

Forum:[Hack Forums](#)

Thread: [Encrypting code at Function level](#)

Tentco - December 20, 2023, 6:51 am

Basic Idea

The basic idea of this trick is to create an encrypted function, where only the current function being executed can be decrypted. This is done by modifying the .text section of the executable to contain the encrypted code. Then, before executing each function, we decrypt it. After executing the function, we re-encrypt it to keep the rest of the program encrypted.

Specific Implementation

To implement this trick, we can use the gcc -finstrument-functions option. This option causes a special function (__cyg_profile_func_enter and __cyg_profile_func_exit) to be called before and after each function call, respectively. By using these functions, we can modify the code to perform the encryption and decryption tasks.

Basically something like this,

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

```
#include <stdlib.h>
```

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

```
#include <windows.h>
```

```
// Hook function
```

```
void __cyg_profile_func_enter(void *this_fn, void *call_site) {  
}
```

```
void __cyg_profile_func_exit(void *this_fn, void *call_site) {  
}
```

```
// Main function
```

```
int main() {  
}
```

```
Code int getfunc(void* func, int mark) {
```

```
    int i = 0;
```

```
    while (1) {
```

```
        unsigned char currentByte = *(unsigned char*)(func + i);
```

```
        if (currentByte == 0xe8) {
```

```
            int relativeOffset = *(int*)(func + i + 1);
```

```
            int targetAddress = (int)func + i + relativeOffset + 5;
```

```
            if (targetAddress == mark) {
```

```
                break;
```

```
            }
```

```
        }
```

```
        i++;
```

```

    }

    return i;
}

```

This allows for an easy calculation of the size of the function. We look for the E8 [offset], or call in x86 asm, opcodes, allowing us to trace and find the absolute address of the resultant location.

```

Code int keylookup(void* func) {
    int j = 0;
    while (keys[j] != 0 && keys[j] != (int)func)
        j += 2;
    if (keys[j] == 0)
        keys[j] = (int)func;
    return j;
}

```

```

void encfn(void* func, int i, const char* key) {
    DWORD a;
    unsigned char* code = func;
    VirtualProtect(code, i, PAGE_EXECUTE_READWRITE, &a);
    for (int j = 0; j < i; j++)
        code[j] ^= key[j % 4];
    VirtualProtect(code, i, PAGE_EXECUTE_READ, &a);
}

```

In the code, keys is an array that holds the encrypted keys for the functions. keylookup is a function that searches for a function's key in the keys array. If it does not find a key, it generates one.

encfn is a function that performs XOR encryption and decryption on the function code. It takes a function pointer, an integer that represents the length of the function code, and a string that serves as the key for the encryption.

So, the basic idea is -

The function `__cyg_profile_func_enter` is called before each function call. In this function, we first decrypt the current function (`__cyg_profile_func_enter` is actually a pointer to the address of the decrypted function). Then, we add the current function to the callstack and call the function using the `__cyg_profile_func_exit` function pointer.

The function `__cyg_profile_func_exit` is called after each function call. In this function, we first call the function using the `__cyg_profile_func_enter` function pointer. Then, we remove the current function from the callstack and re-encrypt the current function.

```

Code int exitMark, enterMark;

```

```

int fakeEntryPoint();// defined somewhere

```

```

int main() {
    srand(time(NULL));
    memset(keysArray, 0, 100 * sizeof(int));
    memset(callstackArray, 0, 100 * sizeof(void*));

    exitMark = ((int)__cyg_profile_func_exit);
    enterMark = ((int)__cyg_profile_func_enter);
}

```

```
printf("%d", fakeEntryPoint()); // Rename original main to avoid conflict
return 0;
}
```

And our address are called successfully.

osn64698 - December 20, 2023, 10:00 am

Good job, I have something like this implemented within my own PIC shellcode crypter.

Upon entry there is decryption and upon exit I have re-encryption with a new KEY, although my shit does not rely on some GCC function.

Also one should consider adding multi-threaded support.

Hackers - December 21, 2023, 6:52 pm

This looks interesting to me but wondering if you can possibly get the address of a function or variable held in memory *from a different program/process*. For example, can ProgramB setup a hook on a function in ProgramA, or would they need to be compiled together? Was mostly curious as if this is possible, I could theoretically listen for events / function use in a few different programs and have my program respond with its own logic. For that use case, it wouldn't be to encrypt and decrypt itself.

For the self decrypting, that's to basically avoid detection on runtime? So for example, you could drop this in, successfully go undetected, then protect yourself from runtime detection by using hooks? If this is true, my main question is what prevents the AV or similar from detecting the decrypted version when it runs before it is re encrypted?

osn64698 - December 22, 2023, 6:32 am

(Dec 21, 2023 07:52 PM)nodes Wrote: [\[CITE\]](#) This looks interesting to me but wondering if you can possibly get the address of a function or variable held in memory *from a different program/process*. For example, can ProgramB setup a hook on a function in ProgramA, or would they need to be compiled together? Was mostly curious as if this is possible, I could theoretically listen for events / function use in a few different programs and have my program respond with its own logic. For that use case, it wouldn't be to encrypt and decrypt itself.

For the self decrypting, that's to basically avoid detection on runtime? So for example, you could drop this in, successfully go undetected, then protect yourself from runtime detection by using hooks? If this is true, my main question is what prevents the AV or similar from detecting the decrypted version when it runs before it is re encrypted?

This is good for 3 things:

1. Emulation engines stalling - as the instructions get decoded on the fly, you can't read the correct OP codes hence it must wait for it to end. You effectively kill the emulator by making it to wait (Brute forcing the key) - AV engines care about how much they impact your computer, hence there is a limit for such an attempt. (X cpu cycles)
2. Memory scans that occur once in a while, when 99% of your code is encrypted and decrypted on the fly, resources stripped / headers removed/ sections wiped.
3. Manual analysis is harder, obviously you can also insert some opaque predicates/junk blocks and other nice stuff.

You should also make sure that you support multi-threading.

Hackers - December 22, 2023, 8:47 am

(Dec 22, 2023 07:32 AM)osn64698 Wrote: [\[CITE\]](#) This is good for 3 things:

1. Emulation engines stalling - as the instructions get decoded on the fly, you can't read the correct OP codes hence it must wait for it to end. You effectively kill the emulator by making it to wait (Brute forcing the key) - AV engines care about how much they impact your computer, hence there is a limit for such an attempt. (X cpu cycles)

2. Memory scans that occur once in a while, when 99% of your code is encrypted and decrypted on the fly, resources stripped / headers removed/ sections wiped.

3. Manual analysis is harder, obviously you can also insert some opaque predicates/junk blocks and other nice stuff.

You should also make sure that you support multi-threading.

I did notice the first issue I noticed this morning was having to support asynchronous functions to prevent blocking and also an issue with my program stealing focus. Thank you for the list here, I am going to have plenty of time on a flight today to think about these concepts and more. I can't say I understand #1 on your list right now as much, but #2 and 3 are familiar to me.

NyQuil - December 23, 2023, 2:21 am

encfn is a function that performs XOR encryption and decryption on the function code. // nice so it does nothing really.

Tentco - December 23, 2023, 3:18 am

(Dec 23, 2023 03:21 AM)NyQuil Wrote: [\[CITE\]](#) encfn is a function that performs XOR encryption and decryption on the function code. // nice so it does nothing really.

You can use encfn for both Encryption/Decryption of XOR, state your question clearly.

redbaron - December 23, 2023, 10:00 am

is there anything like this for MSVC?

Stanley - December 25, 2023, 10:20 pm

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

HiddenEyE - December 26, 2023, 12:24 am

encrypting a function and decrypting the functions is good but you are still accessing the same address

Tentco - December 26, 2023, 12:28 am

(Dec 26, 2023 01:24 AM)HiddenEyE Wrote: [\[CITE\]](#) encrypting a function and decrypting the functions is good but you are still accessing the same address

The idea is that the program is encrypting its own code while it's running, so that any memory dump will only show encrypted code, making it difficult to analyze or reverse engineer the program, this thread is unrelated to AV Evasion + All functions are encrypted using a different key and only the

currently running function is decrypted.

`__cyg_profile_func_enter()` is called, which means the control flow is about to enter a function.

Before entering the function, the code of the function is checked against a stored hash value. If the hash values match, the function is decrypted using the provided keys.

After decryption, the control flow enters the decrypted function, and the function calls are now executed from the decrypted memory location. After the function exits, the code of the function is re-encrypted.

Finally, `__cyg_profile_func_exit()` is called, indicating that the control flow has left the function.

Even though the code is being executed from the same address (where `__cyg_profile_func_exit` is called), the actual function code that is being executed is different each time, as it is decrypted using a different set of keys.

So your statement is kind of invalid.

HiddenEyE - December 26, 2023, 12:35 am

(Dec 26, 2023 01:28 AM)Tentco Wrote: [\[CITE\]](#) The idea is that the program is encrypting its own code while it's running, so that any memory dump will only show encrypted code, making it difficult to analyze or reverse engineer the program, this thread is unrelated to AV Evasion.

that is great, will try it out then.

can this evade memory scanners from seeing the opcode ?.

though i never triend such with my `mem_scan_engine`

Tentco - December 26, 2023, 12:43 am

(Dec 26, 2023 01:35 AM)HiddenEyE Wrote: [\[CITE\]](#) that is great, will try it out then.

can this evade memory scanners from seeing the opcode ?.

though i never triend such with my `mem_scan_engine`

The decrypted function's opcodes will still be present in memory, this could potentially maybe bypass some memory scanner still that's not the main motive of this, you can modify it as you want.

osn64698 - December 26, 2023, 5:46 am

(Dec 26, 2023 01:35 AM)HiddenEyE Wrote: [\[CITE\]](#) that is great, will try it out then.

can this evade memory scanners from seeing the opcode ?.

though i never triend such with my `mem_scan_engine`

You can also use it to fully evade static scanners/scantime. It is also much better to use encryption such as : base64/base58 for better entropy.

Tentco - December 26, 2023, 11:49 pm

Some Improvement Suggestions for anyone who uses it as a base, Add multi-threaded support(suggestion by [osn64698](#)).

Fix relative offsets. Better Encryption + long key size.

Indirect syscalls for Page Protection change In the provided code, there is a `__cyg_profile_func_exit` call at the end of each function that is being called. This call is responsible for

cleaning up the stack and restoring the RWX permissions to their original state. However, the issue arises when the called function enters an

infinite loop, such as `while(1)`.

This is because the infinite loop will keep executing without ever returning control to the caller, and therefore, the `__cyg_profile_func_exit`

call will never be reached. As a result, the RWX permissions for the stack will remain as RW, which leads to crash(because the compiler does

not expect you to exit the loop).

Sleep obfuscation.

HiddenEyE - January 1, 2024, 7:37 am

(Dec 26, 2023 06:46 AM)osn64698 Wrote: [\[CITE\]](#) You can also use it to fully evade static scanners/scantime. It is also much better to use encryption such as : base64/base58 for better entropy.

i use unix to unix encoder i coded in c.

i believe it is better.

osn64698 - January 1, 2024, 7:39 am

(Jan 1, 2024 08:37 AM)HiddenEyE Wrote: [\[CITE\]](#) i use unix to unix encoder i coded in c.

i believe it is better.

What does that mean? lol I do not understand.

HiddenEyE - January 1, 2024, 7:43 am

(Dec 27, 2023 12:49 AM)Tentco Wrote: [\[CITE\]](#) Some Improvement Suggestions for anyone who uses it as a base, Add multi-threaded support(suggestion by [osn64698](#)).

Fix relative offsets. Better Encryption + long key size.

Indirect syscalls for Page Protection change In the provided code, there is a `__cyg_profile_func_exit` call at the end of each function that is being called. This call is responsible for

cleaning up the stack and restoring the RWX permissions to their original state. However, the issue arises when the called function enters an

infinite loop, such as `while(1)`.

This is because the infinite loop will keep executing without ever returning control to the caller, and therefore, the `__cyg_profile_func_exit`

call will never be reached. As a result, the RWX permissions for the stack will remain as RW, which leads to crash(because the compiler does

not expect you to exit the loop).

Sleep obfuscation.

the recommendations are good.

but OP do you know what "`__cyg_profile_func_enter`" calls internally ?

Tentco - January 1, 2024, 7:47 am

(Jan 1, 2024 08:43 AM)HiddenEyE Wrote: [\[CITE\]](#) the recommendations are good.

but OP do you know what "`__cyg_profile_func_enter`" calls internally ?

When you enter a function, `__cyg_profile_func_enter` is called, which in turn calls `_dl_runtime_resolve`. The system-dependent function then performs the actual task of resolving the function address and storing the context.

To refer go here, <https://github.com/bminor/glibc>

HiddenEyE - January 1, 2024, 8:36 am

(Jan 1, 2024 08:39 AM)osn64698 Wrote: [\[CITE\]](#) What does that mean? lol I do not understand.

use google to read about uuencoder and uudecoder

those are good for linux but i wrote it for windows