VIRUS ANALYSIS

Holding the Bady

Costin Raiu
Kaspersky Lab, Romania

After working for over four years with macro and script viruses, I recently came across a piece of malware which gave me cause to dig out my old toolbox and blow the dust off my old disassemblers and debugging tools.

Although the last time I dug out my old toolbox was actually not such a long time ago (that occasion was due to another curious piece of binary data – the executable from the macro virus Class.EZ), this time the reason was not only a little different, but proved to be much, much trickier, and harder to figure out in its deeper internals.

The Bug

On 18 June 2001, *Microsoft* released its 33rd Security Bulletin for this year, dealing with a simple buffer overflow in one of the DLLs used by the indexing service 'idq.dll'. Credited to the people from *eEye Digital Security*, the bug proves, once again, that *Windows NT* and the server software running on *NT* systems are not spared by the most common security vulnerabilities of Unix systems, the buffer overflows.

The original security advisory from *eEye* did not include an exploit, but it wasn't long before a couple were written and started to crawl around. One of them was even posted to the *SecurityFocus* Web site, in the exploits section for this specific vulnerability, therefore becoming generally available to the masses.

However, by far the most interesting exploit came in the form of a computer worm, which not only exploits the vulnerability, but replicates the exploit further, to other servers from the Internet. Initially, the worm was named 'Code Red', but the common name selected by the AV industry for this worm is 'Bady', either in the form of 'Win32/Bady.worm' or the more complex, but even more CARO-compliant name, 'worm://Win32/Bady.A'.

The Worm

The worm code is written in Win32 Intel assembler, and is 3569 bytes long, if we count the data used and carried by the worm along with the executable code.

Due to the nature of this exploit, what was probably one of the trickiest parts was to transfer control to the worm code from the instructions that receive control after they smash the stack. In fact, this is so tricky, that it can work only under very specific conditions, thus limiting the possibility for the worm to spread. Also, it was so tricky that it took

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form without the prior written permission of the publishers.

```
"%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801
%u9090%u9090%u8190%u00c3%u0003%u8b00%u531b%u53ff%u0078%u0000%u00"
```

Figure 1.

four hours before I realized why the worm simply crashed my test system, and didn't want to work.

Basically, the worm sends 224 'N' (0x43) bytes via an HTTP GET request, and it appends a few bytes of executable code after them. The small piece of executable code is encoded in the URL it sends for processing to the ISAPI server extensions, and looks like that shown in Figure 1 above.

The buffer overflow will fill the stack with the 224 'N' bytes expanded to two-byte UNICODE representations of the form {0x4e, 0x0}, which are used as return address when the subroutine in which the buffer overflow took place returns.

After that, the execution flow will hopefully hit one of three eight-byte-long sequences designed to prepare (again!) the stack for another jump, which is designed to hit the real worm code. The jump is performed in quite a tricky manner, and it relies on the fact that at a certain address in memory we can find a specific instruction, a two-byte-long 'call ebx'.

However, the respective instruction, which is supposed to be located in the memory image of the standard system module 'msvcrt.dll' (Microsoft Visual C Runtime Library) at offset 7801CBD3h, is in its place only if the respective library is version 6.10.8637 – exactly the one distributed with *Windows 2000*, Service Pack 0, which is exactly 295000 bytes long.

So, if either SP1 or SP2 has been installed on the machine, the worm will be unable to spread. The same is true if the machine is running *Windows NT 4.0*, and in all these cases, the WWW Publishing Service of IIS will simply crash when attacked.

However, if the system runs the 'good' version of 'msvcrt.dll', the worm performs the jump correctly, and reaches its main code, which begins to take the steps necessary for the worm code to carry the infection further.

First, it will allocate stack space to store 134 (86h) DWORDs, and it will also take care to wipe it using 0CCh bytes. Next, the worm tries to obtain the location of the very useful API GetProcAddress, using a method which is actually very common to most PE infectors. For this, the worm scans the memory range 77E00000h–7800000h, incrementing in steps of 64K, looking for a 'MZ' signature. Obviously, this check attempts to find the memory image of 'kernel32.dll' (which, for example, is found at offset 77e80000h in the initial release of *Windows 2000*).

If the worm does not find the 'MZ' signature of 'kernel32.dll' in that range, it will attempt to look for the same thing starting from 0BFF00000h, obviously assuming that maybe the system is not *NT*, but Win9X (for example, in Win98 the 'kernel32.dll' module is located at the address: 0BFF70000h).

Check and Cross Check

After finding the possible address of the 'kernel32.dll' PE image in memory, the worm will perform a couple of additional checks to be certain that it is indeed the 'kernel32.dll' module. For this, it will check that it is a PE file and then find the export table to check if the module name matches 'KERNEL32'.

If the respective checks fail, the worm code continues scanning. It's amusing to note how careful the author was here to find the correct address of the kernel module image in memory while, a few instructions ago, it simply assumed that 'msvcrt.dll' contains a {0FFh, 0D3h} sequence (call ebx) at 7801CBD3h. I think this was due to the author using the respective code from some PE virus, and he/she didn't bother to remove the Win9X part. Also it seems useless to perform such careful checks for the 'kernel32.dll' module, when the earlier assumption regarding 'msvcrt.dll' has already been made.

After finding the correct address in memory of 'kernel32.dll', a short subroutine is called to determine the offset of the 'GetProcAddress' exported entry. This subroutine will simply parse the export table, and verify if any of the entries is indeed 'GetProcAddress'.

Next, 'GetProcAddress' will be used to obtain the address of other common APIs, which are LoadLibraryA, GetSystemTime, CreateThread, CreateFileA, Sleep, GetSystemDefaultLangID and VirtualProtect. Of these, LoadLibraryA will further be used to load and obtain the memory offsets of the images of 'infocomm.dll', 'WS2_32.dll' and 'w3svc.dll'. The worm then extracts the TcpSockSend subroutine address in 'infocomm.dll', as well for the addresses of the 'socket', 'connect', 'send', 'recv' and 'closesocket' subroutines in 'WS2_32.dll'.

Replication and Payload

Next, the worm spawns 100 threads in memory which are designed to carry the main replication code as well as the payload. However, due to a bug, the worm will try to spawn even more threads for each thread created, therefore quickly eating a huge amount of resources, meaning it is less likely to go unnoticed on an infected server.

Each thread runs exactly the same code, which acts as follows: first, the worm attempts to open a file named 'c:\notworm'. If successful, the worm will start to issue

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form without the prior written permission of the publishers.

'Sleep' calls of about 24 days, ad infinitum. However, if the respective file is not found on disk, the worm continues in its progress.

It will check whether the current day is between 20 and 27 and, if so, it will run the part of the payload which consists of sending 18000h times one-byte-long TCP/IP packets to the IP address 'C689F05Bh' which, in a more readable form, is 198.137.240.91, and which currently resolves to the name 'www.whitehouse.gov'. [*This is no longer the case, since the IP address of the Whitehouse Web site has been changed - Ed.*]

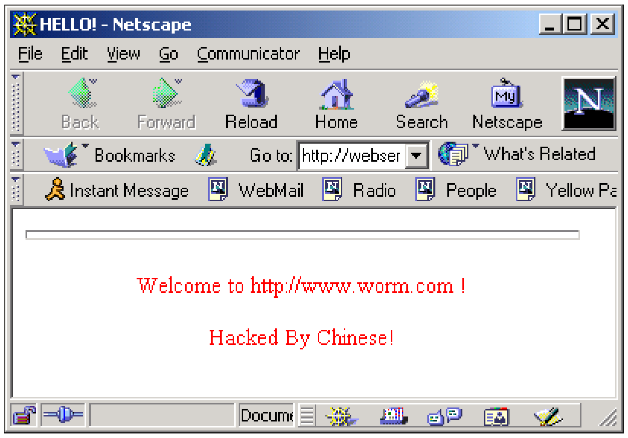
Next, the worm will run its random number generator routine, the purpose of which is to provide targets for infection. The routine uses two things as seeds for the stream of random numbers: the current second/millisecond fields of the current system time, and the thread number.

Combined, these two could produce a lot of different IP streams. I say 'could' because of the way the algorithm works – the entropy provided by the 'second' and 'millisecond' fields of the current time is lost in the computations, so that leaves us with exactly 100 possible streams of IP addresses, which only depend on the thread index, again, only in the range 0–99.

Therefore, whenever a copy of the worm receives control, it will start hitting a predictable invariant stream of IP addresses, thus highly limiting its ability to spread. For example, the stream of IP addresses generated by the first thread in the worm will always start with the following values: 7.107.254.83, 252.118.171.204, 198.83.139.183, 33.250.241.248, and so on.

Interestingly, this mistake seems to have been noticed by the author too; after the initial version of the worm became widespread, another 'fixed' version was reported. The second version seems to have its random number generator routine fixed, thus having much better chances to spread over the Internet.

The worm has another interesting payload which is run only if the current system codepage is 0x409, US English.


HELLO! - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape
Bookmarks Go to: http://webser What's Related
Instant Message WebMail Radio People Yellow Pa
Docume

Welcome to http://www.worm.com !

Hacked By Chinese!

Figure 2: Web page.

First, the worm will run a Sleep call set to two hours, and after that, it prepares to launch the payload. For that, it will scan the import table of 'w3svc.dll' for an API named TcpSockSend. After finding it, the worm replaces it with a pointer to a subroutine inside the worm copy which sends a specific Web page whenever a request to the HTTP server arrives. The Web page is shown in Figure 2.

It should be noted that, while patching the export table of 'w3svc.dll', the worm takes care to write-enable the area of memory in which the module is stored. This is required in order to patch the address of TcpSockSend, the function hooked by the worm. From here, the worm will simply loop again, trying other IP addresses.

Conclusions

There has been much debate around the fact that this may be the first modern worm that doesn't exist at any time in a file, nor use temporary files during replication, as for example *Linux*/Cheese or *Linux*/Ramen do.

'Bady' certainly exists only in memory or as a TCP/IP stream sent around the Internet, thus making it the perfect example for everyone's definition of the term 'worm'. But besides that, the truly important thing about it is that the impact of this worm could have been much, much worse.

If the worm had been written a little more carefully, to infect more than just *Windows 2000* systems running IIS4/5 with the indexing service installed and to use a really 'random' stream of target IPs, then stopping it would have been much more difficult. However, regarding detection, unfortunately the AV world was, in its majority, unprepared to handle 'Bady'. To detect and stop this worm, scanner plug-ins for firewalls are needed and, unfortunately, these are not very common. Also, to detect and clean it in memory, a couple of improvements to the scan engines are probably needed, such as the possibility to scan the memory associated with a thread launched in the memory space of a module attached to a process …

W32/Bady.worm

Aliases:	Code Red, CodeRed.
Type:	Network-propagated worm.
Infects:	*Windows 2000* machines running IIS4/5 with ISAPI enabled.
Payload:	Attempt to flood www.whitehouse.gov between 20th and 28th of each month – hooks all HTTP requests on systems with codepage 0x409, and sends a custom page back to the clients.
Removal:	Stop the WWW service on the affected machine, install the *MS* recommended patch, then restart the WWW service.