Jenkins漏洞分析 CVE-2018-1999001

angel010 / 2018-12-23 09:00:00 / 浏览数 3582 技术文章 翻译文章 顶(0) 踩(0)

CyberArk实验室研究人员发现了开源自动化服务器Jenkins的一些漏洞。本文介绍其中两个漏洞。

CVE-2018-1999001

CVE-2018-1999001的CVSS评分是3.0，属于高危漏洞。

当非授权的攻击者从Jenkins主文件系统移除文件时会导致Jenkins修改其主安全开关，有时甚至会完全放弃所有的安全防护，包括防数据泄露的安全措施，并允许非授权的用

工作原理

不管是Jenkins的用户数据库还是其他用户数据库，每当Jenkins有新用户产生，都会为每个用户名创建一个新的目录和config.xml文件，config.xml文件中的内容就是与token、用户邮件地址和全名。加密的API token是随机生成的token，用于使脚本客户端模仿新用户并调用需要认证的操作。

对每个新用户，Jenkins都会在硬盘上预留空间来保存新用户的详细信息。database有一点误导性，因为Jenkins会把新用户的信息保存在名为users的特殊目录中：

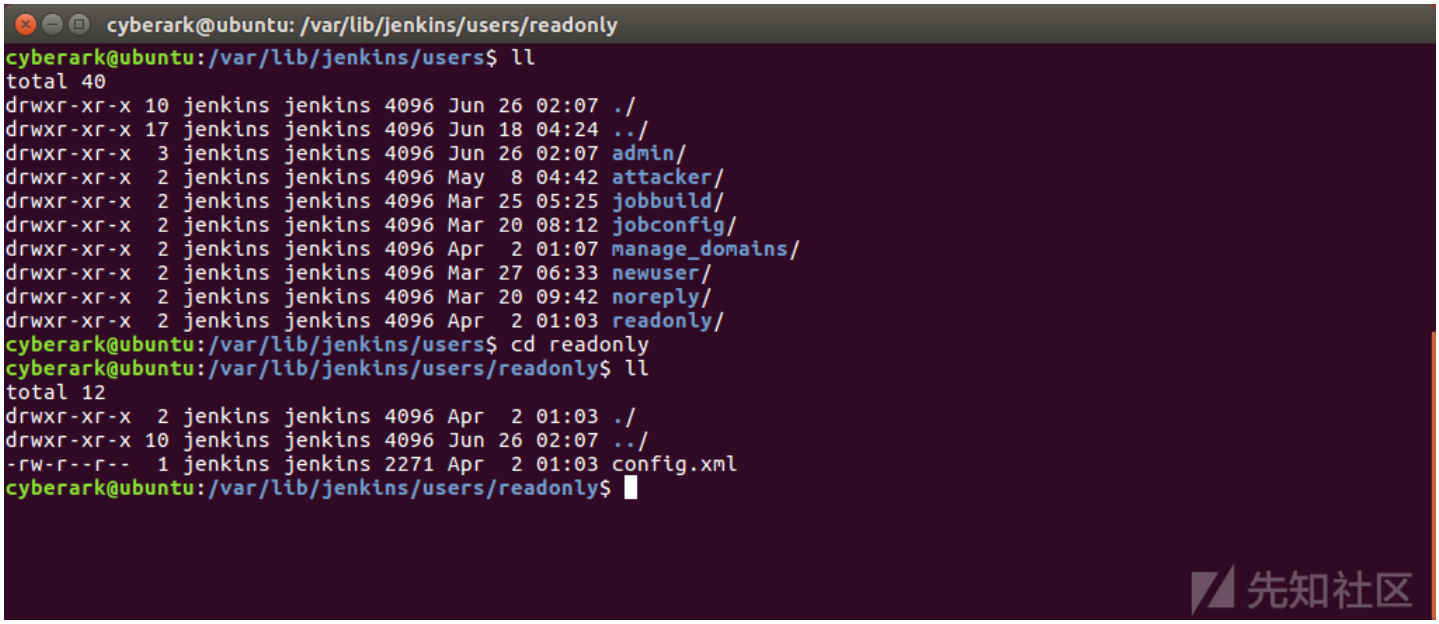


图1: Jenkins用户目录

每个用户名在JENKINS_HOME/users/目录下都有一个对应的文件夹，里面有一个config.xml文件。如果Jenkins被设置为使用自己用户的数据库，那么config.xml文件token、全名等信息。

Jenkins主要安全防护

因为该漏洞主要是滥用Jenkins认证机制来绕过 Jenkins的安全防护，最终使任意用户都可以admin访问。因此，本节介绍Jenkins的主要安全防护措施。

Jenkins Global Security Configuration（全局安全配置）页打开了一个enable security选择框。从Jenkins 2.0开始，该选择框默认是选择的：

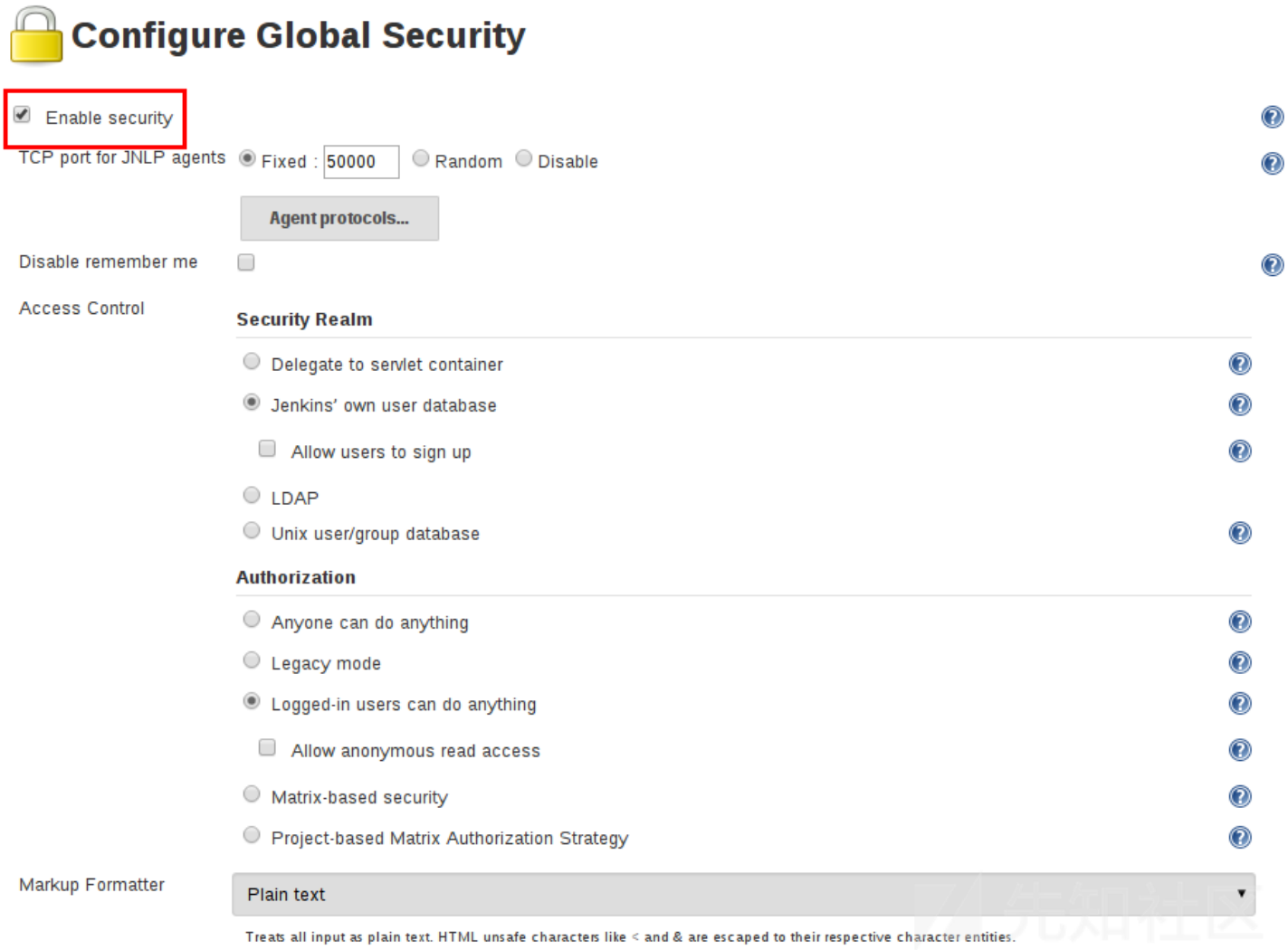


图2: Jenkins Configure Global Security页

Jenkins的`enable security`选择框web UI允许管理员开启、配置、关闭Jenkins所有的安全特征，包括用户认证和授权在内的特征。不选择`enable security`，管理员就可以让所有匿名和非认证的用户访问Jenkins。
Jenkins master的`enable security`状态保存在`JENKINS_HOME`目录的`Jenkins master`配置文件中。配置文件也叫做`config.xml`。

移除主config.xml文件

每个Jenkins用户都可以用密码进行认证，使用API token会使认证过程有一点复杂，因为需要双重认证。
双重认证机制是在`Hudson.Security.BasicAuthenticationFilter.java`中实现的。

为什么是双重认证机制呢？Jenkins支持两种类型的认证：HTTP基本认证和基于form的认证。HTTP基本认证是为脚本客户端预留的，而基于form的认证是为通过web UI以用户名和密码登陆的用户准备的。

使用HTTP基本认证的话，客户端会用用户名和密码或API token组合来认证Jenkins master。在基本认证中，客户端的用户名和密码/API token组合应该是连接在一起、base64编码的，并且在认证的HTTP header中传递：
`Authorization: Basic dm9yZGVsOnZvcmRlbA==`

Jenkins master会用用户数据库来认证用户，比如验证`username:password`的组合，或与保存在用户`config.xml`文件中的本地用户的API token进行比较，即`username:APItoken`认证。

如果`HTTP GET header`是`Authorization: Basic`，就会调用`Hudson.Security.BasicAuthenticationFiltermodule`中的`doFilter`函数，见代码段1第3行。该函数会提取出代码段1第6行的`Authorization header`，并调用第29行来解码base 64信息。

```
String uidpassword = Scrambler.descramble(authorization.substring(6));
```

`uidpassword`中的字符串现在含有解码的base64字符串，格式如`username:password`。
然后填充用户名和密码变量（代码段1，第32和33行）
然后，Jenkins调用第45行的`getById`函数：

```
User u = User.getById(username, true);
```

函数的调用有2个参数：`username`是从`authorization HTTP header`提取的，`true`随后进行介绍。

```
public class BasicAuthenticationFilter implements Filter {
   private ServletContext servletContext;
   public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletExcept
       HttpServletRequest req = (HttpServletRequest) request;
       HttpServletResponse rsp = (HttpServletResponse) response;
       String authorization = req.getHeader("Authorization");

       String path = req.getServletPath();
       if(authorization==null || req.getUserPrincipal() !=null || path.startsWith("/secured/")
       || !Jenkins.getInstance().isUseSecurity()) {
           // normal requests, or security not enabled
           if(req.getUserPrincipal()!=null) {
               // before we route this request, integrate the container authentication
               // to Acegi. For anonymous users that doesn't have user principal,
               // AnonymousProcessingFilter that follows this should create
               // an Authentication object.
               SecurityContextHolder.getContext().setAuthentication(new ContainerAuthentication(req));
           }
           try {
               chain.doFilter(request,response);
           } finally {
               SecurityContextHolder.clearContext();
           }
           return;
       }
       // authenticate the user
       String username = null;
       String password = null;
       String uidpassword = Scrambler.descramble(authorization.substring(6));
       int idx = uidpassword.indexOf(':');
       if (idx >= 0) {
           username = uidpassword.substring(0, idx);
           password = uidpassword.substring(idx+1);
       }

       if(username==null) {
           rsp.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
           rsp.setHeader("WWW-Authenticate","Basic realm=\"Jenkins user\"");
           return;
       }

       {// attempt to authenticate as API token
           // create is true as the user may not have been saved and the default api token may be in use.
           // validation of the user will be performed against the underlying realm in impersonate.
           User u = User.getById(username, true);
           ApiTokenProperty t = u.getProperty(ApiTokenProperty.class);
           if (t!=null && t.matchesPassword(password)) {
               UserDetails userDetails = u.getUserDetailsForImpersonation();
               Authentication auth = u.impersonate(userDetails);

               SecurityListener.fireAuthenticated(userDetails);

               SecurityContextHolder.getContext().setAuthentication(auth);
               try {
                   chain.doFilter(request,response);
               } finally {
                   SecurityContextHolder.clearContext();
               }
               return;
           }
       }


       path = req.getContextPath()+"/secured"+path;
       String q = req.getQueryString();
       if(q!=null)
           path += '?'+q;

       // prepare a redirect
```

```
        rsp.setStatus(HttpServletResponse.SC_MOVED_TEMPORARILY);
        rsp.setHeader("Location",path);

        // ... but first let the container authenticate this request
        RequestDispatcher d = servletContext.getRequestDispatcher("/j_security_check?j_username="+
            URLEncoder.encode(username,"UTF-8")+"&j_password="+URLEncoder.encode(password,"UTF-8"));
        d.include(req,rsp);
    }
```

代码段1: Hudson.Security.BasicAuthenticationFilter.java

分析Hudson.model.User模块，发现Hudson.model.User中的GetById函数调用了另一个函数：

```
public static @Nullable User getById(String id, boolean create) {
            return getOrCreate(id, id, create);
        }
```

代码段2: Hudson.model.User.java

注意前面的true参数这里叫做create。
User.java模块中的getOrCreate会调用另一个getOrCreate函数，但增加了另一个getUnsanitizedLegacyConfigFileFor返回的参数：

```
… User getOrCreate(@Nonnull String id, @Nonnull String fullName, boolean create) {
     return getOrCreate(id, fullName, create, getUnsanitizedLegacyConfigFileFor(id));
}
```

代码段3: Hudson.model.User.java

看起来有一些Jenkins遗留文件系统被用来填充用户的数据库。进一步分析发现两个与遗留数据库相关的兄弟函数：

```
private static final File getConfigFileFor(String id) {
        return new File(getRootDir(), idStrategy().filenameOf(id) +"/config.xml");
    }

    private static File getUnsanitizedLegacyConfigFileFor(String id) {
        return new File(getRootDir(), idStrategy().legacyFilenameOf(id) + "/config.xml");
    }
```

代码段4: Hudson.model.User.java

第一个函数：getConfigFileFor(id)，获取Jenkins的用户数据库目录(JENKINS_HOME/users)，加入一些形式的username
(id)，返回含有JENKINS_HOME/users/{username}/config.xml
的文件名。

第二个函数与第一个函数作用相同，但是从idStrategy().legacyFilenameof()中获取用户名。
legacyFilenameof()函数并不会修改用户名字符串，而getConfigFileFor()使用的filenameOf()函数会将用户名字符串进行处理以防止用户名被作为目录名或其他

然后，Jenkins代码会调用getOrCreate()函数。

```
private static @Nullable User getOrCreate(@Nonnull String id, @Nonnull String fullName, boolean create, File unsanitizedLegacy

        String idkey = idStrategy().keyFor(id);

        byNameLock.readLock().lock();
        User u;
        try {
            u = AllUsers.byName().get(idkey);
        } finally {
            byNameLock.readLock().unlock();
        }

        final File configFile = getConfigFileFor(id);
        if (unsanitizedLegacyConfigFile.exists() && !unsanitizedLegacyConfigFile.equals(configFile)) {
            File ancestor = unsanitizedLegacyConfigFile.getParentFile();
            if (!configFile.exists()) {
                try {
                    Files.createDirectory(configFile.getParentFile().toPath());
                    Files.move(unsanitizedLegacyConfigFile.toPath(), configFile.toPath());
                    LOGGER.log(Level.INFO, "Migrated user record from {0} to {1}", new Object[] {unsanitizedLegacyConfigFile, c
                } catch (IOException | InvalidPathException e) {
                    LOGGER.log(
```

```
                    Level.WARNING,
                    String.format("Failed to migrate user record from %s to %s", unsanitizedLegacyConfigFile, configFil
                    e);
            }
        }

        // Don't clean up ancestors with other children; the directories should be cleaned up when the last child
        // is migrated
        File tmp = ancestor;
        try {
            while (!ancestor.equals(getRootDir())) {
                try (DirectoryStream<Path> stream = Files.newDirectoryStream(ancestor.toPath())) {
                    if (!stream.iterator().hasNext()) {
                        tmp = ancestor;
                        ancestor = tmp.getParentFile();
                        Files.deleteIfExists(tmp.toPath());
                    } else {
                        break;
                    }
                }
            }
        } catch (IOException | InvalidPathException e) {
            if (LOGGER.isLoggable(Level.FINE)) {
                LOGGER.log(Level.FINE, "Could not delete " + tmp + " when cleaning up legacy user directories", e);
            }
```

代码段5: Hudson.model.User.java

函数会将字符串unsanitizedLegacyConfigFile作为一个参数，调用getConfigFileFor(id)来计算处理过的用户名目录。处理是指从用户名中移除恶意数据来预防

如第14-19行代码，如果：
（1）有未处理过的遗留文件存在；
（2）遗留的配置文件与处理过的配置文件不相同；
（3）硬盘中不存在处理过的配置文件

Jenkins就会为用户创建一个新的处理过的目录。未处理过的遗留配置文件就会被移动到新创建的目录，代码段5的第18行就创建了一个新目录，第19行将未处理的配置文件

如何利用该逻辑？

含有Jenkins 安全网关的JENKINS_HOME目录中有一个config.xml文件，可以尝试检查该代码，查看是否可以让Jenkins从硬盘中移除主配置文件。

首先查看代码流图，然后进行分析：

用用户名..来检查流图。需要准备一个含有用户名..和任意密码的格式为username:password的base64字符串：

```
“..:ANYPASSWORD” = Li46QU5ZUEFTU1dPUkQ=”
```

然后用CURL发送:

```
Curl JenkinsURL –H “Authorization: Basic Li46QU5ZUEFTU1dPUkQ=”
```

Jenkins代码就会计算未处理的configFile：
/JENKINS_HOME/users/../config.xml
但是这等价于
/JENKINS_HOME/config.xml
该文件是存在的，因为它是Jenkins的主配置文件，因此通过了test #1。

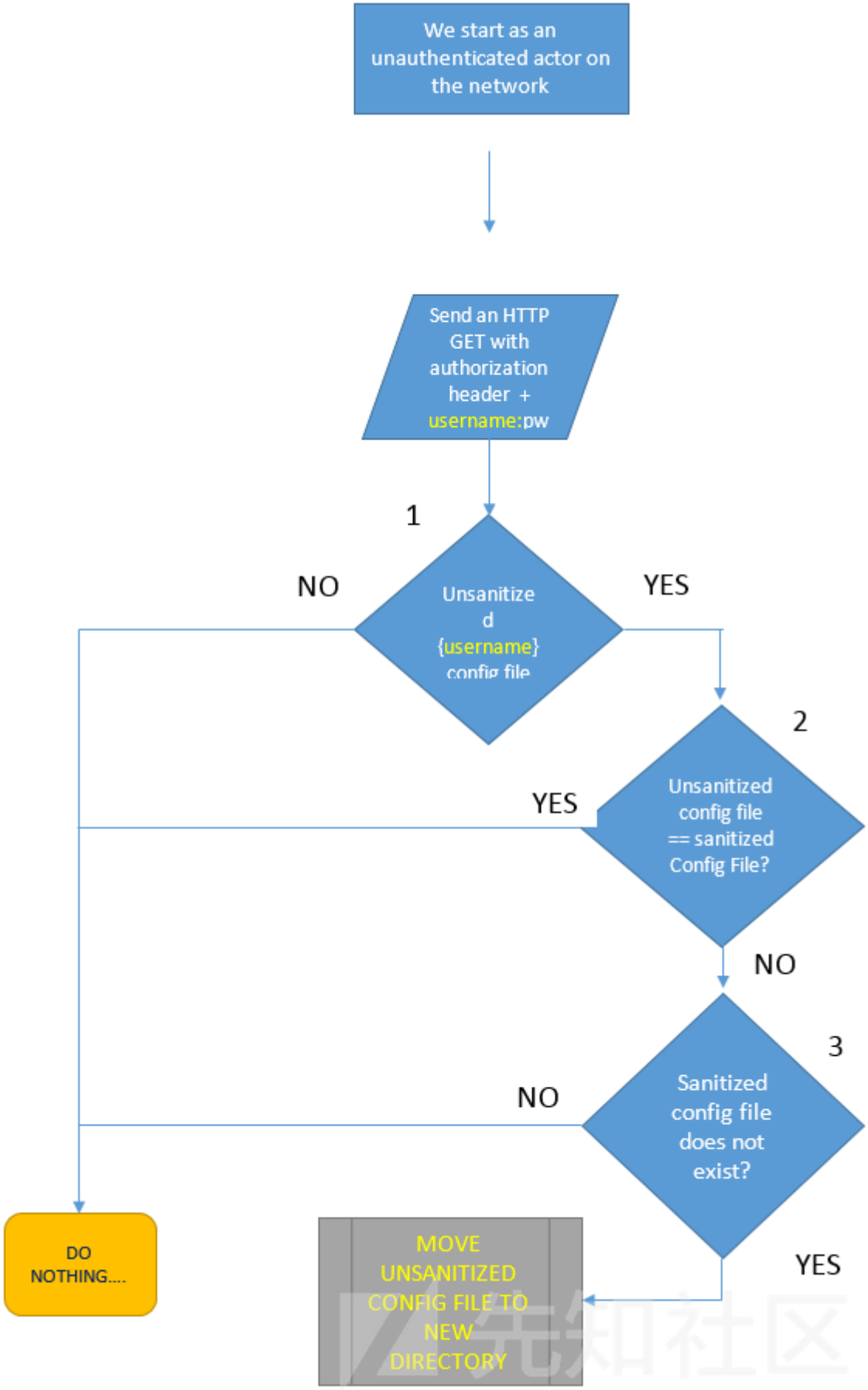
处理过的Jenkins不允许用户名中出现..，因为存在路径遍历问题。所以Jenkins将每个..用字符$002e来替代了。因此处理过的配置文件被计算为：
/JENKINS_HOME/users/$002e$002e/config.xml

很明显，处理过的配置文件与未处理过的配置文件是不同的，因此通过了test #2。

第3个测试是检查处理过的文件是否存在，研究任意认为没有Jenkins安装过程中会用$002e$002e这样的用户名，因此通过了test #3。

Jenkins的含有关于主安全网关描述的主配置文件config.xml已经从JENKINS_HOME目录中移除了。

代码段1表明Jenkins在认证用户，比如比较API token。因为研究人员提供的用户名与Jenkins用户数据库中的用户名不匹配，Curl GET
HTTP命令就会一直报错。可以看出，Jenkins主配置文件已经从Jenkins主目录中移除了。



代码段5流图

主配置文件移除之后呢？

Jenkins
master软件是在Java虚拟机缓存内存中运行的。也就是说当Jenkins服务器运行时，所有的文件都会读取到Java缓存中。当其中有文件被移除后，并不会马上改变Jenkins的

因此，攻击者只需要等Jenkins服务器重启就可以了。

CVE-2018-1999043: 使Java虚拟机奔溃

代码段2中的`User.java`模块函数`User.getById()`是用相同的用户名和参数■`true`■来调用的。分析发现该参数是`create`，看起来可以在不需要经过Jenkins认证的情况
Jenkins Java cache中创建新用户。

每个用户在Java虚拟机内存中都有一定数量的空间，如果用户名比较长，那么该空间的数也会增长。因此，使用含有长用户名的Curl
命令就可以让Java虚拟机因为可用内存太少而本科，迫使Jenkins管理员重启Jenkins服务器。

总结

据上所述，首先将含有重要Jenkins 安全配置的`config.xml`文件从`JENKINS_HOME`移动到新目录，然后通过过长的用户名来使Java
虚拟机奔溃，然后等Jenkins重启后就运行在`Security Disabled`模式了。该模式下不需要认证，任何访问Jenkins master的都拥有管理员权限。

通过从`JENKINS_HOME/users/$002e$002e/`目录读取`config.xml`文件，并对Jenkins文件系统中的文件做必要的修改，攻击者可以将之前移动的`config.xml`复制回原

这样，Jenkins master就会恢复之前的安全配置了，没有人会注意到Jenkins master转变为原来的security disabled的情况。因此攻击活动也不会被发现。

https://www.cyberark.com/threat-research-blog/tripping-the-jenkins-main-security-circuit-breaker-an-inside-look-at-two-jenkins-security-vulnerabilities/