Course website

Google Classroom code: **ytsbpihr**

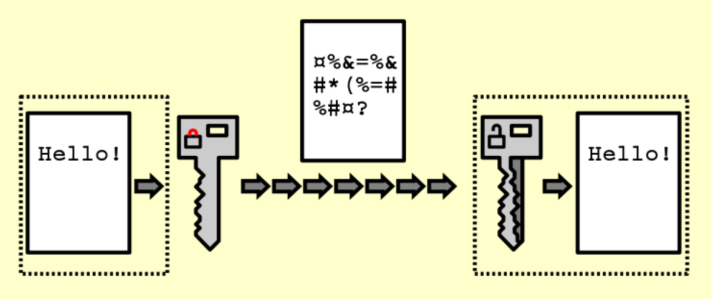
Slides, course info, projects and communications will appear here

whoami

Daniele Friolo

- Ph.D. in **Cryptography**
- Research Interests:
  - Secure Multi-Party Computation
  - Zero-Knowledge
  - Blockchain Applications
  - Advanced Encryption Schemes

Office: Room G24
Viale Regina Elena 295
Hours: By appointment



Security in Software Applications

Introduction to Software Security

SAPIENZA
UNIVERSITÀ DI ROMA

Daniele Friolo

friolo@di.uniroma1.it

https://danielefriolo.github.io

What is Software Security?

Understanding the role that software plays

- in providing *security*
- as source of *insecurity*

Principles, methods & technologies to make software more secure

- Typical *threats* & *vulnerabilities* that make software less secure, and how to avoid them

Why studying Software Security?

Software plays a major role in providing security, and is a major source of security problems.

Software is the weakest link in the security chain, with the possible exception of “the human factor”

Software security does not get much attention

- in other security courses, or
- in programming courses, or indeed, in much of the security literature!

The problem

Sapienza
Università di Roma

The problem

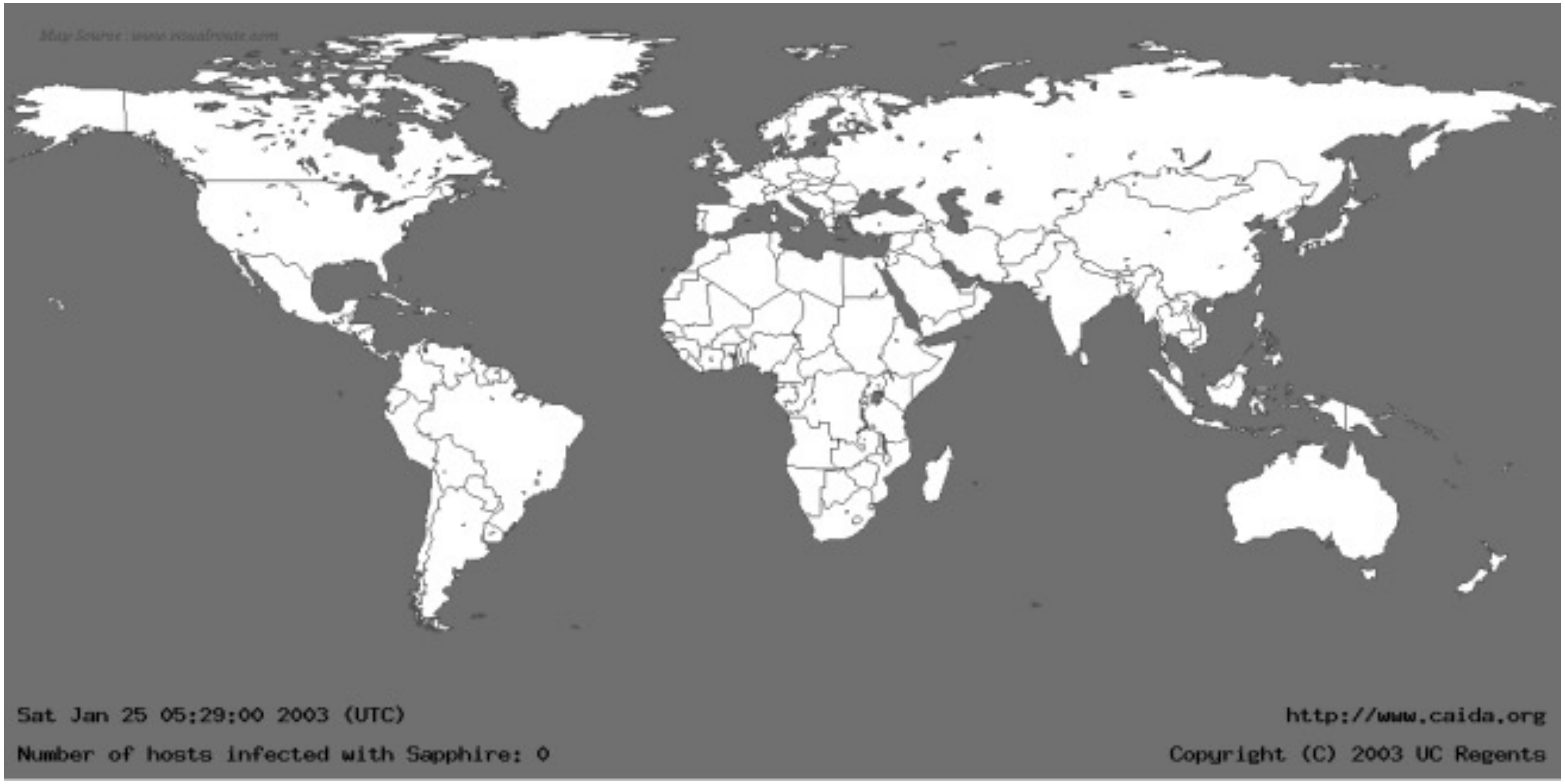
Internet worms and viruses

- **virus** = harmful piece of code that can infect other programs
- **worm** = self-replicating virus; no user action required for spreading infection

First worm: Nov 1988, crashed 10% of internet

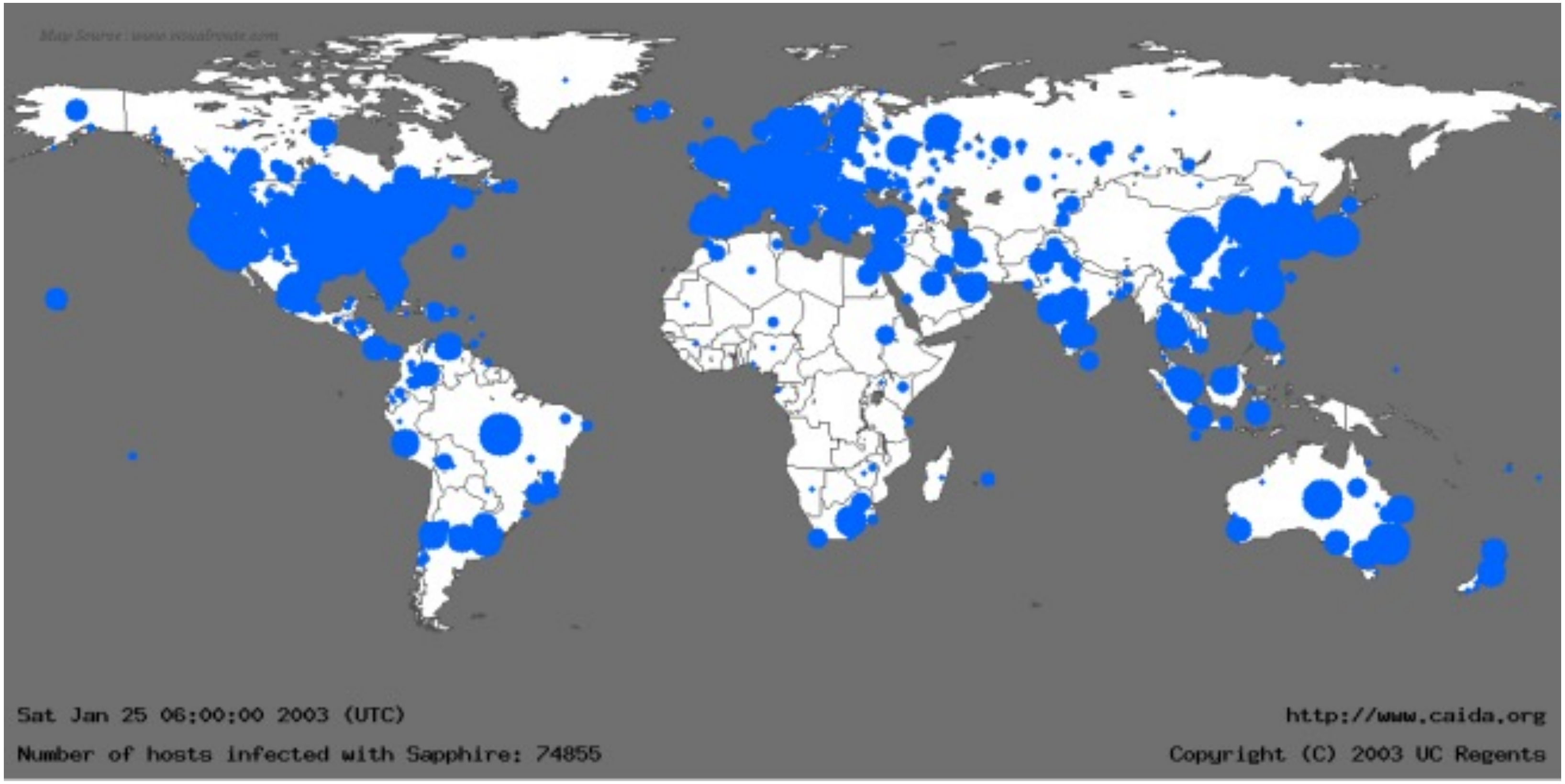
- More recently
  - *email viruses:* I Love You, Kounikova, …
  - *Worms*: Slammer, Blaster, …
- More recently still: attackers have gone, underground & commercial (deep web)

Slammer Worm (Jan 2002)



Pictures taken from *The Spread of the Sapphire/Slammer Worm*, by David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, Nicholas Weaver

Slammer Worm (Jan 2002)



Pictures taken from *The Spread of the Sapphire/Slammer Worm*, by David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, Nicholas Weaver

Famous Vulnerabilities

Cisco Router *(source US-CERT)*

Published: 2011-01-24

Vulnerability No: CVE-2011-0352

CVSS Severity Score: 7.88

Vendor/Product cisco -- linksys_wrt54gc_router_firmware

Buffer overflow in the *web-based management interface* on the Cisco Linksys WRT54GC router with firmware before 1.06.1 allows remote attackers to cause a denial of service (device crash) via a long string in a POST request.

Famous Vulnerabilities

FFmpeg *(source US-CERT)*

Published: 2011-01-24

Vulnerability No: CVE-2010-4705

CVSS Severity Score: 9.3

Vendor/Product: ffmpeg -- ffmpeg

Integer overflow in the vorbis_residue_decode_internal function in libavcodec/vorbis_dec.c in the Vorbis decoder in FFmpeg, possibly 0.6, *has unspecified impact and remote attack vectors*, related to the sizes of certain integer data types. NOTE: this might overlap CVE-2011-0480

Famous Vulnerabilities

Linux/Windows/MACOS

Published: 2011-01-24

Vulnerability : CVE-2011-0638 CVE-2011-0640 CVE-2011-0639

CVSS Severity Score: 9.3

Vendor/Product: Apple Mac OS X/Microsoft – windows/Linux - Linux kernel

```
Microsoft Windows /Mac OS X/ Linux does not properly
warn the user before enabling additional Human Interface
Device (HID) functionality over USB, which allows user-
assisted attackers to execute arbitrary programs via
crafted USB data, as demonstrated by keyboard and mouse
data sent by malware on a smartphone that the user
connected to the computer.
```

Famous Vulnerabilities

Mozilla/Bugzilla

Published: 2011-01-28

Vulnerability : CVE-2010-4568

CVSS Severity Score: 7.5

Vendor/Product: Mozilla - Bugzilla

```
Bugzilla 2.14 through 2.22.7; 3.0.x, 3.1.x, and 3.2.x before
3.2.10; 3.4.x before 3.4.10; 3.6.x before 3.6.4; and 4.0.x
before 4.0rc2 does not properly generate random values for
cookies and tokens, which allows remote attackers to obtain
access to arbitrary accounts via unspecified vectors, related
to an insufficient number of calls to the srand function
```

Famous Vulnerabilities

Tandberg videoconferencing

Published: Feb 2 2011

Vulnerability : CVE-2011-0354

Vendor/Product: Tandberg- video conferencing

```
TANDBERG Videoconferencing Systems Default Account
Lets Remote Users Gain Root Access
The device includes a root administrator account with no
password. A remote user can access the system with root
privileges.
The root user account is used for advanced debugging and is
not required for normal operations.
```

Famous Vulnerabilities

These vulnerabilities seem pretty old, do we need to worry in more modern systems?

Famous Vulnerabilities

These vulnerabilities seem pretty old, do we need to worry in more modern systems?

CVE-2018-13780	The mintToken function of a smart contract implementation for ESH, an Ethereum token, has an integer overflow that allows the owner of the contract to set the balance of an arbitrary user to any value.
CVE-2018-13779	The mintToken function of a smart contract implementation for YLCToken, an Ethereum token, has an integer overflow that allows the owner of the contract to set the balance of an arbitrary user to any value.
CVE-2018-13778	The mintToken function of a smart contract implementation for CGCToken, an Ethereum token, has an integer overflow that allows the owner of the contract to set the balance of an arbitrary user to any value.
CVE-2018-13777	The mintToken function of a smart contract implementation for RRToken, an Ethereum token, has an integer overflow that allows the owner of the contract to set the balance of an arbitrary user to any value.
CVE-2018-13776	The mintToken function of a smart contract implementation for AppleToken, an Ethereum token, has an integer overflow that allows the owner of the contract to set the balance of an arbitrary user to any value.
CVE-2018-13775	The mintToken function of a smart contract implementation for RCKT_Coin, an Ethereum token, has an integer overflow that allows the owner of the contract to set the balance of an arbitrary user to any value.
CVE-2018-13774	The mintToken function of a smart contract implementation for Bitstarti, an Ethereum token, has an integer overflow that allows the owner of the contract to set the balance of an arbitrary user to any value.
CVE-2018-13773	The mintToken function of a smart contract implementation for Enterprise Token Ecosystem (ETE) (Contract Name: NetkillerToken), an Ethereum token, has an integer overflow that allows the owner of the contract to set the balance of an arbitrary user to any value.
CVE-2018-13772	The mintToken function of a smart contract implementation for TheFlashToken, an Ethereum token, has an integer overflow that allows the owner of the contract to set the balance of an arbitrary user to any value.
CVE-2018-13771	The mintToken function of a smart contract implementation for ExacoreContract, an Ethereum token, has an integer overflow that allows the owner of the contract to set the balance of an arbitrary user to any value.
CVE-2018-13770	The mintToken function of a smart contract implementation for UltimateCoin, an Ethereum token, has an integer overflow that allows the owner of the contract to set the balance of an arbitrary user to any value.
CVE-2018-13769	The mintToken function of a smart contract implementation for JeansToken, an Ethereum token, has an integer overflow that allows the owner of the contract to set the balance of an arbitrary user to any value.
CVE-2018-13768	The mintToken function of a smart contract implementation for ZToken, an Ethereum token, has an integer overflow that allows the owner of the contract to set the balance of an arbitrary user to any value.
CVE-2018-13767	The mintToken function of a smart contract implementation for Cornerstone, an Ethereum token, has an integer overflow that allows the owner of the contract to set the balance of an arbitrary user to any value.
CVE-2018-13766	The mintToken function of a smart contract implementation for Easticoin, an Ethereum token, has an integer overflow that allows the owner of the contract to set the balance of an arbitrary user to any value.
CVE-2018-13765	The mintToken function of a smart contract implementation for LandCoin, an Ethereum token, has an integer overflow that allows the owner of the contract to set the balance of an arbitrary user to any value.
CVE-2018-13764	The mintToken function of a smart contract implementation for BiquToken, an Ethereum token, has an integer overflow that allows the owner of the contract to set the balance of an arbitrary user to any value.

https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=smart+contract

Superficial analysis

SAPIENZA
UNIVERSITÀ DI ROMA

Observation 1

All these problems are due to (bad) software

Namely,

- the Linux/Windows/Mac Operating System (OS)
- the router software
- the videoconferencing system software
- the FFmpeg graphics engine
- ...

Such software bugs are why constant patching of system is needed to keep them secure

Observation 2

All these problems are due to (bad) software that

- can be executed over the network (eg. *Slammer worm*)
- executes on (untrusted) input obtained over the network (eg. *Ffmpeg*)

With ever more network connectivity, ever more software can be attacked.

Observation 2

Traditionally, focus on operating system and network security

- regular patching of OS
- firewalls, virus scanners

Increasingly, web applications and web browser are weak link and obvious targets to attack

Also, mobile devices and embedded software are targeted.

- Traditional distinction between OS, network, and application gradually disappearing anyway.

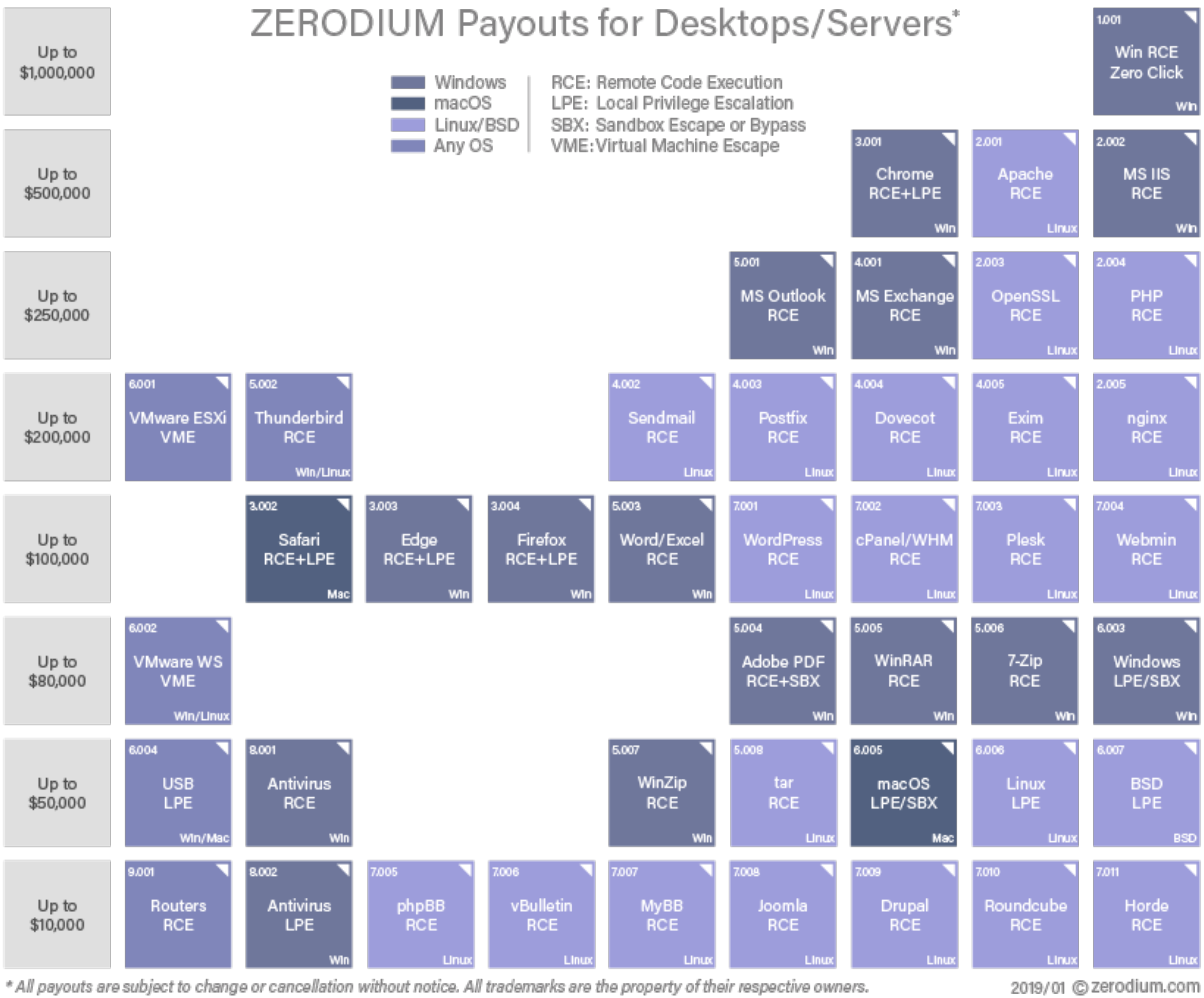
Changing the nature of attackers

- Traditionally, hackers are amateurs motivated by fun, publishing attacks for the prestige
- Increasingly, hackers are professional
  - attackers go underground: zero-day exploits are worth money
- attackers include
  - organized crime with lots of money and (hired) expertise
- government agencies: with even more money & in-house expertise

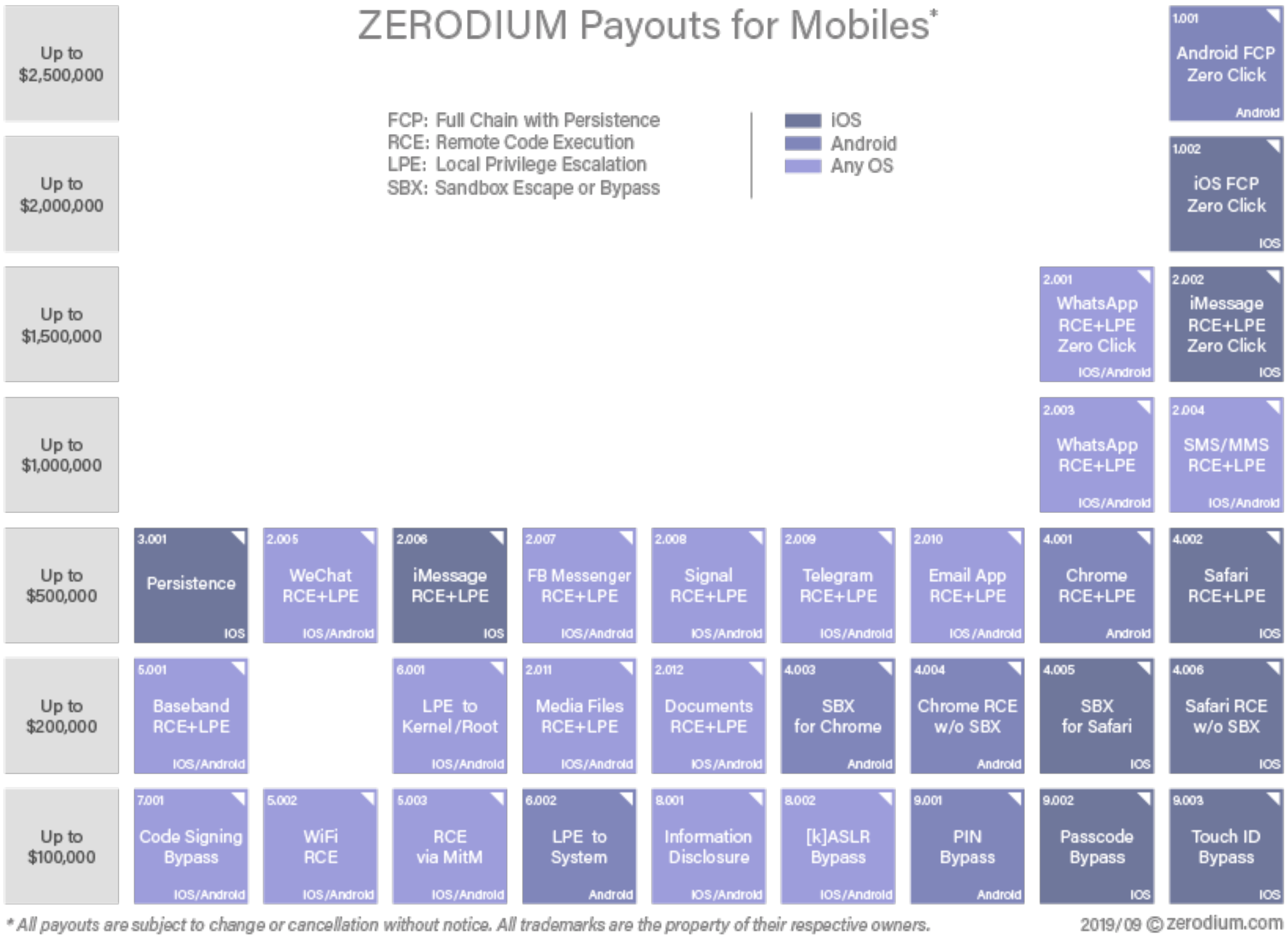
  'Classic' example: *Stuxnet*

  https://www.ted.com/talks/ralph_langner_cracking_stuxnet_a_21st_century_cyber_weapon

Current 0day prices



Current 0day prices



Causes of the problem

SAPIENZA
UNIVERSITÀ DI ROMA

Quick poll

how many of you learned to program in C or C++?

Quick poll

how many of you learned to program in C or C++?

- how many of you have built a web-application?

– in which languages?

Quick poll

how many of you learned to program in C or C++?

- how many of you have built a web-application?

– in which languages?

Major causes of problems are

- lack of *awareness*
- lack of *knowledge*

Functionality vs Security

Security is secondary concern

- Security is always a secondary concern
- primary goal of software is to provide some functionality or services; managing associated risks is a derived/secondary concern

Functionality vs Security

Security is secondary concern

- Security is always a secondary concern
- primary goal of software is to provide some functionality or services; managing associated risks is a derived/secondary concern

There is often a trade-off/conflict between

- security
- functionality & convenience

where security typically looses out

Functionality vs Security

Security is secondary concern

- Security is always a secondary concern
- primary goal of software is to provide some functionality or services; managing associated risks is a derived/secondary concern

There is often a trade-off/conflict between

- security
- functionality & convenience

where security typically looses out

However, due to recent attacks due to wrong usage of smart contracts code (causing loss if billions of dollars), awareness in software security is increasing

(at least for blockchains and the underlying VMs...)

Functionality vs Security

Functionality is about what an application **does.**

Security is about what an application should **not** do.

Unless you think like an attacker, you will be unaware of any potential threats.

Functionality vs Security

Lost battles?

- operating systems
  - huge OS, with huge attack surface (API).
- programming languages
  - buffer overflows, format strings, ... in C
  - public fields in Java
  - ...
- web browsers
  - plug-ins for various formats, javascript, ajax, ...
- email clients

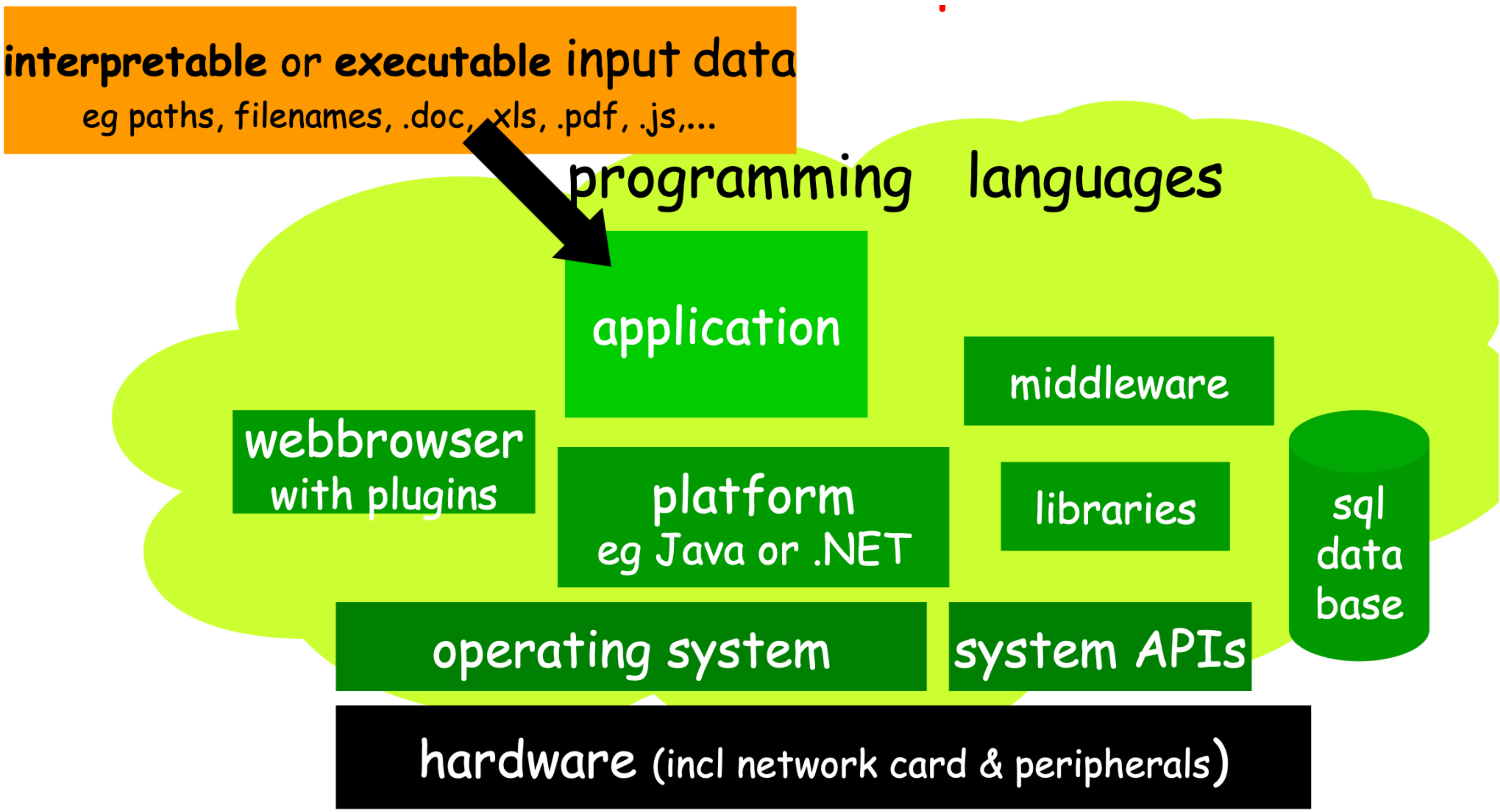
Some programming languages are not designed well enough to minimize the developer's burden of assessing security.

- A desperate case: PHP

First Step: Awareness

- that there might be a problem
- of what needs protecting, from which threats
- of the fact that you might lack knowledge

Weakness in depth



Weakness in depth

Software

- runs on a huge, complicated infrastructure
  - OS, platforms, webbrowser, lots of libraries & APIs, …
- is built using complicated languages
  - programming languages, but also SQL, HTML, XML, …
- using various tools
  - compilers, IDEs, preprocessors, dynamic code downloads

These may have security holes, and may make the introduction of security holes very easy & likely

Flaw or Vulnerability?

Confusing terminology

Security weakness, flaw, vulnerability, bug, error, coding defect ...

Important distinction

- Security weakness / flaw:
  - Something that is wrong or could be better ...
- Security vulnerability
  - Flaw that can be exploited by an attacker to violate a policy

So, a flaw must be

- Accessible: an attacker must have access to it
- Exploitable: an attacker must be able to use it to compromise system

Software Flaws

Software flaws can be introduced at two levels

1. Design flaw – the flaw is introduced during the design
2. Bug / code-level flaw - the flaw is introduced during implementation

Equally common

Vulnerabilities can also arise from other levels

- Configuration flaws – when installing the SW
- “User” flaws
- Unforeseen consequences of intended functionality (e.g., spam ...)

Coding Flaws

Software flaws can be introduced during implementation can be roughly distinguished into

1. Flaws that can be understood by looking at the program
e.g.; typos, confusing program variables, off-by-one, access to array, error in program logic, ...

2. Flaws due to the interaction with the underlying platform or with other systems

- Buffer overflow in C(++) code
- Integer overflow/underflow in most programming languages
- SQL injection, XSS, CSRF, ... in web applications

Spot the security flaws

```
int balance;

void decrease(int amount)
{     if (balance <= amount)
            { balance = balance – amount; }
      else { printf("Insufficient funds\n"); }
}

void increase(int amount)
{     balance = balance + amount;
}
```

Spot the security flaws

```
int balance;

void decrease(int amount)
{     if (balance <= amount)
            { balance = balance – amount; }
      else { printf("Insufficient funds\n"); }
}

void increase(int amount)
{     balance = balance + amount;
}
```

should be >=

what if amount is negative?

what if the sum is too large for an int ?

Spot the security flaws

should be >)

1. Logic error
Can be found by code inspection only

what if amount is negative?

2. Lack of input validation of (untrusted) user
Design flaw or implementation flaw ?

what if the sum is too large for a 64 bit int ?

3. Problem with interaction with underlying platform.
lower level than previous ones

Tackling software security

To prevent standard mistakes, knowledge is crucial

- mistakes may depend on the programming language, on the platform (Op.Sys., DB, Web app, …) and on the (type of) application

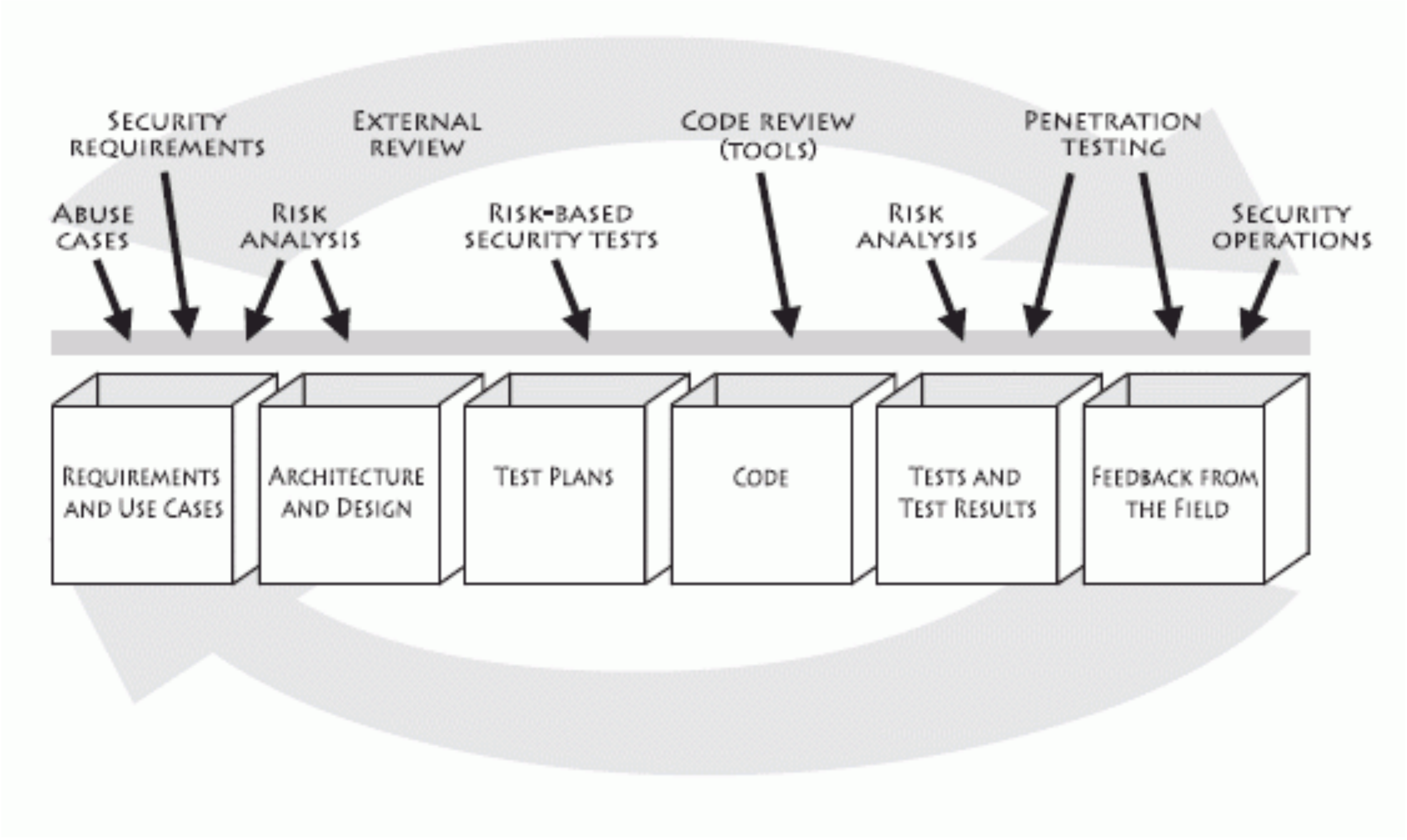
but knowledge alone is not enough: security must be taken into account from the beginning and throughout the software development life cycle

Evolution in tackling software security

Organizations tackle security at the end of the SDLC and with time have moved the concern to earlier stages for example, chronologically:

1. First, *do nothing*
    - Some problem may happen and then we patch
2. then implement support for *regular patching*
3. Products are *pen-tested* pre-emptively
4. Use *static analysis* tools on code produced
5. then *train* programmers to know about common problems
6. then think about *abuse cases* and develop security test for them
7. then start thinking about security *before* starting the development

Security in Software Development Life Cycle



Security concepts

SAPIENZA
UNIVERSITÀ DI ROMA

Security and software

- Security is about regulating access to assets

– eg. information or functionality

- Software provides functionality

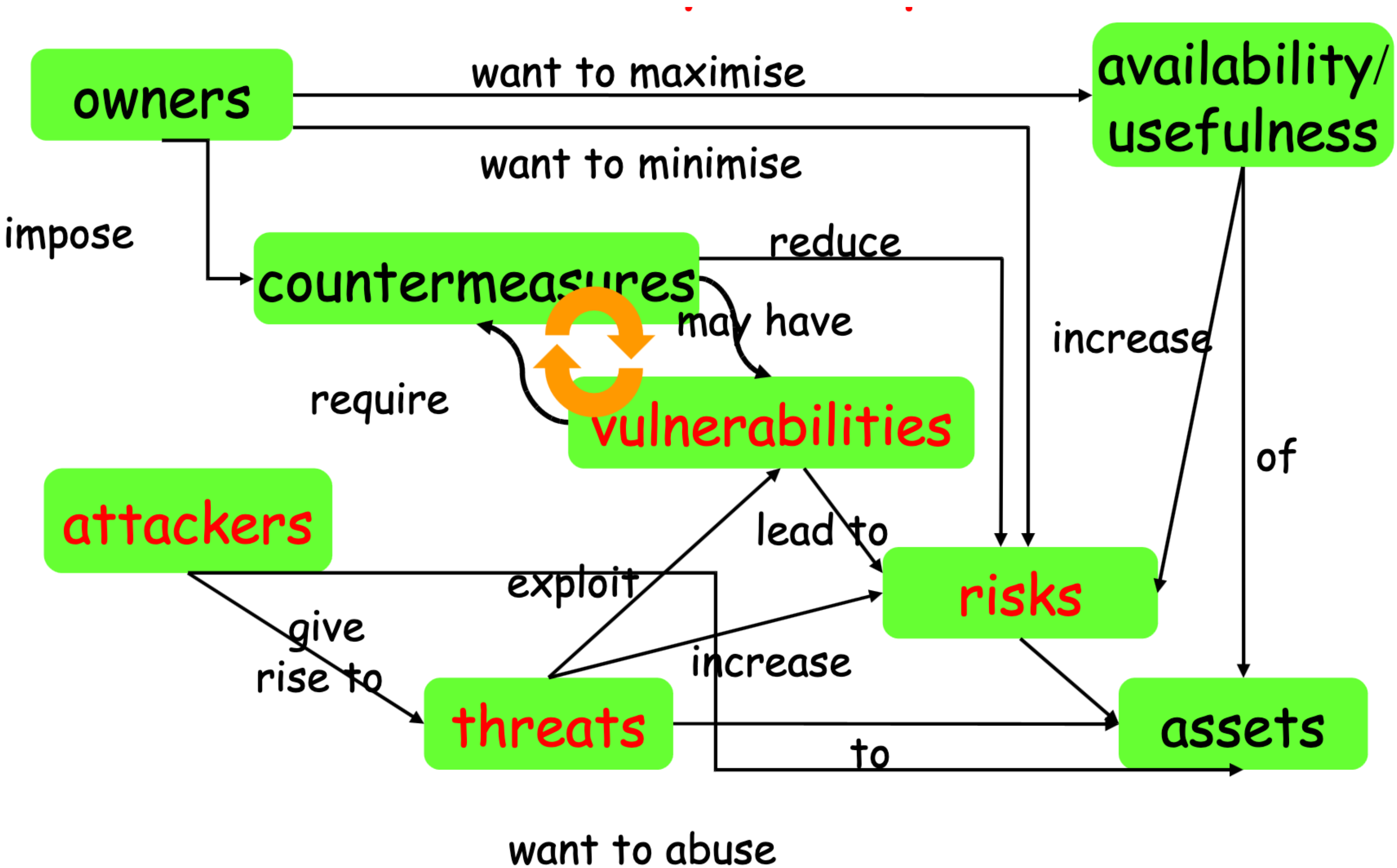
– eg on-line exam results

- This functionality comes with certain risks

– eg what are risks of on-line exam results?

- Software security is about managing these risks

Security concepts



Security concepts

Any discussion of security should start with an inventory of

- the *stakeholders*,
- their *assets*, and
- the *threats* to these assets by possible attackers
- employees, clients, script kiddies, criminals

Any discussion of security without understanding these issues is *meaningless*

Security concepts

Security is about imposing countermeasures to reduce risks to assets to acceptable levels

- A *security policy* is a specification of what *security requirements/goals* the countermeasures are intended to achieve
  - secure against what and from whom ?
- *Security mechanisms* to enforce the policy

- Bottlenecks:
  - expressing what we (don't) want in a policy
  - enforcing this, dynamically or statical

Security Objectives: CIA

- **Confidentiality**
  - unauthorized users cannot *read* information
- **Integrity**
  - unauthorised users cannot *alter* information
- **Availability**
  - authorised users can *access* information

But also

- **Non-repudiation** (for accountability)
  - authorised users cannot deny actions

Security Objectives

Integrity nearly always more important than confidentiality

E.g. think of

- your bank account information
- your medical records
- all your software, incl. entire OS

Availability may be undesirable for privacy

- you want certain data to be or become unavailable

Security goals

The well-known trio

- *confidentiality, integrity, authentication* (CIA)

but there are more "concrete" goals

- *traceability* and *auditing* (forensics)
- *monitoring* (real-time auditing)
- *multi-level security*
- *privacy* & *anonymity*
- ...

and meta-property

- *assurance* – that the goals are me

How to realise security objectives? AAAA

- **Authentication**
  - who are you?
- **Access control/Authorisation**
  - control who is allowed to do what
  - this requires a specification of who is allowed

to do what

- **Auditing**
  - check if anything went wrong
- **Action**
  - if so, take action

How to realise security objectives?

Other names for the last three A's

- **Prevention**
  - measures to stop breaches of security goals
- **Detection**
  - measures to detect breaches of security goals
- **Reaction**
  - measures to recover assets, repair damage, and persecute (and deter) offenders

NB don't be tempted into thinking that good prevention makes detection & reaction superfluous.

Eg. breaking into any house with windows is trivial; despite this absence of prevention, detection & reaction still deter burglars.

Threats vs security requirements

Threats vs security requirements

- information disclosure
  - *confidentiality*
- tampering with information
  - *integrity*
- denial-of-service (DoS)
  - *availability*
- spoofing
  - *authentication*
- unauthorised access
  - *access control*

Countermeasures

Countermeasures can be non-IT related

- physical security of building
- screening of personnel
- legal framework to deter criminals
- police to catch criminals
- ...

but we won't consider these

Also, they can lead to new vulnerabilities

- eg. if we only allow three incorrect logins, as a countermeasure to brute-force password guessing attacks, which new vulnerability do we introduce?

If a countermeasure relies on software, bugs in this software may mean that it is ineffective, or worse still, that it introduces more weaknesses

Countermeasures

Security technologies we can use

- cryptography
  - for threats related to insecure communication and storage
- access control
  - for threats related to misbehaving users
    e.g. role-based access control
- language-based security
  - for threats related to misbehaving programs
    - *typing*, memory-safety
    - *sandboxing*
    - eg Java, .NET/C#

Security technologies

May be provided by the infrastructure/platform an application builds on, for instance

- networking infrastructure
  - which may eg. use SSL
- operating system or database system
  - providing eg. access control
- programming platform
  - for instance Java or .NET sandboxing

Of course, software in such infrastructures implementing security has to be secure

Software infrastructure

Applications are built on top of "infrastructure" consisting of

- operating system
- programming language/platform/middleware
  - programming language itself
    - interface to CPU & RAM
  - libraries and APIs
    - interface to peripherals
    - provider of building blocks
- other applications & utilities
  - eg database

This infrastructure provides security mechanisms,
but is also a source of insecurity

Threats & vulnerabilities

- Knowledge about threats & vulnerabilities crucial

- Vulnerabilities can be specific to programming language, operating system, database, the type of application… and are continuously evolving
  - we cannot hope to cover all vulnerabilities in this course

- “Fortunately”, people keep making the same mistakes and some old favourites never seem to die,
  - esp. public enemy number 1: *the buffer overflow*

and some patterns keep re-emerging

Sources of software vulnerabilities

- Bugs in the application or its infrastructure
  - ie. doesn't do what it should do
- Inappropriate features in the infrastructure
  - ie. does something that it shouldn't do
    - functionality winning over security

- Inappropriate use of features provided by the infrastructure.
  Main causes:
  - complexity of these features
  - functionality winning over security, again
  - ignorance of developers

Topics in rest of this course

- Awareness & knowledge of vulnerabilities (**don'ts**)
  - general (input validation, ...)
  - specific to a kind of application (SQL injection, XSS, ...), or
  - specific to a kind of programming language (buffer overflows, ...)

- Awareness & knowledge of countermeasures (**do's**)
  - at different points in the development lifecycle
  - at level of application, programming language, or platform

  E.g. security technologies (static or dynamic) such as
  - *access control*
  - *untrusted code security*
    - type-safe languages, sandboxing, code-based access control
  - *runtime monitoring*
  - program analyses: typing, static analysis, verification, information flow

About the Exam

The *exam* is divided into

- 1 or 2 individual projects to be done during the course: 20/30% of the final grade
- Written exam: 70/80% of the final grade
- (optional) Oral exam (after the written one)

- **Projects** require the usage of:
    - Static/Dynamic Analysis tools
    - Annotation/Property-based tools
    - Fuzzing tools
    - …
- Might require an additional report