GitHub - knight0x07/BiBi-Windows-Wiper-Analysis: Technical Analysis of Bibi-Windows Wiper Targeting Israeli Organizations

By knight0x07

Archived: 2026-04-05 13:42:09 UTC

On 30th October, Security Joes Incident Response team discovered a new Linux Wiper named **"BiBi-Linux"** Wiper been deployed by Pro-Hamas Hacktivist group to destroy their infrastructure. And then on November 1 2023, ESET Research tweeted about a Windows version of the Bibi Wiper deployed by BiBiGun, a Hamas-backed hacktivist group that initially debuted during the 2023 Israel-Hamas conflict.

In this post, we will look at the Windows version of the BiBi Wiper known as the **"BiBi-Windows Wiper"**

Technical Analysis

Upon execution, the BiBi-Windows Wiper checks to see if any arguments have been passed to the BiBi Wiper. The arguments here is the directory to be destroyed - `bibi.exe <directory_to_destroyed>` . If no argument is provided, it performs the following routine for fetching the target drives

- Reads the hardcoded path: "C:\Users"
- Gets the currently available disk drives using `GetLogicalDrives()` where the return value is the bitmask, then it iterates through the A-Z (26) drives. It next does a bittest with the retrieved bitmask to determine the accessible drives on the system and appends ":\" to the drive name.
- Here it excludes the C drive by checking for `i != 2` where 2 is the bitmask position for C drive
- Then for the available drives except the C drive it executes the `GetDriveTypeA()` which retrives the drive type, The BiBi-Windows Wiper here only targets the following drive types:
    - DRIVE_FIXED
    - DRIVE_REMOVABLE
    - DRIVE_RAMDISK

```
LODWORD(bitmask_drives) = GetLogicalDrives();
for ( i = 26; i >= 0; --i )
{
  if ( i != 2 && _bittest((const signed __int32 *)&bitmask_drives, i) )
  {
    *((_QWORD *)&v48 + 1) = 15i64;
    *(_QWORD *)&v48 = 1i64;
    path[0] = (LPCSTR)(unsigned __int8)(i + 65);
    concat(path, ":\\", 2i64);
    v17 = (const CHAR *)path;
    if ( *((_QWORD *)&v48 + 1) >= 0x10ui64 )
      v17 = path[0];
    v18 = GetDriveTypeA(v17);
    v21 = v18 - 2;
    if ( (v21 & 0xFFFFFFFA) == 0 && v18 != 7 )
```

Therefore the BiBi-Windows Wiper targets

- The hardcoded directory - "C:\Users"
- And all available drives except from "C:" drive

Further it prints the target directories on the console and retrieves the NumberOfProcessors from `GetNativeSystemInfo()` and based on the numberofprocessors it calculates the threads and then prints it onto the console.

```
[+] Path: C:\Users
[+] Path: A:\
[+] CPU cores: 2, Threads: 6
```

Further it creates a new thread which reads the commands stored in reverse, & then creates a new process using `CreateProcessA` to execute those commands. Following are the commands

- cmd.exe /c bcdedit /set {default} recoveryenabled no - Disables Windows Recovery Environment
- cmd.exe / c bcdedit / set {default} bootstatuspolicy ignoreallfailures - Force the system to boot normally rather than into the Windows Recovery Environment
- cmd.exe /c wmic shadowcopy delete - Delete Volume Shadow Copies using WMIC
- cmd.exe /c vssadmin delete shadows /quIet /all - Delete Volume Shadow Copies using VssAdmin

```
memcpyt(v28, "lla/ teIuq/ swodahs   eteled nimdassv  c/ exe.dmc", 49i64);
v0 = v28;
if ( v30 >= 0x10 )
  v0 = (__int64 *)v28[0];
v1 = (char *)v0 + v29;
v2 = v28;
if ( v30 >= 0x10 )
  v2 = (__int64 *)v28[0];
reverse_string(v2, v1);
v3 = v28;
if ( v30 >= 0x10 )
  v3 = (__int64 *)v28[0];
v4 = (CHAR *)(CommandLine - (CHAR *)v3);
do
{
  v5 = *(_BYTE *)v3;
  *((_BYTE *)v3 + (_QWORD)v4) = *(_BYTE *)v3;
  v3 = (__int64 *)((char *)v3 + 1);
}
while ( v5 );
StartupInfo.cb = 104;
*(_OWORD *)&StartupInfo.dwFillAttribute = 0i64;
*(_OWORD *)&StartupInfo.lpReserved = 0i64;
StartupInfo.wShowWindow = 0;
*(_OWORD *)&StartupInfo.lpTitle = 0i64;
*(_OWORD *)&StartupInfo.dwXSize = 0i64;
*(_OWORD *)&StartupInfo.lpReserved2 = 0i64;
*(_OWORD *)&StartupInfo.hStdOutput = 0i64;
CreateProcessA(0i64, CommandLine, 0i64, 0i64, 0, 0x8000001u, 0i64, 0i64, &StartupInfo, &ProcessInformation);
```

Furthermore it creates another thread which executes of the Main Wiper routines. The Wiper routines perform the following actions

- The Arguments to the Wiper Function are:
  - Arg1 - Path of the Directory to be destroyed (Could be provided by the Operator or retrived as explained before)
  - Arg2 - Number of threads
- Then it initiates an infinite loop where the counter is the Round "[+] Round %d\n" value - therefore once the Wiper is executed it would keep destroying the data infinitely!
- Further based on the number of threads, it creates multiple threads in a loop which execute the main Wiper function.
- **The BiBi-Windows wiper excludes the files with ".exe", ".dll" and ".sys" extension**

```
m_copy(v120, (__int64)L".exe", 4ui64);
v121[0] = 0i64;
v121[2] = 0i64;
v121[3] = (void *)7;
m_copy(v121, (__int64)L".dll", 4ui64);
v122[0] = 0i64;
v122[2] = 0i64;
v122[3] = (void *)7;
m_copy(v122, (__int64)L".sys", 4ui64);
```

Wiper Function Analysis

Now lets understand how BiBi-Windows Wiper destroys the data on the machine.

- The Wiper firsty walks through the directory to be encrypted recursively and then retrives the file size of the target files.
- Then it executes the wiper function in loop against the target files based on the file size, where the second argument to the wiper function is "0xFF"
- The Wiper function implements the `Mersenne Twister PseudoRandom Number Generator Algorithm` which generates random numbers as shown below.

```
v8 = v6
   + 435 * (HIBYTE(v14) ^ (435 * (BYTE2(v14) ^ (435 * (BYTE1(v14) ^ (435 * ((unsigned __int8)v14 ^ 2216829733)))))));
v9 = 1i64;
v5[1] = v8;
do
{
  ++v7;
  v8 = v9++ + 1812433253 * (v8 ^ (v8 >> 30));
  *(v7 - 1) = v8;
}
while ( v9 < 624 );
*v5 = 624;
*v4 = v5;
}
```

Mersenne Twister PseudoRandom Number Generator Algorithm

```
v15 = ((((v14 >> 11) & a1[1249] ^ v14) & 0xFF3A58AD) << 7) ^ (v14 >> 11) & a1[1249] ^ v14;
return ((v15 & 0xFFFFDF8C) << 15) ^ v15 ^ ((((v15 & 0xFFFFDF8C) << 15) ^ v15) >> 18);
```

- The random number generated from the function `sub_140008BF0()` then performs modulus (%) with the value "0xFF + 1 = 100" and the output (remainder) of the modulus operation is the byte which is been overwritten in the target file byte by byte.

```
hardcoded_val = v3 + 1;
do
{
  rand_num = 0;
  v13 = 0;
  if ( v3 )
  {
    rand_num = sub_140008BF0(v5);
    v13 = 0xFFFFFFFF;
  }
}
while ( rand_num / hardcoded_val >= v13 / hardcoded_val && v13 % hardcoded_val != v3 );
return rand_num % hardcoded_val;
```

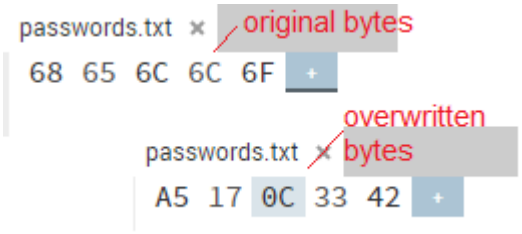
0xFF

Example below against a file consisting of the data "hello" and is been overwritten by these 5 random bytes in a loop.

ogfile_val -> rand_num % (hardcoded_val + 1) -> overwrite_val

```
 h - 0xDD9B40A5 % 0x100 -> 0xA5
e - 0x37905317 % 0x100 -> 0x17
l - 0x54B7A20C % 0x100 -> 0x0C
l - 0xDBD10533 % 0x100 -> 0x33
o - 0x18ED3C42 % 0x100 -> 0x42
```

File Overwritten with Random bytes - **Destroyed!**



Now once the data in the file is overwritten with random bytes and the file is been destroyed, the BiBi Wiper changes the name of the file in the following manner:

- It calls the previous Mersenne Twister function again where the hardcoded argument this time is "0x3D". So once the random number is generated it performs modulus with "0x3E" and the output (remainder) value is stored in the similar manner.
- Further the output remainder value is been multiplied by 2 and then the value is indexed against a wide string as represented below

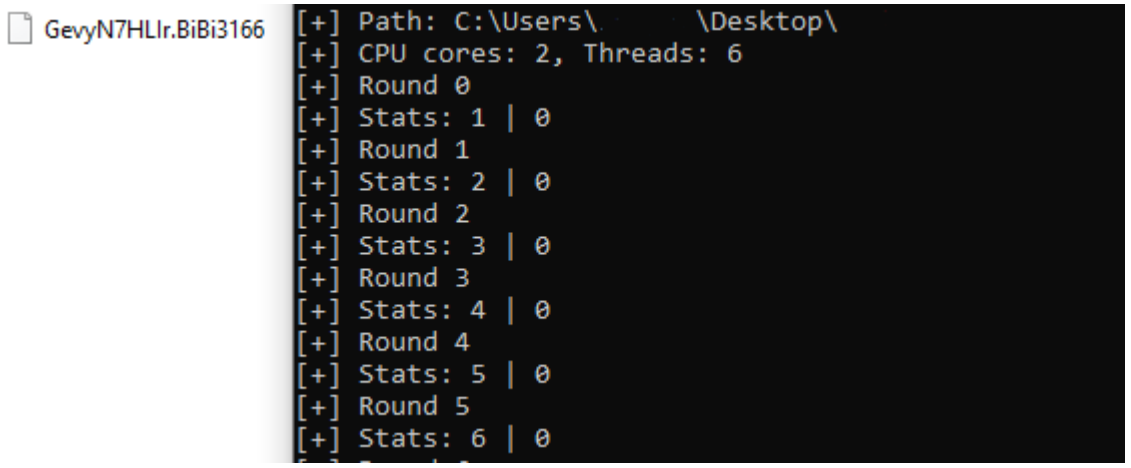
```
 rand_no = twisterfunc()
remainder_output = rand_no % (0x3D + 1)
wide_string = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
filename_byte = wide_string[remainder_output * 2]

Thus,
if random num = 0xC6D15D79
mod_out = 0xC6D15D79 % 0x3E => 0x31
offset = 0x31 × 0x2 = 0x62
so the 62nd indexed value in the wide string becomes one of the character for the filename. In this case the va
```

The same routine is been executed 0xA (10) times and all the indexed values are appended together forming the random file name eg. **1wnRvB6teT** and then the extension **".BiBi" (Bibi is a nickname used for Israel's Prime Minister, Benjamin Netanyahu)** is added along the **round number at the end** in the following manner - **<rand_filename>.BiBi**

Therefore in the following manner the BiBi-Windows Wiper would destroy all the files in the target directories by overwriting the data in the files with random bytes, now as the loop i.e the rounds are till infinity the Wiper will keep on overwriting the files multiple time recursively with random bytes till infinity and wont stop thus destroying the files on the machine completely!

BiBi-Windows Wiper execution showcasing the Target directory, CPU Cores, Threads, Round Number, Stats, and destroyed file with .BiBi extension



Thanks for reading! - knight0x07

Source: https://github.com/knight0x07/BiBi-Windows-Wiper-Analysis?tab=readme-ov-file