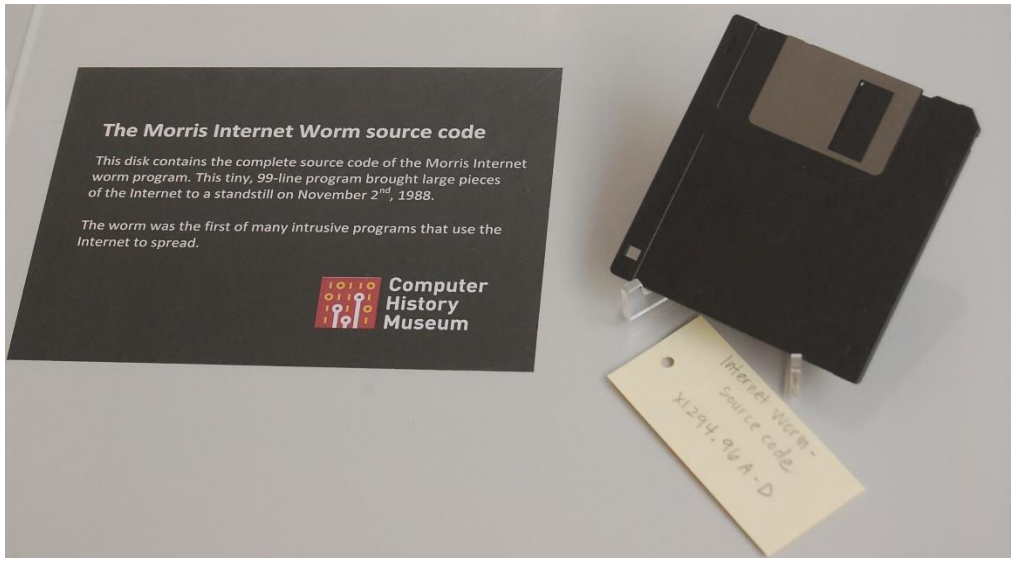
ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA



Conceitos de segurança de software

ISEL – Instituto Superior de Engenharia de Lisboa
Rua Conselheiro Emídio Navarro, 1 | 1959-007 Lisboa

Conceitos base de segurança

- Confidencialidade
  - Ausência de divulgação não autorizada de informação
  - Garantida por meios criptográficos ou de controlo de acessos
- Integridade
  - Ausência de alterações não autorizadas ao sistema ou à informação
  - Verificada por meios criptográficos ou de controlo de acessos
- A política de segurança determina o que é autorizado ou não

- Disponibilidade
  - Prontidão do sistema para fornecer o serviço ou disponibilizar a informação

Vulnerabilidades

Classificação

- Projecto
  - Vulnerabilidade durante a fase de definição de requisitos e desenho da arquitetura.
  - Ex.: Não ter em conta todos os cenários onde a comunicação pode ser observada
- Codificação
  - Erro de código (bug) com implicações de segurança
  - Ex: validação insuficiente do *input*
- Operacional
  - Vulnerabilidade causada por erro de configuração ou pelo ambiente de execução
  - Ex.: contas sem palavras-passe

Vulnerabilidades

Ataques e correções

- Um ataque é uma ação maliciosa que ativa uma ou mais vulnerabilidade
- Um ataque com sucesso, usando um software de *exploit*, resulta numa intrusão
  - Ataque + Vulnerabilidade -> Intrusão
- A deteção e correção de erros faz parte do ciclo de desenvolvimento
  - Alguns estudos referem 15 a 50 erros por para cada 1000 linhas de código
- A publicação e correção das vulnerabilidades leva por vezes à construção de novos *exploits*
- As vulnerabilidades que são desconhecidas da empresa de software ou do público em geral são designadas de dia-zero (*0-day*)

Vulnerabilidades

Classes de vulnerabilidades

- *Common Weakness Enumeration* (CWE) é uma lista de tipos de vulnerabilidades
  - formato CWE-NNN, sendo NNN o número atribuído à classe
- CWE 2021 Top10

Rank	ID	Name	Score	2020 Rank Change
[1]	CWE-787	Out-of-bounds Write	65.93	+1
[2]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.84	-1
[3]	CWE-125	Out-of-bounds Read	24.9	+1
[4]	CWE-20	Improper Input Validation	20.47	-1
[5]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	19.55	+5
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	19.54	0
[7]	CWE-416	Use After Free	16.83	+1
[8]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.69	+4
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	14.46	0
[10]	CWE-434	Unrestricted Upload of File with Dangerous Type	8.45	+5

https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html

Vulnerabilidades

Catálogo de software/bibliotecas vulneráreis

- *Common Vulnerabilities Exposures* (CVE) é um catálogo de vulnerabilidades existentes em software comercial ou aberto
  - Formato CVE-AAAA-NNNN, sendo AAAA o ano em que foi catalogada e NNNN o número atribuído
- A maior parte são vulnerabilidades de projeto ou de codificação
- Vulnerabilidades são reportadas por uma *CVE Numbering Authority* (CNA)
  - Investigadores, empresas, centros de resposta a emergência (CERT), …
  - Atualmente 102 CNAs distribuídas por 17 países
- Exemplo: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-5789
  - *An integer overflow that leads to a use-after-free in WebMIDI in Google Chrome on Windows prior to 73.0.3683.75 allowed a remote attacker who had compromised the renderer process to execute arbitrary code via a crafted HTML page.*

Vulnerabilidades

Grau de gravidade

Common Vulnerability Scoring System (CVSS)

- Vetor de ataque (rede, rede adjacente, local, acesso físico)
- Complexidade do ataque (alta, baixa)
- Privilégios necessários (altos, baixos, nenhuns)
- Interação com o utilizador (nenhum, requerida)
- Âmbito (mesmo, outro)
- Impactos (confidencialidade, integridade, disponibilidade)
- Explorabilidade (código não disponível, prova de conceito, funcional)
- Nível de remediação (solução completa, correção temporária, não oficial)
- https://www.first.org/cvss/calculator/3.0

Ataques

Superfície de ataque

- A interface através da qual um sistema pode ser comprometido é designada *superfície de ataque*
- Ataques técnicos
  - Rede
  - Servidor aplicacional, ambientes de execução
  - Sistema operativo
  - Hardware
- Ataques de engenharia social
  - Utilizadores

Ataques

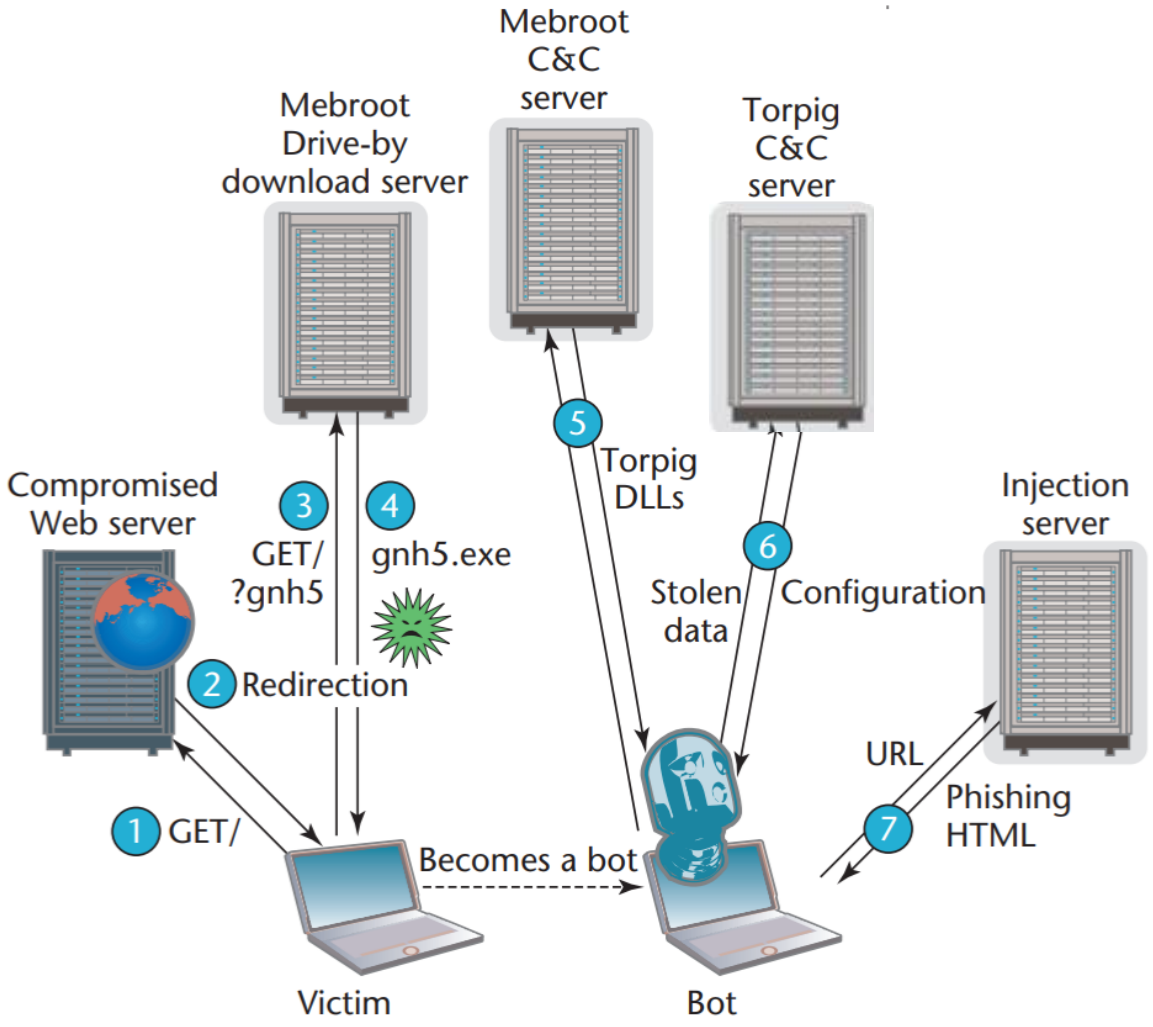
- Programas maliciosos (https://www.virustotal.com/)
  - *Malware*/vírus; Verme/*Worm;* Cavalo de Tróia
- Estes programas têm diferentes estratégias de ataque…
  - *Bot*: programa para obter dados ou executar comandos a partir do exterior (*backdoor*)
  - *Botnet*: rede de bots
  - *Command and Control Server*: controlo remoto da botnet
- … e diferentes objectivos
  - Roubo de identidade ou dados confidenciais
  - Criptomoedas
    - *Ransomware*: bot que espera comando para cifrar todos os dados do computador, entregando chave contra pagamento de criptomoedas
    - *Criptominers*: bot que consume ciclos de CPU minerando criptomoedas para entregar ao atacante

Lockheed Martin *kill chain*

Ordem	Fase	Descrição
1	**Reconhecimento** (*Reconnaissance*)	Procurar, identificar e selecionar os alvos.
2	**Armar** (*Weaponization*)	Ligação do malware com o payload que permite afetar o sistema alvo. Ex: Colocar em ficheiros PDF ou Word o código de ataque.
3	**Entrega** (*Delivery*)	Transmissão da "arma" para o alvo (Ex: Anexos email, Pen USB, Web sites visitados pelos alvos)
4	**Exploração** (*Exploitation*)	Uma vez entregue, a "arma" é ativada executando o código do atacante, explorando a superfície de ataque (sistema operativo, aplicações, utilizador, …)
5	**Instalação** (*Installation*)	A "arma" instala um *backdoor* no sistema alvo, permitindo o acesso permanente ao mesmo
6	**Comando e controlo** (*Command & Control*)	A partir do exterior da organização passa a haver acesso aos sistemas dentro da rede alvo
7	**Ações para o objetivo** (*Actions on Objective*)	O atacante tenta atingir os seus objetivos, os quais podem incluir roubo e destruição de dados ou intrusão em outros alvos

Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains, Lockheed Martin Corporation, 2010
https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/LM-White-Paper-Intel-Driven-Defense.pdf

Exemplo: Torpig



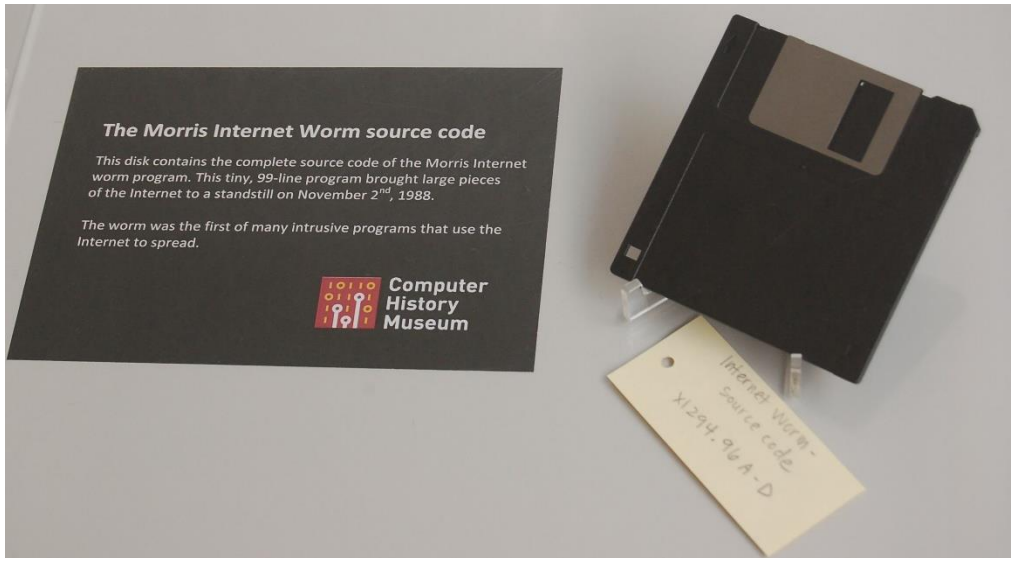
A sombreado os componentes controlados pelos atacantes

- 1: atacantes comprometem web sites vulneráveis
- 2 e 3: páginas modificadas redirecionam o browser da vítima para um *drive-by download*
- 4: a vítima descarrega e executa *Mebroot*, passando a ser um *bot*
- 5: o *bot* obtém os módulos Torpig
- 6: o *bot* faz upload dados roubados do computador da vítima
- 7: quando a vítima visita determinados sites, e páginas selecionadas, o bot obtém páginas de phishing que apresenta ao utilizador (ex: *login*)

https://sites.cs.ucsb.edu/~vigna/publications/2011_SPMagazine_torpig.pdf

ENISA – Threat report 2021

https://www.enisa.europa.eu/publications/enisa-threat-landscape-2021



Segurança e desenvolvimento de software

ISEL – Instituto Superior de Engenharia de Lisboa
Rua Conselheiro Emídio Navarro, 1 | 1959-007 Lisboa

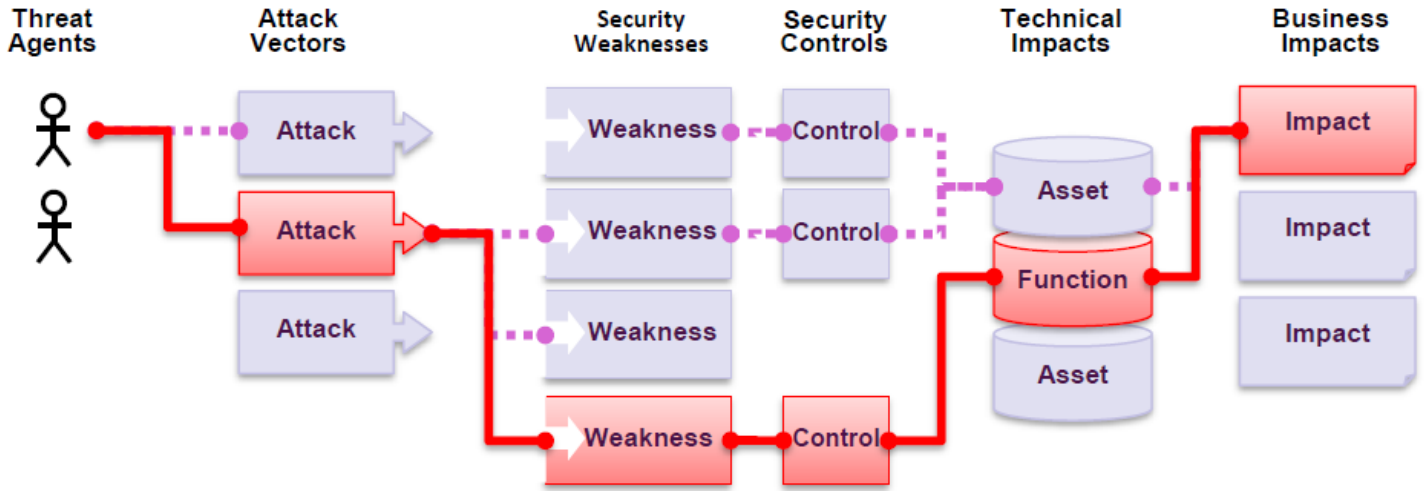
Avaliação de risco

- Quando se planeia software tem-se em conta
  - Funcionalidade, Usabilidade, Desempenho, Simplicidade, Desempenho, Baixo *time-to-market*
- Ter como objectivo desenvolver um sistema sem qualquer vulnerabilidade não é realista
- É preciso avaliar o ***risco***
  - *Risco = probabilidade x impacto*
- Probabilidade de explorar o risco
  - Exposição do sistema afetado, tipo de utilização
  - Grau de vulnerabilidade: Erros de projeto, código ou configuração
- Impacto
  - Impacto nas propriedades de segurança da informação: confidencialidade, integridade, disponibilidade
  - Impacto na reputação da organização

Metodologia de gestão de risco da OWASP

- Quadros normativos como ISO 27001 ou o *Open Web Application Security Project* (OWASP), propõem a seguinte metodologia para a classificação e mitigação de riscos
  1. **Identificar** a situação, os intervenientes e os ativos envolvidos
  2. Fatores para **estimar a probabilidade** de explorar o risco
  3. Fatores para **estimar o impacto**
  4. Determinar a **severidade** do risco
  5. Decidir o que **corrigir**
  6. **Adaptar** o modelo de classificação de risco

Metodologia de gestão de risco da OWASP



- Os agentes podem seguir vários caminhos para prejudicar o sistema
  - Diferentes níveis de motivação, graus de oportunidade, número de atacantes
- Os diferentes caminhos podem ser mais fáceis ou difíceis
  - Facilidade de descoberta e exploração, capacidade de detetar intrusos
- Cada vetor de ataque tem um impacto diferente no negócio
  - Perda: confidencialidade/integridade, dados pessoais, financeira, reputação

https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology#Step_1:_Identifying_a_Risk

https://www.portaldodpo.pt/blog/service/risco/

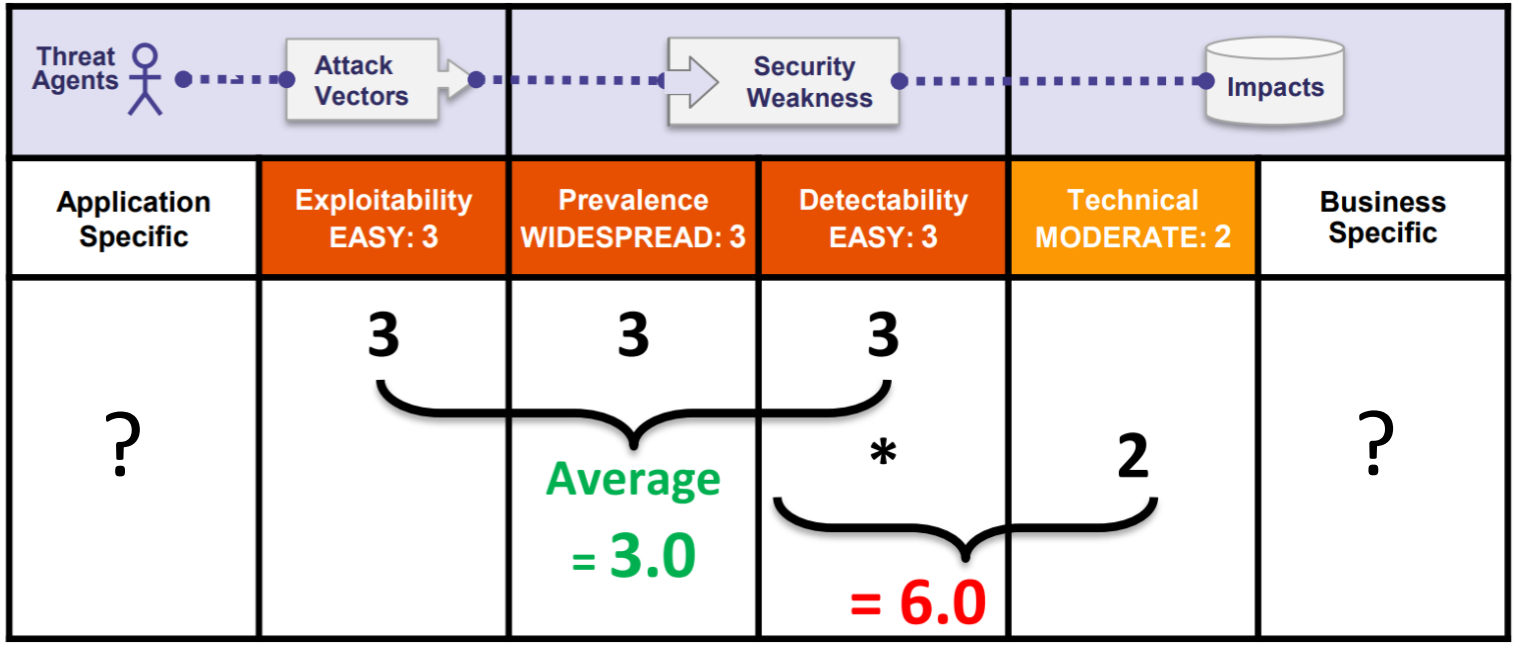
Metodologia de gestão de risco da OWASP

Threat Agents	Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts	Business Impacts
Appli-cation Specific	Easy: 3	Widespread: 3	Easy: 3	Severe: 3	Business Specific
	Average: 2	Common: 2	Average: 2	Moderate: 2	
	Difficult: 1	Uncommon: 1	Difficult: 1	Minor: 1	

- Níveis para estimar probabilidade e impacto
- Níveis mais altos vão dar origem a um nível maior de risco

Exemplo de cálculo de risco para a vulnerabilidade "Má configuração de segurança",

ex: os últimos *updates* disponíveis não estão corretamente instalados



https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology#Step_1:_Identifying_a_Risk

Menos potencial para vulnerabilidades, menos risco

- As linguagem de programação têm implicações na segurança do sistema
  - Linguagens nativas com gestão de memória manual têm maior desempenho mas maiores riscos
  - Linguagens de alto nível (Java, C#, Python) correm num ambiente controlado (*managed*) e com acesso regulado aos recursos nativos
  - O interpretador da linguagem Perl consegue verificar se os dados de entrada chegam a componentes críticos (*tainted mode*)
- Código fechado ou aberto
  - Código fechado (proprietário) mais robusto/profissional? Código livre (aberto) feito por criadores empenhados?
  - Argumento *many eyeballs*: código livre tem maior potencial para escrutínio. Mas ele acontece? Bugs com muitos anos como o ShellSock ou o Heartbleed mostram que nem sempre é assim
  - Argumento *security by obscurity*: código proprietário é mais seguro por ser desconhecido. Mas a frequência de erros descobertos nestes sistemas é elevada.

Evitar as vulnerabilidades mais comuns de projeto

https://ieeecs-media.computer.org/media/technical-activities/CYBSI/docs/Top-10-Flaws.pdf

1. Nunca assumir ou confiar
   - A interface de utilizador por si só não impede acessos a recursos protegidos
   - O armazenamento de chaves do lado do cliente pode levar à sua descoberta
2. Usar mecanismos de autenticação não contornáveis
   - Autenticadores não podem ser fáceis de forjar
   - Usar mais do que um elemento de autenticação
3. Autorizar após autenticar
   - O controlo de acesso feito com base na identidade ou papel na organização é essencial para garantir a confidencialidade, integridade e disponibilidade da informação

Evitar as vulnerabilidades mais comuns de projeto

4. Separar dados de instruções de controlo
   - Misturar dados e instruções de controlo está na base das vulnerabilidades mais comuns em aplicações web e nativas
5. Validar explicitamente todos os dados
   - Está relacionado com o anterior.
   - As aplicações fazem normalmente pressupostos sobre os inputs, é preciso validá-los.
6. Usar corretamente criptografia
   - Não usar algoritmos “caseiros”
   - Má gestão de chaves
   - Fontes de aleatoriedade fracas

Evitar as vulnerabilidades mais comuns de projeto

7. Identificar dados sensíveis e como devem ser tratados
   - Identificar corretamente os dados sensíveis é essencial para a sua proteção
8. Considerar sempre os utilizadores
   - Controlos de segurança suficientemente expressivos mas não excessivos
9. Perceber o impacto dos componentes exteriores na superfície de ataque
   - O uso, inevitável, de componentes externos resulta em herdar as suas fragilidades e limitações
   - Validar a proveniência e integridade do componente externo
10. Conceber antevendo alterações futuras
   - Ex: Alteração segura de partes da aplicação e das chaves

Segurança nos processos de desenvolvimento

SDL Agile

- Processos de desenvolvimento conhecidos com *ágeis* (ex: Extreme Programming, Scrum) são cada vez mais usados
- Organizado em *sprints* (período de 2 a 4 semanas) onde são desenvolvidas as *stories* (conjunto de tarefas), armazenadas no *backlog* do produto
- O consórcio SAFEcode (Software Assurance Forum for Excelence in Code) definiu um conjunto de *stories* e tarefas de segurança
  - Exemplos de *stories* para incorporar: a codificação de saídas é feita corretamente; verificação de índices de buffers; sincronização em acesso concorrente a recursos
  - Exemplos de tarefas operacionais: Usar as últimas versões compiladas; aplicar os patchs disponíveis; rever cuidadosamente código de maior risco
  - Tarefas de especialistas: treinar segurança de codificação e testes; realizar testes de penetração; realizar testes de fuzzing

Segurança nos processos de desenvolvimento

http://safecode.org/wp-content/uploads/2018/01/SAFECode_Agile_Section2a-tables.pdf

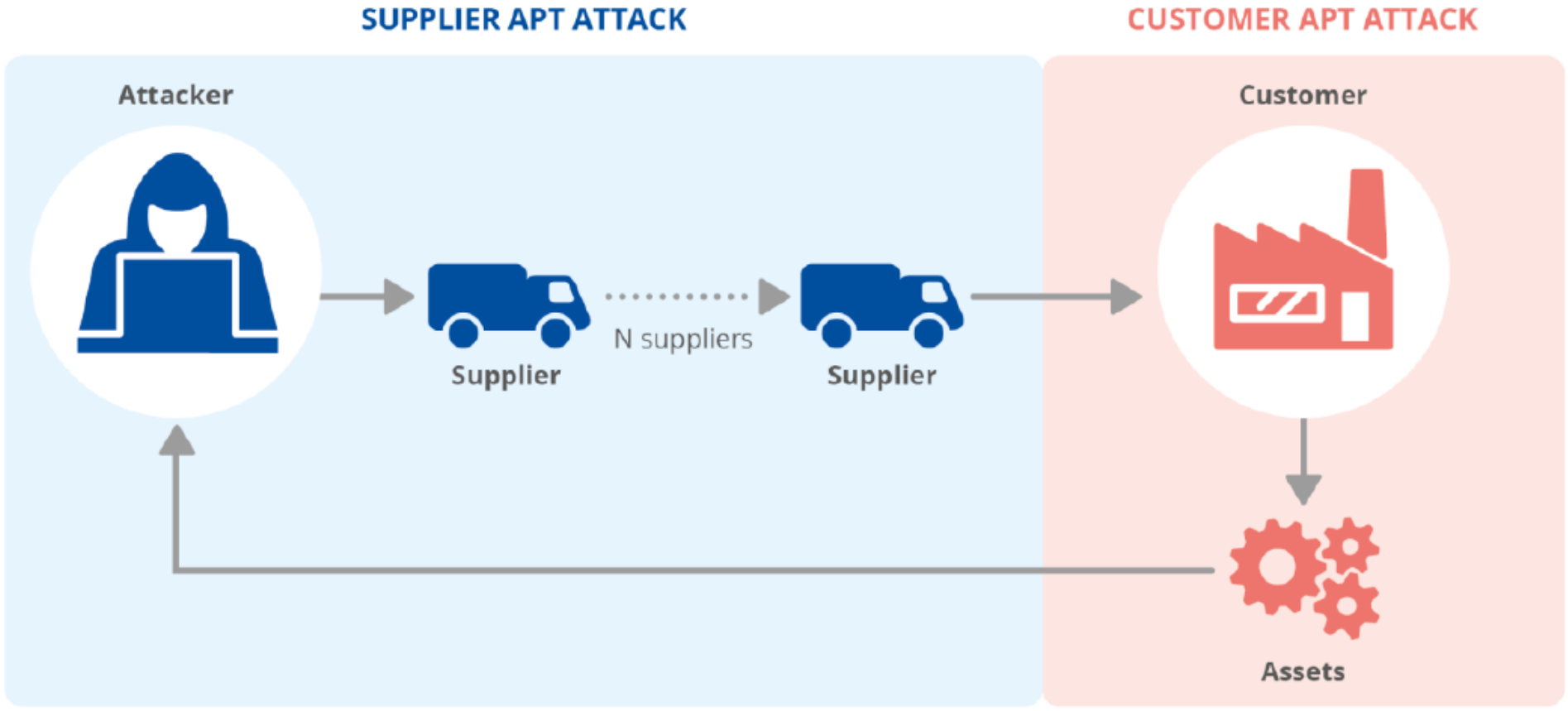
3	As a(n) architect/ developer, I want to ensure **AND** as QA, I want to verify application of or access within index boundaries of buffers and arrays	**[A/D]** Define where buffer operations (on dynamic buffers) occur. Define data types and bounds for buffer operations. **[D]** Adhere to SAFECode's Fundamental Practices for Secure Software Development for prevention of buffer overflows. **[D]** Scan source code for such violations using static code analyzer tools, e.g., Coverity. **[A/D]** Conduct false positive analysis of flagged issues. **[D]** Fix buffer overflow issues analyzed as confirmed. **[T]** Use fuzz testing tools to verify that no process/system crashes/hangs exist. If they do, fix them and re-run the tool.	• Minimize Use of Unsafe String and Buffer Functions • Use a Current Compiler Toolset • Use Static Analysis Tools	CWE-120 CWE-131 CWE-805

SAFECode guidelines

- Conjunto abrangente de recomendações de projeto e codificação (link)
- Arquitetura
  - Princípios vistos anteriormente; Modelação de ameaças
  - Estratégia para cifrar informação (ex: gestão de chaves)
  - Usar mecanismos/protocolos de autenticação *standard;* Estabelecer políticas para logs
- Código
  - Usar convenções de código e boas práticas (OWASP – Secure Coding Practices, Secure Coding Guidelines for Java SE)
  - Usar funções seguras
  - Usar ferramentas para detetar problemas de segurança cedo no desenvolvimento
  - Validar inputs; Tratar erros, dando apenas informação detalhada em *logs* internos

Segurança na cadeira de fornecimento de software

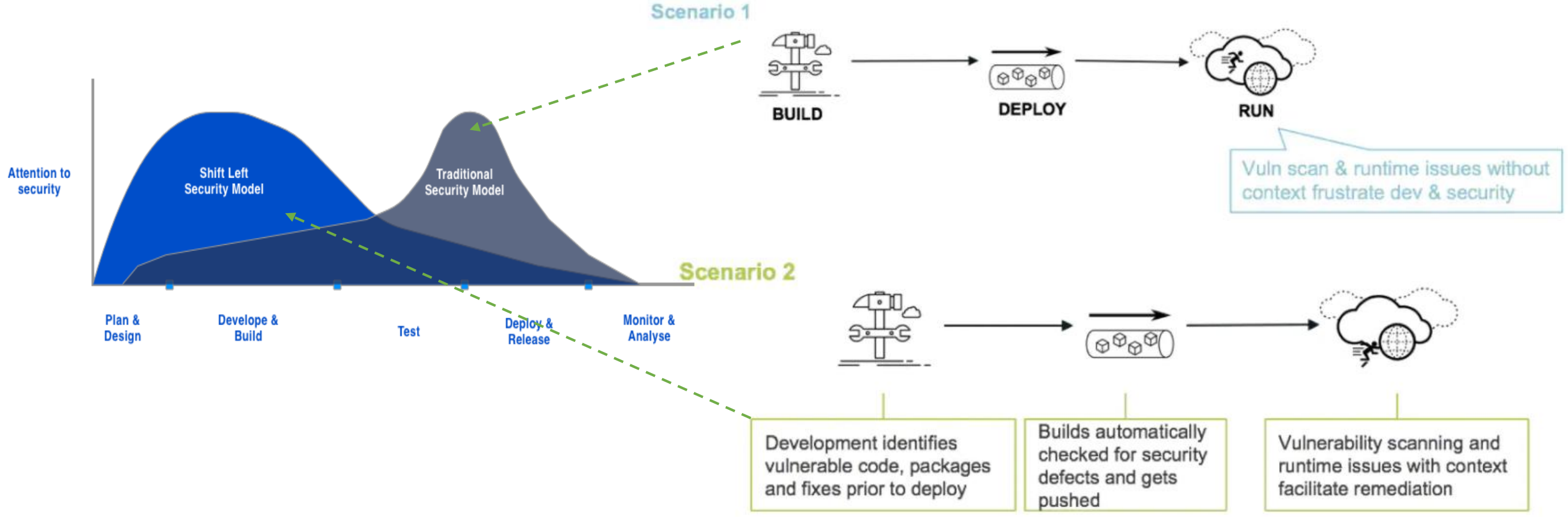
- Na cibersegurança, a cadeia de distribtuição envolve uma ampla gama de recursos (hardware e software), armazenamento (nuvem ou local), mecanismos de distribuição (aplicações web, lojas online), software de gestão.



https://www.enisa.europa.eu/publications/threat-landscape-for-supply-chain-attacks

Resumo – *Shift Left*

- Introduzir verificações de segurança tão cedo quanto possível no ciclo de desenvolvimento
  - Procurar bugs típicos no código; analisar fluxos de dados sensíveis; procurar vulnerabilidades em dependências; testar dinamicamente injentado falhas;



https://www.paloaltonetworks.com/blog/2019/07/4-practical-steps-shift-left-security/
https://devopedia.org/shift-left