VIRUS ANALYSIS 2

MOSTLY HARMLESS

Peter Ferrie and Frédéric Perriot
Symantec Security Response, USA

The LSASS vulnerability of *Microsoft* security bulletin MS04-011 affects *Windows 2000* and *XP*, the two most widespread *Microsoft* operating systems today. It is a stack overflow, hence easily and reliably exploitable – and *eEye* was kind enough to provide the world with thorough documentation of the possible exploitation vectors.

Following in the path of previous high-profile vulnerabilities, the LSASS bug was quickly targeted by proof-of-concept exploits, themselves reused in worms including W32/Sasser.A. Despite the publicity that surrounded Sasser due to its immediate success following its appearance (30 April 2004), this was not the first worm to make use of the vulnerability: some LSASS-exploiting Gaobot variants had surfaced about a week earlier. However, it was the automated infection of new systems that was the decisive factor in making Sasser more widespread.

BISTROMATHICS

Sasser infects new systems by exploiting one of the many vulnerabilities announced in the MS04-011 bulletin. The vulnerability is related to a stack buffer overflow in the file lsasrv.dll, which is normally loaded as part of the lsass.exe process (Local Security Authority Subsystem Service).

In order to find new victims, Sasser scans random IP addresses for vulnerable machines listening on port 445/tcp. Once such a machine is found, it attempts to exploit the LSASS vulnerability by sending a specially crafted RPC request to the LSASS named pipe on the machine. Upon successful exploitation, shell code is injected into the lsass.exe process, which executes a shell (cmd.exe) and binds it to a TCP port. The attacking instance of the worm then connects to this port and sends commands to the shell. These commands download and run the main worm executable on the newly infected system.

The worm download is carried out through FTP, using the default *Windows* ftp.exe program on the client side (victim). On the server side (attacker), Sasser implements its own crude FTP server, which listens on a non-standard TCP port. The infection scheme of Sasser is very similar to that of W32/Blaster (see *VB* September 2003, p.10), with the exception of using FTP instead of TFTP as the main transmission protocol. Worms get more reliable!

Once it is running on a new machine, Sasser installs itself in the Windows directory under the name ‘avserve.exe’ and registers itself in ‘HKLM\...\Run’ as the value ‘avserve.exe’,

in order to run on *Windows* startup. Then Sasser simply tries to infect new systems. There is no intended payload, time-triggered routine, or anything other than replication code in this worm.

INFINITE IMPROBABILITY DRIVE

Sasser generates target IP addresses using three different methods: completely random IPs are used 52 per cent of the time; random IPs located in the same /16 network as the host are used 27 per cent of the time; and random IPs located in the same /8 network as the host are used 21 per cent of the time. This method of skewing probabilities towards nearby hosts was used in W32/Welchia (see *VB* October 2003, p.10). The aim is to increase the probability of hitting vulnerable hosts, on the assumption that nearby machines suffer from the same misconfiguration problems.

The network scanning speed per attack thread is a maximum of four attacks per second, and Sasser spawns 128 attack threads running in parallel.

VOGON POETRY

In addition to the scanning threads, Sasser creates an extra thread devoted to listening for incoming connections on its FTP server port, 5554/tcp. Each incoming FTP connection is then serviced by a newly spawned thread.

Despite its extreme simplicity, the FTP server code in Sasser is buggy. All the code does is reply to five basic FTP commands – 'USER', 'PASS', 'PORT', 'RETR' and 'QUIT' – with some canonical answers, to please the ftp client of the other side of the connection, and serve the worm executable over an FTP data channel.

In the one command requiring a minimal amount of parsing, 'PORT', there lurks a buffer overflow due to the use of a strcpy() call (more on this later).

Sasser creates an embryonic log file in 'c:\win.log'. This may, originally, have been intended as a list of compromised systems, but due to what appears to be a bug, it remains always one line long. The file contains the IP address of the last system successfully infected, and a counter indicating how many systems have been infected from the local machine, in total, since the worm started running.

BABEL FISH

Sasser exploits one vulnerability, but it really makes use of two different exploits, depending on the platform that it is attacking. We shall refer to them as the 'short-form' and 'long-form' exploits. Moreover the 'long-form' exploit has two variants, using two different trampoline addresses.

In order to determine which *Windows* platform it is attacking, the worm fingerprints the remote target system by establishing a NULL session with it, and checking the Native OS field of a session setup response packet. Based on the contents of the Native OS field, '5.1', '5.0', or neither of these two strings, Sasser picks the short-form, long-form (first variant) or long-form (second variant) exploit, targeted respectively at *Windows XP*, *Windows 2000* and unidentified systems. The NULL session packets produced by Sasser seem to originate from a regular *Windows 2000* system, because the author of the exploit captured sample traffic and simply replayed it in the exploit code.

The mode of operation of the short-form exploit, used against *Windows XP*, is to hijack a return address. The trampoline address used in this case points to a 'call esp' instruction located in the address space of the lsass.exe module itself. (This is contrary to the comment in the publicly available source code of the exploit, which mentions it as a 'jmp esp'.)

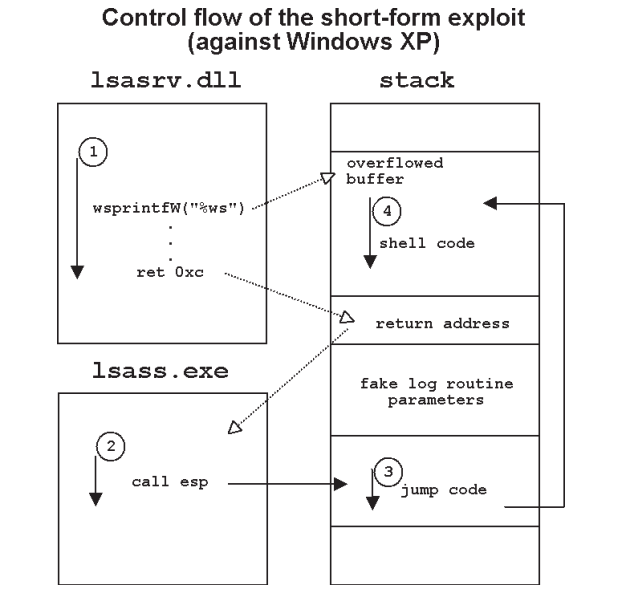
The long-form exploit is more complicated: it attempts to hijack both a return address and an exception handler on the stack. The trampoline address used against unidentified systems points to a 'call ebx' instruction in netrap.dll. The one used against *Windows 2000* systems points to a 'jmp ebx', according to the author of the exploit – but we have not been able to verify this information on our test platforms. The combined hijacking of a return address and an exception handler is probably designed to improve the reliability of the exploitation. In our tests, however, only the exception handler part was needed, and the sole purpose of hijacking the return address was to trigger an exception. (For more detail on the control flow of the exception handler hijacking trick, see *VB*, September 2001, p.4).

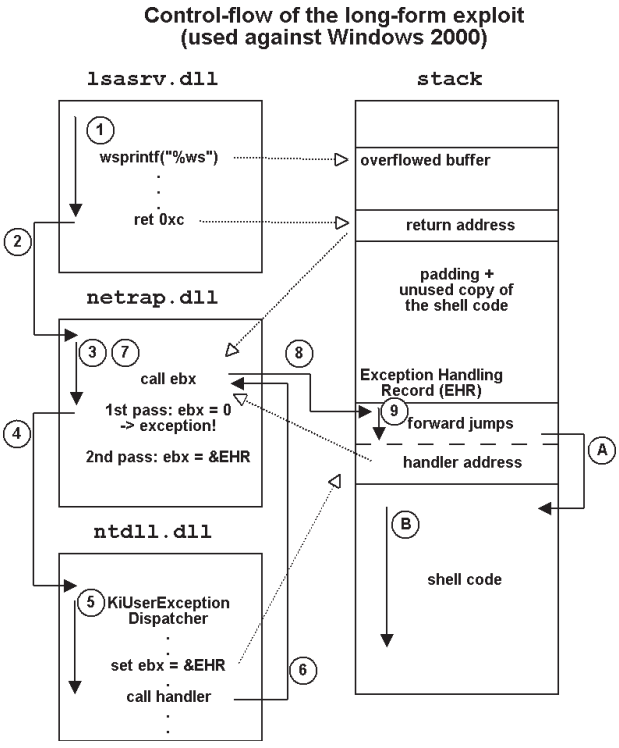
The layout of the attack buffers and the control flow of the exploits are depicted opposite.

HEART OF GOLD

Regardless of the exploit flavor, the attack starts by opening a connection to port 445/tcp of the target system, then going through the same protocol negotiation and session setup as in the fingerprinting. Next, the '\lsarpc' named pipe is opened on the remote system, and an RPC request is made against the LSA_DS (Directory Services) interface.

In the short-form exploit, Sasser sends an approximately 2kb attack buffer to the LSA_DS interface. The RPC request is small enough to fit in one RPC fragment, and the entire request is carried out with a single named pipe transaction. In the long-form exploit, the attack buffer is about 7kb long and the RPC request is split into two fragments.





As for the fingerprinting, the network traffic generated by the Sasser exploits is more akin to a replay than truly synthesized. The author of the HouseOfDabus exploit, which is reused in Sasser, is likely to have obtained it by sniffing traffic from a first-generation exploit that relied on a modified version of netapi32.dll, which was tampered with to allow an extra parameter in the signature of the DsRolerUpgradeDownlevelServer() function. This function employs the LSA_DS RPC interface for legitimate purposes, but normally operates against the local system only. The extra parameter in the version that has been tampered with identifies a machine, thus allowing remote exploitation.

The effect of the malformed request to the LSA_DS interface is a stack buffer overflow in the DsRolepDebugDumpRoutine() logging function of lsasrv.dll. The vulnerable function is normally used to write information to a file called 'DCPROMO.LOG', located in the '%windows%\debug' directory. It employs a 2kb stack buffer to hold log file lines, and it exercises no bounds-checking prior to using a sprintf('%ws') function to fill the buffer. By providing an over-long value corresponding to the '%ws' parameter, Sasser controls the return address and the rest of the stack after the buffer, which allows the execution of arbitrary code. Sasser relies on this to execute its shell code.

The exact sprintf() function in the buggy log routine varies among operating systems and service packs. For *Windows 2000*, the function is imported from msvcrt.dll, and is sprintf() in SP0, and vsprintf() in SP4. For *Windows XP*, the function is imported from user32.dll, and is wsprintfW() in SP0 and wvsprintfW() in SP1a.

The use of *Unicode* and ASCII functions (with or without the leading 'w' in the function name) explains the need for a long-form exploit and a short-form exploit providing more or fewer bytes in the '%ws' parameter for different platforms.

ZAPHOD WAS HERE

Surprisingly, Sasser performs the attack twice for each target machine. The likely explanation for this behaviour is also related to the kind of sprintf() function used in DsRolepDebugDumpRoutine(), more specifically to the use of the user32.dll implementation of sprintf() on *Windows XP* platforms.

The user32.dll implementation of the sprintf() functions has a limit of 1024 characters (not bytes – *Microsoft*'s documentation is not quite correct) that it will place in the destination buffer. Once the limit is reached, no further data are placed in the buffer. On the first exploitation attempt of a *Windows XP* system, when the logging routine is called from DsRolerUpgradeDownlevelServer(), this limitation is encountered because the size of the '%ws' parameter combined with the log line header exceeds 1024 characters.

As a result, the stack buffer is missing a final carriage return and a flag is set, indicating that the next bit of log information should simply be appended to the current line. On the next exploitation attempt, the flag is checked and the long '%ws' parameter is copied to the stack buffer without a

header. The '%ws' parameter alone fits in the buffer, no stack overflow occurs, and 'eventually' (readers are welcome to contact us if they have questions about this) the aforementioned flag is reset, allowing exploitation again.

The net effect is that exploitation of *Windows XP* systems works only every other time – at least under normal conditions. This explains the double attempt at exploitation by Sasser. The author might have assumed that the same condition could occur while attacking *Windows 2000* machines, but we have not observed this in our tests.

Once the lsass.exe process is coerced into running the shell code injected by the worm, the code binds to port 9996/tcp, accepts a connection from the attacker and runs a 'cmd.exe' shell as the SYSTEM user. The commands sent to the shell cause the worm executable to be FTP'd to a file whose name starts with four to five random digits followed by '_up.exe'.

DON'T PANIC!

Despite the obvious success of its spreading mechanism, Sasser suffers from a major limitation: it uses the wrong exploit parameter when it attempts to infect English versions of *Windows 2000* systems. Tests in the lab show that the trampoline address used by Sasser against English versions of *Windows 2000 Workstation*, *Server* and *Advanced Server*, *SP0* and *SP4*, does not correspond to a branch, but to another instruction (a locked 'mov') that causes an exception when reached (including from the exception record hijacked by Sasser, which leads to a crash).

We believe this behaviour is related to the single-byte vs. double-byte character platform issue, and that the Sasser exploit targeted at *Windows 2000* systems works only against double-byte character platforms. This is corroborated by reports from the field: of all Sasser submissions received by *Symantec* from the 'C:\WINNT' directory (the default installation directory for *Windows 2000*, whereas *Windows XP* uses 'C:\WINDOWS' by default), the vast majority originated from machines located in Taiwan and China. The source of the other submissions was unclear, but the names of the users suggested they may have been double-byte character platforms as well.

Thus, Sasser's exploit is effective only against *Windows XP* and some versions of *Windows 2000*. It fails against *Windows 2003 Server* (in fact, the buggy function is not even called).

DISASTER AREA

As in the case of W32/Blaster and other exploit-based worms, the end result of missed exploitation attempts against vulnerable systems is often a crash of the attacked process. The crash – in lsass.exe in the case of Sasser – manifests itself as an error message box warning the user that the system will be shut down. The author of the worm tried to call AbortSystemShutdown() from the main worm executable to prevent this suspicious behaviour, apparently overlooking the fact that the main worm code does not run at all if the exploit fails! On the other hand, if the worm is running successfully on a system that is then incorrectly compromised, the error message box might not appear.

If the exploit succeeds, the shell code terminates with an ExitThread() call after spawning the shell. This is clean enough by itself to ensure that the lsass.exe process keeps running, and makes the use of AbortSystemShutdown() unnecessary.

RESISTANCE IS FUTILE

From the point of view of Network Intrusion Detection, Sasser has one interesting feature: it sends the FTP data and the shell commands byte-by-byte, which results in the network traffic it sends being split into TCP segments starting at unpredictable boundaries. This may have been designed as a way to evade IDS products which do not have the capability to reassemble TCP streams. It may also have been accidental, since no such care is taken when the RPC attack buffer is sent. Nevertheless, the potential exists to mislead some IDS systems.

It was not long before a jealous contender attempted to take advantage of the vulnerability in Sasser's FTP server code. W32/Dabber, which appeared on 14 May 2004, does just this.

Soon a whole new branch of the security industry will appear, specializing in detecting exploitation of worm vulnerabilities: "*Protect your Sasser-infected machines with JamScan. JamScan not only stops Dabber, it also protects you against future worms exploiting the same flaw!*"

W32/Sasser.A

Aliases:	W32.Sasser.Worm, WORM_SASSER, Worm.Win32.Sasser.
Size:	15,872 bytes.
Type:	Internet worm.
Exploits:	LSASS vulnerability, MS04-011, CAN-2003-0533.