

Forum:[Hack Forums](#)

Thread: [\[Advanced\]](#) [\[C\]](#) [Writing a Metamorphic Engine / Real USG; Part 1](#)

movsx - July 22, 2017, 1:40 pm

Introduction

This is the first of a several part tutorial series which will teach you how to write a metamorphic engine / real unique stub generator which will take a subset of the C source language and output unique assembly code. This tutorial is the easiest in the series, each tutorial will be significantly harder than the last. So if you're having difficulty following now, I recommend you go back and study some basic computer science and C coding. If this tutorial is popular, I'll release the second one. After releasing all tutorials, I'll release the full source code on github. This tutorial isn't meant to spoonfeed you, only guide you. Don't know what an abstract syntax tree is? Google it and find out, then come back.

Please note that the purpose of these tutorials aren't for malware, nor for you to copy paste and use in your own projects. The purpose of this series is to demonstrate that metamorphic engines are a real threat to computer security, and hopefully motivate individuals to try and find ways to stop them.

This series is not for the faint of heart, as this simple C subset metamorphic engine spans over 15,000 lines of C code and requires a deep understanding of the C language to follow, as well as you being willing to learn the compiler theory concepts mentioned here and study them well. 15,000 lines of C code is a tiny project, in all reality, but some of you may not be used to projects of this size, nor the actual complexity of the code involved, as each phase of the engine is drastically different from the last, and will begin to use some complex algorithms and data structures.

Drawing from Compiler Theory

A metamorphic engine is essentially a compiler, except its goal is not to compile efficient or small code, but to compile code that is unique and well-functioning. Consequently, there is a significant overlap with writing compilers and metamorphic engines, so the front-end of our engine will essentially be the same as writing a front-end for a compiler. The backend is where metamorphic engines differ from compilers, as you will see in later tutorials.

Brief Discussion of Compiler Frontends

- A source language is input to the front end
- First the front end breaks this input into a sequence of tokens.
- Second the front end generates a parse tree from these tokens
- Third the front end uses syntax directed translation to convert the parse tree into an abstract syntax tree
- Fourth the front end creates a symbol table from the AST
- Fifth the front end runs the AST through the semantic analysis stage to determine if the grammar is semantically valid: proper use of types, not using vars out of scope, not using undeclared vars, etc, etc
- Sixth the front end runs the AST through another syntax directed translator to generate an intermediate representation language, which is then fed into the back end to have code mutation applied to it. This is where the majority of effort/code is at: Control flow modifications, junk code generation, instruction selection, instruction scheduling, register allocation, and function inlining / outlining occur here.

Let's get started! Front end: Stage 1; Lexing

Honestly there's not much to say. The lexer is incredibly easy to implement. I won't even bother discussing it. Play around with lexers, write your own, and understand why they're important.

Here's the code for the lexer:

linkedlist.h <https://pastebin.com/x556r8DL>
linkedlist.c <https://pastebin.com/DV058wt3>
lexer.h <https://pastebin.com/cdxwAdmL>
lexer.c <https://pastebin.com/Emu3rpzx>

Here's an example C block

```
Code {  
    int x;  
    x = 0;  
    while (x < 10) {  
        x = x + 1;  
    }  
}
```

And here's the output of the lexer

```
Code [[] [INT] [Identifier x] [;] [Identifier x] [=] [Int Literal=0] [;] [WHILE ] [[] [Identifier x] [<] [Int  
Literal=10] []] [[] [Identifier x] [=] [Identifier x] [+] [Int Literal=1] [;] []] []]
```

The next tutorial (if this one's popular) will be on recursive descent parsing and will cover generating parse trees -> abstract syntax trees

Malicious Intent - July 22, 2017, 1:54 pm

Interesting. Looking forward to next parts of this tutorial. Not often one comes across something like this presented in readable and compressed form, giving just enough to start off and allowing one to dive deeper into less understood parts by encouraging own research..

maldev0 - July 22, 2017, 2:06 pm

I'm sure this will get very interesting. Especially the prediction of 15k lines of code. Good work, keep it up!

Kind regards

movsx - July 22, 2017, 2:32 pm

(Jul 22, 2017 02:06 PM)maldev0 Wrote: [\[CITE\]](#) I'm sure this will get very interesting. Especially the prediction of 15k lines of code. Good work, keep it up!

Kind regards

It's not a prediction.

ForceJew - July 22, 2017, 5:50 pm

Looks interesting waiting to read the next posts!

TouchMyMalware - July 22, 2017, 6:02 pm

(Jul 22, 2017 05:50 PM)FORC3FI3LD Wrote: [\[CITE\]](#) Looks interesting waiting to read the next posts! There won't be any. He's just outputting some basic code without a finished product. Trust me, he'll never finish this series.

movsx - July 22, 2017, 6:23 pm

(Jul 22, 2017 06:02 PM)TouchMyMalware Wrote: [\[CITE\]](#) There won't be any. He's just outputting some basic code without a finished product. Trust me, he'll never finish this series.

Wait and see buddy. All I see you do is talk shit towards other people. I've never seen you actually

produce anything yourself. Lmao.

TouchMyMalware - July 22, 2017, 6:56 pm

(Jul 22, 2017 06:54 PM)Yani Wrote: [\[CITE\]](#) So what should we be discussing?

This makes absolutely no sense.

It's because he's full of himself. He thinks literally everyone else is wrong and he's right. He also claims to be 23 years old with three bachelor degrees, and an IQ of 140+..No joke.

He thinks because I haven't released anything on HF that I lack any knowledge of programming or computer science. When in reality I couldn't care less to prove myself to that filth. I just like making him look stupid.

movsx - July 22, 2017, 7:01 pm

(Jul 22, 2017 06:54 PM)Yani Wrote: [\[CITE\]](#) So what should we be discussing?

This makes absolutely no sense.

There is a lot to discuss about lexers. For instance. What happens if the user inputs an invalid C program. Does the lexer care about that? What additional information can we propagate to later stages? What do we currently propagate?

How will the lexer be used in the parser?

Try to roll your own lexer, in any language, and think about the next steps.

(Jul 22, 2017 06:59 PM)Yani Wrote: [\[CITE\]](#) That one's great too..

Source code will be released every tutorial, but it's only meant to be used as reference, not copy pasted. If you note, I never mentioned how to actually use the code either.

(Jul 22, 2017 06:56 PM)TouchMyMalware Wrote: [\[CITE\]](#) It's because he's full of himself. He thinks literally everyone else is wrong and he's right. He also claims to be 23 years old with three bachelor degrees, and an IQ of 140+..No joke.

He thinks because I haven't released anything on HF that I lack any knowledge of programming or computer science. When in reality I couldn't care less to prove myself to that filth. I just like making him look stupid.

No, you're the arrogant one, constantly coming to my threads and talking shit. Every time I've talked to you on discord about computer science or coding in general (after you seemingly call me out and accuse me of knowing nothing, with no provocation whatsoever), you've proved to know absolutely jack shit beyond the complete fundamentals you'd read in a "Learn C in 12 hours" book.

I'm not going to argue with a kid who doesn't even know what a TLB is. Please kindly leave my thread and stop trolling, unless you wish to discuss the topic at hand.

TouchMyMalware - July 22, 2017, 7:04 pm

(Jul 22, 2017 07:01 PM)movsx Wrote: [\[CITE\]](#) I'm not going to argue with a kid who doesn't even know what a TLB is. Please kindly leave my thread and stop trolling, unless you wish to discuss the topic at hand.

You need to learn to read mate.

movsx - July 22, 2017, 7:18 pm

(Jul 22, 2017 07:13 PM)Yani Wrote: [\[CITE\]](#) I already played around with lexers, and was quite successful. I was trying to make a runtime interpreter, much like PHP. But gave up as I had more rewarding projects going on.

Anyways, I'm not only talking about the discussion of lexers, but also this "tutorial". I can discuss the code of this lexer, I can discuss metamorphic engines, but really, I can't discuss this "tutorial" because there is none.

There isn't really a way to put this information as a "tutorial". In my eyes you'd better be off writing a paper on it.

There wouldn't be a possible action/branch to take when it reaches the tokens that are invalid. So it can't do anything else than care.

But that's for part 2 of your "tutorial".... We first have to have this useless discussion while we wait for the next parts...

If you're not familiar with octal, get so, as here is a counterexample:

```
int i;  
i = 012399;
```

The lexer would be responsible for reporting this error, namely, when it runs into the literal 012399 because it is an invalid octal value.

Not only this, there's quite a bit more too. But I leave the discussion to those commenting here.

I'm doing this for a reason. Without fundamentally understanding these "useless parts" there is no metamorphic engine. It's a combination of many pieces, the lexer being one of them.

movsx - July 22, 2017, 7:23 pm

(Jul 22, 2017 07:20 PM)Yani Wrote: [\[CITE\]](#) Then just release all parts together, or write a paper.

Stop posting parts like these to feed your ego.

The purpose is to teach the community. Not feed my ego. If I just release the metamorphic engine, nobody will learn anything.

POLY MATH - July 22, 2017, 7:24 pm

Great tutorial but I must say that releasing one part at a time is bad practice around HF.

Try releasing 2-3 Part clusters.

movsx - July 22, 2017, 7:29 pm

(Jul 22, 2017 07:24 PM)Yani Wrote: [\[CITE\]](#) I thought you meant stuff like syntax errors.

Your example doesn't really prove a point tho because you could handle that use-case however you want.

There's actually no other place you would handle it

(Jul 22, 2017 07:24 PM)TRADOC Wrote: [\[CITE\]](#) Great tutorial but I must say that releasing one part at a time is bad practice around HF.

Try releasing 2-3 Part clusters.

I don't think that is true for metamorphic engines. Parsing is a very heavy topic and introducing it at the same time as lexing is a bad idea.

Believe me when I say that other tutorials will be much more involved, likely 5 times the length as this one.

I also plan on releasing the tutorials at given time intervals, not all at once. Otherwise people

wouldn't be able to learn anything.

(Jul 22, 2017 07:24 PM)Yani Wrote: [\[CITE\]](#) But you're not learning anybody anything. You just give them the info you already know.

Don't call it a tutorial if this isn't one, and this isn't one, this is merely a part of it.

I respect this knowledge but please write something like a paper instead of this super short non-tutorial thread.

Or release all of them together idc.

As stated above, I acknowledge this tutorial is short. It's an introduction coupled with lexing. The other tutorials will easily be 5-10x the length of this one, and much more heavily involved. That's just due to their nature.

TouchMyMalware - July 22, 2017, 7:33 pm

(Jul 22, 2017 07:23 PM)movsx Wrote: [\[CITE\]](#) The purpose is to teach the community. Not feed my ego. If I just release the metamorphic engine, nobody will learn anything. You need to learn to read mate.

TouchMyMalware - July 22, 2017, 7:43 pm

(Jul 22, 2017 07:37 PM)Yani Wrote: [\[CITE\]](#) How can we "popularize" this tiny bit of information WHICH IS NOT A TUTORIAL?

You know when he made his last so called "tutorial" on "graph theory" you know the basic shit people would learn on their second year of their CS studies, he was sitting there in Discord complaining about how no one would comment on his thread even though he posted an "A+" tutorial. He literally does this just to feed his ego.

Dadas1337 - July 23, 2017, 4:06 am

why do we even need a semi-complete compiler to achieve metamorphism?

glitchgxd - July 23, 2017, 4:19 am

I don't feel like I've learned anything from this.

I don't think this will get popular if we're not learning anything, and judging by OP, no further guides come out unless there is an interest. So I'm guessing this is the end of the road here and will now move on with life.

Thanks!

movsx - July 23, 2017, 8:31 am

(Jul 23, 2017 04:06 AM)Dadas1337 Wrote: [\[CITE\]](#) why do we even need a semi-complete compiler to achieve metamorphism?

Essentially we need to transform the source language (be it C, assembly, etc) into intermediate representation language so that it is easy for us to manipulate. That is what a compiler does. The metamorphic engine will differ at its backend from the compiler, but their front ends will be nearly identical.

(Jul 23, 2017 04:19 AM).Glitch Wrote: [\[CITE\]](#) I don't feel like I've learned anything from this.

I don't think this will get popular if we're not learning anything, and judging by OP, no further guides come out unless there is an interest. So I'm guessing this is the end of the road here and will now move on with life.

Thanks!

I will be definitely posting a second tutorial which will cover parsing, syntax directed translation, and possibly semantic analysis if I can fit it in there. I don't expect there to be much interest in lexing because its relatively basic.

Just make sure you undertand how to do lexing, and why its done.

It's much easier to parse tokens than it is to parse a string.

lolimsoasd - July 23, 2017, 9:38 am

Basic stuff for now, but as this is basic, why didn't you just write the 2nd part into this same thread as pasting that code to the thread took max 20 minutes? I'm very interested for the oncoming stuff tho.

Malicious Intent - July 23, 2017, 11:20 am

Not sure why so much hate... why not just wait for second part and start judging there .. the way i see this is just introduction part of the tutorial and every long tutorial starts with a primer so sit tight folks and let the guy breath.

movsx - July 23, 2017, 11:23 am

I've started writing part 2 - but I have a problem right now. HF firewall is blocking it when I preview because its too long. Ive PM'd a staff so hopefully it can get posted when its done.

lolimsoasd - July 23, 2017, 11:26 am

(Jul 23, 2017 11:23 AM)movsx Wrote: [\[CITE\]](#) I've started writing part 2 - but I have a problem right now. HF firewall is blocking it when I preview because its too long. Ive PM'd a staff so hopefully it can get posted when its done.

You can always post it in 2 posts :)

movsx - July 23, 2017, 11:28 am

(Jul 23, 2017 11:26 AM)lolimsoasd Wrote: [\[CITE\]](#) You can always post it in 2 posts :)

But when you post two posts in sequence, doesnt the second one get added to the first one. Wouldnt this cause the same problem with the firewall?

lolimsoasd - July 23, 2017, 11:35 am

No idea tbh. Never wrote that long.

0xDEAD10CC - July 23, 2017, 12:29 pm

Why do you duplicate your includes and sometimes separate them between your header and source files?

There's a bit of redundancy I see in the code too.

```
Code void *linkedlist_remove(struct linkedlist *ll)
void *linkedlist_removetail(struct linkedlist *ll)
linkedlist_remove is just a forward call to linkedlist_removetail with the exact same parameter which makes it pretty useless. Things like isnum, isal, are already have standard C library equivalents, as with many of the other things you're doing manually with pointer arithmetic and byte comparison.\
Code while (*buf == ' ' || *buf == '\t' || *buf == '\r' || *buf == '\n') (Jul 23, 2017 11:28 AM)movsx
Wrote: \[CITE\] But when you post two posts in sequence, doesnt the second one get added to the first one. Wouldnt this cause the same problem with the firewall?
That's why there's the :
```

Code [paste] [/paste]

Pastebin tags. Most of the firewall issues I've had were in code I was trying to post. Post length != cloudflare firewall either, it's in the contents not the length of your post.

movsx - July 23, 2017, 12:48 pm

(Jul 23, 2017 12:29 PM)0xDEAD10CC Wrote: [\[CITE\]](#) Why do you duplicate your includes and sometimes separate them between your header and source files?

There's a bit of redundancy I see in the code too.

Code void *linkedlist_remove(struct linkedlist *ll)

void *linkedlist_removetail(struct linkedlist *ll)

linkedlist_remove is just a forward call to linkedlist_removetail with the exact same parameter which makes it pretty useless. Things like isnum, isal, are already have standard C library equivalents, as with many of the other things you're doing manually with pointer arithmetic and byte comparison.\

Code while (*buf == ' ' || *buf == '\t' || *buf == '\r' || *buf == '\n')

That's why there's the :

Code [paste] [/paste]

Pastebin tags. Most of the firewall issues I've had were in code I was trying to post. Post length != cloudflare firewall either, it's in the contents not the length of your post.

These are called good software engineering practices.

linkedlist_remove is called a "wrapper function." I suggest you look into wrapper functions so you can see why they are useful. I use wrapper functions throughout my codebase. The main reason for this is I also implement a stack, which is used as a cactus stack for scope resolution of syntax trees, and the stack is essentially just a huge wrapper for the linked list. It's important for function declarations to be clear, for reuse.

Separating includes is considered good practice, so the client knows what the procedures declarations relies on. The header files generally should only include things necessary for its declarations, while the source file should generally include things required for its internal functioning.

As for coding in general, the most important thing is to push a product, and push a product fast. Nobody is going to spend long periods of time planning to write girlcode, that's just how it works.

0xDEAD10CC - July 23, 2017, 12:55 pm

You also have a logical error:

The wrapper function in this case is redundancy with no added benefit however. If you change it to remove the head you'll end up having to change other code anyways. It's only good for things like your size function where you can change it internally to return a different calculation for size in multiple other places. Explain where I'm missing the benefit of this wrapper function?

>> It's important for function declarations to be clear, for reuse

match1 and match2 however is not very clear in terms of declarations and naming in your lexer code.

movsx - July 23, 2017, 12:57 pm

(Jul 23, 2017 12:55 PM)0xDEAD10CC Wrote: [\[CITE\]](#) You also have a logical error:

The wrapper function in this case is redundancy with no added benefit however. If you change it to remove the head you'll end up having to change other code anyways. It's only good for things like your size function where you can change it internally to return a different calculation for size in multiple other places. Explain where I'm missing the benefit of this wrapper function?

That's not right. Good catch. The semicolon shouldn't be there. I'll fix it up in my own codebase.

My code actually doesn't reach that case, though. Nonetheless, it should be fixed.

EDIT:

The wrappers there because I wanted clients of linkedlist to use remove while using removetail in stack.c for clarity.

Also I'm match1 and match2 are used internally in the lexer. Clients of lexer don't have to care about match1 or match2. And I think the names are pretty good, regardless.

whitegoddess - July 9, 2019, 7:23 pm

Guess he never got to finish the tutorial after all!!! law!!!

vxheavens!!

TGilb94 - July 12, 2019, 2:07 pm

Interesting. Thanks for sharing this with us, much appreciated.