Inconsistent Locking in kcm Leads to Use-after-Free and Other Issues

Gabe Kirkpatrick - g@be-k.biz

Summary

Multiple issues exist in kcm due to missing locks around uses of a global list. This can lead to use-after-free, which can manifest in multiple ways including double-freeing of an allocation, copying memory from a freed allocation back to a client, and potentially other issues.

Tested on the following software version:

```
ProductName: macOS
ProductVersion:  11.1
BuildVersion:    20C69
```

PoC Details

This PoC works by sending sending the server two of the same `KCM_OP_RELEASE_CRED` messages from two different threads. When the timing works out correctly, this will result in the server calling `free_cred` twice on the same object, leading to a double-free. The PoC will continuously do this in a loop, typically causing kcm to crash multiple times.

Repro Steps:
Build and run the PoC by running the following from the `submission2` folder:

```
make
./kcm_uaf
```

This should cause the kcm process to repeatedly crash with the following information and stack:

```
Application Specific Information:
dyld3 mode
abort() called
kcm(14992,0x70000a66d000) malloc: Double free of object 0x7ffd6d606820

Thread 2 Crashed:: Dispatch queue: com.apple.root.default-qos
0   libsystem_kernel.dylib           0x00007fff2033f462 __pthread_kill + 10
1   libsystem_pthread.dylib          0x00007fff2036d610 pthread_kill + 263
2   libsystem_c.dylib                0x00007fff202c0720 abort + 120
3   libsystem_malloc.dylib           0x00007fff201a1430 malloc_vreport + 548
4   libsystem_malloc.dylib           0x00007fff201b5702 malloc_zone_error + 183
5   kcm                              0x0000000106304009 free_cred + 25
6   kcm                              0x000000010630373e kcm_op_release_cred + 318
7   kcm                              0x00000001062feabe kcm_dispatch + 430
8   kcm                              0x00000001062fcf59 kcm_service + 553
9   kcm                              0x0000000106305519 __mheim_do_call_block_invoke + 89
10  libdispatch.dylib                0x00007fff201c15dd _dispatch_call_block_and_release + 12
11  libdispatch.dylib                0x00007fff201c27c7 _dispatch_client_callout + 8
12  libdispatch.dylib                0x00007fff201d19b5 _dispatch_root_queue_drain + 676
```

```
13  libdispatch.dylib                0x00007fff201d1fb8 _dispatch_worker_thread2 + 92
14  libsystem_pthread.dylib          0x00007fff2036a453 _pthread_wqthread + 244
15  libsystem_pthread.dylib          0x00007fff20369467 start_wqthread + 15
```

Bug Details

kcm accepts messages on a mach port "`org.h5l.kcm`". Many of these messages deal with performing operations on credential objects, which are stored in a global list (`ntlm_head`). When accessing these items, accesses should only be performed while holding the `cred_mutex` lock. An example of this done correctly can be seen below:

```
static krb5_error_code
kcm_op_del_cred(krb5_context context,
        kcm_client *client,
        kcm_operation opcode,
        krb5_storage *request,
        krb5_storage *response)
{
    struct kcm_ntlm_cred **cp, *c;
    kcmuuid_t uuid;
    ssize_t sret;

    KCM_LOG_REQUEST(context, client, opcode);

    sret = krb5_storage_read(request, &uuid, sizeof(uuid));
    if (sret != sizeof(uuid)) {
        krb5_clear_error_message(context);
        return KRB5_CC_IO;
    }

    HEIMDAL_MUTEX_lock(&cred_mutex); <- Acquiring before use

    for (cp = &ntlm_head; *cp != NULL; cp = &(*cp)->next) {
    if ((*cp)->type == KCM_NTLM_CRED &&
        memcmp((*cp)->uuid, uuid, sizeof(uuid)) == 0 &&
        kcm_is_same_session(client, (*cp)->uid, (*cp)->session))
        {
            c = *cp;
            *cp = c->next;

            free_cred(c);
            kcm_data_changed = 1;
            break;
        }
```

```
    }

    HEIMDAL_MUTEX_unlock(&cred_mutex); <- Releasing after use

    return 0;
}
```

These calls to `HEIMDAL_MUTEX_lock(&cred_mutex)` & `HEIMDAL_MUTEX_unlock(&cred_mutex)` are missing in a number of places, leading to potential race conditions due to modification of list elements or the list itself simultaneously on multiple threads.

In the PoC, a crash is triggered by calling the `kcm_op_release_cred` function. The function with annotations of the missing lock and it's impacts can be seen below:

```
static krb5_error_code
kcm_op_release_cred(krb5_context context,
            kcm_client *client,
            kcm_operation opcode,
            krb5_storage *request,
            krb5_storage *response)
{
    struct kcm_ntlm_cred **cp;
    kcmuuid_t uuid;
    ssize_t sret;

    KCM_LOG_REQUEST(context, client, opcode);

    sret = krb5_storage_read(request, &uuid, sizeof(uuid));
    if (sret != sizeof(uuid)) {
        krb5_clear_error_message(context);
        return KRB5_CC_IO;
    }

    // Accessing ntlm_head list without acquiring lock
    for (cp = &ntlm_head; *cp != NULL; cp = &(*cp)->next) {
        struct kcm_ntlm_cred *c = *cp;

        if (!kcm_is_same_session(client, c->uid, c->session))
            continue;

        if (memcmp(uuid, c->uuid, sizeof(uuid)) == 0) {
            c->refcount--; // Modifying object without lock
            if (c->refcount < 1) {
                *cp = c->next;
                free_cred(c); // free_cred can be called twice
```

```
            }
            kcm_data_changed = 1;
            return 0;
        }
    }
    return 0;
}
```

To eliminate the issue in the code above, locking using `cred_mutex` should be added around the for loop.

Below are all the message handlers I noticed which lack appropriate locking when handling the `ntlm_head` list:
`kcm_op_get_scram_user_list`
`kcm_op_retain_cred`
`kcm_op_release_cred`
`kcm_op_cred_label_get`

Impact

Due to the number of different operations it is possible the perform on the list and its elements without locking, I would consider this vulnerability very exploitable.

As an additional exploitation factor, I have found a bug which allows arbitrary allocation of heap buffers which are not freed, that can be used as a heap spray.

kcm runs as root, and the mach port on which it receives these messages is accessible from many privilege levels, including for example the WebKit networking process. kcm itself also houses sensitive user data, specifically credentials used for authentication. For this reason I believe this issue qualifies as a sandbox escape, and would like it to be considered for eligibility under the “Unauthorized access to sensitive data” section of the Apple Security Bounty.