安全小课堂第九十七期【web漏洞挖掘之业务逻辑漏洞】

京东安全应急响应中心 5月28日

业务逻辑漏洞是一种设计缺陷，通常表现为设计者或开发者在思考过程中作出的特殊假设存在明显或隐含的错误，攻击者会设法了解设计者与开发者作出的可能假设，从而进行攻破。JSRC **安全小课堂第九十七期**，邀请到**Urahara**作为讲师就**web漏洞之业务逻辑漏洞**为大家进行分享。感谢白帽子盆友的精彩提问与互动~

支付类逻辑漏洞有哪些场景？

京安小妹

Urahara：

1）支付金额大小篡改。这个是比较常见的，一般在支付的关键步骤数据包中直接提交了需要支付该商品的具体金额，但后端并没有对该金额进行校验，传递过程中也没有签名，导致金额在被抓包改包篡改提交至后台后成功支付；

2）支付金额或商品购买数量没有进行负数限制。这种情况也较为常见，如果在相关数据包中能够修改商品的购买数量或者单价，再将其中一个改为负数，那个按照"商品单价x购买数量=需要支付的总额"的逻辑，就会导致一个负数的支付金额，若支付成功，可能导致用户账户余额不减反增的结果。

3）程序执行异常导致支付金额为0或绕过支付逻辑。比如整数溢出的情况，可以参考之前BEC智能合约无限转币的漏洞，逻辑是差不多的，恶意用户通过传入一个特别大的value（商品数量）值，导致代码在处理该value时或者处理运算后的amount总金额时发生整数溢出

4）由于不同语言特性在float或double使用不当时导致的精度丢失问题，可能造成所需支付金额与实际支付金额不对等的情况，甚至根据场景构造达到通过1分钱完成支付特别大金额的订单。这种漏洞比较少也较难发现，一般都是通过审计发现。

5）竞争条件漏洞。这种漏洞一般出现在获取和使用优惠券还有支付等业务场景中，当多个并发的执行线程访问共享资源时，根据执行代码的时间，无意中产生不同的结果，这主要是由于缓存或数据同步时间差导致的。简单举个例子，比如账户余额只有10元，但在进行并发的多订单支付时，成功支付了总价值12元的多个订单。这部分大家可以学习《Race conditions on web》这篇文章。

竞争条件可以参考这个

https://www.josipfranjkovic.com/blog/race-conditions-on-web

讲师

逻辑漏洞和设计缺陷漏洞是如何区分的？

京安小妹

Urahara：

逻辑漏洞一般指开发人员在实现业务功能时由于考虑不全面导致可能出现非预期的业务操作造成的安全问题，而设计缺陷漏洞指业务功能在最初设计时未考虑全面，而使该业务在使用中存在的一些潜在安全风险。

讲师

密码重置逻辑漏洞中，绕过第二步关键短信验证，直接进去第三步重置新密码的漏洞场景，具体是后端代码中的哪些缺陷导致的？

京安小妹

Urahara：

这种属于业务流程乱序，出现这种情况其实都是服务端对业务流程执行顺序检验不足造成的，正确情况下应该是进行每一个步骤之前先检验上一个步骤是否执行或验证通过，也就是说上一个步骤的执行结果是进行下一个步骤执行的充分必要条件，执行链上各环节属于一个预设好的顺序执行过程，而出现业务流程乱序时，每个执行环节可能是独立的，相互之间没有关联，这样就可能造成绕过关键业务流程环节的风险，比如身份验证。

分享一些关于逻辑漏洞的真实代码案例?

京安小妹

Urahara:

分享两个简单一点的逻辑缺陷

第一个任意用户密码重置

```
public String bql_password(){
        he = "mmzh";
        product_name = "密码找回";

        if (StringUtils.isBlank(phone)) {
            hu = "请输入手机号!";
            return "password";
        }

        String sql = "";
        int flag = getUserFlag(phone, 1);
        UserAcctInfo uinfo;
        BoolReturnBO jiekou = new BoolReturnBO();
        if (flag != 0) {
            if (flag == 1) {
            //根据用户手机号找回密码
sql = "update ServiceZhPwdLog set flag=flag+1,code=code+1 where intime=CONVERT(varchar(15),getdate(),23) and telNum='"
                        + phone + "'";
                String newpass = org.apache.commons.lang.RandomStringUtils
                        .randomNumeric(6);
                try {
                    jiekou = nsisvc.resetUserPassword(phone, newpass, RSAUtils.decryptStringByJs(idNum));
                    if (jiekou.getResultcode().equals("0")) {
                        hu = "jkcg";//接口调用成功;

                    } else if(jiekou.getResultcode().equals("9999999")) {
                        hu = "jkcs";//接口超时;
                    } else {
                        hu = "jksb";//接口失败;
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
            bsdao.executeDelOrUpdateHql(sql, false);
```

```
public BoolReturnBO resetUserPassword(String phone, String passwd,
            String psptid) throws Exception {

        HashMap<String, String> map = new HashMap<String, String>();
        BoolReturnBO retbo = new BoolReturnBO();
        map.put("SERIAL_NUMBER", phone);
        map.put("USER_PASSWD", "");
        map.put("X_NEW_PASSWD", passwd);
```

```
        map.put("PS_ID", psid);
```

这个代码的业务功能是从前台获取了一个手机号码 然后根据这个手机号重置了手机号对应账户的密码,重置密码调用resetUserPassword这里边也没有进行身份校验。

正确业务逻辑应该是提交手机号，向该手机号码发送短信验证码，用户输入验证码，后台验证手机号码一致性和短信验证码有效性，验证通过后进行该手机号账户的密码重置。

第二个关于短信验证码重放的。

```
public void doService(CSContext csContext, CSResult result)
    {
        UserInfoBean user = BusinessUtil.getUser(csContext.getRequest());

        Map<String, Object> resultMap = new HashMap<String, Object>();
        String opType = RequestUtil.getStringValueFromRequest(csContext.getRequest(), "opType", "");

        if(null==user && !"0".equals(opType)){
            result.setResultCode(IResultCode.CS_HANDLE_FAILED);
            result.setSystemCode("-100001");
            return;
        }
        //执行发送验证码操作时没有进行频率限制
        if("sendSMSCode".equals(opType)){
            resultMap = sendSMSCode(csContext,user,resultMap);
            resultMap.put("isLogin", "isLogin");
        }
```

发送短信验证码操作无频率限制，可导致短时间内无限制发送大量验证码。

正确业务逻辑短信服务器限制短信发送频率，一般为1分钟1次，并设置单用户每日最大发送次数限额。

讲师

逻辑漏洞的防御手段?

京安小妹

Urahara：

逻辑漏洞因为是和业务紧密联系，这种漏洞在被利用时数据包中是没有payload所言的，所以类似于waf这种应用漏洞的防御手段显然是没有用的，防御和解决业务逻辑漏洞还是要从业务本身出发，做好风险管理，了解掌握各类业务流程和环节中的关键风险节点和风险程度，以及对应有效的风险管理方法和技术，完善风控系统规则等，当然随着技术发展可能会出现机器学习在此类漏洞防御方面的应用，据我所知目前还有这方面的落地产品都还停留在理论阶段或者起步阶段。除此之外，技术方面还应关心漏洞本身的问题，一切漏洞都产生于代码层面和配置层面，加强应用的上线前安全评估和审计必不可少，像逻辑漏洞也是要在代码层面去彻底解决的，由于业务逻辑漏洞和业务关联性太强，每种漏洞都需要单独去考虑解决方案，这里就不展开讲了，简单总结了以下几个方面：

1）完善权限管理和鉴权机制；
2）对重要数据进行签名，防止篡改；
3）重要操作验证在后端进行，比如时间限制，最大值限制等；
4）加强验证码机制和防爆破、防重放机制；
5）业务流程顺序执行关联；

讲师

本期JSRC 安全小课堂到此结束。更多内容请期待下期安全小课堂。如果还有你希望出现在安全小课堂内容暂时未出现，也欢迎留言告诉我们。

安全小课堂的往期内容开通了自助查询，点击菜单栏进入“安全小课堂”即可浏览。

简历请发送：cv-security@jd.com

微信公众号：jsrc_team

新浪官方微博：京东安全应急响应中心