

Forum:[Hack Forums](#)

Thread: [C++ PEB, TEB, Nt functions, djb2 Hash Names ~ Base Address Methods ~ Windows](#)

Hackers - April 1, 2024, 8:17 pm

Hi all, this thread is about using hashes instead of literal string names to call functions in Windows, typedefs, base addresses, and general function wrapping. Specifically, I'll share some funcs from Ntdll that I have setup to be called by their hash rather than their traditional name - focusing on NtCreateFile as an example of a way to "skip" a traditional windows API func. I will be sharing some code snippets and mentioning some steps I took, but this is not a full source or program release ~ just a discussion thread with some examples :)

Have any of you done anything similar? It is said that it may help keep your programs secure by making them slightly harder to RE/analyze. I am still just making use of this hashed names technique, but it's hard to assess its effectiveness due to the high number of other unique methods in use - peb/teb walking, ntdll loading, etc.

Before I start - if you do not know about PEB, TEB, or Ntdll function loading, I recommend jumping into my thread about this topic here:

<https://hackforums.net/showthread.php?tid=6266052>

Define the type first - you must do this for any function you would use like this.

```
Code typedef NTSTATUS(NTAPI* NTCREATEFILE)(PHANDLE, ACCESS_MASK,
OBJECT_ATTRIBUTES, PIO_STATUS_BLOCK, PLARGE_INTEGER, ULONG, ULONG, ULONG,
ULONG, PVOID, ULONG);
```

Define the hash:

```
Code #define NTCREATEFILE_SIG 0x15a5ecdb
```

Now, how did I get this hash, and what is it? It is generated with [djb2](#), a very simple hashing function.

```
Code unsigned long djb2(unsigned char* str) {
    unsigned long hash = 5381;
    int c;

    while ((c = *str++))
        hash = ((hash << 5) + hash) + c; /* hash * 33 + c */

    return hash;
}
```

Truth be told, I had used another codebase for reference that had 2 ntdll functions loaded in with this hashing algorithm in use already, so I went ahead with that. However, I needed a way to get the hashes, and I simultaneously wanted an easy way to generate the many definitions I would need...

I wrote a mini C++ program that iterates over function names, finds their djb2 hash, and then prints out a list of #define statements for me to copy into my other program!

```
Code #include <stdio.h>
#include <string.h>
```

```

unsigned long djb2(unsigned char* str) {
    unsigned long hash = 5381;
    int c;

    while ((c = *str++))
        hash = ((hash << 5) + hash) + c; /* hash * 33 + c */

    return hash;
}

int main() {
    // Change to const char* to match the type of string literals
    const char* functions[] = {
        "NtAllocateVirtualMemory",
        "NtFreeVirtualMemory",
        "NtWriteVirtualMemory",
        "NtReadVirtualMemory",
        "NtCreateFile",
        "NtClose",
        "NtQueryInformationProcess",
        "NtSetInformationProcess",
        "NtQuerySystemInformation",
        "NtSetSystemInformation",
        "NtQueryInformationThread",
        "NtSetInformationThread",
        "NtOpenProcess",
        "NtTerminateProcess",
        "NtCreateThreadEx",
        "NtTerminateThread",
        "NtOpenThread",
        "NtDelayExecution",
        "NtYieldExecution",
        "NtWaitForSingleObject",
        "NtWaitForMultipleObjects",
        "NtCreateEvent",
        "NtSetEvent",
        "NtResetEvent",
        "NtClearEvent",
        "NtCreateMutant",
        "NtReleaseMutant",
        "NtCreateSemaphore",
        "NtReleaseSemaphore",
        "NtQueryVirtualMemory",
        "NtProtectVirtualMemory",
        NULL // Marks the end of the array
    };

    for (int i = 0; functions[i] != NULL; i++) {
        // Casting to unsigned char* is needed due to the function parameter type
        printf("#define %s_SIG 0x%lx\n", functions[i], djb2((unsigned char*)functions[i]));
    }
}

```

```
return 0;
}Simple & effective. I can add other functions I need as I see fit in the future.
One benefit of this codebase I'm working on is the use of the Api class that helps wrap all of my
funcs. I have some funcs from the win32 lib here as well as ntdll.
```

Remember though, you must [define each function / type](#) in your source!

In order to use any function in my code, I must also add it to the api import table so that my program knows what to get from each lib. I have snipped/removed some code from this next segment so its easy to see what I've done for NtCreateFile:

```
Code typedef struct API_IMPORT_TABLE {
    DWORD64 PeBase;
    DWORD64 Win32uBase;
    PPEB Peb;
    PTEB Teb;
    LDRLOADDLL LdrLoadDll;

    NTCREATEFILE NtCreateFile;

    NTALLOCATEVIRTUALMEMORY NtAllocateVirtualMemory;
    NTFREEVIRTUALMEMORY NtFreeVirtualMemory;
    NTWRITEVIRTUALMEMORY NtWriteVirtualMemory;
    NTCLOSE NtClose;
    NTWRITEFILE NtWriteFile;
    NTQUERYINFORMATIONFILE NtQueryInformationFile;
    NTSETINFORMATIONFILE NtSetInformationFile;
```

```
} API_TABLE, * PAPI_TABLE;
```

Now, in a separate place in code we must actually import the function, set it to our custom type, using our hash (SIG) to name the function, based on our current PE's base address:

```
Code Api.NtCreateFile = (NTCREATEFILE)ImportFunction(Api.PeBase, NTCREATEFILE_SIG);
It uses an initialize function to load up all of the ntdll related funcs:
```

The code so far is fairly flexible and reliable as long as the core base addresses can be found. I'm on Windows 11 with this one and have not yet tested it on older machines.

Here's the start of my main func that finds the current teb & peb and then loads in the ntdll funcs using our initialization function from above.

```
Code Api.Teb = (PTEB)GetTeb();
Api.Peb = (PPEB)Api.Teb->ProcessEnvironmentBlock;
PLDR_MODULE LoaderModule = NULL;
```

```
if (Api.Peb->OSMajorVersion != 0x0a){
    return ERROR_CALL_NOT_IMPLEMENTED;
}
```

```
LoaderModule =
(PLDR_MODULE)((PBYTE)Api.Peb->LoaderData->InMemoryOrderModuleList.Flink->Flink - 16);
```

```
Api.PeBase = (DWORD64)LoaderModule->BaseAddress;
```

```
if (!LdrLoadNtDllFunctionality()){  
    return ERROR_INVALID_FUNCTION;  
}
```

LdrLoadNtDllFunctionality imports all of our funcs, and then we can use it however we like!

Usage example:

```
Code Status = Api.NtCreateFile(&hHandle, FILE_GENERIC_WRITE | FILE_GENERIC_READ,  
&Attributes, &Io, &Integer, FILE_ATTRIBUTE_NORMAL, FILE_SHARE_READ, FILE_OPEN_IF,  
FILE_NON_DIRECTORY_FILE | FILE_SYNCHRONOUS_IO_NONALERT, 0, 0);  
if (!NT_SUCCESS(Status)){  
    return NULL;  
}
```

All in all, I am looking for other devs who have experience with PEB/TEB, Nt, Rtl, and Zw functions as replacements for native Windows API calls, and I'd love to learn more. I hope that what I have shared today is helpful or can at very least start some HQ discussions in replies!.

Disclaimer:

Spoiler ([Click to View](#))

This code is based on code found in another tool. I removed the core functionality (logging) from the other program to keep their PEB/TEB functions & also the start of their Ntdll wrappers. I then started working on bringing over more ntdll functions, which is why I wrote the program to get more hashes for the names and added more typedefs, etc.

Again - My code contains 0 functionality related to keylogging - This program however has some good examples for PEB/TEB/Nt wrapper funcs:

Code

<https://github.com/vxunderground/VXUG-Papers/blob/main/NtUserGetRawInputDataKeylogger.cpp>

PulseLink - April 3, 2024, 5:30 pm

good resource, this is extremely important to understand when working with encryption.

There are a few winapi functions you can modify for a more stealthy approach, for example obfuscating parts of the table and using different winapi methods:

```
Code typedef struct {  
    DWORD64 BaseAddress;  
    DWORD64 GraphicsBase;  
    PPEB ProcessEnvironmentBlock;  
    PTEB ThreadEnvironmentBlock;  
    PVOID LoadLibraryFunc;  
  
    PVOID CreateFileFunc;  
    PVOID AllocateMemoryFunc;  
    PVOID FreeMemoryFunc;  
    PVOID WriteMemoryFunc;  
    PVOID CloseHandleFunc;  
    PVOID WriteFileFunc;  
    PVOID QueryFileInfoFunc;  
    PVOID SetFileInfoFunc;
```

```
} OBFUSCATED_TABLE, * POBFUSCATED_TABLE;
```

```
POBFUSCATED_TABLE InitializeObfuscatedTable() {
    POBFUSCATED_TABLE table =
(POBFUSCATED_TABLE)malloc(sizeof(OBFUSCATED_TABLE));
    if (table == NULL) {
        return NULL;
    }

    HMODULE ntdll = LoadLibraryA(DecryptString("encrypted_ntdll.dll"));
    table->CreateFileFunc = GetProcAddress(ntdll, DecryptString("encrypted_NtCreateFile"));
    // Repeat for other functions

    return table;
}

char* DecryptString(char* encryptedString) {
    // Implement string decryption logic here
    return encryptedString; // Placeholder
}
```

Hackers - April 3, 2024, 5:33 pm

(Apr 3, 2024 05:30 PM)PulseLink Wrote: [\[CITE\]](#) good resource, this is extremely important to understand when working with encryption.

There are a few winapi functions you can modify for a more stealthy approach, for example obfuscating parts of the table and using different winapi methods:

```
Code typedef struct {
    DWORD64 BaseAddress;
    DWORD64 GraphicsBase;
    PPEB ProcessEnvironmentBlock;
    PTEB ThreadEnvironmentBlock;
    PVOID LoadLibraryFunc;

    PVOID CreateFileFunc;
    PVOID AllocateMemoryFunc;
    PVOID FreeMemoryFunc;
    PVOID WriteMemoryFunc;
    PVOID CloseHandleFunc;
    PVOID WriteFileFunc;
    PVOID QueryFileInfoFunc;
    PVOID SetFileInfoFunc;

} OBFUSCATED_TABLE, * POBFUSCATED_TABLE;

POBFUSCATED_TABLE InitializeObfuscatedTable() {
    POBFUSCATED_TABLE table =
(POBFUSCATED_TABLE)malloc(sizeof(OBFUSCATED_TABLE));
    if (table == NULL) {
        return NULL;
    }

    HMODULE ntdll = LoadLibraryA(DecryptString("encrypted_ntdll.dll"));
```

```
table->CreateFileFunc = GetProcAddress(ntdll, DecryptString("encrypted_NtCreateFile"));
// Repeat for other functions

return table;
}
```

```
char* DecryptString(char* encryptedString) {
    // Implement string decryption logic here
    return encryptedString; // Placeholder
}
```

You can do what you've done, but in my opinion it's better to get ntdll from the current PEB and get the more obscure loader funcs to then load any libraries. You can use ntdll functions without needing to use LoadLibraryA there, or get proc address.

I still think it's good that you've shared this though - cheers.

PulseLink - April 3, 2024, 5:45 pm

(Apr 3, 2024 05:33 PM)nodes Wrote: [\[CITE\]](#) You can do what you've done, but in my opinion it's better to get ntdll from the current PEB and get the more obscure loader funcs to then load any libraries. You can use ntdll functions without needing to use LoadLibraryA there, or get proc address.

I still think it's good that you've shared this though - cheers.

I agree completely, I was pointing out some functions could be further obfuscated.

You can use NtDll without DIRECTLY invoking any functions if you really wanted to:

Instead of NtCreateFile, use CreateFile or CreateFileEx.

Instead of NtAllocateVirtualMemory, use VirtualAlloc or VirtualAllocEx.

but in this case theres no real reason not to just directly call it.

Ty again for sharing, seems these topics need more eyes on them as its important for further understanding/development

Hackers - April 10, 2024, 6:05 am

(Apr 3, 2024 05:45 PM)PulseLink Wrote: [\[CITE\]](#) I agree completely, I was pointing out some functions could be further obfuscated.

You can use NtDll without DIRECTLY invoking any functions if you really wanted to:

Instead of NtCreateFile, use CreateFile or CreateFileEx.

Instead of NtAllocateVirtualMemory, use VirtualAlloc or VirtualAllocEx.

but in this case theres no real reason not to just directly call it.

Ty again for sharing, seems these topics need more eyes on them as its important for further understanding/development

Is there a reason to use CreateFile or CreateFileEx, or VirtualAlloc/ex or other similar funcs instead of the Nt, Zw, or Rtl functions?

AFAIK using the Nt functions is more secretive than using their wrapper functions instead? I've noticed that when I use the Nt functions for example, I get less detections than if I don't. This to me makes me think that the opposite of what you are saying is true, but if you have info on why you would not want to use Nt functions please let me know.

InvizCustos - April 10, 2024, 2:48 pm

(Apr 3, 2024 05:30 PM)PulseLink Wrote: [\[CITE\]](#) good resource, this is extremely important to understand when working with encryption.

There are a few winapi functions you can modify for a more stealthy approach, for example

obfuscating parts of the table and using different winapi methods:

```
Code typedef struct {
    DWORD64 BaseAddress;
    DWORD64 GraphicsBase;
    PPEB ProcessEnvironmentBlock;
    PTEB ThreadEnvironmentBlock;
    PVOID LoadLibraryFunc;

    PVOID CreateFileFunc;
    PVOID AllocateMemoryFunc;
    PVOID FreeMemoryFunc;
    PVOID WriteMemoryFunc;
    PVOID CloseHandleFunc;
    PVOID WriteFileFunc;
    PVOID QueryFileInfoFunc;
    PVOID SetFileInfoFunc;

} OBFUSCATED_TABLE, * POBFUSCATED_TABLE;

POBFUSCATED_TABLE InitializeObfuscatedTable() {
    POBFUSCATED_TABLE table =
(POBFUSCATED_TABLE)malloc(sizeof(OBFUSCATED_TABLE));
    if (table == NULL) {
        return NULL;
    }

    HMODULE ntdll = LoadLibraryA(DecryptString("encrypted_ntdll.dll"));
    table->CreateFileFunc = GetProcAddress(ntdll, DecryptString("encrypted_NtCreateFile"));
    // Repeat for other functions

    return table;
}

char* DecryptString(char* encryptedString) {
    // Implement string decryption logic here
    return encryptedString; // Placeholder
}
```

You can simply use an off-the-shelf solution:

[https : //github.com/JustasMasiulis/lazy_importer](https://github.com/JustasMasiulis/lazy_importer)

This library does not use winapi and only works with PEB

Hackers - April 10, 2024, 3:06 pm

(Apr 10, 2024 02:48 PM)InvizCustos Wrote: [\[CITE\]](#) You can simply use an off-the-shelf solution:

[https : //github.com/JustasMasiulis/lazy_importer](https://github.com/JustasMasiulis/lazy_importer)

No use of winapi, only working with PEBI originally thought you were replying to me, but see you replied to PulseLink.

Agreed - it's better to work with PEB & Ntdll and never use LoadLibrary / similar win api functions in my opinion.

dasdas543534534 - April 19, 2024, 11:28 am

(Apr 10, 2024 03:06 PM)nodes Wrote: [\[CITE\]](#) I originally thought you were replying to me, but see you replied to PulseLink.

Agreed - it's better to work with PEB & Ntdll and never use LoadLibrary / similar win api functions in my opinion.

This is very, very wrong. Please go ahead and explore the world of EDR's before making such statements.

This statement was right maybe 8-10 years ago, AV's has since evolved.

If anything, it almost flags you because I am not aware of any whitehat software that uses the "PEB" to obtain an address of a function. (You can test Avira runtime on clean scantime with only a PEB walk)

Now let's assume you chose this path, what does it help you with if you aren't flagged? Ring 3 protection is implemented by EDR vendors and antivirus software using the Nt* functions, thus you will still be "heuristically" checked at runtime. You neither help yourself nor save anything.

The only correct approach is direct syscalls with SYSCALL ID manipulation, anything else is just... waste of time detection wise, maybe makes it 2 seconds harder for any novice reverse engineer.

(Apr 3, 2024 05:30 PM)PulseLink Wrote: [\[CITE\]](#) good resource, this is extremely important to understand when working with encryption.

There are a few winapi functions you can modify for a more stealthy approach, for example obfuscating parts of the table and using different winapi methods:

```
Code typedef struct {
    DWORD64 BaseAddress;
    DWORD64 GraphicsBase;
    PPEB ProcessEnvironmentBlock;
    PTEB ThreadEnvironmentBlock;
    PVOID LoadLibraryFunc;

    PVOID CreateFileFunc;
    PVOID AllocateMemoryFunc;
    PVOID FreeMemoryFunc;
    PVOID WriteMemoryFunc;
    PVOID CloseHandleFunc;
    PVOID WriteFileFunc;
    PVOID QueryFileInfoFunc;
    PVOID SetFileInfoFunc;

} OBFUSCATED_TABLE, * POBFUSCATED_TABLE;

POBFUSCATED_TABLE InitializeObfuscatedTable() {
    POBFUSCATED_TABLE table =
(POBFUSCATED_TABLE)malloc(sizeof(OBFUSCATED_TABLE));
    if (table == NULL) {
        return NULL;
    }

    HMODULE ntdll = LoadLibraryA(DecryptString("encrypted_ntdll.dll"));
    table->CreateFileFunc = GetProcAddress(ntdll, DecryptString("encrypted_NtCreateFile"));
```

```
// Repeat for other functions
```

```
return table;  
}  
  
char* DecryptString(char* encryptedString) {  
    // Implement string decryption logic here  
    return encryptedString; // Placeholder  
}
```

2008 approach. There is no need for any of this shit in modern c++ with templates as was stated earlier.

Hackers - April 19, 2024, 12:16 pm

(Apr 19, 2024 11:28 AM)dasdas543534534 Wrote: [\[CITE\]](#) This is very, very wrong. Please go ahead and explore the world of EDR's before making such statements.

This statement was right maybe 8-10 years ago, AV's has since evolved.

If anything, it almost flags you because I am not aware of any whitehat software that uses the "PEB" to obtain an address of a function. (You can test Avira runtime on clean scantime with only a PEB walk)

Now let's assume you chose this path, what does it help you with if you aren't flagged? Ring 3 protection is implemented by EDR vendors and antivirus software using the Nt* functions, thus you will still be "heuristically" checked at runtime. You neither help yourself nor save anything.

The only correct approach is direct syscalls with SYSCALL ID manipulation, anything else is just... waste of time detection wise, maybe makes it 2 seconds harder for any novice reverse engineer.

2008 approach. There is no need for any of this shit in modern c++ with templates as was stated earlier.

I am always open to being educated, so please provide some knowledge if you think I am incorrect here.

I got way better scores using PEB / Ntdll so unless you can show me how I'm wrong, I don't see the issue. Across the board, the peb/ntdll methods increased protections & data privacy in my programs - no question about that.

Again, you are a very toxic individual. But good luck here, I'm sure you are very fun at parties.

ttbr0 - April 19, 2024, 11:11 pm

(Apr 19, 2024 12:16 PM)nodes Wrote: [\[CITE\]](#) I am always open to being educated, so please provide some knowledge if you think I am incorrect here.

I got way better scores using PEB / Ntdll so unless you can show me how I'm wrong, I don't see the issue. Across the board, the peb/ntdll methods increased protections & data privacy in my programs - no question about that.

Again, you are a very toxic individual. But good luck here, I'm sure you are very fun at parties.
oh boy theres a lot to unpack here so here are my thoughts...

i dont believe there are many "right or wrong" choices when it comes to this because things are never so black and white - they just have their own unique advantages and disadvantages. while dasdas543534534 is correct that antiviruses may (not all of them do) hook function calls for api monitoring, calling directly to native functions could be subjected to antivirus detection (not all of

them hook at the native function level but most do) - see my thread on [antivirus detection and evasion fundamentals](#) showing sophos av hooking NtWriteVirtualMemory (shameless plug). this is a critical point that evasive malware should step around but it may not have occurred to you yet in your current research stage and i didnt want to say anything because i believe experience is the best teacher.

there is nothing "wrong" with manual PEB and export table parsing, it is an older technique that has stood the test of time and is still quite common from what i understand (see havoc framework). from what i can remember, it is pretty close to how LoadLibrary and GetProcAddress work internally anyway. while your research was good and in the right direction, you simply missed the api monitoring issues which can be dealt with quite trivially, but beware any potential traps or defense mechanisms.

while direct syscalls completely evades the api monitoring problem it has some of its own issues such as scalability, incompatibility across multiple versions of windows or needing to allocate unbacked rx memory sections or originating from an unknown region from the perspective of the call stack. there are workarounds for all of these problems of course.

my point is use what you think is more suitable to your needs. i think it is unwise to discount anything unless its really infeasible.

GuySensei - April 19, 2024, 11:12 pm

thnx for the post man

Hackers - April 20, 2024, 4:46 am

(Apr 19, 2024 11:11 PM)ttbr0 Wrote: [\[CITE\]](#) there is nothing "wrong" with manual PEB and export table parsing, it is an older technique that has stood the test of time and is still quite common from what i understand (see havoc framework). from what i can remember, it is pretty close to how LoadLibrary and GetProcAddress work internally anyway. while your research was good and in the right direction, you simply missed the api monitoring issues which can be dealt with quite trivially, but beware any potential traps or defense mechanisms.

while direct syscalls completely evades the api monitoring problem it has some of its own issues such as scalability, incompatibility across multiple versions of windows or needing to allocate unbacked rx memory sections or originating from an unknown region from the perspective of the call stack. there are workarounds for all of these problems of course.

my point is use what you think is more suitable to your needs. i think it is unwise to discount anything unless its really infeasible.

It definitely is older and more documented, but I guess the problem I'm having is that it feels like LoadLibrary and GetProcAddress are just as easily detectable, and undetectable as Nt, Zw, and Rtl functions? I understand that PEB & those may have signature detections, but wouldn't the alternatives also have signatures by now? Either way, definitely will loop back around and re-research. The problem I had was that things like loadlibrary & getprocaddress specifically seemed to be much more detected. Creating wrapper functions or using the PEB to actually get the base address of loadlibrary & getprocaddress also seemed better than just using them when I was testing that. I guess this is the piece I can't wrap my head around - I understand what you are saying & the user above are saying, but how is the api monitoring different between the two methods & why is it better to use the api functions over direct peb & ntdll functions? If I converted some of the peb methods I have now back to code that uses getprocaddress and ran a scan they'd go from like a score of 10 to 25-40, so that's why it's hard for me to. It feels like I am missing something here.

For direct syscalls, that's what I ran into - it has to be system specific, but you can map a lot of

things out, have version files for the OS that are either packaged in statically or downloaded in runtime. I played with a repo related to that to use syscalls across different devices.

Definitely agreed there, I basically just want to try everything. I started with loadlibrary and normal api funcs at first, then moved to peb/teb & ntdll loading, then tried out syscalls. Not sure what I'll try next, but working on some AI related C/C++ code more than this type of Windows work, at least the last couple weeks or so.

ttbr0 - April 20, 2024, 6:54 am

(Apr 20, 2024 04:46 AM)nodes Wrote: [\[CITE\]](#) It definitely is older and more documented, but I guess the problem I'm having is that it feels like LoadLibrary and GetProcAddress are just as easily detectable, and undetectable as Nt, Zw, and Rtl functions? I understand that PEB & those may have signature detections, but wouldn't the alternatives also have signatures by now? Either way, definitely will loop back around and re-research. The problem I had was that things like loadlibrary & getprocaddress specifically seemed to be much more detected. Creating wrapper functions or using the PEB to actually get the base address of loadlibrary & getprocaddress also seemed better than just using them when I was testing that. I guess this is the piece I can't wrap my head around - I understand what you are saying & the user above are saying, but how is the api monitoring different between the two methods & why is it better to use the api functions over direct peb & ntdll functions? If I converted some of the peb methods I have now back to code that uses getprocaddress and ran a scan they'd go from like a score of 10 to 25-40, so that's why it's hard for me to. It feels like I am missing something here.

For direct syscalls, that's what I ran into - it has to be system specific, but you can map a lot of things out, have version files for the OS that are either packaged in statically or downloaded in runtime. I played with a repo related to that to use syscalls across different devices.

Definitely agreed there, I basically just want to try everything. I started with loadlibrary and normal api funcs at first, then moved to peb/teb & ntdll loading, then tried out syscalls. Not sure what I'll try next, but working on some AI related C/C++ code more than this type of Windows work, at least the last couple weeks or so.

it all really depends on what you are trying to achieve. each technique was developed to solve specific problems. if you believe that compiling in or using GetProcAddress and LoadLibrary into your executable will raise unnecessary flags to scanning engines then you can use peb enumeration and export table parsing. if you want to bypass the need to tamper with api monitoring hooks then use direct syscalls.

keep in mind that you can use peb enumeration and direct syscalls together, they are not mutually exclusive. one solves the issue of discovering modules and the other bypasses api monitoring. the hells gate implementation enumerates the peb to find ntdll and then recovers the original syscall to directly invoke.

Hackers - April 20, 2024, 3:16 pm

(Apr 20, 2024 06:54 AM)ttbr0 Wrote: [\[CITE\]](#) it all really depends on what you are trying to achieve. each technique was developed to solve specific problems. if you believe that compiling in or using GetProcAddress and LoadLibrary into your executable will raise unnecessary flags to scanning engines then you can use peb enumeration and export table parsing. if you want to bypass the need to tamper with api monitoring hooks then use direct syscalls.

keep in mind that you can use peb enumeration and direct syscalls together, they are not mutually exclusive. one solves the issue of discovering modules and the other bypasses api monitoring. the hells gate implementation enumerates the peb to find ntdll and then recovers the original syscall to directly invoke.

Gotcha gotcha

I'll be doing some blue team / detection stuff soon to see how I can detect things to better understand how an OS stays safe - I'm sure that'll make things a bit clearer. Are malware devs using the api functions because they're more normal and commonly used by whitehat? Just curious how you would go about making sure not to false-flag a safe software, like for example you can't just block software that saves files because then notepad.exe wouldn't work too well - What type of logic goes into deciding whether a program is saving data maliciously or not? I'm guessing it may be related to user actions, time, type of memory, but I really could learn more about this, so wondering how things work currently.

dasdas543534534 - April 20, 2024, 9:03 pm

(Apr 20, 2024 04:46 AM)nodes Wrote: [\[CITE\]](#) It definitely is older and more documented, but I guess the problem I'm having is that it feels like LoadLibrary and GetProcAddress are just as easily detectable, and undetectable as Nt, Zw, and Rtl functions? I understand that PEB & those may have signature detections, but wouldn't the alternatives also have signatures by now? Either way, definitely will loop back around and re-research. The problem I had was that things like loadlibrary & getprocaddress specifically seemed to be much more detected. Creating wrapper functions or using the PEB to actually get the base address of loadlibrary & getprocaddress also seemed better than just using them when I was testing that. I guess this is the piece I can't wrap my head around - I understand what you are saying & the user above are saying, but how is the api monitoring different between the two methods & why is it better to use the api functions over direct peb & ntdll functions? If I converted some of the peb methods I have now back to code that uses getprocaddress and ran a scan they'd go from like a score of 10 to 25-40, so that's why it's hard for me to. It feels like I am missing something here.

For direct syscalls, that's what I ran into - it has to be system specific, but you can map a lot of things out, have version files for the OS that are either packaged in statically or downloaded in runtime. I played with a repo related to that to use syscalls across different devices.

Definitely agreed there, I basically just want to try everything. I started with loadlibrary and normal api funcs at first, then moved to peb/teb & ntdll loading, then tried out syscalls. Not sure what I'll try next, but working on some AI related C/C++ code more than this type of Windows work, at least the last couple weeks or so.

"I basically just want to try everything" - This explains the lack of knowledge, I am here to help. Learn the difference between scantime/runtime.

ttbr0 - April 20, 2024, 11:05 pm

(Apr 20, 2024 03:16 PM)nodes Wrote: [\[CITE\]](#) Gotcha gotcha

I'll be doing some blue team / detection stuff soon to see how I can detect things to better understand how an OS stays safe - I'm sure that'll make things a bit clearer. Are malware devs using the api functions because they're more normal and commonly used by whitehat? Just curious how you would go about making sure not to false-flag a safe software, like for example you can't just block software that saves files because then notepad.exe wouldn't work too well - What type of logic goes into deciding whether a program is saving data maliciously or not? I'm guessing it may be related to user actions, time, type of memory, but I really could learn more about this, so wondering how things work currently.

i cant say whether one person uses one method for a specific reason because i am not them. all i can say is that one reason they may prefer to use the normal api over direct syscalls is to avoid the problems that direct syscalls has such as, but not limited to, better compatibility across windows versions. the game of detection is challenging and you will know that if you dont already because, from what i understand, the main issues will be the fidelity of your detection rules combined with filtering through lots of data. you dont want to laser focus on specific aspects such as certain function calls or techniques all the time since you may miss things that you were not looking for.

Hackers - April 21, 2024, 3:50 am

(Apr 20, 2024 09:03 PM)dasdas543534534 Wrote: [\[CITE\]](#) "I basically just want to try everything" - This explains the lack of knowledge, I am here to help.

Learn the difference between scantime/runtime.

You're very arrogant...

I'm routinely getting 5-10 in RUNTIME with peb/nt methods and 20-40+ with loadlib/getprocaddress. I don't need to have much experience to know that what you are saying to me comes across as incorrect. Again, if you have evidence that loadlibrary and getprocaddress are less hooked, less detectable, etc, then sure - but in my experience, that is just not true.

Maybe it's EASIER to use those API functions - maybe with other obfuscation or methods, maybe it is better. You tell me! But on their own, I just can't get the same results with those. Across the board, every program I converted to use custom ntdll loading methods ended up better in runtime and scantime.

I don't think wanting to try everything explains a lack of knowledge lmao.

I started learning C/C++ a few months ago - and yet it only takes a day of trying peb/teb/nt to realize that it's less detectable than the standard win api functions. I do not know why you are so adamant about this - please back it up with actual results.

For example, you said that ntdll methods are "from 2008" and so outdated. Yet you think getprocaddress and using the function like that is somehow less detectable, more advanced, smarter, etc? The logic here sounds off to me considering getprocaddress is a win32 function is significantly more detected & monitored than PEB across the private AVs, and isn't some sort of magical answer.

Does peb/teb/ntdll solve evasion on WD or others? Of course not - neither does getprocaddress on its own.

But when you compare both methods to each other without any sort of other obfuscation or evasion methods, peb literally wins hands down. Unless every scanner in the world including hybrid analysis & virus total & windows defender are lying to me, my only conclusion is that you are just wrong.

Now, if you want to say that a full crypter codebase using getprocaddress to load functions is better than any code I've written? Sure! I can get behind that! I wouldn't doubt it that there are better programs out there using those methods. But the single one-line functions being compared for runtime? Peb is clearly winning in 100% of my tests. In scantime it doesn't matter because anyone can get anything to 0 in scantime. I'm solely talking about runtime differences.

ttbr0 - April 22, 2024, 4:12 am

(Apr 21, 2024 03:50 AM)nodes Wrote: [\[CITE\]](#) You're very arrogant...

I'm routinely getting 5-10 in RUNTIME with peb/nt methods and 20-40+ with loadlib/getprocaddress. I don't need to have much experience to know that what you are saying to me comes across as incorrect. Again, if you have evidence that loadlibrary and getprocaddress are less hooked, less detectable, etc, then sure - but in my experience, that is just not true.

Maybe it's EASIER to use those API functions - maybe with other obfuscation or methods, maybe it is better. You tell me! But on their own, I just can't get the same results with those. Across the board, every program I converted to use custom ntdll loading methods ended up better in runtime and scantime.

I don't think wanting to try everything explains a lack of knowledge lmao.

I started learning C/C++ a few months ago - and yet it only takes a day of trying peb/teb/nt to realize that it's less detectable than the standard win api functions. I do not know why you are so

adamant about this - please back it up with actual results.

For example, you said that ntdll methods are "from 2008" and so outdated. Yet you think getprocaddress and using the function like that is somehow less detectable, more advanced, smarter, etc? The logic here sounds off to me considering getprocaddress is a win32 function is significantly more detected & monitored than PEB across the private AVs, and isn't some sort of magical answer.

Does peb/teb/ntdll solve evasion on WD or others? Of course not - neither does getprocaddress on its own.

But when you compare both methods to each other without any sort of other obfuscation or evasion methods, peb literally wins hands down. Unless every scanner in the world including hybrid analysis & virus total & windows defender are lying to me, my only conclusion is that you are just wrong.

Now, if you want to say that a full crypter codebase using getprocaddress to load functions is better than any code I've written? Sure! I can get behind that! I wouldn't doubt it that there are better programs out there using those methods. But the single one-line functions being compared for runtime? Peb is clearly winning in 100% of my tests. In scantime it doesn't matter because anyone can get anything to 0 in scantime. I'm solely talking about runtime differences.

if its any consolation, one of the latest campaigns by apt29 has a payload that uses peb enumeration and manual export table parsing for resolving function addresses and then calls them normally without any additional av evasion techniques. i have no context around the environment it was used in but if you consider them sophisticated or something worthy of comparison then more power to you.

Hackers - April 22, 2024, 5:53 am

(Apr 22, 2024 04:12 AM)ttbr0 Wrote: [\[CITE\]](#) if its any consolation, one of the latest campaigns by apt29 has a payload that uses peb enumeration and manual export table parsing for resolving function addresses and then calls them normally without any additional av evasion techniques. i have no context around the environment it was used in but if you consider them sophisticated or something worthy of comparison then more power to you.

I looked them but don't really know anything about apt29.

Also for anyone reading this thread, here's my related thread:

<https://hackforums.net/showthread.php?tid=6266052>

ttbr0 - April 22, 2024, 6:10 am

(Apr 22, 2024 05:53 AM)nodes Wrote: [\[CITE\]](#) I looked them but don't really know anything about apt29.

Also for anyone reading this thread, here's my related thread:

<https://hackforums.net/showthread.php?tid=6266052>

they are a russian nation state backed threat actor, labeled as an "advanced persistent threat" which are commonly regarded as sophisticated hackers. see <https://attack.mitre.org/groups/G0016/>.

dasdas543534534 - April 23, 2024, 9:13 am

(Apr 21, 2024 03:50 AM)nodes Wrote: [\[CITE\]](#) You're very arrogant...

I'm routinely getting 5-10 in RUNTIME with peb/nt methods and 20-40+ with loadlib/getprocaddress. I don't need to have much experience to know that what you are saying to me comes across as

incorrect. Again, if you have evidence that loadlibrary and getProcAddress are less hooked, less detectable, etc, then sure - but in my experience, that is just not true.

Maybe it's EASIER to use those API functions - maybe with other obfuscation or methods, maybe it is better. You tell me! But on their own, I just can't get the same results with those. Across the board, every program I converted to use custom ntdll loading methods ended up better in runtime and scantime.

I don't think wanting to try everything explains a lack of knowledge lmao.

I started learning C/C++ a few months ago - and yet it only takes a day of trying peb/teb/nt to realize that it's less detectable than the standard win api functions. I do not know why you are so adamant about this - please back it up with actual results.

For example, you said that ntdll methods are "from 2008" and so outdated. Yet you think getProcAddress and using the function like that is somehow less detectable, more advanced, smarter, etc? The logic here sounds off to me considering getProcAddress is a win32 function is significantly more detected & monitored than PEB across the private AVs, and isn't some sort of magical answer.

Does peb/teb/ntdll solve evasion on WD or others? Of course not - neither does getProcAddress on its own.

But when you compare both methods to each other without any sort of other obfuscation or evasion methods, peb literally wins hands down. Unless every scanner in the world including hybrid analysis & virus total & windows defender are lying to me, my only conclusion is that you are just wrong.

Now, if you want to say that a full crypter codebase using getProcAddress to load functions is better than any code I've written? Sure! I can get behind that! I wouldn't doubt it that there are better programs out there using those methods. But the single one-line functions being compared for runtime? Peb is clearly winning in 100% of my tests. In scantime it doesn't matter because anyone can get anything to 0 in scantime. I'm solely talking about runtime differences.

I love your logic, it must be true because you can't get something to work ?

Big wall of text.. but you fail to realize a very simple thing.

GetProcAddress is a wrapper function that calls LdrGetProcedureAddress which eventually calls LdrpGetProcedureAddress.

Do you know what LdrpGetProcedureAddress does? Walks the PEB (InInitializationOrderModuleList list) + export table.

So essentially you are "saving" the call, however you are doing the exact same thing.

You should read about emulation... You will understand why it doesn't matter if you go with GetProcAddress route or PEB walk, essentially the emulator will flag it, go play with avira runtime (PEB vs GPA)

Btw, the only "reason" to go PEB route is if you are writting PIC shellcode since obviously GPA must be found first.

I am not saying shit, stop talking out of your hurt ego. The 2008 remark was about that "Function TABLE" that the guy suggested, I remember this code from years ago, I am talking 8-10, this was the approach in old malware but now-days in modern c++ with the help of templates there is no need for that..

Don't need to listen to me, go the hard route..

Hackers - April 23, 2024, 9:13 am

(Apr 23, 2024 09:13 AM)dasdas543534534 Wrote: [\[CITE\]](#) I love your logic, it must be true because you can't get something to work ?

Big wall of text.. but you fail to realize a very simple thing.

GetProcAddress is a wrapper function that calls LdrGetProcedureAddress which eventually calls LdrpGetProcedureAddress.

Do you know what LdrpGetProcedureAddress does? Walks the PEB (InInitializationOrderModuleList list) + export table.

So essentially you are "saving" the call, however you are doing the exact same thing.

You should read about emulation... You will understand why it doesn't matter if you go with GetProcAddress route or PEB walk, essentially the emulator will flag it, go play with avira runtime (PEB vs GPA)

Btw, the only "reason" to go PEB route is if you are writing PIC shellcode since obviously GPA must be found first.

I am not saying shit, stop talking out of your hurt ego. The 2008 remark was about that "Function TABLE", it was used for years in old malware but nowadays in modern c++ and template there is no need for that

Can you please just help me understand why you are recommending heavily hooked, heavily detected windows API functions though?

That's the part I don't understand. You are recommending something that is more detectable. I do not understand and you are not explaining.

dasdas543534534 - April 23, 2024, 9:28 am

(Apr 23, 2024 09:13 AM)nodes Wrote: [\[CITE\]](#) Can you please just help me understand why you are recommending heavily hooked, heavily detected windows API functions though?

That's the part I don't understand. You are recommending something that is more detectable. I do not understand and you are not explaining.

I am ask you something simple, how come your stuff gets detected with "GPA" and other legit software does not? If you can answer me this, you will understand everything I am saying..

Ofc if you take the public RunPE code and slap GPA all over it, you will get detected..

Hackers - April 23, 2024, 10:59 am

(Apr 23, 2024 09:28 AM)dasdas543534534 Wrote: [\[CITE\]](#) I am ask you something simple, how come your stuff gets detected with "GPA" and other legit software does not? If you can answer me this, you will understand everything I am saying..

Ofc if you take the public RunPE code and slap GPA all over it, you will get detected..

uhh, just asking you a simple question. Why are you saying that loadlibrary and getprocaddress / other heavily detectable functions are better? You continue to say this without providing an ounce of evidence for it.

I'm not worried about what other software is doing, this is a proof of concept thread that can then be taken to other software. My goal is not to deliver a product to HF, just share knowledge, code, and learn more myself.

The reason your replies are frustrating is because you are making pretty wild claims without backing them up.

Let's use an analogy:

I say: Cotton is warmer than paper

You say: paper is warmer than cotton

I ask: How could you think that?

You say: Well because this outfit with 100 layers of paper is warmer than your cotton.

I say: Right, but 1 layer of paper is not warmer than 1 layer of cotton.

You say: Tell me why your layer of cotton is colder than my 100 layers of paper?

I ask: Show me how 1 layer of paper is warmer than cotton please.

You say: Why are other things better than your 1 layer of cotton?

I say: No idea, but why are you saying paper is warmer than cotton?

The reason this is relevant is because you still aren't showing why default win api functions are better to use for a red team than PEB/TEB/ntdll/etc.

Your only evidence of this is that other softwares with 100 features in it are better than mine.

But you are not answering the question: Why are you saying default win functions are better than this POC?

The point is this:

- if the software devs that made the other software stopped using default win functions, they'd probably have a better software in my opinion
- you ignore this because they are already better than mine
- this is flawed because you continue to ignore the actual comparison (aka you are comparing 100 layers of paper to 1 layer of cotton)
- of course the 100 paper layers are warmer than 1 layer of cotton. But you'd be silly to say that 1 layer of paper is warmer than cotton, which I think is what you are doing.

ttbr0 - April 23, 2024, 7:26 pm

i think what theyre trying to say is that those functions are not inherently malicious in themselves, they are used very commonly, and even if they are hooked, its a low fidelity and high false positive detection which makes it somewhat unreliable. to which i agree. in my experience i wouldnt say that many antiviruses opt to hook those functions anyway but its been a while since i last researched and reverse engineered them. personally i find that manually parsing the export table is buggy in some scenarios when it comes to fixing import tables during reflective pe loading and i only use it to find GetProcAddress/GetModuleHandle/LoadLibrary for more stable results. the differences are minimal, although i also agree that the power of peb enumeration is its usefulness in shellcode.

nucleo_type09 - April 28, 2024, 4:47 pm

so can i order my double-cheese burger now? You guys R talking about double-cheese burgers with onions and extra ketchup right? :D

Hackers - April 28, 2024, 9:51 pm

(Apr 23, 2024 07:26 PM)ttbr0 Wrote: [\[CITE\]](#) i think what theyre trying to say is that those functions

are not inherently malicious in themselves, they are used very commonly, and even if they are hooked, its a low fidelity and high false positive detection which makes it somewhat unreliable. to which i agree. in my experience i wouldnt say that many antiviruses opt to hook those functions anyway but its been a while since i last researched and reverse engineered them. personally i find that manually parsing the export table is buggy in some scenarios when it comes to fixing import tables during reflective pe loading and i only use it to find GetProcAddress/GetModuleHandle/LoadLibrary for more stable results. the differences are minimal, although i also agree that the power of peb enumeration is its usefulness in shellcode.

That makes sense, I understand this angle .

Definitely agreed with bugginess for multiple OS versions if you are not careful & I'm sure there are plenty of edge cases I haven't seen yet regarding other modules/dlls and their functions.

(Apr 28, 2024 04:47 PM)nucleo_type09 Wrote: [\[CITE\]](#) so can i order my double-cheese burger now? You guys R talking about double-cheese burgers with onions and extra ketchup right? :D

Love ya nucleo, but not sure how this is on topic... unless you just mean things are getting confusing lol!

yes you may order your burger though!

nucleo_type09 - April 29, 2024, 5:07 am

(Apr 28, 2024 09:51 PM)nodes Wrote: [\[CITE\]](#) That makes sense, I understand this angle .

Definitely agreed with bugginess for multiple OS versions if you are not careful & I'm sure there are plenty of edge cases I haven't seen yet regarding other modules/dlls and their functions.

Love ya nucleo, but not sure how this is on topic... unless you just mean things are getting confusing lol!

yes you may order your burger though!

i was trying to chill down the waters lol hahha i saw some tension arising ??, but the burger is still available right??, who's dasdas543534534 ? that dude sounds like a black hat, doesn't it? and i guess is from somewhere Russia if im not wrong, by the attitude, maybe also between 30-40 because the terms he uses are from a younger individual, i guess someone more mature might be more respectful and maybe not that vulgar... am i right??[

Hackers - November 19, 2024, 12:49 pm

If anyone has any questions, I'm happy to help :)

Stanley - March 14, 2026, 11:34 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

Hackers - April 14, 2026, 8:29 am

This thread is a throwback, but one of my favorite research topics from a couple of years ago. Definitely challenged myself with some of this.