Operating Systems
Security

Nick Blundell
University of Birmingham

Lecture 21

Overview

- The Security Problem
- Program Threats
- System and Network Threats
- Cryptography as a Security Tool
- User Authentication
- Firewalling to Protect Systems and Networks

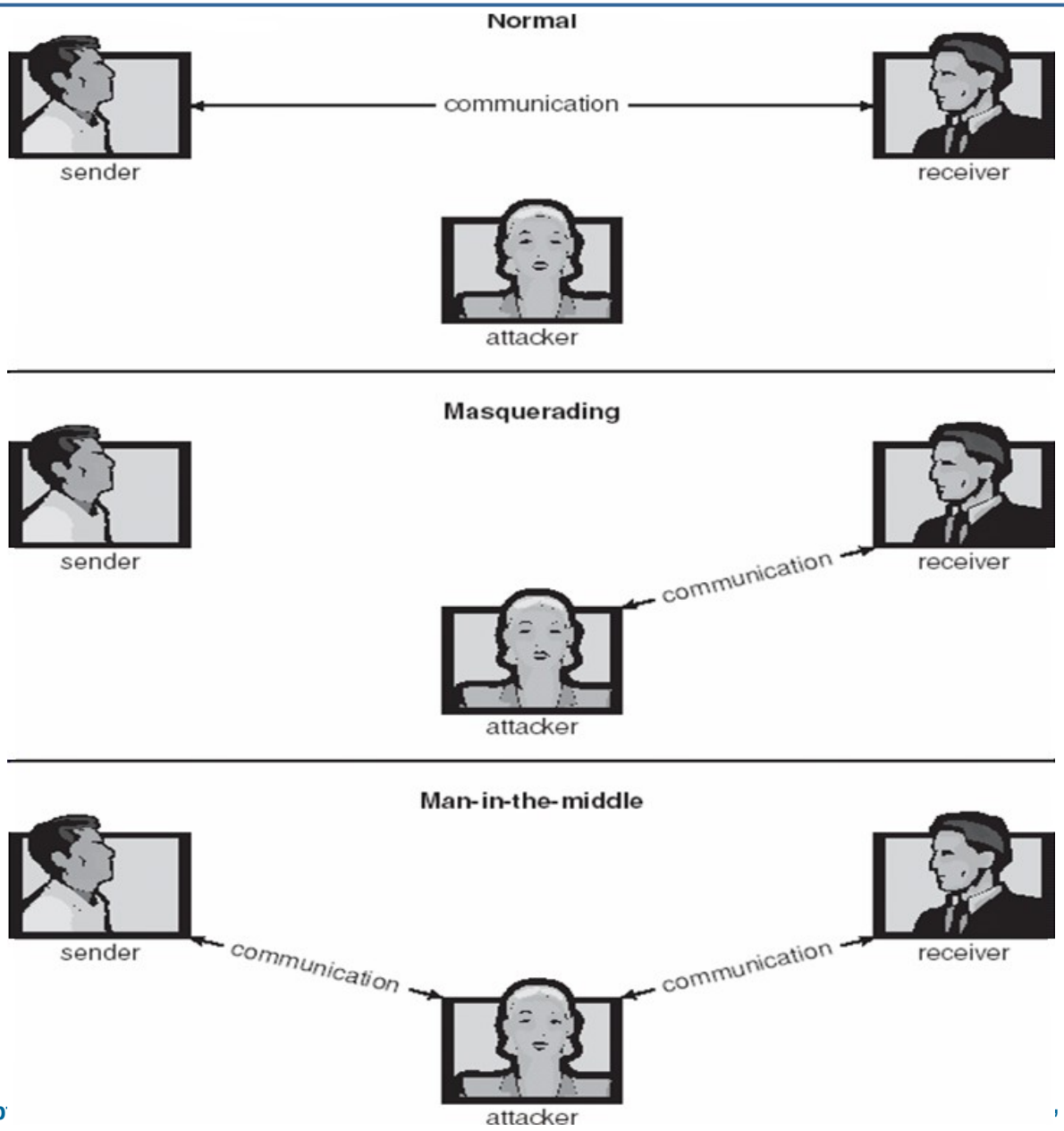
The Security Problem

- Security must consider external environment of the system, and protect the system resources
- Intruders (crackers) attempt to breach security
- Threat is potential security violation
- **Attack** is attempt to breach security
- Attack can be accidental or malicious

Security Violations

- Categories
  - **Unauthorised access/modification/destruction of confidential data/services**
  - **Theft of service – unauthorised use of resources**
  - **Denial of service – preventing the legitimate use of a service**
- Methods
  - **Masquerading** (i.e. pretend to be an authorised party)
  - **Replay attack –** replay earlier messages, possibly modified
  - **Man-in-the-middle attack –** Unbeknown to two communicating parties, all data flows (in one or both directions) through a malicious party.
  - **Session hijacking** – A malicious party intercepts a session between a user and a server (e.g. guess random session ID of a web service)
  - **Timing Attack –** Sometimes secure systems leak information through the time taken for execution that can be used to increase the likelyhood of a successful attack.

Standard Security Attacks



Security Measure Levels

- Security must occur at several levels to be effective:
  - **Physical** – who has physical access to the machine
  - **Human** -
    - social engineering (bribes, etc.), phishing (e.g. fake online banking), dumpster diving (bin diving)
  - **Operating System**
  - **Application Programs**
  - **Network**
- Security is as weak as the weakest link in the chain

Program Threats

- **Trojan Horse**
  - Code segment that misuses its environment
  - Exploits mechanisms for allowing programs written by users to be executed by other users
  - Spyware, pop-up browser windows, covert channels
  - Usually installed by the user, through posing to be a legitimate application
- **Trap Door**
  - Specific user identifier or password that circumvents normal security procedures
  - Could be included in a compiler
  - Can be difficult to detect in a large code base.
- **Logic Bomb**
  - Program that initiates a security incident under certain circumstances
    - There is a famous story, from way back, of a programmer who used errors in rounding numbers of currency transactions to move small amounts of money to a carefully chosen account in the bank. He then watched the balance quickly build. True or myth: we can certainly see how this would be possible.
- **Stack** and **Buffer Overflow**
  - Exploits a bug in a program (overflow either the stack or memory buffers)

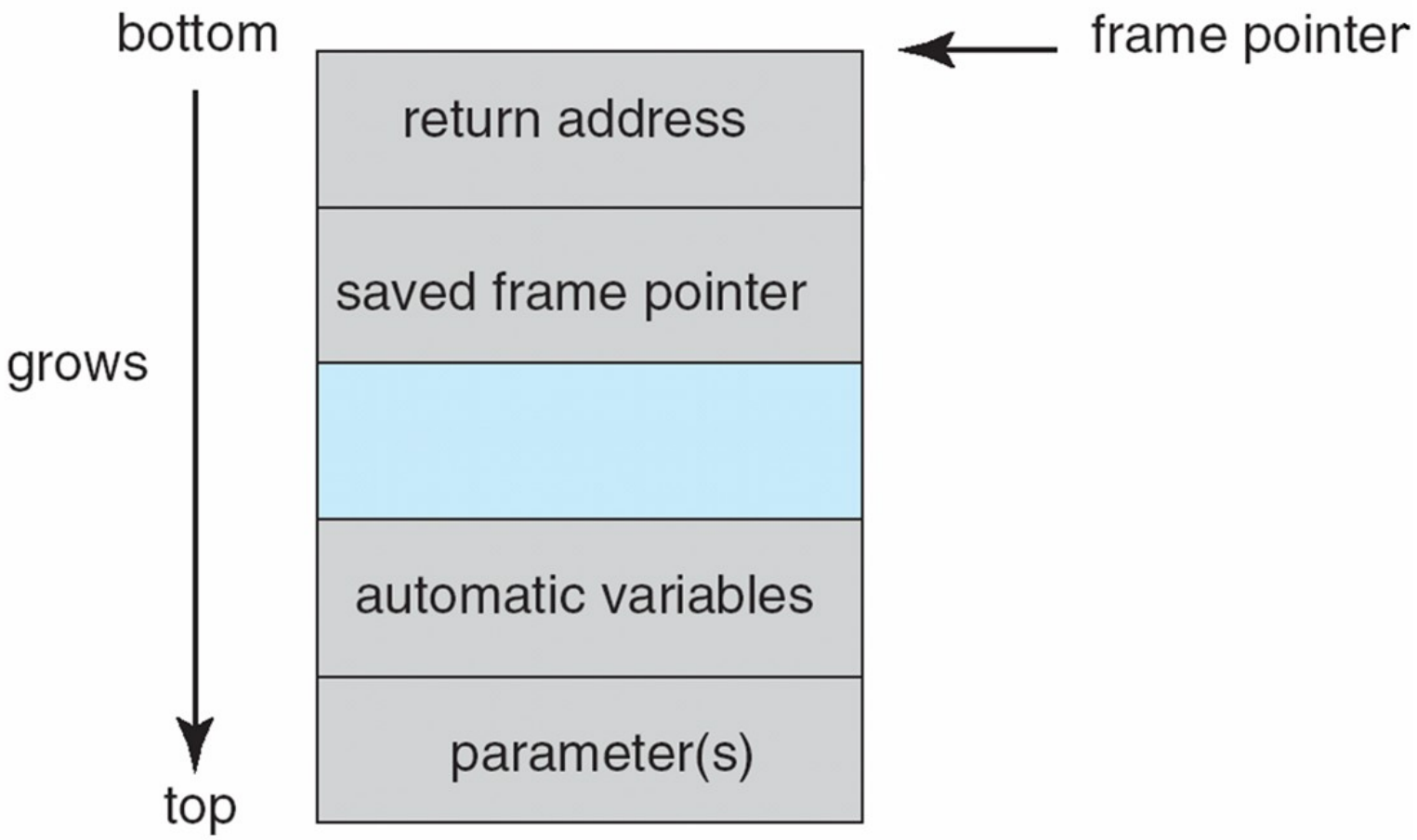
Buffer Overflow Attack

- Buffer overflow attacks are the most common exploits of services
  - In programming languages that do not have automatic bound checking, such as C/C++ it is possible for a buffer (i.e. array) to overflow, such that data is written beyond its limit, thus changing areas of memory that follow the buffer.
  - It is therefore possible for an attacker to carefully craft some input data to:
    - overwrite other variables (e.g. set flags to True or False to alter application behaviour, reset a password held in memory to gain access, etc.)
    - modify the stack frame of the process to run arbitrary code – this is particularly interesting.

Buffer Overflow Attack

- To understand why this is effective it is necessary to understand the process at the machine code level.
- A C/C++ compiler implements function calls by using the stack, such that, when a function is called, the return address (i.e. from where we called) is pushed onto the stack (so we can get back again).
- Then the first thing that happens in the function is that the new **stack frame** (i.e. region of the stack used only by this function) is established by pushing the parent function's stack frame pointer (stored in a well-known register) onto the stack and updating the pointer. Imagine the stack frame as a window over the stack, and so the whole stack will contain stack frames for all of the nested function calls that we are currently inside.
- The compiler then uses the stack frame pointer to make convenient relative references to automatic variables and parameters (i.e. variables that will no longer be needed when the function returns) that are allocated within the stack frame (e.g. the local variable **int my_int;** might be represented internally as **frame_pointer + 0x8**)

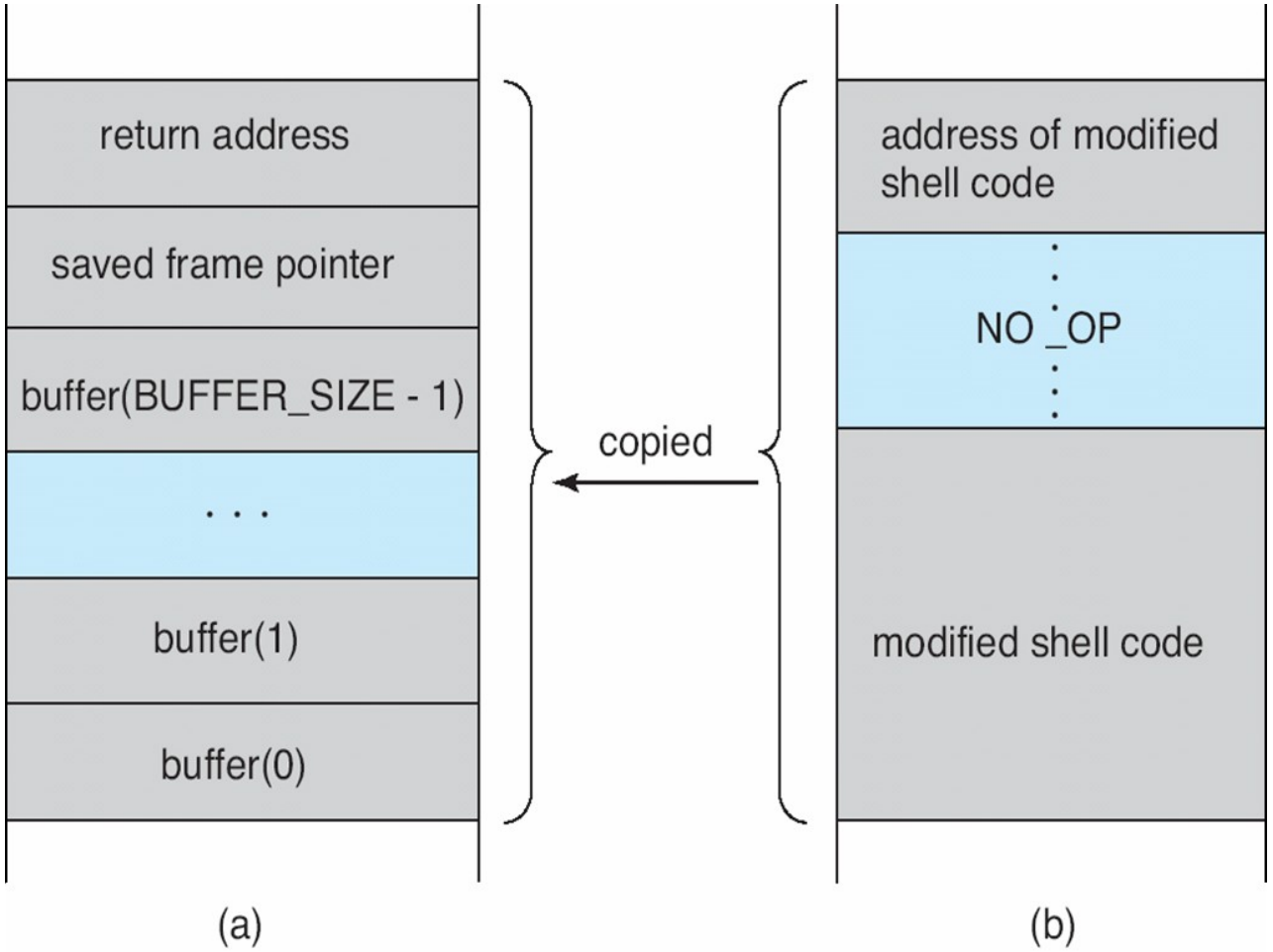
Layout of Typical Stack Frame



Buffer Overflow Attack

- So a possible attack can unravel as follows:
- By examining the code directly or via disassembly, or even by trial and error, the attacker finds that they can overflow a buffer by passing more data to the process than it was expecting (e.g. as a command-line arg, a network packet, a form's field input, etc.)
- Since the buffer resides in the stack frame, the attacker carefully calculates how many additional bytes need to be written in order to overwrite the **return** pointer at the start of the stack frame.
  - By changing this to an arbitrary address, when the current function completes it will jump not back to the calling function but to the new address
- Now the attacker crafts some code, perhaps to start a shell with root permissions, allowing them to run any command on the server (e.g. create themselves a login account for easy access in the future) and compiles this into a minimal machine code payload.
- To complete the attack, the machine code is overflowed over the buffer, complete with sufficient padding to change the return pointer which will point to the start of the malicious code (i.e. the start of the buffer)
  - When the unwitting function calls return.... BAM! The CPU jumps to the carefully crafted attackers payload code.

Hypothetical Stack Frame Before and After Attack



Before attack

After attack

Program Threats (Cont.)

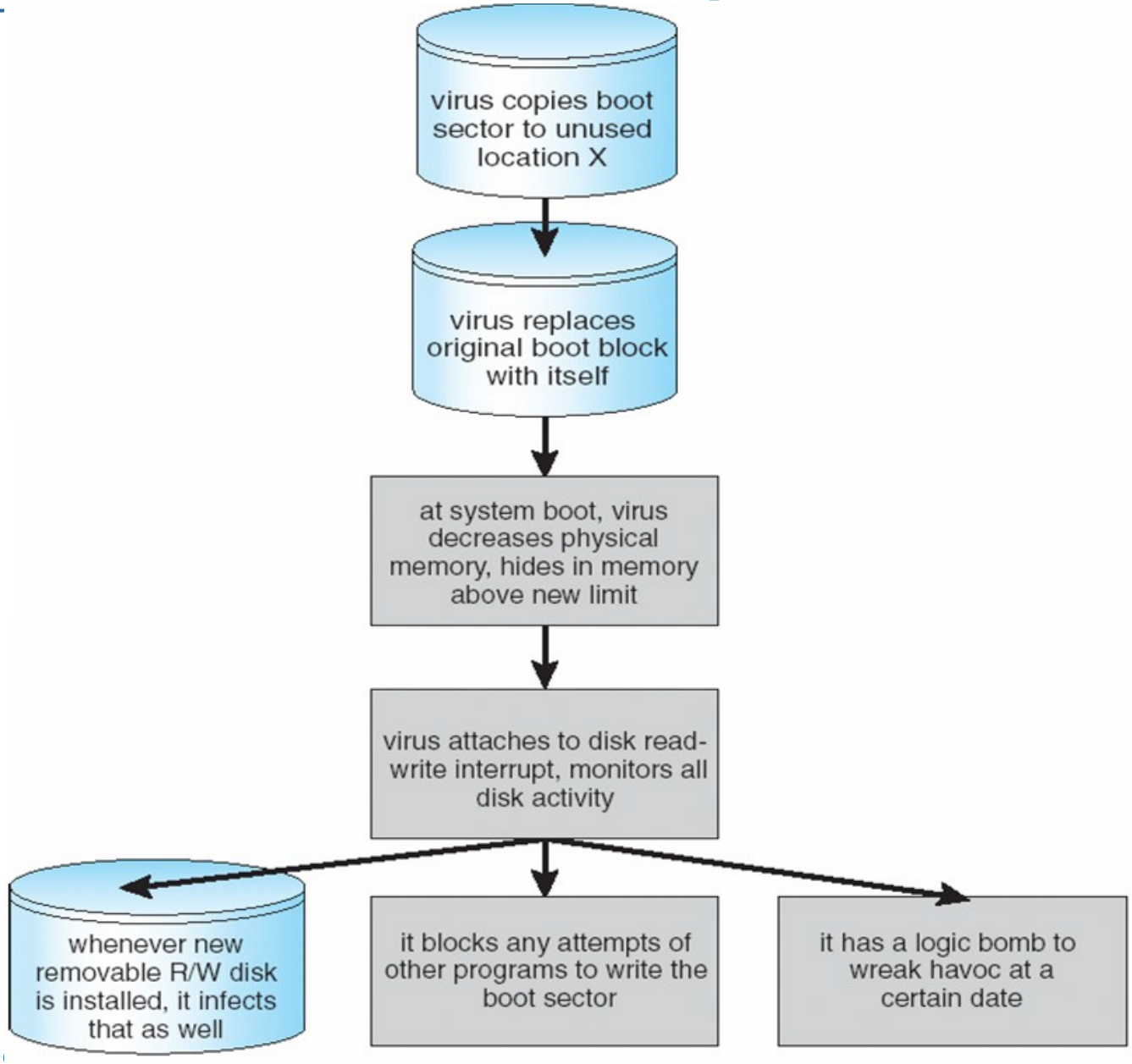
- Viruses
  - Code fragment embedded in legitimate program
  - Very specific to CPU architecture, operating system, applications
  - Usually borne via email or as a macro
    - Visual Basic Macro to reformat hard drive

```
Sub AutoOpen()
Dim oFS
  Set oFS = CreateObject(''Scripting.FileSystemObject'')
  vs = Shell(''c:command.com /k format     c:'',vbHide)
End Sub
```

Program Threats (Cont.)

- Virus dropper part of a virus that inserts virus onto the system, so it stays there – could be a innocent-looking trojan horse that drops the virus
- Many categories of viruses, literally many thousands of viruses
  - Executable file – virus appends/prepends itself to file
  - Boot – installed in boot sector of a disk, and run before OS when disk booted
  - Macro – High-level scripting language virus embedded in documents.
  - Source code – Modifies source code to spread itself.
  - Polymorphic – Virus changes its code every time it runs to avoid detection by signature.
  - Encrypted – Virus conceals itself to avoid detection through encryption, decrypting itself when it runs.
  - Stealth – Alters the system to make its detection harder (e.g. read call by virus checker on an infected file altered to hide virus)
  - Tunneling – Hides itself in an execution chain such as interrupt handlers or device drivers, so that with the virus file deleted, it will reappear sooner or later.

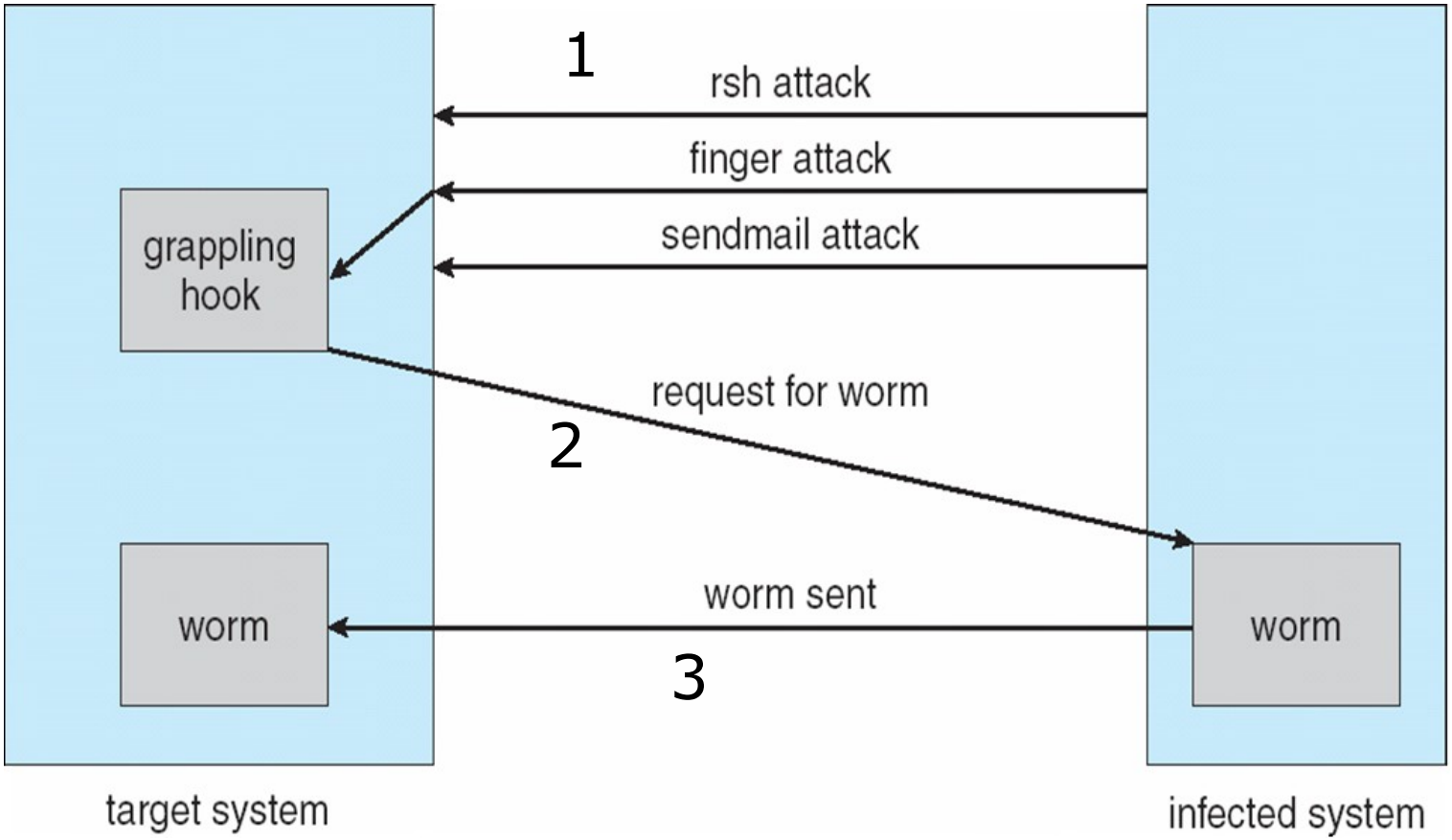
A Boot-sector Computer Virus



System and Network Threats

- **Worms** – use spawn mechanism (i.e. after infecting one computer, connect to others and infect those.
- Morris Internet worm: http://spaf.cerias.purdue.edu/tech-reps/823.pdf
  - Exploited UNIX networking features (remote shell access) and bugs in *finger* and *sendmail* programs
  - Grappling hook program uploaded main worm program
- **Port scanning**
  - Automated attempt to connect to a range of ports on one or a range of IP addresses
- **Denial of Service**
  - Overload the targeted computer, or its network, preventing it from doing any useful work
  - Distributed denial-of-service (DDOS) come from multiple sites at once, perhaps orchestrated by a virus.

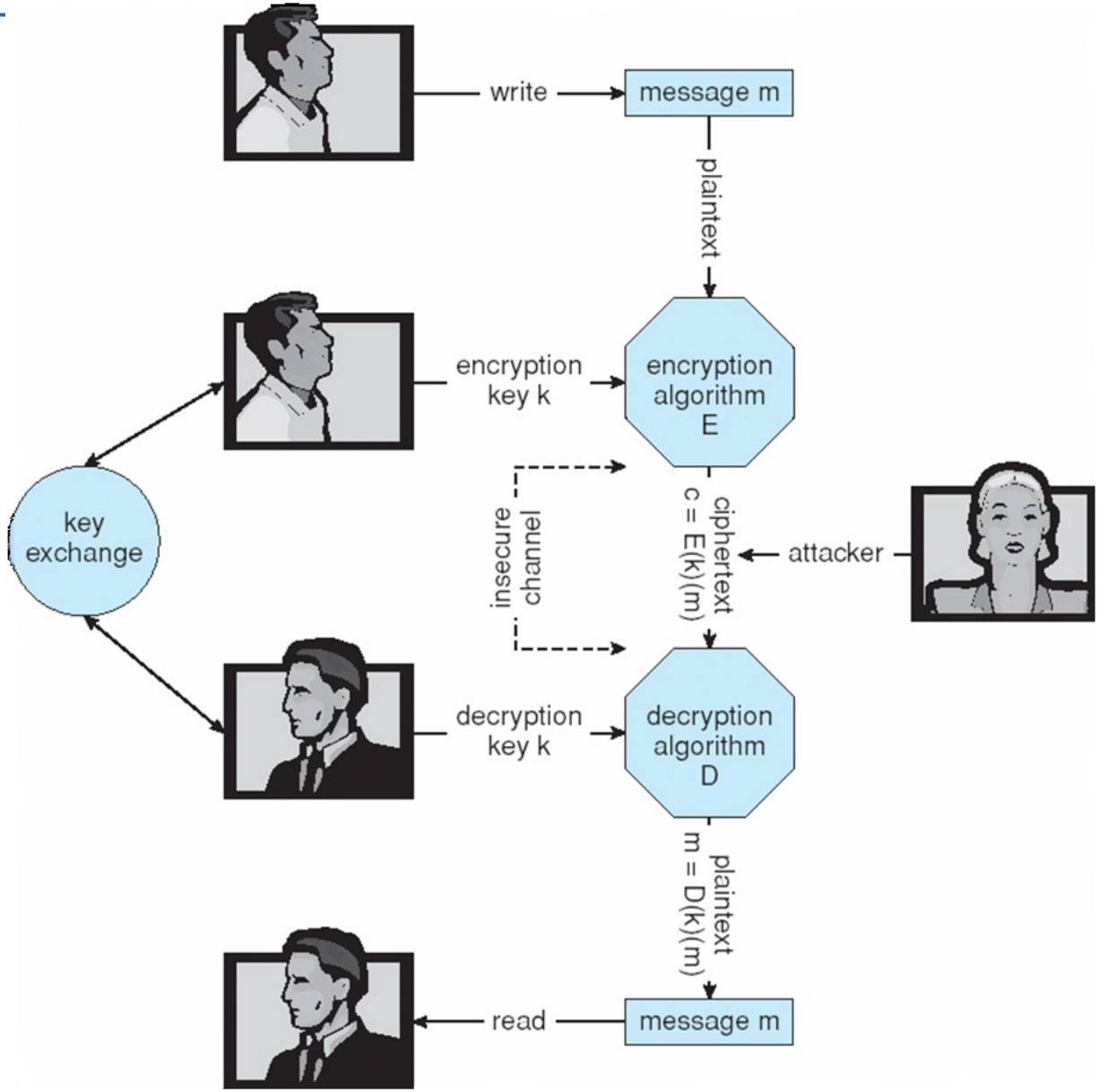
The Morris Internet Worm



Cryptography as a Security Tool

- Broadest security tool available
  - Source and destination of messages cannot be trusted without cryptography
  - Means to constrain potential senders (*sources*) and / or receivers (*destinations*) of *messages*
- Based on secrets (keys) – often the algorithms are widely published, or at least we should assume they could be, but without a key, knowing the algorithm usually not help you
  - Though weaknesses have been found in widely used algorithms, often due to bad implementations (e.g. not enough randomness (aka entropy) used to generate keys, etc.)

Secure Communication over Insecure Medium



Encryption

- Encryption algorithm consists of
  - Set of K keys
  - Set of M Messages
  - Set of C ciphertexts (encrypted messages)
  - A function $E : K \rightarrow (M \rightarrow C)$. That is, for each $k \in K$, $E(k)$ is a function for generating ciphertexts from messages
    - Both E and $E(k)$ for any k should be efficiently computable functions
  - A function $D : K \rightarrow (C \rightarrow M)$. That is, for each $k \in K$, $D(k)$ is a function for generating messages from ciphertexts
    - Both D and $D(k)$ for any k should be efficiently computable functions
- An encryption algorithm must provide this essential property: Given a ciphertext $c \in C$, a computer can compute m such that $E(k)(m) = c$ only if it possesses $D(k)$ (i.e. the decryption algorithm and key).
  - Since ciphertexts are generally exposed (for example, sent on the network), it is important that it be infeasible to derive $D(k)$ from the ciphertexts

Symmetric Encryption

- Same key used to encrypt and decrypt
  - $E(k)$ can be derived from $D(k)$, and vice versa
- DES (data encryption standard) is most commonly used symmetric block-encryption algorithm (created by US Govt)
  - Encrypts a block of data at a time
- Triple-DES considered more secure – applies DES three times to plaintext
- Advanced Encryption Standard (AES), aims to succeed DES
- It is not always convenient to encrypt whole blocks of data, so there are also stream ciphers which encrypt a continuous stream of bytes – less buffering → shorter transmission delays, such as for VoIP streams.
- RC4 (Rivest Cipher 4) is a commonly used symmetric **stream cipher**
  - A stream cipher can be build easily from a block cipher using a technique whereby a pre-buffered random key stream is encrypted in blocks then each byte of the key stream is XOR'd with each byte of plaintext prior to transmission.
- But in order to use symmetric encryption, we must exchange keys somehow, without them being intercepted

Asymmetric Encryption

- Public-key encryption based on each user having two keys:
  - public key – published key used to encrypt data
  - private key – key known only to individual user used to decrypt data
- This gets around the problem of having to distribute a decryption key, that could be intercepted!
- Must be an encryption scheme that can be made public without making it easy to figure out the decryption scheme
  - Most common is RSA (Rivest, Shamir and Adleman) block cipher
- The principal of RSA is that it is computationally infeasible to derive the two prime factors of a product of two prime numbers.
- Based on this, we can create the mathematical equivalent of a padlock:
  - I can send you my open padlock
  - You can put a secret in a box, click my padlock shut, and send it to me
  - Only I have the key to open the padlock.
- For the interested, here is an excellent explanation of the clever RSA: http://www.muppetlabs.com/~breadbox/txt/rsa.html

Cryptography (Cont.)

- Note that symmetric cryptography is usually based on efficient transformations (e.g. shifting bits, adding numbers), whereas asymmetric is based on more complex mathematical functions
  - So asymmetric is much more computationally intensive
  - Typically not used for bulk data encryption
- A beautiful combination, therefore, is to use asymmetric encryption only to securely exchange a short-lifetime randomly generated symmetric key, which can then be used with more efficient symmetric cryptography.

Authentication

- The next question, after securing our messages so that they cannot be read by eavesdroppers, is how do we know that we are communicating with the right person or service?
- Actually, authentication is closely related to public key cryptography, since if we trust that a specific person is the only person to have a private key, we can challenge them by encrypting a random number (termed a *nonce*) with their public key and then ask them to tell us what the number was.
  - Only they should be able to decrypt it, with their private key, thus proving who they are.
- When there is a central authentication server that holds passwords for all domain-wide users, it is often more convenient to use symmetric key encryption, but then we have to be careful about distributing keys. Special protocols (e.g. Needham-Schroeder handshake) were designed for this task.

Authentication without encrypted content – Hash Functions

- Creates small, fixed-size block of data (message digest, hash value) from some arbitrarily long message, *m*
- Hash Function *H* must be collision resistant on *m*
  - Must be infeasible to find two different messages that hash to give the same value.
- So we assume, with a high probability, that a particular hash value could have been produced only from one message
  - A slight change to the input message would give a completely different hash value.
- Common message-digest functions include MD5, which produces a 128-bit hash, and SHA-1, which outputs a 160-bit hash
- These hashing algorithms can be used for digitally signing documents.

Authentication without encrypted content – Digital Signatures

- Sometimes we want to send a document in plain text but with the possibility of later proving that a particular person sent that exact document (i.e. without modification)
- We can do this very simply by combining hashing and public key cryptography.
  - Actually, the trick is to use public key cryptography in reverse: encryption with the private key and decryption with the public key.

Encryption Example - SSL

- Insertion of cryptography at one layer of the ISO network model (the transport layer)
- SSL – Secure Socket Layer (also called TLS)
- Cryptographic protocol that limits two computers to exchange messages only with each other
- Used between web servers and browsers for secure communication (credit card numbers)
- The server is verified with a certificate signed by a trusted third party (e.g. verisign.com), assuring client is talking to correct server
- Asymmetric cryptography used to establish a secure session key (symmetric encryption) for bulk of communication during session
- Communication between each computer then uses symmetric key cryptography

Firewalling to Protect Systems and Networks

- A network firewall is placed between trusted and untrusted hosts
  - The firewall limits network access between these two security domains
- Can be tunneled or spoofed
  - Tunneling allows disallowed protocol to travel within allowed protocols (i.e. telnet inside of HTTP)
  - Firewall rules typically based on host name or IP address, which can be spoofed
- Personal firewall is software layer on given host
  - Can monitor / limit traffic to and from the host
- Application proxy firewall understands application protocol and can therefore restrict access to a service at a finer-grained level (i.e. an SMTP firewall may disallow certain SMTP commands.)
- System-call firewall monitors all important system calls and apply rules to them (i.e. this program can execute that system call)