# 从开发到漏洞挖掘的角色转换

王纬

2020-06-05

# 关于我

- ID: Proteas
  - Weibo: @Proteas
  - Twitter: @ProteasWang
- 3 年 Windows C++ 开发经验。
- 5 年 iOS App 开发经验。
- 5 年 iOS & macOS 漏洞挖掘经验。
- 目前处于技术理想主义与技术现实主义之间。

# 主要内容

- 背景与约束
- 漏洞挖掘与利用
- 为什么要转漏洞挖掘
- 如何转到漏洞挖掘
- 结语

# 背景与约束

- 每个人眼里都有一个世界。
- 每个人都有自己的局限性。

- 经验是不可传承的，
- 每个人都有属于自己的路，
- 但仍希望本文能对你有所帮助。

# 背景与约束

- 二进制方面的漏洞挖掘与利用。
- 领域与目标：默认指 macOS/iOS。
- 面向：开发人员。

# 漏洞挖掘与利用 – 挖掘

- 漏洞挖掘研究的是什么？
- 研究的是可以用于突破安全边界的编程错误。
  - 识别编程错误。
  - 触发编程错误的方法。
  - 自动、高效发现编程错误的方法。

- *本文中的编程错误指：可以用于突破安全边界的编程错误。*

# 漏洞挖掘与利用 – 挖掘

- 大部分编程错误跟“信任”有关系，
- 从广义的角度说，
- 漏洞挖掘研究的是：信任与欺骗。

# 漏洞挖掘与利用 – 利用

- 漏洞利用研究的是：控制与转化。
- 下面是我在日报里写的一句话，
- 编写 iOS 内核利用的过程实际是：
  - 以初始错误类型为起点，
  - 以内核空间任意地址读写为终点，
  - 不断提升控制内核数据能力的过程。

# 漏洞挖掘与利用 – 利用

- 挖掘与利用所依赖的知识有共性，
- 但是差别很大，
  - 待处理的问题
  - 处理问题时的思维模式
- 个人认为这是两个不同的方向。
- 相对于挖掘，在没有公开套路的情况下，
- 利用更具创造性、艺术性。

# 漏洞挖掘与利用 – 思维模式

- 下图是 Minizip 库的解压缩接口的截图。

```
extern int ZEXPORT unzStringFileNameCompare OF ((const char* fileName1,
                                                  const char* fileName2,
                                                  int iCaseSensitivity));

extern unzFile ZEXPORT unzOpen OF((const char *path));

extern unzFile ZEXPORT unzOpen2 OF((const char *path,
                                    zlib_filefunc_def* pzlib_filefunc_def));

extern int ZEXPORT unzClose OF((unzFile file));

extern int ZEXPORT unzGetGlobalInfo OF((unzFile file,
                                        unz_global_info *pglobal_info));
```

# 漏洞挖掘与利用 – 思维模式

```
15
16  extern int ZEXPORT unzGetGlobalComment OF((unzFile file,
17                                             char *szComment,
18                                             uLong uSizeBuf));
19
20  extern int ZEXPORT unzGoToFirstFile OF((unzFile file));
21
22  extern int ZEXPORT unzGoToNextFile OF((unzFile file));
23
24  extern int ZEXPORT unzLocateFile OF((unzFile file,
25                       const char *szFileName,
26                       int iCaseSensitivity));
```

# 漏洞挖掘与利用 – 思维模式

- 开发人员关注什么？
- 大部分关注的是：接口如何使用。
- 一部分会关注：
  - 库的设计
  - 库内部使用的算法。

- 这跟我们所接受的专业训练有关系。

# 漏洞挖掘与利用 – 思维模式

- 漏洞挖掘人员关注什么？
- 攻击面，即：哪些数据是攻击者可控的。
- 库内部的实现：路径穿越，符号问题，溢出，内存操作等。
- 也会关注接口：
  - 主要看接口是否清晰、易用，
  - 是否容易让开发者犯错。

# 漏洞挖掘与利用 – 思维模式

- 开发：直接性。
  - 看到的是接口，关注的也是接口。

- 漏洞挖掘与利用：间接性。
  - 看到的是接口，
  - 关注的是数据对内部实现的影响。
  - 例子：内核利用。

# 漏洞挖掘与利用 – 对抗

- 开发者与厂商的利益是一致的，
- 大家共处一个生态。

- 本质上，漏洞挖掘与厂商处于对抗状态。
  - 如果没有漏洞挖掘人员，
  - 厂商可以降低数量可观的成本。

- 漏洞挖掘的同行之间：撞洞。

# 漏洞挖掘与利用 – 不确定性

- 开发人员及软件工程：
  - 为了保证确定性，
  - 追求的也是确定性。

- 漏洞挖掘与利用领域涉及不确定性，
- 尤其是当前的漏洞利用领域。
- 这会让我们感觉很不舒服、不可靠。

# 漏洞挖掘与利用 – 支撑技术

- 开发领域的一些调侃：
  - 面向 Google 编程。
  - 面向 Github 编程。
  - 面向 QQ 群编程。
- 这些在漏洞挖掘领域基本不存在，
- 遇到的大部分问题都需要自己解决。

# 漏洞挖掘与利用 – 工程化程度

- 工程化是为了保证结果，
- 粗暴地讲就是：如果这么做，就可以得到结果。
- 先定义“结果”再来看工程化程度。

- 如果结果是：打下目标，那工程化程度非常低。
- 如果结果是：获得 CVE，那工程化程度中等偏下。

# 漏洞挖掘与利用 – 工程化程度

- 为什么要谈工程化程度？因为产出。
- 在开发人员的眼中应该不存在 0 产出的情况，
- 但在漏洞挖掘领域，结合具体的目标，
- 是可能存在 0 产出情况的。

- 为了避免这种情况，
- 需要持续不断的思考、总结、学习、对抗。

# 漏洞挖掘与利用 – 习惯失败

- 在大部分情况下，
- 开发人员在解决好不好的问题，
- 漏洞挖掘与利用人员在解决能不能的问题。

# 漏洞挖掘与利用 – 习惯失败

- 在实际的项目开发中，
- 每天都可以输出代码，
- 大部分需求都是可以实现的。

- 在针对有实际价值的目标的漏洞挖掘过程中，
- 基本不可能每天都发现漏洞，
- 所发现的漏洞大部分都不可利用。
- 所设计的利用方案，
- 很可能在某个环节上，被某个缓解措施阻止。

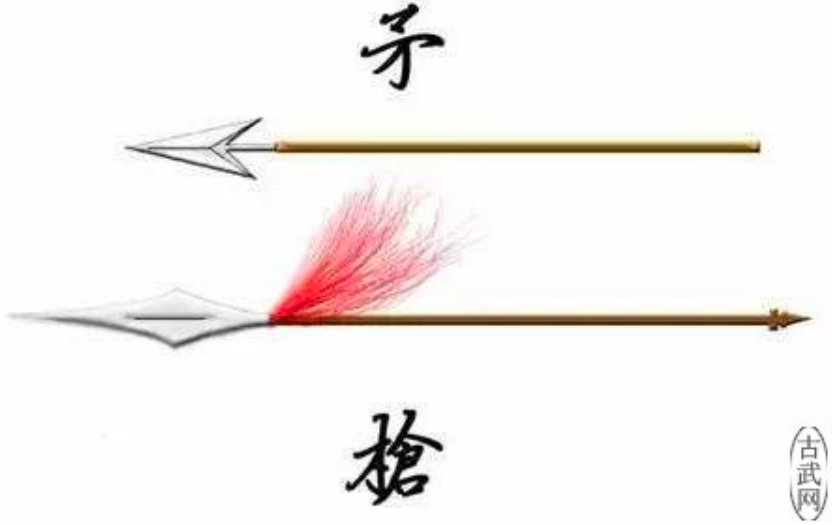
# 漏洞挖掘与利用 – 习惯失败

- 这种情况会对开发人员造成心理冲击，
- 如果内心不强大，很可能会放弃。

- 我们要习惯失败，但仍对目标保有渴望。
- 我们要在多次失败的情况下，
- 继续保持研究的节奏。

# 漏洞挖掘与利用 – 方法论

- 尖 → 专注




图片来源： http://m.6okok.com/wuqi/lenbingqi/6222.html

# 漏洞挖掘与利用 – 方法论

- 实战 → 实操


图片来源： https://zhuanlan.zhihu.com/p/34766047

# 漏洞挖掘与利用 – 价值观

- 开发领域，大家的价值观比较统一。
- 漏洞挖掘领域存在各种价值观：
  - 打下目标。
  - CVE 致谢的数量。
  - 学术价值。
  - ……
- 没有对错、好坏，都在推动这个行业向前发展。

# 为什么要转漏洞挖掘

- 兴趣所在。
- 挑战与快乐。
- 结果更好衡量：
  - 数量
  - 质量

- 可能，除了父母，
- 没有人真的会为我们的未来买单。

# 如何转到漏洞挖掘 – 学习方法

- 对于初学者，
- 个人不提倡“书单式”的学习。

- 个人鼓励“功利式”的学习。
- 学习知识是为了解决问题，
- 知识是为目的服务的。

# 如何转到漏洞挖掘 – Review

- 研究的是：编程错误。
- 最核心的方法论是：专注 + 实操。

# 如何转到漏洞挖掘 – 强化目的

- 目的是：寻找编程错误。
- 其它的都是为这个目的服务：
  - 代码阅读能力。
  - 静态分析能力。
  - 动态分析能力。
  - ……
- 不要本末倒置。

# 如何转到漏洞挖掘 – 强化目的

- 个人认为：
  - 对于职业的漏洞挖掘人员，
  - 针对编程错误的思考与训练，
  - 应该每天都进行，
  - 目的是把其变成一种“肌肉记忆”。

- 大家要记住目的，不要迷失在知识的海洋里。

# 如何转到漏洞挖掘 – 确定目标

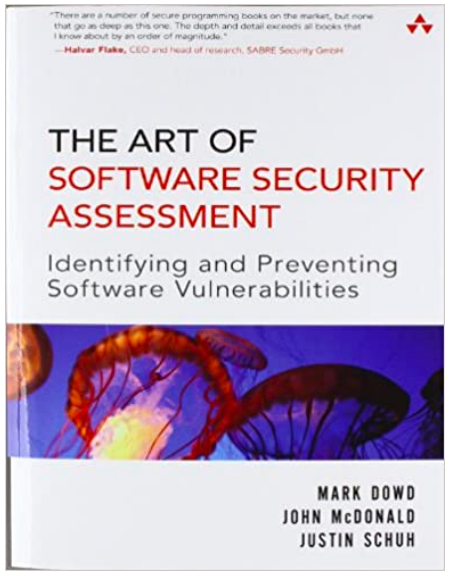
- 由于对抗属性，首先需要确定目标。
- 目标不同，技战术也不同。
- 大家可以结合自己的背景与兴趣选择目标。
- 目标一旦选定，建议一年内不要改变。

# 如何转到漏洞挖掘 – 确定目标

- iOS 沙盒逃逸。
- Android Binder 相关的漏洞。
- macOS XPC 漏洞。
- Linux 用户空间提权。
- 虚拟化产品的逃逸。
- ……

# 如何转到漏洞挖掘 – 编程错误

- 首先要掌握通用的错误类型。
- 《The Art of Software Security Assessment》， by Mark Dowd
- 重点关注其中的错误类型。
- 也可以阅读其它书籍与资料，
- 重点仍然是关注错误类型。
- GP0： https://bugs.chromium.org/p/project-zero/issues/list?q=&can=1



# 如何转到漏洞挖掘 – 编程错误

- 每天积累、关注错误类型。
- 关注每天安全动态中提及的漏洞的错误类型：
  - https://weibo.com/360adlab?is_all=1
  - https://weibo.com/xuanwulab?is_all=1
- 关注 Twitter 上漏洞相关信息中的错误类型。
- 总之，丰富自己所掌握的错误类型，
- 强化自己对编程错误的识别能力。

# 如何转到漏洞挖掘 – 编程错误

- 根据你选择的目标，
- 寻找其历史漏洞。
- 可以选择一个有公开利用的历史漏洞，
- 重点调试分析。

# 如何转到漏洞挖掘 – 编程错误

- #调试分析历史漏洞#
- 可以让我们了解：
  - 所涉及的领域知识，
  - 对目标漏洞有更深的理解，
  - 对利用有一定理解。
- 分析的第一个漏洞，需要的时间比较长。
- 也可以通过这个过程看看自己是否真的喜欢漏洞挖掘。

# 如何转到漏洞挖掘 – 编程错误

- #调试分析历史漏洞#
- 如果一个模块出过漏洞，
- 那它里面存在其它漏洞的可能性更大。
- 另外，可以进行“变种分析”。
- 例子：macOS XPC 提权漏洞。

- *《盘古越狱工具在用户空间的行为》*
- *《iOS 8.1.2 越狱过程详解及相关漏洞分析》*

# 如何转到漏洞挖掘 – 编程错误

- 感觉自己掌握了大部分漏洞类型后，
- 可以通过发现 N-Day 去检验下自己的能力。
- 方法：
  - 个人。
  - 团队。

# 如何转到漏洞挖掘 – 编程错误

- 在学习错误类型以及调试历史漏洞时，
- 可能会涉及逆向及调试。

- 对于漏洞挖掘，
- 逆向与调试只是技术手段，
- 不是技术目的。

# 如何转到漏洞挖掘 – 静态分析

- 强化逆向能力的一个方法：
  - 选择一个缺少文档，
  - 规模合适的功能，
  - 对其进行逆向分析。
- 建议：初学者不要选择 C++ 程序。

# 如何转到漏洞挖掘 – 静态分析

- 在逆向分析的过程中，
- 边看汇编，边查手册，
- 对每一行汇编进行注释。

- *《逆向 iOS SDK – +[UIImage imageNamed:] 的实现》*
- *《逆向 iOS SDK – “添加本地通知”的流程分析》*

# 如何转到漏洞挖掘 – 静态分析

- 建议：
  - 汇编是非常 Low Level 的，
  - 汇编间接反应的是开发者的目的，
  - 不要被汇编淹没，
  - 只见树木不见森林，
  - 要尝试去了解开发者的意图。

# 如何转到漏洞挖掘 – 动态分析

- 程序的有些状态无法静态获得，
- 因此需要调试分析，
- 调试是对逆向的补充、解惑。

# 如何转到漏洞挖掘 – 动态分析

- 调试器有大量的命令与参数，
- 不要去死记这些命令与参数。

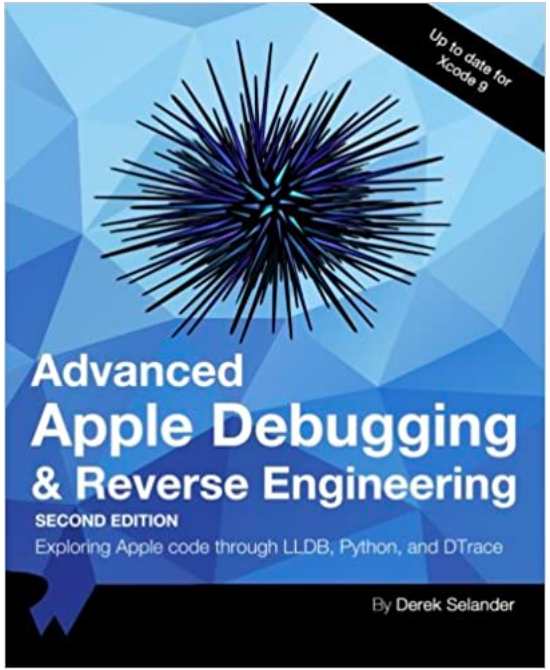
# 如何转到漏洞挖掘 – 动态分析

- 大家可以先略读一本调试相关的书籍，
- 掌握使用调试器解决问题的方法与思路。

- 调试器是工具，
- 调试是技术手段，
- 不是技术目的。

# 如何转到漏洞挖掘 – 动态分析

- 《Advanced Apple Debugging & Reverse Engineering》



# 如何转到漏洞挖掘 – 动态分析

- 调试之前应该想一个调试“方案”：
  - 想通过调试了解、确定什么？
  - 在哪些地址设置断点？
  - 断点命中后，关注哪些寄存器的值？
  - 关注哪些跳转、分支？

# 如何转到漏洞挖掘 – 动态分析

- 练习方法：
- 可以结合前面逆向分析的模块，
  - 使用调试器去获得在逆向时：
    - 无法获得的信息，
    - 或者不容易确定的信息。
  - 在调试器的帮助下，去验证静态分析的结果。

# 如何转到漏洞挖掘 – 方法

- 工程人员主要使用两种方法来挖掘漏洞：
  - 审计
  - 模糊测试（Fuzzing）

- 前面我们不断强调错误类型，
- 因为这是进行审计的基础。

# 如何转到漏洞挖掘 – 审计

- 以功能为主：一对多
  - 功能是“一”，
  - 错误类型是“多”。
  - 需要熟练掌握各种错误类型。

# 如何转到漏洞挖掘 – 审计

- 以错误类型为主：变种分析
  - 错误类型为一，
  - 功能是多。
- 使用这种方法，
- 审计过程相对比较轻松。

# 如何转到漏洞挖掘 – 模糊测试

模糊测试（fuzz testing, fuzzing）是一种软件测试技术。

其核心思想是将自动或半自动生成的随机数据输入到一个程序中，并监视程序异常，如崩溃，断言（assertion）失败，以发现可能的程序错误，比如内存泄漏。模糊测试常常用于检测软件或计算机系统的安全漏洞。

# 如何转到漏洞挖掘 – 模糊测试

- 模糊测试非常适合开发人员，
- 可以利用开发人员的现有经验，
- 发挥出开发人员的优势。

# 如何转到漏洞挖掘 – 模糊测试

- 目前效果比较好的模糊测试方法是：
  - Coverage Guided Fuzzing。
  - 各种 Sanitizer，如：AddressSanitizer。

# 如何转到漏洞挖掘 – 模糊测试

- 目前常用的模糊测试工具：
  - AFL，AFL++
  - libFuzzer
  - Honggfuzz
  - *Peach Fuzzer*
  - libFuzzer + libprotobuf-mutator
  - syzkaller

# 如何转到漏洞挖掘 – 模糊测试

- 体验：
  - 先跟随网上的 Step-by-Step，
  - 去学习如何使用工具（如：AFL 或者 libFuzzer），
  - 然后选一个目标（建议选老版本），
  - 应用 Fuzz 工具。

# 如何转到漏洞挖掘 – 模糊测试

- 应用：
  - 根据自己的经验、兴趣选择一个目标，
  - 目标的接口不一定适合做模糊测试，
  - 对已有接口做封装，
  - 使其适合模糊测试，
  - 最后进行模糊测试。
- 分析哪些程序使用了目标库，
- 向其 SRC 报告发现的问题。

# 结语

- 漏洞挖掘研究的是：可以用于突破安全边界的编程错误。
- 核心方法论：专注 + 实操。
- 分清楚技术手段与技术目的，要有技术方向感。
- 模糊测试比较适合开发人员。
- 当方向与方法对了后，剩下的交给时间。

# 参考


1. Thomas Dullien, Fundamentals of security exploits
2. Thomas Dullien,  Why I Love Offensive Work, Why I don't Love Offensive Work
3. http://www.winimage.com/zLibDll/unzip101e.zip
4. https://zh.wikipedia.org/wiki/%E6%A8%A1%E7%B3%8A%E6%B5%8B%E8%AF%95

# 谢谢！

*意见及反馈：proteas[DOT]wang[AT]gmail[DOT]com*