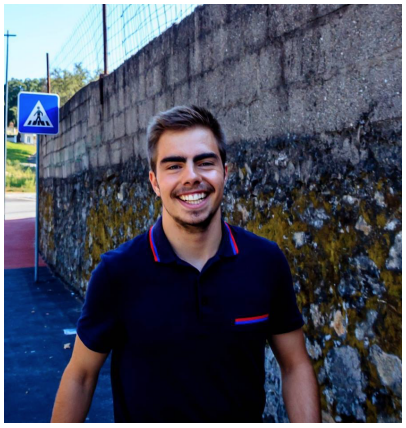
Escola de Engenharia
Universidade do Minho

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
Mestrado em Engenharia Informática
Engenharia de Segurança

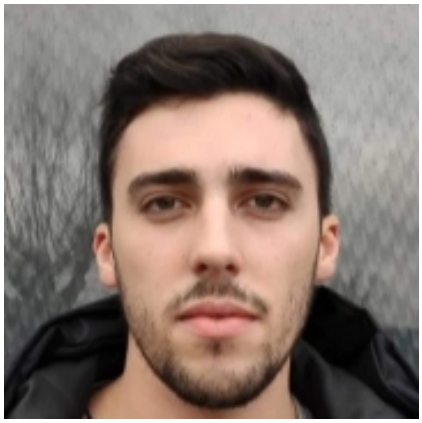
Aula 7

23 de Março de 2020

Grupo 1



Ricardo Pereira a73577



Tiago Ramires pg41101

Braga, 6 de Abril de 2020

1. Vulnerabilidade de codificação

Pergunta 1.1 - *Common Weakness Enumeration* (CWE)

1

Rank 1

A primeira *weakness* tem que ver com operações feitas em *buffers* da memória. O *software* faz operações num determinado buffer numa determinada posição de memória mas consegue também aceder a outras posições da memória, algo que não devia ser permitido. Estes problemas acontecem principalmente com as linguagens **C** e *C++*, verificando-se porém em **Class: Assembly**.

As consequências que esta *weakness* pode ter passam, por exemplo, pela execução de código malicioso através da modificação do *function pointer* (integridade, confidencialidade, disponibilidade), pelo colapso ou indisponibilidade do sistema (disponibilidade, confidencialidade) ou até pelo acesso a informação sensível (confidencialidade).

Rank 2

A segunda *weakness* está relacionada com falta de controlo do software para determinadas ações *input* que são formuladas pelo utilizador e que se reproduzem como *outputs* numa página *web* utilizada por outros. Este problemas não acontecem com uma linguagem específica e estão principalmente relacionados com tecnologias *web*.

As consequências que lhes estão associadas são a execução de ataques através da injeção de *scripts* em *web browsers*, permitindo assim o acesso aos *cookies* de utilizadores (controlo de acesso, confidencialidade), a combinação de outras falhas com a anterior que permite a execução de código arbitrário no computador de um utilizador (integridade, confidencialidade, e disponibilidade) e ainda a execução de ataques *XSS* que partilham algumas das consequências faladas anteriormente (confidencialidade, integridade, disponibilidade, controlo de acesso).

Rank 3

A terceira envolve a má ou inexistente validação de *inputs* num programa, afetando assim o fluxo de controlo ou de dados do mesmo. A informação disponível não especifica o tipo de tecnologias em que ocorre, lembrando apenas que é independente da linguagem inerente ao programa.

As consequências a enfrentar são a introdução de valores inesperados num determinado programa, causando o seu colapso (disponibilidade), a divulgação de dados, caso o atacante tenha controlo das referências de recursos (confidencialidade) e ainda a modificação de dados ou a execução de comandos arbitrários através da introdução de *inputs* maliciosos (integridade, confidencialidade, disponibilidade).

2

Weakness com rank 4

A *weakness* atribuida ao nosso grupo é a que está em 4º no *rank*. O produto em que a mesma se verifica expõe informação a determinados atores que nunca lhes deveria estar acessível. As tecnologias em que a mesma se verifica são transversais às linguagens que utilizam, ocorrendo principalmente em telemóveis. A *weakness* tem grande impacto na confidencialidade dos utilizadores.

A seguir temos um exemplo de código onde reside uma *weakness*. Como se pode ver no mesmo, está a ser tratada uma exceção que o atacante pode conseguir ver, e conseguindo fazê-lo, a exceção contém informação que poderá dar acesso aos ficheiros de configuração e por conseguinte ler as credenciais neles presentes ou alteração da base de dados que a aplicação está a usar.

```
try {
    openDbConnection();
}
//print exception message that includes exception message and configuratio
catch (Exception $e) {
    echo 'Caught␣exception:␣', $e->getMessage(), '\n';
    echo 'Check␣credentials␣in␣config␣file␣at:␣', $Mysql_config_location,
}
```

Listing 1.1: Piece of Java Code

A *weakness* em questão pode-se verificar, entre outros, nos seguintes *CVE*'s:

- CVE-2001-1483
- CVE-2001-1528
- CVE-2004-2150
- CVE-2005-1205

Pergunta P1.2

1

O rácio de *bugs* por linhas de código, ao contrário do que pensamos, é algo relativamente constante e independente das linguagens utilizadas, segundo *Steve McConnell*, autor de *Code Complete*, existem cerca de 15 a 50 *bugs* em 1000 linhas de código.

- *Facebook* - 930 mil - 3.1 milhões.
- *Software* de automóveis - 1.5 milhões - 5 milhões.
- *Linux 3.1* - 225 mil - 750 mil.
- *Google* - 30 milhões - 100 milhões.

Estes valores na realidade é bastante inferior ao estimado anteriormente, dada a dimensão das empresas e o número de *bugs* já resolvidos por as mesmas.

2

Não existe uma relação direta do número de vulnerabilidades e *bugs*, pelo que é difícil estimar e atribuir a cada uma das entidades em questão o número de *vulnerabilidades* que lhe está associado, mas analisando o número de *CVE's* existentes em relação ao número de linhas de código, estima-se que o número de vulnerabilidades seja de 0.05 por 1000 linhas de código.

- *Facebook* - 3.1 mil.
- *Software* de automóveis - 5 mil.
- *Linux 3.1* - 750.
- *Google* - 100 mil.

Pergunta P1.3

Existem três tipos de vulnerabilidades: de projeto, de codificação e operacionais. As *vulnerabilidades de projeto* são introduzidas durante a fase de projeto do *software*, as de codificação durante a programação do *software* e as operacionais existem no ambiente onde o *software* é executado.
Vulnerabilidades de projeto:

- Atrás falamos na má ou **inexistente validação de inputs** e agora verificamos que essa *weakness* se enquadra neste tipo. Mais uma vez, esta *weakness* afeta o fluxo de controlo ou de dados do programa e a solução para a sua resolução passaria por introduzir uma forma de validação desses *inputs*.
- Outra *weakness* que se enquadra neste tipo é a **falta de aviso de ameaças** para o utilizador comum, ou seja, quando este está a ser exposto a determinados perigos sem o saber. Resolver este problema passa por definir nos requisitos da aplicação que devem surgir avisos para o utilizador da aplicação acerca de todos os problemas que o mesmo pode vir a ter por usar aquela aplicação.

Vulnerabilidades de codificação:

- A **execução de programas com privilégios altos** é um problema e ocorre muitas vezes porque um programador insconscientemente o permite quando está a escrever o código daquele programa. A solução para este problema passa por mostrar aos programadores que tipo de aspetos é que devem ter em conta para não permitirem que isto aconteça.
- A **execução de comandos ou utilização de bibliotecas de fontes desconhecidas** pode levar a que um atacante execute comandos maliciosos para seu proveito. Assim, deve-se sempre verificar a autenticidade das fontes de componentes que se utilizam para a construção de programas.

Vulnerabilidades operacionais:

- A **apresentação de mensagens de erro detalhadas** pode ser muitas vezes um problema na medida em que permite ao utilizador comum (possíveis atacantes) deslindar informação que sem essas mensagens nunca seria visível. Com essas mensagens é possível perceber como está o sistema a funcionar e por esse motivo é imperativo reduzir o detalhe de informção que é apresentada.

- A **disponibilidade de um serviço face à sua grande procura** por parte de muitos utilizadores pode abrir caminho para um problema com grande potencial para fazer estragos principalmente a níveis financeiros. Por esta razão é importante fazer a previsão das quantidades de clientes que podem estar ligados a um servidor bem como testes para aferir a disponibilidade permanente destes serviços.

Pergunta P1.4

Uma **vulnerabilidade dia-zero** é bastante diferente de outra vulnerabilidade uma vez que é aquela que é conhecida apenas por um determinados grupo de pessoas, não sendo do conhecimento do resto da comunidade informática. Tais vulnerabilidades conseguem adquirir um valor enorme uma vez que acabam por ser portas de entrada para aceder a sistemas críticos, sendo por isso bastante úteis (ou não) em ambiente militar, empresas cujo trabalho é encontrar as mesmas, entre outros.