VIRUS ANALYSIS

TO CATCH EFISH

Peter Ferrie and Frédéric Perriot
Symantec Security Response, USA

W32/Efish, a member of the W32/Chiton family, contains in its source code (released as part of *29A* magazine) a reference to the television program *The Six Million Dollar Man*. The virus author wanted to call the virus EfishNC ("efficiency"), and referred to it as "Better, Stronger, Faster" (this virus author is not known for humility – in 1994 (s)he named a virus Hianmyt ["high and mighty"]). While the code is indeed better, stronger and faster than comparable viruses, it does have weaknesses. *Symantec* has not received any wild samples of Efish, although the .A variant was published as early as 2002. This suggests that these viruses have not left zoo collections, despite their aggressive infection strategy.

STINGRAY

The infection cycle of Efish starts with an infected program dropping a standalone, unencrypted virus sample to the *Windows* directory, and directing registry hooks to this file. This standalone file, which exists independently of any host program, then infects hosts in a parasitic way. There is no 'direct' infection from one host file to another.

It is worth mentioning that the standalone virus sample is an extremely tortuous, albeit valid, PE file. The PE structure of the file is an abomination of overlapping headers and tables, crafted for the sake of size optimization, which – surprisingly – loads without problem on 32-bit *Windows* platforms, from *Windows 95* to *Windows 2003*. Needless to say, most tools of the trade from *PEDUMP* to *Soft-ICE* have trouble mapping the file properly, and we expect that some anti-virus products would be confused as well.

UNICORN FISH

As with all other known members of the W32/Chiton family, Efish is fully *Unicode*-aware, and selects dynamically between ANSI and *Unicode* routines, as appropriate. These routines include command-line parsing, local network and IP share enumeration, and file enumeration. In some cases, a single routine is capable of processing either type of character sets, by simply changing an AND mask and using the CharNext() API. This is one of the many code optimizations that result in such a small amount of code (around 4kB) that is capable of so much.

There are a number of interesting optimizations in the code. The one that appears most often (and which is the hardest to follow) is the long series of PUSH instructions prior to a series of API calls. The purpose of this seems to be to avoid the reinitialization between API calls of the volatile registers, such as ECX and EDX. One particular example is the code for dropping the standalone virus file, which contains 23 PUSH instructions followed by seven API calls:

```
PUSH    EAX             ;GlobalFree
PUSH    EBP             ;WriteFile
PUSH    ESP             ;WriteFile
PUSH    EDI             ;WriteFile
PUSH    EBP             ;CreateFileA
PUSH    +02             ;CreateFileA
PUSH    +02             ;CreateFileA
PUSH    EBP             ;CreateFileA
PUSH    EBP             ;CreateFileA
PUSH    40000000        ;CreateFileA
PUSH    EAX             ;CreateFileA
LEA     ECX, DWORD PTR [EAX + 7F]
PUSH    ECX             ;MoveFileA
PUSH    EAX             ;MoveFileA
PUSH    EAX             ;GetFileAttributesA
PUSH    EBP             ;SetFileAttributesA
PUSH    EAX             ;SetFileAttributesA
PUSH    ECX             ;DeleteFileA
PUSH    ECX             ;GetTempFileNameA
PUSH    EBP             ;GetTempFileNameA
PUSH    ESP             ;GetTempFileNameA
PUSH    EAX             ;GetTempFileNameA
PUSH    EDI             ;GetWindowsDirectoryA
PUSH    EAX             ;GetWindowsDirectoryA
XCHG    EBP, EAX
CALL    GetWindowsDirectoryA
LEA     EDI, DWORD PTR [EAX + EBP - 01]
CALL    GetTempFileNameA
CALL    DeleteFileA
...
CALL    SetFileAttributesA
CALL    GetFileAttributesA
...
CALL    MoveFileA
CALL    CreateFileA
```

Figure 1. The code for dropping the standalone virus file.

FISH TANKS

Efish is very aggressive when it comes to finding targets. The target selection is contained in three threads.

The first thread periodically enumerates all drive letters, from A: to Z:, looking for fixed and network drives. The second thread periodically enumerates all network shares on the local network, looking for drive resources. The third thread periodically enumerates all network shares on random IP addresses. In each case the virus examines all files in all subdirectories.

BALEENS

Efish examines all files for their potential to be infected, regardless of their extension. First the virus checks if the file is protected by the System File Checker. While the main

file responsible for the protection (sfc.dll) exists in all *Windows* versions that support SFC, the required function is forwarded in *Windows XP/2003* to a file called sfc_os.dll. The method that Efish uses to retrieve the address of exported APIs does not support export forwarding, so the virus resolves the APIs directly from sfc_os.dll on platforms where this .dll exists.

Unprotected files are then checked against a very strict set of filters, which includes the condition that the file being examined must be a character mode or GUI application for the *Intel 386+* CPU, that the file must have no digital certificates, and that it must have no bytes outside of the image. The latter condition is the virus's infection marker.

Additionally, the file must satisfy the needs of the EntryPoint-Obscuring technique (see below).

PILOTFISH

The EPO method that Efish uses is to replace a function prologue with its own code. This method has previously been used by such viruses as Zhengxi (on the DOS platform) and W95/SK (on *Windows*). The virus searches the first section of the file for function prologue code that creates a stack frame, and epilogue code that destroys it. The virus requires that the prologue and epilogue be at least 32 bytes apart, in order for the decryptor to fit.

While it might appear that only the first such sequence is used, this is not always the case. Sometimes a later sequence may be used, or the EPO routine may fail to find a proper sequence even though one exists in the file. This is most likely a coding bug, but it could have been intentional.

FEABUL ENDJINN

Once such a sequence has been found, Efish saves the first 32 bytes of that code, and replaces them with an oligomorphic decryptor. The useful code of the decryptor is 27 to 32 bytes in length, and it is padded up to 32 bytes with ff bytes (an artifact from the memory reuse). The Efish.A engine comprises only about one eighth of the virus body, yet it combines line-swapping, variable load and store instructions and decryption in either a forwards or backwards direction. According to our calculations, there are 23,616 possible valid decryptors and a few invalid ones!

The engine shared by the .B and .C variants adds register replacement, the optional use of 'do-nothing' instructions in the form of INC and DEC of unused registers, and one-byte instructions CMC, STC, and CLD.

The decryptor decrypts the virus body onto the stack and runs it from there. This requires no changes to the attributes of the section in which the virus body is placed within the file, an effective anti-heuristic attack.

DfishNC

A thorough analysis of the engine reveals a lack of optimization in several code sequences and at least two bugs. The result of the first bug is that the PUSH EDI instruction cannot be produced to transfer control to the virus code. However, the code was optimized and that bug was fixed in the .B variant.

The second bug, present in all three variants, causes Efish to produce non-working decryptors in a few rare cases, leading to corrupted replicants. Detection methods based on emulation of the decryptor to recover the virus body are bound to miss such corrupted samples.

BLOWFISH

The decryption is performed using a translate (XLAT) table, in which each unique byte of the virus code is replaced by a unique random value.

The virus author claimed that it is unbreakable, which is clearly untrue, since it is simply a substitution cipher. As we show in our VB2004 conference paper 'Principles and practise of x-raying', several methods exist to break the Efish encryption, and they work quite quickly in practice.

In the .C variant, the author of Efish refined the encryption method a little by taking into account unused byte values from the virus body and reusing slots in the translate table (switching to what is known as a 'homophonic substitution cipher'). Fortunately, efficient attacks still exist against this cipher and, in particular, against the somewhat simplistic implementation in Efish.C. Once again, we refer readers to our paper on x-raying for a thorough explanation.

Efish places its body into the last section of the file, along with the XLAT key table, however it prepends and appends garbage bytes randomly to both the body and the key table, to disguise its true location, and it randomly alters the order in which they are added to the file. If relocation data exist at the end of the file, then the virus moves the data to a larger offset in the file, and places its body and table in the gap that has been created. If there are no relocation data at the end of the file, the virus body and table are placed here (see Figure 2).

STONEFISH

The convoluted code of the virus makes it easy for analysts to overlook one fundamental feature of Efish: the .A and .B variants are 'slow polymorphic' viruses. This term means

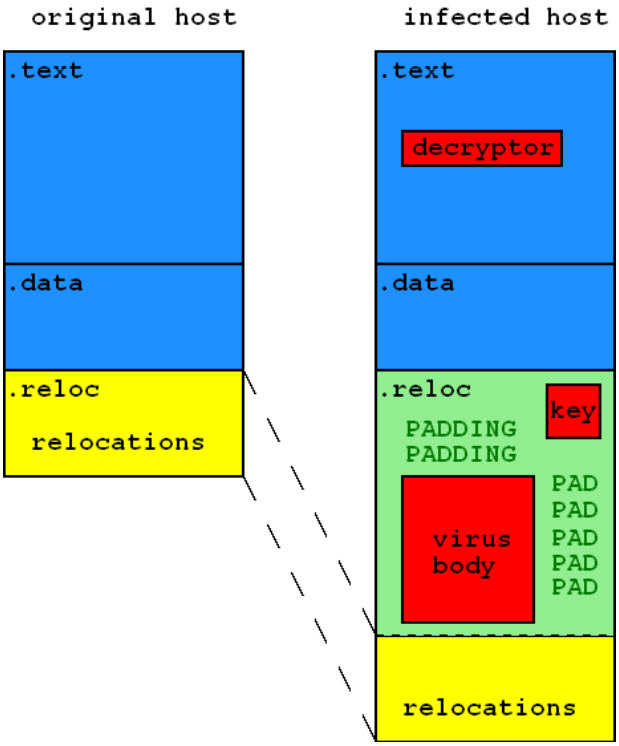

Figure 2.

that the polymorphic decryptor is generated only once in a while, and the same copy is used in the infection of several host programs. In the case of Efish, the decryptor is generated when the standalone sample first runs, before it starts looking for hosts to infect. Additionally, the decryption key, encrypted virus body and layout of the virus segment containing the key, body and random padding, are also generated anew only when the standalone sample starts.

Therefore, all detection methods, whether based on decryptor parsing, emulation, or x-raying of the virus body, must be tested carefully against a range of samples generated from several runs of the virus.

So long, and thanks for all the …

W32/Chiton variant

Type:	Memory-resident parasitic appender/inserter, share crawler, slow polymorph.
Infects:	*Windows* Portable Executable files.
Payload:	None.
Removal:	Delete infected files and restore them from backup. Restore registry.