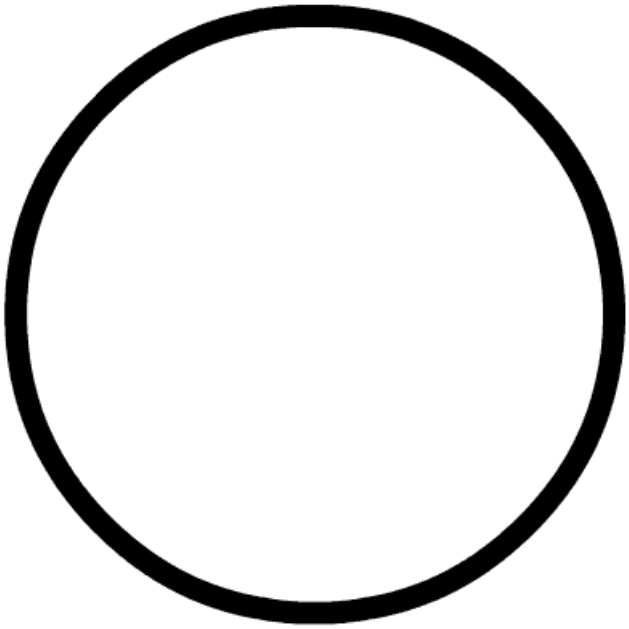
奋战3个月，我们挖到55个苹果漏洞，获得近30万美元奖金

转载

奇安信代码卫士 于 2020-10-10 18:17:37 发布 349 收藏
文章标签： java 安全 信息安全 session acl
原文链接：https://codesafe.qianxin.com/#/home



聚焦源代码安全，网罗国内外最新资讯！

编译：奇安信代码卫士团队

从2020年7月6日至10月6日整整3个月的时间，本文作者、Brett Buerhaus、BenSadeghipour、Samuel Erb 和 TannerBarnes 共同挑战苹果漏洞奖励计划。结果，他们从苹果基础设施中发现了55个漏洞，它们可导致攻击者完全攻陷客户和员工的应用程序、发动能够自动接管受害者iCloud 账户的蠕虫、检索苹果内部项目源代码、完全攻陷苹果使用的工业控制仓库软件，以及接管苹果员工的会话，从而访问管理工具和敏感资源。文章节选编译如下：

我们共发现了55个漏洞，其中11个为“严重”级别、29个为“高危”级别、13个为“中危”级别、2个为“低危”级别。这些严重性等级是我们基于 CVSS评分和对业务的影响力总结得出的。

截至2020年10月6日，苹果公司已修复大部分漏洞，实际上该公司基本在1到2个工作日内就完成修复（有些是在4到6小时内修复）。

引言

7月份我在闲逛推特时看到一名研究员分享了自己发现一个认证绕过漏洞而获得苹果公司10万美元奖金的经历。该漏洞可导致攻击者任意访问苹果任意客户的账户。当时我感到很惊讶，因为之前我理解苹果的漏洞奖励计划仅奖励影响其物理产品的漏洞而影响 Web资产的漏洞并不会得到奖金。

读完文章后，我快速搜索到了苹果公司的漏洞奖励计划页面，发现该公司愿意奖励“对用户具有重大影响”的漏洞，不管它们是否被明确列入漏洞奖励计划内。

这让我蠢蠢欲动，因为我可以调查涵盖范围广泛且功能有趣的新的奖励计划。当时我从未挑战过苹果公司的漏洞奖励计划，因此我不知道最终结果会怎么样，但我决定碰碰运气。

为了增加这个项目的趣味性，我向之前有过合作的黑客发信息询问是否愿意和我一起参加挑战。尽管并不确定最终是否会获得奖金或是否了解该漏洞奖励计划的运作方式，所有人都表示愿意一试，于是我们的苹果 hacking 之旅开始了。

侦察

Hacking 苹果的第一步是找到真正的目标。Ben 和 Tanner在这方面是行家，于是他们开始查找可以访问的所有苹果资产。他们把扫描结果列在控制面板上，包括苹果所有的多种域名下的 HTTP状态码、标头、响应主体和可访问 Web服务器的截屏等。

简言之，苹果的基础设施庞大无比。

苹果的基础设施拥有整个 17.0.0.0/8IP 段，即包括2.5万个 Web服务器，其中1万个在 apple.com下，另外7000个在唯一域名下，而占大头的是 TLD(.apple)。我们主要把精力花在17.0.0.0/8 IP 段、.apple.com 和 .icloud.com 上，因为有趣的功能似乎都集中在这里。

列出所有的 Web服务器后，我们开始在更多有意思的基础设施上运行了目录暴力破解。

通过自动扫描后发现的一些结果包括：

- 受思科 CVE-2020-3452 本地文件读取 1day 漏洞影响的多个 VPN服务器
- 崩溃页面上出错信息中被泄露的 Spotify访问令牌

通过这些进程获取的信息有助于我们了解苹果基础设施中的授权/认证运作方式、已有的客户/员工应用程序、使用的集成/开发工具以及多种可观测到的行为如消耗某些 cookie或重定向到某些应用程序的 Web服务器。

所有扫描完成后，我们已经大概了解了苹果基础设施的运作原理，于是开始针对个体的 Web服务器，因为从直觉来看它们更容易遭到攻击。

自此，我们发现了一系列漏洞，而且加深了对苹果漏洞奖励计划的了解。

发现的漏洞

报告 ID	日期	主题	严重程度	数量
741765282	7/26/2020	绕过授权和认证，实现远程代码执行	严重	1
747008210	9/17/2020	通过配置不当的权限绕过认证，获得全局管理访问权限	严重	1
743305978	8/11/2020	未清洁文件名参数导致命令注入	严重	1
744391544	8/21/2020	机密和管理员工具泄漏导致远程代码执行	严重	1
741014809	7/18/2020	内存泄漏导致员工和用户账户攻陷，进而导致多款内部应用程序遭访问	严重	1
742879257	8/6/2020	未清洁输入参数导致 Vertica SQL 注入	严重	1
742693737	8/5/2020	可蠕虫的存储型 XSS，导致攻击者完全攻陷受害者的 iCloud 账户	严重	1
743695576	8/14/2020	可蠕虫的存储型 XSS，导致攻击者完全攻陷受害者的 iCloud 账户	严重	1
742909161	8/7/2020	完整的响应 SSRF 导致攻击者读取内部源代码和访问受保护的资源	严重	1
745400606	9/1/2020	Blind XSS 导致攻击者访问内部的客户和员工问题追踪的支持门户	严重	1
744376677	8/21/2020	服务器端 PhantomJS 执行导致攻击者访问内部资源和检索 AWS IAM 密钥	严重	1
743197992	8/10/2020	IDOR 导致攻击者读取苹果合作伙伴应用程序的完整用户应用程序详情	高危	2
743661939	8/14/2020	iCloud 上的 IDOR 导致攻击者通过增量的数量识别符检索受害者名称和邮件地址	高危	2
744278275	8/20/2020	Blind XSS 导致攻击者访问 Apple Maps 管理应用程序并修改受保护的资源	高危	2
743337186	8/12/2020	Blind XSS 导致攻击者访问 Apple Books 管理应用程序并修改受保护的资源	高危	2
745697928	9/4/2020	Blind XSS 导致攻击者访问 Apple Books 管理应用程序并修改受保护的资源	高危	2
744190911	8/19/2020	多种双因素认证绕过导致攻击者在无需解决多因素认证机制难题的情况下访问账户详情	高危	2
744172712	8/19/2020	多种苹果认证系统中的反射型 XSS	高危	2
741856747	8/1/2020	通过启用的外部实体处理实现的 XXE	高危	2
745468499	9/2/2020	对苹果应用程序缺乏访问控制，导致攻击者能够检索所有用户的姓名、地址、电话号码和通讯录	高危	2
745467385	9/2/2020	苹果应用程序上的 IDOR 导致攻击者读取用户的受保护信息	高危	2
742383828	8/1/2020	苹果应用程序上的 IDOR 导致攻击者读取用户的受保护信息	高危	2
741425582	7/22/2020	苹果应用程序上的 SSRF 导致攻击者发送内部的 Gopher 请求	高危	2
742326610	7/31/2020	苹果应用程序上的 IDOR 导致攻击者读取用户的受保护信息	高危	2
739884965	7/5/2020	苹果 App Store 上的 IDOR 导致攻击者修改 Apple Store 应用程序的多个组件	高危	2
741782188	7/26/2020	多个 VPN 受本地文件泄漏漏洞的影响	高危	2
742775989	8/5/2020	Apple Application 中的 SSRF 导致攻击者访问受保护资源	高危	2
742778854	8/5/2020	Apple Application 中的 SSRF 导致攻击者访问受保护资源	高危	2
744525657	8/24/2020	Apple Application 中的 SSRF 导致攻击者访问受保护资源	高危	2
744327113	8/21/2020	Apple Application 中的 IDOR 导致攻击者枚举用户信息	高危	2
745360827	9/1/2020	Apple Application 中的 IDOR 导致攻击者枚举用户信息	高危	2
740621036	7/15/2020	Apple Application 中的 IDOR 导致攻击者枚举用户信息	高危	2
743044563	8/11/2020	Apple Application 内的 SSRF 导致攻击者访问受保护资源	高危	2
743675019	8/14/2020	Apple Application 中的 IDOR 导致攻击者枚举用户信息	高危	2
740944358	7/17/2020	Apple Application 内的 SSRF 导致攻击者枚举用户信息	高危	2
742879259	8/6/2020	Apple Application 上的访问控制不当问题导致攻击者泄漏并修改内部应用程序资源	高危	2
740610570	7/16/2020	Apple Application 上的存储型 XSS 导致攻击者提升权限并攻陷 Tenant 应用程序	高危	2

743047726	8/4/2020	对 Apple Application 缺乏评级限制，导致攻击者和验证并访问受保护资源	高危	2
744308191	8/20/2020	不受限的文件上传导致存储型 XSS	高危	2
745710818	9/4/2020	反射型 XSS 导致攻击者完全攻陷 Tenant 资源	中危	3
745356272	9/1/2020	路径遍历导致攻击者枚举系统文件信息	中危	3
739271635	7/7/2020	第三方应用程序上的反射型 XSS 导致攻击者攻陷应用程序	中危	3
740651991	7/15/2020	从低级别用户到高级别用户的 Blind XSS 导致攻击者攻陷应用程序	中危	3
739881737	7/10/2020	反射型 XSS 导致攻击者完全攻陷 Tenant 资源	中危	3
743224534	8/10/2020	不受限的文件上传导致存储型 XSS	中危	3
743675794	8/14/2020	未清洁参数导致反射型 XSS	中危	3
744068810	8/16/2020	未清洁参数导致反射型 XSS	中危	3
744392545	8/21/2020	IDOR 导致信息泄漏	中危	3
744634880	8/24/2020	Apple Application 上的存储型 XSS	中危	3
744991374	8/27/2020	未清洁参数导致反射型 XSS	中危	3
746153073	9/9/2020	未清洁参数导致反射型 XSS	中危	3
740352583	7/16/2020	Stack Trace 导致信息泄漏	中危	3
742387948	8/1/2020	无评级限制的表单登录	低危	4
744276140	8/20/2020	第三方网站上的信息泄漏漏洞	低危	4

Write-Up

我们无法详述所有的漏洞问题，不过挑选了12个更有意思的漏洞作为样本。它们是：

1、通过认证和授权绕过完全攻陷 AppleDistinguished Educators 项目

2、通过认证绕过完全攻陷 DELMIAApriso 应用程序

3、可蠕虫的存储型 XSS漏洞导致攻击者通过经修改的邮件窃取 iCloud数据

4、作者 ePublisher中的命令注入漏洞

5、iCloud 上的完整响应 SSRF问题，导致攻击者检索苹果源代码

6、通过 REST错误泄漏的 Nova管理员调试面板访问权限

7、通过 PhantomJSiTune Banner 和 Book Title XSS泄露的 AWS密钥

8、苹果 eSign上的堆转储导致攻击者攻陷多种外部员工管理工具

9、XML 外部实体处理导致 Java管理 API出现 BlindSSRF 漏洞

10、GBI Vertica SQL 注入和遭暴露的 GSFAPI

11、多个 IDOR 漏洞

12、多个 Blind XSS 漏洞

下文我们将节选前3个漏洞的 Write-up和读者分享。感兴趣的读者可点击文末原文链接，获取更多详情。

通过认证和授权绕过完全攻陷 Apple Distinguished Educators

我们首先hacking 的是“Apple Distinguished Educators”网站。这是一个邀请制 Jive论坛，用户可以通过苹果账户验证身份。这个论坛的有趣之处在于，该 app的某些核心 Jive功能通过苹果构建的一个自定义中间件页面进行移植，以便把自己的认证系统 (IDMSA)移植到底层的Jive 论坛，而在正常情况下需要使用用户名/密码进行认证。

这样做是为了让用户轻松使用已存在的苹果账户验证身份，而不必创建另外一个用户账户。仅使用“通过 Apple登录”就可登录论坛。

被禁止访问论坛的用户的着陆页是一个应用程序门户，你需要提供自己的信息，供论坛版主评估批准。当你提交论坛使用申请时，你提供了几乎关于账户的所有值，就像你正常登录 Jive论坛那样。这使得 Jive论坛能够通过 IDMSAcookie 来了解你，因为它将苹果账户的邮件地址和论坛绑定在一起。

申请页中隐藏的其中一个值是字段“password”的值“###INvALID#%!3”。当你提交的申请中包含用户名、姓氏、名字、邮件地址和雇主时，意味着也提交了“password”值，而这个值之后会秘密地绑定到源自该页面隐藏的输入字段的账户。

```
<div class="j-form-row">
<input id="password" type="hidden" value="###INvALID#%!3">
<div id="jive-pw-strength">
...
```

发现隐藏的默认 password 字段后，我们立即想到,要找到在应用程序上进行手动验证并访问论坛上已获批准账户的方法，而不是尝试通过“通过 Apple登录”系统进行登录。我们调查这件事是因为我们分开注册但所有人的密码是一样的。

如果有人通过该系统进行申请而且存在一个可供你手动验证的功能，那么就能使用默认密码登录账户，并完全绕过“通过 Apple登录”方法。

快速浏览后发现我们无法进行手动验证，但搜索谷歌后发现了一个“cs_login”端点，它的目的是通过用户名和密码登录Jive 应用程序。当我们手动构建供测试用的 HTTP请求，在 AppleDistinguished Developers 应用上进行验证时，我们发现它试图通过展示密码不正确的错误消息进行验证。当我们使用之前申请时用的账户时，该应用提示出错并且以我们尚未获得批准为由禁止认证。如果我们想要进行认证，则必须找到已获批准成员的用户名。

这时，我们将 HTTP请求加载到 BurpSuite 的入侵者并尝试通过登录和默认密码暴力破解长度为1到3个字符的用户名。

大概2分钟后，我们收到了302个响应，成功通过之前找到的默认密码登录用户名长度为3个字符的一名用户。闯关成功！我们的下一个目标是通过权限提升的某人身份进行认证。我们截取了一些访问的屏幕快照并点击“Users”列表，查看哪些用户是管理员。我们登录到在列表上看到的第一个账户，以证明能够通过管理功能实现远程代码执行后果，然而，前面还有一些路障需要清扫。

以管理员账户身份浏览“/admin/”（Jive 管理员控制台）时，应用会将我们重定向到登录页面。这就很奇怪，因为它是 Jive应用程序的自定义行为，而我们之前从未发现。我们猜测苹果公司已经基于 IP地址对管理控制台进行了限制，以确保该应用永远不会被完全攻陷。

我们首先做的事情之一时使用 X-Forwarded-For标头绕过假设的限制，但遗憾的是并未成功。接着我们尝试加载另一种形式的“/admin/”，以防该应用对访问管理控制台具有针对路径的黑名单。

发送更多的 HTTP请求后，我们发现“GET /admin;/”可使攻击者访问管理控制台。我们设置了在 HTTP请求中自动将“GET/POST /admin/”修改为“GET/POST /admin;/”的 BurpSuite 规则自动绕过。

当我们最终导航并加载了管理控制台时，立即发现了异常。我们并没有访问正常功能的权限，展示远程代码执行后果（并不存在模板、插件上传或标准的管理调试功能）。

于是，我们开始思考后续怎么办，并且意识到我们验证通过的账户可能并非该应用程序的“核心”管理员。在最终验证为核心管理员之前我们继续验证了两三个账户，并找到了可以实现远程代码执行的功能。

攻击者能够（1）通过使用一个隐藏的默认登录功能进行手动验证，从而绕过认证，接着（2）通过发送请求中被修改的 HTTP路径访问管理控制台，最终（3）使用“RCE 内置”功能之一如插件上传、模板或文件管理完全攻陷该应用程序。

总言之，它将使攻击者：

在ade.apple.com webserver上执行任意命令

访问用于管理用户账户的内部 LDAP服务

访问大部分的苹果内部网络

至此，我们完成报告并提交。

通过认证绕过，完全攻陷 DELMIA Apriso 应用程序

Hacking 苹果时，我们思考良久的问题是，是否存在和其产品制造以及分发相关的可访问服务。结果我们发现了“DELMIA Apriso”，它是一个第三方“Global Manufacturing Suite”，提供多种仓库解决方案。

遗憾的是，它的交互性不是很强，因为我们只能从可用接口“登录”和“重置密码”。

在数量有限的页面上尝试查找漏洞后，我们发现了奇怪的事：我们通过该网站右上方的栏以“Apple No PasswordUser”的用户身份得以验证。

于是，通过点击“重置密码”，我们以具有使用该页面“权限”的用户身份临时得到验证。

该应用程序的认证模式奏效了，尽管用户具有使用特定页面的特定权限。“重置密码”页面本身是一个页面，因此为了让我们使用该页面，该应用程序自动将我们登录到能够使用该页面的一个账户中。

我们尝试通过多种方法提升权限，但很长时间内并未取得进展。之后，我们向某个 OAuth端点发送了一个HTTP请求，尝试生成一个授权持有者，以便用于探索该 API。结果成功了！

尽管我们的用户账户权限仅限于授权和重置密码，但用户账户能够生成有权访问该应用程序 API版本的持有者。

现在我们可以探索 API 了，希望可以找到某种权限问题，以攻陷该应用程序的某些部分。幸运的是，在侦察过程中，我们发现了该应用程序的 API请求列表。

遗憾的是，我们无法访问大部分 API调用，不过某些部分如“操作 (Operations)”泄漏了大量可用功能。

如果点击 "/Apriso/HttpServices/api/platform/1/Operations"端点，它会返回由近5000个不同的 API调用组成的列表。除了我们最开始发送的初始授权持有者外，均未要求认证。这里泄漏的操作包括：

创建并修改货件

创建并修改员工薪酬日

创建并修改库存信息

验证员工徽章

数百个和仓库相关的操作

我们最为关注的是 "APL_CreateEmployee_SO"。

你可向特定的操作发送一个 GET请求并接收到使用如下格式的预期参数：

```
GET /Apriso/HttpServices/api/platform/1/Operations/operation HTTP/1.1
Host: colormasters.apple.com
```

HTTP 响应如下：

```
{
  "InputTypes": {
    "OrderNo": "Char",
    "OrderType": "Integer",
    "OprSequenceNo": "Char",
    "Comments": "Char",
    "strStatus": "Char",
    "UserName": "Char"
  },
  "OutputTypes": {},
  "OperationCode": "APL_Redacted",
  "OperationRevision": "APL.I.1.4"
}
```

虽然要花费一点时间，但不久后我们意识到要真正调用 API，必须先按照如下格式发送 JSON格式的一个 POST 请求：

```
{
  "inputs": {
    "param": "value"
  }
}
```

如上格式（事后）看似简单易懂，但在 hacking时，我们绝对不知道如何构建这个调用。我甚至还尝试向软件提供商发邮件询问如何构建这些 API调用，但由于我并非付费用户，我并未收到回复。

我用了近6个小时的时间试图搞清楚如何构建如上 API调用，但搞清楚后，一切问题迎刃而解。“创建员工(createemployee)”函数要求提供有赖于 UUID的多个参数，但我们通过其它“操作”检索到了参数并填入。

又过了2个小时，我们最终构建了如下 API请求：

```
POST /Apriso/HttpServices/api/platform/1/Operations/redacted HTTP/1.1
Host: colormasters.apple.com
Authorization: Bearer redacted
Connection: close
Content-Type: application/json
Content-Length: 380
{
  "inputs": {
    "Name": "Samuel Curry",
    "EmployeeNo": "redacted",
    "LoginName": "yourloginname123",
    "Password": "yourpassword123",
    "LanguageID": "redacted",
    "AddressID": "redacted",
    "ContactID": "redacted",
    "DefaultFacility": "redacted",
    "Department": "",
    "DefaultMenuItemID": "redacted",
    "RoleName": "redacted",
    "WorkCenter": ""
  }
}
```

发送这个 API 调用后，我们就可以作为一个全局管理员在应用程序上进行验证。如此，我们就能够完全监控仓库管理软件并很可能通过一些被接受的功能实现 RCE。

数百个不同的功能本可能造成大规模信息泄漏，而且能够破坏用于管理库存和仓库的关键应用程序。

可蠕虫存储型 XSS：通过被修改邮件窃取 iCloud 数据

苹果基础设施的核心组成部分之一是 iCloud平台。该网站用作照片、视频、文档以及苹果产品 app相关数据的自动存储机制。另外，该平台还提供多种服务如邮件服务和“找到我的 iPhone”等。

邮件服务是一种全邮件平台，它可以像 Gmail和 Yahoo那样供用户收发邮件。另外，苹果产品上还默认安装 iOS 和 Mac版本的邮件应用。该邮件服务和所有其它服务如文件和文档存储一起托管在www.icloud.com上。这意味着，从攻击者的角度来看，任意XSS 漏洞均可导致攻击者从 iCloud服务检索任何想要的信息。于是，我们开始查找任何 XSS漏洞。

该邮件应用的运作方式非常简单直接。当服务接收到邮件且用户打开时，数据就会处理为 JSON格式，由 JavaScript进行清理和选择，之后展示给用户。

这意味着，从内容清洁度来看，服务器端并未对邮件进行处理，而且所有用于渲染和处理邮件主体的实际功能都位于 JavaScript中，在客户端进行处理。这样做并不一定就不好，通过了解我们需要在源代码中破解的具体内容，我们简化了识别 XSS的过程。

Style 标记混淆导致的存储型 XSS

在测试该功能时，我最终的拦路虎是“<style>”标记。这个标记很有趣，因为 DOM只会取消末尾带有“<style>”标记的元素。也就是说，如果我们写入“<style><script>alert(1)</script></style>”，并在 DOM中完全渲染，则不会出现任何警报提示，因为从严格意义上来讲，该标记是 CSS样式，而“script”标记在标记内且不超过关闭标记的范围内填充。

从清洁角度来看，苹果需要担心的唯一问题是末尾的 style结束标记，或者，该页面上是否存在敏感信息，通过导入链条实现的 CSS注入。

我决定尝试在苹果不知道的情况下攻破该 style 标记，因为如果可实现的话，则它会是一个十分简单直接的存储型 XSS。

我尝试了各种排列，并最终发现了一些有意思的事：当你的邮件中含有两个 style标记时，它们的内容可能会被拼接成一个 style标记。这意味着如果我们能让“</sty”出现在第一个标记中而“le>”出现在第二个标记中，将有可能诱骗该应用程序认为我们的标记仍然是开放的但实际上并非如此。

我发送了如下 payload，测试它是否起作用：

```
<style></sty</style>
<style>le><script>alert(1)</script></style>
```

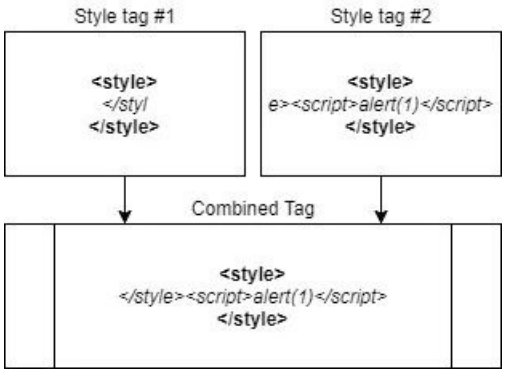
我收到了一封邮件，打开后发现是一个警报提示信息！起作用了!

该页面的 DOM 包含：

```
<style></style><script>alert(1)</script></style>
```

由于该邮件应用托管在www.icloud.com上，意味着我们具有浏览器权限以检索 iCloud服务相应 API的 HTTP响应（如果我们无法溜进 JavaScript）。

如上 payload 的解释如下：



此时，我们认为最酷的 PoC将是窃取所有受害者个人信息（照片、日历信息和文档）并将同样的 exploit发给所有通讯录联系人的东西。

我们构建了一个很好的 PoC，从 iCloudAPI 返回照片 URL 并粘贴到镜像标记中，之后在这些标记下面添加用户账户的通讯录列表。这说明我们可以检索到这些值，但为了提取这些值，我们必须绕过同源策略，也就是说除了“.apple.com”和一些其他域名外，没有其它任何轻易的出站 HTTP请求。

幸运的是，该服务是一个邮件客户端。我们仅使用 JavaScript就能够为自己调用邮件，添加 iCloud照片 URL和通讯录，之后删除受害者签名的所有 iCloud照片和文档 URL。

在恶意方执行的完整利用场景中，攻击者能够静默窃取所有的受害者照片、视频和文档，之后将修改后的邮件转发到受害者的联系人列表并在 iCloud邮件服务中蠕虫 XSSpayload。

由超链接混淆造成的存储型 XSS

之后，我发现了以类似方式影响邮件的另外一个 XSS漏洞。

对于这类半 HTML 应用程序，我一直都会查看它们是如何处理超链接的。虽然将未标记的 URL转换为一个超链接看似简单，但如未能正确清洁或者和其它功能结合，则会变得很麻烦。由于它依赖于正则表达式、InnerHTML 以及可以为 URL添加的所有获批准元素，因此人们常常从这里寻找 XSS漏洞。

该 XSS 的第二个有意思的功能是完全删除某些标记如“<script>”和“<iframe>”。这个功能简单利落，因为某些东西会依赖于多种字符如空格、制表符和换行符，而被删标记遗留的空白可以在无需告知 JavaScript解析器的情况下提供这些字符。这些差异使得攻击者能够混淆该应用程序并带入可调用 XSS的恶意字符。

我对这两个功能都进行了尝试（自动超链接和完全删除某些标记），之后决定结合使用这两个功能，查看产生的结果。令人惊讶的是，如下字符串攻破了超链接功能并混淆了 DOM：

```
https://www.domain.com/abc#<script></script>https://domain.com/abc
```

它通过邮件自我发送后，内容解析如下：

```
<a href="https://www.domain.com/abc#<a href=" https:="" www.domain.com="" abc="&amp;quot;" rel="noopener norefe
```

第一眼看上去这非常有趣，但利用它就难得多了。在标记内定义属性容易（如 src、onmouseover、onclick 等），但提供值就有难度，因为我们仍然必须匹配 URL正则表达式，以便它不会逃逸自动超链接功能。在无需发送单引号、双引号、括号、空格或反撇号的情况下，最终起作用的 payload 是：

```
https://www.icloud.com/mail/#<script></script>https://www.icloud.com/onmouseover=location=/javascript:alert
```

该 payload 在 DOM中生成如下内容：

```
<a href="https://www.icloud.com/mail#<a href=" https:="" www.icloud.com="" onmouseover="location=/javascrip
```

而且提供了警报提示信息。

该 payload 源自 @Blaklis 提供的一个 CTF解决方案。最开始我以为它可能是一个无法遭利用的 XSS，但总会出现边界情况 XSS的解决方案。

我认为的最佳解释是，（1）当加载初始 URL时，“<script></script>”中的字符串在自动化超链接进程中是可接受的而且不会打破进程，（2）删除 script标记后会产生一个空格或某些空白，从而在无需关闭初始超链接功能的情况下重置自动化超链接功能，（3）第二个超链接添加额外的引号，用于打破超链接并创建鼠标悬停事件句柄。

第二个 XSS的影响和第一 XSS一样，除了第二个要求用户触发鼠标悬停事件句柄，但这部分可以简化为通过制作整个邮件的超链接，更轻松触发鼠标悬停事件。

总之，攻击者本可滥用该 XSS漏洞：

- 创建一个能够静默提取/修改包括照片和视频在内的 iCloud账户信息
- 在受害者浏览器中静默执行任意 HTML和 JavaScript

结论

当我们开始着手这个项目时，不曾想到要三个多月才能完成。本来是想偶尔为之的业余项目，但疫情让我们在项目上投入了数百个小时的时间。

总体而言，苹果公司对漏洞报告的响应度很高。更严重漏洞的报告从提交到修复仅有4个小时。

由于没有人真正了解自己的漏洞奖励计划，我们实际上是进入了无主领地并投入如此之多的时间。苹果曾经有着和安全研究员合作不愉快的经历，从我们的情况来看，苹果有了很大的进步。

本文的写作经过也很有趣，因为我们不太确定如何成文。坦率讲，我们找到的每个bug 本来都可以成为完整的writeup。苹果所使用的认证系统十分复杂，不是三言两语就能说清的。苹果的基础设施如 iCloud、App 商店和Developer平台等都是如此。

截止到10月8日，我们已经收到32笔奖金，总额为28.85万美元。

不过，苹果公司似乎是分批支付奖金，可能会在未来几个月的时间里支付其它漏洞奖金。

本文所披露的所有漏洞均已修复并已再次经过测试。

推荐阅读

苹果修复严重的代码执行漏洞，影响 iOS 和 iPadOS 操作系统

不到4个小时，我找到了一枚苹果 0day

呐，一个苹果洞赚10万美元的详细经验都在这里了~

苹果会修复这个 0day吗？有人借机发布越狱包且适用于当前iOS 版本

原文链接

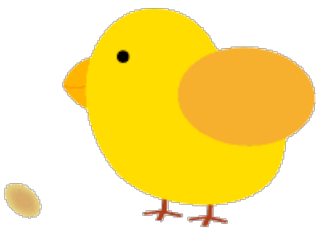
https://samcurry.net/hacking-apple/

题图：Pixabay License

奇安信代码卫士 (codesafe)

国内首个专注于软件开发安全的

产品线。



觉得不错，就点个“在看”吧~