

## 0x01 前言

首先Java下的命令执行大家都知道常见的两种方式：

### 1.使用ProcessBuilder

```
ProcessBuilder pb=new ProcessBuilder(cmd);
pb.start();
```

### 2.使用Runtime

```
Runtime.getRuntime().exec(cmd)
```

也就是说上面cmd参数可控的情况下，均存在命令执行的问题。但是话题回来，不太清楚大家是否遇到过java命令执行的时候，无论是windows还是linux环境下，带有|,<

## 0x02 差别

先选择跟进Runtime.getRuntime().exec(cmd)，样例代码如下所示：

```
import java.io.*;

public class Main {
    public static void main(String[] arg) throws IOException {
        String command="/bin/sh -c echo 111 > 3.txt";
        Process proc = Runtime.getRuntime().exec(command);
        InputStream in = proc.getInputStream();
        BufferedReader br = new BufferedReader(new InputStreamReader(in, "UTF8"));
        String line = null;
        while((line=br.readLine())!=null) {
            System.out.println(line);
        }
    }
}
```

跟进 java.lang.Runtime#exec 的构造方法，下面话题回来，exec的构造方法有以下几种情况，其实根据传入的变量我们大概可以区分的了，一个是根据String command，也就是直接传入一个字符串。另一个是根据String cmdarray[]，也就是传入一个数组。

```
public Process exec(String command) throws IOException {
    return exec(command, null, null);
}

public Process exec(String command, String[] envp) throws IOException {
    return exec(command, envp, null);
}

public Process exec(String cmdarray[]) throws IOException {
    return exec(cmdarray, null, null);
}

public Process exec(String[] cmdarray, String[] envp) throws IOException {
    return exec(cmdarray, envp, null);
}
```

而根据前面代码中，我们传入的命令是如下所示：

```
String command="/bin/sh -c echo 111 > 3.txt";
```

所以会进入Process exec(String command)这个构造方法进行处理，跟进这个方法，发现最后返回exec(command, null, null)。

```
public Process exec(String command) throws IOException {
    return exec(command, null, null);
}
```

继续跟进这个exec方法，看看这个方法的实现。这里代码实例化了StringTokenizer类，并且传入了我们要执行的command命令，简单翻译一下注释：为指定的字符串构造

```

223
224 /**
225  * Constructs a string tokenizer for the specified string. The
226  * tokenizer uses the default delimiter set, which is
227  * <code>"&nbsp;&#92;t&#92;n&#92;r&#92;f"</code>: the space character,
228  * the tab character, the newline character, the carriage-return character,
229  * and the form-feed character. Delimiter characters themselves will
230  * not be treated as tokens.
231  *
232  * @param str a string to be parsed.
233  * @exception NullPointerException if str is <CODE>>null</CODE>
234  */
235 @ public StringTokenizer(String str) { str: "/bin/sh -c echo 111 > 3.txt"
236     this(str, delim: " \t\n\r\f", returnDelims: false; str: "/bin/sh -c echo 111 > 3.txt"
237 }
238
239 /**

```

我们继续往下看，经过StringTokenizer类处理之后会返回一个cmdarray[]，而这里的处理实际上是根据空格针对命令进行了分割，至于为什么结果要是一个array数组，我们

```

441 @ public Process exec(String command, String[] envp, File dir) command: "/bin/sh -c echo 111 > 3.txt" envp: null dir: null
442     throws IOException {
443         if (command.length() == 0)
444             throw new IllegalArgumentException("Empty command");
445
446         StringTokenizer st = new StringTokenizer(command); st (slot_4): StringTokenizer@482 command: "/bin/sh -c echo 111 > 3.txt"
447         String[] cmdarray = new String[st.countTokens()]; cmdarray (slot_5): {" /bin/sh", "-c", "echo", "111", ">", + 1 more}
448         for (int i = 0; st.hasMoreTokens(); i++)
449             cmdarray[i] = st.nextToken(); st (slot_4): StringTokenizer@482
450         return exec(cmdarray, envp, dir); cmdarray (slot_5): {" /bin/sh", "-c", "echo", "111", ">", + 1 more} envp: null dir: null
451

```

Runtime > exec()

Variables

- this = {Runtime@481}
- Variables debug info not available
- command = "/bin/sh -c echo 111 > 3.txt"
- envp = null
- dir = null
- st (slot\_4) = {StringTokenizer@482}
- cmdarray (slot\_5) = {String[6]@483}
  - 0 = "/bin/sh"
  - 1 = "-c"
  - 2 = "echo"
  - 3 = "111"
  - 4 = ">"
  - 5 = "3.txt"
- slot\_6 = 6

我们发现经过一系列的处理，最后又有一个return exec

的处理。继续跟进这个exec的处理，我们可以看到这里最后实例化ProcessBuilder来处理我们传入的cmdarray。到这里实际上可以清楚了Runtime.getRuntime().exec()的

```

public Process exec(String[] cmdarray, String[] envp, File dir)
    throws IOException {
    return new ProcessBuilder(cmdarray)
        .environment(envp)
        .directory(dir)
        .start();
}

```

我们知道ProcessBuilder.start方法是命令执行，那么跟进这个start我们发现，首先prog获取cmdarray[0]也就是我们的/bin/sh，然后判断security是否为null，如果不为null

```

1010 for (String arg : cmdarray)
1011     if (arg == null)
1012         throw new NullPointerException();
1013 // Throws IndexOutOfBoundsException if command is empty
1014 String prog = cmdarray[0]; cmdarray (slot_1): {" /bin/sh", "-c", "echo", "111", ">", + 1 more}
1015
1016 SecurityManager security = System.getSecurityManager();
1017 if (security != null)
1018     security.checkExec(prog);
1019

```

然后继续往下走，这里调用java.lang.ProcessImpl.start。

```
try {
    return ProcessImpl.start(cmdarray,
        env,
        dir,
        red,
        red);
} catch (IOException | IllegalArgumentException e) {
    String exceptionInfo = ":";
    Throwable cause = e;
    if ((e instanceof IOException) && !e.getMessage().contains("Can not disclose the")) {
        try {
            security.checkRead(p);
        } catch (SecurityException se) {
            exceptionInfo = "";
            cause = se;
        }
    }
}
```

cmdarray

cmdarray = {String[6]@492}

- 0 = "/bin/sh"
- 1 = "-c"
- 2 = "echo"
- 3 = "111"
- 4 = ">"
- 5 = "3.txt"

进入之后我们就可以看到最后是调用java.lang.UnixProcess这个类来执行命令，而且我们发现执行命令的时候实际上是根据cmdarray[0]来判断用什么命令。而在java.lang.

```
501 2653 2510 0 9:54AM ?? 0:00.00 (sh)
501 3044 2881 0 9:55AM ttys001 0:00.01 grep --color=auto --exclude-dir
=.bzip2 --exclude-dir=CVS --exclude-dir=.git --exclude-dir=.hg --exclude-dir=.svn
2653
```

这样看可能还不够明显，因为我们知道/bin/sh -c echo 111 > 3.txt在bash命令行下也不会正常执行成功，命令行下需要/bin/sh -c "echo 111 > 3.txt"，看这两段代码的命令执行的效果。

```
String[] command = { "/bin/sh", "-c", "echo 111 > 3.txt" };
String command="/bin/sh -c \"echo 111 > 3.txt\"";
```

首先先看String command="/bin/sh -c \"echo 111 > 3.txt\"";，按照前面的分析，经过StringTokenizer这个类进行拆分之后变成了{"/bin/sh", "-c", "echo", "111", ">", "3.txt"}。

```
public Process exec(String[] cmdarray, String[] envp, File dir) throws IOException {
    return new ProcessBuilder(cmdarray)
        .environment(envp)
        .directory(dir)
        .start();
}
```

cmdarray

cmdarray = {String[6]@480}

- 0 = "/bin/sh"
- 1 = "-c"
- 2 = "echo"
- 3 = "111"
- 4 = ">"
- 5 = "3.txt"

而当前内存开辟一个12473进程，并且确实12473执行sh命令。

```
pid = forkAndExec(mode: LaunchMechanism.ordinal() + 1, pid: 12473,
    helperpath,
    prog, prog: {47, 98, 105, 110, 47, 115, 104, 0}
    argBlock, argc, argBlock: {45, 99, 0, 34, 101, 99, 104, 111, 0, 49, + 12 more} argc: 5
    envBlock, envc, envBlock: null envc: 0
    dir, dir: null
    fds, fds: {28, 29, 31}
    redirectErrorStream); redirectErrorStream: false
```

```
ps -ef | grep 12473
501 12473 9917 0 10:16AM ?? 0:00.00 (sh)
501 12550 2881 0 10:16AM ttys001 0:00.00 grep --color=auto --exclude-dir
=.bzip2 --exclude-dir=CVS --exclude-dir=.git --exclude-dir=.hg --exclude-dir=.svn
12473
```

但是我们发现，经过StringTokenizer这类拆分之后，命令完全变了一个味道，语义完全变了，并不是我们想要的结果，那我们再看看String[] command = {"/bin/sh", "-c", "echo 111 > 3.txt"}的结果。因为我们传入的是array数组类型，这里直接将命令直接带入了ProcessBuilder进行处理，前面完全没有经过StringTokenizer这个类的拆分。也就是他完整的保

```

484 @ public Process exec(@Nonnull String cmdarray[]) throws IOException {
485     return exec(cmdarray, envp: null, dir: null); cmdarray: {"/bin/sh", "-c", "echo 111 > 1.t...}
486 }
487
614 */
615 @ public Process exec(String[] cmdarray, String[] envp, File dir) cmdarray: {"/bin/sh", "-c", "echo 111 > 1.t..
616     throws IOException {
617     return new ProcessBuilder(cmdarray) cmdarray: {"/bin/sh", "-c", "echo 111 > 1.t...}
618         .environment(envp)
619         .directory(dir)
620         .start();
621 }
622

```

也就是说`getRuntime().exec()`如果直接传入字符串会经过`StringTokenizer`的分割，进而破坏其原本想要表达的意思。

下面这段代码是否存在命令执行的问题，要是在PHP下，我会斩钉截铁的说是，但是回到java环境下，我们发现|等一些特殊符号没办法使用，而且传入的是字符串，遇到空

```

String str = request.getParameter("url");
String cmdstr = "ping " + url;
Runtime.getRuntime().exec(cmdstr)

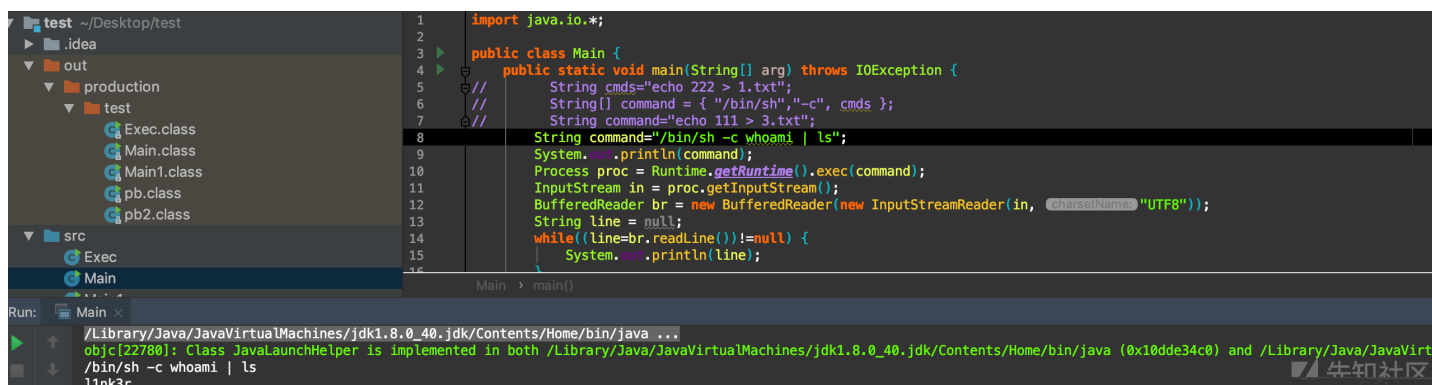
```

再来一段代码，能够执行命令，但是很受限，我们知道命令根据`cmdarray[0]`来确认以什么命令环境启动，这里确实以`/bin/sh`启动了，但是后面的命令执行的时候存在问题

```

String str = request.getParameter("cmd");
String cmdstr = "/bin/sh -c " + cmd;
Runtime.getRuntime().exec(cmdstr)

```



```

1 import java.io.*;
2
3 public class Main {
4     public static void main(String[] arg) throws IOException {
5         String cmds="echo 222 > 1.txt";
6         String[] command = { "/bin/sh", "-c", cmds };
7         String command="echo 111 > 3.txt";
8         String command="/bin/sh -c whoami | ls";
9         System.out.println(command);
10        Process proc = Runtime.getRuntime().exec(command);
11        InputStream in = proc.getInputStream();
12        BufferedReader br = new BufferedReader(new InputStreamReader(in, Charset.forName("UTF8")));
13        String line = null;
14        while((line=br.readLine())!=null) {
15            System.out.println(line);
16        }
17    }
18 }

```

最后再来一段代码，下面这段代码才会是java下命令执行的完全体。

```

String str = request.getParameter("cmd");
String[] cmdstr = { "/bin/sh", "-c", str };
Runtime.getRuntime().exec(cmdstr)

```

后面我翻到一篇文章，实际上也是差不多这个情况，实际上也是这个`StringTokenizer`这个类针对命令进行处理可能会造成非预期的结果。

The only thing to remember is that any white-space sequences vital in your command must be encoded somehow as otherwise it would be eaten by Java's *StringTokenizer*, e.g.:

```
$ java Exec 'sh -c $@|sh . echo /bin/echo -e "tab\trequired"'
```

And to anyone who is interested in what the process tree looks like:

```

$ java Exec 'sh -c $@|sh . echo ps ft'
  PID TTY          STAT       TIME COMMAND
 27109 pts/25      Ss           0:03  /bin/bash
   6904 pts/25      Sl+          0:00  \_ java Exec sh -c $@|sh . echo ps ft
   6914 pts/25      S+           0:00      \_ sh -c $@|sh . echo ps ft
   6916 pts/25      S+           0:00          \_ sh
   6917 pts/25      R+           0:00              \_ ps ft

```

最后还有一个问题，为什么一定要将命令切割成为数组，原因是因为`ProcessBuilder`，看看他的构造方法。

```

public ProcessBuilder(String... command) {
    this.command = new ArrayList<>(command.length);
    for (String arg : command)
        this.command.add(arg);
}

public ProcessBuilder(List<String> command) {
    if (command == null)
        throw new NullPointerException();
    this.command = command;
}

```

实际上它是要求 Array 类型或者 List 类型，如果我们要执行下图中的代码是不行的。

```

4 public class pb {
5     public static void main(String[] arg) throws IOException {
6         String cmds="ping -c 1 www.baidu.com";
7         ProcessBuilder pb= new ProcessBuilder(cmds);
8         Process p = pb.start();
9         InputStream in = p.getInputStream();
10        BufferedReader br = new BufferedReader(new InputStreamReader(in, charsetName: "UTF8"));
11        String line = null;
12        while((line=br.readLine())!=null) {
13            System.out.println(line);
14        }
15    }
16 }

```

pb > main()

...\_40.jdk/Contents/Home/bin/java ...  
 Implemented in both /Library/Java/JavaVirtualMachines/jdk1.8.0\_40.jdk/Contents/Home/bin/java (0x1046474c0) and /Library/...  
 Option: Cannot run program "ping -c 1 www.baidu.com": error=2, No such file or directory  
 ProcessBuilder.java:1048)

原因在于我们传入的类型不对，我们前面说过命令执行是根据cmdarray[0]，确认命令启动环境，这里自然找不到我们要启动的命令。

```

try {
    return ProcessImpl.start(cmdarray,
                             environ...,
                             dir,
                             redirec...,
                             redirec...);
} catch (IOException | IllegalArgumentException e) {
    String exceptionInfo = ": " + e.getMessage();
    Throwable cause = e.getCause();
}

```

cmdarray

cmdarray = {String[1]@701}

0 = "ping -c 1 www.baidu.com"

所以Java下的命令稍微改造一下代码就好。

```

1 import java.io.*;
2
3 public class Main {
4     public static void main(String[] arg) throws IOException {
5         String cmds="echo 111 > 1.txt";
6         String[] command = { "/bin/sh", "-c", cmds };
7         Process proc = Runtime.getRuntime().exec(command);
8     }
9 }

```

还有一种方式就是用编码，linux下可以用bash的base64编码来解决这个特殊字符的问题。

```
1 import java.io.*;
2
3 public class Main1 {
4     public static void main(String[] arg) throws IOException {
5         String cmds="bash -c {echo,ZwNobyAyMiA+IDIudHh0}{base64,-d}{bash,-i}";
6         Process proc = Runtime.getRuntime().exec(cmds);
7         InputStream in = proc.getInputStream();
8         BufferedReader br = new BufferedReader(new InputStreamReader(in, charsetName: "UTF8"));
9         String line = null;
10        while((line=br.readLine())!=null) {
11            System.out.println(line);
12        }
13    }
14 }
15
```

这里在小提一下如果遇到命令执行过滤了ProcessBuilder和getRuntime，可以考虑一下java.lang.ProcessImpl.start

0x03 小结

其实java已经尽量规避命令执行的安全问题，JDK沙盒机制会进行checkExec，执行命令的机制就是仅仅检查并执行命令数组中的第一个，而分隔符后面的所有东西都是默认忽略的。所以在java下如果遇到复杂的命令执行，且参数只能如下所示，且只有一个位置可以控制的话，建议使用base64的编码方式，windows下可以使用powershell的base64。java的反序列化框架利用框架yso，以及一些shiro这类反序列化导致的命令执行实际上很多是用了getRuntime来达到命令执行的目的，且就像我们上面说的，可控位置比较

Reference

[sh-or-getting-shell-environment-from](#)

点击收藏 | 5 关注 | 2

[上一篇：v8 exploit - Real...](#) [下一篇：通过Dpapi获取Windows身份凭证](#)

1. 1 条回复



LandGrey

[land\\*\\*\\*\\*](#) 2019-10-13 14:27:23

Runtime 执行命令，如果用 exec(String command) 方法，要规避管道符 | 和重定向符 < > >>  
用 exec(String[] cmdarray) 或其他几种类似方法，加上 try catch 就可以不考虑平台，在命令中正常用管道符和重定向符了

```
try{
    java.lang.Runtime.getRuntime().exec(new String[]{"bin/bash", "-c", "echo 1 > 1.txt"});
}catch (java.io.IOException e){
    try{
        java.lang.Runtime.getRuntime().exec(new String[]{"cmd", "/c", "echo 1 > 1.txt"});
    }catch (java.io.IOException ee){

    }
}
```

1 回复Ta

[登录](#) 后跟帖

先知社区

[现在登录](#)

[技术文章](#)

[社区小黑板](#)

[目录](#)

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)