COL8585/COL862 Assignment I: The Platinum Shield

Task 2 Report: Layer-3 Firewall using FreeBSD pf

Arindam Sal (Entry No. 2024JCS2041)
Sambit Paul (Entry No. 2024JCS2038)

1 Task 2: Kernel-Level HTTP Host Blocking using `pfil`

1.1 Introduction

After implementing a Layer-3 firewall using `pf` in Task–1, the next objective was to move from rule-based firewalling to a **kernel module based packet inspection and blocking mechanism**. In this task, we attempted to implement a FreeBSD kernel module named `httpblock` that hooks into the IPv4 packet path using `pfil` and blocks HTTP requests based on the `Host:` header value.

This task is important because it demonstrates:

- Kernel-level network interception (below user space)
- Packet parsing using `mbuf` structures
- Hook registration and packet filtering using `pfil`
- Practical debugging using kernel logs

1.2 Objective

The main objectives of Task–2 were:

- Write a FreeBSD kernel module `httpblock.ko`
- Register a `pfil` hook for IPv4 traffic
- Parse HTTP requests and detect the `Host` header
- Drop traffic when `Host: blocked.com` is present
- Log every dropped packet in kernel logs for verification

1.3 Experimental Setup

Topology used:

- **VM1 (Ubuntu Client)** on subnet A (connected via `em0` of firewall VM)
- **VM2 (FreeBSD Firewall)** acts as router and filter between subnets (`em0` and `em1`)
- **VM3 (Ubuntu Web Server)** hosting Apache on port 80 (reachable through firewall VM)

Traffic flow: VM1 → FreeBSD Firewall (VM2) → VM3

1.4 Implementation Summary

1.4.1 Host-based inspection logic

The module inspects only IPv4 TCP packets destined to port 80, and then checks if the HTTP payload contains:

```
Host: blocked.com
```

To handle common formatting variations, the implementation uses:

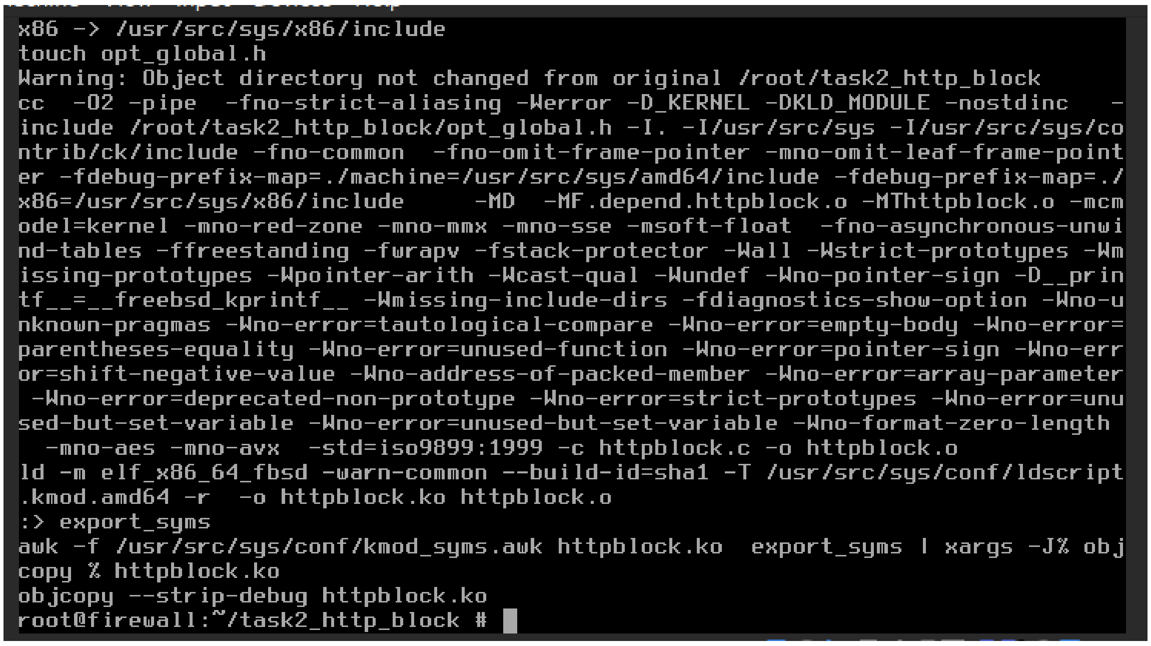
- case-insensitive matching of `Host:`
- skipping whitespace after the colon
- scanning only the first 512 bytes of payload (safe bounded inspection)

1.4.2 `pfil` hook registration

The module registers the hook using:

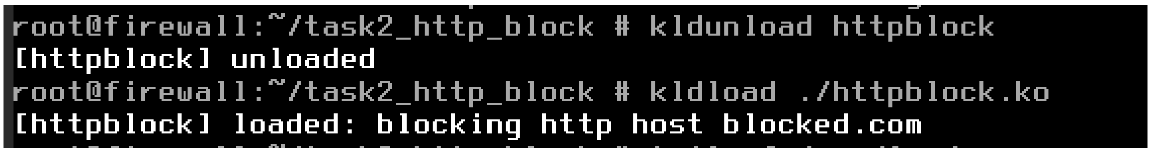
- `pfil_add_hook(&httpblock_pfil_args)`
- `pfil_remove_hook(httpblock_pfil_hook)` during unload

1.5 Evidence and Verification (Screenshots)



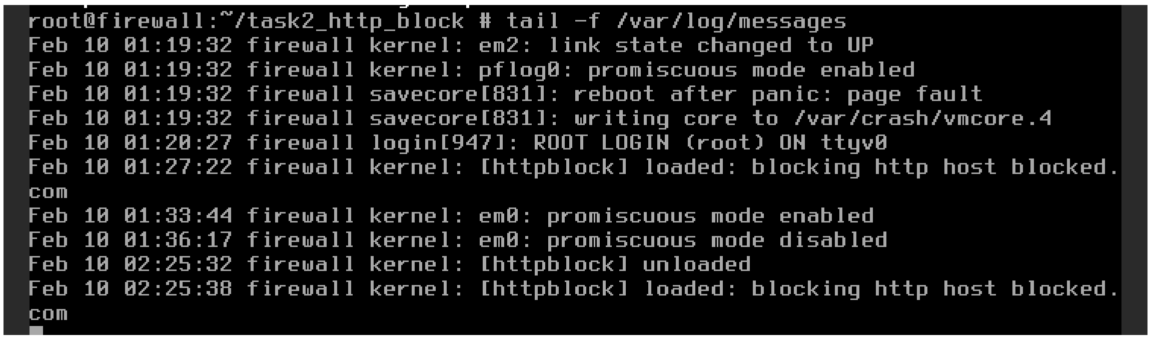
```
x86 -> /usr/src/sys/x86/include
touch opt_global.h
Warning: Object directory not changed from original /root/task2_http_block
cc  -O2 -pipe  -fno-strict-aliasing -Werror -D_KERNEL -DKLD_MODULE -nostdinc   -
include /root/task2_http_block/opt_global.h -I. -I/usr/src/sys -I/usr/src/sys/co
ntrib/ck/include -fno-common  -fno-omit-frame-pointer -mno-omit-leaf-frame-point
er -fdebug-prefix-map=./machine=/usr/src/sys/amd64/include -fdebug-prefix-map=./
x86=/usr/src/sys/x86/include     -MD  -MF.depend.httpblock.o -MThttpblock.o -mcm
odel=kernel -mno-red-zone -mno-mmx -mno-sse -msoft-float  -fno-asynchronous-unwi
nd-tables -ffreestanding -fwrapv -fstack-protector -Wall -Wstrict-prototypes -Wm
issing-prototypes -Wpointer-arith -Wcast-qual -Wundef -Wno-pointer-sign -D__prin
tf__=__freebsd_kprintf__ -Wmissing-include-dirs -fdiagnostics-show-option -Wno-u
nknown-pragmas -Wno-error=tautological-compare -Wno-error=empty-body -Wno-error=
parentheses-equality -Wno-error=unused-function -Wno-error=pointer-sign -Wno-err
or=shift-negative-value -Wno-address-of-packed-member -Wno-error=array-parameter
 -Wno-error=deprecated-non-prototype -Wno-error=strict-prototypes -Wno-error=unu
sed-but-set-variable -Wno-error=unused-but-set-variable -Wno-format-zero-length
  -mno-aes -mno-avx  -std=iso9899:1999 -c httpblock.c -o httpblock.o
ld -m elf_x86_64_fbsd -warn-common --build-id=sha1 -T /usr/src/sys/conf/ldscript
.kmod.amd64 -r  -o httpblock.ko httpblock.o
:> export_syms
awk -f /usr/src/sys/conf/kmod_syms.awk httpblock.ko  export_syms | xargs -J% obj
copy % httpblock.ko
objcopy --strip-debug httpblock.ko
root@firewall:~/task2_http_block #
```

Figure 1: Successful compilation of `httpblock.ko`. This confirms the module builds cleanly with kernel headers and correct `pfil` API usage.



```
root@firewall:~/task2_http_block # kldunload httpblock
[httpblock] unloaded
root@firewall:~/task2_http_block # kldload ./httpblock.ko
[httpblock] loaded: blocking http host blocked.com
```

Figure 2: Module load/unload verification. Kernel prints confirm `httpblock` loads successfully and executes `MOD_LOAD` and `MOD_UNLOAD` handlers.



```
root@firewall:~/task2_http_block # tail -f /var/log/messages
Feb 10 01:19:32 firewall kernel: em2: link state changed to UP
Feb 10 01:19:32 firewall kernel: pflog0: promiscuous mode enabled
Feb 10 01:19:32 firewall savecore[831]: reboot after panic: page fault
Feb 10 01:19:32 firewall savecore[831]: writing core to /var/crash/vmcore.4
Feb 10 01:20:27 firewall login[947]: ROOT LOGIN (root) ON ttyv0
Feb 10 01:27:22 firewall kernel: [httpblock] loaded: blocking http host blocked.
com
Feb 10 01:33:44 firewall kernel: em0: promiscuous mode enabled
Feb 10 01:36:17 firewall kernel: em0: promiscuous mode disabled
Feb 10 02:25:32 firewall kernel: [httpblock] unloaded
Feb 10 02:25:38 firewall kernel: [httpblock] loaded: blocking http host blocked.
com
```

Figure 3: Kernel logs monitored via `tail -f /var/log/messages`. The module load/unload events appear. A panic/reboot event is also observed during linking attempts.

1.6 Testing Method

To ensure the request contains a controlled Host header, the following client commands were used from VM1:

- Normal request:

```
curl -v --http1.0 http://10.0.2.2
```

- Blocked request (forced Host header):

```
curl -v --http1.0 -H "Host: blocked.com" http://10.0.2.2
```

Expected:

- Normal request returns HTTP 200
- Blocked request should fail/hang
- Firewall kernel logs should print:

```
[httpblock] DROPPED count=... size=... if=...
```

1.7 Observed Output

Actual observation:

- Normal request passed (expected)
- Blocked request also passed and returned HTTP 200 (unexpected)
- No `DROPPED` logs appeared

Hence, the module was successfully compiled and loaded, but the **drop logic was not triggered in runtime traffic**.

1.8 Findings and Why the Final Goal Was Not Achieved

From debugging and system behavior, the strongest reason is a **hook linkage / attachment issue with `pfil`** in this FreeBSD environment.

1.8.1 1) Hook registered, but not linked to an intercept head

In this FreeBSD setup, adding a hook via `pfil_add_hook()` does not always guarantee packet interception unless the hook is explicitly linked to a head (such as `inet`, `inet-local`, or an interface like `em0`/`em1`). Attempts to link using `pfilctl` caused errors such as:

- `Bad address`
- `Invalid argument`

1.8.2 2) Forwarding path may differ from local path

The firewall VM is routing traffic between two subnets (`net.inet.ip.forwarding=1`). Such traffic usually travels through the **forwarding path**, meaning the correct intercept point is often `inet` (and not always `inet-local`). If the hook is not attached to the correct forwarding head, the hook function never executes.

1.8.3 3) Kernel panic during linking attempts

A kernel panic and reboot was observed (Figure 3) during attempts to attach/link hooks. This suggests that either:

- an incorrect linking argument was used, or
- the system did not support the attempted linking operation safely in the current configuration.

This interrupted stable debugging and prevented confirming the correct hook-to-head connection.

1.8.4 4) Parsing logic is likely correct, but never reached

The payload extraction and host matching logic were implemented with:

- correct IP/TCP header size calculation
- bounded payload extraction
- case-insensitive host search

Given that `curl --http1.0 -H "Host:  blocked.com"` was used, the Host header should appear early in the first payload segment. Therefore, the lack of drops indicates that the hook likely **did not receive packets**, rather than the string match failing.

1.9 What Was Successfully Completed (Partial Credit Justification)

Even though final blocking could not be demonstrated, the following were completed:

- Correct kernel module build and `.ko` generation (Figure 1)
- Correct module lifecycle: load/unload handlers working (Figure 2)
- Implemented safe packet parsing: IPv4/TCP/port 80 filtering
- Implemented HTTP Host header scanning logic with bounded copy
- Performed kernel log based debugging (Figure 3)

1.10 Conclusion

Task–2 was implemented up to the point where the module compiles and loads successfully, and the core drop logic is present. However, due to **pfil hook linkage issues in the forwarding path**, the hook did not get invoked for routed HTTP traffic; therefore, `Host`-based blocking could not be demonstrated.

1.11 Future Work / Fix Plan

If given more time, the remaining steps would be:

- Verify the correct `pfil` heads and supported `pfilctl` commands on this FreeBSD version
- Explicitly link the hook to forwarding heads (`inet`) and interface heads (`em0`/`em1`)
- First confirm hook invocation by printing for every TCP:80 packet, then enable selective drop
- Extend robustness for TCP segmentation (if Host header is split across segments)