Scurit Informatique
Corrigs des Examens

Session Janvier 2024 & Janvier 2025

Table des matires

1 Examen Janvier 2024

1.1 Question 1 : Vrai ou Faux (10 points)

1.1.1 Q1.1 : Si un chiffrement satisfait la perfect security, alors $|K| \geq |M|$

Solution

Rponse : VRAI
Justification :
Le thorme de Shannon stipule que pour qu'un systme de chiffrement possde la perfect security, il faut obligatoirement que :

$$|K| \geq |M|$$

o $|K|$ est la taille de l'espace des cls et $|M|$ est la taille de l'espace des messages.
Intuition : Si l'espace des cls est plus petit que l'espace des messages, un attaquant peut liminer certains messages possibles en numrant toutes les cls, ce qui viole la perfect security.
Exemple : Le One-Time Pad (OTP) satisfait $|K| = |M|$ et possde la perfect security.

1.1.2 Q1.2 : Le One-Time Pad est pratique pour un usage quotidien

Solution

Rponse : FAUX
Justification :
Le One-Time Pad (OTP), bien qu'offrant une perfect security thorique, est totalement impratique pour un usage quotidien pour plusieurs raisons :

1. Taille de la cl : La cl doit tre aussi longue que le message
2. Distribution : Comment partager de manire scurise une cl de plusieurs gigaoctets ?
3. Usage unique : Chaque cl ne peut tre utilise qu'une seule fois
4. Stockage : Ncessite un stockage scuris massif
5. Gnration : Ncessite une source de vraie alatoire (pas de PRNG)

Conclusion : En pratique, on utilise des chiffrements comme AES avec des cls de 128/256 bits.

1.1.3 Q1.3 : Un PRG (Pseudo-Random Generator) scuris produit une sortie indistinguable d'un alatoire vrai

Solution

Rponse : VRAI
Justification :
La dfinition formelle de la scurit d'un PRG stipule qu'un PRG $G : \{0,1\}^s \to \{0,1\}^n$ (avec $n > s$) est scuris si pour tout algorithme polynomial A :

$$\text{PRGadv}[A, G] = |\Pr[A(G(k)) = 1] - \Pr[A(r) = 1]| < \varepsilon$$

o $k \leftarrow \{0,1\}^s$ (seed alatoire) et $r \leftarrow \{0,1\}^n$ (alatoire pur), et ε est ngligeable.
Interprtation : Aucun adversaire computationnellement born ne peut distinguer la sortie du PRG d'une squence vraiment alatoire avec une probabilit significative.

1.1.4 Q1.4 : AES-CBC ncessite un IV (Initialization Vector) alatoire

Solution

Rponse : VRAI
Justification :
Le mode CBC (Cipher Block Chaining) d'AES requiert un IV alatoire pour garantir la scurit CPA (Chosen-Plaintext Attack) :

- Chiffrement CBC : $C_i = E_k(P_i \oplus C_{i-1})$ avec $C_0 = IV$
- Scurit : Si l'IV est prvisible ou rutilis, un attaquant peut dtecter des messages identiques
- Bonne pratique : IV $\leftarrow \{0,1\}^{128}$ alatoire, diffrent pour chaque message

Point important

L'IV doit tre alatoire pour CBC, mais seulement unique (nonce) pour CTR.

1.1.5 Q1.5 : HMAC est vulnrable aux collisions de la fonction de hachage sous-jacente

Solution

Rponse : FAUX
Justification :
HMAC est conu spcifiquement pour tre rsistant aux collisions de la fonction de hachage :

$$\text{HMAC}_k(m) = H\left((k \oplus \text{opad}) \| H((k \oplus \text{ipad}) \| m)\right)$$

Points cls :

- Mme si H est vulnrable aux collisions, HMAC reste scuris
- HMAC-MD5 est encore considr scuris pour l'authentification, alors que MD5 est cass pour les collisions
- La cl secrte k protge contre les attaques par collision

Note : Cependant, on recommande SHA-256 par prudence moderne.

1.1.6 Q1.6 : bcrypt est prfrable SHA-256 pour le hachage de mots de passe

Solution

Rponse : VRAI
Justification :
bcrypt est spcifiquement conu pour le hachage de mots de passe, contrairement SHA-256 :

Caractristique	bcrypt	SHA-256
Lenteur intentionnelle	✓	×
Facteur de cot ajustable	✓	×
Sel intgr	✓	×
Rsistant GPU/ASIC	✓	×

Exemple :

- SHA-256 : $\sim 10^9$ hashes/sec sur GPU
- bcrypt (cost=12) : ~ 10 hashes/sec

Conclusion : Toujours utiliser bcrypt, scrypt, Argon2 ou PBKDF2 pour les mots de passe.

1.1.7 Q1.7 : Un firewall stateful peut dtecter les attaques au niveau applicatif

Solution

Rponse : FAUX
Justification :
Un firewall stateful maintient l'tat des connexions TCP/UDP (couches 3-4 OSI) mais ne comprend pas les protocoles applicatifs (couche 7) :
Firewall Stateful :

- Suit les connexions TCP (SYN, SYN-ACK, ACK)
- Autorise les paquets de connexions tablies
- Ne regarde PAS le contenu des donnes

Dtection attaques applicatives :

- Injection SQL : Ncessite un WAF (Web Application Firewall)
- XSS : Ncessite un WAF
- Malware HTTP : Ncessite un IPS/IDS

Conclusion : Pour la scurit applicative, on a besoin de WAF, IDS/IPS, ou proxies applicatifs.

1.1.8 Q1.8 : Un buffer overflow peut tre exploit pour excuter du code arbitraire

Solution

Rponse : VRAI
Justification :
Le buffer overflow est une vulnrabilit classique permettant l'excution de code arbitraire :
Mcanisme :

Stack avant overflow:
+----------------+
— Return Address — ¡- Cible
+----------------+
— Saved EBP —
+----------------+
— buffer[64] — ¡- Dbordement
+----------------+

Stack aprs overflow:
+----------------+
— @shellcode — ¡- Return cras !
+----------------+
— NOP NOP NOP —
+----------------+
— Shellcode — ¡- Code malveillant
+----------------+

Exploitation :

1. Attaquant crit au-del du buffer
2. crase l'adresse de retour avec adresse du shellcode
3. Fonction retourne vers le shellcode
4. Shellcode s'excute avec les privilges du programme

Dfenses modernes : Stack canaries, NX bit, ASLR, ROP.

1.1.9 Q1.9 : ASLR (Address Space Layout Randomization) rend les buffer overflows impossibles

Solution

Rponse : FAUX
Justification :
ASLR rend l'exploitation plus difficile mais pas impossible :
Ce que fait ASLR :

- Randomise les adresses mmoire (stack, heap, bibliothques)
- Force l'attaquant deviner les adresses
- Rduit la probabilit de succs d'une attaque

Contournements possibles :

- Information leak : Fuites d'adresses mmoire
- Brute force : Sur systmes 32-bit (entropie limite)
- ROP (Return-Oriented Programming) : Utilise du code existant
- JIT Spray : Dans les navigateurs

Conclusion : ASLR est une dfense importante mais doit tre combine avec NX, canaries, et code scuris.

1.1.10 Q1.10 : Le chiffrement de bout en bout (E2EE) protge contre les attaques man-in-the-middle

Solution

Rponse : FAUX (avec nuance)
Justification :
Le E2EE (End-to-End Encryption) seul ne protge PAS automatiquement contre les MITM :
Problme :

- E2EE chiffre les messages entre Alice et Bob
- Mais comment Alice sait-elle qu'elle parle vraiment Bob ?
- Un attaquant peut tablir deux connexions E2EE spares :
  - Alice $\leftrightarrow$ Attaquant (E2EE avec cl k_1)
  - Attaquant $\leftrightarrow$ Bob (E2EE avec cl k_2)

Protection contre MITM :

- Authentification des cls : Vrification fingerprints (Signal, WhatsApp)
- PKI : Certificats signs par autorit de confiance
- Key transparency : Audits publics des cls

Conclusion : E2EE + authentification forte = protection MITM.

1.2 Question 2 : Rainbow Tables (5 points)

1.2.1 nonc

Comparez les Rainbow Tables et les Tables de Hellman classiques pour le craquage de mots de passe. Expliquez :

1. Le principe de fonctionnement de chaque mthode
2. Les avantages des Rainbow Tables

3. L'impact du salage sur ces attaques

1.2.2 Solution

Solution

1. Principe de fonctionnement
Tables de Hellman classiques :

- Utilise une seule fonction de rduction R
- Cre des chanes : $p_0 \xrightarrow{H} h_1 \xrightarrow{R} p_1 \xrightarrow{H} h_2 \xrightarrow{R} \dots$
- Stocke seulement $(p_{\text{dbut}}, p_{\text{fin}})$ pour chaque chane
- Problme : Collisions de chanes (merging chains)

Exemple chanes Hellman :

Chane 1: password -¿ H -¿ a3f5... -¿ R -¿ admin123 -¿ ...
Chane 2: letmein -¿ H -¿ a3f5... -¿ R -¿ admin123 -¿ ...
^
Collision ! Les chanes fusionnent

Rainbow Tables :

- Utilise une fonction de rduction diffrente chaque tape : $R_1, R_2, \dots, R_t$
- Chanes : $p_0 \xrightarrow{H} h_1 \xrightarrow{R_1} p_1 \xrightarrow{H} h_2 \xrightarrow{R_2} p_2 \xrightarrow{H} \dots$
- Avantage : limine les collisions de chanes !

Exemple chanes Rainbow :

Chane 1: password -¿ H -¿ a3f5... -¿ R -¿ admin123 -¿ H -¿ b2e8... -¿ R -¿ ...
Chane 2: letmein -¿ H -¿ a3f5... -¿ R -¿ user4567 -¿ H -¿ c9d3... -¿ R -¿ ...
^ ^
Mme hash Diffrents mots (R diffrent)
Pas de fusion !

2. Avantages des Rainbow Tables

1. Pas de collisions de chanes :
   - Hellman : $\sim 50\%$ des chanes peuvent fusionner
   - Rainbow : Collisions limines par R_i diffrents
2. Meilleur taux de succs :
   - Pour mme espace mmoire, Rainbow couvre plus de mots de passe
   - Efficacit : $\sim 2\times$ meilleure que Hellman
3. Recherche plus rapide :
   - Moins de fausses alertes dues aux collisions
   - Recherche : $O(t^2)$ oprations pour une table de longueur t

Compromis temps-mmoire :

Mthode	Taux succs	Vitesse recherche
Tables Hellman	$\sim 55\%$	Moyenne
Rainbow Tables	$\sim 99.9\%$	Rapide

3. Impact du salage
Le salage rend ces attaques totalement inefficaces :
Sans sel :

password1 -¿ SHA256 -¿ 5e884898da...
password2 -¿ SHA256 -¿ 5e884898da... (mme hash !)
^
Rainbow table prcalcule peut casser les deux

1.3 Question 3 : Buffer Overflow avec Protection (5 points)

1.3.1 nonc

Considrez le code C suivant avec une variable de garde (canary-like) :

```
void vulnerable(char *input) –
    int guard = 0xDEADBEEF;
    char buffer[64];
    strcpy(buffer, input);

    if (guard != 0xDEADBEEF) –
        printf("Buffer overflow detected!“n”);
        exit(1);
    ″

    printf("Input: %s“n", buffer);
″
```

Questions :

1. Dessinez le stack frame de cette fonction
2. Un attaquant peut-il contourner cette protection ? Si oui, comment ?
3. Quelles amliorations proposeriez-vous ?

1.3.2 Solution

Solution

1. Stack frame
Layout mmoire (x86 32-bit) :

Adresses hautes
+------------------+ ¡- EBP + 8
— Argument: input — (pointeur)
+------------------+ ¡- EBP + 4
— Return Address — ¡- Cible ultime de l'attaque
+------------------+ ¡- EBP
— Saved EBP —
+------------------+ ¡- EBP - 4
— guard (0xDEADBEEF)— ¡- "Canary" maison
+------------------+ ¡- EBP - 8
— buffer[60-63] —
— buffer[56-59] —
— ... —
— buffer[0-3] — ¡- Dbut buffer (EBP - 72)
+------------------+ ¡- ESP
Adresses basses

Calcul des offsets :

- Buffer : EBP - 72 EBP - 9 (64 bytes)
- Guard : EBP - 8 EBP - 5 (4 bytes, valeur 0xDEADBEEF)
- Saved EBP : EBP - 4 EBP - 1
- Return Address : EBP + 4 EBP + 7

2. Contournement de la protection
OUI, l'attaquant peut contourner facilement !
Mthode 1 : Prserver le guard
L'attaquant construit un payload qui :

1. Remplit le buffer (64 bytes)
2. Rcrit le guard avec la bonne valeur (0xDEADBEEF)
3. Continue craser saved EBP et return address

Payload :

[64 bytes de padding] + [0xDEADBEEF] + [4 bytes padding] + [@shellcode]
—¡-- buffer --¿— —¡- guard -¿— —¡- saved EBP -¿— —¡- ret addr-¿—

Exemple concret :

```
payload = b'A' * 64                    # Remplir buffer
payload += b'"xEF"xBE"xAD"xDE'         # Prserver guard (little-endian)
payload += b'B' * 4                    # craser saved EBP
payload += b'"x10"xD0"xFF"xBF'         # Adresse shellcode
payload += shellcode                   # Code malveillant
```

Mthode 2 : Attaque avant la vrification
L'attaquant peut exploiter le printf avant la vrification :

- Format string attack sur printf("Input: %s"n", buffer)
- Fuite d'adresses mmoire
- Exploitation indirecte

3. Amliorations proposes

1. Utiliser les stack canaries du compilateur :

2 Examen Janvier 2025

2.1 Question 1 : Scurit PRG (7 points)

2.1.1 nonc

Partie A : Donnez la dfinition formelle de la scurit d'un PRG (Pseudo-Random Generator).
Partie B : Considrez le PRG suivant dfini sur $\{0,1\}^{128}$:

$$G(s) = s \| \text{AES}_s(0^{128})$$

o s est une cl de 128 bits et AES_s est le chiffrement AES avec la cl s.

Est-ce un PRG scuris ? Justifiez votre rponse.

Partie C : Proposez un PRG manifestement inscuris et prouvez qu'il ne satisfait pas la dfinition de scurit.

2.1.2 Solution

Solution

Partie A : Dfinition formelle
Un PRG (Pseudo-Random Generator) est une fonction dterministe :

$$G : \{0,1\}^s \to \{0,1\}^n$$

avec $n > s$ (expansion), qui est scuris si pour tout adversaire polynomial probabiliste A :

$$\text{PRGadv}[A, G] = \left| \Pr_{k \leftarrow \{0,1\}^s}[A(G(k)) = 1] - \Pr_{r \leftarrow \{0,1\}^n}[A(r) = 1] \right| < \varepsilon$$

o ε est ngligeable en s.
Interprtation :

- Exprience 0 : A reoit $G(k)$ avec k alatoire (pseudo-alatoire)
- Exprience 1 : A reoit r vraiment alatoire
- A doit deviner quelle exprience (sortie 0 ou 1)
- Si G est scuris : avantage de A est ngligeable

Ngligeable : $\varepsilon(s) = o(1/s^c)$ pour tout c, par exemple $\varepsilon(s) = 2^{-s/2}$.
Partie B : Analyse de $G(s) = s \| \text{AES}_s(0^{128})$
Rponse : NON, ce n'est PAS un PRG scuris.
Attaque par distingueur :
Algorithme distingueur A :

1. Recevoir $w \in \{0,1\}^{256}$
2. Parser $w = w_1 \| w_2$ avec $|w_1| = |w_2| = 128$
3. Calculer $c = \text{AES}_{w_1}(0^{128})$
4. Si $c = w_2$: retourner 1 (pseudo-alatoire)
5. Sinon : retourner 0 (alatoire pur)

Analyse des probabilits :
Si $w = G(s)$ (pseudo-alatoire) :

- $w = s \| \text{AES}_s(0^{128})$
- Donc $w_1 = s$ et $w_2 = \text{AES}_s(0^{128})$
- Test : $\text{AES}_{w_1}(0^{128}) = \text{AES}_s(0^{128}) = w_2$
- $\Pr[A(G(s)) = 1] = 1$

Si $w = r$ (alatoire pur) :

- $w = r_1 \| r_2$ avec r_1, r_2 indpendants et alatoires
- Test : $\text{AES}_{r_1}(0^{128}) \stackrel{?}{=} r_2$
- Probabilit de collision : $\Pr[\text{AES}_{r_1}(0^{128}) = r_2] = 1/2^{128}$ (ngligeable)
- $\Pr[A(r) = 1] \approx 0$

Avantage du distingueur :

$$\text{PRGadv}[A, G] = |1 - 0| = 1$$

Cet avantage est non ngligeable, donc G n'est PAS scuris.

Point important

Problme : Exposer la cl s en clair permet l'adversaire de calculer $\text{AES}_s(0^{128})$ lui-mme et de dtecter la structure.

Partie C : PRG manifestement inscuris
Proposition : PRG qui retourne les 128 premiers bits zro :

2.2 Question 2 : IDS et Base-Rate Fallacy (6 points)

2.2.1 nonc

Un IDS (Intrusion Detection System) a les caractristiques suivantes :

- Taux de dtection (True Positive Rate) : TPR = 99%
- Taux de faux positifs (False Positive Rate) : FPR = 1%

Dans un rseau o seulement 0.1% du trafic est malveillant :

Question : Si l'IDS dclenche une alerte, quelle est la probabilit que ce soit rellement une attaque ?

Utilisez le thorme de Bayes et expliquez le phnomne de "base-rate fallacy".

2.2.2 Solution

Solution

Donnes :

- Pr[Attaque] = 0.001 (0.1% du trafic est malveillant)
- Pr[Normal] = 0.999 (99.9% du trafic est lgitime)
- Pr[Alerte|Attaque] = 0.99 (TPR)
- Pr[Alerte|Normal] = 0.01 (FPR)

Question : Calculer Pr[Attaque|Alerte] (probabilit qu'une alerte soit une vraie attaque).
Thorme de Bayes :

$$\Pr[\text{Attaque}|\text{Alerte}] = \frac{\Pr[\text{Alerte}|\text{Attaque}] \times \Pr[\text{Attaque}]}{\Pr[\text{Alerte}]}$$

tape 1 : Calculer Pr[Alerte]
Par la loi des probabilits totales :

$$\Pr[\text{Alerte}] = \Pr[\text{Alerte}|\text{Attaque}] \times \Pr[\text{Attaque}] + \Pr[\text{Alerte}|\text{Normal}] \times \Pr[\text{Normal}]$$

$$\Pr[\text{Alerte}] = 0.99 \times 0.001 + 0.01 \times 0.999$$

$$\Pr[\text{Alerte}] = 0.00099 + 0.00999 = 0.01098$$

tape 2 : Appliquer Bayes

$$\Pr[\text{Attaque}|\text{Alerte}] = \frac{0.99 \times 0.001}{0.01098} = \frac{0.00099}{0.01098} \approx 0.0901$$

Rsultat :

Point important

Probabilit qu'une alerte soit une vraie attaque : $\approx 9\%$ seulement !
Malgr un IDS avec 99% de prcision, 91% des alertes sont des faux positifs.

Tableau rcapitulatif :
Sur 100,000 connexions :

	Attaque relle	Trafic normal	Total
Alerte dclenche	99 (TP)	999 (FP)	1098
Pas d'alerte	1 (FN)	98,901 (TN)	98,902
Total	100	99,900	100,000

Mtriques :

- Vrais positifs (TP) : $100 \times 0.99 = 99$
- Faux ngatifs (FN) : $100 \times 0.01 = 1$
- Faux positifs (FP) : $99{,}900 \times 0.01 = 999$
- Vrais ngatifs (TN) : $99{,}900 \times 0.99 = 98{,}901$

Prcision positive (PPV) :

$$\text{PPV} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{99}{99 + 999} = \frac{99}{1098} \approx 0.09 = 9\%$$

Le phnomne de "Base-Rate Fallacy"
La base-rate fallacy est une erreur cognitive qui consiste ignorer la prvalence de base (taux de base) d'un vnement.

2.3 Question 3 : Dessin de Stack Frame (7 points)

2.3.1 nonc

Considrez le code C suivant (architecture x86 32-bit) :

```
void process(int x, char *data) –
    char buffer[32];
    int counter = 0;
    strcpy(buffer, data);
    printf("Counter: %d“n", counter);
″

int main() –
    char input[100];
    gets(input);
    process(42, input);
    return 0;
″
```

Questions :

1. Dessinez le stack frame complet de process() avec toutes les adresses relatives EBP
2. Calculez l'offset exact pour atteindre l'adresse de retour depuis le dbut de buffer
3. Proposez un payload d'exploitation complet (en supposant ASLR dsactiv)
4. Listez toutes les vulnrabilits prsentes dans ce code

2.3.2 Solution

Solution

1. Stack frame de process()
Layout mmoire dtaill (x86 32-bit) :

Adresses hautes
+----------------------+ ¡- EBP + 12
— Argument: data — (char* pointer, 4 bytes)
+----------------------+ ¡- EBP + 8
— Argument: x — (int, valeur 42, 4 bytes)
+----------------------+ ¡- EBP + 4
— Return Address — ¡- CIBLE de l'attaque (4 bytes)
+----------------------+ ¡- EBP (Saved EBP)
— Saved EBP — (4 bytes)
+----------------------+ ¡- EBP - 4
— counter (int) — (valeur 0, 4 bytes)
+----------------------+ ¡- EBP - 8
— padding (alignment) — (0 bytes si compilateur aligne)
+----------------------+ ¡- EBP - 8
— buffer[28-31] —
— buffer[24-27] —
— buffer[20-23] —
— buffer[16-19] —
— buffer[12-15] —
— buffer[8-11] —
— buffer[4-7] —
— buffer[0-3] — ¡- Dbut buffer (EBP - 40)
+----------------------+ ¡- ESP (stack pointer courant)
Adresses basses

Dtails des offsets depuis EBP :

lment	Offset depuis EBP	Taille
Argument data	EBP + 12	4 bytes
Argument x	EBP + 8	4 bytes
Return Address	EBP + 4	4 bytes
Saved EBP	EBP	4 bytes
counter	EBP - 4	4 bytes
buffer[32]	EBP - 40 EBP - 9	32 bytes

Note : Le compilateur peut ajouter du padding pour aligner les variables. Ici, on suppose un alignement standard.
2. Calcul de l'offset
Distance du dbut de buffer return address :

- Dbut de buffer : EBP - 40
- Return address : EBP + 4
- Offset = (EBP + 4) - (EBP - 40) = 44 bytes

Dcomposition :

— 32 bytes buffer — 4 bytes padding/counter — 4 bytes saved EBP — 4 bytes ret addr —
—¡-- buffer --¿——¡----- 8 bytes gap ----¿——¡--- target ---¿—

Vrification :

- Buffer : 32 bytes
- Counter : 4 bytes (EBP - 4)

3 Annexe : Ressources et Rfrences

3.1 Commandes de Vrification

Vrifier les protections d'un binaire :

```
# Linux
checksec --file=./program

# Manual check
readelf -l ./program — grep GNU˙STACK  # NX bit
readelf -d ./program — grep BIND˙NOW   # RELRO
```

Dsactiver ASLR (pour tests seulement !) :

```
# Temporaire
echo 0 — sudo tee /proc/sys/kernel/randomize˙va˙space

# Ractiver
echo 2 — sudo tee /proc/sys/kernel/randomize˙va˙space
```

Trouver adresses avec GDB :

```
gdb ./program
(gdb) break process
(gdb) run ¡ input.txt
(gdb) info frame        # Voir EBP, ESP
(gdb) x/64x $esp        # Examiner stack
(gdb) print &buffer     # Adresse buffer
```

3.2 Rfrences

- Cryptographie : "The Joy of Cryptography" (Rosulek)
- Buffer Overflow : "Smashing The Stack For Fun And Profit" (Aleph One)
- Network Security : "Computer Security" (Stallings)
- OWASP Top 10 : https://owasp.org/Top10/
- CWE : https://cwe.mitre.org/