Departamento de Informática da
Faculdade de Ciências da
Universidade de Lisboa

# Segurança de Software

Exame de Exemplo

**ATENÇÃO:**

- A duração do exame é de 2,5 horas sem tolerância.
- Só poderá haver desistências do exame após a 1ª hora.
- As cotações das perguntas encontram-se entre parênteses.
- Todas as respostas são dadas directamente neste enunciado.
- Antes da resposta final, utilize folhas de rascunho de modo a não "sujar" este enunciado.
- Se estiver a fazer melhoria, escreva "melhoria" no canto superior direito desta página.
- As respostas de escolha múltipla e do tipo "totobola" são cotadas da seguinte maneira:
  - resposta certa: cotação total.
  - ausência de resposta: zero valores.
  - resposta errada: desconta-se a cotação da pergunta dividida pelo nº de alternativas.

Nome:

Nº:

**1. (1)** Discuta as vantagens e desvantagens, em termos de segurança de software, da utilização de código aberto vs a utilização de código fechado.

**2. (1)** Considere um PC doméstico e um sítio de comércio electrónico, ambos ligados à Internet. Qual deles está sujeito a um maior risco? Justifique.

**3. (1,5)** Descreva os quatro principais tipos de separação entre processos que podem existir num sistema operativo, apresentando as vantagens e desvantagens de cada um deles em termos de segurança e complexidade de concretização.

**4. (3,5)** Considere o programa *oops* que executa com privilégios de *superuser* e é composto pelo seguinte código-fonte:

```
1  void imprime(char *s) {
2     char buf[64];
3     strncpy(buf, s, 64);
4     buf[63] = '\0';
5     printf(buf);
6  }
7
8  int main(int argc, char **argv){
9     imprime(argv[1]);
10    return 0;
11 }
```

**a) (1,5)** Identifique a vulnerabilidade presente neste programa e explique como é que poderia ser explorada por um atacante para ler o conteúdo de endereços de memória arbitrários.

**b) (1)** Como é que uma ferramenta de análise de fluxo de dados (e.g., CQual) processaria o programa *oops* e detectaria a vulnerabilidade identificada na alínea a)?

**c) (1)** Apresente uma versão corrigida do programa *oops* que não contenha a vulnerabilidade identificada na alínea a).

**5. (1,5)** Compare os ataques de *heap overflow* e *stack overflow* em termos de facilidade de execução e danos potenciais.

**6. (1,5)** Descreva os quatro principais tipos de vulnerabilidades de inteiros.

**7. (1,5)** Explique as diferenças entre as vulnerabilidades de *Cross Site Scripting* (XSS) e *Cross Site Request Forgery* (CSRF) e apresente uma discussão sobre qual das duas vulnerabilidades é mais crítica num sistema de *homebanking*.

**8. (1,5)** Relativamente às vulnerabilidades de XSS por reflexão (XSS-R) e XSS por armazenamento (XSS-A), indique quais das seguintes afirmações se aplicam à vulnerabilidade de **XSS-R (1)**, à vulnerabilidade de **XSS-A (2)** ou a **Ambas (X)**.

| | **XSS-R (1)** | **Ambas (X)** | **XSS-A (2)** |
|---|---|---|---|
| O atacante envia o script malévolo para o servidor vulnerável. | | | |
| A vítima não necessita de estar autenticada para a vulnerabilidade ser explorada. | | | |
| Pode revelar informação utilizável mais do que uma vez. | | | |
| A vítima envia o script malévolo para o servidor vulnerável. | | | |
| É parcialmente resolvida através de uma codificação adequada do output. | | | |

**9. (1)** Considere uma aplicação web desenvolvida em Java que utiliza um determinado SGBD e que é acedida pelos utilizadores via browser. Dado que os comandos SQL enviados ao SGBD incluem parâmetros especificados pelo utilizador, esta aplicação inclui potencialmente vulnerabilidades de injecção de SQL. Das seguintes afirmações, indique a única que é **VERDADEIRA**:

- ❑ É difícil explorar este tipo de vulnerabilidade se os comandos SQL forem produzidos pelo código Java da aplicação Web usando *queries* parametrizadas.
- ❑ É difícil explorar este tipo de vulnerabilidade se a interface do utilizador incluir campos de texto que apenas permitem a introdução de valores numéricos.
- ❑ É difícil explorar este tipo de vulnerabilidade se os comandos SQL estiverem *hardcoded* no HTML.
- ❑ É difícil explorar este tipo de vulnerabilidade se a interface do utilizador não incluir campos de texto.
- ❑ Nenhuma das afirmações anteriores é verdadeira.

**10. (1)** Identifique e descreva dois mecanismos de injecção de SQL de primeira ordem e explique o que se entende por injecção de segunda ordem.

**11. (2,5)** Considere o programa *ooops* que executa com privilégios de *superuser* e é composto pelo seguinte código-fonte:

```
1   #include <stdio.h>
2   #include <unistd.h>
3   #include <sys/stat.h>

5   void escreve(char *file) {
6     FILE *f;
7     struct stat statb;
8     if (!lstat(file, &statb)){
9       if (S_ISREG(statb.st_mode)) {
10          f = fopen(file, “wb+”);
11          fwrite(“ola”, 3, 1, f);
12          fclose(f);
13      }
14      else {
15          fprintf(stderr, “Erro: %s eh um link.”, file);
16      }
17    }
18    else {
19      fprintf(stderr, “Erro ao obter informacao.”);
20    }
21  }

23  int main(int argc, char **argv){
24    escreve(argv[1]);
25    return 0;
26  }
```

**a) (1,5)** Identifique a vulnerabilidade presente neste programa e explique como é que poderia ser explorada por um atacante para corromper um ficheiro arbitrário.

**b) (1)** Explique como é que poderia ser corrigida a vulnerabilidade identificada na alínea a).

**12. (1,5)** Relativamente ao plugin Fuzzer do WebScarab e à ferramenta AJECT, indique quais das seguintes afirmações se aplicam ao plugin **Fuzzer (1)**, à ferramenta **AJECT (2)** ou a **Ambos (X)**.

| | **Fuzzer (1)** | **Ambos (X)** | **AJECT (2)** |
|---|---|---|---|
| Permite *fuzzing* por substituição. | | | |
| Suporta protocolos com estado. | | | |
| Permite fazer *fuzzing* a um protocolo arbitrário. | | | |
| Faz monitorização dos resultados. | | | |
| Requer a instalação de um componente no sistema alvo. | | | |

**13. (1)** No contexto das ferramentas de análise estática, explique as principais diferenças entre análise léxica e análise sintáctica.