



Segurança de Software

Ameaças à segurança
informática - com enfoque na
segurança de software

V1.2 nov 2023

Ameaças aos sistemas informáticos

- Dados em repouso
 - Furto (através de acesso físico)
 - Destruição
 - Exfiltração
 - Encriptação (ransomware)
- Dados em trânsito
 - Escuta de dados
 - Alteração de dados
 - MITM - Man in the middle
- Dados em processamento
 - Intrusão
 - Acesso indevido
 - Alteração de dados
 - Exfiltração
 - Destruição
 - Software malicioso
 - Negação de serviço



Alguns conceitos genéricos

❑ Vulnerabilidade

- É uma falha ou fraqueza de configuração ou de conceção das redes, da infraestrutura computacional ou das aplicações que um atacante pode explorar para fazer um ataque informático

❑ Exploração (*Exploit*)

- É a forma como um atacante usa uma vulnerabilidade para lançar um ataque informático
- Também pode ser descrita como um vetor de ataque (attack vector)

❑ Superfície de ataque

- A superfície de ataque de um sistema ou ambiente de software é a soma de todos os pontos que um atacante pode tentar usar para lançar ataques informáticos
- Para maximizar a segurança informática, a superfície de ataque deve ser reduzida ao mínimo

❑ Risco

- O risco corresponde à probabilidade de atacantes usarem exploits para se aproveitarem das vulnerabilidades, e provocarem danos na organização
- O nível de segurança de um sistema informático deve ser um compromisso entre o risco, o valor dos ativos informáticos e o custo da implementação das medidas de segurança

CWE - Common Weakness Enumeration

- A lista de CWE é uma lista de tipos de fraquezas típicas encontradas em software e hardware
- Cada CWE representa uma fraqueza genérica - não identifica produtos específicos que a possuem
- Serve como uma referência comum para identificar fraquezas típicas
- Esta referência é usada por ferramentas e guias que explicam como as prevenir e mitigar

699 - Software Development

- + C API / Function Errors - (1228)
- + C Audit / Logging Errors - (1210)
- + C Authentication Errors - (1211)
- + C Authorization Errors - (1212)
- + C Bad Coding Practices - (1006)
- + C Behavioral Problems - (438)
- + C Business Logic Errors - (840)
- + C Communication Channel Errors - (417)
- + C Complexity Issues - (1226)
- + C Concurrency Issues - (557)
- + C Credentials Management Errors - (255)
- + C Cryptographic Issues - (310)
- + C Key Management Errors - (320)
 - V Use of Hard-coded Cryptographic Key - (321)
 - B Key Exchange without Entity Authentication - (322)
 - V Reusing a Nonce, Key Pair in Encryption - (323)
 - B Use of a Key Past its Expiration Date - (324)
- + C Data Integrity Issues - (1214)
- + C Data Processing Errors - (19)
- + C Data Neutralization Issues - (137)
- + C Documentation Issues - (1225)

A lista: <https://cwe.mitre.org/data/definitions/699.html>

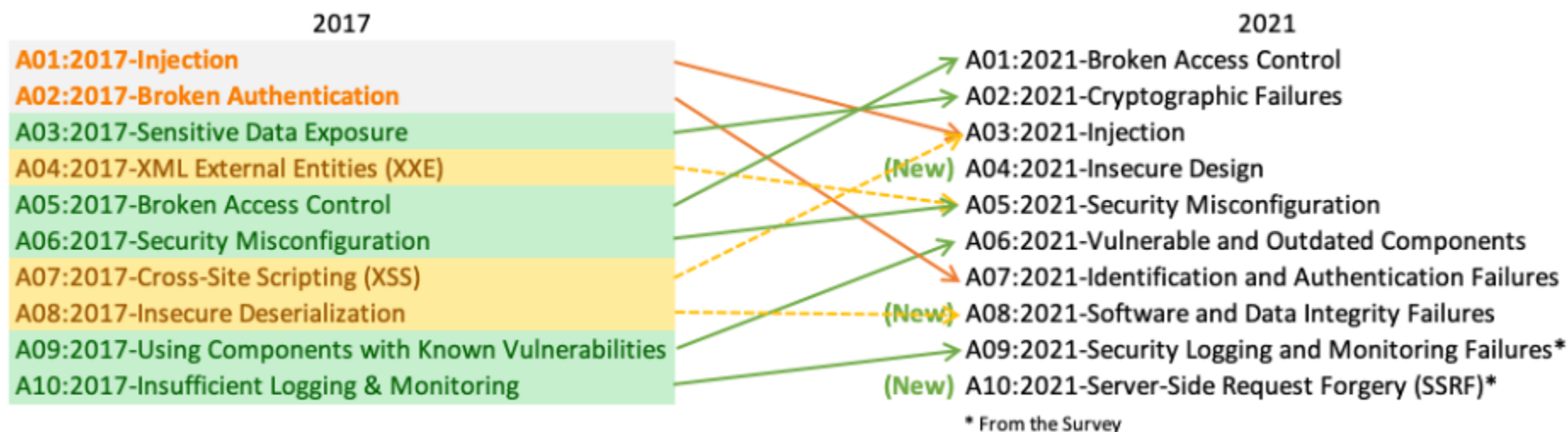
CWE - Top 25 2023

2023 CWE Top 25					
Rank	ID	Name	Score	CVEs in KEV	Rank Change vs. 2022
1	CWE-787	Out-of-bounds Write	63.72	70	0
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.54	4	0
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	34.27	6	0
4	CWE-416	Use After Free	16.71	44	+3
5	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	15.65	23	+1
6	CWE-20	Improper Input Validation	15.50	35	-2
7	CWE-125	Out-of-bounds Read	14.60	2	-2
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.11	16	0
9	CWE-352	Cross-Site Request Forgery (CSRF)	11.73	0	0
10	CWE-434	Unrestricted Upload of File with Dangerous Type	10.41	5	0
11	CWE-862	Missing Authorization	6.90	0	+5
12	CWE-476	NULL Pointer Dereference	6.59	0	-1
13	CWE-287	Improper Authentication	6.39	10	+1
14	CWE-190	Integer Overflow or Wraparound	5.89	4	-1
15	CWE-502	Deserialization of Untrusted Data	5.56	14	-3
16	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	4.95	4	+1
17	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	4.75	7	+2
18	CWE-798	Use of Hard-coded Credentials	4.57	2	-3
19	CWE-918	Server-Side Request Forgery (SSRF)	4.56	16	+2

CWE Top 25: <https://cwe.mitre.org/top25/>

Segurança de aplicações Web - OWASP Top 10

- A fundação OWASP (Open Web Application Security Project®) trabalha com o objetivo de melhorar a segurança de aplicações Web
- O OWASP Top 10 é um documento de referência que descreve as 10 principais preocupações críticas de segurança de aplicações Web



Fonte: <https://owasp.org/www-project-top-ten/>

Vulnerabilidades: são do domínio público

As vulnerabilidades são difundidas publicamente, através das equipas CERT – Computer Emergency Readiness Teams

- Em Portugal, esta tarefa compete ao Centro Nacional de Cibersegurança CERT.PT
- As vulnerabilidades são organizadas em bases de dados públicas, e identificadas por um código CVE - **Common Vulnerability and Exposure**
- Os CVE referem-se a uma vulnerabilidade de um determinado produto específico

The screenshot shows the NVD website interface. At the top, it says 'An official website of the United States government' and 'Here's how you know'. The NIST logo is prominent, along with 'NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY' and 'U.S. DEPARTMENT OF COMMERCE'. A 'NVD MENU' button is in the top right. Below the header, it says 'Information Technology Laboratory' and 'NATIONAL VULNERABILITY DATABASE'. A large 'NVD' logo is on the right. The main content area shows 'There are 1,321 matching records. Displaying matches 1 through 20.' Below this is a table of results. The table has columns for 'Vuln ID', 'Summary', and 'CVSS Severity'. The first four rows are visible, all showing a severity of 9.8 CRITICAL.

Vuln ID	Summary	CVSS Severity
CVE-2022-43029	Tenda TX3 US_TX3V1.0br_V16.03.13.11_multi_TDE01 was discovered to contain a stack overflow via the time parameter at /goform/SetSysTimeCfg. Published: October 19, 2022; 3:15:10 PM -0400	V3.1: 9.8 CRITICAL V2.0:(not available)
CVE-2022-43028	Tenda TX3 US_TX3V1.0br_V16.03.13.11_multi_TDE01 was discovered to contain a stack overflow via the timeZone parameter at /goform/SetSysTimeCfg. Published: October 19, 2022; 3:15:10 PM -0400	V3.1: 9.8 CRITICAL V2.0:(not available)
CVE-2022-43027	Tenda TX3 US_TX3V1.0br_V16.03.13.11_multi_TDE01 was discovered to contain a stack overflow via the firewallEn parameter at /goform/SetFirewallCfg. Published: October 19, 2022; 3:15:10 PM -0400	V3.1: 9.8 CRITICAL V2.0:(not available)
CVE-2022-43026	Tenda TX3 US_TX3V1.0br_V16.03.13.11_multi_TDE01 was discovered to contain a stack overflow via the endIp parameter at /goform/SetPtpServerCfg. Published: October 19, 2022; 3:15:10 PM -0400	V3.1: 9.8 CRITICAL V2.0:(not available)

As principais vulnerabilidades em software

1. Falhas no controlo de acesso (autorização)
2. Falhas no uso de criptografia
3. SQL Injection
4. Cross Site Scripting (XSS) e Cross Site Request Forgery (CSRF)
5. Escrita fora dos limites (overflow)
6. Design inseguro
7. Configuração errada de segurança
8. Utilização de componentes desatualizados e vulneráveis
9. Falhas de identificação e autenticação
10. Falhas de integridade de dados e software
11. Falhas no logging e monitorização





1 - Falhas no controlo de acesso (autorização)

Falhas no controlo de acesso

- Esta fraqueza é provocada pela ausência de uma tática de segurança durante a fase de arquitetura e design
- Esta fraqueza ocorre quando a aplicação não verifica se o utilizador tem a devida autorização **em todos** os recursos restritos, incluindo URLs, scripts e ficheiros
- Nos casos mais graves, o atacante pode ter acesso a determinados recursos, sem estar sequer autenticado
- Os criadores de aplicações com esta fraqueza assumem erradamente que estes recursos só podem ser acedidos se for seguido um determinado caminho de navegação, e por isso só aplicam o controlo de autorização em certos pontos desse caminho



Falhas no controlo de acesso

- Um ataque que consiga obter acesso direto à aplicação através de um URL específico que não tem controlo de acesso, pode ter os seguintes impactos:

- Leitura e modificação de dados
- Execução de código ou comandos
- Assumir outra identidade ou elevar privilégios

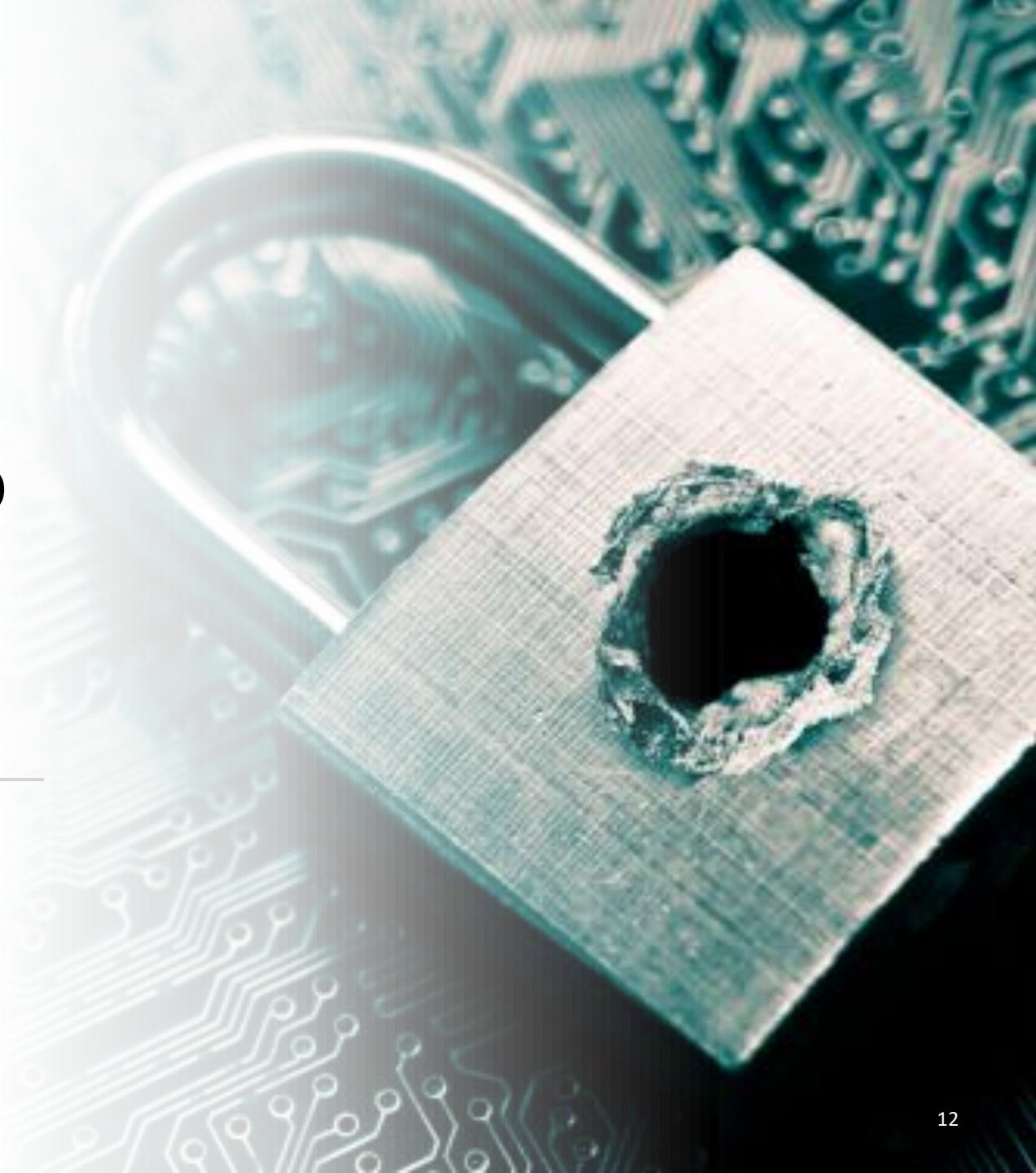
- Esta fraqueza pode ser evitada se forem aplicados controlos de acesso e autorização a todos os recursos restritos.

- Estes controlos podem ser aplicados automaticamente se for usada uma framework de desenvolvimento MVC, como por exemplo Apache Struts





2 - Falhas no uso de criptografia



Falhas no uso de criptografia

- Muitas aplicações lidam com dados sensíveis, incluindo senhas, cartões de crédito, dados de saúde, dados pessoais, segredos empresariais, etc
- Estes dados devem ser protegidos, em repouso, em trânsito e em processamento
- Existe legislação que obriga à encriptação de dados sensíveis, nomeadamente o RGPD da UE ou o PCI DSS (área financeira)
- Esta fraqueza é provocada quando não é usada criptografia para proteger os dados sensíveis ou quando esta é mal usada



Criptografia: algumas falhas

- Dados sensíveis transmitidos em claro (clear text), através de um protocolo não encriptado como HTTP, SMTP ou FTP, mesmo que seja apenas nos componentes backend
- Uso de métodos criptográficos antigos ou fracos, como por exemplo DES, MD5 ou SHA1
- Falha na gestão de chaves: uso de chaves predefinidas, chaves fracas, chaves codificadas no código fonte (hardcoded). Falta de uma política de rotação de chaves.
- Uso de modos de encriptação inseguros como ECB. Utilização de vectores de inicialização fracos ou repetidos. Utilização de encriptação simples quando deveria ser usada encriptação autenticada para assegurar integridade e autenticidade.
- Uso de um gerador de números pseudo aleatórios fraco
- Uso de senhas como chaves, em vez de as chaves serem derivadas das senhas
- Falha na validação de certificados digitais e de toda a cadeia de certificação
- Falha na verificação de assinatura digital ou autenticador MAC

3 - SQL Injection



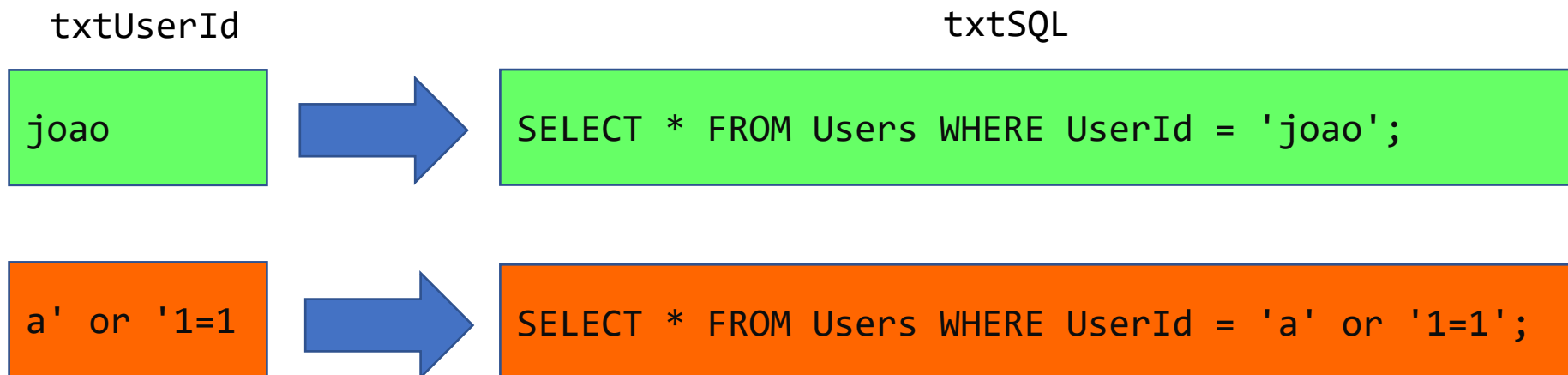
Ataques a aplicações Web: Injeção de SQL (SQL Injection)

❑ Esta vulnerabilidade ocorre no servidor:

- Quando os dados vindos dos browsers não são validados/limpos nos scripts da aplicação Web no servidor
- E quando posteriormente são (mal) construídas queries SQL com esses dados, normalmente através de concatenação de código SQL com os dados vindos do browser

Exemplo simples de um script vulnerável:

```
txtUserId = getRequestString("UserId");  
  
txtSQL = "SELECT * FROM Users WHERE UserId = '" + txtUserId + "'";
```



Injeção de SQL (SQL Injection): alguns códigos para login

- Diferentes sistemas de gestão de base de dados têm sintaxes ligeiramente diferentes, por isso um ataque que funcione com MySQL pode não funcionar com Oracle e vice versa.
- Cada sistema tem uma sintaxe própria de ataque.
- Exemplos de alguns códigos para injeção de SQL em páginas de login:

```
admin' --
```

```
admin' #
```

```
admin'/*
```

```
' or 1=1--
```

```
' or 1=1#
```

```
' or 1=1/*
```

```
') or '1'='1--
```

```
') or ('1'='1--
```

Injeção de SQL (SQL Injection): apagar dados

- Um exemplo em que um atacante poderia apagar uma tabela da base de dados

```
txtUserId = getRequestString("UserId");  
txtSQL = "SELECT * FROM Users WHERE UserId = '" + txtUserId + "'";
```

txtUserId

```
a' or '1=1'; DROP TABLE Users;
```



txtSQL

```
SELECT * FROM Users WHERE UserId = 'a' or '1=1'; DROP TABLE Users;
```

Nota: nem todas as bases de dados suportam *stacking* de comandos SQL

Como evitar ataques de injeção de SQL

- ❑ **VALIDAR TODOS** os dados que vêm dos browsers, apagando ou codificando (escaping) caracteres não autorizados como por exemplo ' ou " ou ; ou \
 - As linguagens de programação Web normalmente têm funções específicas para limpar os dados
- ❑ **NUNCA** construir as queries SQL dinâmicas através da concatenação de strings.
 - Deve ser usada a técnica de "prepared statements", "parameterized queries" ou "stored procedures"
 - Exemplo de query SQL com escaping e prepared statement, em PHP/MySQL:

```
$username = mysqli_real_escape_string($db_connection, $_POST['username']);  
  
$stmt = $dbConnection->prepare('SELECT * FROM Users WHERE UserId = ?');  
$stmt->bind_param('s', $username); // 's' significa o tipo => 'string'  
  
$stmt->execute();
```

Nota: com outras bases de dados, a sintaxe é diferente

Injeção de SQL: efeitos do escaping dos dados de entrada

```
$user_original = $_POST['username'];  
  
$user_sanitizado = mysqli_real_escape_string($db_connection, $user_original );
```

Se o utilizador introduzir no campo username o seguinte texto: `a' or '1=1`

A variável `$user_original` vai ficar com: `a' or '1=1`

Mas a variável `$user_sanitizado` vai ficar com: `a\' or \'1=1`

E finalmente a query SQL vai ficar assim: `SELECT * FROM Users WHERE UserId = 'a\' or \'1=1';`

Desta forma o atacante não vai conseguir introduzir plicas na query SQL

O símbolo de escaping `\` significa que aquilo que aparece a seguir não é para ser interpretado de forma literal. Já devem ter usado em programação o famoso `\n` em que o `n` não significa a letra `n`, mas sim uma nova linha.

Neste caso, o escaping da plica indica que essa plica não serve para abrir ou fechar uma string, mas sim que essa plica faz parte integrante da própria string

Injeção de SQL: efeitos do prepared statement

```
$stmt = $dbConnection->prepare('SELECT * FROM Users WHERE UserId = ?');  
$stmt->bind_param('s', $username); // 's' significa o tipo => 'string'
```

A técnica de prepared statement permite criar a estrutura das queries, deixando espaços vazios para os parâmetros, que são depois lá colocados posteriormente, indicando o seu tipo.

Os parâmetros entram em blocos já depois da estrutura da query estar preparada, pelo que o conteúdo dos parâmetros nunca é confundido com a estrutura da query. Os símbolos que aparecem no parâmetro são sempre interpretados como fazendo parte do parâmetro e nunca da estrutura da query.


Desta forma, um atacante não consegue alterar a estrutura da query.

A query preparada fica assim: `SELECT * FROM Users WHERE UserId =` 

Se a variável username tiver o seguinte texto: `a' or '1=1`

A query que vai ser executada é: `SELECT * FROM Users WHERE UserId =` 

Ou seja, as plicas que entram no parâmetro nunca serão interpretadas como símbolo para abrir ou fechar uma string.



XSS

4 - Cross Site
Scripting (XSS)
e Cross Site
Request
Forgery (CSRF)

Cross-site Scr

Ataques a aplicações Web: Cross Site Scripting (XSS)

- ❑ Esta vulnerabilidade ocorre quando scripts maliciosos são injetados em web sites legítimos e benignos
- ❑ É possível que os scripts sejam depois executados nos browsers dos utilizadores desse web site legítimo
 - Os browsers não têm forma de saber que os scripts são maliciosos pois eles foram descarregados de um web site legítimo e confiável
 - Os scripts maliciosos têm acesso aos cookies, session tokens, ou outra informação sensível mantida pelo browser e usada com aquele site legítimo

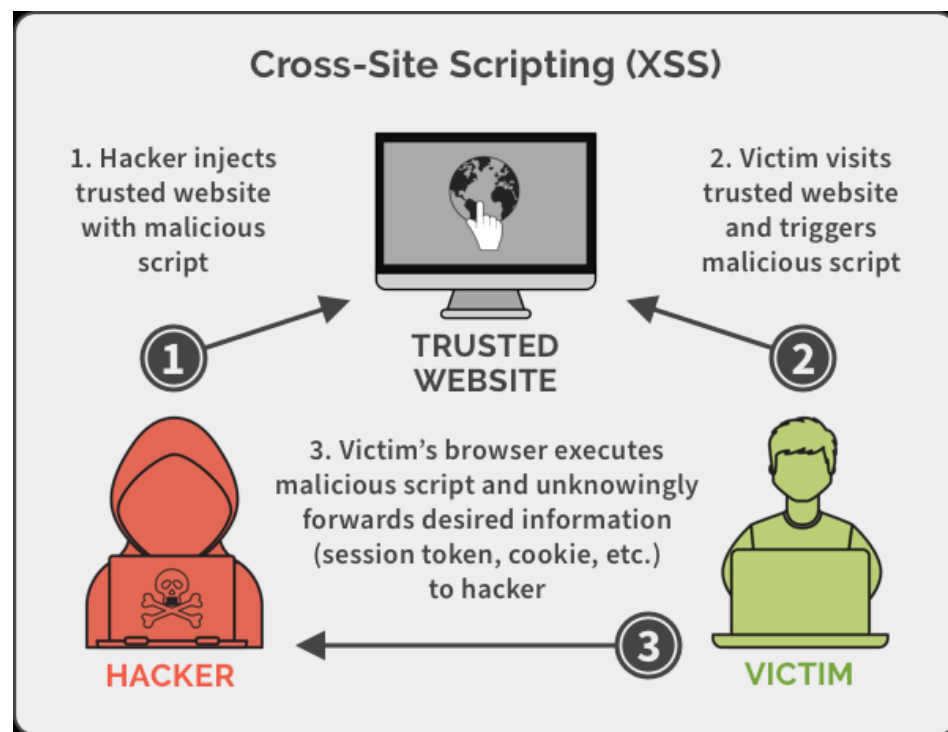


imagem: securitynewspaper.com

Tipos de ataques Cross Site Scripting (XSS)

❑ Stored XSS (AKA Persistent or Type I)

- Quando o script malicioso é armazenado numa base de dados do servidor, num forum, num log de visitantes, num espaço de comentários

❑ Reflected XSS (AKA Non-Persistent or Type II)

- Quando o script malicioso não é armazenado, mas enviado através do URL como parâmetro, e depois provoca uma resposta imediata do servidor sob a forma de uma mensagem de erro, resultados de pesquisa, ou outro tipo de resposta que incorpore esse script malicioso.
- O alvo é levado a clicar num URL para um website legítimo (onde o utilizador está logado) especialmente construído, que contém o código javascript embebido, por exemplo num parâmetro de pesquisa

❑ DOM Based XSS (AKA Type-0)

- Quando o script malicioso não é enviado dentro do HTML (nas respostas do servidor), mas no próprio URL dos pedidos (tipicamente após o sufixo #)
- O script é depois construído e executado em run time no browser, quando alguma página produza código baseado nas variáveis DOM `document.URL`, `document.documentURI`, `location.href`, `location.search`, `location.*`, `window.name` ou `document.referrer`
- Este tipo de ataque é recente, complicado de resolver e estudos mostraram que mais de 50% dos web sites estão vulneráveis a algum tipo de DOM XSS
- A sanitização realizada no servidor não faz absolutamente nada contra um ataque deste tipo

Como evitar ataques Cross Site Scripting (XSS)

- ❑ Há tantas formas de criar ataques XSS que não existem receitas universais simples para proteger um web site
- ❑ As soluções envolvem sempre muito cuidado em introduzir nas páginas HTML conteúdo gerado pelos utilizadores
- ❑ Este tipo de conteúdo deve ser sempre validado e sanitizado antes de ser introduzido nas páginas HTML
- ❑ Deve ser evitada a utilização das variáveis DOM referidas no slide anterior
- ❑ A organização OWASP tem um bom conjunto de regras que devem ser cumpridas - XSS prevention cheat sheet

❑ https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

Sanitização de dados do utilizador

- ❑ Devem ser usadas as funções próprias da linguagem ou plataforma de desenvolvimento
- ❑ As bases de dados também costumam ter funções próprias para sanitizar os dados, antes de eles serem introduzidos na base de dados
- ❑ Também podem ser construídas funções à medida, para bloquear caracteres proibidos, ou para apenas permitir caracteres permitidos
- ❑ O tamanho dos dados também deve verificado e limitado

Ataques a aplicações Web: Cross Site Request Forgery (CSRF)

- ❑ Este ataque leva a vítima a fazer pedidos ilegítimos construídos por um atacante, em aplicações Web onde estão autenticados
- ❑ Desta forma, o atacante pode levar a vítima a praticar ações involuntárias em sites legítimos
- ❑ Na maior parte das vezes, são usadas técnicas de phishing para levar a vítima a clicarem em links que contêm o pedido ilegítimo
- ❑ O servidor não tem forma de saber se o pedido foi voluntário ou não

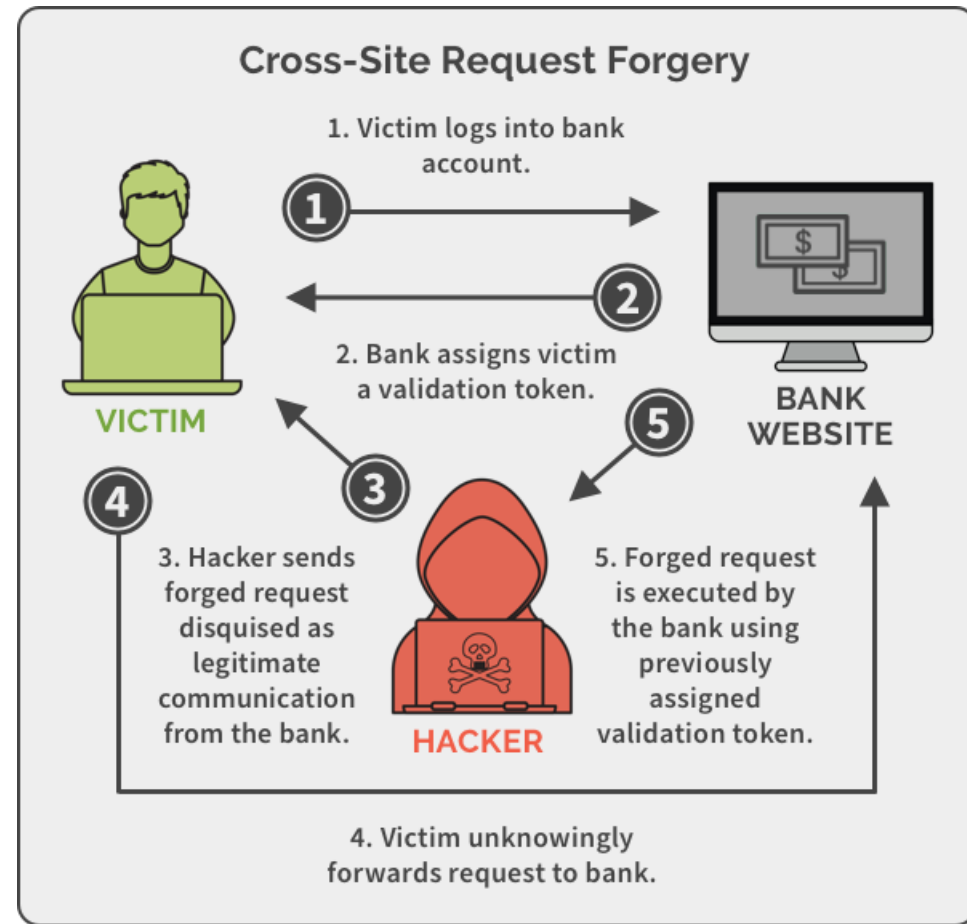


imagem: medium.com

Como evitar ataques Cross Site Request Forgery (CSRF)

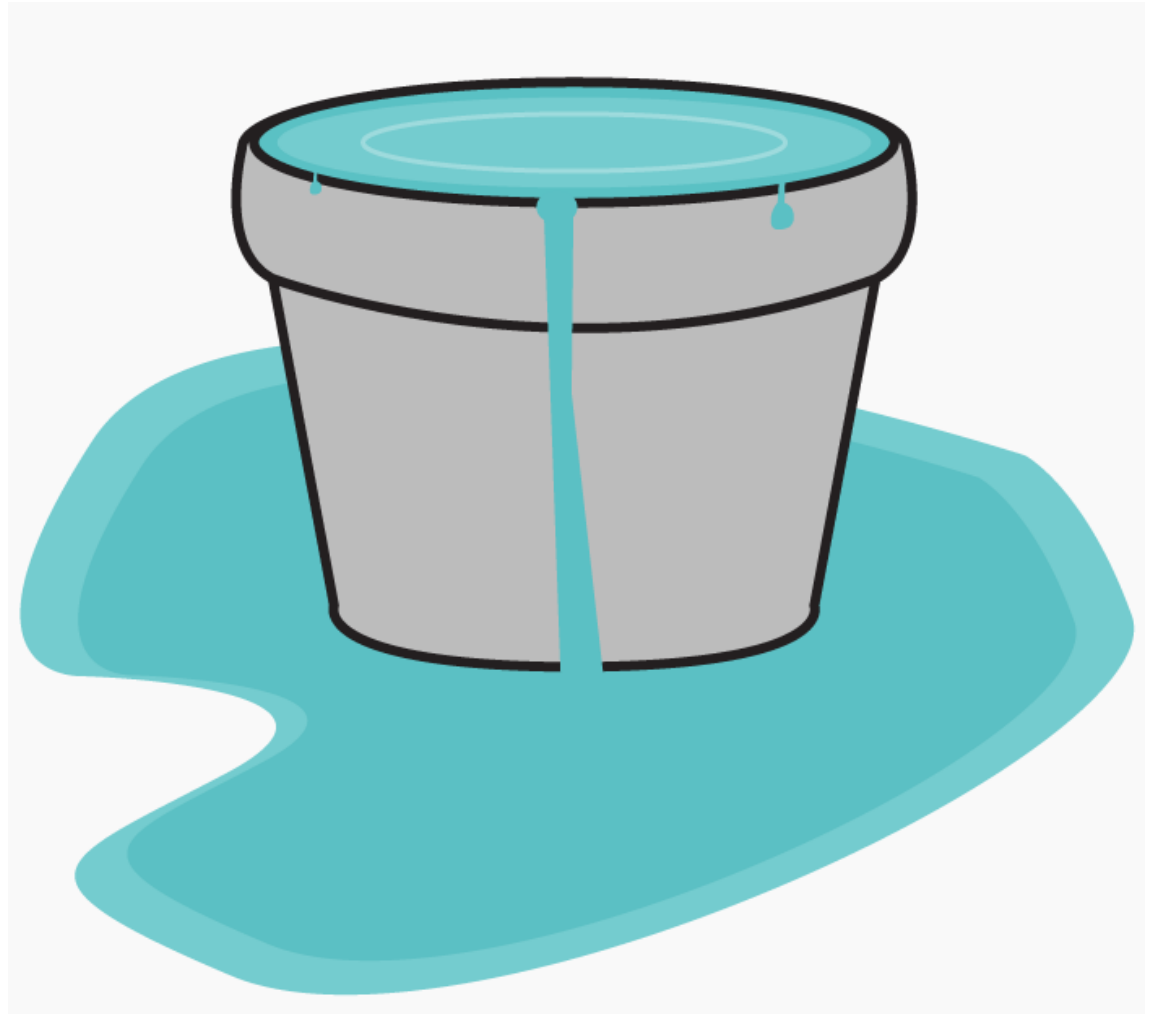
❑ Através da técnica "Synchronizer token pattern"

- Para cada sessão, é gerado um token aleatório no servidor (que complementa o identificador de sessão - session id), que é depois embebido em todas as páginas HTML. O servidor guarda este token numa variável de sessão.
- Todos os pedidos feitos depois pelo cliente têm que enviar este token e o servidor ignora pedidos sem token ou com token incorreto
- Um atacante remoto não será capaz de adivinhar este token, logo não conseguirá forjar pedidos válidos

❑ Através da técnica "Double submit cookie pattern"

- Com esta técnica, também é gerado um token aleatório no servidor, mas não é guardado no servidor, é apenas enviado para o cliente dentro de um cookie.
- A seguir, todos os pedidos do cliente têm que enviar não só este cookie que contém o token (é automático) mas também o token dentro de um parâmetro HTTP
- O servidor só tem que comparar este cookie com o parâmetro HTTP - se forem iguais o pedido é legítimo
- Mais uma vez, um atacante remoto não será capaz de adivinhar este token, logo não conseguirá forjar pedidos válidos

5 - Escrita
fora dos
limites
(overflow)

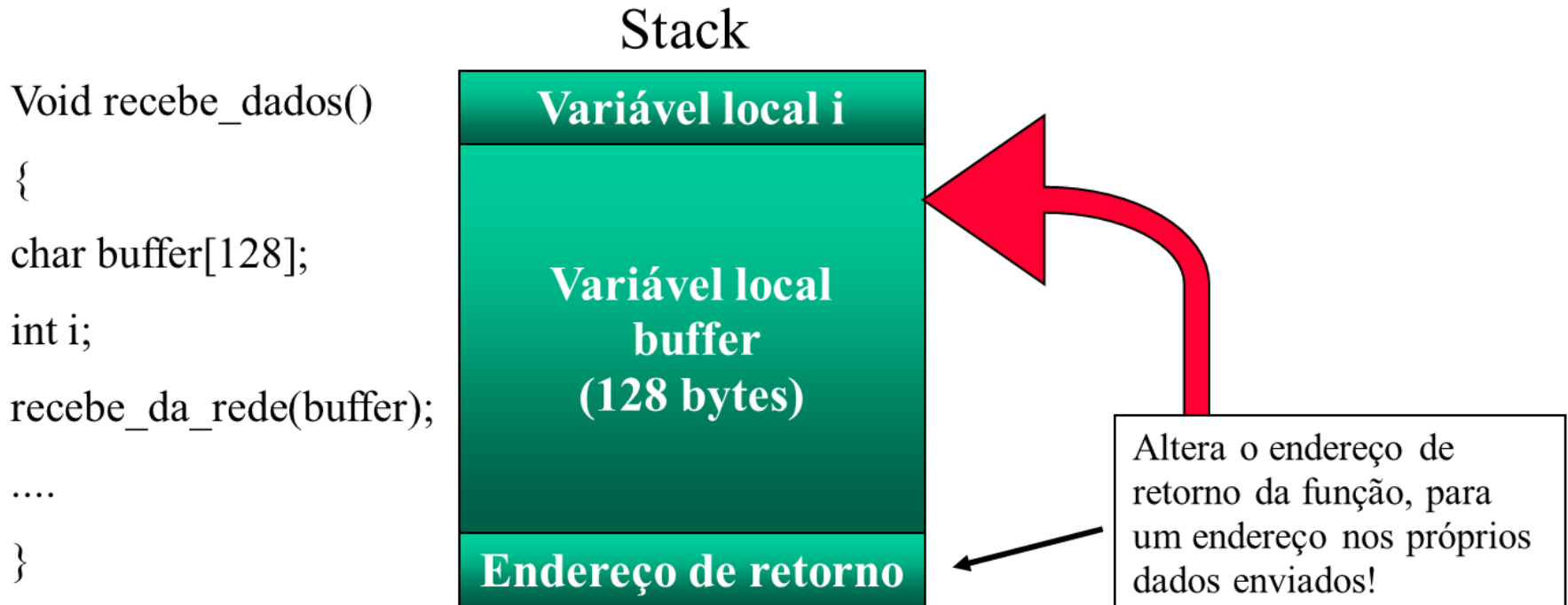


A vulnerabilidade do tipo buffer overflow

- ❑ Esta vulnerabilidade é das mais exploradas e das mais perigosas para penetrar remotamente em sistemas. Exige um conhecimento profundo do *hardware* e *software*
- ❑ A técnica foi utilizada com sucesso em 1988 no famoso '*Internet Worm*' que se disseminou em sistemas UNIX através do serviço '*Finger*' que era vulnerável a este tipo de ataque e continua ativa
- ❑ Os sistemas abertos, como o Linux são mais vulneráveis a este tipo de ataque porque o código é conhecido
- ❑ A resolução deste bug passa por uma programação mais cuidadosa e responsável (de preferência utilizando linguagens de alto nível como o Java)

Buffer overflow: como funciona (exemplo stack overflow)

- Ataque a serviços com permissões elevadas
- O atacante envia dados com tamanhos superiores ao *buffer*, o código não está preparado para evitar isso, e os dados vão corromper a *stack*, re-escrevendo a zona onde se encontra o endereço de retorno da execução do programa
- Se o novo caminho de execução do código máquina for direccionado para os dados, e estes dados possuírem código máquina executável, esse código é executado
- O código pode instruir o sistema operativo para criar um utilizador com permissões elevadas, ou outra qualquer operação de elevadas permissões



Buffer overflow: são descobertos novos casos quase todas as semanas

NVD - Results

https://nvd.nist.gov/vuln/search/results?form_type=Basic&results_type=overview&query=overflow&search_type=last3months

VULNERABILITIES **SEARCH AND STATISTICS**

Search Results (Refine Search)

Sort results by: Publish Date Descending **Sort**

Search Parameters:

- Results Type: Overview
- Keyword (text search): overflow
- Search Type: Search Last 3 Months

There are **438** matching records.
Displaying matches **1** through **20**.

Vuln ID	Summary	CVSS Severity
CVE-2021-30022	There is a integer overflow in media_tools/av_parsers.c in the gf_avc_read_pps_bs_internal in GPAC 1.0.1. pps_id may be a negative number, so it will not return. However, avc->pps only has 255 unit, so there is an overflow, which results a crash. Published: April 19, 2021; 4:15:14 PM -0400	V3.x:(not available) V2.0:(not available)
CVE-2021-30020	In the function gf_hevc_read_pps_bs_internal function in media_tools/av_parsers.c in GPAC 1.0.1 there is a loop, which with crafted file, pps->num_tile_columns may be larger than sizeof(pps->column_width), which results in a heap overflow in the loop. Published: April 19, 2021; 4:15:14 PM -0400	V3.x:(not available) V2.0:(not available)
CVE-2021-30019	In the adts_dmx_process function in filters/reframe_adts.c in GPAC 1.0.1, a crafted file may cause ctx->hdr.frame_size to be smaller than ctx->hdr.hdr_size, resulting in size to be a negative number and a heap overflow in the memcpy. Published: April 19, 2021; 4:15:14 PM -0400	V3.x:(not available) V2.0:(not available)
CVE-2021-30014	There is a integer overflow in media_tools/av_parsers.c in the hevc_parse_slice_segment function in GPAC 1.0.1 which results in a crash. Published: April 19, 2021; 4:15:14 PM -0400	V3.x:(not available) V2.0:(not available)
CVE-2021-29279	There is a integer overflow in function filter_core/filter_props.c:gf_props_assign_value in GPAC 1.0.1. In which, the arg const GF_PropertyValue *value,maybe value->value.data.size is a negative number. In result, memcpy in gf_props_assign_value failed. Published: April 19, 2021; 4:15:14 PM -0400	V3.x:(not available) V2.0:(not available)
CVE-2021-31255	Buffer overflow in the abst_box_read function in MP4Box in GPAC 1.0.1 allows attackers to cause a denial of service or execute arbitrary code via a crafted file. Published: April 19, 2021; 3:15:18 PM -0400	V3.x:(not available) V2.0:(not available)

Buffer overflow: porque ocorrem estes bugs ?

- ❑ A linguagem 'C' é difícil de usar com *strings* e *buffers*
 - O C++ é muito mais fácil, mas muitos programadores continuam a escrever em C, mesmo em compiladores C++.
- ❑ Ainda existe muito código legado
- ❑ Pressão no sentido de lançar as aplicações no mercado
- ❑ Descuido
- ❑ Falta de testes de qualidade do *software*

Buffer overflow: como evitar estes bugs ?

- ❑ Evitar o uso de C e C++. Usar de preferência linguagens de programação de alto nível como JAVA, Kotlin, Javascript, etc, e frameworks de desenvolvimento
- ❑ A possibilidade de ocorrerem buracos em software aumenta com a sua complexidade, por isso deve ter-se em mente a máxima KISS: Keep It Simple, Stupid
- ❑ Validar todas as entradas. Ter sempre cuidado com o tamanho e tipo de dados.
- ❑ Realizar testes exaustivos antes de colocar as aplicações em produção
- ❑ KISS! KISS! KISS! KISS! KISS! KISS! KISS! KISS!

Integer overflow

- Este problema ocorre quando o valor numérico de uma operação aritmética sai fora da gama de números que podem ser representados por uma variável (overflow)
- Por exemplo, uma variável do tipo unsigned int de 32 bit, só pode representar números até 4 294 967 295. Qualquer operação que produza um valor superior, vai provocar um overflow.
- Em caso de overflow, uma vez que o valor armazenado na variável é diferente do que deveria ser, podem ocorrer todo o tipo de problemas
 - Por exemplo, se essa variável for usada para controlar o tamanho de buffers de memória, pode acontecer que seja alocada muito menos memória do que a pretendida

Integer overflow - exemplo em C

```
nresp = packet_get_int();

if (nresp > 0) {
    response = xmalloc(nresp*sizeof(char*));

    for (i = 0; i < nresp; i++)
        response[i] = packet_get_string(NULL);
}
```

- Nota: `sizeof(char*)` devolve usualmente o valor 4
- Aqui neste caso, se a variável **nresp** tiver o valor 1 500 000 000 , a expressão **nresp*sizeof(char*)** vai ser igual a 6 000 000 000, ou seja, superior ao número máximo que pode ser representado por um inteiro de 32 bit, que é 4 294 967 295
- Com o overflow, o resultado que vai sair daquela expressão é 1 705 032 705
- E posteriormente vai ser alocada muito menos memória do que a necessária (`xmalloc`)
- Depois no ciclo `for`, vão ser escritos dados fora da memória que foi realmente alocada
- As consequências podem avassaladoras, desde o simples crash da aplicação até à injeção de código malicioso

6 - Design inseguro



Design inseguro

- Esta categoria está relacionada com as falhas ao nível do design ou arquitetura do software
- Não devemos confundir falhas de design com falhas de implementação: são duas coisas diferentes
- As falhas de design podem ser evitadas se o projeto começar logo a ser pensado para ser seguro por design, suportado em:
 - Modelação de ameaças
 - Padrões seguros de design
 - Arquiteturas de referência
 - Frameworks de desenvolvimento



Design inseguro: algumas falhas típicas de design

- Ausência de um estudo prévio sobre os requisitos de segurança do software
- Ausência de um estudo prévio sobre as eventuais utilizações abusivas do software - exemplo compra bilhetes com reserva temporária
- Não utilização de uma framework de desenvolvimento seguro e padrões de desenvolvimento seguro
- Ausência de testes de integração e unidade no ciclo de desenvolvimento de software
- Geração de mensagens de erro que incluem informação sensível sobre o ambiente de runtime, utilizadores e dados associados
- Armazenamento de credenciais sensíveis (por exemplo senhas de acesso a bases de dados) em ficheiros de configuração, ou pior ainda, no próprio código
- Mistura de dados confiáveis com dados não confiáveis na mesma estrutura de dados - deve haver uma barreira clara entre dados confiáveis e dados não confiáveis

Design inseguro: mais algumas falhas típicas de design

- O software permite que dados enviados pelo utilizador controlem ou influenciem caminhos ou nomes de ficheiros que são usados em operações no sistema de ficheiros (Path Traversal).
- Exemplo:

```
String rName = request.getParameter("reportName");  
  
File rFile = new File("/usr/local/apfr/reports/" + rName);  
...  
rFile.delete();
```

- Neste exemplo, se um atacante enviar no parâmetro reportName a string ".././tomcat/conf/server.xml", o código vai apagar o ficheiro de configuração do sistema server.xml

Design inseguro: mais algumas falhas típicas de design


- O software usa um canal primário para administração ou funcionalidades restritas, mas esse canal não é protegido devidamente (por exemplo com MFA)
- O software permite o upload para o servidor de ficheiros de qualquer tipo, incluindo executáveis, php, js, asp, etc. O atacante pode depois invocar remotamente estes ficheiros
- A interface do utilizador não mostra devidamente informação crítica ao utilizador, permitindo eventualmente a um atacante alterar o estado do software sem que o utilizador se aperceba - facilitando phishing e CSRF
- O software usa inputs que assume como imutáveis (por exemplo campos de formulários escondidos, cookies, parâmetros, URLs), mas não verifica se estes foram alterados. Um atacante pode manipular estes inputs.
- O software guarda informação nos cookies que é usada para controlar a autorização, por exemplo tipo de utilizador. Um atacante pode manipular os dados nos cookies.

Design inseguro: mais algumas falhas típicas de design

- O software permite que sejam guardadas em cache informações sensíveis, que podem posteriormente ser acedidas por outros utilizadores
- O software usa cookies persistentes que contêm dados sensíveis
- A aplicação usa o método HTTP GET e inclui informação sensível nos parâmetros da query string.
- O software implementa mecanismos de proteção do servidor no código no cliente, em vez desses mecanismos serem implementados no servidor. Um atacante tem acesso ao cliente e pode fazer um bypass a esses mecanismos
- O software recorre a "segurança por obscuridade", ou seja, tem mecanismos de segurança cuja força depende essencialmente do facto de os atacantes não os conhecerem. Um atacante pode fazer engenharia reversa para descobrir como funcionam estes mecanismos

Design inseguro: mais algumas falhas típicas de design

- O software não limita o número ou frequência de interações com os utilizadores, por exemplo, o número de pedidos por minuto
- O software não usa uma framework de validação de todas as entradas, ou usa uma framework de forma errada



7 - Configuração errada de segurança



Configuração errada de segurança



- Esta categoria está relacionada com as falhas ao nível das configurações de segurança da própria aplicação ou da infraestrutura que a suporta
- A categoria envolve configurações feitas fora do código fonte, mas com implicações na segurança da aplicação
- Algumas falhas típicas de configuração:
 - Falhas ao nível da blindagem de segurança ao longo da aplicação
 - Permissões mal configuradas em serviços cloud
 - Funcionalidade desnecessárias instaladas ou ativadas (ports, serviços, páginas, contas, privilégios)
 - Contas prédefinidas e respetivas senhas não foram mudadas
 - Software em produção no modo de debugging, mostrando mensagens de erro com dados sensíveis
 - Settings das frameworks não estão no modo de segurança máxima
 - Falha na atualização das funcionalidades de segurança
 - O servidor não envia cabeçalhos HTTP com diretivas de segurança

8 - Utilização de componentes desatualizados e vulneráveis



Utilização de componentes desatualizados e vulneráveis



- Esta categoria está relacionada com a utilização de bibliotecas, módulos ou serviços desatualizados, vulneráveis ou sem manutenção de segurança
- Uma aplicação pode ficar vulnerável se:
 - Se desconhecem as versões de todos os componentes que são usados, tanto no cliente como no servidor
 - Se existem componentes vulneráveis, sem suporte ou desatualizados. Estes componentes incluem sistemas operativos, servidores aplicativos, bases de dados, APIs, bibliotecas, ambientes de runtime, etc
 - Não são feitos regularmente scans de vulnerabilidades
 - Não é feito o upgrade atempado de plataformas, frameworks e das suas dependências
 - Não são feitos testes de compatibilidade das novas versões dos componentes



Utilização de componentes desatualizados e vulneráveis - o exemplo do Log4j

- O Log4j é um pequeno módulo Java usado para gerir ficheiros de log
- Este componente era usado de forma massiva em milhares de aplicações
- Em dezembro de 2021 foi descoberta uma vulnerabilidade que permitia a um atacante correr código arbitrário num servidor com uma aplicação que usasse este componente
- Foi considerada a vulnerabilidade mais perigosa dos últimos 10 anos.
- Afetou centenas de serviços online, incluindo Cloudflare, iCloud, Minecraft: Java Edition, Steam, Tencent QQ, e Twitter.

9 - Falhas de identificação e autenticação



Falhas de identificação e autenticação



- A identificação e autenticação é uma das primeiras barreiras de segurança de um sistema ou aplicação
- Esta categoria envolve as falhas de identificação e autenticação dos atores num sistema informático
- Estes atores podem ser utilizadores, componentes ou serviços
- Falhas da identificação e/ou autenticação podem ser catastróficas, uma vez que um atacante poderá circunscrever o sistema de autenticação ou fazer-se passar por um outro utilizador



Algumas falhas de identificação e autenticação

- O software permite ataques automatizados de força bruta, com uma lista de possíveis nomes de utilizador e senha
- O software tem utilizadores e senhas pré-definidos, com senhas previsíveis e fracas
- O software tem sistemas de recuperação de senhas fraco ou mal desenhado, que permite a um atacante recuperar a senha de outro utilizador
- A senha é guardada em claro, sem sal, e com funções de hashing fracas
- Não é usada autenticação multi factor (MFA)
- O software expõe o identificador de sessão no URL
- O software reutiliza identificadores de sessão
- O software não invalida identificadores de sessão quando é feito o logout ou após um determinado tempo de inatividade



Mais algumas falhas de identificação e autenticação

- O software permite a autenticação de um utilizador malicioso através da técnica de captura-replay
- O software baseia a autenticação em certificados digitais, mas a sua autenticidade (cadeia de certificação) não é verificada
- Um servidor é autenticado através de certificado digital, mas o software não verifica se o certificado corresponde mesmo àquele servidor
- Falta de um passo crítico na sequência de autenticação
- Não existe um mecanismo para limitar a quantidade de autenticações falhadas - no caso de um ataque de força bruta
- O software permite que os utilizadores definam senhas fracas



Mais algumas falhas de identificação e autenticação

- O software permite a um utilizador mudar a senha, sem ter que introduzir a senha anterior - isto pode ser usado por um atacante para mudar a senha de outro utilizador que se tenha esquecido de fazer o logout

10 - Falhas de integridade de dados e software



Falhas de integridade de dados e software



- Esta categoria envolve falhas no controlo da integridade de dados e software, que permitem a sua adulteração
- Falhas no controlo de integridade em updates de software, dados críticos e pipelines CI/CD podem levar à introdução de software malicioso num sistema informático, com consequências dramáticas
- Deve ser dada especial atenção a sistemas que usam plugins, bibliotecas ou módulos de origem não confiável, repositórios públicos e redes de entrega de conteúdo (CDN)



Algumas falhas de integridade de dados e software

- O software não verifica a origem, autenticidade e integridade de dados, de forma a que aceita dados inválidos ou manipulados por um atacante
- O software usa um protocolo de transmissão de dados que não controla a integridade de dados
- O software procura por recursos críticos a partir de uma lista de repositórios controlada externamente
- O software descarrega componentes executáveis de uma localização remota sem verificar a origem e integridade desses componentes
- O software des-serializa dados não confiáveis sem verificar se os dados resultantes serão válidos



Algumas falhas de integridade de dados e software

- O software baseia-se em dados presentes em cookies para decisões críticas, sem verificar a integridade dos cookies
- O software baseia-se em dados presentes em cookies para decisões críticas, sem verificar se aqueles dados se aplicam ao utilizador "logado"
- O software inclui uma funcionalidade externa (por exemplo uma widget web) de outro domínio não confiável



Falhas de integridade de dados e software - o caso SolarWinds

- A SolarWinds é uma grande empresa de software que produz ferramentas de gestão de redes e infraestrutura, que são usadas por centenas de organizações em todo o mundo
- Em 2019, atacantes (eventualmente a soldo de um estado cujo nome começa por R e acaba em A) conseguiram penetrar na rede da SolarWinds e ter acesso ao repositório do código fonte da sua ferramenta Orion
- Em 2020, introduziram alterações no código fonte da ferramenta, que lhes permitia o acesso a todas as entidades que instalassem a ferramenta Orion
- Quando a SolarWinds compilou o código e fez a atualização automática nos clientes, estes ficaram infetados com o código malicioso
- Entre as empresas comprometidas estão Microsoft, Intel, Cisco, Nvidia, VMware, Belkin, e a empresa de segurança FireEye - que descobriu o problema

11 - Falhas no logging e monitorização



Falhas no logging e monitorização



- O registo de eventos e atividades relevantes no âmbito da utilização de um sistema informático é crítico para detetar e responder a ataques informáticos
- A monitorização em tempo real é essencial para responder em tempo útil a ameaças
- O logging é essencial para a responsabilização, visibilidade, alertas de incidentes e análise forense
- Estes sistemas de logging normalmente alimentam sistemas **SIEM - Security Information and Event Management** , que usam automação e inteligência artificial para detetar atividade suspeita
- Falhas no sistema de logging e monitorização podem comprometer a forma como as entidades podem responder a ataques informáticos



Falhas no logging e monitorização

- O software não faz o logging de eventos cruciais, como logins, logins falhados, transações críticas
- Avisos e erros não geram mensagens de log claras e adequadas
- Os logs de aplicações e APIs não são monitorizados - quer manualmente quer automaticamente
- Os logs são guardados apenas localmente
- Não há processos de alertas e escalonamento de respostas a incidentes
- A realização de testes de penetração (PEN testing) e scans de segurança dinâmicos (DAST) não fazem disparar alertas