VIRUS ANALYSIS 2

CHIBA WITTY BLUES

Peter Ferrie, Frédéric Perriot, Péter Ször
Symantec, USA

W32/Witty is a UDP-based worm employing a vulnerability in *ISS* security products, such as the *BlackICE* firewall, to spread. More specifically, Witty uses a stack buffer overflow in the code that parses ICQ v5 packets.

Witty is very similar to last year's W32/Slammer (see *VB*, March 2003, p.6) in a number of ways: it is short (only 647 bytes for the attack buffer, excluding the variable UDP payload padding), its sending rate is limited only by available bandwidth, and it selects random target IP addresses. Unlike Slammer, however, Witty features a very destructive payload: it overwrites random portions of the hard drives of machines it infects.

THE PARSING EXPEDITION

The vulnerability used by Witty, discovered by *eEye*, was published on 18 March 2004 (PST). The worm appeared late on 19 March 2004 (PST). This is perhaps the shortest timespan we have experienced between the public announcement of a buffer overflow vulnerability and the spread of the corresponding worm. One day leaves little time for countermeasures to be deployed, especially since this vulnerability was announced on a Friday!

The bug in *ISS*'s software is in the Protocol Analysis Module (PAM) and is related to the parsing of ICQ v5 datagrams supposed to originate from ICQ servers. The PAM module is located in a DLL called iss_pam1.dll (for the *BlackICE* product). The module takes a UDP source port of 4000 as an indication that an incoming datagram is an ICQ server answer (it does so regardless of the destination port – this assumption is necessary, given the connectionless nature of the UDP protocol). Then the module parses the packet according to the ICQ protocol. If an ICQ v5 SRV_MULTI compound message is received, containing an SRV_USER_ONLINE packet followed by a specially crafted SRV_META_USER packet, it may result in a stack buffer being overwritten. Witty sends just such a message, overflowing an email address field, and hijacking a return address in a typical stack-smashing attack.

The SRV_USER_ONLINE and SRV_META_USER packets are obviously anomalous. In fact, they are not valid messages according to the ICQ v5 protocol. Instead, their fields are tuned to force a specific code path to be taken in the PAM module. The SRV_USER_ONLINE portion of the datagram contributes in setting up the parameters required for the PAM module to parse the malformed SRV_META_USER portion. The packet sizes are the exact minimum required. Most fields are zeroed out, but the spoofed user IP address is set to one in order to bypass a check for a non-zero value. To be able to produce such an optimised exploit in such a short time, it looks like the author enjoyed an uncanny knowledge of the inner workings of the vulnerable code.

MONA LISA OVERFLOW

The overlong email address submitted by Witty (which gives the virus its name – it starts with an ASCII string that includes 'insert witty message here') is copied into a 512-byte stack buffer through an sprintf() call. After sprintf() returns, the function whose stack frame holds the overflowed buffer attempts to return to its caller. Since the return address is overwritten, the control flow is hijacked. Instead of returning, the function jumps to a 'jmp esp' instruction located at a constant offset in iss_pam1.dll. The 'jmp esp' instruction in turn transfers control to the top of the stack, which contains a backwards jump pointing to the beginning of the worm body.

The first step in the execution of the worm body is to initialize register 'edi' to point to the beginning of the UDP packet payload. This register is later used when sending copies of the worm to new machines. Rather than rebuilding the attack buffer from scratch, Witty locates the original copy of its attack buffer on the heap. It does so by following a saved pointer on the stack frame of the procedure calling the vulnerable ICQ parsing routine. The obtained location is a pointer to the IP packet payload, so the worm then skips the UDP header by adding 8 to register 'edi'.

Witty then alters the stack pointer to avoid clobbering its own code by pushing data on the stack.

KUANG EXPERT TYPE 11H

The worm functionality is fairly simple: it creates a UDP socket and binds it to port 4000, and starts sending copies of itself to random IP addresses, on random destination ports. Periodically, after every infection cycle (consisting of 20,000 attacks) it runs its destructive payload. Witty relies on the Import Address Table entries of iss_pam1.dll to call the kernel32 APIs it needs. It resolves the Winsock APIs dynamically.

The randomization is carried out by a Pseudo-Random Number Generator (PRNG) similar to the Linear Congruential Generator in Slammer (it uses the same multiplier and delta.) The PRNG is seeded with the value returned by GetTickCount() on the beginning of each infection cycle. As a result of a single PRNG being used to

randomize the target IP, destination port, and size of the UDP payload, these three parameters are correlated: 90 per cent of the IP address space will be attacked in a single way. To a given IP address in this space, the worm always sends a payload of a constant size to a constant destination port. This is independent of the attacking machine. The other 10 per cent of the IP address space may be attacked in two different ways (and no more than two). For instance, the IP address 209.134.161.35 may receive worm datagrams with the following characteristics:

209.134.161.35 attack type 1: 882 bytes to port 23280/udp
209.134.161.35 attack type 2: 1030 bytes to port 13615/udp

Occasionally, the PRNG of Witty will generate some IP addresses finishing in '.255'. Since the *eEye* advisory mentions explicitly that the vulnerability is exploitable by broadcasting malformed packets, it is legitimate to wonder whether the worm takes advantage of such a mass-propagation. Fortunately, the use of a socket to broadcast datagrams requires a specific option to be set, and the worm author did not take this step.

After every infection cycle, Witty attempts to overwrite a random 64k area of one of the first eight hard disks with the beginning of the memory image of iss_pam1.dll.

PATTERN RECOGNITION

We believe that the variable destination port and UDP payload size used by Witty were designed by the author to evade IDS products. A consequence of the variable payload size is that additional padding is sent following the worm body. The padding just happens to be the content of heap memory after the IP payload of the worm packet, and as such it is variable. Thus the worm packets will not only have variable size, but also variable checksums in the general case. In addition, as we mentioned above, the target IP, destination port, and payload size are correlated, which may lead to confusion when looking at the worm traffic from a single IP (one may think that the UDP payload is a constant size).

All in all, Witty had some interesting characteristics that probably allowed it to fly under a number of IDS radars. It is likely that the author of the worm was familiar with IDS systems.

BURNING CHROME

There have been several vulnerabilities discovered recently in security software, from bugs in OpenSSL (exploited by the Linux/Slapper worm) to ones in *Microsoft*'s ISA Server and there are surely more to come. Since security software layers are naturally at the front line of defence, vulnerabilities in them are hard to mitigate. Vulnerable security software may lull the user into a false sense of security: disabling network services and closing ports does not prevent the parsing of incoming data destined for these services. This is particularly striking in the case of Witty: most clients do not have any ICQ v5 client installed because the current ICQ protocol version is 8, and version 5 has been obsolete for years!

COUNT ZERO

According to the *CAIDA* analysis of the spread of the Witty worm (http://www.caida.org/analysis/security/witty/), the early propagation of the worm does not match the expected rate of a natural infection. The pool of infected machines appearing in the first moments of the epidemic is too high to be explained by regular network scanning and exploitation. Instead, the *CAIDA* analysts propose that the population of the worm was seeded, by the worm's author injecting the worm code manually into a few pre-scanned vulnerable machines. We agree.

Rather than inferring this from epidemiological data, our evidence relies on the observation of a side-effect of the worm propagation method: when Witty sends itself to a target machine, it randomizes the size of the datagram it sends. The size of the datagram is between 768 bytes and 1279 bytes, therefore the datagram systematically includes a portion of unused data following the 647 meaningful bytes of the worm. Had the worm originated from a single instance, all of its replicants would share the same tail (between bytes 648 and 767). Such is not the case: firewall logs show that at least two different tails exist (there may be more such tails, the seeding population could be determined by counting them).

CONCLUSION

Insert witty conclusion here. [sic]

W32/Witty

Aliases:	W32.Witty.Worm, W32/Witty.worm, WORM_WITTY.A, Worm.Win32.Witty.
Size:	Variable, between 768 and 1279 bytes for the UDP payload.
Type:	Internet worm.
Exploits:	buffer overflow in ICQ parsing routine of the ISS PAM module CAN-2004-0362.