VIRUS ANALYSIS 1

64-BIT RUGRATS

Peter Ferrie and Péter Ször
Symantec Security Response, USA

On 26 May 2004, we received the first known virus for the 64-bit *Windows* operating system on the *Intel Itanium* platform. We decided to call it W64/Rugrat.3344.A.

Just like some of its predecessors (specifically W32/Chiton – see *VB*, June 2002, p.4), Rugrat is aware of Thread Local Storage, helping it to make the first successful tip toe towards painless infection of *Windows* DLLs – at least in the .B variant of the virus.

BLAST FROM THE PAST

As might be expected, the text in the virus body suggests that Rugrat and Chiton share the same author: 'Shrug - roy g biv', who is now a member of the notorious 29A virus-writing group.

W64/Rugrat uses a fairly simple idea: take the 32-bit code and port it to 64 bits – but the devil is in the detail.

Writing assembly for the *Itanium* is not simply a case of an everyday port of a 32-bit C application to 64-bit, which even your grandmother could do with a little advice. This is in contrast to the impression we were given when *Microsoft* introduced the platform with a demonstration: "Here is Larry who ported a million lines of C code in two weeks!".

Obviously Larry was not the kind who used to cast his pointers with DWORDs in front. Larry probably did not need to port a GUI either, and he obviously was not interested in writing a memory scanner to scan the 64-bit address space for virus code. Larry needed just one thing: earplugs to reduce the ventilation noise coming out of the strange box that he first mistook for an atomic reactor. But the earplugs were disposed of a long time ago, along with the beta boards, and nowadays it is the beautiful *Itanium2* that resides under Larry's desk.

While Larry did not care to open the guide for the *Itanium* instruction set, 'roy g biv' did. One question comes to mind: might 'roy g biv' have owned an *IA64* box? He probably did, but he could equally have used an emulator on a bulked up PC.

Intel's IA64 assembly code is designed for explicit parallelism, so coding well in *IA64* assembly code requires the ability to 'think in parallel'. Unfortunately, when this is done well, the resulting code can be very difficult to read, especially when the virus code is further obfuscated. So, in turn, we started to think in parallel to share the fun of the analysis of this new kid on the block.

ROUND AND ROUND

The first time Rugrat is executed, it checks the event that caused its execution. The virus replicates only during the DLL_PROCESS_DETACH event, which occurs when an application is exiting. The reason that Rugrat does this could be because an application taking an extended period of time to terminate is far less noticeable than an application taking an extended period of time to start.

During Thread Local Storage events, it is NTDLL.DLL (the NATIVE API) that calls the Thread Local Storage entry point. That call leaves in the B0 register of *Itanium* processors a pointer into the NTDLL.DLL address space. The virus uses this fact to gain access to NTDLL.DLL, in order to retrieve the addresses of the API functions that it requires.

The virus uses a CRC method to match the API names. The use of the CRC method means that the API names are not visible in the virus code, while also reducing the size of the virus code.

The virus uses a few Win64 APIs from three different libraries: NTDLL.DLL, SFC_OS.DLL and KERNEL32.DLL. From NTDLL.DLL it picks LdrGetDllHandle(), RtlAddVectoredExceptionHandler() and RtlRemoveVectoredExceptionHandler(). The virus supports vectored exception handling to avoid crashing during infections. The use of LdrGetDllHandle() makes it simpler to gain access to other modules. From SFC_OS.DLL, Rugrat uses the SfcIsFileProtected() function to avoid infecting executables that are protected by the System File Checker (SFC).

The following 16 functions are used from KERNEL32.DLL to implement a standard direct action file infection of an *IA64* Portable Executable image using file mapping:

CreateFileMappingA()	GlobalAlloc()
CreateFileW()	GlobalFree()
CloseHandle()	LoadLibraryA()
FindFirstFileW()	MapViewOfFile()
FindNextFileW	SetCurrentDirectoryW()
FindClose()	SetFileAttributesW()
GetFullPathNameW()	SetFileTime()
GetTickCount()	UnmapViewOfFile()

As expected the virus will set the host's time/date stamp back after each infection, as well as its attributes, which it clears before infection.

Rugrat shows one major functional difference from Chiton – Rugrat uses only Unicode functions, and does not support ANSI functions, perhaps because 64-bit *Windows* is based on *Windows NT*, which is entirely Unicode under the hood.

The virus searches for files in the current directory and all subdirectories, using a linked list instead of a recursive function. This is important from the point of view of the virus author, because the .B variant of Rugrat infects DLLs, whose stack size can be very small.

FILTERS

Files are examined for their potential to be infected, regardless of their suffix, and will be infected if they pass a very strict set of filters. The first of these filters is the support for the System File Checker that exists in *Windows XP/2003 (Note the SFC_OS module name)*. The remaining filters include the condition that the file being examined must be a character mode or GUI application for the *Intel IA64* CPU, that the file must have no digital certificates, and that it must have no bytes outside of the image.

The *IA64* CPU introduces the concept of 'predication' to the execution flow, which allows a programmer to remove certain branches from the code, and to replace a block with a predicate check instead. A sample check for the file type might look like this:

```
     ld2 r30 = [r32]
     mov r31 = 0x5A4D;;
     cmp.eq p1 = r30, r31
(p1) ld4 r30 = [r8]
(p1) mov r31 = 0x4550;;
(p1) cmp.eq p1 = r30, r31
```

The first 'cmp' instruction sets the P1 register only if the condition is met. If it is not met, any instruction that is predicated by the P1 register will be ignored by the processor. The filtering code could be considered 'predication abuse', since it contains more than 30 predicated instructions in a row, including predicated compares, which reuse active predicate registers. The infection code contains several such blocks.

TOUCH AND GO

When a file that meets the infection criteria is found, it will be infected. If relocation data exist at the end of the file, the virus will move the data to a larger offset in the file, and place its code in the gap that has been created. If there are no relocation data at the end of the file, the virus code will be placed here. For the .A variant of Rugrat, which does not infect DLLs, the relocation data check is almost never used, since the majority of executable files do not contain relocation data (a 'global pointer' is used instead, see below). The .B variant of Rugrat infects DLLs in the same way as for applications, meaning that even 'resource-only' DLLs that have no main entry point can still be a source of infection, since the Thread Local Storage entry point will still be called.

The virus carries its own Thread Local Storage directory, which will be used if the target file contains no directory at all. The virus carries its own callback array for those hosts whose Thread Local Storage directory contains no callbacks.

When it encounters a host that already has a Thread Local Storage directory containing callbacks, the virus will save the address of the first callback and replace it with the address of the virus code.

Once the infection is complete, the virus will calculate a new file checksum, if one existed previously, before continuing to search for more files.

When the file searching has finished, the virus will allow the application to exit by forcing an exception to occur. This technique appears twice in the virus code, and is an elegant way to reduce the code size, in addition to functioning as an effective anti-debugging method.

Since the virus has protected itself against errors by installing a Vectored Exception Handler, the simulation of an error condition results in the execution of a common block of code to exit a routine. This avoids the need for separate handlers for successful and unsuccessful code completion. The Vectored Exception Handling is a heap-based dynamic exception-handling mechanism of newer *Windows* releases, which provides an alternative to the Structured Exception Handling, a stack-based mechanism. SEH is more vulnerable to exploitation (see *VB*, September 2001, p.4) than VEH, and VEH has some other benefits such as up-front exception control. Nonetheless, the exploitation of VEH by attackers is also becoming common in the field.

EXCEPTION TO THE RULE

The cause of the exception is more subtle in Rugrat than in Chiton. On the x86 CPU, exception-causing instructions such as INT 3 can appear anywhere in the code, without restriction. On the *IA64* CPU, though, instructions are placed in 'slots', and collected in 'bundles' that execute in parallel, so an exception-causing instruction in a bundle that contains other instructions could cause those instructions to be interrupted. Additionally, when resuming from an exception handler, execution continues from the slot in which the exception occurs, which results in the instructions in the earlier slots of the same bundle not being executed.

These two restrictions are the likely reason why the virus author chose an instruction that is always placed in the first slot of a bundle. The instruction itself, LD1 R8 = [R0], also looks legitimate, until it is understood that the R0 register always contains the value 0, and attempting to access the 0th byte of memory always causes an exception.

GLOBALISATION

The *IA64* uses a global pointer to access variables. This avoids the costly application of relocation items when a file is loaded to an address other than its default virtual address. This also makes the *IA64* code a little more compact, although it still appears large to eight-bit eyes.

The global pointer value can be retrieved from the Portable Executable GlobalPtr header field. Every structure – exported function addresses, Vectored Exception Handlers, even the Thread Local Storage Callback itself – is required to be in the form of a virtual address followed by a global pointer value, and the virus supplies these structures correctly. This structure is called a PLABEL_DESCRIPTOR, and the compiler and the debugger do a perfect job of hiding it, so guys like Larry need not worry about it.

CONCLUSION

The adoption of *IA64* machines appears to be slow. The lack of *IA64* machines restricts the scope of any potential outbreak, and this virus was simply a proof that it was possible. What can we expect next? A 32-bit and 64-bit cross-infector seems obvious.

Additionally, the news of *AMD64* is spreading, and systems are sold for as little as $700, "introducing the only Windows-compatible 64-bit processor and the smooth transition to 64 bits". *Microsoft* will be ready with the *AMD64* release by the end of this year. How much smoother could the transition be for the virus writers?

> *Chuckie: "So, we got a baby now."*
>
> *Lillian DeVille: "I wished we'd a talked about it first. I don't know if I'm ready."*
>
> *Rugrats*, Klasky Csupo Inc.

Let's hope that this baby doesn't grow up.

W64/Rugrat.3344!IA64

Type:	Direct-action parasitic appender/inserter.
Infects:	*Windows IA64* PE files.
Payload:	None.
Removal:	Delete infected files and restore them from backup.