

Forum: [Hack Forums](#)

Thread: [True USG](#)

NyQuil - March 18, 2017, 10:41 pm

ive tried wrapping my head around the subject for along time since i been programming thou its never been a serious thought till now as im now (legal/illegal) writing my own streams into a process but my wonders are what makes usg really. on the internet you see randomizing every string, var, bool name, type parameter names, etc as well as you can see things such as adding all these different lines that make the full code thrown in a array and randomly laid out in a random codedom environment. as well as say "path.combine" for documents location can be write in 100 ways as they say does this mean that running an iteration random through them to create a new function per random layout per random stub?

Hoody - March 21, 2017, 11:13 pm

When most hf kiddies say their crypter has usg they mean skid usg (aka re-name variables, then c+p Eternal's codedom class)

What would real usg look like... i have a basic example :)

This example is from a crypter of mine, the code you are looking at (below) is the C loader that executes the shellcode. Basic idea behind it is very simple. Break your code down into parts, then find different ways to complete that same part. At the end you should be able to mix-match the parts into a "unique" stub.

Kinda big so.. [\(Click to View\)](#)

btw _start is the entry (not main) because it compiles with Tiny C Compiler

This might be more difficult for a large crypter. But why wouldn't you always wrap your crypter in a loader... refud +1000 line crypter vs. refud 100-200 line loader

ShadowCloudGH - March 22, 2017, 2:56 am

(Mar 21, 2017 11:13 PM)Hoody Wrote: [\[CITE\]](#) ...Sensible stuff said here

This isn't really my area of expertise, but on a complete side note, why wouldn't you just write the loader against an interface that you retrieve from your control server dynamically and then inject the selected implementation of that interface at compile time for the "unique" stub?

HFDone - March 22, 2017, 5:16 am

(Mar 21, 2017 11:13 PM)Hoody Wrote: [\[CITE\]](#) When most hf kiddies say their crypter has usg they mean skid usg (aka re-name variables, then c+p Eternal's codedom class)

What would real usg look like... i have a basic example :)

This example is from a crypter of mine, the code you are looking at (below) is the C loader that executes the shellcode. Basic idea behind it is very simple. Break your code down into parts, then find different ways to complete that same part. At the end you should be able to mix-match the parts into a "unique" stub.

Kinda big so.. [\(Click to View\)](#)

btw _start is the entry (not main) because it compiles with Tiny C Compiler

This might be more difficult for a large crypter. But why wouldn't you always wrap your crypter in a loader... refud +1000 line crypter vs. refud 100-200 line loader

? https://en.wikipedia.org/wiki/Polymorphic_code

ShadowCloudGH - March 22, 2017, 5:29 am

(Mar 22, 2017 05:16 AM)HFDone Wrote: [\[CITE\]](#) ? https://en.wikipedia.org/wiki/Polymorphic_code
Yes?

I mean still inject the code from the concrete in CodeDom, but now you just need to deal with an engine that builds random implementations of specific interfaces? I'm guessing you can probably use something like GetMethodBody and then just encrypt it using something specific to each compilation?

You are not going to achieve true polymorphic code in a managed language. (at least not in a meaningful way that's truly polymorphic per run rather than per compilation)

So my question is why you don't have a loader that's simply coded against interfaces (So no method body) and then have the loader retrieve a single concrete implementation of the selected interface, load up the method body, then encrypt it with some symmetrical algorithm and inject that into the loader. The loader would just need to know how to decrypt these? You have small "hotswappable" concrete implementations which would be much easier to maintain?

Aristocles - March 22, 2017, 8:50 am

(Mar 22, 2017 05:29 AM)ShadowCloudGH Wrote: [\[CITE\]](#) Yes?

I mean still inject the code from the concrete in CodeDom, but now you just need to deal with an engine that builds random implementations of specific interfaces? I'm guessing you can probably use something like GetMethodBody and then just encrypt it using something specific to each compilation?

You are not going to achieve true polymorphic code in a managed language. (at least not in a meaningful way that's truly polymorphic per run rather than per compilation)

So my question is why you don't have a loader that's simply coded against interfaces (So no method body) and then have the loader retrieve a single concrete implementation of the selected interface, load up the method body, then encrypt it with some symmetrical algorithm and inject that into the loader. The loader would just need to know how to decrypt these? You have small "hotswappable" concrete implementations which would be much easier to maintain?

Such engine wouldn't be random. Your engine would have to somehow select a pre-written interface from a list.

ShadowCloudGH - March 22, 2017, 10:36 am

(Mar 22, 2017 08:50 AM).NETGuard Wrote: [\[CITE\]](#) Such engine wouldn't be random. Your engine would have to somehow select a pre-written interface from a list.

Indeed, however as per my previous post, you could then have a much easier time of implementing new ones and then encrypt those depending on the build so the encrypted version is unique.

In any event, while this might get past static analysis it won't be much help against dynamic

analysis?

Aristocles - March 22, 2017, 11:56 am

(Mar 22, 2017 10:36 AM)ShadowCloudGH Wrote: [\[CITE\]](#) Indeed, however as per my previous post, you could then have a much easier time of implementing new ones and then encrypt those depending on the build so the encrypted version is unique.

In any event, while this might get past static analysis it won't be much help against dynamic analysis?

Dynamic analysis quality is only based upon who is doing the analysis. Therefore nobody can infer anything from your interrogation.

Static analysis nowadays are based on very efficient algorithm and large database of heuristic detections, therefore I even doubt it would pass this step : having a program which create code ad so is shady in itself, therefore it would triggered AV.

HFDone - March 22, 2017, 12:15 pm

the engine is written in asm as its the only way to do it.

mov eax, 1 = xor eax, eax > add eax, 1 etc ...

and its done at runtime with jmp call mov instructions.

all the other ways that mentioned doesnt provide real usg!

Hoody - March 22, 2017, 3:15 pm

(Mar 22, 2017 02:56 AM)ShadowCloudGH Wrote: [\[CITE\]](#) This isn't really my area of expertise, but on a complete side note, why wouldn't you just write the loader against an interface that you retrieve from your control server dynamically and then inject the selected implementation of that interface at compile time for the "unique" stub?

If done in a OOP language (c#, vb.cancer, java) that'd be the "correct" way to do it. Considering you dont have interfaces (to my knowledge) in c cant really do that lol

Actually... using an interface for this would be a really go idea. Imagine a "community crypter" where you can mix match parts from authors on hf to get a fud stub. Doubt thatd happen, but its a cool thought

(Mar 22, 2017 05:16 AM)HFDone Wrote: [\[CITE\]](#) ? https://en.wikipedia.org/wiki/Polymorphic_code
The sudo code in the first example on the wiki is a loader... same as mine.
why the ? face... mine is exactly that...

You might be thinking of metamorphism?

ShadowCloudGH - March 22, 2017, 8:08 pm

(Mar 22, 2017 11:56 AM).NETGuard Wrote: [\[CITE\]](#) Dynamic analysis quality is only based upon who is doing the analysis. Therefore nobody can infer anything from your interrogation.

Static analysis nowadays are based on very efficient algorithm and large database of heuristic detections, therefore I even doubt it would pass this step : having a program which create code ad so is shady in itself, therefore it would triggered AV.

O, while I completely agree the builder would trigger the AV (So when we are dealing with picking up the correct code and encrypting the method bodies) but I don't think static analysis would pick up the built stub since we are no longer adding the code dynamically, it's been compiled before?

HFDone - March 23, 2017, 7:16 am

(Mar 22, 2017 03:15 PM)Hoody Wrote: [\[CITE\]](#) If done in a OOP language (c#, vb.cancer, java) that'd be the "correct" way to do it. Considering you dont have interfaces (to my knowledge) in c cant really do that lol

Actually... using an interface for this would be a really go idea. Imagine a "community crypter" where you can mix match parts from authors on hf to get a fud stub. Doubt thatd happen, but its a cool thought

The sudo code in the first example on the wiki is a loader... same as mine.
why the ? face... mine is exactly that...

You might be thinking of metamorphism?
"This example is not really a polymorphic code ..."
read my last reply ?

webinjectsformgrabberhvc - *March 23, 2017, 7:18 pm*

Instructions such as MOV are turing complete. True USG is completed within the primitive instruction set.