포렌식 도구 이해

> [19반] 정민석_7000

요약

dmp 바이너리 파일에서 `0x64 0x??{6-8} 0x40 0x06 0x??{18} 0x5a 0x0c 0x00[2}` 의 패턴을 정규표현식을 이용하여 찾고 flag를 찾는 실습을 진행하였습니다.

실습 내용

먼저 volatility를 설치하고 환경설정을 합니다.

```
git clone https://github.com/volatilityfoundation/volatility3.git
cd volatility3
git checkout stable
pip install -r requirements.txt
python vol.py --save-config config.json -f OtterCTF.vmem windows.info
```

이후 다음의 명령어를 통하여 키를 찾고 컴퓨터의 정보를 추출합니다

```
python vol.py -c config.json -f OtterCTF.vmem windows.lsadump.Lsadump

# Hostname
python vol.py -c config.json -f OtterCTF.vmem windows.envars.Envars | fin
dstr -i computername
python vol.py -f otterctf.vmem windows.registry.printkey --key controlset0
01\Services\Tcpip\Parameters
python vol.py -f otterctf.vmem windows.registry.printkey --key controlset0
01\Control\ComputerName

# IP
python vol.py  -c config.json -f otterctf.vmem windows.netstat
python vol.py  -c config.json -f otterctf.vmem windows.netscan
```

```
python vol.py  -c config.json -f otterctf.vmem windows.registry.printkey --k
ey controlset00[1-2]\Services\Tcpip\Parameters\interfaces
# Process
python vol.py  -c config.json -f otterctf.vmem windows.pslist
python vol.py  -c config.json -f otterctf.vmem windows.pstree
python vol.py  -c config.json -f otterctf.vmem windows.psscan
```

여기에서 사용자가 Lunar-3라는 708번의 프로세스에서 게임을 실행하고 있는 것을 파악하고, 708번 프로세스의 메모리 덤프를 아래와 같이 생성합니다.

```
python vol.py -c config.json -f OtterCTF.vmem windows.memmap --pid 70
8 --dump
```

이렇게 pid.708.dmp 파일이 생성되었습니다. 여기에서 `0x64 0x??{6-8} 0x40 0x06 0x??{18} 0x5a 0x0c 0x00[2}` 의 패턴 뒤의 문자를 출력해야합니다.

python을 이용하여 dmp파일 중 `0x64 0x??{6-8} 0x40 0x06 0x??{18} 0x5a 0x0c 0x00[2}` 의 패턴을 찾습니다. 해당 코드는 정규표현식을 만들고, 파일을 바이트 형식으로 읽어온 뒤, 일치한 패턴의 뒤 16 바이트를 hex와 문자로 출력하는 기능을 수행합니다.

```
import re

def bytes_to_printable_str(byte_data):
    return ''.join(chr(b) if 0x20 <= b <= 0x7E else '.' for b in byte_data)

def find_pattern_in_dmp(file_path, preview_bytes=16):
    pattern = (
        b'\x64' +           # 0x64
        b'(.{6,8})' +       # 6~8 bytes
        b'\x40\x06' +       # 0x40 0x06
        b'.{18}' +          # 18 bytes
        b'\x5a\x0c\x00\x00'   # 0x5a 0x0c 0x00 0x00
    )

    with open(file_path, 'rb') as f:
        data = f.read()

    match = re.search(pattern, data, re.DOTALL)
```

```
    if not match:
        print("패턴을 찾을 수 없습니다.")
        return

    offset = match.start()
    end = match.end()
    follow_bytes = data[end:end + preview_bytes]

    hex_str = ' '.join(f"{b:02x}" for b in follow_bytes)
    char_str = bytes_to_printable_str(follow_bytes)

    print(f"hex:   {hex_str}")
    print(f"text:  {char_str}")

find_pattern_in_dmp("pid.708.dmp", preview_bytes=16)
```

결과

`CTF{M0rtyL0L}` 이라는 flag를 확인할 수 있습니다.

pid.708.dmp | dmp.py U

volatility3 > dmp.py

```python
import re

def bytes_to_printable_str(byte_data):
    return ''.join(chr(b) if 0x20 <= b <= 0x7E else '.' for b in byte_data)

def find_pattern_in_dmp(file_path, preview_bytes=16):
    pattern = (
        b'\x64' +             # 0x64
        b'(.{6,8})' +         # 6~8 bytes
        b'\x40\x06' +         # 0x40 0x06
        b'.{18}' +            # 18 bytes
        b'\x5a\x0c\x00\x00'   # 0x5a 0x0c 0x00 0x00
    )

    with open(file_path, 'rb') as f:
        data = f.read()

    match = re.search(pattern, data, re.DOTALL)
    if not match:
        print("패턴을 찾을 수 없습니다.")
        return

    offset = match.start()
    end = match.end()
    follow_bytes = data[end:end + preview_bytes]

    hex_str = ' '.join(f"{b:02x}" for b in follow_bytes)
    char_str = bytes_to_printable_str(follow_bytes)

    print(f"hex:   {hex_str}")
    print(f"text:  {char_str}")

find_pattern_in_dmp("pid.708.dmp", preview_bytes=16)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\j93es\Desktop\volatility3> python .\dmp.py
hex:   4d 30 72 74 79 4c 30 4c 00 00 00 00 00 00 00 21
text:  M0rtyL0L.......!
PS C:\Users\j93es\Desktop\volatility3>
```