The magic dump 

when something in node goes wrong

A Cluedo like experience

Who Am I?

@lucamaraschi

Software Architect @icemobile

"{core}Dumps Watching"

Life of a debugger!

Disclaimers

it did not take ONLY 20 minutes

I do not work for Joyent!

But all these things...
only work on SmartOS!
sorry Linux!

So...

3AM...and this

```
while(true) {
ELB kills machines();
ELB spawns new ones();
}
```

ELB trashes the machines!
no way to get it back!

but it was just a Nightmare!

because it already happened in the afternoon!

EC2

Let's the hack begin!

Time to move the app to a SmartOS machine!

WHY?

dtrace + mdb!!!

Let's Dtrace the nginx machine...

```
pid$target::*ngx_http_process_request:entry
{
    this->request = (ngx_http_request_t *)copyin(arg0,sizeof(ngx_http_request_t));
    this->request_line = stringof(copyin((uintptr_t)this->request->request_line.data,
                                         this->request->request_line.len));
    printf("request line = %s\n", this->request_line);
    printf("request start sec = %d\n", this->request->start_sec);
}
```

Shows all the requests informations

and...HIGH LATENCY!

BINGO!

but it's not NGINX!

D'OH!

Let's move to the app...

restify

or die!

+ steroids ===

virgilio-http

Our logger of choice is

BUNYAN

or die!

Step 1

Let me check what's going wrong

```
prstat -c -x $(pgrep -o -x node)
```

memory ++

CPU ++

==> latency!

Looks like the usual problem!

which does not help at all!

Step 2
Check the logs!
Log Snooping via dtrace! ;-)

Thanks Trent Mick

Let's enable all logs

without restarting the machine! ;-)

```
dtrace -x strsize=4k -qn 'bunyan*:::log-*{printf("%d: %s: %s", pid, probefunc, copyinstr(arg0))}'
```

or just

```
bunyan -p $(pgrep -o -x node)
```

but still nothing... interesting or useful

Useless logs!

Step 3

Enable all the node http probes

```
nhttpsnoop -gl
```

A lot of garbage collection time...

Interesting...

Step 4

Let me see who is taking so much time...

Check for live latency on a siege attack!

```
#!/usr/sbin/dtrace -s
#pragma D option quiet
#pragma D option aggpack
#pragma D option aggsortkey

restify*:::route-start
{
   track[arg2] = timestamp;
}

restify*:::handler-start
/track[arg3]/
{
   h[arg3, copyinstr(arg2)] = timestamp;
}

restify*:::handler-done
/track[arg3] && h[arg3, copyinstr(arg2)]/
{
   @handlers[copyinstr(arg1),copyinstr(arg2)] =
     lquantize((timestamp - h[arg3, copyinstr(arg2)]) / 1000000, 0, 25, 1);
   h[arg3, copyinstr(arg2)] = 0;
}

restify*:::route-done
/track[arg2]/
{
   @routes[copyinstr(arg1)] =
     lquantize((timestamp - track[arg2]) / 1000000, 0, 25, 1);
   track[arg2] = 0;
}

END
{
        printf("ROUTE LATENCY (milliseconds)");
        printa(@routes);
        printf("\nHANDLER LATENCY (milliseconds)\n");
        printa(@handlers);
}
```

EUREKA!

I found the piece of code...

Unfortunately it's a big method! :-(

Only way to find the "leak" is to put my hand in the DUMP!

It's a dirty job...

Luckly...

we run node with

--abort-on-uncaught-exception

Generates a core dumps when the app crashes.

What is a core dump?

memory segment snapshot

That's the exciting moment...

Step 0

Run node with low level memory infos

```
UMEM_DEBUG=“default,audit=200” node –expose-gc –always-compact server.js
```

Do not do it at home!

High overhead!

Step 1

Take a dump every minute!

It keeps clean and we can see what's wrong!

```
for i in $(seq 1 100); do
  pmap -x $1 >> ./shared_volume/
  gcore $1 >> ./shared_volume/
  sleep 60
done
```

Check the memory allocation HEAP, anon or stack?

pmap diff heap++ == troubles!

Recap

- Concurrency: peak of 200 user/sec
- User signed requests
- Response Payload: ~ 16kb
- Every machine crashed every ~ 5 minutes
- Memory sky high
- CPU just on drugs!

Ok let's start

- load the core file `mdb /shared_volume/core.[pid]`
- load the v8 symbols `::load v8.so`
- I feel lucky `::findleaks`
- Too bad... :-(

But luckily something nice was added...

::findjsobjects -p {objectname}

and we can print it out...

{add_ptr}::jsprint

Unleash the beast...

::jsfunctions

and we find the ptr to the handler

Almost there...

::jsscope

shows all the variable references

Tattarattata

OK, I got the property

Now let's see all the references

ptr::findjsobjects -r

WHAT???

~ 1M references to the prop

Strange...

Ok, let me see the source...

ptr::jssource

It doesn't look right...

Reference to the non-cached var!

Is anybody referencing The wrong object?


TYPO!!!

1 char changed!
Back ALIVE!

"One reason I think debugging is so hard: you need to know much more about how computers work to debug than you do to program."

Dave Pacheco(Twitter - 18 August 2015)

THANK YOU!!!