

# Mestrado em Engenharia Informática (MEI)

# Mestrado Integrado em Engenharia Informática

## (MiEI)

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

Engenharia de Segurança

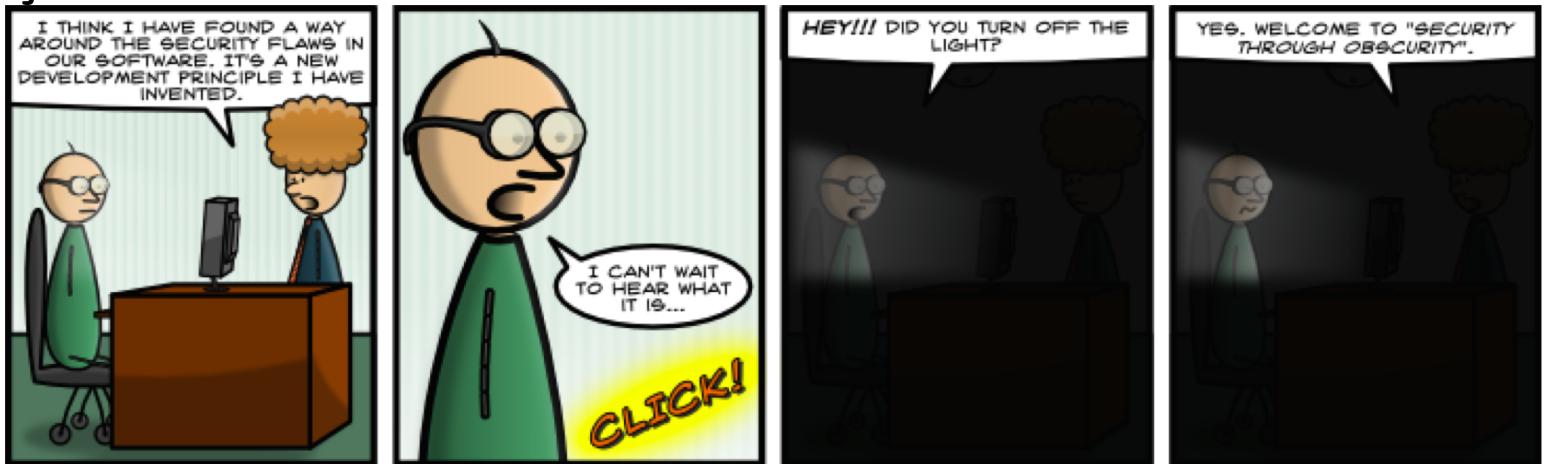


# Tópicos

- Segurança de Software

(Bibliografia: Segurança no Software, Miguel Pupo Correia, Paulo Jorge Sousa)

## Motivação



# Segurança de Software

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

- Relevância

- Ataques a sistemas financeiros, médicos, governamentais e infraestruturas críticas têm aumentado em número e gravidade;
- Com as infraestruturas digitais cada vez mais complexas e interligadas, a dificuldade de disponibilizar sistemas seguros aumenta de forma exponencial;
- Facilidade de exploração de erros.

- Objectivo

- Ensinar programação robusta, ou como escrever programas fiáveis, no sentido em que executam as tarefas codificadas e lidam com inputs ou eventos inesperados, de uma forma razoável;
- Conhecer e perceber a importância dos princípios de programação segura;
- Preparar futuros profissionais de software para desenhar e implementar software seguro.



# Segurança de Software

- Mitos
  - Se os estudantes aprenderem a escrever programas seguros, o estado de segurança do software e sistemas informáticos vai melhorar drasticamente
    - Programas dependem de sistemas operativos, de APIs/bibliotecas de terceiros e de outros serviços externos;
    - Segurança de um sistema informático depende de ser criado e configurado de acordo com os requisitos do ambiente particular em que é utilizado – o intervalo de diferença entre a configuração teórica necessária e a prática usual é desconhecido, mas provavelmente grande;
    - Não é claro se as *software houses* estão dispostas a pagar o preço associado à codificação segura nem se os clientes estarão dispostos a pagar preços mais altos, e suportar tempos de desenvolvimento mais longos.
  - As Universidades sabem o que e como fazer para ensinar programação segura
    - As Universidades não sabem como ensinar programação segura. Têm ideias e muita teoria, mas não sabem o que vai funcionar e quando.



# Segurança de Software - Conceitos

- Objetivo da **segurança** é garantir os seguintes atributos:
  - **confidencialidade** – ausência de divulgação não autorizada de informação;
  - **integridade** – ausência de alterações não autorizadas ao sistema ou à informação;
  - **disponibilidade** – prontidão da informação para ser acedida ou, do sistema para fornecer um serviço;
  - **autenticidade** – informação ou serviço fornecido são genuínos.

# Segurança de Software - Conceitos

- Objetivo da **segurança** é garantir os seguintes atributos:
  - **confidencialidade** – ausência de divulgação não autorizada de informação;
  - **integridade** – ausência de alterações não autorizadas ao sistema ou à informação;
  - **disponibilidade** – prontidão da informação para ser acedida ou, do sistema para fornecer um serviço;
  - **autenticidade** – informação ou serviço fornecido são genuínos.

A **privacidade** é a confidencialidade de dados pessoais (nome, telefone, morada, ...).

# Segurança de Software - Conceitos

- Objetivo da **segurança** é garantir os seguintes atributos:
  - **confidencialidade** – ausência de divulgação não autorizada de informação;
  - **integridade** – ausência de alterações não autorizadas ao sistema ou à informação;
  - **disponibilidade** – prontidão da informação para ser acedida ou, do sistema para fornecer um serviço;
  - **autenticidade** – informação ou serviço fornecido são genuínos.

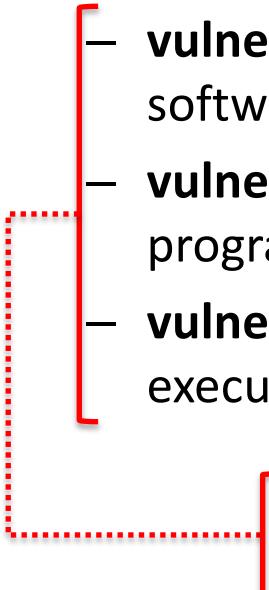
A definição do que é ou não autorizado, i.e., o conjunto de requisitos de segurança que têm de ser satisfeitos pelo sistema, constitui a sua **política de segurança**.

# Segurança de Software - Conceitos

- **Vulnerabilidade** é um defeito do sistema (software ou outro) que pode ser explorado por um atacante com o objetivo de violar a política de segurança.
- As vulnerabilidades de software podem ser classificadas com base em três categorias:
  - **vulnerabilidade de projeto** – introduzida durante a fase de projeto do software (obtenção de requisitos e desenho);
  - **vulnerabilidade de codificação** (implementação) – introduzida durante a programação do software, i.e., um bug com implicações de segurança;
  - **vulnerabilidade operacional** – causada pelo ambiente no qual o software é executado ou pela sua configuração.

# Segurança de Software - Conceitos

- **Vulnerabilidade** é um defeito do sistema (software ou outro) que pode ser explorado por um atacante com o objetivo de violar a política de segurança.
- As vulnerabilidades de software podem ser classificadas com base em três categorias:
  - **vulnerabilidade de projeto** – introduzida durante a fase de projeto do software (obtenção de requisitos e desenho);
  - **vulnerabilidade de codificação** (implementação) – introduzida durante a programação do software, i.e., um bug com implicações de segurança;
  - **vulnerabilidade operacional** – causada pelo ambiente no qual o software é executado ou pela sua configuração.

 **Segurança de software** diz respeito a estes três tipos de vulnerabilidades.

# Segurança de Software - Conceitos

- **Ataque** é uma ação maliciosa que visa ativar uma ou mais vulnerabilidades de modo a subverter a política de segurança do sistema.
- Ataque com sucesso (**intrusão**), quando ativa uma ou mais vulnerabilidades.
- **Exploit** é um pedaço de código capaz de ativar determinada vulnerabilidade num pacote de software.
  - Exploit também é usado (em inglês) como verbo para denominar a ação de realizar um ataque

ataque + vulnerabilidade(s) ➔ intrusão

# Vulnerabilidade de codificação

- Vulnerabilidade de codificação tem ciclo de vida próprio – quando um pacote de software começa a ser comercializado contém inúmeros **bugs**.
- Estima-se que qualquer pacote de software tem uma média de 5 a 50 bugs por cada 1.000 SLOC (*Source Lines Of Code* – linhas de código fonte, excluindo linhas de comentário), alguns dos quais são vulnerabilidades.
  - Limite superior (50 bugs por 1.000 LOC) para software normal;
  - Limite inferior (5 bugs por 1.000 LOC) para software desenvolvido utilizando métodos de desenvolvimento rigorosos.

Pacote de Software	Ficheiros	SLOC
Windows XP (2001)	n/a	45 milhões <sup>(1)</sup>
MacOSX 10.4 (2005)	n/a	86 milhões <sup>(1)</sup>
Linux kernel 3.6 (2012)	n/a	15,9 milhões <sup>(1)</sup>
MariaDB 10.1	5.014	2.127.699 <sup>(2)</sup>
PostgreSQL 9.4.4	3.864	1.698.471 <sup>(2)</sup>
Python 2.7.10	3.240	998.697 <sup>(2)</sup>
Perl 5.22.0	3.434	903.874 <sup>(2)</sup>

(1) [https://en.wikipedia.org/wiki/Source\\_lines\\_of\\_code](https://en.wikipedia.org/wiki/Source_lines_of_code)

(2) <https://github.com/aldanial/cloc>



# Vulnerabilidade de codificação

- Vulnerabilidade de codificação tem ciclo de vida próprio – quando é descoberta uma vulnerabilidade, a empresa que criou o software deve criar um **remendo (patch/fix)**.
- *Patch* deve ser instalado pelos administradores ou utilizadores do software, de modo a removerem a vulnerabilidade.
- Aspetos negativos dos *patch*:
  - Publicar um *patch* é declarar publicamente que o produto tem problemas;
  - Publicar um *patch* permite que atacantes descubram a(s) vulnerabilidade(s) que o *patch* vem resolver, e criem exploits que tirem partido da(s) mesma(s). Qual o objetivo?
  - Instalar um *patch* tem custos de administração;
  - Instalar um *patch* pode resolver uma vulnerabilidade, mas introduzir outra(s).
- Para minimizar estes efeitos, alguns fabricantes de software publicam *patch* com uma periodicidade fixa, resolvendo diversas vulnerabilidades (excepto vulnerabilidades muito críticas, para as quais publicam *patch* imediatos).
- Outros (como é normal nos dispositivos móveis) disponibilizam novas versões de software com novas funcionalidades (e resolvendo vulnerabilidades).



# Vulnerabilidade

- As vulnerabilidades podem não ser conhecidas na comunidade de segurança informática, mas serem conhecidas em meio restrito. Qual o objetivo?
  - grupo de piratas informáticos (ataques ou venda na *dark web*),
  - meio militar de um país (ciber-armas).
- Usualmente denominadas de **vulnerabilidades dia-zero (0-day vulnerability)**

**Ataques dia-zero** – Permite atacar sistemas administrados por equipas competente e com bons conhecimentos de segurança.

Pos	\$25k-\$100k	Trend	DL	Today↓	0-day
1	AMD EPYC Server/Ryzen/Ryzen Pro/Ryzen Mobile Hardware Validated Boot MASTERKEY privilege escalation	=	x	\$25k-\$100k	\$100k and more
2	Cisco IOS/IOS XE LLDP Subsystem memory corruption [CVE-2018-0175]	=	x	\$25k-\$100k	\$25k-\$100k
3	Cisco IOS/IOS XE Smart Install memory corruption [CVE-2018-0171]	=	x	\$25k-\$100k	\$25k-\$100k
4	Cisco IOS XE Switch Integrated Security unknown vulnerability	=	x	\$25k-\$100k	\$25k-\$100k
5	Apple iOS WebKit Assertion unknown vulnerability	+	x	\$25k-\$100k	\$100k and more

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

# Vulnerabilidade

- Quando uma vulnerabilidade é tornada pública, torna-se passível de ser usada para atacar sistemas.
  - Porque não é impedido, por legislação, a publicação de vulnerabilidades?
  - Qual é a forma ética de publicar uma vulnerabilidade?

29 March, 2018

## Cisco Smart Install Remote Code Execution Introduction

- Application: Cisco IOS, Cisco IOS-XE
- Vendor: [Cisco](#)
- Bugs: Stack-based buffer overflow [[CWE-20](#)], [[CWE-121](#)]
- Risk: Critical; AV:N/AC:L/Au:N/C:C/I:C/A:C (10.0)

A stack-based buffer overflow vulnerability was found in [Smart Install Client](#) code. This vulnerability enables an attacker to remotely execute arbitrary code without authentication. So it allows getting full control over a vulnerable network equipment.

## Diclosure Timeline

- 13/05/2017 – The vulnerability was presented at the [GeekPWN 2017 Hong-Kong](#). Under the terms of the competition, interaction with the vendor for fixing the vulnerability is the right and responsibility of the GeekPWN organizers.
- 26/09/2017 – We informed the vendor about our planned talk "[How to cook Cisco. The exploit development for Cisco IOS](#)" at the [EKOPARTY 2017](#) conference and clarified the planned date of disclosure of the vulnerability. Vendor's response is below:

Thanks for sharing the slides.

Regarding the SMI RCE that you found at GeekPwn 2017 and mentioned on Slide 10. The team there did share the details and the vulnerability is well underway being patched.

For Cisco IOS Software we typically make public the vulnerabilities on the 4th Wed of Sept or March of each year. For the vulnerability, you reported it is slated for public release in March 2018, unless we see any public exploitation of the vulnerability or increased public awareness.



# Vulnerabilidade

- Quando uma vulnerabilidade é tornada pública, torna-se passível de ser usada para atacar sistemas.
  - Porque não é impedido, por legislação, a publicação de vulnerabilidades?
  - Qual é a forma ética de publicar uma vulnerabilidade?
- Catalogação de vulnerabilidades
  - *Common Weakness Enumeration (CWE)* – classificação de classes de vulnerabilidade, em que atribui a cada classe de vulnerabilidades um identificador com o formato CWE-NNNN, sendo NNNN o número atribuído à classe.
  - *Common Vulnerabilities and Exposures (CVE)* – catálogo de vulnerabilidades (de projeto e codificação) existentes em software comercial ou aberto, com identificador com formato CVE-AAAAA-NNNN, sendo AAAA o ano em que a vulnerabilidade foi catalogada e NNNN o seu número.
  - *Common Configuration Enumeration (CCE)* – catálogo de vulnerabilidades operacionais (ou de configuração) existentes em software comercial ou aberto, com identificador com formato CCE-NNNNN-M, sendo NNNNN o seu número, e M utilizado para verificar a integridade de NNNNN [histórico].
  - *Open Sourced Vulnerability Database (OSVDB)* – iniciativa *open source* similar ao CVE [histórico].

Geridas pela MITRE Corporation, e apadrinhadas pelo NIST que as divulga sob a designação de *National Vulnerability Database (NVD)*.



# Vulnerabilidade

- Quando o *exploit* de uma vulnerabilidade é tornado público, o perigo torna-se imediato.
- Fazer um ataque depois de conhecido o *exploit*, tende a ser simples.
- Catálogos de *exploits*
  - *Exploit Database* – arquivo de *exploits* públicos, identificando o CVE da vulnerabilidade e/ou software que explora, para utilização por investigadores de vulnerabilidades e *penetration testers*.
  - *Google Hacking Database* – arquivo de Google *dorks* (*query* de pesquisa que retorna a informação sensível) que embora sendo uma forma de *exploits*, são também utilizados para criar novos *exploits*.

Utilizada em ferramentas como Metasploit e similares.

# Desenvolvimento de Software Seguro

20 years-old Orpheus' Lyre vulnerability in Kerberos fixed this week

July 14, 2017 By Pierluigi Paganini

[f My Page](#)

[Like 0](#)

[G+](#)

**A 20 years-old vulnerability in Kerberos, dubbed Orpheus' Lyre, was patched this week for both Microsoft and Linux distros.**

A 20 years-old vulnerability in Kerberos was patched this week for both Microsoft and Linux distros.

The vulnerability dubbed [Orpheus' Lyre](#) has been found three months ago by Jeffrey Altman, founder of AuriStor, and Viktor Dukhovni and Nicolas Williams from Two Sigma Investments. The issue

The flaw, tracked as CVE-2017-11103, was found in Heimdal, an open-source implementation of Kerberos, like the mythological character Orpheus played his lyre with such grace that it lulled Cerberus to sleep, this issue can bypass Kerberos.

The issue could result in remote privilege escalation and credential theft, an attacker can trigger it to access the target network.

Demonstra a necessidade de existirem processos de desenvolvimento de software adequados que evitem a criação de vulnerabilidades de projeto como estas.

## 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.



Meltdown



Spectre

As vulnerabilidades identificadas têm uma característica comum interessante:

existiam há mais de vinte anos  
sendo que uma era em código aberto (Orpheus Lyre) e outra em "microcódigo fechado" do processador.

# Desenvolvimento de Software Seguro

- Segurança é apenas um de vários objetivos antagónicos existentes no desenvolvimento de software.
- Objetivos no desenvolvimento de software:
  - **Funcionalidade** – qualidade dos produtos de software é muitas vezes avaliada pela quantidade de funcionalidades fornecidas, o que conduz a um elevado grau de complexidade, em evidente contradição com a segurança;
  - **Usabilidade** – facilidade de um utilizador tirar partido do produto de software, sendo a segurança muitas vezes considerada como um entrave à usabilidade, a ponto de ser necessário estabelecer um “compromisso” entre as duas;
  - **Desempenho** do software – mecanismos criptográficos ou verificações de segurança gastam tempo de CPU, com efeito negativo no desempenho;
  - **Simplicidade** – diminui custos de produção e manutenção, não compatíveis com a necessidade de mecanismos de segurança;
  - **Time-to-market** – colocação rápida do produto no mercado, sendo a segurança (testes e avaliações) um entrave.



# Desenvolvimento de Software Seguro

- Objetivo do desenvolvimento de software seguro (produto de software ou sistema informático):  
Zero vulnerabilidades ?
- Zero vulnerabilidades só é possível em software muito simples.
- [Os fatores a ter em conta no desenvolvimento de software seguro é o **risco, nível de ameaça** (potencial para existirem ataques), **grau de vulnerabilidade, probabilidade do ataque ter sucesso** e o **impacto** (ou **custo de falha**)].
- Exemplos:
  - Produto de software muito vulnerável, usado em computadores pessoais de poucas pessoas;
  - Página web institucional de uma Universidade;
  - Sistema valioso e exposto na Internet (e.g., carteira de *bitcoins*).

[probabilidade do ataque ter sucesso = nível da ameaça \* grau de vulnerabilidade  
risco = probabilidade de ataque ter sucesso \* impacto]

**risco** = nível de ameaça \* grau de vulnerabilidade \* impacto



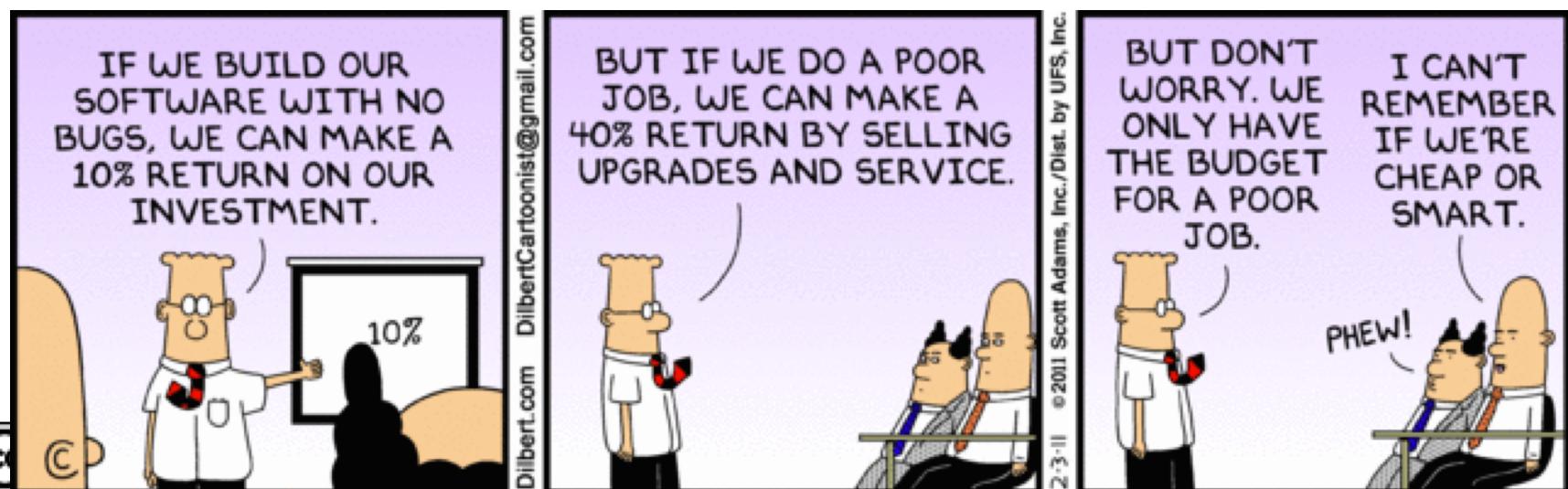
# Desenvolvimento de Software Seguro

- Objetivo do desenvolvimento de software seguro (produto de software ou sistema informático):

**Reducir o risco** para níveis aceitáveis

**risco** = nível de ameaça \* grau de vulnerabilidade \* impacto

- Nos exemplos vistos, qual o valor que faz sentido gastar em segurança?
  - Produto de software muito vulnerável, usado em computadores pessoais de poucas pessoas;
  - Página web institucional de uma Universidade;
  - Sistema valioso e exposto na Internet (e.g., carteira de bitcoins).



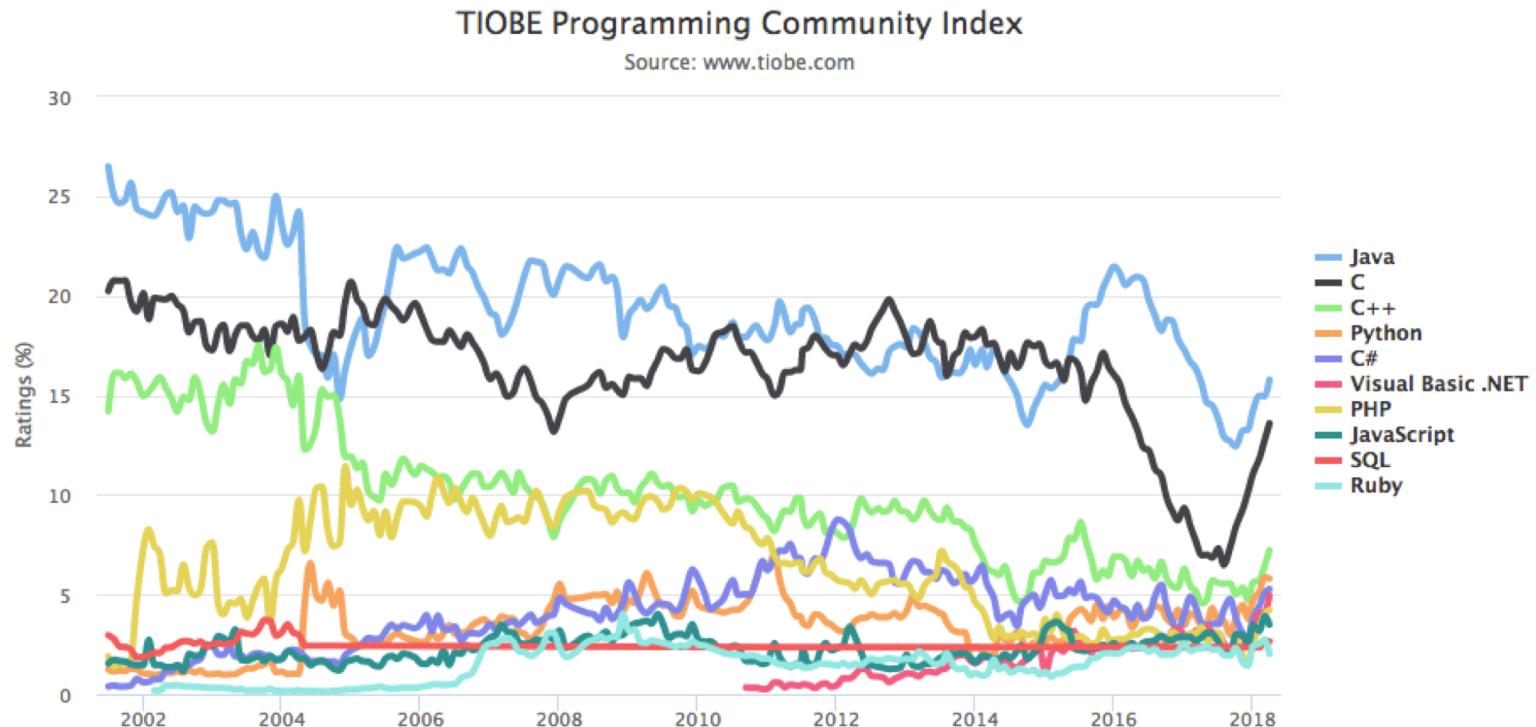
# Desenvolvimento de Software Seguro

- Objetivo do desenvolvimento de software seguro (produto de software ou sistema informático):  
**Reducir o risco** para níveis aceitáveis  
**risco** = nível de ameaça \* grau de vulnerabilidade \* impacto
- Nos exemplos vistos, qual o valor que faz sentido gastar em segurança?
  - Produto de software muito vulnerável, usado em computadores pessoais de poucas pessoas;
  - Página web institucional de uma Universidade;
  - Sistema valioso e exposto na Internet (e.g., carteira de bitcoins).
- Valor que não seja superior ao impacto (ou custo das falhas) de segurança, que se pode traduzir na perda de clientes e/ou em processos judiciais que levem a indemnizações.
- Qual o fator de risco mais controlável por quem desenvolve o software?
  - Em geral, o fator em que se pode atuar é o grau de vulnerabilidade.



# Desenvolvimento de Software Seguro

- Alguns fatores que influem sobre o grau de vulnerabilidade
  - Existência de vulnerabilidades de projeto, de codificação e operacionais;
  - Mecanismos (ou controlos) de segurança implementados;
  - Nível de maturidade dos processos de segurança da organização que utiliza o software;
  - Linguagem de programação utilizada



# Desenvolvimento de Software Seguro

- Alguns fatores que influem sobre o grau de vulnerabilidade
  - Existência de vulnerabilidades de projeto, de codificação e operacionais;
  - Mecanismos (ou controlos) de segurança implementados;
  - Nível de maturidade dos processos de segurança da organização que utiliza o software;
  - Linguagem de programação utilizada
    - C/C++ – linguagem compilada em que cada instrução C/C++ é traduzida para instruções em linguagem máquina. Vantagem: Excelente desempenho. Desvantagem: Gestão de memória deficiente que leva a inúmeras vulnerabilidades.
    - Java – linguagem compilada para bytecodes, que são interpretados e opcionalmente compilados em tempo de execução. Vantagem: Gestão de memória cuidada; Pode ser executado num *sandbox* que impõe políticas de controlo de acesso a recursos do sistema. Desvantagem: Desempenho inferior.
    - Perl – linguagem interpretada com possibilidade de operar no *taint mode* (segurança na validação do input da aplicação, garantindo que o input não é passado a comandos perigosos sob o ponto de vista de segurança, antes de serem validados).
      - Note-se que a linguagem X não é melhor que a linguagem Y, mas o aspeto de segurança proporcionado pela linguagem deve ser ponderado cuidadosamente no momento da sua escolha (e esse aspeto não é estático, já que varia com o passar do tempo), assim como deve ser ponderada a escolha de APIs, servidor aplicacional, componentes de software, sistema operativo, ....
      - Note-se que muitas vezes a linguagem disponibiliza mecanismos de segurança, mas os programadores desconhecem que existem, que devem ser usados ou, como devem ser usados.

# Desenvolvimento de Software Seguro

- Alguns fatores que influem sobre o grau de vulnerabilidade
  - Existência de vulnerabilidades de projeto, de codificação e operacionais;
  - Mecanismos (ou controlos) de segurança implementados;
  - Nível de maturidade dos processos de segurança da organização que utiliza o software;
  - Linguagem de programação utilizada
    - C/C++ – linguagem compilada em que cada instrução C/C++ é traduzida para instruções em linguagem máquina. Vantagem: Excelente desempenho. Desvantagem: Gestão de memória deficiente que leva a inúmeras vulnerabilidades.
    - Java – linguagem compilada para bytecodes, que são interpretados e opcionalmente compilados em tempo de execução. Vantagem: Gestão de memória cuidada; Pode ser executado num *sandbox* que impõe políticas de controlo de acesso a recursos do sistema. Desvantagem: Desempenho inferior.
    - Perl – linguagem interpretada com possibilidade de operar no *taint mode* (segurança na validação do input da aplicação, garantindo que o input não é passado a comandos perigosos sob o ponto de vista de segurança, antes de serem validados).
  - Código fechado versus código aberto
    - Assunto controverso, que envolve carga ideológica e interesses económicos;
    - Do ponto de vista da segurança do software, há um argumento aparentemente forte a favor da segurança do código aberto: estar aberto para escrutínio (*many eyeballs*). Será que isso significa que no código aberto, os milhares de olhos, olham realmente para o código e são capazes de descobrir vulnerabilidades (muitas vezes *subtis*)? E só olha para o código quem está disposto a contribuir para aumentar a sua segurança?
    - Do lado do código fechado, “escondido é mais seguro”?