fishnet™ SECURITY

Check Point Kernel Debugging, In-Depth

John Petrucci

July 2013

Rev. 2 – December 2014

SECURELY ENABLING **YOUR BUSINESS**

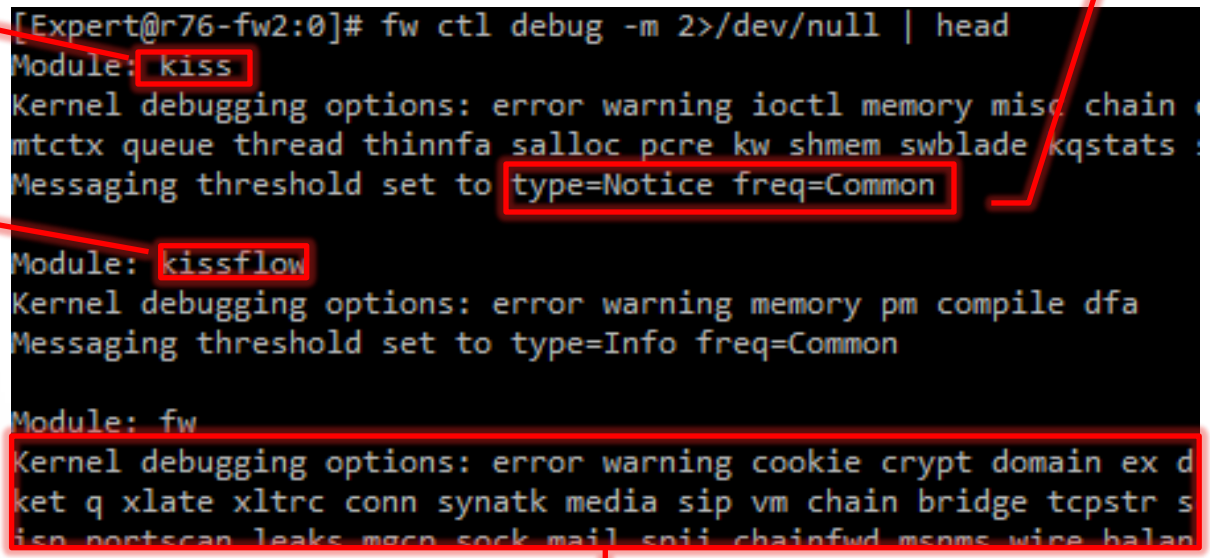
Outline

1. Terminology Used
2. zdebug vs kdebug
3. fw ctl debug –i filterfile
4. Buffers and Suppression
   a. The Buffer
   b. Debug Buffer Size (-buf)
   c. kdebug Has a Buffer Too
   d. dmesg; the Kernel Ring Buffer
   e. kdprintf_limit
5. Gotchas
   a. Type / Frequency Thresholds
   b. Printing Additional Fields w/ kdebug
   c. Printing Additional Fields w/ Kernel Parameters
   d. SecureXL Debugs
   e. Failed Buffer Allocations
   f. Stuck Debug Flags
   g. “Corrupted” Output with -o
6. Enumerating All Possible Debugs
   a. Debug Sampler Bash Script

Terminology Used

modules

type & frequency thresholds



flags

fishnet™ SECURITY

zdebug vs kdebug

zdebug vs kdebug

zdebug is intended for quick, transient debugging. It is typically used when you plan to review the output immediately such as while in a remote session with a client and looking for something specific like a rulebase drop.

zdebug is a shortcut which automatically:

1. sets the buffer to 1MB
2. enables the selected flags
3. starts printing the debug buffer contents to the screen
4. resets the buffer and modules to the defaults when you're finished

zdebug is limited in that you cannot:

- modify the buffer size
- specify type/frequency thresholds
- debug on flags from more than one module
- enable timestamps

zdebug vs kdebug (cont.)

kdebug requires a bit more setup work, but can do much more:

- Set any combination of module and flags
- Display second or microsecond timestamps
- Create a cyclic set of files to keep disk space from being consumed (e.g. debug.0, debug.1, …, debug.n)
- Print additional “fields” (columns) of data (see Printing Additional Fields w/ kdebug)

Think of kdebug as a way to view the kernel message buffer. The actual manipulation of what appears in the buffer; modules, flags, types, and frequencies is done with `fw ctl debug`. In fact, as long as you set a buffer first with `fw ctl debug` you can start viewing it with kdebug and manipulate the modules and flags while kdebug is running.

kdebug can be used to view the messages on the console in realtime just like zdebug, but it is more useful when we redirect the output to a file.

zdebug vs kdebug (cont.)

Cyclic file creation is one of the most under-utilized features of kdebug. It can be used so that a debug may be run for an extended period of time without ever consuming too much disk space. You should still be wary of the CPU overhead though, when asking a client to run a heavy debug during production hours. To set up a cyclic kdebug use the syntax:

```
fw ctl kdebug -s <Size_of_each> -m <nuMber_rotations> -o <Output_name> -f
```

This is more resistant to file corruption and disk space exhaustion than using a file descriptor redirect such as:

```
fw ctl kdebug -f > debug.out
```

An interesting and undocumented behavior is seen when the –s and –m are omitted, but –o is still used to redirect the output. The output file will appear to be corrupted, however it actually is encoded with additional data and can be "played back" with '`fw ctl kdebug -i <file>`'. You can then switch timestamps on and off, and toggle –p data display during the playback instead of during the collection.

In practice there is little reason for using –i in this way, but if you receive a debug which appears corrupted in your text editor this might be the reason.

The ‘fw ctl debug’ command also has a –i argument (though very different) which is described as defining a “filter file”. No further details are given by the documentation.

```
[Expert@c1]# fw ctl debug -h | head
Usage: fw ctl debug [-d <strings>] [-s "<string>"] [-v ("<vs ids>"|all)] [-x] [-m <module>] [-i <filter-file|-> | -u] [+|-] <options | a
```

I have not found a practical way to leverage this yet. Trying various forms of syntax within the filter file I do know that a single IP address followed by a semicolon will be accepted as valid when compiling the filter, however the debug does not seem to be restricted to only patterns matching the filter.

ClusterXL ATRG makes reference to filters, specifically that there is a difference between filter syntax and INSPECT syntax.

** This needs further research and could prove very useful if we can reverse engineer the expected syntax. **

fishnet™ SECURITY

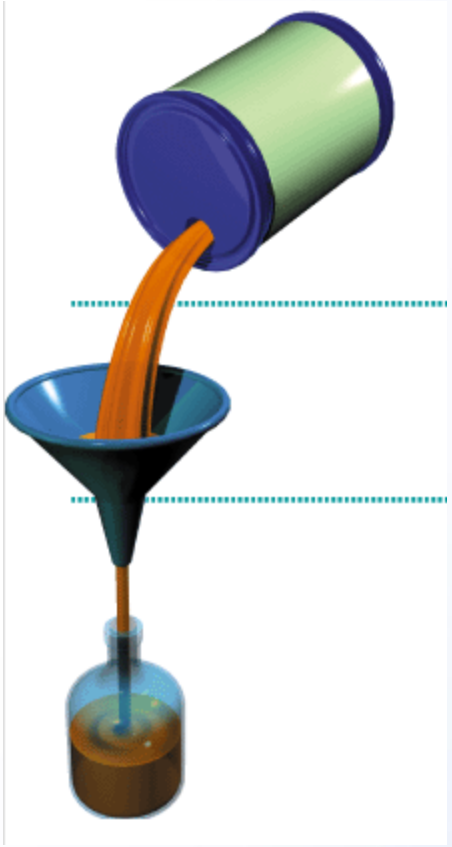
Buffers and Suppression

When kdebug or zdebug is used to view the messages in the buffer or write them to a file we introduce a bottleneck. The system must now put all of these messages somewhere- and the hard drive / terminal read this data much slower than the kernel spits them out.

Consider the analogy of a quickly pouring liquid through a funnel. The small part of the funnel is our slow bottleneck and the liquid represents kernel messages. The jar below is the screen or hard drive reading the messages.

The top part of the funnel is the buffer, which is where messages are held until they are read.

If the kernel writes messages in short bursts the HDD has time to catch up (liquid drains out) and no messages are lost.



It's guaranteed that the kernel will write messages into the buffer faster than they can be read – but if this continues for too long and without a break we will start to lose messages.

By setting the buffer size larger more messages can be held while the HDD is processing the earlier ones.

```
;13Jul2013 11:50:09;[cpu_0];[fw4_1];;
;13Jul2013 11:50:09;[cpu_0];[fw4_1];fw_findifnum: ifn=0 dev=ffff81007f0ae000;
;13Jul2013 11:50:09;[cpu_0];[fw4_1];fw_findifnum: ifn=1 dev=ffff810079cb4000;
;13Jul2013 11:50:09;[cpu_1];[fw4_0];fw_kmalloc_impl: kiss_ioctl_handler: allocate
;13Jul2013 11:50:09;[cpu_0];[fw4_1];fw_findifnum: ifn=2 dev=ffff81007a342000;
fwkdebug: buffer full. messages lost.
 fwkdebug: buffer full. messages lost.
;13Jul2013 11:50:09;[cpu_0];[fw4_1];fw_acct_update_do: dst is gw, probably proxy
2 IPP 6, bytes[0][1]: 492, bytes[0][0]: 0 ;
```

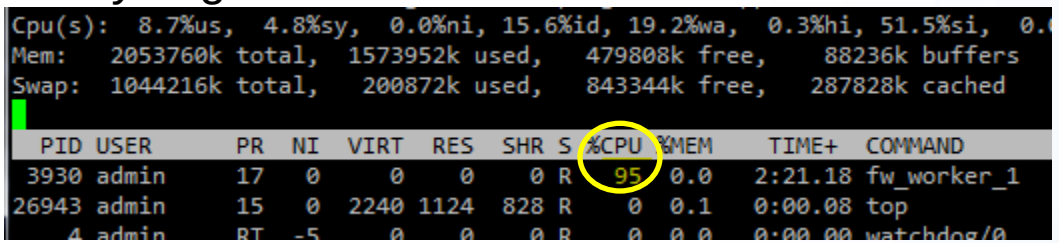
Debug Buffer Size (-buf)

So we should just max it out, right?

Not exactly. The resource intensive part of the operation is transferring the messages from the buffer onto the screen or into a file. If you enable too many debug flags and set the buffer too large, then tell the system to process everything there can be a serious performance hit.

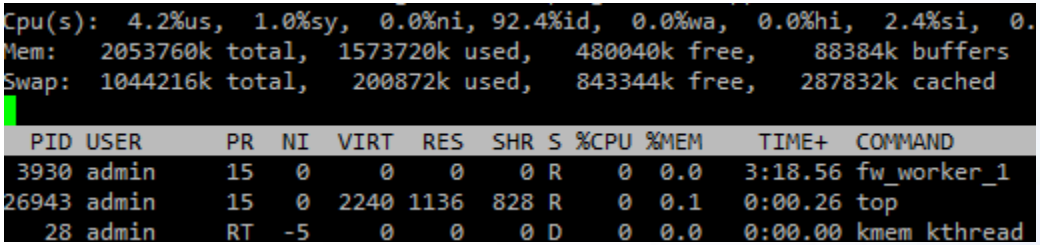
Test this for yourself. Max buffer + many flags + write to the console:

```
fw ctl debug –buf 32000
fw ctl debug –m fw + all
fw ctl kdebug –f –T
```



Now send it to /dev/null so we don't attempt to write to the console…

```
fw ctl debug –buf 32000
fw ctl debug –m fw + all
fw ctl kdebug –f –T &>/dev/null
```



kdebug Has a Buffer Too

The same can be demonstrated by outputting to the console but setting the buffer to a small size such as 500KB. In this scenario we'll lose tons of messages but the console is able to keep up with the buffer.

Up to this point we've been discussing the buffer which holds the messages temporarily before they are written to disk or console. There is also a buffer used by kdebug when moving messages out of the ring buffer and onto your screen or into a file.

```
Usage: fw ctl kdebug [-i <file> | [-f] -o <file>] [-b <buffer size>]
```

The result of setting this to smaller value is that messages are drained from the ring buffer more slowly.

I have not found a good use for this yet, but I do want to make sure the difference between buffers is addressed.

dmesg; the Kernel Ring Buffer

Even when you're not actively trying to run a debug, kernel messages can be written to /var/log/messages or flood the console. This is because messages are always being written to the kernel ring buffer whether kdebug / zdebug are listening or not. The contents of the buffer can be viewed at any time with `dmesg`.

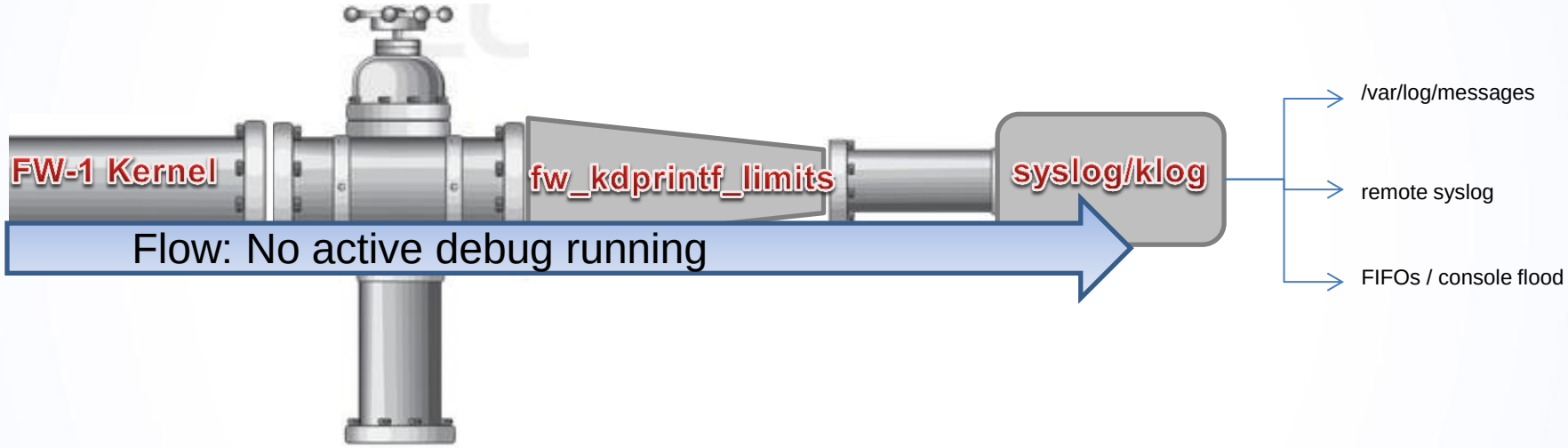
Example broadcast message:

```
-rw-rw-rw- 1 admin root      0 Jul 13 20:36 senderid_white
-rw-rw---- 1 admin root    380 Jul 13 10:45 tail
[Expert@r76-fw2:0]#
Message from syslogd@ at Sat Jul 13 20:47:00 2013 ...
r76-fw2 kernel: Sample kernel message!
```

Kernel messages alongside an sshd message in /var/log/messages:

```
)ec 13 10:30:17 sabretooth kernel: [fw_1];fwloghandle_check_string: invalid char in string (ascii 10)
)ec 13 10:30:17 sabretooth kernel: [fw_1];fwloghandle_register_string_obfuscated: failed to obfuscate given string!
)ec 13 10:32:16 sabretooth kernel: [fw_0];cpas_tcp_pass_data: asked to transfer 547 bytes which is more than in q(470)
)ec 13 10:34:37 sabretooth sshd[23079]: Did not receive identification string from 192.168.12.102
)ec 13 10:35:26 sabretooth kernel: [fw_1];cpas_tcp_pass_data: asked to transfer 610 bytes which is more than in q(533)
)ec 13 10:35:54 sabretooth kernel: [fw_2];fwloghandle_check_string: invalid char in string (ascii 10)
)ec 13 10:35:54 sabretooth kernel: [fw_2];fwloghandle_register_string_obfuscated: failed to obfuscate given string!
)ec 13 10:36:02 sabretooth kernel: [fw_0];fwloghandle_check_string: invalid char in string (ascii 10)
```

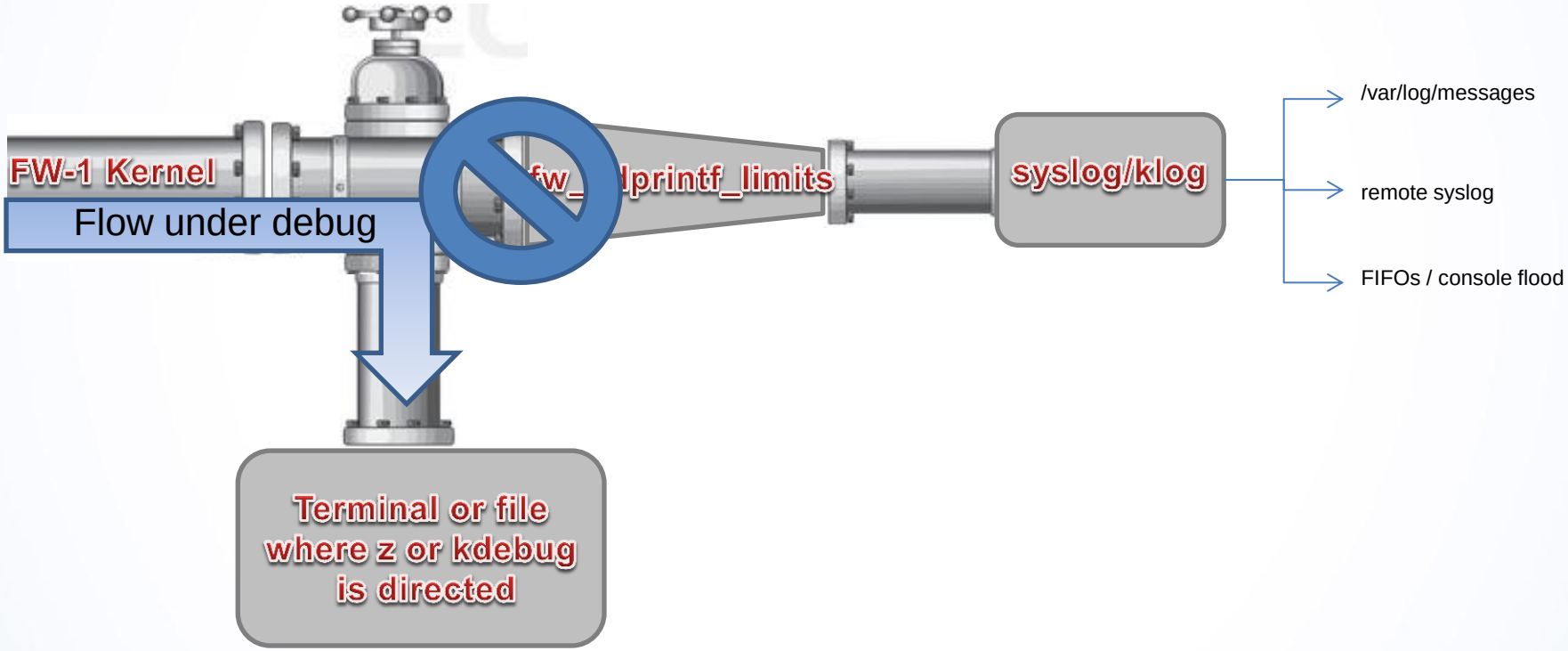
From the ring buffer, messages are typically picked up by the syslog and klog daemons, and the buffer is cleared. The messages can then be redirected to files, remote syslog servers, pipes and FIFOs, or flooded to every console.



Operating in this mode, there are two kernel parameters for log suppression. They are **fw_kdprintf_limit** and **fw_kdprintf_limit_time**, and their job is to prevent excessive debug messages when we're not actively trying to debug.

```
Jul 13 21:26:35 r76-fw2 bootsplash: Connect Failed
Jul 13 21:28:30 r76-fw2 kernel: FW-1: lost 3406 debug messages
Jul 13 21:28:30 r76-fw2 kernel: [fw4_0];fw_kfree_impl: fwkdebug_stop: freei
```

kdebug and zdebug work as a toggle, so while they are running the normal flow is interrupted.



This is why the fw_kdprintf parameters are not respected during an active debug.

fishnet™ SECURITY

Gotchas

Type and frequency thresholds are not documented very well. They are a means to control debug verbosity, but similar to the user space "TDERROR_ALL_ALL=[1-5]", the impact is arbitrary. Unless we know that a particular message only appears at a higher "type" it is best to leave these at the most verbose setting. As of the time of this writing the available levels are as follows:

Type	Frequency
INFO	COMMON
NOTICE	RARE
WRN	
ERR	
NONE	

Type and frequency are changed with `fw ctl debug -t <type> -f <freq>`.

Careful! Unlike all other levels of "type", when NONE is set there is no feedback printed to the console to warn you that your debug will be empty.

Printing Additional Fields w/ kdebug

kdebug accepts an optional argument “[-p fld1[,fld2..]” which can be used to further control fields that get printed.

This is the only exception to the earlier statement*;

“*Think of kdebug as a way to view the kernel message buffer. The actual manipulation of what appears in the buffer; modules, flags, types, and frequencies is done with `fw ctl debug`*”

The contextual help shows all possible values for -p:

```
Usage: fw ctl kdebug [-i <file> | [-f] -o <file>] [-b <buffer size>] [-t|-T] [-p fld1[,fld2..] [-m num [-s size]]
       -t/-T  to print the time field (seconds/microseconds)
       -p  to print specific fields all|proc|pid|date|mid|type|freq|topic|time|ticks|tid|text|err|host|vsid|cpu
              -m - number of cyclic files, -s - size of each
```

Among these is a “type” and “freq”, which can be leveraged to find at which threshold a known message exists.

```
[Expert@c1]# fw ctl kdebug -p type,freq | grep -viE "^(module|enabled|$)" | head
Kernel debugging buffer size: 32768KB
Messaging threshold set to type=Info freq=Common
;None;Common;[cpu_0];[fw_1];[0.0.0.0:8116 -> 192.168.25.0:8116] [SID: 00000] appi_cmi_handler_match_cb: match on context 183;
;None;Common;[cpu_0];[fw_1];[0.0.0.0:8116 -> 192.168.25.0:8116] [SID: 00000] appi_cmi_handler_match_cb: parser context;
;None;Common;[cpu_0];[fw_1];[0.0.0.0:8116 -> 192.168.25.0:8116] [SID: 00000] appi_cmi_handler_match_cb: conn returned: action [Accept];
;None;Common;[cpu_0];[fw_1];[0.0.0.0:8116 -> 192.168.25.0:8116] [SID: 00000] appi_cmi_handler_match_cb: match on context 183;
;None;Common;[cpu_0];[fw_1];[0.0.0.0:8116 -> 192.168.25.0:8116] [SID: 00000] appi_cmi_handler_match_cb: parser context;
;None;Common;[cpu_0];[fw_1];[0.0.0.0:8116 -> 192.168.25.0:8116] [SID: 00000] appi_cmi_handler_match_cb: conn returned: action [Accept];
;None;Common;[cpu_0];[fw_1];[0.0.0.0:8116 -> 192.168.25.0:8116] [SID: 00000] appi_cmi_handler_match_cb: match on context 183;
;None;Common;[cpu_0];[fw_1];[0.0.0.0:8116 -> 192.168.25.0:8116] [SID: 00000] appi_cmi_handler_match_cb: parser context;
[Expert@c1]#
```

Printing Additional Fields w/ Kernel Parameters

There are a few kernel parameters which can be adjusted (*fw ctl set int* or *fwkern.conf*) to influence the output of a debug. Most notably are those mentioned in the ClusterXL ATRG, such as `fwha_dprint_io`.

To find more like this we can list all the parameters and search for “print”, “dprint”, or similar. Not all matches are necessarily relevant. See SK33156 for more on this.

```
[Expert@g7720vsx1:0]# fw ctl zdebug 2>/dev/null | grep print &
[1] 31961
[Expert@g7720vsx1:0]# fw ctl set int listparams 1
[Expert@g7720vsx1:0]# ;[kern];[tid_2];[fw4_0];fw: fwpslglue_print_config (int) 0 has callback;
;[kern];[tid_2];[fw4_0];fw: psl_print_stack_threshold_on_handle_pkt (int) 0 ;
;[kern];[tid_2];[fw4_0];fw: print_string_dictionary (int) 0 has callback;
;[kern];[tid_2];[fw4_0];fw: print_log_messages (int) 0 has callback;
;[kern];[tid_2];[fw4_0];fw: print_sd_log_messages (int) 0 has callback;
;[kern];[tid_2];[fw4_0];fw: fwha_dprint_cu (int) 0 ;
;[kern];[tid_2];[fw4_0];fw: fwha_dprint_all_net_check (int) 0 ;
;[kern];[tid_2];[fw4_0];fw: fwha_dprint_io (int) 0 ;
;[kern];[tid_2];[fw4_0];fw: fwha_print_all_drops (int) 0 ;
;[kern];[tid_2];[fw4_0];fw: fwha_dprint_state (int) 0 ;
;[kern];[tid_2];[fw4_0];fw: fwha_dprint_sync_buffer (int) 0 ;
;[kern];[tid_2];[fw4_0];fw: fwha_dprint_arp (int) 0 ;
;[kern];[tid_2];[fw4_0];fw: fwha_dprint_trusted (int) 0 ;
;[kern];[tid_2];[fw4_0];fw: fw_print_conn_format (int) 1 ;
;[kern];[tid_2];[fw4_0];fw: ws_print_stats_enabled (int) 0 ;
;[kern];[tid_2];[fw4_0];fw: ws_print_stats_state (int) 0 ;
;[kern];[tid_2];[fw4_0];fw: print_str_map_table (int) 0 has callback;
```

Modifier flags

Similarly, there are some **flags** which act as modifiers to each line of output. Rather than causing additional lines of data to be shown in the debug they print additional *columns*. Just one of many examples of this is the 'address' flag of the APPI module.

```
Kernel debugging buffer size: 32768KB
Module: APPI
Enabled Kernel debugging options: error warning info verbose policy
Messaging threshold set to type=Info freq=Common

;[cpu_0];[fw_1];appi_rad_uf_cmi_handler_match_cb: rad service should be used in kernel;
;[cpu_0];[fw_1];appi_rad_uf_cmi_handler_prepare_rad_request: request buffer: fishnetsecurity.com/sites/default/files/resourceimages/business.png (67);
;[cpu_0];[fw_1];appi_rad_uf_cmi_handler_match_cb: found in cache;
;[cpu_0];[fw_1];appi_rad_uf_cmi_handler_match_cb: conn returned: action [Accept];
~
```

```
Kernel debugging buffer size: 32768KB
Module: APPI
Enabled Kernel debugging options: error warning info verbose address policy
Messaging threshold set to type=Info freq=Common

;[cpu_0];[fw_1];[192.168.59.10:38871 -> 50.112.125.251:80] [SID: 00994] appi_cmi_handler_match_cb: conn returned: action [Accept];
;[cpu_0];[fw_1];[192.168.59.10:38871 -> 50.112.125.251:80] [SID: 00994] appi_rad_uf_cmi_handler_match_cb: rad service should be used in kernel;
;[cpu_0];[fw_1];[192.168.59.10:38871 -> 50.112.125.251:80] [SID: 00994] appi_rad_uf_cmi_handler_prepare_rad_request: request buffer: fishnetsecu
;[cpu_0];[fw_1];[192.168.59.10:38871 -> 50.112.125.251:80] [SID: 00994] appi_rad_uf_cmi_handler_match_cb: found in cache;
;[cpu_0];[fw_1];[192.168.59.10:38871 -> 50.112.125.251:80] [SID: 00994] appi_rad_uf_cmi_handler_match_cb: conn returned: action [Accept];
~
```

Notice that the lines are the same, just now with IP information included.

SecureXL device and API debugs are enabled outside of `fw ctl debug`, as they are not part of the firewall kernel. They have their own suites of modules and flags, and everything is controlled with `sim dbg` and `fwaccel dbg`, respectively.

Unfortunately their output is sent to the same buffer as the firewall kernel module and this means that although it appears only one targeted firewall flag is being debugged, logs can be flooded with Performance Pack debug messages.

```
[Expert@r76-fw2:0]# fwaccel dbg
Usage: fwaccel dbg [-m <...>] [resetall | reset | list | all | +/- <flags>]
   -m <module>              - module of debugging
   -h                       - this help message
   resetall                 - reset all debug flags for all modules
   reset                    - reset all debug flags for module
   all                      - set all debug flags for module
   list                     - list all debug flags for all modules
   -f reset | "<5-tuple>"   - filter debug messages
   + <flags>                - set the given debug flags
   - <flags>                - unset the given debug flags

List of available modules and flags:
```

```
[Expert@r76-fw2:0]# sim dbg
Usage: sim dbg [-m <...>] [resetall | reset | list | all | mask | +/- <flags>]
   -m <module>                                  - module of debugging
   -f <sip>,<sport>,<dip>,<dport>,<proto>       - set filter ('*' for wildcard)
   -f reset                                     - reset the filter
   -h                                           - this help message
   resetall                                     - reset all debug flags for all modules
   reset                                        - reset all debug flags for module
   all                                          - set all debug flags for module
   list                                         - list all debug flags for all modules
   mask                                         - hex number of debug flags mask (with 0x prefix)
   + <flags>                                    - set the given debug flags
   - <flags>                                    - unset the given debug flags
```

If the kernel has no spare memory available you might encounter an error when trying to set the buffer with `fw ctl debug –buf <number>`, such as:

```
FW-1: Failed to allocate debugging buffer (<number>K)
```

This issue is described in sk41862, but essentially the kernel cannot spare any additional memory and a reboot will be required to allocate that amount of memory. This issue is indicative of a larger memory issue.

After rebooting you should be able to immediately allocate the maximum amount of memory to the debug buffer and nothing else will be able to ‘steal’ it away.

There is a known issue in R77.10 described in sk98625, where the translation from the user space commands to the kernel are not correctly parsed on a system using CoreXL.

If the command to set or unset a module & flag does not contain a +/- symbol it may not be applied to FW workers except for fw_worker_0.

fishnet™ SECURITY

Practical Use

Enumerating All Debugs

There are many different possible kernel debugs when you consider each flag of each module.

```
  Where possible options are:

Module: kiss
Kernel debugging options: error warning ioctl memory misc chain driver pools handles vbuf pm rem sm dfa pmdump pmint htab bench ghtab mtctx queue thread thinnfa sa
Messaging threshold set to type=Notice freq=Common

Module: kissflow
Kernel debugging options: error warning memory pm compile dfa
Messaging threshold set to type=Info freq=Common

Module: fw
Kernel debugging options: error warning cookie crypt domain ex driver filter hold if install ioctl kbuf ld log machine memory misc packet q xlate xltrc conn synatk
 ipopt link nat cifs drop route citrix misp portscan leaks mgcp sock mail spii chainfwd msnms wire balance dynlog smtp wap content mrtsync sam sock malware cmi asp
epq cvpnd cptls ftp nac span ucd acct dlp ua icmptun dnstun ips rad zeco user shmem utest
Messaging threshold set to type=Info freq=Common

Module: h323
```

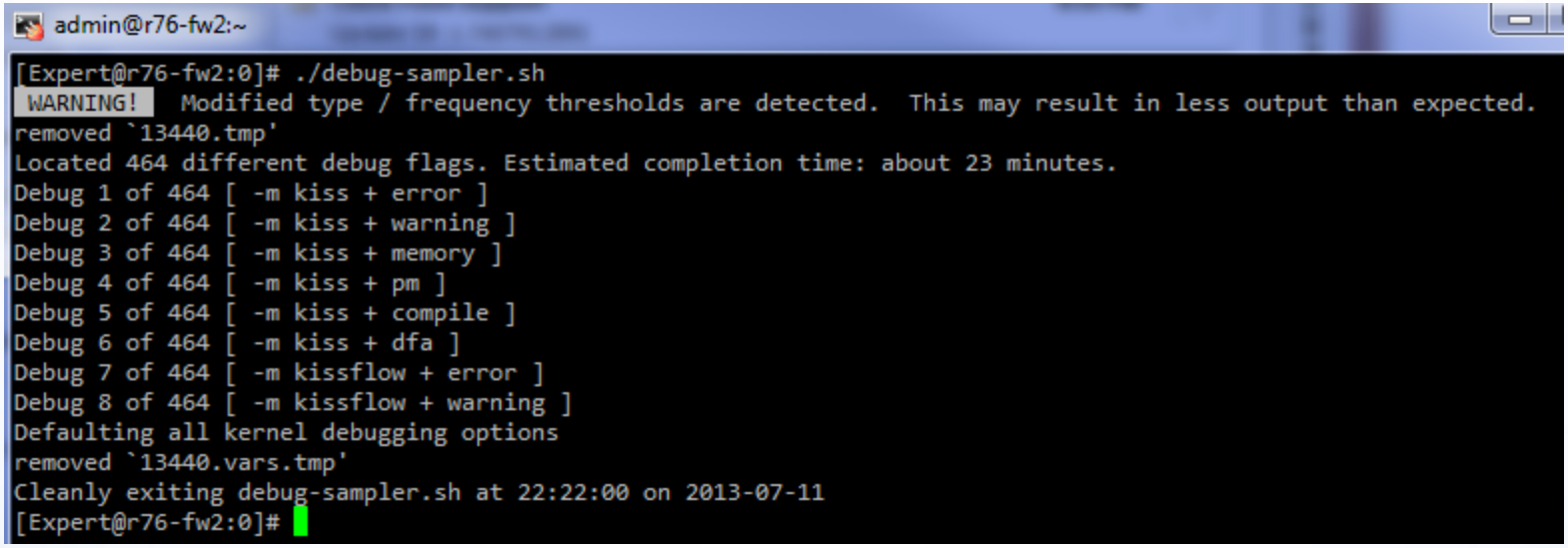
As of R77.20 there are **600+** different flags that can be debugged.

How can we possibly know which ones to enable for the information we want to see?

Enumerating All Debugs (cont)

Introducing `debug-sampler.sh`.

This is a bash script which runs kdebug with each flag for a short period of time, one at a time. The goal is to collect sample data so that we can see which flags may be interesting. The script is flexible in that it "learns" what flags are available automatically. Alternatively you can specify a file containing a module + flag pair per line if you are only interested in a specific subset of debugs.



Only ran for a few seconds to show an example.

Enumerating All Debugs (cont)

Script is available from these locations:

- http://johncpetrucci.com/archive/debug-sampler.v4.sh
- https://github.com/Jcpetrucci/bash-debug-sampler

Try it for yourself. If you find any interesting debugs let me know!

Do note that unfortunately, `debug-sampler` has no way of accounting for modifier flags* because this requires additional logic to combine several flags.

```
fwaccel dbg > $$.fwaccel.tmp 2>/dev/null
printf "Generating temporary file: %s\n" "$$.sim.tmp" >&2
sim dbg > $$.sim.tmp 2>/dev/null

# Check that the type/frequency thresholds are not modified.
grep -qEi "(type\=(none|err|wrn|notice))|(freq\=rare)" $$.fw1.tmp && echo "$(tput smso) WARNING! $(tput r

# Define the function to zero-out debug options, set a single debug flag, start the debug, let it run X s

runDebug() {
        TYPE="$1"
        MODULE="$2"
        FLAG="$3"
        printf "Debug %s of %s %s: [ -m %s + %s ]\n" "$i" "$NUMBEROFFLAGS" "$TYPE" "$MODULE" "$FLAG"
        fw ctl debug -x >/dev/null # Zero out debug modules and flags.
        fwaccel dbg resetall >/dev/null # Zero out debug modules and flags.
        sim dbg resetall >/dev/null # Zero out debug modules and flags.
        fw ctl debug -buf 16000 >/dev/null # Set the buffer size.
        RESULTFILE=Debug_log_for_${TYPE}_${MODULE}_${FLAG// /_}.txt # Create file for output and name it

        # Selector for different types of debugs (fw1/fwaccel/sim).
        case "$TYPE" in
                fw1)        fw ctl debug -m ${MODULE} ${FLAG} >/dev/null ;; # Set the module and flag
                fwaccel)    fwaccel dbg -m ${MODULE} + ${FLAG} >/dev/null; # Requires "+ flag", unlik
                            fwaccel dbg list | cat <(printf "Current fwaccel debugs:\n") - >>
                sim)        sim dbg -m ${MODULE} + ${FLAG} >/dev/null; # Requires "+ flag", unlike fw
                            sim dbg list | cat <(printf "Current sim debugs:\n") - >> "$RESUL

        esac
```