锤子科技 Smartisan OS 云服务

内部研讨报告

1 概述

1.1 锤子科技

本篇报告阐述了锤子科技（北京）有限公司出品的 Smartisan OS 手机操作系统中云服务存在的安全缺陷，测试过程，以及缺陷导致的危害。并从公司管理的角度指出了信息安全，以及一只公司专属的安全团队对锤子科技的重要性。

由于本篇报告包含技术细节过多，因此仅作内部交流使用。 出于对锤子科技声誉的保护以及法律责任考虑，本报告应高度保密，不得外传。

1.2 Smartisan OS 及其云服务

Smartisan OS 是锤子科技推出的基于 Android 定制的移动操作系统。该系统于 2013 年 3 月 27 日在北京国家会议中心的“锤子智能手机操作系统发布会”上，由锤子科技的创始人罗永浩首次公布。 该操作系统目前由锤子科技官方适配四款手机。由于 2014 年 5 月 20 日发布会的临近，目前该系统版本号非常接近 1.0.0.0，但是并没有除优化便签等应用用户体验外的更大改进。

Smartisan OS 在 2014 年 4 月 11 日发布的 V0.9.9.7 alpha 版本中新增了云服务功能，用来供用户同步联系人，日历以及便签到锤子科技的服务器上。

目前上述云服务使用基于 Nginx 定制的 scws/2.4.9 作为 Web 服务器。scws 即 Smartisan Cloud Web Server 之意。同时，上述云服务使用锤子科技自行颁发的 X.509 证书 以及 TLS 1.2 协议对访问流量进行加密。

1.3 TLS 以及中间人攻击

TLS，即 Transport Layer Security，传输层安全协议，是 SSL（安全套接字）的继任者，二者均使用公开密钥技术对网络流量进行非对称加密。

中间人攻击，即 Man-in-the-middle Attack，是指攻击者分别与通信的两端建立连接，转发两者之间的流量，从而窃听通信内容的攻击方式。而 TLS 和 SSL 有防范中间人攻击的机制。在本次测试过程中，实际上更多地使用了中间人转发通信内容的方式获取锤子科技云服务的加密流量的明文以进行分析，而并非窃听他人的通信。

由于锤子科技云服务采用了 TLS 加密其 HTTP 流量，因此，在中间人代理服务器与锤子科技云服务通信时，流量采用锤子科技的 X.509 证书加密，而在云服务客户端程序与中间人代理服务器通信时，流量采用代理服务器自行生成的 X.509 证书加密。于是，中间人代理服务器便能够监听并篡改云服务客户端程序和云服务服务器之间的通信。

2 测试过程

2.1 测试过程概述

本次针对锤子科技云服务（下称云服务）的测试主要由三部分组成：反编译关键应用，模拟器环境下加密流量劫持以及针对 JSON API 的黑盒测试。实际上，上述三部分并没有如此清晰的划分，而是一个不断获取新的信息，不断深入挖掘安全缺陷的过程。第一部分，即反编译关键应用，揭示了云服务的基本请求格式以及 URL。第二部分探索了在 Android 模拟器环境下调试流量加密应用的基本原理和方法。第三部分通过上述一、二部分收集的信息，对云服务的 JSON API 进行了一系列安全检测，并发现了安全缺陷。

2.2 测试工具

本次测试中用到的工具主要有 Poster，Android 模拟器和 mitmproxy。
Poster 是 Firefox 浏览器的一款插件。通过这款插件，用户可以定义 HTTP 请求的几乎一切参数，从请求 URL 到请求头一应俱全。除常见的 GET，POST 以及 PUT 等 HTTP 方法外，Poster 还支持 HEAD，MOVE 等不常用的 HTTP 方法。
本次测试中使用的 Android 模拟器由锤子科技论坛网友使用 Google Android SDK 中的模拟器，通过打包 Smartisan OS 的系统文件并替换模拟器中的 system.img，定制而成，运行早期 Smartisan OS。通过修改设置中移动网络的 APN（接入点名称），配置 HTTP 代理服务器，可以实现流量转发，从而达到分析流量的目的。
mitmproxy，即 Man-in-the-Middle Proxy。顾名思义，mitmproxy 是一款通过中间人攻击的原理来进行流量分析和修改的代理服务器软件。mitmproxy 使用 Python 开发，支持针对 TLS 以及 SSL 的加密流量劫持。mitmproxy 有透明代理，HTTP 代理等多种模式。在 HTTP 代理模式下可以很好地配合 Android 模拟器工作。同时，mitmproxy 对 TLS 以及 SSL 加密流量的中间人劫持功能也使得针对加密的 HTTP 流量的分析成为可能。

2.3 反编译关键应用

在本次测试的第一部分，由于受限于硬件条件，无法进行真机调试，因此从静态分析入手，即通过分析云服务客户端程序的代码，而不是通过实际运行程序，来获得云服务系统的相关信息。
在下载锤子科技官方网站上的 V0.9.9.7 alpha 版压缩包后，解压 system 目录下的 CloudServiceSmartisan.apk，并使用通用的 apk 反编译工具对其进行反编译，生成 jar 文件。而后使用 Java Decompiler 查看由字节码反编译而成的源代码。

```
private void sendAccountInfoToServer()
{
  this.mContextHandler.sendMessage(this.mContextHandler.obtainMessage(7, Integer.valueOf(2131296355)
  JSONObject localJSONObject = new JSONObject();
  try
  {
    if (CloudSyncCommonUtil.validEmail(this.mAccountName)) {
      localJSONObject.put("email", this.mAccountName);
    }
    if (CloudSyncCommonUtil.ValidMobileNumber(this.mAccountName)) {
      localJSONObject.put("cellphone", this.mAccountName);
    }
    if (!TextUtils.isEmpty(this.mVerificationCodeValue)) {
      localJSONObject.put("captcha", this.mVerificationCodeValue);
    }
    localJSONObject.put("pwd", this.mPassword);
    CommonHttpUtils.postBackground("https://api.account.smartisan.cn/tickets.json", localJSONObject,
    return;
  }
  catch (Exception localException)
  {
    for (;;)
    {
      showToast(getString(2131296421), 0);
    }
  }
}
```

图 1 CloudServiceSmartisan.apk 登录相关部分代码

在名为 CommonHttpUtils 的类中可以看到一些反编译错误，以及认证用的名为 Ticket 的 HTTP 头，如下图所示。

```
public static void get(String paramString1, HttpCallBackListener paramHttpCallBackListener, String paramString2,
  throws Exception
{
  sBadTokenRetryTimes = 2;
  HashMap localHashMap = new HashMap();
  localHashMap.put("Ticket", paramString2);
  if (!TextUtils.isEmpty(paramString3)) {
    localHashMap.put("Local-Version", paramString3);
  }
  localHashMap.put("Content-type", "application/json");
  initParam(localHashMap);
  if (paramBoolean) {
    doHttpInCurrentThread("GET", paramString1, null, paramHttpCallBackListener, localHashMap);
  }
  for (;;)
  {
    return;
    doHttpBackground("GET", paramString1, null, paramHttpCallBackListener, localHashMap);
  }
}
```

图 2 GET 请求相关代码

而在 NoteTask 类中，可以看到更新，新建，删除便签的操作细节。如下图所示，创建便签操作的请求是一个发往 https://api.sync.cloud.smartisan.cn/USER_ID/notes.json 的 PUT 请求。

```
protected void upload(JSONArray paramJSONArray)
  throws Exception
{
  try
  {
    JSONObject localJSONObject = new JSONObject();
    localJSONObject.put("notes", paramJSONArray);
    CommonHttpUtils.putInCurrentThread("https://api.sync.cloud.smartisan.cn/" + this.mAccount.getId() + "/notes.json"
    {
      public void onEnd(String paramAnonymousString, HashMap<String, String> paramAnonymousHashMap)
        throws Exception
      {
        JSONObject localJSONObject1 = new JSONObject(paramAnonymousString);
        try
        {
          JSONObject localJSONObject2 = localJSONObject1.getJSONObject("notes").getJSONObject("syncid");
          Iterator localIterator = localJSONObject2.keys();
          while (localIterator.hasNext())
          {
            CloudNote localCloudNote = new CloudNote();
            String str = (String)localIterator.next();
            localCloudNote.mOperation = CloudObject.Operation.CREATE;
            localCloudNote.mId = str;
            localCloudNote.mSyncId = localJSONObject2.getString(str);
            NoteTask.this.addSnote(localCloudNote);
          }
          return;
        }
        catch (Exception localException) {}
      }

      public void onError(int paramAnonymousInt, String paramAnonymousString)
        throws Exception
      {
        throw new Exception("upload note error");
      }

      public void onStart(String paramAnonymousString) {}
    }, this.mAccount.getToken(), this.dbHelper.getLastSyncVerByType(getTaskId(), this.mAccount.getId()));
    return;
  }
```

图 3 NoteTask 类中上传便签相关代码

2.4 模拟器环境下加密流量劫持

在本次测试的第二部分，上文所述 Android 模拟器，即“锤子系统电脑版”，被用来运行第一部分中所提到的云服务客户端程序，即 CloudServiceSmartisan.apk。同时，一台运行 Ubuntu 操作系统的虚拟机被用来运行 mitmproxy，作为 Android 模拟器的 HTTP 代理服务器以劫持云服务客户端程序与服务器的通信。

启动“锤子系统电脑版”后发现，这款 Android 模拟器的所载入的 Smartisan OS 发布于 2013 年 7 月，并不具有 V0.9.9.7 alpha 版本的云服务功能。由于在使用最新版的 Smartisan OS 重新打包 system.img 文件后，模拟器并不能正常启动，因此转而使用 Android 系统调试工具 adb 安装 CloudServiceSmartisan.apk。安装成功后可在“设置”中选择“添加账户”来登录或注册云服务账户。

图 4 Smartisan OS 云服务客户端

根据第一部分中对云服务客户端程序反编译获得的源代码的分析，以及概述中对中间人攻击的介绍，由于中间人代理服务器与云服务客户端之间的通信会使用代理服务器自行生成的 X.509 证书，因此必须要将该证书导入 Android 模拟器所加载的操作系统中。导入过程并不复杂，既可以将 BASE64 编码的证书文件直接复制到/etc/security/cacerts 目录，并设置权限为 644；也可以将相应证书文件复制到/sdcard 目录后，在“设置”中的“安全”菜单选择“从 SD 卡安装”。作为 Android 内建安全措施之一，导入证书时采用后者将会强制用户设置锁屏密码。

在安装并第一次运行 mitmproxy 后，mitmproxy 生成的 X.509 证书可在当前用户 home 目录下的.mitmproxy 下找到。将其中的 mitmproxy.cer 通过 adb 复制到模拟器的/sdcard 目录下，进行安装。

此外，还需要设置恰当的 APN 来配置操作系统使用中间人代理服务器作为 HTTP 代理。打开“设置”菜单，选择“无线和网络下”的“更多”，选择“移动网络”，“接入点名称（APN）”。“名称”任意设置，“APN”设置为“Internet”，并将代理服务器地址和端口设置为运行 mitmproxy 的虚拟机的 IP 地址和端口，默认为 8080 端口。

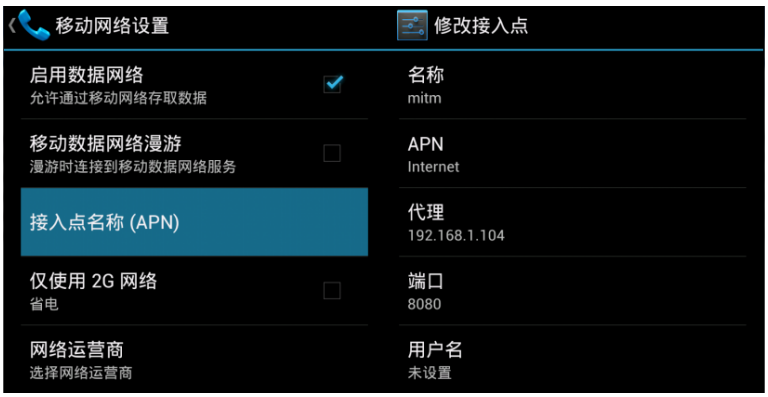

图 5 Android 模拟器代理设置

使用任意账号密码测试登录，在 mitmproxy 界面可以看到请求的 URL 以及 JSON 请求格式，与第一部分中登录部分相关代码相符。

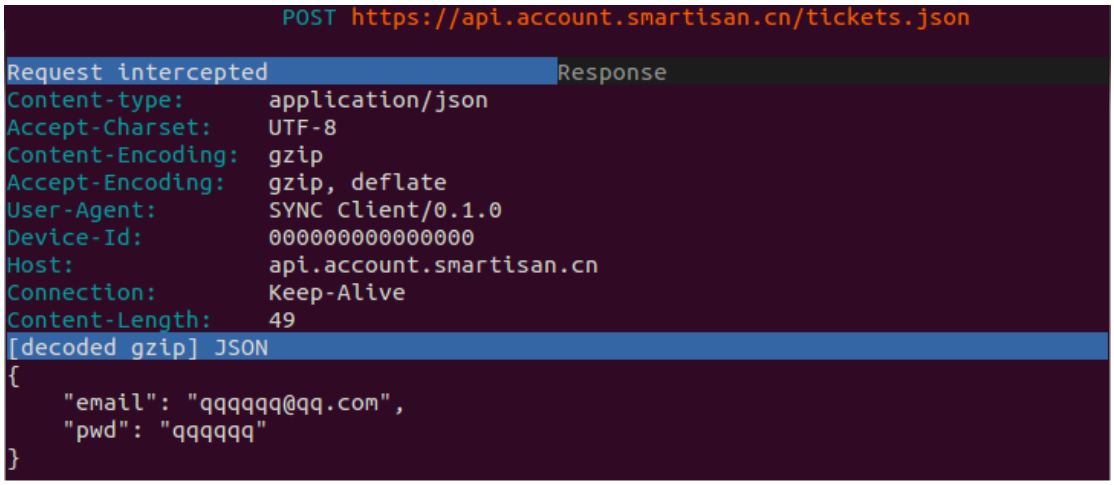

图 6 mitmproxy 截获的登录请求

2.5 针对 JSON API 的黑盒测试

根据前两部分获得的云服务相关信息可以发现，云服务接口实际上是一系列 JSON API。而这些 API 分为两台服务器：api.account.smartisan.cn 和 api.sync.cloud.smartisan.cn。其中前者负责账户相关 API 请求，后者负责实际同步内容相关 API 请求。下表列出了部分常用 API 以及用途。

URL	备注
https://api.account.smartisan.cn/tickets.json	POST 方法；登录
https://api.account.smartisan.cn/user/prereg.json	POST 方法；短信验证相关
https://api.account.smartisan.cn/user/USER_ID.json	GET 方法；用户基本信息
https://api.account.smartisan.cn/user/USER_ID/avatar	GET 方法；用户头像
https://api.sync.cloud.smartisan.cn/USER_ID/notes/list.json	GET 方法；便签同步列表
https://api.sync.cloud.smartisan.cn/USER_ID/notes.json	GET/PUT 方法；便签操作
https://api.sync.cloud.smartisan.cn/USER_ID/ contacts/list.json	GET 方法；联系人同步列表
https://api.sync.cloud.smartisan.cn/USER_ID/contacts.json	GET/PUT 方法；联系人操作
https://api.sync.cloud.smartisan.cn/USER_ID/calendars/list.json	GET 方法；日历同步列表
https://api.sync.cloud.smartisan.cn/USER_ID/calendars.json	GET/PUT 方法；日历操作

表 1 部分 API URL

其中用户基本信息、便签、联系人以及日历 API 均需要验证身份。身份验证是通过验证 HTTP 请求头中的“Ticket”字段进行的。

通常对于计算机程序和系统的测试都是从用户输入开始的。根据以上的信息，云服务的 API 主要有三类用户可控的输入：URL 参数，即用户 ID，HTTP 请求头中的字段，即“Ticket”等，以及 JSON 格式的请求主体。

经过一系列检测，在第一类用户输入中并没有发现云服务 API 响应异常，推测原因为

API 过滤了所有非数字字符。

而 HTTP 请求头中，版本参数，User-Agent 等字段均不能导致 API 响应异常。真正有趣的地方在身份认证使用的“Ticket”字段上。下表列出了针对用户基本信息 API 的“Ticket”字段部分测试输入以及响应。

测试输入	API 响应
999' #	{"code":404125}
999' or 1=1 #	{"code":401}
999' order by 6#	{"code":500013}
999' order by 5#	{"code":404125}
999' union select 3,3,3,3,3#	{"code":404122}

表 2 用户基本信息 API Ticket 字段部分测试结果

由此，用户基本信息 API 被确认在“Ticket”字段部分存在 SQL 注入漏洞，并且可以确认检查令牌的 SQL 查询返回字段数为 5。可以推测出，响应中的“401”意为找到了令牌但是用户 ID 与 URL 中的 USER_ID 参数不符，因此认证失败；“404125”意味着没有找到相应令牌记录；而“404122”则意味着并未找到相应的用户。

此外，在针对登录和短信验证相关 API 的测试中发现，请求的 JSON 格式数据中的“email”字段也存在注入。由于云服务 API 在处理“email”字段时过滤了部分不符合电子邮件地址命名规则的字符（如空格，逗号，全角字符等），因此此处 SQL 注入点威胁并不大。下表列出了针对登录 API 的部分测试输入以及响应。

“email”字段测试输入	API 响应
qqq'/**/order/**/by/**/3#@qq.com	{"code":404123}
qqq'/**/order/**/by/**/4#@qq.com	{"code":500013}
qqq'@qq.com	{"code":500013}

表 3 登录 API JSON 请求数据 email 字段部分测试结果

令人感到奇怪的是，提交到 api.sync.cloud.smartisan.cn 服务器的所有请求的 HTTP 请求头中的“Ticket”字段均经过了良好的过滤，并不存在 SQL 注入漏洞。而提交到 api.account.smartisan.cn 服务器的请求头中的“Ticket”字段并未经过过滤。这意味着两台服务器运行着完全不同的用户身份认证模块，而这对于一个有着如此预期用户规模的云服务来说，是不可想象的。

3 安全缺陷

3.1 用户头像泄露

由于用于获取用户头像的 API 并没有身份认证，因此可以根据用户 ID 任意下载用户头像。下图展示了用户 ID 小于 200 的部分内部测试用头像。

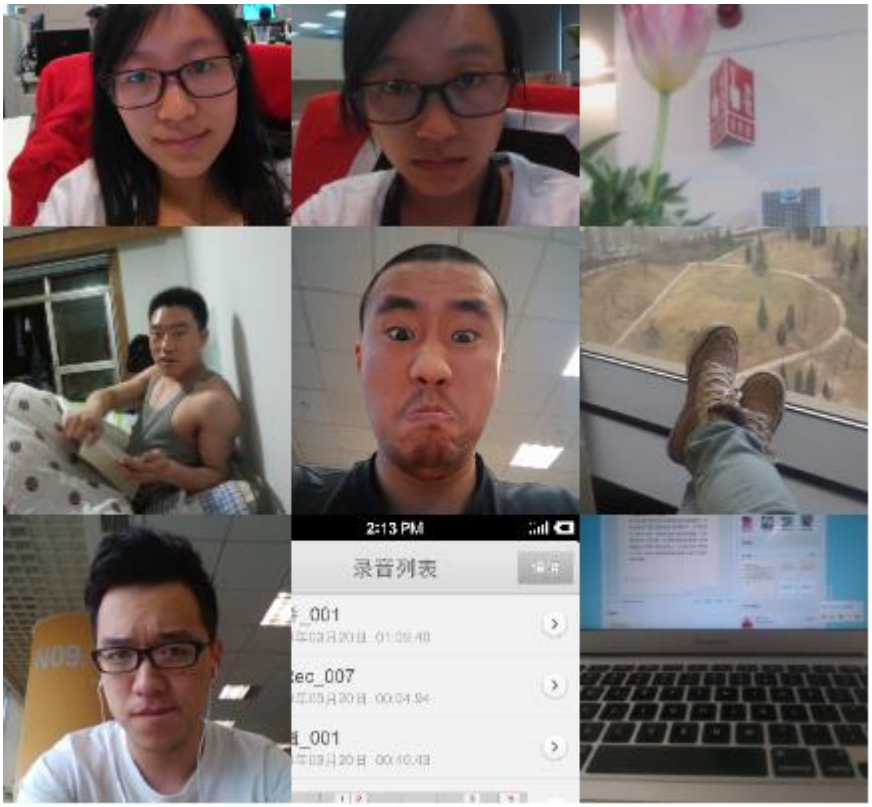

图 7 用户 ID 小于 200 的部分测试用头像

用户 ID 大于 1000000 的为公开测试用户，目前约 3000 名用户中有 707 人设置了头像。下图是部分公开测试用户的头像。

图 8 部分公开测试用户头像

如果从安全缺陷的严格定义出发，用户头像并不是重要的隐私信息，并且用户头像本身就存在一定的社交属性。因此不经身份认证也可访问用户头像并不属于安全缺陷。

3.2 用户个人信息泄露

经过针对用户个人信息 API 的测试，已经确定用户基本信息 API 请求头中的“Ticket”字段存在 SQL 注入漏洞。通过闭合“Ticket”字段查询条件，并加入 OR 条件指明“uid”字段取值，可以绕过用户身份认证，获取用户电话号码，邮箱地址，注册时间，以及是否通过短信和邮件认证等信息。

图 9 通过 Ticket 字段注入获得 uid 为 1002337 用户的个人信息

由上图可见，uid 为 1002337 的这位用户电话号码为 18651488100，邮箱地址为 348037907@qq.com。

3.3 用户同步令牌泄露

根据上文 3.5 节中对表 2 测试结果的分析，用户基本信息 API 在未找到请求中“Ticket”字段对应的令牌时，返回代码“404125”，即令牌未找到；而在找到令牌，但用户 ID 与 URL 中请求的 USER_ID 参数不符时，返回代码“401”，即认证失败。由此，可在用户基本信息 API 请求中“Ticket”字段后加入 OR 条件来测试任意 SQL 表达式。如果表达式为真，用户基本信息 API 则返回代码“401”，如果表达式为假，用户基本信息 API 则返回代码“404125”，从而使 SQL 盲注成为可能。

经过针对用户基本信息 API 的 SQL 盲注测试发现，可以通过提交精心构造“Ticket”字段的 API 请求来获取 SQL 表名，字段名，以及当前运行 SQL 版本等信息。经过检测，数据库名为 cloud_account，存在用户表 7 个，分别是 account_cellphone_uid，account_email_uid，account_failedlogin，account_seccode，account_seccodes，account_tickets 以及 account_users。其中 account_tickets 表存在 5 个字段，分别是 id，ticket，uid，create_time 以及 expire_time。

通过利用这一安全缺陷，可以获取指定用户 ID 的同步令牌，从而进一步获取该用户的便签，联系人和日历信息，此外还可以修改用户昵称，头像，甚至添加、修改联系人电话号码，实施电信诈骗。

3.4 安全缺陷成因分析及临时解决方案

针对第一点缺陷，如何处理用户头像的访问权限问题，显然要根据具体的软件需求来决定，但是仅就目前来说，为了防止用户头像被恶意抓取，可以在 URL 中加入 hash。

上述第二、三两点安全缺陷的成因是 API 将“Ticket”字段内容，以及用户输入的“email”字段内容直接代入了 SQL 查询语句，从而导致云服务 API 存在 SQL 注入缺陷。在测试过程中可以发现，虽然云服务 API 对用户输入进行了合理性过滤，如 URL 中 USER_ID 参数只允许数字字符，用户输入的“email”字段不允许逗号，空格等非电子邮件地址字符，但是却完全没有进行安全性过滤，从而导致用户输入直接被代入执行。

此外，云服务 API 应尽量少地将应用程序内部状态暴露给外界。如果不能通过云服务 API 的响应区分令牌未找到和认证未通过，攻击者将无从判断精心构造的恶意“Ticket”字段内容是否起效，便不会有进一步的攻击。而最小化对外界暴露无关信息，也是开发接口的重要意义之一。

作为临时解决方案，可以在用户输入内容被云服务 API 处理前，对其进行安全性过滤，并修改 ORM 模块，使用预编译 SQL 语句的方式，而不是字符串拼接的方式处理 SQL 查询，从而避免 SQL 注入攻击。

5 总结

本次测试的结果说明，锤子科技云服务开发团队严重缺乏安全意识，认识不到对用户输入数据过滤的重要性。并且对作为最基本的攻击类别之一的 SQL 注入攻击的成因以及危害完全没有认识。从软件工程的角度来说，锤子科技云服务开发团队没有使用业界先进的程序设计技术，开发的产品存在不一致性，基础模块（即用户身份认证模块）不能通用。

总而言之，锤子科技管理层应提高自身安全意识，同时对开发以及运维部门进行相关培训。尤为重要的是，锤子科技应尽快聘请或建立一支安全团队，以便对突发安全事件进行紧急响应，并对网络服务的日常运行日志进行审计。除此之外，拥有公司专属的安全团队还可以提高公司办公网络基础架构的安全性，降低公司遭受有针对性、有预谋的综合性社会工程学攻击的危害。

安全绝不是部署某种或某些安全产品，采用某种相对安全的网络架构就可以高枕无忧的问题。安全是一个全方位的系统，其中最重要的部分是人，也即锤子科技的全体员工。