Mebromi: the first BIOS rootkit in the wild

webroot.com//blog/2011/09/13/mebromi-the-first-bios-rootkit-in-the-wild/

Blog Staff September 13, 2011

By Marco Giuliani

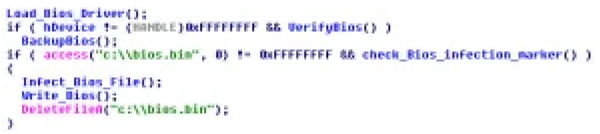
In the past few weeks a Chinese security company called **Qihoo 360** blogged about a new BIOS rootkit hitting Chinese computers. This turned to be a very interesting discovery as it appears to be the first real malware targeting system BIOS since a well-known proof of concept called **IceLord** in 2007. The malware is called **Mebromi** and contains a bit of everything: a BIOS rootkit specifically targeting Award BIOS, a MBR rootkit, a kernel mode rootkit, a PE file infector and a Trojan downloader. At this time, Mebromi is not designed to infect 64-bit operating system and it is not able to infect the system if run with limited privileges.

The infection starts with a small encrypted dropper that contains five crypted resource files: *hook.rom*, *flash.dll*, *cbrom.exe*, *my.sys*, *bios.sys*. The goal of these files will be presented later in this analysis.

The infection is clearly focused on Chinese users, because the dropper is carefully checking if the system it's going to infect is protected by Chinese security software **Rising Antivirus** and **Jiangmin KV Antivirus**. To gain access to the BIOS, the infection first needs to get loaded in kernel mode so that it can handle with physical memory instead of virtual memory.

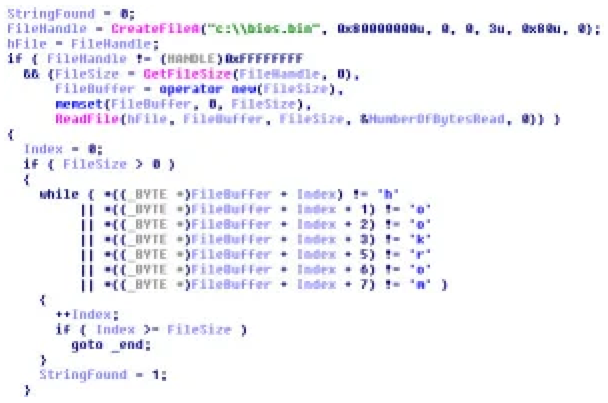
Many of you may recall the old **CIH/Chernobyl** infection, the infamous virus discovered in 1998 that was able to flash the motherboard BIOS, erasing it. Even CIH needed to gain kernel mode access to reach the BIOS, though at the time the virus was exploiting a privilege escalation bug in Windows 9x operating system which allowed it to overwrite the Interrupt Descriptor Table with its own payload from user mode, then triggering the overwritten interrupt handler and its malicious code is executed in kernel mode. Mebromi does not use such kind of privilege escalation trick anymore, it just needs to load its own kernel mode driver which will handle the BIOS infection. To do so, it uses two methods: it could either extract and load the flash.dll library which will load the bios.sys driver, or it stops the beep.sys service key, overwriting the beep.sys driver with its own bios.sys code, restart the service key and restore the original beep.sys code.

The bios.sys driver is the code which handle the BIOS infection. To read the BIOS code, it needs to map the physical memory located at physical memory address *0xF0000*, this is where the BIOS ROM usually resides. Once read, the driver verifies if the BIOS ROM is Award BIOS, by checking the presence of the string: *$@AWDFLA*. If found, the driver tries to locate the **SMI port** that will be used by the rootkit to flash the BIOS ROM.



```
Load_Bios_Driver();
if ( hDevice != (HANDLE)0xFFFFFFFF && VerifyBios() )
  BackupBios();
if ( access("c:\\bios.bin", 0) != 0xFFFFFFFF && check_Bios_infection_marker() )
{
  Infect_Bios_File();
  Write_Bios();
  DeleteFileA("c:\\bios.bin");
}
```

If the BIOS ROM matches the string, the rootkit saves a copy of the BIOS to the file C:bios.bin and pass the next step to the user mode component of the infection. The dropper extracts two files: cbrom.exe and hook.rom. Cbrom.exe is a legitimate tool developed by Phoenix Technologies, used to modify the Award/Phoenix BIOS ROM binaries. Hook.rom is the rootkit ISA BIOS ROM that is added to the BIOS binary, containing the rootkit infection. The dropper executes cbrom.exe with the /isa switch parameter, passing the hook.rom file. Before actually injecting the malicious ISA ROM, the dropper checks the BIOS ROM code looking for the "*hook rom*" string, used as a marker of the infection. If found, it means that the BIOS is already infected and it doesn't need to be infected again.

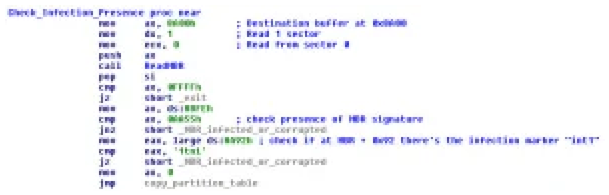


```
StringFound = 0;
FileHandle = CreateFileA("c:\\bios.bin", 0x80000000u, 0, 0, 3u, 0x80u, 0);
hFile = FileHandle;
if ( FileHandle != (HANDLE)0xFFFFFFFF
  && (FileSize = GetFileSize(FileHandle, 0),
      FileBuffer = operator new(FileSize),
      memset(FileBuffer, 0, FileSize),
      ReadFile(hFile, FileBuffer, FileSize, &NumberOfBytesRead, 0)) )
{
  Index = 0;
  if ( FileSize > 0 )
  {
    while ( *((_BYTE *)FileBuffer + Index) != 'h'
         || *((_BYTE *)FileBuffer + Index + 1) != 'o'
         || *((_BYTE *)FileBuffer + Index + 2) != 'o'
         || *((_BYTE *)FileBuffer + Index + 3) != 'k'
         || *((_BYTE *)FileBuffer + Index + 5) != 'r'
         || *((_BYTE *)FileBuffer + Index + 6) != 'o'
         || *((_BYTE *)FileBuffer + Index + 7) != 'm' )
    {
      ++Index;
      if ( Index >= FileSize )
        goto _end;
    }
    StringFound = 1;
  }
```

After that the bios.bin file has been modified, the bios.sys driver send to the BIOS SMI port the command *0x29*, used to erase the BIOS flash, and then the command *0x2F* used to write the new BIOS ROM code to the BIOS ROM.

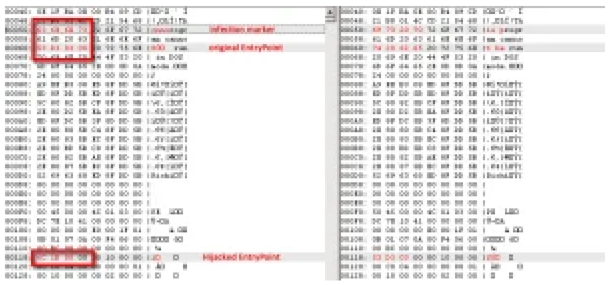
The BIOS is now infected, and the dropper goes to its next step: **infecting the Master Boot Record**. The infection is 14 sectors long and the original MBR is stored to the sector 7. To avoid potential startup issues, the infected MBR stores a copy of the original MBR's partition table. Finally the dropper extracts the my.sys driver on the root of the C: drive. My.sys is a kernel mode rootkit that hijacks disk.sys's IRP major functions, by redirecting the *IRP_MJ_READ/WRITE* and *IRP_MJ_DEVICE_CONTROL* native functions. It is used to hide the infection on the disk. Even if the BIOS infection doesn't succeed, the rootkit does infect the MBR.

At the next system startup, after the BIOS POST phase, the malicious code injected inside it prepares the full MBR infection (all the first 14 sectors are stored inside the malicious BIOS rom, 7168 bytes in total) and checks the MBR code of the hard drive looking if the infection is already present. To do it, the BIOS malicious code checks for the presence of the string "*int1*" at the offset 0x92. If the string is not found, the BIOS malicious rom will overwrite all the first 14 sectors of the hard drive, thus restoring the MBR infection.



```
Check_Infection_Presence proc near
                mov     ax, 0600h       ; Destination buffer at 0x0600
                mov     dx, 1           ; Read 1 sector
                mov     ecx, 0          ; Read from sector 0
                push    ax
                call    ReadMBR
                pop     si
                cmp     ax, 0FFFFh
                jz      short _exit
                mov     ax, ds:07FEh
                cmp     ax, 0AA55h      ; check presence of MBR signature
                jnz     short _MBR_infected_or_corrupted
                mov     eax, large ds:0092h ; check if at MBR + 0x92 there's the infection marker "intT"
                cmp     eax, 'Ttni'
                jz      short _MBR_infected_or_corrupted
                mov     ax, 0
                jmp     copy_partition_table
```

The system startup procedure continues and the control now passes to the malicious master boot record. Here the malicious payload analyzes the original MBR partition table and looks for the active partition, checking if it's using a NTFS or FAT32 file system. The malicious MBR code **contains indeed NTFS/FAT32 parser routines**, used to get inside the file system to look for *winlogon.exe* or *wininit.exe* file. When found, the malicious code contains a file infection payload, able to inject malicious code inside the specified file and hijack the entry point of it. Before infecting the file, the MBR malicious code checks if it is already infected, by looking for the string "*cnns*" at the offset 0x50 from the beginning of the PE file. This is the infection marker. If the string is not found, the infection stores a crypted payload – about 600 bytes of code – inside winlogon.exe or wininit.exe and hijacks the PE entry point to the beginning of it, saving the original entry point at the offset 0x60.



The job of the MBR infection ends here, waiting for the Windows startup which will load the patched executable. When loaded, the payload self-decrypt its malicious code and loads in memory the my.sys driver. Then it tries to download an additional infection from the (now unavailable) URL address: http://dh.3515.info:806/test/91/calc[removed].

The concept behind Mebromi is not new. In fact **we must recall the IceLord BIOS rootkit published in 2007**, a public proof of concept able to target Award BIOS rom, using an approach very similar to the Mebromi one – or should we say that Mebromi is more than just inspired by the IceLord rootkit?

Storing the malicious code inside the BIOS ROM could actually become more than just a problem for security software, giving the fact that even if an antivirus detect and clean the MBR infection, it will be restored at the next system startup when the malicious BIOS payload would overwrite the MBR code again. Developing an antivirus utility able to clean the BIOS code is a challenge, because it needs to be totally error-proof, to avoid rendering the system unbootable at all. The job of handling with such specific system codes should be left to the developers of the specific motherboard model, who release BIOS updates along with specific tool to update the BIOS code.

On the other hand, although this kind of infection is potentially one of the most persistent infections known out there in the wild, it will hardly become a major threat because of the level of complexity needed to achieve the goal. While a kernel mode infection or a MBR infection could still work in a generic way among all the PC out there – and they still have a huge available free space to play with, a BIOS-based rootkit needs to be fully compatible with all major BIOS rom out there, it should be able to infect all the different releases of Award, Phoenix, AMI BIOS's out there; **a level of complexity that is simply unasked for writing a good persistent infection** (e.g. TDL rootkit, various Rustock releases, ZeroAccess rootkit among all). In fact, why is Mebromi only targetting Award BIOS rom? Perhaps because there was already a known proof of concept that is 5 years old targeting Award BIOS ROM available online.

Are BIOS rootkits a real threat? Yes, we can consider Mebromi the first real BIOS rootkit incident discovered in the wild – let's consider IceLord BIOS rootkit more a proof of concept. Should we be concerned about BIOS rootkits? Well, while we try to discover whether our PC is infected by an unknown and super-stealth BIOS rootkit, let's try and look if there is a more "humble" kernel mode rootkit which is already infecting our PC, allowing a remote attacker to silently own our system.

About the Author

Blog Staff

The Webroot blog offers expert insights and analysis into the latest cybersecurity trends. Whether you're a home or business user, we're dedicated to giving you the awareness and knowledge needed to stay ahead of today's cyber threats.