Advanced Mac OS X Rootkits

Dino Dai Zovi

Chief Scientist

Endgame Systems

Overview

- Mac OS X and Mach
- Why use Mach for rootkits?
- User-mode Mach rootkit techniques
- Kernel Mach rootkit techniques

Why Mach Rootkits?

- Traditional Unix rootkit techniques are well understood
- Mach functionality is more obscure
- Rootkits using obscure functionality are less likely to be detected or noticed
- Mach is fun to program

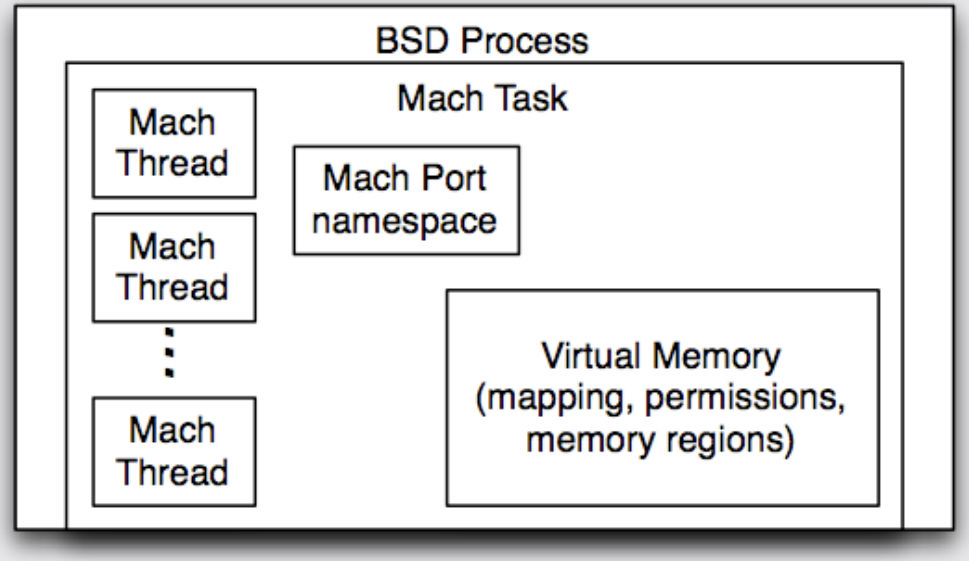
Introduction to Mach

- Mac OS X kernel (xnu) is a hybrid between Mach 3.0 and FreeBSD
  - FreeBSD kernel top-half runs on Mach kernel bottom-half
  - Multiple system call interfaces: BSD (positive numbers), Mach (negative)
  - BSD sysctls, ioctls
  - Mach in-kernel RPC servers, IOKit user clients, etc.
- Mach inter-process communication (IPC)
  - Communicates over uni-directional ports, access controlled via rights
  - Multiple tasks may hold port send rights, only one may hold receive rights

Tasks and Processes

ENDGAME SYSTEMS

- Mach Tasks own Threads, Ports, and Virtual Memory
- BSD Processes own file descriptors, etc.
- BSD Processes <=> Mach Task
  - task_for_pid(), pid_for_task()
- POSIX Thread != Mach Thread
  - Library functions use TLS



Mach Task/Thread System Calls

- task_create(parent_task, ledgers, ledgers_count, inherit_memory, *child_task)
- thread_create(parent_task, *child_activation)
- vm_allocate(task, *address, size, flags)
- vm_deallocate(task, address, size)
- vm_read(task, address, size, *data)
- vm_write(task, address, data, data_count)

User-mode Mach Rootkits

- Not as “sexy” as kernel mode rootkits
- Can be just as effective and harder to detect
- Are typically application/process -specific
- Based on thread injection or executable infection
- Would you notice an extra bundle and thread in your web browser?

Injecting Mach Threads

- Get access to another task's task port
  - task_for_pid() or by exploiting a local privilege escalation vulnerability
- Allocate memory in remote process for thread stack and code trampoline
- Create new mach thread in remote process
  - Execute trampoline with previously allocated thread stack segment
  - Trampoline code promotes Mach Thread to POSIX Thread
    - Call _pthread_set_self(pthread_t) and cthread_set_self(pthread_t)

Mach Exceptions

- Tasks and Threads generate exceptions on memory errors
- Another thread (possibly in another task) may register as the exception handler for another thread or task
- Exception handling process:
  1. A Thread causes a runtime error, generates an exception
  2. Exception is delivered to thread exception handler (if exists)
  3. Exception is delivered to task's exception handler (if exists)
  4. Exception converted to Unix signal and delivered to BSD Process

Injecting Mach Bundles

- Inject threads to call functions in the remote process
  - Remote thread calls injected trampoline code and then target function
  - Function returns to chosen bad address, generates an exception
  - Injector handles exception, retrieves function return value
- Call dlopen(), dlsym(), dlclose() to load bundle from disk
- Inject memory, call NSCreateObjectFileImageFromMemory(), NSLinkModule()
- Injected bundle can hook library functions, Objective-C methods

inject-bundle

- inject-bundle
  - Inject a bundle from disk into a running process
  - Usage: inject_bundle path_to_bundle [pid]
- Sample bundles
  - test: Print output on load/run/unload
  - isight: Take a picture using iSight camera
  - sslspy: Log SSL traffic sent through SecureTransport
  - ichat: Log IMs from within iChat

Hooking and Swizzling

- Hooking C functions is basically the same as on any other platform
  - see Rentzsch's mach_override
- Objective-C runtime has hooking built-in:
  - method_exchangeImplementations()
  - or just switch the method pointers manually
  - all due to Obj-C's dynamic runtime
  - use JRSwizzle for portability

DEMO

Rootkitting the Web Browser

- What client system doesn't have the web browser open at all times?
- Will be allowed to connect to *:80 and *:443 by host-based firewalls (i.e. Little Snitch)
- Background thread can poll a known site for command and control instructions or look for instructions in HTML content from any site
- Injected bundles do not invalidate dynamic code signatures (used by Keychain, etc)

Kernel Mach Rootkits

- Mach system calls allow Mach RPC to in-kernel servers which perform task, thread, and VM operations
- RPC routines are stored in the `mig_buckets` hash table by subsystem id + subroutine id
- Analogous to `sysent` table for Unix system calls
- Incoming Mach messages sent to a kernel-owned port are dispatched through `mig_buckets`
- We can interpose on these function calls or inject new RPC servers by modifying this hash table

Example: inject_subsystem


```
int inject_subsystem(const struct mig_subsystem * mig)
{
        mach_msg_id_t h, i, r;
        // Insert each subroutine into mig_buckets hash table
        for (i = mig->start; i < mig->end; i++) {
                mig_hash_t* bucket;
                h = MIG_HASH(i);
                do { bucket = &mig_buckets[h % MAX_MIG_ENTRIES];
                } while (mig_buckets[h++ % MAX_MIG_ENTRIES].num != 0 &&
                                h < MIG_HASH(i) + MAX_MIG_ENTRIES);
                if (bucket->num == 0) {     // We found a free spot
                        r = mig->start - i;
                        bucket->num = i;
                        bucket->routine = mig->routine[r].stub_routine;
                        if (mig->routine[r].max_reply_msg)
                                bucket->size = mig->routine[r].max_reply_msg;
                        else
                                bucket->size = mig->maxsize;
                        return 0;
                }
        }
        return -1;
}
```

Mach Kernel RPC servers

- In-kernel Mach RPC subsystems are enumerated in the `mig_e` table and interfaces are in /usr/include/mach/subsystem.defs
  - mach_vm, mach_port, mach_host, host_priv, host_security, clock, clock_priv, processor, processor_set, is_iokit, memory_object_name, lock_set, ledger, semaphore, task, thread_act, vm_map, UNDReply, default_pager_object, security

Machiavelli

- Mach RPC provides high-level remote control
  - –vm_alloc(), vm_write(), thread_create() on kernel or any task
- Want to still use MiG generated client RPC stubs
- Machiavelli Proxy runs as background thread in control utilities on attacker's system
- Machiavelli Agents run on the remote compromised host as user-mode process or in kernel

NetMessage and NetName servers

- Network transparency of IPC was a design goal
- Old Mach releases included the NetMessage Server
  - Mach servers could register themselves on the local NetName server
  - Clients could lookup named servers on remote hosts
  - Local NetMessage server would act as a proxy, transmitting Mach IPC messages over the network
- These features no longer exist in Mac OS X

Machiavelli Architecture

- Machiavelli Proxy
  - Runs as background thread of a Machiavelli utility
  - Receives messages on proxy ports and sends to remote Agent
  - Replaces port names in messages received from Agent with proxy ports
- Machiavelli Agent
  - Receives messages over network from Proxy, sends to real destination
  - Receives and transmits reply message if a reply is expected
- Machiavelli Utilities
  - Run on control host, use Proxy to control compromised host

Mach messages

- Mach messages are structured and unidirectional
- 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;
```

- Body consists of typed data items

Complex Mach Messages

- “Complex” Mach messages contain out-of-line data and may transfer port rights and/or memory pages to other tasks
- In the message body, descriptors describe the port rights and memory pages to be transferred
- Kernel grants port rights to the receiving process
- Kernel maps transferred pages to receiving process, sometimes at message-specified address

Proxying Mach Messages

- Proxy maintains a Mach port set
  - A port set has the same interface as a single port and can be used identically in mach_msg()
  - Each proxy port in the set corresponds to the real destination port name in the remote Agent
  - Port names can be arbitrary 32-bit values, so port set names are pointers to real destination port name values
- Received messages must be translated (local <=> remote ports and descriptor bits)
- Messages are serialized to byte buffers and then sent

Serializing Mach Messages

- Serializing “simple” messages is simple as they don’t contain any out-of-line data
- Out-of-line data is appended to the serialized buffer in order of the descriptors in the body
- Port names are translated during deserialization
  - Translating to an intermediate “virtual port name” might be cleaner

Deserializing Mach Messages

- Port names in the mach message must be replaced with local port names
- On Agent, this is done to receive the reply
- On Proxy, this is done to replace transferred port names with proxy port names
  - Ensures that only the initial port must be manually obtained from the proxy, the rest are handled automatically
- OOL memory is mapped+copied into address space

Machiavelli example


```
int main(int argc, char* argv[])
{
    kern_return_t kr;
    mach_port_t port;
    vm_size_t page_size;

    machiavelli_t m = machiavelli_init();
    machiavelli_connect_tcp(m, "192.168.13.37", "31337");
    port = machiavelli_get_port(m, HOST_PORT);

    if ((kr = _host_page_size(port, &page_size)) != KERN_SUCCESS) {
        errx(EXIT_FAILURE, "_host_page_size: %s", mach_error_string(kr));
    }

    printf("Host page size: %d\n", page_size);

    return 0;
}
```

DEMO

Miscellaneous Agent services

- Agent must provide initial Mach ports:
  - host port
  - task_for_pid() (if pid == 0 => returns kernel task port)
- As OS X is a Mach/Unix hybrid, just controlling Mach is not enough
  - i.e. How to list processes?
- Instead of implementing Unix functionality in Agent, inject Mach RPC server code into pid 1 (launchd)

Network Kernel Extensions (NKEs)

- NKEs can extend or modify kernel networking functionality via:
  - Socket filters
  - IP filters
  - Interface filters
  - Network interfaces
  - Protocol plumbers

Conclusion

- Mach is a whole lot of fun
- Mach IPC can be made network transparent and provides a good abstraction for remote host control
- I wish my desktop was as secure as my iPhone
- For updated slides and tools go to:
  - http://trailofbits.com/