Software and Web Security 1

The big problem in computer security

Fairy tales: a problem...

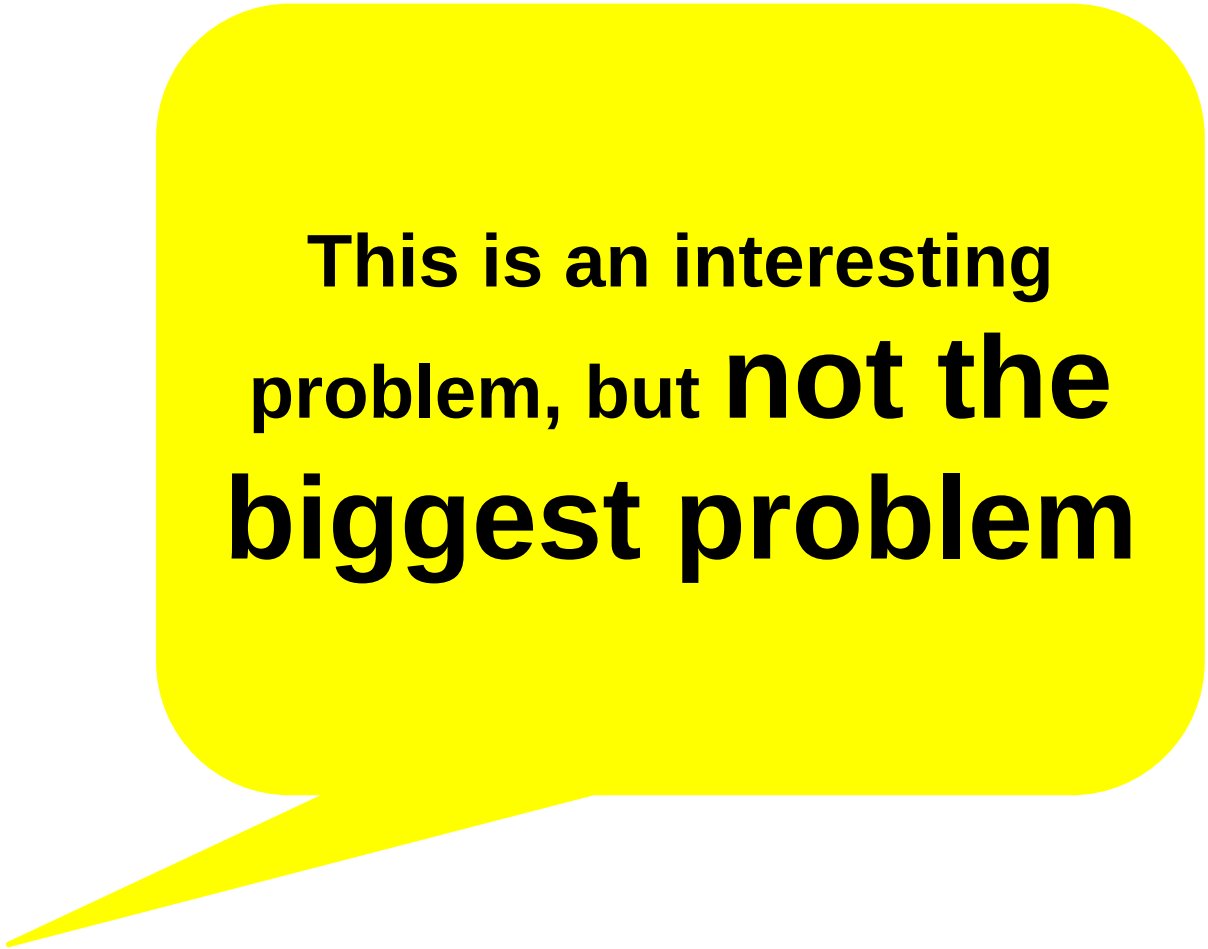
Many discussions of security begin with Alice and Bob

Alice

Eve

Bob

Problem: how can Alice communicate securely with Bob,
when Eve can modify or eavesdrop on the communication?


This is an interesting problem, but not the biggest problem

Fairy tales... a bit more realistic

How can Alice's *computer* communicate securely with Bob's *computer* when Eve can modify or eavesdrop on the communication?

Even if Alice can trust Bob, can she trust his computer?

Reality and the ***big*** problem

Alice's computer is communicating with *some other computer* on the internet

possibly malicious input

Can Alice's computer be *hacked*,
when it communicates with some other computer?

NB solving the first problem - securing the communication - does *not* help here!

Why is this a problem? Why can't we solve it?

- *Why can PCs, laptops, tablets, smartphones, web-sites, servers, routers, printers, smartcards, cars, ATMs .... be hacked?*

  Because there is software inside!

- Software is the most complex artifact mankind has ever created
- The good news:

  software is incredibly powerful & flexible, and shaping the world
- The bad news:

  we are not (yet?) capable of producing software without bugs

- By sending malicious input to software, attackers can try to exploit such bugs

From simple attacks to malware

- You can exploit vulnerabilities in software
  - to simply crash a program
  - to reveal or corrupt some data on that computer
  - to interfere with services offer by that computer

- To do more interesting damage, *you want to get some software running on your victim's computer*.

  malware = software with some malicious intent

  NB here the power & flexibility of software is used *against* us.

software security vulnerabilities

Terminology

Common use of terminology can be very confused & confusing:

(security) weakness, flaw, vulnerability, bug, error, coding defect, ...

One important distinction we can make:

- a security weakness/flaw:
  something that is wrong or could be better
- a security vulnerability
  a weakness/flaw that can actually be exploited by an attacker,
  This requires the flaw to be
  1. *accessible - attacker has to be able to get at it*
  2. *exploitable – attacker has to be able to do some damage with it*

Eg by unplugging your network connection, some vulnerabilities become flaws

Warning: there is no standardised terminology for the distinction above!

design vs implementation flaws

Software vulnerabilities can be introduced at different “levels”

- design flaws
  - fundamental error in the design
- implementation flaws or coding error
  - introduced when implementing

focus of this course

The precise border is not precise: for some flaws it is debatable whether they are design or implementation flaws

Vulnerabilities can also arise on higher levels (out of scope for this course)

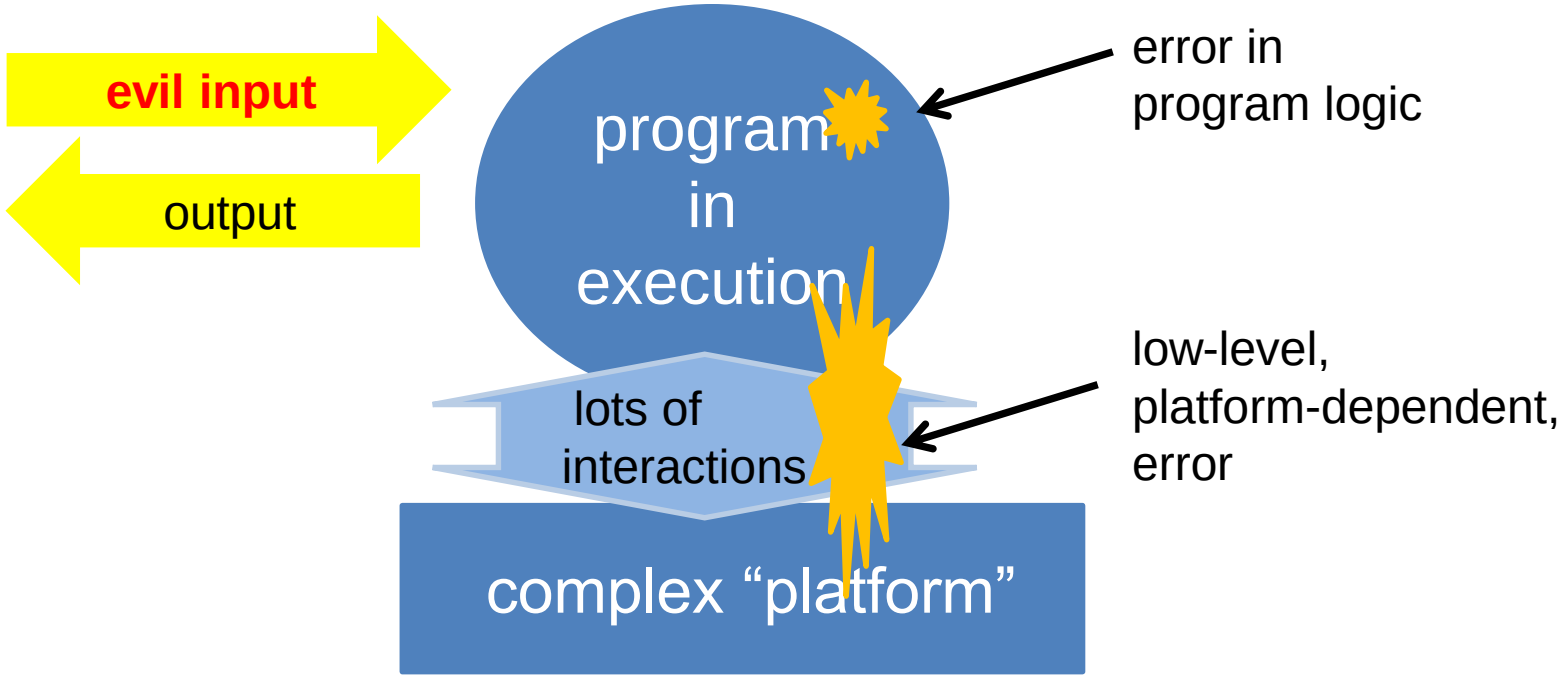
- configuration flaw when installing software on a machine
- the user
- unforeseen consequence of the *intended* functionality (eg. spam)

Different kinds of implementation flaws

1. Some implementation flaws can be spotted just by looking at the program itself (and understanding what it is meant to do!)
   - eg. simple typos, confusing two program variables, off-by-one errors in a for-loop or array access, ...
   - sometimes called logic errors, as opposed to syntax errors, or an error in the program logic

focus of this course

2. Some lower-level implementation flaws can only be spotted if you understand how the underlying platform of the program works.
   - in the case of software running on a normal machine, the platform consists of CPU, OS, memory (RAM+disk), and I/O peripherals
   - in the case of software interacting over the web, this platform is "the web" (IP/HTTP/...) plus on the client side, a web browser and, on the server side, a web application/server (incl. some back-end database)



The platform can be

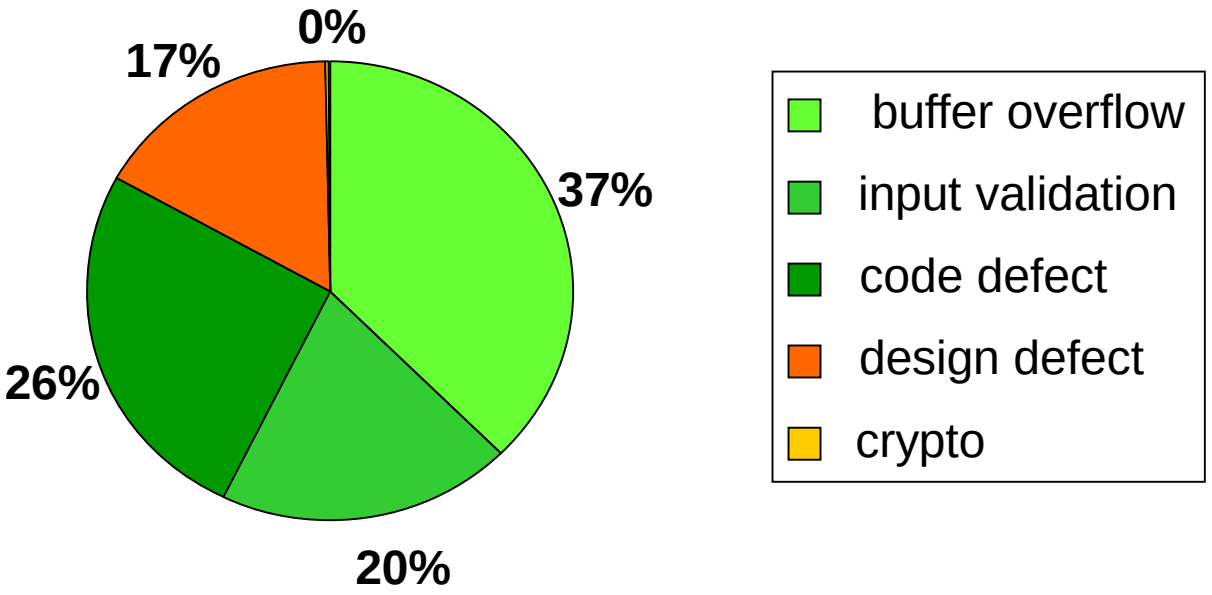
- CPU, OS, memory, peripherals
- web-browser and web-application server

focus of sws 1

focus of sws 2

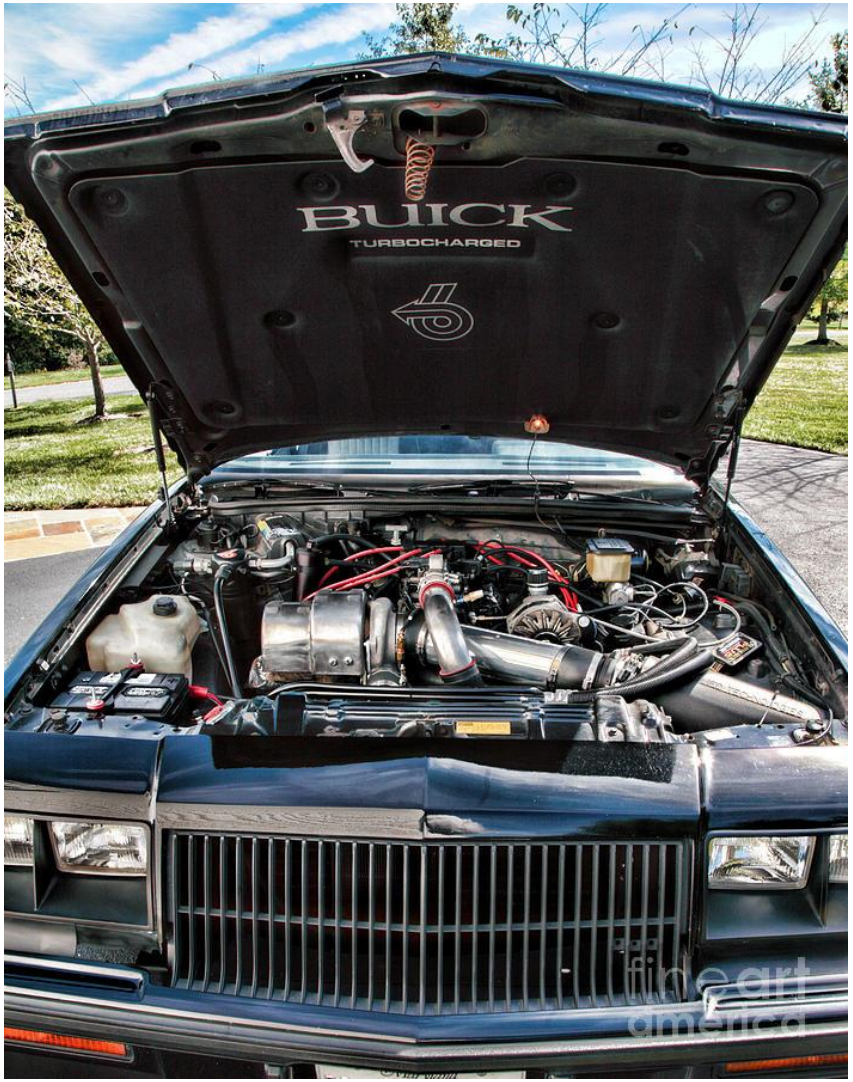
Typical software security vulnerabilities

Security bugs found in Microsoft's first security bug fix month (2002)



Here *buffer overflows* are platform-specific.
Some of the *code defects* and *input validation* problems might also be.
Crypto problems are much rarer, but can be very high impact.

under the hood of a programming language



A programming language is an abstraction layer

- A programming language tries to provide a convenient abstraction layer over the underlying hardware

- So that the programmer does not have to worry about
  - machine instructions of the CPU
  - precisely where in main memory or disk data is allocated
  - how to change some pixels on the screen to show some text
  - ....

CPU


RAM


disk


I/O peripherals

abstraction

In the following piece of code

```
int main(int j){
  printf(”Hello, world\n”);
  printf(”The answer is %i”, 2*j/(6+j));
  return 0; // convention: returning 0 means OK
 }
```

we abstract from

- *how* the data is represented
- *where* in memory (in CPU, RAM or on disk) this data is stored
- *which* machine instructions are executed
- *how* data is printed to the screen

This abstraction is provided the programming language
together with the operating system (OS)

The operating system is responsible for some abstractions, esp.

- memory management
- handling Input and Output (I/O)
  - incl. file system

For I/O, the OS will provide some standard libraries to the programmer, described as part of the programming language specification.

- Eg functions such as **`printf(), fopen(),...`**

Different levels of abstraction for data

1. In programming language we can write a string

   `"hello, world\n"`

   and not care *how* this data is represented or *where* it is stored

2. At a lower level, we can think of memory as a sequence of bytes

					h	e	l	l	o	,		w	o	r	l	d	\n	\0				

3. At the level of hardware, these bytes may be spread over the CPU (in registers and caches), the main memory, and hard disk


There are still lower levels, but then we get into electronics and physics.

Does the programmer have to know how the programming language works under the hood?

- In the *ideal* situation the programmer does *not* need to know how this works
  - *except* to understand the efficiency of programs

- However, for most programming languages, the programmer *does* have to understand this to understand the behaviour of programs under unusual circumstances

  esp. when program is attacked with malicious input

Recap

- The biggest problem is computer security is insecure software
- Security flaws in software can be
  - design flaws
  - implementation flaws

  If they can be exploited, they become vulnerabilites

- Implementation flaws in code can be
  - logic errors, “local” to the program, or
  - lower-level coding defects caused by strange interaction with the underlying platform

    To understand these, we have to look under the hood of the programming language