Netdev 0.1 – Netfilter BoF

Pete Bohman, Platform Security Engineer
Joshua Hunt, Kernel Engineer

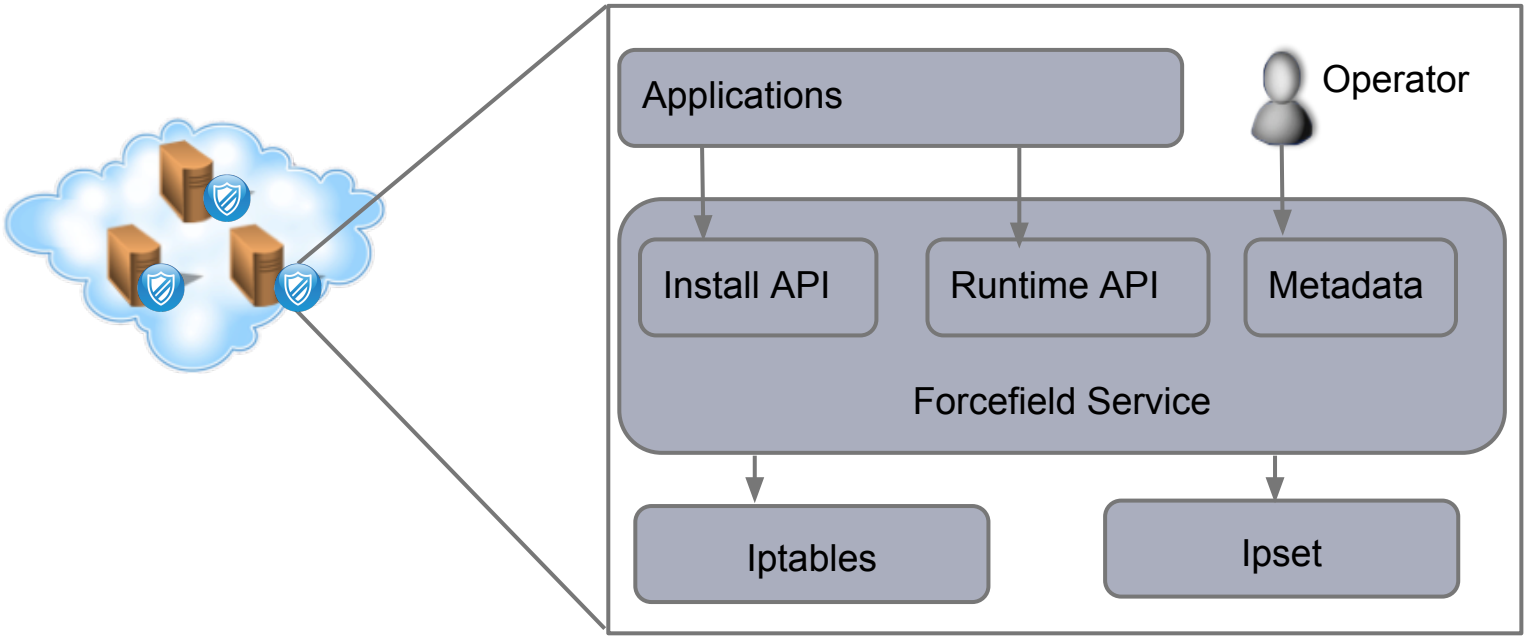
BoF Overview

- Logistics
  - Notes being kept on etherpad: https://etherpad.mozilla.org/RvVanIdtZ8
- Akamai and netfilter
  - Akamai's use of netfilter components
  - Desired netfilter interfaces
  - Scaling netfilter components
  - Nftables considerations
  - QA
- TBD Presentations
- Open Discussion

(etherpad link)

Forcefield Firewall Manager

- Internal IP Firewall Manager running on every Akamai machine
  - Deployed in 2010 to help manage firewall
  - First line of defense against attacks
  - Centralized policy management
  - Goal: Safely expose all netfilter functionality
  - Goal: Provide easy to use abstractions for common use cases
- Globally distributed
  - 160,000+ servers, 2,600+ locations, 107 countries



Forcefield Firewall Manager - Interfaces and Use Cases

- Install API
  - Open ports
  - Close ports
  - Protect ports (default policy)
    - Applies an Akamai Source IP ACL to the port
    - Current ACL is a hash:net set type with 460K cidrs (10M IPs)
  - Install application specific chains and ipsets
    - White/black list sets (~5M entries) and rules
    - Rate limiting rules using hashlimit module
    - Attack signature rules using u32 and string match modules
    - Random sampling to nfqueue using statistic module

```
# HTTP and HTTPS are open on machines running ghost.
$ffapi->set_protection( \&is_ghost,
                                    level => $ffapi->OPEN,
                                    ports => [ "80,443/tcp" ],
                                    priority => 10,
                                  );
```

Forcefield Firewall Manager - Interfaces and Use Cases

- Runtime API
  - Dynamically modify the contents of ipsets
    - Updating white/black lists
    - Toggling rules by populating ipsets

```
# -m set --match-set whitelist_tog dst -j filter
# ipset add whitlist_tog $MYIP
```

  - Retrieve ipset contents and counters

- Metadata Interface
  - Allows operators to push changes to sets of machines
  - Quick and safe response mechanism for DoS attacks
    - Add a rule for newly detected attack signatures
    - Block miss configured requests coming from a set of customer IPs

Forcefield Firewall Manager - Monitoring

- Monitor state of iptables and ipset
  - Are chains/sets loaded?
  - Are ipset/rule packet count rates abnormally high?
- Traffic analysis
  - Sample of dropped packets are logged
    - Message framing with error correction
    - Log binary payload only to avoid deep packet inspection
    - Considering ulogd2 framework with custom output module
  - Logs aggregated from across the network
  - Logs processed to produce forcefield dashboard (next slide)

Akamai | ForceField Defense Report *beta*

60,768,042 probes blocked last hour by all target IPs

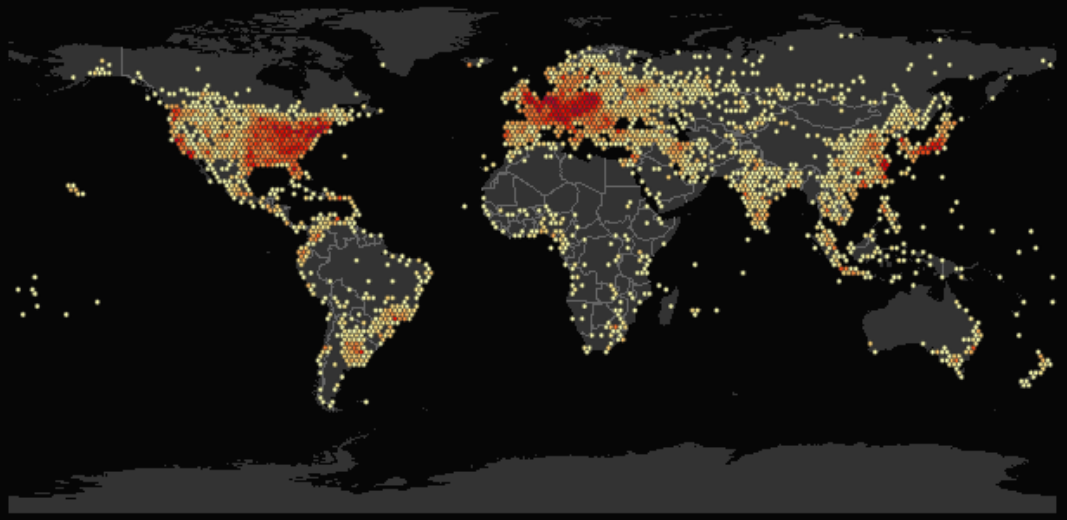
57,883,522 probes blocked last hour by Service IPs

2,884,520 probes blocked last hour by Darknet IPs

Source Countries avg probes/hour

	Last Hour	24-hr Trend	Last Day	15-day Trend
USA	14,398,959		16,012,393	
China	7,559,351		7,873,949	
Netherl...	5,102,926		2,268,340	
France	2,659,771		5,289,345	
Canada	2,265,911		3,216,578	
Germany	2,085,760		1,785,372	
Russia	1,685,765		2,182,569	
Brazil	1,663,492		1,339,197	
India	1,631,507		1,340,778	
Turkey	1,621,539		1,432,572	

Source Cities avg probes/hour



Displaying data from Thu, 06 Nov 2014 14:00:00 GMT | retrieved at Thu, 06 Nov 2014 17:03:26 GMT

Source Networks avg probes/hour

	Last Hour	24-hr Trend	Last Day	15-day Trend
AS 4134 No.31	4,493,239		4,706,625	
AS 15169 Google Inc.	3,151,201		312,743	
AS 16276 OVH	3,055,112		7,026,885	
AS 10439 CariNet	1,353,180		1,344,801	
AS 29073 AS29073	992,511		851,582	
AS 3320 DTAG Deutsch...	982,324		795,986	
AS 9121 TTNET Turk Tel...	869,524		753,262	
AS 3462 HINET Data Co...	768,153		668,818	

Target Ports avg probes/hour

	Last Hour	24-hr Trend	Last Day	15-day Trend
UDP:500 isakmp	9,408,467		8,023,514	
UDP:137 NETBIOS Nam...	8,188,247		6,598,565	
TCP:445 Microsoft-DS	7,547,738		6,859,702	
TCP:23 Telnet	3,956,273		3,791,936	
TCP:22 The Secure Shel...	1,294,647		1,495,288	
TCP:8080 HTTP Alternat...	1,255,507		1,375,003	
TCP:53 Domain Name S...	1,222,606		215,825	
TCP:1234 Infoseek Sear...	870,914		1,224,147	

Netfilter Integration Improvements

- Integration Pain Points
  - Textual CLI
    - strcmp() differences hinder read and lookup operations

```
> ipset create foo hash:net  maxelem 1024
> ipset save foo
create foo hash:net family inet hashsize 1024 maxelem 1024
```

    - Overhead of spawning processes for each operation
  - Non atomic ipset operations
    - A failure may leave partial state committed in ipset
    - Workaround involves:
      - Create temporary set (max set name size limited to)
      - Copy existing set to temporary set for incremental operations
      - Perform operations on temporary set
      - Swap temporary set and target set
        - Lose counter updates during above steps
      - Delete temporary set
  - Change monitoring and logging
    - Must poll iptables/ipset to monitor changes and persist state

Netfilter Integration Improvements

- Supported user space API
  - CRUD (Create, Read, Update, Delete) operations
  - Operations performed on a handle returned from Create
  - Message based operations with defined extensible fields
  - Registered callbacks for modifications

Scaling netfilter components

Akamai's use of iptables:

- Pete provided the high-level overview of our usage and identified some of the issues we've hit with iptables. Most of those revolve around the lack of a supported library.
- For the most part we run iptables with no modifications.
- Able to utilize the great work done by the community here.
- We have hit some problems with the hashlimit match recently.

Scaling netfilter components

Hashlimit issues

- Problem 1:

  Modification of hash parameters via iptables restore doesn't work.

  - Patch proposed earlier this year: http://permalink.gmane.org/gmane.comp.security.firewalls.netfilter.devel/53515
  - Modifying hash config parameters, for ex, changing the rate limit for a rule and trying to replace it with iptables-restore, does not enforce the change of the rate.
  - Ex:

```
original rule:
    -A INPUT -s 10.18.40.44/32 -i eth1 -p tcp -m hashlimit --
hashlimit-upto 10/sec --hashlimit-burst 10
--hashlimit-name test -j ACCEPT

new rule:
    -A INPUT -s 10.18.40.44/32 -i eth1 -p tcp -m hashlimit --
hashlimit-upto 1000/sec --hashlimit-burst 10
--hashlimit-name test -j ACCEPT
```

Scaling netfilter components

Hashlimit continued:

- Basic Problem:
  - htable_find_get() only checks against name and family when looking to see if a hash already exists
- Upstream feedback:
  - While the functionality isn't ideal it's been this way forever and so we can't break this now since people may be relying on this behavior.
  - There's at least one other problem here, reported in the thread by Florian Westphal that it silently allows you to use the same hash with different params in the same ruleset:

Note that:

```
-A INPUT -m hashlimit --hashlimit-upto 10/sec --hashlimit-burst 10 --hashlimit-name
test
-A INPUT -m hashlimit --hashlimit-upto 1/sec --hashlimit-burst 10 --hashlimit-name
test
```

doesn't work as expected either (rule #2 uses config options of #1).

Scaling netfilter components

Hashlimit continued:

- Problem 2: Max rate is capped at 10,000 pps
  - This is too low for some of the systems we run.
  - Appears to be because it's using a 32-bit type to define the rateinfo

- For Problem 2, the ratelimit problem, I think we can fix this in the existing hashlimit implementation, but based on the original feedback, Problem 1, the config change issue, will require a new implementation?

- If so, we'd like to create a hashlimit2 to fix this and any other issues currently in hashlimit which can't be done in the current implementation.

- **Thoughts?**
- Other hashlimit problems which may not be solvable in the current implementation?

Scaling Netfilter Components

Akamai's ipset usage:

- Currently have sets with up to 2 million entries
- We expect this to grow to 25 million possibly in the near future
  - Some of this can be reduced with more efficient combination of set information.

    Ex: ability to match against a port range inside the set
- Using v4 and v6 versions of:
  - hash:ip
  - hash:ip,port
  - hash:net
  - hash:net,net
  - list
  - etc
- A # of internal teams looking to utilize ipsets.
- Tend to be used for white and blacklisting.
- Relying on libipset and our own firewall management layer to manage these sets.

Scaling netfilter components

libipset:

- We understand it's not fully supported/stable API.
- We're willing to take this risk and handle any problems there.
- It doesn't expose everything that we'd like to see wrt set metadata.
- We've recently started trying to push patches to expose some of this metadata like:
  - Exposing set size info:

    Add element count to hash headers

    http://patchwork.ozlabs.org/patch/422902/

- Our applications want/need more visibility into the details of the set, both from a configuration and reporting standpoint.
- Not thread-safe :(

Scaling netfilter components

Working with large sets:

- Adding entries to sets directly can take a long time:

```
ipset create foo hash:ip hashsize 1048576 maxelem 2000000
cat 1M.ips.rand | xargs -n1 ipset add foo
```

336.552358884 seconds

- Work around the add problem by using restore to create a temporary set and then swap them:

```
ipset restore < 1M.set
```

1.914640349 seconds

- Time to save large sets:

```
ipset save > 1M.set
```

0.854251486 seconds

Scaling netfilter components

- Problems with ipset:
  - These operations aren't atomic, so a failure mid-way leaves you with a partial set.
  - Sets grow, but they don't shrink.
  - Separation with iptables makes it difficult to work with at times.

Scaling netfilter components

- How do nftables sets compare?
  - We'll start with one of the simple table definitions from one of the wikis:

```
table ip filter {
        set blackhole {
             type ipv4_addr
        }
        chain input {
                  type filter hook input priority 0;
        }

        chain forward {
                  type filter hook forward priority 0;
        }

        chain output {
                  type filter hook output priority 0;
        }
}
```

Scaling netfilter components

- Tried adding 1 million elements… took a long time so scaled back to 10k:

```
cat 10k.ips | xargs -I entry nft add element filter blackhole { entry }
      real 3m34.749s
      user 0m3.773s
      sys  0m18.719s
```

- As we saw before ipset also has issues adding entries one at a time, but not quite as bad. Here's how long it takes ipset:

```
cat 10k.ips | xargs -n1 ipset add foo
      real    0m20.439s
      user    0m1.940s
      sys     0m18.757s
```

Scaling netfilter components

- With ipsets we saw a similar problem where adds take much longer than restores. Maybe this is the better way to do it in nftables also?

```
time nft -f test-10k-nft
real 0m0.100s
user 0m0.009s
sys  0m0.070s
```

- Win! So lets check 1 million now:

```
time nft -f test-1M-nft

Message from syslogd@a198-18-40-32 at Fri Feb 13 11:41:14 2015 ...
a198-18-40-32 kernel: [  328.092675] NMI watchdog: BUG: soft lockup - CPU#4 stuck
for 22s! [nft:3921]

Message from syslogd@a198-18-40-32 at Fri Feb 13 11:41:42 2015 ...
a198-18-40-32 kernel: [  356.069085] NMI watchdog: BUG: soft lockup - CPU#4 stuck
for 22s! [nft:3921]
```

- :(

Scaling netfilter components

- Reported in “softlockups when trying to restore an nft set of 1M entries”:

  https://marc.info/?l=netfilter-devel&m=142382876825876&w=2

  - Cong Wang suggested a patch to add a cond_resched() in nf_tables_newsetelem(). This appears to get rid of the softlockup in my initial testing. However there still appear to be some other underlying problems.

- Started investigating and noticed restores don’t increase the # of buckets. If I have a default setting of 4 buckets and do a restore, no expansion takes place:

```
[ 1408.311877] __rhashtable_insert:562: ffff88040a1bce90[1] elts:9999
[ 1408.311893] __rhashtable_insert:562: ffff88040a1bce90[3] elts:10000
[ 1408.311903] rhashtable_expand:385: Expanding ffff88040a1bce90 to:8
```

- Why doesn’t it grow on restore?
  - Not sure yet. It’s on my TODO list. I think Patrick and Thomas have a good idea as to what’s going on here.

Scaling netfilter components

- When we do an add of 10k elements starting with 4 buckets that grows to 16384.
- What happens if I provide a more appropriate bucket hint? Say 1M buckets for a 1M entry set:

```
nft -f test-1M-nft

real    0m1.370s
user    0m0.829s
sys     0m0.488s
```

- Yay! On par with ipset now, which was 1.9s

Scaling netfilter components

- Why do nftables set restores take so much longer?
  - Can't expand the hash table
    - Underlying data structure being chosen is a hashtable, which is implemented using rhashtables.
    - Current net-next implementation of nft hash sets default to 4 buckets.
    - nft_hash doesn't define a max_shift param to rhashtables. max_shift defines the max # of buckets the table can grow too.
    - If no max_shift is defined, then the table is not allowed to expand.
    - Currently this is not enforced as a requirement if tables want the ability to expand. I've pushed a patch up to resolve this: https://patchwork.ozlabs.org/patch/438122/
  - No way to set initial # of buckets
    - nft sets support a 'size' parameter. This is currently used to provide the a initial bucket hint (nelem_hint) for rhashtables, but it's also used in other nft set code to define the max # of elements.
    - The use of 'size' in nft sets seems incorrect to me.

Scaling netfilter components

- I've proposed the following two patches to:
  - Use 'size' to define max_shift so its consistent with the other nft set code (defaults to a max of 1024 elements if no size param is provided):

    https://patchwork.ozlabs.org/patch/438118/
  - Introduce a new parameter to set the # of initial buckets, 'init_size', to pass as the nelem_hint (still defaults to 4):

    http://patchwork.ozlabs.org/patch/438120/

Scaling netfilter components

- With all of that how do things look now ?
  - Add of 10k elements (Using 16384 initial buckets):

```
time cat 10k.ips | xargs -I entry nft add element filter blackhole {entry}
real    3m8.715s
user    0m2.211s
sys     0m3.087s
```

  - Restore of 10k elements (Using 16384 initial buckets):

```
time nft -f test-10k-nft

real    0m0.035s
user    0m0.008s
sys     0m0.005s
```

  - Restore of 1M elements (Using 2M initial buckets):

```
time nft -f test-1M-nft

real    0m1.645s
user    0m0.804s
sys     0m0.486s
```

Scaling netfilter components

- Revisiting adds with nftables sets. Why do they take so long and is there anything we can do about it?
  - With new patches in place we can define initial # of buckets so that it doesn't have to resize the set during adds, but that does not affect performance noticeably.
  - Why do they take so long?
    - Another item for my TODO list.
    - Is it just the syscall overhead?
    - Thoughts?

Scaling netfilter components

- Alternatives?
  - nft supports batch adding. (Tests before were run with adding one element at a time.)
  - What happens if we batch add 1M entries (provided bucket hint of 1M)?

```
Batching with 2048 entries:
real    0m14.430s
user    0m8.038s
sys     0m1.116s

Batching with 4096 entries:
real    0m12.356s
user    0m12.903s
sys     0m0.988s

Batching with 8192 entries:
real    0m22.602s
user    0m23.393s
sys     0m0.871s
```

Scaling netfilter components

- Batch adding vs single element adding of 1M entries:
  - 4k entries takes ~12 seconds.
  - Single element takes many minutes. Longer than I was willing to wait around and figure out… :)
  - Batching allows you to combine multiple entries into a single netlink message.

Scaling netfilter components

Add/Restore Performance Comparison for 1M Entry Sets

Operation	**Software**	**Time**
Add 1M ipset entries	ipset add	336.5s
Restore 1M ipset entries	ipset restore	1.91s
Add 1M nft set entries	nft add	Still going...
Add 1M nft set entries	nft add 4K batch w/patches	12s
Restore 1M nft set entries	nft -f w/patches	1.65s

Scaling netfilter components

Initial impressions/questions of nftables sets:

- With my patches restore performance seems acceptable.
- Atomicity is a big win for us.
- Ability to combine ipv4 and ipv6 rulesets will reduce duplicate code for us.
- Even though rbtree is a possible set type, I was not able to find a config which chose it. The hash table was preferred in the cases I tried. Is there a setup where rbtree is used?
  - Thoughts on providing a set type hint, that way the user could force the usage of a certain set type?
- Open to accepting other underlying data structures to be selected for set types?
  - I think we're interested in looking at alternatives to hash tables for certain sets to help reduce memory overhead.

Scaling netfilter components

Initial impressions/questions of nftables sets (continued):

- Storing large sets in a single rules file is a bit cumbersome.
  - Think 20 sets with 1 million entries each. In addition to the normal ruleset. What about the ability to save sets to separate files and then reference them as includes?
- Is the plan to reach feature parity with ipsets in terms of supported set types?
- The more user-configurable we allow sets to be I think the more valuable they become, even if this is only at the library level:
  - Number of hash initial hash buckets
  - grow and shrink thresholds

Scaling netfilter components

rhashtable:

- Should rhashtable enforce a maxelems value or should that be left up to the user of the hash?
  - Currently it caps the # of buckets to grow, but not the total size of the table. Meaning the chains can grow as long as you have available memory.

Scaling netfilter components

My personal nftables TODO:

- Understand why the table doesn't grow during restore.
- ipsets vs nftables sets performance comparison
  - Will send that to mailing list when I get this data.
- Understand why single element adds to a set take so long.
- Add tests to nftables to exercise some of the cases I've described.

Nftables Considerations

libnftables

- Very important to have a supported library to take full advantage of netfilter components.

- Concerned there will be 3 places to update: nftables, libnftnl, libnftables. Maybe 4 if you have to update the kernel too.

- Requirements:
  - thread-safe
  - expose kernel-level rules/set metadata
    - memory usage
    - set size
    - etc
  - atomicity for combined set and rule operations
  - random matching (BPF "`ld rand`")

Nftables Considerations

nftables compatibility tool:

- Cool idea.
- For it to be usable for us it would need ipset support.
  - Does it make sense to extend this or add a separate tool which would do the ipset to nft set conversion?

Q&A

- Any questions?


Akamai
FASTER FORWARD