Resumão de Análise e Exploração de Vulnerabilidades

Vulnerabilidades

Uma é uma **fraqueza** no sistema que permite ao atacante violar uma politica de segurança (**CIA**).

O crescimento de vulnerabilidades é **proporcionalmente direto** ao crescimento de software.

Podem existe devido a **bugs** ou **falhas**:

- **Bug**: Erro de implementação do software
- **Falha**: Erro de design or arquitetónico do software

CIA

- **Confidencialidade**: Assegurar que as informações sejam acessíveis apenas para aqueles que têm autorização para fazê-lo.
- **Integridade**: Garantir que as informações não sejam alteradas ou destruídas de forma não autorizada.
- **Disponibilidade**: Garantir que os utilizadores autorizados tenham acesso às informações e aos recursos relacionados sempre que necessário.

CWE (Common Weakness Enumeration)

Lista extensa de anti-padrões que podem levar a sistemas inseguros.

CVE (Common Vulnerability and Exposures)

Documento que visa declarar a existência de uma falha ou vulnerabilidade de um sistema numa determinada versão.

CVSS (Common Vulnerability Scoring System)

Pontuação usada para determinar a severidade de uma vulnerabilidade.

A pontuação é dada pelos seguintes grupos:

- Base Metric Group: Qual a severidade da vulnerabilidade?
- Temporal Metric Group: Qual a severidade da vulnerabilidade no momento?
- Environmental Metric Group: Qual a severidade da vulnerabilidade no meu ambiente?

Base Score formula

É feito usando o **ISS** (Impact Subscore), do **ESS** (Exploitability Subscore) e o **Impacto**.

Divulgação da vulnerabilidade

A divulgação de uma vulnerabilidade pode ser feita de duas formas:

- **None**: A vulnerabilidade não é divulgada

- **Coordinated**: O analista informa o fabricante da vulnerabilidade, o fabricante implementa e distribui a correção.
- **Full**: O analista divulga a vulnerabilidade sem informar o fabricante. O fabricante é pressionado a implementar e distribuir a correção.

Avaliação de vulnerabilidades em redes

Análise de vulnerabilidades

O processo de procurar e analisar novas vulnerabilidades.

É pertinente para:

- **Product Owners**: Para garantir que o produto é seguro
- **Desenvolvedores**: Para perceber o que criou a vulnerabilidade e como pode ser evitada
- **Administradores**: Para perceber como pode ser mitigada a vulnerabilidade
- **Analistas**: Para perceber como pode ser explorada a vulnerabilidade

Avaliação de vulnerabilidades

O processo de analisar, avaliar e rever entidades.

Assessment vs Audit

- **Audit**: Determina a conformidade de um standard.
- **Assessment**: Determina o quão mau/bom algo se encontra.

Existem 3 tipos de avaliação:

- **Black Box**: Não tem acesso ao código fonte nem à arquitetura do sistema.
- **White Box**: Tem acesso ao código fonte e à arquitetura do sistema.
- **Grey Box**: Tem acesso ao código fonte ou à arquitetura do sistema (alguma informação).

Penetration Testing

Semelhante à avaliação, mas o o seu foco é analisar a infraestrutura **a partir de fora** (emula o atacante).

Scope

O scope é o conjunto de entidades (sistemas, software, endpoints) que serão analisadas.

WAF (Web Application Firewall)

Filtra pedidos HTTP e impede que a execução de pedidos indevidos.

IDS (Sistema de deteção de intrusão)

Conjunto de ferramentas que monitorizam a rede e alertam sobre possíveis ataques (e.g. Windows Defender).

Firewall

Dispositivo que controla o tráfego de dados definindo regras de acesso e filtragem de pacotes.

Tipos de Ataques

Estes são alguns dos tipos de ataques que podem ser feitos a um sistema:

- **Ativos** : corre software para descobrir/testar a rede e o sistema
- **Passivos** : corre software para monitorizar o tráfego da rede
- **Externos**: foca-se em exposições públicas
- **Internos**: foca-se em exposições internas
- **Host-Based** : foca-se nas más configurações/permissões existentes no software
- **Rede**: foca-se em comunições das infraestruturas da rede
- **Aplicação**: foca-se em explorar vulnerabilidades de aplicações (erros lógicos, autenticação, bases de dados, etc.)
- **Wireless**: foca-se em atacar comunições de redes sem fios

Life-cycle de vulnerabilidades

1. Estabelecer as bases: definir os assets e definir prioridades.
2. Avaliar vulnerabilidades: procurar vulnerabilidades dentro do scope, contruir um relatório detalhado (responde às perguntas: O que foi encontrado? Quais são as entidades afetadas? Quais são as recomendações para tradar do mesmo?) sem necessariamente explorar essa vulnerabilidade num todo (não é um teste de penetração).
3. Avaliar risco: quantificar o impacto que um determinado risco tem, para melhor decidir qual vulnerabilidade priorizar.
4. Remedição: corrigir ou minimizar o impacto das vulnerabilidades que encontram-se no software.
5. Verificação: verificação da eficácia das correções feitas.
6. Monitorização: analisar algumas falhas que não foram ou foram corrigidas (pode envolver configuração de firewalls, IDS/NIDS/HIDS).

Enumeração

É o processo de descobrir informação sobre o sistema.

Estas informações podem vir de:

- **Erros**: Por vezes, mensagens de erro podem conter informação sensível (versão do software, nome de utilizadores, etc.)
- **Portas**: As portas abertas podem indicar que serviços estão a correr na máquina.
- **Banner**: Alguns serviços podem ter banners que contêm informação sobre o serviço.
- **Cookies**: Alguns cookies enviados por pedidos HTTP contêm informação sobre o server stack (tipo de framework usada, etc.)
- **OS Fingerprinting**: Alguns serviços podem ter banners que contêm informação sobre o sistema operativo.

Para mitigar estas informações, é recomendado:

- **Desativar mensagens de erro ou torná-las genéricas**

- **Manter abertas apenas as portas necessárias**
- **Restringir o acesso a banners ou usar banners forjados**
- **Detetar scanners de portas utilizando regras de firewall**

Injeção de comandos

Este tipo de vulnerabilidade ocorre quando um software constroi um comando (ou parte dele) a partir de dados de entrada externos, não validando os mesmos.

Pode trazer problemas de:

- Confidencialidade;
- Controlo de acesso;
- Integridade;
- Não repúdio.

Pitfalls

- **Confiar em dados de entrada**: Nunca confiar em dados de entrada, mesmo que sejam provenientes de uma fonte confiável.
- **Confiar em sistemas internos ou APIs**: Nunca confiar em sistemas internos ou APIs, pois estes podem ser comprometidos.
- **Confiar em dados provenientes da base de dados**: Nunca confiar em dados provenientes da base de dados, pois estes podem ser comprometidos.
- **Ignorar como os dados são usados externamente**: Nunca ignorar como os dados são usados externamente, pois estes podem ser usados para explorar vulnerabilidades.

Propriedades ACID

A informação deve ser:

- **Atomica**: Todas as operações devem ser executadas ou nenhuma.
- **Consistente**: A informação deve estar sempre consistente.
- **Isolada**: As operações devem ser isoladas.
- **Durável**: A informação deve ser persistente.

SQLi (SQL Injection)

Este tipo de vulnerabilidade ocorre quando um software constroi uma query SQL (ou parte dela) a partir de dados de entrada externos, não validando os mesmos.

Contém os seguintes **tipos**:

- **Inband**: O atacante usa o mesmo canal de comunicação que a vítima.
  - **Error-based**: O atacante usa mensagens de erro para obter informação sobre a base de dados.
  - **Union-based**: O atacante usa a cláusula `UNION` para obter informação sobre a base de dados.
  - **Blind**: O atacante usa a cláusula `WHERE` para obter informação sobre a base de dados.
    - **Boolean-based**: O atacante usa a cláusula `WHERE` para obter informação sobre a base de dados.

Time-based: O atacante manipula a cláusula WHERE introduzindo atrasos temporais na execução da consulta para inferir informações sobre a base de dados com base no tempo de resposta.

- **Out-of-band**: O atacante usa um canal adicional para filtrar informação da base de dados.

Para **mitigar esta vulnerabilidade**, é recomendado:

- Sanitizar os dados de entrada;
- Usar *prepared statements*;

OS Command Injection

Esta vulnerabilidade permite aos atacantes executar comandos no sistema operativo. Isto acontece porque não é feita uma neutralização de elementos especiais usados em comandos.

Command Injection pode ser feita de algumas formas:

Command Override: A aplicação aceita o input e usa-o para selecionar qua programa executar, bem como que comandos usar.

Argument Injection: Um payload pode executar comandos do utilizador antes ou depois do comando previsto, modificando os argumentos.

Exemplo:

```
<?php
  $host = $_GET['host']; // ex: host=localhost; rm -rf /
  $command = 'ping -c 4 ' . $host; // ex: ping -c 4 localhost; rm -rf /
  system($command);
```

Variáveis de ambiente: A execução de comandos é afetada por variáveis de ambiente.

Exemplo:

```
$ ls -la
-rw-r--r--  1 root  root  0 Jan  1  1970 ls
$ export PATH=.:$PATH
$ ls -la
Código malicioso
```

Shell Shock

A vulnerabilidade ShellShock permite que um atacante execute comandos arbitrários em um servidor que esteja executando o Bash como shell.

CGI: Common Gateway Interface: Especificação para executar programas externos, tipicamente em servidores web.

Parameter Expansion: Permite que a Shell expanda variáveis usando um padrão de substituição.

```
$ ls *
File.txt
$ touch -- '-la'
$ ls
-la File.txt
$ ls * # O comando será ls -la File.txt
-rw-r--r--  1 root  root  0 Jan  1  1970 File.txt
```

Mitigação

Para mitigar esta vulnerabilidade, é recomendado:

- **Nunca executar comandos do sistema operativo por uma aplicação**;
- **Ser cuidadoso com dependências**;
- **Caso seja necessário a execução de comandos, primeiro processar todos os dados de entrada**;

Broken Authentication

Esta vulnerabilidade permite o comprometimento de credenciais, tokens de sessão ou chaves. Isto acontece pois as funcionalidades relacionadas com autenticação e gestão de sessões são implementadas de forma incorreta.

Autenticação

Visa determinar a identidade de uma entidade. Isto é feito através de uma verificação, usando:

- **Algo que a entidade sabe**: e.g. password
- **Algo que a entidade tem**: e.g. cartão de cidadão
- **Algo que a entidade é**: e.g. impressão digital

MFA e 2FA

MFA: Autenticação de múltiplos fatores **2FA**: Autenticação de dois fatores

HTTP

O HTTP é um protocolo **stateless**, ou seja, não guarda informação sobre o estado de uma entidade.

Os pedidos HTTP podem ser:

- **GET**: Obter informação
- **POST**: Enviar informação
- **DELETE**: Apagar informação
- ...

Sessões

As sessões são usadas para manter o estado de uma entidade (e.g preferências do utilizador, credenciais, etc.). Isto é feito através de um identificador único, que é usado para identificar a entidade.

Podem ser guardados como:

- **Cookies**: Guardados no cliente (na *cookie jar*) e reenviados em cada pedido HTTP.
- **JSON Web Tokens (JWT)**: Guardados no cliente, provém mecanismos para *token refresh*, limitando o impacto de um *token* comprometido.
- **SESSION_ID**: Variável definida no servidor, que é enviada para o cliente e reenviada em cada pedido HTTP.
- ...

Use of URL

O URL é usado para identificar recursos na web. No entanto, este pode conter informação sensível (e.g. credenciais), deve ser evitado.

Broken Authentication exploração

O que torna uma aplicação vulnerável:

- As credencias podem ser adivinhadas ou rescritas devido a fracas funções de gestão de contas;

O que permite a:

- Ataques de força bruta (*brute force attacks*);
- Ataques de dicionário (*credential stuffing*);

Para mitigar esta vulnerabilidade, é recomendado:

- Evitar o uso de palavras-passe o mais possível;
- Usar um armazenamento seguro (e.g. PKBDF2, scrypt, etc.);
- Adicionar delays após tentativas falhadas;
- Usar MFA/2FA;

Session Hijacking

Ocorre quando um atacante obtém acesso a uma sessão válida de um utilizador.

Pode ser feito de algumas formas:

- **Sniffing/Interceptação**: O atacante obtém o identificador da sessão através de interceptação de tráfego.
- **Brute Force**: O atacante tenta adivinhar o identificador da sessão, por este motivo as variáveis de sessão deve ter grande entropia.
- **Session Fixation**: O atacante força o utilizador a usar um identificador de sessão conhecido.

Session Fixation

Ocorre quando um atacante força um utilizador a usar um identificador de sessão conhecido.

Pode ser feito de algumas formas:

- **Pre-generated SID**: O atacante obtem um SID válido e força o utilizador a usá-lo, através de um link malicioso. Após o utilizador aceder ao link, o atacante pode usar o SID para aceder à sessão.

- **Cross-domain cookie**: O atacante força o utilizador a usar um SID válido para aquele domínio, através de um link malicioso. Após o utilizador aceder ao link, o atacante pode usar o SID para aceder à sessão.

XSS (Cross-site scripting)

Ocorre quando um atacante consegue injetar código malicioso no lado do cliente.

Existem 3 tipos de XSS:

- DOM-based: Código malicioso é executado no cliente, sem envolvimento do servidor.
- Reflected: Código injetado é refletido no cliente, muitas vezes na resposta do servidor.
- Stored: Código malicioso é armazenado no servidor e refletido no cliente.

Tem um impacto:

- Moderado para DOM-based e Reflected XSS;
- Severo para Stored XSS;

Para mitigar esta vulnerabilidade, é recomendado:

- **Synchronized tokens**: Usar tokens para validar pedidos HTTP;
- **Cookie-to-header**: Quando estabelecida a conexão, o servidor envia um cookie para o cliente, que é reenviado em cada pedido HTTP;
- **SameSite cookies**: Cookies que só são enviados para o mesmo domínio;
- **Double submit cookies**: São usadas duas cookies, session cookie (indentifica o utilizador durante o tempo de sessão) e CSRF cookie (muda dinamicamente a cada pedido) para validar pedidos HTTP;

CSRF (Cross-site request forgery)

Ocorre quando um atacante consegue fazer com que um utilizador execute ações indesejadas no lado do servidor.

Funciona da seguinte forma:

1. O utilizador acede a um site malicioso;
2. O site malicioso faz um pedido HTTP para o servidor;
3. O servidor responde ao pedido HTTP.
4. O utilizador acede ao site malicioso e o atacante consegue obter informação sobre o utilizador.

Same Origin Policy

Política de segurança que impede que scripts de um domínio acedam a recursos de outro domínio. De modo a prevenir ataques de CSRF e XSS.

CORS (Cross-Origin Resource Sharing)

Mecanismo que permite que recursos de um domínio sejam acedidos por outro domínio. Por padrão o Same Origin Policy restrige solicitações de recursos de uma origem diferente. O CORS é uma maneira de relaxar essas restrições e permite que servidores especifiquem quais origens estão autorizadas a acessar seus recursos.

O Pre-flight (pedido enviado antes do pedido HTTP) não é necessário se:

- O método HTTP é GET, HEAD ou POST via XHR;
- Body é um texto limpo;
- Não há cabeçalhos personalizados;

O servidor responde com um CORS header, que contém:

- **Access-Control-Allow-Origin**: Domínios que podem aceder ao recurso;
- **Simple Request**: GET, HEAD, POST;

CORS apresenta alguns riscos, como:

- **Permissão Universal**: uso do * (universal) allow nos headers;
- A aplicação que permite CORS pode ser vulnerável a CSRF;
- **Misplaced trust**: Dados trocados por dois sites é baseado em confiança mútua, caso um dos sites seja comprometido, o outro também será;
- **Access Control based on Origin**: O Origin Header indica de onde é feito o pedido HTTP, no entanto, este pode ser forjado;

Concorrência

Ambientes distribuidos geralmente causam operações concorrentes, onde duas ou mais operações são executadas simultaneamente.

Condições de corrida

Ocorre quando duas ou mais operações tentam manipular o mesmo recurso ao mesmo tempo.

Serialização

É o processo de garantir que apenas uma operação é executada de cada vez.

Dado um conjunto de transações, dois escalonamentos são equivalentes se:

- **Read-Write**: Se as transações que leem o mesmo item, leem o mesmo valor;
- **Write-Write**: Se as transações que escrevem o mesmo item, escrevem o mesmo valor;

TOCTOU (Time-of-check to time-of-use)

É uma vulnerabilidade onde o atacante pode aproveitar-se de uma janela entre o tempo de um recurso ser verificado e o tempo de uso desse mesmo recurso.

Para mitigar esta vulnerabilidade, é recomendado:

- **Afirmar que as ações são serializadas como esperado**: pode exigir conhecimento da camada inferior;
- **Forçar a serialização manualmente** (para BDs e outros objetos partilhados);
- **Se possível, enviar macro-operações para sistemas (transações completas) que bloqueiam recursos na fonte**;
- **Reduzir o uso de nomes de arquivos para uma única chamada e, em seguida, usar descritores de arquivos**;

Covert Channels

Um canal secreto é um caminho que pode ser usado para transferir informação de uma forma não projetada pelos desenvolvedores.

Microarchitectural covert channels: são canais secretos que exploram o comportamento do hardware.

MeltDown

É um ataque que explora a vulnerabilidade de execução especulativa, que permite que um atacante leia a memória do kernel.

Spectre

Semelhante ao Meltdown, mas explora a vulnerabilidade de execução especulativa, que permite que um atacante leia a memória de outros processos.

Para mitigar ambas as vulnerabilidades, é recomendado:

- Para sistemas remotamente expostos (browser, network), limitar a precisão do temporizador;
- Para sistemas locais, atualizações de microcódigo e kernel;

Buffer Overflow

Ocorre quando um programa tenta escrever mais dados num buffer do que o buffer pode armazenar (if input > buffer).

Hoje em dia não acontece tanto pois:

- Existem linguagens de programação que não permitem o acesso direto à memória;
- Foram criadas ferramentas para detetar este tipo de vulnerabilidade;

Os softwares vulneráveis a este tipo de vulnerabilidade são:

- Qualquer software que receba informação de fontes externas: sockets, Pipes, ficheiros, etc.;
- Qualquer software desenvolvido em linguagens com acesso direto à memória: C, C++, etc.;

Estrutura de memória 101

O Kernel organiza a memória em páginas (4096 bytes). Os processos operam no espaço de memória virtual.

O Kernel divide o programa em segmentos:

- **Stack Segment**: Armazena variáveis locais e argumentos de funções.
- **Shared Libraries Segment**: Armazena bibliotecas partilhadas.
- **Heap Segment**: Armazena variáveis dinâmicas.
- **BSS Segment**: Armazena variáveis globais não inicializadas.
- **Data Segment**: Armazena variáveis constantes.
- **Code Segment**: Instruções do programa.

Stack

É uma estrutura de dados que armazena informação de forma LIFO (Last In First Out). A Stack está organizada em frames, que contém:

- **Return information**: Endereço de retorno, argumentos, etc.
- **Local variables**: Variáveis locais e temporárias.
- **Arguments to following functions**: Argumentos para funções chamadas.

Return information tem dois objetivos principais:

- **Chaining frames**: Permite que o programa volte ao estado anterior.
- **Return to the next instruction**: Permite que o programa continue a executar.

A stack pode ser subvertida de modo a conduzir a possiveís ataques, como:

- **DoS** (Denial of Service): o programa crasha.
- **Memory Disclosure**: o programa revela informação sensível.
- **Mudança de fluxo de execução**;
- **Injeção de código malicioso**;

RBP (Return Base Pointer): Ele é um ponteiro que aponta para o endereço de retorno da função atual. Ele é usado para restaurar o estado da função anterior.

RIP (Return Instruction Pointer): Ele é um ponteiro que aponta para a próxima instrução a ser executada.

Estratégia de ataque:

- **Sobrescrever buffer sobre RBP/RIP** (Return Base Pointer/Return Address), mudando qualquer um destes valores, o ataque pode direcionar a execução do programa para um código malicioso.

Frame Chaining vs Function Chaining

- **Frame Chaining**: Refere-se à manipulação do RBP (Return Base Pointer), envolve a manipulação do ponteiro de base da pilha para controlar o fluxo de execução do programa.
- **Function Chaining**: Refere-se à manipulação do RIP (Return Instruction Pointer), envolve a manipulação do endereço de retorno na pilha para direcionar a execução do programa para sequências específicas de instruções ou gadgets durante um ataque (ROP).

Medidas de proteção

Data Execution Prevention (DEP)

É uma tecnologia de segurança que impede a execução de código em áreas de memória não designadas para execução.

Canaries

É um valor que é colocado antes do endereço de retorno, que é verificado antes de retornar da função. Se o valor for alterado, o programa crasha.

Exemplo: (Variaveis locais) -> (Canary) -> (Endereço de retorno)

ROP (Return Oriented Programming)

É uma técnica que permite que um atacante modificar a pilha de execução de um programa para os ponteiros de retorno apontarem para sequências de instruções existentes no programa.