CS 458: Computer Security and Privacy

Michael Noukhovitch

Spring 2016, University of Waterloo

Notes written from Erinn Atawater's lectures.

Contents

1 Introduction

1.1 Security

Security can be defined as:

confidentiality access to systems is limited to authorized

integrity getting the correct data

availability system is there when you want it

1.2 Privacy

There are many definitions but we will stick to **informational self-determination**, where you control the information about you

1.3 Terminology

assets things we want to protect

vulnerabilities weaknesses in a system that can be exploited

threats loss or harm that may befall a system

- interception
- interruption
- modification
- fabrication

threat model set of threats to defend against (who/what)

attack an action which exploits a vulnerability to execute a threat

control removing or reducing a vulnerability

1.4 Types of Defence

Defend against an attack:

- **prevent** stop the attack from happening
- **deter** make the attack more difficult
- **deflect** make it less attractive for attacker
- **recover** mitigate effects of the attack

Make sure that defence is correct with principles:

- **easiest penetration** system is only as strong as weakest link
- **adequate protection** don't spend more on defence than the value of the system

1.5 Methods of Defence

- Software controls: passwords, virus scanner ...
- Hardware controls: fingerprint reader, smart token ...
- Physical controls: locks, guards, backups ...
- Policies: teaching employees, password changing rules

2 Program Security

2.1 Flaws, faults, and failures

2.1.1 Defintions

flaw problem with a program

fault a potential error inside the logic

failure an actual error visible by the user

2.1.2 Unexpected Behaviour

A spec will list the things a program will do but an implementation may have additional behaviour. This can cause issues as these behaviours might not be tested and would be hard to test.

2.2 Unintentional Security Flaws

2.2.1 Types of Flaws

- **intentional**
  - **malicious**: inserted to attack system
  - **nonmalicious**: intentional features meant to be in the system but can cause issues
- most flaws are **unintentional**

2.2.2 Buffer Overflow

Most common exploited type of security flaw when program reads or writes past the bounds of the memory that it should use. If the attacker exploits it they can override things like the *saved return address*. Targets programs on a local machine that run with setuid priveleges or network daemons

Example 2.1. basic buffer overflow

```
#define LINELEN 1024

char buffer[LINELEN];
strcpy(buffer, argv[1]);
```

Types:

- only a single byte can be written past the end of the buffer
- overflow of buffers on the heap (instead of the stack)
- jump to other parts of the program or libraries (instead of shellcode)

Defences:

- language with bounds-checking
- non-executable stack (mem is never both writable and executable)
- stack at random virtual addresses for each process
- "canaries" detect if stack has been overwritten before return

2.2.3 Integer Overflows

Program may assume integer is always positive, and below certain value. Overflow will make a too large singed integer negative, violating assumptions.

2.2.4 Format String Vulnerabilities

Format strings can have unexpected consequences, `printf`

- `buffer` parse buffer for \%s and use whatever is on the stack to process found format params
- `%s%s%s%s` may crash your program
- `%x%x%x%x` dumps parts of the stack
- `%n` will write to an address on the stack

2.2.5 Incomplete Mediation

mediation ensure what the user has entered is a meaningful request

incomplete mediation application accepts incorrect user data

Though **client-side** mediation is helpful to the user, you should always perform **server-side** mediation

- check values entered by user
- check state stored by client

TOCTTOU "time of check to time of user" errors are race conditions that may affect correct access to resources

Defend by making all access control information **constant** between TOC and TOU

- keep private copy of request
- act on object itself as opposed to symlinks . . .
- use locks on object

2.3 Malware

- written with malicious intent
- needs to executed to cause harm

2.3.1 Virus

virus malware that infects other files (with copies of itself)

infect modify existing program (*host*) so opening gives control to virus

payload end goal of virus (e.g. corrupt, erase ...)

Protection can come in two forms:

signature-based keep a list of all known viruses (but how to deal with polymorphic viruses?)

behaviour-based look for suspicious system behaviour

2.3.2 Worm

worm self-containing piece of code that replicated with little/no user input

- often use security flaws in widely deployed software
- searches for other unprotected sources to spread to

2.3.3 Other Types

trojan horse claim to do something normal, but hide malware

scareware scaring user into agreeing

ransomware ransoming user's resource

logic bomb written by insider, already on your computer waiting to be triggered

2.4 Other Malicious Code

2.4.1 General Attacks

web bug tiny object in web page, fetched from a different server that can track you

backdoor instructions set to bypass normal authentication, come from

- forgetting to remove
- testing purposes
- law enforcement
- malicious purposes

salami attack attack from many smaller attacks

privelege escalation raises privelege of attacker, can cause legitimate higher privelege code to execute attack

rootkit tool to gain priveleged access and then hide itself

- cleans logs
- modify basic commands `ls`...
- modify kernel so no programs can see it

2.4.2 Man-in-the-middle

intercepts communication but passes it on to intended party eventually

keystroke logger logs keyboard input and spies on user

- application-specific
- system logger (all keystrokes)
- hardware logger (physical device)

interface illusions tricks user to execute malicious action with UI

phishing make fake website look real to extract user information

2.5 Nonmalicious flaws

covert channels transfer data through secret/non-standard channel (e.g. hide data in published report)

side channels attack based on knowledge from physical behaviour of computer

- RF emissions
- power consumption
- cpu usage
- reflection of the screen

2.6 Security Controls

2.6.1 Design

Design programs so they're less likely to have flaws

- modularity
- encapsulation
- information hiding
- mutual suspicion
- confinement/sandboxing

2.6.2 Implementation

When actually coding, reduce security flaws

- don't use C
- static code analysis
- formal methods
- genetic diversity (run varied code)
- educate yourself

2.6.3 Change Management

Make sure that all changes to the code maintain security

- track changes in a system (CVS . . .)
- do post-mortems of security flaws
- code reviews
  - guided code reviews
  - easter-egg code reviews (intentional flaws)

2.6.4 Testing

Make sure implementation meets specification *and nothing else*

black box testing treat code as an opaque interface

fuzz testing submit completely random data

white box testing testing which understands how it works, good for regression testing

2.6.5 Documentation

For posterity, write down:

- choices made
- things that didn't work
- security checklist

2.6.6 Maintenance

Make sure that code out there gets better not worse

standards rules to incorporate controls at each software stage

process formal specs of how to implement each standard

audits externally verify your processes are correct and followed

3 Operating System Security

3.1 Protection in a General-Purpose System

3.1.1 Overview

Protect a user from attacks, and protect resources:

- CPU
- memory
- I/O devices
- programs
- data
- networks
- OS

3.1.2 Separation/Sharing

Keep one users' objects separate from others'

- **physical** use different physical resource
- **temporal** execute at different times
- **logical** give impression that no other users exist
- **cryptographic** enrypt data to make it unintelligible

OS' can allow for **flexible sharing**, not "all or nothing"

3.1.3 Memory Protection

Prevent one program from corrupting other programs, OS, data...

fence register exception if memory access below address in fence register

base/bounds register pair exception if memory not between register pair

tagged architechture each memory word has extra bits that identify access to that word

3.1.4 Segmentation

segmentation each program has different mem segments for code, data, stack. Virtual address contains `<segment name, offset within segment>` and segment name is mapped to physical address in *Segment Table*

+ each address reference is checked for protection
+ different levels of protection
+ share/restrict access to a segment
- external fragmentation
- costly: dynamic length out-of-bounds checks, segment names

3.1.5 Paging

pages equal divisions of virtual address space

frames equal divisions of physical memory (size same as page)

page table maps page number to corresponding frame `<page #, offset within page>`, also contains memory protection bits for each page

+ each address reference is checked for protection
+ share/restrict access to a page with different rights
+ unpopular pages can be moved to disk
- internal fragmentation
- infeasible to have different levels of protection for different data

3.1.6 x86

Includes both segmentation and paging, with memory protection bits:

- no access
- read access
- read/write access
- *no execute

3.2 Access Control

3.2.1 Overview

We need to protect more than just memory, with three goals

- check every access
- enforce least privelege
- verify acceptable use

Create an **access control matrix** for a set of protected objects 0, subjects S, and rights R (r,w,x,o).

3.2.2 Access Control Methods

Access Control Lists each object has a list of subjects and their access rights

Capability unforgeable token that gives own some access rights to an object

RBAC admin assign users to roles and grants access rights to roles

- can be hierarchical
- a user can have multiple roles for different tasks
- **separation of duty** same person can't be responsible for two different roles on a task

3.3 User Authentication

3.3.1 Overview

Computers systems have to identify and authenticate users before authorizing them

identify who are you

authenticate prove your identify

3.3.2 Authentication Factors

Four classes, something the user:

- **knows** password, PIN
- **has** ATM card, physical key
- **is** biometrics
- **context** location, time

Using multiple factors ("two-factor" authentication) improves security if they are different types

3.3.3 Passwords

Attacks:

- shoulder surfing
- keystroke logging
- phishing
- password re-use
- password guessing

User defenses:

- choosing good passwords (long and not easily guessable)
- hygiene
  - write down > store insecurely
  - change regularly
  - site specific passwords
  - don't reveal
  - don't enter sensitive info on public computers

Admin defenses:

- store only cryptographic hashes

salt user-specific addition to hash (based on time of day ...)

pepper salt not stored alongside password

- use expensive hash
  - SHA-x take microseconds
  - bcrypt takes hundreds of milliseconds
  - scrypt also uses a lot of memory
- use a **MAC** message authentication code
  - also uses secret key to compute fingerprint
  - impossible to crack without secret key
  - as secure as hashing if key does leak
- make password recovery force a reset
  - don't email them the password, since you shouldn't be storing it
- one-time passwords
  - fight interception attacks
  - **challenge-response** to generate the password

3.3.4 Biometrics

Authenticate user if *physical characteristic* is sufficiently similar to stored trait, but this has issues

- remote authentication can't test if you are trying to bypass it
- since we check for similarity, not equality, false positives are more likely
- facial recognition software still not good enough for security
- privacy issue if your stored biometrics leak
- if leaked, you can change a password, but not biometrics
- some of your biometrics are not secret (easy to get photos of...)

3.4 Security Policies and Models

3.4.1 Trusted OS

We trust an OS if we have confidence that it provides security services which build on four factors:

policy set of rules outlining what is secured and why

model implementes the policy and can be used for reasoning about it

design spec of how OS implements model

trust assurance that OS is implemented according to design

3.4.2 Trusted Software

Software that does what we expect it to do and nothing more

functional correctness software works correctly

enforcement of integrity wrong inputs don't impact correctness

limited privelege access rights are minimized

appropriate confidence level rated as required

3.4.3 Security Policies

Basic military model:

- each object has a sensitivity level
- each object is assigned to one or more compartments
- subject can access if it *dominates* the requirements

Chinese Wall security policy that once you access info you can't access info about competitors

ss-property read access by subject to object if each object previously accessed is either from the same company, or a different type of company (no conflict)

***-property** write access by subject to object if all readable object by subject are either from the same company or have been *sanitized*

lattice security model where there is a unique upper and lower bound for any two points, i.e. each level is distinct

3.4.4 Security Models

Bell-La Padula regulates information flow in lattices, so users get information only with their clearance

- no read up (read more secure documents)
- no write down (write less secure documents)

Biba prevent inappropriate modification of data

- subjects and objects ordered by integrity
- no read down (don't contaminate reliable person with unreliable info)
- no write up (don't contaminate reliable info with unreliable person)

Low Watermark Property instead of enforcing integrity rules, just reduce integrity when it is violated

- **subject** if subject s reads object o, then $I(s)$ = greatest lower bound of $I(s)$ and $I(o)$
- **object** if a subject writes to an object, the reduce integrity of object to greatest lower bound

3.5 Trusted OS Design

3.5.1 Security Design Principles

least privelege use least priveleges possible

economy of mechanism protection mechanism should be simple

open design avoid security by obscurity (secret keys not algorithms)

complete mediation check everything

permission based default is no permissions

separation of priveleges two or more conditions to get access

ease of use make it easy or no one will use it

3.5.2 Security Features

identification/authentication access control

- mandatory: central authority determines access
- directionary: owners of objects have some control over who can access
- role-based: central authority defines roles

object reuse protection

- stored data should be inaccessible to next user
- deleting a file should actually wipe it (not hidden!)

complete mediation

- all accesses must be checked

trusted path

- defend against illusions
- assure that keystrokes and mouse movements are sent correctly

accountability and audit

- log all security-related events
- find good middle-ground for granularity of logs

intrusion detection

- correlated actual behaviour with "normal" behaviour
- alarm if behaviour looks abnormal

3.5.3 Trusted Computing Base

TCB part of the trusted OS that is necessary to enforce OS policies

security kernel runs between OS and hardware maintaining security

rings security level where processes can only access rings that level or above

reference monitor monitor with collection of access controls that must be tamperproof, unbypassable, and analyzable

virtualization way to provide logical separation/isolation

- memory: page mapping to give separate address space
- machines: virtualize I/O, files, printers ...

3.5.4 Least Privelege

chroot sandbox/jail a command by changing its root directory

compartmentalization split application into parts and apply least privelege to each part

setuid bit causes executable to run under identity of owner not caller

- careful of **confused deputy** attack where you convince program another user is executing a setuid'd program

3.5.5 Assurance

Convince others to trust out OS through

- testing
- formal verification
- validation (requirements ...)

Can also have third party evaluate it based on

- **Orange Book** security ratings from DoD
- **common criteria** international effort, protection profiles against security threats and objectives

4 Network Security

4.1 Network Concepts

4.1.1 Overview

Internet has

- decentralized control

- uncontrollable traffic flow through different nodes
- different types of nodes/links

TCP/IP suite covers

- network access: ethernet, wifi ...
- internet: network, IP ...
- transport: TCP, UDP ...
- application: HTTP, FTP, SSL ...

Protocols were based on *trust* and *non-malicious* intent

4.2 Threats

4.2.1 Port Scan

Each application runs on a **port** and unsecure systems (**loose lips**) can reveal what is running on each port, allowing the attacker to target weakest application

- nmap tool can identify application
- login application can reveal OS, version ...

4.2.2 Intelligence

Find information out in the open:

- social engineering
- dumpster diving
- eavesdropping
- google searching
- facebook searching

4.2.3 Wiretapping

Owner of a node can mointor all communication through it

- copper cable
  - inductance allows for eavesdropping without physical contact
  - cable cutting splicing otherwise
- fiber optic
  - no inductance, and splicing is possible but maybe detectable
  - cable "bending"
- satellite

    - signal path is wide so nearby attacker can eavesdrop

- wifi
    - easily intercepted nearby or far away using directed antenna
    - need authentication for basic security
- LAN
    - attacker can un-ignore wrongly delivered packets using a **packer sniffer**

4.2.4 Impersonation

Impersonate by stealing a password

- guessing
- exploit default password
- sniff for password between nodes
- social engineering

and exploit trust beween machines and accounts where `rhosts/rlogin` can allow another use on another machine to act as themselves on their machine without having to enter a password

4.2.5 Spoofing

spoofing when one object masquerades as another

- **url spoofing** uwaterlo.ca
- **evil twin** attacks wifi access points using the same name
- **session hijacking** stealing cookies or session key to hijack
- **man-in-the-middle** attacker becomes stealth intermediate node

4.2.6 Traffic Analysis

Discovering the *existance* of communication between two parties (e.g. whistleblower) can be sensitive, and can be discovered by attacker looking at send and recieve addresses in headers

4.2.7 Integrity Attacks

Modify packets as they are being transmitted

- change payload
- change sender/reciever
- replay previously seen packets ("buy 10 stocks" over and over)
- delete or create packets

- TCP will can use a checksum but that can be easily defeated

Poison DNS cache

- feed wrong mappings to system redirecting them to your address
- DNNSEC created to combat this

4.2.8 Protocol Failures

TCP and other protocols can be attacked directly:

- assumes all nodes implement protocol faithfully
- optional requests of systems to help flow control (can be ignored)
- may not check if packet is well formatted
- may include broken security mechanisms (e.g. WEP)

4.2.9 Website Vulnerabilities

- website defacement
- send malicious URL to web server (to exploit)
- submit modified state (cookie, session id) to exploit incomplete mediation
- code injection

  XSS steal sensitive information on page and send it to attacker

  CSRF perform malicious action on some website (e.g. bank) if user is logged in there

4.2.10 Denial of Service

- cutting a wire, jamming a signal
- **ping flood** overload ability to respond to pings
- **smurf attack** spoof sender of ping as victim, and have return pings all go there
- **syn flood** overflow memory with stored SYN packets
- send packet fragments that can't be assembled
- send packets that are all hashed into same bucket
- **black hole attack** router advertises very low cost for destination and then drops all packets
- **DNS cache poisoning** make packets route to wrong host

4.2.11 Distributed Denial of Service

DDoS makes it difficult to stop the source of the attack by using lots of machines

zombie/bot computer controlled with malware

botnet network of computers used to DDoS

amplification use nodes that respond with more data than they are queried

reflection send queries with victim's address as source

SNMP simple network management protocol on many home routers which is usually vulnerable

Botnets can be controlled in many ways:

- central node: all bots connect and it gives commands
- p2p distributing updates
- fast flux: single host name maps to hundreds of addresses, constantly swap in/out of DNS to make tracking difficult
- domain generation algorithm: generate a large set of domain and contact a tue subset of them for instructions

4.2.12 Active Code

Server can ask client to execute code on its behalf

- Java
- Javascript
- ActiveX
- Flash

Can be dangerous depending on sandboxing, "trusted" source

4.3 Network Security Controls

4.3.1 Segmentation and Separation

- have servers on different machines
- segment by functional and access requirements
- split by vulnerability (inside/outside firewall)

4.3.2 Redundancy

- avoid single points of failure
- different software, genetic diversity
- sync redundant servers closely (so they can easily take over)

4.3.3 Access Controls

- router ACL: drop packets with particular source and address
  - expensive for high traffic
  - difficult to gather logs
  - source addresses can be spoofed
- firewalls: filter based on other critera than addresses

4.4 Firewalls

4.4.1 Overview

- all traffic goes through these choke points, **firewalls**
- carefully examine traffic, possibly refuse it access
- does not protect against attacks from inside

4.4.2 Types

packet filtering gateways make decision based on header of a packet, can drop spoofed traffic

stateful inspection keep state to identify packets that belong together

application proxies app-specific firewall with full knowledge about communication and sophisticated processing

forward proxy accessing server outside the company

reverse proxy accessing company from outside

personal home computer with a whitelist, protect against attacks on servers

Demilitarized Zone subnet containing external services with internal and exteranl firewall

4.5 Honeypots

4.5.1 Overview

- set up unprotected computer as a trap for attacker
- computer should have no activity
- monitor for any activity as sign of break-in
- continue monitoring honeypot to learn about new threats

4.5.2 Types

low interaction daemon that emulates running host(s)

+ easy to install/maintain
- limited amount of information gathered
- easier to detect

high interaction deploy real hardware/software, tracking with stealth network switches and keyloggers

+ can capture lots of information
+ can caputure unexpected behaviour
- complex

4.6 Intrusion Detection Systems

4.6.1 Overview

IDS monitor activity to identify malicious events

- recieve events from sensors
- store and analyze
- take action

4.6.2 Host-based

Run on a host to protect this host

+ can exploit lots of information
- miss out on information of other hosts
- if host gets subverted, so does IDS (single point of failure)

4.6.3 Network-based

Run on a dedicated node to protect all attached hosts

+ difficult to subvert
- has to rely on information available in packets

Distributed IDS combination of host-based and network-based

4.6.4 Signature-based

Detect attack "signatures"

+ may exploits statistical analysis to catch tough stuff
- may fail if attacker modifies attack (polymorphic worms)

4.6.5 Heuristic/Anomaly-based

Look for behaviour that is out of the ordinary

- model good behaviour and alert if system doesn't follow model
- model bad behaviour and alert if system resembles model
- classify as
  - good
  - suspicious
  - unknown
- learn to classify unknown events over file