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

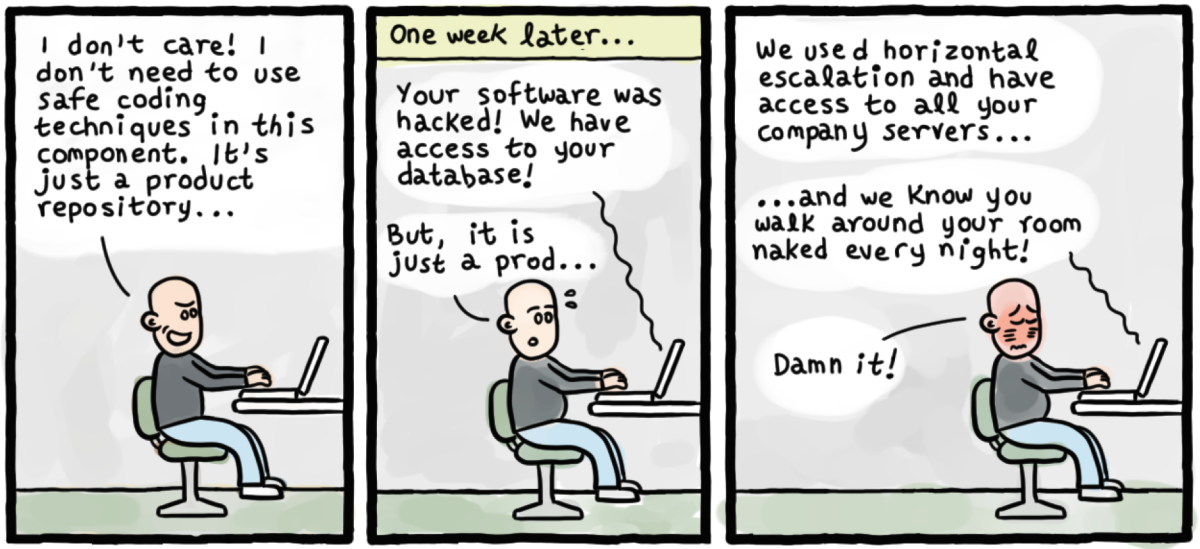
Perfil de Especialização **CSI** : Criptografia e Segurança da Informação

Engenharia de Segurança

jose.miranda@devisefutures.com

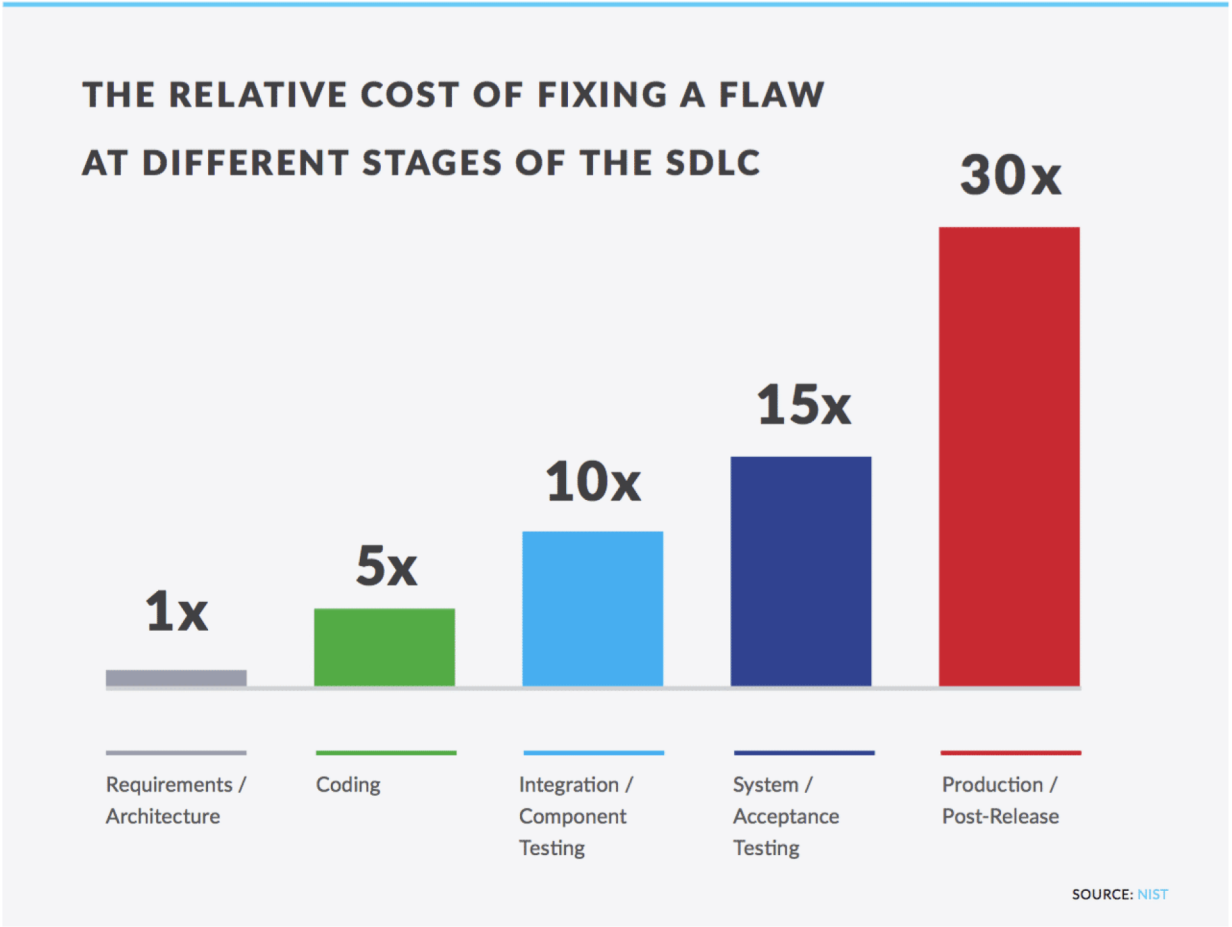
Tópicos de Segurança de Software

- Princípios de projeto de software
- Evitar as Vulnerabilidades de Projeto predominantes



Vulnerabilidades de Projeto

- Tal como visto em aula anterior, as **vulnerabilidades de projeto** são introduzidas durante a fase de projeto do software (obtenção de requisitos e desenho).



Princípios de Projeto de software

- Princípios de projeto de software a seguir para evitar os erros mais comuns:
  - **Economia de mecanismos** – manter o projeto tão simples quanto possível, já que a complexidade causa vulnerabilidades. No âmbito da segurança, aplica-se, por exemplo, aos mecanismos de controlo de acesso ou de validação de *inputs*;
  - **Decisão segura por omissão** – Utilizar escolhas por omissão à prova de falhas. Decisão por omissão é o "não acesso", sendo o acesso permitido apenas por decisão explícita. No âmbito da segurança, as aplicações deste princípio são evidentes, assim como os exemplos de violação deste princípio (sistemas de acesso sem senhas definidas por omissão, ou com senhas conhecidas; acesso a objetos/dados definidos por omissão e não para quem necessita de aceder);
  - **Mediação completa** – deve ser verificada a autorização de todos os acessos realizados a qualquer objeto, ou seja, não deve ser possível contornar o controlo de acesso;
  - **Projeto aberto** – segurança não deve depender do segredo do projeto ou desenho, já que a "segurança por obscuridade" gera apenas uma ilusão de segurança;
  - **Separação de privilégios** – defende que um mecanismos que tenha de ser destrancado por duas chaves distintas é mais robusto e flexível do que outro que requeira apenas uma chave. No âmbito da segurança, aplica-se, por exemplo no caso da autenticação (autenticação multifator – *something you know, something you have, something you are*);

Princípios de Projeto de software

- Princípios de projeto de software a seguir para evitar os erros mais comuns:
  - **Privilégios mínimos** (Confiança com relutância) – cada processo e cada utilizador devem operar tendo as permissões mínimas (“*need to know*”) para fazerem o que precisam, de modo a limitar os estragos causados por um erro acidental ou intrusão. No âmbito da segurança, aplica-se, por exemplo, em executar servidores Web/mail/... com privilégios baixos, em vez de *root*/administrador;
  - **Mecanismos comuns mínimos** – deve ser minimizada a dependência dos processos e utilizadores em relação a recursos partilhados entre eles, já que qualquer coisa partilhada (e.g., ficheiros ou memória) constitui um caminho pelo qual podem ser passados dados (privados ou confidenciais);
  - **Adequação psicológica** – defende que a interface pessoa-máquina deve ser projetada para ser simples de usar. Tema de usabilidade da segurança, cujo objetivo é os utilizadores conseguirem aplicar corretamente os mecanismos de proteção, de forma rotineira e automática.
  - **Seguro por omissão** – software deve ser projetado tendo em conta a possibilidade de ocorrência de intrusões. Por exemplo, serviços desnecessários desligados e, configurações por omissão devem ser conservadoras;
  - **Considerar a segurança desde o início do projeto**;

Princípios de Projeto de software

- Princípios de projeto de software a seguir para evitar os erros mais comuns:
  - **Seguro na instalação** – software deve conter:
    - Ferramentas e documentação que permitam que seja configurado corretamente, sob o ponto de vista da segurança;
    - Ferramentas de administração e análise de segurança que permitam ao administrador determinar e configurar o nível de segurança pretendido;
    - Ferramentas para a instalação de atualizações e *patches*;
  - **Comunicação** – devem existir processos para resposta a vulnerabilidades, assim como o envolvimento dos programadores com a comunidade de utilizadores para resposta a dúvidas sobre segurança e vulnerabilidades;
  - **Defesa em profundidade** (*Defend in depth*) – vários níveis ou mecanismos de proteção (*security by diversity*), de modo a que o software não seja comprometido mesmo que esteja mal configurado, seja vulnerável ou que um dos níveis/mecanismos de proteção falhe;
  - **Proteger o elo mais fraco** – os atacantes tendem a atacar o ponto mais fraco do sistema, que por isso deve ser a primeira preocupação a ter em conta. Note-se que o elo mais fraco de muitos sistemas são os utilizadores (pouco alertados para as questões de segurança);
  - **Falhar de forma segura** – quando ocorre uma falha, o sistema deve ficar num estado considerado seguro. Ou seja, as condições de erro devem ser tratadas com muito cuidado.

Evitar as Vulnerabilidades de Projeto predominantes

- Como evitar vulnerabilidades de projeto predominantes[1]:
  1. **Nunca assumir ou confiar sem validar**
     - São inerentemente inseguros os projetos de soluções de software que coloquem lógica de autorização, controle de acesso, efetivação de políticas segurança e quaisquer dados sensíveis em software cliente, pensando que estes mecanismos não serão descobertos/ modificados/expostos por utilizadores inteligentes ou por atacantes.
     - Uma regra de ouro na construção de software seguro é nunca confiar nos *inputs*. Sempre que se projete um sistema que receba dados de componentes-cliente não confiáveis, os dados recebidos devem assumir-se como comprometidos até prova em contrário, pelo que têm de ser validados adequadamente antes de serem processados.
     - Por outro lado, se dados sensíveis ou propriedade intelectual necessitam de ser enviadas ou armazenadas em software cliente, o sistema deve ser concebido para ser capaz de tolerar um potencial comprometimento dos mesmos.
  2. **Usar um mecanismo de autenticação que não possa ser contornado ou comprometido**
     - Um dos objetivos de desenho de software seguro é prevenir que alguém aceda ao sistema ou ao serviço sem antes se autenticar, assim como prevenir que um utilizador mude de identidade (ou efetue operações críticas) sem se reautenticar. Recomendada a utilização de técnicas de autenticação multifator (*something you know, something you have and something you are*).
     - Autenticação abrange também componentes residentes em máquinas diferentes que têm de se autenticar mutuamente.
     - O ato de autenticação resulta, normalmente, na criação de um *token* que é usado como representação da entidade autenticada em todas as suas interações com o sistema. Se estes *tokens* puderem ser derivados deterministicamente a partir de informação fácil de obter (p.ex., *username*), ou se esses *tokens* continuarem válidos após períodos de inatividade, será fácil um atacante personificar um utilizador válido do sistema.

[1] Relatório IEEE – https://www.computer.org/cms/CYBSI/docs/Top-10-Flaws.pdf

Evitar as Vulnerabilidades de Projeto predominantes

- Como evitar vulnerabilidades de projeto predominantes[1]:

3. Autorizar após autenticar

- Autorizar após autenticar é igualmente importante, já que nem todos os utilizadores autenticados devem ter acesso a todas as operações do sistema.
- A autorização das operações, realizadas por utilizadores autenticados, deve ser explícita. Estas autorizações podem depender dos privilégios associados ao utilizador autenticado, assim como ao contexto da operação, incluindo a hora de operação e o local onde se encontra o utilizador.

4. Separar os dados, das instruções de controlo

- Misturar dados e instruções de controlo numa mesma entidade (p.ex., uma *string*) pode levar a "vulnerabilidades de injeção" que permitem o controlo do fluxo de execução do sistema por dados não confiáveis.

5. Validar explicitamente todos os dados

- Os pressupostos (explícitos ou implícitos) relativos aos dados sobre os quais o software opera (p.ex, formato, tamanho, ...) devem ser explicitamente validados.

[1] Relatório IEEE – https://www.computer.org/cms/CYBSI/docs/Top-10-Flaws.pdf

Evitar as Vulnerabilidades de Projeto predominantes

- Como evitar vulnerabilidades de projeto predominantes[1]:

6. Usar corretamente a criptografia

- Criptografia é uma das ferramentas mais importantes para construir sistemas seguros (confidencialidade e integridade dos dados, autenticar a origem dos dados, entre outros).
- Armadilhas a evitar:
  - Utilização de algoritmos criptográficos caseiros, já que a concepção de um algoritmo criptográfico requer competências matemáticas significativas e raras;
  - Utilização incorreta de algoritmos criptográficos, já que por exemplo, pode ser necessário escolher o vetor de inicialização (IV) com determinadas propriedades para o algoritmo ser seguro;
  - Má gestão de chaves, como por exemplo a colocação de chaves no código-fonte do software, a impossibilidade de revogação ou rotação de chaves, a utilização de chaves fracas e a utilização de mecanismos fracos de distribuição de chaves;
  - Aleatoriedade não aleatória, já que as operações criptográficas requerem números aleatórios com fortes propriedades de segurança: para além de terem de ser criptograficamente aleatórios, não podem ser reutilizados.

7. Identificar dados sensíveis e como devem ser tratados

- Sensibilidade dos dados depende de vários fatores de contexto, incluindo a regulamentação, política da organização, obrigações contratuais e expetativas dos utilizadores;
- Tratamento de dados deve ser alvo da elaboração de política de proteção de dados que permita definir uma abordagem unificada que cumpra todas as regulamentações aplicáveis;
- Requisitos de proteção de dados podem mudar ao longo do tempo, à medida que evolui o ambiente de negócio, muda a regulamentação, os sistemas ficam cada vez mais interligados, e são incorporadas novas fontes de dados no sistema.

[1] Relatório IEEE – https://www.computer.org/cms/CYBSI/docs/Top-10-Flaws.pdf

Evitar as Vulnerabilidades de Projeto predominantes

- Como evitar vulnerabilidades de projeto predominantes[1]:

8. Considerar sempre os utilizadores

- Software deve ser desenhado de forma a ser fácil de instalar, configurar, utilizar e atualizar de modo seguro, tendo também em consideração as habilidades cognitivas, a cultura, os hábitos e o nível cultural dos seus **utilizadores-alvo**.
- Desenho do software deve:
  - Facilitar a configuração e a utilização segura por aqueles interessados em fazê-lo;
  - Motivar e incentivar a utilização segura por aqueles não particularmente interessados em segurança de software;
  - Prevenir ou mitigar a utilização abusiva por aqueles que pretendam fragilizar ou comprometer o sistema.

[1] Relatório IEEE – https://www.computer.org/cms/CYBSI/docs/Top-10-Flaws.pdf

Evitar as Vulnerabilidades de Projeto predominantes

- Como evitar vulnerabilidades de projeto predominantes[1]:
  9. **Perceber o impacto dos componentes externos na superfície de ataque**
     - O desenho/projeto de software herda as fragilidades e limitações de segurança dos componentes externos utilizados (binários, bibliotecas ou código-fonte).
     - Para minimizar o impacto negativo de componentes externos, devem ser adotadas as seguintes medidas de segurança (sempre que possível):
       - Isolar os componentes externos utilizando contentores e *sandboxes*;
       - Configurar os componentes externos para ativar apenas as funcionalidades necessárias;
       - Documentar o risco que se está a aceitar, caso não seja possível alinhar as propriedades de segurança do componente externo com os requisitos de segurança do software;
       - Validar a proveniência e integridade do componente externo, assim como manter uma cópia local do código-fonte do componente;
       - Manter uma lista dos componentes externos utilizados e verificar periodicamente se as versões utilizadas correspondem às versões mais seguras disponíveis;
       - Autenticar o fluxo de dados entre o sistema e os componentes externos;
       - Estar atento a funcionalidades/comunicações extra (não documentadas) dos componentes externos.

[1] Relatório IEEE – https://www.computer.org/cms/CYBSI/docs/Top-10-Flaws.pdf

Evitar as Vulnerabilidades de Projeto predominantes

- Como evitar vulnerabilidades de projeto predominantes[1]:

10. Conceber antevendo alterações futuras

- Muitas alterações acontecem ao longo do tempo, com eventual impacto na segurança: alterações no próprio software, nos ambientes onde o software executa, assim como alterações a nível das ameaças e ataques ao software.
- Projeto de software seguro deve ser flexível para minimizar o impacto de alterações futuras, adotando os seguintes princípios:
  - Atualizações seguras – o sistema que é atualizado deve verificar a integridade e a proveniência dos pacotes de atualização, através da assinatura de código;
  - Capacidade para ativar e desativar funcionalidades – as vulnerabilidades identificadas podem não ser mitigadas em tempo útil, e os sistemas críticos não podem ser desligados, pelo que deve ser possível desativar funcionalidades com problemas através da configuração do software;
  - Alterações de segredos – palavras-chave (e mesmo chaves criptográficas) podem ser comprometidas, pelo que o sistema deve estar preparado para ser possível mudar os seus segredos em qualquer momento (necessário na fase de desenho identificar o que é afetado com essa mudança);
  - Alterações dos requisitos de segurança de componentes externos – devem existir camadas intermédias de abstração entre o código do sistema e a API de componentes externos (por exemplo, porque código de componente externo deixou de ser mantido, ou o seu licenciamento mudou), de forma a ser possível mudar esses componentes, sem mudar muito código do sistema;
  - Alterações de permissões especiais – podem existir acessos privilegiados para equipas de assistência (ou outros), sendo que a necessidade desses acessos privilegiados varia ao longo do tempo, devendo o sistema ter formas de revogar ou desativar o acesso a todas/algumas das funcionalidades quando não é necessário aceder-lhes, ou quando algum utilizador deixa de ter necessidade dessas permissões especiais.

[1] Relatório IEEE – https://www.computer.org/cms/CYBSI/docs/Top-10-Flaws.pdf