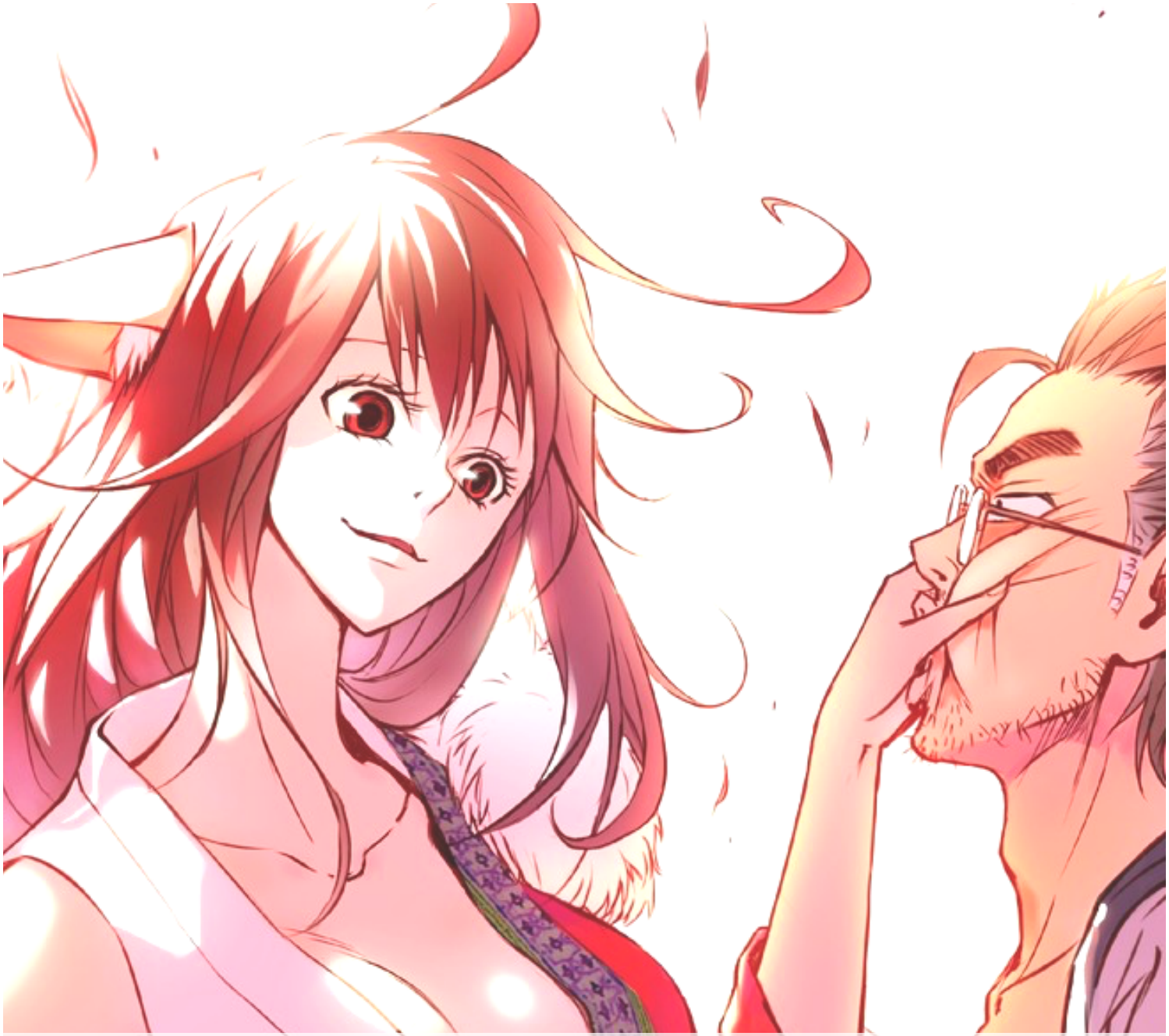

Object-C 测试工具与 Frida

译者：lockdown(看雪 ID：小调调)

原文链接：https://rotlogix.com/2016/03/20/objective-c-instrumentation-with-frida/

原文作者：Rotlogix

微信公众号：看雪 iOS 安全小组 我们的微博：weibo.com/pediyiosteam

我们的知乎：zhihu.com/people/pediyiosteam

Object-C 测试工具与 Frida

Overview

一个朋友最近在问怎么使用 Frida 来测试 Objective-C 的应用程序，我将这个过程分享给大家，这个简要的教程阐述了我是如何教他一步一步做到的。

Frida 是一个非常强大的跨平台测试工具。对于那些不熟悉 Frida 是什么、能做什么的人，我强烈建议在开始这个教程之前，先去下面的网页查看一下。

http://www.frida.re/

在本教程中，我们将使用 Frida 在一个非常简单的 Objective-C 应用中，拦截一个方法（函数）。我们将使用 Frida 的 Python 通用接口来实现。

Objective-C Application

我们的 Objective-C 程序中有一个类叫 Hook 和一个实例化方法叫 hookMe。该实例化方法有一个参数，并通过 NSLog 打印输出。

```
-(void)hookMe:(NSString *)arg {
    NSLog(@"%@", arg);
}
```

在应用程序的 main 函数中，我们创建一个新的 Hook 实例，然后调用 hookMe。

```
Hook *hook = [[Hook alloc]init];
[hook hookMe:@"Hello, World!"];
```

我们的目标是编写一个脚本，它将拦截 hookMe 并打印出其参数内容以及一些附加信息。

Hook.py

想要阅读完整的脚本，你可以在这里查看这里。

在本教程中，我们仅关注注入到我们的目标进程中的 JavaScript 代码。我更倾向于对所有的 Frida Python 脚本使用相同的框架代码，所以你可以根据需要复制粘贴，并根据自己的需求添加自己所需要的功能。

```
for(var className in ObjC.classes) {
    if (ObjC.classes.hasOwnProperty(className)) {
        if(className == "Hook") {
            send("Found our target class : " + className);
        }
    }
}
```

如果你查看 Frida 的文档中对 Objective-C 的支持，你会看到 ObjC.classes 返回当前注册类的映射表。 这些类可以很容易地通过 for 循环遍历，并且应该允许我们查找我们的目标类 Hook。

它不包括在脚本中，但是可以使用以下的方式执行访问方法。

```
ObjC.classes.Hook.$methods;
```

你可以使用特殊属性 $methods 访问对象公开的方法数组。这将包括 hookMe 方法以及通过继承可用的任何其他方法。

事情从这里开始变的有趣了。

```
if(ObjC.available) {
    for(var className in ObjC.classes) {
        if (ObjC.classes.hasOwnProperty(className)) {
            if(className == "Hook") {
                send("Found our target class : " + className);
```

```
            }
        }
    }
    var hook = ObjC.classes.Hook["- hookMe:"];
    Interceptor.attach(hook.implementation, {
        onEnter: function(args) {
            var receiver = new ObjC.Object(args[0]);
            send("Target class : " + receiver);
            send("Target superclass : " + receiver.$superClass);
            var sel = ObjC.selectorAsString(args[1]);
            send("Hooked the target method : " + sel);
            var obj = ObjC.Object(args[2]);
            send("Argument : " + obj.toString());
        }
    });
} else {
    console.log("Objective-C Runtime is not available!");
}
```

首先，我们创建一个包含我们要拦截的方法的新变量。

```
var hook = ObjC.classes.Hook["- hookMe:"];
```

在 Objective-C 中 “-“ 是描述实例方法的语法。如果你遍历$methods 特殊属性返回的所有方法，你将看到 hookMe 以这种方式展现出来。

Objective-C 运行时将所有方法调用转换为 objc_msgSend 调用。objc_msgSend 函数声明如下：

```
id objc_msgSend(id self, SEL op, args);
```

当我们拦截的方法被 Objective-C 运行时调用时，Frida 通过拦截方法的参数提供对底层 objc_msgSend 函数的访问接口。

```
var receiver = new ObjC.Object(args[0]);
send("Target class : " + receiver);
send("Target superclass : " + receiver.$superClass);

var sel = ObjC.selectorAsString(args[1]);
send("Hooked the target method : " + sel);

var obj = ObjC.Object(args[2]);
send("Argument : " + obj.toString());
```

第一个参数是一个指针，指向接收消息的实例类。在我们的例子中它指向 Hook 。 您还可以通过 Frida 还提供的$ superClass 特殊属性访问对象的超类。

objc_msgSend 声明中的第二个参数是 SEL，它本质上是一个包含调用方法名的 CString(selector)。你可以使用 Frida 的 selectorAsString JavaScript 函数将其转换并将其发送回您的 Python 代码。

我们可以访问传递给 args [2]中的 hookMe 方法的参数，并从中创建一个新的 ObjC.Object。 Frida 对对象的处理允许我们简单地调用 toString 以获得参数的值。下面是我们的 hook.py 脚本的最终输出：

```
[*] Found our target class : Hook
[*] Target class : <Hook: 0x7fd7cb4005e0>
[*] Target superclass : NSObject
[*] Hooked the target method : hookMe:
[*] Argument : Hello, World!
```

Conclusion

我希望这些信息对你有帮助。再次，你可以仔细阅读整个脚本。我强烈建议深入了解 Frida 的文档和 frida-gum，以更熟悉 Frida。

References

https://github.com/frida/frida-gum/blob/master/bindings/gumjs/gumjs-objc.js