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

3.1.3 Memory Protection

3.1.4 Segmentation

3.1.5 Paging