Investigating memory leaks with Valgrind

A Node.js process may run out of memory due to excessive consumption of native memory. Native Memory is memory which is not managed by the V8 Garbage collector and is allocated either by the Node.js runtime, its dependencies or native addons.

This guide provides information on how to use Valgrind to investigate these issues on Linux platforms.

Valgrind

Valgrind is a tool available on Linux distributions which can be used to investigate memory usage including identifying memory leaks (memory which is allocated and not freed) and other memory related problems like double freeing memory.

To use Valgrind:

- Be patient, running under Valgrind slows execution significantly due to the checks being performed.
- Reduce your test case to the smallest reproduce. Due to the slowdown it is important to run the minimum test case in order to be able to do it in a reasonable time.

Installation

It is an optional package in most cases and must be installed explicitly. For example on Debian/Ubuntu:

```
apt-get install valgrind
```

Invocation

The simplest invocation of Valgrind is:

```
valgrind node test.js
```

with the output being:

```
user1@minikube1:~/valgrind/node-addon-examples/1_hello_world/napi$ valgrind node
test.js
==28993== Memcheck, a memory error detector
==28993== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==28993== Using valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==28993== Command: node test.js
==28993==
==28993== Use of uninitialised value of size 8
==28993==    at 0x12F2279: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==28993==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==28993==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==28993==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
```

```
==28993==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==28993==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==28993==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==28993==    by 0x12F3E9C: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==28993==    by 0x12F3C77: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==28993==    by 0xC7C9CF: v8::internal::(anonymous
namespace)::Invoke(v8::internal::Isolate*, v8::internal::(anonymous
namespace)::InvokeParams const&) (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==28993==    by 0xC7CE87: v8::internal::Execution::Call(v8::internal::Isolate*,
v8::internal::Handle<v8::internal::Object>,
v8::internal::Handle<v8::internal::Object>, int,
v8::internal::Handle<v8::internal::Object>*) (in /home/user1/valgrind/node-v12.14.1-
linux-x64/bin/node)
==28993==    by 0xB4CF3A: v8::Function::Call(v8::Local<v8::Context>,
v8::Local<v8::Value>, int, v8::Local<v8::Value>*) (in /home/user1/valgrind/node-
v12.14.1-linux-x64/bin/node)
==28993==
--28993-- WARNING: unhandled amd64-linux syscall: 332
--28993-- You may be able to write your own handler.
--28993-- Read the file README_MISSING_SYSCALL_OR_IOCTL.
--28993-- Nevertheless we consider this a bug.  Please report
--28993-- it at http://valgrind.org/support/bug_reports.html.
==28993==
==28993== HEAP SUMMARY:
==28993==     in use at exit: 6,140 bytes in 23 blocks
==28993==   total heap usage: 12,888 allocs, 12,865 frees, 13,033,244 bytes
allocated
==28993==
==28993== LEAK SUMMARY:
==28993==    definitely lost: 0 bytes in 0 blocks
==28993==    indirectly lost: 0 bytes in 0 blocks
==28993==      possibly lost: 304 bytes in 1 blocks
==28993==    still reachable: 5,836 bytes in 22 blocks
==28993==         suppressed: 0 bytes in 0 blocks
==28993== Rerun with --leak-check=full to see details of leaked memory
==28993==
==28993== For counts of detected and suppressed errors, rerun with: -v
==28993== Use --track-origins=yes to see where uninitialised values come
```

This reports that Node.js is not *completely* clean as there is some memory that was allocated but not freed when the process shut down. It is often impractical/not worth being completely clean in this respect. Modern operating systems will clean up the memory of the process after the shutdown while attempting to free all memory to get a clean report may have a negative impact on the code complexity and shutdown times. Node.js does a pretty good job only leaving on the order of 6 KB that are not freed on shutdown.

An obvious memory leak

Leaks can be introduced in native addons and the following is a simple example leak based on the "Hello world" addon from node-addon-examples.

In this example, a loop which allocates approximately 1 MB of memory and never frees it has been added:

```
void* malloc_holder = nullptr;
napi_value Method(napi_env env, napi_callback_info info) {
  napi_status status;
  napi_value world;
  status = napi_create_string_utf8(env, "world", 5, &world);
  assert(status == napi_ok);

  // NEW LEAK HERE
  for (int i=0; i < 1024; i++) {
    malloc_holder = malloc(1024);
  }

  return world;
}
```

When trying to create a memory leak you need to ensure that the compiler has not optimized out the code that creates the leak. For example, by assigning the result of the allocation to either a global variable or a variable that will be read afterwards the compiler will not optimize it out along with the malloc and Valgrind will properly report the memory leak. If `malloc_holder` in the example above is made into a local variable then the compiler may freely remove it along with the allocations (since it is not used) and Valgrind will not find any leaks since they will no longer exist in the code being run.

Running Valgrind on this code shows the following:

```
user1@minikube1:~/valgrind/node-addon-examples/1_hello_world/napi$ valgrind node
hello.js
==1504== Memcheck, a memory error detector
==1504== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==1504== Using V#algrind-3.13.0 and LibVEX; rerun with -h for copyright info
==1504== Command: node hello.js
==1504==
==1504== Use of uninitialised value of size 8
==1504==    at 0x12F2279: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==1504==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==1504==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==1504==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==1504==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==1504==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
```

```
==1504==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==1504==    by 0x12F3E9C: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==1504==    by 0x12F3C77: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==1504==    by 0xC7C9CF: v8::internal::(anonymous
namespace)::Invoke(v8::internal::Isolate*, v8::internal::(anonymous
namespace)::InvokeParams const&) (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==1504==    by 0xC7CE87: v8::internal::Execution::Call(v8::internal::Isolate*,
v8::internal::Handle<v8::internal::Object>,
v8::internal::Handle<v8::internal::Object>, int,
v8::internal::Handle<v8::internal::Object>*) (in /home/user1/valgrind/node-v12.14.1-
linux-x64/bin/node)
==1504==    by 0xB4CF3A: v8::Function::Call(v8::Local<v8::Context>,
v8::Local<v8::Value>, int, v8::Local<v8::Value>*) (in /home/user1/valgrind/node-
v12.14.1-linux-x64/bin/node)
==1504==
--1504-- WARNING: unhandled amd64-linux syscall: 332
--1504-- You may be able to write your own handler.
--1504-- Read the file README_MISSING_SYSCALL_OR_IOCTL.
--1504-- Nevertheless we consider this a bug.  Please report
--1504-- it at http://valgrind.org/support/bug_reports.html.
world
==1504==
==1504== HEAP SUMMARY:
==1504==     in use at exit: 1,008,003 bytes in 1,032 blocks
==1504==   total heap usage: 17,603 allocs, 16,571 frees, 18,306,103 bytes allocated
==1504==
==1504== LEAK SUMMARY:
==1504==    definitely lost: 996,064 bytes in 997 blocks
==1504==    indirectly lost: 0 bytes in 0 blocks
==1504==      possibly lost: 3,304 bytes in 4 blocks
==1504==    still reachable: 8,635 bytes in 31 blocks
==1504==                       of which reachable via heuristic:
==1504==                         multipleinheritance: 48 bytes in 1 blocks
==1504==         suppressed: 0 bytes in 0 blocks
==1504== Rerun with --leak-check=full to see details of leaked memory
==1504==
==1504== For counts of detected and suppressed errors, rerun with: -v
==1504== Use --track-origins=yes to see where uninitialised values come from
==1504== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Valgrind is reporting a problem as it shows 996,604 bytes as definitely lost and the question is how to find where that memory was allocated. The next step is to rerun as suggested in the output with `--leak-check=full`:

```
user1@minikube1:~/valgrind/node-addon-examples/1_hello_world/napi$ valgrind --leak-
check=full node hello.js
==4174== Memcheck, a memory error detector
==4174== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
```

```
==4174== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==4174== Command: node hello.js
==4174==
==4174== Use of uninitialised value of size 8
==4174==    at 0x12F2279: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F3E9C: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F3C77: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0xC7C9CF: v8::internal::(anonymous
namespace)::Invoke(v8::internal::Isolate*, v8::internal::(anonymous
namespace)::InvokeParams const&) (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0xC7CE87: v8::internal::Execution::Call(v8::internal::Isolate*,
v8::internal::Handle<v8::internal::Object>,
v8::internal::Handle<v8::internal::Object>, int,
v8::internal::Handle<v8::internal::Object>*) (in /home/user1/valgrind/node-v12.14.1-
linux-x64/bin/node)
==4174==    by 0xB4CF3A: v8::Function::Call(v8::Local<v8::Context>,
v8::Local<v8::Value>, int, v8::Local<v8::Value>*) (in /home/user1/valgrind/node-
v12.14.1-linux-x64/bin/node)
==4174==
--4174-- WARNING: unhandled amd64-linux syscall: 332
--4174-- You may be able to write your own handler.
--4174-- Read the file README_MISSING_SYSCALL_OR_IOCTL.
--4174-- Nevertheless we consider this a bug.  Please report
--4174-- it at http://valgrind.org/support/bug_reports.html.
world
==4174==
==4174== HEAP SUMMARY:
==4174==     in use at exit: 1,008,003 bytes in 1,032 blocks
==4174==   total heap usage: 17,606 allocs, 16,574 frees, 18,305,977 bytes allocated
==4174==
==4174== 64 bytes in 1 blocks are definitely lost in loss record 17 of 35
==4174==    at 0x4C3017F: operator new(unsigned long) (in
/usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==4174==    by 0x9AEAD5: napi_module_register (in /home/user1/valgrind/node-
v12.14.1-linux-x64/bin/node)
==4174==    by 0x4010732: call_init (dl-init.c:72)
```

```
==4174==    by 0x4010732: _dl_init (dl-init.c:119)
==4174==    by 0x40151FE: dl_open_worker (dl-open.c:522)
==4174==    by 0x5D052DE: _dl_catch_exception (dl-error-skeleton.c:196)
==4174==    by 0x40147C9: _dl_open (dl-open.c:605)
==4174==    by 0x4E3CF95: dlopen_doit (dlopen.c:66)
==4174==    by 0x5D052DE: _dl_catch_exception (dl-error-skeleton.c:196)
==4174==    by 0x5D0536E: _dl_catch_error (dl-error-skeleton.c:215)
==4174==    by 0x4E3D734: _dlerror_run (dlerror.c:162)
==4174==    by 0x4E3D050: dlopen@@GLIBC_2.2.5 (dlopen.c:87)
==4174==    by 0x9B29A0: node::binding::DLOpen(v8::FunctionCallbackInfo<v8::Value>
const&)::{lambda(node::binding::DLib*)#1}::operator()(node::binding::DLib*) const
(in /home/user1/valgrind/node-v12.14.1-linux-x64/bin/node)
==4174==
==4174== 304 bytes in 1 blocks are possibly lost in loss record 27 of 35
==4174==    at 0x4C31B25: calloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-
linux.so)
==4174==    by 0x40134A6: allocate_dtv (dl-tls.c:286)
==4174==    by 0x40134A6: _dl_allocate_tls (dl-tls.c:530)
==4174==    by 0x5987227: allocate_stack (allocatestack.c:627)
==4174==    by 0x5987227: pthread_create@@GLIBC_2.2.5 (pthread_create.c:644)
==4174==    by 0xAAF9DC: node::inspector::Agent::Start(std::string const&,
node::DebugOptions const&, std::shared_ptr<node::HostPort>, bool) (in
/home/user1/valgrind/node-v12.14.1-linux-x64/bin/node)
==4174==    by 0x9A8BE7:
node::Environment::InitializeInspector(std::unique_ptr<node::inspector::ParentInspecto
 std::default_delete<node::inspector::ParentInspectorHandle> >) (in
/home/user1/valgrind/node-v12.14.1-linux-x64/bin/node)
==4174==    by 0xA1C9A5: node::NodeMainInstance::CreateMainEnvironment(int*) (in
/home/user1/valgrind/node-v12.14.1-linux-x64/bin/node)
==4174==    by 0xA1CB42: node::NodeMainInstance::Run() (in
/home/user1/valgrind/node-v12.14.1-linux-x64/bin/node)
==4174==    by 0x9ACB67: node::Start(int, char**) (in /home/user1/valgrind/node-
v12.14.1-linux-x64/bin/node)
==4174==    by 0x5BBFB96: (below main) (libc-start.c:310)
==4174==
==4174== 2,000 bytes in 2 blocks are possibly lost in loss record 33 of 35
==4174==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-
linux.so)
==4174==    by 0x9794979: Method(napi_env__*, napi_callback_info__*) (in
/home/user1/valgrind/node-addon-
examples/1_hello_world/napi/build/Release/hello.node)
==4174==    by 0x98F764: v8impl::(anonymous
namespace)::FunctionCallbackWrapper::Invoke(v8::FunctionCallbackInfo<v8::Value>
const&) (in /home/user1/valgrind/node-v12.14.1-linux-x64/bin/node)
==4174==    by 0xBA6FC8: v8::internal::MaybeHandle<v8::internal::Object>
v8::internal::(anonymous namespace)::HandleApiCallHelper<false>
(v8::internal::Isolate*, v8::internal::Handle<v8::internal::HeapObject>,
v8::internal::Handle<v8::internal::HeapObject>,
v8::internal::Handle<v8::internal::FunctionTemplateInfo>,
v8::internal::Handle<v8::internal::Object>, v8::internal::BuiltinArguments) (in
/home/user1/valgrind/node-v12.14.1-linux-x64/bin/node)
==4174==    by 0xBA8DB6: v8::internal::Builtin_HandleApiCall(int, unsigned long*,
```

```
v8::internal::Isolate*) (in /home/user1/valgrind/node-v12.14.1-linux-x64/bin/node)
==4174==    by 0x1376358: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==
==4174== 997,000 bytes in 997 blocks are definitely lost in loss record 35 of 35
==4174==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-
linux.so)
==4174==    by 0x9794979: Method(napi_env__*, napi_callback_info__*) (in
/home/user1/valgrind/node-addon-
examples/1_hello_world/napi/build/Release/hello.node)
==4174==    by 0x98F764: v8impl::(anonymous
namespace)::FunctionCallbackWrapper::Invoke(v8::FunctionCallbackInfo<v8::Value>
const&) (in /home/user1/valgrind/node-v12.14.1-linux-x64/bin/node)
==4174==    by 0xBA6FC8: v8::internal::MaybeHandle<v8::internal::Object>
v8::internal::(anonymous namespace)::HandleApiCallHelper<false>
(v8::internal::Isolate*, v8::internal::Handle<v8::internal::HeapObject>,
v8::internal::Handle<v8::internal::HeapObject>,
v8::internal::Handle<v8::internal::FunctionTemplateInfo>,
v8::internal::Handle<v8::internal::Object>, v8::internal::BuiltinArguments) (in
/home/user1/valgrind/node-v12.14.1-linux-x64/bin/node)
==4174==    by 0xBA8DB6: v8::internal::Builtin_HandleApiCall(int, unsigned long*,
v8::internal::Isolate*) (in /home/user1/valgrind/node-v12.14.1-linux-x64/bin/node)
==4174==    by 0x1376358: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==
==4174== LEAK SUMMARY:
==4174==    definitely lost: 997,064 bytes in 998 blocks
==4174==    indirectly lost: 0 bytes in 0 blocks
```

```
==4174==      possibly lost: 2,304 bytes in 3 blocks
==4174==    still reachable: 8,635 bytes in 31 blocks
==4174==                       of which reachable via heuristic:
==4174==                         multipleinheritance: 48 bytes in 1 blocks
==4174==         suppressed: 0 bytes in 0 blocks
==4174== Reachable blocks (those to which a pointer was found) are not shown.
==4174== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==4174==
==4174== For counts of detected and suppressed errors, rerun with: -v
==4174== Use --track-origins=yes to see where uninitialised values come from
==4174== ERROR SUMMARY: 5 errors from 5 contexts (suppressed: 0 from 0)
```

This is the most interesting part of the report:

```
==4174== 997,000 bytes in 997 blocks are definitely lost in loss record 35 of 35
==4174==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-
linux.so)
==4174==    by 0x9794979: Method(napi_env__*, napi_callback_info__*) (in
/home/user1/valgrind/node-addon-
examples/1_hello_world/napi/build/Release/hello.node)
==4174==    by 0x98F764: v8impl::(anonymous
namespace)::FunctionCallbackWrapper::Invoke(v8::FunctionCallbackInfo<v8::Value>
const&) (in /home/user1/valgrind/node-v12.14.1-linux-x64/bin/node)
==4174==    by 0xBA6FC8: v8::internal::MaybeHandle<v8::internal::Object>
v8::internal::(anonymous namespace)::HandleApiCallHelper<false>
(v8::internal::Isolate*, v8::internal::Handle<v8::internal::HeapObject>,
v8::internal::Handle<v8::internal::HeapObject>,
v8::internal::Handle<v8::internal::FunctionTemplateInfo>,
v8::internal::Handle<v8::internal::Object>, v8::internal::BuiltinArguments) (in
/home/user1/valgrind/node-v12.14.1-linux-x64/bin/node)
==4174==    by 0xBA8DB6: v8::internal::Builtin_HandleApiCall(int, unsigned long*,
v8::internal::Isolate*) (in /home/user1/valgrind/node-v12.14.1-linux-x64/bin/node)
==4174==    by 0x1376358: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==4174==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
```

From the stack trace we can tell that the leak came from a native addon:

```
==4174==    by 0x9794979: Method(napi_env__*, napi_callback_info__*) (in
/home/user1/valgrind/node-addon-
```

```
examples/1_hello_world/napi/build/Release/hello.node)
```

What we can't tell is where in the native addon the memory is being allocated. This is because by default the addon is compiled without the debug symbols which Valgrind needs to be able to provide more information.

Enabling debug symbols to get more information

Leaks may be either in addons or Node.js itself. The sections which follow cover the steps needed to enable debug symbols to get more info.

Native addons

To enable debug symbols for all of your addons that are compiled on install use:

```
npm install --debug
```

Any options which are not consumed by npm are passed on to node-gyp and this results in the addons being compiled with the debug option.

If the native addon contains pre-built binaries you will need to force a rebuild.

```
npm install --debug
npm rebuild
```

The next step is to run Valgrind after the rebuild. This time the information for the leaking location includes the name of the source file and the line number:

```
==18481== 997,000 bytes in 997 blocks are definitely lost in loss record 35 of 35
==18481==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-
linux.so)
>>>> ==18481==    by 0x9794989: Method(napi_env__*, napi_callback_info__*)
(hello.cc:13)  <<<<<
==18481==    by 0x98F764: v8impl::(anonymous
namespace)::FunctionCallbackWrapper::Invoke(v8::FunctionCallbackInfo<v8::Value>
const&) (in /home/user1/val  grind/node-v12.14.1-linux-x64/bin/node)
==18481==    by 0xBA6FC8: v8::internal::MaybeHandle<v8::internal::Object>
v8::internal::(anonymous namespace)::HandleApiCallHelper<false>(v8::internal::
Isolate*, v8::internal::Handle<v8::internal::HeapObject>,
v8::internal::Handle<v8::internal::HeapObject>,
v8::internal::Handle<v8::internal::FunctionTem  plateInfo>,
v8::internal::Handle<v8::internal::Object>, v8::internal::BuiltinArguments) (in
/home/user1/valgrind/node-v12.14.1-linux-x64/bin/node)
==18481==    by 0xBA8DB6: v8::internal::Builtin_HandleApiCall(int, unsigned long*,
v8::internal::Isolate*) (in /home/user1/valgrind/node-v12.14.1-linux-x64/bin/node)
==18481==    by 0x1376358: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==18481==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==18481==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==18481==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
```

```
x64/bin/node)
==18481==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==18481==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
==18481==    by 0x12F68A3: ??? (in /home/user1/valgrind/node-v12.14.1-linux-
x64/bin/node)
```

This new output shows us exactly where the leak is occurring in the file `hello.cc`:

```
 6 void* malloc_holder = nullptr;
 7 napi_value Method(napi_env env, napi_callback_info info) {
 8   napi_status status;
 9   napi_value world;
10   status = napi_create_string_utf8(env, "world", 5, &world);
11   assert(status == napi_ok);
12   for (int i=0; i< 1000; i++) {
13     malloc_holder = malloc(1000);  // <<<<<< This is where we are allocating the
memory that is not freed
14   }
15   return world;
16 }
```

Node.js binary

If the leak is not in an addon and is instead in the Node.js binary itself, you may need to compile node yourself and turn on debug symbols. Looking at this entry reported by Valgrind, with a release binary we see:

```
 ==4174== 304 bytes in 1 blocks are possibly lost in loss record 27 of 35
==4174==    at 0x4C31B25: calloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-
linux.so)
==4174==    by 0x40134A6: allocate_dtv (dl-tls.c:286)
==4174==    by 0x40134A6: _dl_allocate_tls (dl-tls.c:530)
==4174==    by 0x5987227: allocate_stack (allocatestack.c:627)
==4174==    by 0x5987227: pthread_create@@GLIBC_2.2.5 (pthread_create.c:644)
==4174==    by 0xAAF9DC: node::inspector::Agent::Start(std::string const&,
node::DebugOptions const&, std::shared_ptr<node::HostPort>, bool) (in
/home/user1/valgrind/node-v12.14.1-linux-x64/bin/node)
==4174==    by 0x9A8BE7:
node::Environment::InitializeInspector(std::unique_ptr<node::inspector::ParentInspecto
 std::default_delete<node::inspector::ParentInspectorHandle> >) (in
/home/user1/valgrind/node-v12.14.1-linux-x64/bin/node)
==4174==    by 0xA1C9A5: node::NodeMainInstance::CreateMainEnvironment(int*) (in
/home/user1/valgrind/node-v12.14.1-linux-x64/bin/node)
==4174==    by 0xA1CB42: node::NodeMainInstance::Run() (in
/home/user1/valgrind/node-v12.14.1-linux-x64/bin/node)
==4174==    by 0x9ACB67: node::Start(int, char**) (in /home/user1/valgrind/node-
v12.14.1-linux-x64/bin/node)
==4174==    by 0x5BBFB96: (below main) (libc-start.c:310)
```

This gives us some information of where to look (`node::inspector::Agent::Start`) but not where in that function. We get more information than you might expect (or see by default with addons) because the Node.js binary exports many of its symbols using `-rdynamic` so that they can be used by addons. If the stack gives you enough information to track down where the leak is, that's great, otherwise the next step is to compile a debug build of Node.js.

To get additional information with Valgrind:

- Check out the Node.js source corresponding to the release that you want to debug. For example:

```
git clone https://github.com/nodejs/node.git
git checkout v12.14.1
```

- Compile with debug enabled (for additional info see building a debug build). For example, on *nix:

```
./configure --debug
make -j4
```

- Make sure to run with your compiled debug version of Node.js. Having used `./configure --debug` , two binaries will have been built when `make` was run. You must use the one which is in `out/Debug` .

Running Valgrind using the debug build of Node.js shows:

```
==44112== 592 bytes in 1 blocks are possibly lost in loss record 26 of 27
==44112==    at 0x4C2BF79: calloc (vg_replace_malloc.c:762)
==44112==    by 0x4012754: _dl_allocate_tls (in /usr/lib64/ld-2.17.so)
==44112==    by 0x586287B: pthread_create@@GLIBC_2.2.5 (in /usr/lib64/libpthread-
2.17.so)
==44112==    by 0xFAB2D2: node::inspector::(anonymous
namespace)::StartDebugSignalHandler() (inspector_agent.cc:140)
==44112==    by 0xFACB10: node::inspector::Agent::Start(std::string const&,
node::DebugOptions const&, std::shared_ptr<node::HostPort>, bool)
(inspector_agent.cc:777)
==44112==    by 0xE3A0BB:
node::Environment::InitializeInspector(std::unique_ptr<node::inspector::ParentInspecto
 std::default_delete<node::inspector::ParentInspectorHandle> >) (node.cc:216)
==44112==    by 0xEE8F3E: node::NodeMainInstance::CreateMainEnvironment(int*)
(node_main_instance.cc:222)
==44112==    by 0xEE8831: node::NodeMainInstance::Run() (node_main_instance.cc:108)
==44112==    by 0xE3CDEC: node::Start(int, char**) (node.cc:996)
==44112==    by 0x22D8BBF: main (node_main.cc:126)
```

Now we can see the specific file name and line in the Node.js code which caused the allocation (inspector_agent.cc:140).

We can examine that line (and its surrounding code) to find a solution for the memory leak.