Trolling with Math

```
base26_t YOU = 0x038C2767;
```
Trolling with Math

wat?

• frank^2
• pronounced “frank 2” (the carat is ~flare~)
• “that guy with the hat”
• DC949
• DC310
Trolling with Math

When we last left our heroes...

- “I have no idea what the fuck frank^2 is talking about, but its awesome.”
- “More content, less bullshit.”
Content
Trolling with Math

ya we pumpin
Trolling with Math

MATH!

• It’s very possible your math teacher made this more complicated than it needs to be.
• \( f(x) = x \times 7 \)
• \((\lambda x: x \times 7)\)
• public static int multiplyBySevenAndReturn(Integer x)
  { return x \times 7; }

• Mathematic functions can get even more complicated, but this is all we need for now.
Trolling with Math

ASSEMBLY!

- JMP and CALL instructions are not specific with immediate values. They’re offsets.
- JMP 00401000 is more like JMP-A-FEW-BYTES-AHEAD. It’s the same with CALL.
- ...except CALL sticks its dick in your stack.
- **Hot.**
Trolling with Math

ASSEMBLY!

- Oh, except for when you stick an address in a register. *Totally different.*
- When you stick an address in a register and then do something like CALL EAX, it specifically goes to whatever value is in EAX.
- Same goes for CALL [EAX] or JMP [EAX]-- it just dereferences EAX and jumps to that address.
Trolling with Math

ASSEMBLY!

• Let’s talk about JMP SHORT.
• This is essentially a jump within the range of -127 8==0==D 127.
• Regular JMP instructions are more like -2147483647 8=====0=====D 2147483647
Trolling with Math

ASSEMBLY!

• There is no such thing as CALL SHORT.
• I know, right?
• What the hell.
Trolling with Math

ASSEMBLY!

• Here’s some computer science witchcraft.
• Technically you can define the space between each instruction as...
Trolling with Math

nullspacelabs
Trolling with Math

ASSEMBLY!

• Each instruction is executed one after another.
• This can be interpreted as an unconditional jump to the next instruction.
• This gives us space between each assembly instruction so long as each instruction is subsequently linked by an unconditional jump.
Trolling with Math

ASSEMBLY!

```
MOV    EAX, 5355434B
MOV    EBX, 20412044
XOR    EAX, EBX
CALL   49434B20
MOV    EDX, EAX
SUB     AL, 46
XOR    EAX, 41545459
```
Trolling with Math

ASSEMBLY!

```
MOV    EAX, 5355434B
JMP    0
MOV    EBX, 20412044
JMP    0
XOR    EAX, EBX
JMP    0
CALL   49434B20
JMP    0
MOV    EDX, EAX
JMP    0
SUB     AL, 46
JMP    0
XOR    EAX, 41545459
JMP    0
```
Trolling with Math

ASSEMBLY!

- It is therefore possible to place every single assembly instruction in an arbitrary location in memory if and only if each singular instruction is followed by an accompanying unconditional jump to the next instruction.
Trolling with Math

ASSEMBLY!

• A one-dimensional array can technically be interpreted as a two-dimensional array as well. It just requires a little math.
• This gives us the ability to interpret a location in memory as an X/Y grid.
• Coupled with interpreting null space between instructions as unconditional jumps, we can literally draw instructions.
• This is awesome.
Trolling with Math

Let’s do this

• Disassemble each instruction.
• Allocate space in memory significantly larger than the collection of instructions.
• For each instruction, determine $f(x)$
• Place each instruction at the corresponding $(x,y)$ location in memory.
• Join the instruction with an unconditional jump.
• Mark memory executable and run.
Bullshit
Trolling with Math

FUCK!

• Like gravity, that shit only works in theory.
• In practice, we’re fucked.
• Totally and utterly fucked.
Trolling with Math

FUCK!

• All of your JMP instructions? Fucked.
• All of your CALL instructions? Fucked.
• Any self-referential code? Fucked.
• Self-modifying code that relies on iteration? You better BELIEVE it’s fucked.
Trolling with Math

FUCK!

• Let’s start with JMP instructions.
• Since JMPs are offsets, when placed in an arbitrary location, they no longer point to where you think they’re pointing at.
• Short JMPs are in a similar situation. When arbitrarily placed by your $f(x)$ function, they will very likely not point to where you think they will.
• Short JMPs are easily fixed. Long JMPs? Not so much.
Trolling with Math

FUCK!

- Dealing with register-based JMPs are going to be an issue as well.
- Since they require hard offsets and may be calculated at run-time, there is no easy way to determine where they’re going.
- So unless you want to do some extra work to get this working... you may as well ignore it.
Trolling with Math

FUCK!

- \( f(x) \) formulas aren’t nearly as elegant in code as they are on paper.
- This requires all sorts of strange voodoo magic if we want to use arbitrary formulas—function pointers, class pointers, the whole shebang.
Deal with it
Trolling with Math

At disassembly, convert all your JMP SHORTs to JMP PANTS before storing them away.

Simple enough!
Trolling with Math

The actual offset data though? *Hoo.*

All instructions which you’ve detected have offsets that will move when the code is moved need to be recalculated.

This means you need to:

- Keep track of the instructions.
- Keep track of the targets.
- See the source for an example of how I accomplished this.
Trolling with Math

he;l$\psi$

- After all the instructions are placed, replace the old offsets with the new offsets.
- Assuming you didn’t fuck up the offsets, those problems are now solved.
Trolling with Math

he;lρ

• Now that we have the caveats out of the way, we have a path to a potential higher-level implementation.

• It goes like this:
Trolling with Math

Implementation

- Disassemble instructions.
- Prepare buffer.
- Initialize $f(x)$ function constants.
- Iterate over $f(x)$ values and determine data pointers by which your code will be written to while tracking fucked instructions.
- Write the instructions to the corresponding pointers.


**Trolling with Math**

*Implementation*

• Repair all your conditional jumps.
• Mark the new section of memory as executable.
• **RUN!**
COOL STORY, BRO
Trolling with Math

Who cares?

• The isolation of assembly instructions and numerical steps to calculate \( f(x) \) allows us to place assembly instructions anywhere in the buffer we want to with little to no interaction from the user.

• In order to obfuscate the clarity of the codepath, all you have to do is write a function and point the MATHEMACHINAE at some assembly.
Trolling with Math

Who cares?

- This makes accomplishing various polymorphic techniques a little bit simpler as well.
- Instead of writing code that manipulates your code in a specific way each time, you can write a series of functions which randomly determine the location of your code, then select those functions at random, etc.
Trolling with Math

Who cares?

• Anti-reversing isn’t about how cool and fresh your anti-debug techniques are.
• Anti-reversing isn’t about how much of a boner you get from breaking out of IDA and spawning Last Measure all over a reverser’s desktop (but it is pretty goddamn funny).
• Anti-reversing is about being a dick.
Who cares?

- Everyone knows where to Google for anti-debug techniques.
- You can’t Google for creativity, though.
- The most creative anti-reversing assholes among you will be the direct result of broken fingers and fist-sized holes in plastered walls.
- And that’s something to be proud of!
Trolling with Math

Yo dawg, I heard this joke was played out...
Trolling with Math

...but it’s contextual, so fuck the haters
Trolling with Math

At least I didn’t use memegenerator
Trolling with Math

AW FUCK

USE UNCONDITIONALS

SUCK AT OBFUSCATION
Trolling with Math

*Shit sucks*

- But the example code only uses unconditional jumps.
- Unconditional jumps only go in one goddamn direction.
- Conditional jumps go in two.
- That makes them *better*.
Trolling with Math

wait wh

- If we require conditional jumps yet need to use unconditional jumps... what the fuck?
- Opaque predicates save the day!
- But why stop there?
Trolling with Math

Hardening

• Consider the null-space expansion posited earlier.

• If a set of instructions has an unconditional jump between each instruction, it also follows that a series of assembly instructions which do not have direct affect on the result of our desired instructions can precede or proceed a single instruction.

• This is even more awesome.
Trolling with Math

Hardening

pre-amble
assembly data
post-script
Trolling with Math

Hardening

• The pre-amble section can be used for two things:
  • Repairing the after-effects of the previous pre-amble’s opaque predicate.
  • Anti-debug code chunks.
Trolling with Math

Hardening

• The post-script is a whole lot more fun.
• This section can be used for:
  • Opaque predicates and obfuscated jumps
  • Anti-debug and general control-flow obfuscation
  • Encryption/decryption of various chunks of code within the program
Trolling with Math

Hardening

CALL IsDebuggerPresent
CMP EAX, 1
JE FuckYouNeighbor
MOV EAX, 5355434B
PUSH EAX
XOR EAX, EAX
JZ nextBlock

POP EAX
MOV EBX, 20412044
CLC
JNC nextBlock

POP EAX
MOV EBX, 20412044
CLC
JNC nextBlock
Trolling with Math

Hardening

• This obviously introduces a whole lot more issues than our baseline does, such as after-effects and a complication of generating all the different sets of instructions.

• So throw your Shmooballs if you got ‘em, I’m about to be That Guy:

COMING SOON
(AKA READ MY SHITTY BLOG)
Trolling with Math

**Hardening**

- Our $f(x)$ formulas don’t necessarily need to be calculated iteratively, e.g. $f(1)$, $f(2)$, ... $f(n)$
- There’s nothing to stop us from randomly calculating them as well!
Hardening

- If our code is generated from a predictable formula, then it follows the entry point is predictable, i.e. it can be calculated at runtime.
- Oh, hi mister Debugger! What’s that? You want to ride the snake?
- **NOT WORTHY**
Drawbacks
Trolling with Math

Drawbacks

• This technique assumes sanely compiled code.

• This means if you’re trying to obfuscate assembly that would make the Conficker gang say “bravo!”, you’re screwed.
Trolling with Math

Drawbacks

• Massive memory footprint.
• You are likely going to be dealing with a HUGE dataset when you’re done.
• This gets significantly larger when you obfuscate more than just one functions.
• I sure hope your FuDG3p4Ck3R v6.66 is efficient!
Trolling with Math

Drawbacks

• You think your function pointers are so clever? Yeah? Fuck you, they’re broken.

• Wise-guy thought he might be smart by using C++ and the STL to get by and make his code more efficient? Fuck your OOP paradigm.
Drawbacks

• The more clever you get with generating obfuscation pairs and manipulating the assembly, the more complicated it gets to repair.

• It’s a slippery-slope from “oh, hey, neat, I can just stick a JMP or two in here” to “HOW THE FUCK DO I MEMORIZE THE DRAGONBOOK IN AS SHORT AMOUNT OF TIME AS POSSIBLE?!”
Trolling with Math

End!

- @franksquared
- http://argolithmic.blogspot.com
- http://argolith.ms