beginning Saturday, January 25 at approximately 12:30 a.m. EST, a distributed denial-of-service attack spread rapidly throughout the global Internet. Within 10 minutes, most of the vulnerable hosts on the Internet were infected. By morning, Bank of America customers could not withdraw money from 13,000 ATMs. Continental Airline's Web site was offline, forcing manual check-in. Normally heavy Internet trading on the South Korean stock market vanished. Many other Web sites and Internet services were also rendered inaccessible by the Sapphire (or Slammer) worm responsible for the attack.

Sapphire is a 376-byte worm that infects Microsoft SQL Server 2000 hosts via the SQL Resolution Service running on UDP port 1434. The worm does no damage to the infected machine: it does not modify disk files, alter or inspect database contents, or interrupt database execution. It merely probes for other SQL Servers to infect, by generating random IP addresses and sending UDP packets to port 1434 on those addresses. Since most of these IP addresses are not local, the resulting flood of packets saturates the host's connection to the Internet.

That such a small worm could so effectively disrupt so many servers so rapidly was somewhat surprising. The CodeRed, I Love You, and Nimda worms were all much bigger than the single Sapphire packet and took much longer to propagate. That such a simple attack was possible was no surprise at all. Sapphire exploits a buffer overflow vulnerability in SQL Server 2000 for which CERT issued a security advisory and Microsoft issued a patch six months earlier. When the Resolution Server receives a packet of type 04, it uses data from the packet to build a registry key in a fixed-size buffer on the stack. The unpatched code performs no length checks on the registry key it constructs. If the received packet is long enough, the constructed registry key overflows the stack-allocated buffer and overwrites the current function's return address, which follows the buffer in memory. The Sapphire worm consists of a single overly long packet that causes this return address to be overwritten with the address in the buffer where the worm's code resides. The worm code begins executing when the function returns.

Buffer overflow bugs of this nature are a common source of security vulnerabilities in programs written in languages like C and C++. They also arise frequently in ordinary program development in such languages, where they cause memory corruption that leads to erratic program behavior, application crashes, and machine crashes. They can be difficult to debug because the resulting program behavior usually cannot be explained abstractly in terms of functions, variables, expressions, and statements, but must be understood at the machine level in terms of addresses and bytes. In turning a buffer overflow bug into a security breach, an adversary exploits this abstraction gap to corrupt memory in a carefully calculated way.

Some modern programming languages (for example, Ada, C#, Common Lisp, Eiffel, Java, Modula 3, Scheme, and Standard ML) prevent this failure of abstraction. Such a language is said to be “type-safe” because its implementation ensures, via some combination of compile- and runtime checks, that the value a variable takes on or a function is passed always matches the language's notion of the variable's or function's type. When a type violation arising from a programming error is about to occur, such as an access to the 257th element of a 256-element buffer, the language implementation either terminates the program or raises a language-level exception.

Type safety makes program development and debugging easier by making program behavior more understandable. More importantly, type safety prevents an adversary from turning a type violation into a security breach. While an adversary might be able to provide inputs that trigger a runtime check, memory corruption cannot occur. There is no way the adversary can cause a buffer to overflow and be reinterpreted as a sequence of machine instructions. (For type safety to prevent security breaches, the capability some compilers provide to turn off runtime checks must not be used.)

Type safety is not a panacea for security. Other kinds of bugs besides type violations can lead to security problems. Even the termination of an application due to a type violation can result in denial of service. But type-safe languages make it much more difficult for an adversary to turn a type violation into a more serious security breach. Type-safe languages provide an important line of defense in developing applications safe for today's networked world.

Andrew Wright (akwright@acm.org) is research staff at a large networking company.