Uninitialized Memory Use in kcm

Gabe Kirkpatrick - g@be-k.biz

Summary

An issue exists in kcm due to insufficient validation of return values, that results in uninitialized stack memory being used. This can result in free being called on an invalid address, as well as potentially causing an integer overflow, leading to be buffer overflow.

Tested on the following software version:

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

PoC Details

This PoC works by sending a malformed message to kcm. This malformed message will result in uninitialized memory being used as a pointer, and passed to free.

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

```
make
./kcm_poc
```

This should cause the distnoted process to repeatedly crash with the following stack:

```
Thread 2 Crashed:: Dispatch queue: com.apple.root.default-qos
0   libsystem_kernel.dylib            0x00007fff2033e462 __pthread_kill + 10
1   libsystem_pthread.dylib           0x00007fff2036c610 pthread_kill + 263
2   libsystem_c.dylib                 0x00007fff202bf720 abort + 120
3   libsystem_malloc.dylib            0x00007fff201a0430 malloc_vreport + 548
4   libsystem_malloc.dylib            0x00007fff201a34c8 malloc_report + 151
5   com.apple.Heimdal                 0x00007fff2b6b0768 krb5_data_free + 40
6   kcm                               0x00000001070c4bca kcm_op_cred_label_set + 506
7   kcm                               0x00000001070bfabe kcm_dispatch + 430
8   kcm                               0x00000001070bdf59 kcm_service + 553
9   kcm                               0x00000001070c6519 __mheim_do_call_block_invoke + 89
10  libdispatch.dylib                 0x00007fff201c05dd _dispatch_call_block_and_release + 12
11  libdispatch.dylib                 0x00007fff201c17c7 _dispatch_client_callout + 8
12  libdispatch.dylib                 0x00007fff201d09b5 _dispatch_root_queue_drain + 676
13  libdispatch.dylib                 0x00007fff201d0fb8 _dispatch_worker_thread2 + 92
14  libsystem_pthread.dylib           0x00007fff20369453 _pthread_wqthread + 244
15  libsystem_pthread.dylib           0x00007fff20368467 start_wqthread + 15
```

Bug Details

kcm accepts messages on a mach port "org.h5l.kcm". When parsing these messages, the return values of the functions which read from the messages must be validated. An example of this being done properly can be seen below:

```
static krb5_error_code
kcm_op_add_ntlm_cred(krb5_context context,
             kcm_client *client,
             kcm_operation opcode,
             krb5_storage *request,
             krb5_storage *response)
{
    struct kcm_ntlm_cred *cred, *c;
    krb5_error_code ret;

    cred = create_cred(KCM_NTLM_CRED);
    if (cred == NULL)
    return ENOMEM;

    ret = krb5_ret_stringz(request, &cred->user);
    if (ret) <-- Return value checked properly
    goto error;

    ret = krb5_ret_stringz(request, &cred->domain);
    if (ret) <-- Return value checked properly
    goto error;

    ret = krb5_ret_data(request, &cred->nthash);
    if (ret) <-- Return value checked properly
    goto error;
```

In the above code, all calls beginning in `krb5_ret` read data from the client provided message, and return an error code indicating the success of the call. If there is an error after attempting to read one of these values, the function immediately exits by going to the error label.

An issue exists in the `kcm_op_cred_label_set` function due to failing to check to return values when calling `krb5_ret_stringz` and `krb5_ret_data`. If processing a message that does not contain a string or data block when these functions are called, the destination parameter passed to them will remain uninitialized. The code can be seen below, with the missing checks and use of uninitialized memory annotated:

```
static krb5_error_code
kcm_op_cred_label_set(krb5_context context,
             kcm_client *client,
             kcm_operation opcode,
             krb5_storage *request,
             krb5_storage *response)
{
    struct kcm_ntlm_cred *c;
    kcmuuid_t uuid;
    krb5_data data;
    char *label = NULL;
```

```
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;
}

krb5_ret_stringz(request, &label); <-- Return value not checked
krb5_ret_data(request, &data); <-- Return value not checked

HEIMDAL_MUTEX_lock(&cred_mutex);

for (c = ntlm_head; c != NULL; c = c->next) {

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

if (memcmp(uuid, c->uuid, sizeof(uuid)) == 0) {
    heim_string_t s;

    // Uses of potentially uninitialized data
    s = heim_string_create(label);

    if (data.length) {
    heim_data_t d;

    d = heim_data_create(data.data, data.length);

    heim_dict_set_value(c->labels, s, d);
    heim_release(d);
    } else {
    heim_dict_delete_key(c->labels, s);
    }
    kcm_data_changed = 1;
    heim_release(s);
    break;
}
}

HEIMDAL_MUTEX_unlock(&cred_mutex);

krb5_data_free(&data);<--Free of potentially uninitialized pointer
free(label); <--Free of potentially uninitialized pointer

if (c == NULL)
return ENOENT;
```

```
    return 0;
}
```

In the PoC provided, kcm will crash when calling `krb5_data_free`, which then calls malloc. The uninitialized data is also usable in the body of the loop, with `heim_data_create` even containing a malloc call using an uninitialized size value added with a constant, causing the potential for an integer overflow, leading to a buffer overflow.

This bug should be relatively easy to fix, by adding checks for the return values of `krb5_ret_stringz` and `krb5_ret_data` in the function shown above.

Impact

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.