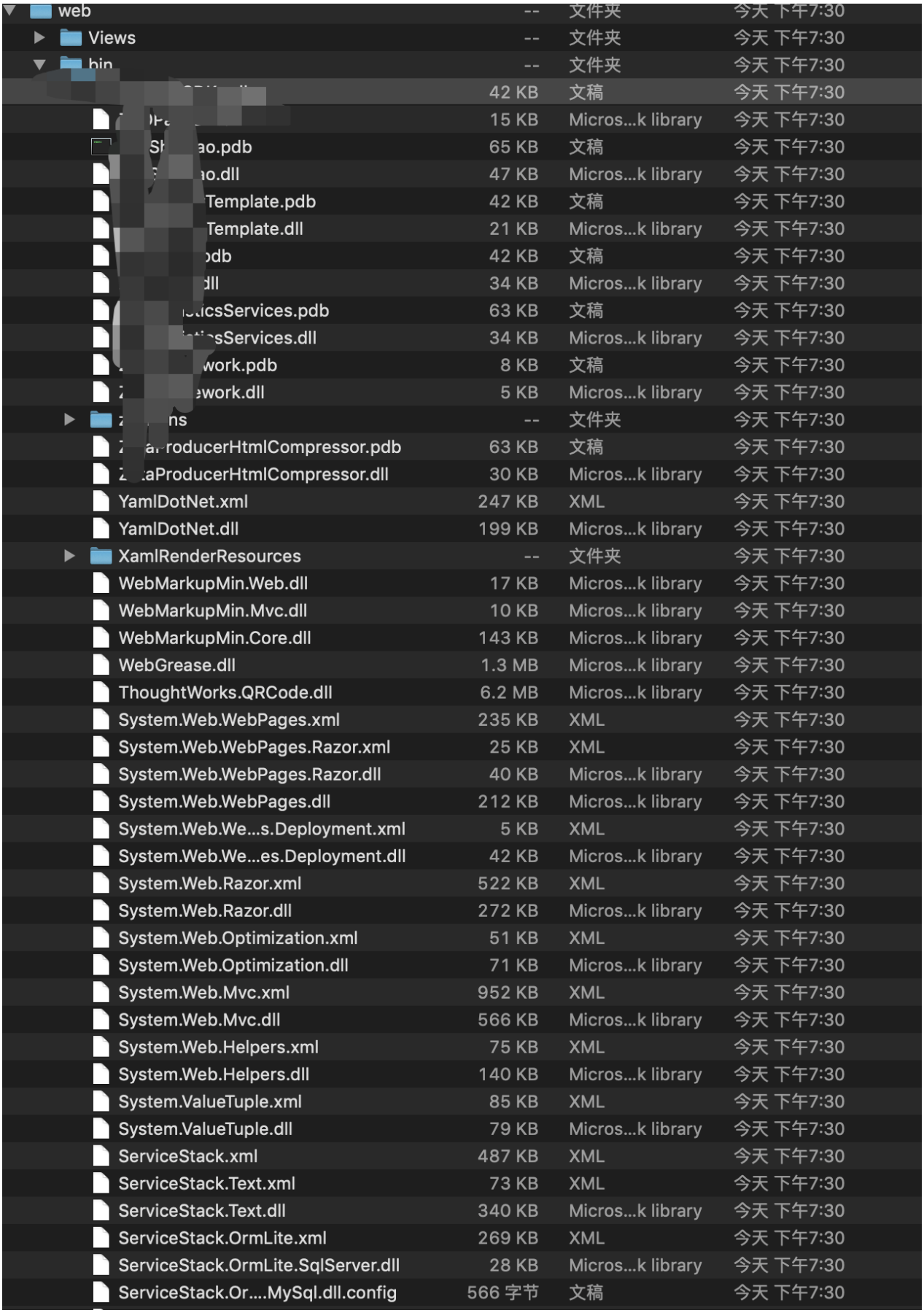
前言

最近在测试某src的时候。发现了某个站点存在源码泄露，xxx.com/www.zip可直接下载源码。通过分析该站点源码以及加上一些黑盒测试，最后成功getshell该站点。

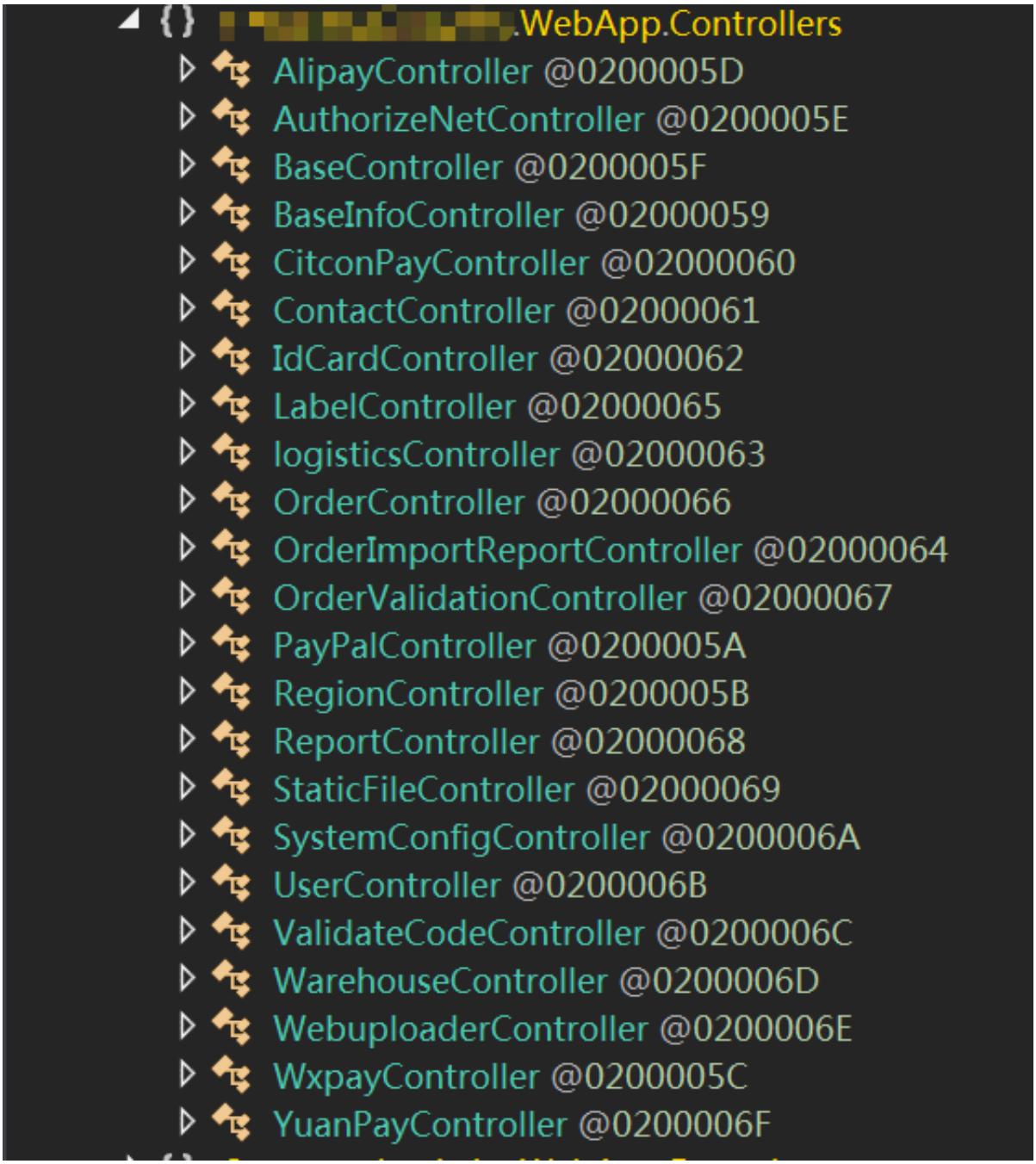
过程

当拿到源码的时候首先看了一下该源码的结构，如图所示：



一看就是.net源码。话不多说，开始反编译代码。我这里使用DNSPY进行反编译。反编译后查看其架构，是mvc。

然后找到控制器代码。如图所示：



剩下的就是逐步看代码。首先找到了一处上传：

```
326         [HttpPost]
327         public ContentResult Upload()
328         {
329             HttpFileCollectionBase files = base.Request.Files;
330             bool flag = files != null;
331             if (flag)
332             {
333                 for (int i = 0; i < files.Count; i++)
334                 {
335                     HttpPostedFileBase file = files[i];
336                     bool flag2 = file == null;
337                     if (!flag2)
338                     {
339                         string pic = "/" + Path.GetFileName(file.FileName);
340                         string path = Path.Combine(base.Server.MapPath(base.Request.Form["currentPath"] ?? ""), pic);
341                         file.SaveAs(path);
342                     }
343                 }
344             }
345             return new ContentResult
346             {
347                 Content = "uploaded successfully."
348             };
349         }
```

用户注册登录进去后，构造上传包进行上传，但是服务器却抛出错误了，此处代码不可利用。继续找。

“/”应用程序中的服务器错误。

运行时错误

说明:
服务器上出现应用程序错误。此应用程序的当前自定义错误设置禁止远程查看应用程序错误的详细

详细信息: 若要使他人能够在远程计算机上查看此特定错误消息的详细信息，请在位于当前 Web 应用程序根目录下的“web.config”配置文件中创建一个 <customErrors> 标记。然后应将此 <customErrors> 标记的“mode”特性设置为“Off”。

```
<!-- Web.Config 配置文件 -->
<configuration>
    <system.web>
        <customErrors mode="Off"/>
    </system.web>
</configuration>
```

然后又发现一处代码，如下图所示：

```
[HttpPost]
public JsonResult UploadImage()
{
    string extension = base.Request.Headers["X-File-Extension"];
    string fileType = base.Request.Headers["X-File-Type"];
    string[] fileTypeProps = fileType.Split(new char[]
    {
        '/'
    });
    bool flag = fileTypeProps.Length != 2;
    JsonResult result;
    if (flag)
    {
        result = base.Json(new
        {
            result = false,
            message = "文件类型不正确"
        });
    }
    else
    {
        bool flag2 = fileTypeProps[0] != "image";
        if (flag2)
        {
            result = base.Json(new
            {
                result = false,
                message = "文件类型不是图片"
            });
        }
        else
        {
            int fileSize = Convert.ToInt32(base.Request.Headers["X-File-Size"]);
            string localPath = Path.Combine(HttpRuntime.AppDomainAppPath, "Uploads/XamlLabelImage");
            bool flag3 = !Directory.Exists(localPath);
            if (flag3)
            {
                Directory.CreateDirectory(localPath);
            }
            Stream fileContent = base.Request.InputStream;
            fileContent.Seek(0L, SeekOrigin.Begin);
            byte[] fileBytes = fileContent.ReadFully();
            string hashFileName = null;
            try
            {
                using (MD5 md5Hash = MD5.Create())
                {
                    string hash = MD5Helper.GetMd5Hash(md5Hash, Convert.ToBase64String(fileBytes));
                    hashFileName = string.Format("{0}.{1}", hash, extension);
                }
                using (FileStream save = new FileStream(Path.Combine(localPath, hashFileName), FileMode.OpenOrCreate,
                  FileAccess.ReadWrite))
                {
                    save.Write(fileBytes, 0, fileBytes.Length);
                }
                result = base.Json(new
                {
                    result = true,
                    data = string.Format("/Uploads/XamlLabelImage/{0}", hashFileName)
                });
            }
            catch (Exception ex)
            {
                result = base.Json(new
                {
                    result = false,
                    message = "上传文件出现异常: " + ex.Message
                });
            }
        }
    }
    return result;
}
```

该处代码判断文件类型的时候有一个很明显的缺点，它取文件类型的时候是取的是

```
base.Request.Headers["X-File-Type"]
```

而取后缀的时候是这个

```
ase.Request.Headers["X-File-Extension"]
```

该段代码只是简单的判断了一下X-File-Type的值而已，对后缀并没有进行判断，因此确定该处存在任意文件上传。

确定有机会getshell了后，开始构造上传报文，但是回显如下：

Error.

An error occurred while processing your request.

然后细心排查了一下，发现是因为自己没有访问该控制器的权限导致的，该控制器声明了权限：

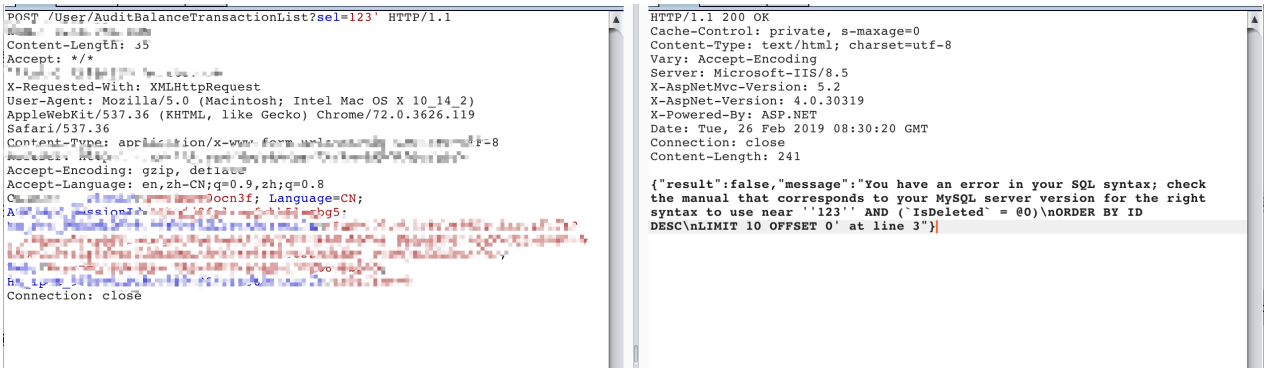
```
18      [Permissions("WEB_LabelManage")]
19      public class LabelController : BaseController
```

而我们注册用户的权限是没有的。那么现在要么找到一个越权漏洞，要么就找个注入获取高权限用户的密码信息。

不出意外的在UserController里面找到了如下这样的代码：

```
1642        [HttpPost]
1643        public string AuditBalanceTransactionList(string sel, int pageNumber, int pageSize)
1644        {
1645            string filter = " filename is not null and status is not null";
1646            bool flag = !string.IsNullOrWhiteSpace(sel);
1647            if (flag)
1648            {
1649                filter = string.Format(" filename is not null and status='{0}'", sel);
1650            }
1651            PagedResponse<BalanceTransaction> transactionListResponse =
                  ServiceClient.Send<PagedResponse<BalanceTransaction>>(new GetBalanceTransactionList
1652            {
1653                Filter = filter,
1654                PageNumber = pageNumber,
1655                PageSize = pageSize,
1656                Sort = "ID DESC"
1657            }, null, false, null);
1658            bool flag2 = transactionListResponse.ResponseStatus != null;
```

很明显这个函数的sel参数是存在注入的。但是由于该代码使用的ServiceStack框架，我一直找不到该框架反编译后的Service的信息（**有大佬知道的请告诉我**）。但是没有具体的Service的逻辑关系也不大，只要知道这个地方有注入就OK了。然后开始试着注入：



OK，注入确实存在。试着注入出数据：

```
{"result":false,"message":"Potential illegal fragment detected:
filename is not null and status='1' and (select 1) '-1'"}
```

发现使用select的时候被拦截了，找一下拦截点。

在ServiceStack.OrmLite.OrmLiteUtils中有这样的检测：

```
408     public static string SqlVerifyFragment(this string sqlFragment)
409     {
410         return sqlFragment.SqlVerifyFragment(OrmLiteUtils.IllegalSqlFragmentTokens);
411     }
412
413     // Token: 0x060006E2 RID: 1762 RVA: 0x000159FC File Offset: 0x00013BFC
414     public static string SqlVerifyFragment(this string sqlFragment, IEnumerable<string> illegalFragments)
415     {
416         if (sqlFragment == null)
417         {
418             return null;
419         }
420         string text = sqlFragment.StripQuotedStrings('\'').StripQuotedStrings('"').StripQuotedStrings
              ('`').ToLower();
421         foreach (string value in illegalFragments)
422         {
423             if (text.IndexOf(value, StringComparison.Ordinal) >= 0)
424             {
425                 throw new ArgumentException("Potential illegal fragment detected: " + sqlFragment);
426             }
427         }
428         return sqlFragment;
429     }
430
```

然后这个IllegalSqlFragmentTokens的内容如下：

```
938         public static string[] IllegalSqlFragmentTokens = new string[]
939         {
940             "--",
941             ";--",
942             ";",
943             "%",
944             "/*",
945             "*/",
946             "@@",
947             "@",
948             "char",
949             "nchar",
950             "varchar",
951             "nvarchar",
952             "alter",
953             "begin",
954             "cast",
955             "create",
956             "cursor",
957             "declare",
958             "delete",
959             "drop",
960             "end",
961             "exec",
962             "execute",
963             "fetch",
964             "insert",
965             "kill",
966             "open",
967             "select",
968             "sys",
969             "sysobjects",
970             "syscolumns",
971             "table",
972             "update"
973         };
974
```

好吧.有框架的防注入，应该是不行了。那么就只有找越权的点了，找了很久，发现了如下代码：

```
565         [HttpPost]
566         public JsonResult UpdateUser(UserAccountDetails model)
567         {
568             string editResult = "用户新建成功.";
569             UserClaim uc = new UserClaim();
570             bool flag = model.UserClaimList == null;
571             if (flag)
572             {
573                 model.UserClaimList = new List<UserClaim>();
574             }
575             bool flag2 = model.UserAccount.ID > 0L;
576             if (flag2)
577             {
578                 try
579                 {
580                     model.UserAccount.ModifiedDate = new DateTime?(DateTime.Now);
581                     bool flag3 = base.Request.Form["IDNumber"] != null;
582                     if (flag3)
583                     {
584                         bool flag4 = long.Parse(base.Request.Form["ClaimID"]) > 0L;
585                         if (flag4)
586                         {
587                             uc.Type = "IDNumber";
588                             uc.Value = base.Request.Form["IDNumber"];
589                             uc.ID = long.Parse(base.Request.Form["ClaimID"]);
590                             model.UserClaimList.Add(uc);
591                         }
592                         else
593                         {
594                             uc.Type = "IDNumber";
595                             uc.Value = base.Request.Form["IDNumber"];
596                             model.UserClaimList.Add(uc);
597                         }
598                     }
599                     Response<bool> response = ServiceClient.Send<Response<bool>>(new UpdateUserAccountWithClaimList
600                     {
601                         Model = model.UserAccount,
602                         ClaimList = model.UserClaimList
603                     }, null, false, null);
604                     editResult = "编辑用户成功.";
605                 }
606                 catch
607                 {
608                     editResult = "编辑用户失败.";
609                 }
610             }
611             else
612             {
```

这个函数是用来更新用户的，该入口在普通用户也是看不到的，应该是提供给管理员操作的，毕竟管理员这种才可以新建用户和更新已有用户信息。但是该函数并没有做权限的控制，普通用户可以直接构造报文访问该函数。因此我试着更新id为1的用户的邮箱为我自己的邮箱，因为id为1的用户大概率是管理员用户，更改该用户的邮箱后，然后通过找回密码功能即可将重置链接发送至我邮箱进行密码重置。

如下图：

```
POST /user/UpdateUser HTTP/1.1
Content-Length: 164
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_2)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Accept-Encoding: gzip, deflate
Accept-Language: en,zh-CN;q=0.9,zh;q=0.8
Connection: close

UserAccount.ID=1&UserAccount.Username=test&UserAccount.Email=
ail.com&UserAccount.MobilePhoneNumber=
93&ClaimID=-1
```

修改完毕后就可以直接通过找回密码功能重置密码，如下图所示。

尊敬的客户：

test，您好！

您在 提交找回密码请求，请点击下面的链接重设新的密码:

点击重设密码

如果您无法点击以上按钮，请将此以下链接复制到浏览器地址栏后访问：

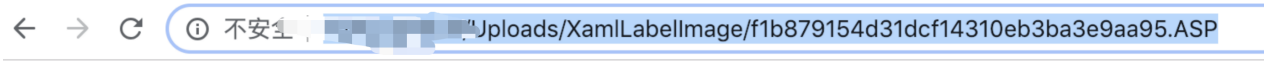
User/PasswordReset?token=8b34c 5a528e ad578

为了保证您帐号的安全性，该链接有效期为24小时，并且重设密码成功之后将失效! 如果您误收到此电子邮件，则可能是其他用户在尝试帐号设置时的误操作，如果您并未发起该请求，则无需再进行任何操作，并可以放心地忽略此电子邮件。

此邮件为自动发送，请勿回复，谢谢！

然后重设密码登陆后果然是管理员权限，构造最开始的上传报文即可Getshell(上传asp文件会被删掉，所以更改了下后缀名大小写绕过)。

访问之后成功解析。如下图。



security test!

后记

运气好运气好。