Universidade do Minho

Mestrado Integrado em Engenharia Informática

Engenharia de Segurança

Trabalho TP6

Grupo 2
Paulo Gameiro - A72067
Pedro Rodrigues - PG41092
Rafaela Soares - A79034

Braga, Portugal
6 de Abril de 2020

Conteúdo

1 Pergunta P1.1 **2**
1.1 1 2
1.1.1 CWE-119 2
1.1.2 CWE-79 2
1.1.3 CWE-20 3
1.2 2 4

2 Pergunta P1.2 **6**
2.1 1 6
2.2 2 6

3 Pergunta P1.3 **7**
3.0.1 Vulnerabilidades de Projeto 7
3.0.2 Vulnerabilidades de Codificação 7
3.0.3 Vulnerabilidades Operacionais 7

4 Pergunta P1.4 **8**

1 Pergunta P1.1

1.1 1

1.1.1 CWE-119

A *Weakness* CWE-119, *Improper Restriction of Operations within the Bounds of a Memory Buffer*, permite que o *software* tenha a possibilidade de ler/escrever a partir de/num local de memória que não seja o definido no *buffer* de memória, no qual está a executar operações.

Tal é possível, uma vez que "algumas linguagens de programação permitem o endereçamento direto dos locais da memória e não garantem automaticamente que esses locais sejam válidos para o *buffer* de memória que está a ser referenciado", o que leva a que operações de leitura e escrita possam ser executadas em locais fora do pretendido.

Relativamente às linguagens de programação que permitem o que foi descrito, destacam-se as linguagens C (*Often Prevalent*), C++ (*Often Prevalent*) e *Class*: *Assembly* (*Undetermined Prevalence*).

Caso um *attacker* tenha conhecimento desta *Weakness*, este pode colocar colocar em causa a confidencialidade, integridade e disponibilidade do sistema em questão, através de *memory corruption*.

As consequências mais comuns são a execução de código ou comandos não autorizados e modificação de memória, o que tem um impacto na confidencialidade (o *attacker* pode redirecionar um ponteiro de função para o seu próprio código malicioso), integridade (o *attacker* pode modificar o *buffer*) e disponibilidade (pode ocorrer *overflow*, devido à injeção de código arbitrário por parte do *attacker*).

Para além disso, podem explorar o *exploit* DoS (o que tem impacto na disponibilidade) e ler a memória (o que coloca em causa a confidencialidade).

1.1.2 CWE-79

A *Weakness* CWE-79, *Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting'*, refere-se à falta de validação ou validação incorreta, por parte do *software*, do *user-controllable input*, antes de este ser colocado no *output* utilizado como página *web*. De referir que a página *web* abrange outros utilizadores.

Esta *Weakness* resulta em *Cross-site Scripting* - ataque comum baseado na injeção de *client-side script*, de forma maliciosa, através de uma aplicação *web*.

Tal pode ocorrer, de acordo com CWE, quando:

1. Dados não confiáveis entram num aplicação *web*, geralmente a partir de uma solicitação *web*.
2. A aplicação *web* gera dinamicamente uma página *web* que contém esses dados não confiáveis.
3. Durante a geração da página, a aplicação não impede que os dados contenham conteúdo executável por um navegador *web*, como JavaScript, HTML *tags*, atributos HTML, eventos de rato, Flash, ActiveX, entre outros não menos relevantes.
4. A "vítima"(utilizador alvo) visita a página *web*, gerada através de um navegador *web*, que contém *scripts* mal-intencionados (injetados mediante os dados não confiáveis).

5. Como os *scripts* são provenientes de uma página *web* enviada pelo servidor, o navegador da "vítima"executa os *scripts* mal-intencionados, no contexto do domínio do servidor.

Como já foi mencionado, tal se pode suceder pela utilização da tecnologia *Web Based* (*Often Prevalent*) e por *Class: Language-Independent (Undetermined Prevalence)*.

Caso um *attacker* tenha conhecimento desta *Weakness*, este pode colocar colocar em causa a confidencialidade, integridade, controlo de acesso e disponibilidade.

As consequências mais comuns são a execução não autorizada de código e comandos (o que pode colocar em causa a integridade, disponibilidade e confidencialidade), leitura de dados da aplicação (como *cookies*, por exemplo, o que coloca em causa a confidencialidade) e *Bypass Protection Mechanism* (que condiciona o controlo de acesso e confidencialidade).

1.1.3 CWE-20

A *Weakness* CWE-20, *Improper Input Validation*, refere-se à falta de validação ou validação incorreta das entradas que podem condicionar o fluxo de controlo ou o fluxo de dados de um programa.

Se o *software* não validar os *inputs* corretamente, o *attacker* tem a possibilidade de elaborar *inputs* imprevisíveis, o que pode levar a inputs não intencionais, alteração do fluxo, controlo arbitrário de recursos e execução arbitrária de código. Tal pode se suceder pela utilização de *Class: Language-Independent (Undetermined Prevalence)*.

Caso um *attacker* tenha conhecimento desta *Weakness*, este pode colocar colocar em causa a confidencialidade, integridade e disponibilidade do sistema em questão.

As consequências mais comuns são a exploração do *exploit* DoS (o que tem impacto na disponibilidade), leitura de memória, ficheiros e diretorias (o que coloca em causa a confidencialidade), modificação de memória (o que tem impacto na integridade) e execução de códigos ou comandos não autorizados(o que pode colocar em causa a disponibilidade e confidencialidade).

1.2 2

A *Weakness* CWE-125, *Out-of-bounds Read*, consiste na leitura de dados após o final ou antes do início do *buffer* em questão pelo *software*, o que permite que um *attacker* possa ler informações confidenciais de outros locais de memória ou causar falhas.

Relativamente às linguagens de programação que permitem o que foi descrito, destacam-se as linguagens C (*Undetermined Prevalence*) e C++ (*Undetermined Prevalence*).

Caso um *attacker* tenha conhecimento desta *Weakness*, este pode colocar colocar em causa a confidencialidade. As consequências mais comuns são a leitura de memória e *Bypass Protection Mechanism* (ao ser lido memória fora dos limites estabelecidos, um *attacker* pode obter valores que não deviam ser expostos, como endereços de memória).

No intuito de contornar tais consequências, é necessário verificar se o *index* a ser consultado está dentro dos limites do *buffer*. Isto é, se é maior ou igual a 0 e menor que o tamanho do *buffer*. Caso não seja, é necessário retornar como *output* um valor que não seja o armazenado, como por exemplo, -1.

Segue-se, de seguida, como sugestão do CWE, uma função em C que implemente o descrito:

```
int getValueFromArray(int *array, int len, int index) {

  int value;

  // check that the array index is less than the maximum
  // length of the array

  if (index >= 0 && index < len) {

     // get the value at the specified index of the array
     value = array[index];
  }

  // if array index is invalid then output error message
  // and return value indicating error

  else {
     printf("Value is: %d\n", array[index]);
     value = -1;
  }

  return value;
}
```

Já um exemplo de uma CVE que inclua a *Weakness* abordada, CWE-20, é a CVE-2004-0183. Esta vulnerabilidade permite aos *attackers* remotos explorarem o *exploit* DoS, através de pacotes ISAKMP (*Internet Security Association and Key Management Protocol*), que contêm *Delete payload*

com um elevado número de SPIs, o que leva à leitura, fora dos limites estabelecidos, do *buffer* em questão. Tal se sucede em TCPDUMP 3.8.1 e versões anteriores.

Desta forma, e como podemos constatar intuitivamente na imagem abaixo, tal é feito através de Internet (AV), uma vez que o *attacker* envia pacotes ISAKMP. Como não é necessário autenticação, apresenta baixa complexidade de acesso (AC).

Embora não coloque em causa a integridade do sistema, esta vulnerabilidade tem impacto na disponibilidade, originando interrupção do mesmo.

De referir que apesar de não ser referido nesta imagem que esta vulnerabilidade tem impacto na confidencialidade, acredita-se que sim, uma vez que existe possibilidade de leitura.

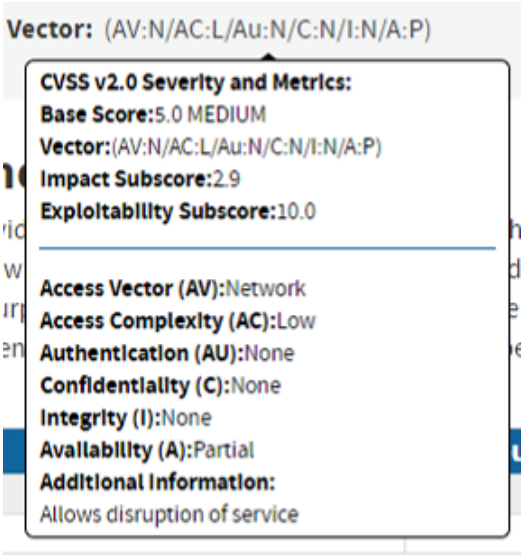

Figura 1: Vulnerabilidade CVE-2004-0183

2 Pergunta P1.2

2.1 1

De acordo com os *slides* fornecidos da aula 9, "estima-se que qualquer pacote de *software* tem uma média de 5 a 50 *bugs* por cada 1000 SLOC (*Source Lines Of Code* – linhas de código fonte, excluindo linhas de comentário".

De referir que o limite superior para *software* normal é de 50 *bugs* por 1.000 SLOC e que o limite inferior para *software* desenvolvido utilizando métodos de desenvolvimento rigoroso é de 5 *bugs* por 1.000 SLOC.

Ou seja, para estimar o limite superior do número de *bugs* de X linhas de código, recorrer-se-á à expressão **número de *bugs* = (50* X) /1000** .

Já para se estimar o limite inferior do número de *bugs* de X linhas de código, recorrer-se-á à expressão **número de *bugs* = (5* X) /1000**.

Posto isto, apresenta-se, de seguida, a estimativa do número de *bugs* do Facebook, *Software* de automóveis, Linux 3.1 e de todos os serviços Internet da Google, tendo sido utilizado o *site* **informationisbeautiful** como referência para o número de linhas de código.

	Nº linhas de código	**Limite inferior**	**Limite superior**
Facebook	62 milhões linhas	310 mil *bugs*	3.1 milhões *bugs*
Software de automóveis	100 milhões linhas	500 mil *bugs*	5 milhões *bugs*
Linux 3.1	15 milhões linhas	75 mil *bugs*	750 mil *bugs*
Serviços Internet Google	2 mil milhões linhas	10 milhões *bugs*	100 milhões *bugs*

2.2 2

No que concerne ao número de quantos destes *bugs* são vulneráveis, tal é inconclusivo, uma vez que não é possível estimar o número de vulnerabilidades por número de *bugs* encontrados.

3 Pergunta P1.3

Depois de encontradas, é atribuída uma categoria a cada uma das vulnerabilidades, sendo que as vulnerabilidades de Projeto estão associadas a falhas que ocorrem ainda durante a fase de planeamento do *software*. As vulnerabilidades de Codificação correspondem a falhas ocorridas durante o desenvolvimento

No caso das vulnerabilidades Operacionais, estas estão associados ao sistema ou ambiente onde o *software* é executado, através de falhas que podem permitir aceder a outras funcionalidades fora das que seria suposto.

3.0.1 Vulnerabilidades de Projeto

CVE-2019-5638

Esta vulnerabilidade presente no **Rapid7 Nextpose 6.5.50** representava uma expiração de sessões insuficiente, por exemplo, no caso de um *leak* de credenciais dos utilizadores, se o administrador alterasse as *passwords* dos *users*, as sessões iniciadas continuariam válidas e como tal seria possível para os atacantes continuar a utilizar com a sessão iniciada.

Esta vulnerabilidade poderia ser resolvida se a sessão expirasse após um determinado tempo.

CVE-2015-4495

O leitor de PDF das versões do Mozilla Firefox anteriores a 39.0.3, entre outras, permitia que atacantes conseguissem ler ficheiros ou escalar privilégios remotamente, sendo que esta vulnerabilidade foi resolvida nas versões mais recentes do *browser*.

3.0.2 Vulnerabilidades de Codificação

CVE-2018-5447

A validação de *input* deve realizada corretamente, de forma a evitar a ocorrência de vulnerabilidades como esta, em que um atacante poderia aceder remotamente aos recursos do sistema e afetar a sua disponibilidade.

CVE-2019-14027

Esta vulnerabilidade está associada à falta de verificação de tamanho máximo de um canal, utilizado num *loop* em diversos processadores snapdragon, que levava à ocorrência de um *buffer ouverflow*.

3.0.3 Vulnerabilidades Operacionais

CVE-2019-3826

Através desta vulnerabilidade, um atacante poderia fazer com que um utilizador autenticado (num *site* num servidor Prometheus) visitasse um determinado url de um *site* armazenado no mesmo servidor, o que permitia que esse atacante executasse *scripts* arbitrariamente. Esta vulnerabilidade poderia ser corrigida utilizando um servidor diferente.

4 Pergunta P1.4

Uma vulnerabilidade é uma falha não intencional encontrada no *software* ou sistema operativo. Já uma vulnerabilidade dia-zero é uma falha de *software* que é conhecida pela entidade patronal responsável pelo *software*, mas que não existe nenhum *patch*, de momento, que consiga solucionar a falha.

Posto isto, a diferença entre as duas é que a vulnerabilidade dia-zero é uma vulnerabilidade que ainda não tem um *"patch"* que consiga solucionar a falha.