pwntools / pwndbg for starters

A gentle introduction to tooling (and more, this is a trap!)

Workshop room 1

hanemile - 2025-02-28

Theory

Practice

Nerd

Hacker™

Skriptkiddie

Pwntools, wasn’ das?

https://ctf.emile.space

Interaction, Automation

No communication, no hack!

Erste Challenge: http://138.199.213.51:31337

```
; ./flag_guesser
…
```

```
; nc 138.199.213.51 31337
…
```

```
; # Just use whatever floats your boat^Wpwning env
; # https://github.com/Gallopsled/pwntools

; nix flake init -t https://github.com/hanemile/hefe#ctf
…

; nix develop
…
```

```
; $EDITOR hack.py
#!/usr/bin/env python3
from pwn import *

p = process(“./flag_guesser”)
# p = remote(“138.199.213.51”, 31337)

p.interactive()

; python3 hack.py
```

https://ctf.emile.space

pwntools!

All you'll ever need [citation needed]

```
; pwn
usage: pwn [-h]
{asm,checksec,constgrep,cyclic,debug,disasm,disablenx,elfdiff,elfpatch,
errno,hex,libcdb,phd,pwnstrip,scramble,shellcraft,template,unhex,update
,version}
```

asm

context

elf

gdb

rop

shellcraft

tubes

utils

https://ctf.emile.space

```
; pwn template
[*] Automatically detecting challenge binaries...
[+] Found challenge binary '.ape-1.9'
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# This exploit template was generated via:
# $ pwn template
from pwn import *

# Set up pwntools for the correct architecture
exe = context.binary = ELF(args.EXE or './challenge')

# Many built-in settings can be controlled on the command-line and show up
# in "args".  For example, to dump all data sent/received, and disable ASLR
# for all created processes...
# ./exploit.py DEBUG NOASLR

def start(argv=[], *a, **kw):
    '''Start the exploit against the target.'''
    if args.GDB:
        return gdb.debug([exe.path] + argv, gdbscript=gdbscript, *a, **kw)
    else:
        return process([exe.path] + argv, *a, **kw)

# Specify your GDB script here for debugging
# GDB will be launched if the exploit is run via e.g.
# ./exploit.py GDB
gdbscript = '''
continue
'''.format(**locals())
```

```
#===========================================================
#                    EXPLOIT GOES HERE
#===========================================================

io = start()

# shellcode = asm(shellcraft.sh())
# payload = fit({
#     32: 0xdeadbeef,
#     'iaaa': [1, 2, 'Hello', 3]
# }, length=128)
# io.send(payload)
# flag = io.recv(...)
# log.success(flag)

io.interactive()
```

`p.sendline()` `p32()` `u32()` `p.interactive()`

`p.readline()` `p64()` `u64()`

`p.sendlineafter(b“term”, b“content”)`

`; python3 solve.py GDB DEBUG`

`hex(u64(b”\xdd\xcc\xbb\xaa\x00\x00\x00\x00")) => 0xaabbccdd`

`p64(0xaabbccdd) => b'\xdd\xcc\xbb\xaa\x00\x00\x00\x00'`

GDB + Pwndbg = ❤️

https://ctf.emile.space

Introspection, Dynamic analysis

LIVE DEMO!

https://ctf.emile.space