

# Protection and Security

## 1 Systems Security

Security is one of the new buzz-words in systems. Although security has been a serious issue in computing for decades, most of the work was carried out by shadowy 'haX0r' figures and companies cranking out patches for their software. The proliferation of the internet has opened up many individuals to electronic attack, and now companies like Microsoft and Cisco are making security a priority. Researchers can now get funding to do 'security research', and OS textbooks now discuss security issues (albeit abstractly).

As you will see, many of the security vulnerabilities out there 'in the wild' are the result of an insufficient understanding of the (often nebulous) semantics of C. Buffer overflows, heap overflows, etc. are caused by programmers insufficiently isolating user input from the program's internals. Some security vulnerabilities are the result of design flaws, that no amount of coding can overcome, but many of the day-to-day vulnerabilities are caused by the malicious exploitation of subtle properties of the C language. This last point is worth repeating, because modern type-safe languages (e.g. Java, Lisp, ML, Smalltalk, etc.) are immune from many of these vulnerabilities. A garbage-collected language cannot have double-free errors, for instance. A type safe language cannot accidentally overflow arrays, therefore conventional buffer overflows are impossible. Additionally, type-safety usually means that type casting is checked and guarded, which makes it impossible to overflow ints, for example. Of course, type safety is no silver bullet. Type safety merely eliminates the possibility of large classes of errors occurring in the code.

And that's one of the frustrating things about security work (from an academic point of view), most of the problems would be solved if programmers just used a type-safe language. The reason they don't is historical and sociological. Rather than just use a safer language, companies would rather continue to develop in an unsafe language (C++ usually) and then spend MORE money to hire a security consultant to audit their code base. Of course, any problems uncovered during the audit may not be resolved by the time the deadline arrives. It seems odd that companies (and consumers) are willing to pay more money incrementally to continue developing unsafe code than to pay more money upfront to switch over to a safe environment.

### 1.1 Modern Insecurity

Modern security work has its own 'hacker' vocabulary. A vulnerability is a software defect that can be used by an 'attacker' to do 'something bad'. An 'exploit' is a bit of software that exploits a vulnerability to do 'something bad'. An 'attack' is an event where an exploit is used on a system with a vulnerability. Note that 'something bad' is usually highly contextual. It could be as bad as launching a root shell or as innocuous as slowing down the machine being attacked (with leaking sensitive information somewhere in between).

There are many different types of security violations, a partial list running from more to less severe is:

1. Total loss of control (zombies/bots, rootshell)

2. Information destruction/modification (vandalism or accidental)
3. Information disclosure (SSN, credit card info, classified material)
4. Unauthorized access (break-ins)
5. Denial of service (D.O.S., spam)

These run the gamut from inconvenience to absolute disaster, with various gray levels of awfulness in-between. Chances are that you've recieved some spam in your life, and if you've ever used a windows machine connected to a network, chances are you've had to deal with viruses. So, we've all had to deal with some security shortfalls. But how can this happen? What are all these evil 'haX0rs' doing that lets them take over my machine?

## 1.2 Common Vulnerabilities

One common vulnerability is the bad password. This is an instance of human error or inexperience. Using your favorite color, or your spouse's name for your password is usually a bad idea. They can be easily guessed (and easily automatically guessed). Additionally, humans are vulnerable to 'social engineering', which is basically when someone tries to get you to divulge your password. This can be remarkably simple, just calling the person and claiming to be from tech support usually does the trick. This vulnerability exploits human nature and doesn't rely on any actual coding errors, and is basically impossible to correct in software. (Except passwords, where the system can run an automatic password guesser to yell at you if your password is too simple).

Another vulnerability is an unprotected network allowing attackers to eavesdrop on traffic. In the old days, when networks were rare, most traffic was sent unencrypted down the wire. If you had a machine on the network you could just put the ethernet card in 'promiscuous mode' (no, I'm not making that up) and read all the bits flowing over the wire. You'd see for instance the string "username:" going from machine A to machine B, then you'd see "lcarrol\n" going from B to A. Then "password:" going from A to B. Then "jabberwock\n" going back from B to A. You didn't even have to talk to anyone and you could still get all their info! This scenario almost never happens anymore, `ssh` is used almost exclusively for remote connections and web browsers also employ encryption to hide their sensitive traffic. Of course, wireless networks are introducing all kinds of new opportunities to eavesdrop :)

The classic vulnerability is one caused by software flaws. Perhaps most classic is the buffer overflow, where a coding error allows user input to overwrite program data (this is usually used to inject code and modify return addresses). The famous morris worm of 1988 used a buffer overflow to take control of a system process. There are many, many different kinds of software vulnerabilities, but for the purposes of this class there are essentially two: bugs arising from subtleties in the C programming language; and bugs arising from a misunderstanding of the interface of an external system (a SQL Database, say). Hackers in the wild actually perform a kind of deep reverse engineering, digging through programs at the assembly level looking for exploitable errors. As you can imagine, this kind of skill level limits exploit writing to a fairly small group of people. However,

what often happens is that an exploit is crafted and then conveniently packaged as a windows program, so that anyone with a network connection and a grudge can use the exploit to wreak havok (these are the so-called 'script kiddies').

The last class of vulnerabilities I'll mention is the aptly-named back door. A back door is an undocumented access point for a system. Sometimes manufacturers intentionally create backdoors in their software (often to spy on their customers), but usually a backdoor is an illicit software system installed on a machine by an attacker. The purpose of the backdoor is to give the attacker a hidden communication channel with the system. A good example is the so-called rootkit. A rootkit is a chunk of software that modifies the target system so that the attacker can connect as root but not show up to any legitimate system administrators. For instance, root kits will often replace system utilities (like `ps`, `top`, `who`, etc) so that illicit processes and users are not displayed (or logged). A rootkit enables an attacker to use a system without leaving a trace. Therefore, an attacker will exploit a software vulnerability in order to gain control of a machine for long enough to install a rootkit.