NPF: NetBSD's new firewall

Zoltán Arnold Nagy
The NetBSD Foundation
zoltan@netbsd.org

Abstract

In today multicore world, bandwidth requirements continue to rise. In order to cope with this, there was a need for a firewall which was designed with SMP in mind and embracing concurreny. NPF is this firewall. While it's not as feature-heavy as other firewall implementations available, with it's from scratch design it brings scalability to the table. It will debut in NetBSD 6.0.

1 Background

1.1 Motivation

If we look around the market today, multicore processors are everywhere; from 12-core high-end server CPUs to dual-core cellphones, it's been very hard to not bump into them. Along with this, datacenter's bandwidth requirements are staggering - as more and more people start to enjoy online streaming, IPTv, Video on Demand services, we're saturating hundreds of gigabits of bandwidth. However, in order to build secure networks we need firewalls, and thus in turn, these firewalls need to handle these bandwidth requirements. So far NetBSD distributed two packet filters - IPFilter and PF -, but we realized they lack a number of features, which are the following:

- Multicore design
- Concurrent algorithms
- Fast ruleset manipulation

1.2 History overview

On 12 September, 2010 NPF has been announced [4]. It has been written by Mindaugas Rasiukevicius from scratch, and was sponspored by The NetBSD Foundation. The main reason for funding the development of a packet filter was to improve NetBSD's network capabilities, and keeping in mind the importance of SMP systems, it was a logical step. During the 2011 Google Summer of Code [1], IPv6 support has been added. [6]

2 Design

Inspired by the Berkeley Packet Filter (BPF) [2], NPF uses "n-code" to describe the filtering ruleset. Each rule is compiled into a low-level machine code, which is then in turn interpreted, and evaluated. This low-level machine code essentially represents a directed acyclic control flow graph. Packet inspection is done arithemtically (meaning the interpreted bytecode does not depend on the packets themselves; it doesn't matter what kind of packet is currently being processed, the bytecode ran against it will be always the same, except for the control instructions, which manipulate flow). All processing is done in 32-bit words (including instructions and their arguments). The two main extension points are rule procedures, which are designed to perform custom actions on packets, and can be implemented in a kernel module by registering it's procedure handler, and Application Level Gateways (ALGs), which allow application-driven dynamic firewall configurations (such as SIP/RTSP connection tracking).

2.1 Instruction set

The n-code bytecode interpreter supports both RISC and CISC like instructions. [3]

2.1.1 Fixed length instructions

- Flow-control instructions are: return, advance, jump, tag and invalidate
- Set and load instructions are move and load word (lw)
- Comparison instructions
- Arithmetic instructions are add, sub, multiply and divide
- Bitwise instructions are: not, and, or and xor

2.1.2 Variable length instructions

While in general the bytecode engine avoids internal protocol knowledge, the instructions supporting the rulesets do use protocol knowledge in order to perform their checks. They are supposed to be fully optimized.

- IP4MASK/IP6MASK are used to check the packet's IP address against an other masked IP address; the whole bytecode fragment consists of 4/7 words respectively: the opcode, the direction, the address and the netmask
- TABLE is used to match the specified address against a table; the whole bytecode fragment consists of 3 words: the opcode, the destination and the table id
- ICMP4 is used to match ICMP type and code; the whole fragment consists of 2 words: the opcode, and the 8-bit type and 8-bit fit together into a 32-bit word
- TCP_PORTS/UDP_PORTS is used to match TCP/UDP ports; it consists of 3 words: opcode, direction and port range encoded into a 32-bit word.
- TCP_FLAGS is used to match TCP flags/-mask; the whole fragment consists of 2 words: the opcode and the flag and the mask together in a 32-bit word

3 Usage

The configuration syntax for NPF is very similiar to PF/IPFilter syntax. Generally, the configuration resides in /etc/npf.conf, while NPF can be controller by the npfctl(8) utilty.

3.1 Basics

In NPF, each rule must belong to a group, and there's always at least one group, called the default.

Let's have a look at an example:

```
ext_if = "wm0"
int_if = "wm1"

group (name "ext", interface $ext_if) {
        pass all
}

group (name "int", interface $int_if) {
        pass all
}

group (default) {
        block all
}
```

First, we define two variables, and give them two interface names. Then we define two groups - the first to match packets that pass the interface defined in the ext_if variable, and a second one to do the same with int_if. If a packet arrives that matches either of the groups, it will get inspected by the associated group; else, the default group's rules apply.

3.2 Tables

Frequently changing the configuration and reloading the configuration all the time can be an expensive task. Also, if you have the same rule listed many times just with different IP addresses, it can be inefficient to evaluate. To solve this problem, NPF supports the creation of tables. A table can be backed up either by a Patricia tree or by a hash. Example use:

```
table <1> type tree dynamic
group (default) {
        block in quick from <1>
}
```

After that, we only need to manipulate the table with id 1 to change which packets are blocked:

```
server# npfctl table 1 add 10.1.2.1/32
server# npfctl table 1 add 10.1.1.0/24
server# npfctl table 1 rem 10.1.3.0/24
```

3.3 Rule procedures

Rule procedures can be used to apply different transformations to packets matching a rule. There are two rule procedures currently implemented: logging and normalization. Example:

```
procedure "log" {
        log: npflog0
}

procedure "norm": {
        normalise: "random-id"
}
group (default):
        block in quick from <1> apply "log"
```

To declare procedures, you have to specify the rule procedure's name and parameters, and give it a user defined procedure name. Then, in order to execute this procedure in a rule on the matching packet, you can use the apply keyword followed by the user-defined name.

4 Future

With NetBSD 6.0 NPF will get a much more wider exposure among users, which in turn will make the code base more mature; NPF implements what many would call basic requirements for a viable firewall. Currently, the most pressing matter impacting NPF is the single-threadedness of the networking code in NetBSD, but changing this would require a hug effort. Especically for this reason, funding has been offered for it [5], sadly with no takers. NetBSD is participating in Google Summer of Code again, and NPF will be on the list of proposed student projects. Here's an incomplete list of things that could be done generally around NPF or the networking code, with no particular order regarding either the difficulty or the importance of the task:

- CARP support
- rate-limiting support
- layer 7 filtering support
- ALGs for common protocols (SIP for VoIP, FTP, IRC, ...)
- more detailed statistics
- Multi-queue NIC support with per-CPU queue bindings
- On-NIC packet classification support

References

[1] Google. Summer of Code. Available at `http://code.google.com/soc/`.

[2] Steven McCanne and Van Jacobson. The bsd packet filter: a new architecture for user-level packet capture. In *Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings*, USENIX'93, pages 2–2, Berkeley, CA, USA, 1993. USENIX Association.

[3] "NPF". "n-code header". `http://cvsweb.netbsd.org/bsdweb.cgi/src/sys/net/npf/npf_ncode.h?rev=1.6&content-type=text/x-cvsweb-markup`.

[4] The NetBSD Foundation. Introducing NPF, NetBSD's new packet filter. Available at `http://mail-index.netbsd.org/netbsd-announce/2010/09/13/msg000110.html`.

[5] The NetBSD Foundation. Plan and funding of SMP Networking projects. Available at `http://mail-index.netbsd.org/netbsd-announce/2011/11/25/msg000136.html`.

[6] Zoltan Arnold NAGY. IPv6 support for NPF firewall. `http://www.google-melange.com/gsoc/proposal/review/google/gsoc2011/nagyz/1001`.