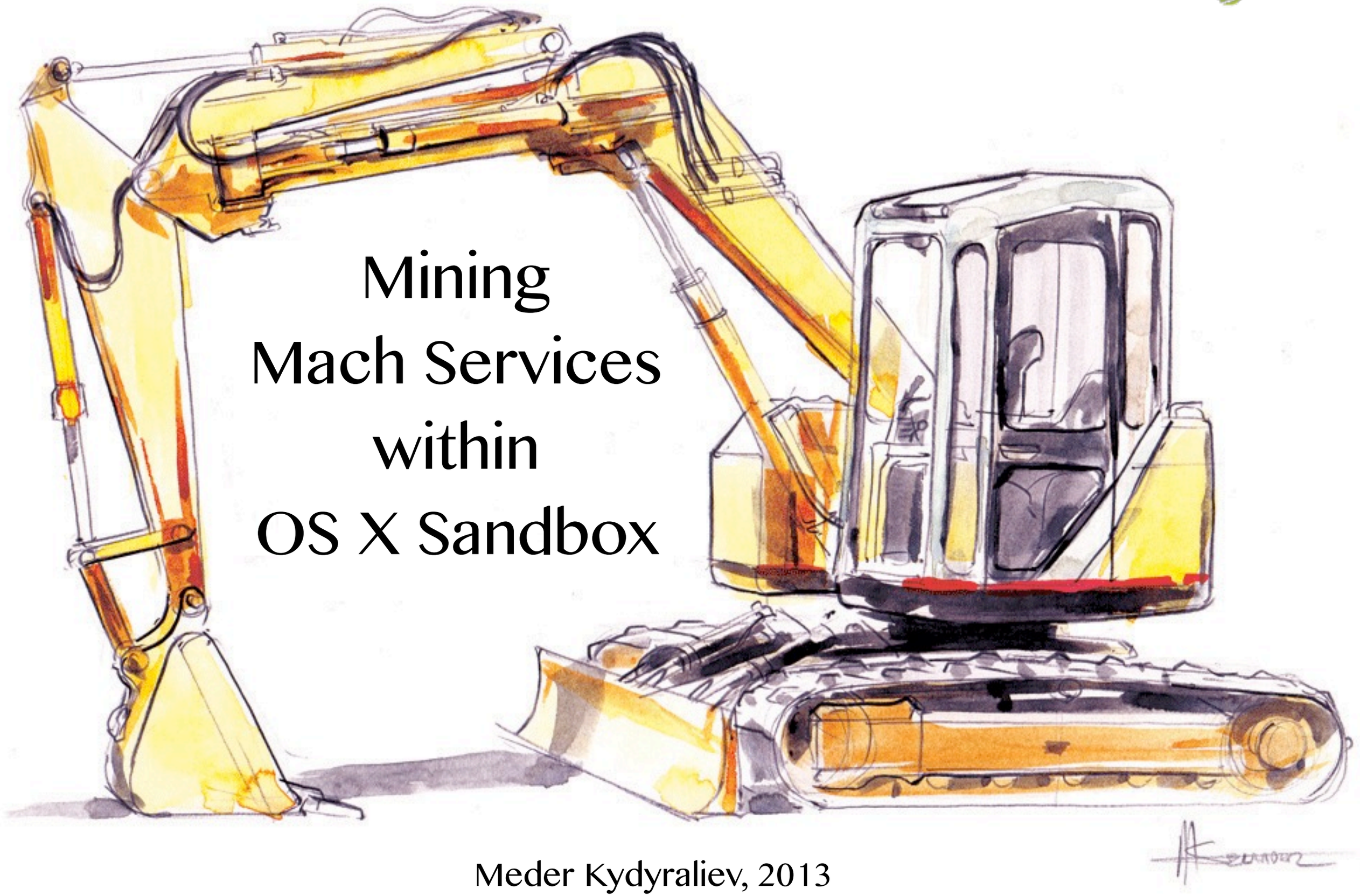
ruxcon



Mining Mach Services within OS X Sandbox

Meder Kydyraliev, 2013

Agenda

- Sandboxing: what? why? how?
- OS X Sandbox aka Seatbelt
  - Quick overview
  - Enumerating attack surface
- Mach services
  - Quick overview
  - Fuzzing!

Disclaimer

IANAR

(I Am Not A Reverser)

Sandboxing: What?

- Sandbox - a mechanism for segregation and containment of a piece of code exposed to untrusted inputs
- MAC & RBAC
- Sandbox flavors:
    - LSM: SELinux, AppArmor, TOMOYO
    - TrustedBSD MAC: Seatbelt

Sandboxing: Why?

- Good software is hard
- Fixing bugs in software security researchers used to not care about is even harder (e.g. Adobe Reader)
- Indicator of “acceptance” by software vendors

Sandboxing: How?

- It all boils down to hooks:

```
int connect_nocancel(__unused proc_t p, struct connect_nocancel_args *uap,
                     __unused int32_t *retval) {
...
#if CONFIG_MACF_SOCKET_SUBSET
    if ((error = mac_socket_check_connect(kauth_cred_get(), so, sa)) != 0) {
        if (want_free)
            FREE(sa, M_SONAME);
        goto out;
    }
#endif /* MAC_SOCKET_SUBSET */
...
}
```

Sandboxing: How?

- It all boils down to hooks:

```
int connect_nocancel(__unused proc_t p, struct connect_nocancel_args *uap,
                     __unused int32_t *retval) {
...
#if CONFIG_MACF_SOCKET_SUBSET
    if ((error = mac_socket_check_connect(kauth_cred_get(), so, sa)) != 0) {
        if (want_free)
            FREE(sa, M_SONAME);
        goto out;
    }
#endif /* MAC_SOCKET_SUBSET */
...
}
```

OS X Sandbox

- Based on TrustedBSD MAC Framework
- Prior work:
  - Iozzo, V. (2012).“A Sandbox odyssey”.
  - Blazakis, D. (2011). “The Apple Sandbox”.

TrustedBSD MAC

- A bunch of hooks are sprayed throughout the kernel
- Hooks loop over registered policy modules invoking corresponding functions (e.g. `mac_vnode_check_open`)
- Allows coexistence of multiple implementations
- Provides multiplex system call `mac_syscall()` for modules to expose functionality

Issues with TrustedBSD/MAC

- Relies on hooks to be present (missing vs. unimplemented)
- Argument parsing prior to hooks represents attack surface
- XNU extras:
  - Retrofitted for Mach (more hooks sprayed in user-land Mach services, e.g. `mach-lookup`)

mac_syscall()

`_syscall_set_profile`	applies profile to a process
`_syscall_set_profile_builtin`	applies default builtin profile to a process
`_syscall_check_sandbox`	checks specified action(e.g. mach-lookup) against policy
`_syscall_note`	associated “note” with current proc’s sandbox label
`_syscall_container`	???
`_syscall_suspend`	Suspends sandbox checks on supplied PID by setting boolean value on proc's label. PID must belong to the same user. Calling process must have `com.apple.private.security.sandbox-manager` *entitlement* and target process has to either have `com.apple.security.print` entitlement value set to `1` or `com.apple.security.temporary-exception.audio-unit-host` set to `1`.
`_syscall_unsuspend`	Resume suspended sandbox checks.
`_syscall_passthrough_access`	??? Seems to take a descriptor, get corresponding vnode and call vnode_authorize() on parent's vnode.
`_syscall_vtrace`	

mac_syscall()

`_syscall_set_profile`	applies profile to a process
`_syscall_set_profile_builtin`	applies default builtin profile to a process
`_syscall_check_sandbox`	checks specified action(e.g. mach-lookup) against policy
`_syscall_note`	associated “note” with current proc’s sandbox label
`_syscall_container`	???
`_syscall_suspend`	Suspends sandbox checks on supplied PID by setting boolean value on proc's label. PID must belong to the same user. Calling process must have `com.apple.private.security.sandbox-manager` *entitlement* and target process has to either have `com.apple.security.print` entitlement value set to `1` or `com.apple.security.temporary-exception.audio-unit-host` set to `1`.
`_syscall_unsuspend`	Resume suspended sandbox checks.
`_syscall_passthrough_access`	??? Seems to take a descriptor, get corresponding vnode and call vnode_authorize() on parent's vnode.
`_syscall_vtrace`	

mac_syscall()

_syscall_extension_issue	returns extension for a file operation or Mach lookup
_syscall_extension_consume	uses the above extension to augment current proc's policy by adding action authorized by the extension to the policy
_syscall_extension_release	disassociates "consumed" extension from the policy
_syscall_extension_update_file	???
_syscall_extension_twiddle	???

Seatbelt Extensions

Usage:

```
extension = _syscall_extension_issue("com.apple.app-sandbox.read", "/etc/passwd");

_syscall_extension_consume(extension);

FILE* f = fopen("/etc/passwd","r");
```

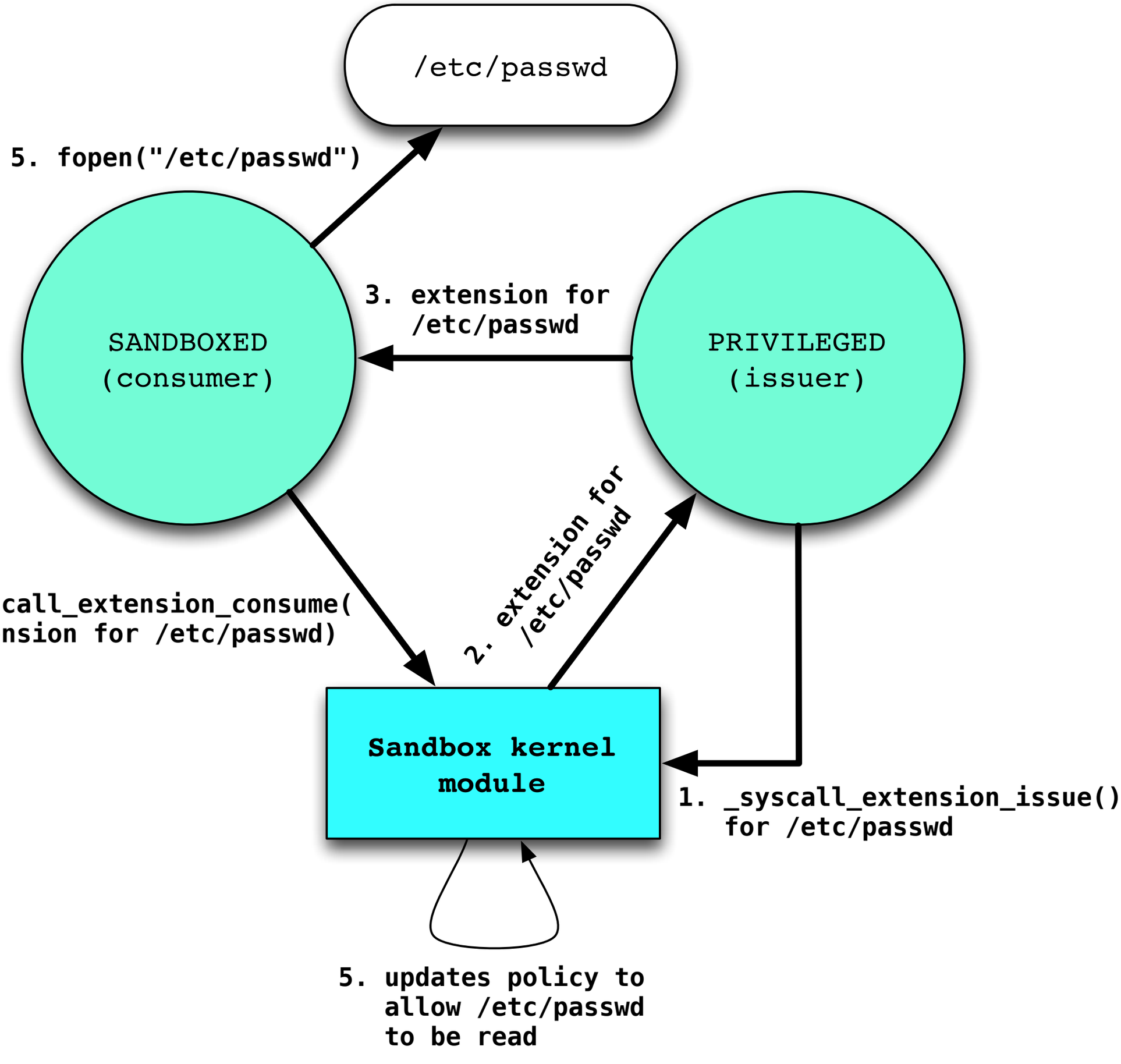
Issuer Policy:

```
(allow file-issue-extension ....
```

Consumer Policy:

```
(allow file-read* (extension "com.apple.app-sandbox.read"))

(allow mach-lookup (extension "com.apple.app-sandbox.mach"))
```


/etc/passwd
5. fopen("/etc/passwd")
SANDBOXED
(consumer)
3. extension for
/etc/passwd
PRIVILEGED
(issuer)
2. extension for
/etc/passwd
4. _syscall_extension_consume(
extension for /etc/passwd)
Sandbox kernel
module
1. _syscall_extension_issue()
for /etc/passwd
5. updates policy to
allow /etc/passwd
to be read

Seatbelt Extensions

- How to they work?

```
58ffd694274b2b5575eff5e497fe4de1de815124;00000000;000000000000001a;com.app
le.app-sandbox.read;00000001;01000004;00000000000041ee;/private/etc/passwd
```

`58ffd6942....5e497fe4de1de815124`	SHA1 HMAC value.
`0`	0
`000000000000001a`	Length of the extension type string that follows.
`com.apple.app-sandbox.read`	Extension type
`1`	1
`1000004`	File system ID (fsid_t) with type truncated.
`00000000000041ee`	inode number
`/private/etc/passwd`	file path

Seatbelt Extensions

- SHA1 HMAC key generated on startup by Sandbox.kext
- Constant time comparison

Users

- Used by Google Chrome.
- Used by Adobe Reader X(I).
- In some form or the other used by most OS X apps.

Challenges

- Complex interactions between components (server vs client apps)
- One-time sensitive resource access (e.g. config load on startup)
- Legacy apps: sandbox-aware vs. sandbox-unaware

Process Warm-up

- What?
  - exercise of code paths prior to sandbox being enabled
- Why?
  - communications channels are established
  - files are open/read/written

Enumerating Attack Surface

- BSD system calls
  - code that runs before MAC hook
  - hooks provided by MAC, but not implemented by Seatbelt
- Mach Services
  - in-kernel
  - user-land
- I/O Kit

BSD system calls

- There are a number of system call MAC hooks not implemented/allowed by Seatbelt, e.g.:
  - socket() - AF_INET/AF_LOCAL sockets of SOCK_DGRAM/ SOCK_STREAM type are allowed
  - setsockopt(), ioctl(), mmap() - unimplemented by Seatbelt
  - getfsstat(), readdir() - unimplemented and provide the fsid and inode for extensions (if you already have the key)

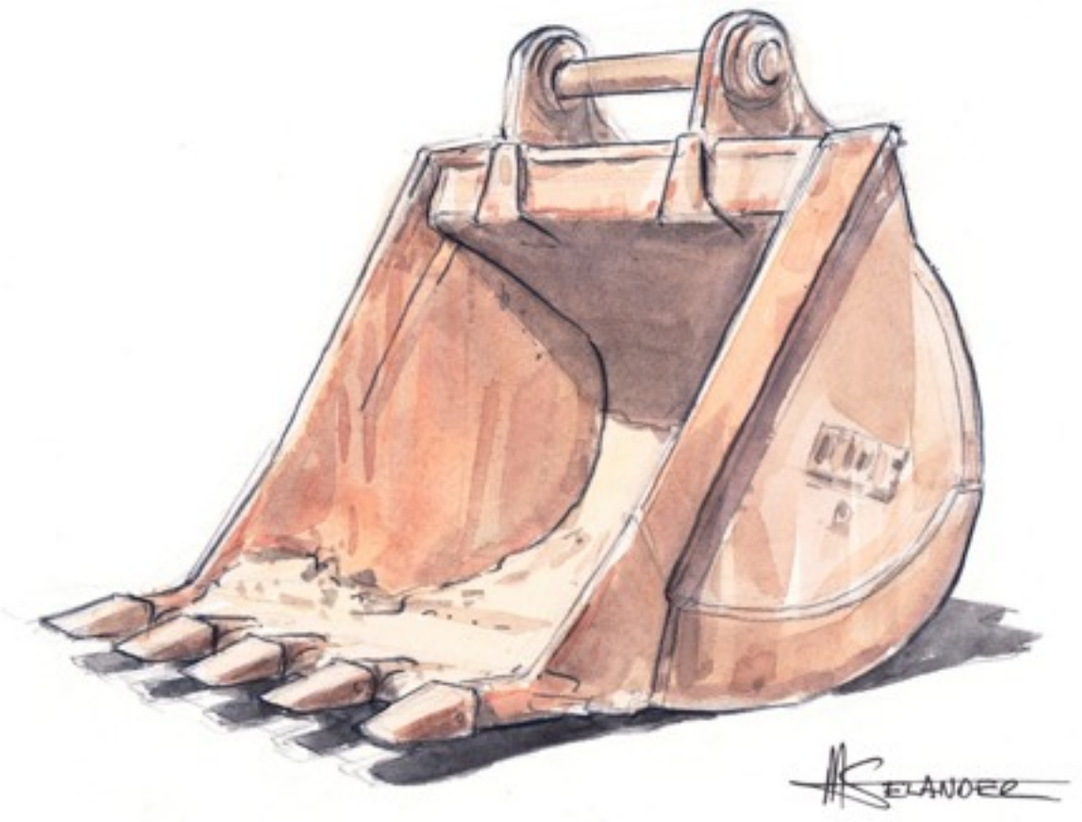
...speaking of setsockopt()

- Turns out you can set SO_EXECPATH(0x1085) to a path of a preauthorized binary to bypass firewall prompts:

  - `/usr/libexec/configd`
  - `/usr/sbin/mDNSResponder`
  - `/usr/sbin/racoon`
  - `/usr/bin/nmblookup*` - doesn't exist hence prompt is displayed
  - `/System/Library/PrivateFrameworks/Admin.framework/Versions/A/Resources/readconfig`

- Setting 0x1085 to any string without ‘/’ results in `panic()` (NULL deref)

Mach Services



Remember warm-up?

- Exercising code paths leaves some interesting artifacts...Mach ports.
- Chrome renderer:
  - policy: only `fontd`
  - reality (`showipcint` from kgmacros):
    - `coreservicesd`
    - `cfprefsd`
    - `notifyd`
    - `distnoted`

Mach Services Intro

- Mach service is essentially a queue consumer
- Mach ports represent descriptors for queues
- Send == enqueue, receive == dequeue
- Sender/receiver can be either:
  - another process(e.g. `coreservicesd`)
  - kernel (e.g. `thread_set_state` ptrace replacement)
- See also Dai Zovi's "Hacking at Mach Speed"

Mach Ports

- Port just a descriptor of a port in task's (i.e. proc's) IPC namespace
- Lots of types end up being defined as `mach_port_t` (e.g. `clock_serv_t`)
- Mach ports can be obtained by:
  - calling Mach traps (e.g. `task_self_trap`, `task_for_pid`, `mach_port_allocate`)
  - via boostrap/launchd...

Mach Lookup via launchd

```
mach_port_t bootstrap, svc_port;
task_get_bootstrap_port(mach_task_self(), &bootstrap);

bootstrap_look_up(bootstrap,
                  "com.apple.FontObjectsServer",
                  &svc_port);
...
mach_msg(...);
```

launchd

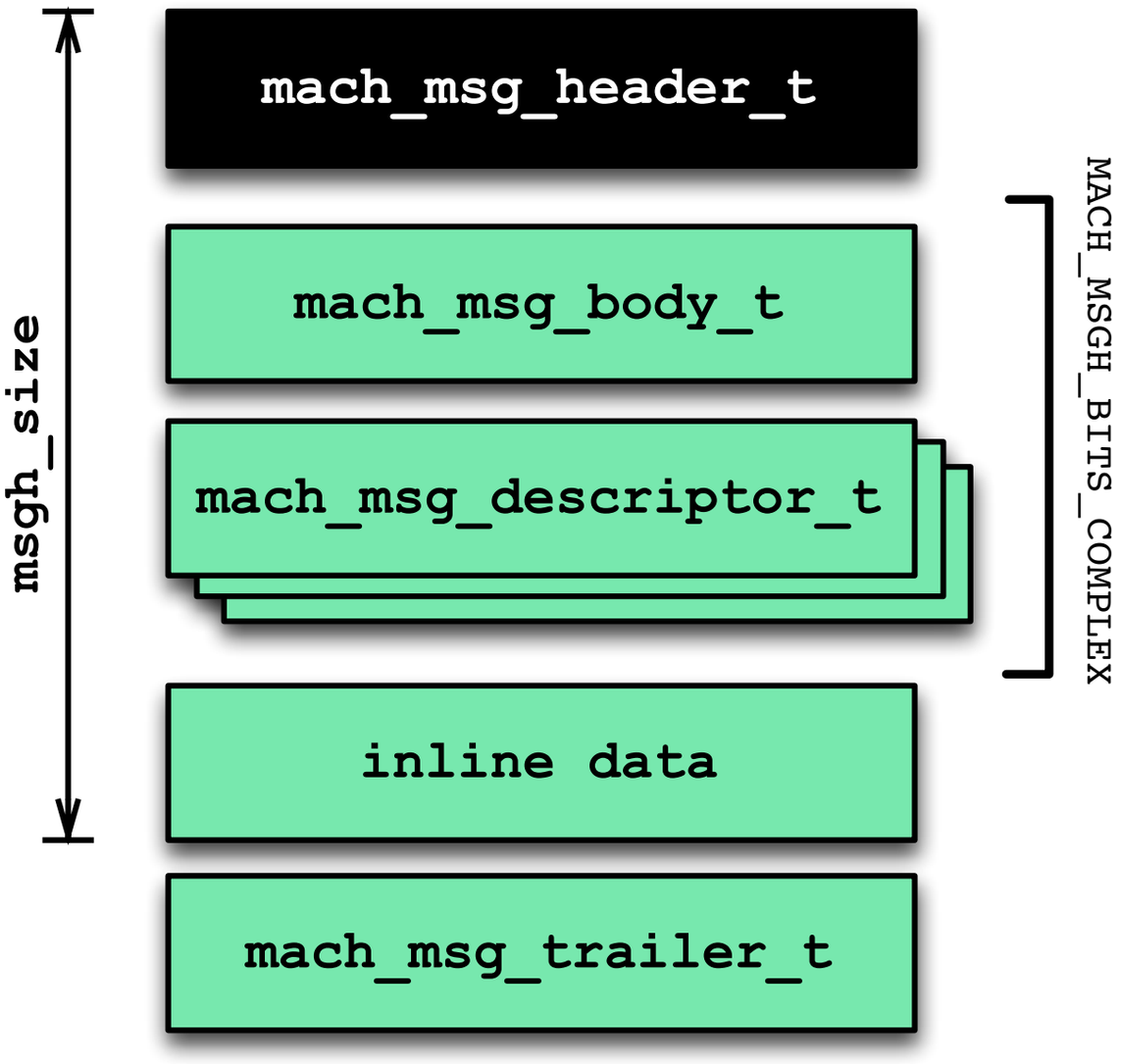
- And here's what happens in launchd:

```
job_mig_look_up2(job_t j, mach_port_t srp, name_t servicename, mach_port_t *serviceportp, pid_t
target_pid, uuid_t instance_id, uint64_t flags)
{
     ...
#endif

#if HAVE_SANDBOX
     /* We don't do sandbox checking for XPC domains because, by definition, all
      * the services within your domain should be accessible to you.
      */
     if (!xpc_req && unlikely(sandbox_check(ldc->pid, "mach-lookup", per_pid_lookup ?
               SANDBOX_FILTER_LOCAL_NAME : SANDBOX_FILTER_GLOBAL_NAME, servicename) > 0)) {
          return BOOTSTRAP_NOT_PRIVILEGED;
     }
#endif
```

Anatomy of a Mach Message

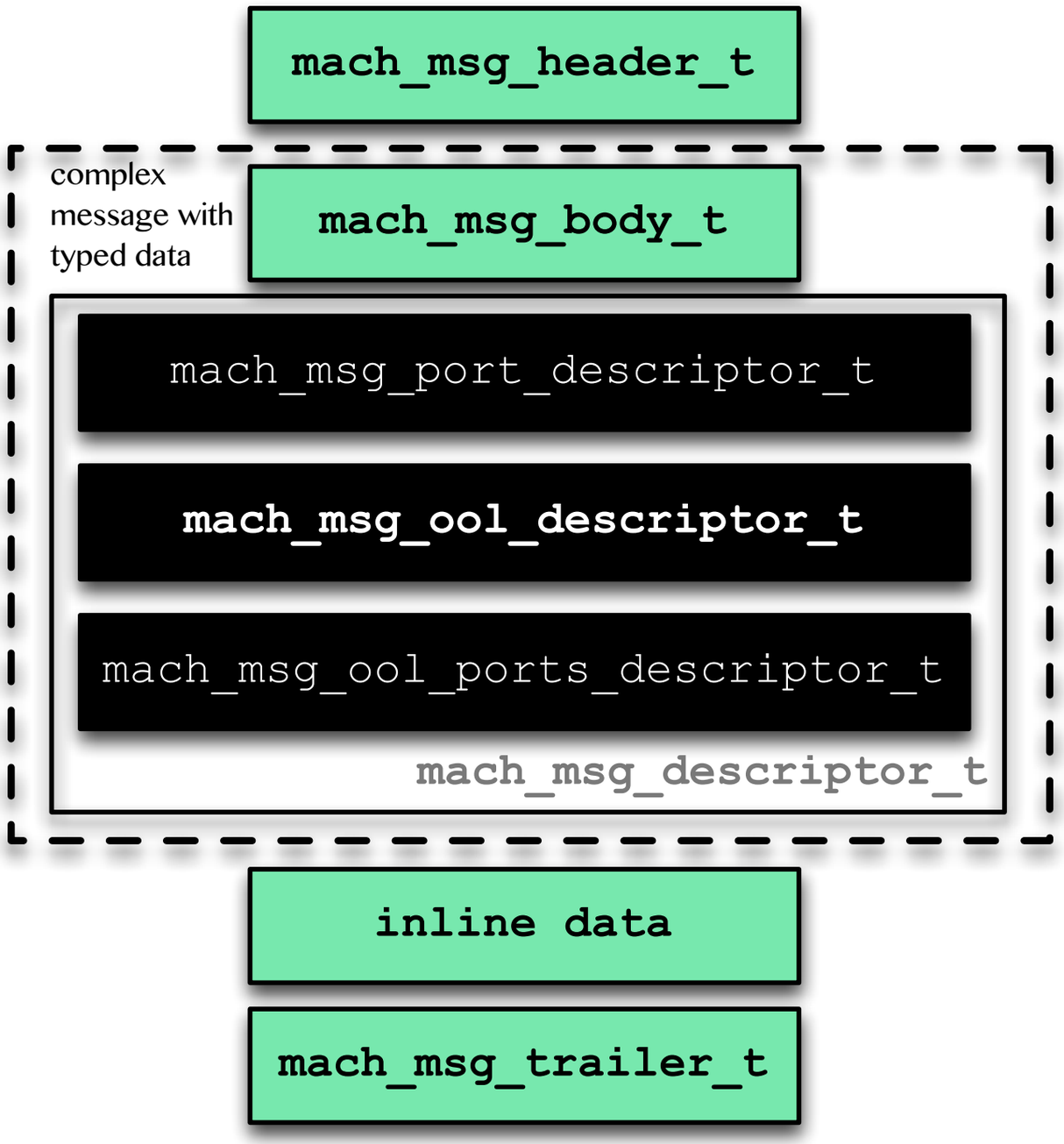
Mach Message - Header




```
typedef struct
{
  mach_msg_bits_t msgh_bits;
  mach_msg_size_t msgh_size;
  mach_port_t     msgh_remote_port;
  mach_port_t     msgh_local_port;
  mach_msg_size_t msgh_reserved;
  mach_msg_id_t   msgh_id;
} mach_msg_header_t;
```

- **msgh_bits** - determines how msgh_remote_port and msgh_local_port are handled and specifies if message is complex (MACH_MSGH_BITS_COMPLEX)
- **msgh_local_port** - reply port
- **msgh_id** – used by services to demux calls.

Mach Message - Body

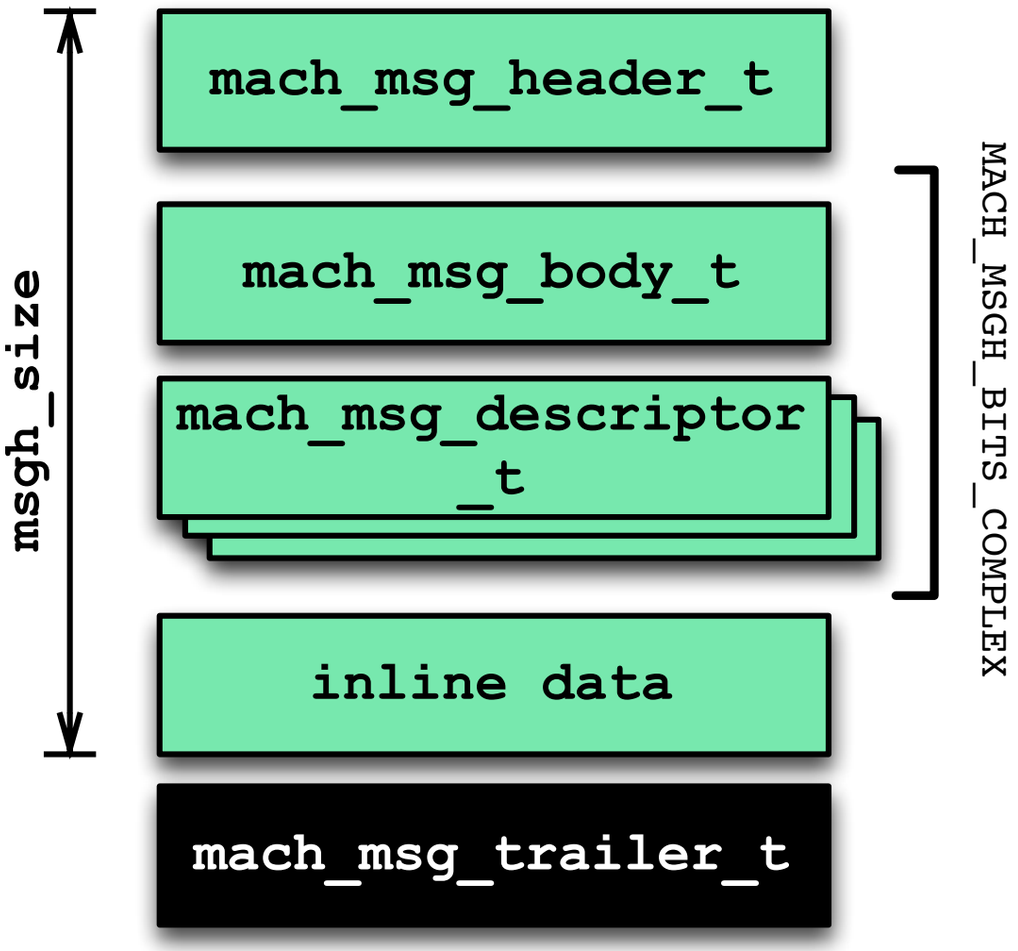



```
typedef struct
{
  mach_msg_size_t msgh_descriptor_count;
} mach_msg_body_t

typedef struct
{
  void*                          address;
  boolean_t                   deallocate: 8;
  mach_msg_copy_options_t        copy: 8;
  unsigned int                   pad1: 8;
  mach_msg_descriptor_type_t  type: 8;
  mach_msg_size_t                   size;
} mach_msg_ool_descriptor_t;
```

- **`address, size`** - pointer to out-of-line(OOL) memory and it's size
- **`copy`** - option instructing kernel on how to treat the memory (MACH_MSG_VIRTUAL_COPY or MACH_MSG_PHYSICAL_COPY)

Mach Message - Body




```
typedef struct
{
  mach_msg_trailer_type_t       msgh_trailer_type;
  mach_msg_trailer_size_t       msgh_trailer_size;
  mach_port_seqno_t             msgh_seqno;
  security_token_t              msgh_sender;
  audit_token_t                 msgh_audit;
  mach_vm_address_t             msgh_context;
  int                         msgh_ad;
  msg_labels_t                      msgh_labels;
} mach_msg_mac_trailer_t;
```

- Trailer is requested by supplying `MACH_RCV_TRAILER_TYPE(MACH_RCV_TRAILER_SENDER)` option to `mach_msg()`.
- `audit_token_to_au32()` extracts {r,e}uid/gid and pid from `audit_token_t`

Sending/Receiving

`mach_msg()/mach_msg_overwrite()` are used to both send and receive messages:

```
mach_msg_return_t   mach_msg
                    (mach_msg_header_t          msg,
                     mach_msg_option_t        option,
                     mach_msg_size_t       send_size,
                     mach_msg_size_t    receive_limit,
                     mach_port_t         receive_name,
                     mach_msg_timeout_t       timeout,
                     mach_port_t               notify);
```

- option - `MACH_SEND_MSG, MACH_RCV_MSG`

Fuzzing Mach Services

https://github.com/meder/mach-fuzz

...but before that

- Mach services generally considered less sexy:
  - local privesc
  - require extra knowledge
- Some Mach services are a b!@#$% to fuzz...

coreservicesd

- Runs as root
- Will segfault within seconds of fuzzing
  - Out-of-memory reads
  - Huge allocations
- ...which brings down lots of other stuff (must restart)
- Instead of fuzzing explored APIs exposed over Mach...

coreservicesd

- Has checks sprinkled to check if app is sandboxed
- Checks are missing/wrong in:
  - `__XSetContentTypeHandler`
  - `__XSetSchemeHandler`
  - `_XRegisterItemInfo`

coreservicesd

- `__XSetContentTypeHandler` - associates arbitrary registered bundle ID with arbitrary MIME type. Attack: associate `public.plain-text` with `com.apple.JarLauncher`

- `__XSetSchemeHandler` - associated URL schemes with arbitrary registered bundle ID. Attack: change default browser, mail agent, PDF reader.

- `_XRegisterItemInfo` - registers items (e.g. applications) with launchd. Used by mdworker to automatically register any valid .app directories on your HDD (e.g. if you unzipped something with .app and Info.plist) with launchd. Newly registered bundle ID can be used in above calls. NOTE: calls `processIsInAppSandbox()`, which returns false for Chrome.

...back to fuzzing

- Usual steps involved:
  - pick target
  - collect samples
  - fuzz

Collecting Samples

- Can't download off the internet :(
- Random generation ineffective
  - `msgh_id` - correct ID range is crucial for reaching target code:

```
mov    eax,0xffffc180;
add    eax,DWORD PTR [rdi+0x14] ; msgh_id
cmp    eax,0x21                 ; 0x3e80
jbe    process_message
xor    eax,eax
jmp    return
```

  - `msgh_size` - size is often checked right after msgh_id for expected size.

Collecting Samples

- mach_dump.py on target process
- trigger code paths (e.g. drag and drop, install stuff, visit web pages)
- uses gdb + python
    - OS X gdb 6.3.50-20050815
    - Compile + sign latest gdb for Python support(symbols are borked)
    - use both!

mach_dump.py

- Parses Mach message and saves it on disk (including OOL memory)
- Implements GDB breakpoint
- Must be set right after server-side `mach_msg()` call and given register name with mach message
- To find the right spot:

```
break mach_msg
commands
  bt
  c
end
```

Sample stacktrace

```
#0  0x00007fff8389dc0d in mach_msg ()
#1  0x00007fff8c030835 in serverMainHandler ()
#2  0x00007fff8c623e40 in __CFMachPortPerform ()
#3  0x00007fff8c623d09 in
        __CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE1_PERFORM_FUNCTION__ ()
#4  0x00007fff8c623a49 in __CFRunLoopDoSource1 ()
#5  0x00007fff8c656c02 in __CFRunLoopRun ()
#6  0x00007fff8c6560e2 in CFRunLoopRunSpecific ()
#7  0x00007fff8c664dd1 in CFRunLoopRun ()
#8  0x00007fff8c02fff7 in main_handler ()
#9  0x00007fff8cf807e1 in start ()
```

Fuzzing

- Basic fuzzer
- Parses messages saved with mach_dump.py
- Allocates ports where needed
- Can cycle through a range of msgh_ids

Results (fontd)

- Incorrectly bounded `rol` loop on stack with attacker controlled count
- Arbitrary `vm_deallocate()` on a pointer
- Over reading in `memcpy`/`memmove`
- Huge allocations

Recommendations

- Know the descriptors and ports accessible from sandboxed process
- Close unused mach ports

  OR

- Broker out mach calls (tricky!)

References

- Iozzo, V. (2012). “A Sandbox odyssey”: http://prezi.com/lxljhvzem6js/a-sandbox-odyssey-infiltrate-2012/
- Dai Zovi. (2011). “Hacking at Mach2”: http://blog.trailofbits.com/2011/01/11/hacking-at-mach-2/
- Blazakis, D. (2011). “The Apple Sandbox”: http://securityevaluators.com/files/papers/apple-sandbox.pdf
- Tinnes, Evans (2009). Security In-Depth for Linux Software: https://www.cr0.org/paper/jt-ce-sid_linux.pdf