Mestrado em Engenharia Informática (MEI)
Mestrado Integrado em Engenharia Informática (MiEI)

Perfil de Especialização **CSI** : Criptografia e Segurança da Informação

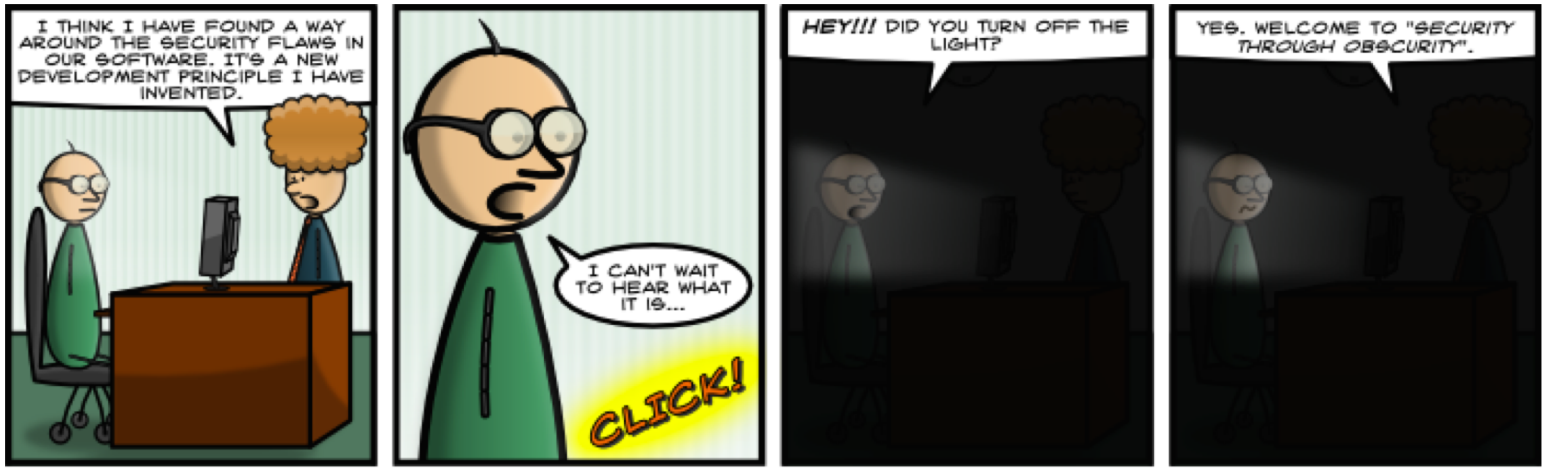
Engenharia de Segurança

jose.miranda@devisefutures.com

Topics

- Software Security

(Bibliography: Segurança no Software, Miguel Pupo Correia, Paulo Jorge Sousa,
Security Software: Building security in, Gary McGraw)



Software Security

It is far harder to write solid code than to destroy it (OWASP DevGuide)

- Relevance
  - Attacks on financial, medical, government, and critical infrastructure systems have increased in number and severity;
  - With digital infrastructures increasingly complex and interconnected, the difficulty of providing secure systems increases exponentially;
  - Easier to explore errors.

- Objective
  - Teach robust programming, or how to write reliable programs, in the sense that programs perform coded tasks and deal with unexpected inputs or events in a reasonable way;
  - Know and realize the importance of secure programming principles;
  - Prepare future software professionals to design and implement software that is guided by these principles.

Software Security

- Myths
  - If students learn to write secure programs, the security state of software and computer systems will dramatically improve.
    - Programs depend on operating systems, APIs / third-party libraries, and other external services;
    - Security of a computer system depends on being created and configured according to the requirements of the particular environment in which it is used - the gap between the required theoretical configuration and the usual practice is unknown, but probably large;
    - It is not clear whether *software houses* are willing to pay the price associated with secure coding, or whether customers are willing to pay higher prices and support longer development times.
  - Universities know what and how to teach secure programming
    - Universities have ideas and a lot of theory, but they do not know what will work and when. Secure programming relies heavily on programmers, teamwork and company culture.

Software Security - Concepts

- **Security** purpose is to ensure the following properties:
  - **confidentiality** – no unauthorized disclosure of information;
  - **integrity** – no unauthorized changes to the system or information;
  - **availability** – readiness of the information to be accessed or, of the system to provide a service;
  - **authenticity** – Information or service provided is genuine.

Software Security - Concepts

- **Security** purpose is to ensure the following properties:
  - **confidentiality** – no unauthorized disclosure of information;
  - **integrity** – no unauthorized changes to the system or information;
  - **availability** – readiness of the information to be accessed or, of the system to provide a service;
  - **authenticity** – Information or service provided is genuine.

Privacy is the confidentiality of personal data (name, phone, address, ...).

Software Security - Concepts

- **Security** purpose is to ensure the following properties:
  - **confidentiality** – no unauthorized disclosure of information;
  - **integrity** – no unauthorized changes to the system or information;
  - **availability** – readiness of the information to be accessed or, of the system to provide a service;
  - **authenticity** – Information or service provided is genuine.

The definition of what is or is not allowed, i.e. the set of security requirements that the system must satisfy, is included in the **security policy**.

Software Security - Concepts

- **Vulnerability** is a system bug (software or other) that can be exploited by an attacker to violate the security policy.
- Software vulnerabilities can be classified in three categories:
  - **project vulnerability** – introduced during the software design phase (design and requirements gathering);
  - **implementation vulnerability** – introduced during software development, i.e., a bug with security implications;
  - **operational vulnerability** – caused by the environment in which the software runs or by its configuration.

Software Security - Concepts

- **Vulnerability** is a system bug (software or other) that can be exploited by an attacker to violate the security policy.
- Software vulnerabilities can be classified in three categories:
  - **project vulnerability** – introduced during the software design phase (design and requirements gathering);
  - **implementation vulnerability** – introduced during software development, i.e., a bug with security implications;
  - **operational vulnerability** – caused by the environment in which the software runs or by its configuration.

Software security concerns to these three types of vulnerabilities.

Software Security - Concepts

- **Attack** is a malicious action that aims to activate one or more vulnerabilities in order to subvert the security policy of the system.
- The attack is successful (**intrusion**) when one or more vulnerabilities are activated.
- **Exploit** is a piece of code capable of activating a vulnerability in a software package.
  - Exploit is also used as a verb to denote the action of performing an attack

attack + vulnerability(ies) ➡ **intrusion**

Implementation vulnerability

- Implementation vulnerability has its own lifecycle - When a software package begins to be sold it contains numerous **bugs**.
- It is estimated that any software package has an average of 5 to 50 bugs per 1.000 SLOC (*Source Lines Of Code*, excluding comment lines), some of which are vulnerabilities.
  - Upper limit (50 bugs per 1,000 LOC) for normal software;
  - Lower limit (5 bugs per 1,000 LOC) for software developed using rigorous development methods.

Software Package	Files	SLOC
Windows XP (2001)	n/a	45 milhões (1)
MacOSX 10.4 (2005)	n/a	86 milhões(1)
Linux kernel pre4.2 (2015)	n/a	20,2 milhões(1)
MariaDB 10.1	5.014	2.127.699(2)
PostgreSQL 9.4.4	3.864	1.698.471(2)
Python 2.7.10	3.240	998.697(2)
Perl 5.22.0	3.434	903.874(2)

(1) https://en.wikipedia.org/wiki/Source_lines_of_code
(2) https://github.com/aldanial/cloc

Implementation vulnerability

- Implementation vulnerability has its own lifecycle - When a vulnerability is discovered, the company that created the software must create a **patch** (patch / fix).
- Patch must be installed by administrators or users of the software in order to remove the vulnerability.
- Negative aspects of the patch:
  - Publishing a patch is a public declaration that the product has problems;
  - Publishing a patch allows attackers to discover the vulnerability(ies) the patch resolves, and create exploits that take advantage of the vulnerability(ies). What is the purpose?
  - Installing a patch has administration costs;
  - Installing a patch can resolve a vulnerability, but may give rise to other vulnerability(ies).

- To minimize these effects, some software vendors publish a patch at fixed intervals, resolving various vulnerabilities (except very critical vulnerabilities, for which they publish immediate patches).
- Other vendors (as is normal for mobile devices) provide new versions of software with new features (and fixing vulnerabilities).

Vulnerability

- Vulnerabilities may not be known in the computer security community, but known in a restricted environment. What is the purpose?
  - group of hackers (for attacks or sale on the *dark web*),
  - military forces of one country (for cyber-weapons).
- Usually referred to as **zero-day vulnerability**.

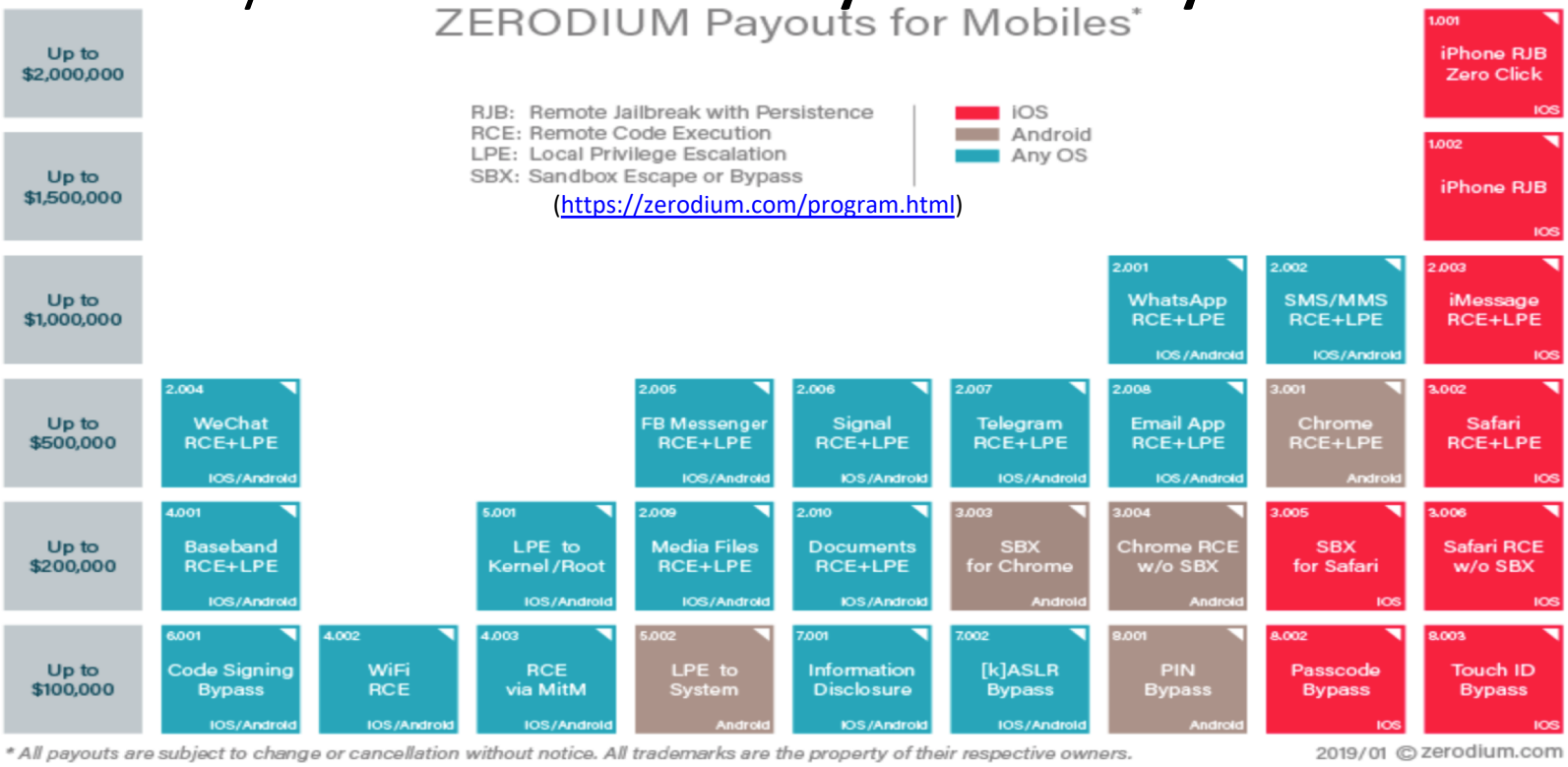
Zero-day Attacks – Allows to attack systems managed by skilled teams with good security knowledge.

Pos	$25k-$100k	Link	Trend	Today	0-day
1	Microsoft Windows DLL Loader privilege escalation [CVE-2019-5921]	✗	□	$25k-$100k	$100k and more
2	Tesla Model 3 Entertainment System Code Execution [CVE-2019-9977]	✗	□	$25k-$100k	$25k-$100k
3	Cisco IOS/IOS XE Secure Storage privilege escalation [CVE-2019-1762]	✗	□	$25k-$100k	$25k-$100k

(https://vuldb.com/?exploits.top)

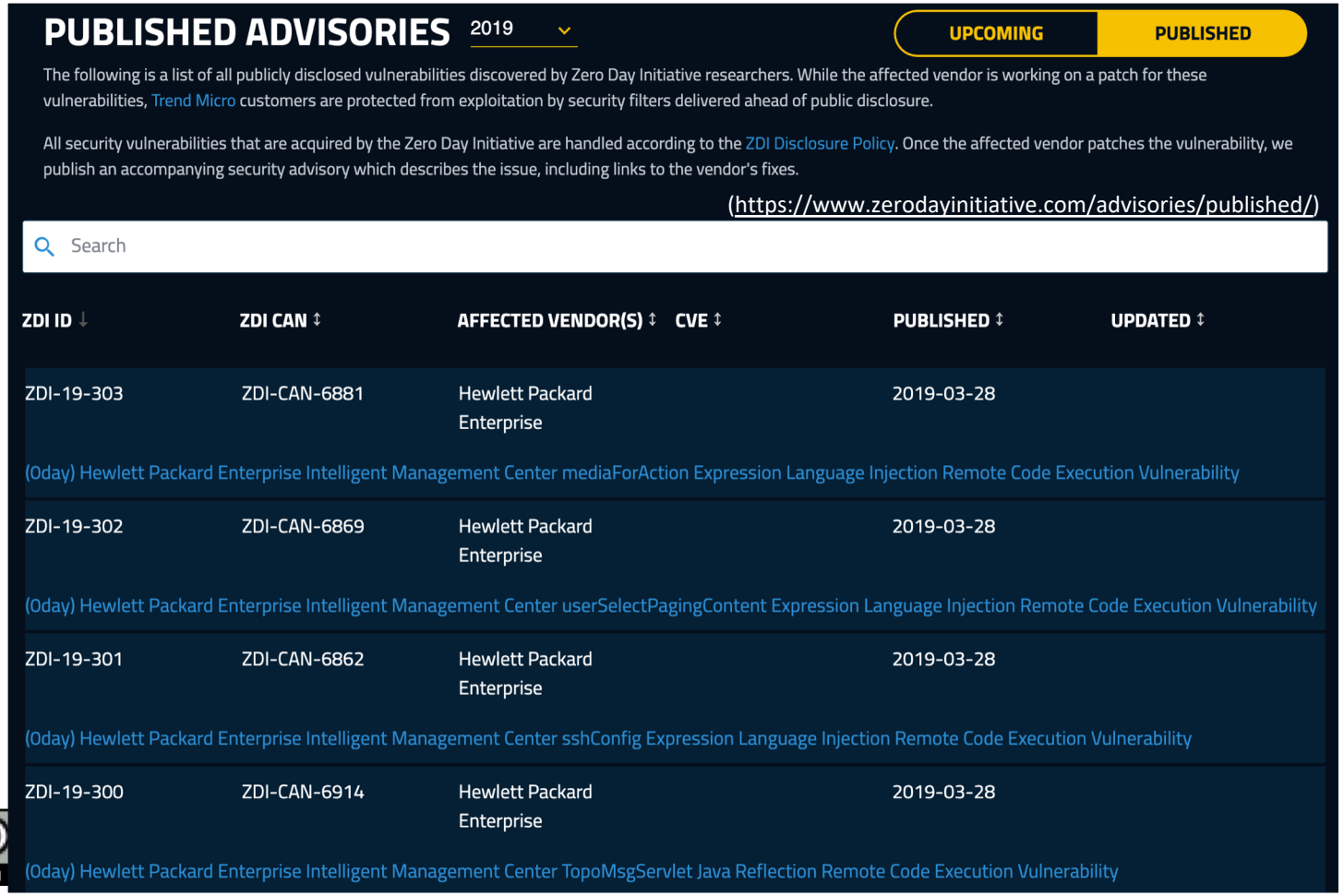
Vulnerability

- Vulnerabilities may not be known in the computer security community, but known in a restricted environment. What is the purpose?
  - group of hackers (for attacks or sale on the *dark web*),
  - military forces of <u>one</u> country (for cyber-weapons).
- Usually referred to as **zero-day vulnerability**.



Vulnerability

- When a vulnerability is made public, it becomes usable to attack computing systems.
  - Why is not the publication of vulnerabilities prevented by legislation?
  - What is the ethical way to publish a vulnerability?



PUBLISHED ADVISORIES 2019 — UPCOMING | PUBLISHED

The following is a list of all publicly disclosed vulnerabilities discovered by Zero Day Initiative researchers. While the affected vendor is working on a patch for these vulnerabilities, Trend Micro customers are protected from exploitation by security filters delivered ahead of public disclosure.

All security vulnerabilities that are acquired by the Zero Day Initiative are handled according to the ZDI Disclosure Policy. Once the affected vendor patches the vulnerability, we publish an accompanying security advisory which describes the issue, including links to the vendor's fixes.

(https://www.zerodayinitiative.com/advisories/published/)

Search

ZDI ID	ZDI CAN	AFFECTED VENDOR(S)	CVE	PUBLISHED	UPDATED
ZDI-19-303	ZDI-CAN-6881	Hewlett Packard Enterprise		2019-03-28	
(Oday) Hewlett Packard Enterprise Intelligent Management Center mediaForAction Expression Language Injection Remote Code Execution Vulnerability					
ZDI-19-302	ZDI-CAN-6869	Hewlett Packard Enterprise		2019-03-28	
(Oday) Hewlett Packard Enterprise Intelligent Management Center userSelectPagingContent Expression Language Injection Remote Code Execution Vulnerability					
ZDI-19-301	ZDI-CAN-6862	Hewlett Packard Enterprise		2019-03-28	
(Oday) Hewlett Packard Enterprise Intelligent Management Center sshConfig Expression Language Injection Remote Code Execution Vulnerability					
ZDI-19-300	ZDI-CAN-6914	Hewlett Packard Enterprise		2019-03-28	
(Oday) Hewlett Packard Enterprise Intelligent Management Center TopoMsgServlet Java Reflection Remote Code Execution Vulnerability					

Vulnerability

- When a vulnerability is made public, it becomes usable to attack computing systems.
  - Why is not the publication of vulnerabilities prevented by legislation?
  - What is the ethical way to publish a vulnerability?

March 26th, 2019 (https://www.zerodayinitiative.com/advisories/ZDI-19-290/)

Apple macOS SCSITaskUserClient Out-Of-Bounds Write Privilege Escalation Vulnerability

ZDI-19-290
ZDI-CAN-7889

CVE ID	CVE-2019-8529
CVSS SCORE	7.8, (AV:L/AC:H/PR:L/UI:N/S:C/C:H/I:H/A:H)
AFFECTED VENDORS	Apple
AFFECTED PRODUCTS	macOS
VULNERABILITY DETAILS	This vulnerability allows local attackers to escalate privileges on vulnerable installations of Apple macOS. An attacker must first obtain the ability to execute low-privileged code on the target system in order to exploit this vulnerability. The specific flaw exists within the SCSITaskUserClient module. The issue triggers a write past the end of an allocated data structure. An attacker can leverage this vulnerability to escalate privileges and execute code in the context of the kernel.
ADDITIONAL DETAILS	Apple has issued an update to correct this vulnerability. More details can be found at: https://support.apple.com/en-us/HT209599
DISCLOSURE TIMELINE	2019-01-21 - Vulnerability reported to vendor 2019-03-26 - Coordinated public release of advisory
CREDIT	Juwei Lin(@panicall) of TrendMicro Research

Vulnerability

- When a vulnerability is made public, it becomes usable to attack computing systems.
  - Why is not the publication of vulnerabilities prevented by legislation?
  - What is the ethical way to publish a vulnerability?
- Vulnerability Cataloging
  - *Common Weakness Enumeration* (**CWE**) – classification of vulnerability classes, in which an identifier with the format CWE-NNNN is assigned to each class of vulnerabilities, where NNNN is the number assigned to the class
  - *Common Vulnerabilities and Exposures* (**CVE**) – catalog of vulnerabilities (design and implementation vulnerabilities) existing in commercial or open software, with identifier format CVE-AAAA-NNNN, being AAAA the year in which the vulnerability was cataloged and NNNN its number.
  - *Common Configuration Enumeration* (**CCE**) – catalog of operational (or configuration) vulnerabilities existing in commercial or open software, with identifier format CCE-NNNNN-M, NNNNN being its number, and M used to verify the integrity of NNNNN [deprecated].
  - *Open Sourced Vulnerability Database* (**OSVDB**) – open source initiative similar to CVE [deprecated].

Managed by MITRE Corporation, and sponsored by NIST that discloses them under the designation of National Vulnerability Database (**NVD**).

Vulnerability

- When the exploit of a vulnerability is made public, the danger is immediate.
- Making an attack after the exploit is known, tends to be simple.
- Exploit catalogs
  - *Exploit Database* – archive of public exploits, identifying the CVE of the vulnerability and/or software it exploits, for use by vulnerability researchers and penetration testers.
  - *Google Hacking Database* – Google dorks (search query that returns sensitive information) archive that although being a form of exploits, are also used to create new exploits.

Used in tools like Metasploit and similar.

Secure Software Development

How to protect your PC against the major 'Meltdown' CPU security flaw

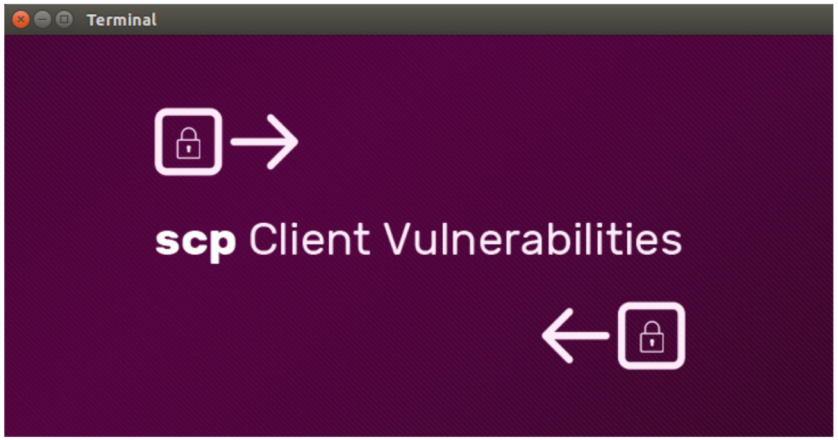
Everything we know so far

By Tom Warren | @tomwarren | Jan 4, 2018, 8:12am EST

Details have emerged on two major processor security flaws this week, and the industry is scrambling to issue fixes and secure machines for customers. Dubbed "Meltdown" and "Spectre," the flaws affect nearly every device made in the past 20 years. The Meltdown flaw primarily affects Intel and ARM processors, and researchers have already released proof-of-concept code that could lead to attacks using Meltdown.

36-Year-Old SCP Clients' Implementation Flaws Discovered

January 15, 2019 Mohit Kumar



A set of 36-year-old vulnerabilities has been uncovered in the Secure Copy Protocol (SCP) implementation of many client applications that can be exploited by malicious servers to overwrite arbitrary files in the SCP client target directory unauthorizedly.

Identified vulnerabilities have an interesting common feature:

existed for more than twenty years

being one in open code (OpenSSH scp) and another in "closed microcode" of the processor.

Proves the need for proper software development processes that prevent the creation of project vulnerabilities like these.

Secure Software Development

- Security is just one of several conflicting goals in software development.
- Goals in software development:
  - **Functionality** – quality of software products is often assessed by the amount of functionality provided, which leads to a high degree of complexity, in obvious conflict with security;
  - **Usability** – the ease of a user to take advantage of the software product, with security often being considered as an obstacle to usability, to the point where it is necessary to establish a "compromise" between the two;
  - Software **performance** – cryptographic mechanisms or security checks spend CPU time, with a negative effect on performance;
  - **Simplicity** – reduces production and maintenance costs, not compatible with the need for security mechanisms;
  - **Time-to-market** – rapid product placement on the market, with security (testing and evaluation) being a hindrance.

Secure Software Development

- Goal of developing secure software (software product or computer system):

  Zero vulnerabilities ?

- Zero vulnerabilities is only possible in very simple software.
- The factors to take into account in the development of secure software are **risk**, **threat level** (potential for attacks), **degree of vulnerability**, **probability of attack to succeed** and **impact** (or **cost of failure**).
- Examples:
  - Very vulnerable software product, used on personal computers of few people;
  - Institutional website of an University;
  - Valuable and exposed system on the Internet (e.g., bitcoins wallet).

probability of attack to succeed = threat level * degree of vulnerability
risk = probability of attack to succeed * impact

risk = threat level * degree of vulnerability * impact

Secure Software Development

- Goal of developing secure software (software product or computer system):

Reduce risk to acceptable levels

risk = threat level * degree of vulnerability * impact

- In the examples seen, how much should be spent on security?
  - Very vulnerable software product, used on personal computers of few people;
  - Institutional website of an University;
  - Valuable and exposed system on the Internet (e.g., bitcoins wallet).
- Value that does not exceed the impact (or cost of failures) of security, which can translate into loss of customers and/or in legal proceedings that lead to compensations.
- What is the risk factor that software developers can control best?
  - In general, the degree of vulnerability.

Secure Software Development

- Goal of developing secure software (software product or computer system):

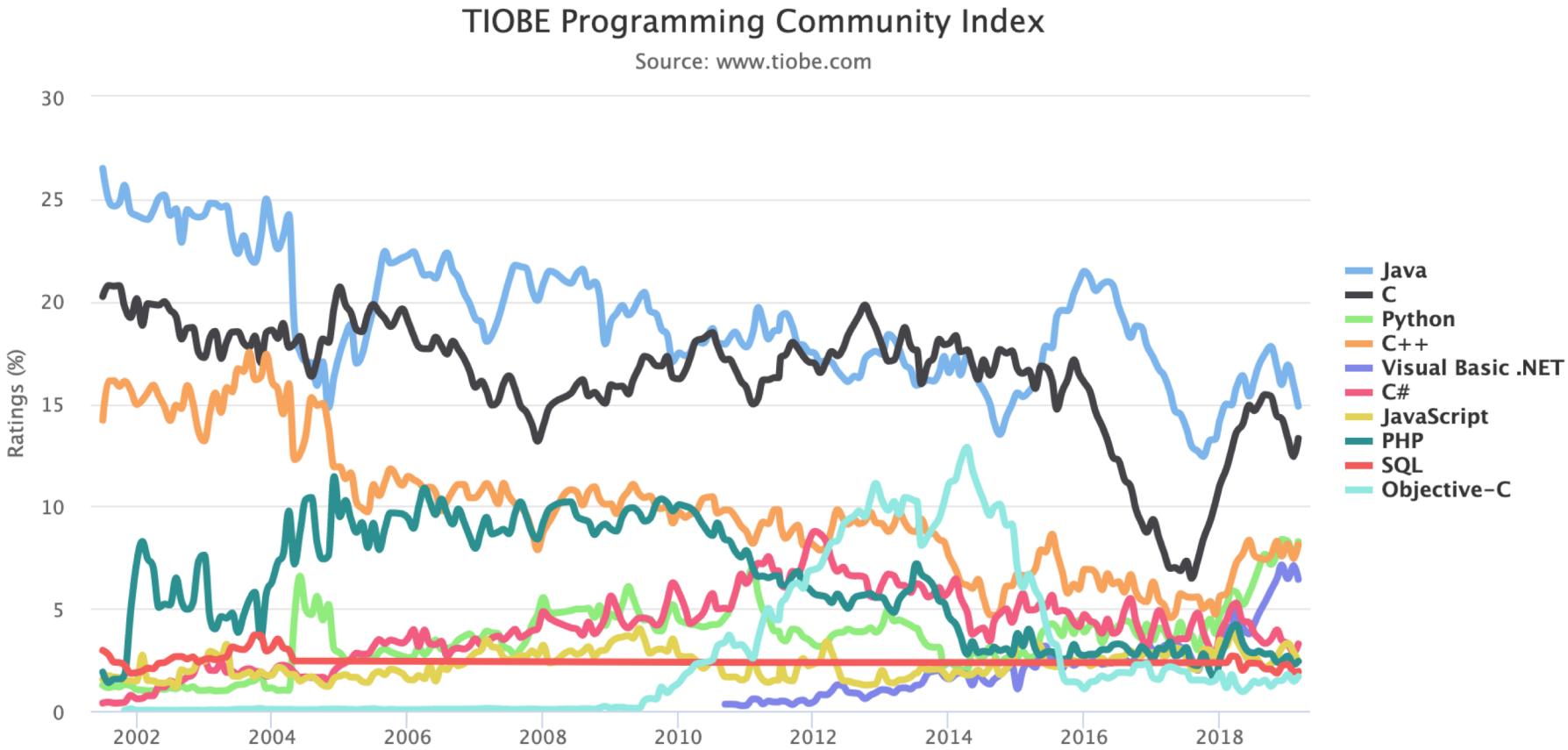
<u>Reduce risk</u> <u>to acceptable levels</u>

risk = threat level * degree of vulnerability * impact

- In the examples seen, how much should be spent on security?
  - Very vulnerable software product, used on personal computers of few people;
  - Institutional website of an University;
  - Valuable and exposed system on the Internet (e.g., bitcoins wallet).

Secure Software Development

- Some factors that influence the degree of vulnerability
  - Design, implementation and operational vulnerabilities;
  - Implemented security mechanisms (or controls);
  - Level of maturity of the security processes of the organization that uses the software;
  - Programming language used.



Secure Software Development

- Some factors that influence the degree of vulnerability
  - Design, implementation and operational vulnerabilities;
  - Implemented security mechanisms (or controls);
  - Level of maturity of the security processes of the organization that uses the software;
  - Programming language used.
    - C/C++ – compiled language in which each C/C ++ statement is translated into machine language instructions. Advantage: Excellent performance. Disadvantage: Poor memory management leading to numerous vulnerabilities.
    - Java – language translated into bytecodes, which are interpreted and optionally compiled at runtime. Advantage: Careful memory management; It can be run in a sandbox that enforces access control policies to system resources. Disadvantage: Lower performance.
    - Perl – interpreted language with the ability to operate in *taint mode*, ensuring that the input is not sent to safety-critical commands before being validated.

    - It should be noted that language X is not better than language Y, but the security aspect provided by the language must be weighed carefully at the time of its choice (and this aspect is not static, as it varies over time), as well as the choice of APIs, application server, software components, operating system, ... should be considered.
    - Note that often the language provides security mechanisms, but programmers are unaware of which should be used or how they should be used.

Secure Software Development

- Some factors that influence the degree of vulnerability
  - Design, implementation and operational vulnerabilities;
  - Implemented security mechanisms (or controls);
  - Level of maturity of the security processes of the organization that uses the software;
  - Programming language used.
    - C/C++ – compiled language in which each C/C ++ statement is translated into machine language instructions. Advantage: Excellent performance. Disadvantage: Poor memory management leading to numerous vulnerabilities.
    - Java – language translated into bytecodes, which are interpreted and optionally compiled at runtime. Advantage: Careful memory management; It can be run in a sandbox that enforces access control policies to system resources. Disadvantage: Lower performance.
    - Perl – interpreted language with the ability to operate in *taint mode*, ensuring that the input is not sent to safety-critical commands before being validated.
  - Closed versus open source code
    - Controversial subject, involving ideological ballast and economic interests;
    - From the point of view of software security, there is a seemingly strong argument in favor of open source security: being open for scrutiny (many eyballs). Does this mean that in open source, the thousands of eyes actually look at the code and are able to discover vulnerabilities (often subtle)? And only who is willing to contribute to increase the security of the code looks into the code?
    - Do lado do código fechado, “escondido é mais seguro”?
    - On the closed code side, "hidden is more secure"?