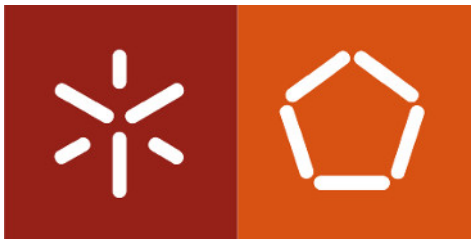

Mestrado em Engenharia Informática
Universidade do Minho

Engenharia de Segurança

Aula 11 TP - 11/05/2020

João Miranda - PG41845
Sandro Cruz - PG41906

11 de Maio de 2020

Conteúdo

Capítulo 1

Validação de Input

1.1 Experiência 1.1

Analise os ficheiros InputValidation.cpp e InputValidation.java.

```
user@CSI:~/Desktop/engseg$ java InputValidation
Insira um número: adsadsadasd
Exception in thread "main" java.util.InputMismatchException
        at java.base/java.util.Scanner.throwFor(Scanner.java:860)
        at java.base/java.util.Scanner.next(Scanner.java:1497)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2161)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2115)
        at InputValidation.main(InputValidation.java:14)
user@CSI:~/Desktop/engseg$
```

Figura 1.1: Output recebido dando como input uma string, no programa "InputValidation.java"

```
user@CSI:~/Desktop/engseg$ java InputValidation
Insira um número: 1111111111111111111111111111111111111111111111111111
Exception in thread "main" java.util.InputMismatchException: For input string: "
1111111111111111111111111111111111111111111111111111"
        at java.base/java.util.Scanner.nextInt(Scanner.java:2167)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2115)
        at InputValidation.main(InputValidation.java:14)
user@CSI:~/Desktop/engseg$
```

Figura 1.2: Output recebido dando como input um inteiro superior ao esperado, no programa "InputValidation.java"

```
user@CSI:~/Desktop/engseg$ ./InputValidation
Insira um número: sdadasdsad
O quadrado de 0 é 0
```

Figura 1.3: Output recebido dando como input uma string, no programa "InputValidation.cpp"

```
user@CSI:~/Desktop/engseg$ ./InputValidation
Insira um número: 11111111111111111111111111111111111111111111111111111111111111
O quadrado de 18446744073709551615 é 4294967295
user@CSI:~/Desktop/engseg$ ./InputValidation
```

Figura 1.4: Output recebido dando como input um inteiro superior ao esperado, no programa "InputValidation.cpp"

Os problemas existentes na utilização do "scanner"/"cin"é que não existe uma validação do input do utilizador, o que pode levar a um conjunto de vulnerabilidades associados com a má validação dos dados. O acesso directo do input para o "array"pode levar a um "buffer"overflow, o que por si só pode causar um panóplia de vulnerabilidades.
Uma forma de mitigar este problema era validar os dados do input, tanto o inteiro máximo e mínimo que se pode inserir, verificar se input recebido é realmente só constituído por inteiros.

1.2 Experiência 1.2

Os problemas existentes são semelhantes aos da experiência 1.1, pois os dados não são validados depois do utilizador os ter inserido, como por exemplo é possível inserir dados que não são inteiros como por exemplo uma string ou um número que é superior ao valor máximo da variável "int", o que leva a que excepções sejam lançadas.

```java
import java.util.*;

public class WhileEx
{
  public static void main(String[] args)
  {
    Scanner console = new Scanner(System.in);
    int age;
    int total = 0;
    int i = 0;

    System.out.println("Insira 10 idades: ");
    while(i<10){
      try{
        System.out.println("Insira a "+(i+1)+" idade ");
        age = console.nextInt();
        total = total + age;
        i++;
      }
      catch(InputMismatchException e){
        System.out.println("Dados inseridos invalidos tenhem de ser do tipo int");
        total = -1;
        break;
      }
      catch(Exception e){
        System.out.println("Dados inseridos invalidos: " + e.toString());
        total = -1;
        break;
      }
    }
    if(total != -1){
      System.out.println("Média de idades: " + (float)total/10);
    }
  }
}
```

Figura 1.5: Alterações feitas no "WhileEx.java"de forma a ser mais seguro.

1.3 Experiência 1.3

No método "getWich"não verificado se o input recebido é superior ao tamanho do "array", também não é feita uma verificação se o input recebido é um inteiro dado que estes dado vai ser utilizado num método seguinte e pode causar problemas.

```
int getWhich()
{
  size_t x;
  while (true) {
    cout << "Escolha o nome #: ";
    std::cin >> x;
    if(std::cin.fail()){
      std::cout << "Error" << std::endl;
      std::cin.clear();
      std::cin.ignore(256,'\n');
    }
    else if(x < 0 || x > 5){
      std::cin.clear();
      std::cout << "Numero Invalido" << '\n';
    }
    else{
      break;
    }
  }
  return x;
}
```

Figura 1.6: Alterações feitas no "Input.cpp"de forma a ser mais seguro.

1.4 Experiência 1.4

Tal como na experiência 1.3 método "getWich"não verificado se o input recebido é superior ao tamanho do "array", também não é feita uma verificação se o input recebido é um inteiro dado que estes dado vai ser utilizado num método seguinte e pode causar problemas.
No método "getArraySize"não é verificado se o input recebido foi realmente um inteiro, o que pode levar a excepções serem lançadas e nunca apanhadas.

```java
public static int getArraySize(Scanner scan, int limite) {
  int n;
  while(true){
    System.out.print("Quantos nomes? ");
    String s = scan.nextLine();
    if(!s.matches("-?\\d+")){
      System.out.print("Input invalido \n");
    }
    else if(s.length() > limite){
      System.out.print("Input demasiado longo\n");
    }
    else if(Integer.parseInt(s) < -1){
      System.out.print("Input invalido: "+s+"\n");
    }
    else{
      n = Integer.parseInt(s);
      break;
    }
  }
  return n;
}
```

Figura 1.7: Alterações feitas na função getWich() de forma a ser mais seguro.

```java
public static int getWhich(Scanner scan, int limite, int sz) {
  int x;
  while(true){
    System.out.print("Escolha o nome #: ");
    String s = scan.nextLine();
    if(!s.matches("-?\\d+")){
      System.out.print("Input invalido \n");
    }
    else if(s.length() > limite){
      System.out.print("Input demasiado longo \n");
    }
    else if(Integer.parseInt(s) < -1 || Integer.parseInt(s) > sz){
      System.out.print("Input invalido: "+s+"\n");
    }
    else{
      x = Integer.parseInt(s);
      break;
    }
  }
  return x;
}
```

Figura 1.8: Alterações feitas na função getArraySize() de forma a ser mais seguro.

1.5 Pergunta 1.1

Analise o programa filetype.c que imprime no ecran o tipo de ficheiro passado como argumento.

1.5.1 Existem pelo menos dois tipos de vulnerabilidades estudadas na aula teórica de "Validação de Input"que podem ser exploradas. Identifique-as.

Uma das vulnerabilidades existentes é a não validação das variáveis ambientes, o que pode levar a execução de um outro programa quando achamos que estamos a executar o "filetype", uma outra vulnerabilidade é a injecção de separadores no input pois apenas com um ";"é possível encadear vários comandos que iram ser executados, estas duas vulnerabilidades são possíveis dada a utilização do "system()"para ler o "buffer".

1.5.2 Forneça o código/passos/linha de comando que permitem explorar cada uma das vulnerabilidades identificadas na linha anterior.

```
user@CSI:~/Desktop/engseg$ ./filetype file;ls
file: ASCII text, with very long lines
file         Input.cpp              InputValidation.java  WhileEx.class
filetype     Input.java             readfile.c            WhileEx.java
filetype.c   InputValidation        string_formato2.c
Input        InputValidation.class  string_formato.c
Input.class  InputValidation.cpp    WhileEx2.java
user@CSI:~/Desktop/engseg$
```

Figura 1.9: Explorar a injecção de separadores utilizando o ";"e um comando unix

Para demonstrar a validação das variáveis ambientes criarmos um pequeno "bash script"só para demonstrar a vulnerabilidade.

```
user@CSI:~/Desktop/engseg$ cat file
#!/bin/bash

echo "Well this is awkward"
```

Figura 1.10: "Bash script"criado

Para que o "script"possa ser executado tivemos que dar permissões de execução ao ficheiro criado. Depois tivemos que alterar a variável "PATH"para por exemplo: "/usr/local/bin".

```
user@CSI:~/Desktop/engseg$ export PATH=/usr/local/bin:$PATH
user@CSI:~/Desktop/engseg$ sudo cp file /usr/local/bin/file
user@CSI:~/Desktop/engseg$
```

Figura 1.11: Comandos para alterar a variável "PATH"e copiar o "bash script"criado par o novo caminho armazenado na variável

Depois removemos o "bash script"do directório em que estamos a executar o programa, só para demonstrar não é necessário fazer dado que alteramos a variável ambiente, e depois executamos o programa chamando o "bash script"criado.

```
user@CSI:~/Desktop/engseg$ rm file
user@CSI:~/Desktop/engseg$ ./filetype file
Well this is awkward
user@CSI:~/Desktop/engseg$
```

Figura 1.12: Exemplificação da vulnerabilidade

1.6 Experiência 1.6

1.6.1 Faça algumas experiências com vários valores de input tanto com o programa com vulnerabilidades como sem vulnerabilidades e tire as suas conclusões.

Análogo aos exemplos que foram utilizados na aula teórica. Foi possível verificar com o programa string_formato.c que se podem obter informações para leitura e escrita de valores na stack. Relativamente ao string_formato2.c já houve a correção desse problema apresentado corretamente a representação lógica do que era pretendido do programa que era a escrita do *input* que foi dado em *output* em string.

1.7 Experiência 1.7

1.7.1 Teste a sua habilidade de identificar problemas de segurança durante a revisão de código.

Esta experiência foi realizada com a utilização do *website* indicado.

1.8 Experiência 1.8

1.8.1 Uma estratégia importante de segurança é a "defense in depth". Explique o que significa. De que modo é que a "defense in depth"está relacionada com a validação de input?

A estratégia "defense in depth"é uma implementação de segurança que possui camadas de segurança implementadas para proteger um ativo contra o acesso ou modificação não autorizada. Cada camada de segurança compõe uma "defense in depth". Caso alguma das camadas não consiga efetuar a proteção, a próxima camada estará no seu lugar para fazer essa proteção. A implementação de uma única camada de sanitização e de validação de input é um ponto de falha caso seja ignorada ou a sua implementação principal origine uma falha. É muito melhor abordar a implementação das camadas de validação de input que pode proteger de vulnerabilidades tais como a injeção de SQL, de código ou de comandos e do XSS.

1.9 Pergunta 1.2

O programa foi desenvolvido em Python denominado de validar.py e possui os seguintes métodos:

- **validateValue()**: Este método valida a introdução de um valor por parte do utilizador aceitando somente números inteiros ou, caso seja decimal, com 2 casas decimais. Foi considerado que o valor é um preço e por isso caso seja metido outro input que não este não é validado.
- **validateDate()**: A validação da data foi feita com base numa biblioteca denominada datetime que faz a verificação do formato de data que o utilizador apresentou tendo de ser coerente.
- **validateName()**: No caso do nome foram consideradas todas as palavras juntamente com a verificação da primeira letra ser maiúscula de cada nome.
- **validateNIF()**: Este método foi realizado de acordo com um algoritmo genérico já existente (algoritmo de módulo 11).
- **validateCC()**: Consiste na validação do cartão de cidadão. Cada letra existente no cartão de cidadão (no seu número) possui uma letra que corresponde a um número. Por isso, foi feita a verificação do número juntamente com a segunda parte que possui a letra.
- **validateCreditCard()**: A validação do cartão de crédito foi feita com base num algoritmo já existente (Algoritmo de Luhn).

- **validateDateCreditCard()**: Este método valida o estado do cartão, ou seja, se já atingiu a expiração ou não. É verificado com base na data apresentada por parte do utilizador e a data atual.
- **validateCVCCVV()**: O método que valida o CVC/CVV vai validar unicamente pela utilização de quatro algarismos na sua introdução por parte do utilizador.