

Forum:[Hack Forums](#)

Thread: [Computing the stub size of a crypter](#)

Ozedath - March 6, 2016, 1:06 am

This thread is a continuation of this one:

<https://hackforums.net/showthread.php?tid=5193244>

In the following code:

```
Code void _xor_block(unsigned char *pblock, unsigned int blocksize)
{
    unsigned int i;

    for(i = 0; i < blocksize; i++)
        pblock[i] ^= i;

    return;
}
```

```
int _xor_chunk(unsigned char* pchunk, unsigned long chunksize, unsigned int blocksize)
{
    if(chunksize % blocksize || blocksize % 8)
        return -1;

    unsigned long index = 0;

    while(index < chunksize)
    {
        _xor_block(pchunk + index, blocksize);
        index += blocksize;
    }

    return 0;
}
```

```
__declspec(naked) void __stdcall _stub(void)
{
    __asm
    {
        push 0xFEFEFEFE //blocksize
        push 0xFDFDFDFD //chunksize
        push 0xFCFCFCFC //pchunk

        call _xor_chunk //decrypt

        mov eax, 0x7FFFFFFF //oep
        jmp eax //go go
    }
}
```

```

    }
}

__declspec(naked) int __end(void)
{
    __asm ret 8
}

```

Why isn't the stub size computed correctly?

Code `DWORD dwStubSize = (DWORD) __end - (DWORD) __xor_block;`
 The `dwStubSize` is `0xfffff56`, for example, for `__end` being loaded at `0x00d515a0` and `__xor_block` at `0x00d51450`.

Why isn't the correct value of the difference being set?

NTTITON376 - March 6, 2016, 1:44 am

You're a fucking dumb ass.

Ozedath - March 6, 2016, 2:00 am

(Mar 6, 2016 02:44 AM)NTTITON376 Wrote: [\[CITE\]](#) You're a fucking dumb ass.

lol. Why? I'm trying to get a fully functional crypter and start working on my own from there.

Only3 - March 6, 2016, 2:04 am

size of `__xor_block` function = `(DWORD) __end < address of __end function - (DWORD) __xor_block < address of __xor_block function;`

Ozedath - March 6, 2016, 2:40 am

(Mar 6, 2016 03:04 AM)Only3 Wrote: [\[CITE\]](#) size of `__xor_block` function = `(DWORD) __end < address of __end function - (DWORD) __xor_block < address of __xor_block function;`

But shouldn't `__end` have the highest address value since it's the last function declared of those?

morph - March 7, 2016, 1:12 am

(Mar 6, 2016 03:40 AM)Ozedath Wrote: [\[CITE\]](#) But shouldn't `__end` have the highest address value since it's the last function declared of those?

Optimization

Kenshin13 - March 10, 2016, 8:23 am

Disable optimization.

Apply some sort of padding at the end of functions. (Simply checking for `0xCCs` usually work)

RISC - March 10, 2016, 9:23 am

You don't need to disable optimisation. You can simple use the `noinline` attribute.

push rcx - March 10, 2016, 10:16 am

Regardless if padding is applied, he's getting a negative distance, `0xfffff56`

You're running this with debugging mode, therefore it's not actually pointing at a function, it's pointing at a relative JMP instruction.

To avoid this run it in release mode or by disabling incremental linking. (Be careful, when disabling incremental linking the code structure will not be aligned, you'd have to enable optimization and modify reference settings, it's alot of unnecessary setting configurations)

Code Link Incrementally is not selected (/INCREMENTAL:NO)

/OPT:REF is selected

/OPT:ICF is selected
/OPT:LBR is selected
/ORDER is selected
For example

printing `&_end` in debugging mode, won't actually be the `&end` function, rather an address that executes a relative jmp instruction to your function.

I'd leave the incremental linking to default settings and use this function on debugging mode, possibly with some optimization:

```
Code __declspec(naked) unsigned long _end()
{
    __asm
    {
        call foo
foo:
        pop eax
        sub eax, 5+_xor_block
        ret
    }
}
printf("%2X\n", _end());
```

Only3 - March 10, 2016, 11:18 am

if he need to use return then he can define it as unsigned int

Ozedath - March 10, 2016, 2:53 pm

(Mar 10, 2016 11:16 AM)push rcx Wrote: [\[CITE\]](#) Regardless if padding is applied, he's getting a negative distance, 0xfffff56

You're running this with debugging mode, therefore it's not actually pointing at a function, it's pointing at a relative JMP instruction.

To avoid this run it in release mode or by disabling incremental linking. (Be careful, when disabling incremental linking the code structure will not be aligned, you'd have to enable optimization and modify reference settings, it's alot of unnecessary setting configurations)

Code Link Incrementally is not selected (/INCREMENTAL:NO)

/OPT:REF is selected
/OPT:ICF is selected
/OPT:LBR is selected
/ORDER is selected
For example

printing `&_end` in debugging mode, won't actually be the `&end` function, rather an address that executes a relative jmp instruction to your function.

I'd leave the incremental linking to default settings and use this function on debugging mode, possibly with some optimization:

```
Code __declspec(naked) unsigned long _end()
{
```

```
    __asm  
    {  
        call foo  
foo:  
        pop eax  
        sub eax, 5+_xor_block  
        ret  
    }  
}
```

```
printf("%2X\n", _end());
```

Many thanks :). I am/was running it in debug mode. Am retrying with the suggested compilation parameters.