pwntools

INF600C

Sécurité des logiciels et exploitation de vulnérabilités

Philippe Pépos Petitclerc

Hiver 2024

Université du Québec à Montréal

Introduction

Pwntools

Projet

Pwntools is a CTF framework and exploit development library. Written in Python, it is designed for rapid prototyping and development, and intended to make exploit writing as simple as possible.

- Github
- Documentation
- Tutoriel

Installation

```
1  apt-get update
2  apt-get install python3 python3-pip python3-dev git libssl-dev \
3      libffi-dev build-essential
4  python3 -m pip install --upgrade pip
5  python3 -m pip install --upgrade pwntools
```

workspace

`pwntools` y est déjà installé.

Prologue

```python
from pwn import *  # import à la racine

# contexte global pour la suite des operations
context(arch='amd64', os='linux')  # arch='i386' pour 32-bits

...
```

Contexte

Un objet global qui défini les configurations d'exécution utilisées par `pwntools`.

Documentation

Communiquer avec le monde

Tubes

Abstraction qui offre un API commun pour communiquer avec différents type de processus.

- Processus locaux
- Sockets
- Processus distant via SSH
- etc.

Exemples

```
1  io = process('./chal')  # processus local
2  io = remote('workspace.kaa', 80)  # sockets TCP
```

Lire et écrire

- `send(data)`: Envoie `data` au processus.
- `sendline(data)`: Envoie `data` et un saut de ligne au processus.
- `sendlineafter(delim, data)`: Envoie `data` et un saut de ligne au processus après avoir reçu `delim`
- `recv()`: Retourne un paquet de données du processus.
- `recvline()`: Retourne une ligne de données du processus.
- `recvuntil(delim)`: Retourne toute la donnée reçue jusqu'à `delim`.
- `interactive()`: Bascule en mode interactif.

RTFM pour le reste de l'API

Note

Attention au `bytes`. C'est du Python 3.

Exemple

TODO: Exemple avec prog

Quelques utilitaires

pack et unpack

- Pour coder et décoder des nombres en leur représentation binaire.
- Essentiellement la même chose que `struct.pack` et `struct.unpack`, mais plus simple.
- Repose sur la configuration du contexte global pour choisir le boutisme par défaut.

API

- `p8`, `p16`, `p32`, `p64`: Code l'argument en binaire et retourne les octets.
- `u8`, `u16`, `u32`, `u64`: Décode un entier des octets des l'argument.

Exemple d'utilisation

```
>>> context.endian = 'little'
>>> n = 0xc0fe
>>> p16(n)
b'\xfe\xc0'
>>> u32(b'\xfe\xc0\x00\x00') == n
True
```

Suites de de Bruijn

- Utilitaires de suite d'octets uniques
- Analogue à `pattern_create`, `pattern_find` dans *Peda*.
- `cyclic(n)` pour générer
- `cyclic_find(b'ABAA')` pour trouver le décalage
- `cyclic_find(0x61616162)` fonctionne aussi. (Utilise `context` pour le boutisme)

Assembleur et Shellcraft

Assembler et désassembler

Pwntools offre un API simple pour assembler et désassembler.

- `asm(assembly)` Assemble `assembly` et retourne les octets du code machine.

```
asm('nop ; nop')
>>> b'\x90\x90'
```

- `disasm(machinecode)` Désassemble `machinecode` et retourne les instructions assembleurs.

```
c = b'\x90\x90'
print(disasm(c))
    0:   90                      nop
    1:   90                      nop
```

Note

Les informations d'architecture proviennent du contexte global.

Shellcraft

Pwntools a une librairie de *shellcodes* préfabriqués que vous pouvez choisir d'utiliser. RTFM pour voir ce qui existe.

```
1  sc = shellcraft.sh()  # Attention au context()
2
3  # Instructions d'un shellcode pour `execve("/bin/sh")`
4  print(sc)
5
6  # On l'assemble pour avoir le code octet à utiliser dans un payload.
7  sc = asm(sc)
8
9  payload = b'A' * 32 + p32(0xffff8588) + b'\x90' * 100 + sc
10 io.send(payload)
```

Analyse de fichiers ELF

Objet ELF

Chargement

```
>>> e = ELF("./libc.so")
```

Symboles

```
>>> e.symbols  # Dictionnaire des symboles et adresses
>>> hex(e.symbols['system'])  # Adresse de `system` dans le fichier
>>> e.got  # Fonctionne aussi pour la GOT
>>> e.plt  # Fonctionne aussi pour la PLT
```

RTFM

Documentation

Outils de débogage

Module GDB

- Requiert d'avoir `gdb` et `gdbserver` installés sur le système
- Fonctionne bien localement avec un terminal graphique, ou par ssh dans une session `tmux`
- `io = gdb.debug("/bin/true", gdbscript="continue")` pour déboguer un processus local
  - Lance le processus à l'intérieur de GDB.
- `io = process("./chal")` puis `gdb.attach(io, "continue")`
  - Lance le processus sans GDB, puis s'attache avec `gdb -p PID`

Conclusion

Conclusion

- Pwntools est une librairie très puissante et très pratique.
- Ne laissez pas une librairie nuire à votre compréhension.
- RTFM