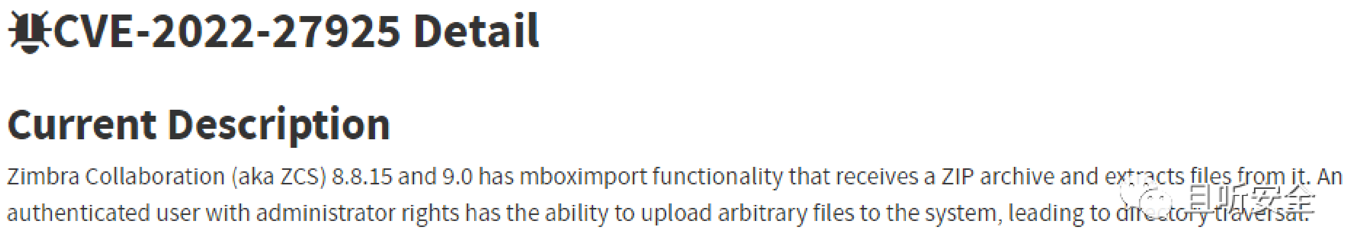
CVE-2022-27925 Zimbra Collaboration 存在路径穿越漏洞最终导致RCE

白帽子社区 2022-06-18 17:10

以下文章来源于且听安全， 作者QCyber

漏洞信息

前段时间 Zimbra 官方通报了一个 RCE 漏洞 CVE-2022-27925 ，也有小伙伴在漏洞空间站谈到了这个漏洞，上周末在家有时间完成了漏洞的分析与复现。漏洞原理并不复杂，但在搭建环境的过程中遇到了一些坑，下面将分析过程分享给大家。


CVE-2022-27925 Detail

Current Description

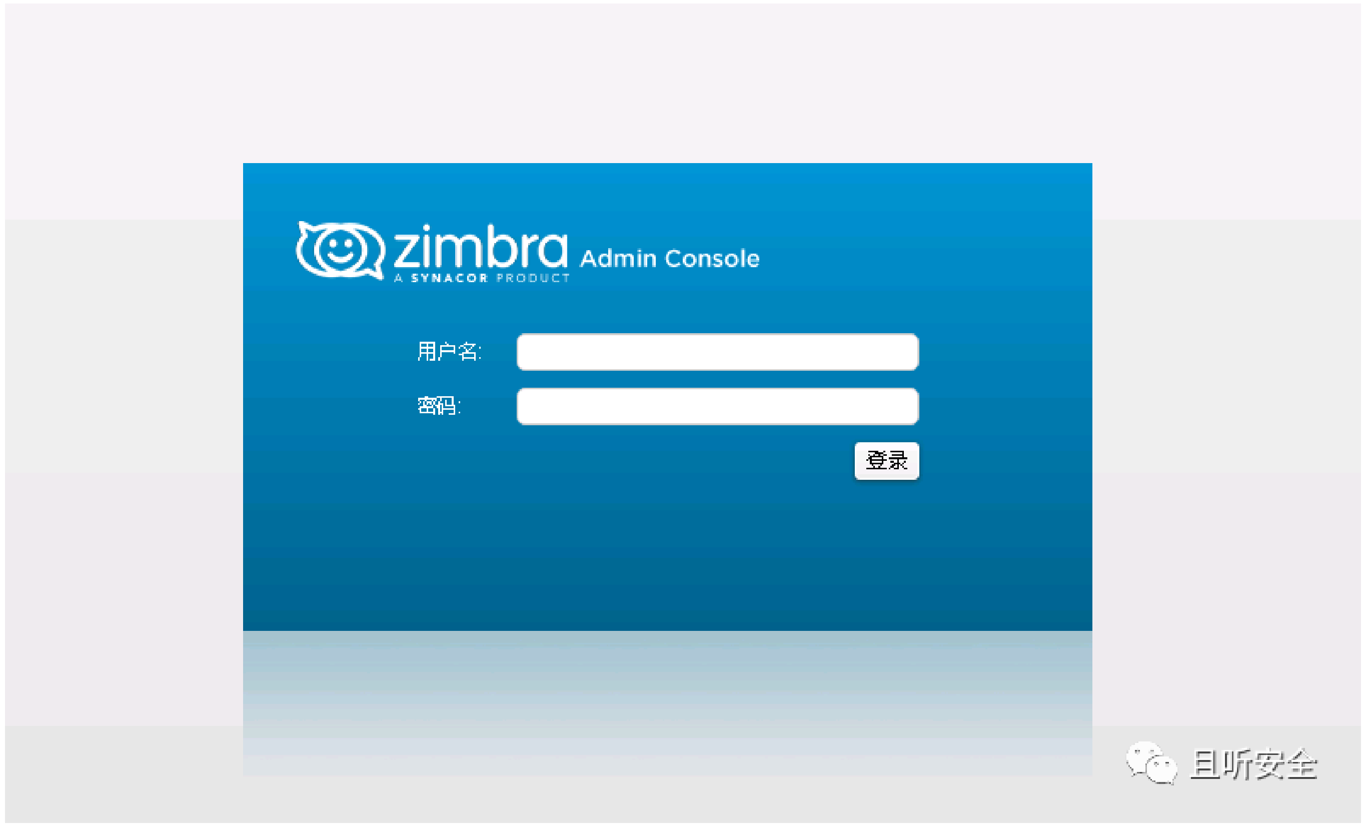
Zimbra Collaboration (aka ZCS) 8.8.15 and 9.0 has mboximport functionality that receives a ZIP archive and extracts files from it. An authenticated user with administrator rights has the ability to upload arbitrary files to the system, leading to directory traversal.

从描述来看，这是一个 ZIP 压缩包解析导致路径穿越类型的漏洞。

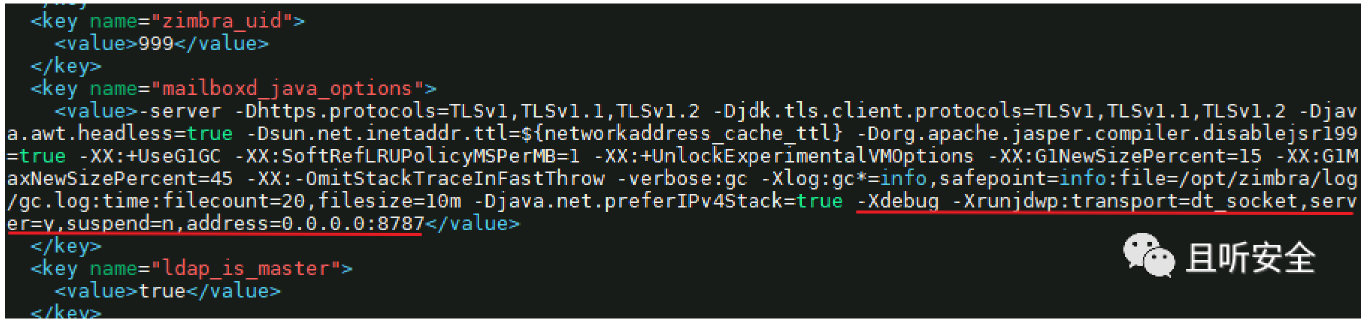
环境搭建

由于直接安装 v9.0.0 或 v8.8.15 默认就是最新版，因此选择安装 v8.8.12。安装过程非常曲折，**环境搭建有疑惑的小伙伴可以加入漏洞空间站进行交流。**

最终完成安装并启动成功：

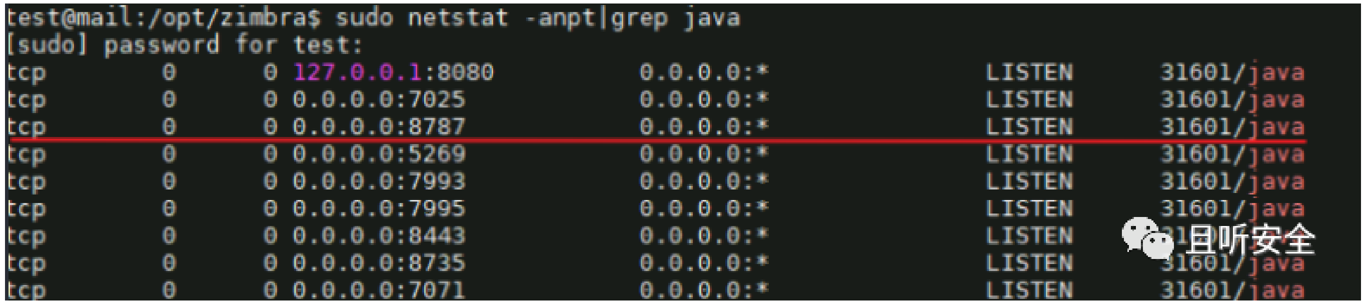


通过配置 `mailboxd_java_options` 加入调试信息：



```
  <key name="zimbra_uid">
    <value>999</value>
  </key>
  <key name="mailboxd_java_options">
    <value>-server -Dhttps.protocols=TLSv1,TLSv1.1,TLSv1.2 -Djdk.tls.client.protocols=TLSv1,TLSv1.1,TLSv1.2 -Djav
a.awt.headless=true -Dsun.net.inetaddr.ttl=${networkaddress_cache_ttl} -Dorg.apache.jasper.compiler.disablejsr199
=true -XX:+UseG1GC -XX:SoftRefLRUPolicyMSPerMB=1 -XX:+UnlockExperimentalVMOptions -XX:G1NewSizePercent=15 -XX:G1M
axNewSizePercent=45 -XX:-OmitStackTraceInFastThrow -verbose:gc -Xlog:gc*=info,safepoint=info:file=/opt/zimbra/log
/gc.log:time:filecount=20,filesize=10m -Djava.net.preferIPv4Stack=true -Xdebug -Xrunjdwp:transport=dt_socket,serv
er=y,suspend=n,address=0.0.0.0:8787</value>
  </key>
  <key name="ldap_is_master">
    <value>true</value>
  </key>
```

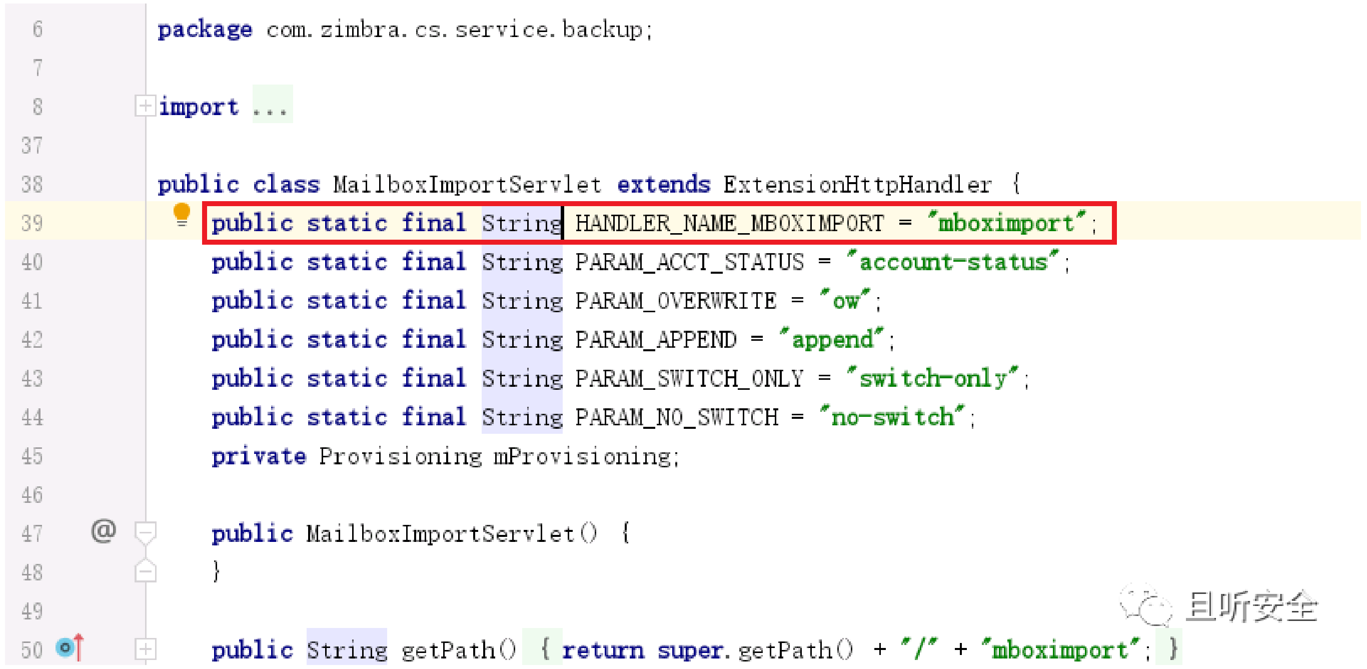
重启 Zimbra 服务即可打开远程调试：



```
test@mail:/opt/zimbra$ sudo netstat -anpt|grep java
[sudo] password for test:
tcp        0      0 127.0.0.1:8080          0.0.0.0:*               LISTEN      31601/java
tcp        0      0 0.0.0.0:7025            0.0.0.0:*               LISTEN      31601/java
tcp        0      0 0.0.0.0:8787            0.0.0.0:*               LISTEN      31601/java
tcp        0      0 0.0.0.0:5269            0.0.0.0:*               LISTEN      31601/java
tcp        0      0 0.0.0.0:7993            0.0.0.0:*               LISTEN      31601/java
tcp        0      0 0.0.0.0:7995            0.0.0.0:*               LISTEN      31601/java
tcp        0      0 0.0.0.0:8443            0.0.0.0:*               LISTEN      31601/java
tcp        0      0 0.0.0.0:8735            0.0.0.0:*               LISTEN      31601/java
tcp        0      0 0.0.0.0:7071            0.0.0.0:*               LISTEN      31601/java
```

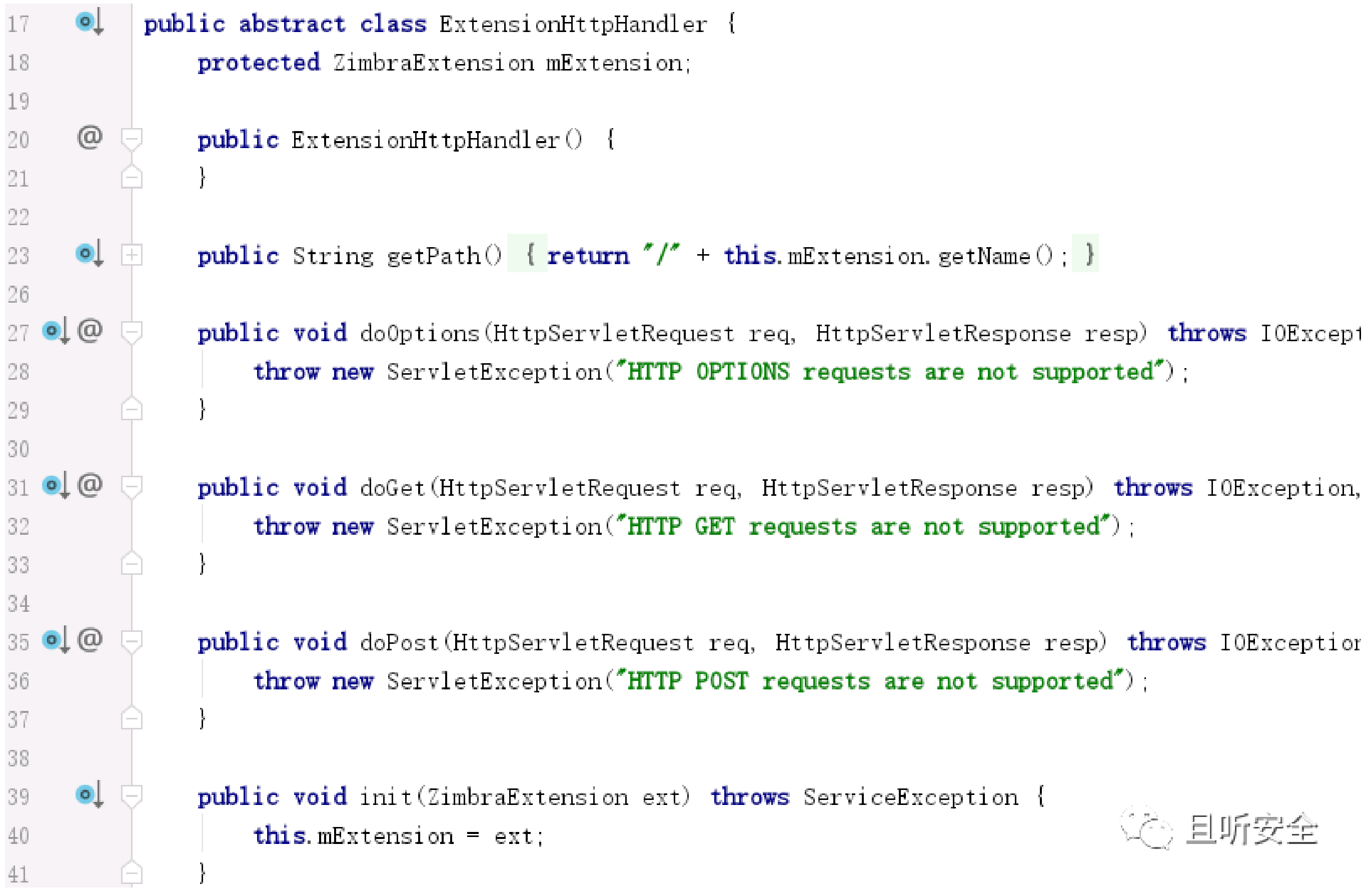
寻找调用链

漏洞出现在 `mboximport` 相关的功能中，全盘搜索定位到位于 `zimbrabackup.jar` 中的 `MailboxImportServlet`：



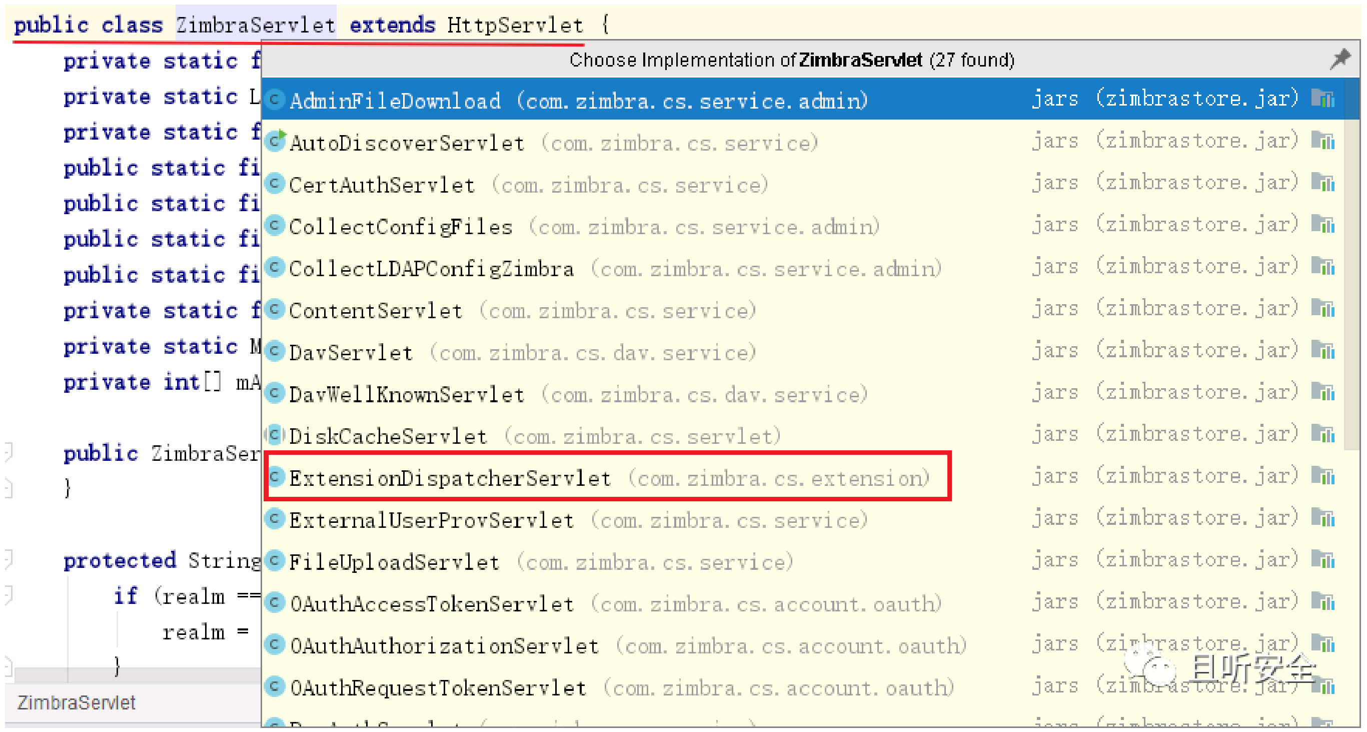
```
package com.zimbra.cs.service.backup;

import ...

public class MailboxImportServlet extends ExtensionHttpHandler {
    public static final String HANDLER_NAME_MBOXIMPORT = "mboximport";
    public static final String PARAM_ACCT_STATUS = "account-status";
    public static final String PARAM_OVERWRITE = "ow";
    public static final String PARAM_APPEND = "append";
    public static final String PARAM_SWITCH_ONLY = "switch-only";
    public static final String PARAM_NO_SWITCH = "no-switch";
    private Provisioning mProvisioning;

    public MailboxImportServlet() {
    }

    public String getPath() { return super.getPath() + "/" + "mboximport"; }
```

从命名规则和存在的成员函数 `doPost` 来看， `MailboxImportServlet` 应该对应一个 `Servlet` 对象，但是 `MailboxImportServlet` 继承于 `ExtensionHttpHandler` 而非 `HttpServlet` ：



```java
public abstract class ExtensionHttpHandler {
    protected ZimbraExtension mExtension;

    public ExtensionHttpHandler() {
    }

    public String getPath() { return "/" + this.mExtension.getName(); }

    public void doOptions(HttpServletRequest req, HttpServletResponse resp) throws IOExcept
        throw new ServletException("HTTP OPTIONS requests are not supported");
    }

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException,
        throw new ServletException("HTTP GET requests are not supported");
    }

    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOExceptior
        throw new ServletException("HTTP POST requests are not supported");
    }

    public void init(ZimbraExtension ext) throws ServiceException {
        this.mExtension = ext;
    }
```

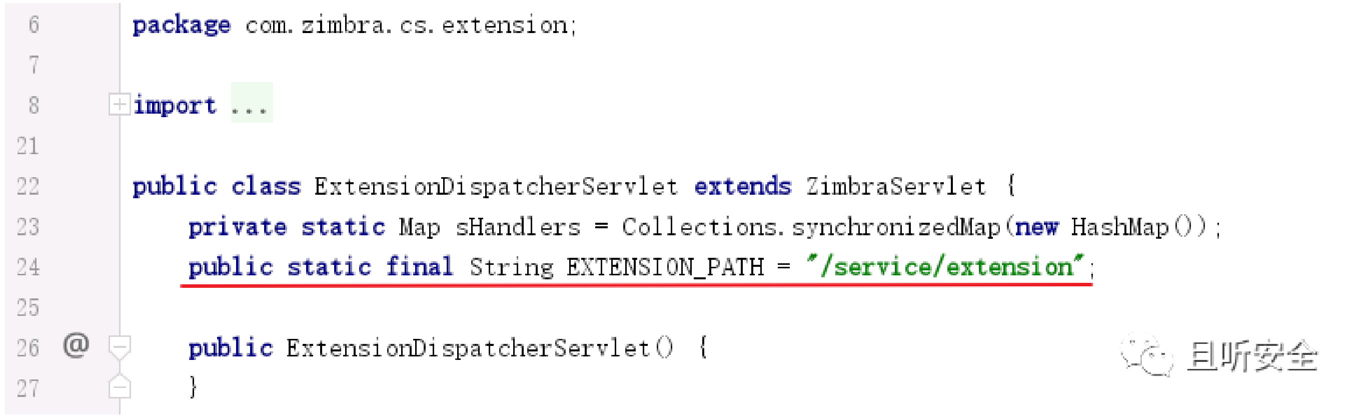
所以还需要寻找某种相互之间的转换关系。我们知道 Zimbra 自定义了 `Servlet` 对象的基类 `ZimbraServlet` ，搜索其子类：



```
public class ZimbraServlet extends HttpServlet {
    private static f
    private static L
    private static f
    public static fi
    public static fi
    public static fi
    public static fi
    private static f
    private static M
    private int[] mA

    public ZimbraSer
    }

    protected String
        if (realm ==
            realm =
        }
```

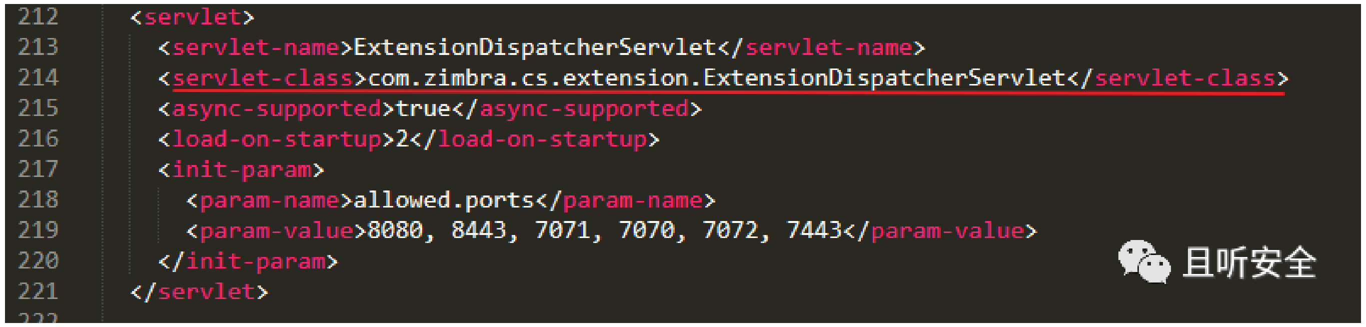
```
Choose Implementation of ZimbraServlet (27 found)
AdminFileDownload (com.zimbra.cs.service.admin)              jars (zimbrastore.jar)
AutoDiscoverServlet (com.zimbra.cs.service)                  jars (zimbrastore.jar)
CertAuthServlet (com.zimbra.cs.service)                      jars (zimbrastore.jar)
CollectConfigFiles (com.zimbra.cs.service.admin)             jars (zimbrastore.jar)
CollectLDAPConfigZimbra (com.zimbra.cs.service.admin)        jars (zimbrastore.jar)
ContentServlet (com.zimbra.cs.service)                       jars (zimbrastore.jar)
DavServlet (com.zimbra.cs.dav.service)                       jars (zimbrastore.jar)
DavWellKnownServlet (com.zimbra.cs.dav.service)              jars (zimbrastore.jar)
DiskCacheServlet (com.zimbra.cs.servlet)                     jars (zimbrastore.jar)
ExtensionDispatcherServlet (com.zimbra.cs.extension)         jars (zimbrastore.jar)
ExternalUserProvServlet (com.zimbra.cs.service)              jars (zimbrastore.jar)
FileUploadServlet (com.zimbra.cs.service)                    jars (zimbrastore.jar)
OAuthAccessTokenServlet (com.zimbra.cs.account.oauth)        jars (zimbrastore.jar)
OAuthAuthorizationServlet (com.zimbra.cs.account.oauth)      jars (zimbrastore.jar)
OAuthRequestTokenServlet (com.zimbra.cs.account.oauth)       jars (zimbrastore.jar)
```

定位 `ExtensionDispatcherServlet` ：

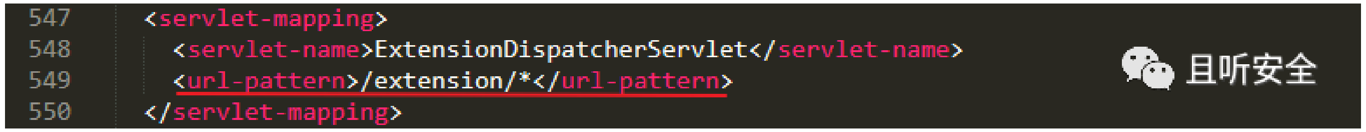


```
package com.zimbra.cs.extension;

import ...

public class ExtensionDispatcherServlet extends ZimbraServlet {
    private static Map sHandlers = Collections.synchronizedMap(new HashMap());
    public static final String EXTENSION_PATH = "/service/extension";

    public ExtensionDispatcherServlet() {
    }
```

可以找到相关配置：

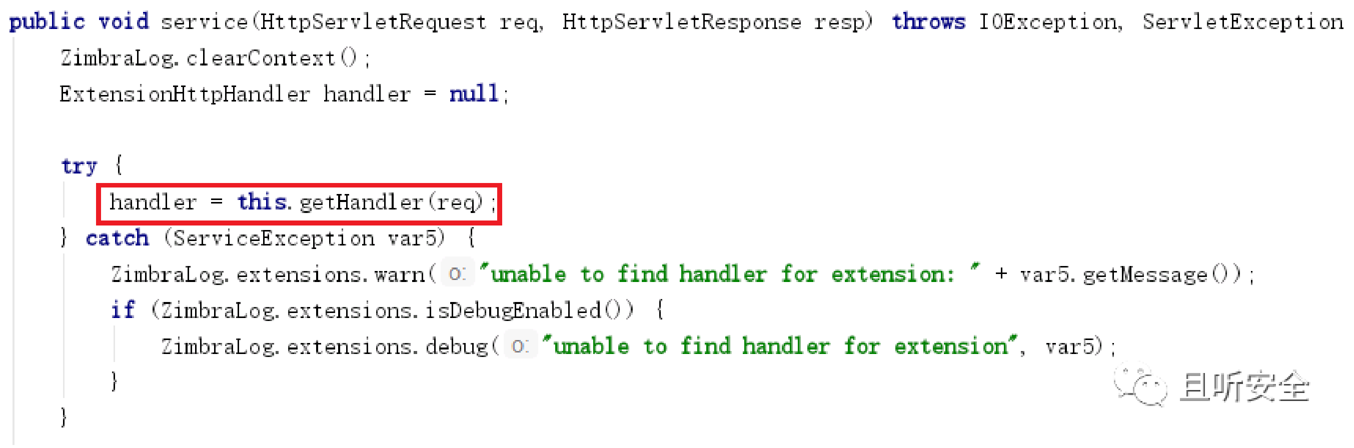


```xml
<servlet>
  <servlet-name>ExtensionDispatcherServlet</servlet-name>
  <servlet-class>com.zimbra.cs.extension.ExtensionDispatcherServlet</servlet-class>
  <async-supported>true</async-supported>
  <load-on-startup>2</load-on-startup>
  <init-param>
    <param-name>allowed.ports</param-name>
    <param-value>8080, 8443, 7071, 7070, 7072, 7443</param-value>
  </init-param>
</servlet>
```



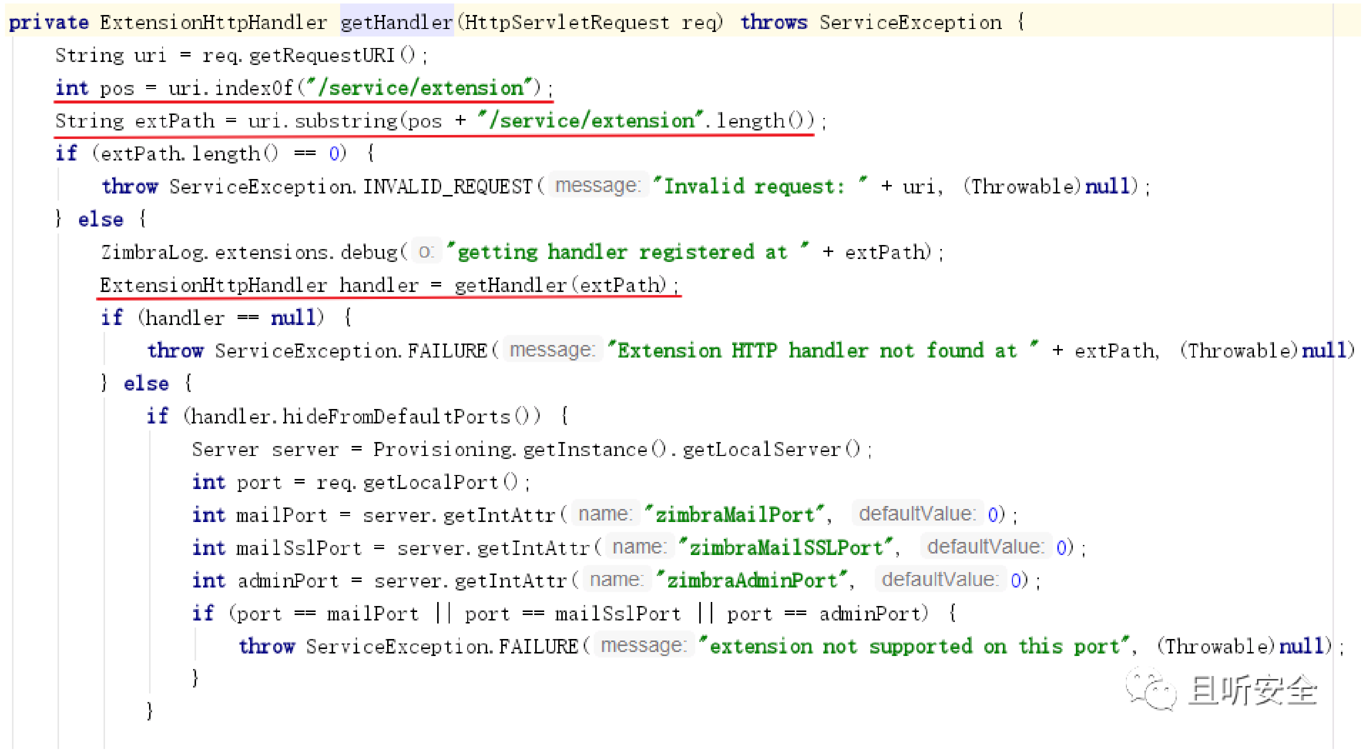
```xml
<servlet-mapping>
  <servlet-name>ExtensionDispatcherServlet</servlet-name>
  <url-pattern>/extension/*</url-pattern>
</servlet-mapping>
```

所 以 `ExtensionDispatcherServlet` 对 应 的 URL 规 则 为 `/service/extension/*` ， 回 到 `ExtensionDispatcherServlet#service` 函数：



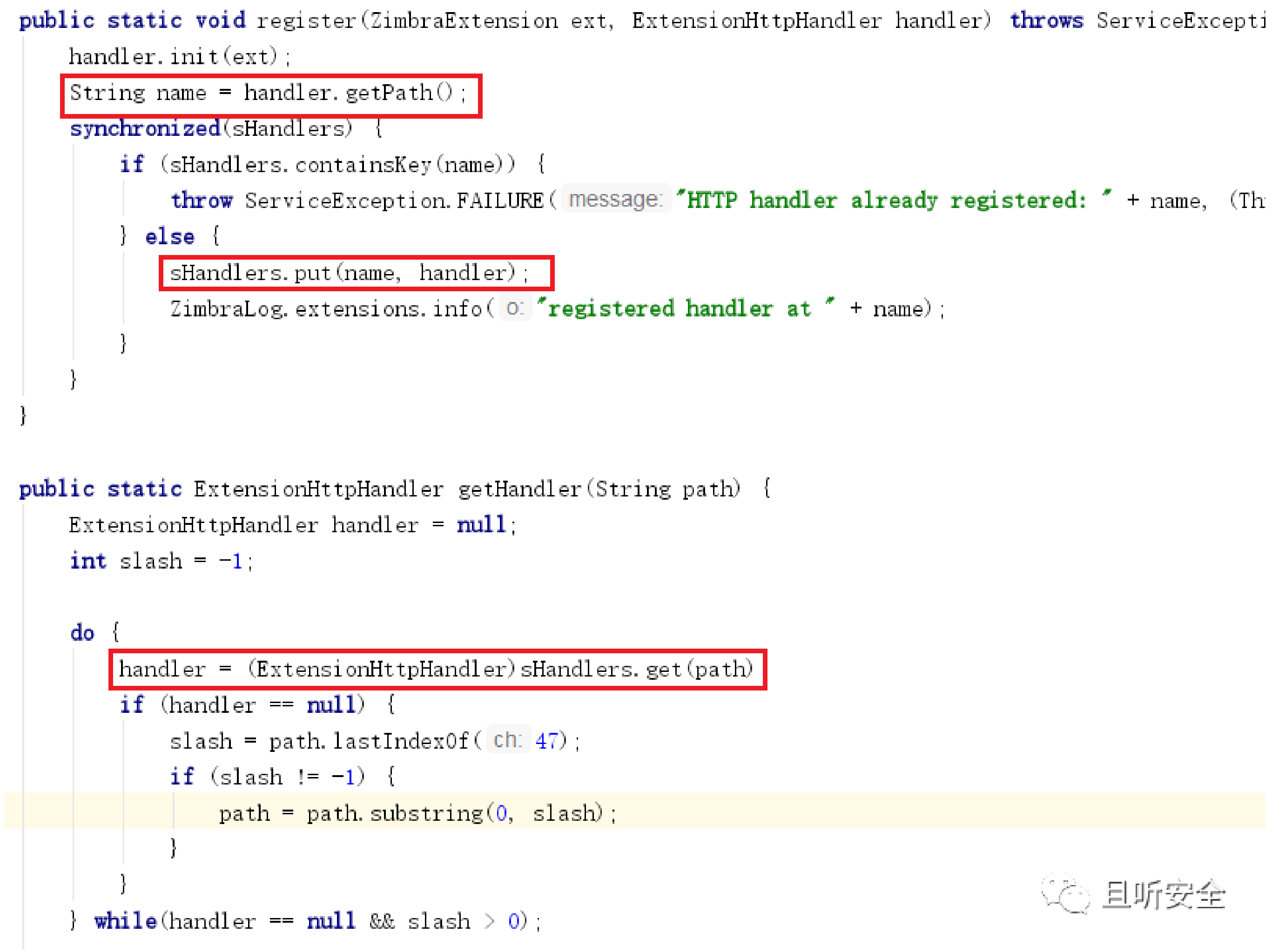
```java
public void service(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException
    ZimbraLog.clearContext();
    ExtensionHttpHandler handler = null;

    try {
        handler = this.getHandler(req);
    } catch (ServiceException var5) {
        ZimbraLog.extensions.warn( o: "unable to find handler for extension: " + var5.getMessage());
        if (ZimbraLog.extensions.isDebugEnabled()) {
            ZimbraLog.extensions.debug( o: "unable to find handler for extension", var5);
        }
    }
```

通过 `getHandler` 函数来寻找对应的 `ExtensionHttpHandler` 对象 `handler` （前面定位的 `MailboxImportServlet` 正好继承于 `ExtensionHttpHandler`），进入 `getHandler` 函数：



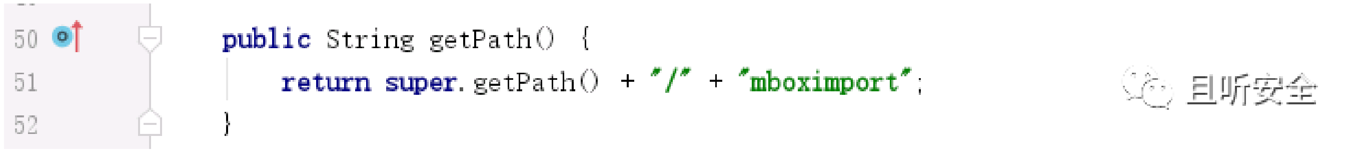
```
private ExtensionHttpHandler getHandler(HttpServletRequest req) throws ServiceException {
    String uri = req.getRequestURI();
    int pos = uri.indexOf("/service/extension");
    String extPath = uri.substring(pos + "/service/extension".length());
    if (extPath.length() == 0) {
        throw ServiceException.INVALID_REQUEST( message: "Invalid request: " + uri, (Throwable)null);
    } else {
        ZimbraLog.extensions.debug( o: "getting handler registered at " + extPath);
        ExtensionHttpHandler handler = getHandler(extPath);
        if (handler == null) {
            throw ServiceException.FAILURE( message: "Extension HTTP handler not found at " + extPath, (Throwable)null)
        } else {
            if (handler.hideFromDefaultPorts()) {
                Server server = Provisioning.getInstance().getLocalServer();
                int port = req.getLocalPort();
                int mailPort = server.getIntAttr( name: "zimbraMailPort",  defaultValue: 0);
                int mailSslPort = server.getIntAttr( name: "zimbraMailSSLPort",  defaultValue: 0);
                int adminPort = server.getIntAttr( name: "zimbraAdminPort",  defaultValue: 0);
                if (port == mailPort || port == mailSslPort || port == adminPort) {
                    throw ServiceException.FAILURE( message: "extension not supported on this port", (Throwable)null);
                }
            }
```

提取 URL 中 `/service/extension` 之后的字符串并赋值给 `extPath` ，带入 `getHandler` 函数：



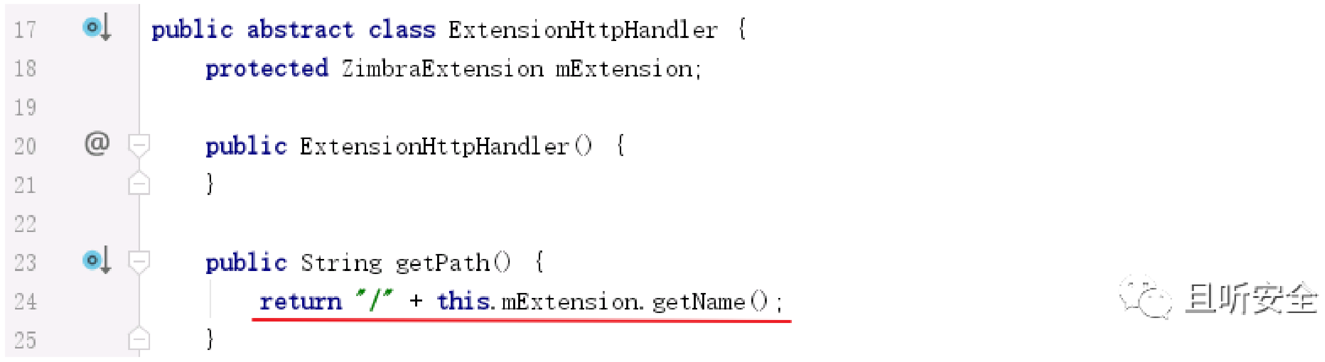
```java
public static void register(ZimbraExtension ext, ExtensionHttpHandler handler) throws ServiceExcepti
    handler.init(ext);
    String name = handler.getPath();
    synchronized(sHandlers) {
        if (sHandlers.containsKey(name)) {
            throw ServiceException.FAILURE( message: "HTTP handler already registered: " + name, (Th
        } else {
            sHandlers.put(name, handler);
            ZimbraLog.extensions.info( o: "registered handler at " + name);
        }
    }
}

public static ExtensionHttpHandler getHandler(String path) {
    ExtensionHttpHandler handler = null;
    int slash = -1;

    do {
        handler = (ExtensionHttpHandler)sHandlers.get(path)
        if (handler == null) {
            slash = path.lastIndexOf( ch: 47);
            if (slash != -1) {
                path = path.substring(0, slash);
            }
        }
    } while(handler == null && slash > 0);
```

返回的 `ExtensionHttpHandler` 对象来自于 `sHandlers` 键值对，其中的 `key` 来自于 `ExtensionHttpHandler#getPath` 函数，查看定义：

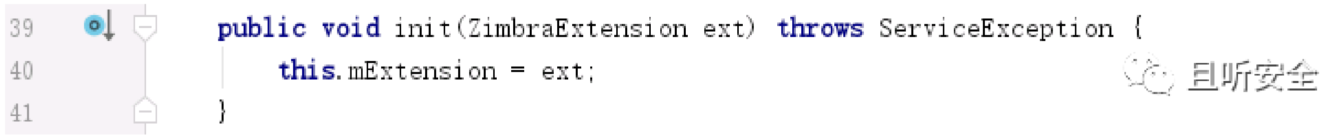


```java
public String getPath() {
    return super.getPath() + "/" + "mboximport";
}
```

`ExtensionHttpHandler#getPath`：

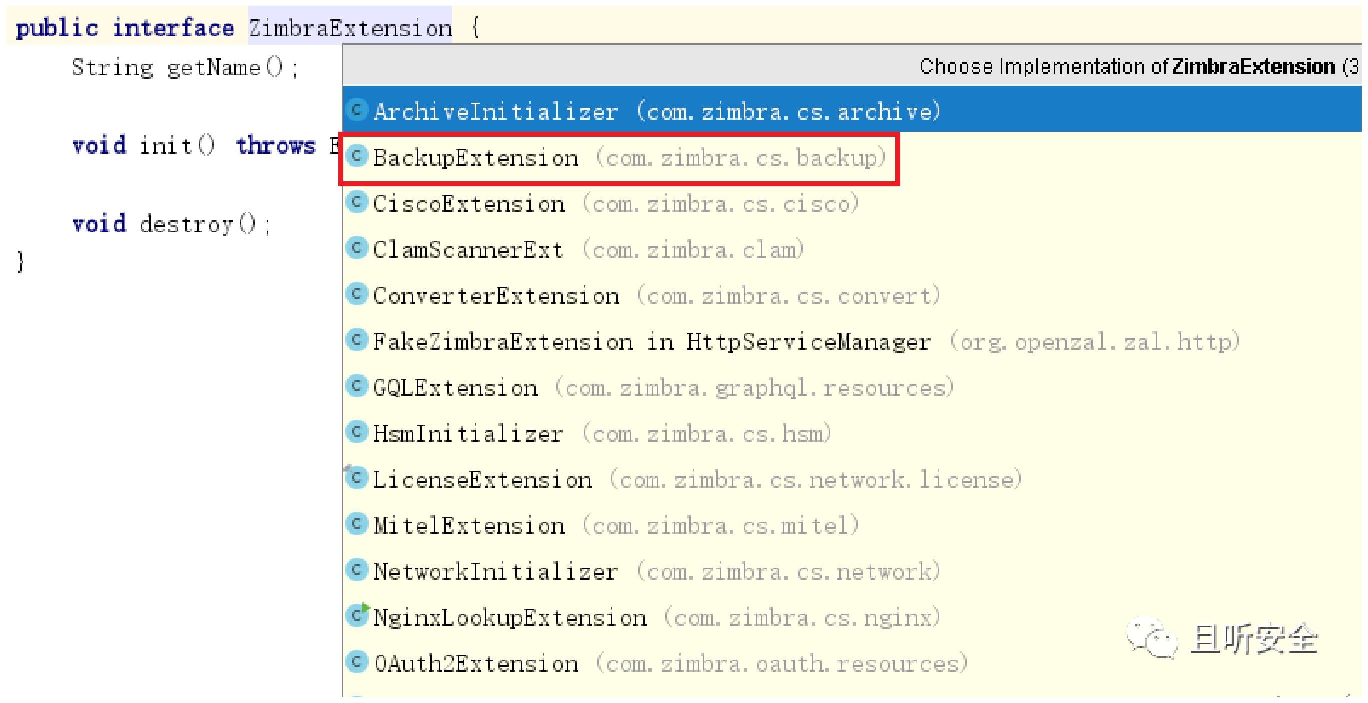


```java
public abstract class ExtensionHttpHandler {
    protected ZimbraExtension mExtension;

    public ExtensionHttpHandler() {
    }

    public String getPath() {
        return "/" + this.mExtension.getName();
    }
```



```java
public void init(ZimbraExtension ext) throws ServiceException {
    this.mExtension = ext;
}
```

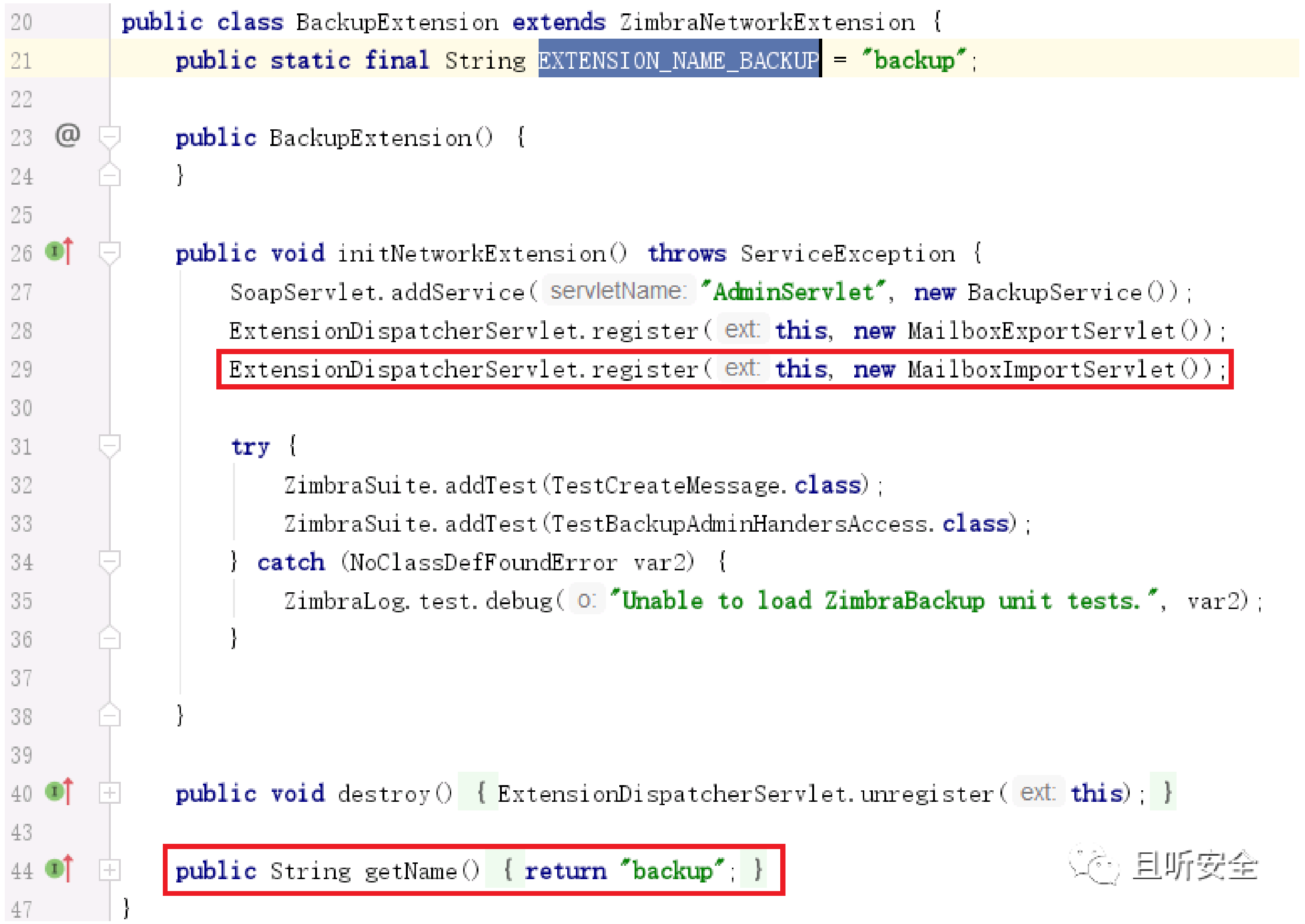
`mExtension` 为 `ZimbraExtension` 类型，并且在 `init` 函数中完成初始化，搜索 `ZimbraExtension` 子类：



```
public interface ZimbraExtension {
    String getName();

    void init() throws E

    void destroy();
}
```

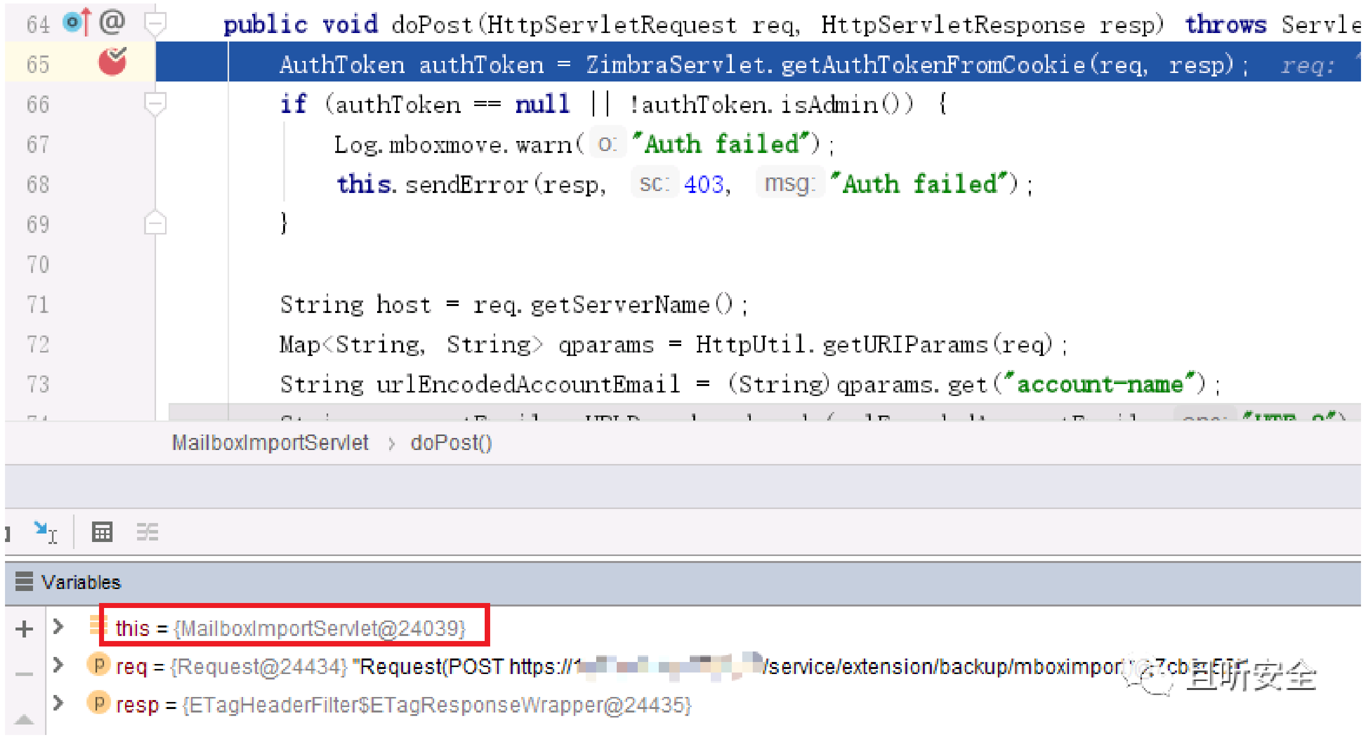
```
Choose Implementation of ZimbraExtension (3
ArchiveInitializer (com.zimbra.cs.archive)
BackupExtension (com.zimbra.cs.backup)
CiscoExtension (com.zimbra.cs.cisco)
ClamScannerExt (com.zimbra.clam)
ConverterExtension (com.zimbra.cs.convert)
FakeZimbraExtension in HttpServiceManager (org.openzal.zal.http)
GQLExtension (com.zimbra.graphql.resources)
HsmInitializer (com.zimbra.cs.hsm)
LicenseExtension (com.zimbra.cs.network.license)
MitelExtension (com.zimbra.cs.mitel)
NetworkInitializer (com.zimbra.cs.network)
NginxLookupExtension (com.zimbra.cs.nginx)
OAuth2Extension (com.zimbra.oauth.resources)
```

定位 `BackupExtension` ，里面刚好注册了 `MailboxImportServlet` 类型：



```
public class BackupExtension extends ZimbraNetworkExtension {
    public static final String EXTENSION_NAME_BACKUP = "backup";

    public BackupExtension() {
    }

    public void initNetworkExtension() throws ServiceException {
        SoapServlet.addService( servletName: "AdminServlet", new BackupService());
        ExtensionDispatcherServlet.register( ext: this, new MailboxExportServlet());
        ExtensionDispatcherServlet.register( ext: this, new MailboxImportServlet());

        try {
            ZimbraSuite.addTest(TestCreateMessage.class);
            ZimbraSuite.addTest(TestBackupAdminHandersAccess.class);
        } catch (NoClassDefFoundError var2) {
            ZimbraLog.test.debug( o: "Unable to load ZimbraBackup unit tests.", var2);
        }

    }

    public void destroy() { ExtensionDispatcherServlet.unregister( ext: this); }

    public String getName() { return "backup"; }
}
```

所以构造特定 URL 将调用 `MailboxImportServlet` ，测试如下：



```
public void doPost(HttpServletRequest req, HttpServletResponse resp) throws Servle
    AuthToken authToken = ZimbraServlet.getAuthTokenFromCookie(req, resp);   req:
    if (authToken == null || !authToken.isAdmin()) {
        Log.mboxmove.warn( o: "Auth failed");
        this.sendError(resp, sc: 403, msg: "Auth failed");
    }

    String host = req.getServerName();
    Map<String, String> qparams = HttpUtil.getURIParams(req);
    String urlEncodedAccountEmail = (String)qparams.get("account-name");
```

MailboxImportServlet › doPost()

Variables

```
this = {MailboxImportServlet@24039}
req = {Request@24434} "Request(POST https://1[redacted]/service/extension/backup/mboximpor
resp = {ETagHeaderFilter$ETagResponseWrapper@24435}
```

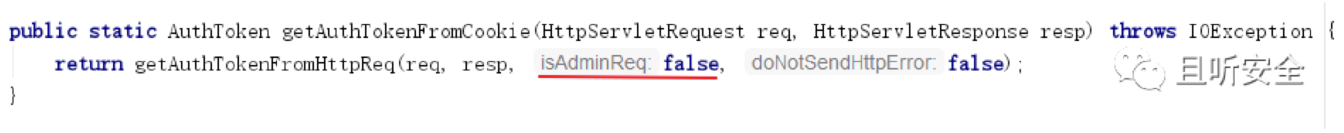
成功进入 `MailboxImportServlet#doPost` 函数处理逻辑。

权限认证分析

下面分析一下 `doPost` 函数的处理逻辑，首先通过 `getAuthTokenFromCookie` 从 Cookie 中提取 token 认证信息，并检查是否为管理员权限：

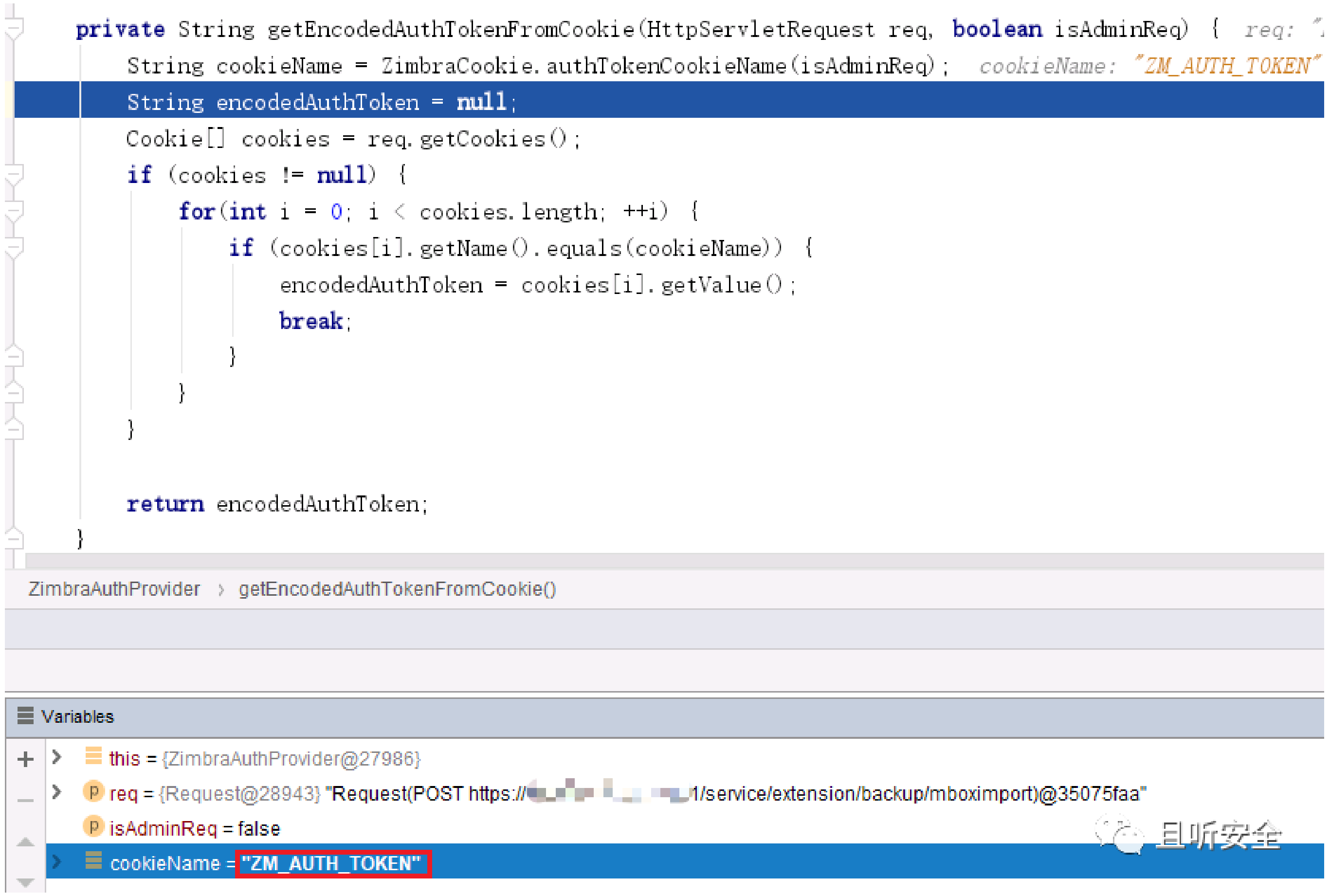
```
AuthToken authToken = ZimbraServlet.getAuthTokenFromCookie(req, resp);
if (authToken == null || !authToken.isAdmin()) {
    Log.mboxmove.warn("Auth failed");
    this.sendError(resp, 403, "Auth failed");
}
```

进入 `getAuthTokenFromCookie`：



```
public static AuthToken getAuthTokenFromCookie(HttpServletRequest req, HttpServletResponse resp) throws IOException {
    return getAuthTokenFromHttpReq(req, resp,  isAdminReq: false,  doNotSendHttpError: false);
}
```

因为这里 `isAdminReq` 默认为 `false`，因此认证后需要携带 `ZM_AUTH_TOKEN` 的 Cookie 值，而非 `ZM_ADMIN_AUTH_TOKEN`：



```
private String getEncodedAuthTokenFromCookie(HttpServletRequest req, boolean isAdminReq) {  req: "
    String cookieName = ZimbraCookie.authTokenCookieName(isAdminReq);  cookieName: "ZM_AUTH_TOKEN"
    String encodedAuthToken = null;
    Cookie[] cookies = req.getCookies();
    if (cookies != null) {
        for(int i = 0; i < cookies.length; ++i) {
            if (cookies[i].getName().equals(cookieName)) {
                encodedAuthToken = cookies[i].getValue();
                break;
            }
        }
    }

    return encodedAuthToken;
}
```

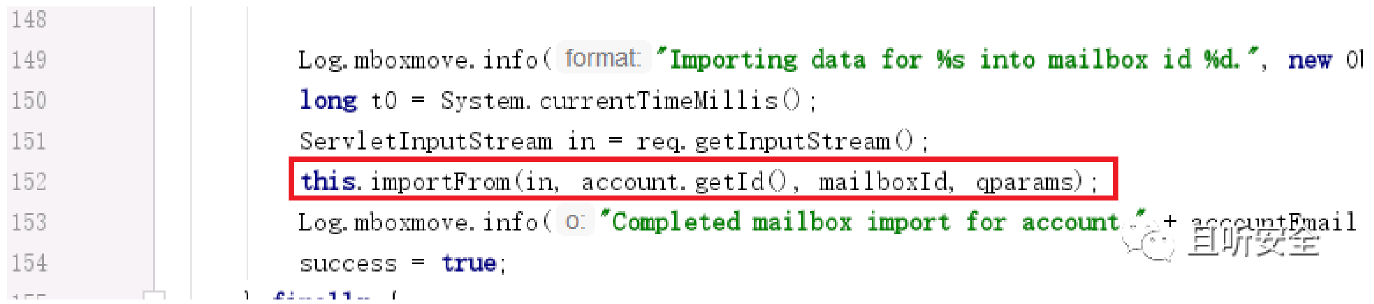
ZimbraAuthProvider › getEncodedAuthTokenFromCookie()

Variables

```
this = {ZimbraAuthProvider@27986}
req = {Request@28943} "Request(POST https://[redacted]1/service/extension/backup/mboximport)@35075faa"
isAdminReq = false
cookieName = "ZM_AUTH_TOKEN"
```

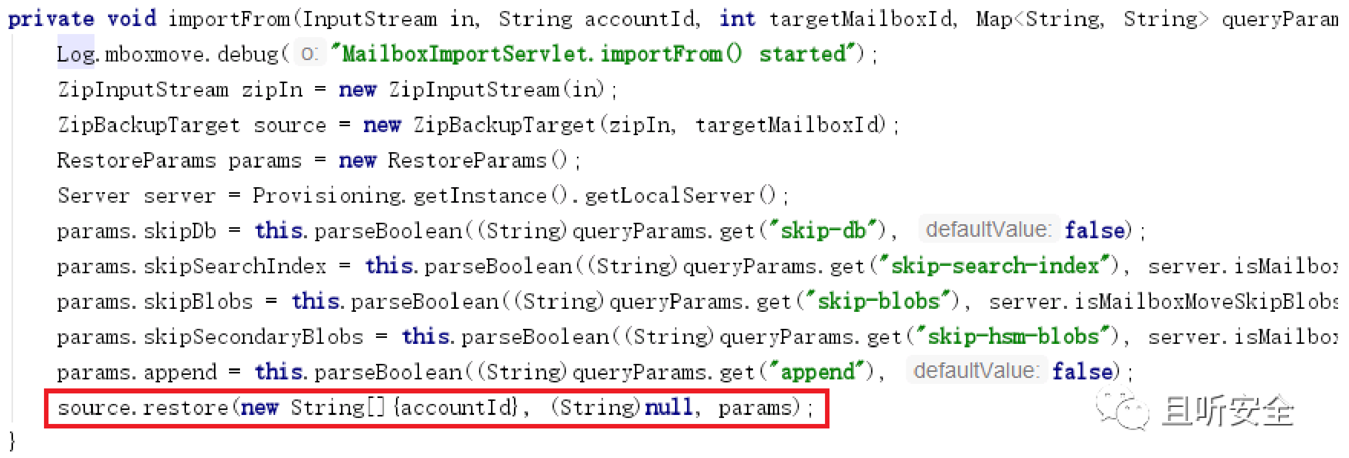
漏洞点定位

通过权限检查后，将会进行一系列参数提取与判断，当提供的 `account-name` 等参数通过验证后，将进入第 152 行 `importFrom` 函数：



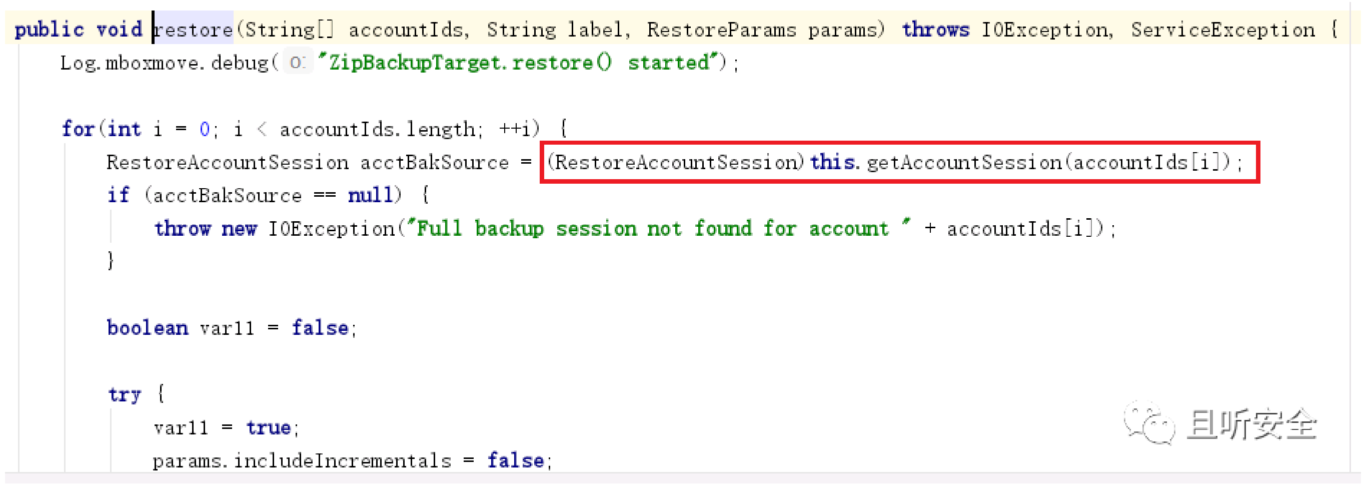
```
            Log.mboxmove.info( format: "Importing data for %s into mailbox id %d.", new O
            long t0 = System.currentTimeMillis();
            ServletInputStream in = req.getInputStream();
            this.importFrom(in, account.getId(), mailboxId, qparams);
            Log.mboxmove.info( o: "Completed mailbox import for account " + accountEmail
            success = true;
        } finally {
```

其中 `in` 来自于 POST 请求数据包，进入 `importFrom` 函数：



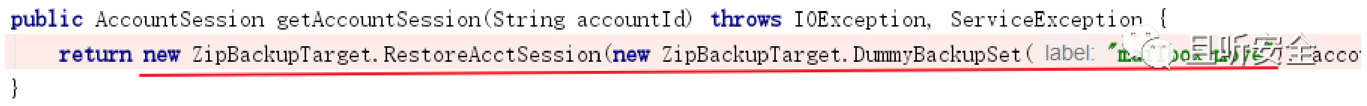
```
private void importFrom(InputStream in, String accountId, int targetMailboxId, Map<String, String> queryParam
    Log.mboxmove.debug( o: "MailboxImportServlet.importFrom() started");
    ZipInputStream zipIn = new ZipInputStream(in);
    ZipBackupTarget source = new ZipBackupTarget(zipIn, targetMailboxId);
    RestoreParams params = new RestoreParams();
    Server server = Provisioning.getInstance().getLocalServer();
    params.skipDb = this.parseBoolean((String)queryParams.get("skip-db"),  defaultValue: false);
    params.skipSearchIndex = this.parseBoolean((String)queryParams.get("skip-search-index"), server.isMailbo
    params.skipBlobs = this.parseBoolean((String)queryParams.get("skip-blobs"), server.isMailboxMoveSkipBlobs
    params.skipSecondaryBlobs = this.parseBoolean((String)queryParams.get("skip-hsm-blobs"), server.isMailbo
    params.append = this.parseBoolean((String)queryParams.get("append"),  defaultValue: false);
    source.restore(new String[]{accountId}, (String)null, params);
}
```

提取 ZIP 压缩包，调用 `restore` 函数：



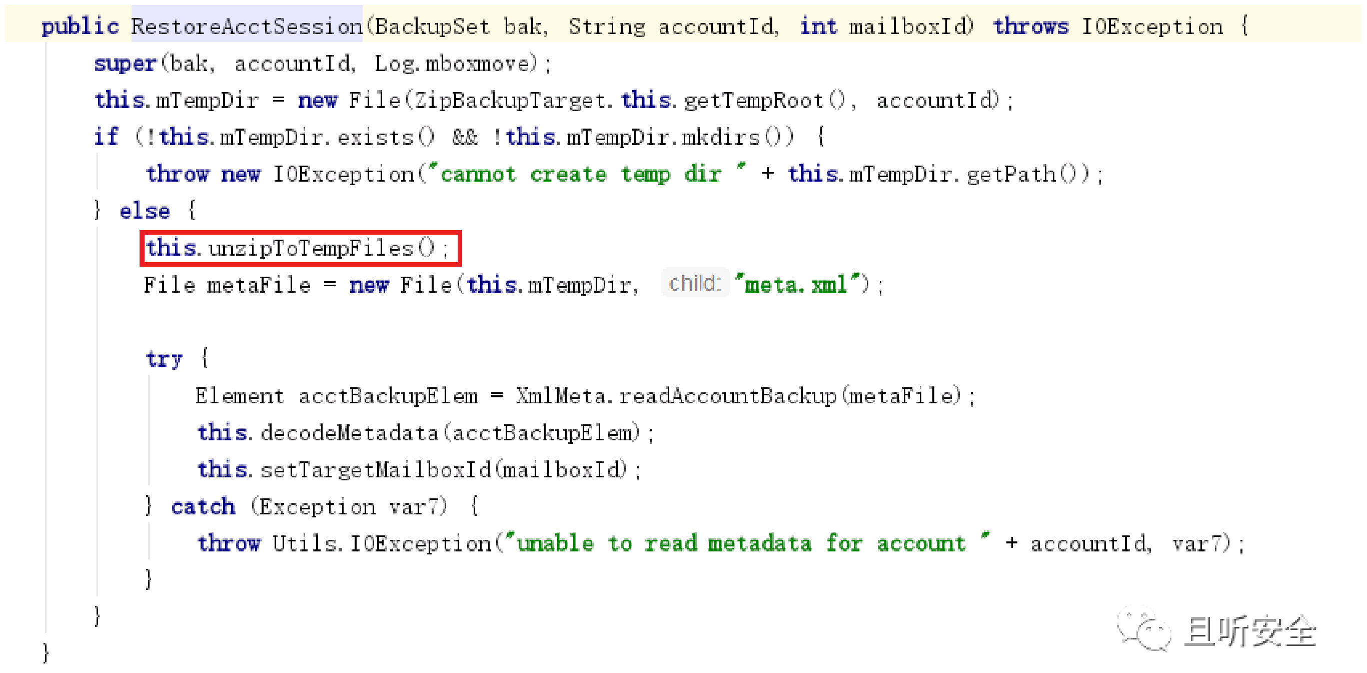
```java
public void restore(String[] accountIds, String label, RestoreParams params) throws IOException, ServiceException {
    Log.mboxmove.debug( 0: "ZipBackupTarget.restore() started");

    for(int i = 0; i < accountIds.length; ++i) {
        RestoreAccountSession acctBakSource = (RestoreAccountSession)this.getAccountSession(accountIds[i]);
        if (acctBakSource == null) {
            throw new IOException("Full backup session not found for account " + accountIds[i]);
        }

        boolean var11 = false;

        try {
            var11 = true;
            params.includeIncrementals = false;
```

进入 `getAccountSession` 函数:

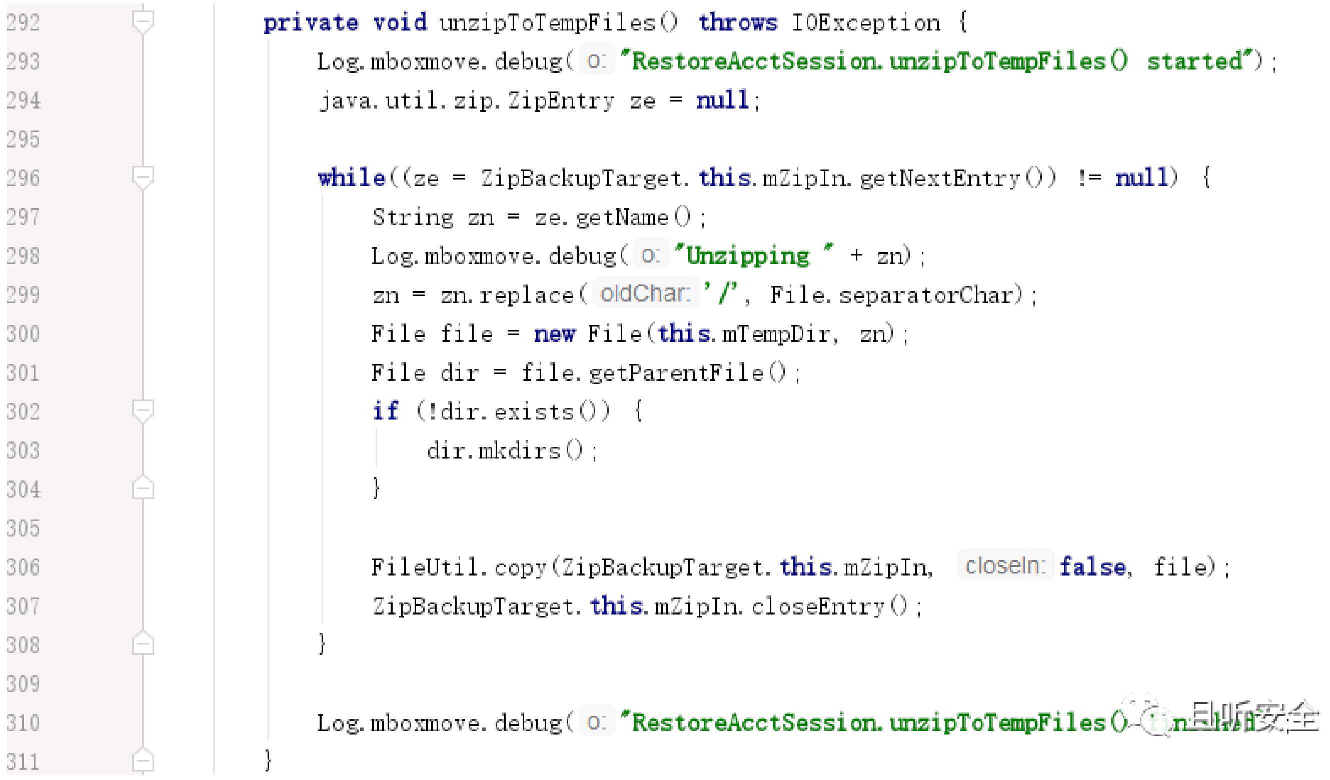


```java
public AccountSession getAccountSession(String accountId) throws IOException, ServiceException {
    return new ZipBackupTarget.RestoreAcctSession(new ZipBackupTarget.DummyBackupSet( label: "mailboxmove"), acco
}
```

实例化 `ZipBackupTarget.RestoreAcctSession` 对象，进入构造函数:


```
public RestoreAcctSession(BackupSet bak, String accountId, int mailboxId) throws IOException {
    super(bak, accountId, Log.mboxmove);
    this.mTempDir = new File(ZipBackupTarget.this.getTempRoot(), accountId);
    if (!this.mTempDir.exists() && !this.mTempDir.mkdirs()) {
        throw new IOException("cannot create temp dir " + this.mTempDir.getPath());
    } else {
        this.unzipToTempFiles();
        File metaFile = new File(this.mTempDir,  child: "meta.xml");

        try {
            Element acctBackupElem = XmlMeta.readAccountBackup(metaFile);
            this.decodeMetadata(acctBackupElem);
            this.setTargetMailboxId(mailboxId);
        } catch (Exception var7) {
            throw Utils.IOException("unable to read metadata for account " + accountId, var7);
        }
    }
}
```

跟进 `unzipToTempFiles` 函数：

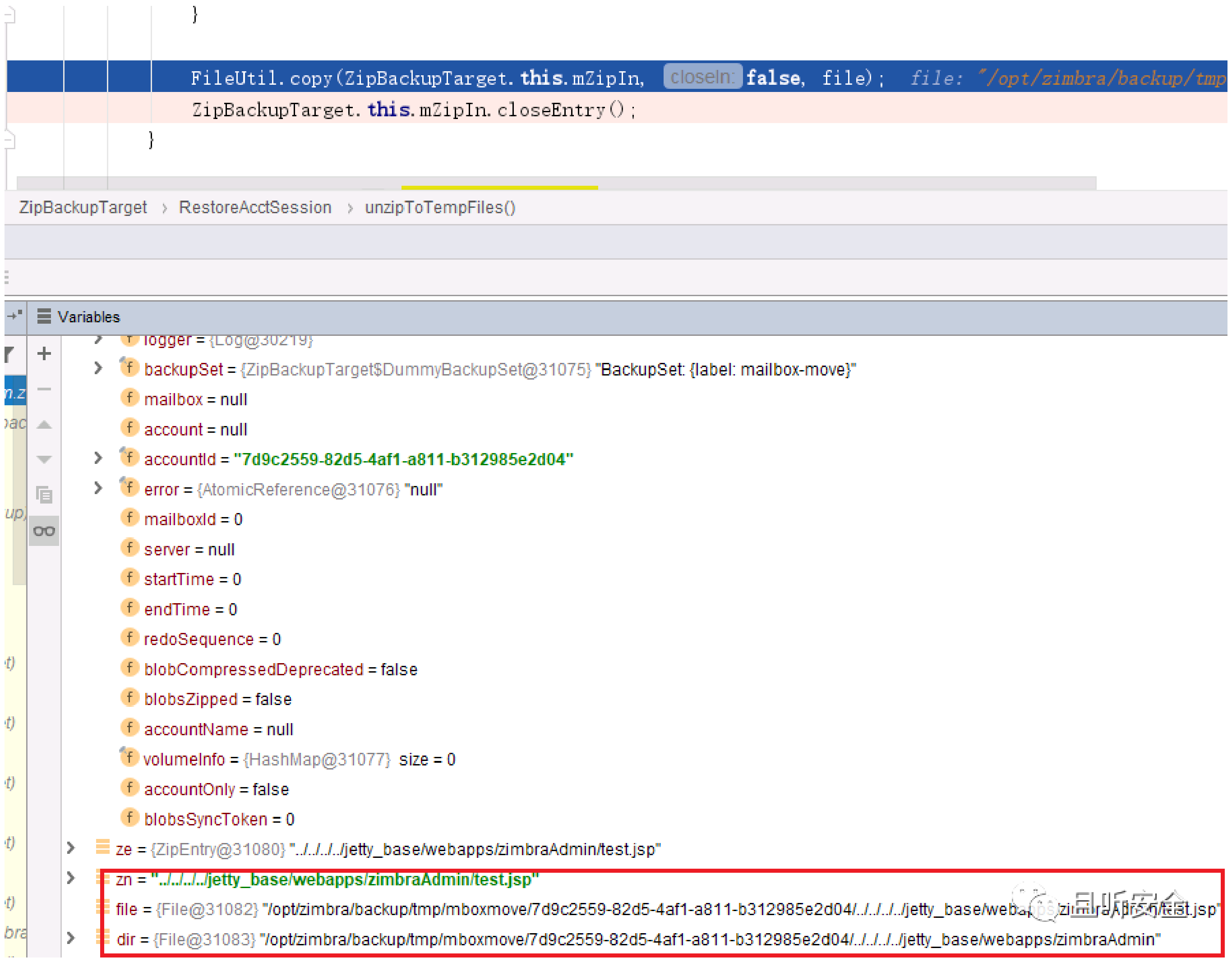


```
private void unzipToTempFiles() throws IOException {
    Log.mboxmove.debug( o: "RestoreAcctSession.unzipToTempFiles() started");
    java.util.zip.ZipEntry ze = null;

    while((ze = ZipBackupTarget.this.mZipIn.getNextEntry()) != null) {
        String zn = ze.getName();
        Log.mboxmove.debug( o: "Unzipping " + zn);
        zn = zn.replace( oldChar: '/', File.separatorChar);
        File file = new File(this.mTempDir, zn);
        File dir = file.getParentFile();
        if (!dir.exists()) {
            dir.mkdirs();
        }

        FileUtil.copy(ZipBackupTarget.this.mZipIn,  closeIn: false, file);
        ZipBackupTarget.this.mZipIn.closeEntry();
    }

    Log.mboxmove.debug( o: "RestoreAcctSession.unzipToTempFiles() fin[illegible]
}
```

ZIP 压缩包解压过程存在路径穿越漏洞，导致可以向任意路径写入 shell 。

漏洞复现

通过上述分析，我们可以构造一个存在路径穿越的 ZIP 压缩包，并发送特定 POST 请求实现压缩包解压路径穿越：



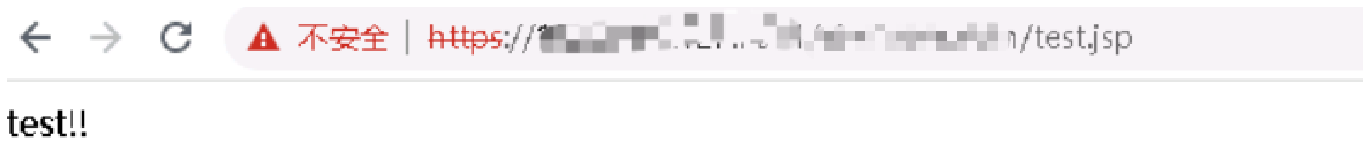
```
            }
        FileUtil.copy(ZipBackupTarget.this.mZipIn, closeIn: false, file);   file: "/opt/zimbra/backup/tmp
        ZipBackupTarget.this.mZipIn.closeEntry();
    }
```

ZipBackupTarget › RestoreAcctSession › unzipToTempFiles()

Variables

```
  logger = {Log@30219}
> backupSet = {ZipBackupTarget$DummyBackupSet@31075} "BackupSet: {label: mailbox-move}"
  mailbox = null
  account = null
> accountId = "7d9c2559-82d5-4af1-a811-b312985e2d04"
> error = {AtomicReference@31076} "null"
  mailboxId = 0
  server = null
  startTime = 0
  endTime = 0
  redoSequence = 0
  blobCompressedDeprecated = false
  blobsZipped = false
  accountName = null
  volumeInfo = {HashMap@31077}  size = 0
  accountOnly = false
  blobsSyncToken = 0
> ze = {ZipEntry@31080} "../../../../jetty_base/webapps/zimbraAdmin/test.jsp"
> zn = "../../../../jetty_base/webapps/zimbraAdmin/test.jsp"
  file = {File@31082} "/opt/zimbra/backup/tmp/mboxmove/7d9c2559-82d5-4af1-a811-b312985e2d04/../../../../jetty_base/webapps/zimbraAdmin/test.jsp"
> dir = {File@31083} "/opt/zimbra/backup/tmp/mboxmove/7d9c2559-82d5-4af1-a811-b312985e2d04/../../../../jetty_base/webapps/zimbraAdmin"
```

最终写入 shell ：


不安全 | ~~https~~://[redacted]/test.jsp

test!!