RevBits

Cyber Security Solutions

OPERATION SAUDI

MONTHLY INTELLIGENCE BRIEFING

FEBRUARY 7, 2017

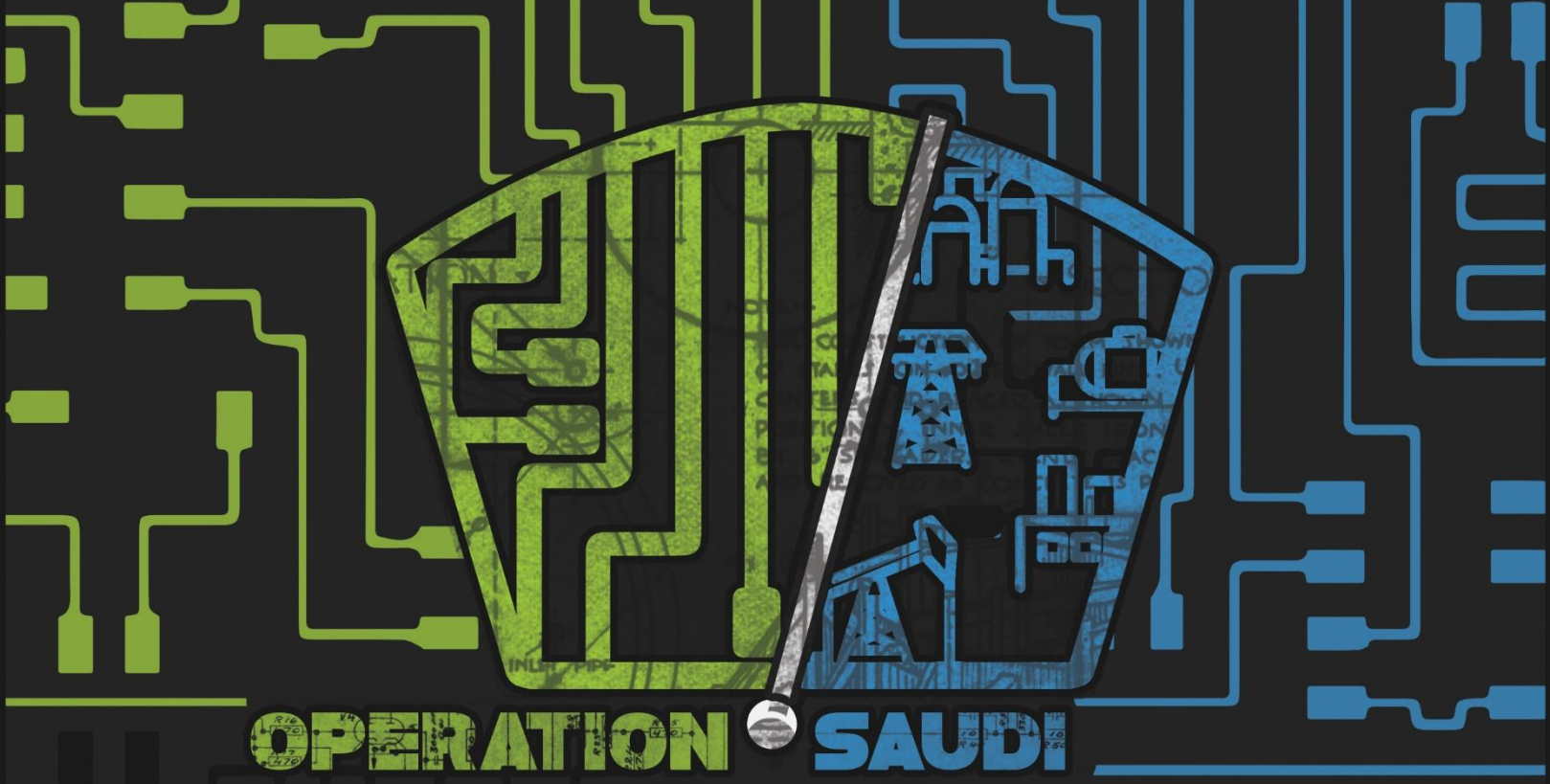

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Shamoon, a malicious wiper software, poses a continuing threat to the oil and gas industry since it was noticed in 2012. This software erases all networked workstations by gaining access to network credentials and installing itself on the system. Initially, the malware was effective, but the code was clumsy.

Today's version of **Shamoon** shows more skill and increasing knowledge from the actors and developers. **RevBits** recently analyzed a new, more sophisticated version of the known threat, **Shamoon**. Full details remain unknown regarding the origin of this malware, however it is suspected that **an Iranian group** is behind **the threat**.

SUPPORTIVE REASONING

While it is very difficult to assign or attribute malicious software to any specific nation or individual origin, we believe **Shamoon** originates from **Yemen** or **Iran** based on language identifiers found in the code. One can plainly see use of the **Persian** and **Arabic** languages set as the resource language identifier in the most recent samples of **Shamoon** we have analyzed. Images are below.

Further supporting this conclusion is that the victims of **Shamoon** are all clustered **in Saudi Arabia**. We have reviewed multiple malware samples from this specific country.

Curiously, the actors did not encrypt system data from infiltrated networks or hold the same for ransom. Instead, they simply replaced the data with a **JPEG photo**.

Iran also has the ability to gain access to Saudi energy Companies networks in order to wipe their systems, essentially crippling them.

Iran based cyber attacks also generally present with technical crudeness. This lack of sophistication is also found in the **Shamoon** samples. There is evidence of .NET language used in the development of the program and public tools and codes have been reused, all of which are attributable to Iran's style of cyber threat.

The current political climate is also considered in forming our conclusion. For instance, we note that **Shamoon** is the Arabic pronunciation of "***Simon***", which is a symbolically Hebrew name. We believe it is possible that the criminals behind this attack used this name in the hopes of casting suspicion on Hebrew or Jewish hackers. In addition, **Shamoon** is written "***Shamoun***" by native Arabic speakers. The deliberate misspelling could be another tactic to throw investigators off the true trail to the source.

We feel all of the signatures and evidence extracted from the malware point to Iran as the actor-nation in all **Shamoon** attacks in **Saudi Arabia**.

We have not included any names of clients or identification of the source of the samples or data we analyzed, in accordance with our **NDA** and confidentiality agreements.

THE PROGRESSION OF SHAMOON FROM V2 TO V3

Shamoon follows the common process of Iranian cyber threats. It began in rudimentary but effective form and is taking shape as the developers gain knowledge and resources.

The industry's first exposure **to Shamoon** occurred in 2012 and the malware takes its name from a PDB path embedded in early versions of the program. The string found is: "**c:\shamoon\ArabianGulf\wiper\release.pdb**" and we believe reference to *the Arabian Gulf* is a further attempt by the actors to obfuscate their true identity. *Arabian Gulf* is a term most often used by Arab countries to refer to their location, in rejection of the universally accepted Persian Gulf designation.

The **Shamoon** malware has grown progressively more sophisticated, although it still lacks in comparison to actors from other parts of the world. We have found that attackers occasionally altered the metadata as it relates to date and time as well as language identifiers. We conclude that this activity is meant to mislead any forensic analysts.

For comparison and to illustrate the evolution and ever changing data associated with **Shamoon** attacks, we may review one of the first appearances of malware with the **MD5** hash of **B14299FD4D1CBFB4CC7486D978398214**, which occurred **in mid August of 2012**. This instance of the software does not show any changes to the date and time of its compilation. The malware sample's complication date and time reads **08/09/2012 10:46:22PM** which appears to be objectively correct.

As we see in later occurrences of **Shamoon** malware breaches, there are changes to the compilation date and time. The most recent attack we have access to occurred recently in Saudi Arabia, but conversely shows a compilation time and date of **06/07/2011 01:30:56 PM**. We know this is untrue based on the other data we reviewed which contradicts the displayed compilation time, including the embedded credentials, the initial breach to the network, and the time and date on the execution of the attack.

It is very easy to link occurrences of **Shamoon** attacks because the actors clearly re-use code in all versions of **Shamoon**. We surmise this is either a deliberate attempt to send a message to Saudi Arabia, or indication that the actors are unsophisticated or do not care about this shortcoming in their code. The **Shamoon** samples we analyzed from separate entities and clients in Saudi Arabia are linked by this code re-use. Some of the samples definitively trace to Iran.

CRUMBS ON THE DIGITAL TRAIL: CODE RE-USE

One of the ways **Shamoon** is easy to identify is because of the pieces of code that are used over and over again. The actors also used similar strings and techniques in the code. We look for this known codes during malware analysis.

- All versions of **Shamoon** malware use RawDisk from Eldos for kernel-mode driver to have low level disk access. See, https://www.eldos.com/rawdisk/
- The actors use an unusual **IDE**, **WxWidget**, which helps us zero in on **Shamoon**. All samples we've analyzed contains embedded "**Dinkumware**" license strings and **a _ts_open** function, which we note during forensics.
- The license key holds the key here too. All of the S**hamoon** samples we analyzed use the license key of a fictitious company. There's no obvious reason a company like South Machine which is associated with the binnatova@bsunanotechnology.com key, needs a RawDisk driver from Eldos.

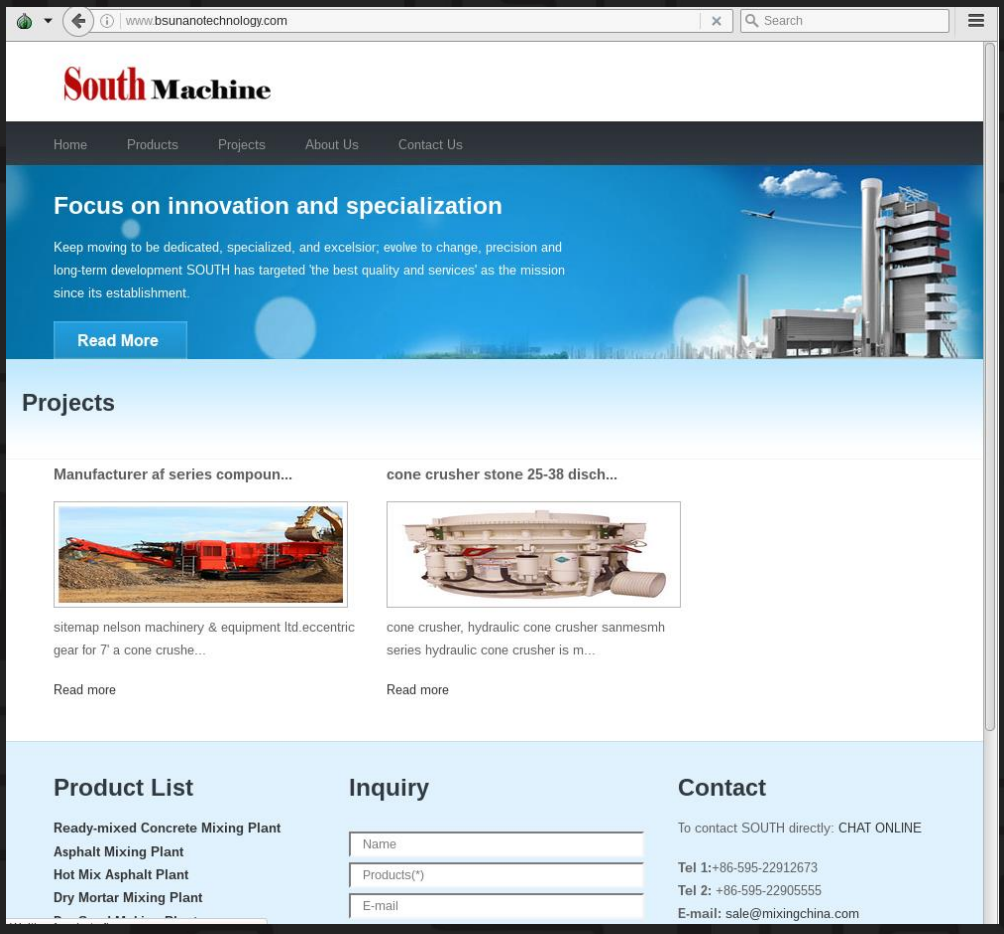

Fig 1 – *bsunanotechnology.com website used for RawDisk driver license*

- Additionally, the samples all contain RawDisk driver as a resource and have **x64** and **x32** payload embedded inside. **4-byte XOR** encryption with similar function is also present in all versions.
- The earliest known **Shamoon** samples contain no string obfuscation but current samples show evidence of this. There is even process hollowing or forking technique at play in **Shamoon v3**. The actors are gaining knowledge and becoming stronger, which makes it important to study them now so we can follow their process. We analyzed the new forked/hollowed process and still find evidence of known **Shamoon** patterns.

SHAMOON V3

The most recent samples we analyzed showed more sophistication than **Shamoon** is known for. Therefore, we must first form a viable link between **Shamoon v3** and earlier samples of the malware.

Upon deeper review, some things in the coding has changed, but much remains consistent.

One advancement in **Shamoon v3** is a hollowing/forking processes never before seen in this malware. This version is cloaked, encrypted, and uses anti-debugging techniques as well.

Another change includes manual modifications to both the compilation date and time and the language identifier, which we trace back to Yemen and Iran.

This sample has a language identifier of **1065 (0x429)** which indicates that Persian was the OS language where the code was written in.

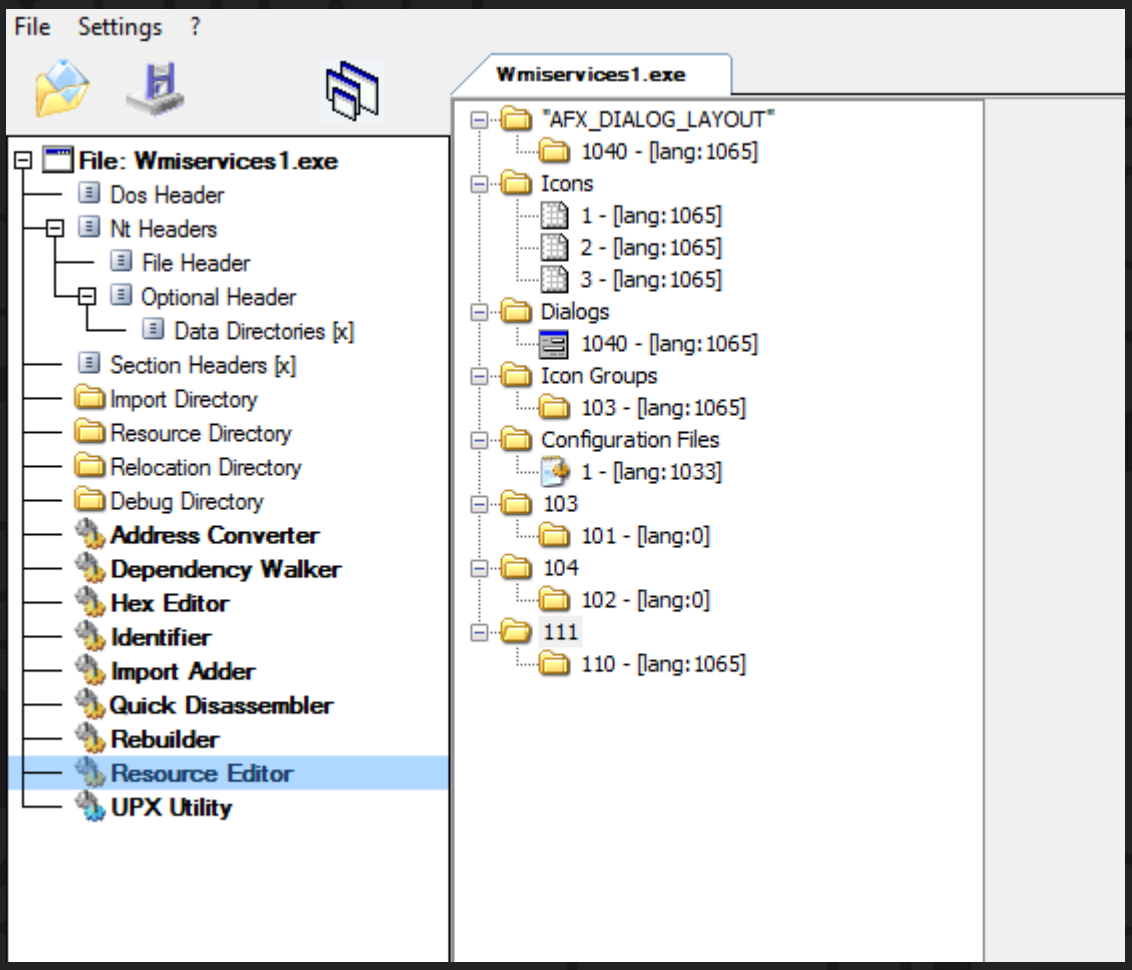

Fig 2 –*Examples of Persian language identifier in **Shamoon v3***

0x0482	Occitan (oc)	0x82	**LANG_OCCITAN**	France (FR)	0x01	**SUBLANG_OCCITAN_FRANCE**
0x0448	Odia (or)	0x48	**LANG_ORIYA**	India (IN)	0x01	**SUBLANG_ORIYA_INDIA**
0x0463	Pashto (ps)	0x63	**LANG_PASHTO**	Afghanistan (AF)	0x01	**SUBLANG_PASHTO_AFGHANISTAN**
0x0429	Persian (fa); see note 6	0x29	**LANG_PERSIAN**	Iran (IR)	0x01	**SUBLANG_PERSIAN_IRAN**
0x0415	Polish (pl)	0x15	**LANG_POLISH**	Poland (PL)	0x01	**SUBLANG_POLISH_POLAND**
0x0416	Portuguese (pt)	0x16	**LANG_PORTUGUESE**	Brazil (BR)	0x01	**SUBLANG_PORTUGUESE_BRAZILIAN**

Fig 3 – *Key To Language identifier ID code, from Microsoft*

This sample illustrates a manually altered compilation date. This is another recent advancement seen in Shamoon in an attempt to block forensic analysis.

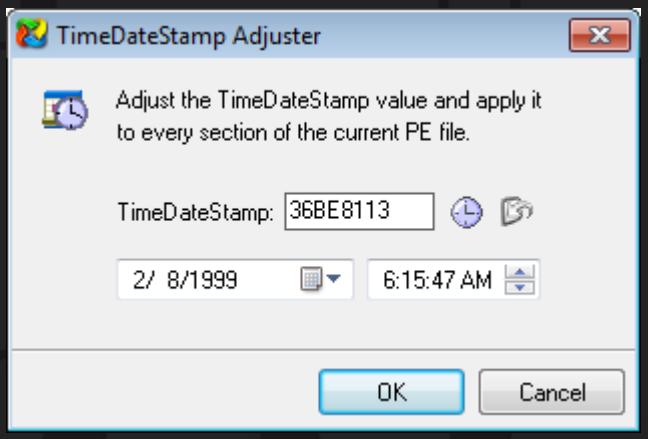

Fig 4 – *Most recent **Shamoon** sample with compilation date changed*

It is more difficult but not impossible to dissect and review the code in **Shamoon v3**. Evidence of forking illustrates advancement in code and that development is ongoing. The actors are gaining skills they can use to make forensics more difficult which shows growth.

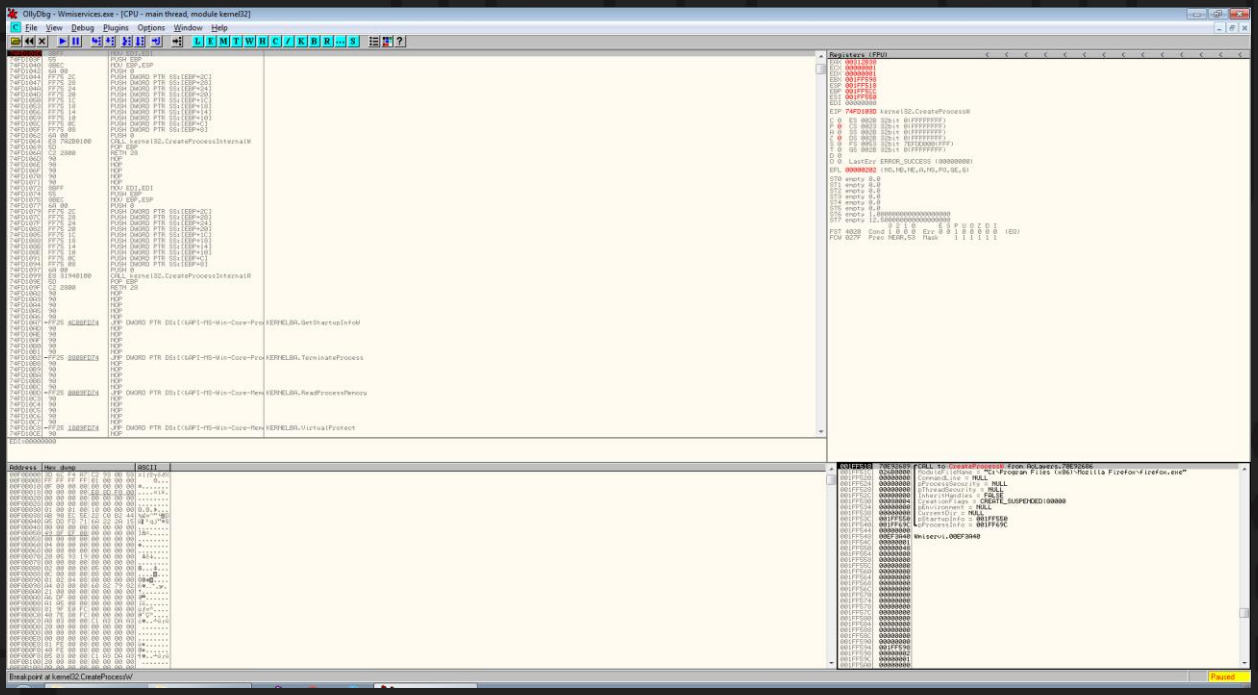

Fig 5 – *Process Forking/Hollowing by **Shamoon v3***

Shamoon now performs process forking through browsers installed as hollow process names in the endpoint machine.

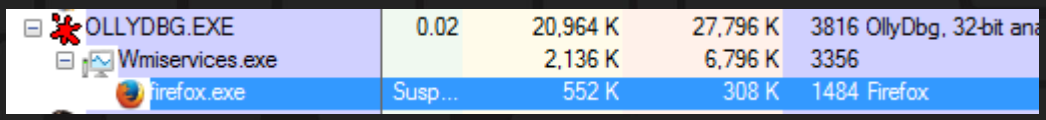

Fig 6 – *Firefox process spawned as suspended process for use as process hollowing*

Shamoon v3 also uses **VBScript** instead of batch script for self-deleting:

```
WScript.Sleep(10 * 1000)
On Error Resume Next
Set WshShell = CreateObject("Scripting.FileSystemObject")
While WshShell.FileExists("C:\Users\Win7KernelDev\Desktop\samples\Wmiservices.exe")
WshShell.DeleteFile "C:\Users\Win7KernelDev\Desktop\samples\Wmiservices.exe"
Wend
WScript.Sleep(10 * 1000)
WshShell.DeleteFile "C:\Users\WIN7KE~1\AppData\Local\Temp\C-Dlt-C-Org-T.vbs"
Set WshShell = Nothing
```

Fig 7 – *VBScript deleting original malware file*

The changes are somewhat isolated thus far. Once the forked process is dumped, **Shamoon v3** behaves almost exactly like the original samples. We can easily see instances of the re-used code which points directly to **Shamoon**. This is further confirmation that the recently analyzed threat is a more sophisticated version of the known threat, **Shamoon**.

For instance, this malware sample also has the **Dinkumware** C library copyright notice.

```
.data:0042F160 aCopyrightC1992 db 'Copyright (c) 1992-2004 by P.J. Plauger, licensed by Dinkumware, '
.data:0042F160                 db 'Ltd. ALL RIGHTS RESERVED.',0
.data:0042F1BB                 align 10h
.data:0042F1C0 off_42F1C0      dd offset off_4285AC    ; DATA XREF: .rdata:0042BCF4↑o
```

Fig 8 – ***Dinkumware*** *License notice, shared across all* ***Shamoon*** *samples*

Further examples of unique code used across all samples in the **Shamoon** malware family are below.

- Using Registry values to discover boot device to identify location of MBR

```
debug020:006F3D38 aSystembootdevi:
debug020:006F3D38 unicode 0, <SystemBootDevice>,0
debug020:006F3D5A db 0ABh ; ½
```

Fig 9 - ***SystemBootDevice*** *registry value for identifying MBR location*

```
debug020:006F3D78 aRdisk_0:
debug020:006F3D78 unicode 0, <rdisk(>,0
debug020:006F3D86 db 0ABh ; ½
```

Fig 10 – *Parsing "rdisk(" piece of returned registry value*

- Identical INF strings:

  Shamoon v2 strings:

```
debug024:00AF4310 aInfNetimm173_pnf:
debug024:00AF4310 unicode 0, <\inf\netimm173.pnf>,0
debug024:00AF4336 db 0ABh ; ½
```

  And

```
debug024:00AF42C8 aInfUsbvideo324_pnf:
debug024:00AF42C8 unicode 0, <\inf\usbvideo324.pnf>,0
debug024:00AF42F2 db 0ABh ; ½
```

  And exact same strings in **Shamoon v3**:

```
debug020:006F1B38 aInfUsbvideo324:
debug020:006F1B38 unicode 0, <\inf\usbvideo324.pnf>,0
debug020:006F1B62 db 0ABh ; ½
```

  And

```
debug024:00AF4310 aInfNetimm173_pnf:
debug024:00AF4310 unicode 0, <\inf\netimm173.pnf>,0
debug024:00AF4336 db 0ABh ; ½
```

- Identical CLSID:

  Shamoon v2:

```
.data:0042F000                 dd offset a82b5234fDf6146 ; "{82B5234F-DF61-4638-95D5-341CAD244D19}"
.data:0042F004                 dd offset aSystemCurrentc ; "System\\CurrentControlSet\\Control\\Net"...
.data:0042F008 off_42F008      dd offset off_4285AC    ; DATA XREF: .rdata:0042BA2C↑o
.data:0042F008                                         ; .rdata:off_42BA50↑o ...
```

Shamoon v3:

```
.rdata:000F8318 aSystemCurrentc:                        ; DATA XREF: .data:000FF004↓o
.rdata:000F8318 unicode 0, <System\CurrentControlSet\Control\NetworkProvider\Order>,0
.rdata:000F8386 align 4
.rdata:000F8388 a82b5234fdf6146 db '{82B5234F-DF61-4638-95D5-341CAD244D19}',0
.rdata:000F8388                                         ; DATA XREF: .data:000FF000↓o
.rdata:000F83AF align 10h
```

- Identical License strings:

  Shamoon v2:

```
debug024:009B1F90 unicode 0, <8F71FF7E2831A05D0B88FDAACFAC818E936FCAAA4534041804196 62BE>
debug024:009B1F90 unicode 0, <D76E9D70384F056F03ADF3C917CB8C3EE12832F7A7EC3E234BC7FBD04>
debug024:009B1F90 unicode 0, <76CFC9F58AC1A1C248DA06E531D133A071>,0
debug024:009B20BA db 0ABh ; ½
debug024:009B20BB db 0ABh ; ½
```

  Shamoon v3:

```
debug020:006F1F68 unicode 0, <8F71FF7E2831A05D0B88FDAACFAC818E936FCAAA4534041804196 62BE>
debug020:006F1F68 unicode 0, <D76E9D70384F056F03ADF3C917CB8C3EE12832F7A7EC3E234BC7FBD04>
debug020:006F1F68 unicode 0, <76CFC9F58AC1A1C248DA06E531D133A071>,0
debug020:006F2092 db 0ABh ; ½
```

From these figures we can see that **Shamoon v2** and **Shamoon v3** are easily linked through code similarities. There is obvious advancement, however, including improved anti-debugging and anti-analysis techniques. Additionally, once the conclusion is reached that the recent samples provided to us are **Shamoon**, we can make further conclusions about the origin of this attack.

HOW SHAMOON WORKS - THE WIPING PROCESS

Shamoon deletes every file on a computer through a combination of Windows API calls and Eldos RawDisk driver. **Shamoon** gains access to the disk using RawDisk driver. From there, it begins wiping MBR.

Some samples show the files overwritten with images including a burning US flag or a photograph of a Syrian boy who drowned as his family tried to cross from Turkey to Greece. Occasionally, the data is replaced by random junk.

Fig 11 – *Photo used by **Shamoon** malware to replace disk data*

It is possible to recover some previously deleted or mirrored files since **Shamoon** doesn't affect the free drive space. The majority of data is sadly gone, however.

Once **Shamoon** is executed and date/time for wiping arrives, it installs a driver and opens all disk and partitions using the RawDisk helper library. Eventually, it overwrites all the sectors.

The wiping function is inherent in all **Shamoon** samples, as illustrated in the samples below.

```
if ( SystemTime.wMonth <= Mon11
  && (SystemTime.wMonth != Mon11
   || SystemTime.wDay <= Day26
   && (SystemTime.wDay != Day26
    || SystemTime.wHour <= v7 && (SystemTime.wDay != Day26 || SystemTime.wHour != v7 || SystemTime.wMinute <= v8))) )
{
  v2 = v9;
}
else
```

Fig 12 – ***Shamoon*** *with hardcoded date and time for wiping*

```
  hFile = CheckDateTime(word_159050, 0xC0000000);
  if ( hFile != (HANDLE)-1 )
  {
    WriteFile(hFile, lpBuffer, nNumberOfBytesToRead, &NumberOfBytesWritten, 0);
    v0 = WipeWithDriver(hFile);
    sub_1272B0(
      hFile,
      v0,
      SHIDWORD(v0),
      10 * dword_157B50 * dword_157B48,
      (unsigned __int64)(10i64 * dword_157B50 * dword_157B48) >> 32,
      lpBuffer,
      nNumberOfBytesToRead,
      dword_15932C);
    CloseHandle(hFile);
  }
  ms_exc.registration.TryLevel = -2;
```

Fig 13 – *Call to Wipe function*

Although devastating, this function is not advanced. It is the same function used in the Sony hack. In fact, the Sony hack was more sophisticated still, as it used a stolen license instead of using a trial one-month license and rolling back the date and time on the infected machine in order to use the driver.

CLASSIFYING THE ACTORS BEHIND SHAMOON

The attackers show some skill in finding their ways into corporate networks, initially through exploiting web vulnerabilities of companies and from there elevating their privilege and finding their ways to inside network. After finding required credentials and access for executing **Shamoon** across the network, attackers compile their sample with client specific information and they deploy it through the network.

The technical skill of the actors was weak initially but shows improvement. The previous version of **Shamoon** lacked hollowing/forking and had clumsy programming. In fact, numerous basic **C++** data type mistakes can clearly be seen in their code. This shows lack of **C++** experience and skill in the authors of **Shamoon**. With **Shamoon v3** and advanced multi-staged process hollowing/forking, they show some signs of capability, but still use all the old code. Just for reference, here are two samples of their coding practices which clearly demonstrates their substandard **C++** skill:

- Use of registry for identifying OS architecture:
  All samples of **Shamoon** malware use a strange and weak method for identifying OS architecture which can be accomplished with a very simple API call. Instead, they store two copies of "**amd64**" string, one lower case, one upper case and they query a registry value and compare the results once time with "**amd64**" and 2nd time with "**AMD64**". There are multiple problems with this approach:
  a) Instead of using two **amd64** strings (one lowercase and one uppercase), they could use case-insensitive string comparison function. To do otherwise clearly points to lack of **C++** expertise in authors
  b) They unnecessarily encrypt and decrypt both "**amd64**" strings.
  c) Their comparison is flawed, manually changing "**amd64**" string to "**aMd64**" in registry would force **Shamoon** to assume 32-bit OS.
  d) Too many API calls are used for just checking OS architecture which is inefficient.
- Use of wrong (double) size for decrypting Unicode strings:
  The authors are unaware of the data types in **C++** and therefore they make the mistake of double allocating required space for copying Unicode strings. They also unnecessarily allocate new strings in memory (instead of decoding strings in-place) and they are never freed.

```
v0 = 0;
if ( !RegOpenKeyExW(HKEY_LOCAL_MACHINE, EnvironmentRegPath, 0, 0x20019u, &phkResult) )
{
  Type = 0;
  cbData = 100;
  if ( !RegQueryValueExW(phkResult, PROCESSOR_ARCHITECTURE_STR, 0, &Type, &Data, &cbData) )
  {
    v1 = cbData;
    if ( cbData > 0 )
    {
      memcpy_0(v7, &Data, cbData);
      v7[v1 >> 1] = 0;
      if ( !wcscmp((const unsigned __int16 *)AMD64_STR, (const unsigned __int16 *)v7)
        || !wcscmp((const unsigned __int16 *)amd64_lower_str, (const unsigned __int16 *)v7) )
      {
        v0 = 1;
      }
    }
  }
  RegCloseKey(phkResult);
}
```

Fig 14 – *Odd OS Architecture check code*

As mentioned previously, most attacks originating from Iran do not show great technical sophistication. This is specifically true in their malware. Most malware coming from Iran is written in .NET which makes it easy to decompile them. **Shamoon** is actually one of the few malware originating in Iran that is written in **C++**.

Additionally, we have never seen any kernel module/code from Iranian hackers and **Shamoon** is no exception. All kernel task is done by RawDisk driver from Eldos.

IOC AND PROTECTION

The SHA1 and MD5 checksum of main dropper samples we've received and analyzed are listed below:

Ver	SHA1	MD5
V1	7C0DC6A8F4D2D762A07A523F19B7ACD2258F7ECC	B14299FD4D1CBFB4CC7486D978398214
V1	502920A97E01C2D022AC401601A311818F336542	D214C717A357FE3A455610B197C390AA
V2	E7C7F41BABDB279C099526ECE03EDE9076EDCA4E	5446F46D89124462AE7ACA4FCE420423
V3	0A4FFCE8F301546100D7B00BA017F5E24D1B2D9B	FB21F3CEA1AA051BA2A45E75D46B98B8
V3	279FF728023EEAA1715403EC823801BF3493F5CA	0CCC9EC82F1D44C243329014B82D3125

CONCLUSION

The malicious wiper software **Shamoon** continues to be a threat to the oil and gas sector in Saudi Arabia and likely originates from Iran based on clues in the language identifier in the coding and similarities with other known Iranian malware actors.

Recent samples of malware identified as **Shamoon** show both clear links to earlier versions through code and implemented techniques as well as a growing acumen in the actors. Cumbersome processes and clumsy code remains, but more advanced anti-debugging and anti-analysis techniques have emerged.

It is crucial to continue to analyze samples as they strike so we can fully assess this threat.

THE REVBITS TEAM'S SECURITY PROWESS IS PROVEN AND WE ARE COMMITTED TO PROVIDING COMPANIES WITH NEXT-LEVEL SUPPORT AGAINST SHAMOON AND ALL SUSPICIOUS ACTIVITY.

RevBits
Cyber Security Solutions