

Forum:[Hack Forums](#)

Thread: [dll injection with nim?](#)

RowdyCat - August 3, 2025, 9:44 am

hi ive been messing around with nim and wonder how do i compild a dll with nim and load it. ive created the injector exe file that uses the windows api. i tested that i can successfully load system dll files but my own dll file compiled with nim seems to fail. any help would be nice, thanks

??????????? - August 4, 2025, 3:08 am

(Aug 3, 2025 09:44 AM)RowdyCat Wrote: [\[CITE\]](#) hi ive been messing around with nim and wonder how do i compild a dll with nim and load it. ive created the injector exe file that uses the windows api. i tested that i can successfully load system dll files but my own dll file compiled with nim seems to fail. any help would be nice, thanks

You're almost certainly hitting one of the usual gotchas: calling convention/name decoration mismatch, bitness mismatch, missing export, or a dependency issue. Here's a minimal working workflow in Nim for a DLL you can load with `LoadLibrary`/`GetProcAddress` from a Windows injector:

1. Nim DLL (cdecl, simplest)

```
```nim
mylib.nim
import os, strutils
proc add(a: cint, b: cint): cint {.exportc.} = a + b
```
```

Compile it as a DLL:

```
```sh
nim c --app:lib --opt:speed --out:mylib.dll mylib.nim
```
```

`exportc` uses the C (cdecl) calling convention and exports the symbol as `add` (no mangling). Verify with:

```
```sh
dumpbin /EXPORTS mylib.dll
```
```

(or use `objdump`/Dependency Walker) to see that `add` is exported.

2. If your injector expects stdcall (name decoration)

Change to:

```
```nim
proc add(a: cint, b: cint): cint {.exportc, stdcall.} = a + b
```
```

Then the symbol will be decorated (e.g., ``add@8``). You can avoid export-name issues by supplying a `.def` file:

```
mylib.def
...
LIBRARY mylib
EXPORTS
    add
...
```

And compile with that in the same directory (the Nim-generated build will pick it up if named appropriately or you can post-process the generated `.lib`/link` with it).

3. Common pitfalls to check if your DLL “fails to load”:

Bitness mismatch: Injector is 64-bit but DLL is 32-bit (or vice versa). They must match.

Export not found: Use ``dumpbin /EXPORTS mylib.dll`` to confirm the name your injector is requesting (``"add"`` vs ``"add@8"``). Calling convention mismatch: If your function is `stdcall` but you call it as `cdecl`, you’ll crash. Make sure your function pointer typedef in the injector matches:

```
```c
typedef int (__cdecl *add_t)(int,int); // for exportc only
// or
typedef int (__stdcall *add_t)(int,int); // if you used stdcall
```
```

Dependencies: Nim code may pull in runtime support; use Dependency Walker or ``dumpbin`` to see if the DLL is missing a dependent runtime (e.g., mingw/MSVC runtimes). Building statically can help: try adding Nim compile flags to reduce external dependencies if needed. Path / loader issues: Use a full absolute path in ``LoadLibraryW``, and check ``GetLastError()`` if it fails.

AV / security: Custom DLL injection can be blocked—ensure the target process isn’t killing your injection prematurely.

4. Example loader snippet (C-style)

```
```c
HMODULE h = LoadLibraryA("C:\\full\\path\\mylib.dll");
if (!h) { printf("LoadLibrary failed: %lu\n", GetLastError()); return; }
typedef int (__cdecl add_t)(int,int); // adjust if stdcall
add_t add = (add_t)GetProcAddress(h, "add");
if (!add) { printf("GetProcAddress failed\n"); return; }
int res = add(2,3);
printf("2+3 = %d\n", res);
```
```

5. Debugging tips

Use ``dumpbin /DEPENDENTS mylib.dll`` or Dependency Walker to spot missing DLLs. Print ``GetLastError()`` after ``LoadLibrary`/`GetProcAddress`` to get the failure reason.

If symbol names look weird, use a `.def` file to force clean exports.

* Match injector and DLL architecture (use ``file`` or ``dumpbin`` to confirm).

Bshol - August 19, 2025, 11:52 pm

To compile a DLL with Nim, use `nim c --app:lib --noMain yourfile.nim` and make sure to export your

functions with `{.exportc, dynlib.}` pragmas. The most common issue is that Nim DLLs need proper function exports - add `{.exportc: "functionName", dynlib.}` to each function you want to call from your injector, and ensure you're using the correct calling convention (usually `{.stdcall.}` for Windows APIs).

You can also check if your DLL exports are correct using `dumpbin /exports yourdll.dll` or Dependency Walker to verify the function names are properly exported.

??????????? - August 20, 2025, 12:36 am

(Aug 3, 2025 09:44 AM)RowdyCat Wrote: [\[CITE\]](#) hi ive been messing around with nim and wonder how do i compild a dll with nim and load it. ive created the injector exe file that uses the windows api. i tested that i can successfully load system dll files but my own dll file compiled with nim seems to fail. any help would be nice, thanks

So was it of any help?

RowdyCat - August 20, 2025, 11:48 am

(Aug 20, 2025 12:36 AM)??????? Wrote: [\[CITE\]](#) So was it of any help?
yeah i just decided learn C which its compiler works great in my lab. i used rundll32.exe to execute bypass the antivirus which allows me to inject the shellcode flawlessly.

??????????? - August 20, 2025, 11:58 pm

(Aug 20, 2025 11:48 AM)RowdyCat Wrote: [\[CITE\]](#) yeah i just decided learn C which its compiler works great in my lab. i used rundll32.exe to execute bypass the antivirus which allows me to inject the shellcode flawlessly.

Seems lika a great update.