The PLUGIN_xxx macros don't initialize all members of the plugin_feature_t structure (only the 'relevant' members are initialized). However, in plugin-feature.c a hash is calculated on all members of the plugin_feature_t (plugin_feature_hash()). This forces an assumption that two different plugin_feature_t objects with the same initialization (PLUGIN_xx(...)) will have the same hash.

The reason this works is because the rest of the members are initialized to 0 if the plugin_feature_t is a static variable.

This is almost always true, except for in libcharon/daemon.c where an automatic instance of plugin_feature_t is defined.

Attached is a patch to fix this. It is not elegant, and perhaps a different method should be preferred (for instance, perhaps plugin_feature_hash() should be modified to hash only relevant fields).

Related issues:
Related to Bug #1503: Incorrect reference counting of policies depending on c... Closed 09.06.2016

Associated revisions
Revision 27b3358f - 20.03.2014 13:42 - Tobias Brunner
plugin-feature: Hash only the actually used feature argument

Clang does not initialize padding in union members so hashing the complete "arg" union could lead to different hashes if the hashed plugin_feature_t does not have static storage duration.

Fixes #549.

History
#1 - 18.03.2014 12:04 - Tobias Brunner
- Status changed from New to Feedback
- Assignee set to Tobias Brunner

The PLUGIN_xxx macros don't initialize all members of the plugin_feature_t structure (only the 'relevant' members are initialized).

We use this kind of initialization all over the place (mostly via the INIT macro, which makes use of compound literals). The reason this works is sentence 1677 in section 6.7.8 of the C99 standard. Basically, any struct member that is not listed explicitly in the designator is initialized to zero.

Do you have any indication this is not the case with a specific compiler or on a specific platform?

#2 - 19.03.2014 16:56 - Evgeniy Stepanov
When this code is built with Clang, parts of plugin_feature_t are left uninitialized.

You've got a union with fields of varying size. Clang does not clear padding bytes when a shorter field is initialized. I believe it is aligned with the standard, because reading from a field of a union other the last assigned one is UB, so you should never be able to observe this uninitialized padding.

Find a relevant compiler issue by this link: http://llvm.org/bugs/show_bug.cgi?id=14352

struct S {
    int a;
    int b;
    union {
}
struct {
    void * p;
    void * q;
} a;

int b;
}
c;

void g(struct S*);

void f(int x, int y) {
    struct S s = (struct S){x, x, {.b = y}};
    g(&s);
}

f():
  0: 48 83 ec 18          sub  $0x18,%rsp
  4: 89 3c 24            mov  %edi,(%rsp)
  7: 89 7c 24 04         mov  %edi,0x4(%rsp)
 b: 89 74 24 08         mov  %esi,0x8(%rsp)
 f: 48 8d 3c 24     lea   (%rsp),%rdi
13: e8 00 00 00 00     callq 18 <f+0x18>
14: R_X86_64_PC32  g-0x4
18: 48 83 c4 18        add  $0x18,%rsp
1c: c3                 retq

#3 - 20.03.2014 09:14 - Tobias Brunner
- Tracker changed from Issue to Bug
- Status changed from Feedback to Assigned
- Target version set to 5.1.3

Thanks Evgeniy for the additional information. That's an interesting difference between the two compilers. And thinking about it I can see how not initialising the padding makes totally sense, and doesn't usually matter. I also don't think we rely on it other than in this particular case.

I think the best fix for this is to only hash the actually used members in plugin_feature_hash(). Otherwise, we'd have to rely on the users of this struct and macros to know that they have to define it statically. I will implement this later today.

#4 - 20.03.2014 14:09 - Tobias Brunner
- Status changed from Assigned to Closed
- Resolution set to Fixed

#5 - 09.06.2016 11:51 - Tobias Brunner
- Related to Bug #1503: Incorrect reference counting of policies depending on compiler/platform added

Files

daemon.patch  1.19 KB  18.03.2014  Noam Lampert