both answers being "very few". My eyes are opened to this hiding technique and I added a check underneath my Malware Searches examination step to account for it. If I ever need to determine if something is leveraging NTFS Extended Attributes then this is the process I will use.

1. Parse the MFT so that NTFS Extended Attributes are shown for every file. One tool to use is AnalyzeMFT with the command

analyzeMFT.py -f $MFT -o parsed_mft.txt

Import analyzeMFT's output into Excel or Calc (output is comma delimited format). Apply a sort on the $EA attribute being true to show every file with a NTFS Extended Attribute.



	A	C	D	H	AU	AV
1	Record Number	Active	Record type	Filename #1	EA Information	EA
21488	21486	Active	File	/Windows/winsxs/x86_cxfalpal_ibv32.inf_31bf3856ad364e35_6.1.7600.16385_none_bd	FALSE	TRUE
21491	21489	Active	File	/Windows/winsxs/x86_cxraptor_fm1236mk5_ibv32.inf_31bf3856ad364e35_6.1.7600.16	FALSE	TRUE
21496	21494	Active	File	/Windows/winsxs/x86_cxraptor_fm1236mk5_ibv32.inf_31bf3856ad364e35_6.1.7600.16	FALSE	TRUE
21497	21495	Active	File	/Windows/winsxs/x86_cxraptor_fm1236mk5_ibv32.inf_31bf3856ad364e35_6.1.7600.16	FALSE	TRUE
21501	21499	Active	File	/Windows/winsxs/x86_cxraptor_fm1236mk5_ibv32.inf_31bf3856ad364e35_6.1.7600.16	FALSE	TRUE
21667	21665	Active	File	/Windows/winsxs/x86_mdmbr00a.inf_31bf3856ad364e35_6.1.7600.16385_none_7d002	FALSE	TRUE
21973	21971	Active	File	/Windows/winsxs/x86_ph3xibc9.inf_31bf3856ad364e35_6.1.7600.16385_none_4482afc	FALSE	TRUE
21979	21977	Active	File	/Windows/winsxs/x86_ph3xibc9.inf_31bf3856ad364e35_6.1.7600.16385_none_4482afc	FALSE	TRUE
21999	21997	Active	File	/Windows/System32/DriverStore/FileRepository/ph6xib32c1.inf_x86_neutral_569a6f9	FALSE	TRUE
25169	25167	Active	File	/Windows/winsxs/X8E68D~1.163/Brmf3wia.dll	FALSE	TRUE
25173	25171	Active	File	/Windows/winsxs/X8E68D~1.163/BrUs2Sti.dll	FALSE	TRUE
27193	27191	Active	File	/Windows/winsxs/x86_microsoft-windows-p..unterinfrastructure_31bf3856ad364e35_	FALSE	TRUE
27208	27206	Active	File	/Windows/winsxs/x86_microsoft-windows-m..-downlevelmanifests_31bf3856ad364e3	FALSE	TRUE
27216	27214	Active	File	/Windows/winsxs/x86_microsoft-windows-m..-downlevelmanifests_31bf3856ad364e3	FALSE	TRUE
27222	27220	Active	File	/Windows/winsxs/x86_microsoft-windows-m..-downlevelmanifests_31bf3856ad364e3	FALSE	TRUE
27228	27226	Active	File	/Windows/winsxs/x86_microsoft-windows-m..-downlevelmanifests_31bf3856ad364e3	FALSE	TRUE
38859	38857	Active	File	/Windows/winsxs/x86_ehome-bdatunepia_31bf3856ad364e35_6.1.7600.16385_none_c	FALSE	TRUE
38860	38858	Active	File	/Windows/winsxs/X86_EH~4.163/mcstoredb.dll	FALSE	TRUE
39952	39950	Active	File	/Windows/System32/wbem/OfflineFilesWmiProvider_Uninstall.mof	FALSE	TRUE
39990	39988	Active	File	/Program Files/DVD Maker/audiodepthconverter.ax	FALSE	TRUE
39991	39989	Active	File	/Windows/winsxs/x86_microsoft-windows-sonic-directshowtap_31bf3856ad364e35_6	FALSE	TRUE
39997	39995	Active	File	/Windows/winsxs/x86_microsoft-windows-sonic-rtstreamsink_31bf3856ad364e35_6.1	FALSE	TRUE
39999	39997	Active	File	/Windows/winsxs/x86_microsoft-windows-sonic-colorconverter_31bf3856ad364e35_6	FALSE	TRUE
41537	41535	Active	Folder	/Windows/CSC/V20~1.6	TRUE	TRUE
42748	42746	Active	Folder	/Users/lab/AppData/Local/{5da39e95-8007-4308-c6cf-bcce61795d0d}/U	TRUE	TRUE
42754	42752	Active	Folder	/Windows/Installer/{5da39e95-8007-4308-c6cf-bcce61795d0d}/U	TRUE	TRUE
42758	42756	Active	File	/Windows/System32/services.exe	TRUE	TRUE

2. Locate the files' of interest MFT records. AnalyzeMFT's output contains this information.

3. Identify the partition's starting sector. One tool to use is TSK's mmls with the command:

mmls \\.\PHYSICALDRIVE#

4. Review the attributes associated with any files of interest. One tool to use is TSK's istat with the command

istat -o 2048 \\.\PHYSICALDRIVE3 12345

5. Make note of the NTFS Extended Attributes $EA_INFORMATION and $EA. In istat's output, next to the attribute name will be a value inside parentheses.

6. Dump the data from the $EA and $EA_INFORMATION for closer inspection. Use TSK's icat with the commands:

```
icat -o 2048 \\.\PHYSICALDRIVE# 12345-208-# > EA_INFO.txt
icat -o 2048 \\.\PHYSICALDRIVE# 12345-224-# > EA.txt
```

Happy Hunting
