

Forum:[Hack Forums](#)

Thread: [.dll Whitelist / Auto-unload Modules in Runtime with C++ // Proof of Concept //](#)

Hackers - March 18, 2024, 3:28 pm

DLL Whitelist

Proof of Concept

+ Notes

Implementing a module whitelist is a way to start focusing on the integrity of your program in runtime. For example, if your program doesn't need user32.dll, then why load it? This POC (proof of concept) implements a strict whitelist of allowed modules (or .dll files) and unloads any others.

Benefits for Intellectual Property Protections & Runtime Security

- Due to this being a whitelist & not a blacklist, a program does not have to maintain a list of "unwanted modules" and instead only needs its own "approved modules" list. This is a smaller amount of data, is preserved over time, and is easier for devs to maintain
- The POC can be modified to exit the program or implement different security features upon unknown modules being loaded , rather than simply unloading them

Known Concerns / Dev Notes

- a bad actor could use the name of a whitelisted module & try to load it, tricking your software
- other software can always re-load the module after unloading and this tool doesn't prevent that, it just auto unloads every X milliseconds (defined in code)
- there are other, perhaps better, ways of preventing unwanted modules being loaded
- this is not meant to be a standalone class that accounts for everything you want to do. It's a POC - Take the code as-is & modify it if you want to.

Implications for Anti-*, Blue Team, etc

As this POC is entirely educational & for research purposes only, I find it important to mention some implications here.

Although an auto-unloader does not prevent a module from being loaded again, it can cause issues if a software relies on its dll being loaded into a program. For example, at least 1 well-known provider may be unable to stop file changes on a locked folder vault. In this test case, the provider's anti-ransomware tool was unable to stop a file on Desktop being modified due to its dll being unloaded repeatedly. This may be a unique case and not the norm, but the well-known provider is vulnerable to dll/module unloading attacks for that specific feature while its other main protection software does not have the same issue. It is important for all providers to ensure that their detections do not rely on a module always being loaded, or to detect patterns related to auto-unloading.

<https://github.com/LNodesL/DLL-Whitelist-POC>

If you want the FULL code, head to the git repo. The following code samples are just highlights:

The Whitelist

Code EnumProcessModules(hProcess, hModules, sizeof(hModules), &cbNeeded)- Using this we can get all of the modules loaded into our program/process

Code int moduleCount = cbNeeded / sizeof(HMODULE);

```
std::vector<std::string> knownModules = {
    "ntdll.dll",
    "KERNEL32.DLL",
    "KERNELBASE.dll",
    "apphelp.dll",
    "ucrtbase.dll",
    "msvcrt.dll",
    "kernel.appcore.dll"
};
```

knownModules.push_back(GetExecutableFileName());- A standard vector list/array of std::string

- I added some common modules. To test, remove 1 that you know will load on your system to see it detected & unloaded

- We have to add our own executable name, which will show up in the module list

Code GetModuleFileNameExA(hProcess, hModules[i], szModName, sizeof(szModName)- We can use this to get the filename instead of a path

- You may want to consider using pathing instead of just filenames in future, especially if the origin of the module matters to you & not just the name

- Remember the known concerns list above - for 100% security in runtime, this POC won't cover you 100%

Code bool isKnownModule = false;

```
for (const auto& knownModule : knownModules) {
    if (moduleName.find(knownModule) != std::string::npos) {
        isKnownModule = true;
        break;
    }
}
```

```
if(!isKnownModule) {
    std::cout << "Unknown module detected: " << moduleName << std::endl;
    UnloadDllByName(moduleName);
}
```

- This is the loop that checks to see if anything is in our whitelist or not. If not, it triggers the unload function!

- I recommend removing all std::cout / err / etc from your final program release

Unloading by name:

Code bool UnloadDllByName(std::string dllName) {

```
HMODULE hModule = GetModuleHandleA(dllName.c_str()); // Use ANSI version directly
if (hModule == NULL) {
    std::cerr << "DLL not found: " << dllName << std::endl;
    return false;
}
```

```
if (FreeLibrary(hModule)) {
    std::cout << "DLL unloaded successfully: " << dllName << std::endl << std::endl;
    return true;
} else {
    std::cerr << "Failed to unload DLL: " << dllName << std::endl << std::endl;
    return false;
}
```

- In this code we are using standard functions to obtain the module handle & then use FreeLibrary to unload.

- You should look into [LdrUnloadDll from ntdll](#) if you want a more direct, lower level method

Using a Thread

One thing you will notice in main.cpp (our prototype / test program) is that we use a detached thread to run the auto unloader:

Code `std::thread t(ForceDllWhitelist, 500);`
`t.detach();` We do this so that the loop inside of dll-whitelist.cpp doesn't steal the main thread of our int main program.

The int is the number of milliseconds to delay between each module check / loop. 500 = 500ms, or 1/2 of a second (ex: use 1000 for 1 second delay)

- This doesn't delay the first check, just the delay in between

Non-threaded usage:

Code `ForceDllWhitelist(500);` - This will cause your main program to freeze up / not continue if no changes are made in there.

- I just wanted to push the POC out and write this up - in hindsight I should've just used threads in the included file, not in main

Custom Sleep

The custom sleep function can be worked on or improved, but does work as-is on personal systems. I'm not sure if people know about how clocks can be manipulated in a VM, or if that matters, but I wanted to mention this just in case you haven't thought about that yet. I haven't tested how that works with loop ticks & clocks being manipulated & the implications for this method although I plan on releasing a cryptographic-cracking formula that has a scaling difficulty to implement rough-estimate timing functions without the ability to be skipped - that will be a different POC repo on my gits.

```
Code void customSleep(int milliseconds) {
    auto start = std::chrono::high_resolution_clock::now();
    auto end = start + std::chrono::milliseconds(milliseconds);
    while (std::chrono::high_resolution_clock::now() < end) {
        // Busy-wait loop.
    }
}
```

One more thought...

I just used this format for the first time today:

Code `#include "inc/dllWhitelist.cpp"`- Usually I would include this in the build / compile command or just use a header and not do it in the file like this, but I figured out that this works too so kept it in. Not sure how others will feel about this.

7 - March 18, 2024, 4:31 pm

Am I reading this wrong or can the

"Known Concerns / Dev Notes" have a positive malicious use

Great read though! Again HQ as always

Hackers - March 18, 2024, 6:08 pm

(Mar 18, 2024 04:31 PM)NulldB Wrote: [CITE](#) Am I reading this wrong or can the "Known Concerns / Dev Notes" have a positive malicious use

Great read though! Again HQ as always

The known concerns, to me, are flaws in the exact definition of what this tool does for auto-unloading. It's also basically a list of things I'd assume people would call out because this tool can't prevent a module from loading 100% as-is.

I'm sure any tool can be used for malicious uses, as people will always find a way to use products for BH activities - even Microsoft & Apple can't stop people using their products for that.

If I had to theorize how the known concerns could possibly be used, a red team could use that knowledge:

- using a dummy dynamic module can prevent another module from being loaded, if you have priority in load order

- using an internal, static module can prevent another module of same name being loaded

edit:

important to note that it is highly unlikely you'll be able to utilize this just on its own simply by naming something the same & throwing it in CWD. this preload method specifically require best practices for library loading to be ignored - but over years many softwares have had this vuln .

Depending on the blue team / anti-* software, it may be possible to even use default load order just by placing a dummy dll somewhere & not even loading it into your own program. For example, if a software checks a path that your program has access to, the dummy dll can be placed into it causing that other program to use it.

Any program that doesn't clear out their CWD as a potential dll path that also uses loadlibrary or similar is vulnerable to a pre-loading attack due to load order priority:

- For example, if you know my software calls "user32.dll" without a path, you can put user32.dll into the run-directory of my program and I'll call your dll instead of the default one I want.

That's outlined here on an official MS page:

<https://support.microsoft.com/en-us/topic/secure-loading-of-libraries-to-prevent-dll-preloading-attacks-d41303ec-0748-9211-f317-2edc819682e1>

Red teams may be interested in pre-loading DLLs by statically linking their own versions (or dummy dll files).

Red teams may also try to do something like unloading a detected unapproved module and then

on-the-fly creating a dll of the same name, to try to load it before the real dll can re-load into the program. However, I am not sure if this works because of load orders and such & the 'calling program' would most likely use a direct path to their own dll to load into yours - it wouldn't call loadlib functions from your program (which is how CWD is taken advantage of in preloading).

- However, blue teams could also make the mistake of loading a library via search (like 'test.dll' instead of 'path/to/test.dll') and then getting the path after it's dynamically resolved, and then they could push that dll to your program. In this case the red team could put test.dll into the CWD of blueteam's anti* software so that it pulls the dummy test.dll before pushing that into red team's program. However this is total theory and requires blue team to be vulnerable to pre-loading and also relying on that same vuln to find the path to a target driver they're pushing to yours. This is very unlikely to happen in this sequence imo.

SIM-5457022 - March 18, 2024, 10:37 pm

(Mar 18, 2024 06:08 PM)nodes Wrote: [\[CITE\]](#) The known concerns, to me, are flaws in the exact definition of what this tool does for auto-unloading. It's also basically a list of things I'd assume people would call out because this tool can't prevent a module from loading 100% as-is.

I'm sure any tool can be used for malicious uses, as people will always find a way to use products for BH activities - even Microsoft & Apple can't stop people using their products for that.

If I had to theorize how the known concerns could possibly be used, a red team could use that knowledge:

- using a dummy dynamic module can prevent another module from being loaded, if you have priority in load order

- using an internal, static module can prevent another module of same name being loaded

edit:

important to note that it is highly unlikely you'll be able to utilize this just on its own simply by naming something the same & throwing it in CWD. this preload method specifically require best practices for library loading to be ignored - but over years many softwares have had this vuln .

Depending on the blue team / anti-* software, it may be possible to even use default load order just by placing a dummy dll somewhere & not even loading it into your own program. For example, if a software checks a path that your program has access to, the dummy dll can be placed into it causing that other program to use it.

Any program that doesn't clear out their CWD as a potential dll path that also uses loadlibrary or similar is vulnerable to a pre-loading attack due to load order priority:

- For example, if you know my software calls "user32.dll" without a path, you can put user32.dll into the run-directory of my program and I'll call your dll instead of the default one I want.

That's outlined here on an official MS page:

<https://support.microsoft.com/en-us/topic/secure-loading-of-libraries-to-prevent-dll-preloading-attack-s-d41303ec-0748-9211-f317-2edc819682e1>

Red teams may be interested in pre-loading DLLs by statically linking their own versions (or dummy dll files).

Red teams may also try to do something like unloading a detected unapproved module and then on-the-fly creating a dll of the same name, to try to load it before the real dll can re-load into the program. However, I am not sure if this works because of load orders and such & the 'calling program' would most likely use a direct path to their own dll to load into yours - it wouldn't call loadlib functions from your program (which is how CWD is taken advantage of in preloading).

- However, blue teams could also make the mistake of loading a library via search (like 'test.dll' instead of 'path/to/test.dll') and then getting the path after it's dynamically resolved, and then they

could push that dll to your program. In this case the red team could put test.dll into the CWD of blueteam's anti* software so that it pulls the dummy test.dll before pushing that into red team's program. However this is total theory and requires blue team to be vulnerable to pre-loading and also relying on that same vuln to find the path to a target driver they're pushing to yours. This is very unlikely to happen in this sequence imo.

So, what you're saying that when we write our programs we better double check what files it's grabbing and make sure it's a legit dll that needs to be loaded and not a fake one that a hacker has laying around in the same folder as the program.

Hackers - March 18, 2024, 11:13 pm

(Mar 18, 2024 10:37 PM)SIM-5457022 Wrote: [\[CITE\]](#) So, what you're saying that when we write our programs we better double check what files it's grabbing and make sure it's a legit dll that needs to be loaded and not a fake one that a hacker has laying around in the same folder as the program. I feel like 90% of git is probably vulnerable to load priority in the program directory

ttbr0 - March 19, 2024, 12:43 am

code looks good for unloading dlls. great work! something to note is that modules have reference counts for however many times LoadLibrary was called on it. calling FreeLibrary once may not unload the module so you need to check the reference count and unload it until it is zero and the library actually gets unloaded. i believe the reference count is in the peb's LDR_DATA_TABLE_ENTRY which you already know from your previous thread on peb walking.

some suggestions for keeping dlls unloaded:

1. primitive method, but you could hook LoadLibrary or LdrLoadDll or NtMapViewOfSection and check the dll name
2. consider using the PROCESS_CREATION_MITIGATION_POLICY_BLOCK_NON_MICROSOFT_BINARIES_ALWAYS_ON mitigation to prevent non ms signed binaries from being loaded into the process (probably sufficient for most use cases since ms binaries are not malicious)
3. quite aggressive but using the PROCESS_CREATION_MITIGATION_POLICY_PROHIBIT_DYNAMIC_CODE_ALWAYS_ON mitigation could prevent manual mapping of dlls. there are compatibility issues with dynamic code, so a process with the c# runtime might implode

Hackers - March 19, 2024, 7:29 am

(Mar 19, 2024 12:43 AM)ttbr0 Wrote: [\[CITE\]](#) code looks good for unloading dlls. great work! something to note is that modules have reference counts for however many times LoadLibrary was called on it. calling FreeLibrary once may not unload the module so you need to check the reference count and unload it until it is zero and the library actually gets unloaded. i believe the reference count is in the peb's LDR_DATA_TABLE_ENTRY which you already know from your previous thread on peb walking.

some suggestions for keeping dlls unloaded:

1. primitive method, but you could hook LoadLibrary or LdrLoadDll or NtMapViewOfSection and check the dll name
2. consider using the PROCESS_CREATION_MITIGATION_POLICY_BLOCK_NON_MICROSOFT_BINARIES_ALWAYS_ON mitigation to prevent non ms signed binaries from being loaded into the process (probably sufficient for most use cases since ms binaries are not malicious)
3. quite aggressive but using the PROCESS_CREATION_MITIGATION_POLICY_PROHIBIT_DYNAMIC_CODE_ALWAYS_ON

mitigation could prevent manual mapping of dlls. there are compatibility issues with dynamic code, so a process with the c# runtime might implode

Thanks, I appreciate that a lot!

The reference counts is something I did read about, but honestly forgot about that issue. I'm going to update this to use the PEB count & make sure all are unloaded no matter what. Thanks for this reminder :)

I am not familiar with the flags/definitions you provided for the policies but am going to look into those to see how that can work.

- I appreciate your comments across my threads man! You seem like an HQ user

Update for blue teams

- although anti-ransomware on 1 provider was bypassed w/ unloading method, the signature detect was fast

- I have now broken the anti-ransomware from 2 separate codebases using unloading

- at this time I haven't determined how the signature is detected or if it's possible for a read team to re-bypass without rewriting/recompiling their code

ttbr0 - March 19, 2024, 6:18 pm

(Mar 19, 2024 07:29 AM)nodes Wrote: [\[CITE\]](#) Thanks, I appreciate that a lot!

The reference counts is something I did read about, but honestly forgot about that issue. I'm going to update this to use the PEB count & make sure all are unloaded no matter what. Thanks for this reminder :)

I am not familiar with the flags/definitions you provided for the policies but am going to look into those to see how that can work.

- I appreciate your comments across my threads man! You seem like an HQ user

Update for blue teams

- although anti-ransomware on 1 provider was bypassed w/ unloading method, the signature detect was fast

- I have now broken the anti-ransomware from 2 separate codebases using unloading

- at this time I haven't determined how the signature is detected or if it's possible for a read team to re-bypass without rewriting/recompiling their code

thanks dude, happy to help someone who is into research and improving their skills! i just remembered that dll pinning exists using GetModuleHandleEx with the

GET_MODULE_HANDLE_EX_FLAG_PIN flag which makes FreeLibrary refuse to decrement reference counts and unload the library. from what i quickly looked at, it seems to just set the peb reference count to 0xFFFF as an identifier for pinning. unsetting that should easily undo the pinning.

SIM-5457022 - March 19, 2024, 11:54 pm

(Mar 18, 2024 11:13 PM)nodes Wrote: [\[CITE\]](#) I feel like 90% of git is probably vulnerable to load priority in the program directory

but to combat the load priority issue would we as programmers need to double check which files to load into our program? Should we be checking signatures that are signed?

what would avoid getting a load priority vulnerability?

Hackers - March 20, 2024, 6:27 am

(Mar 19, 2024 06:18 PM)ttbr0 Wrote: [\[CITE\]](#) thanks dude, happy to help someone who is into research and improving their skills! i just remembered that dll pinning exists using GetModuleHandleEx with the GET_MODULE_HANDLE_EX_FLAG_PIN flag which makes FreeLibrary refuse to decrement reference counts and unload the library. from what i quickly looked at, it seems to just set the peb reference count to 0xFFFF as an identifier for pinning. unsetting that should easily undo the pinning. Ah, good to know. Maybe I need to do a v3 of this where I use the PEB for count & manual unloading rather than funcs. I will do that soon probably.

(Mar 19, 2024 11:54 PM)SIM-5457022 Wrote: [\[CITE\]](#) but to combat the load priority issue would we as programmers need to double check which files to load into our program? Should we be checking signatures that are signed?

what would avoid getting a load priority vulnerability?

I don't know too much about this part, but yes I believe signature checking anything you load in, using full paths to trusted binaries/libs, etc.

To avoid the load priority vuln you can use direct paths / dynamic paths to known, trusted folders rather than just using the no-path name in a load library func. However, another step would be needed (such as the checksum / signature checking) to make sure the trusted folder's file hasn't been altered. I know anti-* softwares use minifilters to monitor their own source files.

dasdas543534534 - April 19, 2024, 12:02 pm

(Mar 18, 2024 03:28 PM)nodes Wrote: [\[CITE\]](#) -

Cute but won't work against statically linked DLL's that are abused for side loading. (You can reverse engineer why)

As far as code:

```
Code HMODULE hModules[1024];
DWORD cbNeeded;
Always initialize variables.
```

```
Code for (int i = 0; i < moduleCount; i++) {
Ugly, will never be used in production code, use std::span and for each loop.
```

```
Code std::string moduleName = szModName;
Un-necessary copy created, if you know you don't need szModName anymore, use std::move, or
std::string_view?
```

This entire code :

```
Code bool UnloadDllByName(std::string dllName) {
    HMODULE hModule = GetModuleHandleA(dllName.c_str()); // Use ANSI version directly
    if (hModule == NULL) {
        std::cerr << "DLL not found: " << dllName << std::endl;
        return false;
    }

    if (FreeLibrary(hModule)) {
        std::cout << "DLL unloaded successfully: " << dllName << std::endl << std::endl;
    }
}
```

```

    return true;
} else {
    std::cerr << "Failed to unload DLL: " << dllName << std::endl << std::endl;
    return false;
}
}

```

Can be converted to :

Code `return GetModuleHandleA(dllName.c_str()) ? FreeLibrary(hModule) : false;`

Code `int intervalMilliseconds = 500`

Why not `const`?

Again..

```

Code for (const auto& knownModule : knownModules) {
    if (moduleName.find(knownModule) != std::string::npos) {
        isKnownModule = true;
        break;
    }
}

```

Just ugly? why not <https://en.cppreference.com/w/cpp/algorithm/ranges/find> , if u already use c++, utilize latest.

Hackers - April 19, 2024, 12:06 pm

(Apr 19, 2024 12:02 PM)dasdas543534534 Wrote: [\[CITE\]](#) Cute but won't work against statically linked DLL's that are abused for side loading. (You can reverse engineer why)

As far as code:

```

Code HMODULE hModules[1024];
DWORD cbNeeded;
Always initialize variables.

```

```

Code for (int i = 0; i < moduleCount; i++) {
Ugly, will never be used in production code, use std::span and for each loop.

```

```

Code std::string moduleName = szModName;
Un-necessary copy created, if you know you don't need szModName anymore, use std::move, or
std::string_view?

```

This entire code :

```

Code bool UnloadDllByName(std::string dllName) {
    HMODULE hModule = GetModuleHandleA(dllName.c_str()); // Use ANSI version directly
    if (hModule == NULL) {
        std::cerr << "DLL not found: " << dllName << std::endl;
        return false;
    }
}

```

```

if (FreeLibrary(hModule)) {
    std::cout << "DLL unloaded successfully: " << dllName << std::endl << std::endl;
    return true;
} else {
    std::cerr << "Failed to unload DLL: " << dllName << std::endl << std::endl;
    return false;
}
}

```

Can be converted to :

```
Code return GetModuleHandleA(dllName.c_str()) ? FreeLibrary(hModule) : false;
```

Code `int intervalMilliseconds = 500`

Why not `const`?

Nice, thanks!

I am still learning C++, only started learning a few months ago so this is helpful. Much appreciated!!

Also, this is a POC not a final release. I consider "POC" as a minimum viable product - meaning if I start coding and get it working for what I am trying to do (or proof), that's my POC. I usually do not spend a single minute past "it working" to do cleanup or optimizations.

This one is a mix of my own code, some code from GPT/LLM & the aftermath of a lot of trial & errors.

nucleo_type09 - April 23, 2024, 7:21 am

Looks good, bro. You just need a bit more practice with C++, but your PoC looks nice. Even though, I would suggest using `const` where possible. Constants are allocated in the program's data segment, and using stack allocations instead of heap is preferable, as they're faster. By the way, how are you? Have you left the company you were working with? :)

Hackers - April 23, 2024, 7:42 am

(Apr 23, 2024 07:21 AM)nucleo_type09 Wrote: [\[CITE\]](#) Looks good, bro. You just need a bit more practice with C++, but your PoC looks nice. Even though, I would suggest using `const` where possible. Constants are allocated in the program's data segment, and using stack allocations instead of heap is preferable, as they're faster. By the way, how are you? Have you left the company you were working with? :)

Hey thanks for this, and yes agreed. Going back to some basics with C/C++ instead of getting too advanced too quickly so I can have a better foundation.

Not yet lol, just stressing and losing hair still!

nucleo_type09 - April 23, 2024, 8:04 am

(Apr 23, 2024 07:42 AM)nodes Wrote: [\[CITE\]](#) Hey thanks for this, and yes agreed. Going back to some basics with C/C++ instead of getting too advanced too quickly so I can have a better foundation.

Not yet lol, just stressing and losing hair still!

?? IDK why, but i somehow was expecting that, don't stress bro, why don't you work on your own program in the meanwhile? you can create a website, flood it with ads, and receive money, multiply that for 10, and you get a full salary haha

Hackers - April 28, 2024, 9:44 pm

(Apr 23, 2024 08:04 AM)nucleo_type09 Wrote: [\[CITE\]](#) ?? IDK why, but i somehow was expecting that, don't stress bro, why don't you work on your own program in the meanwhile? you can create a website, flood it with ads, and receive money, multiply that for 10, and you get a full salary haha
haha that's true & to certain degree that is what I'm doing. I have some website projects on the side ongoing that I don't talk about as much.

nucleo_type09 - April 29, 2024, 5:41 am

(Apr 28, 2024 09:44 PM)nodes Wrote: [\[CITE\]](#) haha that's true & to certain degree that is what I'm doing. I have some website projects on the side ongoing that I don't talk about as much.

do u use react or pure js for the front end? i used to use react for front end, it was fun but way too easy to digest, i typically like more complex stuff so i don't get bored, maybe that's why i completely left web-dev haha, btw, what db r u using? mongo i guess?

Hackers - November 19, 2024, 12:50 pm

(Apr 29, 2024 05:41 AM)nucleo_type09 Wrote: [\[CITE\]](#) do u use react or pure js for the front end? i used to use react for front end, it was fun but way too easy to digest, i typically like more complex stuff so i don't get bored, maybe that's why i completely left web-dev haha, btw, what db r u using? mongo i guess?

Both, as well as non-JS solutions too. Depends on the client, budget, project requirements, etc for what I would recommend.

Web dev is in fact very boring to me at this point in time haha.

Stanley - March 16, 2026, 8:20 am

I'm Stanley HF's helper bot. The OP (original poster) of this thread wants everyone to see this page and has elected to bump it to the top of the showthread listing. Since HF doesn't want a ton of LQP (Low Quality Posts) this is the best method of bumping your content. There is a small ?yte fee for each bump. Any OP can bump any of their threads on HF from the linked icon on the top of their first post.

If you haven't already read the first post of this thread we encourage you to do so.

Thanks,
Stanley