Catálogo de vulnerabilidades

FURB
UNIVERSIDADE DE BLUMENAU

Catálogo de vulnerabilidades

- Os seguintes catálogos cobrem vulnerabilidades comuns na codificação de aplicações web:
  - OWASP – Open Web Application Security Project
  - PCI DSS – Payment Card Industry Data Security Standard
- CWE – Common Weakness Enumeration
- Os catálogos fornecem guias para orientar desenvolvedores para identificar e prevenir fraquezas.

Brief Listing of the Top 25

This is a brief listing of the Top 25 items, using the general ranking.

NOTE: 16 other weaknesses were considered for inclusion in the Top 25, but their general scores were not high enough. They are listed in a separate "On the Cusp" page.

Rank	Score	ID	Name
[1]	93.8	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
[2]	83.3	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
[3]	79.0	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[4]	77.7	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
[5]	76.9	CWE-306	Missing Authentication for Critical Function
[6]	76.8	CWE-862	Missing Authorization
[7]	75.0	CWE-798	Use of Hard-coded Credentials
[8]	75.0	CWE-311	Missing Encryption of Sensitive Data
[9]	74.0	CWE-434	Unrestricted Upload of File with Dangerous Type
[10]	73.8	CWE-807	Reliance on Untrusted Inputs in a Security Decision
[11]	73.1	CWE-250	Execution with Unnecessary Privileges
[12]	70.1	CWE-352	Cross-Site Request Forgery (CSRF)
[13]	69.3	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
[14]	68.5	CWE-494	Download of Code Without Integrity Check
[15]	67.8	CWE-863	Incorrect Authorization
[16]	66.0	CWE-829	Inclusion of Functionality from Untrusted Control Sphere
[17]	65.5	CWE-732	Incorrect Permission Assignment for Critical Resource

- Várias vulnerabilidades podem ser causadas por falta de “boas práticas de programação”.

- “(...) seguir boas práticas de programação tornam os sistemas mais robustos, confiáveis e, evidentemente, seguros.” (ALBUQUERQUE, 2002).

Validação de entrada inadequada

Validação de entrada inadequada

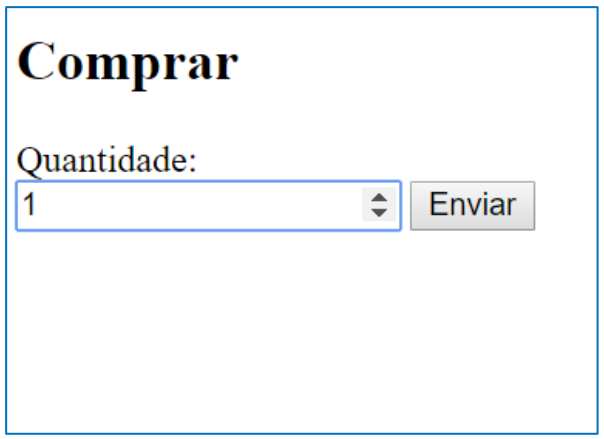
- Este é o tipo de vulnerabilidade em que o software não faz validações adequadas das entradas de dados, podendo afetar o fluxo de execução do programa

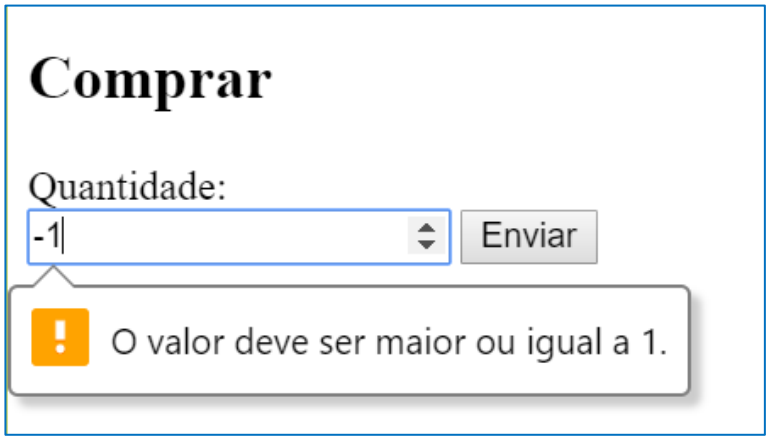
- Ocorre pois o programador não acredita que certas entradas podem ser modificadas pelo agente e com isso, o programador pode não realizar qualquer validação sobre os dados de entrada.

- A entrada pode ser feita através de interação com a aplicação (como teclado, mouse), importação de dados ou API

Exemplo 1

- Este exemplo demonstra a interação de um shopping virtual na qual o usuário consegue especificar a quantidade de itens que serão comprados e o total é calculado.






```
<input type="number" name="quantidade" min="1" size="50%">
```

Exemplo 1

O agente pode editar o *html* gerado para desviar de determinadas consistências

```
<input type="number" name="quantidade" min="1"
size="50%"> == $0
```

Considerar que o serviço executa o código abaixo:

```
...
int quantity = dataSession.getAttribute("quantidade");
double total = price * quantity;
chargeUser(total);
...
```

Este código não consiste a quantidade recebida.
Ao receber valor negativo, o usuário pode ser creditado, ao invés de debitado.

Validação de entrada inadequada

- Em aplicações web, muitos programadores acreditam que os cookies e campos ocultos do formulário não podem ser modificados no navegador.

- Os campos ocultos são utilizados para enviar um valor do servidor para o cliente e (sem que o valor seja alterado), enviar novamente para o servidor.

Exemplo 2

- Neste exemplo, o programa solicita ao usuário as dimensões de um tabuleiro de jogo.

```
...
#define MAX_DIM 100
...
/* board dimensions */

int m,n, error;
board_square_t *board;
printf("Please specify the board height: \n");
error = scanf("%d", &m);
if ( EOF == error ) {
    die("No integer passed\n");
}

printf("Please specify the board width: \n");
error = scanf("%d", &n);
if ( EOF == error ) {
    die("No integer passed\n");
}

if ( m > MAX_DIM || n > MAX_DIM ) {
    die("Value too large!\n");
}

board = (board_square_t*) malloc( m * n * sizeof(board_square_t));
...
```

O programa não recusa números negativos, sendo possível informar dois números negativos.

O agente pode provocar o uso excessivo de memória.

Validação de entrada
consequências

- Um agente pode submeter valores inesperados que podem abortar a execução do software, causar loopings infinitos ou consumo excessivo de recursos
- Para evitar tais vulnerabilidades, deve-se aplicar a seguinte prática para desenvolvimento de sistemas seguros:

> Tratar todas as entradas do sistema como não confiáveis

- "(...) sempre que receber dados, seja através de entrada direta de dados pelo usuário, seja pelo recebimento de dados de outro sistema, estes devem passar por uma verificação de integridade e consistência." (ALBULQUERQUE, 2002)

Validação de entrada

- Deve-se assumir que todas as entradas podem fornecer dados maliciosos.

- A recomendação é criar uma lista de entradas aceitáveis (“lista branca”). Desta forma, qualquer entrada que não seguir esta definição, deverá ser recusada (ou transformada).

- A recomendação é de não confiar exclusivamente em observar entradas maliciosas ou malformadas (isto é, não confiar numa “lista negra”).
  - É mais provável que seja esquecido alguma entrada não desejada
  - Listas negras podem ser úteis para detectar ataques potenciais

Validação de entrada

- Quando for desenvolver validação de dados de entrada, deve-se considerar todas as propriedades relevantes, incluindo:
  - quantidade de caracteres,
  - tipo de dado de entrada,
  - o intervalo completo de valores aceitáveis,
  - consistência entre campos relacionados e
  - conformidade às regras de negócios.

Funções seguras

- Outra prática para construção de software seguro é:

> Criar e usar funções intrinsecamente seguras

- Todas as funções deveriam validar os dados de entrada para impedir perda de controle do sistema ou qualquer tipo de falha.
  - Isso significa que cada função (método) deve verificar se os parâmetros estão dentro de um intervalo de valores aceitáveis e não contém caracteres estranhos ou má formação, antes de qualquer tentativa de uso.

- Também recomenda-se sempre retornar exceção quando algum problema for detectado na entrada ou durante a execução da função

Implicação de práticas seguras

- “(...) muitas vezes um código seguro implica uma certa perda de desempenho. Isto é inevitável devido ao maior controle dentro do sistema, mas pode ser facilmente compensado com um hardware mais rápido. Já para um código inseguro, não há hardware que dê jeito” (ALBUQUERQUE, 2002).