

Mestrado em Engenharia Informática (MEI)

Mestrado Integrado em Engenharia Informática

(MiEI)

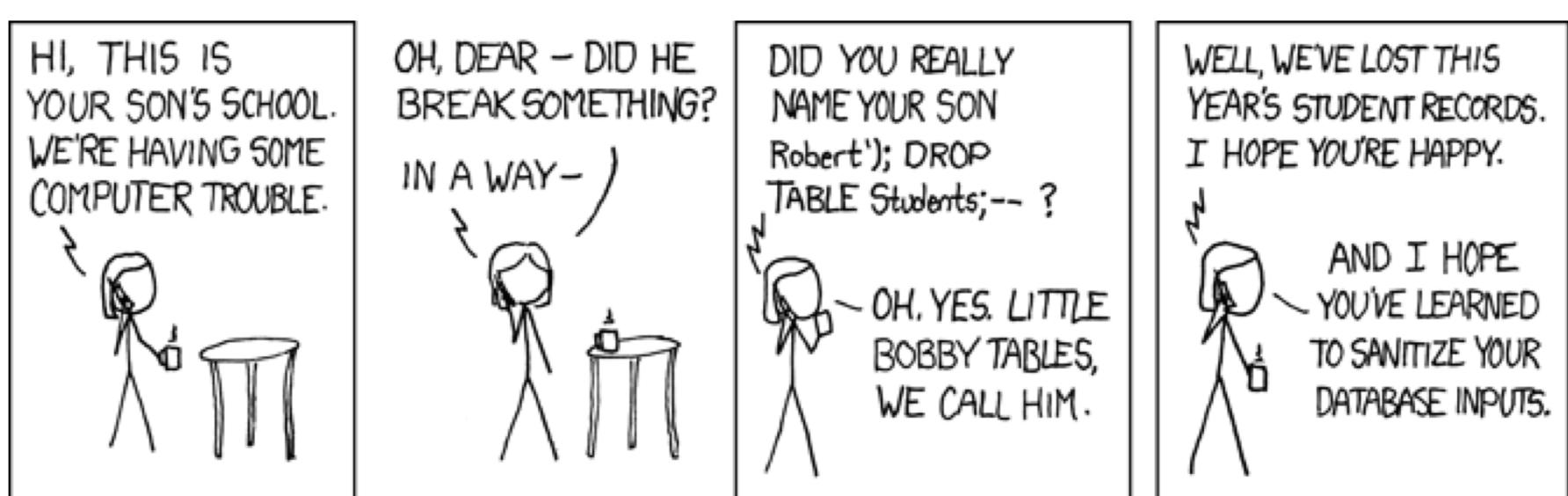
Perfil de Especialização **CSI** : Criptografia e Segurança da Informação

Engenharia de Segurança



Tópicos de Segurança de Software

- Validação de *Input*



Validação de Input

- Todo o input para um programa (a partir de utilizador através de um teclado, rede, ficheiros, dispositivos externos, variáveis de ambiente, ...) pode ser a fonte de vulnerabilidades de segurança e bugs;
- Qualquer programa que processa dados de input sem validação adequada, é susceptível a vulnerabilidades de segurança;
- Um atacante pode passar argumentos mal formados a qualquer parâmetro do programa;
- Todo o input deve ser tratado como potencialmente perigoso.
- Estas questões são particularmente relevantes em programas com permissões *setuid root* (i.e., os utilizadores executam o programa como se fossem *root*), ou que executem em modo privilegiado.



Validação de Input

CVE ID	Vulnerability type	Publish Date	CVSS Score	Description
CVE-2018-1000125		2018-03-13	7.5	inversoft prime-jwt version prior to version 1.3.0 or prior to commit 0d94dcef0133d699f21d217e922564adbb83a227 contains an input validation vulnerability in JWTDecoder.decode that can result in a JWT that is decoded and thus implicitly validated even if it lacks a valid signature. This attack appears to be exploitable via an attacker crafting a token with a valid header and body and then requests it to be validated.
CVE-2018-9142		2018-03-30	7.6	On Samsung mobile devices with N(7.x) software, attackers can install an arbitrary APK in the Secure Folder SD Card area because of faulty validation of a package signature and package name.
CVE-2018-8998	DoS	2018-03-24	6.1	In Advanced SystemCare Ultimate 11.0.1.58, the driver file (Monitor_x86.sys) allows local users to cause a denial of service (BSOD) or possibly have unspecified other impact because of not validating input values from IOCTL 0x9c4060cc.
CVE-2018-8711	File Inclusion	2018-03-14	7.5	A local file inclusion issue was discovered in the WooCommerce Products Filter (aka WOOF) plugin before 2.2.0 for WordPress. The vulnerability is due to the lack of args/input validation on render_html before allowing it to be called by extract(), a PHP built-in function. Because of this, the supplied args/input can be used to overwrite the \$pagepath variable, which then could lead to a local file inclusion attack.
CVE-2018-7187	Exec Code	2018-02-16	9.3	The "go get" implementation in Go 1.9.4, when the -insecure command-line option is used, does not validate the import path (get/vcs.go only checks for ":// anywhere in the string), which allows remote attackers to execute arbitrary OS commands via a crafted web site.
CVE-2018-6476		2018-01-31	10.0	In SUPERAntiSpyware Professional Trial 6.0.1254, the SASKUTIL.SYS driver allows privilege escalation to NT AUTHORITY\SYSTEM because of not validating input values from IOCTL 0x9C402114 or 0x9C402124 or 0x9C40207c.

Validação de Input

- **Superfície de ataque** de uma aplicação é constituída pelo conjunto de interfaces através das quais podem ser recebidas entradas vindas do exterior:
 - Mecanismos de comunicação remota (e.g., *sockets, web services, ...*);
 - Mecanismos de comunicação entre processos (e.g., sinais, semáforos, ...);
 - Interface programática da aplicação (API);
 - Ficheiros utilizados pela aplicação;
 - Interface utilizado (e.g., argumentos da aplicação, input do utilizador, ...);
 - Sistema operativo (e.g., variáveis de ambiente, ...).
- Regra de ouro na construção de software seguro é **nunca confiar no input.**

Validação de Input proveniente do processo-pai

- Nos sistemas operativos da família Unix, as aplicações são tipicamente lançadas a partir de uma *shell* e, executadas como processo-filho dessa *shell*.
- Atacante com acesso a essa *shell*, pode lançar a aplicação, fornecendo input que explore alguma vulnerabilidade:
 - Argumentos com tamanho indevido (podendo provocar *buffer overflow*);
 - Rotinas de tratamento de sinais com código malicioso;
 - Definir as permissões por omissão dos ficheiros criados pela aplicação (através do comando *umask*);
 - Variáveis de ambiente com valores erróneos.
 - PATH guarda as diretórias onde podem estar os programas executáveis (quando é pedida a execução de um programa sem se especificar o seu caminho absoluto, o sistema operativo percorre as diretórias guardadas na PATH pela ordem em que aparecem nessa variável, até encontrar o programa e executá-lo).



Validação de Input proveniente do processo-pai

Exemplo:

- As funções C *system(command)* e *popen(command, type)* executam o programa/comando passado como argumento, com as variáveis de ambiente do processo-pai → **evite ambas as funções**
- Suponha que um programa inclui a instrução *system("ls")* para listar o conteúdo da diretoria atual (note que o programa *ls* legítimo está guardado na diretoria /bin)
- O que aconteceria se um atacante:
 - Alterasse o valor da PATH para “.:./usr/local/bin:/usr/bin:/bin”, e
 - Efectuasse o comando bash “cp ./programa_malicioso ./ls” ?
- E o que aconteceria se o programa tivesse permissões *setuid root*?



Validação de Input proveniente do processo-pai

- Que outras variáveis de ambiente são utilizadas pelo seu programa ou pelas bibliotecas/APIs que utiliza?
- Se as bibliotecas/API que utiliza não efetuam o controlo adequado das variáveis de ambiente:
 - Cabe-lhe a si efetuar esse controlo, ou
 - Defina você mesmo a variável.



Validação de Input: Metacaracteres

- Metacaracteres são fonte de especial preocupação, uma vez que são responsáveis por um grande número de vulnerabilidades, tipicamente em aplicações que lidam com *strings*.
- Solução para este tipo de vulnerabilidade é simples: validar os inputs recebidos, controlando os (meta)caracteres aceites
 - Optar por técnicas de ***white listing*** onde é definida uma lista com os caracteres válidos aceites pela aplicação.
 - Não optar por ***black listing***, i.e., pela lista de caracteres que não devem ser aceites pela aplicação. Porquê?
 - Por exemplo, se estiver a ser utilizado o *encoding* UTF-8, o caracter ‘.’ (ponto) também pode ser escrito como ‘%2e’ e ‘%c0%ae’, entre outros.

Validação de Input: Metacaracteres

- Três tipos de ataques mais comuns baseados em Metacaracteres:
 - Delimitadores embebidos
 - Quando input para aplicação inclui diferentes tipos de informação separada por delimitadores.
 - Suponha uma aplicação que armazena nomes de utilizadores com respetivas senhas num ficheiro em que cada linha tem o formato *utilizador:password\n*
 - O que pode acontecer se a Alice ao alterar a password, escolher *batata\nhacker:ola123* ?
 - Injeção do caracter \0
 - Injeção de separadores.



Validação de Input: Metacaracteres

- Três tipos de ataques mais comuns baseados em Metacaracteres:
 - Delimitadores embebidos
 - Perigoso porque nem sempre é interpretado como terminador de string (por exemplo, no Perl e Java, ao contrário do C/C++)
 - Considere uma aplicação Web construída em Perl e que permite abrir ficheiros de texto terminados com a extensão .txt.
 - O que acontece se a Alice fornecer como nome de ficheiro a string `/etc/passwd\0.txt` ?
 - Injeção de separadores.

Validação de Input: Metacaracteres

- Três tipos de ataques mais comuns baseados em Metacaracteres:
 - Delimitadores embedidos
 - Injeção do carácter \0
 - Injeção de separadores
 - Injeção de separadores de comandos, que podem permitir a execução de comandos arbitrários. Nos Unix, recorrendo ao metacaracter ‘;’.
 - Injeção de separadores de pastas, geralmente denominado por *path traversal attack* podem permitir a leitura e/ou escrita de ficheiros arbitrários.
 - Considere uma aplicação Web que dado um utilizador imprime estatísticas recorrendo a `system("cat", "/var/stats/$username");`
 - Como é que a Alice pode aproveitar esta vulnerabilidade e imprimir um ficheiro qualquer do sistema, por exemplo o ficheiro /etc/passwd ?

Validação de Input: Vulnerabilidade de String de formato

- Classe de vulnerabilidades na qual:
 - A falta de validação de entradas permite a um atacante controlar a execução de uma aplicação;
 - A validação de entradas necessária para evitar a vulnerabilidade é extremamente simples.
- Classe de vulnerabilidades mais prevalente e perigosa no C e C++, embora outras linguagens (e.g., Java, Perl, PHP, Python e Ruby) também permitam strings de formato com vulnerabilidades relacionadas.

Validação de Input: Vulnerabilidade de String de formato

- Exemplo simples (e clássico) da vulnerabilidade de string de formato:

```
1 #include <stdio.h>
2
3 int main(int argc, char **argv) {
4     char buf[1024];
5
6     if(argc > 1) {
7         strcpy(buf, argv[1], 1023);
8         buf[1023] = '\0';
9         printf(buf);
10    }
11 }
```

- O primeiro argumento da função *printf* está sob controlo do utilizador da aplicação, e pode ser usada para especificar o formato de diferentes tipos de dados (e.g., %d indica uma variável inteira, %s uma string, ...).

```
$ ./a.out "string - %s | apontador - %p | inteiro - %d"
string - (null) | apontador - 0xfffffffffffffff | inteiro - 19
```

Validação de Input: Vulnerabilidade de String de formato

- Exemplo simples (e clássico) da vulnerabilidade de string de formato:

```
1 #include <stdio.h>
2
3 int main(int argc, char **argv) {
4     char buf[1024];
5
6     if(argc > 1) {
7         strncpy(buf, argv[1], 1023);
8         buf[1023] = '\0';
9         printf(buf);
10    }
11 }
```

- Sempre que a string de formato possa ser controlado por um atacante, estamos perante uma vulnerabilidade de string de formato.
 - Ocorre em todas as famílias de funções que têm como argumento strings de formato (e.g., printf, err, syslog).

Validação de Input: Vulnerabilidade de String de formato

- Exemplo simples (e clássico) da vulnerabilidade de string de formato:

```
1 #include <stdio.h>
2
3 int main(int argc, char **argv) {
4     char buf[1024];
5
6     if(argc > 1) {
7         strncpy(buf, argv[1], 1023);
8         buf[1023] = '\0';
9         printf(buf);
10    }
11 }
```

- Como se resolve esta vulnerabilidade?
 - Substituindo na linha 9 *printf(buf)* por *printf("%s", buf)* .

```
$ ./a.out "string - %s | apontador - %p | inteiro - %d"
string - %s | apontador - %p | inteiro - %d
```



Validação de Input: Vulnerabilidade de String de formato

- Exemplo simples (e clássico) da vulnerabilidade de string de formato:

```
1 #include <stdio.h>
2
3 int main(int argc, char **argv) {
4     char buf[1024];
5
6     if(argc > 1) {
7         strncpy(buf, argv[1], 1023);
8         buf[1023] = '\0';
9         printf(buf);
10    }
11 }
```

- Qual o maior problema desta vulnerabilidade?

- Permite ler / escrever valores da stack, através do uso apropriado de formatadores de string.
 - Ex: (o valor 41 hexadecimal corresponde ao valor 65 em decimal, que corresponde à letra ‘A’).

```
$ ./a.out AAAAAAAA%p%p%p%p%p%p%p%p%p%p%p%p%p%p  
AAAAAAA0x1d0x7ffeeda8db9400x1d0x00xfffffff00000000x00x7ffeeda8db3700x7ffeeda8db7a80x20x4141414141414141
```



Validação de Input

- Risco

- Se os dados de input não são validados para garantir que contêm o tipo, a quantidade e a estrutura correta de informação, problemas podem (e vão) acontecer;
- Erros de validação de input podem levar a *buffer overflows* se os dados forem utilizados como índices para um array, ou utilizados como base de SQL injection permitindo aceder/alterar/apagar dados privados numa Base de Dados;
- Os atacantes podem utilizar inputs cuidadosamente escolhidos, de forma a causar a execução de código arbitrário. Esta técnica pode ser usada para apagar dados, causar danos, propagar worms, ou obter informações confidenciais.

Validação de Input

- Validação “responsável” de **todo** o input
 - Tipo: validar que o input tem o tipo de dados expectável, por exemplo a idade é *int*. Muitos programas lidam com os dados de input assumindo que é uma string, verificando depois que essa string contém os caracteres apropriados, e convertendo-a para o tipo de dados desejado;
 - Tamanho: validar que o input tem o tamanho expectável (por exemplo, o número de telefone tem 9 dígitos);
 - Intervalo: validar que o input se encontra dentro do intervalo expectável (por exemplo, o valor do mês encontra-se entre 1 e 12);
 - Razoabilidade: validar que o input tem um valor razoável (por exemplo, o nome não contém caracteres de pontuação nem não alfanuméricos);
 - Divisão por Zero: validar o input de modo a não aceitar valores que possam causar problemas posteriormente no programa, como por exemplo a divisão por zero;
 - Formato: validar que os dados de input estão no formato adequado (por exemplo, a data está no formato DD/MM/YYYY);
 - Dados obrigatórios: garantir que o utilizador insere os dados obrigatórios;
 - Checksums: muitos números de identificação possuem *check digits* (dígitos adicionais inseridos o final do número para validação). Ver *check digit* de [Cartão de Cidadão](#), [Passaporte](#), [cartões de crédito](#), [algoritmo de Luhn](#).



Validação de Input

- Utilização das ferramentas da linguagem de programação
 - Linguagens como o C e C++ lêem (por omissão) o input para um buffer de caracteres, sem validarem o limite do buffer, causando problemas de buffer overflow e de validação de input. Contudo existem disponíveis bibliotecas específicas de leitura, mais robustas, desenhadas a pensar na segurança;
 - Linguagens fortemente tipadas, como o Java e C++, exigem que o tipo dos dados armazenado numa variável seja conhecido a-priori (o que leva a incompatibilidades de tipo quando, por exemplo, uma string é inserida em resposta a um pedido de inteiro);
 - Linguagens não tipadas, como o PHP ou Python não têm esses requisitos – qualquer variável pode armazenar qualquer valor. Este facto não eliminam os problemas de validação (teste o input de uma string para ser utilizado como índice de um array);
- Recuperação apropriada
 - Um programa robusto deve tratar um input inválido de um modo apropriado, correcto e seguro, repetindo o pedido de input ou continuando com valores pré-definidos (truncar ou reformatar dados de modo a ajustá-los ao pretendido deve ser evitado).