reverse专题 – 2 混淆

outline

- 上节课的一些遗留内容
- 混淆
- 还有时间分享一些逆向小心得

壳，加密，解密？

VM 代码及分析

如果能模拟执行是不是更酷

VM 最难处理的部分 – 跳转逻辑

尽可能把 custom ISA 转化成你熟悉的 ISA

- 直接翻译成C
- disassembler 写得好甚至可以想办法塞回 ida

VM pwn ☺

混淆

obfuscation

credit to SGFVamll for many materials, really thanks

感觉一切增加理解难度的都可以称之为混淆

不妨来做一个 JS 逆向题先 （5min rush）

credit: https://obfuscator.io/

obfuscation built by many **techniques**

比如刚刚看的 js 代码

- **Identifiers Transformations**
  - 变量名简直治疗高血压
  - https://deobfuscate.io/

- **Strings Transformations**
  - 类的 string 方法通过 index 访问
  - .....

- **more over**

完全恢复还不如动态完成；（但由于语义的隐藏动态也并不方便

back to our native program

识别混淆类型，理解混淆逻辑，并进行去混淆

重点介绍三个混淆方式

- instruction subsititution
- bogus control flow
- control flow flattening

Instructions Substitution

The goal of this obfuscation technique simply consists in replacing standard binary operators (like **addition**, **subtraction** or **boolean** operators) **by functionally equivalent, but more complicated sequences of instructions.**

上一次学逆向专题是下周的上周的明天的昨天的大前天的 UTC+8 的 14:15

equals

上节逆向课星期一下午

addition

```
a = b + c -> a = b - (-c)
                or
                a = -(-b + (-c))

          -> a = b + ?
             a = a + c
             a = a - ?

       -> a = b - ?
          a = a + b
          a = a + r
```

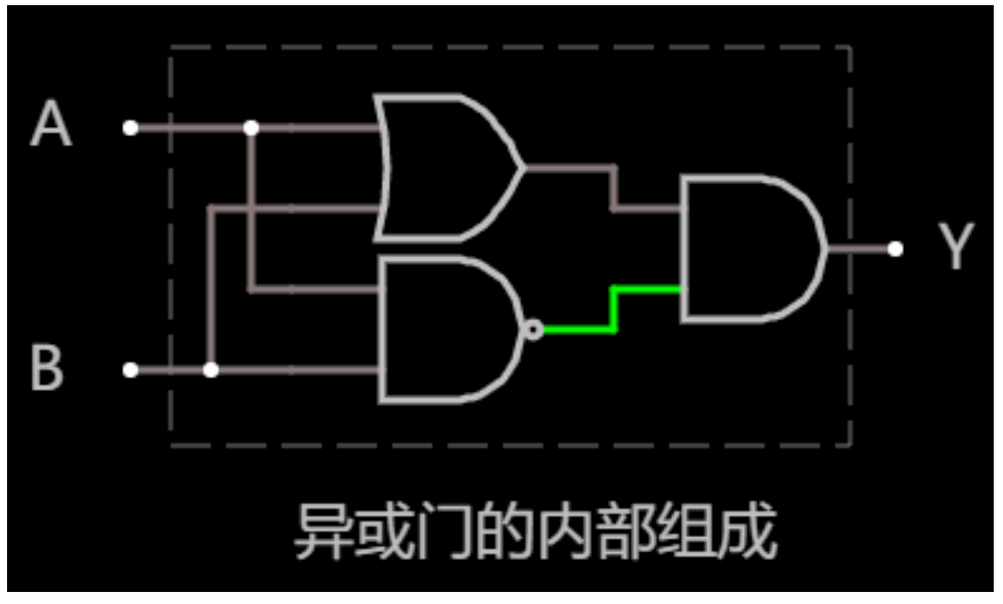
moreover

有数逻那感觉了

```
b & c = (b ^ ~c) & b
```

```
b | c = (b & c) | (b ^ c)
```

```
b ^ c = (~b & c) | (b & ~c)
```



手工完成混淆

```
#define ADD(x, y) (-(-x + (-y)))
#define SUB(x, y) (ADD(x, -y))
#define AND(x, y) ((x ^ ~y) & x)
#define OR(x, y)  ((x & y)|(x ^ y))
#define XOR(x, y) ((~x & y)|(x & ~y))
```

宏好像没啥用 why

and

```
uint32_t ADD(x, y) { return (-(-x + (-y))); }
uint32_t SUB(x, y) { return (ADD(x, -y)); }
uint32_t AND(x, y) { return ((x ^ ~y) & x); }
uint32_t OR(x, y) { return ((x & y) | (x ^ y)); }
uint32_t XOR(x, y) { return ((~x & y) | (x & ~y)); }
```

ollvm 一行搞定

你不懂我

靓仔，你懂我不?

套娃

```
b & c = (b ^ ~c) & b
      = ((~b & ~c) | (b & c)) & b
```

demo (attachment) ……

```
1  __int64 __fastcall sub_400686(unsigned int *a1, _DWORD *a2)
2  {
3    __int64 result; // rax
4    unsigned int v3; // [rsp+1Ch] [rbp-24h]
5    unsigned int v4; // [rsp+20h] [rbp-20h]
6    int v5; // [rsp+24h] [rbp-1Ch]
7    unsigned int i; // [rsp+28h] [rbp-18h]
8
9    v3 = *a1;
10   v4 = a1[1];
11   v5 = 0;
12   for ( i = 0; i <= 0x3F; ++i )
13   {
14     v5 += 1166789954;
15     v3 += (v4 + v5 + 11) ^ ((v4 << 6) + *a2) ^ ((v4 >> 9) + a2[1]) ^ 0x20;
16     v4 += (v3 + v5 + 20) ^ ((v3 << 6) + a2[2]) ^ ((v3 >> 9) + a2[3]) ^ 0x10;
17   }
18   *a1 = v3;
19   result = v4;
20   a1[1] = v4;
21   return result;
22 }
```

简单看看 pass 代码

咱也不懂 C++，也不懂 llvm，就当 C 代码看看咯

https://github.dev/bluesadi/Pluto-Obfuscator/blob/main/llvm/lib/Transforms/Obfuscation/Substitution.cpp

如何去 substitution 混淆？

1. 编译器优化
   - maybe useless at all

2. 经验性手动处理
   - 一杯茶 一包烟 …

3. 写脚本？
   - 简介一下思路和看一下效果

4. 要不，还是动调?

Bogus (虚假) Control Flow

This method modifies a **function call graph** by **adding a basic block before the current basic block**. This new basic block contains an **opaque predicate** and then makes a **conditional jump to the original basic block.**

就我个人来说，大前天课对我的意义，不能不说非常重大。 而这些并不是完全重要，更加重要的问题是， 就我个人来说.....不能不说非常重大。 对我个人而言，大前天课不仅仅是一个重大的事件，还可能会改变我的人生（from 狗屁不通文章生成器）**对了周一下午是不是上了逆向**

equals

上节逆向课在星期一下午

A Simple Example with demo

```c
#include <stdlib.h>

int main(int argc, char **argv)
{
    int a = atoi(argv[1]);
    if (a == 0)
        return 1;
    else
        return 10;
    return 0;
}
```

为啥 **opaque predicate**

编译器「感觉」这个全局变量可能有地方引用？不敢下判断

如何去 bogus control flow 混淆？

编译器优化赛高

- *注意对于**bss** 对象的修饰*

还有一个贼恶心的

Control Flow Flattening

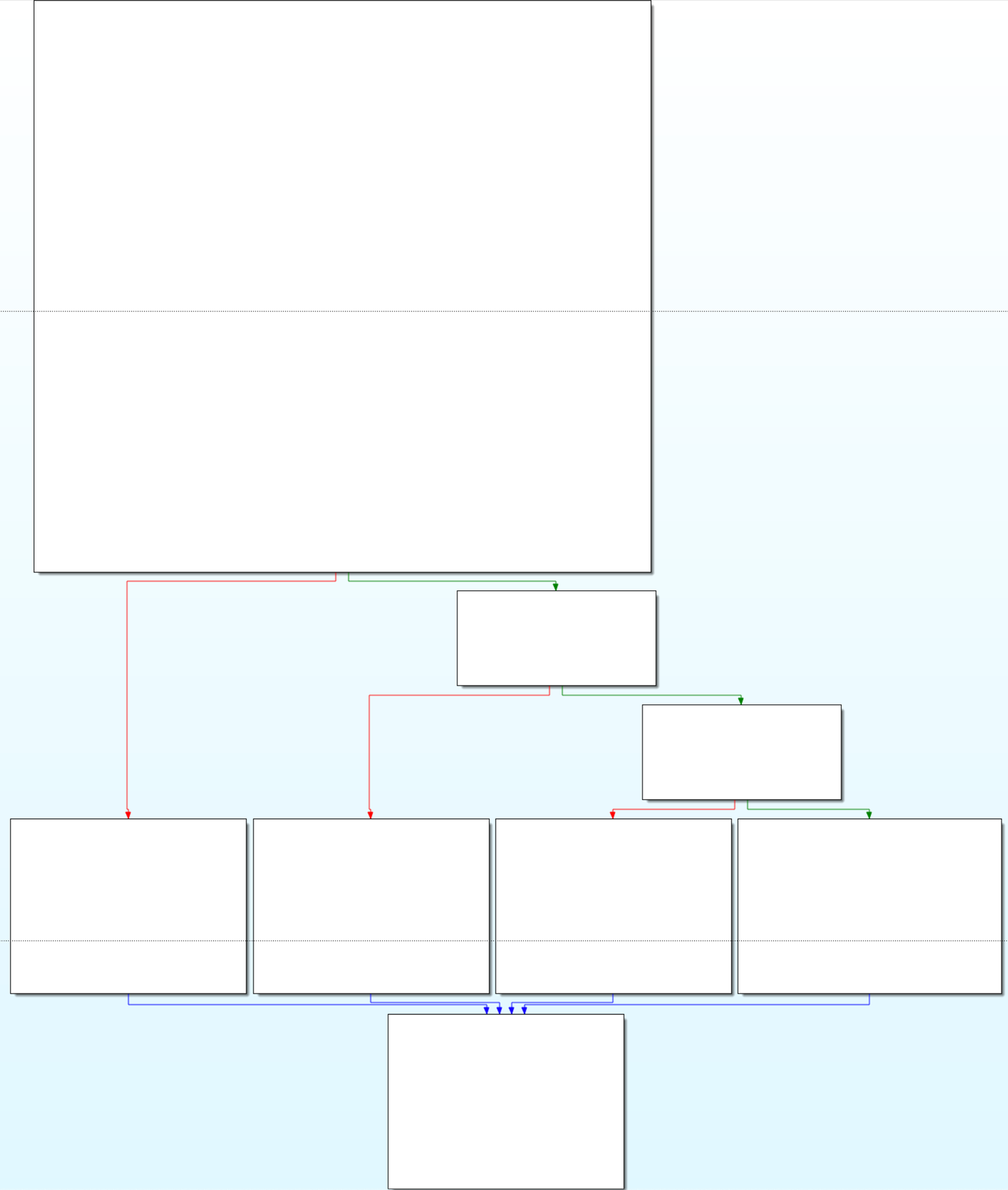
The purpose of this pass is to **completely flatten (**展开、展平**)** the control flow graph of a program.

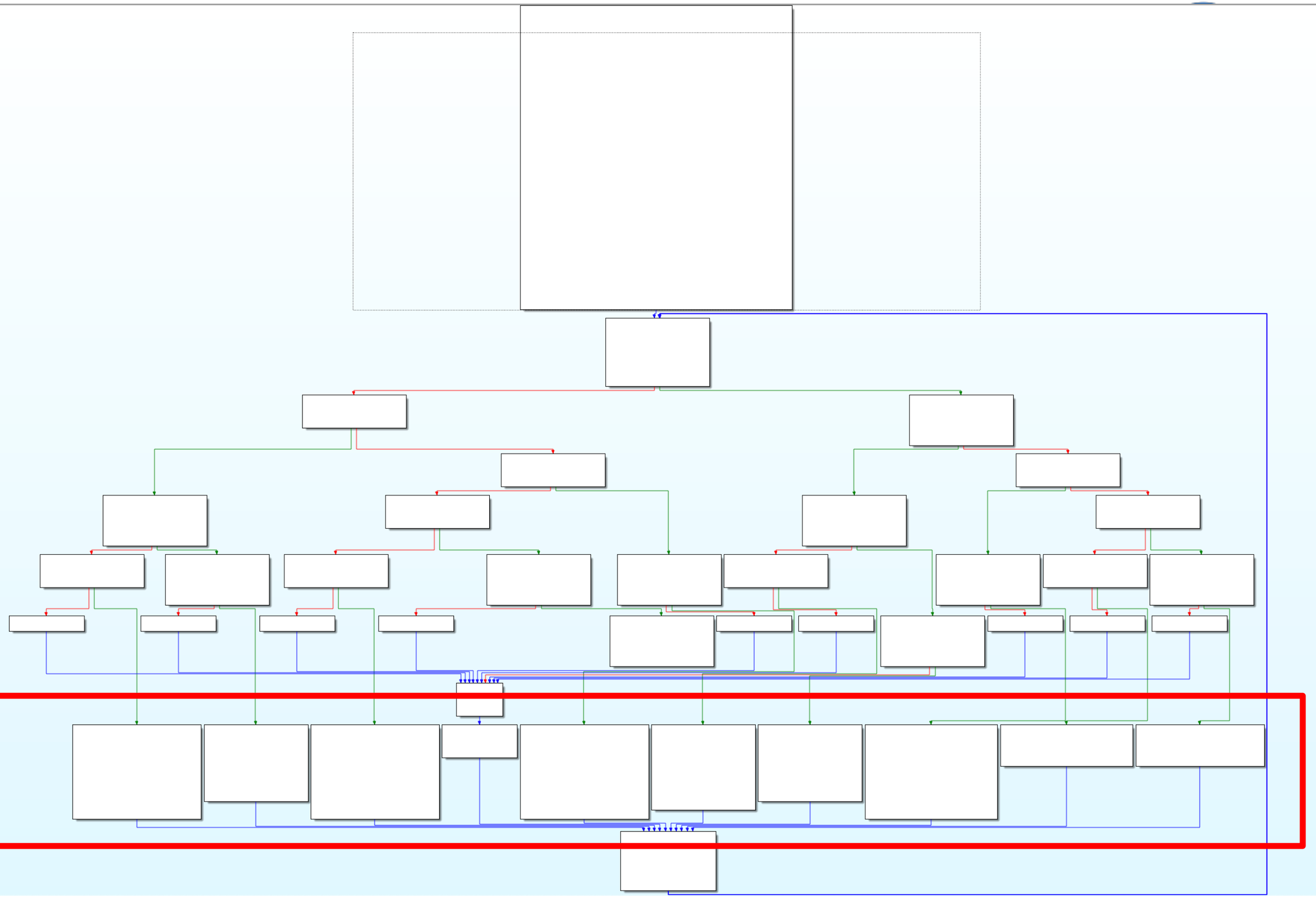
first proposed

László, Tımea, and Ákos Kiss. "Obfuscating C++ programs via control flow flattening." Annales Universitatis Scientarum Budapestinensis de Rolando Eötvös Nominatae, Sectio Computatorica 30.1 (2009): 3-19.

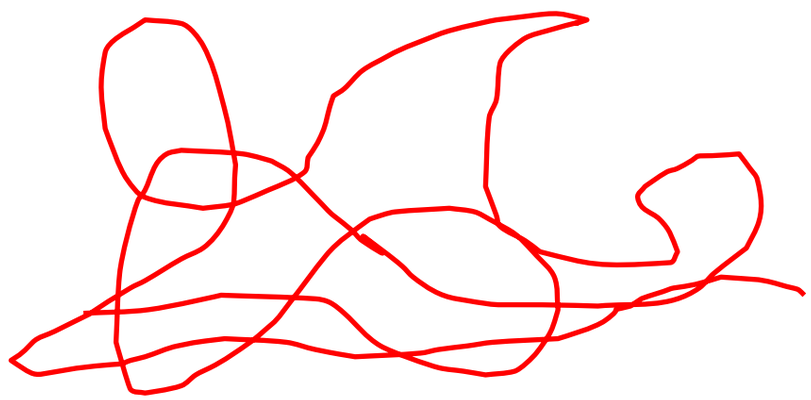
什么是展开? see demo

```c
int main(int argc, char *argv)
{
    unsigned int mod = argc % 4;
    unsigned int result = 0;

    if (mod == 0)
        result = (argc | 0xBAAAD0BF) * (2 ^ argc);

    else if (mod == 1)
        result = (argc & 0xBAAAD0BF) * (3 + argc);

    else if (mod == 2)
        result = (argc ^ 0xBAAAD0BF) * (4 | argc);

    else
        result = (argc + 0xBAAAD0BF) * (5 & argc);

    return result;
```





hence



一节上逆课在向星下午期

equals

上节逆向课在星期一下午

算法流程简述

1. 入口块连接到分发块（dispatchBB)，创建返回块

2. 其他基本块保存

3. 为每个保存的基本块分配 switch – case 值

4. 修改每个基本块末尾的跳转

如何去 fla 混淆 （个人jio的

- 先确定入口块和结束块内容

- 理清所有基本块
  - 理清其「**功能**」
  - 理清其「**约束**」
  - 理清其「**路由**」/「**跳转**」

- 写脚本恢复控制流，还原 C 代码
  - 或许也可以直接 ida 上 patch

当然，实战时可能动态 + log的做法更多更直接

人工去一下试试

虽然很慢很无聊，但是希望大家能 follow ☺

基础作业中相关的混淆你已经完全掌握了

还有其他混淆么

有，多得是 .....

- 异常处理
- 间接调用
- 字符串加密
- MBA (Mixed Boolean-Arithmetic)

或许还有时间

分享一些逆向心得

- 如何想办法也得跑起来
  - 保证逻辑一致

- 能动态拿到的东西不用自己写

- 字符串起大作用

- 找源代码

- 猜、猜、猜

- 试图去理解出题人