

Security

- While protection has been discussed throughout the class — kernel vs. user mode, protected memory, file permissions — these mechanisms have generally been focused on protection from accidental misuse (software bugs, novice users, corrupted data)
- The issue of *security* arises when we need to protect against attempts to undermine the intended use and function of a computer system's components and data
- Security requires good protection mechanisms, but good protection mechanisms don't ensure security

Definitions

- A system is *secure* if its resources are used and accessed as intended under all circumstances; any instance when this is not the case is a *security violation*
- An *intruder* or *cracker* is a party that is intentionally attempting to breach the security of a system
- A *threat* is the potential for a security violation
- An *attack* is an actual attempt at violation (which may or may not be successful)

Types of Security Violations

“Inappropriate use of system resources” can really be anything, but there are a number of typical categories

- *Breach of confidentiality*: Unauthorized reading of data
- *Breach of integrity*: Unauthorized modification of data
- *Breach of availability*: Unauthorized destruction of data
- *Theft of service*: Unauthorized use of resources
- *Denial of service (DoS)*: Preventing legitimate use of a system or service

Typical Violation Methods

- *Masquerading*: One party pretending to be another

Successful masquerading usually results in *breach of authentication* — the intruder gains more privileges than they should typically have in a system

- *Replay attack*: Malicious or fraudulent repetition of a (previously) valid transaction

Replay attacks typically involve *message modification*, where the replay is changed in a key way to again give the intruder additional privileges

- *Man-in-the-middle*: “Concurrent” masquerading where the attacker intercepts two-way communication

A precursor to a man-in-the-middle attack is *session hijacking*, where a valid communication session is intercepted and subverted

Levels Needing Security

Security isn't just about the technology itself:

- The *physical* level involves actual touch-access to the devices to be protected
 - The *human* level involves the personnel who are working with the devices to be protected — they require adequate training and awareness
 - When security issues do reach the system, they typically involve the *operating system* level
 - These days, the *network* level is now a major factor too
-
- The whole “weakest link” aphorism applies here — for instance, a highly secure operating system is useless if network transmissions are not encrypted, or if users display their passwords on post-it notes
 - The multilevel nature of security leads to the slogan “security must be *built-in*, not *tacked on*”
 - OS protection mechanisms (user accounts, permissions, privileged operations, logging and instrumentation) help to *implement* security, and are not security measures in and of themselves
 - However, we must balance good security against user convenience — if security measures are a pain to follow, then users will try to bypass or ignore them

Program/Software Threats

- A computer system is all about the processes that run on it — thus, it is no surprise that many security breaches are achieved by programs or software
- Pre-Internet, the creation of a security-breaching program was the main cracker activity; these days, we also have *network threats* which, though capable of their own security breaches, are often used to subsequently install a program threat
- Terms may vary, and this isn't necessarily an exhaustive list — new program threats are certain to arise

Trojan Horse

- A *trojan horse* is a program or code that abuses its environment to cause a security breach — typically, a trojan horse tries to get itself invoked by a highly privileged user such as an admin
- Trojan horses manage to get executed through some form of deception or misdirection, such as:
 - ◆ “Masking” a valid program with another program of the same name earlier in a user's program search path (modern GUI equivalent: adapting the name, icon, and maybe the location of a well-known or frequently used application)
 - ◆ Emulating another program, such as a login screen
 - ◆ Accompanying another program surreptitiously, e.g., spyware

Trap Doors and Logic Bombs

- A *trap door* is a special condition or data item within a program that results in a threat — sort of like an “evil twin” to an *Easter egg*
- Particularly nasty: place a trap door in a *compiler* — thus, all programs built by that compiler pose a threat
- Related to a trap door is a *logic bomb* — an internal condition check (current date, user status, availability of a Web site) that results in a threat if true, but does nothing if false

Stack and Buffer Overflow

- The *stack- or buffer-overflow* technique has been a wildly popular (and effective) way to breach security; it requires some technical savvy, but the payoff is great
- Software frequently copies data into *buffers*; if data *exceeding* the buffer size can be sent in, then we potentially overwrite the code, particularly the stack
- Usual technique is to send executable code in the buffer, and go past the buffer so we reach the top of the stack; at that location, we deposit the address of the buffer, which now holds the bad code

Combatting Buffer Overflow

- Program-level protection: Never do an unbounded copy (*strcpy*, *memcpy*)
- System-level protection: Disallow code execution from certain blocks of memory (stack, code pages)
- “Social” protection: Open-source software allows more parties to inspect code, and thus spot buffer-overflow vulnerabilities
- Buffer overflow requires technical savvy to figure out, but much less skill to mimic (i.e., by *script kiddies*)

Viruses

- The most well-known program threat is probably the *virus* — a program that can replicate itself
- Viruses start with a *virus dropper* — code that installs the “first copy” of a virus — delivered via other breach techniques (trojan horse, buffer overflow)
- Viruses are typically installed in a context that allows them to spread easily — privileged executables, macro-capable documents, boot sectors; *anti-virus* software works mainly by searching for the byte patterns of known viruses on these items

System and Network Threats

- While a program threat comes from a process executing *within* a computer, system and network threats come from data arriving from *outside*
- Such threats are tricky, because (a) we *do* want to open our computer to communications from the outside, but (b) we need techniques to ensure that the information that does arrive at our computer is valid (valid sources, valid content)
- Some threats, such as *phishing*, have a strong human factor, and so protection must go beyond technology

Worms

- A *worm* is a network-delivered program threat, made possible by abuse of a vulnerable network service
- A worm searches the network for vulnerable services, such as bypassing password restrictions (e.g., old-style *rsh*), buffer-overflow bugs (e.g., old-style *finger*), or delivering trojan horses (e.g., e-mail attachments)
- A service is compromised by making it execute a *grappling hook* or *vector* — software similar to a virus dropper that, on execution, downloads the worm onto the new networked victim; then, “rinse and repeat”

Port Scanning

- *Port scanning* is not an attack in its own right, but a technique for detecting system vulnerabilities
- A port scanner attempts connections over a variety of ports; when a port responds, tests are conducted to see if the service on the other end might be vulnerable (e.g., specific program or version check)
- Port scanning *is* detectable and traceable (if we're watching for it), so savvy attackers usually perform this via *zombie systems* — machines that are already unknowingly compromised

Denial of Service

- A *denial-of-service* (DoS) attack seeks to disable some network service provided by the target host — not quite the same as seeing private data or running arbitrary software, but highly disruptive nonetheless
- A typical DoS attack involves initiating and sustaining a very high volume of network connections; the attack can be *distributed* (DDoS), and like port scanning can be launched from zombies
- We can't really *prevent* DoS, only detect it (maybe) and initiate some kind of contingency plan

Phishing

- *Phishing* is interesting because it isn't a completely technological attack; it has a human, "social engineering" element
- Phishing is analogous to a network trojan horse: the attacker sends an e-mail to a potential victim, with links leading to deceptively trusted Web sites; the victim may then enter sensitive data into these sites
- To "phight phishing" (heh), user education is paramount: be careful what you click on, and be aware of Web addresses being used

SQL Injection

- *SQL injection* takes advantage of the general structure of many Web applications: arguments entered into these applications usually lead to some type of database query
- If an application is not carefully written, an SQL injection attack can actually expand the query sent to the database, resulting in unauthorized data access
- Note that guarding against this attack has to take place at the application level; there is no actual violation of network- or OS-level services

Cryptography

- Note how a common element in many of these attacks involves the equivalent of “don’t talk to strangers” — how can we make sure that (a) data we send can only be seen by their intended recipients, and (b) data that we receive comes from a known or trusted sender?
- For this, *cryptography* presents a viable set of techniques and algorithms: *encryption* helps to ensure condition (a), while *authentication* ensures (b)
- Cryptography allows us to send trustworthy data through an untrustworthy *medium* (the network)

Encryption

- *Encryption* is the act of transforming data so that, if seen in transit, the transformed version is not useful
- Specifically, we have an *encryption function* that can convert a *message* into a *ciphertext*; a *decryption function* performs the reverse
- These functions use a *key* as a parameter; the trick to effective encryption is making sure that we can determine the decryption function *if and only if* we know the key

Symmetric Encryption

- When the same key is used for both encryption and decryption, we have *symmetric encryption*
- The key becomes the focus of protection — it must be *shared* in a secure manner (e.g., offline), and must not be vulnerable to exposure
- Some algorithms (DES, AES) are *block ciphers* — they encrypt fixed chunks of bits at a time
- Other algorithms are *stream ciphers* — they encrypt a byte/bit at a time

Asymmetric Encryption

- *Asymmetric encryption* involves *two* keys: one for encryption and one for decryption — typically, the encryption key is the *public key* because this allows anyone to send you a message; the decryption key is the *private key* because this makes sure that *no one but you* can transform the ciphertext back to the message
- Asymmetric encryption relies on *one-way functions* — transformations which can't be reversed to unique values; in terms of real-world impact, these functions are computationally more expensive

Authentication

- Encryption constrains the *receivers* of a message; authentication constrains the *senders*
- Authentication is accomplished by accompanying a message with an *authenticator*; like a ciphertext, an authenticator is the output of some function that relies on a key
- A boolean *verifier* function checks to see if an authenticator corresponds to its message
- Authentication is the basis of *nonrepudiation* — proving that something could be done only by a specific party

Key Distribution

- The trick with cryptography is *protecting the key* — best done “offline” or *out-of-band*, but unfortunately that technique does not scale
- A technique that does scale is the use of *certificate authorities* — “publicly trustable” entities that distribute their public keys with software that needs them; these keys, in turn, can be used to sign and thus verify the keys of other parties
- These authorities (e.g., VeriSign) require proof of identification to get this *certificate* (the signed key)

User Authentication

- Not to be confused with *message* authentication is *user* authentication — how do we know that an individual is really who they claim to be?
- *Passwords* are the most common mechanism; they function as the *shared secret* between a user and the system, so they must be closely guarded (from *shoulder surfing* or *sniffing*) and difficult to guess
- *Biometrics* is approaching wide use — this involves devices that can read a person's unique physical traits, such as palm or fingerprint readers

Security Defenses

- Generally, effective security defense requires a multilayer approach; a *security policy* establishes what must be secured, and specifies security procedures at the *physical* and *human* levels
- A *vulnerability assessment* then checks a specific system for properties that make them susceptible to attacks
- Some form of *intrusion detection and protection* must then be established to catch or avoid potential attacks (e.g., virus detection for program threats, DMZs and firewalls for some network threats)