Universidade do Minho

Mestrado em Engenharia Informática

Engenharia de segurança

Ficha de exercício 12

Grupo 1

Rui Carlos Azevedo Carvalho - PG47633
Daniel Barbosa Miranda - PG47123
Ana Luísa Lira Tomé Carneiro - PG46983

Validação de Input

1.1 Pergunta P1.1

1.1.1 Exercicio 1

Neste problema era pretendido que explora-se-mos a vulnerabilidade de validação de *input* e *buffer overflow* de forma a que o programa imprimi-se ficheiros que não terminem em '.txt'.

O programa readfile.c começa por ler o ficheiro passado como argumento, ao qual acrescenta o sufixo '.txt' de modo a garantir que só deixa ler ficheiros em texto, através da função *snprintf*, tal como está em baixo.

```
char buf[64]
snprintf(buf, sizeof(buf), "%s.txt", argv[1]);
```

Contudo, a variável *buf* que vai servir para armazenar o *input* com o sufixo '.txt' pode ser manipulada para não acrescentar esse sufixo ao *input* enviado. Como a variável *buf* tem de tamanho 64 bytes e como estamos a ler esse tamanho todo do *input* do utilizador, é possível que o *input* enviado possa ter um tamanho maior ou igual a este, fazendo com que o sufixo '.txt' não "caiba" no *buffer* pois, este foi totalmente preenchido com o *input* do utilizador. Desta forma, podemos ver que o programa readfile.c apresenta uma vulnerabilidade de *buffer overflow*, pois é possível enviar mais bytes do que aqueles que são permitidos fazendo com que o programa não execute de acordo com o seu objetivo.

Além da falta do *buffer overflow*, este programa também apresenta uma vulnerabilidade de validação de *input*, pois não verifica se o ficheiro que o utilizador pretende abrir é um ficheiro '.txt'. O programa em vez de fazer essa avaliação, assume que todos os *inputs* enviados vão ter como o sufixo '.txt', pois este é acrescentado a cada *input*. Contudo, esta limitação pode ser superada através da exploração da vulnerabilidade de *buffer overflow*.

1.1.2 Exercicio 2

De forma a explorar a vulnerabilidade mencionada no exercício anterior para que o programa leia o ficheiro */etc/passwd* podemos, tal como foi descrito acima, inserir como *input* no programa a *string* *./../../../../../../../../../../../../../../../../../../etc/passwd*. Desta forma, o programa como vai receber um *input* com cerca de 64 bytes, não vai conseguir acrescentar à variável *buf* o sufixo '.txt' e por a *string* que irá ficar armazenada no *buffer* será a mesma que foi enviada pelo o utilizador. Além disso, o utilizador pode se aproveitar do facto de que numa árvore de diretorias, podemos

utilizar o “..” para irmos para a diretoria anterior e o “.” para a diretoria actual. Assim, abrir um ficheiro com que se encontre na diretoria *etc/*, pode ser facilmente obtido se andarmos múltiplas diretorias para trás, tal como podemos ver na imagem.

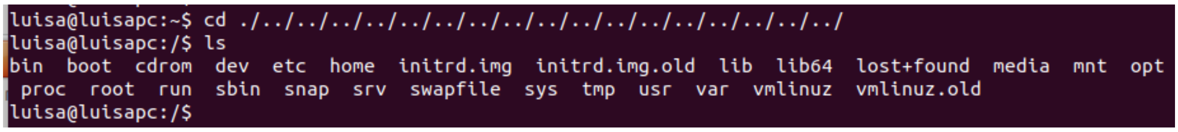

Figure 1.1: Diretoria onde se encontra a pasta *etc*

Desta forma, conseguimos obter o conteúdo do ficheiro */etc/passwd* se acrescentarmos /../ suficientes de forma a não só irmos obter à diretoria onde se encontra a pasta */etc/* como também a preencher o *buf* de forma a não ser possível adicionar o sufixo ’.txt’. Em baixo, encontra-se a exploração das vulnerabilidades apresentadas assim como parte do *output* do programa, tal como era pretendido.

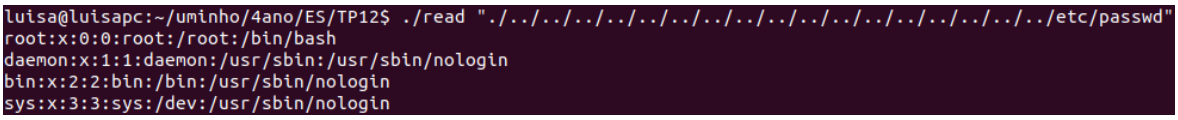

Figure 1.2: Explorações das vulnerabilidades mencionadas

1.2 Pergunta P1.2

Para implementar o programa em questão decidimos criar uma função para cada um dos inputs. Estas funções devem receber o *input*, fazer a sua verificação e caso seja válido então deve ser devolvido, caso contrário deve aparecer uma mensagem de erro a indicar o *input* inválido e o programa deve terminar logo a seguir.

Desta forma, a função que se segue verifica o *input* do montante. Assim sendo, começa-se por utilizar uma expressão regular para verificar se existe pelo menos um número antes do ”.” e se existem exatamente 2 números depois do ”.”, dado que os montantes devem ser do tipo ”0.12 €” ou ”150.24”, por exemplo. Caso a expressão regular não encontre um *match* com o *input* introduzido então é porque o montante está mal formatado e deve ser devolvida a mensagem que aquilo que o utilizador introduziu está errado. Por conseguinte, caso esteja bem formatado, é feita uma verificação adicional no `cast` do valor para um `float`. Se não for possível fazer esta conversão é lançada uma exceção que é capturada, sendo que, neste momento informa-se mais uma vez que o valor introduzido não é correto e o programa termina.

Por fim, caso o valor esteja corretamente introduzido devolve-se o resultado do `cast` do montante para `float`.

```python
def getAmount():
    amount = input("Introduza o montante a pagar: ")
    res = re.match("[0-9]+\.[0-9]{2}", amount)


    if not res:
```

```
7           print("Valor do montante não é válido")
8           sys.exit()
9
10      try:
11          amount = float(amount)
12
13      except(ValueError, TypeError):
14          print("Valor do montante não é válido")
15          sys.exit()
16
17      return amount
```

A função que se segue recebe a data de nascimento, sendo que, tal como a anterior começa-se por verificar se a data introduzida pelo utilizador encontra-se na forma "YYYY/MM/DD" através de uma expressão regular e caso não esteja introduzida corretamente é devolvido um erro para o ecrã do utilizador. Por conseguinte, utiliza-se a classe `datetime` do python para verificar se a data é válida, ou seja, se os meses encontram-se entre 1 e 12, se foi introduzido o dia 29 para o mês 2 então o ano tem de ser bissexto, entre outras verificações. Caso este passo não lance nenhuma exceção então este objeto "datetime" é devolvido pela função.

```
1   def getBirthdate():
2
3       birthdate = input("Introduza a data de nascimento (YYYY/MM/DD): ")
4       res = re.match("[0-9]{4}\/[0-9]{2}\/[0-9]{2}", birthdate)
5
6       if not res:
7           print("Data de nascimento inválida")
8           sys.exit()
9
10      else:
11          res = birthdate.split("/")
12          year = int(res[0])
13          month = int(res[1])
14          day = int(res[2])
15
16          try:
17              birthdate = datetime.date(year = year, month = month, day =
                ↪  day)
18              return birthdate
19
20          except(ValueError, TypeError):
21              print("Data de nascimento inválida!")
22              sys.exit()
23
24      return birthdate
```

De seguida, a função que verifica o nome introduzido apenas tem cuidado de verificar

se o nome introduzido não apresenta caracteres diferentes de a-z ou A-Z, através da utilização de uma expressão regular.

```
def getName():
    name = input("Introduza o seu nome: ")

    res = re.match("[^a-zA-Z]", name)

    if res == None:
        print("Nome inválido")
        sys.exit()

    return name
```

As seguintes funções permitem a validação do número de identificação fiscal (NIF), sendo que a segunda verifica se o *input* é um conjunto de 9 números, cada um entre 0 e 9 e se o for, é invocada a primeira função, que valida o número em si e é baseada no módulo 11 do Governo Português.

```
def validNIF(numero):

    soma = sum([int(dig) * (9 - pos) for pos, dig in
    ↪  enumerate(numero)])
    resto = soma % 11
    if (numero[-1] == '0' and resto == 1):
        resto = (soma + 10) % 11

    return resto == 0

def getNIF():

    nif = input("Introduza o seu NIF: ")

    res = re.match("^[0-9]{9}$", nif).group(0)

    if res == None:
        print("NIF inválido")
        sys.exit()

    if not validNIF(res):
        print("NIF inválido")
        sys.exit()

    return nif
```

O número de identificação de cidadão (NIC) é validado tendo como base uma expressão regular que indicate que este é constituído por 8 números de 0 a 9.

```
def getNIC():

```

```
    nic = input("Introduza o seu NIC: ")

    res = re.match("^[0-9]{8}$", nic)

    if res == None:
        print("NIC inválido")
        sys.exit()

    return nic
```

Para validar as informações de um cartão de crédito são utilizadas 3 funções auxiliares, invocadas pela função abaixo.

```
def getCCInfo():

    ccnum = getCCNum()

    expiryDate = getExpiryDate()

    cvc = getCVC()

    return ccnum, expiryDate, cvc
```

A primeira das funções auxiliares está demonstrada abaixo e, trata da validação do número do cartão de crédito. Este deverá ser representado por 4 conjuntos, separados entre si por um espaço, de 4 números, cada um compreendido entre 0 e 9.

```
def getCCNum():

    ccnum = input("Introduza o seu número de cartão de crédito: ")

    res = re.match("^[0-9]{4}\s[0-9]{4}\s[0-9]{4}\s[0-9]{4}$", ccnum)

    if res == None:
        print("Número de cartão de crédito inválido")
        sys.exit()

    return ccnum
```

A segunda função auxiliar valida a data de validade do cartão de cidadão. Nela é utilizada uma expressão regular que aceita apenas valores sobre o formato MM/YY (mês/ano), sendo portanto um total de quatro dígitos em que os 2 primeiros são o mês e os últimos o ano. De seguida, se houver uma correspondência para a expressão mencionada verifica-se se o mês está contido entre 1 e 12.

```
def getExpiryDate():

    expirydate = input("Introduza a data de validade (MM/YY): ")
```

```
    res = re.match("[0-9]{2}\/[0-9]{2}", expirydate)

    if res == None:
        print("Data de validade inválida")
        sys.exit()

    else:
        res = expirydate.split("/")
        month = int(res[0])

        if month >= 1 and month <= 12:
            return expirydate

        else:
            print("Data de validade inválida!")
            sys.exit()
```

Quanto à última função auxiliar, esta trata da validação do CVC/CVV e indica que este deverá ser constituído por 3 valores compreendidos entre 0 e 9.

```
def getCVC():

    cvc = input("Introduza o seu CVC/CVV: ")

    res = re.match("^[0-9]{3}$", cvc)

    if res == None:
        print("NIC inválido")
        sys.exit()

    return cvc
```

O ficheiro com o código completo encontra-se no seguinte link.