Profiling

(in production)

Justin Azoff - Sr. Support Engineer
justin@corelight.com

What do I meen by in production? There are a lot of things you can do to profile a program to figure out what it is doing. With zeek you can process a pcap file and test various settings and scripts to determine what the problem is, but that doesn't help if you are seeing a problem with live traffic.

Process

1. **Profile**
2. **Analyze**
3. Hypotheses
4. Reproduce
5. Workaround
6. Fix

This is almost the scientific method!
Workaround can be as simple as commenting out a script that is not important

Types of profiling

- Memory profiling
- Core profiling
- Script profiling

Memory issues?

Zeek often has memory leaks?

Memory issues?

Zeek often has ~~memory leaks~~ unbounded state growth.

Unplug traffic, does memory drop? Not a leak.

State?

1320279566.452687 %1 start 192.168.2.76:52026 > 132.235.215.119:80
1320279566.831473 %2 start 192.168.2.76:52027 > 72.21.211.173:80
1320279566.748201 %1 GET / (200 "OK" [109978] www.reddit.com)
1320279566.898309 %2 GET /SVUtep3Rhg5FTRn4.jpg (200 "OK" [2562] e.thumbs.redditmedia.com)

Lets take a quick trip back 10 years or so and look at what the http log used to look like. Imagine something like reporting on every IP that requested www.reddit.com. If we just look for www.reddit.com in the logs, we would find %1. But what connection was %1?

State!

"id.resp_h": "132.235.215.119", "id.resp_p": 80,
"method": "GET", "host": "www.reddit.com", "uri": "/",
"version": "1.1",
"user_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6;...",
"request_body_len": 0, "**response_body_len**": 109978,
"status_code": 200, "status_msg": "OK",
"resp_mime_types": ["text/html"]

Now, we log “fat” records containing aggregate metadata for a single connection. The downside to this is that the metadata needs to be buffered until the request is done. In order to log response_body_len along with the {method,host,uri}, those need to be stored until the request is complete. This means that longer requests put more of a demand on memory usage.

Unbounded state :-(

HTTP Multipart responses cause unbounded http.log entries #289

Closed **JustinAzoff** opened this issue on Feb 26 · 0 comments

JustinAzoff commented on Feb 26 Contributor

Multipart responses are often used for webcams. This is seen on the network as a single http request with multiple file responses. This will cause resp_fuids, resp_mime_types, and maybe resp_filenames to grow indefinitely.

Attached is a PCAP that demonstrates this with a response containing ~1000 parts.

multipart.zip

1

Bounded state

```
global connections: count = 0;

event new_connection(c: connection) {
   ++connections;
}

event zeek_done() {
   print fmt("Saw %d connections", connections);
}
```

Unbounded state

```
global connections: set[conn_id];

event new_connection(c: connection) {
   add connections[c$id];
}

event zeek_done() {
   print fmt("Saw %d connections", |connections|);
}
```

"Works in pcap"

Cardinality matters

```
global connections: set[port];

event new_connection(c: connection) {
   add connections[c$id$resp_p];
}

event zeek_done() {
   print fmt("Saw %d connections", |connections|);
}
```

Does this blow up memory? Why not?

Expirations matter

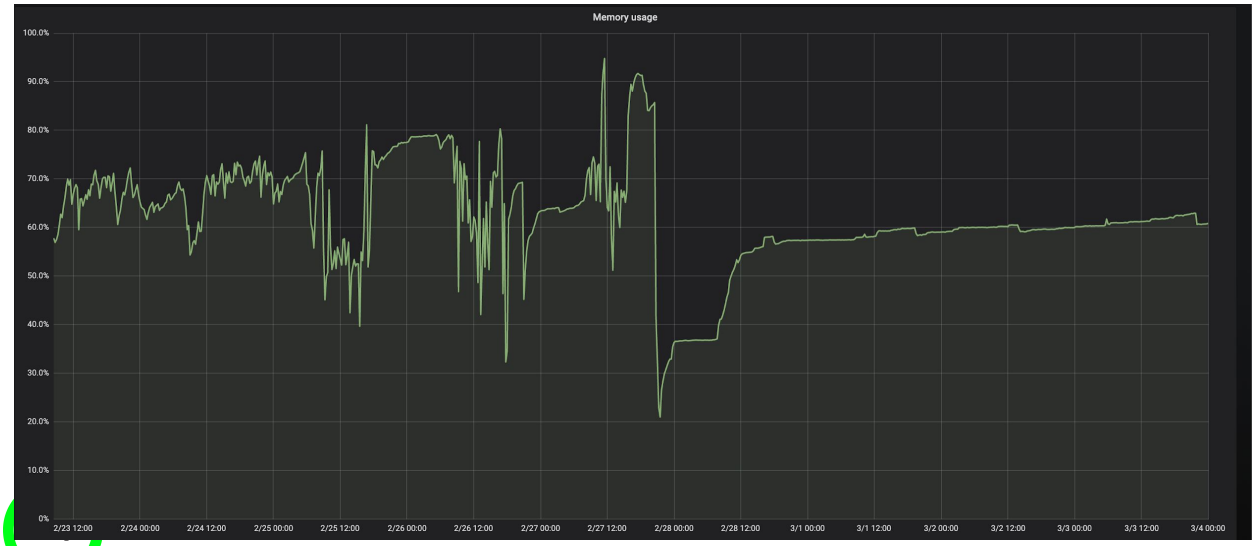
```
global conns: set[conn_id] &create_expire=1hrs;
```

Important lessons

- Know the **cardinality** of what you are storing
- Use **expiration** timers to avoid tracking things for longer than you need or expect
- **Delete** tracked data as soon as possible

For high cardinality, consider HyperLogLog
Currently most state is removed on connection_state_remove, NOT when it's last used

Case study: OOM



Case study: OOM - jemalloc

- like tcmalloc, but maintained
- **May** or **may not** perform better than glibc
- Has excellent profiling capabilities
- Zeek: ./configure --enable-jemalloc

Case study: OOM - jemalloc - verify profiling

MALLOC_CONF=stats_print:true zeek -NN 2> /tmp/jemalloc_stats

```
___ Begin jemalloc statistics ___
Version: "5.2..."
Build-time option settings
  config.cache_oblivious: true
  config.debug: false
  config.fill: true
  config.lazy_lock: false
  config.malloc_conf: "..."
  config.opt_safety_checks: false
  config.prof: true
  config.prof_libgcc: false
  config.prof_libunwind: true
  config.stats: true
  config.utrace: false
  config.xmalloc: false
...
```

No output: not using jemalloc

Case study: OOM - jemalloc - build with profiling

```
./configure
--enable-prof
--disable-prof-libgcc
--disable-prof-gcc
--enable-prof-libunwind
```

No output: not using jemalloc

Case study: OOM - jemalloc profiling

Uses environment variable to enable

```
MALLOC_CONF=
prof:true,
prof_prefix:jeprof.out,
lg_prof_interval:32
```

Case study: OOM - jemalloc profiling - output files

```
jeprof.out.5165.3094.i3094.heap
jeprof.out.5165.3095.i3095.heap
jeprof.out.5165.3096.i3096.heap
jeprof.out.5165.3097.i3097.heap
jeprof.out.5165.3098.i3098.heap
jeprof.out.5165.3099.i3099.heap
jeprof.out.5165.3100.i3100.heap
```

Uhhhh now what?

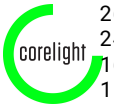
Case study: OOM - jemalloc profiling - .heap files

```
@ 0x7fa14d9a1a32 0x7fa14d94c8b4
0x7fa14d9c6669 0x556df0853469 0x556df08f4468
0x556df06785bb 0x556df0679ee9 0x556df067ba4d
0x556df0658c37 0x556df08b1d90
 0x556df0658ed9 0x556df0509ab3 0x7fa14cb8209b
0x556df051663a
  t*: 10: 2560 [0: 0]
  t0: 10: 2560 [0: 0]
```

Not human readable. Contain memory addresses of the stack trace

Case study: OOM - jemalloc profiling - jeprof

```
# jeprof $(which zeek) $(ls /usr/local/zeek/spool/worker-1-1/jeprof* -tr|tail -n 1)
Using local file /usr/local/zeek/bin//zeek.
Using local file /usr/local/zeek/spool/worker-1-1/jeprof.out.7973.130.i130.heap.
Welcome to jeprof!  For help, type 'help'.
(jeprof) top20
Total: 1011.9 MB
   108.8  10.8%  10.8%    108.8  10.8% BroString::Set@2c6430
    99.0   9.8%  20.5%    207.7  20.5% file_analysis::X509::ParseSAN
    76.1   7.5%  28.1%     78.6   7.8% i2d_ASN1_SET_ANY
    68.0   6.7%  34.8%     68.0   6.7% BroString::Set@2c64f0
    63.9   6.3%  41.1%     63.9   6.3% ASN1_STRING_set
    58.0   5.7%  46.8%     58.0   5.7% CRYPTO_zalloc
    50.0   4.9%  51.8%    200.8  19.8% StringVal::StringVal
    36.0   3.6%  55.3%     38.0   3.8% RuleMatcher::InitEndpoint
    35.5   3.5%  58.8%    361.0  35.7% file_analysis::X509Common::ParseExtension
    34.0   3.4%  62.2%     34.0   3.4% safe_malloc (inline)
    31.2   3.1%  65.3%     31.2   3.1% std::vector::_M_default_append (inline)
    27.5   2.7%  68.0%     27.5   2.7% safe_realloc (inline)
    26.0   2.6%  70.6%     26.0   2.6% analyzer::pia::PIA::AddToBuffer
    24.0   2.4%  73.0%     24.0   2.4% TableVal::Init
    16.5   1.6%  74.6%     17.0   1.7% ValManager::ValManager
    15.5   1.5%  76.1%     32.5   3.2% Connection::BuildConnVal
```



This is showing an issue with large amounts of SSL related allocations.

Case study: OOM - jemalloc profiling - jeprof cumulative

```
(jeprof) top50 --cum
Total: 1011.9 MB
     0.0   0.0%   0.0%   1004.3  99.3% __libc_start_main
     0.0   0.0%   0.0%   1004.3  99.3% _start
     2.5   0.2%   0.2%   1003.8  99.2% main
     0.0   0.0%   0.2%    938.7  92.8% net_run
     0.0   0.0%   0.2%    936.7  92.6% iosource::PktSrc::Process
     0.0   0.0%   0.2%    936.7  92.6% iosource::PktSrc::Process (inline)
     0.0   0.0%   0.2%    936.7  92.6% net_packet_dispatch
     0.0   0.0%   0.2%    917.7  90.7% NetSessions::DoNextPacket
     0.0   0.0%   0.2%    917.7  90.7% NetSessions::NextPacket
     0.0   0.0%   0.2%    855.7  84.6% Connection::NextPacket
     0.0   0.0%   0.2%    855.7  84.6% analyzer::Analyzer::NextPacket
     0.0   0.0%   0.2%    855.2  84.5% analyzer::tcp::TCP_Analyzer::DeliverPacket
     0.0   0.0%   0.2%    810.2  80.1% analyzer::tcp::TCP_Analyzer::DeliverPacket (inline)
     0.0   0.0%   0.2%    810.2  80.1% analyzer::tcp::TCP_Endpoint::DataSent
```

We can also look at the cumulative allocations..

Case study: OOM - jemalloc profiling - jeprof cumulative

```
0.0   0.0%   0.2%    810.2  80.1% analyzer::tcp::TCP_Reassembler::DataSent
0.0   0.0%   0.2%    799.2  79.0% analyzer::Analyzer::ForwardStream
0.0   0.0%   0.2%    799.2  79.0% analyzer::Analyzer::NextStream
0.0   0.0%   0.2%    799.2  79.0% analyzer::tcp::TCP_Reassembler::BlockInserted
0.0   0.0%   0.2%    799.2  79.0% analyzer::tcp::TCP_Reassembler::DeliverBlock
0.0   0.0%   0.2%    738.1  72.9% analyzer::ssl::SSL_Analyzer::DeliverStream
0.0   0.0%   0.2%    738.1  72.9% binpac::SSL::SSLPDU::ParseBuffer
0.0   0.0%   0.2%    738.1  72.9% binpac::SSL::SSLPDU::ParseBuffer (inline)
0.0   0.0%   0.2%    738.1  72.9% binpac::SSL::SSLRecord::ParseBuffer
0.0   0.0%   0.2%    738.1  72.9% binpac::SSL::SSL_Flow::NewData
0.0   0.0%   0.2%    736.1  72.8% analyzer::ssl::SSL_Analyzer::SendHandshake
0.0   0.0%   0.2%    736.1  72.8% binpac::SSL::Handshake::Parse
0.0   0.0%   0.2%    736.1  72.8% binpac::SSL::Handshake::Parse (inline)
0.0   0.0%   0.2%    736.1  72.8% binpac::SSL::PlaintextRecord::Parse
0.0   0.0%   0.2%    736.1  72.8% binpac::TLSHandshake::Handshake::Parse
```

We can also look at the cumulative allocations..

Case study: OOM - jemalloc profiling - jeprof cumulative

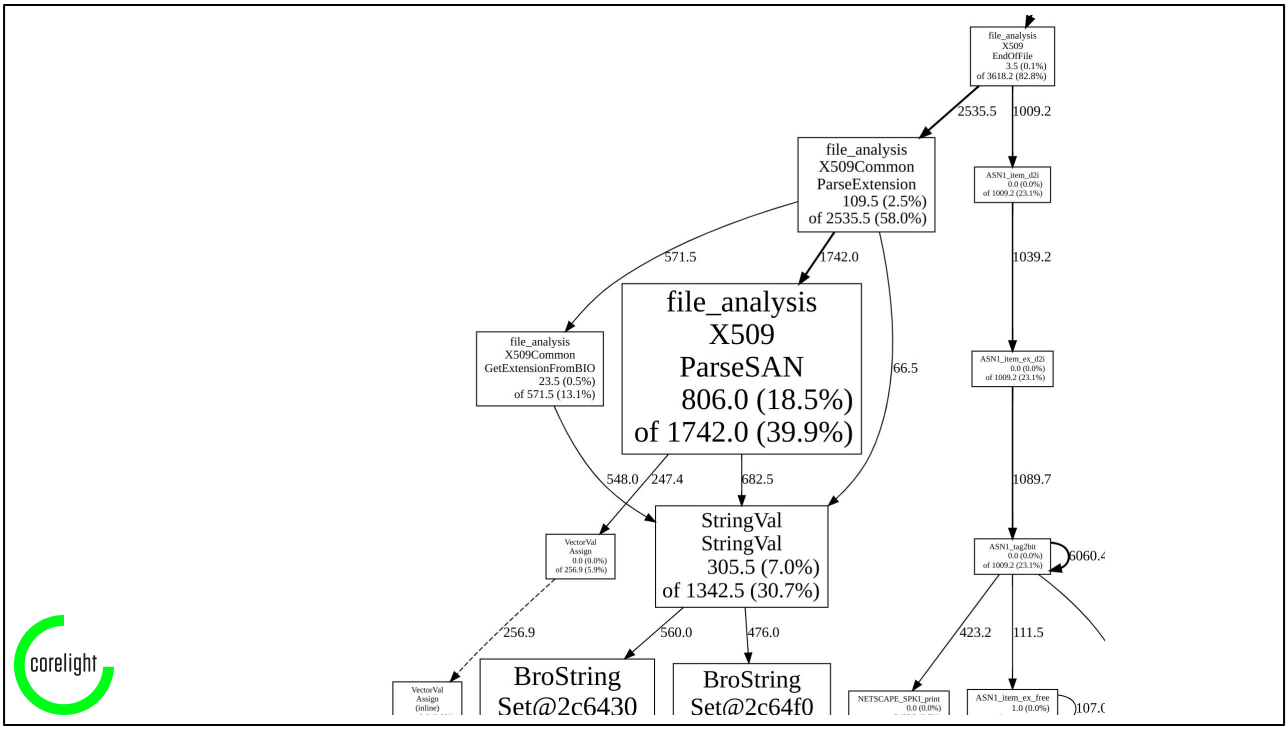
```
     0.0   0.0%   0.2%    736.1  72.8% binpac::TLSHandshake::HandshakePDU::ParseBuffer
     0.0   0.0%   0.2%    736.1  72.8% binpac::TLSHandshake::HandshakeRecord::ParseBuffer
     0.0   0.0%   0.2%    736.1  72.8% binpac::TLSHandshake::Handshake_Flow::NewData
     0.0   0.0%   0.2%    735.6  72.7% binpac::TLSHandshake::Certificate::Parse
     0.0   0.0%   0.2%    735.6  72.7% binpac::TLSHandshake::Handshake_Conn::proc_certifica
     0.0   0.0%   0.2%    735.6  72.7% binpac::TLSHandshake::Handshake_Conn::proc_v3_certif
     0.0   0.0%   0.2%    627.1  62.0% file_analysis::File::EndOfFile
     0.0   0.0%   0.2%    627.1  62.0% file_analysis::Manager::RemoveFile
     0.0   0.0%   0.2%    619.1  61.2% file_analysis::X509::EndOfFile
    35.5   3.5%   3.8%    361.0  35.7% file_analysis::X509Common::ParseExtension
     0.0   0.0%   3.8%    221.1  21.8% ASN1_item_d2i
     0.0   0.0%   3.8%    221.1  21.8% ASN1_item_ex_d2i
```

We can also look at the cumulative allocations.. Finally towards the end we see X509Common::ParseExtension

Case study: OOM - jemalloc profiling - jeprof svg

```
# jeprof `which zeek` jeprof.out.5165.5521.i5521.heap --svg > /tmp/prof.svg
Using local file /usr/local/zeek/bin//zeek.
Using local file jeprof.out.5165.5521.i5521.heap.
Dropping nodes with <= 65.4 MB; edges with <= 13.1 abs(MB)

# jeprof `which zeek` jeprof.out.5165.5521.i5521.heap --dot > /tmp/prof.dot
# dot -Tsvg < prof.dot > prof.svg
```

This can often be easier visualized as an SVG



This can often be easier visualized as an SVG

Case study: OOM - ssl-clear-memory

Delete tracked data as soon as you know you no longer need it

```
hook SSL::ssl_finishing(c: connection) &priority=-200
{
    if (c$ssl?$cert_chain)
        delete c$ssl$cert_chain;
    if (c$ssl?$client_cert_chain)
        delete c$ssl$client_cert_chain;
}
```

More of a workaround. The bigger fix would be an overhaul of how the ssl analyzer parses certificates

Case study: RuleMatcher - jeprof top20

```
# jeprof $(which zeek) $(ls /usr/local/zeek/spool/worker-1-1/jeprof* -tr|tail -n 1)
Using local file /usr/local/zeek/bin//zeek.
Using local file /usr/local/zeek/spool/worker-1-1/jeprof.out.7973.130.i130.heap.
Welcome to jeprof!  For help, type 'help'.
(jeprof) top20
Total: 1011.9 MB
   108.8  10.8%  10.8%    108.8  10.8% BroString::Set@2c6430
    99.0   9.8%  20.5%    207.7  20.5% file_analysis::X509::ParseSAN
    76.1   7.5%  28.1%     78.6   7.8% i2d_ASN1_SET_ANY
    68.0   6.7%  34.8%     68.0   6.7% BroString::Set@2c64f0
    63.9   6.3%  41.1%     63.9   6.3% ASN1_STRING_set
    58.0   5.7%  46.8%     58.0   5.7% CRYPTO_zalloc
    50.0   4.9%  51.8%    200.8  19.8% StringVal::StringVal
    36.0   3.6%  55.3%     38.0   3.8% RuleMatcher::InitEndpoint
    35.5   3.5%  58.8%    361.0  35.7% file_analysis::X509Common::ParseExtension
    34.0   3.4%  62.2%     34.0   3.4% safe_malloc (inline)
    31.2   3.1%  65.3%     31.2   3.1% std::vector::_M_default_append (inline)
    27.5   2.7%  68.0%     27.5   2.7% safe_realloc (inline)
    26.0   2.6%  70.6%     26.0   2.6% analyzer::pia::PIA::AddToBuffer
    24.0   2.4%  73.0%     24.0   2.4% TableVal::Init
    16.5   1.6%  74.6%     17.0   1.7% ValManager::ValManager
    15.5   1.5%  76.1%     32.5   3.2% Connection::BuildConnVal
```

Another thing seen in the previous profile was RuleMatcher::InitEndpoint using a lot of memory, what is that?

Case study: RuleMatcher - list RuleMatcher::InitEndpoint

```
     .      .  733:         if ( hdr_test->level <= RE_level )
     .      .  734:             {
     .      .  735:             for ( int i = 0; i < Rule::TYPES; ++i )
     .      .  736:                 {
     .      .  737:                 loop_over_list(hdr_test->psets[i], j)
     .      .  738:                     {
     .      .  739:                     RuleHdrTest::PatternSet* set =
     .      .  740:                         hdr_test->psets[i][j];
     .      .  741:
     .      .  742:                     assert(set->re);
     .      .  743:
     .      .  744:                     RuleEndpointState::Matcher* m =
  12.5   12.5  745:                         new RuleEndpointState::Matcher;
  42.0   48.0  746:                     m->state = new RE_Match_State(set->re);
     .      .  747:                     m->type = (Rule::PatternType) i;
     .      .  748:                     state->matchers.append(m);
     .      .  749:                     }
     .      .  750:                 }
     .      .  751:             }
```

Case study: RuleMatcher - bug report

file-magic sigs are being checked in places they shouldn't be #554

Closed **JustinAzoff** opened this issue 9 days ago · 3 comments · Fixed by #559

JustinAzoff commented 9 days ago — Member

I've been seeing RuleMatcher::InitEndpoint show up in memory profiling as one of the top allocations.

Digging into it, the allocations are coming from

```
RuleEndpointState::Matcher* m =
    new RuleEndpointState::Matcher;
m->state = new RE_Match_State(set->re);
```

Assignees

jsiwek

Labels

Area: File Ana

Area: Protocc

Area: Signatu

Type: Bug

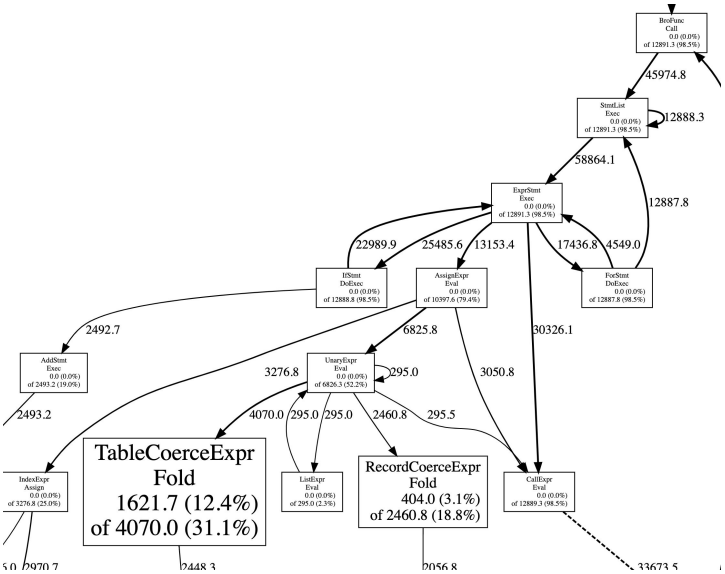
https://github.com/zeek/zeek/issues/554

jeprof - sometimes not helpful

```
# jeprof `which zeek` jeprof.out.5165.5521.i5521.heap
Using local file /usr/local/zeek/bin//zeek.
Using local file jeprof.out.5165.5521.i5521.heap.
top10
Welcome to jeprof!  For help, type 'help'.
(jeprof) Total: 13088.2 MB
  2450.8  18.7%  18.7%   2450.8  18.7% TableVal::Init
  2020.3  15.4%  34.2%   2020.3  15.4% Dictionary::Init
  1621.7  12.4%  46.6%   4070.0  31.1% TableCoerceExpr::Fold
  1278.1   9.8%  56.3%   1278.1   9.8% safe_realloc (inline)
  1244.2   9.5%  65.8%   1244.2   9.5% safe_malloc (inline)
   649.0   5.0%  70.8%    649.0   5.0% CompositeHash::ComputeHash
   628.5   4.8%  75.6%   4435.7  33.9% Dictionary::Insert@2dd790
   590.0   4.5%  80.1%   5122.0  39.1% TableVal::Assign@389b40
   588.5   4.5%  84.6%    588.5   4.5% BifFunc::bro_network_time
   491.0   3.8%  88.3%   1754.6  13.4% Dictionary::Insert@2dd2f0
```

Not perfect: safe_* shouldn’t really show up, don’t know which dictionary or dictionaries it is

jeprof - sometimes not helpful



Quiz

What is the most common Dict size?

Quiz

```
 void Dictionary::DeInit()
        {
+       printf("DictEntry::DeInit length=%d max=%d total=%d\n", Length(), MaxLength(), NumCumulativeInserts());
```

```
justin@z420:/dev/shm$ cat dicts.txt |sort|uniq  -c|sort -rn
  70216 DictEntry::DeInit length=0 max=0 total=0
  20000 DictEntry::DeInit length=0 max=1 total=1
  10129 DictEntry::DeInit length=1 max=1 total=1
     14 DictEntry::DeInit length=2 max=2 total=2
      3 DictEntry::DeInit length=5 max=5 total=5
      3 DictEntry::DeInit length=3 max=3 total=3
      2 DictEntry::DeInit length=8 max=8 total=8
      2 DictEntry::DeInit length=4 max=4 total=4
      2 DictEntry::DeInit length=37 max=37 total=37
      1 DictEntry::DeInit length=9 max=9 total=9
      1 DictEntry::DeInit length=742 max=742 total=1690
```

Defer initialization of Dicts and Lists #277

Merged jsiwek merged 1 commit into `master` from `topic/jazoff/datastructures-defer-init` on Mar 13

jeprof future improvements

- **Patch jemalloc to understand zeek stack traces and replace BroFunc::Call with script function names**
- Cache symbols to increase performance
- Automate the profiling

BroFunc::Call is not helpful

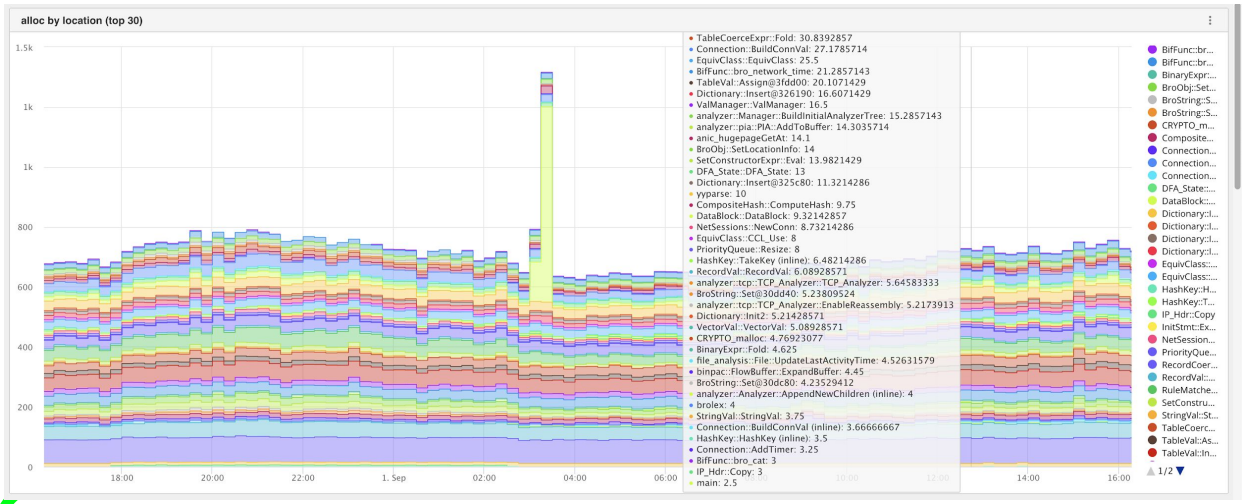
zeek-jemalloc-profiling

Automates the profiling :-)

`zkg install zeek-jemalloc-profiling`

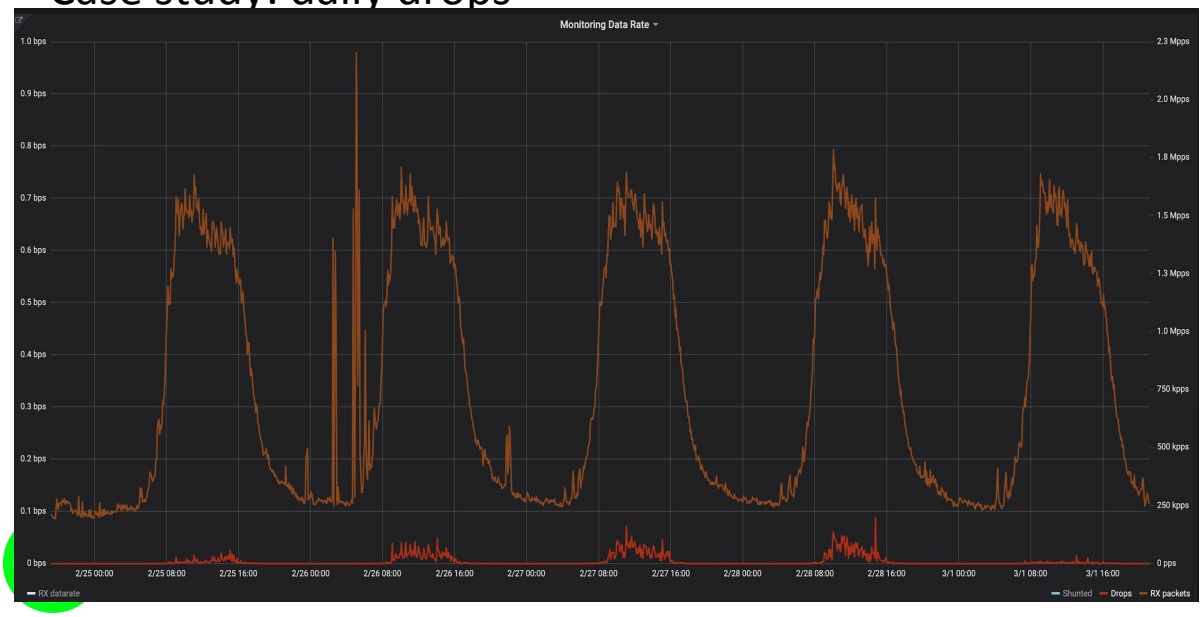
- Helps set env vars
- Converts output files into json for humio/splunk/elastic

jeprof output visualized

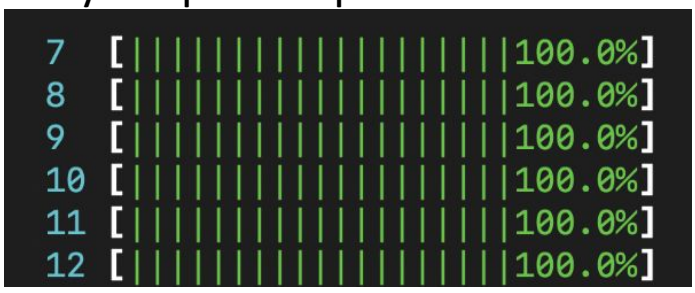


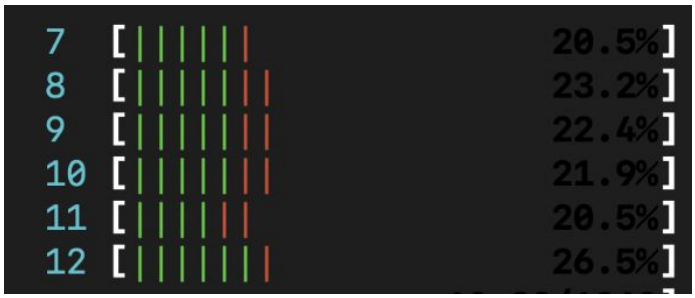
corelight

Case study: daily drops



Case study: daily drops - htop before and after restart





corelight

Core profiling with perf

```
perf top -F 199 -p 7973 # process
perf top -F 199 -C 4    # core
perf record -F 199 -C 4 # record
perf record -F 199 -C 4 -- sleep 60 # for 60s
```

Case study: daily drops - perf top

```
 21.74%  bro                     [.] BaseList::remove
 20.56%  bro                     [.] BaseList::remove_nth
  1.25%  bro                     [.] Dictionary::DoRemove
  0.33%  bro                     [.] Dictionary::NextEntry
```

Case study: daily drops - gdb

```
0x0000563eb87f10c0 in BaseList::remove (this=0x563f14155928, a=a@entry=0x563eff1a9100) at /home/justin/src/bro-2.6.3/src/List.cc:120
120             for ( i = 0; i < num_entries && a != entry[i]; ++i )
(gdb) bt
#0  0x0000563eb87f10c0 in BaseList::remove (this=0x563f14155928, a=a@entry=0x563eff1a9100) at /home/justin/src/bro-2.6.3/src/List.cc:120
#1  0x0000563eb877d64e in DictEntryPList::remove (a=0x563eff1a9100, this=<optimized out>) at /home/justin/src/bro-2.6.3/src/Dict.h:13
#2  Dictionary::DoRemove (this=this@entry=0x563ebc5bfd00, entry=entry@entry=0x563eff1a9100, h=h@entry=4150, chain=chain@entry=0x563edcfb8e90,
    chain_offset=chain_offset@entry=0) at /home/justin/src/bro-2.6.3/src/Dict.cc:254
#3  0x0000563eb877d78c in Dictionary::Remove (this=0x563ebc5bfd00, key=0x563f12697db0, key_size=48, hash=<optimized out>,
    dont_delete=dont_delete@entry=false) at /home/justin/src/bro-2.6.3/src/Dict.cc:209
#4  0x0000563eb884c946 in TableEntryValPDict::RemoveEntry (key=0x563f0e706130, this=<optimized out>) at /home/justin/src/bro-2.6.3/src/Hash.h:67
#5  TableVal::Delete (this=this@entry=0x563ebc5bfbb0, index=index@entry=0x563f46da96e0) at /home/justin/src/bro-2.6.3/src/Val.cc:2084
#6  0x0000563eb87a0241 in IndexExpr::Delete (this=<optimized out>, f=0x563f2aedd200) at /home/justin/src/bro-2.6.3/src/Val.h:286
#7  0x0000563eb882f0a8 in DelStmt::Exec (this=<optimized out>, f=<optimized out>, flow=<optimized out>) at /home/justin/src/bro-2.6.3/src/Stmt.cc:1159
#8  0x0000563eb8830287 in StmtList::Exec (this=0x563ebc5cc460, f=0x563f2aedd200, flow=@0x7ffebadeb554: FLOW_NEXT)
    at /home/justin/src/bro-2.6.3/src/List.h:76
#9  0x0000563eb87cac46 in BroFunc::Call (this=<optimized out>, args=0x563ed5f66610, parent=0x563f33061e60) at /usr/include/c++/8/bits/stl_vector.h:948
#10 0x0000563eb87a89b1 in CallExpr::Eval (this=0x563ebc5cf510, f=0x563f33061e60) at /home/justin/src/bro-2.6.3/src/Expr.cc:4841
#11 0x0000563eb8832a64 in ExprStmt::Exec (this=0x563ebc5cf6e0, f=0x563f33061e60, flow=@0x7ffebadeb784: FLOW_NEXT)
    at /home/justin/src/bro-2.6.3/src/Stmt.cc:352
#12 0x0000563eb8830287 in StmtList::Exec (this=0x563ebc5ceb90, f=0x563f33061e60, flow=@0x7ffebadeb784: FLOW_NEXT)
    at /home/justin/src/bro-2.6.3/src/List.h:76
#13 0x0000563eb87cac46 in BroFunc::Call (this=<optimized out>, args=0x563ec2fd0700, parent=0x0) at /usr/include/c++/8/bits/stl_vector.h:948
#14 0x0000563eb884e3df in TableVal::CallExpireFunc (this=<optimized out>, idx=0x563f167e97b0) at /home/justin/src/bro-2.6.3/src/Val.h:252
#15 0x0000563eb884e61a in TableVal::DoExpire (this=0x563ebc5bfbb0, t=1567362103.1253321) at /home/justin/src/bro-2.6.3/src/Val.cc:2367
```

Case study: daily drops - gdb

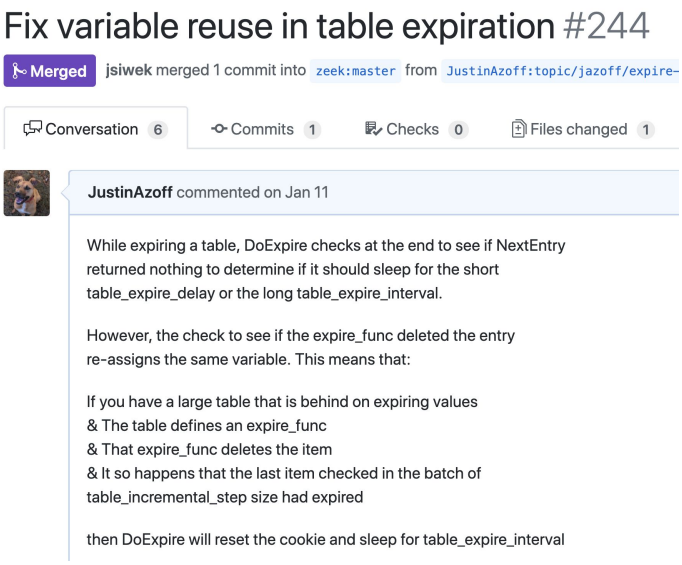
```
(gdb) print this->location->filename
value has been optimized out
(gdb) up
#5  TableVal::Delete (this=this@entry=0x563ebc5bfbb0, index=index@entry=0x563f46da96e0) at /ho
2084            TableEntryVal* v = k ? AsNonConstTable()->RemoveEntry(k) : 0;
(gdb) print this->location->filename
$1 = 0x563ebc4c1960 "/home/justin/src/bro-2.6.3/scripts/base/frameworks/tunnels/./main.bro"
(gdb) print this->location->first_line
$2 = 82
```

From https://www.zeek.org/support/reporting-problems.html

Case study: daily drops - tunnels/main

```
global active: table[conn_id] of Info = table()
&read_expire=expiration_interval &expire_func=expire;

function close(tunnel: Info, action: Action) {
        tunnel$action = action;
        tunnel$ts = network_time();
        Log::write(LOG, tunnel);
        delete active[tunnel$id]; }

function expire(t: table[conn_id] of Info, idx: conn_id):
interval {
        close(t[idx], EXPIRE);
        return 0secs; }
```

Case study: daily drops - pull request



Fix variable reuse in table expiration #244

Merged | jsiwek merged 1 commit into zeek:master from JustinAzoff:topic/jazoff/expire-

Conversation 6 | Commits 1 | Checks 0 | Files changed 1

JustinAzoff commented on Jan 11

While expiring a table, DoExpire checks at the end to see if NextEntry returned nothing to determine if it should sleep for the short table_expire_delay or the long table_expire_interval.

However, the check to see if the expire_func deleted the entry re-assigns the same variable. This means that:

If you have a large table that is behind on expiring values
& The table defines an expire_func
& That expire_func deletes the item
& It so happens that the last item checked in the batch of table_incremental_step size had expired

then DoExpire will reset the cookie and sleep for table_expire_interval

corelight

Case study: daily drops - pull request

JustinAzoff commented on Jan 12 Author Member

I have found out that a side affect of this bug actually causes a much larger problem.. the real reason why performance tanks is because DoExpire was leaking RobustCookies.

Normally, NextEntry is what removes the robust cookie at the end of iterations. However, since DoExpire was mistakingly thinking it was finished and starting over, the iteration was never finishing and tbl->cookies was gaining a cookie at each call to DoExpire.

If that was not bad enough, and this where things go completely off the rails:

```
void Dictionary::StartChangeSize(int new_size)
    {
    // Only start resizing if there isn't any iteration in progress.
    if ( cookies.length() > 0 )
        return;
```

Because the cookie list is never empty, the dictionary is never resized, and basically becomes a giant linked list... making the table expiration that is already behind eventually dominate 100% of runtime.

corelight

Case study: 2.6 drops

Description

Observing packet drops in 2.6 (not in 2.5) but not seeing any ***malfs.*** I had 12 hour and 24 hour runs This is only in 2.6 and not in 2.5. Both appliances were sensing the same traffic pattern at the same time.

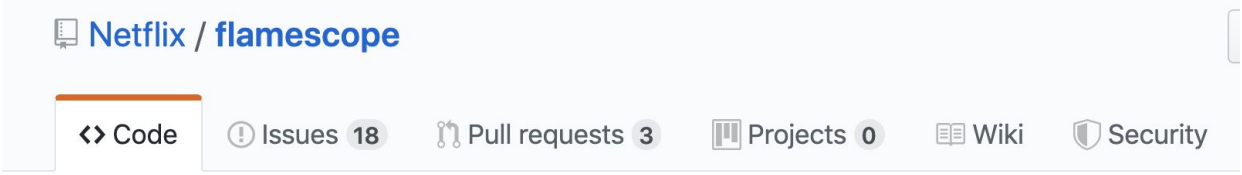
We ran into this issue while load testing 2.6 using our IXIA

Case study: 2.6 drops - process

```
perf record -F 199 -C 1-.. -- sleep 60
```

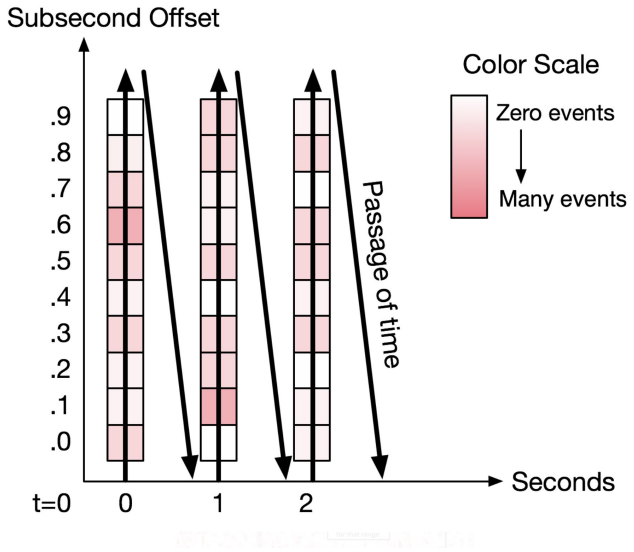
- In a loop, capture perf output for a minute at a time for all worker cores.
- Delete output file unless drops are detected.
- Filter the output to the individual core that drops were seen on.
- Set this up and come back in 8 hours.

Case study: 2.6 drops - tools



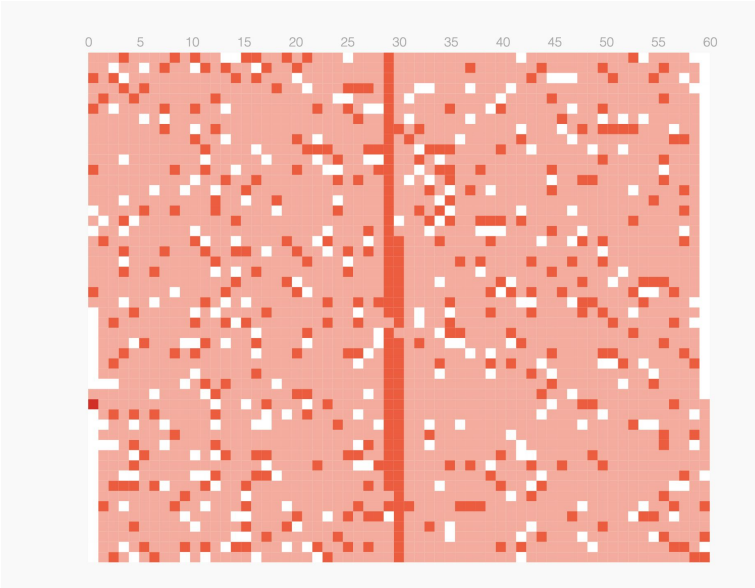
FlameScope is a visualization tool for exploring different time ranges as Flame Graphs.

Case study: 2.6 drops - tools



corelight

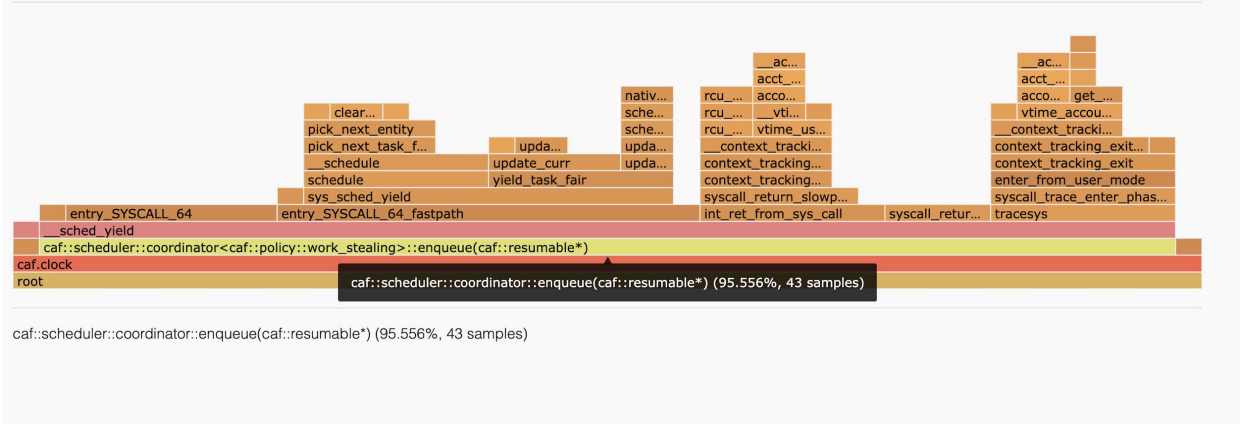
Case study: 2.6 drops - FlameScope view



corelight

Red bad. Big red line more bad

Case study: 2.6 drops - FlameGraph view



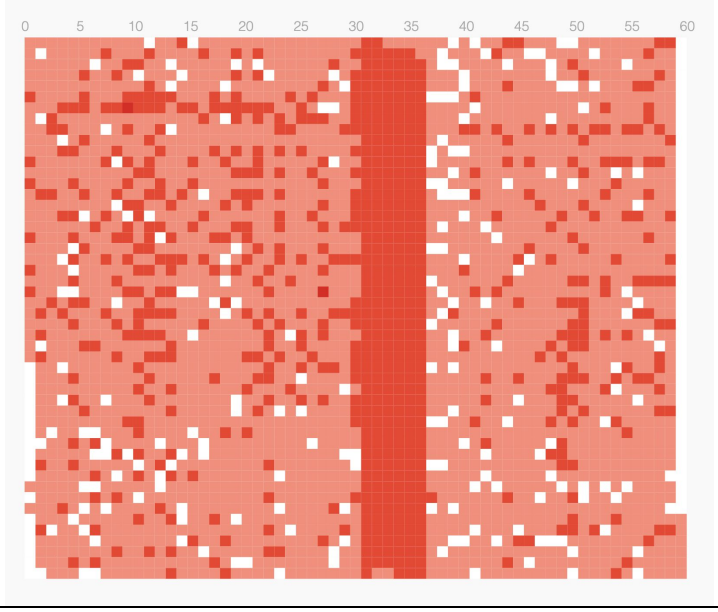
corelight

Found the problem to be inside the caf scheduler

Case study: reassembler - perf top

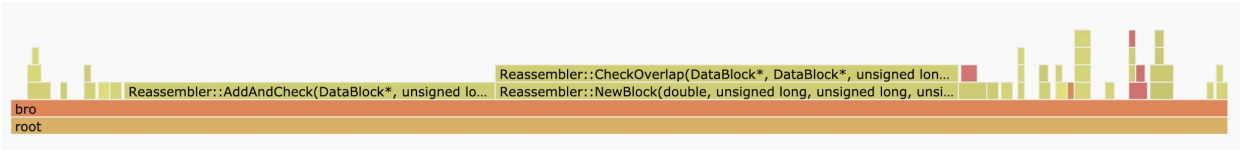
```
PerfTop:     199 irqs/sec  kernel: 1.5%  exact:  0.0% [199Hz cycles],  (target_pid: 63291)
-------------------------------------------------------------------------------------------------------------------------------

  44.49%  bro                          [.] _ZN11Reassembler12CheckOverlapEP9DataBlockS1_mmPKh.part.5
  44.08%  bro                          [.] _ZN11Reassembler11AddAndCheckEP9DataBlockmmPKh
   3.47%  libcaf_io.so.0.16.2          [.] _ZN3caf2io15abstract_broker6wr_bufENS0_17connection_handleE
   0.63%  libjemalloc.so.2             [.] malloc
   0.33%  bro                          [.] _ZNK10Dictionary6LookupEPKvim
   0.29%  bro                          [.] _ZN13PriorityQueue10BubbleDownEi
   0.23%  [kernel]                     [k] timerqueue_add
```

Case study: reassembler - FlameScope view



corelight

Case study: reassembler - FlameGraph view



Case study: reassembler - process

```
perf record -F 199 -C 5 -- sleep 60
```

- In a loop, capture perf output for a minute at a time for one worker.
- Use the -w option to write out full pcap from that worker
- Delete output file and pcap unless drops are detected.
- Delete the pcap unless the profile contained Reassembler
- Set this up and come back in ~~8 hours~~ 2 days.

Case study: reassembler

Improve reassembly worst-case performance #576

Merged rsmmr merged 8 commits into `master` from `topic/jsiwek/reassembly-improvements-map` 8 days ago

Conversation 7 Commits 8 Checks 1 Files changed 9

jsiwek commented 19 days ago • edited Member

Fixes #575

Changes the internal reassembly data structure to use `std::map` instead of a linked-list.

corelight

Case study: reassembler

Pathological Case

Using `pathological-reassembly-duplicate-segments.pcap`

Master:

```
$ time -f "%e" zeek -b -r pathological-reassembly-duplicate-segments.pcap
15.77
```

Branch:

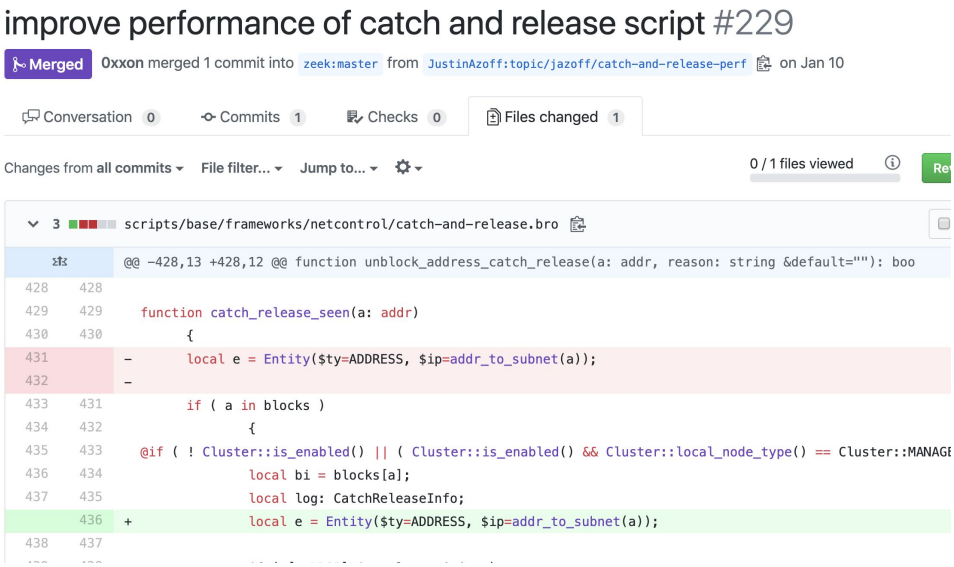
```
$ time -f "%e" zeek -b -r pathological-reassembly-duplicate-segments.pcap
0.21
```

Way better.

Script profiling

We have a plugin, but it’s not ready yet.. In the meantime:

Script profiling - lessons - defer work



improve performance of catch and release script #229

Merged 0xxon merged 1 commit into zeek:master from JustinAzoff:topic/jazoff/catch-and-release-perf on Jan 10

Conversation 0 Commits 1 Checks 0 Files changed 1

Changes from all commits File filter... Jump to... 0 / 1 files viewed

3 scripts/base/frameworks/netcontrol/catch-and-release.bro

```
@@ -428,13 +428,12 @@ function unblock_address_catch_release(a: addr, reason: string &default=""): boo
428 428
429 429   function catch_release_seen(a: addr)
430 430         {
431     -       local e = Entity($ty=ADDRESS, $ip=addr_to_subnet(a));
432     -
433 431         if ( a in blocks )
434 432                 {
435 433   @if ( ! Cluster::is_enabled() || ( Cluster::is_enabled() && Cluster::local_node_type() == Cluster::MANAGE
436 434                 local bi = blocks[a];
437 435                 local log: CatchReleaseInfo;
    436 +               local e = Entity($ty=ADDRESS, $ip=addr_to_subnet(a));
438 437
```

corelight

Catch and release was showing up in profiling

Script profiling - lessons - don't work at all

Move catch-and-release and unified2 scripts from base/ to policy/ #379

Closed **jsiwek** opened this issue on May 24 · 1 comment

jsiwek commented on May 24 Member

Simply loading these scripts incurs a performance cost due to them containing handlers for common events, and they are not useful to the most general population of users, so they should not be loaded by default

corelight

Even after fixing it was still showing up, but don't need it anyway!

Script profiling - lessons - strings can be slow

Appending to strings is slow

```
local s = "my string";
s += "more";
s += "more";
s += "even more";
```

Python optimized this a while back, but we haven't yet.

Script profiling - lessons - string_vecs are fast

Do this instead

```
local s = string_vec();
s += "my string";
s += "more";
s += "more";
s += "even more";
local res = join_string_vec(s, "");
```

+= for vectors is new :-)

Script profiling - lessons - `md5_hash_update` is faster

Are you just computing the md5?

```
local h = md5_hash_init();
md5_hash_update(h, "my string");
md5_hash_update(h, "more");
md5_hash_update(h, "more");
md5_hash_update(h, "even more");
local res = md5_hash_finish(h);
```

Do you even need the full string?

Profiling future improvements

- Read Brendan Gregg's new book and learn how to do custom profiling using ebpf and bcc-tools.
- Implement a version of **perf record** that does rolling captures without restarting
- Implement profiling tools that understand zeek stack frames

BroFunc::Call is not helpful

Links

- https://github.com/jemalloc/jemalloc/wiki/Use-Case:-Leak-Checking
- http://www.brendangregg.com/perf.html
- https://github.com/Netflix/flamescope
- https://medium.com/netflix-techblog/netflix-flamescope-a57ca19d47bb
- https://www.amazon.com/BPF-Performance-Tools-Brendan-Gregg/dp/0136554822 - BPF Performance Tools
- https://github.com/esnet/zeek-exporter
- https://github.com/JustinAzoff/zeek-jemalloc-profiling
- https://github.com/JustinAzoff/zeek_benchmarks

Links

- https://github.com/zeek/zeek/issues/289 - HTTP Multipart responses cause unbounded http.log entries
- https://github.com/zeek/zeek/issues/554 - file-magic sigs are being checked in places they shouldn't be
- https://github.com/zeek/zeek/pull/229 - improve performance of catch and release script
- https://github.com/zeek/zeek/pull/231 - Val pre-allocation
- https://github.com/zeek/zeek/pull/244 - Fix variable reuse in table expiration
- https://github.com/zeek/zeek/pull/277 - Defer initialization of Dicts and Lists
- https://github.com/zeek/zeek/pull/500 - Remove redundant buffering in contentline
- https://github.com/zeek/zeek/pull/501 - Avoid buffering all http headers
- https://github.com/zeek/zeek/issues/379 - Move catch-and-release and unified2 scripts from base/ to policy/
- https://github.com/zeek/zeek/pull/276 - Improve performance of dns policy
- https://github.com/zeek/zeek/pull/477 - Add DPD::max_violations option
- https://github.com/zeek/zeek/pull/576 - Improve reassembly worst-case performance