### Suricata - Bug #3705

**VarNameStoreLookupById: Assertion `!(current == ((void *)0))'**

<table>
<thead>
<tr>
<th>Status:</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td></td>
</tr>
<tr>
<td>Category:</td>
<td></td>
</tr>
<tr>
<td>Target version:</td>
<td>5.0.3</td>
</tr>
<tr>
<td>Affected Versions:</td>
<td>5.0.3</td>
</tr>
<tr>
<td>Effort:</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

We're trying to use the multi-tenant support in Suricata. In our test setup we were not having any problems, but trying to load a multi-tenant configuration (both device and vlan based) on a system that is monitoring a relatively busy link we get this error (often within a few minutes):

```
[May 18 14:48:38] 18/5/2020 -- 14:48:38 - <Notice> - Signature(s) loaded, Detect thread(s) activated.
```

It seems related to [https://redmine.openinfosecfoundation.org/issues/2518](https://redmine.openinfosecfoundation.org/issues/2518) only we're not trying to perform a reload. We've tried both vlan and device based multi-tenant configurations. We tried setting "loaders" to 1 and 3. We tried "default" on and off. But this always happens.

As soon as we disable multi-select everything runs fine.

**History**

**#1 - 05/19/2020 09:14 AM - Justin Ossevoort**

I've run Suricata in gdb with a breakpoint on the `g_varnamestore_current_sc` global variable

```
(gdb) watch g_varnamestore_current_sc Atomic__
Hardware breakpoint 1: g_varnamestore_current_sc_ATOMIC__
```

```
(gdb) run
Starting program: /usr/bin/suricata --af-packet=mon3
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
19/5/2020 -- 09:00:37 - <Notice> - This is Suricata version 5.0.3 RELEASE running in SYSTEM mode
[New Thread 0x7fffe069700 (LWP 683)]
```

Thread 1 "Suricata-Main" hit Hardware breakpoint 1: g_varnamestore_current_sc.Atomic__

Old value = (VarNameStore *) 0x0
New value = (VarNameStore *) 0x7fffe804f6c0
0x00000000055555555d05bf in VarNameStoreActivateStaging () at util-var-name.c:348
348 in util-var-name.c
```

```
(gdb) c
Continuing.
[New Thread 0x7fffdcd1f700 (LWP 684)]
[New Thread 0x7fffc7ff700 (LWP 685)]
[New Thread 0x7fffc7ff700 (LWP 686)]
[New Thread 0x7fffc6ff700 (LWP 687)]
[New Thread 0x7fffc6ff700 (LWP 688)]
[New Thread 0x7fffc67ff700 (LWP 689)]
[New Thread 0x7fffc5ff700 (LWP 690)]
[New Thread 0x7fffc4ff700 (LWP 691)]
[New Thread 0x7fffc4ff700 (LWP 692)]
19/5/2020 -- 09:02:44 - <Notice> - all 4 packet processing threads, 4 management threads initialized, engine started.
19/5/2020 -- 09:02:44 - <Notice> - rule reload starting
```

Thread 1 "Suricata-Main" hit Hardware breakpoint 1: g_varnamestore_current_sc.Atomic__

Old value = (VarNameStore *) 0x7fffe804f6c0
New value = (VarNameStore *) 0x0
I'm not claiming any fundamental understanding of the codebase, but it seems the following is happening:

- Suricata starts up and instantiates a `DetectEngineCtx`, which in turn initializes the "VarNameStore"
- After initial startup it triggers a load reload (probably so that loading and reloading use the same codepaths)
- In `DetectEngineReload` a new `DetectEngineCtx` is created (which doesn't initialize a new "VarNameStore" as one is already present)
- It swaps the old and new context in `DetectEngineReloadThreads`
- Afterwards it triggers the cleanup of the old `DetectEngineCtx` using `DetectEngineCtxFree` in `DetectEnginePruneFreeList`, which in turn calls `VarNameStoreFree` which cleans up the current "VarNameStore"

Now we have new instances of the `DetectEngineCtx` running without a current "VarNameStore", which leads to us triggering the assertion.

My first guess would be that we need either a refcount for the "VarNameStore" to keep it alive across reloads or we need to perform a swap/handover during reload and a flag in the `DetectEngineCtx` that indicates if it owns the "VarNameStore".

Now have new instances of the `DetectEngineCtx` running without a current "VarNameStore", which leads to us triggering the assertion.

My first guess would be that we need either a refcount for the "VarNameStore" to keep it alive across reloads or we need to perform a swap/handover during reload and a flag in the `DetectEngineCtx` that indicates if it owns the "VarNameStore".

Ah, it appears the part I was missing is the "de_ctx_version" logic, which acts effectively what I meant with the handover logic. It should only free the current "VarNameStore" if the version being freed matches the version being active.

This is possibly where the multi-tenant shenanigans come in.

My first guess would be that we need either a refcount for the "VarNameStore" to keep it alive across reloads or we need to perform a swap/handover during reload and a flag in the `DetectEngineCtx` that indicates if it owns the "VarNameStore".

Ah, it appears the part I was missing is the "de_ctx_version" logic, which acts effectively what I meant with the handover logic. It should only free the current "VarNameStore" if the version being freed matches the version being active.

This is possibly where the multi-tenant shenanigans come in.

I've been running Suricata in gdb some more:

- Around reload the new `DetectEngineCtx` allocate a new "VarNameStore", which is put in the "g_varnamestore_staging" global variable
- When "VarNameStoreFree" (after the reload when cleaning up the old `DetectEngineCtx`) the new "VarNameStore" is still in the "g_varnamestore_staging" variable en the one in "g_varnamestore_current" matches the old "de_ctx_version" that is being cleaned up, so a NULL pointer is put in it's place.

Looking at the code I suspect in between these two lines someone should have called `VarNameStoreActivateStaging` (which puts "g_varnamestore_current" in "g_varnamestore_old" and "g_varnamestore_staging" in "g_varnamestore_current").

This function is currently called in two places:

- `detect-engine-build.c`: At the end of `SigGroupBuild` there is:
  ```c
  if (!DetectEngineMultiTenantEnabled()) {
    VarNameStoreActivateStaging();
  }
  ```
- `detect-engine.c`: At the end of `DetectEngineMultiTenantSetup`

I suspect the first codepath is why the not-multi-tenant reload is working. The second callsite is only triggered at startup, but not around reloads.

I'm starting to think this is exactly [https://redmine.openinfosecfoundation.org/issues/2518](https://redmine.openinfosecfoundation.org/issues/2518) as this problem is around reloads and I can't seem to find
any logic that would handle multi-tenant reloading in the reload logic and consequently call the `VarNameStoreActivateStaging` for this case.

#5 - 05/19/2020 02:34 PM - Justin Ossevoort

Some further puzzle pieces:

- The `DetectEngineReload` function appears to want only support non DETECT_ENGINE_MT_STUB type detect engines
- In `detect-engine.c` there is a function `DetectEngineMTApply` that contains parts of the logic from `DetectEngineReload` that also deal with `DETECT_ENGINE_MT_STUB` detect engine types, but this method appears to never be called in the normal reload logic, only around unix socket commands

#6 - 05/19/2020 02:52 PM - Justin Ossevoort

It appears I've zeroed in on the problem:

As before it appears that `DetectEngineReload` function doesn't support DETECT_ENGINE_MT_STUB, so why is it actually being called in the first place? And it should have short-circuited in the case of DETECT_ENGINE_MT_STUB, so why is it being called and runs to completion?

Well according to the breakpoint I set right before the check:

```plaintext
(gdb) print old_de_ctx->type
$9 = DETECT_ENGINE_TYPE_DD_STUB
```

After a little more digging I found this is used to implement `detect.delayed-detect` (which is an option we've always had enabled). After disabling this option the entire `DetectEngineReload` is never called and Suricata starts succesfully.

But triggering a reload using the command channel will indeed trigger the same assertion so the `reload-rules` is indeed still broken (aka: [https://redmine.openinfosecfoundation.org/issues/2518](https://redmine.openinfosecfoundation.org/issues/2518))

So the current state is:

- Bug: `multi-detect` can not be combined with `detect.delayed-detect` (leading to the earlier assertion in "VarNameStore"); It might be enough to declare this unsupported and reorder the check in `PostConfLoadedDetectSetup` so that `multi-detect` takes precedence over `detect.delayed-detect`
- Bug: `reload-rules` can not be used with `multi-detect` (again leading to the earlier assertion in "VarNameStore"); This probably needs a special case in `DetectEngineReload` and could possible use loop over the tenants and `DetectEngineMTApply` for activation.
- I feel there is still some strange corner case around `multi-detect` and `default: true` that should be handled, but I don't understand how that is supposed to work in the first place