This issue is related to https://redmine.openinfosecfoundation.org/issues/4842 and possibly https://redmine.openinfosecfoundation.org/issues/4580

While testing the workaround for 4842 by setting a stream-depth:20mb for SMB, it was seen that there was higher worker thread CPU utilization and hence higher packet processing latency when doing large file transfers between Windows 10 clients. Issue is seen in 5.0.8, but not in 6.0.4 as 6.x code does not have buffering in the SMB rust protocol parser [https://github.com/OISF/suricata/commit/674b8dc0fb35c7f91074cd937b925f1987027].

On using perf top, it was evident that __memmove_avx_unaligned_erm() was getting called frequently

```
PerfTop:  375 irqs/sec kernel: 6.7%  exact: 0.0% [300Hz cycles], (target_pid: 23192)
```

```
41.69%  libc-2.27.so  [.]  __memmove_avx_unaligned_erm
4.59%  suricata  [.]  AppLayerParserGetStateProgress
4.06%  suricata  [.]  rs_smb_state_get_tx_iterator
3.58%  suricata  [.]  GetDetectTx
2.75%  libpthread-2.27.so  [.]  pthread_mutex_unlock
2.61%  libpthread-2.27.so  [.]  pthread_mutex_trylock
2.39%  suricata  [.]  FlowGetProtoMapping
2.27%  suricata  [.]  AppLayerParserTransactionsCleanup
2.10%  suricata  [.]  DetectRunTx
2.10%  libc-2.27.so  [.]  cfree@GLIBC_2.2.5
1.72%  suricata  [.]  AppLayerParserGetTxDetectFlags
1.70%  suricata  [.]  FlowTimeoutHash
1.57%  suricata  [.]  SMBGetTxIterator
1.12%  suricata  [.]  rs_smb_tx_get_alstate_progress
0.99%  suricata  [.]  InspectionBufferClean
0.94%  suricata  [.]  SMBGetAlstateProgress
```

From Intel VTune, it was clear that most calls to __memmove_avx_unaligned_erm() were getting called from suricata::smb::smb::SMBState::parse_tcp_data_ts(). Using rust-gdb, this was further narrowed down to the following call stack.

```
```

03/14/2022
Looking at the code at parse_tcp_data_ts() code in rust/src/smb/smb.rs, it's seen that self.tcp_buffer_ts.len() is called post call to
self.tcp_buffer_ts.split_off(0). But it's seen that
self.tcp_buffer_ts.len() always returns the length as zero since I think Vec::split_off() function sets the length to zero post call to
self.tcp_buffer_ts.split_off(0), which nullifies the greater than 100000 check.

@ From: "rust/src/smb/smb.rs"
0 => i,
._ => {
    v = self.tcp_buffer_ts.split_off(0);
    if self.tcp_buffer_ts.len() + i.len() > 100000 {
        self.set_event(SMBEvent::RecordOverflow);
        return 1;
    };
    v.extend_from_slice(i);
    v.as_slice()
},

From: "rust-1.53.0/library/alloc/src/vec/mod.rs"
pub fn split_off(&mut self, at: usize) -> Self
where
    A: Clone,
{cold
#inline(never)
    fn assert_failed(at: usize, len: usize) -> ! {
        panic!("at` split index (is {}) should be <= len (is {})", at, len);
    }
assert_failed(at, self.len());
if at == 0 {
    // the new vector can take over the original buffer and avoid the copy
    return mem::replace(
        self,
        Vec::with_capacity_in(self.capacity(), self.allocator().clone()),
    );
    };
if at == 0 {
    let other_len = self.len - at;
    let mut other = Vec::with_capacity_in(other_len, self.allocator().clone());
    // Unsafely `set_len` and copy items to `other`.
    unsafe {
        self.set_len(at);
        other.set_len(other_len);
    }
    }
    };
if at > self.len() {
    assert_failed(at, self.len());
    }

From: "rust/src/smb/smb.rs"
// sanity check to avoid memory exhaustion
if state.defrag_buf_ts.len() + buf.len() > 100000 {
    SCLogDebug!("rs_smb_parse_request_tcp: TCP buffer exploded {}",
        state.defrag_buf_ts.len(), buf.len());
    return 1;
};

To fix the issue, the following code change is required (based on similar code in rust/src/smb/smb.rs). Similar fix will also be needed in rust/src/nfs/nfs.rs as well as the bug seems to exists for NFS parser as well.

diff
@@ -1374,11 +1374,11 @@ impl SMBState {
 let tcp_buffer = match self.tcp_buffer_ts.len() {
 0 => i,
```rust
impl SMBState {
    let tcp_buffer = match self.tcp_buffer_tc.len() {
        0 => i,
        _ => {
            v = self.tcp_buffer_tc.split_off(0);
            if self.tcp_buffer_tc.len() + i.len() > 100000 {
                self.set_event(SMBEvent::RecordOverflow);
                return 1;
            }
            v.extend_from_slice(i);
            v.as_slice()
        }
    };
    // ...
}
```

Post the change, the CPU utilization comes down.

To reproduce the issue, you can use the provided Suricata config and Intel VTune screenshot. I also have a 800Mb pcap (compressed size 54MB) which can reproduce the issue. Request the Suricata team to include the pcap as part of Suricata regression tests.

```text
md5sum smb_small_ixia.pcap.7z
0506633f0e9351354429f70b1597b2eb smb_small_ixia.pcap.7z
```

Content of smb.rules (single ETOpen rule is enough to trigger the issue)
drop smb $HOME_NET any -> any any (msg:“ET EXPLOIT ETERNALBLUE Probe Vulnerable System Response MS17-010”;
flow:from_server,established; content:“[[[SMB][25 05 02 00 c0 98 01]]” offset:4 depth:11 content:“[00 00 00 00 00 00 00 00 00 00 00]”;
distance:3 within:10 content:“[00 00 00]” distance:6 within:3 isdataat:!1 relative threshold: type limit track by_src count 1,
seconds 30; reference:url,github.com/rapid7/metasploit-framework/blob/master/modules/auxiliary/scanner/smb/smb_ms17_010.rb;
classtype:trojan-activity; sid:2025650; rev:2; metadata:affected_product Windows_XP_Vista_7_8_10_Server_32_64_Bit,
attack_target Client_Endpoint, created_at 2018_07_11, deployment Internal, former_category EXPLOIT, signature_severity Major,
tag Metasploit, tag ETERNALBLUE, updated_at 2019_09_28;
)

Related issues:
Copied to Bug #5118: smb: excessive CPU utilization and higher packet process... Assigned
Copied to Bug #5119: smb: excessive CPU utilization and higher packet process... Assigned

History
#1 - 02/13/2022 08:43 PM - Victor Julien
- Status changed from New to Assigned
- Assignee set to Victor Julien

#2 - 02/14/2022 04:33 PM - Victor Julien
- Assignee changed from Victor Julien to Jason Ish
- Target version set to 7.0rc1
- Label Needs backport to 5.0, Needs backport to 6.0 added

#3 - 02/16/2022 05:57 PM - Jason Ish
Suricata appears to enter this condition when the parser enters a state when it considers the SMB messages to be in an overflow state. The extra CPU usage is seen as we are copying large amount of data just before throwing the data away. However, we can detect this condition before the copy which reduces CPU usage.

6.0.4 appears to have the same problem but its not as apparent due to more efficient buffering.

PR for 5.0.x: https://github.com/OISF/suricata/pull/7014

#4 - 02/16/2022 05:57 PM - Jason Ish
- Status changed from Assigned to In Review

#5 - 02/18/2022 01:54 PM - Jeff Lucovsky
- Copied to Bug #5118: smb: excessive CPU utilization and higher packet processing latency due to excessive calls to Vec::extend_from_slice() added

#6 - 02/18/2022 01:55 PM - Jeff Lucovsky
- Copied to Bug #5119: smb: excessive CPU utilization and higher packet processing latency due to excessive calls to Vec::extend_from_slice() added

Files
<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>Date</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>config.yaml</td>
<td>73.7 KB</td>
<td>01/10/2022</td>
<td>Srini J</td>
</tr>
<tr>
<td>vtune_smb.png</td>
<td>133 KB</td>
<td>01/10/2022</td>
<td>Srini J</td>
</tr>
</tbody>
</table>