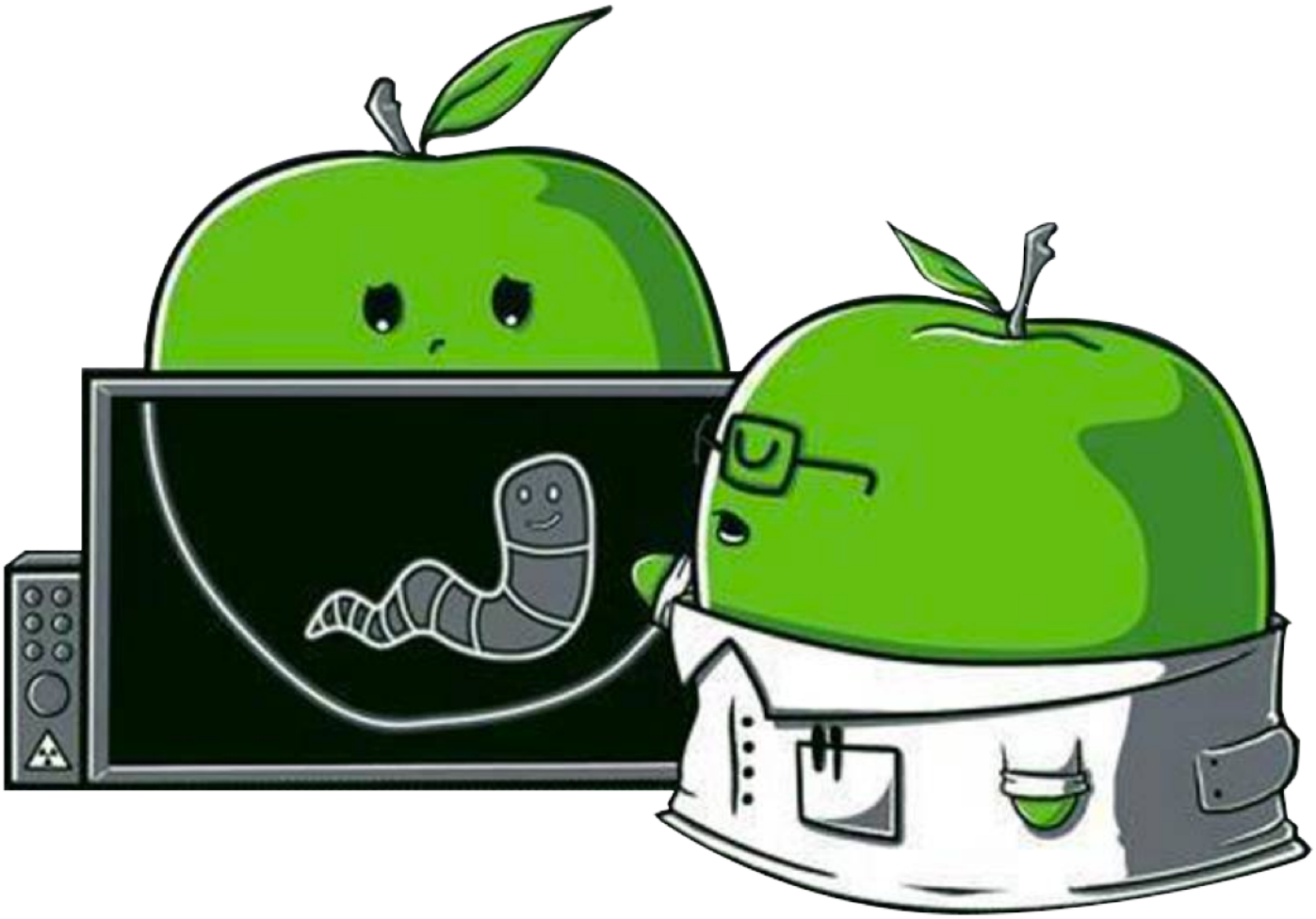


第0x7章：反编译和反编译

笔记:

这本书正在进行中。

我们鼓励您直接在这些页面上发表评论 ...建议编辑、更正和/或其他内容！

要发表评论，只需突出显示任何内容，然后单击 出现在屏幕上的图标
就在文档的边界上）。

根据定义，Mach-O二进制文件是“二进制的” ...这意味着，虽然计算机很容易读取，但它们编译的二进制代码并不是为人类直接读取而设计的。

由于绝大多数Mach-O恶意软件仅以这种编译的二进制形式“可用”（即其源代码不可用），我们作为恶意软件分析师依赖于能够从此类二进制文件中提取有意义信息的工具。

在前一章中，我们介绍了各种静态分析工具，它们可以帮助对未知的Mach-O二进制文件进行分类。然而，如果我们真的想全面了解Mach-O二进制文件（例如，一个似乎是Mac恶意软件新部件的样本），就需要其他更复杂的工具。

高级逆向工程工具提供了反汇编、反编译（甚至动态调试）二进制文件的能力。在本章中，我们将坚持反汇编和反编译的静态分析方法（尽管在后面的章节中，我们也将介绍动态调试）。虽然这些工具至少需要对低级反转概念（如汇编代码）有一个基本的了解，并且可能会导致耗时的分析会话，但它们的分析能力是无价的，无与伦比的。即使是最复杂的恶意软件样本也无法与使用这些工具的熟练分析师匹敌！

在讨论反汇编程序和反编译程序的细节之前，需要对汇编代码进行一次简短的探讨。

汇编语言基础

笔记:

整本书都是关于二进制代码的反汇编和汇编语言的。

在这里，我们只提供基本知识（并在简化各种概念方面采取一些自由），并假设读者熟悉各种基本的反转概念（如寄存器等）。

有两本关于逆向工程（包括组装/拆卸）的优秀书籍：

软件（包括恶意软件）是用编程语言编写的 ...一种明确的“人性化”语言，可以被翻译（编译）成二进制代码。我们在第0x5章（“非二进制分析”）中讨论的脚本本身不是编译的，而是在运行时“解释”为系统理解的命令或代码。

如前所述，在分析被怀疑是恶意的已编译Mach-O二进制文件时，通常无法获得原始源代码。我们必须利用一个能够理解已编译的二进制机器级代码的工具，并将其翻译回更可读的东西：汇编代码！这个过程被称为分解。

汇编语言是一种低级编程语言，直接翻译成二进制指令。这种直接翻译意味着编译后的二进制代码可以（稍后）直接编译回汇编。例如，二进制序列100100000111000000111000可以在汇编代码中表示为：add rax, 0x38（“向rax寄存器添加38个十六进制”）。

反汇编程序的核心是将已编译的二进制文件（如恶意软件样本）作为输入，并将其转换回汇编代码。当然，这取决于我们如何理解提供的组件！

笔记:

汇编有各种“版本”。我们将重点介绍x86_64（x86指令集的64位版本），即采用英特尔语法的System V ABI

汇编指令“由一个助记符表示，它通常与一个或多个操作数相结合” [3]。助记符通常描述指令:

助记符	示例	说明
添加	添加rax, 0x100	将第二个操作数（例如0x100）添加到第一个操作数。
mov	mov rax, 0x100	将第二个操作数（例如0x100）移到第一个操作数中。
jmp	jmp 0x100000100	跳转到操作数中的地址（即继续执行）。
打电话	打电话给rax	执行操作数地址指定的子例程。

通常，操作数是寄存器（CPU上的命名内存“插槽”）或数值。在反转64位Mach-O二进制时会遇到一些寄存器

包括rax、rbx、rcx、rdx、rdi、rsi、rbp、rsp和r8. - r15. 正如我们将很快看到的，通常特定的寄存器被一致地用于特定的目的，这简化了逆向工程的工作。

笔记:

所有64位寄存器也可以由其32位（或更小）组件“引用”...在二进制分析中，你仍然会遇到。

“所有寄存器都可以在16位和32位模式下访问。在16位模式下，寄存器由上面列表中的两个字母缩写标识。在32位模式下，这个两个字母缩写的前缀是‘E’（扩展）。例如，‘EAX’是作为32位值的累加器寄存器。

类似地，在64位版本中，“E”被替换为“R”（寄存器），因此64位版本的“EAX”被称为“RAX”[3]

在结束对汇编代码的（粗略）讨论之前，让我们简要讨论一下调用约定。这将让我们了解如何进行API（方法）调用，如何传入参数，以及如何处理响应 ...在汇编代码中。

为什么这是相关的？通过研究Mach-O二进制文件调用的系统API方法，通常可以对其有一个相当全面的了解。例如，一个恶意二进制文件调用“write file” API方法，传入~/Library/LaunchAgents目录下的属性列表和路径，很可能会作为启动代理持久存在！

因此，我们通常不需要花费数小时来理解二进制文件中的所有汇编指令，而是可以专注于“围绕” API调用来理解的指令：

- 调用了什么（API）调用
- 向（API）调用传递了哪些参数
- 它根据（API）调用的结果采取什么操作

...通常，这种理解足以对我们正在分析的（潜在恶意的）二进制样本获得相对全面的理解。

为了便于解释调用约定和方法调用（在程序集级别），我们将重点介绍一个代码段Objective-C,它创建一个NSURL对象，该对象的初始化为“www.google.com":

```
01    //url对象
02    NSURL* url = [NSURL URLWithString:@"www.google.com"];
```

当程序想要调用方法或系统API调用时，它首先需要“准备”

通话的理由。在上面的源代码中，当调用URLWithString:方法（该方法需要一个string对象作为其唯一参数）时，Objective-C代码在字符串中传递“www.google.com".

在程序集级别，关于如何将参数传递给方法或API函数，有一些特定的“规则”。这被称为呼叫约定。调用约定的规则在应用程序二进制接口（ABI）中明确规定，64位macOS系统的规则如下：

论据	寄存器
第一个论点	rdi
第二个论点	rsi
第三个论点	rdx
第四个论点	rcx
第五个论点	r8
第六个论点	r9

macOS（英特尔64位）呼叫约定

由于这些规则始终适用，因此作为恶意软件分析师，我们可以准确地了解通话的方式。例如，对于采用单个参数的方法，在调用之前，该参数（参数）的值将始终存储在rdi寄存器中！

因此，一旦在反汇编中识别了调用（通过调用助记符），在汇编代码中向后看就会发现传递给方法或API的参数值。这通常可以提供对代码逻辑的有价值的洞察（即恶意软件样本试图连接的URL、打开的文件路径等）。

调用指令什么时候返回呢？咨询ABI发现，方法或API调用的返回值将始终存储在rax寄存器中。因此，一旦NSURL的URLWithString: method调用返回，新构造的NSURL对象将位于rax寄存器中。

由于在call指令完成时，rax寄存器保存返回值，因此您经常会看到使用call指令进行反汇编，紧接着是指令检查，并根据rax寄存器中的值的结果执行操作。例如（我们将很快看到），如果检查网络连接的函数在rax寄存器中返回零（NO/false），则恶意样本选择不感染系统。

当反转Objective-C二进制代码时，必须理解的另一件事是

objc_msgSend [4]函数。

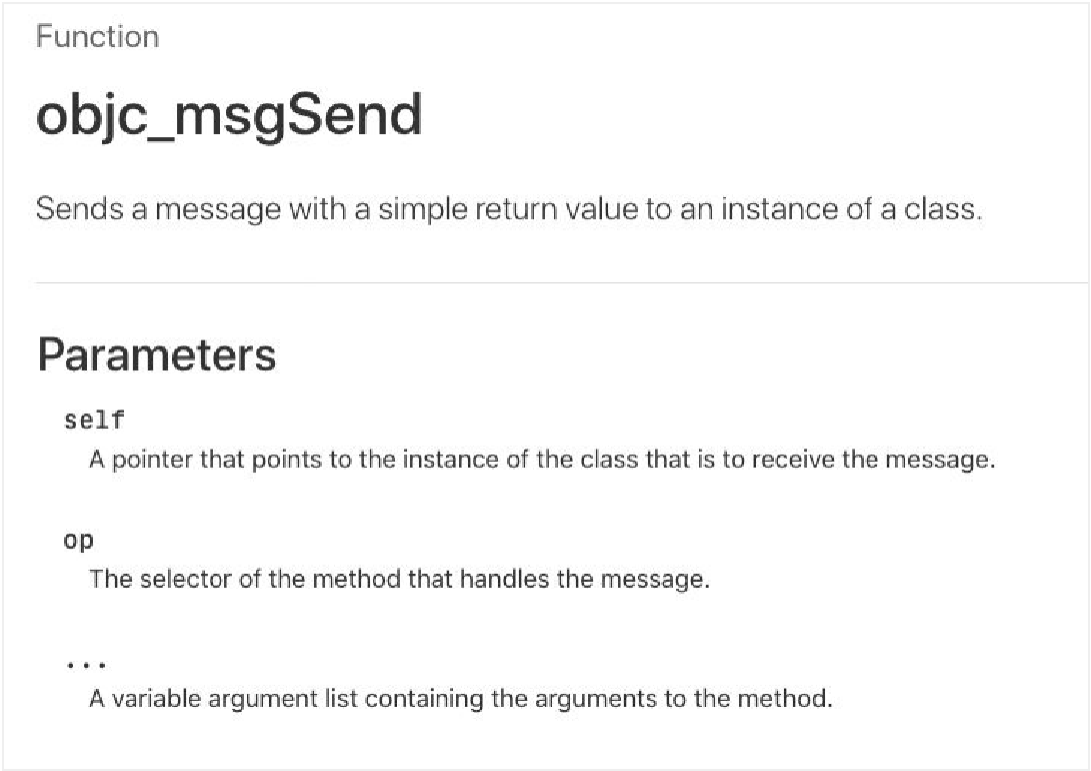
回想一下下面的Objective-C代码，它只是构造了一个URL对象：

```
01    //url对象
02    NSURL* url = [NSURL URLWithString:@"www.google.com"];
```

编译此代码时，编译器（llvm）会将此Objective-C调用（以及大多数其他Objective-C调用）转换为调用objc_msgSend的代码。或者正如苹果所解释的那样：

> “当遇到[Objective-C]方法调用时，编译器生成对*...objc_msgSend*” [4]

Apple developer文档包含此函数的一个条目，说明它“向类的实例发送带有简单返回值的消息” [4]：



Function

objc_msgSend

Sends a message with a simple return value to an instance of a class.

Parameters

self
A pointer that points to the instance of the class that is to receive the message.

op
The selector of the method that handles the message.

...
A variable argument list containing the arguments to the method.

objc_msgSend函数

由于绝大多数Objective-C调用都是通过此函数进行路由的，因此在反转编译的Objective-C代码时必须理解它。所以，让我们把它分解一下！

首先，什么是“发送信息” ...一个类的实例“甚至是什么意思？”？简单地说，这意味着调用对象的方法。

笔记:

Objective-C运行时基于发送消息的概念，以及其他相当独特的源于对象的范例。

有关Objective-C运行时及其内部的深入讨论，请参阅nemo提供的以下内容:

其次，objc_msgSend的参数呢:

- 第一个参数（self）是“指向要接收消息的类实例的指针” [4]。或者更简单地说，它是调用该方法的对象。如果方法是类方法，那么它将是类对象的一个实例（作为一个整体），而对于实例方法，self将指向类的一个实例化实例作为一个对象。
- 第二个参数（op）是“处理消息的方法的选择器” [4]。更简单地说，这只是方法的名称。
- 其余参数是方法（op）所需的任何值。

最后，objc_msgSend返回方法（op）返回的任何内容。

回想一下，ABI定义了如何将参数传递给函数调用。因此，我们可以准确地映射在调用时哪些寄存器将保存objc_msgSend的参数:

论据	寄存器	（适用于）objc_msgSend
第一个论点	rdi	self：调用方法的对象
第二个论点	rsi	op：方法的名称
第三个论点	rdx	方法的第一个参数
第四个论点	rcx	方法的第二个参数

第五个论点	r8	方法的第三个参数
第六个论点	r9	方法的第四个论点
7+参数	rsp+ (在堆栈上)	方法的5+参数

当然，寄存器rdx、rcx、r8、r9仅在被调用的方法需要它们（用于参数）时使用。例如，只接受一个参数的方法将只使用rdx寄存器。

此外，与任何其他函数或方法调用一样，一旦对objc_msgSend的调用完成，rax寄存器将保存返回值（实际上是调用的方法的返回值）。

这就结束了我们关于汇编语言基础知识的简短讨论。有了这个低级语言的基础知识，现在让我们看一下拆解二进制代码。

拆卸

反汇编包括将二进制代码（1和0）转换回汇编指令。然后可以分析这些汇编代码，以全面了解二进制代码。反汇编程序（将在稍后讨论）是一个能够执行这种转换并方便分析已编译二进制文件的程序。

在这里，我们将讨论各种反汇编概念，并通过真实世界的示例（直接取自恶意代码）加以说明。重要的是要记住，一般来说，分析恶意代码的目的是全面了解其逻辑和功能 ...不一定要理解每一个装配说明。正如我们前面提到的，关注方法和函数调用的逻辑通常可以提供一种有效的方法来获得这样的理解。

因此，让我们简单地看一个反汇编代码的示例，以说明如何识别此类调用、参数和（API）响应。最终结果如何？全面理解反汇编代码片段。

恶意软件有时包含检查其主机是否连接到互联网的逻辑。如果受感染的系统处于脱机状态，恶意软件通常会在尝试连接到其命令和控制服务器以执行任务之前等待（睡眠）。

检查网络连接的恶意软件的一个具体例子是OSX。Komplex [7],其中包含一个名为connectedToInternet的函数。通过研究这个民族国家后门的反汇编二进制代码，我们可以确认这个功能确实可以检查受感染的系统是否在线，并了解它是如何完成这个检查的。

具体来说，我们的分析将揭示恶意软件通过Apple的NSData类，调用dataWithContentsOfURL: method [8]来检查网络连接。如果无法访问远程URL（www.google.com）（即受感染的系统处于脱机状态），则呼叫将失败，表明系统处于脱机状态。

现在让我们深入了解OSX的分解。Komplex的ConnectedPointerNet功能（为清晰起见进行了注释）。请注意，我们将逐个分解函数，首先显示一个Objective-C表示，它是从反汇编中重建的。

```
connectedToInternet() {

   //url对象
   NSURL* url = [NSURL URLWithString:@"www.google.com"];
```

前面我们提到，Objective-C方法调用被“转换”为对objc_msgSend函数的调用。因此，在反汇编中看到对该函数的调用也就不足为奇了：

```
connectedToInternet

;将指向NSURL类的指针移动到rdi中
; rdi寄存器保存第一个参数（“self”）
mov     rdi, qword [objc_cls_ref_NSURL]

;将指向方法名“URLWithString:”的指针移动到rsi中
; rsi寄存器保存第二个参数（“op”）
利亚     rsi, qword [URLWithString:]

;在rdx中加载url的地址
; rdx寄存器保存第三个参数，这是传递给rdx的第一个参数
; 正在调用的方法（URLWithString:）
利亚     rdx, qword [_www_google_com]

;将指向objc_msgSend的指针移动到rax寄存器中
; 然后调用它
mov     rax, cs:_objc_msgSend_ptr
打电话   rax

;将响应保存到名为“url”的（堆栈）变量中
; rax寄存器保存方法调用的结果
mov         qword [rbp+url], rax
```

我们还看到，一行Objective-C代码（NSURL* url = [NSURL URLWithString:@“www.google.com”]）被翻译成了几行汇编代码。

首先初始化参数（在预期寄存器中）以调用objc_msgSend,然后进行调用，并保存结果。

具体来说，rdi寄存器（第一个参数）加载了对NSURL类的引用。然后，用方法的名称加载第二个参数（rsi）：URLWithString:。最后用字符串“www.google.com”初始化rdx。现在可以制作objc_msgSend了。调用完成后，新初始化的NSURL对象将返回到rax寄存器中，并存储到局部变量中。

一旦构建了NSURL对象，恶意软件就会调用NSData的dataWithContentsOfURL:方法。同样，在查看反汇编之前，让我们在Objective-C中构造一个可能的表示：

```
01    //数据对象
02    //通过尝试连接/读取谷歌进行初始化。通用域名格式
03    NSData* data = [NSData dataWithContentsOfURL:url];
```

以下是OSX的（相关）反汇编代码。Komplex的连接点网络法：

```
01  ;以下代码准备了相关的寄存器
02  ; 然后通过objc_msgSend函数进行objective-c调用
03
04  ;将指向NSData类的指针移动到rdi中
05  ; rdi寄存器保存第一个参数（“self”）
06  mov     rdi, qword [objc_cls_ref_NSData]
07
08  ;将指向方法名“dataWithContentsOfURL:”的指针移动到rsi中
09  ; rsi寄存器保存第二个参数（“op”）
10  利亚    rsi, qword [dataWithContentsOfURL:]
11
12  ;将（先前创建的）url对象移动到rdx中
13  ; rdx寄存器保存第三个参数，这是传递给rdx的第一个参数
14  ; 正在调用的方法（'dataWithContentsOfURL:'）
15  mov     rdx, qword [rbp+url]
16
17  ;将指向objc_msgSend的指针移动到rax寄存器中
18  ; 然后调用这个函数，进行objective-c调用
19  mov     rax, cs:_objc_msgSend_ptr
```