0x00 前言

2023.8.15 夜里

非常欢迎使用我的魔刀千刃，并且欢迎各位师傅对我的开源代码进行指导！

```
--Offense without defense,

    unparalleled in the world.--
Welcome to the universe of N1nEmAn.

    To find detailed usage instructions for evilblade,

        please visit cnblogs.com/9man.
```

访问连接：https://pypi.org/project/evilblade/

0x01 安装

安装直接如下命令。

```
pip3 install evilblade
```

本包还依赖包LibcSearcher和pwntools，如未安装，请安装。

```
pip3 install pwntools
pip3 install LibcSearcher
```

0x02 内置帮助

为了兼容问题内置帮助我没有写中文，在这里写中文版本的。
内置帮助命令： `python -m pydoc evilblade` ，按 `q` 退出。

0x03 中文帮助

```
    dp(name, data)
        # 数据打印

    dpx(name, data)
        # 数据以十六进制格式打印

    evgdb(*argv)
        # 设置 gdb，也称为 evil-gdb
        # 如果需要设置断点，请将 'b address/defname' 作为参数。

    fmt(offset, begin, end, size, written)
        # 用于格式化字符串漏洞，但实用性不高。

    getbase(add, defname, *args)
```

```
        # 计算 libc 的基地址。"add" 是泄露的地址，"defname" 是库函数名，"*args" 是需要减去的额外偏移量。

    getd(i, j)
        # 用于十进制格式
        # 类似于 getx32

    getx(i, j)
        # 用于十六进制格式
        # 类似于 getx32

    getx32(i, j)
        # 用于 32 位
        # 在使用 tet() 测试过的 '\xff' 格式之后，你可以接收地址。
        # "i" 参数表示接收数据的开头，"j" 表示结尾。
        # 通常 "i" 设置为 0，"j" 设置为 -1。
        # 不断修改 "i" 和 "j" 的值，直到获得满意的结果为止。

    getx64(i, j)
        # 用于 64 位
        # 类似于 getx32

    gotadd(defname, *args)
        # 没有 PIE 保护，如果参数只有 "defname"，得到 GOT 表地址。
        # 有 PIE 保护，如果参数只有 "defname"，得到 GOT 表地址偏移。加上第二个参数作为基地址，得到实际 GOT 表地址。

    ia lambda (...)
        # 启动交互式 shell

    libset(libc_val)
        # 设置你的 libc，也称为 libc-set

    n2b lambda x
        # 将数字转换为字节

    pltadd(defname, *args)
        # 类似于 gotadd，但获取的是 PLT 表的地址

    rgetbase(add, defname, *args)
        # 类似于 'getbase'，但用于远程 libc 库

    rl lambda (...)
        # 接收一行数据，相当于 p.recvline()

    rlibset(defname, add)
        # 设置远程 libc 库。

    rop lambda r
        # 将值列表转换为 ROP 链（64 位）

    rsetup(mip, mport)
        # 建立远程连接

    rsymoff(defname, *args)
        # 类似于 'symoff'，但用于远程 libc 库

    ru lambda s
```

```
        # 接收数据，直到特定字符串，相当于 p.recvuntil(s)

    rv lambda x
        # 接收数据，相当于 p.recv(x)

    sa lambda t, s
        # 在特定字符串后发送数据，相当于 p.sendafter(t, s)

    sd lambda s
        # 发送数据，相当于 p.send(s)

    setup(p_val)
        # 设置你的进程和 ELF

    sl lambda s
        # 发送带有 '\n' 的数据，相当于 p.sendline(s)

    sla lambda t, s
        # 在特定字符串后发送一行数据，相当于 p.sendlineafter(t, s)

    sn lambda n
        # 发送一个带有 '\n' 的数字，相当于 sl(n2b(n))

    sna lambda t, n
        # 在特定字符串后发送一个数字，相当于 sla(t, n2b(n))

    symadd(defname, *args)
        # 类似于 gotadd，但获取的是 SYM 表的地址

    symoff(defname, *args)
        # 如果参数只有 'defname'，你会得到偏移量。
        # 如果有第二个参数作为基地址，你会得到函数的实际地址。

    terset(get)
        # 如果你不能直接运行 GDB，请根据情况使用 'terset' 来设置终端。使用 'echo $TERM' 的输出作为参数。

    tet()
        # 接收一行数据，并将其显示给你

    uu64 lambda data
        # 从字节中解包一个 64 位无符号整数

```

0x02 开源代码

```python
from pwn import *
from LibcSearcher import *

'''
明知道是陷阱，
    为什么还要来。
'''
# Convert a number to bytes
n2b = lambda x: str(x).encode()

```

```python
# Receive data, equivalent to p.recv(x)
rv = lambda x: p.recv(x)

# Receive a line of data, equivalent to p.recvline()
rl = lambda: p.recvline()

# Receive data until a specific string, equivalent to p.recvuntil(s)
ru = lambda s: p.recvuntil(s)

# Send data, equivalent to p.send(s)
sd = lambda s: p.send(s)

# Send data with '\n', equivalent to p.sendline(s)
sl = lambda s: p.sendline(s)

# Send a number with '\n', equivalent to sl(n2b(n))
sn = lambda n: sl(n2b(n))

# Send data after a specific string, equivalent to p.sendafter(t, s)
sa = lambda t, s: p.sendafter(t, s)

# Send a line of data after a specific string, equivalent to p.sendlineafter(t, s)
sla = lambda t, s: p.sendlineafter(t, s)

# Send a number after a specific string, equivalent to sla(t,n2b(n))
sna = lambda t, n: sla(t, n2b(n))

# Start an interactive shell
ia = lambda: p.interactive()

# Convert a list of values to a ROP chain (64-bit)
rop = lambda r: flat([p64(x) for x in r])

# Unpack a 64-bit unsigned integer from bytes
uu64 = lambda data: u64(data.ljust(8, b'\x00'))

##Set your libc, aka libc-set
def libset(libc_val):
    global libc
    libc = ELF(libc_val)

#Set your prosecc and ELF
def setup(p_val):
    global p
    global elf
    p = process(p_val)
    elf = ELF(p_val)
#Establish remote connection
def rsetup(mip, mport):#设置远程连接 remote setup
    if args.P:
        global p
        p = remote(mip,mport)
##Recieve a line of data, and show it for you
def tet():
    #test,测试接收数据
    p = globals()['p']
    r = ru('\n')
    print('\n---------------\n','add','is >>> ',r,'\n---------------')
```

```
    return r

#For 64-bit
#Just like getx32
def getx64(i,j):
    if i != 0:
        r = (ru('\n'))[i:j]
        dp('getx64',r)
        r = u64(r.ljust(8,b'\0'))
        print('\n---------------\n','add','is >>> ',hex(r),'\n---------------')
        return r
    else:
        r = (ru('\n'))[:j]
        dp('getx64',r)
        r = u64(r.ljust(8,b'\0'))
        print('\n---------------\n','add','is >>> ',hex(r),'\n---------------')
        return r

#For 32-bit
#After testing with tet(), addresses can be received for the '\xff' format.
#The 'i' parameter represents the start of the received data, while 'j' indicates the end.
#Usually, 'i' is set to 0, and 'j' is set to -1.
#Continuously adjust the values of 'i' and 'j' until you obtain the desired result.
def getx32(i,j):
    if i != 0:
        r = (ru('\n'))[i:j]
        dp('getx32',r)
        r = u32(r.ljust(4,b'\0'))
        print('\n---------------\n','add','is >>> ',hex(r),'\n---------------')
        return r
    else:
        r = (ru('\n'))[:j]
        dp('getx32',r)
        r = u32(r.ljust(4,b'\0'))
        print('\n---------------\n','add','is >>> ',hex(r),'\n---------------')
        return r

#For the hex format
#Just like getx32
def getx(i,j):
    if i != 0:
        r = (ru('\n'))[i:j]
        dp('geti',r)
        r = int(r,16)
        print('\n---------------\n','add','is >>> ',hex(r),'\n---------------')
        return r
    else:
        r = (ru('\n'))[:j]
        dp('geti',r)
        r = int(r,16)
        print('\n---------------\n','add','is >>> ',hex(r),'\n---------------')
        return r


#For the decimal format
#Just like getx32
```

```python
def getd(i,j):
    if i != 0:
        r = (ru('\n'))[i:j]
        dp('geti',r)
        r = int(r,10)
        print('\n---------------\n','add','is >>> ',hex(r),'\n---------------')
        return r
    else:
        r = (ru('\n'))[:j]
        dp('geti',r)
        r = int(r,10)
        print('\n---------------\n','add','is >>> ',hex(r),'\n---------------')
        return r

'''
只攻不防，
    天下无双~~~~~
        魔刀千刃。
'''
##计算世界

#Calculate the base address of libc. "add" is the leaked address, "defname" is the library function name, and "*args" are the excess offsets that need to be subtracted.
def getbase(add,defname,*args):
    #计算libcbase，args作为多余参数相减   get libcbase
    base = add - libc.sym[defname]
    for num in args:
        base -= num
    print('\nloading...')
    print('\n---------------\nget!your base is >>> ',hex(base),'\n--------------
')
    return base

ter = 'NULL'
#If you are unable to directly run GDB, please use the 'terset' to set the terminal according to your situation. Use the output of 'echo $TERM' as the parameter.
def terset(get):
    global ter
    dp('ter',ter)
#Set gdb,aka evil-gdb
#If you need to set a breakpoint, please use 'b address/defname' as the parameter.
def evgdb(*argv):
    p = globals()['p']
    ter = globals()['ter']
    #获取全局变量值
    dp('gdbter',ter)
    if ter!='NULL':
        context.terminal = [ter, '-e']
    if args.G:
        if(len(argv)==0):
            gdb.attach(p)
        else:
            gdb.attach(p,argv[0])
#If the parameter is only 'defname', you will get the offset.
#If there's a second parameter as the base address, you will get the actual address of the function.
```

```
177 def symoff(defname,*args):#计算或者设置偏移symbol's offset
178     if(len(args)>0):
179         ba = args[0]
180         print('\n---------------\nyour ',defname,'offset is >>>
    ',hex(libc.sym[defname]),'\n---------------')
181         print('\n---------------\nyour ',defname,'is in >>>
    ',hex(ba+libc.sym[defname]),'\n---------------')
182         return libc.sym[defname]+ba
183     else:
184         print('\n---------------\nyour ',defname,'offset is >>>
    ',hex(libc.sym[defname]),'\n---------------')
185         return libc.sym[defname]
186 #Without PIE, if only "defname", obtain the address of the GOT table.
187 #With PIE, if only "defname", obtain the offset of the GOT table. Adding the
    second parameter as the base address will give you the actual GOT table address.
188 def gotadd(defname,*args):#获取got表地址got'sadd
189     if (len(args) > 0):
190         return elf.got[defname]+args[0]#有pie的时候
191     return elf.got[defname]
192 #Jusr like gotadd,but obtain the address of the PLT table
193 def pltadd(defname,*args):#获取got表地址got'sadd
194     if (len(args) > 0):
195         return elf.plt[defname]+args[0]#有pie的时候
196     return elf.plt[defname]
197
198 #Just like gotadd,but obtain the address of the SYM table
199 def symadd(defname,*args):#获取got表地址got'sadd
200     if (len(args) > 0):
201         return elf.sym[defname]+args[0]#有pie的时候
202     return elf.sym[defname]
203 #Data print
204 def dp(name,data):#打印数值data print
205         print('\n---------------\nyour ',name,' is >>> ',(data),'\n---------------
    ')
206 #Data print as hex
207 def dpx(name,data):#hex打印数值data print
208         print('\n---------------\nyour ',name,' is >>> ',hex(data),'\n------------
    ---')
209
210 '''
211 因为,
212     我有想要保护的人。
213 '''
214
215 ##查库世界
216
217 #Set the remote libc library.
218 def rlibset(defname,add):
219     #远程libc设置
220     global rlibc
221     rlibc = LibcSearcher(defname, add)
222
223 #Just like 'getbase', but for remote libc library
224 def rgetbase(add,defname,*args):
225     #计算远程libcbase，args作为多余参数相减   get libcbase
226     base = add - rlibc.dump(defname)
227     for num in args:
228         base -= num
```

```
    print('\nloading...')
    print('\n---------------\nget!your base is >>> ',hex(base),'\n--------------
')
    return base
#Just like 'symoff',but for remote libc library
def rsymoff(defname,*args):#计算或者设置偏移symblol's offset
    if(len(args)>0):
        ba = args[0]
        print('\n----------------\nyour ',defname,'offset is >>>
',hex(rlibc.dump(defname)),'\n---------------')
        print('\n----------------\nyour ',defname,'is in >>>
',hex(ba+rlibc.dump(defname)),'\n---------------')
        return rlibc.dump(defname)+ba
    else:
        print('\n---------------\nyour ',defname,'offset is >>>
',hex(rlibc.dump(defname)),'\n---------------')
        return rlibc.dump(defname)

#攻击世界

#For fmt, but the reliability is not high.
def fmt(offset,begin,end,size,written):
    #fmt利用
    payload = fmtstr_payload(offset,{begin: end},write_size =
size,numbwritten=written)
    return payload
'''
    offset (int) – 您控制的第一个格式化程序的偏移量
    字典 (dict) – 被写入地址对应->写入的数据，可多个对应{addr: value, addr2: value2}
    numbwritten (int) – printf函数已写入的字节数
    write_size (str) – 必须是byte，short或int。告诉您是否要逐字节写入，短按short或int
(hhn，hn或n)
'''
```