## VIRUS ANALYSIS

# TIME MACHINE

*Peter Ferrie*
Symantec Security Response, USA

It is commonly reported that the first known full stealth file-infecting virus was Frodo, in 1989. In fact, that is true only for the *IBM PC* world. The *Commodore 64* world had been infected three years earlier by what was perhaps *truly* the first full stealth file-infecting virus: C64/BHP.A (not to be confused with the boot-sector virus for the *Atari*, also known as BHP).

All of the descriptions of BHP that were published at the time were inaccurate, some of them even giving incorrect descriptions of how the infection worked. This article takes a look at what it really did.

### BASIC INSTINCT

As with all *Commodore 64* programs, BHP began with some code written in Basic. This code consisted of a single line, a SYStem [*sic*] call to the assembler code, where the rest of the virus resided. Unlike many programs, the virus code built the address to call dynamically. This may have been written by a very careful coder, but it proved to be unnecessary because the address did not change in later versions of the machine.

Once the assembler code gained control, it placed itself in the block of memory that was normally occupied by the I/O devices when the ROM was banked-in.

At this point, it is necessary to describe some of the *Commodore 64* architecture in more detail.

### DOWN MEMORY LANE

The *Commodore 64* used a MOS 6510 CPU, a later version of the MOS 6502 chip used by several competing machines of the time, including the *Apple II*-series and the *Atari 400* and *800*.

Since the 6502's data bus (and therefore the 6510's data bus) was only 16 bits wide, the maximum directly addressable memory range was 64kb. In order to accommodate more memory, a 'banking' architecture was implemented, allowing different memory regions to be mapped in under the user's control, simply by writing the appropriate value to a specific memory-mapped port.

### NOW YOU SEE ME ...

The *Commodore 64* allowed quite a large address space in comparison with other machines at that time: potentially eight banks of 64kb (a total of 512kb!) of memory were available, though most machines did not have the chips installed to provide that much.

Since the mapped regions all needed to be within the 64kb range, a few memory ranges provided the base for all of the banked memory, in order to give the maximum amount of memory that would always be available. This greatly reduced the complexity of the average program.

On the other hand, however, several steps were required for a program that ran in one memory bank to access data in another memory bank. The first step was to place code in non-banked memory and run it. The next steps were for that code to bank out the program, bank in the required data, access that data and save them, then bank out the data, bank in the program again, restore the data, and return control to the program.

### ... NOW YOU DON'T

A side-effect of memory-banking was that it was a great way to hide a program, since the program was not visible if its memory was not banked in. This is the reason why BHP placed its code in banked memory.

After copying itself to banked memory, the virus restored the host program to its original memory location and restored the program size to its original value. This allowed the host program to execute as though it were not infected. However, at this time the virus would verify the checksum of the virus's Basic code, and would overwrite the host memory if the checksum did not match.

An interesting note about the checksum routine is that it missed the first three bytes of the code, which were the line number and SYS command. This made the job easier for the person who produced the later variant of the virus. Although the later variant differed only in the line number, this was sufficient to defeat the BHP-Killer program, because BHP-Killer checked the entire Basic code, including the line number.

### CAPTAIN HOOK

The virus checked whether it was running already by reading a byte from a specific memory location. If that value matched the expected value, the virus assumed that another copy was running. Thus, writing that value to that memory location would have been an effective inoculation method. Similar methods were used against viruses for the *Commodore Amiga* machines.

If no other copy of the virus was running, the virus would copy some code into a low address in non-banked memory, and hook several vectors, pointing them to the copied code.

The virus hooked the ILOAD, ISAVE, MAIN, NMI, CBINV and RESET vectors.

The hooking of MAIN, NMI, CBINV and RESET made the virus Break-proof, Reset-proof, and Run/Stop-Restore-proof. These hooks ensured that the virus did not lose control while the machine restarted. This technique was similar to the Ctrl-Alt-Delete hooks that were used later in DOS viruses on the *IBM* PCs, or the Ctrl-Amiga-Amiga hooks that viruses used on the *Commodore Amiga*.

Once the hooks were in place, the virus ran the host code. The main virus code would be called on every request to load or save a file.

## HEAVY LOAD

The ILOAD hook was reached when a disk needed to be searched. This happened whenever a directory listing was requested, and could happen when a search was made using a filename with wildcards, or the first time that a file was accessed. Otherwise, the drive hardware cached up to 2kb of data and returned it directly.

The virus called the original ILOAD handler, then checked whether an infected program had been loaded. If an infected program had been loaded, the virus restored the host program to its original memory location and restored the program size to its original value. Otherwise, even if no file had been loaded, the virus called the infection routine.

## DON'T FORGET TO SAVE

The ISAVE hook was reached whenever a file was saved. The virus called the original ISAVE handler to save the file, then called the infection routine.

The infection routine began by checking that the requested device was a disk drive. If so, then the virus opened the first file in the cache. The first file in the cache would be the saved file if this code was reached via the ISAVE hook, otherwise it would be the first file in the directory listing.

If the file was a Basic program, then the virus performed a quick infection check by reading the first byte of the program and comparing it against the SYS command. The virus read only one byte initially, because disk drives were serial devices on the *Commodore 64*, and therefore very slow. However, if the SYS command was present, the virus verified the infection by reading and comparing up to 27 subsequent bytes. A file was considered infected if all 27 bytes matched.

If the file was not infected, the virus switched to reading data from the hardware cache. The first check was for a standard disk layout: the directory had to exist on track 18, sector 0, and the file to infect had not to have resided on that track.

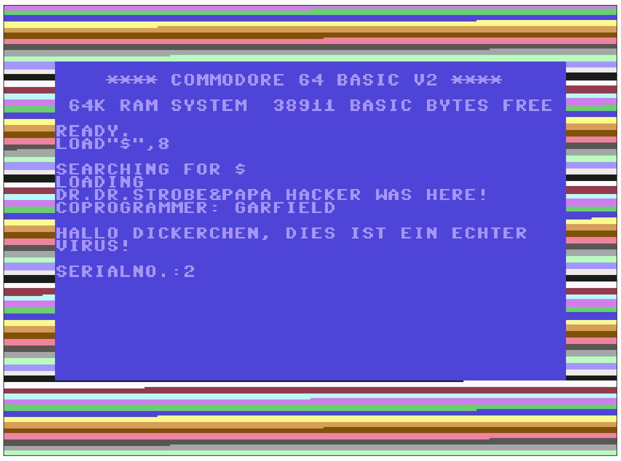
## LESS TALK, MORE ACTION

If these checks passed, the virus searched the track list for free sectors. It began with the track containing the file to infect, then moved outwards in alternating directions. This reduced the amount of seeking that the drive had to perform in order to read the file afterwards, and was a very interesting optimisation, given that some multi-sector boot viruses on the *IBM PC* placed their additional code at the end of the disk, leading to very obvious (read: audible) seeking by the drive.

If at least eight free sectors existed on the same track, then the virus allocated eight sectors for itself and updated the sector bitmap for that track. The code to update the sector bitmap was beautiful, allocating the sectors and creating the list of sector numbers at the same time. The code could have been shortened slightly, though, by reordering some of the instructions.

This was the case throughout the virus – overall, the code was very tight (as it needed to be), but there were some pieces of code that could have been optimised in very obvious (and some less obvious) ways. There were also a couple of harmless bugs. However, given the size of the code, the only resulting advantage would have been that the payload (see below) could have contained a longer message or more effects.

By comparison, the code used to write the virus to the disk was a horrible mess – suggesting, perhaps, that it was written by a co-author. The virus wrote itself to disk in the following manner: the first sector of the host was copied to the last sector allocated by the virus, then that first sector



```
    **** COMMODORE 64 BASIC V2 ****

 64K RAM SYSTEM  38911 BASIC BYTES FREE

READY.
LOAD"$",8

SEARCHING FOR $
LOADING
DR.DR.STROBE&PAPA HACKER WAS HERE!
COPROGRAMMER: GARFIELD

HALLO DICKERCHEN, DIES IST EIN ECHTER
VIRUS!

SERIALNO.:2
```

*Figure 1. BHP's payload. The text was displayed one character at a time, while the colours of the border cycled.*

was replaced by the first sector of the virus. After that, the remaining virus code was written to the remaining allocated sectors.

The directory stealth was present here, and it existed without any effort on the part of the virus writer(s). It was a side-effect of the virus not updating the block count in the directory sector. The block count was not used by DOS to load files, its purpose was informational only, since it was displayed by the directory listing.

In fact, the same problem existed on DOS for the *Apple II* series of machines and such a virus would have been much easier to write there, since communication with the hardware is much simpler on those machines. The only obvious effect in the case of BHP was that the number of free blocks on the disk was visibly reduced, because the value was calculated using the sector bitmap, not the directory listing.

## PAYLOAD

After any call to ILOAD or ISAVE, the virus checked whether the payload should activate. The conditions for the payload activation were the following: that the machine was operating in 'direct' mode (the command-prompt), that the seconds field of the jiffy clock was a value from 2–4 seconds, and that the current scan line of the vertical retrace was at least 128. This made the activation fairly random. The payload was to display a particular text, one character at a time, while cycling the colours of the border (see Figure 1).

The serial number that was displayed was the number of times the payload check was called. It was incremented once after each call, and it was carried in replications. It reset to zero only after 65,536 calls.

## CONCLUSION

So now we know: BHP was a virus ahead of its time.

### C64/BHP.A

| | |
|---|---|
| Size: | 2030 bytes. |
| Type: | Memory-resident parasitic prepender. |
| Infects: | *Commodore 64* Basic files. |
| Payload: | Displays text under certain conditions. |
| Removal: | Delete infected files and restore them from backup. |