

Conceitos Básicos

Comprometer a máquina de um utilizador... para quê?

1. Roubar credenciais do utilizador (normalmente via engenharia social)
2. Slansomware
3. Troubar poder de processamento
4. Usar o endereço de rede para parecer um utilizador legítimo

Comprometer servidores... para quê?

1. Fugas de dados
2. Motivações geo-políticas
3. Como forma de infectar máquinas dos utilizadores

Ataque à "Supply-Chain": infectar os servidores que distribuem software
Ataque ao Servidor Web: infectar o servidor web que é acedido pelos 1

0) que é "segurança"? Não existe definição universal
1. "O sistema comportar-se como esperado"

2. "A proteção de sistemas de computadores e redes contra divulgação da informação, roubo ou dano ao hardware, softwares ou dados eletrónicos, bem como contra a disruptão ou desorientação dos serviços que fornecem"

3. "Um sistema que permanece confiável diante da malícia"

"Engenharia de Segurança": "segurança de software é integrar práticas de segurança na forma como se constrói software; não integrar funcionalidades de segurança no código"

- A segurança é relativa → por si só não significa nada, depende sempre de quem a define e como
- A segurança muda com o contexto → tudo depende da aplicação concreta
- A segurança é defensiva → é definida na negativa: coisas más não podem acontecer

Atores: entidades que intervêm no sistema

- A segurança é definida da perspectiva dos atores!
- Alguns atores são confiáveis, outros não potenciam atacantes

Adversário/Atacante: ator com intenção explícita de usar o sistema/recursos de forma indireta e/ou invadir o seu uso

- Nenhum ator é seguro contra todos os atacantes
- É essencial conhecer o adversário (motivação, capacidades, ações)
- Um atacante vai sempre encontrar o elo mais fraco

ADVERSARIAL

THINKING

Modelo de Segurança

Correção: bom input → bom output

Correção c Segurança

Segurança: mau input ↛ mau output

↳ integridade: dados fiáveis

↳ confidencialidade: dados sensíveis nunca não revelados

Modelo Binário: seguro VS inseguro

- Defini formalmente as capacidades dos atacantes X "security prof"
- Defini formalmente os objetivos de segurança Y "secure by design"
- "Nenhum atacante limitado a X pode quebrar a segurança Y"

- ~ não escala para sistemas complexos
- ~ modelo abstrato ≠ sistema concreto
- ~ modelos formais podem ignorar problemas reais

ciclos mais longos

Modelo de Gestão de Risco: mais seguro VS menos seguro mitigação

- Minimizar o risco em relação às ameaças mais prováveis resistência
- "Mitigar o custo das medidas de segurança VS eventuais perdas" risco

- ~ a análise de risco pode estar errada

ciclos mais curtos

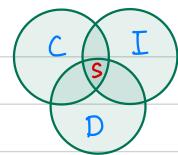
- ~ nunca se está realmente seguro

- ~ uma ameaça mal catalogada pode querer todas as garantias

Adversário descobre um novo ataque
PROCESSO CONTÍNUO

Defesa descobre uma nova defesa

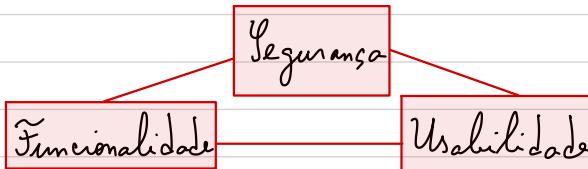
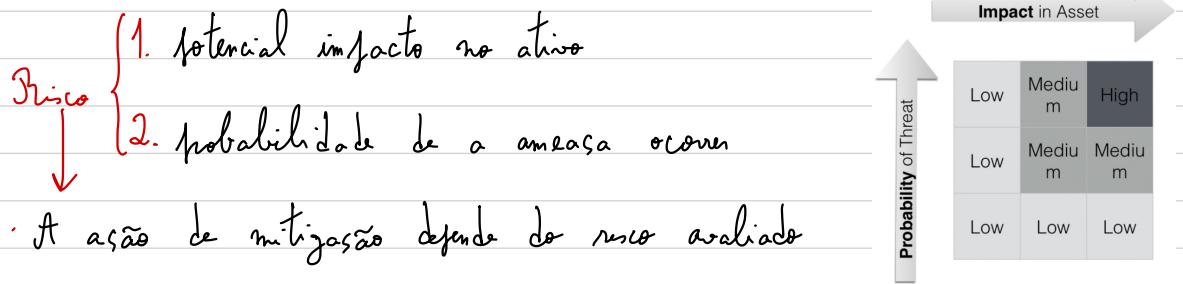
- 3 riscos
- Confidencialidade
 - Integridade
 - Disponibilidade



Quanto custa o ataque?
GESTÃO DO CUSTO
Quanto custa a defesa?

Ativo: recurso com valor para um ator do sistema

- Gera-se o risco de ativos relevantesarem usados inadequadamente ou indizponíveis e, assim, perderem o seu valor



Vulnerabilidade: falha latente que é acionada a um ataque, normalmente não intencional e originada de erros de design

Ataque: ocorre quando alguém tenta explorar uma vulnerabilidade

- Quando um ataque é bem-sucedido, o sistema fica comprometido

Ataque	<ul style="list-style-type: none"> · Passivo · Ativo · Dos 	Estrutura	<ul style="list-style-type: none"> · Motivo / Ameaça · Oportunidade / Vulnerabilidade · Método / Exploit: como explorar
---------------	---	------------------	--

Reportar: processo da comunicação para mitigar vulnerabilidades
 1. identificar 2. classificar 3. divulgar 4. detectar 5. mitigar 6. eliminar

Ameaça: incidente possível que pode trazer consequências negativas para o sistema, pessoa ou organização, dependente do contacto

- Tipicamente definida de acordo com o tipo de ataque/atacante e origem
- Identificada e classificada de acordo com a relevância

Modelo de Ameaça

Objetivo: proteger que ativos e contra quem?

Adversário: motivação? capacidade? tipo de acção?

Pensamento: contra que tipos de ataques devemos proteger?

Âmbito: que tipos de ataques podemos ignorar?

Perímetro de Segurança: fronteira que delimita um contacto com o mesmo nível de segurança — todos os impactos externos são suspeitos

Superfície de Ataque: pontos de contacto do perímetro de segurança com o exterior

Mecanismo de Segurança: método, ferramenta ou procedimento que permite implementar (parte de) uma política de segurança

- Parte do modelo de confiança consiste em confiar que um (conjunto de) mecanismo(s) de segurança serve(m) o seu propósito e está(o) bem implementados
- A política de segurança determina um conjunto de processos/mecanismos que deve ser implementado para garantir segurança de acordo com um modelo de ameaça definido — para prevenção/detecção/recuperação
- Em sistemas complexos, a segurança é lidada em camadas

Engenharia de Segurança: construir confiança via argumentos rigorosos

1. Defini o problema: análise dos requisitos e definição do modelo de ameaça
2. Defini o modelo de confiança
3. Defini a solução: políticas (o quê) e mecanismos (para quê)
4. Validar a solução

↳ Os mecanismos de segurança não são suficientes, juntamente com os caminhos de segurança, para implementar a política de segurança

• Auditorias de segurança provam validade deste processo

Testes de Penetração: método para validar empiricamente a solução de ^{segurança} de

Hacking Ético: simular um ataque para procurar vulnerabilidades que podem ser exploradas por um adversário

"Black Box": simulação de ataques reais

"White Box": com conhecimento privilegiado do sistema

Segurança de Software

1. Fráquezas no processo de desenvolvimento
2. Vulnerabilidades
3. Exploits

- Um usuário honesto interage com uma aplicação do software
- Um atacante malicioso tenta explorar uma vulnerabilidade do software
- A plataforma computacional em que o software executa é sempre uma potencial ameaça para essa plataforma, podendo permitir ganhar controle sobre ela

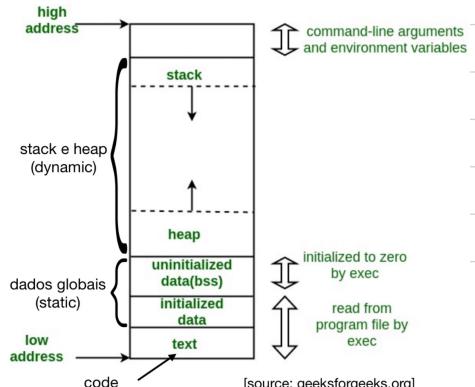
Violação da "Integridade do Fluxo de Controle": o atacante altera o fluxo de controle do programa para executar o seu código na máquina

Modelo de Ataque: o código do usuário recebe input desde fora do perímetro de segurança

Ex: buffer overflow - quando o programa acede fora da memória alocada

Modelo de Memória: cada processo tem acesso à memória virtual, simulada e gerida pelo sistema operativo

- Memory w/ code (**text**)
- Memory w/ global/static data (**data**)
- Memory allocated "on demand" (**heap**)
- Memory allocated automatically (**stack**):
 - return address
 - **temp** variables
 - function/procedure arguments



[source: geeksforgeeks.org]

Stack Smashing: ocorre quando um input externo leva a que o programa quebra a convenção de gestão da memória, substituindo a sequência esperada de instruções por instruções controladas pelo atacante

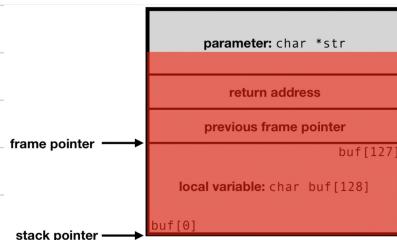
Stack Smashing

Stack Frame: contexto local no qual a função corre

Frame Pointer: apontador para o início/fim do stack frame

Stack Pointer: apontador para o fim/topo do stack frame

```
Ex: void func (char *st) {  
    char buf[128];  
    strcpy (buf, st);  
    msn (buf);  
}
```



- A função escreve para posições de memória crescentes
- Se st vier de fora do perímetro de segurança, o atacante pode preencher a stack com código, substituir o endereço de retorno e executar código arbitrário
- Não há separação entre código e dados!

Código Shell: sequência de instruções Assembly que executa comandos da shell — "lou" para injetar na memória e executar no retorno

Detalhes: preceber o código com NOP; não contém '\0' na codificação...

Um exploit robusto vai funcionar consistentemente em execuções arbitrárias do código vulnerável em plataformas similares

Heap Buffer Overflow

```
Ex: int main() {  
    char *buffer;  
    buffer = (char *) malloc(5 * sizeof(char));  
    char *string;  
    string = (char *) malloc(5 * sizeof(char));  
    strcpy(buffer, "abcdefghijklmnopqrstuvwxyz1234567890");  
    printf("%s", string);  
    return 0;  
}
```

Alocações para Funções; exception handlers; ...

Alocações para Funções: um objeto é guardado na heap, contendo um apontador para a sua tabela de funções internas (vTable)

Exception Handlers: o stack frame de uma função inclui uma lista ligada de apontadores para diferentes rotinas de atendimento de exceções

```
Ex: int main() {  
    char *buff1, *buff2;  
    buff1 = malloc(40);  
    buff2 = malloc(40);  
    gets(buff1);  
    free(buff1);  
    return 0;  
}
```

Heap Smashing:

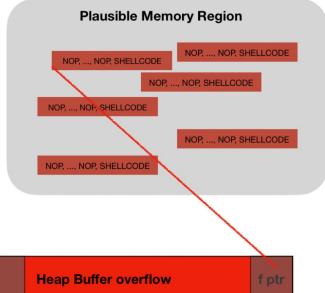
1. escrever o "payload" para a memória heap → por cima dos apontadores para blocos malloc/free
2. free alínea outras localizações de memória → escreve por cima de outros endereços de funções
- afaga um bloco numa lista ligada, reservando apontadores → se o bloco anterior/seguinte está livre, juntá-lo
- acaba por escrever por cima de um apontador para função com o endereço de código shell

Como criar um exploit?

1. Dominar programação Assembly de baixo-nível e debug de código binário
2. Perceber como fazer o overflow num ambiente controlado
3. Replicar o ambiente de execução do código alvo
 - a. Prever o endereço sobre o qual se pretende ganhar controlo
 - b. Prever o endereço no qual o código shell vai ser escrito
 - c. Enviar washers antes de ganhar controlo

Modelo de Memória do Browser: um servidor web malicioso que queria ganhar controlo da máquina de clientes pode executar código javascript nelas

"Heap Spraying": inundar a memória com código javascript malicioso com cópias de código shell e NOPs → com o objetivo de escrever por cima de apontadores na região alvo da memória, aumentando as possibilidades de ganhar controlo



"Use after Free": se, em linguagens com gestão dinâmica de memória, o código accede uma região da memória anteriormente libertada ou, em linguagens orientadas a objetos, o código chama um método de uma instância destituída de um objeto, então o adversário pode conseguir executar código shell, colocando-o nessa região de memória

Ex:

```
char *t1 = malloc(100);  
free(t1);  
char *t2 = malloc(100);  
strcpy(t2, bad);  
ptrs(t1); //apontador solto
```

Overflow de Inteiros: quando a representação de inteiros da máquina causa perda de informação — aritmética módulo N bits

Como usar um exploit?

1. Se o adversário controla um inteiro no input
2. Pode causar um overflow de inteiros numa computação intermédia
3. Causar a alocação de um buffer menor do que o esperado
4. Levar a um buffer overflow
5. Pode potencialmente executar código arbitrário ou causar uma falha

Leitura Fora dos Limites: valores negativos ou "off-by-one" — podem ler dados inválidos ou executar código

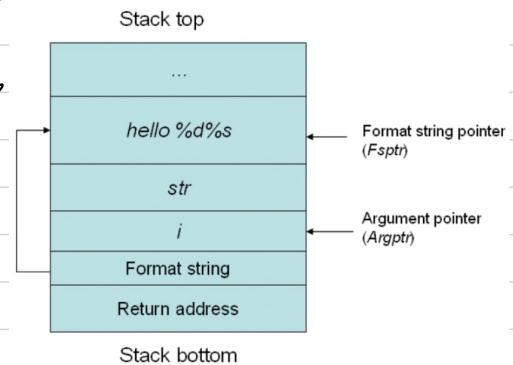
Ex: Heartbleed OpenSSL

Format Strings: a função `printf` recebe como primeiro argumento uma string de formatação e depois um número variável de parâmetros que vão para a stack (de cima para baixo)

1. string de formatação (array de caracteres na stack)

2. argumentos da string por ordem reversa

3. endereço da string de formatação



• `printf` atravessa a string de formatação e lê os argumentos imediatamente acima na stack

`printf ("%./d\n")`: imprime algo distante da stack

`printf ("%./s\n")`: imprime algo apontado por um endereço na pilha

`printf ("%<address>%./d%./d...%./s")`: permite ler qualquer endereço de memória (a string de formatação está na pilha!)

• Também é possível escrever num array de memória com `printf (dst, src)`, sendo que se `src` é controlado pelo atacante e contém `%./n`, o tamanho da região de memória a escrever é o tamanho da string de origem `src`

`if (stolen (src) < sizeof (dst)) printf (dst, src)`

%./n: armazena no endereço do argumento o número de caracteres impressos até ao momento

Se o atacante controlar a string de formatação, pode escrever para memória arbitrária

Para roubar o controle, um atacante tem de injetar código malicioso (código shell) em memória e mudar o controle para saltar para essa área e executar o código malicioso

CONTRA-MEDIDAS

1. Prevenção da Execução de Dados (DEP) ou W^X (Write XOR Execute): uma página de memória executável não pode ser escrita e vice-versa

2. Alteração do Layout das Estruturas de Endereçamento: a localização da memória de partes críticas do programa não baralhadas em cada exec.

"Just-In-Time Patching": como o código malicioso é compilado "on-the-fly", é possível alterar o compilador JIT para inserir código malicioso em tempo de execução — compilar constantes e saltar para "não endereços"; código executável; patch de código shell

Bibliotecas Ligadas: permitem reutilizar código que já está na memória executável, tendo endereços fáceis de puxar

Como usar libc como código shell? Usar técnicas de overflow, mas o endereço de retorno vai o de uma função da biblioteca — preparar a stack para simular uma chamada genuína a funções

A função "chamante" guarda parâmetros e o endereço de retorno

A função "chamado" guarda o frame pointer anterior e aloca suas próprias variáveis locais

Cada função limpa o seu espaço usado da stack, deixando o ambiente com o endereço de retorno automaticamente e a função "chamante" limpa os parâmetros

Como chamam a função X?

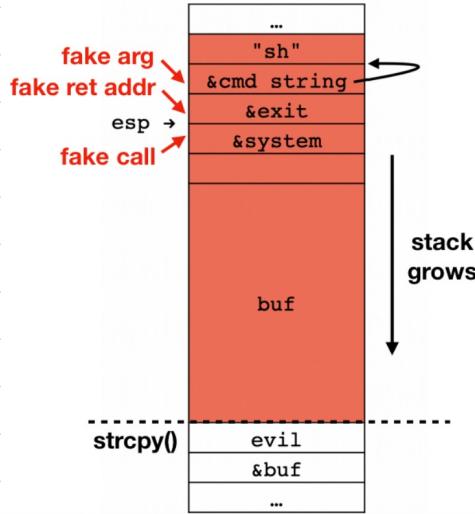
1. Substituir o endereço de retorno da função atual por X
2. Configurar a pilha para uma chamada válida a X:

X = system: o endereço da string com o comando shell invocado (parâmetros) e o endereço de retorno para quando system terminar

X = mprotect: o endereço de memória do código shell, o tamanho e as permissões (parâmetros) e o endereço de retorno para quando mprotect terminar → código shell

SYSTEM

```
void foo(char *evil) {  
    char buf[32];  
    strcpy(buf, evil);  
}
```

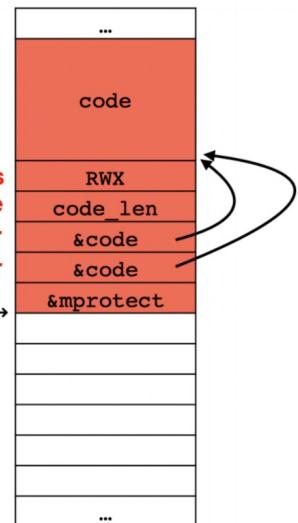


MPROTECT

```
int mprotect(void *addr,  
size_t len,  
int prot);
```

new permissions
memblock size
memblock addr
ret addr

esp →



loit

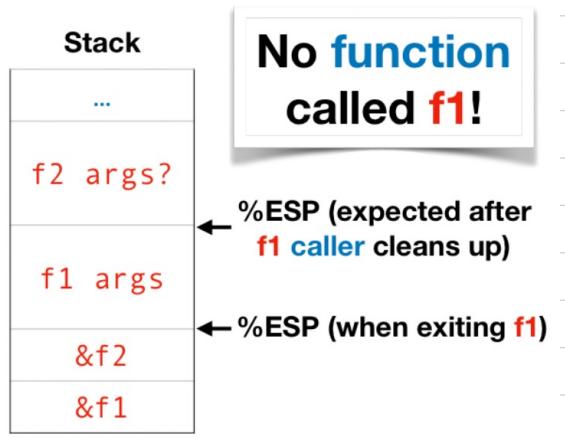
code

Programação Orientada aos Retornos: permite executar N funções em sequência

Como?

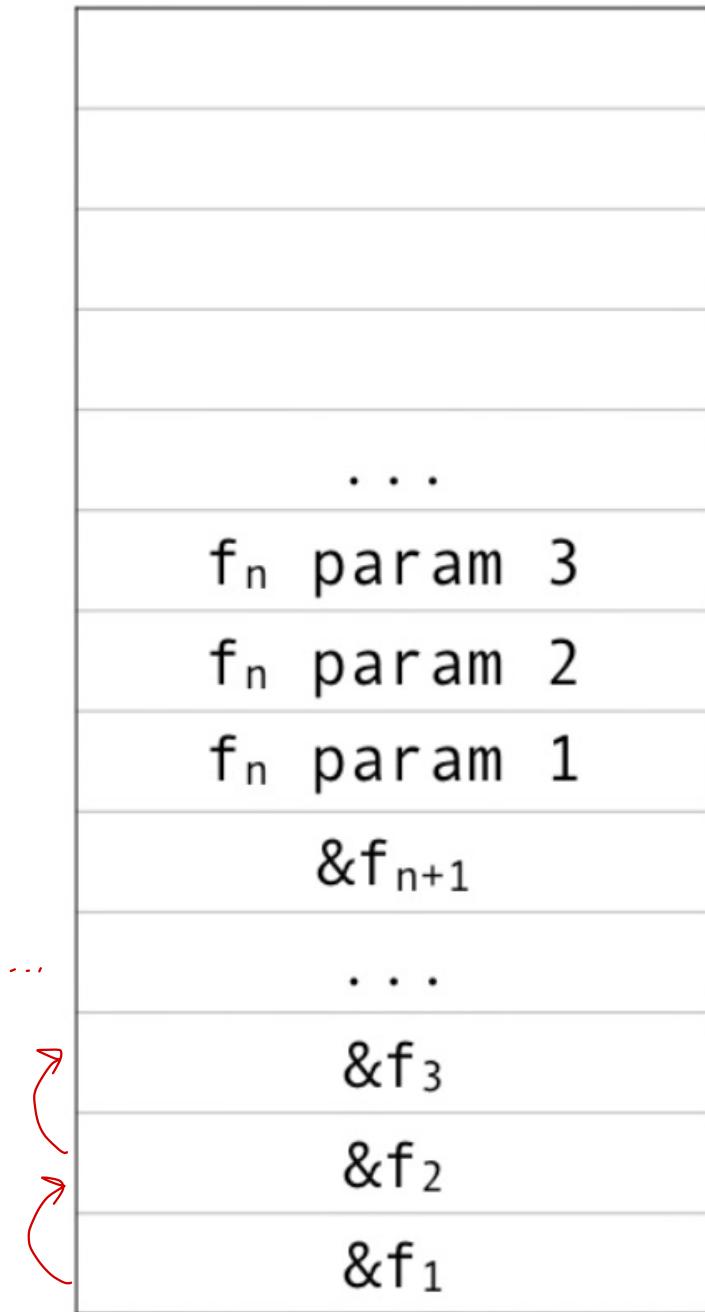
- retornar a primeira função a ser chamada
- parâmetros + endereço de retorno = segunda função a ser chamada

ao entrar em f_2 , a stack ainda tem o "lixo" da chamada anterior
nenhuma função (f_1 ou f_2) limpa isto



- Se uma função não recebe parâmetros, a função que a chama só guarda o endereço de retorno na stack e limpa-o
- Para executar N funções sem parâmetros, basta colocar os seus endereços na stack pela ordem inversa
- Depois de uma sequência de N funções que não recebem parâmetros, é possível chamar uma função f_{N+1} que recebe parâmetros e terminar com uma função f_{N+2} que não recebe parâmetros

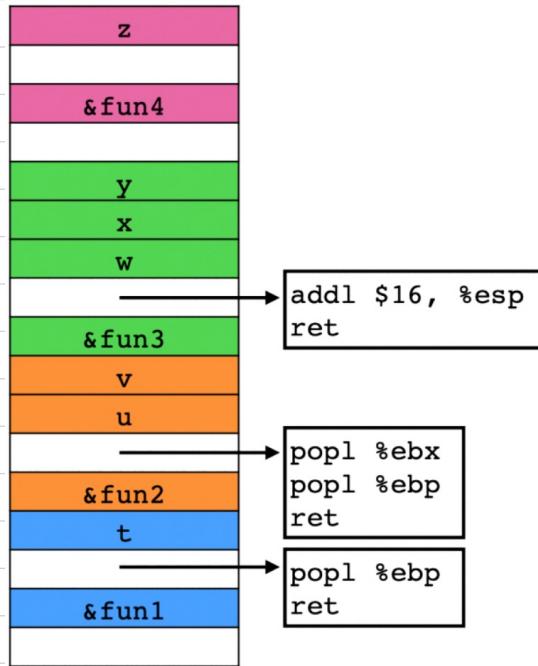
Stack



E para chamar funções com parâmetros? Existem versões alternativas das funções que limpam os seus próprios parâmetros da stack

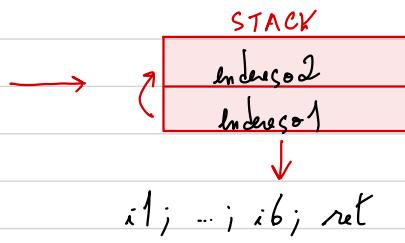
Como? Colocar na pilha, entre f1 e f2, um endereço que aponte para código que

1. limpe os bytes necessários da stack (ou move o stack pointer)
2. retorne para a próxima função (f2)



Caso Geral:

endereço 1: i1; i2; i3; ret
endereço 2: i4; i5; i6; ret



Gadget: sequência de código armazenado em memória que termina em rot
reutilizar e juntar para atacar

1. Procurar automaticamente no código alvo por um conjunto de gadgets
2. Usar gadgets para conseguir comportamento equivalente ao esperado

Operações aritméticas/lógicas (códigos lineares) não simples, MAS
Depende do código ao qual se tem acesso

if (c)
 f();
else
 g();



if (c)
 SP → &f
else
 SP → &g

1. Usando uma sequência de gadgets, inicializar um registo m para 0x00...00 se c é verdadeiro e 0xffffffff se c é falso
2. Aponta o stack pointer para a sequência que começa em &f
3. Calcular o deslocamento necessário para que o stack aponte para a aponta para &g
4. Computar $SP = m \& offset + SP$ usando um conjunto de gadgets

Como encontrar vulnerabilidades? erro de gestão de memória → crash

1. Testar o programa contra inputs arbitrários
2. Se crashar, investigar porquê
3. Se o erro de gestão de memória não confirma, é uma vulnerabilidade

Buffer Overflow

Se o atacante puder...

1. influenciar algum input:

- é vital garantir que todos os acessos a memória são corretos

2. escrever fora da região de memória designada:

- pode subvertеть o fluxo de controle do programa
- pode executar código malicioso arbitrário

3. ler fora da região de memória designada:

- pode ler informações sensíveis
- pode executar código (potencialmente malicioso) residente na memória

Stack/Heap Buffer Overflow: escrever por cima do endereço de retorno na stack/heap para executar código malicioso

Use After Free: aproveitar um programa que está a usar memória de fora do seu controlo para ler código malicioso dessa memória

Format Strings: controlar strings de formatação e/ou explorar número incorrecto de argumentos para ler/escrivver dados na stack/heap

Integer Overflows: abusar de erros de precisão aritmética para alocar memória erradamente

Defesa

Problema: Dados de fora do perímetro de segurança podem interferir com o controlo de fluxo do programa

Defesa em Profundidade:

1. Linguagem de Programação:

- a. usar linguagens "memory safe"
- b. verificação do programa
- c. compilação segura

2. Aplicação:

- a. detectar tentativas de ataque
- b. monitorizar a stack
- c. etiquetar memória

3. Sistema Operativo: prevenir a execução de código malicioso

4. Hardware:

- a. conjunto de instruções
- b. ambientes de execução confiável / controlada
- c. inicialização segura

Contamedidas do Sistema Operativo

Prevenção da Execução de Dados: a memória nunca pode simultaneamente ser escritível por um programa e conter código executável

Limitações:

1. Por si só não impede ataques de revitalização de código
2. Cria problemas para JIT

Aleatorização da Organização do Espaço de Endereçamento (ASLR): a localização do código e dos dados em memória é determinada de forma aleatória em cada execução — imprevisível para o atacante

1. ASRL em Tempo de Execução: aleatoriza endereços em cada execução
2. Kernel ASRL: não aleatoriza em tempo de inicialização
3. Aleatorização da Ligação ao Endereço do Kernel: o código do Kernel é aleatorizado

KASLR: antes de uma chamada de sistema, verifica que o ret foi precedido por um call → garante que os rets não precedidos por calls (ROP)

Limitações:

1. Não impede JOP
2. Perdas de precisão e desempenho

Contamedidas da Aplicação

Executáveis Independentes da Posição (PIE): o compilador gera código cuja execução é independente dos endereços absolutos → de cada vez que é executado, pode ser carregado num endereço diferente

Instrumentação do Código: montanhação em tempo de execução — o compilador gera código que verifica se a função esperada é realizada

Canários da Stack

Objetivo: prevenir injções de código através da detecção de alterações na stack

Ideia: introduzir canários gerados dinamicamente entre as variáveis locais e o frame pointer ~~e os valores de endereço de retorno~~ guardados na stack, verificando o canário antes de usar o endereço de retorno

Implementação: o compilador instrumentaliza entradas/saídas de funções

Canário Aleatório: no início da execução, o programa escolhe um array de bytes aleatórios para gerar canários que são colocados nos stack frames e verificados antes de retornar, abortando/terminando se o canário

Canário de Terminação: usar /0, /\n ou EOF para as funções de manipulação de strings paraem sempre nesses valores

- +: simples/fácil de ser em prática
- : perda de espaço e desempenho

Como derrotar os canários?

1. Ler o canário usando outras vulnerabilidades e com stack overflow colocar lá o valor de volta
2. Contornar o canário usando apontadores existentes: se o código tiver um apontador na stack e escrever para esse endereço o valor de outra variável local, um stack overflow pode mudar ambos e esconder em memória
3. Escrivê-lo por cima e ignorar o canário usando apontadores para funções: se um programa receber um apontador para função como argumento, mudá-lo pode permitir saltar para um endereço antes de a função retornar
4. Aativar o canário: se o canário for reutilizado muitas vezes, é suscetível a ataques de força bruta e fácil de descobrir

Etiquetagem de Memória: limita os apontadores para as regiões de memória apontadas (cores) e as tags não comparadas em alocar a memória, de modo a lançar exceções em overflow ou "use after free"

Integridade do Controlo de Fluxo:

1. Em tempo de compilação, determinar um grafo de controlo de fluxo (conjunto de fontes origem para cada alvo)
2. Em tempo de execução, verificar a consistência com essa informação

• Não é necessário proteger chamadas diretas, cujo alvo está no código

• É necessário proteger chamadas indiretas, cujos endereços não manipuladas dinamicamente pelo programa, incluindo apontadores para funções

BÁSICO

VS.

CLÁSSICO

VS.

AVANÇADO

Contramedidas da Linguagem de Programação

Segurança de um Programa: a execução do programa permanece dentro do comportamento esperado na semântica da linguagem

Segurança de Memória ⊂ Segurança do Programa

Segurança Espacial: a validade da memória depende da localização

Segurança Temporal: a validade da memória depende do tempo de vida do objeto/variável

Linguagens fortemente tipadas verificam segurança em tempo de compilação
Linguagens fracamente tipadas verificam segurança em tempo de execução

Análise de Manhais: o input do utilizador é manchado e as funções críticas não aceitam argumentos for manhais

Análise em Tempo Constante: proteção contra temporização de canais

Compilação Segura: provar que o processo de compilação não introduz erros de segurança

Contramedidas do Hardware

- Ambientes de Execução Controlada / Confiable
- UEFI Secure Boot

Segurança de Sistemas

1. Economia de Mecanismos: um sistema só deve ter apenas as funcionalidades necessárias e os mecanismos mais simples de segurança devem ser adotados — facilita implementação, usabilidade, validação, ...

2. Predefinição Imune a Falhas: a configuração de qualquer sistema deve, por defeito, obrigar a um nível conservativo de proteção

"Falha Fechada": numa falha, o sistema reverte para o seu fechado/ bloqueio conservativo

"Falha Aberta": numa falha, o sistema recupera para o seu estado permanente

3. Design Aberto: a arquitetura de segurança e o funcionamento interno de um sistema devem ser públicos — o argumento de segurança não deve ser baseado em esconder os mecanismos de segurança (os negócios são um número pequeno de parâmetros configuráveis do sistema)

4. Defesa em Profundidade: não se deve depositar toda a confiança num único mecanismo — blindar o perímetro e assumir que não existem ameaças internas

5. Mínimo Mecanismo Comum: favorecer o design transversal e limitar a partilha de mecanismos entre camadas — muitas vezes são adicionados mecanismos para as mesmas ameaças, enquanto outras ameaças podem continuar ignoradas

6. Privilégio Mínimo: cada utilizador/comportamento/programa deve ter apenas o mínimo de privilégios/permisões possíveis para desempenhar o seu papel — contradiz este princípio amplificando desencorajantemente o potencial impacto de uma falha de segurança

7. Separação de Privilégios: o uso de recursos e funcionalidades deve ser compartmentalizado — cada comportamento deve ser isolado dos outros, em domínios de confiança separados

8. Mediação Completa: para todos os recursos, um sistema deve definir uma política de segurança clara e validar todos os acessos aos recursos de acordo com essa política de segurança

Controlo de Acessos: família de mecanismos de segurança que instância os princípios 6, 7 e 8

Ator: entidade que realiza uma ação

Recurso: onde a ação é realizada

Operação: ação concreta realizada

Permissões: para (recurso, operação)

	R1	R2	R3
A1	r	rw	n
A2	rw	n	r
A3	r	r	r

Matriz de Controlo de Acessos: descreve todos os acessos possíveis

o quê?

Lista de Controlo de Acessos: para cada recurso, as permissões dos atores sobre eles — controlo de acessos dentro de fronteiras/limites

quem?

Capacidades: para cada ator, todas as operações que pode efectuar em cada recurso — controlo de acessos fora de fronteiras/limites

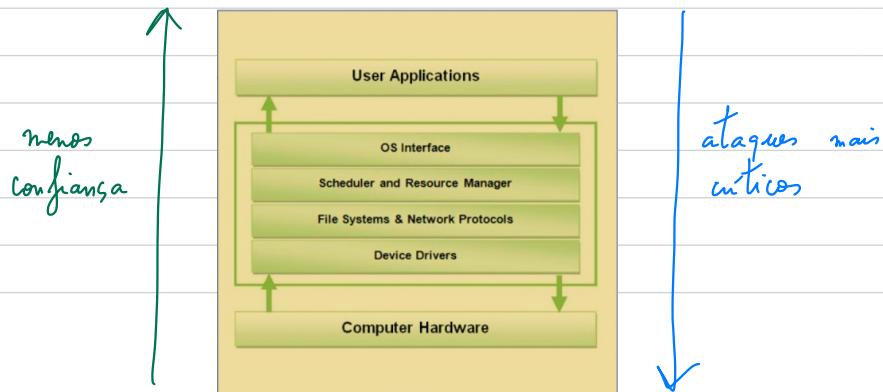
Controlo de Acesso Baseado em Papeis: relaciona os atores a perfis/papel e perfis/papel a permissões — quem gera depende da relação ator/permissões

Controlo de Acesso Baseado em Atributos: atores e recursos têm atributos associados e a matriz de ações descreve as permissões com base nos atributos — a decisão vai ao encontro da política de segurança

Segurança de Sistemas Operativos

Sistema Operativo: interface do computador entre utilizadores e hardware, que gere acessos/partilha de recursos por utilizadores/aplicações

- Usuários/programas podem controlar utilizadores, processos e/ou ficheiros maliciosos
- Os utilizadores e as aplicações não são potenciais ameaças
- Os recursos não os bens a serem protegidos
- Os SOs devem garantir isolamento virtual entre utilizadores, aplicações e processos, para melhor partilhar os recursos do sistema, através da imposição de mecanismos de medição de acessos



Processo Honesto: corre isoladamente

Processo Desonesto: tenta quebrar o isolamento e/ou abusar recursos

Kernel: parte do SO que realiza as operações mais críticas

Modo Kernel: o código pode realizar quase todas as operações do sistema

Modo do Utilizador: o código não tem acesso direto aos recursos do sistema

- Qualquer troca de informação entre os dois núcleos é parte da superfície de ataque
- A Kernel está protegida dos processos do modo do utilizador, tendo o seu próprio espaço de memória gerido de forma independente
- Qualquer processo em modo do utilizador deve ser capaz de aceder aos recursos do sistema usando chamadas ao sistema, que executam em modo Kernel
- Os pontos de entrada nas chamadas ao sistema não críticos: para causar danos críticos, um processo em modo do utilizador tem de invocar uma chamada ao sistema — a solução é implementar mecanismos de monitorização que controlam essas chamadas, simples e sempre presentes
- Para escalar privilégios, o código a correr em modo de utilizador deve preparar argumentos, identificar um ponto de entrada permitido para aceder ao modo Kernel e executar uma instrução especial que lhe dá controlo à Kernel — superfície de ataque bem definida com um número limitado de pontos de entrada

Processo: instância de um programa em execução — em memória e com um ID

- Cada processo deve executar num contexto em que tenha acesso a um conjunto de recursos, disponíveis independentemente dos outros processos
- A fronteira entre processos é um limite de confiança: os processos têm de ser confiados/isolados entre eles
- Para cada processo, o kernel mede os acessos aos recursos (tempo/efetua)
- Quando o utilizador lança uma aplicação, um processo do SO cria um novo processo
- O SO gera uma hierarquia de processos em que os descendentes herdam os privilégios do criador
- As permissões de um processo dependem de quem o criou (UID/GID)
- Chamadas ao sistema podem servir para comunicação entre processos

Daemon / Serviços: processos que não são diretamente visíveis para o utilizador, normalmente lançados antes dos processos que interagem com o utilizador, com mais privilégios e sobrevisam a sessões

• A confiança colocada nos processos é induktiva: o código na BIOS depois de uma instalação limpa do SO é confiável, o processo de inicialização coloca esse código em memória e para-lhe o controlo — nenhum novo processo pode mudar esse estado “confiável”

"Confiable": o sistema faz exatamente (e só) aquilo que foi especificado para fazer

• Vulnerabilidades que afetam a implementação do mecanismo de inicialização podem contaminar completamente o modelo de confiança do sistema (em que o processo de inicialização é a âncora)

Rota de Confiança do SO: UEFI Secure Boot — a BIOS só inicializa componentes de firmware assinados digitalmente pelo fornecedor da placa-mãe

Monitização: detecta fugas do modelo de confiança

Mediação: a instalação de código usando assinaturas digitais protege o sistema contra malware

Controle de dados: isolamento entre processos — um processo não pode aceder ao espaço de memória de outro processo — a confidencialidade, integridade e o controlo de fluxo do Kernel deve ser protegido dos processos do nível do utilizador

• Os acessos à memória são mediados por mecanismos de hardware e software geridos pelo Kernel

Memória Virtual: o espaço de memória lido do SO é muito maior do que o espaço físico disponível

• A tradução de endereços é usada para implementar mecanismos de memória virtual

Isolamento: cada processo acede a uma região de memória que não existe fisicamente e tem uma visão limitada dos recursos

- A memória virtual está dividida em páginas de tamanho fixo
- O SO armazena localização física de cada página (em uso)
- Parte da memória virtual do Kernel é mapeada diretamente para a memória virtual de cada processo, mas com permisões diferentes
- Parte do espaço de memória de um processo é gerida pelo Kernel, o processo não tem acesso a esse espaço, mas pode interagir com ele via chamadas ao sistema
 - Quando um processo invoca uma chamada ao sistema, não é necessário alterar o mapeamento de páginas do sistema porque a memória relevante do Kernel já está mapeada, existindo no mesmo espaço de endereçamento do processo
 - Quando se altera entre processos do utilizador, as tabelas de páginas são modificadas, mas não aquelas que pertencem à memória do Kernel

Defesa em Profundidade:

1. Nem o Kernel deve violar a regra W^X
2. Prevê-se que o Kernel escreva em regiões de memória do utilizador protegidas contra fugas de informação e execução de código malicioso
3. As tabelas de páginas do utilizador e do Kernel devem estar separadas

Sistema de Ficheiros

Actores: utilizadores/processos

Recursos: ficheiros/pastas

Ações: R/W/X

- Cada utilizador pertence a um grupo
- Cada recurso tem um proprietário e um grupo

ACL Estendido: estende RBAC com ACLs com grupos/utilizadores nomeados

Superusers: root/nodo \rightarrow uid = 0

- Tudo é permitido aos superusers/administradores
- O proprietário pode ser mudado pelo superuser
- O grupo pode ser mudado pelo proprietário ou pelo superuser

Controlo de Acesso Discrecional: o proprietário pode mudar permissões

Controlo de Acesso Mandatório: só o superuser pode mudar permissões

- Os utilizadores interagem com o sistema via processos em que cada processo tem um user id que determina as permissões do processo

Login: o processo de login descecala os privilégios (root \rightarrow user)

Retried: link associado a um ficheiro que liga o utilizador sobre o qual um processo é criado ao proprietário do executável \rightarrow escalação de privilégios

- Um processo normalmente executa com o UID do utilizador que o criou, podendo aceder aos seus recursos
- Os processos root não executados com o UID do proprietário
- Os processos da Kernel são lançados com UID=0, tendo acesso a todos os recursos (privilegio máximo)

User ID Efectivo (EUID): determina as permissões

User ID Real (RUID): utilizador que lança o processo

User ID Guardado (SUID): utilizador anterior, para transmitir privilégios

O root pode mudar os UIDs

Um utilizador pode mudar EUID para RUID ou SUID

Um processo pode reduzir temporariamente os seus privilégios com setuid que apenas altera EUID e preserva RUID e SUID !!

diminui os privilégios de forma permanente permite que o utilizador mude EUID de volta para RUID com setuid

Tudo é um ficheiro, por isso os mecanismos de controlo de acesso não sempre os mesmos

O mecanismo de controlo de acesso em *nix é uma implementação de Listas de Controlo de Acesso estendida com RBAC — quando um atacante se torna root, não há forma de diminuir os privilégios

Confinamento

Porquê? pode ser necessário executar código não confiável numa plataforma confiável

Objetivo: se o código se "comportar mal", neutralizá-lo

Solução: garantir que código potencialmente não pode afetar o resto do sistema malicioso

Aliás: confinamento físico ao nível do hardware

Processos: uma vista virtual da memória e dos recursos garante que as ações de P1 não afetam o contexto de P2 e vice-versa

Isolamento de Falhas de Software (SF): isolamento de processos que partilham espaço de endereçamento do utilizador — isolamento de memória

Interrupção de Chamadas de Sistema (SCI): medição de todas as chamadas de sistema — separação Kernel vs. utilizador

Monitor de Referência: componente central sempre invocado, omnipresente e simples que media completamente todos os acessos a pedidos

Máquinas Virtuais: uma vista virtual do hardware para cada SO garante que as ações no SO1 não afetam as ações no SO2 e vice-versa — cores SOs inteiros sobre o hardware ou outros SOs — hypervisor

Tipo 1: SO fino sobre hardware - "metal fino"

Tipo 2: camada de software sobre o SO - "alojado"

Caixa de areia: confinamento adicional dentro de uma aplicação

Contentores: partilham o Kernel do SO anfitrião e isolam componentes do utilizador

chroot: permite criar celas que só podem ser usadas pelo root, transformando o ambiente da shell visto pelos utilizadores/processos/aplicações

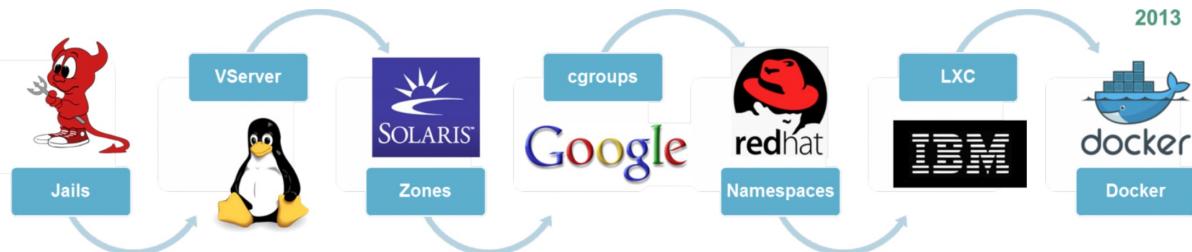
FreeBSD: restringe comunicação na rede, comunicação entre processos e privilégios root dentro da cela

Contentores Linux: root dentro do container é utilizador fora do container

Groups de Controlo: limitam/definem a quantidade/qualidade dos recursos

Namespaces: limitam/definem a disponibilidade dos recursos

Docker: foco em aplicações individuais em ambientes isolados e reproduzíveis
- o daemon que gera os containers corre como root



Intervenção de Chamadas de Sistema

- A superfície de ataque está limitada às chamadas de sistema
- Solução: monitorizar chamadas de sistema e bloquear as não autorizadas
- Implementação: dentro da Kernel, com um mecanismo recomp

ptace: permite a um processo monitor ligar-se a um processo alvo, sendo notificado quando ele invocar uma chamada de sistema e matando-o se a chamada não for autorizada

Vulnerabilidade: ataques TOCTOU - Time of Check Time of Use

recomp + bpf: permite políticas de controlo configuráveis às chamadas de sistema

• Um processo pode instalar múltiplos filtros BPF

• Dada uma vulnerabilidade no Kernel, uma mitigação possível é bloquear chamadas de sistema específicas e limitar acesos a certos ficheiros

extended BPF: correr de forma eficiente e segura programa do utilizador na Kernel, verificando a sua segurança formalmente

Isolamento de Falhas de Software

Alternativa: usar diferentes esquemas de endereçamento

Objectivo: limitar a região de memória disponível para uma aplicação

Operações Perigosas: load/store/jump - proteger com guardas

Máquinas Virtuais

- É improvável que o SO e hypervisor estejam vulneráveis ao mesmo tempo
- Um hypervisor é (geralmente) mais simples do que o SO

Canais Secundários: observar efeitos colaterais

Provedores modernos não incopram de garantir isolamento completo

Pode um programa saber que está a correr numa máquina virtual? SIM!

Para quê? evitar análise em ambientes virtuais
Como? detectar os recursos limitados do SO

"Phishing": correr o browser numa máquina virtual e explorar websites

Hypervisors

Compatibilidade: garantir que o software funciona fácil & diretamente

Desempenho: minimizar impacto da virtualização

NÃO Transparência: comportamento indistinguível quando virtualizado

Segurança Web

Protocolo HTTP: um recurso é identificado por um URL
esquema :// domínio: porta /caminho / ...

- É um protocolo "stateless"
- O cliente realiza pedidos individuais
- Servidor responde com estado + corpo (opcional)

GET: obtém um recurso - não deve mudar o estado do servidor

POST: cria um novo recurso - usado para quase tudo com efeitos colaterais

Cookie: de cada vez que um URL no mesmo servidor é acessado, o servidor envia um cookie e o browser guarda-o

- As páginas web são programadas
- Uma janela do browser pode ter conteúdo de diferentes origens

Frame: divisão rígida e visível

iFrame: objeto flutuante embalido num documento

Porquê? delegam uma área do ecrã para outra origem e aproveitam o isolamento entre frames que o browser fornece

DOM: interface orientada a objetos que representa a hierarquia de uma página completa - processo recursivo de carregamento da página

Código JavaScript pode ler ou mudar o estado de uma página

Modelo de Confiança: quem é que pode confiar em quem? NINGUÉM

- O servidor e o utilizador podem ser maliciosos
- O browser pode ter vulnerabilidades ou malware

Modelos de Ataque na Web:

1. Atacante no Browser: explora vulnerabilidades ou injeta malware no browser
2. Atacante Externo / na Rede: só controla o meio de comunicação
3. Atacante Interno / na Web: controla parte da aplicação web (cliente/revisor)
4. Atacante como Utilizador: tenta subverter o comportamento da página
5. Atacante como Página Web: controla (parte de) uma página web no cliente

Origem: o contexto que define o perímetro de segurança das páginas web

Recursos: elementos DOM das páginas

Isolamento: Same-Origin Policy (SOP)

Confidencialidade: dados de uma origem não podem ser acecidos por código de uma origem diferente

Integridade: dados de uma origem não podem ser alterados por código de uma origem diferente

• Cada frame tem uma origem: (esquema, domínio, porta)

SOP: código num frame só pode aceder a dados da mesma origem

• Os frames podem comunicar entre eles através de mensagens

- Os cookies só são enviados pelo browser ao servidor com a mesma origem da que os criou
- Uma página/frame pode fazer pedidos HTTP para fora da sua origem, podendo enviar dados (efeitos colaterais) ou incorporar recursos

HTML: pode criar frames com código de outra origem, mas não infeciona-lo nem modifica o conteúdo → XFS

JavaScript: pode obter scripts de outras origens, mas não infeciona-lo nem manipula-lo → XSS

↓
• Pode receber respostas de pedidos dinâmicos feitos à mesma origem

Imagens: os pixels não se mostram, mas o tamanho é revelado

• O servidor não pode avisar que os recursos públicos enviados para a página são inofensivos, porque podem revelar informação sensível sobre o estado do servidor

Cookies: a origem obrigatória é (domínio, caminho), o esquema e a porta são opcionais

• Um site pode definir um cookie para o seu domínio ou subdomínios

• Se uma página cria um frame com outra origem, os cookies associados ao frame não são acessíveis à página pai

• Os browsers enviam os cookies se:

- (1) o domínio do cookie fornece um sufixo do domínio da URL
- (2) o caminho do cookie fornece um sufixo do caminho da URL

• Os browsers modernos só enviam se o atributo Name Site=Nome definido pelo servidor na resposta que fornece o cookie

Name Site=Exact: só envia o cookie quando o pedido tem a mesma origem da página de alto nível

Name Site-Lax: distingue pedidos e abre exceções entre domínios para mandar cookies

• Cookies regulares só são enviados através de HTTPS

DOM: acesso ao DOM é permitido via JavaScript → pode ler a variável document.cookie (risco da versão) ← HTTP Only impede isso

CORS: permite relaxar quais pedidos entre origens para recursos não autorizados

Pedido Simple: não deve causar efeitos colaterais - o browser faz o pedido e verifica se o código de A pode aceder aos recursos de B

Pedido "Pre-Flight": pode causar efeitos colaterais - o browser faz um pedido sem efeitos colaterais e verifica se o código de A pode aceder aos recursos de B: se sim, faz o pedido real

Injeção

Injeção: input malicioso causa execução anómala de uma requisição escolhida pelo atacante → PROBLEMAS

1. Input não validado
2. Falta de separação entre input e aplicação

• Scripting do lado do servidor permite execução de código e/ou chamadas à shell (`eval()`, `system()`, `exec()`, ...)

CGI: o servidor corre programas bash para processar pedidos

• O servidor web tem um diretório próprio para armazenar scripts que correm em pedidos HTTP para gerar output enviado para o browser

• Os cabeçalhos HTTP que representam argumentos do script não traduzem em variáveis de ambiente

whitelisting > blacklisting

SQL Injection: scripting do lado do servidor gera dinamicamente comandos SQL, envia esses comandos para a base de dados de modo que executem os pedidos do cliente e podem incluir inputs do cliente

SQLi "cego": o atacante não vê o output da query, mas pode ver efeitos colaterais

Proteções: nunca criar consultas dinamicamente como strings

- Não mostram erros aos clientes, mas registra-los e criar alertas
- Usar instrumentos oferecidos pelas linguagens de programação

1. Sanitização de Inputs: garanti que só input seguro / sanitizado é aceito
- o problema real é falta de tipagem, mas escapar caracteres resolve

2. Querier Parametrizadas: a query é enviada diretamente para o servidor, separando o comando parametrizado e os parâmetros - o servidor define a query independentemente dos parâmetros

3. Bibliotecas ORM: uma API no paradigma orientado a objetos implementa operações de tradução de objetos para consultas

HTML Injection: atacantes podem injetar elementos no DOM da página

Cross-Site Scripting (XSS): o atacante pode convencer o cliente a carregar código malicioso dentro da página de um site legítimo — ignora SOP porque é o cliente a carregar o pedido e o atacante não deve ver a resposta

1. Refletido (utilizadores + servidores + atacantes): o atacante força o utilizador a fazer um pedido a um site legítimo e conteúdo malicioso é refletido para executar no cliente

2. Armazenado (utilizadores + servidores): o conteúdo malicioso é armazenado num recurso do site legítimo

3. DOM (utilizadores): código malicioso corre só do lado do cliente, nada é enviado para o servidor

Prevenções: sanitização e validação de inputs ("whitelisting" > "blacklisting")

1. Política de Segurança do Conteúdo (CSP): cada site pode permitir scripts de uma dada origem através da definição de um atributo — scripts de uma linha não são executados, só aqueles cuja origem é permitida

2. Integridade dos Subcursos (SRI): inclui a hash do código que se espera receber

3. Tipos Confidenciais: o browser rejeita atribuições de strings a elementos perigosos do DOM nem o tipo correto — o browser avisa que só funções confidenciais do DOM (que sanitizam inputs) geram strings com os tipos corretos



Broken Access Control

Insecure Direct Object Reference (IDOR): controle de acessos
inexistente — Adulteração de Parâmetros Web

Cross-Site Resource Forgery (CSRF):

1. O atacante engana o utilizador legítimo a clicar num link
2. O utilizador legítimo faz um pedido nem quer/sabem a um site legítimo
3. O site legítimo executa operações em nome do utilizador legítimo

O atacante pode desimular o pedido que causa um efeito colateral no servidor



SOP: não impede o atacante de ver a resposta nenhuma origem, mas não impede que o efeito colateral seja executado

XS-Leaks: o atacante pode fazer pedidos em nome do utilizador, não ver a resposta, mas usar canais alternativos

Prevenções:

1. Permite ao servidor saber que um pedido veio de uma página confiável
 - a. Tokens CSRF: token secreto (dinâmico) no form HTML que o atacante não pode dezer quando simula um pedido por POST
 - b. SameSite=Strict: proíbe o uso de cookies em pedidos "cross-site"
 - c. SameSite=Lax: permite seguir links "cross-site", mas não permite pedidos cross-site para carregar imagens ou frames

2. Permite aos servidores tomar decisões com base na origem do pedido
- a. Validação do "Referer": ao seguir links, validação explícita do servidor dos cabeçalhos Referer e Origin
 - b. Cabeçalhos Fetch Metadata: fornecidos pelo browser
 - i. Sec-Fetch-Init: que site originou o pedido?
 - ii. Sec-Fetch-Mode: que tipo de pedido?
 - iii. Sec-Fetch-User: o pedido foi feito explicitamente pelo utilizador?

3. Defesa em Profundidade: o cabeçalho Cross-Origin-Opener-Policy garante que o site que faz um pedido "cross-origin" numa nova página perde a referência a esse pedido → previne XSS-Leaks

Falhas de Identificação e Autenticação:

- Gestão da Sessão do Lado do Servidor: o servidor armazena e gera toda a informação da sessão, criando um token aleatório para servir de referência para o cookie do cliente
- Gestão da Sessão do Lado do Cliente: o cliente armazena e gera toda a informação da sessão → armazenamento dos cookies
- O servidor pode encriptar cookies para garantir confidencialidade, mas os cookies não devem incluir informação sensível
- O servidor pode e deve usar MACs para garantir integridade

Design Inseguro: abrange erros da lógica do negócio

Vulnerabilidade:

- validar input só do lado do cliente
- o servidor gera mensagens de erro com informação sensível
- o servidor pode ser corrompido
- o servidor pode guardar credenciais de forma insegura

Segurança Mal-Configurada:

1. Publicação Indesejável: aplicações web expõem dados sensíveis inadvertidamente

↓ CONTRA MEDIDAS

- garantir controlo de acesso apropriado
- lidar com exceções adequadamente
- desativar listagem de diretórios

2. Exibição de Ficheiros: todo o input é potencialmente malicioso

↓ CONTRA MEDIDAS

- atribuir nomes difíceis de administrar
- não permitir acesso direto via a página web
- fazer scan com um antivírus
- ter cuidado com formatos de compressão

3. Clickjacking: utilizador visita um site malicioso e o atacante faz-o ver num frame mas a interação é com outro frame

↓ PROTEÇÕES

• Caixas de areia HTML5

• Cabeçalho que impede uma página de ser redirecionada num frame noutra origem

• CSP que impede um site de ser "framed" por outro

4. Phishing: o atacante envia uma mensagem fraudulenta que engana o utilizador a visitar uma imitação de um site confiável, visitando um site malicioso e revelando dados sensíveis → 2FA < V2F



23/30

+1/1/60+

19,5/30

This is a multiple-choice test that will be corrected automatically. Please follow these rules:

- Mark your answer using only **blue** and **black** pen. No pencil or light-coloured pens.
- Check only inside each box and be generous on ink. Erased boxes will not be detected automatically.
- Only the boxes matter for the automatic correction. You may underline text or take notes on the sides.

The test is marked for 20 points. There are 30 questions in total, each with 4 options. Each question is worth 20/30 points. Students can check one or two choices per question. The scoring for each question is as follows:

- One checked correct answer (100%).
- One checked incorrect answer (-20%).
- No checked answers (0%).
- Two checked answers, one correct (50%).
- Two checked answers, none correct (-20%).
- More than two checked answers (-20%).

□ 0	□ 0	□ 0	□ 0	□ 0	□ 0	□ 0	□ 0	□ 0	□ 0
□ 1	□ 1	□ 1	□ 1	□ 1	□ 1	□ 1	□ 1	□ 1	□ 1
□ 2	□ 2	□ 2	□ 2	□ 2	□ 2	□ 2	□ 2	□ 2	□ 2
□ 3	□ 3	□ 3	□ 3	□ 3	□ 3	□ 3	□ 3	□ 3	□ 3
□ 4	□ 4	□ 4	□ 4	□ 4	□ 4	□ 4	□ 4	□ 4	□ 4
□ 5	□ 5	□ 5	□ 5	□ 5	□ 5	□ 5	□ 5	□ 5	□ 5
□ 6	□ 6	□ 6	□ 6	□ 6	□ 6	□ 6	□ 6	□ 6	□ 6
□ 7	□ 7	□ 7	□ 7	□ 7	□ 7	□ 7	□ 7	□ 7	□ 7
□ 8	□ 8	□ 8	□ 8	□ 8	□ 8	□ 8	□ 8	□ 8	□ 8
□ 9	□ 9	□ 9	□ 9	□ 9	□ 9	□ 9	□ 9	□ 9	□ 9

← code your 9-digit upYYYYXXXXX student number horizontally on the left, and replicate it below. Write also your first and last name below.

Student Number:

up									
----	--	--	--	--	--	--	--	--	--

First and Last Name:

.....

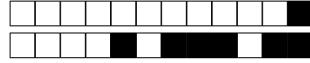
Group 1 Introduction (4 questions)

4/4 2,5/4

✓ **Question 1.1 ♣** Which of the following concepts is **not** associated with the **risk management** approach to security?

- Security proof. ✓
 Cost/benefit analysis.

- Mitigation. ✓
 Threat analysis. ✓



✓ **Question 1.2 ♣** Which of the following options is **not** usually categorised as an exploit?

- JavaScript performing heap spraying.
- ? Overflow in an integer computation. **VULNERABILIDADE**
- ? Text filled into edit box that contains JavaScript.
- Message that triggers an array access out of bounds.

✓ **Question 1.3 ♣** Which of the following is **not** a common reason for an attacker to compromise a server that answers to requests from a large number of clients/users?

- Geo-political and strategic motivation. ✓
- ? To make the machine part of a botnet. **CLIENTE**
- As part of a supply-chain attack. ✓
- ? To cause a data breach. ✓

✓ **Question 1.4 ♣** Pick the **incorrect** statement or indicate that all are correct.

- A trust model describes which assumptions we can rely on to build security. ✓
- ? All are correct.
- ? A security goal is stated in terms of avoiding loss of value to an asset.
- A security policy describes how security mechanisms are used to realize a security model. ✓

Group 2 Software Security (10 questions) **6,5/10** **8/10**

✗ **Question 2.1 ♣** In the buffer overflow **lab**, you created a malicious input that overwrites the return address of the vulnerable function with an address within your buffer where you stored **shellcode** to be executed. Which security technique had to be **disabled during compilation** to allow you to **overwrite** the return address of the vulnerable function?

- Address Space sation Prevention
- Layout Randomi
- Data Execution ✓
- Stack Canary ✓
- None of the other choices ✓

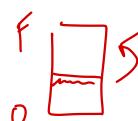


✓ **Question 2.2 ♣** Certain format strings can make a program crash (with very high probability). Which of the following **printf** commands is very likely to crash the program?

- int n=1; printf("%s%d",n); ✓
- int n=0; printf("%f%d",n); ✓
- int n=0; printf("%s%x",&n); ✓
- int n=1; printf("%n%f",&n); ✓

✗ → **Question 2.3 ♣** Recall the buffer overflow **lab** that you have performed, where a buffer in a local variable receives an untested input. For the exploitation strategy where the injected code (**shellcode**) is placed at the end of the input bytes, you should:

- In the place of the vulnerable function's return address, write another address pointing to a lower place in the stack.
- ? In the place of the vulnerable function's return address, write another address pointing to a higher place in the stack.
- In the place of the vulnerable function's return address, write the address of the buffer.
- None of the other options.





Question 2.4 ♣ *Information-flow analysis* encompasses a class of program analysis techniques beyond safety. The C-like program on the right illustrates an example of:

- ? Static taint analysis.
 Control-flow integrity.
 Dynamic taint analysis.
 Constant-time analysis.

```
int printf(untainted char *fmt, ...);  

tainted char *fgets(...);  

tainted char *name = fgets(..., network_fd);  

printf(name); // FAIL: tainted not untainted
```

verificada en tiempo de compilación



Question 2.5 ♣ Suppose you intend to use Return Oriented Programming to execute the instructions of functions f_1, f_2, f_3, f_4, f_5 (in some order) and you have placed their addresses in the stack as illustrated below. Which of the following functions could take parameters and the ROP strategy would still work without any changes to the part of the stack shown in the figure?

- Function f_3 .
 Function f_4 .
 All functions could take parameters.
 It is not possible to pass any parameters.

PARAMS	
High address	& f_3
	& f_1
	& f_5
	& f_4
Low address	& f_2



Question 2.6 ♣ Recall what you have studied about the classical configuration of the stack region managed by a function f that is called by a function g . Indicate which of the following options is correct for the positioning of the following data pieces: ① local variables of f , ② frame pointer of g , ③ parameters passed by g , ④ return address back to g .

P→R→F P→V

- ③ \Rightarrow A, ② \Rightarrow B, ① \Rightarrow C, ④ \Rightarrow D
 ① \Rightarrow A, ③ \Rightarrow B, ② \Rightarrow C, ④ \Rightarrow D
 ④ \Rightarrow A, ③ \Rightarrow B, ① \Rightarrow C, ② \Rightarrow D
 ③ \Rightarrow A, ④ \Rightarrow B, ② \Rightarrow C, ① \Rightarrow D

High address 3, 4, 2, 1
Low address

<i>g</i>	PARAMS	A	1
<i>g</i>	ret	B	4
<i>g</i>	FP	C	2
<i>g</i>	LOCAL	D	1

g → f

g
f



Question 2.7 ♣ Consider the code below, which introduced a critical vulnerability in openSSH. An attacker that controls the value of `nresp` can cause a memory management problem because:

- It can lead to not allocation enough memory.
 It can lead to allocating too much memory.
 It can force the ~~if~~ statement not to execute.
 None of the other choices.

```
nresp = packet_get_int();      OVERFLOW
if (nresp > 0) {
    response = xmalloc(nresp*sizeof(char*));
    for (i = 0; i < nresp; i++)
        response[i] = packet_get_string(NULL); }
```



Question 2.8 ♣ A canary built from string terminating bytes offers better security than a random canary because:

- They can be generated much more efficiently.
 It is harder to guess than a random canary.
- It makes it more difficult to overwrite the stack.
 Random bytes can be all 0.



Question 2.9 ♣ In a heap overflow attack, the crucial information that is usually overwritten is:

- Return addresses. STACK
 Pointers to functions.
- None of the other choices.
 Frame pointers. STACK



X → Question 2.10 ♣

a localização do código em memória é aleatória em cada execução

One can bypass Address Space Layout Randomization (ASLR) protections by:

- Trial and error.
- All of the other choices.
- Jumping to system code using Return Oriented Programming.
- Extracting addresses using other vulnerabilities.

Group 3 Systems Security (10 questions)

7,5/10

5,5/10

✓ Question 3.1 ♣ Remember the environment variable and setuid program lab. The Linux file system associates a owner, a group, and 12 permission bits with each entry. For a file with the 12th bit (the Set-UID bit) active, it can be executed:

- Only by users whose set (owner, group, others) in the permission bits, has the executable bit active.
- Only by the owner or by a user belonging to the group.
- Only by the owner.

✓ Question 3.2 ♣ Many modern operating systems require system support for an hardware component called a Trusted Platform Module (TPM). Which is the main rationale for that requirement?

- To protect the user login process.
- To guarantee that the user only installs official software.
- To store user's cookies when browsing the web.
- To prevent malicious bootloaders from compromising the operating system's boot process.

✓ Question 3.3 ♣ Which is not a systems security principle that we have studied in the classes?

- Separation of privilege
- Economy of mechanism
- ? Compromise recording
- Closed design
OPEN

✓ → Question 3.4 ♣ Under Linux, Docker crucially relies on seccomp-bpf to confine containers. Select the incorrect choice or mark that all choices are correct.

? All choices are correct.

- Using the default Docker seccomp-bpf filters, an attacker that acquires root in the container does not directly acquire root in the host OS.
- ? If an attacker has access to certain system calls inside a container, it can exploit Docker to acquire root in the host OS.
- Users can manually configure seccomp-bpf filters to block various system calls inside the container.

X → Question 3.5 ♣ The Android operating system is a particular distribution built on top of Linux. Which is not an additional domain-specific restriction that Android implements?

- There is no root user, to prevent applications from escalating privileges.
- It implements a form of Mandatory Access Control, so that no application can change its permissions.
- It isolates different applications by registering each application with a different UNIX user.
- It enforces Manifest Permissions per application, to restrict its system capabilities.



Question 3.6 ♣ Virtual machines are commonly used as a security mechanism. Which of the following sentences is **not** true?

- A disadvantage of virtual machines is that malware may exploit the virtualisation layer to remain undetected. ✓
- An advantage of virtual machines is that acquiring root in the virtual machine does not grant root in a host operating system. ✓
- A disadvantage of virtual machines is that malware may detect that it is being virtualised. ✓
- An advantage of virtual machines is that software may be transparently executed as in a non-virtualised operating system.



Question 3.7 ♣ Which fundamental mechanism does a UNIX operating system have in place to enforce isolation?

- System call interposition, by virtualising the address space of different user processes, monitoring all virtual address translations.
- System call interposition, by preventing different user processes to communicate via system calls.
- ? Software fault isolation, by virtualising the address space of different user processes, monitoring all virtual address translations.
- ? Software fault isolation, by ensuring that the control-flow of user processes never accesses invalid virtual memory regions.



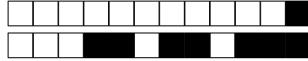
Question 3.8 ♣ For efficiency reasons, the address space of the Linux kernel is not completely independent from that of user processes due to *kernel mapping*. Which of the following is correct?

- ? Part of the kernel memory space is mapped into the memory space of each user process, to reduce the number of cache misses.
- Since the discovery of speculative execution attacks such as Meltdown and Spectre, Linux no longer uses *kernel mapping*, as it violates the principle of *separation of privilege*.
- Part of each user process's memory is mapped into the kernel memory space, to reduce the size of the address translation tables.
- ? Part of the kernel memory space is mapped into the memory space of each user process, to speed up system calls.



→ **Question 3.9 ♣** In the environment variable and setuid program lab we have experimented with environment variables and setuid programs. Which of the following sentences is **not** true?

- If the shell detects that it is being run under a setuid process, it may drop its privilege. ✓
- The system function allows calling a shell function within a program with the program's environment.
- When a process forks a child process, it passes on its environment, excluding some critical variables if the fork is a system call. ✓
- When the shell forks a child process, it passes on its environment, excluding some critical variables if it has a different effective user id. ✓



✓ Question 3.10 ♣ The UNIX filesystem can be seen as an instance of:

- Access control lists, since only the owner of each file (or root) may read, write or execute it.
- Role-based access control, since each file has an access control list for three fixed roles (owner, group, others) and the association between users and groups may be modified independently.
- Capability lists, since each file enumerates the permissions for all the users who may read, write or execute it.
- Attribute-based access control, since users may change their group membership without the need to change file permissions.

Group 4 Web Security (6 questions)

5/6

3,5/6

✓ Question 4.1 ♣ Cookies may be used for various purposes. Name the incorrect one below.

- Tracking user activity. ✓
- Personalising user experience. ✓
- Encrypting user communication.
- Managing user sessions. ✓

✗ Question 4.2 ♣ Which sentence best characterises a Cross-Site Request Forgery (CSRF) attack?
“When a malicious origin can send requests to a server in another origin ...” :

- ... and measure side-channels such as response time. XCS-Yeahs
- ... where the user is already logged in.
- ... and read its response.
- ... and trigger a side-effect on the server.

→ ✓ Question 4.3 ♣ According to the Same-Origin Policy (SOP), which of the following is allowed?

- JavaScript code in a page from origin A can inspect the HTML code of a frame from origin B.
- HTML code in a page from origin A can send a POST request to a page from origin B.
- HTML code in a page from origin A can send and read the response of a GET request to a page from origin B.
- JavaScript code in a frame from origin A can exchange data with a frame from origin B.

✓ Question 4.4 ♣ Consider the following SQL query, written in some server-side library, that is vulnerable to SQL injection. Which of the following statements is true?

```
uName = getRequestString("username");
uPass = getRequestString("userpassword");
sql = 'SELECT * FROM Users WHERE Name=' + uName + ' AND Pass=' + uPass + '';
```

- A SQL injection attack would not be possible if the clauses for the Name and Pass fields did not enclose strings with double quotes ("").
- A SQL injection attack may be able to delete the USERS table.
- A SQL injection attack that bypasses authentication is only possible because the clause for the Name field appears before the clause for the Pass field.
- A SQL injection attack will only be able to read existing data from the USERS table.



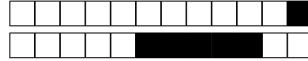
✓ **Question 4.5 ♣** Which assignment can be seen as a valid analogy in the table below?

- ? ①=Pages,②=Cookies,③=HTTP,④=Frames
- ? ①=Pages,②=HTML,③=JavaScript,④=Popups
 - ①=DOM,②=Cookies,③=iFrames,④=Fetch
 - ①=Frames,②=DOM,③=Images,④=Sub-frames

Systems Security	Web Security
Processes	①
Files	②
Sockets	③
Sub-processes	④

✓ → **Question 4.6 ♣** A classical protection against Cross-Site Scripting (XSS) attacks is for the site to adopt a Content Security Policy (CSP). Which of the following statements is true?

- DOM-based XSS attacks cannot be prevented with a CSP because they occur on the server-side when processing a user request.
- Preventing reflected XSS attacks requires both CSP and SRI (Subresource Integrity).
- XSS attacks using inline scripts are blocked by a default CSP.
- Stored XSS attacks cannot be prevented with a CSP because the malicious payloads are already stored in the server.



21/30

+1/1/60+

19/30

Computer Security Foundations
03/02/2023

Duration: 1H

LEIC
Resit Exam - Part 1

This is a multiple-choice test that will be corrected automatically. Please follow these rules:

- No pencil or light-coloured pens. Mark your answer using only **blue** and **black** pen.
- Erased boxes are not detected automatically. Check only inside boxes and be generous on ink.
- You may underline text or take notes on the sides. Only boxes matter for automatic correction.

The test is marked for 20 points and contains 30 questions, each with 4 options. **Only one of those options is accepted as correct, which could be an option indicating all options are correct.**

You can check one or two choices per question. The scoring for each question is as follows:

- One checked correct answer (100%).
- One checked incorrect answer (-20%).
- No checked answers (0%).
- Two checked answers, one correct (50%).
- Two checked answers, none correct (-20%).
- More than two checked answers (-20%).

<input type="checkbox"/> 0									
<input type="checkbox"/> 1									
<input type="checkbox"/> 2									
<input type="checkbox"/> 3									
<input type="checkbox"/> 4									
<input type="checkbox"/> 5									
<input type="checkbox"/> 6									
<input type="checkbox"/> 7									
<input type="checkbox"/> 8									
<input type="checkbox"/> 9									

← code your 9-digit upYYYYXXXXX student number horizontally on the left, and replicate it below. Write also your first and last name below.

Student Number:

up

First and Last Name:

.....

Group 1 Introduction (4 questions)

1,5/4

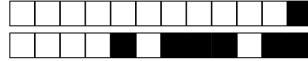
2/4

✗ Question 1.1 ♣ Which of the following options is **not** usually categorised as a **threat?**

- JavaScript performing heap spraying. **EXPLOIT** Company that competes in the same area.
 Activist that disagrees with company policy. Natural catastrophe.

✗ Question 1.2 ♣ Which of the following concepts is **not** associated with the **binary model of security?**

- Formal definition. ✓ Security proof. ✓ Resilience. ✓ Cryptography.



- ✓ **Question 1.3 ♣** Identify a typical reason for an attacker to compromise a user's machine.

- Playing games on the user's machine.
 As part of a supply chain attack.
 Stealing user's credentials.

- ✗ **Question 1.4 ♣** The rows and columns of a risk assessment matrix correspond to:

- Level of severity/- Cost of mitigation
? Probability of occurrence/Level of severity
 Level of severity/ Accuracy of information
? Probability of occurrence/Cost of mitigation

Group 2 Software Security (10 questions)

6/10 6,5/10

- ✓ **Question 2.1 ♣** During the buffer overflow lab, you explored a buffer overflow vulnerability using shellcode. Which technique could be used to make it impossible run shellcode in the stack?

- Address Space Randomisation
 Stack Canary
 Data Execution Prevention
 None of the other choices



- ✗ **Question 2.2 ♣** In the buffer overflow lab, you created a malicious input that overwrites the return address of the vulnerable function with the address of the shellcode. Which protection had to be disabled during compilation?

- ? Stack Canary
 Data Execution Prevention
 None of the other choices
 Address Space Randomisation

- ✓ **Question 2.3 ♣** Certain format strings can make a program crash (with very high probability). Which of the following printf commands is very likely to crash the program?

- int n=1; printf("%n%f",&n);
 int n=1; printf("%s%d",n);
 int n=0; printf("%f%d",n);
 int n=0; printf("%s%x",&n);

- ✗ **Question 2.4 ♣** Remember the program on the right with a stack overflow vulnerability from the buffer overflow lab. Assume that strlen(str)=400 and that we have a shellcode with 20 bytes. Consider a 32-bit architecture where valid addresses are aligned with multiples of 4. Which of the following input strings would not yield a successful attack?

```
int bof(char *str)
{
    char buffer[100];
    strcpy(buffer,str);
    return 1;
}
```

- OVERRIDE RETURN ADDRESS**
 Place the shellcode at str[380], write &buffer+300 to all positions from 0 to 100 and fill all remaining positions with NOPs.
 Place the shellcode at str[380], write &buffer+300 to all positions from 0 to 200
75
and fill all remaining positions with NOPs.
 Place the shellcode at str[0] and write &buffer to all positions from 20 to 400.
 Place the shellcode at str[380] and write &buffer+380 to all positions from 0 to 380.
75



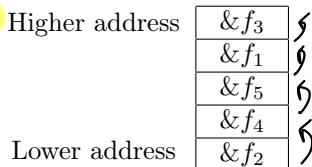
Question 2.5 ♣ Just-In-Time (JIT) compilation is problematic in terms of security mainly because:

- W^X*
- It violates the W^X principle.
 - It may remove protection mechanisms.
 - It inhibits static analysis of the program.
 - It may introduce vulnerabilities.

Question 2.6 ♣ In a use-after-free attack, the crucial information that is usually overwritten is:

- Pointers to functions.
- Exception handlers.
- Virtual function tables.
- All of the other choices.

Question 2.7 ♣ Suppose you intend to use Return Oriented Programming (ROP) to execute the instructions of functions f_1, f_2, f_3, f_4, f_5 (in some order) and you have placed their addresses in stack as illustrated below. In which order are they executed?



- 2,4,5,1,3.
- ROP does not use the stack.
- 3,1,5,4,2.
- ROP can return to at most one function.

Question 2.8 ♣ The code `int n; printf("%d",n);` has a common vulnerability catalogued in the CERT Secure C Coding Standard. It is a violation of: *UNSPECIFIED BEHAVIOR*

- Constant-time security
- Control-flow integrity
- Program safety
- Memory safety

Question 2.9 ♣ In heap spraying, the attacker does the following:

- Uses a heap buffer overflow to fill the entire memory with shellcode.
- Creates multiple copies of the shellcode in the heap.
- None of the other choices.
- Uses a programming technique known as spraying to control the heap.

Question 2.10 ♣ A shadow stack is a security mechanism that:

- Inverts the stack to make it harder to modify.
- Uses two copies of the stack.
- Cryptographically denies stack modification.
- Hides the stack from the attacker.

Group 3 Systems Security (10 questions)

7,5/10

7,5/10

Question 3.1 ♣ Remember the program `int main() { system("ls"); return 0; }` from the environment variable and `setuid` program lab. Which of the following statements is **not** true?

- None of the other choices is false.
- Another executable called `ls` in the working directory may be executed instead of the system `/bin/ls` command.
- If the program runs with root `setuid`, it may list all files in the current directory.



✓ **Question 3.2 ♣** There are several confinement approaches in operating systems. Which is true?

- A container allows a guest OS to run independently of a host OS. ✗
- A sandbox allows applications to run independently of an OS. ✗
- A container allows an OS to run indepen-

dently from the underlying hardware through the use of virtual machines. ✗

- An hypervisor allows an OS to run independently from the underlying hardware through the use of virtual machines.

✓ **Question 3.3 ♣** Which is **not** a systems security principle that we have studied in the classes?

- Minimum privilege ✓
- Open design ✓
- Complete mediation ✓
- Economy of privilege

✗ **Question 3.4 ♣** In a UNIX operating system, a process can change its permissions at runtime, by controlling the Effective User ID (EUID). The **setuid** primitive allows:

- The root user to change the permissions of the process into a non-privileged user, without the possibility to regain root privileges. ✗
- The root user to change the permissions of the process into a non-privileged user, without the possibility to regain root privileges. ✗

SETEUID

- A user to change the permissions of the process into another user, without the possibility to gain root privileges. ✗

✓ **Question 3.5 ♣** In Linux, Secure Computing Mode, **seccomp** for short, is an important reference monitor for system calls. Which of the following sentences is **not** true? *SÓ PERMITE RETS E FICHEIROS JÁ ABERTOS* ✓

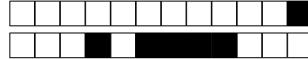
- ? All options are true.
- The seccomp reference monitor kills any process that violates its permissions.
- A program in seccomp cannot open new files.
- A program in seccomp cannot fork child processes.

✓ **Question 3.6 ♣** For efficiency reasons, the address space of the kernel is not completely independent from that of user processes due to *kernel mapping*. Which is **not** an implemented countermeasure by UNIX systems to prevent a user program from accessing kernel memory?

- ? The kernel memory management abides by the W^X restrictions. ✓
- Kernel Address Space Layout Randomisation (KASLR) loads the kernel to a random address each time the system boots. ✓
- Accessing kernel memory requires special kernel privileges, divided into distinct kernel capabilities.
- The kernel memory is placed below the stack program memory, to make sure that a stack overflow cannot reach the kernel.

✗ **Question 3.7 ♣** Which is **not** an important assumption made by a modern operation system to guarantee that user files are kept secure?

- The hardware has not been compromised.
- The user never installs malicious applications with administrator privileges.
- No attacker has physical access to memory/disk when the system reboots or hibernates. ✓
- The bootloader has not been compromised. ✓



✓ **Question 3.8 ♣** Which of the following is an advantage of role-based access control?

- Unlike access control lists, it allows defining the permissions that each actor has on every resource.
- It is a more general form than access control lists, capabilities or attribute-based access control.
- It allows decoupling access control policies into a relation between actors and roles and a relation between resources and roles.

✓ **Question 3.9 ♣** In the environment variable and setuid program lab we have experimented with environment variables and setuid programs. Which of the following sentences is not true?

- If the shell detects that it is being run under a setuid process, it may drop its privilege.
- The system function allows calling a shell function within a program with the program's environment.
- When a process forks a child process, it passes on its environment, excluding some critical variables if the fork is a system call.
- When the shell forks a child process, it passes on its environment, excluding some critical variables if the fork has a different effective user id.

✓ **Question 3.10 ♣** The UNIX filesystem permissions are an important element to ensure security of critical administrative operations. Which of the following sentences is true?

- The group of a file can only be changed by the root user.
- Only the owner of a file or the root user can change its permissions.
- The owner of a file can only be changed by the owner itself.
- Not even the root user can change the permissions of files he does not own.

6/6 3/6

Group 4 Web Security (6 questions)

✓ **Question 4.1 ♣** According to the Same-Origin Policy (SOP), how can a web developer prevent a cookie from being sent by a HTML link from origin A that makes a request to origin B?

- Defining cookies as being HTTPOnly.
- Defining SameSite=Lax.
- Defining SameSite=Strict.
- Using only secure (HTTPSS) cookies.

✓ **Question 4.2 ♣** Which is not an example of a Cross-Site Request Forgery (CSRF) attack?

- A hidden JavaScript script in a malicious site tries to log in into the users' web router in his home network, using default credentials, to install spyware.
- The source of an image in a malicious site sends a login request in a legitimate site with the attacker's credentials.
- The submit button of a HTML form in a malicious site sends a money wiring request to the legitimate site of a bank where the user is logged in.
- The submit button of a HTML form in a malicious site sends, alongside a request to a legitimate site, a malicious JavaScript whose execution steals the user's cookies.



✓ **Question 4.3 ♣** According the Same-Origin Policy (SOP), which of the following is allowed?

- JavaScript code in frame with origin A can manipulate the HTML code of page with origin B.
- JavaScript code in page with origin A can send a GET request to page with origin B.
- JavaScript code in page with origin A can send a POST request to frame with origin B.
- HTML code in frame with origin A can inspect the HTML code of sub-frame with origin B.

✓ **Question 4.4 ♣** Consider the following SQL query, written in some server-side library, that is vulnerable to SQL injection. How could we prevent a SQL injection attack on this query?

```
uName = getRequestString("username");
uPass = getRequestString("userpassword");
sql = 'SELECT_*_FROM_Users_WHERE_Name=' + uName + '"_AND_Pass=' + uPass + '';
```

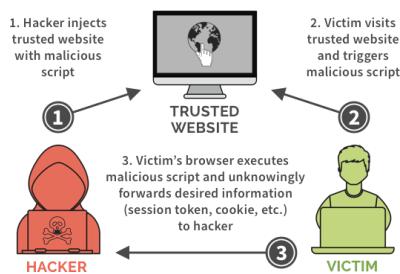
- Using Trusted Types, a feature of SQL engines that enforces input sanitisation.
- Not using string concatenation, and filling the username and userpassword arguments using prepared statements.
- The client-side application generates a fixed SQL query (for username and userpassword), sent securely via HTTPS.
- Changing the order of the clauses for the Name and Pass fields within the WHERE clause.

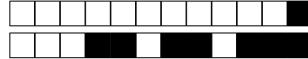
Question 4.5 ♣ It is **not** recommended for HTTP GET requests to perform server side-effects because, by default:

- They are allowed across any origin.
- An attacker can always see its response.
- An attacker can steal cookies.
- They are not recorded in the server logs.

✓ **Question 4.6 ♣** The figure illustrates a typical Cross-Site Scripting (XSS) attack. It is an example of:

- A DOM-based XSS attack.
- A reflected XSS attack.
- A stored XSS attack.
- A subtype called a Cross-Frame Scripting (XFS) attack.





Question 4.3 ♣ According the Same-Origin Policy (SOP), which of the following is allowed?

- JavaScript code in frame with origin A can manipulate the HTML code of page with origin B.
- JavaScript code in page with origin A can send a GET request to page with origin B.
- JavaScript code in page with origin A can send a POST request to frame with origin B.
- HTML code in frame with origin A can inspect the HTML code of sub-frame with origin B.

Question 4.4 ♣ Consider the following SQL query, written in some server-side library, that is vulnerable to SQL injection. How could we prevent a SQL injection attack on this query?

```
uName = getRequestString("username");
uPass = getRequestString("userpassword");
sql = 'SELECT_*_FROM_Users_WHERE_Name=' + uName + '"_AND_Pass=' + uPass + '';
```

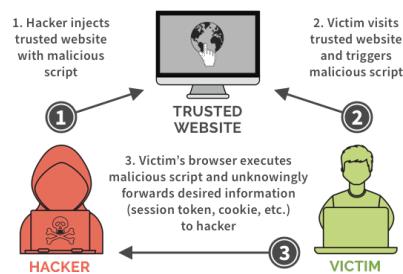
- Using Trusted Types, a feature of SQL engines that enforces input sanitisation.
- Not using string concatenation, and filling the `username` and `userpassword` arguments using prepared statements.
- The client-side application generates a fixed SQL query (for `username` and `userpassword`), sent securely via HTTPS.
- Changing the order of the clauses for the Name and Pass fields within the WHERE clause.

Question 4.5 ♣ It is **not** recommended for HTTP GET requests to perform server side-effects because, by default:

- They are allowed across any origin.
- An attacker can always see its response.
- An attacker can steal cookies.
- They are not recorded in the server logs.

Question 4.6 ♣ The figure illustrates a typical Cross-Site Scripting (XSS) attack. It is an example of:

- A DOM-based XSS attack.
- A reflected XSS attack.
- A stored XSS attack.
- A subtype called a Cross-Frame Scripting (XFS) attack.



Encriptação Simétrica

A encriptação garante confidencialidade

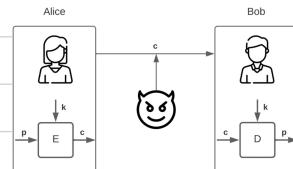
Encriptação: transforma texto limpo em texto cifrado usando uma chave

$$c \leftarrow E(K, p)$$

a encriptação é normalmente aleatória

$$p \leftarrow D(K, c)$$

a desencriptação é determinística



Encriptação Simétrica: é usada a mesma chave dos dois lados

Língua de Círculo: chave fixa ($\gg 3$)

insegura porque o espaço de chave é muito pequeno

Língua de Substituição: escolher "shifts" diferentes para letras diferentes
inseguras por causa de ataques de frequência de letras

Máquina de Hebern: a chave é o disco - tabela de substituição
↳ "Enigma": a chave é a configuração inicial de "rotors" (que rodam)

"One-Time Pad": escolher uma string aleatória de bits, $K \leftarrow \{0, 1\}^n$
Encriptar: $c \leftarrow m \text{ XOR } K$
Desencriptar: $m \leftarrow c \text{ XOR } K$

É perfeitamente seguro (desde que as chaves só sejam usadas uma vez), mas as chaves têm de ter o mesmo tamanho das mensagens...

Princípio de Kerckhoff: todos os detalhes de uma operação criptográfica são públicos, o único segredo é a chave

NEVER USE YOUR OWN CRYPTO

Cifra de Bloco: definida por dois algoritmos determinísticos

Encriptação $E(K, \text{f}_1)$: recebe uma chave $K \in \{0, 1\}^\lambda$ e um bloco de texto $\text{f} \in \{0, 1\}^B$, retornando um bloco cifrado $c \in \{0, 1\}^B$

Desencriptação $D(K, c)$: recebe uma chave $K \in \{0, 1\}^\lambda$ e um bloco cifrado $c \in \{0, 1\}^B$, retornando um bloco de texto $\text{f} \in \{0, 1\}^B$

Uma cifra de blocos é invertível - K define uma permutação

Ex: AES; 3DES = $E(K_1, D(K_2, E(K_3, \text{f})))$, $\lambda = 56$

ADVANCED ENCRYPTION STANDARD

$B = 128$; $\lambda \in \{128, 192, 256\}$

- Mantém um estado interno de 128 bits: array 4×4 de 16 bits
- O estado é transformado usando uma rede de substituição-permutação
- As cifras de blocos não são primitivas a partir das quais se constroem esquemas de encriptação e se prova a segurança da encriptação

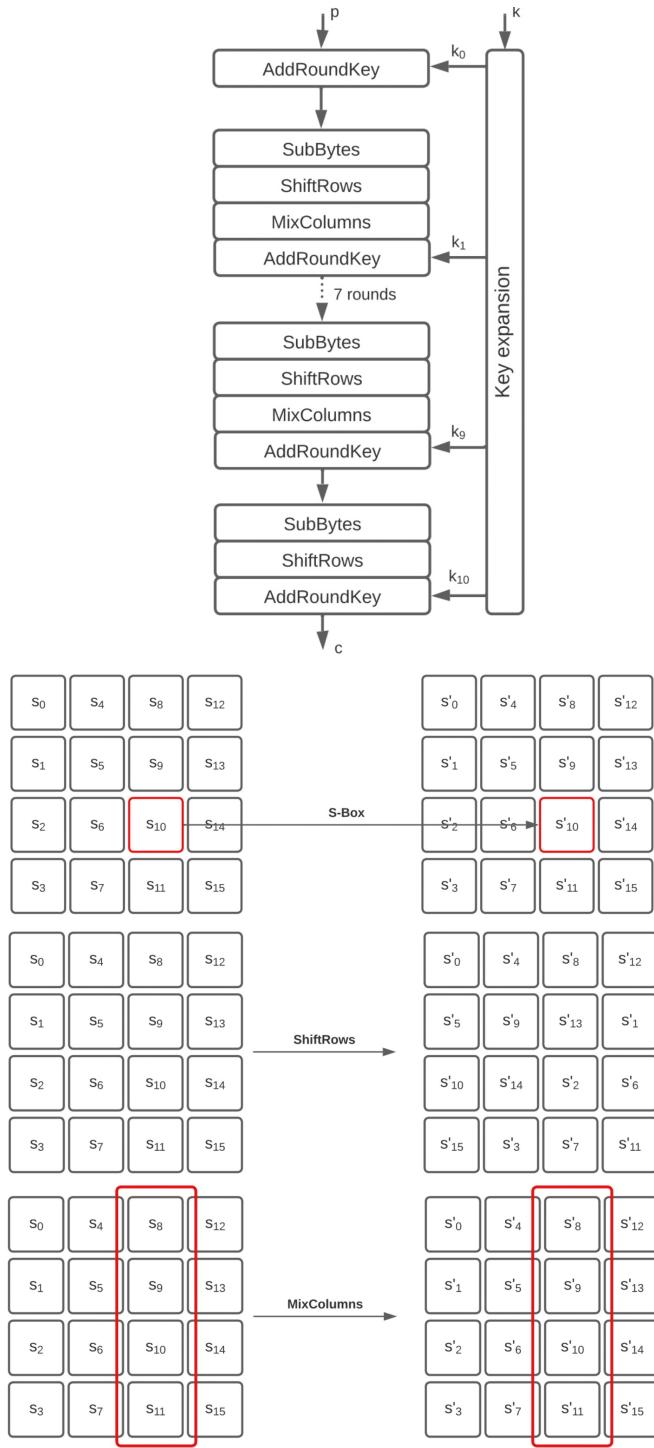
Electronic-Code-Book (ECB): divide a mensagem em blocos de texto $\text{f}_1 \dots \text{f}_n$ e encripta independentemente cada bloco $c_i \leftarrow E(K, \text{f}_i)$

Cipher Block Chaining (CBC): tem um vetor de inicialização (IV) e os blocos dependem uns dos outros

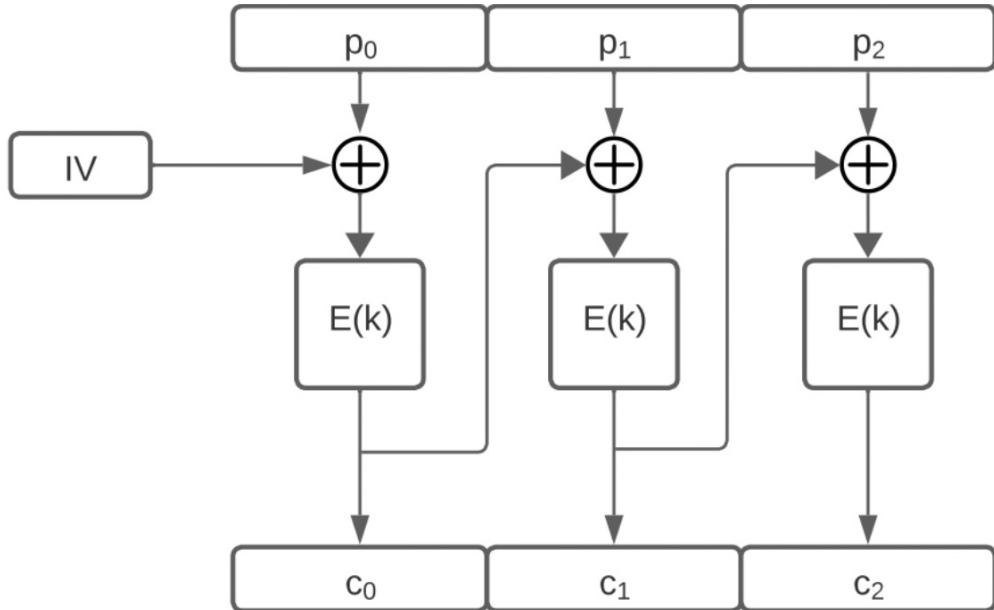
Counter Block Mode (CTR): muitas vezes usado com base num número N de uso único, mas não necessariamente aleatório, tornando a encriptação "stateful"

- o fluxo da chave pode ser pré-processado
- qualquer parte dos dados pode ser accedita eficientemente
- a encriptação/desencriptação pode ser paralelizada

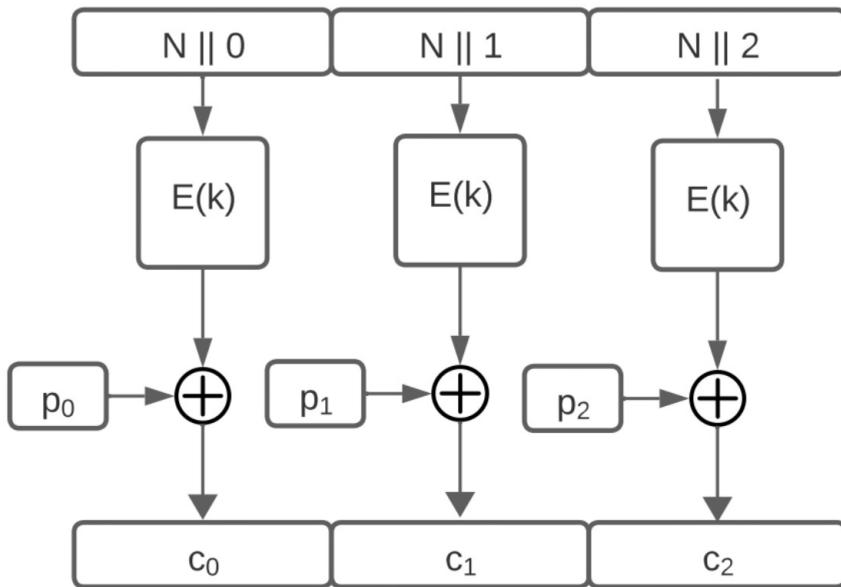
AES



CBC



CTR



Padding: é sempre adicionado, normalmente, sendo $K > |M|$ o múltiplo seguinte de B (em bytes), adicionando $K - |M|$ bytes com valor $K - |M|$, de modo que o último byte revele quanto "padding" foi adicionado

Criptografia Simétrica: gerada uniformemente de forma aleatória, derivada de algo usando uma Função de Derivação de Chave

Criptografia Assimétrica: o algoritmo de geração de chaves gera um par de chaves, o detentor da chave privada gera ambas e publica a pública

• Idealmente, a chave deve ser armazenada num hardware seguro externo, podendo ser "wrapped" antes de armazenada para encriptar com outra chave

• Processos aleatorizados não descritos usando distribuições de aleatoriedade

• Um processo V mostra da distribuição uniforme se: $\forall \lambda \in S$, $P(\lambda = \lambda) : \lambda \in V) = 1/|S|$

• Testes estatísticos não são suficientes/relevantes para Segurança

Linuc: PRG acessível em /dev/random

Variante: /dev/random bloqueia se não houver "entropia suficiente"

• Um tamanho comum para chaves é 128 bits

Segurança de N bits: o melhor ataque para quebrar o esquema requer $2^{N/2}$ passos

• Chaves de N bits não podem ter mais do que segurança de N bits

• Chaves de L bits podem ter até a segurança de N bits, sendo $N < L$

Regra 2¹²⁸: designs para os quais o melhor ataque requer 2^{128} passos

MACs e Encriptação Autenticada

Função de Hash: $m \rightarrow \text{HASH} \xrightarrow{h}$ tamanho constante e fixo ($256/512$)
... Criptograficamente segura: output imprevisível e difícil encontrar colisões
COMO?

1. Computar valores como num ataque de força-bruta
2. Armazená-los numa estrutura de dados indexada pelo valor da imagem
3. Cada nova imagem é procurada na estrutura de dados
4. Repetir até encontrar uma colisão

Quantas operações? Como, depois de N valores verificamos $N \times (N-1)/2$ pares, verificam 2^N pares precisa de $\approx \sqrt{2^N}$ valores — a complexidade geral é a de encontrar a pré-imagem de uma hash com $N/2$ bits de output (metade do alcance)

Construção de Merkle-Damgård: baseia-se numa função de compressão de $m+n$ para n bits para construir uma função de hash cujo comprimento do output é n para comprimentos de input arbitrários

- H_0 é o valor inicial: constante e público
- M é partido em blocos de tamanho m : M_1, M_2, \dots

Ex:	MD4	SHA-1
MD5	$512 \rightarrow 128$	$512 \rightarrow 160$
SHA-256	$512 \rightarrow 256$	$512 \rightarrow 256$

SHA-512: $1024 \rightarrow 512$

Construção em Espanha: usa uma permutação de l bits para construir uma função de hash para comprimentos arbitrários de input e output

AES-XTS: fixada um valor inicial h_0 , acumular gradualmente a mensagem (dividida em blocos de tamanho r) no estado, fazendo o XOR dos blocos

ESPELHAR: processo dual de construir o output iterativamente, bloco a bloco

Ex: SHA-3

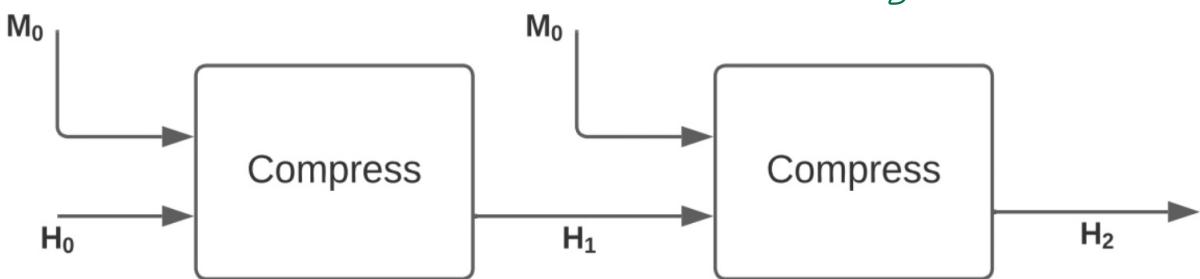
Função de Hash



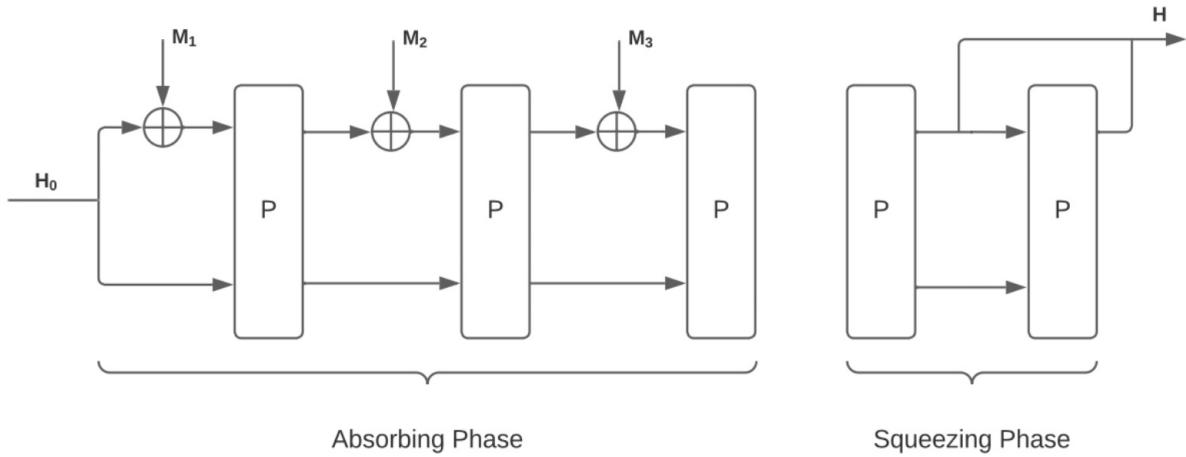
Any length

Short, fixed length:
usually 256 or 512 bits

Constução de Merkle - Damgård



Constução em Esponja



MD5: output de 128 bits — quebrada, demora segundos a encontrar colisões

Secure Hash Function (SHA): padão internacional

SHA-1: output de 160 bits e blocos de 512 bits — insegura

SHACAL → função de compressão que realiza cinco adições de 32 bits

SHA-2... → o identificador de três dígitos define o comprimento do output

→ parâmetros aumentados e cifras de blocos internas melhoradas

SHA-3: blocos de 1152, 1088, 832 ou 576 bits para outputs de, respectivamente, 224, 256, 384 ou 512 bits — funções SHAKE

Message Authentication Codes (MACs): autenticação simétrica $t \leftarrow \text{MAC}(k, m)$

- t garante que m foi produzido por alguém que conhece k , implicando que a mensagem m não foi alterada desde a sua criação — assinatura digital
- São uma forma de configurar/concordar em estabelecer uma chave comum k : o emissor computa $t \leftarrow \text{MAC}(k, m)$ e envia (m, t) , enquanto o receptor recebe (m, t) e recomputa $t' \leftarrow \text{MAC}(k, m)$ — se $t \neq t'$, a mensagem é rejeitada porque não é possível computar t sem k
- MACs não fornecem confidencialidade — garantem integridade
- MACs são otogonais à encriptação
- MACs são construídos a partir de funções de hash e cifras de blocos

Construção Simples: preficar a chave — $\text{MAC}(k, m) = H(K \parallel M)$

MD: dado (m, t) , o atacante retorna $H(K \parallel M \parallel \text{pad} \parallel M')$, que pode ser computado só a partir de t' e m' — ataque de extensão do comprimento

Construção HMAC: quando instanciada com a construção MD, HMAC é simplesmente $H((K \oplus opad) \parallel H((K \oplus ipad) \parallel m))$

\downarrow \downarrow

restrições para alinhar os tamanhos do bloco

Construção CMAC: usa o modo de operação CBC e fixa o IV para todos os blocos zero, tornando a cifra do último bloco como tag

- As chaves k_1 e k_2 não derivadas de k :

- $| \leftarrow E(k_1, 0)$
- $k_1 = (| \ll 1) \oplus (0x\ 00..0087 \times \text{LSB}(|))$
- $k_2 = (k_1 \ll 1) \oplus (0x\ 00..0087 \times \text{LSB}(k_1))$

Funções Universais de Hash: não tendo de ser resistentes a colisões, parametrizadas por uma chave $UH(k, m)$, garantem que, para duas mensagens fixas $m_0 \neq m_1$, então: $P[UH(k, m_0) = UH(k, m_1)] \leq \epsilon$

- É possível usar uma função de hash universal como MAC desde que só se autentique uma mensagem

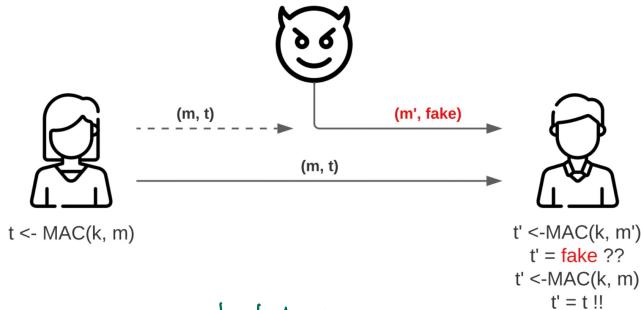
Construção de Wegman-Carter: usando uma PRF (como AES) para fortalecer a UH , converte-se UH num MAC totalmente seguro

Encryptar Valor da Hash Universal: $UH(k_1, m) \oplus \text{PRF}(k_2, n)$

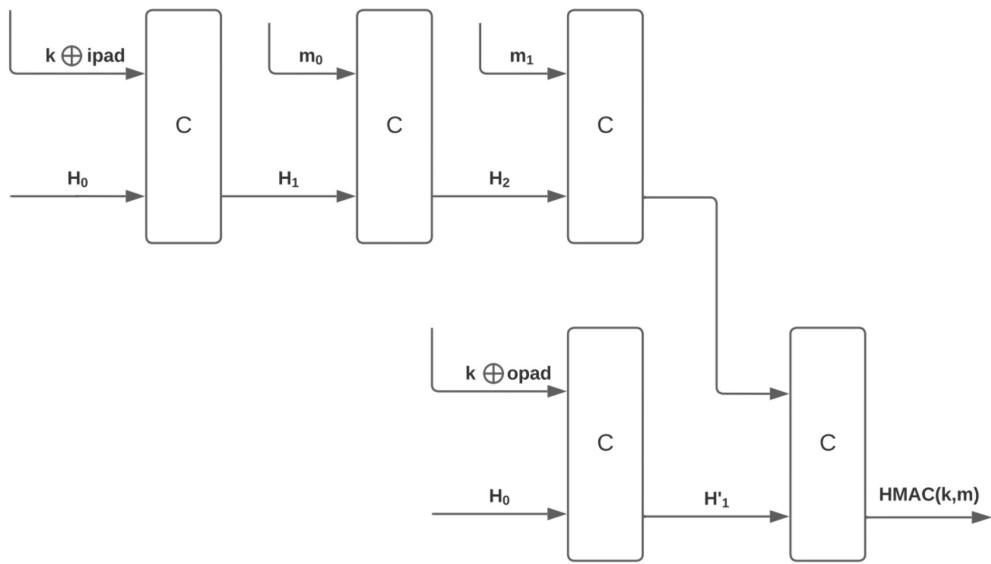
- A chave MAC completa é (k_1, k_2)
- N é um número (público) de uso único

Ex: $\text{Poly 1305}((k_1, k_2), m) = (m_1 K + \dots + m_n K^n \pmod{t}) + \text{AES}(k_2, n)$

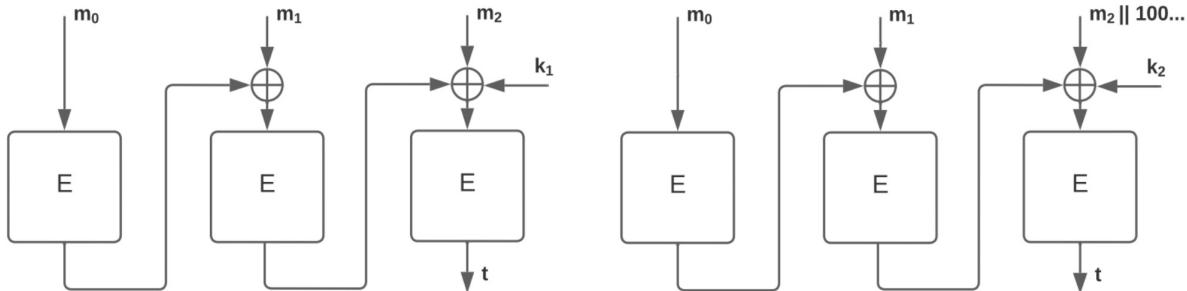
MAC



HMAC



CMAC



Encriptação: confidencialidade + MACs: autenticidade = Encriptação Autenticada

1. Encrypt-and-MAC: encriptação e MAC da mensagem
→ AE com $k = (k_1, k_2)$ feita com processamento paralelo

$$\cdot c \leftarrow E(k_1, m) \quad \cdot t \leftarrow \text{MAC}(k_2, m) \quad \text{Output: } (c, t)$$

- ⋮ c potencialmente maligno desencriptado antes da autenticação
- ⋮ MACs não foram desenhados para garantir confidencialidade
- ⋮ a contusão pode ser segura para alguns MACs
- ⋮ SSH: $\text{MAC}(k_2, m||n)$, sendo n o número de sequência

2. MAC-then-Encrypt: encriptar a mensagem e a sua autenticação
→ AE com $k = (k_1, k_2)$ feita sequencialmente

$$\cdot t \leftarrow \text{MAC}(k_1, m) \quad \cdot c \leftarrow E(k_2, m||t) \quad \text{Output: } c$$

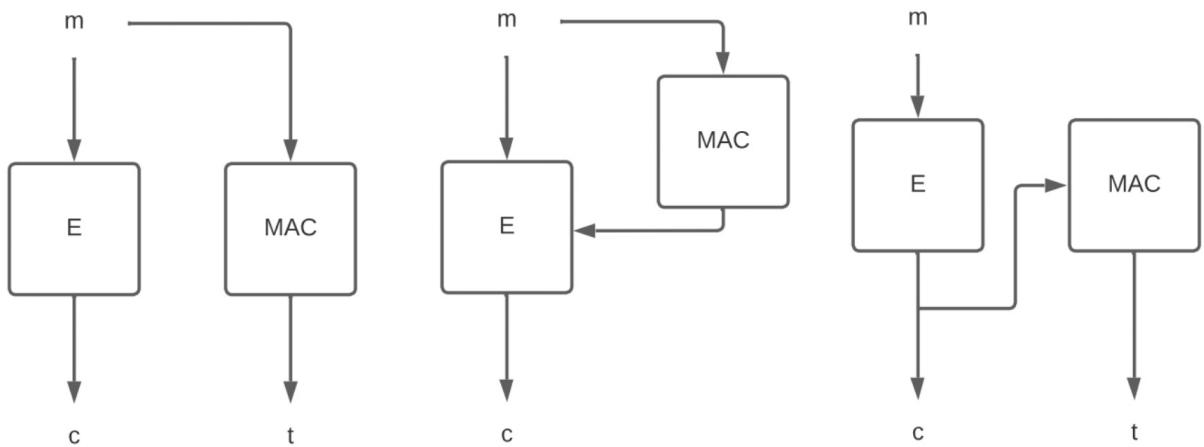
- ⋮ c potencialmente maligno desencriptado antes da autenticação
- ⋮ A desencriptação falhou por causa do padding ou por causa do MAC?
- ⋮ TLS < 1.3

3. Encrypt-then-MAC: autenticar a encriptação da mensagem
→ AE com $k = (k_1, k_2)$ feita sequencialmente

$$\cdot c \leftarrow E(k_1, m) \quad \cdot t \leftarrow \text{MAC}(k_2, c) \quad \text{Output: } c$$

- ⋮ O texto cifrado não é desencriptado até ser autenticado
- ⋮ A verificação MAC é rápida — útil contra DoS

Encriptação Autenticada



Encrypt-and-MAC

MAC-then-encrypt

Encrypt-then-MAC

Como otimizar "encryt-then-MAC"? encrutar/desencrutar blocos em paralelo

Modo Galois-Counter (GCM): CTR mais camada de autenticação

- Chaves de 128 bits
- Nonce de 96 bits
- CTR desde 1

• A função universal de hash é GHASH (hk, c_1, \dots, c_n), sendo AES usada como PRF para a hash do valor no input $n \parallel 0$, isto é:

- $hk \leftarrow \text{AES}(k, 0)$
- $P(x) = (1 \oplus D(A), 1 \oplus D(c), |A| \parallel |C|)$ em hk
- $P(x) = x \times (x \times (x \times (\dots + a_2) + a_1) + a_0)$

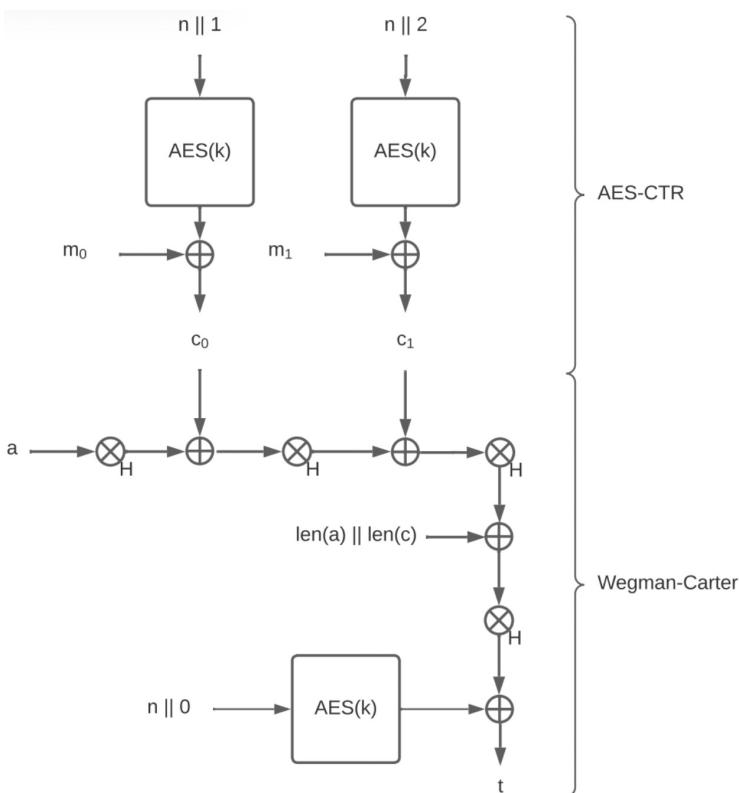
• A camada de encriptação herda o paralelismo do modo CTR
• A camada de autenticação bloqueia se a Nô é conhecido no fim
• Se a for conhecido desde o início, os blocos cifrados são computados e autenticados "on the fly", sendo a autenticação do bloco anterior acumulada enquanto o bloco atual está a ser encriptado

Offset Codebook (OCB): o offset depende da chave e do Nonce, sendo incrementado para cada novo bloco - o último offset é computado a partir do último bloco de texto processado

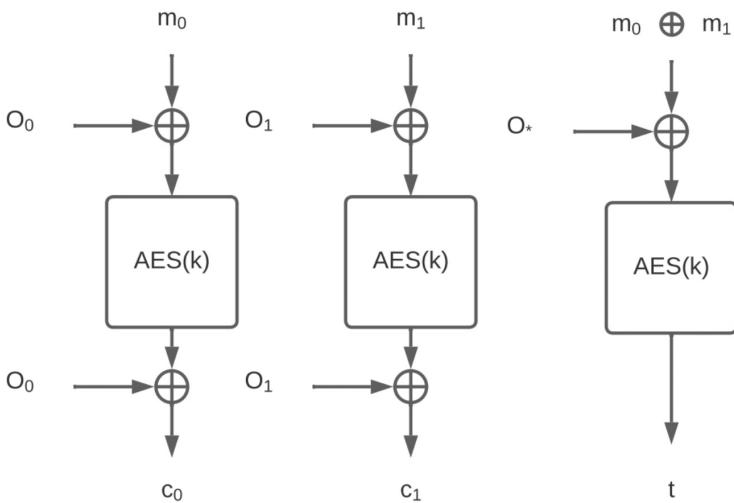
• Tag $T = E(K, S \oplus O_*) \oplus E(K, a_0 \oplus O_0) \oplus \dots \oplus E(K, a_n \oplus O_n)$,
Nó $S = m_0 \oplus \dots \oplus m_n$ e os valores de offset para AD diferentes dos usados para encriptar m_0, \dots, m_n

• Se reutilizam o Nonce, o atacante pode identificar blocos duplicados, pelo que Nonces distintos podem garantir a autenticidade do OCB: um atacante pode combinar blocos de mensagens autenticadas com OCB para criar outra mensagem autenticada, mas, ao contrário de GCM, não consegue extrair a chave

CGM



OCB



Modo IV sintético (SIV): refusa AES-GCM contra a reutilização do Nonce, através de uma composição genérica de encriptação baseada em Nonce e uma PRF

- $t \leftarrow \text{PRF}(k_1, a \parallel f \parallel n)$
- $c \leftarrow E(k_2, n=t, 1)$

Output: (c, t)

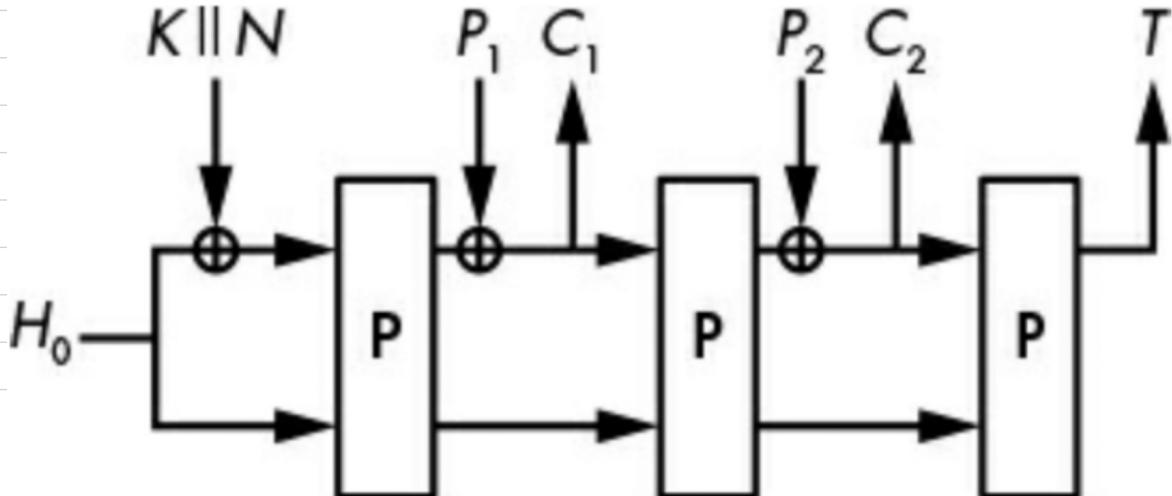
- Uma tag é usada enquanto o Nonce for provavelmente fresco
- Este esquema não é "streamable"

AE baseada em Permutações:

- rápido
- "streamable"

- P é uma permutação fixa
- h_0 é um valor público
- o último bloco deve ser padded

- Inversibilidade não afetada
- O primeiro bloco de texto e blocos seguintes com prefixos comuns não afetados
- Os textos continuam confidenciais depois da divergência



Criptografia de Chave Pública

Criptografia Simétrica: N participantes $\rightarrow N(N-1)/2$ chaves

Solução Centralizada: Centro de Distribuição de Chaves $\rightarrow N$ chaves

↓
Guarda uma chave de longo-prazo partilhada com cada participante

Chave de Longo-Prazo: altos requisitos de segurança para armazenar

Chave de Transação: efímera; dados limitados se corrompida

Problemas:

1. Chaves simétricas de longo-prazo partilhadas
 - a. sistemas abertos assíncronos \rightarrow encriptação de chave pública
 - b. sistemas abertos síncronos \rightarrow acordos de chave + assinaturas digitais
2. Não-Repúdio: qualquer pessoa com a mesma chave pré-partilhada pode
SISTEMAS ABERTOS produzir uma mensagem autenticada assinaturas digitais

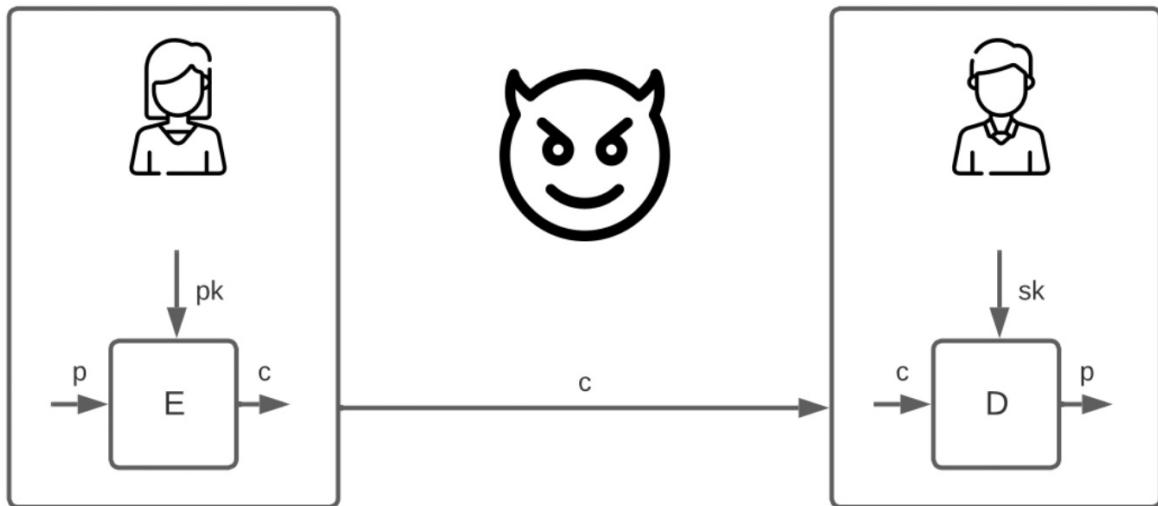
Criptografia de Chave Pública: qualquer pessoa com PK pode encriptar, mas só alguém com SK pode desencriptar

- $c \leftarrow E(pk, m)$ - encriptação com a chave pública
- $m \leftarrow D(sk, c)$ - desencriptação com a chave secreta

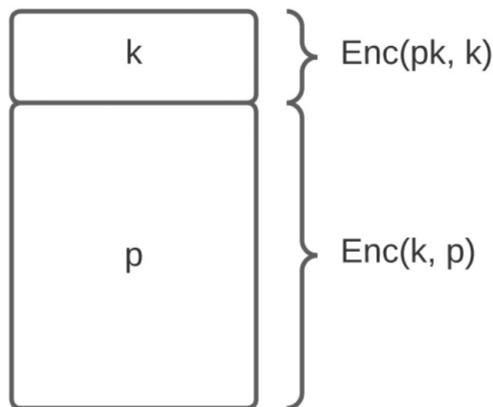
Mecanismos de Encapsulação de Chaves:

1. O emissor gera uma chave simétrica de sessão K para encriptar m
2. O emissor obtém pk e usa-a para encriptar K
3. O receptor recebe dois textos cifrados, correspondentes a (1) e (2)

Criptografia de Chave Pública



Mecanismos de Encapsulação de Chaves



Como construir criptografia de chave pública?

1. Função "trapzoo" de Número único: dados "trapzoo" n e s , é possível encontrar m tal que $F(m) = s$
2. Permutação "trapzoo" de Número único: dados "trapzoo" n e s , é possível computar $F^{-1}(s)$

↓
RSA:

1. Escolha um expoente público (normalmente $O(n \cdot 1000)$)
2. Escolher dois primos grandes p e q (> 2048)
3. Calcular $n \leftarrow p \times q$; $\varphi \leftarrow (p-1)(q-1)$
4. Computar d tal que $d \times e \bmod \varphi = 1$

$$\begin{aligned} \cdot pk &\leftarrow (e, n) & F(pk, x) &= x^e \bmod n \\ \cdot sk &\leftarrow (d, n) & F^{-1}(sk, y) &= y^d \bmod n \\ && n^{ed} \bmod n &= x \end{aligned}$$

• Não pode ser usada só RSA para assinatura porque não é aleatorizada!

Assinaturas: asseguram a autoria da mensagem/documento e que ele não é alterado depois da assinatura NÃO falsificação; reutilização; repúdio

Autenticação Assimétrica de Mensagens: só alguém com SK pode assinar, mas qualquer pessoa com PK pode verificar

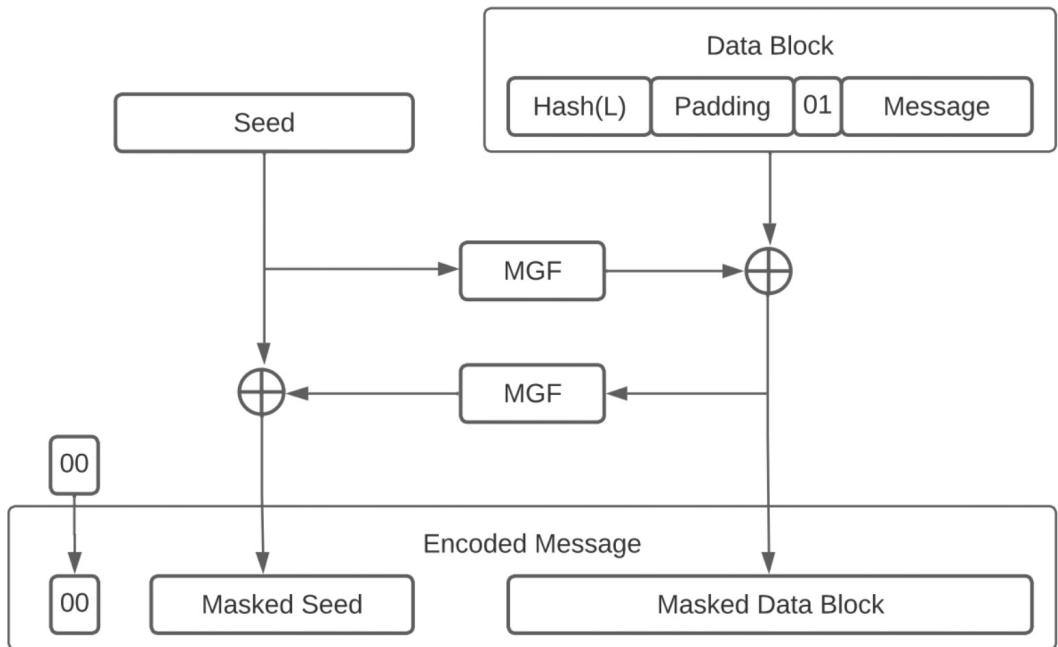
- $s \leftarrow S(sk, t_f)$ — assinatura com a chave privada
- $T/L \leftarrow V(pk, t_f, s)$ — verificação com a chave pública

• Assinaturas garantem autenticidade e integridade \approx MACs assimétricos

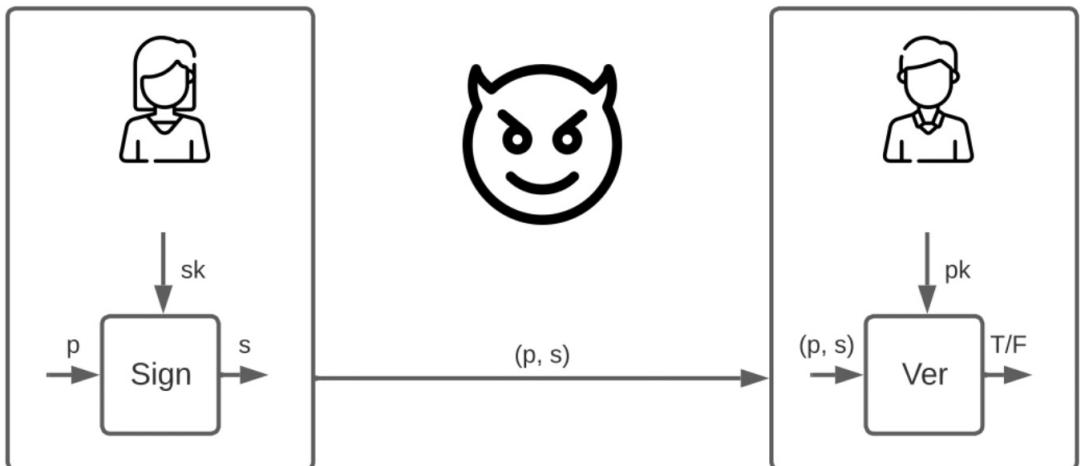
↳ não requerem uma chave secreta pré-partilhada

↳ garantem não-repúdio porque quem assina não pode negar a computação da assinatura

O AEP



Autenticação Assimétrica de Mensagens



Como construir assinaturas digitais?

1. $(\gamma_k, \alpha_k) \leftarrow \text{Gen}()$
2. $\sigma \leftarrow F^{-1}(\gamma_k, H(m))$ — expoente secreto sobre a hash da mensagem
3. $T \leftarrow F(\gamma_k, H(m)) = \sigma$ — inserir a assinatura e renunciar MD

Criptografia de Chave Pública:

- Autenticidade e Não-Repúdio com assinaturas digitais
- Confidencialidade com encriptação de chave pública, usada para transportar chaves simétricas

"Perfect Forward Secrecy": chaves de longo-prazo comprometidas não devem comprometer chaves de sessão

↓
PKs não podem ser usadas para transportar chaves simétricas

Protocolo Diffie-Hellman: os parâmetros públicos são (G, g, \cdot)

Grupo G: valores $[1..f]$ para um número primo grande f

Oeração: realiza dois grupos de elementos num terceiro

Grupo Gerador g: permite codificar invariavelmente um grande inteiro para o grupo

Para $x \in \{0 \dots f-1\}$, $g^x = g \cdot g \cdot g \dots \cdot g$ produz elementos diferentes em G

$$k = (g^n)^y = g^{ny} = g^{y^n} = (g^y)^n = k$$

DH



$x <-\$[0..q]$
 $X <- g^x$

$K <- Y^x$

X

Y



$y <-\$[0..q]$
 $Y <- g^y$

$K <- X^y$



$x <-\$[0..q]$
 $X <- g^x$

$K <- Y^x$

X

Y



$y <-\$[0..q]$
 $Y <- g^y$

$K <- X^y$

- DH garante anonimidade
- Se as mensagens forem simplesmente encaminhadas, um ataque "Man-in-the-Middle" é indetectável porque o atacante pode usar K_A e K_B

Como estabelecer canais seguros?

1. Chaves Públicas Autenticadas → proteger acordo de chaves
2. Protocolos de Acordo de Chaves
 - ↳ proteger autenticidade de chaves simétricas
 - garantir segredo das chaves para confidencialidade das mensagens
3. AEAD para trocar mensagens

Como instanciar canais seguros? TLS 1.3

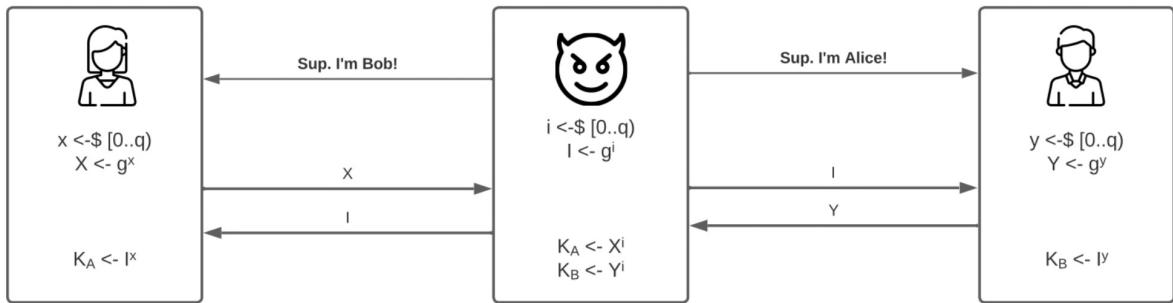
Problema: dada pk_A , como é que o Bob sabe que corresponde à chave secreta sk_A , conhecida apenas pela Alice?

Infraestrutura de Chave Pública: a confiança em chaves é validada pela confiança numa autoridade central, garantindo-se escalabilidade através de validação hierárquica das chaves

Request For Comments (RFC): documento público que denota como é que os protocolos devem ser implementados

Nonce: um número que só é utilizado uma vez

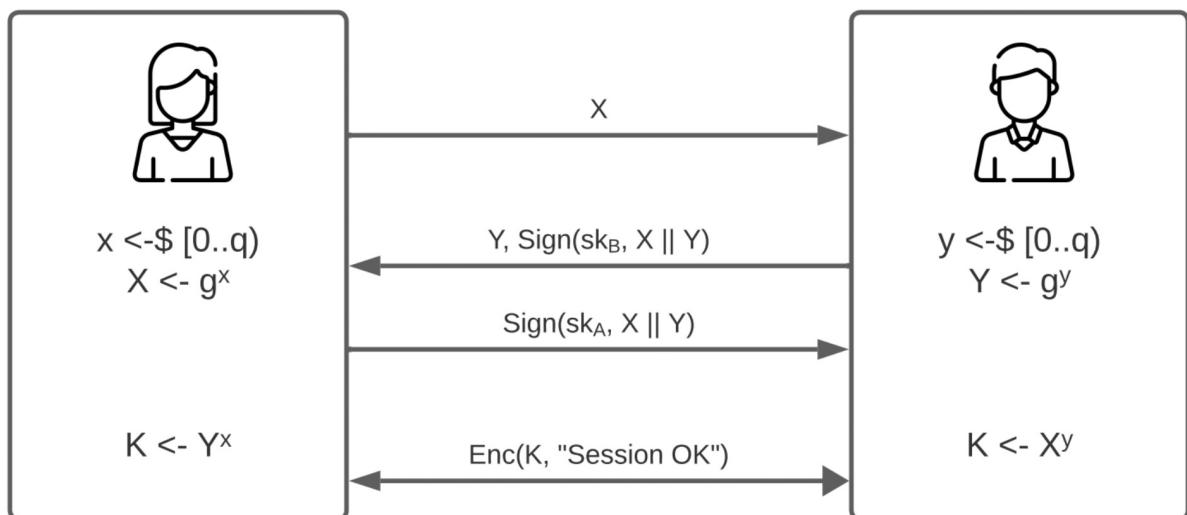
Man - in - the - Middle



DH Autenticação

Assumption:
 (sk_A, pk_B)

Assumption:
 (sk_B, pk_A)



Infraestruturas de Chave Pública

Criptografia de chave pública permite chaves públicas autênticas, caso contrário podem ocorrer ataques de "Man-in-the-Middle"

Normas Técnicas: que algoritmos usar

Standardização: como é esperado que as normas técnicas sejam aplicadas & responsabilidades e direitos dos participantes

Leis: garantias formais e responsabilização na violação de regras

Autoridade Central Confidada: fornece assinaturas que confirmam a identidade das pessoas - certificados

Infraestrutura de Chave Pública: A e B podem não confiar em CA_A ou CA_B, mas confiam em CA_{root} que certifica outras CAs que, por sua vez, certificam chaves públicas, permitindo a troca de certificados - hierarquia de confiança

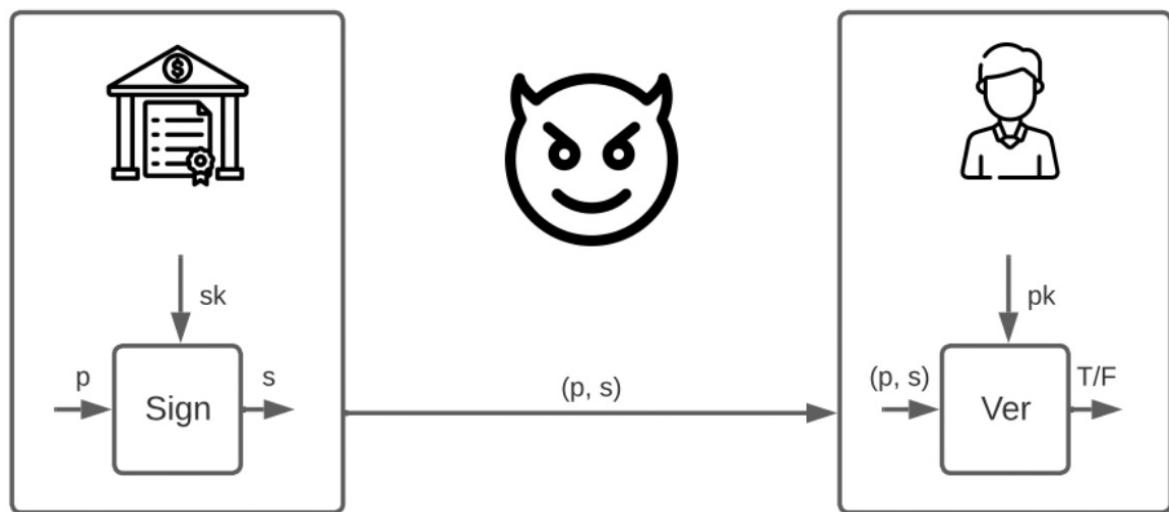
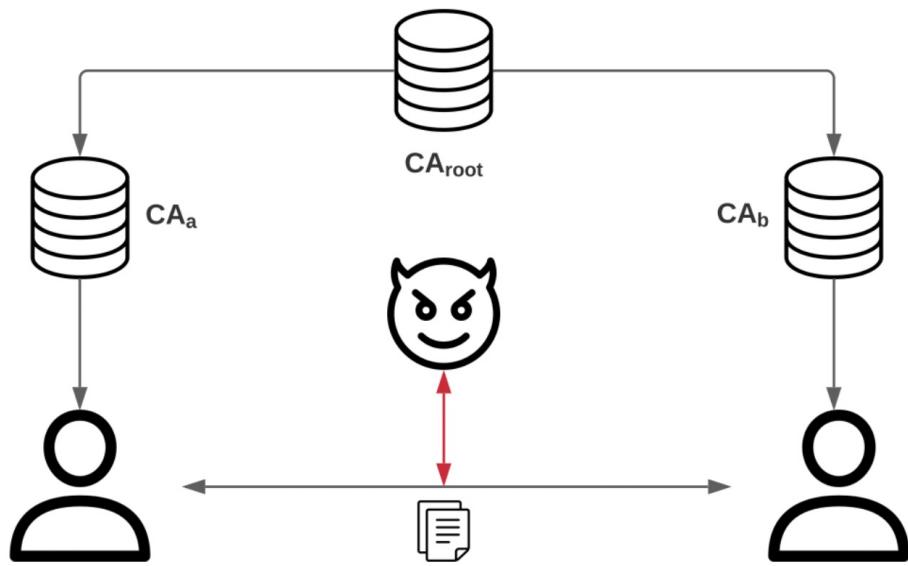
Trusted Computing Base (TCB): componentes que se acreditam funcionar como esperado

Trusted Third Party (TTP): confirma/desmente se X detém pk_x

1. Como construir um canal seguro entre A e TTP?
 2. O que fazer se TTP estiver inacessível?
 3. Como assegurar que A e B confiam em TTP?
 4. O que realmente significa confiar em TTP?
- } CERTIFICADOS } REGULAÇÃO

Como resolver? Com certificados de chave pública, usando assinaturas digitais

Infraestrutura de Chave Pública



• A Autoridade Certificadora (CA) deve validar toda a informação referida no certificado (identidade, chave pública, validade, ...), assinando um documento digital com esta informação

Certificado: documento codificado com Abstract Syntax Notation 1 (ASN1)

Que verificam?

1. coincidência entre a identidade e a chave com o certificado
 2. prazo de validade do certificado
 3. metadados
 4. confiança na CA
 5. assinatura do certificado pela CA
- } PKI

Certificado n.º 509

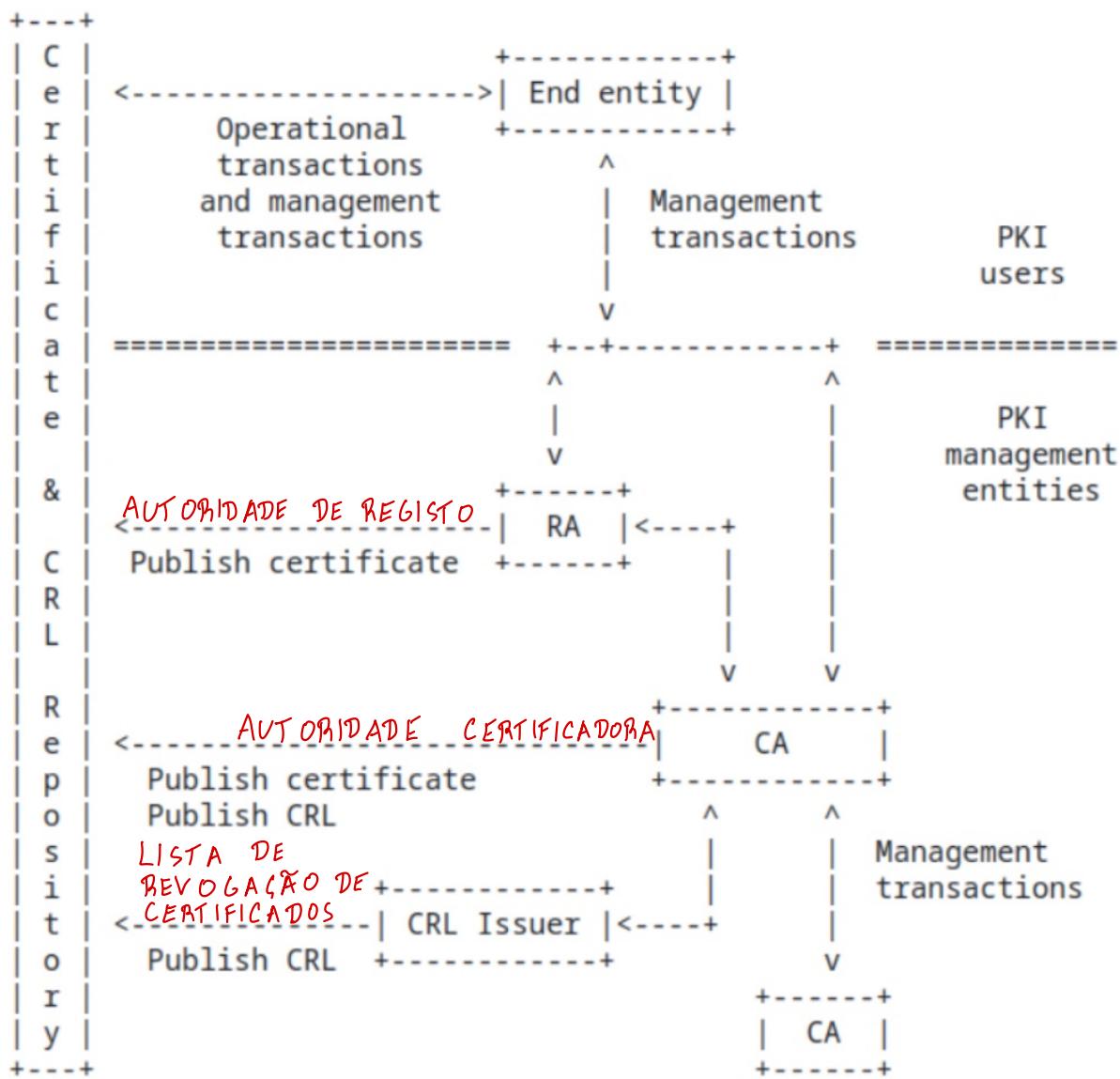
<ul style="list-style-type: none">• Sujeito: identidade do utilizador• Emissor: identidade da CA• Validade• Chave Pública• Número de Série
--

Extensões n.º 509

<ul style="list-style-type: none">• Sujeito / Autoridade Identificadora da Chave: hash da PK• Restrições Básicas: flags que limitam o certificado original• Uso da Chave: contexto em que pode ser usada
--

Infraestrutura de Chave Pública: permite que a identidade de um utilizador seja garantida através de um certificado produzido por uma autoridade certificadora confiável, ligando uma chave pública a um utilizador específico e potencialmente sobre um dado contexto de utilização, com responsabilidades bem definidas para todos os participantes

RFC 5280



Como são distribuídos e armazenados os certificados? Com especificações concretas nos múltiplos protocolos, mas sempre codificados de forma a garantir interoperabilidade

Como verifica os requisitos de uma PKI? Os utilizadores devem estabelecer alguma comunicação com a CA

todas as chaves públicas não codificadas em certificados X.509 e alguns certificados especiais contêm chaves públicas de CAs

Como confiar numa CA? Implicitamente (pré-instaladas ou estatais)

PROBLEMA

certificados de raiz auto-armados

Qualquer pessoa pode gerar um certificado auto-armado

VALIDÁ-LO IMPLICA

Confiar que a SK associada pertence à CA

Confiar que a CA é confiável, pelo que os seus certificados também

Hierarquia de CAs: se CA_A assina CA_B, a confiança em B é menor ou igual que a confiança em A

Quando é que um certificado é invalidado/revogado?

1. Depois do período de validade
2. Se as chaves secretas forem perdidas
3. Se as CAs forem comprometidas
4. Se os metadados deixarem de ser válidos

Como? Via Listas de Revogação de Certificados (CRLs)

CRL: CA publica periodicamente uma "black-list" dos certificados

1. Trusted Service Provider Lists (TSL): atualiza frequentemente "white-list"
2. Online Certificate Status Protocol (OCSP): serviço seguro verifica revogação
3. "Pinning" de Certificados: geridos individualmente pelos serviços/browsers/apps, identificam revogações críticas de certificados

Pretty Good Privacy (PGP): privacidade e autenticação na comunicação de dados, baseada numa rede de confiança descentralizada

- Confiança direta estabelecida por canais seguros
- Confiança indireta através do reto de confiança distos

Imprensa Digital PGP: chave pública conta impresa

Autorização: decidir que ações um utilizador pode realizar no sistema - conjunto bien definido de restrições sobre os recursos do sistema

Autenticação: determinar se um utilizador deve ter acesso ao sistema

- Numa máquina local, com algo que sabe (palavra-passe), tem (cartão) ou é (dados biológicos)
- Numa rede, com protocolos de segurança - regras de comunicação seguidas numa aplicação de segurança Ex: SSL; IPsec; SSH; ...

Protocolo de Autenticação: provar a identidade e estabelecer uma chave de sessão para efeitos criptográficos

Ataque de Repetição: o adversário observa a interação e usa as mensagens para repetir um falso de comunicação

↓ PREVENÇÃO

Challenge-Resposta: um desafio é escolhido de maneira que a resposta não seja fornecida, só A fornece a resposta correta e B consegue verificá-la eficientemente

Desafio:Nonce - cada pedido para autenticação deve usar um diferente

Resposta: Hash (resistente a colisões) - a mensagem usada para a primeira autenticação não vai funcionar para nenhuma das seguintes

→ Alivinhos Passwords

Keyloggers: verificar palavras-passe armazenadas, na cache do browser...

Hardware: dispositivos entre o teclado e o computador

Software: malware que intercepta teclas

Ataques de Dicionário: adicionar sal aleatório e único para cada utilizador, armazenando $(r, H(r \parallel pw))$ em vez de $H(pw)$

Phishing: o adversário simplesmente convence o utilizador a dar-lhe a chave

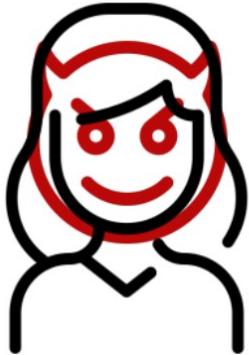
HTTPS: apresentar um servidor incorreto, nem o certificado

HTTP: mudar o website para um falso

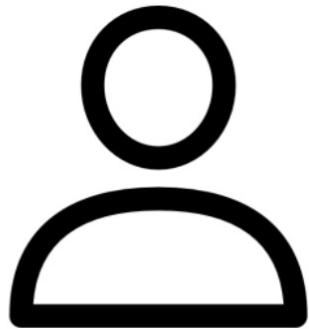
Homófogo: registar domínios semelhantes aos amplamente populares

Ataque de Repetição

Alice



Bob



I'm Alice.

Prove it.

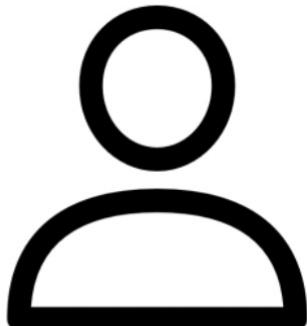
My pwd is "banana"

Challenge - Response

Alice



Bob



I'm Alice.

nonce

$h("banana" \parallel nonce)$

Protocolos de Segurança de Redes

World Wide Web: aplicação cliente-servidor que corre na internet e em intranets TCP/IP

Camadas Open System Interconnection (OSI):

Sockets, SSL, FTP, HTTP, Utilizador Final } 7. Aplicação

6. Apresentação

5. Sessão

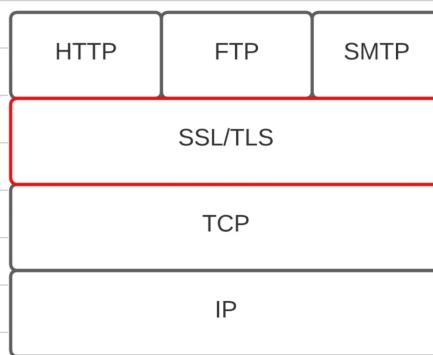
4. Transporte

3. Rede

Protos IP + TCP/UDP } 2. Ligação de Dados

1. Física

Estrutura Física + Ethernet } 2. Ligação de Dados

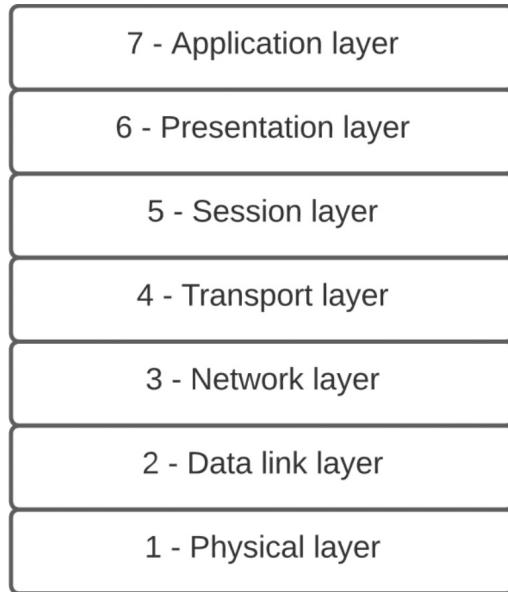


- SSL/TLS é middleware entre a aplicação e TCP
- IPsec refina o protocolo IP

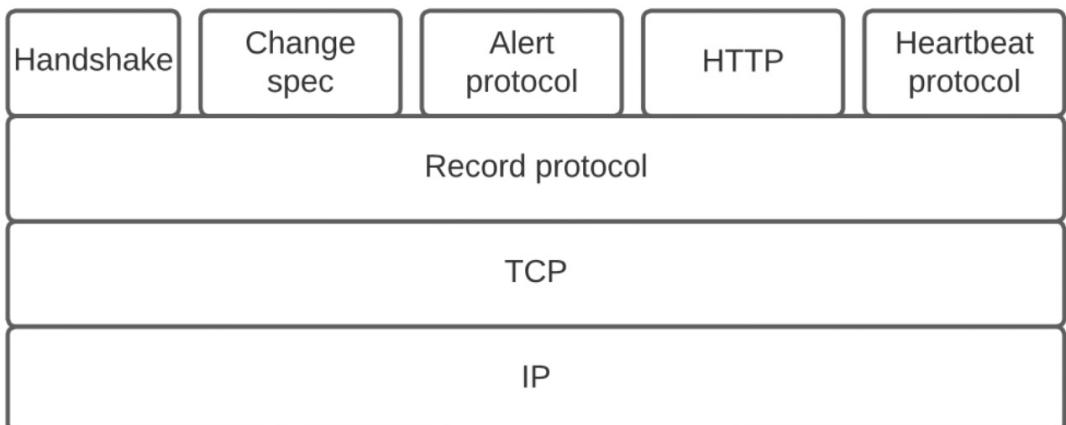
Secure Socket Layer (SSL): protocolo usado para a maioria das transações seguras na internet

Transport Layer Security (TLS): evolução do protocolo SSL comercial

Camadas OSI



Pilha do Protocolo SSL/KLS



Conexão TLS: transporte que fornece um tipo de serviço adequado — as conexões não P2P, transientes e cada uma associada a uma sessão

Versão TLS: associação entre um cliente e um servidor, criada pelo protocolo de handshake, que define um conjunto de parâmetros de segurança criptográfica partilhados entre múltiplas conexões, usado para evitar etapas de negociação diferentes no início de cada conexão

Protocolo Record: Integridade e Confidencialidade da mensagem, usando a chave acordada no handshake — a unidade resultante é transmitida via TCP e o receptor desencripta, verifica, descompõe e "re-monta"

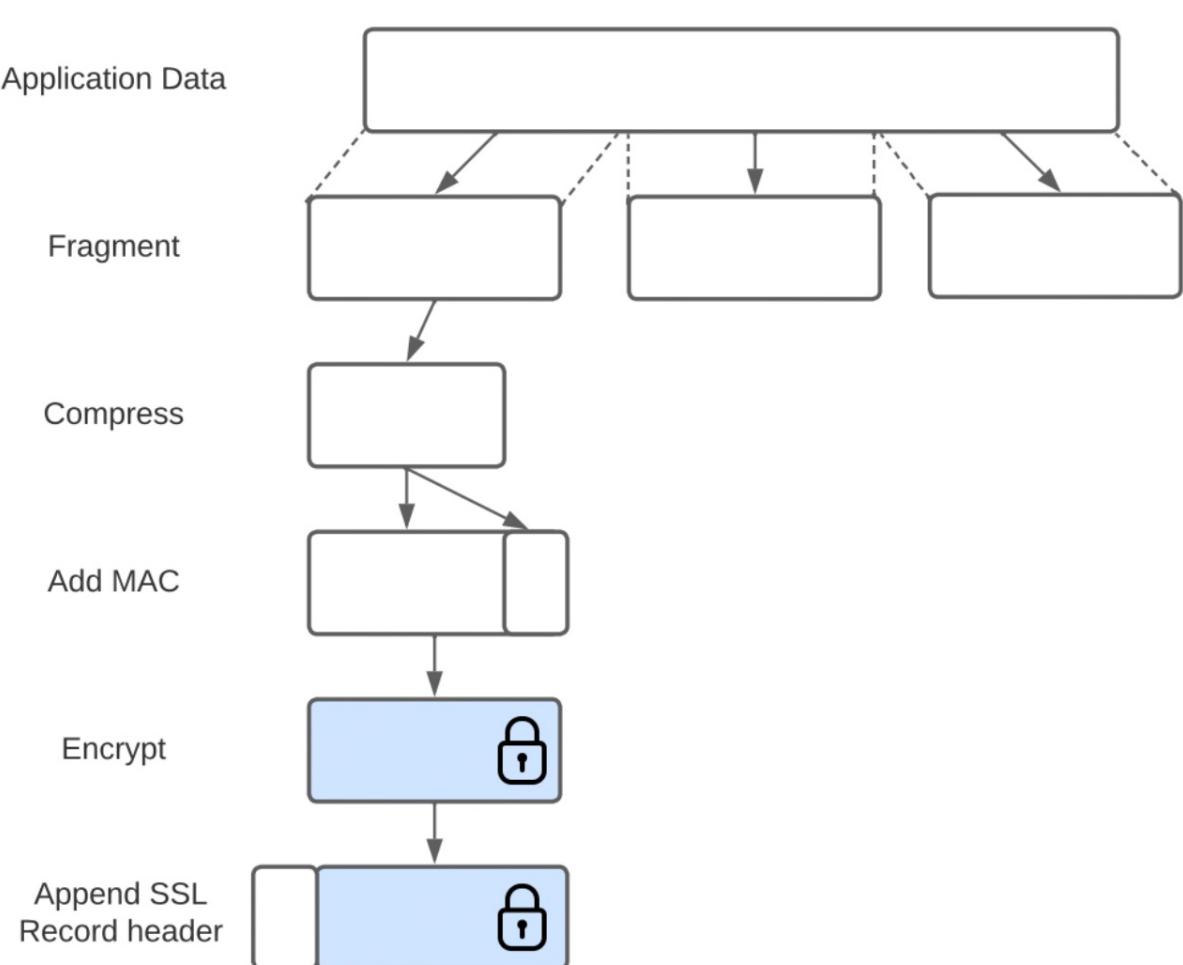
Handshake: crucial para estabelecer uma chave criptográfica, modo antes de quaisquer dados da aplicação serem transmitidos e constituído por uma série de mensagens trocadas pelo cliente e servidor — permite autenticação mútua, bem como negociar os algoritmos de encriptação e de MAC e as chaves criptográficas

1. Hello! → especificações: versão TLS, ID da sessão, Cipher Suite e método de compunção

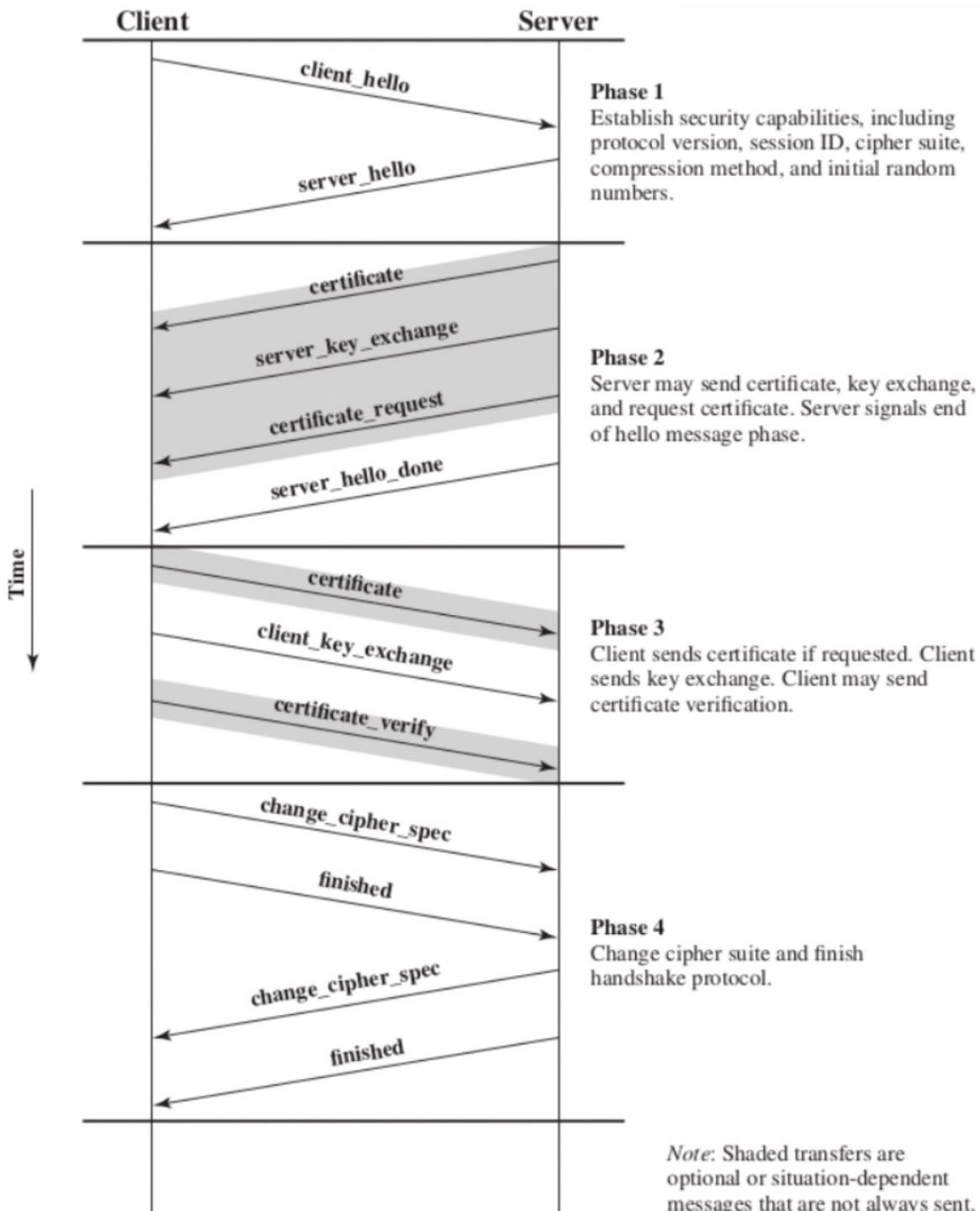
2./3. troca e verificação de certificados, bem como acordo da chave

4. cliente envia as especificações da cifra e uma mensagem de fim protegida com encriptação autenticada usando os novos algoritmos, negociações e chaves; o servidor verifica e faz o mesmo

Record



Handshake



Change Cipher Spec: mensagem única (um byte a 0 ou 1) que estabelece as especificações da cifra acordada, atualizando o Cipher Suite em uso
- mensagem de confirmação que faz copiar o estado pendente para o atual

Protocolo de Alerta: alertas TLS que podem provocar avisos (não-fatais) ou terminar conexões (fatais) - cada mensagem, comprimida e encriptada, é constituída por dois bytes: o primeiro refere-se à severidade e o segundo especifica

Nota: com avisos fatais/não-fatais (?), outras conexões da sessão podem continuar, mas não são estabelecidas conexões adicionais

Protocolo Heartbeat: faz "ping" regularmente e previne a conexão de terminar - sinal gerado periodicamente para indicar operação normal ou para sincronizar com outras partes do sistema, monitorizando a disponibilidade de uma entidade do protocolo

HEARTBEAT REQUEST: "para que estás vivo"
HEARTBEAT RESPONSE: "eu estou, de facto, vivo"

Heartbeat: enviar a mensagem de "heartbeat", esperar, preparar e enviar a resposta com o tamanho exato do conteúdo esperado

Heartbleed: pequeno "payload" desfasado de grande, cuja resposta mal preparada contém muito mais do que o esperado

HTTPS: combinação de HTTP e SSL para implementar comunicação segura entre o cliente e o servidor - o agente atua como cliente HTTP e TLS
🕒 URL
🕒 conteúdos do documento, de formulários e do cabeçalho HTTP
🕒 cookies

Secure Shell Protocol (SSH): fornece um caminho autenticado e encriptado para uma linha de comandos do SO sobre a rede, protegendo contra ataques de "spoofing" e modificação de dados

Transport Layer Protocol: fornece autenticação do servidor, confidencialidade e integridade

1. Acordo do protocolo e de versões de software
2. Troca de algoritmos suportados
3. Fim da troca da chave
4. Último ponto a executar

Acordo do Algoritmo: algoritmo de encriptação para confidencialidade, algoritmo MAC para autenticação e algoritmo de compreensão

User Authentication Protocol: autentica o utilizador do lado do cliente no servidor
Chave Pública: o cliente envia uma mensagem com a sua chave pública e assinada com a chave privada; o servidor verifica se a chave é aceitável para autenticação e se a assinatura está correta

Password: o cliente envia uma mensagem com a password, encriptada via TLP

Hostbased: o cliente envia uma assinatura criada com a chave privada do seu anfitrião, sendo esta identidade verificada - fornece anonimato de grupo

UAC

Connection Protocol: multiplica o túnel encriptado em vários canais lógicos

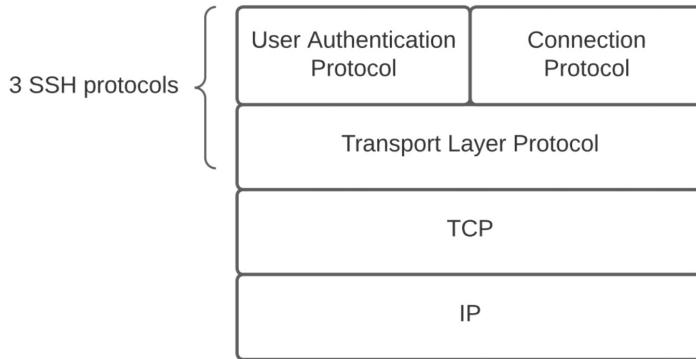
Sessão: execução remota de um programa

X11: X Windows System que fornece uma GUI para computadores em rede

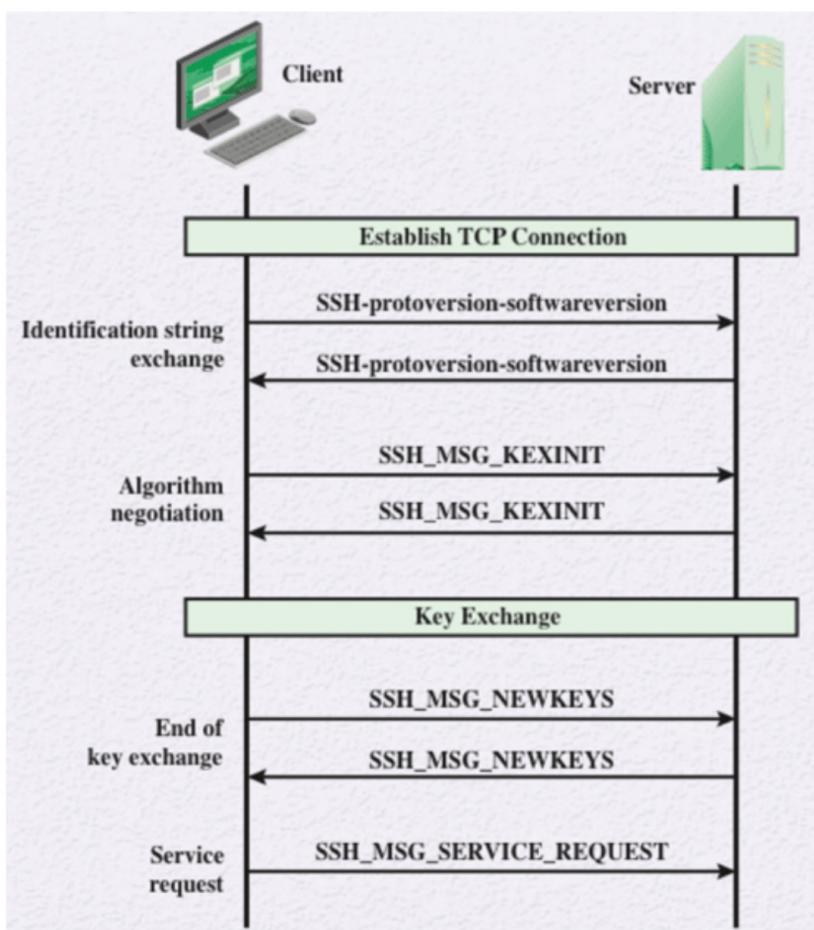
forwarded-tcpip: encaminhamento remoto dos portos (PC remoto → PC local)

direct-tcpip: encaminhamento local dos portos (conexão TCP segura → túnel SSH)

SSH



TLS



IPSec: Segurança implementada na camada de rede — prospera em aplicações em que a mesma segurança é sempre necessária e as mesmas técnicas de segurança podem ser aplicadas a todas as aplicações

- Quando implementado numa firewall ou router, fornece segurança forte a todo o tráfego que entra o perímetro — contexto claro do âmbito
- Transforma as aplicações e utilizadores finais
- Protege a arquitetura de rotas contra IP Spoofing, com autenticação e integridade para todas as mensagens de rotas

1. Gestão de Troca de Chaves: IKE
2. 2 extensões de segurança de cabeçalhos: AH e ESP
3. 2 modos de operação: Transporte e túnel

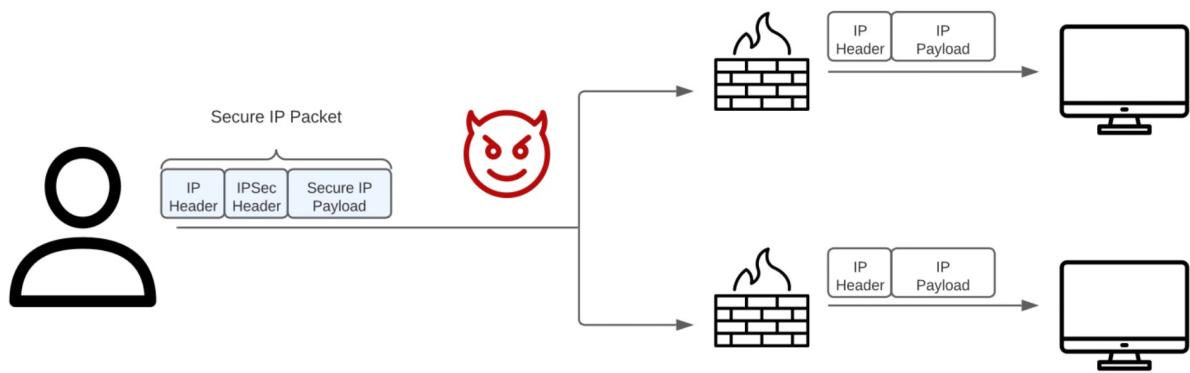
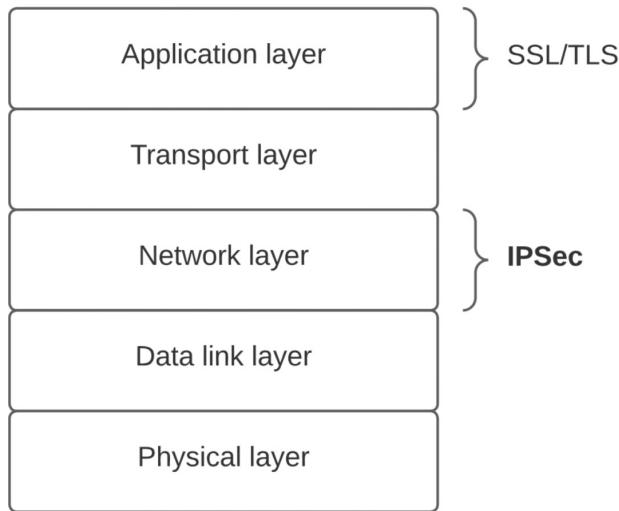
Internet Key Exchange (IKE):

1. Cookies impedem ataques de entrupimento
2. Especifica os parâmetros globais usados por DH
3. Usa Nonce para prevenir contra ataques de repetição
4. Permite que DH troque valores das chaves públicas
5. Autentica DH contra ataques "man-in-the-middle"

1. **IKE security association (SA):** handshake; seleção dos parâmetros criptográficos para autenticação e troca de chaves; escolha de um regredor master \approx sessão SSL/TLS

2. **IPSec security association (SA):** efêmera; usa a fase 1 para selecionar as chaves de encriptação / MAC \approx conexão SSL/TLS — serviços para conexões seguras (é uma conexão de um envio entre um emissor e um receptor que trata das regras de segurança para o tráfego nela transportado)

IP Sec



- a. Security Parameters Index (SPI): número numinal de 32 bits atribuído, apenas com significância local
- b. Identificador do Protocolo de Segurança: indica se associação AH ou ESP
- c. Endereço IP de Destino: sistema do utilizador final ou rede (firewall/router)
- Em qualquer pacote IP, a SA é identificada de forma única por (c) no cabeçalho IPv4/IPv6 e (a) no cabeçalho da extensão (AH/ESP)

Security Policy Database (SPD): meios pelos quais o tráfego IP se relaciona com SAs - cada entrada define o subconjunto de tráfego IP e aponta para SAs.

- Um cabeçalho IP não pode ser encRIPTADO porque os routers não têm acesso à chave de RSA, mas precisam ver o endereço de destino e alterar campos de dados para encaminhar o pacote
- O tráfego web encapsula dados iterativamente

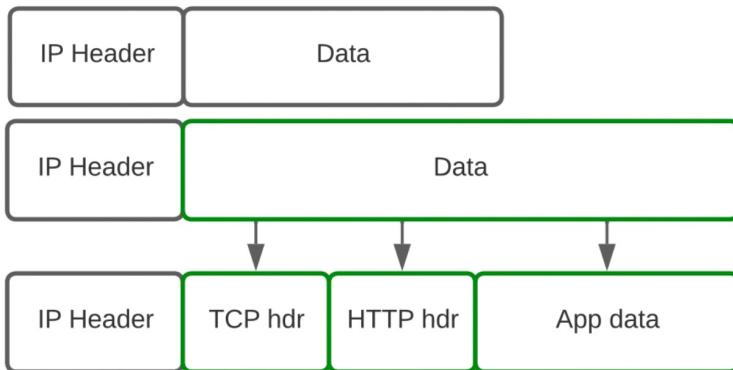
Modo de Transporte: adiciona informação de segurança ao pacote original

- desenhado para comunicação host-to-host
- muito eficiente - mínimo cabeçalho extra
- o cabeçalho original mantém-se - um atacante pode ver quem comunica

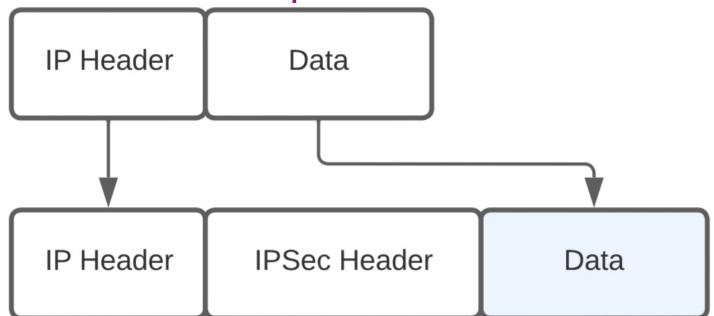
Modo de Tunnel: protege o pacote original, encapsulando-o num novo pacote IP

- desenhado para tráfego firewall-to-firewall
- o pacote IP original é encapsulado em IPsec
- o cabeçalho IP original (com os anfitriões no domínio) não é visível ao atacante, mas o novo cabeçalho (com as firewalls) é

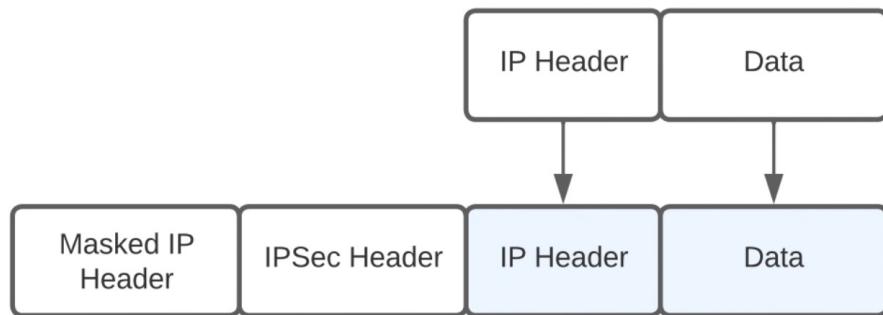
Datagrama IP



Modo Transporte



Modo Tunel



Encapsulated Security Payload (ESP): função de troca de chaves, combinada para autenticação/encrustação — regras Integridade e Confidencialidade, protegendo tudo para além do cabeçalho IP

Authentication Header (AH): função só de autenticação — apenas Integridade, protegendo tudo para além do cabeçalho IP e os campos invariantes do cabeçalho

Porque? ESP não protege a integridade do cabeçalho IP e encrustar dados previne que um firewall inspecione o seu conteúdo

SSL/TLS

- Camada de Socket
- Espaço do Utilizador
- Encrustação
- Integridade
- Autenticação
- Simples
- Elegante

• Aplicações devem estar cientes, SO não

- Para segurança ao nível da aplicação
- Fácil de adaptar às necessidades
- Não protege contra IP spoofing
- Relativamente em retroadaptação aplicações

IPsec

- Camada de Rede
- Espaço do SO
- Encrustação
- Integridade
- Autenticação
- Complexo

• SO deve estar ciente, aplicações não

- Usado em VPNs
- Túnel seguro
- Comunicações devem ser confidenciais e autenticadas
- Não amplamente desenvolvido

Ameaças e Contramedidas

Intusos:

1. Libercuminosos: indivíduos ou membros de grupo de crime organizado com o objetivo de uma recompensa financeira
2. Organizações Patrocinadas pelo Estado: grupos de hackers patrocinados pelos governos para conduzirem atividades de espionagem ou sabotagem - Ameaças Avançadas Persistentes: cobertas e persistentes
3. Ativistas: indivíduos (insiders ou membros de um grupo maior) motivados por causas sociais ou políticas, cujo objetivo é promover/publicizar a causa
4. Outros: motivados por desafios técnicos, estímulos de grupo e reputação

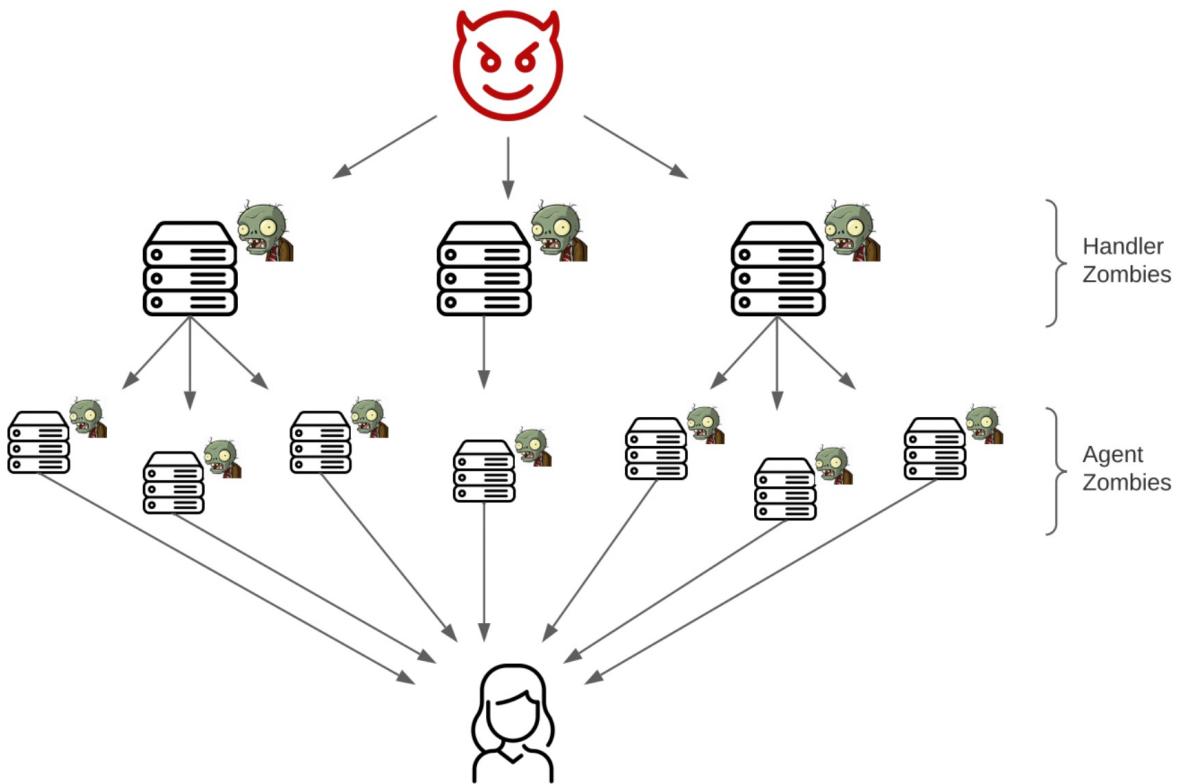
Níveis de Capacidade:

1. Aprendiz: com conhecimento mínimo, recorrem a ferramentas existentes
2. "journeymen": com conhecimentos técnicos suficientes para modificar e estender ferramentas de ataque para vulnerabilidades novas/recentes
3. Mestre: com altas capacidades técnicas, capazes de descobrir novas vulnerabilidades

Denial-of-Service (DoS): forma de ataque na disponibilidade dos serviços

1. Largura de Banda da Rede: capacidade das ligações da rede conectarem um servidor à Internet
2. Recursos do Sistema: software que lida com a rede
3. Recursos da Aplicação: fazer muitos pedidos a um servidor do sistema abusivo

DDoS



Flooding Ping: sobrecarregam a capacidade da conexão de rede com a originação da rede — leva a descartar pacotes

• A origem do ataque é claramente identificada, a não ser que se use um endereço "spoofed" (servidor zumbi)

Botnet: rede de computadores infectados com software malicioso que permite que sejam controlados por um atacante (zumbis)

Attack-as-a-Service: serviços de Controle e Comando (C & C) são responsáveis por comandar computadores infectados — "bot-hander" é o atacante

UPD Flood: o hacker envia pacotes UDP para um porto aleatório, gerando pacotes UDP ilegítimos e fazendo com que o sistema gaste recursos ao enviar os pacotes de volta

Ataques de Reflexão: o atacante envia pacotes para um serviço conhecido no intermediário com um endereço de origem "spoofed" na vítima, fazendo com que o intermediário responda à vítima — gerar volume de pacotes suficiente para inundar a ligação ao sistema alvo sem alertar o intermediário

Echo-CharGen: o serviço de eco (porto 07) devolve tudo o que recebe e CharGen é um serviço de geração de caracteres

Requisito: spoofing do endereço de origem

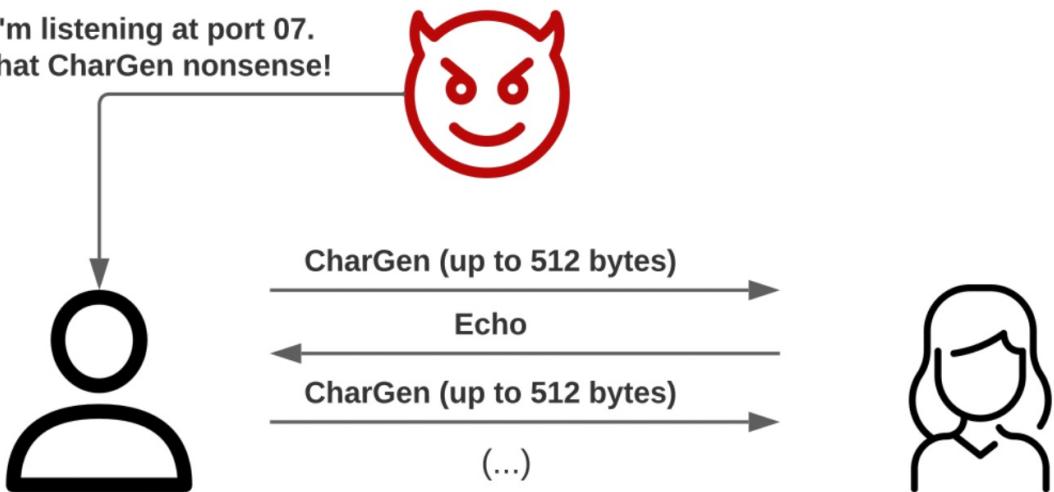
Smurf: o serviço faz broadcast de eco para toda a rede

Requisito: spoofing do endereço de origem

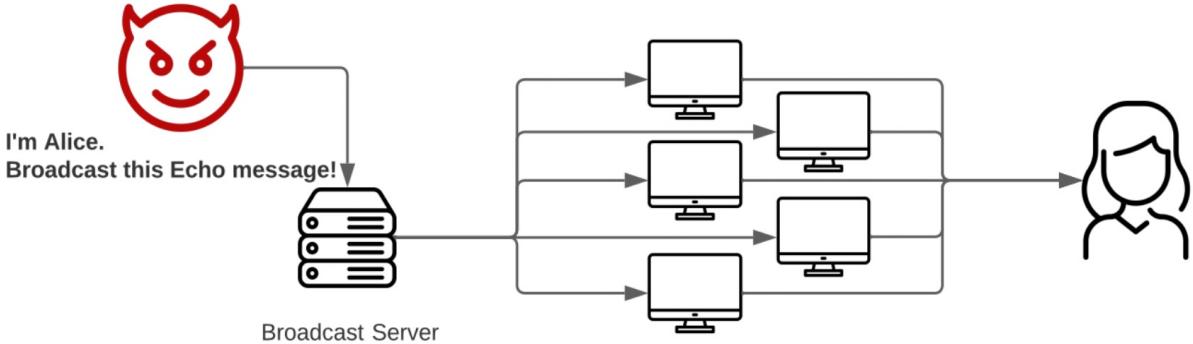
Requisito: acesso a um serviço na rede

Echo - CharGen

I'm Alice, I'm listening at port 07.
Send me that CharGen nonsense!



Smurf



SYN Flooding: o atacante envia SYN com uma origem "falsa" (inexistente) e o servidor responde com SYN-ACK sucessivamente até timeout

Amplificação de DNS: pedidos DNS (com o argumento ANY para conter um pedido pequeno numa resposta grande) com o endereço IP de origem "falsa" como o alvo — pedidos a muitos servidores ligados inundam o alvo

Ataques Enjetor: com preparação sofisticada — não aplicações triviais de vulnerabilidades conhecidas

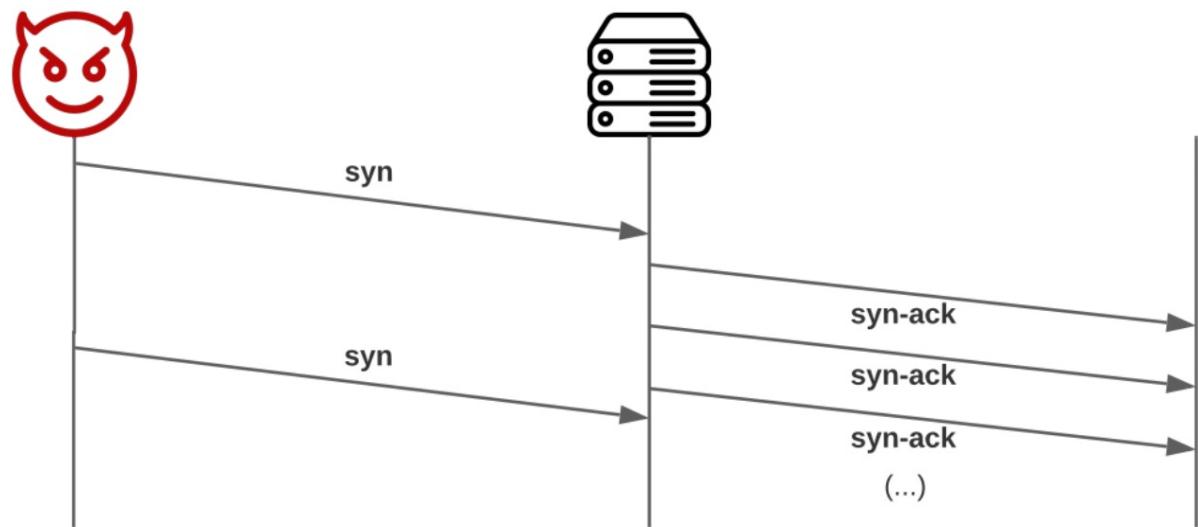
Contingências:

1. Prevenção e Preempção: ANTES — políticas para consumo de recursos e recursos de backup disponíveis
2. Deteção e Filtros: DURANTE — procurar pacotes suspeitos e filtrar
3. Isolamento e Identificação: DURANTE/DEPOIS — identificar origens e preparar listas brancas/verdes
4. Reação: DEPOIS — eliminar efeitos do ataque, limpando o sistema

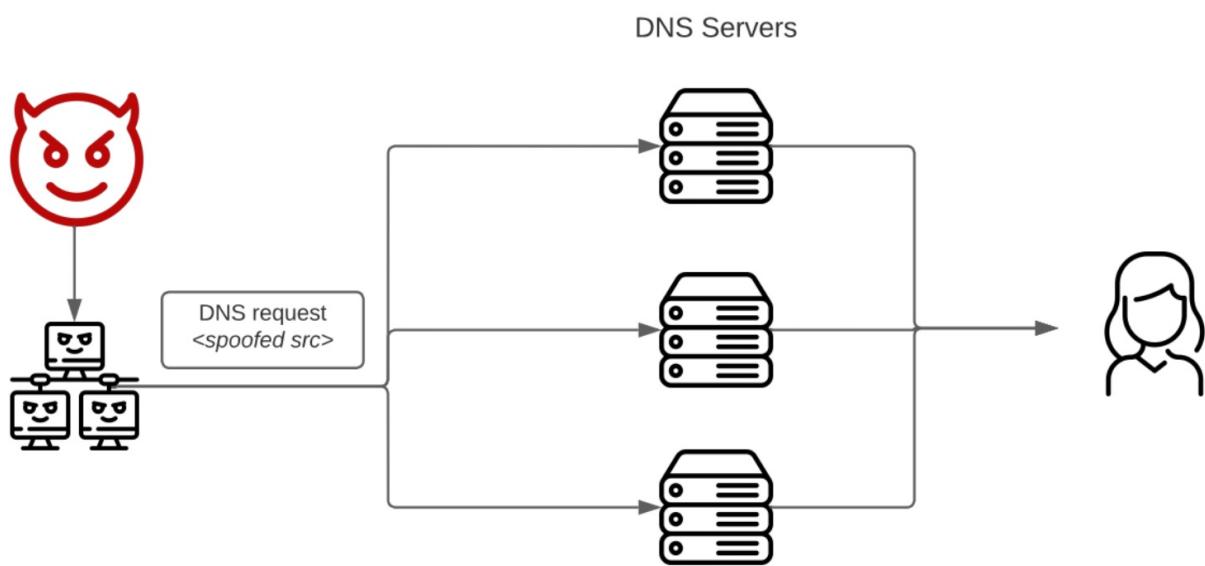
Firewall: decide o que entra e sai de uma rede interna — controlo acesso com base numa política de acesso, tipo de tráfego, endereço, protocolo, ...

- :- ponto de estrangulamento único
- :- localização para monitorizar eventos de segurança
- :- plataforma conveniente para várias funções
- :- pode servir a plataforma para IPsec (modo túnel)
- :- não pode proteger contra "keyfests" nem ameaças internas
- :- um dispositivo pode ser infectado fora e usado dentro
- :- LAN nem fios inseguir podem ser acedidas a partir de fora

SYN Spoofing



Amplificação de DNS



Filtro de Pacotes: opera na camada de rede, observando pacotes IP e avaliando a sua importância — configurado via ACL

- ↪ fácil; rápido; transparente
- ↪ nem todo; vulnerável a ataques em bytes TCP/IP; não pode ver conexões TCP; desconhece os dados e o contexto da aplicação

Filtro de Pacotes Stateful: opera na camada de Transporte e adiciona estado ao filtro de pacotes, lembrando-se de conexões TCP e pacotes UDP

- ↪ pode fazer tudo o que um filtro de pacotes faz; regista conexões em curso; confia na lógica do protocolo para detectar maus comportamentos
- ↪ não pode ver dados da aplicação (falta lógica interna e diminui exatidão); mais lento

Proxy: algo que atua em nome de outro

Proxy da Aplicação (Gateway ao Nível da Aplicação): vê os dados que chegam à aplicação, verificando se são seguros antes de passarem

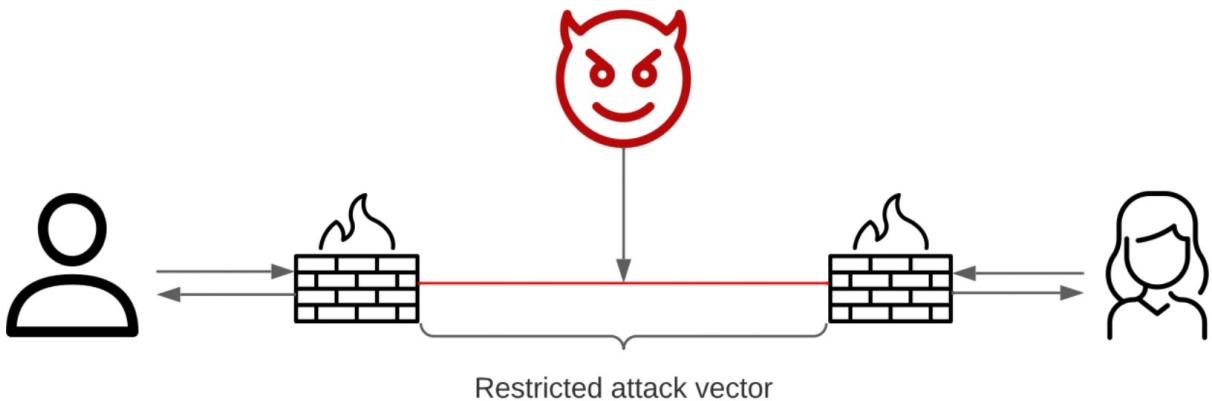
- ↪ vista completa de conexões e dados da aplicação; filtra na camada de aplicação, previnindo erros de software e exploração de vulnerabilidades; dificulta "spoofing"; configurável; específico; granular
- ↪ pior desempenho; cada aplicação tem de ter um código; muito problema/

Política da Firewall

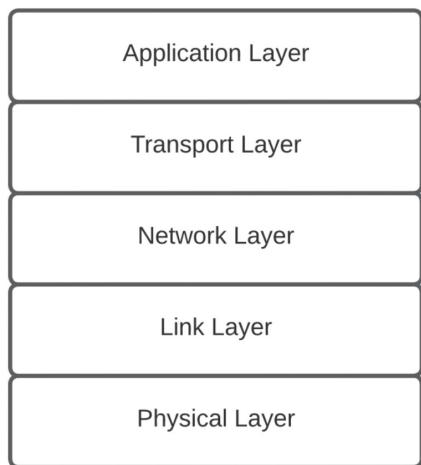
Permissiva: permite por defeito; bloquear alguns — falhas de segurança

Restritiva: bloquear por defeito; permitir alguns — falhas de disponibilidade

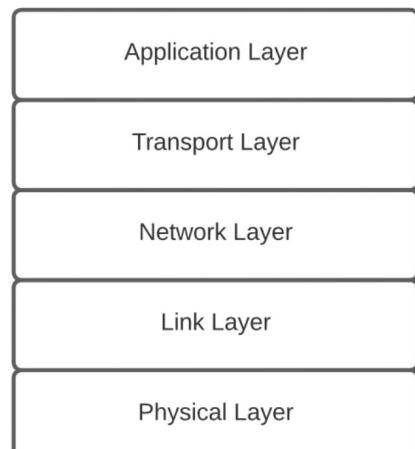
Firewall



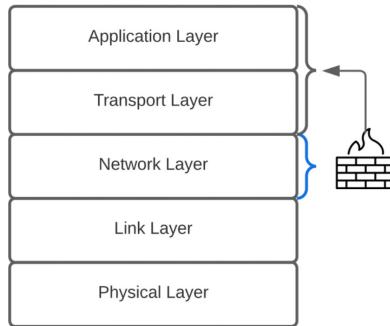
Filtro de Pacotes



Filtro de Pacotes Stateful



Proxy da Aplicação



Sistema de Detecção de Intrusos:

1. Disponibilidade: corre continuamente e fornece degradação graciona do serviço
2. Resiliência: tolera falhas e resiste à subversão
3. Desempenho: impõe um "overhead" mínimo e escala bem
4. Adaptabilidade: configurado de acordo, mas adaptável e reconfigurável

Host-Based IDS: monitoriza atividades nos anfítrios, podendo detectar intrusões externas e internas, mas com pouca visão das atividades da rede

Network-Based IDS: monitoriza atividade em pontos selecionados da rede para ataques conhecidos, examinando protocolos ao nível do transporte, da rede e da aplicação, mas com pouca visão de ataques baseados nos anfítrios ≈ firewall

Deteção de Assinaturas: só identifica ataques com base num determinado conjunto de padrões de dados maliciosos ou regras de ataques — um padrão identificado como uma assinatura origina um aviso

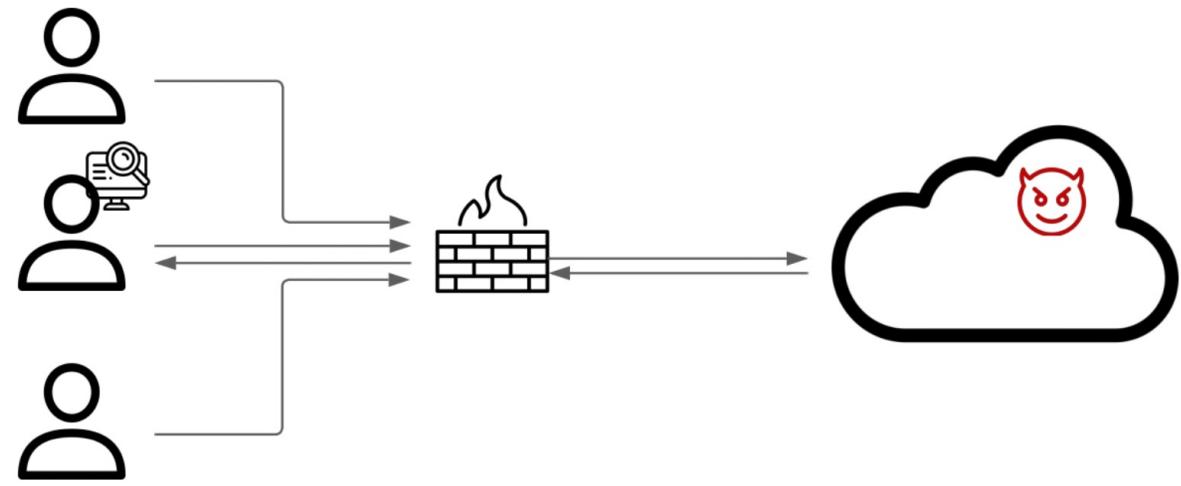
- simples; deteta ameaças comuns/conhecidas; identifica exatamente os ataques detectados; eficiente
- assinaturas devem ser atualizadas e podem ser muitas; só deteta ataques já conhecidos, pelo que variações podem evitá-la (ataca mas não impede)

Deteção de Anomalias: o comportamento observado é analisado para determinar se corresponde a um utilizador legítimo ou a um intruso — tem de medir o comportamento normal do sistema, que está em constante evolução, pelo que um atacante pode manipular o comportamento e convencer o IDS

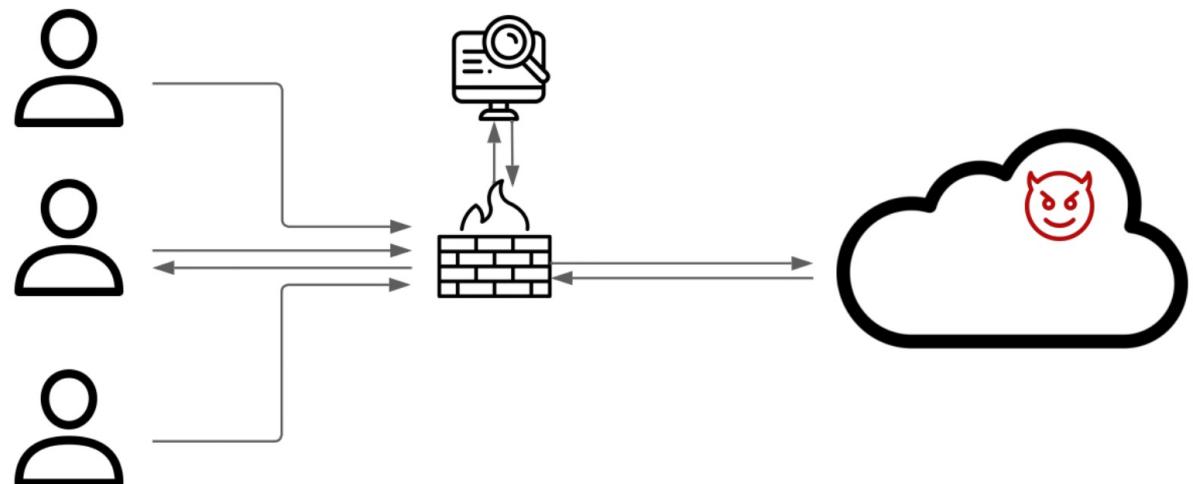
Falácia do Náscio-Baix: probabilidade de um evento condicional ser avaliada nem considerar o "náscio-baix" do evento

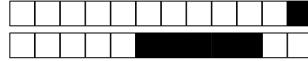
Falso Positivo (FP): identificado quando não ocorre Falso Negativo (FN): não identificado quando ocorre

Host - Based IDS



Network - Based IDS





This is a multiple-choice test that will be corrected automatically. Please follow these rules:

- No pencil or light-coloured pens. Mark your answer using only **blue** and **black** pen.
- Erased boxes are not detected automatically. Check only inside boxes and be generous on ink.
- You may underline text or take notes on the sides. Only boxes matter for automatic correction.

The test is marked for 20 points. Each question is worth 20/30 points.

There are 30 questions in total, each with 4 options. **Only one of those options is accepted as correct, which could be an option indicating all options are correct.**

You can check one or two choices per question. The scoring for each question is as follows:

- One checked correct answer (100%).
- One checked incorrect answer (-20%).
- No checked answers (0%).
- Two checked answers, one correct (50%).
- Two checked answers, none correct (-20%).
- More than two checked answers (-20%).

0 0 0 0 0 0 0 0
 1 1 1 1 1 1 1 1
 2 2 2 2 2 2 2 2
 3 3 3 3 3 3 3 3
 4 4 4 4 4 4 4 4
 5 5 5 5 5 5 5 5
 6 6 6 6 6 6 6 6
 7 7 7 7 7 7 7 7
 8 8 8 8 8 8 8 8
 9 9 9 9 9 9 9 9

← code your 9-digit upYYYYXXXXX student number horizontally on the left, and replicate it below. Write also your first and last name below.

Student Number:

up

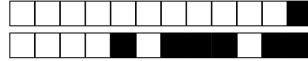
First and Last Name:

.....

✓ Group 1 Cryptography (10 questions)

Question 1.1 Consider a scenario where A sends (pk, m, σ) to B where m is a message and $\sigma = \text{Sig}(sk, m)$ is a (secure) digital signature of m . A Man-In-The-Middle (MitM) attacker could persuade receiver B that A sent $m' \neq m$ by:

- MitM attacks are not possible when using digital signatures.
- Keeping pk , changing m for m' and generating a new signature σ' .
- Keeping pk , and replacing (m, σ) with (m', σ) .
- Convincing B that A owns a public key different than pk .



✓ ? **Question 1.2 ♣** The difference between an authenticated encryption scheme (AE) and an authenticated encryption scheme with associated data (AEAD) is that:

- AEAD probabilistic whereas AE is deterministic.
- AEAD allows binding public metadata to the ciphertext, while AE does not.
- AEAD allows associating metadata to encryption keys, which the AE does not allow.
- AEAD guarantees metadata confidentiality, which the AE does not allow.

✓ **Question 1.3 ♣** The statement “*Public-key cryptography made symmetric cryptography obsolete*” is:

- False: the two techniques are always used together for performance reasons.
- True, but only in applications where one cannot have a pre-shared secret key.
- True, except for applications where we need protection against quantum computers.
- False: if a PKI is not available one always uses symmetric cryptography.

✓ **Question 1.4 ♣** The Electronic Code Book operation mode is:

- A secure MAC scheme construction that uses a block cipher.
- An insecure symmetric encryption scheme construction that uses a block cipher.
- An insecure MAC scheme construction that uses a block cipher.
- A secure symmetric encryption scheme construction that uses a block cipher.

✓ **Question 1.5 ♣** In a KDS, a Key Distribution Center interacts with N agents and:

- Stores \cancel{N} long-term key that it uses to establish an arbitrary number of session keys.
- Stores $N * (\cancel{N} - 1)/2$ long-term keys, which it makes available when needed for communication.
- Caches a varying number of session keys, which are gradually provided to the participants.
- Stores N long-term keys that it uses to establish an arbitrary number of session keys.

✗ **Question 1.6 ♣** The Perfect Forward Secrecy property guarantees that:

- Corrupting a session key does not compromise past session keys.
- Corrupting a long-term key does not compromise past session keys.
- Corrupting a session key does not compromise future session keys.

✓ **Question 1.7 ♣** Which is a correct assignment for (A) RSA signature, (B) AES-CTR, (C) RSA-OAEP and (D) HMAC?

	Confidentiality	Authenticity
Symmetric	(1) B	(2) D
Asymmetric	(3) C	(4) A

- 1-B;2-D;3-C;4-A.
- 1-A;2-~~B~~;3-C;4-D.
- 1-C;2-A;3-~~D~~;4-B.
- 1-~~A~~;2-C;3-B;4-D.

✓ **Question 1.8 ♣** The non-repudiation property is important in the context of message authentication. Which of the following sentences is true? Note that MAC stands for Message Authentication Code.

- Asymmetric ciphers guarantee this property, as long as the public-key is authentic.
- A MAC does not guarantee this property.
- Digital signatures guarantee this property, even after the long-term secret key is compromised.
- A MAC guarantees this property, provided that the sender is trusted.



✓ **Question 1.9 ♣** Recall that a Message Authentication Code (MAC) has the following syntax $\text{MAC}(k, m) = t$. A MAC guarantees:

- Message integrity and authentication.
- Confidentiality, integrity and authentication for a sequence of messages.
- Message confidentiality, integrity and authentication.
- Integrity and authentication for a sequence of messages.

? **Question 1.10 ♣** A common construction of a secure symmetric encryption scheme is of the form $\text{Enc}(k, n, m) = \text{PRG}(k, n) \oplus m$. The following property shows that this construction *does not* provide integrity guarantees:

- Flipping a single ciphertext bit causes a single bit flip in the recovered message.
- The PRG generator produces a uniform distribution.
- The XOR operation cancels out:
$$\text{PRG}(k, n) \oplus \text{PRG}(k, n) \oplus m = m.$$
- The PRG generator does not produce a uniform distribution.

Group 2 Public-Key-Infrastructure (5 questions)

$A \rightarrow B$

✗ **Question 2.1 ♣** When using public-key certificates to transfer encrypted information from A to B using a public-key encryption scheme:

- B must get and a priori verify A's certificate.
- A must get and a priori verify B's certificate.
- A and B must hold certificates issued by the same certification authority.
- A and B must exchange and a priori verify each others' certificates.

✓ **Question 2.2 ♣** For a certificate authority, a Certificate Revocation List (CRL)

- Contains all issued certificates that, while being within the validity period, should not be used.
- Contains all issued certificates that should not be used.
- Contains only issued certificates that are within the validity period and can be used.
- Contains only issued certificates that can be used.

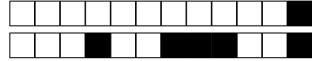
✓ **Question 2.3 ♣** Recall what you have studied about certificate chains. Suppose certification authority A signed the certificate of certification authority B, and that all you know about A and B is what is written in this certificate.

$CA_A \text{ signs } CA_B \quad t_B \leq t_A$

- Trust in A can never be greater than trust in B.
- Trust in B can never be greater than trust in A.
- B cannot operate until it also signs a certificate for A.
- A trusts B to sign a certificate for A.

✗ **Question 2.4 ♣** The public-key infrastructure solves the following problem:

- Sharing of asymmetric secret keys.
- Public key authentication.
- Public key authentication and confidentiality.
- Symmetric key sharing using public keys.



✓ **Question 2.5 ♣** Which is the most common channel for a user to get information about the certification authorities that work as trust roots in a PKI?

- Their certificates are distributed by the web sites the users visit.
- Users only get that information when they need a public key certificate for a web server.
- Their certificates are pre-installed in operating systems and browsers.
- Users only get that information when they need a personal public key certificate.

Group 3 Authentication (4 questions)

✗ **Question 3.1 ♣** Which is the best way for a web application to store session tokens on the client side?

- In cookies.
- A combination of all the other options.
- In the content of links.
- In hidden form fields.

✓ **Question 3.2 ♣** Which of the following is not an attack on a password-based authentication mechanism?

- Data breach in a server reveals user passwords.
- Phishing site steals user credentials.
- User chooses a weak password.
- Malware registers user keystrokes.

✓ **Question 3.3 ♣** Which does not represent a security risk for biometric authentication systems?

- Stealing characteristics from individuals.
- High rate of false positives.
- High rate of false negatives.
- Forging characteristics from individuals.

? **Question 3.4 ♣** Which is the main difference between message authentication (MA) e entity authentication (EA)?

- EA is intended to verify that an entity participates at real time in the protocol.
- In MA there exists typically a requirement that a message was sent recently, by the correct entity.
- In EA the receiver has the guarantee that a message was sent by a specific entity, contrary to MA where the receiver only knows that the sent message is valid.
- An EA mechanism requires the use of a MA mechanism, but not vice-versa.

Group 4 Network Security (6 questions)

✓ **Question 4.1 ♣** Which of the following is an attack at the level of the transport layer?

- Rogue DHCP.
- MAC flooding.
- DNS cache poisoning.
- TCP session hijacking.



✗ **Question 4.2 ♣** In the context of *firewall* packet filtering, which of the following statements is true?

- A *Default allow* policy typically offers more protection than a *Default deny* policy.
- Packet filtering does not distinguish inbound and outbound traffic.

Stateful filtering has the disadvantage of being harder to implement.

Stateless filtering has the disadvantage of being harder to configure exceptions for legitimate users.

✓ **Question 4.3 ♣** Consider attacks to the DNS protocol. Which of the following statements is **not** true?

- Both DNS spoofing and DNS cache poisoning permit directing users to malicious machines.
- DNS spoofing can be performed by malware directly in the user's machine.

DNS cache poisoning is an attack directed at a legitimate DNS server.

DNS spoofing consists in flooding a DNS server with IP registers.

✓ ? **Question 4.4 ♣** Which of the following attacks to the UDP/TCP protocols is harder to concretize?

- TCP session spoofing.
- UDP session hijacking.

TCP session hijacking.

Sending RST messages.

✓ ? **Question 4.5 ♣** At the level of network communications, which of the following statements is true?

- An *on-path* can only send packets.
- A *man-in-the-middle* attacker can control all communications.

An *off-path* can only receive packets.

An *eavesdropper* just cannot modify packets.

✗ **Question 4.6 ♣** A MAC address physically identifies a machine in a network. Which of the following statements is **not** true?

- A MAC spoofing attack allows usurping the MAC address of another machine.
- A MAC flooding attack can force a switch to broadcast all packets.

A MAC flooding attack intends to perform Denial of Service of a switch.

A MAC spoofing attack permits impersonating a hub/router/switch.

Group 5 Malware and Detection (3 questions)

✓ **Question 5.1 ♣** In what consists the concept of signature-based malware detection?

- Digitally signing the software so that malware cannot modify its execution.
- Detecting personal signatures that hackers leave in the code of the malware that they create.

Detecting patterns of known attacks.

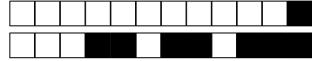
Identifying digital signatures of servers to which malware tries to connect.

✓ **Question 5.2 ♣** Which of the following statements is **not** true?

- A *worm* can be used to create a *botnet*.
- A *botnet* is a network of malware computers with a common control.

A *worm* is a self-propagating malware.

A *worm* is a malware used as "bait" to fool users.



✓ **Question 5.3 ♣** Which is **not** a strategy that a modern *virus* uses to avoid being detected?

- Dissimulate itself as a regular file and mutate when executing. ✓
- Terminating the processes launched by the antivirus.
- Behave differently when executed in a *sandbox*. ✓
- Encrypt its code in a probabilistic fashion on each infection.

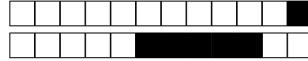
Group 6 Transport Layer Security (TLS) (*2 questions*)

✓ **Question 6.1 ♣** Which of the following attacks can be avoided using TLS connections?

- DNS spoofing.
- Man-in-the-middle attacks, as long as the client validates the server's certificate.
- Network traffic analysis to obtain meta-data.
- Web pages that include HTTP/HTTPS mixed content.

✗ **Question 6.2 ♣** Which is the difference in TLS 1.3 handshake to prior versions?

- Corrupting the server keys does not affect previous sessions.
- It makes use of RSA instead of authenticated Diffie-Hellman.
- Does not make use of long-term keys.
- For reasons of performance, connections do not always guarantee perfect forward secrecy.



This is a multiple-choice test that will be corrected automatically. Please follow these rules:

- No pencil or light-coloured pens. Mark your answer using only **blue** and **black** pen.
- Erased boxes are not detected automatically. Check only inside boxes and be generous on ink.
- You may underline text or take notes on the sides. Only boxes matter for automatic correction.

The test is marked for 20 points and contains 30 questions, each with 4 options. **Only one option is accepted as correct, which could indicate all options are correct.**

You can check one or two choices per question. The scoring for each question is as follows:

- One checked correct answer (100%).
- One checked incorrect answer (-20%).
- No checked answers (0%).
- Two checked answers, one correct (50%).
- Two checked answers, none correct (-20%).
- More than two checked answers (-20%).

□ 0	□ 0	□ 0	□ 0	□ 0	□ 0	□ 0	□ 0	□ 0	□ 0
□ 1	□ 1	□ 1	□ 1	□ 1	□ 1	□ 1	□ 1	□ 1	□ 1
□ 2	□ 2	□ 2	□ 2	□ 2	□ 2	□ 2	□ 2	□ 2	□ 2
□ 3	□ 3	□ 3	□ 3	□ 3	□ 3	□ 3	□ 3	□ 3	□ 3
□ 4	□ 4	□ 4	□ 4	□ 4	□ 4	□ 4	□ 4	□ 4	□ 4
□ 5	□ 5	□ 5	□ 5	□ 5	□ 5	□ 5	□ 5	□ 5	□ 5
□ 6	□ 6	□ 6	□ 6	□ 6	□ 6	□ 6	□ 6	□ 6	□ 6
□ 7	□ 7	□ 7	□ 7	□ 7	□ 7	□ 7	□ 7	□ 7	□ 7
□ 8	□ 8	□ 8	□ 8	□ 8	□ 8	□ 8	□ 8	□ 8	□ 8
□ 9	□ 9	□ 9	□ 9	□ 9	□ 9	□ 9	□ 9	□ 9	□ 9

← code your 9-digit upYYYYXXXXX student number horizontally on the left, and replicate it below. Write also your first and last name below.

Student Number:

up

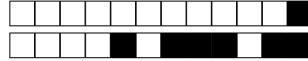
First and Last Name:

.....

Group 1 Cryptography (10 questions)

✓ Question 1.1 ♣ A key exchange protocol does **not** guarantee Perfect Forward Secrecy when:

- Corrupting session keys reveals all past session keys.
- Corrupting long-term keys permits knowing all future session keys.
- Corrupting long-term keys reveals all past session keys.



✓ **Question 1.2 ♣** Remember that a Message Authentication Code (MAC) has the syntax $\text{MAC}(k, m) = t$. To protect a sequence of messages, a MAC must be used:

- In the context of a challenge-response protocol.
- Guaranteeing that each message is only transmitted once.
- Without additional concerns: a MAC allows precisely to protect sequences of messages.
- With confirmation in both directions.

✗ **Question 1.3 ♣** The non-repudiation property is important in the context of message authentication. When using public-key encryption and signatures, what should you do to guarantee this property?

- Sign the plaintext.
- Sign the ciphertext.
- Sign the plaintext and the ciphertext.
- Encryption should never be used when this property is needed.

✓ **Question 1.4 ♣** The *encrypt* operation in a modern symmetric cipher has which 3 input parameters?

- Key, Nonce and Tag.
- Key, Nonce and Message.
- Key, Tag and Message.
- Sender's key, Receiver's Key and Message.

✓ **Question 1.5 ♣** Nowadays, the construction of an authenticated cipher using a symmetric cipher and a Message Authentication Code (MAC) uses the construction:

- Encrypt-And-Mac.
- MAC-Then-Encrypt.
- None of the others.
- Encrypt-Then-MAC.

✗ **Question 1.6 ♣** In modern operating systems, the generation of random numbers for cryptography is generally done using:

- A random number generator fed by another random number generator.
- Dedicated hardware to generate entropy that is directly used by applications.
- A random number generation algorithm that feeds a pseudo-random one.
- None of the other options.

✓ **Question 1.7 ♣** Which is a correct assignment between protection requirements and applications?

<input checked="" type="checkbox"/> 1-B;2-C;3-A.	<input type="checkbox"/> 1-B;2-A;1-C.	B (1) in transit, synchronous communications C (2) in transit, asynchronous communications A (3) at rest	(A) disk encryption (B) https (C) email
<input type="checkbox"/> 1-A;2-B;3-C.	<input type="checkbox"/> 1-C;2-B;3-A.		

✗ **Question 1.8 ♣** In a key distribution system, a Key Distribution Center (KDC) interacts with N agents that trust in the KDC:

- Only to store long-term keys, but the KDC does not know the session keys.
- Because it knows all the keys and could violate the security of the communications.
- Only to store session keys, which are made available when necessary.
- The KDC does not need to be trusted.



✓ **Question 1.9 ♣** The Counter Mode (CTR) mode of operation is:

- A secure construction of a symmetric cipher based on a block cipher.
- An insecure construction of a MAC based on a block cipher.
- A secure construction of a MAC based on a block cipher.

✓ **Question 1.10 ♣** The Electronic Code Book mode of operation is insecure because:

- It reveals patterns in the message.
- Is not insecure and is used in practice.
- Uses a key that is too small.
- Uses an insecure block cipher.

Group 2 Public-Key-Infrastructure (5 questions)

$A \rightarrow B$

✗ **Question 2.1 ♣** When using public-key certificates to transfer digitally-signed information from A to B:

- A has to know and validate a priori B's certificate.
- B has to know and validate a priori A's certificate.
- A and B need to have certificates issued by the same Certification Authority.
- A and B have to exchange and validate certificates a priori.

✓ **Question 2.2 ♣** Recall the PKI Lab. During the execution of a Man-In-The-Middle attack, Alice created a site to emulate the behaviour of a bank's website to obtain Bob's access credentials. Additionally, she created a CA and generated a valid certificate for her site, she performed a cache poisoning by mapping her server to the bank's site on Bob's computer and added her CA to Firefox's valid certificate list on Bob's computer. If Bob decides to check his bank account from his computer on Firefox, will Alice be able to steal his credentials?

- Yes.
- It depends. If a device in the communication route between Bob's computer and the malicious server knows the correct site's IP address, it will override the cache poisoning routing and redirect Bob to the correct site.
- No, Firefox will prevent Bob to access the

target insecure site because the CA authenticating it was added to Firefox's trusted certificates and does not belong to the operative system's certificate repository.

- No, if Bob uses the HTTPS protocol Firefox will warn him that the target site can be malicious.

✓ **Question 2.3 ♣** The pinning of public-key certificates has the goal of allowing:

- A certification authority to account for the falsification of certificates in a given context.
- A software or service provider to account for the falsification of certificates in a given context.
- A certification authority to detect certificates that must be excluded from CRLs.
- A software or service provider to function as a certification authority.

✗ **Question 2.4 ♣** A public-key certificate of a server is signed by the Certification Authority to:

- Allow it to be sent over insecure channels.
- All the other options.
- Guarantee that it was not modified.
- Attest who issued it.



✓ **Question 2.5 ♣** Which of the following is correct in the context of certificate chains?

- A certificate chain may be formed by two certificates: an end-user certificate and an intermediate certification authority certificate.
- A certificate chain may include many certificates that do not belong to certification authorities.
- A certificate chain be formed by three certificates: an end-user certificate, an intermediate certification authority certificate and a root certification authority certificate.
- A certificate chain includes all trusted root certificates.

Group 3 Authentication (4 questions)

✓ **Question 3.1 ♣** Which of the following attacks is **not** a potential theft of session tokens?

- Attacker is a Man-in-the-Middle in the communications between client and server.
- Attacker uses SQL Injection on the server side to invalidate all tokens in use.
- Attacker convinces the user to login with his token.
- Attacker uses Cross-Site Scripting to read hidden tokens in forms in the client side.

✓ **Question 3.2 ♣** In a multi-factor authentication scenario, which of the following additional factors may **not** be considered?

- Challenge-response of a digital signature using a smartcard. ✓
- Challenge-response with the same application in another device. ✓
- User is requested a second and stronger password.
- Application sends a unique code and with a short validity by SMS. ✓

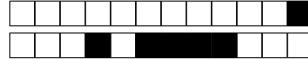
✓ ? **Question 3.3 ♣** Which is **not** a common component of a biometric authentication system?

- Collecting samples. ✓
- Extracting templates. ✓
- Detecting duplicated samples/templates.
- Matching samples with templates. ✓

✓ **Question 3.4 ♣** Which of the following examples of authentication mechanisms permit identifying a user according to each of the principles in the below table?

- ①=password; ②=smartcard; ③=biometry
- ①=mensaje/SMS; ②=cellphone; ③=biometry
- ①=cellphone number; ②=biometry; ③=administrator
- ①=client number; ②=password; ③=user profile

Principle	Mechanism
Something that is known	①
Something that is owned	②
Something that is intrinsic	③



Group 4 Network Security (6 questions)

✓ **Question 4.1 ♣** There are various ways to perform **spoofing attacks in networks**. Which of the following statements is **not** true?

- IRDP spoofing allows announcing a fake router.
- ARP spoofing allows usurping the IP of another machine. ✓
- MAC spoofing allows usurping the MAC address of another machine. ✓
- DNS spoofing allows controlling the translation of IPs into MAC addresses.

✓ **Question 4.2 ♣** Which of the following is a possible way to bypass a firewall?

- Configuring a proxy for a specific application.
- Using a NAT spoofing attack.
- Tunneling packets of a protocol inside another protocol.
- Using NAT to avoid the distinction among internal IPs.

✓ **Question 4.3 ♣** It is possible to increase the security of a network at various levels. Which of the following examples is **not** true?

- At the network layer, e.g. IPSec. ✓
- At the transport layer, e.g. TLS. ✓
- At the application layer, e.g., WhatsApp.
- At the physical layer, e.g. PKI.

✗ **Question 4.4 ♣** Mechanisms such as *firewalls* and *proxies* permit protecting machines connected in a network. Which of the following statements is not **not** true?

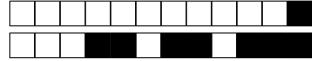
- A proxy is oriented to specific protocols.
- A local firewall may include filters for specific applications. ✓
- A network firewall controls external accesses using specific proxies.
- A firewall is oriented to packets. ✓

? **Question 4.5 ♣** In a rogue DHCP attack:

- A DHCP server can force a client to renew its statically-defined IP.
- A DHCP server can advertise a fake router to a client.
- A DHCP server never returns acknowledgements to client requests.
- A client can perform Denial of Service of a DHCP server.

✓ **Question 4.6 ♣** One of the most basic network attacks is the so-called *wiretapping*. Which of the following statements is **not** true?

- It permits Denial of Service of a communication channel. ✓
- It is typically at the level of the physical/logical layer. ✓
- It does not permit modifying packets in a network.
- It permits eavesdropping/sending packets in a network.



Group 5 Malware and Detection (3 questions)

✓ Question 5.1 ♣ Which of the following is **not** a technique used for detecting botnets?

- Detecting communication traffic with the command and control system. ✓
- Creating a new command and control system that allows deactivating the bots.
- Exposing vulnerable machines to be compromised and monitored. ✓
- Detecting malware in a compromised machine. ✓

✓ Question 5.2 ♣ In what consists the concept of *malvertising*?

- Pages on the *darkweb* that auction malware.
- ads* with wrong information.
- Malware that advertises itself.
- Using *ads* systems to explore vulnerabilities in browsers.

✓ Question 5.3 ♣ Which of the following is an advantage of *host-based detection systems* relatively to *network-based detection systems* or vice-versa?

- A *host-based detection system* does not use system resources while executing.
- A *network-based detection system* permits protecting more systems.
- A *network-based detection system* permits inspecting the contents of encrypted packets.
- A *host-based detection system* is simpler to deploy in a large organization.

Group 6 Transport Layer Security (TLS) (2 questions)

? Question 6.1 ♣ In what consists a *SSL stripping* attack?

- When an attacker connects to the server using HTTP and to the client using HTTPS.
- When an attacker connects to the server using HTTPS and to the client using HTTP.
- When an attacker connects to the server and the client using HTTPS.
- When an attacker is able to remove the TLS/SSL headers from HTTPS packets.

✓ Question 6.2 ♣ When a Certification Authority is corrupted in a TLS/HTTPS connection:

- All the security of past sessions is potentially lost.
- Attacker won't be able to decrypt future messages.
- It is only problematic for versions prior to TLS 1.3.
- It will allow Man-in-the-Middle attacks.

1

Segurança: CIA

Ataque: MOM

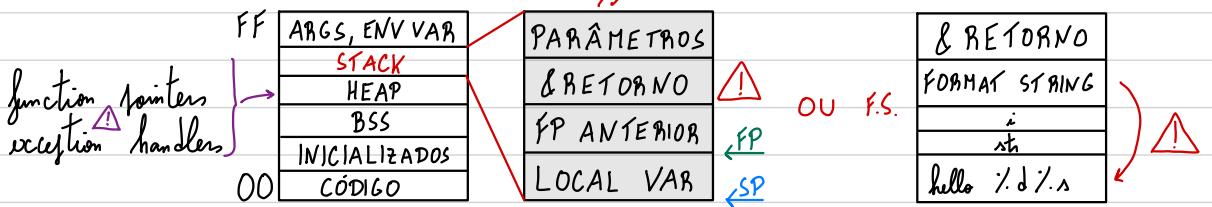
Correção: bom input \Rightarrow bom output

Segurança: man input $\not\Rightarrow$ man output

Binário: seguro VS. insseguro

Gestão de Risco: mais seguro VS. menos seguro

Modelo de Ativaca: objetivos de segurança; adversários; âmbito; pensamento adversário



JIT Spraying: contorna W^X e ASLR - código malicioso em runtime

ROP: retornar f1 - parâmetros + endereço de retorno = f2

Calling f: armazena parâmetros e endereço de retorno

Called f: armazena FP anteriores e aloca variáveis locais

Defesas do SO:

1. DEP / W^X: não impede ROP e cria problemas para JIT
2. ASLR: aleatoriza a localização do código e dos dados por execução
3. K Bouncer: antes de uma syscall, verificar se ret precedido por call

Defesas da Aplicação / Compilador:

1. PIE: executáveis independentes da posição (endereços absolutos)
2. Instrumentação: código que monitora se a funcionalidade é esperada
3. Canários: depois das variáveis locais - ler, contornar, escrever/ignorar, adicionar
4. Tag de Memória: limitar apontadores a regiões de memória
5. CFI: determinar um grafo de controle de fluxo na compilação e verificá-lo na execução

- Defesas da Linguagem: Segurança da Memória e Segurança do Programa
1. Programação Defensiva
 2. Análise de Tinta
 3. Análise de Tempo Constante
 4. Compilação Segura

Defesas do Hardware: TPM; TEE; UEFI Secure Boot

- | | |
|---------------------------|-----------------------------|
| 1. Economia de Mecanismo | 5. Mínimo Mecanismo Comum |
| 2. "Fail-Safe Defaults" | 6. Mínimo Privilegio |
| 3. Design Aberto | 7. Separação de Privilegios |
| 4. Defesa em Profundidade | 8. Mediação Completa |

- Controle de Acessos
- | |
|---|
| 1. Matriz: todas as combinações (ator, recurso, operação) |
| 2. Lista: por recurso - "inbound" / "o que" |
| 3. Capacidades: por ator - "outbound" / "quem" |
| 4. Papéis: atores ↔ papéis ↔ permissões |
| 5. Atributos: atores e recursos têm atributos |

Um processo executa com o UID de quem o lanza (Kernel → UID = 0) ou do dono do arquivo, se setuid = 1 — (EUID, RUID, SUID)
PERMISSÕES LANçOU ANTERIOR

- | | |
|---|--------------|
| 1. "Aí Gap": confinamento ao nível de hardware | 2. Processos |
| 3. SFI + SCI: monitor de referência mede as syscalls e isola processos | |
| 4. Máquinas Virtuais: Hypervisor permite partilhas HW | 5. Sandboxes |
| 6. Containeres: partilham o Kernel do SO anfitrião e isolam componentes do utilizador | |

- SOP: pode fazer pedidos para outra origem, mas não analisa respostas
- CORS: pedidos simples (sem efeitos) ou "pre-flighted" (com efeitos)
- XSS: refletido (atacantes), armazenado (serviços) ou DOM (utilizadores)

Encriptação: confidencialidade

2

MACs: integridade & autenticidade

Simétrica: mesma chave dos dois lados - cifra de blocos é invertível AES
ECB: $c_i \leftarrow E(k, p_i)$ - inseguro CBC: tem IV
A chave é gerada a partir de uma distribuição aleatória uniforme

Função de Hash: colisões podem ser encontradas com trabalho $\sqrt{2^n}$ SHA
Merkle-Damgård: MD4, MD5, SHA1, SHA256, SHA512 Entra: SHA3

MACs: Autenticação Simétrica $t \leftarrow MAC(k, m) \approx$ assinaturas digitais
MD: $MAC(k, m) = H(k \parallel M) \rightarrow H(k \parallel M \parallel pad \parallel M)$ HMAC CMAC

Função Universal de Hash: desde que só se autentique uma mensagem
 $\Rightarrow UH(k_1, m) \oplus PRF(k_2, \text{Nonce})$ - Construção de Wegman-Carter

AEAD: impede repetição/remoção/omissão de mensagens - metadados públicos

1. Encrypt-and-MAC 2. MAC-then-Encrypt 3. Encrypt-then-MAC
GCM OCB SIV

Centro de Distribuição de Chaves: N chaves e 1 de longo-prazo com todos
partilha de chaves simétricas de longo-prazo \rightarrow encriptação de chave pública
não-repúdio \rightarrow assinaturas digitais \rightarrow acordos de chave + AD

Encriptação de Chave Pública: $c \leftarrow E(gk, t), t \leftarrow D(sk, c)$ RSA

Autenticação Assimétrica de Mensagem: $s \leftarrow S(sk, t), t/l \leftarrow V(pk, t/s)$

Assinaturas Digitais garantem autenticidade, integridade e não-repúdio

"Perfect Forward Secrecy": chaves longo-prazo comprometidas não comprometem senhas

$$DH: K = (g^\gamma)^n = g^{\gamma n} = g^{n\gamma} = (g^n)^\gamma = K$$

Protocolos de acordo de chave protegem autenticidade de chaves simétricas

- Criptografia de Chave Pública pressupõe chaves públicas autênticas + MitM
- CA: valida identidade e chave pública - certificado assinado válido
- certificados de raiz não auto-assinados e implicitamente confiados
- Se CA_A assina CA_B , confiança (B) \ll confiança (A)
- Listas de Revogação de Certificados: "black-list" periódica
- 1. TSL: "white-list" 2. OCSP: serviço seguro 3. Pinning: browser/apps

Autenticação: se um utilizador pode ter acesso a um sistema

Autorização: que ações um utilizador pode realizar no sistema

Desafio-Resposta: desafio escolhido de modo a impossibilitar reutilização Nonce

- TLS/SSL: entre aplicação e TCP - cada conexão associada com uma sessão
1. Record: integridade e confidencialidade das mensagens com a chave acordada
 2. Handshake: estabelecimento da chave criptográfica - via sessão AUTENTICAÇÃO MÚTUA
 3. Change Cipher Spec: mensagem única que estabelece as especificações acordadas
 4. Alerta: pode provocar avisos ou terminar conexões - segurança e especificidade
 5. Heartbeat: pinga regularmente e impede a conexão de terminar

IPSec: na camada de rede - em firewalls/routers, transparente às apps.

↳ autenticação e integridade para as mensagens; proteção contra IP spoofing

IKE: 1. IKE SA \approx sessão (parâmetros) 2. IPsec SA \approx conexão (revisão)

ESP: autenticação/integridade e confidencialidade - para além do cabeçalho

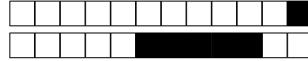
AH: só autenticação/integridade (não confidencialidade) - algum cabeçalho

Modo Transporte: adiciona informação/segurança ao pacote original F2F H2H

Modo Tunel: proteger o pacote original, encapsulando-o num novo pacote IP

(D)DoS: inundação UDP; Echo-Changen; Smurf; SYN Flooding; Amplificação DNS

Filtros de Pacotes: camada de rede ... Stateful: camada de transporte



This is a multiple-choice test that will be corrected automatically. Please follow these rules:

- No pencil or light-coloured pens. Mark your answer using only **blue** and **black** pen.
- Erased boxes are not detected automatically. Check only inside boxes and be generous on ink.
- You may underline text or take notes on the sides. Only boxes matter for automatic correction.

The test is marked for 20 points and contains 30 questions, each with 4 options. **Only one of those options is accepted as correct, which could be an option indicating all options are correct.**

You can check one or two choices per question. The scoring for each question is as follows:

- One checked correct answer (100%).
- One checked incorrect answer (-20%).
- No checked answers (0%).
- Two checked answers, one correct (50%).
- Two checked answers, none correct (-20%).
- More than two checked answers (-20%).

□ 0	□ 0	□ 0	□ 0	□ 0	□ 0	□ 0	□ 0	□ 0	□ 0
□ 1	□ 1	□ 1	□ 1	□ 1	□ 1	□ 1	□ 1	□ 1	□ 1
□ 2	□ 2	□ 2	□ 2	□ 2	□ 2	□ 2	□ 2	□ 2	□ 2
□ 3	□ 3	□ 3	□ 3	□ 3	□ 3	□ 3	□ 3	□ 3	□ 3
□ 4	□ 4	□ 4	□ 4	□ 4	□ 4	□ 4	□ 4	□ 4	□ 4
□ 5	□ 5	□ 5	□ 5	□ 5	□ 5	□ 5	□ 5	□ 5	□ 5
□ 6	□ 6	□ 6	□ 6	□ 6	□ 6	□ 6	□ 6	□ 6	□ 6
□ 7	□ 7	□ 7	□ 7	□ 7	□ 7	□ 7	□ 7	□ 7	□ 7
□ 8	□ 8	□ 8	□ 8	□ 8	□ 8	□ 8	□ 8	□ 8	□ 8
□ 9	□ 9	□ 9	□ 9	□ 9	□ 9	□ 9	□ 9	□ 9	□ 9

← code your 9-digit upYYYYXXXXX student number horizontally on the left, and replicate it below. Write also your first and last name below.

Student Number:

up									
----	--	--	--	--	--	--	--	--	--

First and Last Name:

.....

Group 1 Introduction (4 questions)

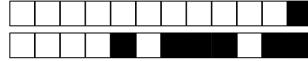
✓ **Question 1.1** ♣ Which of the following options is **not** usually categorised as a **threat**?

- JavaScript performing heap spraying. **ATTAQUE** Company that competes in the same area.
 Activist that disagrees with company policy. Natural catastrophe.

✓ **Question 1.2** ♣ Which of the following concepts is **not** associated with the binary model of security?

- Formal definition. Security proof. Resilience. Cryptography.

GESTÃO
DE RISCO



✓ **Question 1.3 ♣** Identify a typical reason for an attacker to compromise a user's machine.

- Playing games on the user's machine.
 As part of a supply-chain attack.
 Stealing user's credentials.

✓ **Question 1.4 ♣** The rows and columns of a risk assessment matrix correspond to:

- Level of severity/-
Cost of mitigation
 Probability of oc-
currence/Level of
severity
 Level of severi-
ty/Accuracy of in-
formation
 Probability of oc-
currence/Cost of
mitigation

Group 2 Software Security (10 questions)

✓ **Question 2.1 ♣** During the buffer overflow lab, you explored a buffer overflow vulnerability using *shellcode*. Which technique could be used to make it impossible run *shellcode* in the stack?

- Address Space
Layout Randomi-
sation
 Stack Canary
 Data Execution
Prevention
 None of the other
choices

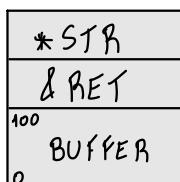
✓ **Question 2.2 ♣** In the buffer overflow lab, you created a malicious input that overwrites the return address of the vulnerable function with the address of the *shellcode*. Which protection had to be disabled during compilation?

- Stack Canary
 Data Execution
 Prevention
 None of the other
choices
 Address Space
Layout Randomi-
sation

✓ **Question 2.3 ♣** Certain format strings can make a program crash (with very high probability). Which of the following *printf* commands is very likely to crash the program?

- int n=1; printf("%n%f",&n);
 int n=1; printf("%s%d",n);
 int n=0; printf("%f%d",n);
 int n=0; printf("%s%x",&n);

✓ **Question 2.4 ♣** Remember the program on the right with a stack overflow vulnerability from the buffer overflow lab.



Assume that *strlen(str)=400* and that we have a *shellcode* with 20 bytes. Consider a 32-bit architecture where valid addresses are aligned with multiples of 4. Which of the following input strings would **not** yield a successful attack?

- Place the *shellcode* at *str[380]*, write
&buffer+300 to all positions from 0 to 100
and fill all remaining positions with NOPs.
 Place the *shellcode* at *str[380]*, write
&buffer+300 to all positions from 0 to 200

```
int bof(char *str)
{
    char buffer[100];
    strcpy(buffer,str);
    return 1;
}
```

- and fill all remaining positions with NOPs.
 Place the *shellcode* at *str[0]* and write
&buffer to all positions from 20 to 400.
 Place the *shellcode* at *str[380]* and write
&buffer+380 to all positions from 0 to 380.



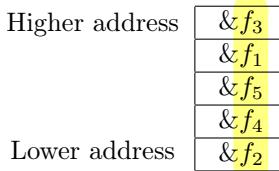
✓ Question 2.5 ♣ Just-In-Time (JIT) compilation is problematic in terms of security mainly because:

- It violates the W^X principle. It may remove protection mechanisms. It inhibits static analysis of the program. It may introduce vulnerabilities.

✓ Question 2.6 ♣ In a use-after-free attack, the crucial information that is usually overwritten is:

- Pointers to functions. Exception handlers. Virtual function tables. All of the other choices.

✓ Question 2.7 ♣ Suppose you intend to use Return Oriented Programming (ROP) to execute the instructions of functions f_1, f_2, f_3, f_4, f_5 (in some order) and you have placed their addresses in stack as illustrated below. In which order are they executed?



- 2,4,5,1,3. ROP does not use the stack. ROP can return to at most one function.

✓ Question 2.8 ♣ The code `int n; printf("%d",n);` has a common vulnerability catalogued in the CERT Secure C Coding Standard. It is a violation of:

- Constant-time security Control-flow integrity Program safety Memory safety

✓ Question 2.9 ♣ In heap spraying, the attacker does the following:

- Uses a heap buffer overflow to fill the entire memory with shellcode. None of the other choices. Creates multiple copies of the shellcode in the heap. Uses a programming technique known as spraying to control the heap.

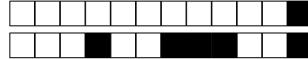
✓ Question 2.10 ♣ A shadow stack is a security mechanism that:

- Inverts the stack to make it harder to modify. Cryptographically denies stack modification. Uses two copies of the stack. Hides the stack from the attacker.

Group 3 Systems Security (10 questions)

✓ Question 3.1 ♣ Remember the program `int main() { system("ls"); return 0; }` from the environment variable and `setuid` program lab. Which of the following statements is **not** true?

- None of the other choices is false. Another executable in the PATH may be executed instead of the system /bin/ls command. If the program runs with root `setuid`, it may list all files in the current directory.
- Another executable called ls in the working directory may be executed instead of the system /bin/ls command.



✓ **Question 3.2 ♣** There are several confinement approaches in operating systems. Which is true?

- A container allows a guest OS to run independently of a host OS.
- A sandbox allows applications to run independently of an OS.
- A container allows an OS to run indepen-
- dently from the underlying hardware through the use of virtual machines.
- An hypervisor allows an OS to run independently from the underlying hardware through the use of virtual machines.

✓ **Question 3.3 ♣** Which is **not** a systems security principle that we have studied in the classes?

- Minimum privilege
- Open design
- Complete mediation
- Economy of privilege

✓ **Question 3.4 ♣** In a UNIX operating system, a process can change its permissions at runtime, by controlling the Effective User ID (EUID). The `setuid` primitive allows:

- The root user to change the permissions of the process into a non-privileged user, without the possibility to regain root privileges.
- The root user to change the permissions of the process into a non-privileged user, without the possibility to regain root privileges.
- The root user to change the permissions of the process into a non-privileged user, with the possibility to regain root privileges.
- A user to change the permissions of the process into another user, without the possibility to gain root privileges.

✓ **Question 3.5 ♣** In Linux, Secure Computing Mode, `seccomp` for short, is an important reference monitor for system calls. Which of the following sentences is **not** true?

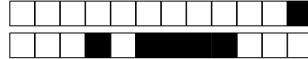
- All options are true.
- The `seccomp` reference monitor kills any process that violates its permissions.
- A program in `seccomp` cannot open new files.
- A program in `seccomp` cannot fork child processes.

✓ **Question 3.6 ♣** For efficiency reasons, the address space of the kernel is not completely independent from that of user processes due to *kernel mapping*. Which is **not** an implemented countermeasure by UNIX systems to prevent a user program from accessing kernel memory?

- The kernel memory management abides by the W^X restrictions.
- Kernel Address Space Layout Randomisation (KASLR) loads the kernel to a random address each time the system boots.
- Accessing kernel memory requires special kernel privileges, divided into distinct kernel capabilities.
- The kernel memory is placed below the stack program memory, to make sure that a stack overflow cannot reach the kernel.

✓ **Question 3.7 ♣** Which is **not** an important assumption made by a modern operation system to guarantee that user files are kept secure?

- The hardware has not been compromised.
- The user never installs malicious applications with administrator privileges.
- No attacker has physical access to memory/disk when the system reboots or hibernates.
- The bootloader has not been compromised.



✓ **Question 3.8 ♣** Which of the following is an advantage of role-based access control?

- Unlike access control lists, it allows defining the permissions that each actor has on every resource.
- It is a more general form than access control lists, capabilities or attribute-based access control.
- It allows decoupling access control policies into a relation between actors and roles and a relation between resources and roles.

✓ **Question 3.9 ♣** In the environment variable and `setuid` program lab, we have experimented with environment variables and `setuid` programs. Which of the following sentences is **not** true?

- If the shell detects that it is being run under a `setuid` process, it may drop its privilege.
- The `system` function allows calling a shell function within a program with the program's environment.
- When a process forks a child process, it passes

on its environment, excluding some critical variables if the fork is a system call.

- When the shell forks a child process, it passes on its environment, excluding some critical variables if the fork has a different effective user id.

✓ **Question 3.10 ♣** The UNIX filesystem permissions are an important element to ensure security of critical administrative operations. Which of the following sentences is true?

- The group of a file can only be changed by the root user.
- Only the owner of a file or the root user can change its permissions.
- The owner of a file can only be changed by the owner itself.
- Not even the root user can change the permissions of files he does not own.

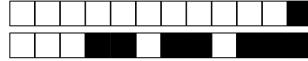
Group 4 Web Security (6 questions)

✓ **Question 4.1 ♣** According the Same-Origin Policy (SOP), how can a web developer prevent a cookie from being sent by a HTML link from origin A that makes a request to origin B?

- Defining cookies as being `HTTPOnly`.
- Defining `SameSite=Lax`.
- Defining `SameSite=Strict`.
- Using only secure (`HTTPS`) cookies.

✓ **Question 4.2 ♣** Which is **not** an example of a Cross-Site Request Forgery (CSRF) attack?

- A hidden JavaScript script in a malicious site tries to log in into the users' web router in his home network, using default credentials, to install spyware.
- The source of an image in a malicious site sends a login request in a legitimate site with the attacker's credentials.
- The submit button of a HTML form in a malicious site sends a money wiring request to the legitimate site of a bank where the user is logged in.
- The submit button of a HTML form in a malicious site sends, alongside a request to a legitimate site, a malicious JavaScript whose execution steals the user's cookies.



✓ **Question 4.3 ♣** According the Same-Origin Policy (SOP), which of the following is allowed?

- JavaScript code in frame with origin A can manipulate the HTML code of page with origin B.
- JavaScript code in page with origin A can send a GET request to page with origin B.
- JavaScript code in page with origin A can send a POST request to frame with origin B.
- HTML code in frame with origin A can inspect the HTML code of sub-frame with origin B.

✓ **Question 4.4 ♣** Consider the following SQL query, written in some server-side library, that is vulnerable to SQL injection. How could we prevent a SQL injection attack on this query?

```
uName = getRequestString("username");
uPass = getRequestString("userpassword");
sql = 'SELECT_*_FROM_Users_WHERE_Name=' + uName + '"_AND_Pass=' + uPass + '';
```

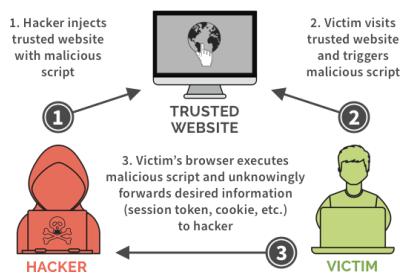
- Using Trusted Types, a feature of SQL engines that enforces input sanitisation.
- Not using string concatenation, and filling the `username` and `userpassword` arguments using prepared statements.
- The client-side application generates a fixed SQL query (for `username` and `userpassword`), sent securely via HTTPS.
- Changing the order of the clauses for the Name and Pass fields within the WHERE clause.

✓ **Question 4.5 ♣** It is **not** recommended for HTTP GET requests to perform server side-effects because, by default:

- They are allowed across any origin.
- An attacker can always see its response.
- An attacker can steal cookies.
- They are not recorded in the server logs.

✓ **Question 4.6 ♣** The figure illustrates a typical Cross-Site Scripting (XSS) attack. It is an example of:

- A DOM-based XSS attack.
- A reflected XSS attack.
- A stored XSS attack.
- A subtype called a Cross-Frame Scripting (XFS) attack.





This is a multiple-choice test that will be corrected automatically. Please follow these rules:

- Mark your answer using only **blue** and **black** pen. No pencil or light-coloured pens.
- Check only inside each box and be generous on ink. Erased boxes will not be detected automatically.
- Only the boxes matter for the automatic correction. You may underline text or take notes on the sides.

The test is marked for 20 points. There are 30 questions in total, each with 4 options. Each question is worth 20/30 points. Students can check one or two choices per question. The scoring for each question is as follows:

- One checked correct answer (100%).
- One checked incorrect answer (-20%).
- No checked answers (0%).
- Two checked answers, one correct (50%).
- Two checked answers, none correct (-20%).
- More than two checked answers (-20%).

<input type="checkbox"/> 0									
<input type="checkbox"/> 1									
<input type="checkbox"/> 2									
<input type="checkbox"/> 3									
<input type="checkbox"/> 4									
<input type="checkbox"/> 5									
<input type="checkbox"/> 6									
<input type="checkbox"/> 7									
<input type="checkbox"/> 8									
<input type="checkbox"/> 9									

← code your 9-digit upYYYYXXXXX student number horizontally on the left, and replicate it below. Write also your first and last name below.

Student Number:

up	<input type="checkbox"/>							
----	--------------------------	--------------------------	--------------------------	--------------------------	--------------------------	--------------------------	--------------------------	--------------------------

First and Last Name:

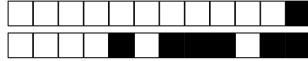
.....

Group 1 Introduction (4 questions)

✓ **Question 1.1 ♣** Which of the following concepts is **not** associated with the risk management approach to security?

- Security proof. **BINARIO**
- Cost/benefit analysis.

- Mitigation.
- Threat analysis.



✓ **Question 1.2 ♣** Which of the following options is **not** usually categorised as an exploit?

- JavaScript performing heap spraying.
- Overflow in an integer computation. *VULNERABILIDADE*
- Text filled into edit box that contains JavaScript.
- Message that triggers an array access out of bounds.

✓ **Question 1.3 ♣** Which of the following is **not** a common reason for an attacker to compromise a server that answers to requests from a large number of clients/users?

- Geo-political and strategic motivation.
- To make the machine part of a botnet.
- As part of a supply-chain attack.
- To cause a data breach.

✓ **Question 1.4 ♣** Pick the **incorrect statement** or indicate that all are correct.

- A trust model describes which assumptions we can rely on to build security.
- All are correct.
- A security goal is stated in terms of avoiding loss of value to an asset.
- A security policy describes how security mechanisms are used to realize a security model.

Group 2 Software Security (10 questions)

✓ **Question 2.1 ♣** In the buffer overflow lab, you created a malicious input that overwrites the return address of the vulnerable function with an address within your buffer where you stored *shellcode* to be executed. Which security technique had to be disabled during compilation to allow you to overwrite the return address of the vulnerable function?

- Address Space sation
- Layout Randomi
- Prevention
- Stack Canary
- None of the other choices

✓ **Question 2.2 ♣** Certain format strings can make a program crash (with very high probability). Which of the following `printf` commands is very likely to crash the program?

- `int n=1; printf("%s%d",n);`
- `int n=0; printf("%f%d",n);`
- `int n=0; printf("%s%x",&n);`
- `int n=1; printf("%n%f",&n);`

✓ **Question 2.3 ♣** Recall the buffer overflow lab that you have performed, where a buffer in a local variable receives an untested input. For the exploitation strategy where the injected code (*shellcode*) is placed at the end of the input bytes, you should:

- In the place of the vulnerable function's return address, write another address pointing to a lower place in the stack.
- In the place of the vulnerable function's return address, write another address pointing to a higher place in the stack.
- In the place of the vulnerable function's return address, write the address of the buffer.
- None of the other options.



+1/3/58+

✓ **Question 2.4 ♣** *Information-flow analysis* encompasses a class of program analysis techniques beyond safety. The C-like program on the right illustrates an example of:

- Static taint analysis.
- Control-flow integrity.
- Dynamic taint analysis.
- Constant-time analysis.

```
int printf(untainted char *fmt, ...);
tainted char *fgets(...);

tainted char *name = fgets(..., network_fd);
printf(name); // FAIL: tainted not untainted
```

✓ **Question 2.5 ♣** Suppose you intend to use Return Oriented Programming to execute the instructions of functions f_1, f_2, f_3, f_4, f_5 (in some order) and you have placed their addresses in the stack as illustrated below. Which of the following functions could take parameters and the ROP strategy would still work without any changes to the part of the stack shown in the figure?

- Function f_3 .
- Function f_2 .
- All functions could take parameters.
- It is not possible to pass any parameters.

High address	& f_3
	& f_1
	& f_5
	& f_4
Low address	& f_2

✓ **Question 2.6 ♣** Recall what you have studied about the classical configuration of the stack region managed by a function f that is called by a function g . Indicate which of the following options is correct for the positioning of the following data pieces: ① local variables of f , ② frame pointer of g , ③ parameters passed by g , ④ return address back to g .

- ③ \Rightarrow A, ② \Rightarrow B, ① \Rightarrow C, ④ \Rightarrow D
- ① \Rightarrow A, ③ \Rightarrow B, ② \Rightarrow C, ④ \Rightarrow D
- ④ \Rightarrow A, ③ \Rightarrow B, ① \Rightarrow C, ② \Rightarrow D
- ③ \Rightarrow A, ④ \Rightarrow B, ② \Rightarrow C, ① \Rightarrow D

High address	PARA A 3
	RET B 4
	FP C 2
	L D 1

✓ **Question 2.7 ♣** Consider the code below, which introduced a critical vulnerability in openSSH. An attacker that controls the value of nresp can cause a memory management problem because:

- It can lead to not allocation enough memory.
 - It can lead to allocating too much memory.
 - It can force the if statement not to execute.
 - None of the other choices.
- ```
nresp = packet_get_int();
if (nresp > 0) {
 response = xmalloc(nresp*sizeof(char *));
 for (i = 0; i < nresp; i++)
 response[i] = packet_get_string(NULL); }
```

✓ **Question 2.8 ♣** A canary built from string terminating bytes offers better security than a random canary because:

- They can be generated much more efficiently.
- It makes it more difficult to overwrite the stack.
- Random bytes can be all 0.

✓ **Question 2.9 ♣** In a heap overflow attack, the crucial information that is usually overwritten is:

- Return addresses.
- Pointers to functions.
- None of the other choices.
- Frame pointers.



**Question 2.10 ♣** One can bypass Address Space Layout Randomization (ASLR) protections by:

- Trial and error.
- All of the other choices.
- Jumping to system code using Return Oriented Programming.
- Extracting addresses using other vulnerabilities.



## Group 3 Systems Security (10 questions)

**Question 3.1 ♣** Remember the environment variable and `setuid` program lab. The Linux file system associates a owner, a group, and 12 permission bits with each entry. For a file with the 12th bit (the Set-UID bit) active, it can be executed:

- Only by users whose set (owner, group, others) in the permission bits, has the executable bit active.
- Only by the owner or by a user belonging to the group.
- Only by a user belonging to the group.
- Only by the owner.



**Question 3.2 ♣** Many modern operating systems require system support for an hardware component called a Trusted Platform Module (TPM). Which is the main rationale for that requirement?

- To protect the user login process.
- To guarantee that the user only installs official software.
- To store user's cookies when browsing the web.
- To prevent malicious bootloaders from compromising the operating system's boot process.



**Question 3.3 ♣** Which is **not** a systems security principle that we have studied in the classes?

- Separation of privilege
- Economy of mechanism
- Closed design



**Question 3.4 ♣** Under Linux, Docker crucially relies on `seccomp-bpf` to confine containers. Select the **incorrect** choice or mark that all choices are correct.

- All choices are correct.
- If an attacker has access to certain system calls inside a container, it can exploit Docker to acquire root in the host OS.
- Using the default Docker `seccomp-bpf` filters, an attacker that acquires root in the container does not directly acquire root in the host OS.
- Users can manually configure `seccomp-bpf` filters to block various system calls inside the container.



**Question 3.5 ♣** The Android operating system is a particular distribution built on top of Linux. Which is **not** an additional domain-specific restriction that Android implements?

- There is no root user, to prevent applications from escalating privileges.
- It implements a form of Mandatory Access Control, so that no application can change its permissions.
- It isolates different applications by registering each application with a different UNIX user.
- It enforces Manifest Permissions per application, to restrict its system capabilities.



✓ **Question 3.6 ♣** Virtual machines are commonly used as a security mechanism. Which of the following sentences is **not** true?

- A disadvantage of virtual machines is that malware may exploit the virtualisation layer to remain undetected.
- An advantage of virtual machines is that acquiring root in the virtual machine does not grant root in a host operating system.
- A disadvantage of virtual machines is that malware may detect that it is being virtualised.
- An advantage of virtual machines is that software may be transparently executed as in a non-virtualised operating system.

✓ **Question 3.7 ♣** Which fundamental mechanism does a UNIX operating system have in place to enforce isolation?

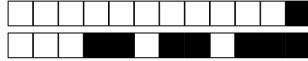
- System call interposition, by virtualising the address space of different user processes, monitoring all virtual address translations.
- System call interposition, by preventing different user processes to communicate via system calls.
- Software fault isolation, by virtualising the address space of different user processes, monitoring all virtual address translations.
- Software fault isolation, by ensuring that the control-flow of user processes never accesses invalid virtual memory regions.

✓ **Question 3.8 ♣** For efficiency reasons, the address space of the Linux kernel is not completely independent from that of user processes due to *kernel mapping*. Which of the following is correct?

- Part of the kernel memory space is mapped into the memory space of each user process, to reduce the number of cache misses.
- Since the discovery of speculative execution attacks such as Meltdown and Spectre, Linux no longer uses *kernel mapping*, as it violates the principle of *separation of privilege*.
- Part of each user process's memory is mapped into the kernel memory space, to reduce the size of the address translation tables.
- Part of the kernel memory space is mapped into the memory space of each user process, to speed up system calls.

✓ **Question 3.9 ♣** In the environment variable and `setuid` program lab, we have experimented with environment variables and `setuid` programs. Which of the following sentences is **not** true?

- If the shell detects that it is being run under a `setuid` process, it may drop its privilege.
- The `system` function allows calling a shell function within a program with the program's environment.
- When a process forks a child process, it passes on its environment, excluding some critical variables if the fork is a system call.
- When the shell forks a child process, it passes on its environment, excluding some critical variables if it has a different effective user id.



✓ **Question 3.10 ♣** The UNIX filesystem can be seen as an instance of:

- Access control lists, since only the owner of each file (or root) may read, write or execute it.
- Role-based access control, since each file has an access control list for three fixed roles (owner, group, others) and the association between users and groups may be modified independently.
- Capability lists, since each file enumerates the permissions for all the users who may read, write or execute it.
- Attribute-based access control, since users may change their group membership without the need to change file permissions.

## Group 4 Web Security (6 questions)

✓ **Question 4.1 ♣** Cookies may be used for various purposes. Name the **incorrect** one below.

- Tracking user activity.
- Personalising user experience.
- Encrypting user communication.
- Managing user sessions.

✓ **Question 4.2 ♣** Which sentence best characterises a Cross-Site Request Forgery (CSRF) attack?  
“When a malicious origin can send requests to a server in another origin ...” :

- ... and measure side-channels such as response time.
- ... where the user is already logged in.
- ... and read its response.
- ... and trigger a side-effect on the server.

✓ **Question 4.3 ♣** According the Same-Origin Policy (SOP), which of the following is allowed?

- JavaScript code in a page from origin A can inspect the HTML code of a frame from origin B.
- HTML code in a page from origin A can send a POST request to a page from origin B.
- HTML code in a page from origin A can send and read the response of a GET request to a page from origin B.
- JavaScript code in a frame from origin A can exchange data with a frame from origin B.

✓ **Question 4.4 ♣** Consider the following SQL query, written in some server-side library, that is vulnerable to SQL injection. Which of the following statements is true?

```
uName = getRequestString("username");
uPass = getRequestString("userpassword");
sql = 'SELECT_*_FROM_Users_WHERE_Name=' + uName + '_AND_Pass=' + uPass + '';
```

- A SQL injection attack would not be possible if the clauses for the Name and Pass fields did not enclose strings with double quotes ("").
- A SQL injection attack may be able to delete the USERS table.
- A SQL injection attack that bypasses authentication is only possible because the clause for the Name field appears before the clause for the Pass field.
- A SQL injection attack will only be able to read existing data from the USERS table.



✓ **Question 4.5 ♣** Which assignment can be seen as a valid analogy in the table below?

- ①=Pages,②=Cookies,③=HTTP,④=Frames
- ①=Pages,②=HTML,③=JavaScript,④=Popups
- ①=DOM,②=Cookies,③=iFrames,④=Fetch
- ①=Frames,②=DOM,③=Images,④=Sub-frames

| Systems Security | Web Security |
|------------------|--------------|
| Processes        | ①            |
| Files            | ②            |
| Sockets          | ③            |
| Sub-processes    | ④            |

✓ **Question 4.6 ♣** A classical protection against Cross-Site Scripting (XSS) attacks is for the site to adopt a Content Security Policy (CSP). Which of the following statements is true?

- DOM-based XSS attacks cannot be prevented with a CSP because they occur on the server-side when processing a user request.
- Preventing reflected XSS attacks requires both CSP and SRI (Subresource Integrity).
- XSS attacks using inline scripts are blocked by a default CSP.
- Stored XSS attacks cannot be prevented with a CSP because the malicious payloads are already stored in the server.



This is a multiple-choice test that will be corrected automatically. Please follow these rules:

- Mark your answer using only blue and black pen. No pencil or light-coloured pens.
- Check only inside each box and be generous on ink. Erased boxes will not be detected automatically.
- Only the boxes matter for the automatic correction. You may underline text or take notes on the sides.

The test is marked for 20 points. There are 30 questions in total, each with 4 options. Each question is worth 20/30 points. Students can check one or two choices per question. The scoring for each question is as follows:

- One checked correct answer (100%).
- One checked incorrect answer (-20%).
- No checked answers (0%).
- Two checked answers, one correct (50%).
- Two checked answers, none correct (-20%).
- More than two checked answers (-20%).

0  0  0  0  0  0  0  0  
 1  1  1  1  1  1  1  1  
 2  2  2  2  2  2  2  2  
 3  3  3  3  3  3  3  3  
 4  4  4  4  4  4  4  4  
 5  5  5  5  5  5  5  5  
 6  6  6  6  6  6  6  6  
 7  7  7  7  7  7  7  7  7  
 8  8  8  8  8  8  8  8  8  
 9  9  9  9  9  9  9  9

← code your 9-digit upYYYYXXXXX student number horizontally on the left, and replicate it below. Write also your first and last name below.

Student Number:

up **2 0 2 1 0 8 7 4 4**

First and Last Name:

**MANUEL NETO**

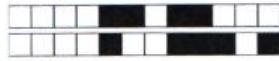
## Group 1 Introduction (4 questions)

✓ Question 1.1 ♣ Which of these statements is more precise? **Adversarial thinking** is about:

- 1/1
- Integrating secure development practices into the way you build software.
  - Analysing and balancing the risk / cost of possible attacks.
  - Defining security from the perspective of the various attackers.
  - Questioning the security assumptions and understanding how to break a system.

✓ Question 1.2 ♣ Which is **not** a common motivation for malware to compromise a user machine?

- 1/1
- To use the machine for a cyber theft ring.
  - To demand a ransom on the data.
  - To perform a typo-squatting attack.
  - To steal user credentials.



✓ Question 1.3 ♣ Which of the following properties defines a zero-day vulnerability?

- 1/1
- It was mitigated zero days after being found.
  - No one in the world knows it exists.
  - Its exploit was never publicly known before.
  - Some users do not know it exists.

✓ Question 1.4 ♣ Security is often described in terms of three broad axes, them being: CIA

- 1/1
- Security / Exposure / Usability
  - Confidentiality / Functionality / Usability
  - Confidentiality / Integrity / Availability
  - Confidentiality / Integrity / Authentication

## Group 2 Software Security (10 questions)

✓ Question 2.1 ♣ Data Execution Prevention, also called W^X, is a defense at the layer of:

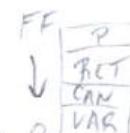
- 1/1
- The running application.
  - The operating system.
  - The programming language.
  - The hypervisor hardware.

✓ Question 2.2 ♣ There are various countermeasures to avoid attackers from usurping control of an application. Which of these statements is **not** true?

- 0.2/1
- Position-independent executables ensure that the relative positions of generated code can be randomised on each execution.
  - Memory tagging checks in hardware that pointers to allocated memory regions do not overflow.
  - A shadow stack keeps copies of control structures and checks at program run-time that the original stack has not been compromised.
  - Control-flow integrity enforces that direct calls to system calls in the program follow a pre-determined control-flow graph.

✓ Question 2.3 ♣ If the stack grows from high to low addresses, a stack canary is usually placed:

- 1/1
- Immediately below the return address.
  - Immediately above the return address.
  - It is a global variable, not stored in the stack.



✓ Question 2.4 ♣ This code introduced a critical vulnerability in openSSH. An attacker controlling nresp can cause a memory management problem because it can trigger:

```
nresp = packet_get_int();
if (nresp > 0) {
 response = xmalloc(nresp * sizeof(char*));
 for (i = 0; i < nresp; i++)
 response[i] = packet_get_string(NULL); }
```

- 1/1
- An integer overflow and a subsequent heap buffer overflow.
  - An integer overflow and a subsequent stack buffer overflow.
  - An heap overflow and a subsequent integer buffer overflow.
  - An integer overflow and a subsequent use after free.

✓ Question 2.5 ♣ Why is it more complicated to simulate the sequential call to multiple functions with parameters in a return-oriented programming attack? Because:

- 1/1
- We are writing over the stack frames of the calling functions.
  - The caller to each called function does not clean the passed parameters from the stack.
  - Each called function does not clean its parameters from the stack as it should.
  - We can only call functions with arguments already present in the original stack.



✓ **Question 2.6 ♣** Why is a JIT spraying attack particularly effective?

- 1/1
- Because the JIT compiler disables address randomisation and an attacker can predict the addresses of the injected *shellcode*.
  - Because an attacker can reuse code generated "just-in-time" to build a *shellcode* and "spray" the memory to make it executable.
  - Because the code is compiled "on-the-fly" with no stack protections or run-time instrumentation.
  - Because the JIT compiler relaxes Write Or Execute restrictions and an attacker can inject multiple copies of the *shellcode*.

✓ **Question 2.7 ♣** Recall the format strings lab and consider a program with a simple `printf` call and the format string controlled by user input. Select the achievable exploit or mark all other options.

- 1/1
- All of the other options.
  - Knowing any value in the stack stored before the format string.
  - Knowing the values of other parameters of the calling function.
  - Modifying the value of a program variable, possibly not in the stack.

✓ **Question 2.8 ♣** In documented exploits, how many bytes of an overflow were needed to be successful?

- 1/1
- Not less than 4, to overwrite an address.
  - Not less than 8, to overwrite an address.
  - One byte can be enough.

✓ **Question 2.9 ♣** In the buffer overflow (BO) and the format strings (FS) labs, we have seen how to read and write data in the stack. Assume that we a large BO that overwrites bytes up to the stack frame of the calling function, followed by a FS vulnerability. Mark one or all as true.

- 1/1
- All the other options.
  - We can exploit the FS to find the address of the local buffer and then exploit the BO to execute *shellcode*.
  - We can exploit the BO to write an arbitrary address to the stack and then exploit the FS to read the value stored in that address.
  - We can exploit the FS to read the stack canary and then exploit the BO to overwrite the current function's return address.

✓ **Question 2.10 ♣** In the buffer overflow lab, we overwrote the buffer with a malicious payload, leading to the execution of *shellcode*. In the used script, part the payload was filled with 0x90, the bytecode representation of the NOP instruction. Its purpose is to spend CPU cycles, while:

- 1/1
- Counting how many times NOP was executed and storing it in a register.
  - Clearing the %EBP register.
  - None of the other options.
  - Increasing the program counter but not changing any memory or register.

✓ **Group 3 Systems Security (9 questions)**

✓ **Question 3.1 ♣** Which is **not** an important protection to ensure that an OS is secure?

- 1/1
- OS virtual page tables stored in RAM memory shall be cleared when the OS shuts down.
  - A Trusted Platform Module ensures that the kernel is digitally signed by root.
  - UEFI Secure Boot ensures that the firmware is digitally signed by the manufacturer.
  - The OS hibernation files stored on disk shall always be encrypted.



✓ **Question 3.2 ♣** For efficiency reasons, the address space of the Linux kernel is not completely independent from that of user processes due to *kernel mapping*. Which of the following is correct?

- 0.2/1
- Part of each user process's memory is mapped into the kernel memory space, to pass parameters to system calls.
  - Part of the kernel memory space is mapped into the memory space of each process, allowing it to write to mapped kernel memory.
  - Kernel Page-Table Isolation keeps a minimal kernel memory mapping for system calls into the memory space of each process.
  - Part of the kernel memory space is mapped differently across different user processes.  
↑

✓ **Question 3.3 ♣** A critical component of an Operating System (OS) is its reference monitor, which controls process invocations to system calls. Which of the following statements is **not** true?

- 1/1
- The `ptrace` system call offers a possible way to implement a reference monitor. ✓
  - The design of the reference monitor shall be simpler to validate than that of the OS. ✓
  - There should be a large number of system calls, with the most sharing of mechanisms. LEAST COMMON ECONOMY
  - If the reference monitor crashes, all monitored processes have to be killed. ✓

✓ **Question 3.4 ♣** In the environment variable and `setuid` program lab, we have experimented with environment variables and processes. Which of the following sentences is true?

- 0.2/1
- When a process forks a child process, it passes on its environment, excluding variables defined by the current user. ✗
  - The `execve` function always calls a shell function within a program with the program's environment.
  - When the shell forks a child process, it passes on its environment, but it may exclude some critical variables. ✗
  - If the shell is launched inside a process, it sets its privilege to that of the process's owner. ?

✓ **Question 3.5 ♣** Which is **not** a systems security principle that we have studied in the classes?

- 1/1
- Least common mechanism ✓
  - MINIMUM Sharing of privilege
  - Work factor ✓
  - Fail-safe defaults ✓

✓ **Question 3.6 ♣** Consider a bank with a set of safe deposit boxes which store valuables from clients.

- 1/1
- The bank teller checking client access to boxes is an example of Capabilities.
  - Each client having a card which opens a set of boxes is an example of Capabilities.
  - A set of shared cards which open a family's box is an example of Access Control Lists.
  - With Access Control Lists, two clients cannot have access to the same box.

✓ **Question 3.7 ♣** Virtualisation is a common security mechanism. Which sentence is **not** true?

- 1/1
- Hardware support improves performance of virtualised programs. ✓
  - Malware can detect that it is running in a virtualised environment. ✓
  - Hardware support ensures indistinguishable behavior for virtualised programs.



+216/5/26+

✓ Question 3.8 ♣ An important detail of the UNIX file permissions is the User ID (UID) of a process. A process is launched with the UID:

- 1/1
- Of the owner, unless the `setuid` bit is set to 1.
  - Of the owner, unless that user is root.
  - Of the user who launched it, unless the `setuid` bit is set to 1.
  - Of the root user, If `setuid` bit is set to 1

✓ Question 3.9 ♣ Recall the Linux Environment CTF. For the program on the right, which functions could be overridden using the `LD_PRELOAD` environment variable?

- 1/1
- `test_heap`, `test_sum`, `main`
  - `printf`, `fgets`, `malloc`
  - `puts`, `printf`, `my_puts`
  - `test_heap`, `test_sum`, `my_puts`

```
#include <stdlib.h>
#include <stdio.h>
char* test_heap() {
 char* buf = (char*) malloc(10);
 fgets(buf, 9, stdin); return buf; }
int test_sum() { return 2 * 2; }
void my_puts(char* str) {
 printf("%s\n", str); }
int main() { puts("Hello!"); printf("%d\n", test_sum()); my_puts(test_heap()); return 0; }
```

## Group 4 Web Security (7 questions)

✓ Question 4.1 ♣ Assuming the traditional configuration `SameSite = None`, when does a browser send a cookie to a server based on its origin? Choose the correct filling for the below table.

| Request to URL | Domain=login.fsi.pt; Path=/ | Domain=fsi.pt; Path=/ | Domain=fsi.pt; Path=/my/home |
|----------------|-----------------------------|-----------------------|------------------------------|
| login.fsi.pt   | 1 ✓                         | 2 ✗                   | 3 ✗                          |
| fsi.pt/my      | 4 ✗                         | 5 ✓                   | 6 ✗                          |

COOKIE

- 0.2/1
- 1=No, 2=Yes, 3=Yes, 4=No, 5=No, 6=Yes.
  - 1=Yes, 2=Yes, 3=No, 4=No, 5=Yes, 6=Yes.
  - 1=Yes, 2=Yes, 3=No, 4=Yes, 5=Yes, 6=Yes.
  - 1=Yes, 2=Yes, 3=No, 4=No, 5=No, 6=Yes.

✓ Question 4.2 ♣ We have seen various web attacks. Which of the following sentences is true?

- 1/1
- Cross-Site Scripting (XSS) is a class of server-side attacks.
  - Insecure Direct Object Reference (IDOR) is a class of client-side attacks.
  - XS-Leaks are a class of client-side attacks.

✓ Question 4.3 ♣ It is dangerous for GET requests to have side-effects. Because they are typically:

- 0.2/1
- Not blocked by the Same Origin Policy.
  - Not pre-flighted according to the CORS.
  - Kept in server logs and browser history.
  - All the other options.



**Question 4.4 ♣** Consider the following SQL query, that we have seen in the SQL Injection Lab, and is vulnerable to SQL injection. How could we change the query to list all users in the database?

```
$input_uname = $_GET['Username']; $input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);
$sql = "SELECT id, name, eid, salary, birth, ssn, address, email, nickname,
 Password FROM credential WHERE name= '$input_uname' AND Password=' $hashed_pwd '";
$result = $conn -> query($sql);
```

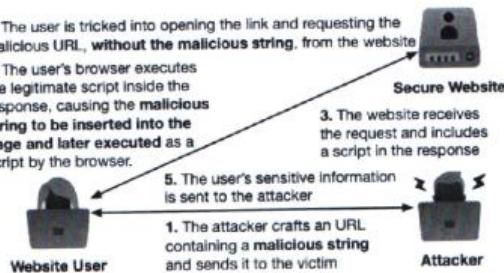
- 1/1
- Defining Username="" OR TRUE # and Password="" OR TRUE # .
  - Defining Username="admin" and Password="" OR 1=1 # .
  - Defining Username="admin" and Password="1=1#".
  - It is only possible to list one or less users.

**Question 4.5 ♣** Cross-Site Request Forgery (CSRF) is a classical web security attack, that many web proposals seek to mitigate. Which of the following statements is true?

- 0.2/1
- CSRF tokens allow a web server to receive a whitelist of trustworthy origins.
  - Fetch Metadata headers allow a web server to take decisions on which client requests to accept based on their origin.
  - SameSite=Strict cookies allow a web server to take decisions on which client requests to accept based on their origin.
  - Fetch Metadata headers allow a web server to receive a whitelist of trustworthy origins.

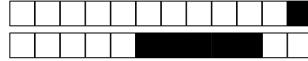
→ **Question 4.6 ♣** The figure illustrates a typical example of Cross-Site Scripting (XSS) attack, namely:

- 0.2/1
- A reflected XSS subtype.
  - A Cross-Frame Scripting (XFS) subtype.
  - A DOM-based XSS subtype.
  - A stored XSS subtype.



→ **Question 4.7 ♣** Which is the most secure form of session management in web applications?

- 0.2/1
- The server manages all client state, stored in an encrypted and MAC-signed session cookie.
  - The server stores all client state, bound to the same identifier for all cookies of a client.
  - The client manages all its state in its session cookie, which is stored by the server.
  - The server stores all client state, bound to a random token in each session cookie.



This is a multiple-choice test that will be corrected automatically. Please follow these rules:

- No pencil or light-coloured pens. Mark your answer using only **blue** and **black** pen.
- Erased boxes are not detected automatically. Check only inside boxes and be generous on ink.
- You may underline text or take notes on the sides. Only boxes matter for automatic correction.

The test is marked for 20 points and contains 30 questions, each with 4 options. **Only one option is accepted as correct, which could indicate all options are correct.**

You can check one or two choices per question. The scoring for each question is as follows:

- One checked correct answer (100%).
- One checked incorrect answer (-20%).
- No checked answers (0%).
- Two checked answers, one correct (50%).
- Two checked answers, none correct (-20%).
- More than two checked answers (-20%).

|     |     |     |     |     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| □ 0 | □ 0 | □ 0 | □ 0 | □ 0 | □ 0 | □ 0 | □ 0 | □ 0 | □ 0 |
| □ 1 | □ 1 | □ 1 | □ 1 | □ 1 | □ 1 | □ 1 | □ 1 | □ 1 | □ 1 |
| □ 2 | □ 2 | □ 2 | □ 2 | □ 2 | □ 2 | □ 2 | □ 2 | □ 2 | □ 2 |
| □ 3 | □ 3 | □ 3 | □ 3 | □ 3 | □ 3 | □ 3 | □ 3 | □ 3 | □ 3 |
| □ 4 | □ 4 | □ 4 | □ 4 | □ 4 | □ 4 | □ 4 | □ 4 | □ 4 | □ 4 |
| □ 5 | □ 5 | □ 5 | □ 5 | □ 5 | □ 5 | □ 5 | □ 5 | □ 5 | □ 5 |
| □ 6 | □ 6 | □ 6 | □ 6 | □ 6 | □ 6 | □ 6 | □ 6 | □ 6 | □ 6 |
| □ 7 | □ 7 | □ 7 | □ 7 | □ 7 | □ 7 | □ 7 | □ 7 | □ 7 | □ 7 |
| □ 8 | □ 8 | □ 8 | □ 8 | □ 8 | □ 8 | □ 8 | □ 8 | □ 8 | □ 8 |
| □ 9 | □ 9 | □ 9 | □ 9 | □ 9 | □ 9 | □ 9 | □ 9 | □ 9 | □ 9 |

← code your 9-digit upYYYYXXXXX student number horizontally on the left, and replicate it below. Write also your first and last name below.

Student Number:

up

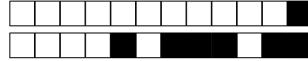
First and Last Name:

.....

## Group 1 Cryptography (10 questions)

✓ Question 1.1 ♣ A key exchange protocol does **not** guarantee Perfect Forward Secrecy when:

- Corrupting session keys reveals all past session keys.
- Corrupting long-term keys permits knowing all future session keys.
- Corrupting long-term keys reveals all past session keys.



✓ **Question 1.2 ♣** Remember that a Message Authentication Code (MAC) has the syntax  $\text{MAC}(k, m) = t$ . To protect a sequence of messages, a MAC must be used:

- In the context of a challenge-response protocol.
- Guaranteeing that each message is only transmitted once.
- Without additional concerns: a MAC allows precisely to protect sequences of messages.
- With confirmation in both directions.

✓ **Question 1.3 ♣** The non-repudiation property is important in the context of message authentication. When using public-key encryption and signatures, what should you do to guarantee this property?

- Sign the plaintext.
- Sign the ciphertext.
- Sign the plaintext and the ciphertext.
- Encryption should never be used when this property is needed.

✓ **Question 1.4 ♣** The *encrypt* operation in a modern symmetric cipher has which 3 input parameters?

- Key, Nonce and Tag.
- Key, Nonce and Message.
- Key, Tag and Message.
- Sender's key, Receiver's Key and Message.

✓ **Question 1.5 ♣** Nowadays, the construction of an authenticated cipher using a symmetric cipher and a Message Authentication Code (MAC) uses the construction:

- Encrypt-And-Mac.
- MAC-Then-Encrypt.
- None of the others.
- Encrypt-Then-MAC.

✓ **Question 1.6 ♣** In modern operating systems, the generation of random numbers for cryptography is generally done using:

- A random number generator fed by another random number generator.
- Dedicated hardware to generate entropy that is directly used by applications.
- A random number generation algorithm that feeds a pseudo-random one.
- None of the other options.

✓ **Question 1.7 ♣** Which is a correct assignment between protection requirements and applications?

|                                                  |                                       |                                               |                     |
|--------------------------------------------------|---------------------------------------|-----------------------------------------------|---------------------|
| <input checked="" type="checkbox"/> 1-B;2-C;3-A. | <input type="checkbox"/> 1-B;2-A;1-C. | B (1) in transit, synchronous communications  | (A) disk encryption |
| <input type="checkbox"/> 1-A;2-B;3-C.            | <input type="checkbox"/> 1-C;2-B;3-A. | C (2) in transit, asynchronous communications | (B) https           |

B (1) in transit, synchronous communications  
C (2) in transit, asynchronous communications  
A (3) at rest

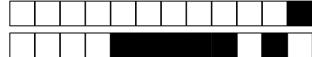
A (3) at rest

C (2) in transit, asynchronous communications

B (1) in transit, synchronous communications

✓ **Question 1.8 ♣** In a key distribution system, a Key Distribution Center (KDC) interacts with  $N$  agents that trust in the KDC:

- Only to store long-term keys, but the KDC does not know the session keys.
- Because it knows all the keys and could violate the security of the communications.
- Only to store session keys, which are made available when necessary.
- The KDC does not need to be trusted.



✓ Question 1.9 ♣ The Counter Mode (CTR) mode of operation is:

- A secure construction of a symmetric cipher based on a block cipher.
- An insecure construction of a MAC based on a block cipher.

✓ Question 1.10 ♣ The Electronic Code Book mode of operation is insecure because:

- It reveals patterns in the message.
- Is not insecure and is used in practice.
- Uses a key that is too small.
- Uses an insecure block cipher.

## Group 2 Public-Key-Infrastructure (5 questions)

✓ Question 2.1 ♣ When using public-key certificates to transfer digitally-signed information from A to B:

- A has to know and validate a priori B's certificate.
- B has to know and validate a priori A's certificate.
- A and B need to have certificates issued by the same Certification Authority.
- A and B have to exchange and validate certificates a priori.

✓ Question 2.2 ♣ Recall the PKI Lab. During the execution of a Man-In-The-Middle attack, Alice created a site to emulate the behaviour of a bank's website to obtain Bob's access credentials. Additionally, she created a CA and generated a valid certificate for her site, she performed a cache poisoning by mapping her server to the bank's site on Bob's computer and added her CA to Firefox's valid certificate list on Bob's computer. If Bob decides to check his bank account from his computer on Firefox, will Alice be able to steal his credentials?

- Yes.
- It depends. If a device in the communication route between Bob's computer and the malicious server knows the correct site's IP address, it will override the cache poisoning routing and redirect Bob to the correct site.
- No, Firefox will prevent Bob to access the

target insecure site because the CA authenticating it was added to Firefox's trusted certificates and does not belong to the operative system's certificate repository.

- No, if Bob uses the HTTPS protocol Firefox will warn him that the target site can be malicious.

✓ Question 2.3 ♣ The pinning of public-key certificates has the goal of allowing:

- A certification authority to account for the falsification of certificates in a given context.
- A software or service provider to account for the falsification of certificates in a given context.
- A certification authority to detect certificates that must be excluded from CRLs.
- A software or service provider to function as a certification authority.

✓ Question 2.4 ♣ A public-key certificate of a server is signed by the Certification Authority to:

- Allow it to be sent over insecure channels.
- All the other options.
- Guarantee that it was not modified.
- Attest who issued it.



✓ **Question 2.5 ♣** Which of the following is correct in the context of certificate chains?

- A certificate chain may be formed by two certificates: an end-user certificate and an intermediate certification authority certificate.
- A certificate chain may include many certificates that do not belong to certification authorities.
- A certificate chain be formed by three certificates: an end-user certificate, an intermediate certification authority certificate and a root certification authority certificate.
- A certificate chain includes all trusted root certificates.

### Group 3 Authentication (4 questions)

✓ **Question 3.1 ♣** Which of the following attacks is **not** a potential theft of session tokens?

- Attacker is a Man-in-the-Middle in the communications between client and server.
- Attacker uses SQL Injection on the server side to invalidate all tokens in use.
- Attacker convinces the user to login with his token.
- Attacker uses Cross-Site Scripting to read hidden tokens in forms in the client side.

✓ **Question 3.2 ♣** In a multi-factor authentication scenario, which of the following additional factors may **not** be considered?

- Challenge-response of a digital signature using a smartcard.
- Challenge-response with the same application in another device.
- User is requested a second and stronger password.
- Application sends a unique code and with a short validity by SMS.

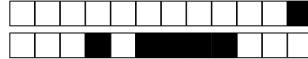
✓ **Question 3.3 ♣** Which is **not** a common component of a biometric authentication system?

- Collecting samples.
- Extracting templates.
- Detecting duplicated samples/templates.
- Matching samples with templates.

✓ **Question 3.4 ♣** Which of the following examples of authentication mechanisms permit identifying a user according to each of the principles in the below table?

- ①=password; ②=smartcard; ③=biometry
- ①=mensagem SMS; ②=cellphone; ③=biometry
- ①=cellphone number; ②=biometry; ③=administrator
- ①=client number; ②=password; ③=user profile

| Principle                   | Mechanism |
|-----------------------------|-----------|
| Something that is known     | ①         |
| Something that is owned     | ②         |
| Something that is intrinsic | ③         |



## Group 4 Network Security (6 questions)

✓ Question 4.1 ♣ There are various ways to perform spoofing attacks in networks. Which of the following statements is **not** true?

- IRDP spoofing allows announcing a fake router.
- MAC spoofing allows usurping the MAC address of another machine.
- DNS spoofing allows controlling the translation of IPs into MAC addresses.

✓ Question 4.2 ♣ Which of the following is a possible way to bypass a firewall?

- Configuring a proxy for a specific application.
- Using a NAT spoofing attack.
- Tunneling packets of a protocol inside another protocol.
- Using NAT to avoid the distinction among internal IPs.

✓ Question 4.3 ♣ It is possible to increase the security of a network at various levels. Which of the following examples is **not** true?

- At the network layer, e.g. IPSec.
- At the transport layer, e.g. TLS.
- At the application layer, e.g., WhatsApp.
- At the physical layer, e.g. PKI.

✓ Question 4.4 ♣ Mechanisms such as *firewalls* and *proxies* permit protecting machines connected in a network. Which of the following statements is **not** true?

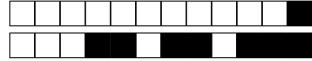
- A proxy is oriented to specific protocols.
- A local firewall may include filters for specific applications.
- A network firewall controls external accesses using specific proxies.
- A firewall is oriented to packets.

✓ Question 4.5 ♣ In a rogue DHCP attack:

- A DHCP server can force a client to renew its statically-defined IP.
- A DHCP server can advertise a fake router to a client.
- A DHCP server never returns acknowledgements to client requests.
- A client can perform Denial of Service of a DHCP server.

✓ Question 4.6 ♣ One of the most basic network attacks is the so-called *wiretapping*. Which of the following statements is **not** true?

- It permits Denial of Service of a communication channel.
- It is typically at the level of the physical/logical layer.
- It does not permit modifying packets in a network.
- It permits eavesdropping/sending packets in a network.



## Group 5 Malware and Detection (3 questions)

✓ Question 5.1 ♣ Which of the following is **not** a technique used for detecting botnets?

- Detecting communication traffic with the command and control system.
- Creating a new command and control system that allows deactivating the bots.
- Exposing vulnerable machines to be compromised and monitored.
- Detecting malware in a compromised machine.

✓ Question 5.2 ♣ In what consists the concept of *malvertising*?

- Pages on the *darkweb* that auction malware.
- ads* with wrong information.
- Malware that advertises itself.
- Using *ads* systems to explore vulnerabilities in browsers.

✓ Question 5.3 ♣ Which of the following is an advantage of *host-based detection systems* relatively to *network-based detection systems* or vice-versa?

- A *host-based detection system* does not use system resources while executing.
- A *network-based detection system* permits protecting more systems.
- A *network-based detection system* permits inspecting the contents of encrypted packets.
- A *host-based detection system* is simpler to deploy in a large organization.

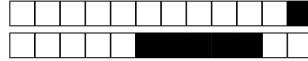
## Group 6 Transport Layer Security (TLS) (2 questions)

✓ Question 6.1 ♣ In what consists a *SSL stripping* attack?

- When an attacker connects to the server using HTTP and to the client using HTTPS.
- When an attacker connects to the server using HTTPS and to the client using HTTP.
- When an attacker connects to the server and the client using HTTPS.
- When an attacker is able to remove the TLS/SSL headers from HTTPS packets.

✓ Question 6.2 ♣ When a Certification Authority is corrupted in a TLS/HTTPS connection:

- All the security of past sessions is potentially lost.
- Attacker won't be able to decrypt future messages.
- It is only problematic for versions prior to TLS 1.3.
- It will allow Man-in-the-Middle attacks.



This is a multiple-choice test that will be corrected automatically. Please follow these rules:

- No pencil or light-coloured pens. Mark your answer using only **blue** and **black** pen.
- Erased boxes are not detected automatically. Check only inside boxes and be generous on ink.
- You may underline text or take notes on the sides. Only boxes matter for automatic correction.

The test is marked for 20 points. Each question is worth 20/30 points.

There are 30 questions in total, each with 4 options. **Only one of those options is accepted as correct, which could be an option indicating all options are correct.**

You can check one or two choices per question. The scoring for each question is as follows:

- One checked correct answer (100%).
- One checked incorrect answer (-20%).
- No checked answers (0%).
- Two checked answers, one correct (50%).
- Two checked answers, none correct (-20%).
- More than two checked answers (-20%).

|                            |                            |                            |                            |                            |                            |                            |                            |                            |
|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|
| <input type="checkbox"/> 0 |
| <input type="checkbox"/> 1 |
| <input type="checkbox"/> 2 |
| <input type="checkbox"/> 3 |
| <input type="checkbox"/> 4 |
| <input type="checkbox"/> 5 |
| <input type="checkbox"/> 6 |
| <input type="checkbox"/> 7 |
| <input type="checkbox"/> 8 |
| <input type="checkbox"/> 9 |

← code your 9-digit upYYYYXXXXX student number horizontally on the left, and replicate it below. Write also your first and last name below.

Student Number:

|    |                          |                          |                          |                          |                          |                          |                          |                          |
|----|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| up | <input type="checkbox"/> |
|----|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|

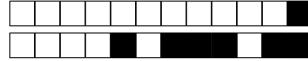
First and Last Name:

|       |
|-------|
| ..... |
|-------|

## Group 1 Cryptography (10 questions)

**Question 1.1** ♣ Consider a scenario where A sends  $(pk, m, \sigma)$  to B where  $m$  is a message and  $\sigma = \text{Sig}(sk, m)$  is a (secure) digital signature of  $m$ . A Man-In-The-Middle (MitM) attacker could persuade receiver B that A sent  $m' \neq m$  by:

- MitM attacks are not possible when using digital signatures.
- Keeping  $pk$ , changing  $m$  for  $m'$  and generating a new signature  $\sigma'$ .
- Keeping  $pk$ , and replacing  $(m, \sigma)$  with  $(m', \sigma)$ .
- Convincing B that A owns a public key different than  $pk$ .



✓ **Question 1.2 ♣** The difference between an authenticated encryption scheme (AE) and an authenticated encryption scheme with associated data (AEAD) is that:

- AEAD probabilistic whereas AE is deterministic.
- AEAD allows binding public metadata to the ciphertext, while AE does not.
- AEAD allows associating metadata to encryption keys, which the AE does not allow.
- AEAD guarantees metadata confidentiality, which the AE does not allow.

✓ **Question 1.3 ♣** The statement “*Public-key cryptography made symmetric cryptography obsolete*” is:

- False: the two techniques are always used together for performance reasons.
- True, but only in applications where one cannot have a pre-shared secret key.
- True, except for applications where we need protection against quantum computers.
- False: if a PKI is not available one always uses symmetric cryptography.

✓ **Question 1.4 ♣** The Electronic Code Book operation mode is:

- A secure MAC scheme construction that uses a block cipher.
- An insecure symmetric encryption scheme construction that uses a block cipher.
- An insecure MAC scheme construction that uses a block cipher.
- A secure symmetric encryption scheme construction that uses a block cipher.

✓ **Question 1.5 ♣** In a KDS, a Key Distribution Center interacts with  $N$  agents and:

- Stores 1 long-term key that it uses to establish an arbitrary number of session keys.
- Stores  $N * (N - 1)/2$  long-term keys, which it makes available when needed for communication.
- Caches a varying number of session keys, which are gradually provided to the participants.
- Stores  $N$  long-term keys that it uses to establish an arbitrary number of session keys.

✓ **Question 1.6 ♣** The Perfect Forward Secrecy property guarantees that:

- Corrupting a session key does not compromise past session keys.
- Corrupting a long-term key does not compromise future session keys.
- Corrupting a long-term key does not compromise past session keys.
- Corrupting a session key does not compromise future session keys.

✓ **Question 1.7 ♣** Which is a correct assignment for (A) RSA signature, (B) AES-CTR, (C) RSA-OAEP and (D) HMAC?

|            | Confidentiality | Authenticity |
|------------|-----------------|--------------|
| Symmetric  | (1) <b>B</b>    | (2) <b>D</b> |
| Asymmetric | (3) <b>C</b>    | (4) <b>A</b> |

- 1-B;2-D;3-C;4-A.
- 1-A;2-B;3-C;4-D.
- 1-C;2-A;3-D;4-B.
- 1-A;2-C;3-B;4-D.

✓ **Question 1.8 ♣** The non-repudiation property is important in the context of message authentication. Which of the following sentences is true? Note that MAC stands for Message Authentication Code.

- Asymmetric ciphers guarantee this property, as long as the public-key is authentic.
- A MAC does not guarantee this property.
- Digital signatures guarantee this property, even after the long-term secret key is compromised.
- A MAC guarantees this property, provided that the sender is trusted.



- ✓ **Question 1.9 ♣** Recall that a Message Authentication Code (MAC) has the following syntax  $\text{MAC}(k, m) = t$ . A MAC guarantees:

- Message integrity and authentication.
- Message confidentiality, integrity and authentication.
- Confidentiality, integrity and authentication for a sequence of messages.
- Integrity and authentication for a sequence of messages.

- ✓ **Question 1.10 ♣** A common construction of a secure symmetric encryption scheme is of the form  $\text{Enc}(k, n, m) = \text{PRG}(k, n) \oplus m$ . The following property shows that this construction *does not* provide integrity guarantees:

- Flipping a single ciphertext bit causes a single bit flip in the recovered message.
- The PRG generator produces a uniform distribution.
- The XOR operation cancels out:  
$$\text{PRG}(k, n) \oplus \text{PRG}(k, n) \oplus m = m.$$
- The PRG generator does not produce a uniform distribution.

## Group 2 Public-Key-Infrastructure (5 questions)

- ✓ **Question 2.1 ♣** When using public-key certificates to transfer encrypted information from A to B using a public-key encryption scheme:

- B must get and a priori verify A's certificate.
- A must get and a priori verify B's certificate.
- A and B must hold certificates issued by the same certification authority.
- A and B must exchange and a priori verify each others' certificates.

- ✓ **Question 2.2 ♣** For a certificate authority, a Certificate Revocation List (CRL)

- Contains all issued certificates that, while being within the validity period, should not be used.
- Contains all issued certificates that should not be used.
- Contains only issued certificates that are within the validity period and can be used.
- Contains only issued certificates that can be used.

- ✓ **Question 2.3 ♣** Recall what you have studied about certificate chains. Suppose certification authority A signed the certificate of certification authority B, and that all you know about A and B is what is written in this certificate.

- Trust in A can never be greater than trust in B.
- Trust in B can never be greater than trust in A.
- B cannot operate until it also signs a certificate for A.
- A trusts B to sign a certificate for A.

- ✓ **Question 2.4 ♣** The public-key infrastructure solves the following problem:

- Sharing of asymmetric secret keys.
- Public key authentication.
- Public key authentication and confidentiality.
- Symmetric key sharing using public keys.



### ✓ Question 2.5 ♣ Which is the most common channel for a user to get information about the certification authorities that work as trust roots in a PKI?

- Their certificates are distributed by the web sites the users visit.
- Users only get that information when they need a public key certificate for a web server.
- Their certificates are pre-installed in operating systems and browsers.
- Users only get that information when they need a personal public key certificate.

## Group 3 Authentication (4 questions)

### ✓ Question 3.1 ♣ Which is the best way for a web application to store session tokens on the client side?

- In cookies.
- A combination of all the other options.
- In the content of links.
- In hidden form fields.

### ✓ Question 3.2 ♣ Which of the following is **not** an attack on a password-based authentication mechanism?

- Data breach in a server reveals user passwords.
- Phishing site steals user credentials.
- User chooses a weak password.
- Malware registers user keystrokes.

### ✓ Question 3.3 ♣ Which does **not** represent a security risk for biometric authentication systems?

- Stealing characteristics from individuals.
- High rate of false positives.
- High rate of false negatives.
- Forging characteristics from individuals.

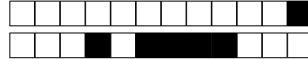
### ✓ Question 3.4 ♣ Which is the main difference between *message authentication* (MA) e *entity authentication* (EA)?

- EA is intended to verify that an entity participates at real time in the protocol.
- In MA there exists typically a requirement that a message was sent recently, by the correct entity.
- In EA the receiver has the guarantee that a message was sent by a specific entity, contrary to MA where the receiver only knows that the sent message is valid.
- An EA mechanism requires the use of a MA mechanism, but not vice-versa.

## Group 4 Network Security (6 questions)

### ✓ Question 4.1 ♣ Which of the following is an attack at the level of the transport layer?

- Rogue DHCP.
- MAC flooding.
- DNS cache poisoning.
- TCP session hijacking.



✓ **Question 4.2 ♣** In the context of *firewall* packet filtering, which of the following statements is true?

- A *Default allow* policy typically offers more protection than a *Default deny* policy.
- Packet filtering does not distinguish inbound and outbound traffic.
- Stateful filtering has the disadvantage of being harder to implement.
- Stateless filtering has the disadvantage of being harder to configure exceptions for legitimate users.

✓ **Question 4.3 ♣** Consider attacks to the DNS protocol. Which of the following statements is **not** true?

- Both DNS spoofing and DNS cache poisoning permit directing users to malicious machines.
- DNS spoofing can be performed by malware directly in the user's machine.
- DNS cache poisoning is an attack directed at a legitimate DNS server.
- DNS spoofing consists in flooding a DNS server with IP registers.

✓ **Question 4.4 ♣** Which of the following attacks to the UDP/TCP protocols is harder to concretize?

- TCP session spoofing.
- UDP session hijacking.
- TCP session hijacking.
- Sending RST messages.

✓ **Question 4.5 ♣** At the level of network communications, which of the following statements is true?

- An *on-path* can only send packets.
- A *man-in-the-middle* attacker can control all communications.
- An *off-path* can only receive packets.
- An *eavesdropper* just cannot modify packets.

✓ **Question 4.6 ♣** A MAC address physically identifies a machine in a network. Which of the following statements is **not** true?

- A MAC spoofing attack allows usurping the MAC address of another machine.
- A MAC flooding attack can force a switch to broadcast all packets.
- A MAC flooding attack intends to perform Denial of Service of a switch.
- A MAC spoofing attack permits impersonating a hub/router/switch.

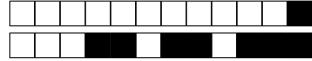
## Group 5 Malware and Detection (3 questions)

✓ **Question 5.1 ♣** In what consists the concept of signature-based malware detection?

- Digitally signing the software so that malware cannot modify its execution.
- Detecting personal signatures that hackers leave in the code of the malware that they create.
- Detecting patterns of known attacks.
- Identifying digital signatures of servers to which malware tries to connect.

✓ **Question 5.2 ♣** Which of the following statements is **not** true?

- A *worm* can be used to create a *botnet*.
- A *botnet* is a network of malware computers with a common control.
- A *worm* is a self-propagating malware.
- A *worm* is a malware used as "bait" to fool users.



✓ **Question 5.3 ♣** Which is **not** a strategy that a modern *virus* uses to avoid being detected?

- Dissimulate itself as a regular file and mutate when executing.
- Terminating the processes launched by the antivirus.
- Behave differently when executed in a *sandbox*.
- Encrypt its code in a probabilistic fashion on each infection.

## Group 6 Transport Layer Security (TLS) (*2 questions*)

✓ **Question 6.1 ♣** Which of the following attacks can be avoided using TLS connections?

- DNS spoofing.
- Man-in-the-middle attacks, as long as the client validates the server's certificate.
- Network traffic analysis to obtain meta-data.
- Web pages that include HTTP/HTTPS mixed content.

✓ **Question 6.2 ♣** Which is the difference in TLS 1.3 handshake to prior versions?

- Corrupting the server keys does not affect previous sessions.
- It makes use of RSA instead of authenticated Diffie-Hellman.
- Does not make use of long-term keys.
- For reasons of performance, connections do not always guarantee perfect forward secrecy.



+76/1/30+

Fundamentos de Segurança Informática  
15/1/2024

Duração: 1H

LEIC  
Teste Intermédio

Este teste de escolha múltipla será corrigido automaticamente. Siga por favor as seguintes indicações:

- Não utilize lápis ou cores leves. Marque a sua resposta utilizando apenas caneta azul e preta.
- Rasuras não são detectadas automaticamente. Marque apenas caixas, sendo generoso na tinta.
- Pode sublinhar texto ou tirar notas nas margens. Apenas as caixas importam para a correção.

O teste está cotado para 20 valores e tem 25 perguntas, cada uma com 4 opções e igual cotação.  
**Apenas uma opção é correta, podendo indicar que todas as outras opções estão corretas.**  
Pode marcar uma ou duas escolhas por questão. A cotação é atribuída da seguinte forma:

- Uma resposta correta marcada (100%).
- Uma resposta incorreta marcada (-20%).
- Nenhuma resposta marcada (0%).
- Duas resp. marcadas, uma correta (50%).
- Duas resp. marcadas, zero corretas (-20%).
- Mais do que duas resp. marcadas (-20%).

|                                       |                                       |                                       |                                       |                            |                                       |                                       |                                       |                            |                            |
|---------------------------------------|---------------------------------------|---------------------------------------|---------------------------------------|----------------------------|---------------------------------------|---------------------------------------|---------------------------------------|----------------------------|----------------------------|
| <input type="checkbox"/> 0            | <input checked="" type="checkbox"/> 0 | <input type="checkbox"/> 0            | <input checked="" type="checkbox"/> 0 | <input type="checkbox"/> 0 | <input type="checkbox"/> 0            | <input type="checkbox"/> 0            | <input type="checkbox"/> 0            | <input type="checkbox"/> 0 | <input type="checkbox"/> 0 |
| <input type="checkbox"/> 1            | <input type="checkbox"/> 1            | <input checked="" type="checkbox"/> 1 | <input type="checkbox"/> 1            | <input type="checkbox"/> 1 | <input type="checkbox"/> 1            | <input type="checkbox"/> 1            | <input type="checkbox"/> 1            | <input type="checkbox"/> 1 | <input type="checkbox"/> 1 |
| <input checked="" type="checkbox"/> 2 | <input type="checkbox"/> 2            | <input checked="" type="checkbox"/> 2 | <input type="checkbox"/> 2            | <input type="checkbox"/> 2 | <input type="checkbox"/> 2            | <input type="checkbox"/> 2            | <input type="checkbox"/> 2            | <input type="checkbox"/> 2 | <input type="checkbox"/> 2 |
| <input type="checkbox"/> 3            | <input type="checkbox"/> 3            | <input type="checkbox"/> 3            | <input type="checkbox"/> 3            | <input type="checkbox"/> 3 | <input type="checkbox"/> 3            | <input type="checkbox"/> 3            | <input type="checkbox"/> 3            | <input type="checkbox"/> 3 | <input type="checkbox"/> 3 |
| <input type="checkbox"/> 4            | <input type="checkbox"/> 4            | <input type="checkbox"/> 4            | <input type="checkbox"/> 4            | <input type="checkbox"/> 4 | <input type="checkbox"/> 4            | <input type="checkbox"/> 4            | <input checked="" type="checkbox"/> 4 | <input type="checkbox"/> 4 | <input type="checkbox"/> 4 |
| <input type="checkbox"/> 5            | <input type="checkbox"/> 5            | <input type="checkbox"/> 5            | <input type="checkbox"/> 5            | <input type="checkbox"/> 5 | <input type="checkbox"/> 5            | <input type="checkbox"/> 5            | <input type="checkbox"/> 5            | <input type="checkbox"/> 5 | <input type="checkbox"/> 5 |
| <input type="checkbox"/> 6            | <input type="checkbox"/> 6            | <input type="checkbox"/> 6            | <input type="checkbox"/> 6            | <input type="checkbox"/> 6 | <input type="checkbox"/> 6            | <input type="checkbox"/> 6            | <input type="checkbox"/> 6            | <input type="checkbox"/> 6 | <input type="checkbox"/> 6 |
| <input type="checkbox"/> 7            | <input type="checkbox"/> 7            | <input type="checkbox"/> 7            | <input type="checkbox"/> 7            | <input type="checkbox"/> 7 | <input type="checkbox"/> 7            | <input checked="" type="checkbox"/> 7 | <input type="checkbox"/> 7            | <input type="checkbox"/> 7 | <input type="checkbox"/> 7 |
| <input type="checkbox"/> 8            | <input type="checkbox"/> 8            | <input type="checkbox"/> 8            | <input type="checkbox"/> 8            | <input type="checkbox"/> 8 | <input checked="" type="checkbox"/> 8 | <input type="checkbox"/> 8            | <input type="checkbox"/> 8            | <input type="checkbox"/> 8 | <input type="checkbox"/> 8 |
| <input type="checkbox"/> 9            | <input type="checkbox"/> 9            | <input type="checkbox"/> 9            | <input type="checkbox"/> 9            | <input type="checkbox"/> 9 | <input type="checkbox"/> 9            | <input type="checkbox"/> 9            | <input type="checkbox"/> 9            | <input type="checkbox"/> 9 | <input type="checkbox"/> 9 |

← Codifique o seu número de estudante de 9 dígitos upYYYYXXXXXX horizontalmente na grelha à esquerda. Escreva também o seu número de estudante e o primeiro e último nomes em baixo.

Número de estudante:

up **2 0 2 1 0 8 7 4 4**

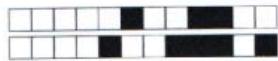
Primeiro e último Nome:

**MANUEL NETO**

## Grupo 1 Encriptação e Autenticação de Mensagens (13 questões)

**Questão 1.1 ♣** Considera um algoritmo de geração de chaves que produz chaves como bit-strings. O que significa dizer que este segue uma distribuição uniforme?

- 1/1
- Um adversário nunca pode dar *brute-force* nas chaves produzidas por este algoritmo
  - As chaves geradas podem ser usadas por qualquer algoritmo criptográfico
  - Padrões comuns (e.g. 00000 or 11111) não ocorrem nas chaves geradas
  - A probabilidade de cada bit na chave ser 0 é de  $\frac{1}{2}$



✓ Questão 1.2 ♣ O que indica o princípio de Kerkhoff?

- 1/1
- Todos os algoritmos criptográficos devem ser públicos. A única informação secreta é a chave
  - Algoritmos criptográficos devem ser rigorosamente analisados antes de serem usados
  - Todos os algoritmos criptográficos devem ser desenhados por especialistas
  - Nenhum algoritmo criptográfico deve ser revelado publicamente

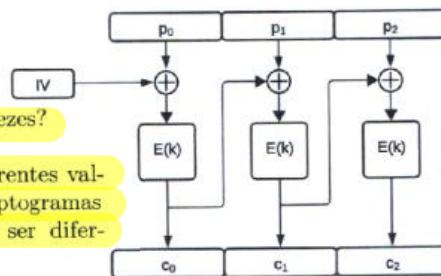
✓ Questão 1.3 ♣ Considera uma cifra de substituição, que para cifrar faz o *shift* de letras, de acordo com uma chave gerada, e reverte este processo na decifração. O que faz com que este esquema seja inseguro, de acordo com os padrões modernos de segurança?

- 1/1
- O tamanho do espaço da chave é demasiado pequeno. Podemos encontrar a chave por força-bruta
  - O criptograma vai revelar padrões nas letras, que podem ser usados para inferir informação
  - Este esquema é impossível de decifrar
  - Porque o criptograma não está assinado pelo remetente

✓ Questão 1.4 ♣ Considera o modo de cifra CBC, apresentado à direita

O que acontece se cifrarmos a mesma mensagem duas vezes?

- 1/1
- Os criptogramas resultantes vão ser sempre diferentes
  - Se usarmos diferentes valores de IV, os criptogramas resultantes vão ser diferentes ✓
  - Os criptogramas resultantes vão ser sempre iguais
  - Se usarmos o mesmo valor de IV, os criptogramas resultantes vão ser diferentes



✓ Questão 1.5 ♣ Um esquema de cifra é caracterizado por que combinação de algoritmos?

- 1/1
- Geração de chaves, Assinatura, Verificação
  - Geração de chaves, Cifração, Validação
  - Transmissão, Cifração, Decifração
  - Geração de chaves, Cifração, Decifração

✓ Questão 1.6 ♣ Considera uma função de hash criptográfica  $H$ , e duas mensagens diferentes  $m_1, m_2$ , tal que  $m_2$  é o resultado de trocar um bit de  $m_1$ . Qual das seguintes deve ser verdade?

- 1/1
- $H(m_1) = H(m_2)$
  - $H(m_1) > H(m_2)$  se  $m_1 > m_2$
  - $H(m_1)$  e  $H(m_2)$  não têm correlação
  - $H(m_1) \approx H(m_2)$ , i.e. são similares

✓ Questão 1.7 ♣ Considera as seguintes frases sobre funções de hash com chave  $H(k, m)$ , onde  $k$  é a chave e  $m$  é a mensagem. Qual delas não é verdadeira?

- 0.2/1
- Funções de hash com chave podem ser usadas para proteger a confidencialidade dos dados
  - Funções de hash com chave são usadas em combinação com esquemas de cifra para construir esquemas de cifras autenticadas ✓
  - Funções de hash com chave protegem contra mensagens forjadas, já que para produzir um output, o remetente tem que ter a chave  $k$
  - Funções de hash com chave podem ser usadas para proteger a integridade das mensagens



✓ Questão 1.8 ♣ O que significa dizer que uma função de hash  $H$  é resistente a colisões?

- 1/1
- É computacionalmente inviável encontrar dois valores  $m_1 \neq m_2$ , para os quais  $H(m_1) = H(m_2)$
  - Independentemente do input,  $H$  produz sempre o mesmo resultado
  - $H$  é resistente a ataques de *man-in-the-middle*
  - Não podem existir dois valores  $m_1 \neq m_2$ , para os quais  $H(m_1) = H(m_2)$

✓ Questão 1.9 ♣ Qual é a relação entre assinaturas digitais e MACs?

- 1/1
- Ambos são métodos para cifrar mensagens, em paradigmas diferentes
  - Ambos são métodos para assegurar a integridade das mensagens, em paradigmas diferentes
  - Assinaturas digitais devem ser usadas em combinação com MACs para assegurar a integridade de mensagens
  - Assinaturas digitais são um método mais poderoso para assegurar a integridade de mensagens que MACs

✓ Questão 1.10 ♣ O RSA não pode ser usado diretamente para cifração, por muitas razões. Qual das seguintes é uma razão válida para isso?

- 1/1
- Porque a fatorização do módulo de RSA é um problema simples por si só
  - O RSA não é randomizado, e como tal cifrar duas vezes a mesma mensagem produzirá dois criptogramas iguais
  - Porque não há forma de o inverter, i.e. recuperar a mensagem original
  - O problema de RSA é computado sobre números, e muitas vezes as mensagens são bit-strings

A A B?! B

✓ Questão 1.11 ♣ Liga os protocolos ao paradigma que assumem. 1 :AES-CTR; 2 :RSA-OAEP; 3 :Diffie-Hellman; 4 :HMAC; A :Criptografia de chave simétrica; B :Criptografia de chave pública

- 1/1
- 1 - A | 2 - A | 3 - B | 4 - A
  - 1 - B | 2 - A | 3 - A | 4 - B
  - 1 - B | 2 - B | 3 - B | 4 - A
  - 1 - A | 2 - B | 3 - B | 4 - A

✓ Questão 1.12 ♣ O paradigma híbrido é um sistema para cifração de chave pública, onde o corpo da mensagem  $m$  é primeiro cifrado utilizando uma chave simétrica fresca  $c_1 \leftarrow Enc(k, m)$ . Este criptograma  $c_1$  é então enviado, bem como uma cifração desta chave, utilizando a chave pública do destinatário  $c_2 \leftarrow Enc(pk, k)$ . Como podemos decifrar esta mensagem?

- 1/1
- $m \leftarrow Dec(sk, c_1, c_2)$
  - $k \leftarrow Dec(sk, c_1); m \leftarrow Dec(c_1, c_2)$
  - $k \leftarrow Dec(pk, c_2); m \leftarrow Dec(k, c_1)$
  - $k \leftarrow Dec(sk, c_2); m \leftarrow Dec(k, c_1)$

✓ Questão 1.13 ♣ Considera a troca de chaves de Diffie-Hellman, onde a Alice e o Bob geram  $x$  e  $y$ , respectivamente, e enviam  $X \leftarrow g^x$  e  $Y \leftarrow g^y$  pela rede. Como é que isto pode corretamente ser usado para trocar um segredo?

- 1/1
- Alice pode computar  $Y^x$ , Bob pode computar  $X^y$ , que são ambos  $XY^{xy}$
  - Alice pode computar  $g^{xy}$ , e enviar isto ao Bob
  - Alice pode computar  $Y^x$ , Bob pode computar  $X^y$ , que são ambos  $g^{xy}$
  - Porque  $g^x$  e  $g^y$  são o mesmo valor, e podem ser usados como o segredo



## Grupo 2 Infraestrutura de Chave Pública e Autenticação (3 questões)

✓ Questão 2.1 ♣ Qual é o benefício de Public-Key Infrastructures (PKIs) para cífras de chave pública?

- 1/1
- PKI estabelece canais seguros, que depois vão ser usados para fazer cífras de chave pública ✓
  - PKI valida a autenticidade de chaves, que vão ser usadas em algoritmos criptográficos de chave pública ✓
  - PKI mantém chaves secretas partilhadas par-a-par, que depois podem ser usadas para cifração
  - PKI mantém a infra-estrutura tecnológica usada para transmitir mensagens entre os participantes

✓ Questão 2.2 ♣ Qual das seguintes frases **mais completamente** descreve a informação tipicamente contida num certificado X.509?

- 1/1
- Subject, Issuer, Public Key e Validity ✓
  - Subject, Serial number, Firewall configuration e TLS version
  - Subject, Issuer e Validity ✓
  - Subject, Issuer, Public Key, Secret key e Validity ✓

✓ Questão 2.3 ♣ Uma contramedida comum contra ataques de repetição é a utilização de um nonce. Qual das seguintes alternativas não é um mecanismo válido para o implementar?

- 0.5/1
- Usar o millisegundo atual como nonce ✓
  - Usar um valor grande aleatório como nonce ✓
  - Usar a hora atual como nonce ✓
  - Começar com o nonce a 0, e incrementar sempre que uma mensagem é enviada

## Grupo 3 TLS e IPsec (4 questões)

✓ Questão 3.1 ♣ Qual é o propósito do protocolo de handshake do TLS?

- 1/1
- Frequentemente trocar sinais entre participantes, dessa forma assegurando que o canal de comunicação está operacional ~~NETBEAM~~
  - Enviar mensagens de alerta críticas ou de gestão, e fazer o registo do comportamento do canal de comunicação ~~ALEPIT~~
  - Autenticar os participantes e trocar material criptográfico, para mais tarde ser usado na comunicação ✓
  - Encapsular mensagens de payload usando cífras e códigos de autenticação de mensagens ~~RECORD~~

✓ Questão 3.2 ♣ Comunicação via IPsec é protegida de acordo com dois protocolos principais: Authentication Header (AH) e Encapsulated Security Payload (ESP). Qual das seguintes frases descreve estes com mais precisão?

- 1/1
- AH assegura a integridade das mensagens, enquanto que o ESP assegura confidencialidade, mas não integridade
  - AH assegura a integridade das mensagens, enquanto que o ESP assegura confidencialidade e integridade
  - ESP assegura a integridade das mensagens, enquanto que o AH assegura confidencialidade e integridade
  - AH e ESP asseguram níveis semelhantes de segurança, mas a autenticação dos participantes é feita de forma diferente



✓ Questão 3.3 ♣ Quais das seguintes não é um benefício de usar IPSec? REDE

1/1

- IPSec providencia segurança forte ao tráfego que cruza o perímetro da firewall ✓
- IPSec permite um controlo fino ao nível aplicacional dos dados trocados
- IPSec é transparente aos programadores de aplicações, comportando-se ao nível da rede
- IPSec previne *spoofing* de IPs, e como tal fortalece a arquitetura de routeamento

✓ Questão 3.4 ♣ As seguintes frases referem-se a protocolos de segurança de redes. Qual destas é verdadeira para o TLS, mas não para o IPSec?

-0.2/1

- O protocolo permite um controlo mais granular da segurança ao nível da aplicação, permanecendo oblivio aos mecanismos de segurança na camada de rede
- O protocolo é resiliente contra ataques de repetição, e contra o envio de mensagens forjadas por atacantes externos ✓?
- O protocolo permite uma arquitetura mais robusta de routeamento, permanecendo oblivio aos comportamentos no nível aplicacional
- O protocolo assegura comunicação segura entre dois participantes ✓?

## Grupo 4 Ataques e Contramedidas (5 questões)

1/1

✓ Questão 4.1 ♣ Quais são as desvantagens mais comuns de seguir políticas de firewall restritivas? DISPONIBILIDADE

1/1

- Configurações de firewall incorretas levam a brechas nos esquemas de cifração utilizados
- Configurações de firewall incorretas expõem o sistema a ataques de *IP spoofing*
- Configurações de firewall incorretas levam a serviços não reconhecidos interagir com o sistema
- Configurações de firewall incorretas levam a perda de disponibilidade do sistema

1/1

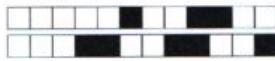
✓ Questão 4.2 ♣ No contexto de um ataque de *denial-of-service*, é comum usar servidores Zombie. Das seguintes, qual não é uma vantagem de depender destas máquinas?

- Servidores Zombie permitem que um atacante escale o ataque, utilizando múltiplas máquinas ✓
- Servidores Zombie permitem que um atacante delegue a computação necessária para executar um ataque, para outras máquinas
- Servidores Zombie permitem esconder a origem original do ataque
- Servidores Zombie permitem a execução de ataques de *spoofing* de IP

✓ Questão 4.3 ♣ O que é um ataque de *SYN Spoofing*?

1/1

- Um ataque ao TLS que visa atrasar o servidor, iniciando várias sessões falsas de TLS
- Um ataque ao TLS que visa quebrar o esquema de cifra usado nas comunicações
- Um ataque ao UPS que encaminha mensagens a destinos incorretos
- Um ataque ao Chargen, que envia uma grande quantidade de dados a um sistema vítima



✓ **Questão 4.4** Qual é a diferença de um Host-based IDS, quando comparado com um Network-based IDS?

- 1/1
- Host-based* IDS pode detetar imediatamente ataques, enquanto que *Network-based* IDS só pode registar ataques para análise posterior
  - Melhor controlo de segurança a nível aplicacional, sacrificando segurança que depende do contexto de rede
  - Network-based* IDS comporta-se como um *Host-based* IDS, com adicional contexto a nível de rede para as mensagens trocadas

✓ **Questão 4.5** Qual das seguintes adequadamente caracteriza a definição de uma firewall do tipo application proxy?

- 1/1
- Existe na camada de rede, requere configurações complexas, permite um controlo grosso da segurança
  - Existe na camada aplicacional, requere configurações complexas, permite um controlo fino da segurança
  - Existe na camada aplicacional, requere configurações simples, permite um controlo fino da segurança
  - Existe na camada aplicacional, requere configurações complexas, permite um controlo grosso da segurança