Université de Perpignan
C. Legrand

L3 Informatique
Année 2026

TP Noté Sécurité Informatique

Projet CryptoShell – Ransomware éducatif en Bash

Informations pratiques :

- **Distribution :** 16 février 2026
- **Date de rendu :** 08 mars 2026 à 23h59 sur Moodle
- **Modalités :** Travail **individuel**. Déposer sur Moodle une archive `.zip` contenant :
  - Tous les scripts Bash (Phases 1–6)
  - Le rapport d'analyse (Phase 7) au format PDF
- **Barème :** 25 points ramenés à /20. Chaque phase est évaluée indépendamment.

Avertissement éthique et légal

Les techniques présentées dans ce TP sont étudiées dans un **cadre strictement académique** afin de comprendre les mécanismes utilisés par les logiciels malveillants et ainsi pouvoir mieux s'en protéger.
Toute utilisation de ces techniques en dehors du cadre pédagogique est illégale et passible de sanctions pénales (chapitre III du Code pénal, articles 323-1 à 323-8 – *Des atteintes aux systèmes de traitement automatisé de données*) :

- Accès ou maintien frauduleux dans un STAD (art. 323-1) : **3 ans** d'emprisonnement et **100 000 €** d'amende
- Entrave au fonctionnement d'un STAD (art. 323-2) : **5 ans** d'emprisonnement et **150 000 €** d'amende
- Introduction frauduleuse de données dans un STAD (art. 323-3) : **5 ans** d'emprisonnement et **150 000 €** d'amende

Peines portées à **7 ans** et **300 000 €** lorsque le système visé traite des données à caractère personnel mis en œuvre par l'État, et jusqu'à **10 ans** et **300 000 €** en bande organisée (art. 323-4-1). *Peines mises à jour par la loi LOPMI n° 2023-22 du 24 janvier 2023.*

Scénario : Vous êtes analyste junior dans une équipe de réponse aux incidents (*CERT*). Votre responsable vous demande de comprendre le fonctionnement d'un ransomware récent ciblant des serveurs Linux, puis de développer les outils de détection et de récupération correspondants. Pour cela, vous allez **construire progressivement** un ransomware éducatif en Bash, baptisé **CryptoShell**, puis **changer de posture** pour créer les contre-mesures.

Prérequis : Chapitres 01 (logiciels malveillants) et 02 (virus interprétés), TD01 à TD04.

Méthode de travail : Pour chaque phase, procédez comme suit :

- Exécutez `setup_lab.sh` pour créer un environnement de test propre.
- Travaillez **uniquement** dans le répertoire `lab/` créé par le script.
- Testez chaque phase **avant** de passer à la suivante.
- Recréez l'environnement de test entre chaque phase si nécessaire.

Fichiers fournis :

- `xor_helper.sh` — Script helper contenant les fonctions `generate_key()` et `xor_file()` (à utiliser avec `source`)
- `setup_lab.sh` — Script de mise en place de l'environnement de test
- `cible1.txt`, `cible2.txt`, `cible3.sh` — Fichiers cibles de test
- `rapport_template.tex` — Template du rapport d'analyse (Phase 7)

Helper XOR (`xor_helper.sh`) — ce script vous est fourni en boîte noire :

```
#!/bin/bash
# CryptoShell XOR Helper
# Usage: source xor_helper.sh

# Génère une clé aléatoire de N octets (en hex)
generate_key() {
    local length=${1:-8}
    head -c "$length" /dev/urandom | xxd -p
}

# Chiffre/déchiffre un fichier avec une clé XOR (en hex)
# Usage: xor_file <input_file> <output_file> <hex_key>
xor_file() {
    local input="$1" output="$2" key="$3"
    local key_len=${#key}
    local i=0
    xxd -p "$input" | tr -d '\n' | fold -w2 | while read byte; do
        local ki=$(( i % (key_len / 2) ))
        local key_byte="${key:$((ki*2)):2}"
        printf "%02x" $(( 16#$byte ^ 16#$key_byte ))
        i=$((i + 1))
    done | xxd -r -p > "$output"
}
```

xor_helper.sh (fourni)

Barème détaillé :

Phase	Intitulé	Durée	Points	Difficulté
1	Reconnaissance	30 min	3 pts	⋆⋆
2	Chiffrement XOR	45 min	4 pts	⋆⋆⋆
3	Propagation et rançon	45 min	4 pts	⋆⋆⋆
4	Déclencheur	30 min	3 pts	⋆⋆
5	Furtivité	30 min	3 pts	⋆⋆⋆
6	Détection et récupération	45 min	4 pts	⋆⋆⋆
7	Rapport d'analyse	45 min	4 pts	⋆⋆⋆⋆
	Total	**∼4h30**	**25 pts → /20**	

Exercice 1 (Projet CryptoShell — Ransomware éducatif)**.**

Phase 1 — Reconnaissance et découverte de fichiers **[3 points]**

Rappel de cours

Classification d'Adleman et cycle de vie d'un virus :
Leonard Adleman (1988) a proposé une classification formelle des programmes malveillants :

- **Programmes simples :** exécutent leur action sans se reproduire (bombe logique, ransomware pur, spyware).
- **Programmes auto-reproducteurs :** se copient dans d'autres programmes ou systèmes (virus, vers).

Le **cycle de vie** d'un virus comporte 3 phases :

1. **Infection** : le virus se copie dans de nouveaux hôtes (routine de recherche + copie)
2. **Incubation** : le virus est présent mais inactif (attente du déclencheur)
3. **Maladie** : la charge utile s'exécute (action destructrice)

La **routine de recherche de fichiers** est la première étape de tout malware ciblant des fichiers. Elle détermine la **portée** de l'attaque : plus elle est efficace (récursive, multi-extensions), plus l'impact est important.

1a) Écrire un script `recon.sh` qui prend un répertoire en argument et affiche tous les fichiers `.txt` et `.dat` qu'il contient (récursivement). Pour chaque fichier, afficher : le chemin, la taille en octets, la date de modification. Utiliser `find` ou un parcours récursif avec une boucle.

1b) Modifier le script pour que le nombre total de fichiers trouvés et la taille totale soient calculés. Stocker la liste des fichiers cibles dans un fichier `targets.list`.

1c) **Question théorique :** Dans la classification d'Adleman, où se situe un ransomware ? Est-ce un programme simple ou auto-reproducteur ? Justifier.

Phase 2 — Chiffrement XOR [4 points]

Rappel de cours

Chiffrement XOR et polymorphisme viral :
Le XOR (ou exclusif) possède 4 propriétés algébriques fondamentales :

1. **Commutativité :** $A \oplus B = B \oplus A$
2. **Associativité :** $(A \oplus B) \oplus C = A \oplus (B \oplus C)$
3. **Élément neutre :** $A \oplus 0 = A$
4. **Auto-inverse :** $A \oplus A = 0$

Conséquence pour le chiffrement : Si $C = M \oplus K$ (chiffrement), alors $C \oplus K = M \oplus K \oplus K = M \oplus 0 = M$ (déchiffrement). La **même opération** avec la **même clé** permet de chiffrer ET déchiffrer.

Lien avec le chiffrement de Vernam : Le XOR avec une clé aussi longue que le message et véritablement aléatoire constitue le **masque jetable** (*one-time pad*), prouvé incassable par Shannon (1949).

Polymorphisme viral par chiffrement XOR : Dans un virus polymorphe, le code viral est chiffré avec une clé XOR différente à chaque infection. Seule la **routine de déchiffrement** (en clair) reste constante — c'est un **invariant détectable**.

Table de vérité du XOR :

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

2a) Expliquer mathématiquement pourquoi XOR permet de chiffrer ET déchiffrer avec la même opération et la même clé. Donner les 4 propriétés algébriques utilisées.

2b) Calculer manuellement le résultat du XOR de la chaîne `"HACK"` avec la clé `0x5A`. Fournir le tableau complet (caractère, ASCII hex, clé hex, résultat hex, résultat caractère).

2c) En utilisant `xor_helper.sh`, écrire un script `encrypt.sh` qui : (1) génère une clé aléatoire de 8 octets, (2) chiffre un fichier passé en argument, (3) renomme le fichier chiffré avec l'extension `.locked`, (4) supprime l'original, (5) sauvegarde la clé dans un fichier caché `.cryptoshell_key`.

2d) Écrire `decrypt.sh` qui : lit la clé depuis `.cryptoshell_key`, déchiffre un fichier `.locked`, restaure le nom original, et supprime le fichier `.locked`.

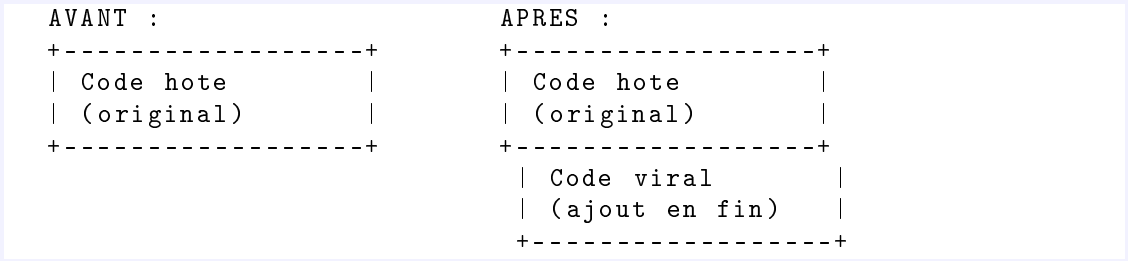
Phase 3 — Propagation et rançon [4 points]

Rappel de cours

Techniques d'infection classiques et anti-surinfection :
Les virus classiques utilisent 3 techniques principales d'infection de fichiers :

- **Écrasement :** le virus remplace le début du fichier hôte (destructif, simple)
- **Ajout en fin :** le virus s'ajoute après le code hôte (non destructif)
- **Compagnon :** le virus crée un fichier portant le même nom mais avec une priorité d'exécution plus élevée

Schéma textuel avant/après infection par ajout :



Anti-surinfection : Un virus doit éviter de réinfecter un fichier déjà infecté. Stratégies : marqueur (signature), comparaison tail, exécution test.

3a) Écrire `cryptoshell.sh` qui : (1) génère une clé aléatoire, (2) parcourt récursivement un répertoire cible, (3) chiffre chaque fichier `.txt` et `.dat` trouvé, (4) renomme en `.locked`, (5) dépose un fichier `RANSOM_NOTE.txt` dans chaque répertoire contenant des fichiers chiffrés.

3b) Ajouter un mécanisme d'**anti-surinfection** : le script ne doit PAS re-chiffrer un fichier déjà `.locked`. Implémenter cette vérification.

3c) La note de rançon doit contenir : un message menaçant, un identifiant unique de la victime (généré aléatoirement), et des instructions (fictives). Écrire la fonction `create_ransom_note()`.

3d) **Question théorique :** Comparer la technique d'infection d'un ransomware avec les 3 techniques classiques de virus (écrasement, ajout, compagnon). Quelles sont les similitudes et différences fondamentales ?

Phase 4 — Déclencheur (bombe logique) [3 points]

Rappel de cours

Bombes logiques et déclencheurs :
Une **bombe logique** (*logic bomb*) est un programme dont la charge utile ne s'active que lorsqu'une **condition** est remplie. Types de déclencheurs :

- **Temporel :** date, heure, jour de la semaine
- **Par compteur :** après N exécutions
- **Événementiel :** présence d'un fichier, connexion réseau, login utilisateur
- **Environnemental :** variables d'environnement, nom de machine

Commandes utiles :

- `date +%u` : jour de la semaine (1=lundi, ..., 7=dimanche)
- `date +%d` : jour du mois (01–31)
- `date +%m` : mois (01–12)

Exemples historiques : Omega Engineering (1996) — un administrateur licencié a planté une bombe logique qui a supprimé tous les logiciels de production 20 jours après son départ. Dark Seoul (2013) — malware activé à une date/heure précise ciblant des banques et médias sud-coréens.

Lien avec les virus : Un virus avec un déclencheur différé est une bombe logique montée sur un vecteur viral. La séparation déclencheur/charge utile est un **pattern architectural** fondamental.

4a) Modifier `cryptoshell.sh` pour que le chiffrement ne s'active que si une **condition temporelle** est remplie : le script doit vérifier si nous sommes un dimanche (`date +%u` renvoie 7). Sinon, le script se termine silencieusement.

4b) Implémenter un **déclencheur alternatif par compteur** : un fichier caché `.cs_count` est incrémenté à chaque exécution. Le chiffrement ne s'active qu'après 5 exécutions. Après activation, le compteur est supprimé.

4c) **Question théorique :** Citer deux exemples historiques de bombes logiques célèbres. Expliquer pourquoi les antivirus ont du mal à détecter les bombes logiques avant leur déclenchement.

Phase 5 — Furtivité et anti-détection [3 points]

Rappel de cours

Techniques de furtivité (*stealth*) :
Un malware furtif cherche à dissimuler les traces de son activité pour retarder la détection :

- **Préservation des horodatages :** la modification d'un fichier change sa date de dernière modification (`mtime`). Un administrateur surveillant les dates peut détecter une anomalie.
- **Nettoyage des fichiers temporaires :** tout artefact laissé par le malware (fichiers temporaires, logs) est une preuve.
- **Suppression des erreurs :** les messages d'erreur sur `stderr` peuvent alerter l'utilisateur.

Commandes clés :

- `touch -r reference fichier` : applique les horodatages de `reference` à `fichier`
- `stat -format="%y" fichier` : affiche la date de modification
- `trap "commandes" EXIT SIGINT SIGTERM` : exécute `commandes` à la sortie du script (même en cas d'interruption Ctrl+C ou kill)
- `2>/dev/null` : redirige `stderr` vers `/dev/null`

5a) Ajouter la **préservation des horodatages** : avant de chiffrer chaque fichier, sauvegarder sa date de modification, puis la restaurer sur le fichier `.locked`. Utiliser `touch -r` pour copier les horodatages.

5b) Ajouter un **mécanisme de nettoyage** : utiliser `trap` pour garantir la suppression de tous les fichiers temporaires (clé, fichiers intermédiaires) même en cas d'interruption (SIGINT, SIGTERM). Rediriger stderr avec `2>/dev/null` sur les commandes critiques.

5c) **Question théorique :** Le polymorphisme par réordonnancement de lignes (vu au TD04) et le polymorphisme par chiffrement XOR (vu au TD02) visent le même objectif : échapper à la détection par signature. Comparer ces deux approches : lequel est plus robuste ? Lequel laisse un invariant détectable ? Un antivirus pourrait-il détecter CryptoShell malgré ces techniques ?

Phase 6 — Détection et récupération [4 points]

Rappel de cours

Indicateurs de compromission (IoC) et types de détection :
Un **indicateur de compromission** (*Indicator of Compromise*, IoC) est un artefact observable qui signale une infection ou une intrusion.

Types de détection antivirus :

- **Par signature** (*pattern matching*) : recherche de chaînes de caractères ou patterns connus dans les fichiers (rapide, précis, mais contournable par polymorphisme)
- **Heuristique** : analyse de code suspect sans signature connue — détection de patterns comportementaux dans le code source (ex : un script qui chiffre massivement des fichiers)
- **Comportementale** : surveillance des actions **à l'exécution** — détection de l'intention du programme (ex : accès massif aux fichiers + écriture de fichiers chiffrés)

Commandes `grep` utiles pour l'analyse :

- `grep -q "pattern" fichier` : mode silencieux, code de sortie 0 si trouvé
- `grep -r "pattern" répertoire` : recherche récursive
- `grep -l "pattern" *.sh` : liste les fichiers contenant le pattern
- `grep -c "pattern" fichier` : compte les occurrences

6a) Écrire `scanner.sh` qui parcourt un répertoire et détecte les **indicateurs de compromission (IoC)** de CryptoShell :

- Présence de fichiers `.locked`
- Présence de `RANSOM_NOTE.txt`
- Présence de fichiers cachés `.cs_count`, `.cryptoshell_key`
- Patterns suspects dans les scripts : `xor`, `locked`, `RANSOM`

Le scanner doit afficher un rapport synthétique avec le nombre d'IoC trouvés par catégorie.

6b) Écrire `recovery.sh` qui : (1) recherche le fichier `.cryptoshell_key`, (2) si trouvé, déchiffre tous les fichiers `.locked` du répertoire, (3) supprime les notes de rançon et les fichiers d'état.

6c) **Question théorique :** Remplir un tableau d'IoC pour chaque phase (1–5) avec : la technique utilisée, l'IoC détectable, le type de détection (signature, heuristique, comportemental). Proposer une stratégie de détection combinant les 3 approches.

Phase 7 — Rapport d'analyse [4 points]

Un template de rapport (`rapport_template.tex`) vous est fourni. Remplissez les sections suivantes :

R1) Classification d'Adleman : Où se situe CryptoShell dans la classification ? Programme simple ou auto-reproducteur ? Justifier.

R2) Cycle de vie : Dessiner (en ASCII art ou schéma) le cycle de vie complet de CryptoShell en identifiant les 3 phases classiques (infection, incubation, maladie). Indiquer quelle phase du TP correspond à chaque étape.

R3) Virus, Ver ou Cheval de Troie : CryptoShell tel qu'implémenté est-il un virus, un ver, ou un cheval de Troie ? Comment le modifier pour qu'il devienne un ver (propagation réseau) ? Comment le transformer en cheval de Troie ?

R4) Comparaison avec les virus de documents : Un macro-virus Word et CryptoShell ciblent tous deux des fichiers de données. Quelles sont les similitudes et différences fondamentales (vecteur d'infection, format cible, interpréteur, persistance) ?

R5) Tableau des IoC par phase : Remplir un tableau complet [Phase | Technique | IoC | Type de détection].

R6) Réflexion éthique : En quoi l'étude de la construction de malware dans un cadre académique est-elle bénéfique pour la cybersécurité ? Citer au moins un exemple concret.

Le rapport est évalué sur la qualité de l'analyse, pas sur la longueur.