



纸 如 们 逆向基础

2017.8 北京/南京

Null/Tea Deliverers 管云超

(初音未来)

自我介绍



- 刚毕业的高中生
- 淮北邮大一新生
- Nu1L逆向/杂项选手
- Tea Deliverers摸鱼党
- 主攻Windows逆向、
软件保护、IDA扩展
- 热爱修电脑
- 业余Io娘+coser一枚
(+各种附加属性)
- (看我多原谅啊)

什么是逆向



- 有许多无辜的妹纸们
- 邪恶的肥宅们偷偷在她们体内植入了控制意识的东西（正向开发）
- 然后让一部分妹纸去服务其他妹纸（封装）
- 于是正义的我们决心要打破这样的格局，深入妹纸们的内心
- 用崭新的视角去看这个世界，改变这一切



是不是很神
奇？
是不是很有
趣？
所以废话少说
开(=^.=)枪(=^.=)



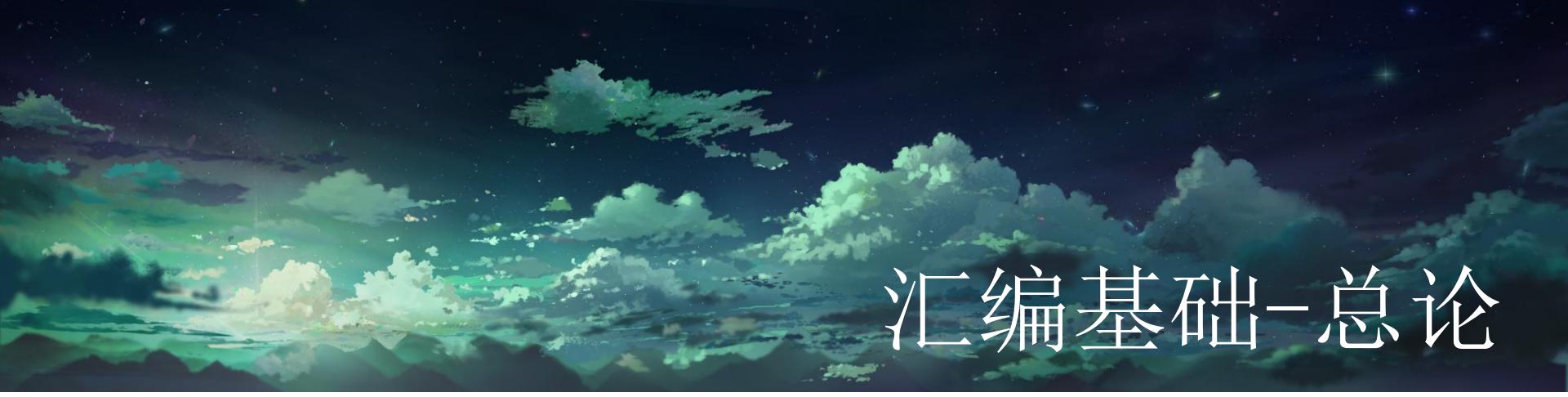
Duang !

- 我们来到了名为**CTF**的星球
- 啊，出现了一只萌妹
- 请选择操作：
- （无）
- 你把人家吓跑了
- 看起来我们需要学习一些技能呢（废话）

新手教程-目录

- 汇编基础
- 常见逆向思路
- 初级工具使用
- Demo & 实战





汇编基础-总论

- 想要救(gōng)出(lue)妹纸们,
- 我们就要首先认识并了解她们
- 机智的我们派人潜入了敌方阵营
- 目前计算机的常见指令集有:
 - x86/x64
 - ARM/ARM64 (AArch64)
 - 其他相对小众的架构 (如MIPS)

汇编基础-总论

- 汇编其实非常简单，将每一条指令单拿出来看，都是十分简单易懂的
- 将C代码拆开写，其实也就成了汇编
- 接着我们了解一下CPU的运行方式：
- 取出下一条要执行的指令+执行当前的指令+输出结果
- 什么都不管只会往前跑的蠢妹纸（一脸无辜）
- （都知道四驱车怎么跑的吧）





汇编基础-总论

- 常用的汇编指令只有：
- 运算指令：
 - 加减乘除 与或非 异或
- 数据转移指令：
 - 读取/写入不同长度的内存
 - 写入寄存器
- 跳转指令
 - 无条件跳转/条件跳转
- (栈操作指令)
 - 入栈、出栈

看呢，和C语言差不多吧
无非是把C语言简洁的运算符变成了一条指令了而已啦，童鞋们要有信心啊（拍肩

汇编基础-x86寄存器

- x86支持使用一个寄存器的不同长度的值，如下图所示（只包含通用寄存器）

AH	AL	EAX	RAX	R8	R8B	R8W	R8D	R12	R12B	R12W	R12D	
BH	BL	BX	EBX	RBX	R9	R9B	R9W	R9D	R13	R13B	R13W	R13D
CH	CL	CX	ECX	RCX	R10	R10B	R10W	R10D	R14	R14B	R14W	R14D
DH	DL	DX	EDX	RDX	R11	R11B	R11W	R11D	R15	R15B	R15W	R15D
BP	EBP	RBP	DI	EDI	RDI	SP	ESP	RIP				
SI	ESI	RSI	SP	ESP	RSP							

☆注意这张图这里打错了，是IP和EIP☆

其中：
RIP—指向下一条执行的命令
RSP—指向栈顶
RBP—指向栈帧中心
(稍后解释栈和栈帧)

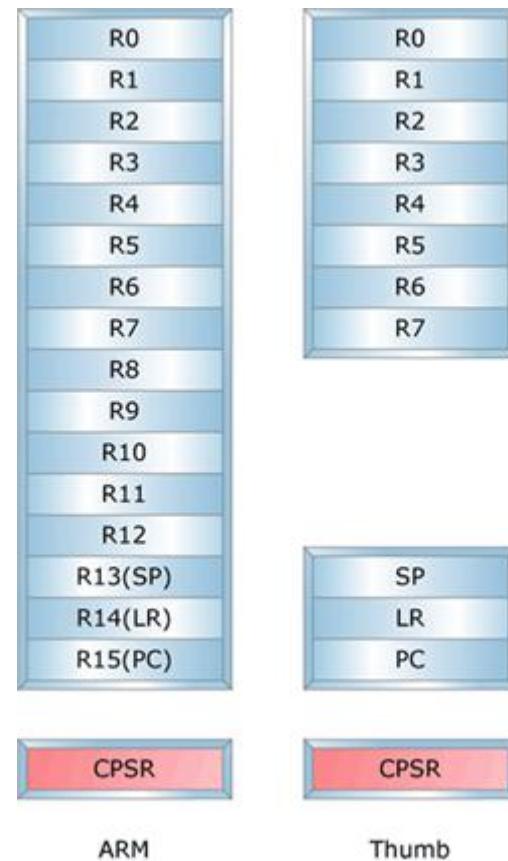
看这只Intel娘多萌
(^-^)/



汇编基础-ARM寄存器

- ARM寄存器

其中SP与x86中esp相当
R11与x86中ebp相当
LR是link register的简称，
用于保存调用者的地址方便返
回
PC在ARM中指向两条指令之后
(由于三级流水线的历史遗留
问题)



汇编基础-常用指

- X86和ARM的汇编指令多的惊人
- 但是只是知道常用的就够啦

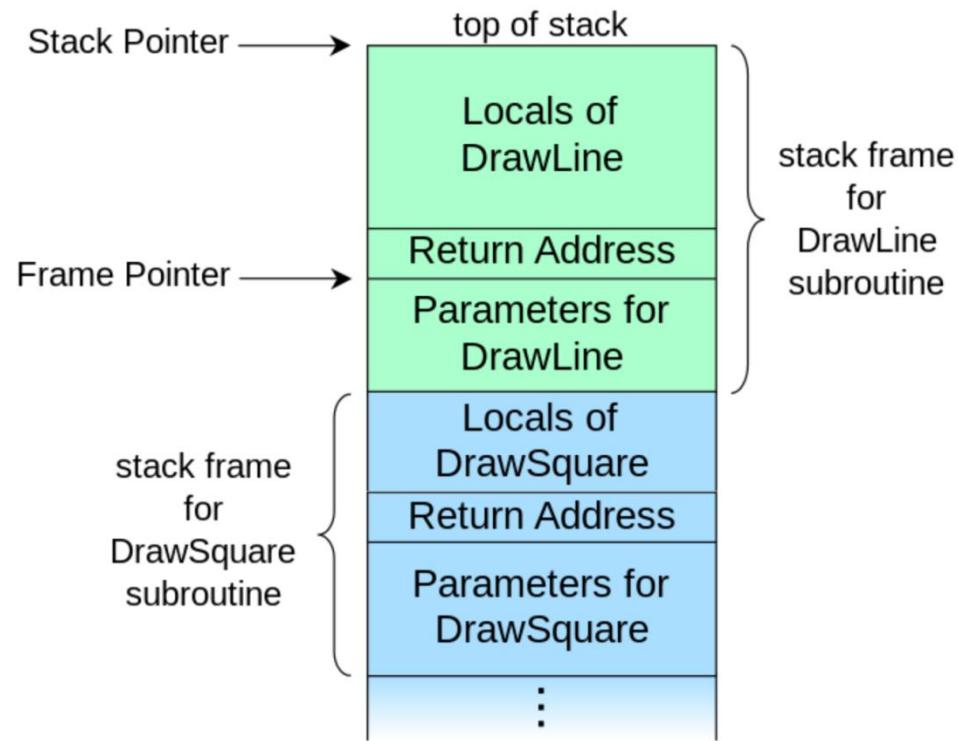
* 仅在部分ARM
指令集中支持

操作	X86	ARM	操作	X86	ARM
加	add	add	左移	shl/sal	lsl
减	sub	sub	逻辑右移	shr	lsr
乘	mul	imul	算术右移	sar	asr
除	idiv	sdiv udiv*	读内存	mul	str系列指令
与	and	and	写内存	idiv	str系列指令
或	or	orr	转移指令	mov	mov
非	not	mvn	跳转指令	jmp/jCC	b/bx/bCC
异或	xor	eor	函数调用	call	bl/blx



汇编基础-总论

- 一般指令执行时都会将返回的结果存储于寄存器中。但寄存器数量很有限，于是我们需要经常的将它存入内存。
- 然后聪明的先人就创造了栈。
- 大概后人也都觉得这个idea很不错于是大家都这样干了233333





开始勾搭
— 基本工具使
用

Repository

```
// marshal obj type of version 2
// version 2 is backward compatible to version 1 (for read)
enum <char> ObjType {
    TYPE_NULL      = '0',
    TYPE_NONE      = 'N',
};

char Header[16];
local unsigned int min data block = FileSize();
```

Variables

Name	Value	Start	Size	Color
char Header[16]	racrr	0h	10h	Fq: Bg:
struct Entry entry[0]		10h	24h	Fq: Bg:
int header	1	10h	4h	Fq: Bg:
int block addr	328	14h	4h	Fq: Bg:
int nameS	21	18h	4h	Fq: Bg:
char name[22]	lib particle candy.lu	1Ch	16h	Fq: Bg:
char padding[2]		32h	2h	Fq: Bg:
struct FileEntry file entry[0]		148h	8BD8h	Fq: Bg:
int header	2	148h	4h	Fq: Bg:
int block addr	35792	14Ch	4h	Fq: Bg:
int size	35786	150h	4h	Fq: Bg:
char data[35786]		154h	8BCAh	Fq: Bg:
char padding[2]		8D1Eh	2h	Fq: Bg:
struct Entry entry[1]		34h	14h	Fq: Bg:
struct FileEntry file entry[1]		8D20h	1074h	Fq: Bg:
struct Entry entry[2]		48h	18h	Fq: Bg:
struct FileEntry file entry[2]		104h	104h	Fq: Bg:
struct Entry entry[3]		248h	24h	Fq: Bg:
struct FileEntry file entry[3]		15040h	15040h	Fq: Bg:
struct Entry entry[4]		188h	14h	Fq: Bg:
struct FileEntry file entry[4]		21448h	52C0h	Fq: Bg:
struct Entry entry[5]		9Ch	14h	Fq: Bg:
struct FileEntry file entry[5]		26708h	C0F4h	Fq: Bg:
struct Entry entry[6]		B0h	1Ch	Fq: Bg:
struct FileEntry file entry[6]		327FCh	7A4h	Fq: Bg:
struct Entry entry[7]		CCh	38h	Fq: Bg:
File		20516h	10750h	Fq: Bg:

解析的结果会以列表的形式显示在Variables界面中

Inspector Variables Bookmarks F0 Functions

```
}while (FTell() < min_data_block);
    return 0;
}
```

初级工具使用

• 开始勾搭

CFF Explorer VIII - [e39768f6-bc0f-4999-8867-d7053c77ae2f.exe]

文件 设置 帮助

属性

属性	值
文件名	C:\Users\THSCSLab\Desktop\REMRAM SUMMER ayamy.exe
文件类型	可移植可执行32
文件信息	No match found.
文件大小	93.39 KB (95628 bytes)
PE 大小	67.00 KB (68608 bytes)
创建时间	星期二 01 八月 2017, 13.22
修改时间	星期二 01 八月 2017, 13.22
访问时间	星期二 01 八月 2017, 13.14
MD5	B58823BA022D810081C3
SHA-1	F06009EC39C534C1B95B

有许多方便的小功能会在以后用到

- <http://bbs.pediy.com/thread-141055.htm>





初级工具使用

- 引诱上床=非侵入式
- 4. 反汇编器
 - IDA: Interactive Disassembler
 - 反汇编界面使用
 - 不同界面说明
 - 设置Data、Code (d、c)
 - 查看Cross Reference (x)
 - 查看String List (Shift-F12)
 - F5反编译器使用
 - 设置Type (变量、函数) (Y)
 - 设置Calling Convention (Y)

初级工具使用



- ~~使用器具-侵入式~~
- 3. 调试器
 - 命令行调试器
 - gdb
 - WinDBG
 - 图形界面调试器
 - OllyDBG
 - x64dbg
 - IDA内置调试器
- 调试器的功能十分相似，都是单步、查看运行过程中变量的值

初级工具使用



- 于是一般会有如下操作
 - 步进
 - 单步执行，遇到call调用时会跟进而不是跳过去
 - 步过
 - 单步执行，遇到call调用时会和别的指令一样跳过去
 - 运行至指定位置
 - 相当于临时下断点，碰到循环时可以方便的跳过去
 - 运行
 - 直接跑起来，而不是一条一条指令执行

初级工具使用

• 使用器具-侵入式

- 3. 调试器
 - 命令行调试器

- gdb
 - 支持平台全面，有强大社区支持
 - 支持配合gdbserver进行远程调试
 - 社区维护大量插件



初级工具使用

- 使用器具-侵入式
 - 3. 调试器
 - 命令行调试器
 - gdb
 - WinDBG
 - 仅支持Win平台
 - 强于解析结构体，尤善于Windows内部结构的解析与显示
 - 同样支持插件，但数量相对较少
 - 支持Win内核调试
 - 支持Win远程调试



初级工具使用



- ~~使用器具-侵入式~~
- 3. 调试器
 - 图形界面调试器
 - OllyDBG
 - Win下老牌调试器
 - 有强大的社区，支持脚本和插件扩展
 - 仅支持32位
 - 快捷键：F8步过 F7步进 F4运行至 F9运行 F2下断点

初级工具使用



- ~~使用器具-侵入式~~
- 3. 调试器
 - 图形界面调试器
 - OllyDBG
 - x64dbg
 - 支持32和64位
 - 仍在活跃开发中
 - 插件相对较少
 - 快捷键与OD相同
 - IDA内置调试器
 - 快捷键与OD相同

初级工具使用

- 使用器具-侵入式
- 3. 调试器
 - 图形界面调试器
 - OllyDBG
 - x64dbg
 - IDA内置调试器
 - 支持多种后端:
 - » IDA内置调试器
 - » gdb、windbg
 - 支持远程调试
 - » 兼容gdbserver
 - » 官方支持ARM/ARM64、x86/x64指令集远程调试
 - » 支持Android iOS Win Linux
 - » 支持配合Hox-Rays调试





恋爱循环 -搭建调试环境

搭建调试环境

搭建IDA远程调试环境

-Linux

- 这个简单一些
- 配置好虚拟的网络为NAT(网络地址转换)模式或Host Only(仅主机)模式，关闭虚拟机防火墙
- 调试器选择Remote Linux Debugger，点击Debugger→Process Options，填入虚拟机的ip和端口即可开始调试啦~~
- 注意32位和64位区别哦



搭建调试环境

搭建IDA远程调试环境

– Android 虚拟机

- 请参照
<http://www.cnblogs.com/hellowzd/p/5858875.html>
搭建好Android SDK环境，
并新建模拟器
 - 注意要下载好ARMv7的System Image
- 利用adb push 将IDA安装目录下的android_server上传到/data/local/tmp



搭建调试环境

搭建IDA远程调试环境

- Android 虚拟机

- 在adb shell中进入上述目录并执行
 - chown 0.0 ./android_server
 - chmod 755 ./android_server
 - ./android_server
- 新开一个cmd窗口，执行adb forward tcp:23946 tcp:23946进行端口转发
- 在IDA的Debugger中选择Remote ARMLinux/Android Debugger，Debugger→Process Options中填入地址127.0.0.1端口23946即可开始调试



搭建调试环境

搭建IDA远程调试环境

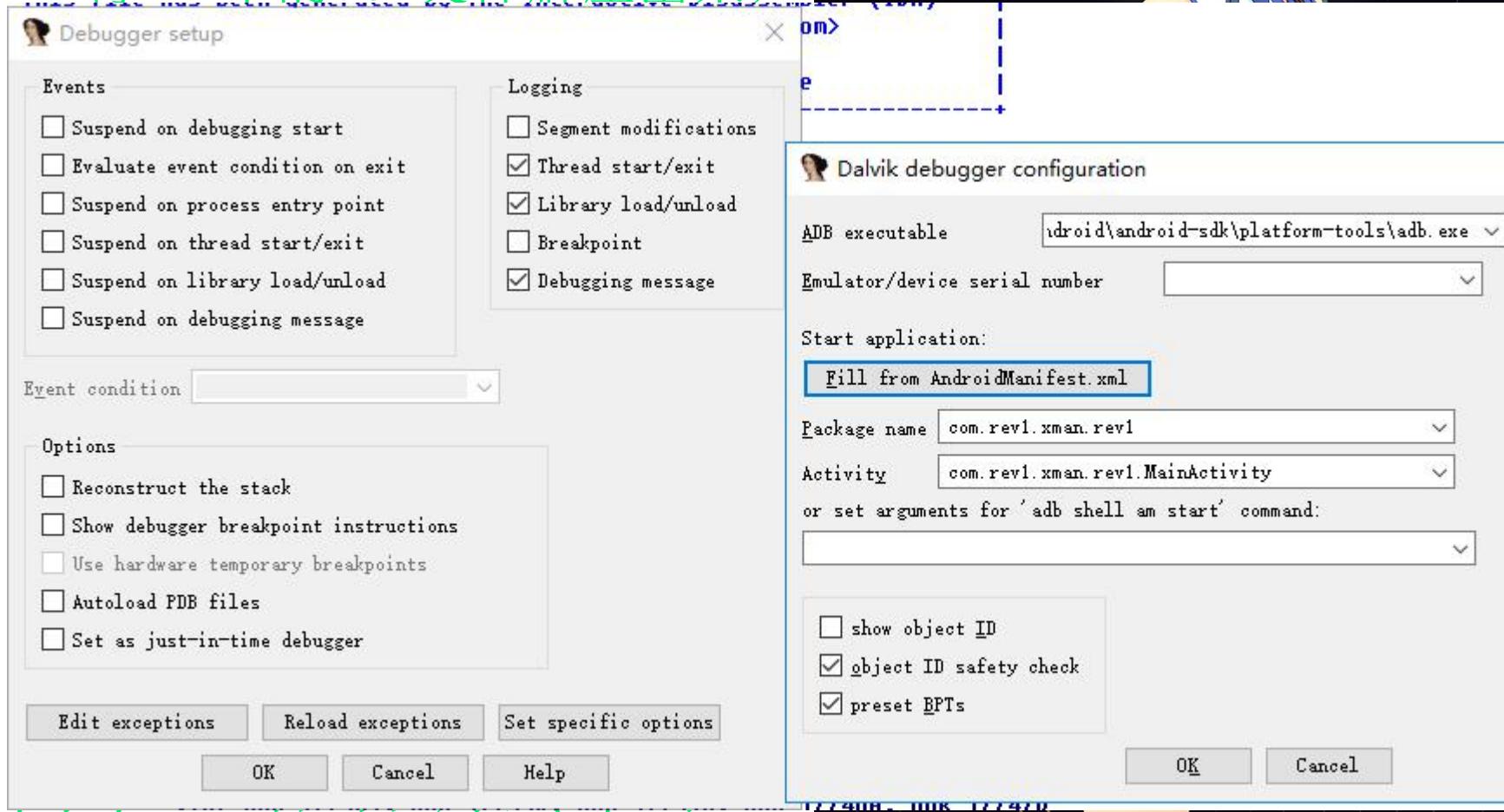
— Android真机

- 推荐先将机器Root
 - 否则大部分应用由于 debuggable=false 而无法调试
- 安装Xposed框架
- 安装XInstaller，在其他设置中勾选调试应用
 - 这样才能调试应用
- 剩余步骤同虚拟机



调试流程

- IDA调试Android程序
 - 将APK中的dex文件解压出来,





放下戒心
— 去除软件保
护





去除软件保护

- 1. 僵壳
 - PEiD、ExeInfo.
- 2. 脱壳
 - 搜索脱壳机
 - 热门壳: UPX、ASPack.
 - ESP定律快速脱壳
 - 只针对压缩壳
- 3. 去除花指令
 - 使用OllyDBG脚本
 - 手动总结特征码+修改
- 4. 去除混淆

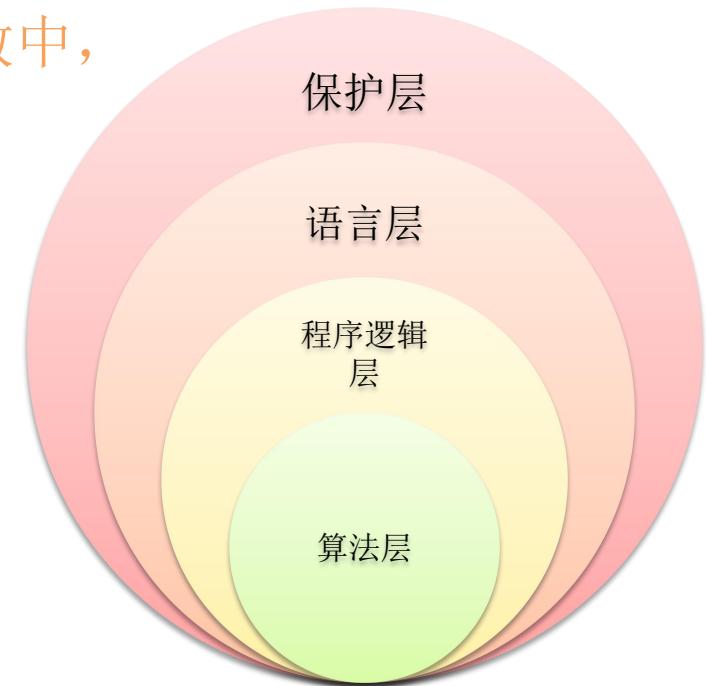


心意相连
-定位验证代码



定位验证代码

- 定位验证代码前
- 我们先需要知道肥宅们的意图
- 肥宅的目的是要我们拿到flag
- 而flag就藏在验证函数中，
- 于是我们的任务就是
 - 破除保护外壳
 - 理解程序逻辑
 - 找到验证函数
 - 逆推得到flag
- (肥宅的目的分明是找到萌妹然后prpr)





定位验证代码

- 1. 正面长驱直入
 - 从程序的入口点开始，逐步分析
 - 层层深入最后抵达验证函数
 - 静态分析（生撸）（硬肛）
- 2. 从信息输入/输出处寻找
 - 查找引用了输入输出函数的位置
 - 进而回溯找到验证函数
- 3. 利用字符串寻找
 - 寻找关键字符串位置从而找到验证函数

古人云 .. 百合无限好 只是生不了~~~

心心相印 —— 常见算法分析与逆向





算法分析与逆向

CTF中常见的类型有：

- 1. 没算法：
 - 直接输出flag
- 2. 常见的算法：
 - 简单异或
 - 带雪崩效应的异或
 - 加密算法（RSA、AES）
 - 散列算法（MD5、SHA1）
 - 解方程



算法分析与逆向

- 3. 有趣的算法：
 - 走迷宫
 - (各种脑洞)



经验加成
-加速CTF解题

外星人做法



- 1. 边信道攻击
 - CTF中可以使用pintools
 - 原理: 监测程序执行指令数
 - 应用:
 - 逐字节验证的题目
- 2. Google大法好
 - 加双引号搜索S-Box等常量快速确定算法

外星人做法

- 3. 逆向小trick
 - 快速找main位置：
 - 寻找到一个**大**跳转
 - 快速定位关键位置
 - 从Function List靠前的位置开始乱翻
 - 从main函数旁边翻
 - 编译时不同的源文件会被分别编译为.o再由编译器：合并
 - 编译命令行中标准库一般在最后面





外星人做法

- 3. 逆向小trick

- 应对MFC程序：

- 使用 `xspy` 工具查看消息处理函数

- 在看雪论坛中搜索`xspy`即可下载

- 极其傻瓜化的一个工具

- 拖上去看感兴趣的函数就可以了

- 比如`OnClick` `OnCommand`等

- 手动加载Signature

- 碰到无法自动识别库函数时

- (`Shift - F5`) `View` → `Open Subviews` → `Signatures`

- (`Shift - F11`) `View` → `Open Subviews` → `Signatures`



外星人做法

- 4. 调试小技巧
 - 如何得知MessageBox后在程序在哪里继续运行
 - 在0D或x64dbg中找到内存布局列表（0D：Alt-M或查看→内存）
(x64dbg：在窗口栏点击内存布局)
 - 找到自己程序的代码段（通常
是. text，按F2）（下区段断点）
 - 返回程序点击确定即可

CTF讲解时间~~





天空中划过一道流星
新的世界即将到来
CTF未来趋势



残酷现实 -真实的逆 向

欢迎回到地球~~

CTF V. S. 现实

CTF:

- 代码量小
- 结构简单
 - 单文件
- 编码单一
- 现代语言特性少
 - 面向过程
- 加密壳/优化少
- 语言常见
 - C C++ ASM





CTF V. S. 现实

现实：

- 代码量巨大
- 结构复杂
 - 大量静态库 动态库
- 各种乱码
- 大量现代语言特性
 - C++、Template
- 优化和加密壳十分常见
- 语言可能十分神奇
 - Go VB Delphi



砾戈秣马
-IDA高级使
用

IDA高级使用

- 1. 设置字符串编码&格式
 - 快捷键: Alt-A
 - 可以设置字符串类型
 - Unicode字符串 (WCS)
 - 多字节字符串 (MBS)
 - 其他...
 - 可以设置字符串编码
 - 需要系统支持对应编码



IDA高级使用

- 2. 导入/导出数据
 - 快捷键: Shift-E
 - 菜单名:
 - Edit→Import Data
 - Edit→Export Data
 - 操作:
 - 选定后按快捷键
 - 可以方便地提取数据或修改idb数据库



IDA高级使用

- 3. 选定大段数据
 - 快捷键:Alt-L
 - 菜单名:
 - Edit→Begin Selection
 - 将标定选择起始点
 - 之后可随意滚动或按g跳转到结束位置
 - (这样就用一直傻傻的滚动了)



IDA高级使用

- 4. 批量应用类型
 - 组合技
 - 操作：
 - 设置好第一个的类型
 - 利用刚才的技巧选定数据
 - 如果不选定数据，IDA会尽可能多的设置类型
 - 按*或按d弹出建立数组对话框
 - 不勾选Create as array
 - 大量减少工作量XD



IDA高级使用

- 5. 设置间接跳转地址
 - 快捷键: Alt-F11
 - 菜单:
 - Edit→Plugins→Change the callee address
 - 将利用动态调试获取到的调用/跳转地址填入，可以极大地帮助IDA和F5的分析（比如参数个数、调用约定什么的），获得更准确的结果



Cassandra
Pixiv Fantasia RD

IDA高级使用

- 6. 修复跳转表
 - 快捷键：
 - 默认无，可手动设置
 - 菜单：
 - Edit→Other→Specify switch idiom
 - 当程序存在PIE时可能会导致跳转表分析失败，于是需要我们手动修复来获得更好的分析结果



IDA高级使用

- 7. IDAPython
 - IDA自带支持脚本
 - 可以使用几乎所有IDA提供的API
 - 可以快速完成大量重复性操作
 - 可以方便的了解IDA内部的数据和结构



Cassandra
Pixiv Fantasia RD



妙手回春

-HexRays出错处理

F5出错处理



- 常见F5出错信息：
格式: XXX: XXX
 - positive sp value
 - call analysis failed
 - cannot convert to microcode
 - stack frame is too big
 - local variable allocation failed
- F5结果不正确

F5出错处理



- 1. positive sp value
 - 成因：IDA会自动分析SP寄存器的变化量，由于缺少调用约定、参数个数等信息，导致分析出错
 - 解决方案：
 - 推荐：在 Option→Generala中设置显示Stack pointer，然后去检查对应地址附近调用的函数的调用约定以及栈指针变化
 - 不推荐：在对应地址处按 Alt-K然后输入一个较大的负值（有风险）

F5出错处理

- 2. call analysis failed
 - 成因: F5在分析调用时，未能成功解析参数位置/参数个数
 - 解决方案:
 - 对于间接调用(类似call eax等)，可使用之前讲过的设置调用地址的方法解决
 - 对于直接调用，查看调用目标的type是否正确设置。可变参数是引发这种错误的主要原因之一



F5出错处理



- 3. cannot convert to microcode
 - 成因：部分指令无法被反编译
 - 解决方案：
 - 最常见起因是函数中间有未设置成指令的数据字节，按c将其设置为指令即可
 - 其次常见的是x86中的rep前缀，比如repXX jmp等，可以将该指令的第一个字节（repXX前缀的对应位置）patch为0x90（NOP）

F5出错处理



- 4. stack frame is too big
 - 成因：在分析栈帧时，IDA出现异常，导致分析出错
 - 解决方案：
 - 找到明显不合常理的stack variable offset，双击进入栈帧界面，按u键删除对应stack variable
 - 如果是壳导致的原因，先用OD等软件脱壳
 - 可能由花指令导致，请手动或自动检查并去除花指令

—十分罕见



F5出错处理

5. local variable allocation failed

- 成因：分析函数时，有部分变量对应的区域发生重叠，多见于ARM平台出现Point、Rect等8字节、16字节、32字节结构时尤其多见
- 解决方案：
 - 1. 修改对应参数为多个int
 - 2. 修改ida安装目录下的hexrays.cfg中的HO_IGNORE_OVERLAPS



F5出错处理

6. F5分析结果不正确

- 成因: F5会自动删除其认为不可能到达的死代码
- 常见起因是一个函数错误的被标注成了noreturn函数
- 解决方案:
 - 1. 进到目前反编译结果, 找到最后被调用的函数(被错误分析的函数), 双击进入, 再返回(迫使HexRays重新分析相应函数)
 - 2. 如果上述方案不成功, 那么进到被错误分析的函数, 按Tab切换到反汇编界面, 按Alt-P进入界面取消函数的Does not return属性



F5出错处理

6. F5分析结果不正确

- 成因: F5会自动删除其认为不可能到达的死代码
- 常见起因是一个函数错误的被标注成了noreturn函数
- 解决方案:
 - 1. 进到目前反编译结果，找到最后被调用的函数(被错误分析的函数)，双击进入，再返回(迫使HexRays重新分析相应函数)
 - 2. 如果上述方案不成功，那么进到被错误分析的函数，按Tab切换到反汇编界面，按Alt-P进入界面取消函数的Does not return属性

F-[ízəm] Vol.10

Murakami Suigun's Fetishistic Illustrations

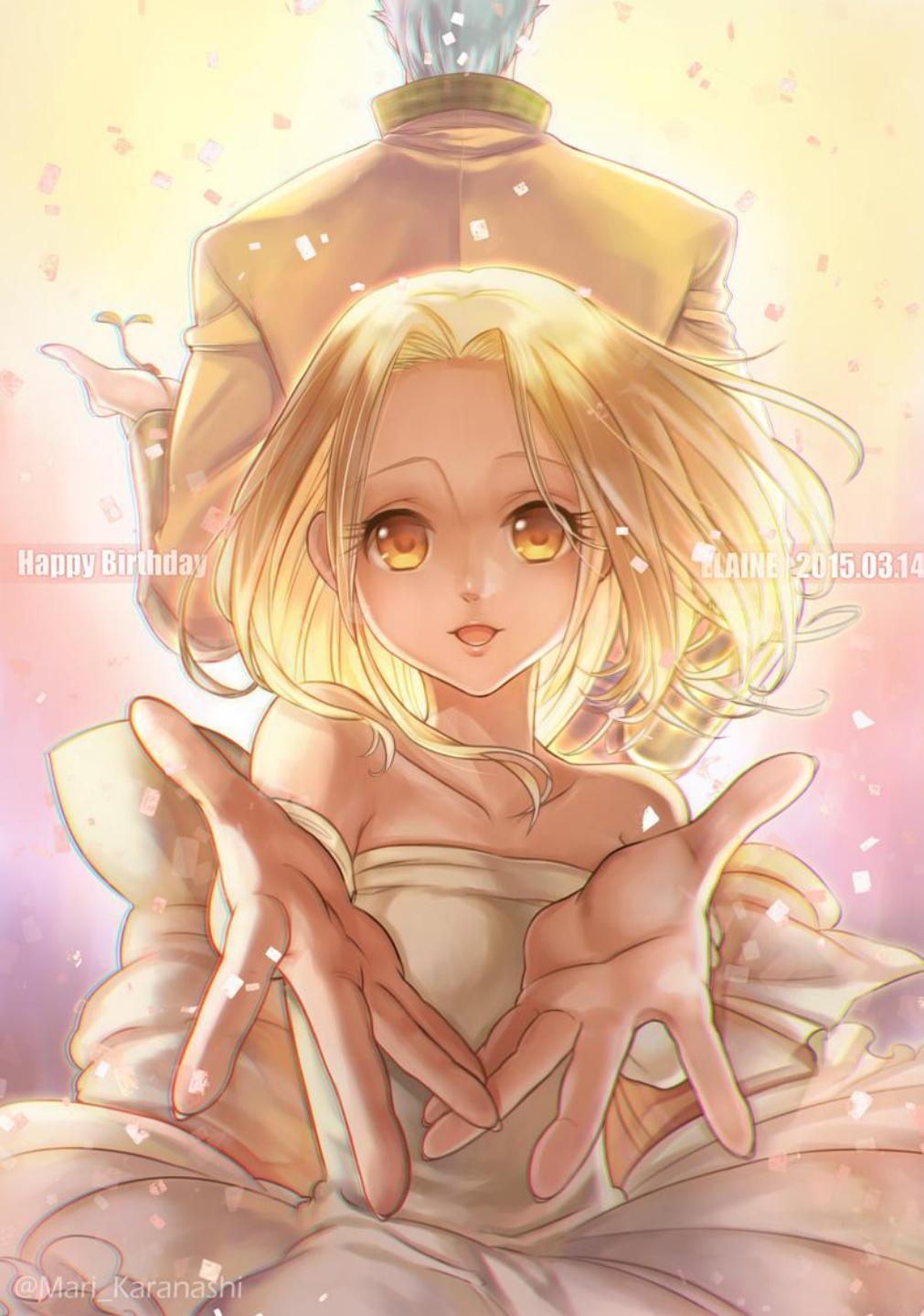
砾戈秣马

-HexRays高级使用



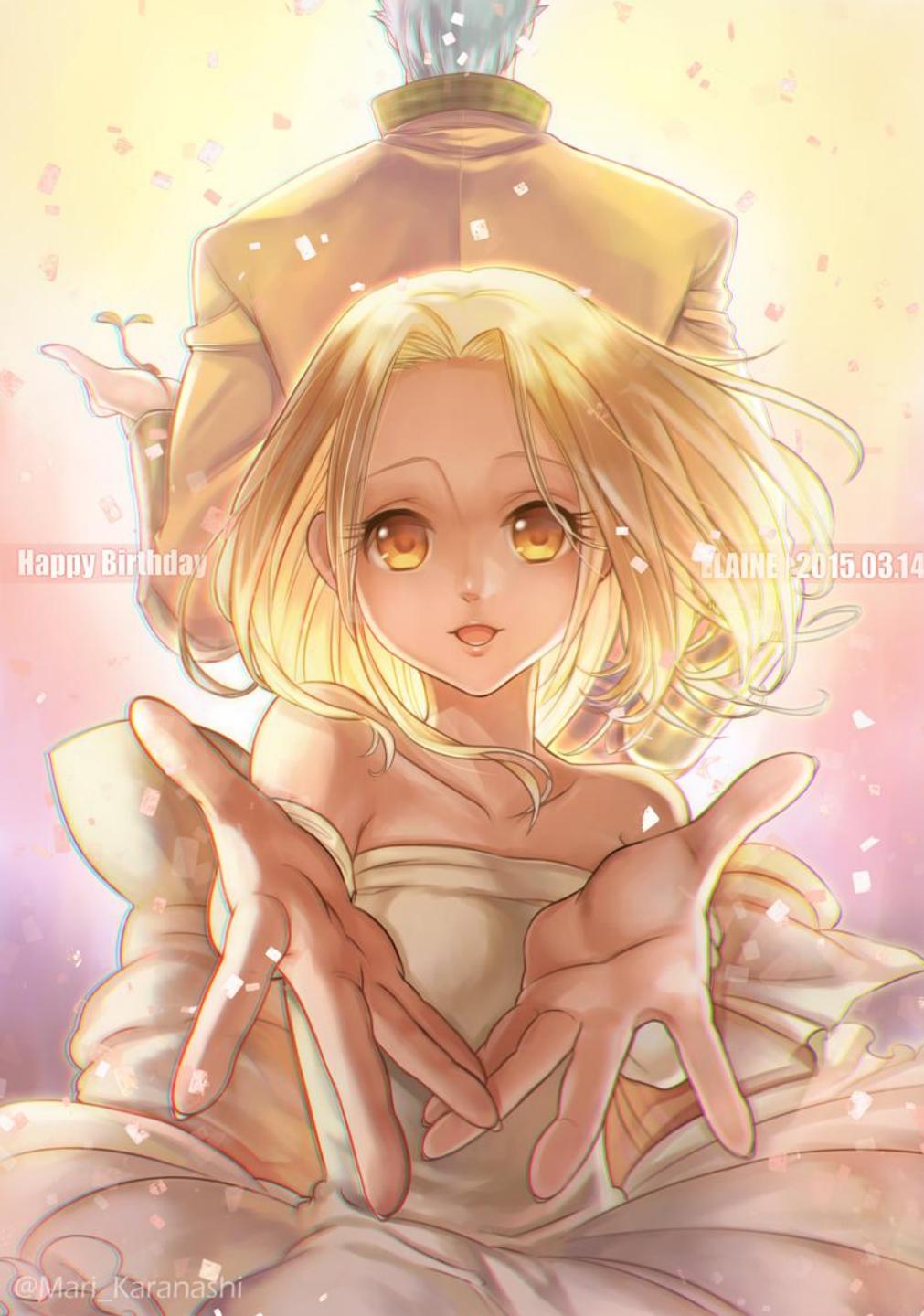
F5高级使用

- 1. 自定义寄存器传参
 - 组合技
 - 使用IDA中的_usercall和_userpurge调用约定（两个下划线）
 - 设置范例
 - int test<eax>(int a1<ebx>) ;
 - <>中为对应值的位置
 - 第一个<>中为返回值位置，注意返回值的位置即使不变也要填



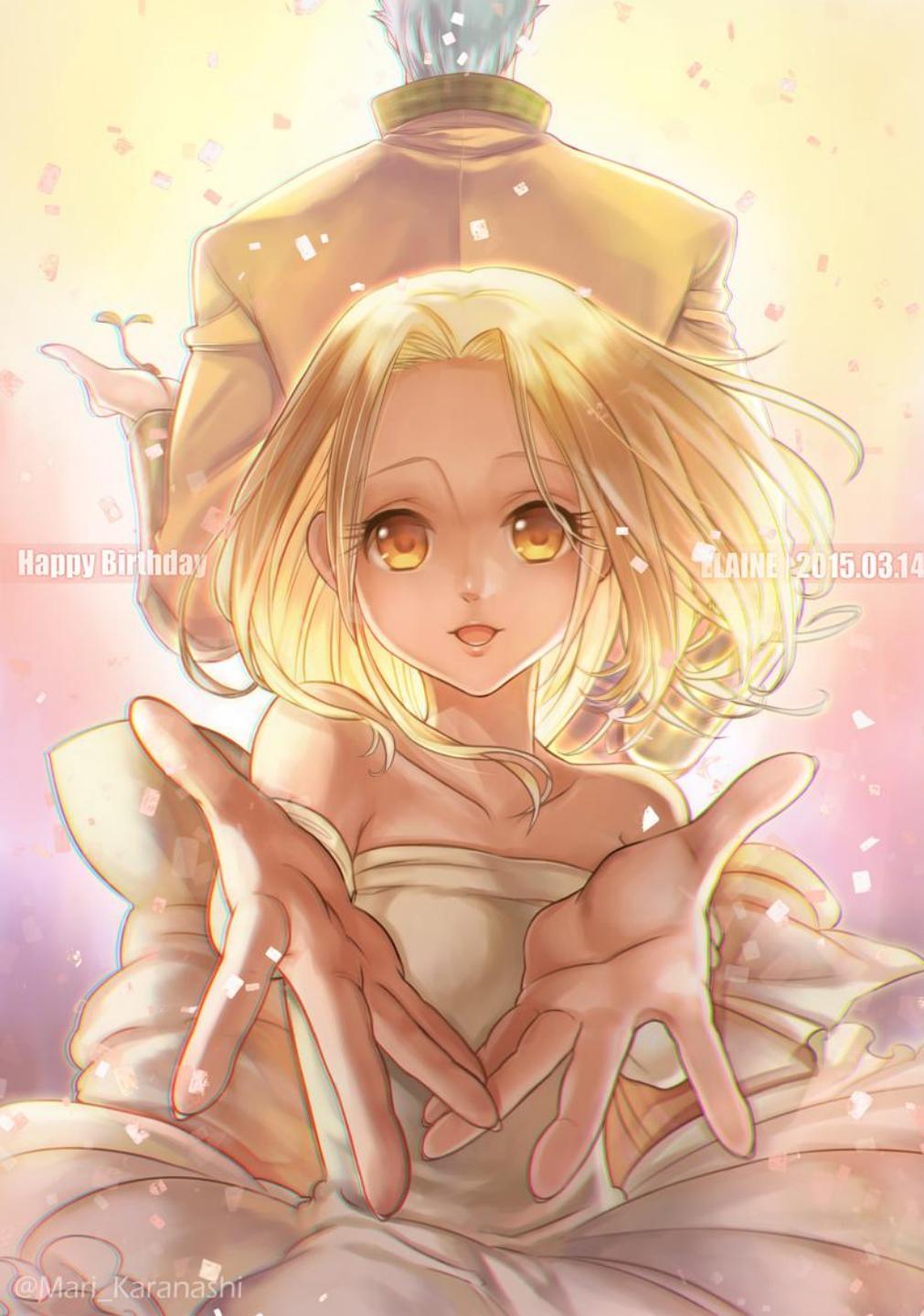
F5高级使用

- 2. HexRays源码级调试
 - 与IDA的调试过程相似
 - 点击在反编译行前的圆圈即可下断点
 - 鼠标移到对应变量上可显示对应值
 - Debugger→Debugger windows→Locals可以查看所有变量的列表
 - 可以在F5界面中使用各种调试操作如单步、运行至给定位置



F5高级使用

- 2. HexRays源码级调试
 - 注意：
 - F5中显示的变量很可能不是变量原来的值，尤其是寄存器变量，尽量在赋值位置断下并查看值
 - 对于优化后的代码，F5调试中bug很多
 - F5的单步不太稳定，在ARM平台可能会跑飞
 - F5的运行至指定位置功能同样不稳定，可能会跑飞



见招拆招
—编写IDA
Processor





IDA Processor

- 请童鞋们打开
- <http://github.com/gyc990326/IDABinaryTranslator>
- <https://github.com/gyc990326/cLEMENCyTools>
- 未整理，凑合看啦~



见招拆招

-IDA 手动加载



Manual Load

- 遇到特殊情况，例如
 - 32位和64位代码混合时
 - 程序没有对应的加载器时，就需要我们进行手动加载（Manual Load）
- 在打开文件后的对话框中，勾选Manual Load即可

见招拆招
件

FIDA 加载头文
vol.10

Murakami Suigun's Fetishistic Illustrations



导入头文件



- 如果能拿到程序的一部分头文件，比如程序公开的API接口，那么可以通过头文件让IDA了解这个结构并辅助分析
- IDA只支持导入C的头文件，所以C++中的模板、类什么的需要转换
- 对于没有虚函数的class，可以将成员函数直接删除，并改掉class private public等关键字，将其转为C结构体



见招拆招
-恢复部分符
号



恢复符号

- 1. 找程序的旧版本
 - 大量的程序早期版本中**安全意识不强**，没有删除符号等各种信息，可以利用搜索引擎搜索旧版本参考利用
 - 对于苹果应用，可以使用AppAdmin获取低版本应用



恢复符号

- 2. Rizzo匹配
 - <https://github.com/dvttys0/ida/tree/master/plugins/rizzo>
 - 这种方法对于冷门平台的程序有奇效
 - 今年DEFCON的新平台中，Rizzo十分成功的匹配了大量函数
 - 可以搜索并借用前人的工作成果



恢复符号

- 3. 看程序自带的 string
 - 许多程序**自带了大量调试信息**以方便程序员调试，然而在发布时它们有的**并没有被去除**，于是函数的基本功能和名称就轻松拿到啦~~



恢复符号

- 4. Google搜索源代码
 - 许多程序早期版本或其他分支源码被流出过
 - 典型例子：CS、iOS
 - 大量库开源
 - 按获得信息搜索源码（因为大量程序其实）
 - 直接套用Stack Overflow答案
 - 直接套用CSDN代码

恢复符号

- 5. 自己制作
Signature
 - IDA提供自动制作
Signature的工具
 - 请打开IDASDK68文件夹，
找到flair68文件夹





见招拆招

-多角度联合切入

多角度切入

- 一个功能会有多个切入位置
- 例如想让Win10 14393允许关闭UAC时运行Windows Store App就有如下切入点：
 - (反向)注册表中EnableLUA键值控制UAC开闭
 - (反向)“无法使用内置管理员帐户”提示
 - (正向)从Store App启动流程入手
 - Calc. exe



多角度切入

- 这么多角度，最后有成有败：
- 未能从Enable LUA键值深入
- 提示文本处于判断的下游，之前的调用流程因为RPC而不可见
- Calc.exe和AppxLauncher最后调用了相同的接口，通过单步跟踪确定到关键位置





多角度切入

- 然后调用流程大概是这样的。 。 。 。 :

```
1 (launcher)      ShellExecute...etc.  
2 (launcher)      IApplicationActivationManager::ActivateApplication::::  
3 (ActXPrxy.dll) ===RPC====  
4 (sihost.exe)    twinui.appcore.dll::CAplicationActivationManager:::  
5 (ActXPrxy.dll) ===RPC====  
6 (sihost.exe)    ActivationManager.dll::Execution::ActivationManagerShim::ActivateApplicationForProtocol  
7 (sihost.exe)    ActivationManager.dll::Execution::ActivationManagerShim::ActivateApplicationForContractByAcid  
8 (sihost.exe)    ActivationManager.dll::Execution::ActivationManagerShim::ActivateApplicationForContractByAcidAsUserWithHost  
9 (sihost.exe)    ActivationManager.dll::Execution::ActivationManagerShim::_ActivateApplicationForContractByAcid  
10 (sihost.exe)   twinui.appcore.dll::CAplicationActivationManager::ActivateApplicationForContractByAcidAsUserWithHost  
11 (sihost.exe)   twinui.appcore.dll::CAplicationActivationManager::_ActivateExtensionForContractHelper  
12 (sihost.exe)   twinui.appcore.dll::CAplicationActivationManager::s_ResolveMonitorCheckRequirements  
13 (ActXPrxy.dll) ===RPC====  
14 (explorer.exe) twinui.dll::IActivationErrorPopupFactory:::::  
15 (explorer.exe) twinui.dll::CActivationErrorPopupFactory::VerifyNotElevated  
16 (explorer.exe) twinui.dll::::|
```

<https://gist.github.com/gyc990326/3bcfa09eaa900221803a1f603b486da1>

- 
- 所以如果在前面两条线路
上继续纠结，死的就很
惨。。



见招拆招
一直觉硬上

直觉硬上



- 如何区分用户代码：
 - 用户代码一般在程序靠前的地方
 - 用户代码一般不会有奇怪算法
 - ~~一般程序员都不是算法超人~~
 - 库函数同样分布在一起，如果发现了`_S_construct`之类的典型库函数字符串，那么那一片就都不用看了



见招拆招 -搞定虚表

jomzi



搞定虚表

- 虚表是个业界难题
- 虚函数的处理会因编译器采取的ABI的不同而不同
- IDA本身不支持虚表的各种操作，而且似乎并没有要支持的意思，于是有各路大神开发了插件
 - HexraysCodeXplorer
 - hexrays_tool
 - HexRaysPyTools
- 最后的一个经过测试效果最好~~



搞定虚表

首先是安装插件.....

- 快捷键: Shift-F
- 菜单: 右键→Deep Scan Variable
- 操作:
 - 在构造函数中执行上述操作, 等待扫描完毕
 - 按Alt-F8打开Structure Builder, 看到有黄色部分后选择不正确的field点Disable
 - 清除所有的黄色部分后即可点击Finalize, 在新弹出的窗口中修改名称



最后忠告
-保护自己

保护自己



- 在逆向现实世界的软件时的几点小提示：
 - 先把程序的后缀名去掉
 - 避免意外误运行
 - 绝对不要在自己的真机上运行程序
 - 部分软件，尤其是国内的外挂和辅助，有许多暗桩（会偷偷执行的代码），其中不乏重启关机甚至格盘的样本
 - 推荐打开360或与其同等强度的HIPS防御软件（开到最大！！！）

• 推荐书目：感谢聆听~~
《加密与数学》
如果有问题的话可以来问哦
欢迎小伙伴们来勾搭
《软件调试》
《IDA Pro权威指南》~

