黑客是如何攻击 WebSockets 和 Socket.io的

mss**** / 2018-08-14 08:21:57 / 浏览数 8141

原文：https://www.blackhillsinfosec.com/how-to-hack-websockets-and-socket-io/

WebSockets概述

WebSockets是一种允许浏览器和服务器建立单个TCP连接，并进行双向异步通信的技术。这种技术非常适合Web应用程序，采用该技术之后，浏览器无需在后台发送数百个

除了Burp Suite之外，还有一些其他工具也能用来处理WebSockets。虽然我们已经尝试过所有的工具，但没有一个完全符合我们的胃口。

- Zed Attack Proxy（ZAP）
- Pappy Proxy
- Man-in-the-Middle Proxy（mitmproxy）
- WebSocket/Socket.io（WSSiP）

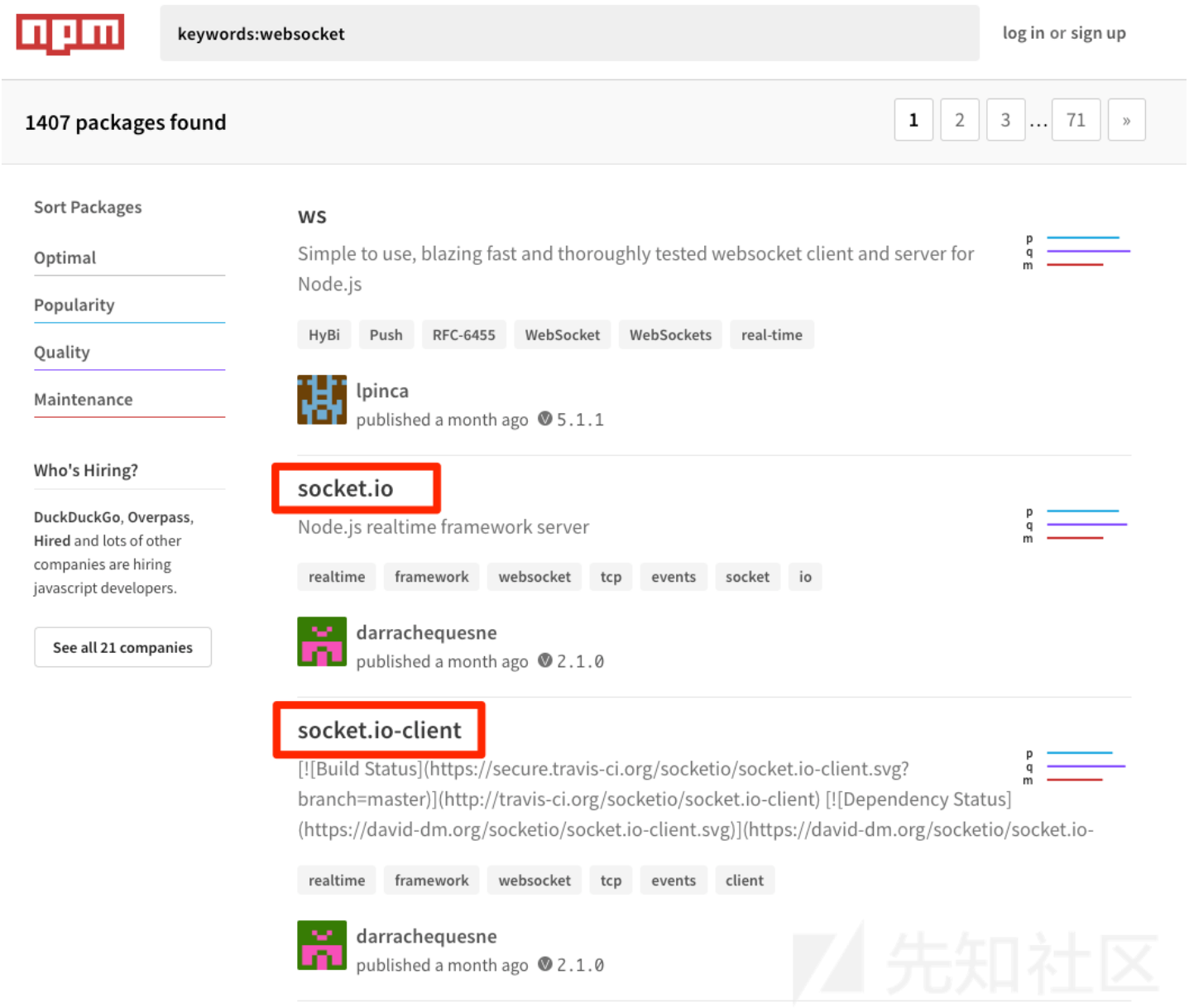
对于希望通过WebSockets来绕过进攻端的安全检测的读者来说，可以参阅下面这篇文章。

```
https://www.blackhillsinfosec.com/command-and-control-with-websockets-wsc2/
```

在本文中，我们关注的重点是socket.io，这是一个流行的JavaScript
WebSockets库。然而，需要说明的是，文中介绍的攻击思路不仅适用于其他库，同时，也适用于WebSockets协议。

那么，socket.io到底有多受欢迎呢？它在Github上收获了41,000多颗星。

同时，在NPM网站的WebSockets包排行榜上，它们还占据了第二名和第三名的位置。



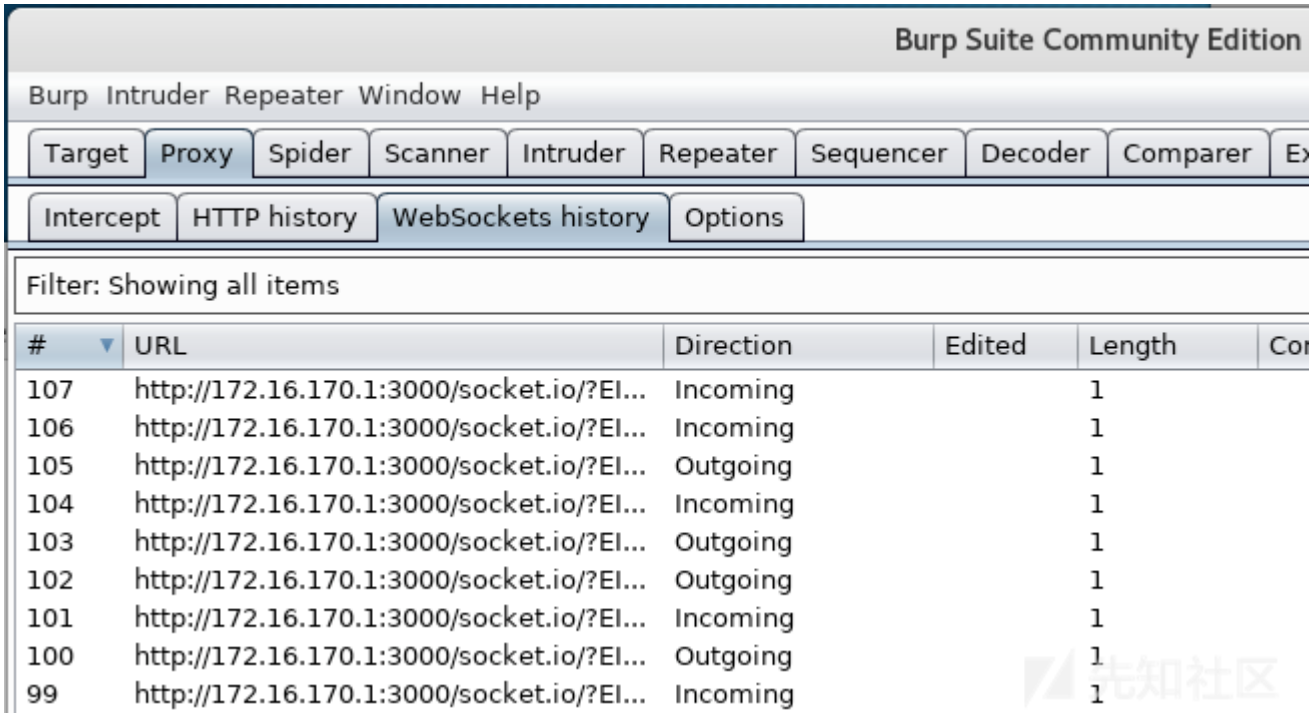
事实上，就连优秀的[OWASP Juice-Shop项目]也采用了socket.io库，因此，我们决定使用socket.io来完成相应的演示。

```
https://github.com/bkimminich/juice-shop/search?utf8=%E2%9C%93&q=socket.io&type=
```

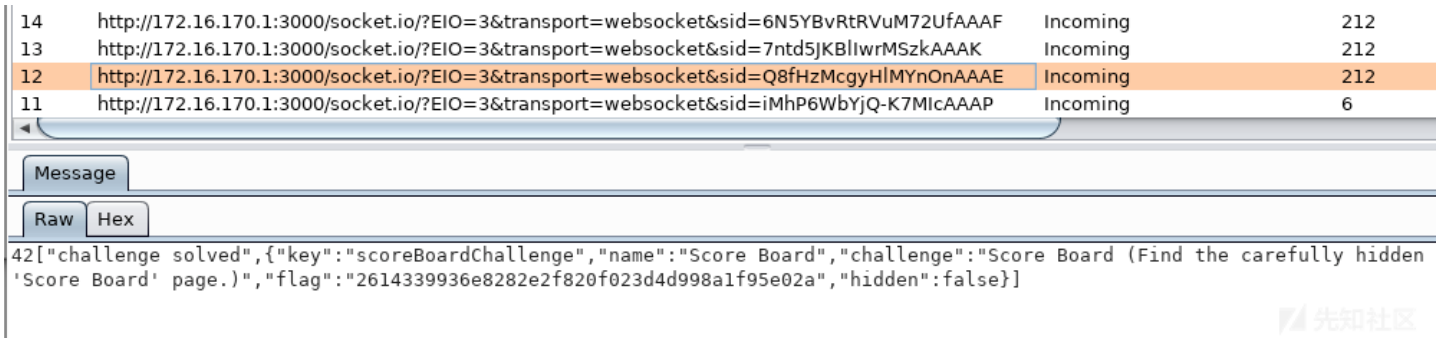
在本文中，我们假设读者可以熟练使用Burp
Suite测试Web应用程序，同时，文中涉及的所有测试工作，都可以利用该软件的社区版本来完成。废话少说，直入主题吧！

如果通过浏览器访问Juice-Shop的话，就可以在后台快速考察WebSocket的流量了。为此，可以打开Burp，然后转到Proxy->
WebSockets历史记录，从这里就可以看到相关的流量了。

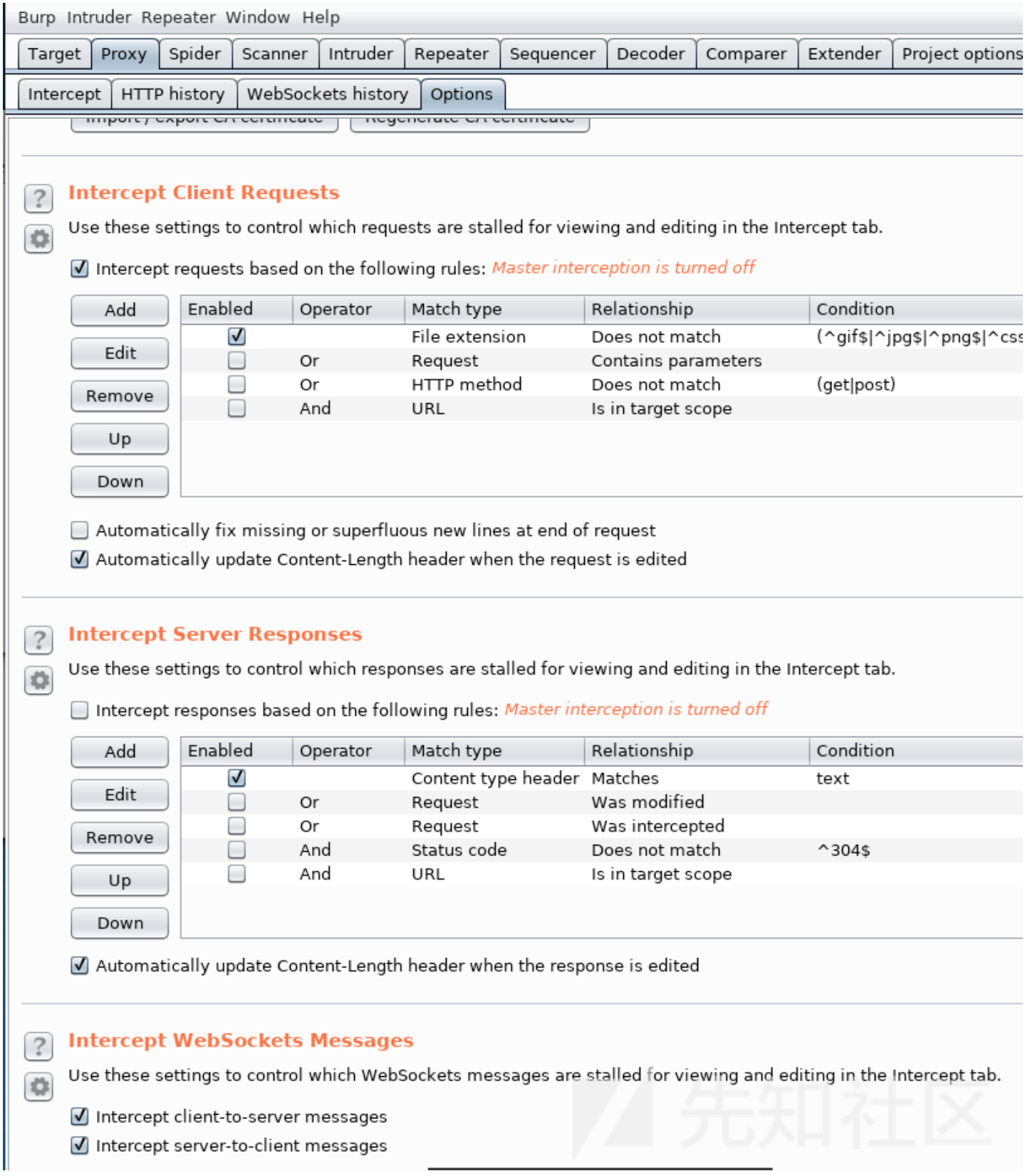
我们知道，HTTP是一种无状态协议，所以，它需要不停的发送请求/响应对；与此相反，WebSockets则是一种有状态协议。这就意味着，我们可以从服务器获得任意数量的

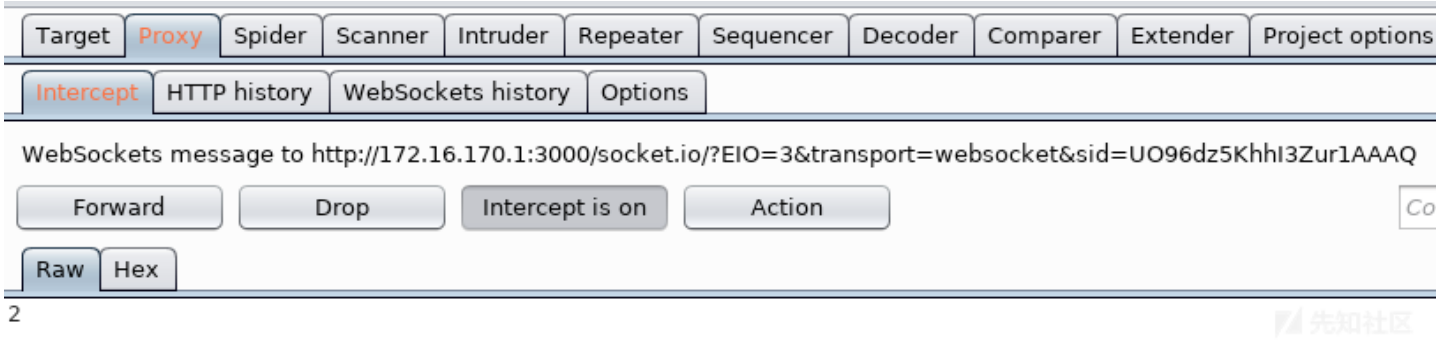


在该视图中，我们看到的，主要是发送和接收的单字节消息。但是，当应用程序执行一些有趣的操作时，我们将看到带有更大的有效载荷的消息。

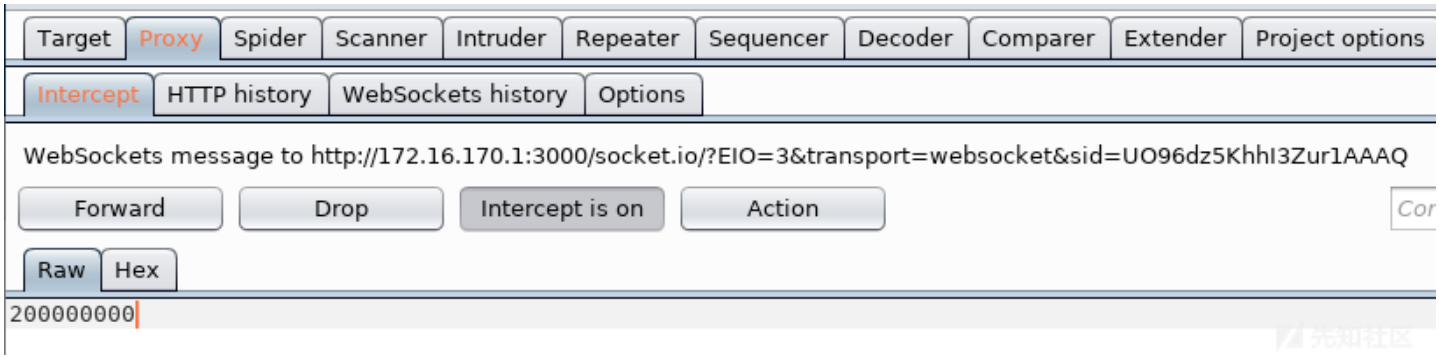


Burp提供的许多功能，也可以用来测试WebSockets。比如，Burp可以实时拦截和修改WebSocket消息，遗憾的是，Burp仍然缺乏针对WebSockets的Repeater、Scanne

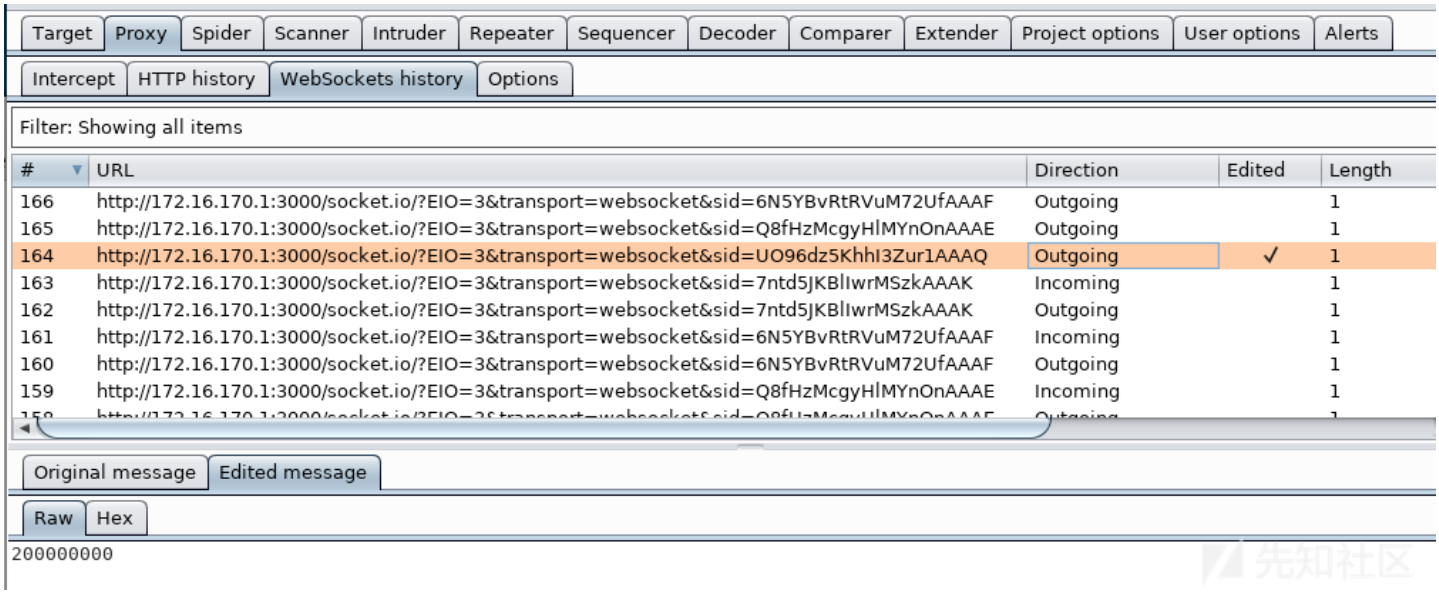




我们将会收到截获的WebSocket消息，这里跟处理HTTP消息的方式别无二致。同样，我们也可以在拦截窗口中编辑这些消息。



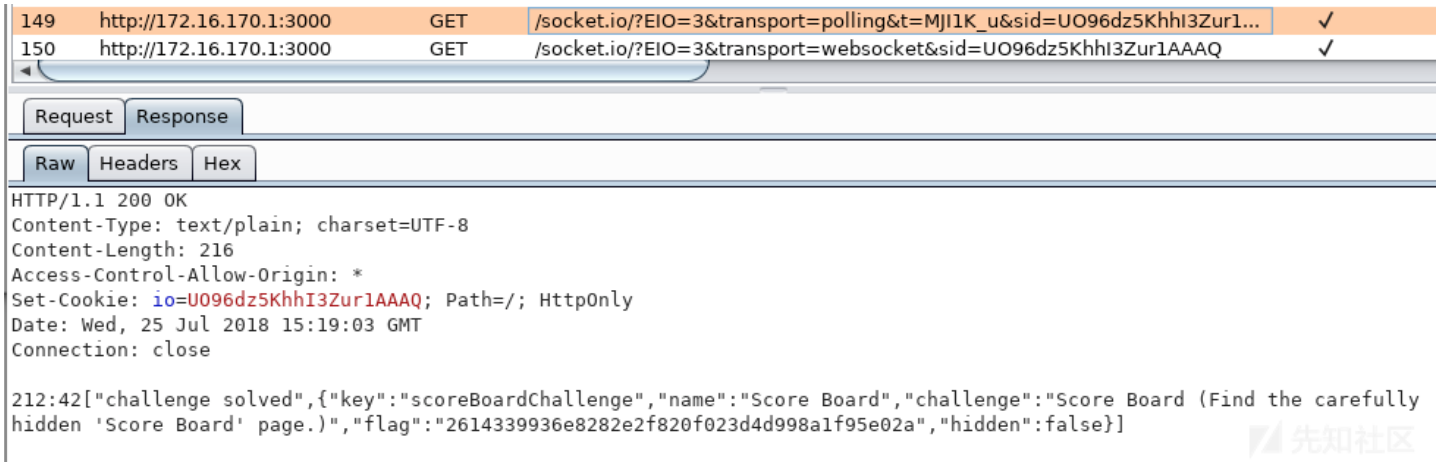
之后，就可以在WebSockets历史记录选项卡中查看编辑后的消息了。



将WebSockets降级为HTTP

方法1：活用Socket.io的HTTP备用机制

我很快注意到了一件奇怪的事情：有时，我会在HTTP历史记录中看到类似于在WebSockets历史记录中所见过的消息。实际上，这个有趣的WebSockets消息与回答记分板



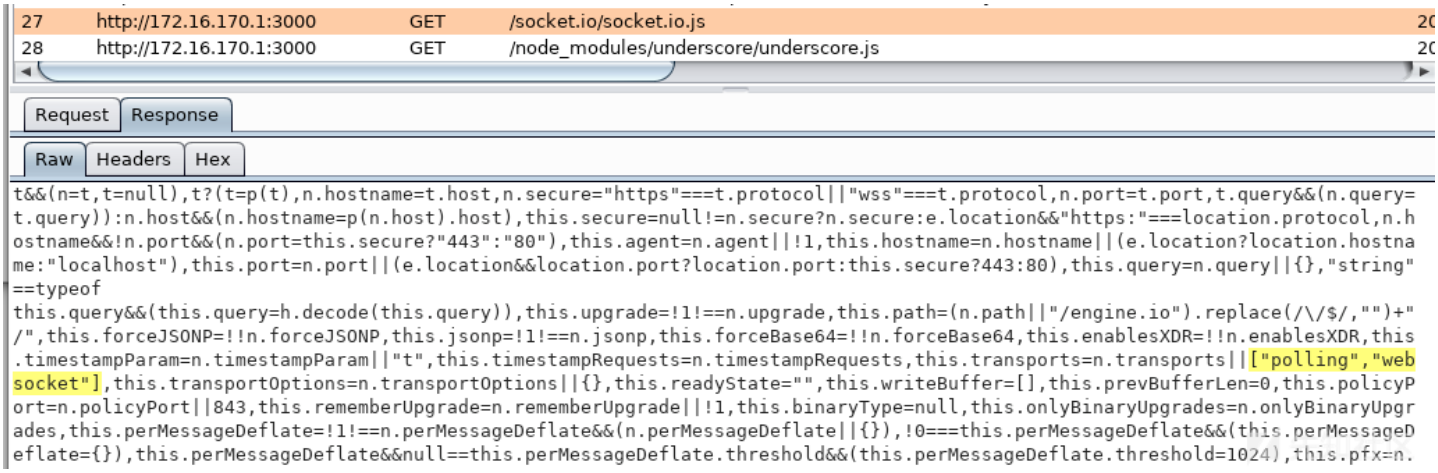
之所以允许使用HTTP，根据我的推测，是为了在WebSockets不受支持或因某种原因而被阻止的情况下，使应用程序仍然可以正常运行。传输参数之所以会引起我的注意，

在socket.io的文档中，有一个章节对“polling”和“websockets”这两个默认传输选项的运行机制进行了介绍。同时，它还介绍了如何通过将WebSockets指定为唯一的传输机

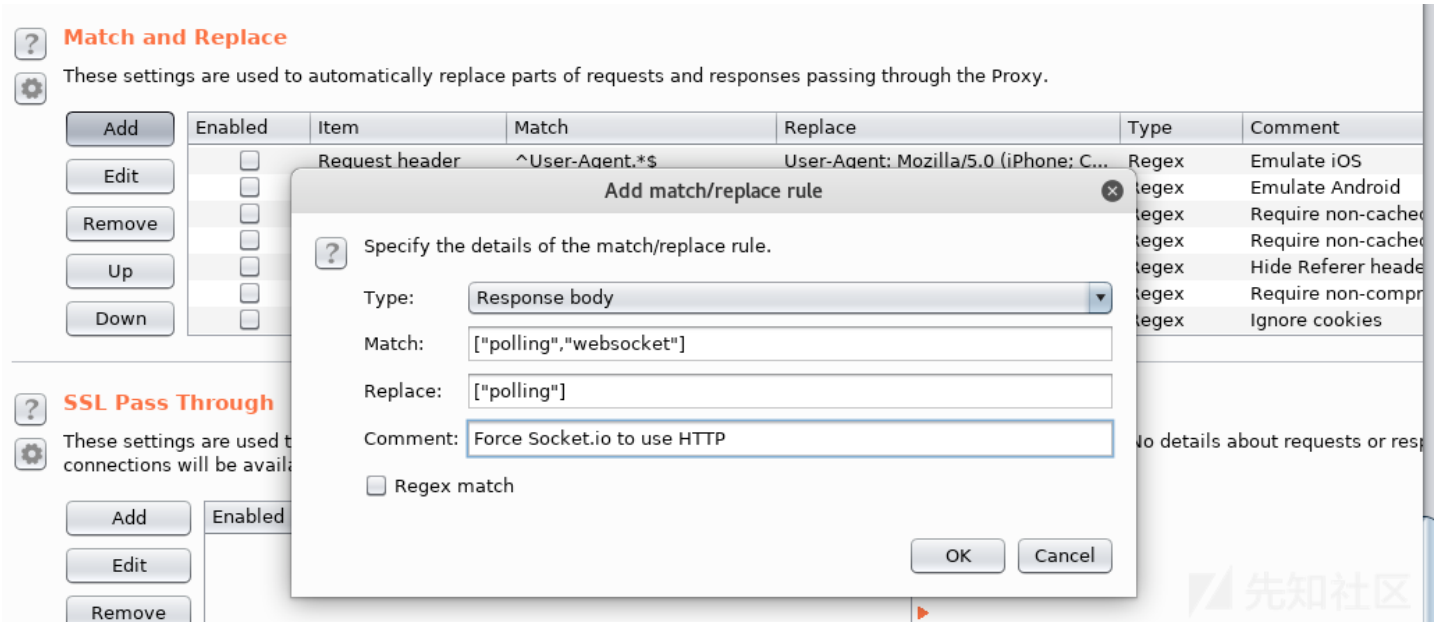
`https://socket.io/docs/client-api/#with-WebSocket-transport-only`

在浏览socket.io.js源代码过程中，我无意中发现了以下代码，看起来对我们非常有用。

```
this.transports=n.transports||["polling","WebSocket"]
```

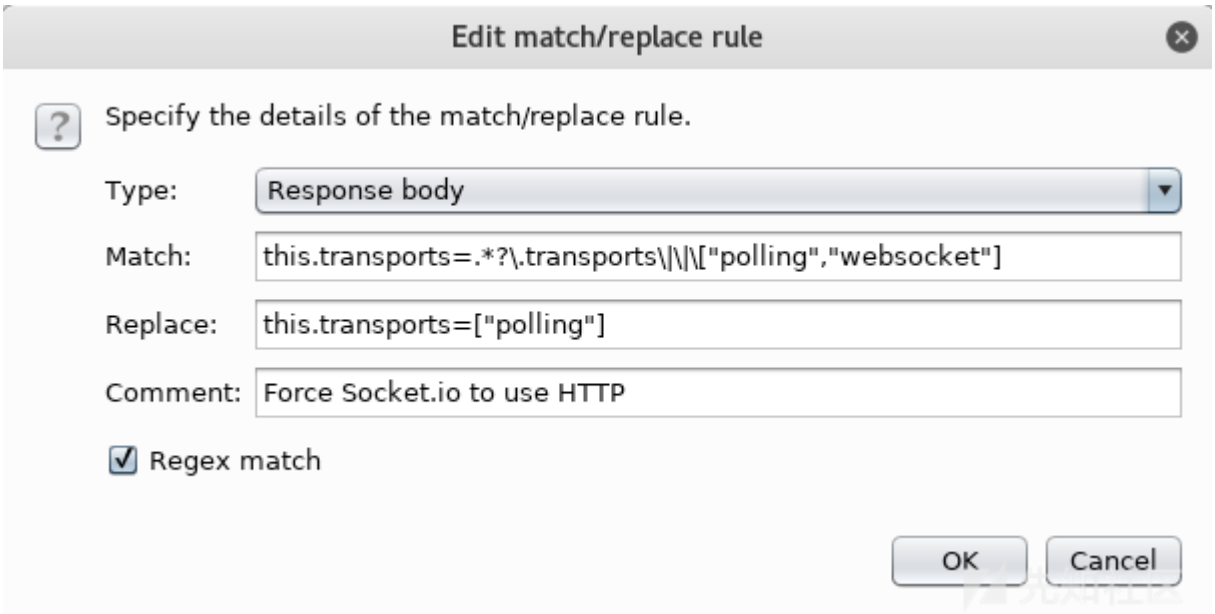


上面这行代码会将一个名为"transports"的内部变量设置为传入的某个值，但是，如果传入的值为false/empty的话，就将其设为默认的 ["polling","websocket"] 。到目前为止，这当然符合我们对轮询和WebSockets的默认传输的理解。那么，接下来让我们看看，当我们在Burp的Proxy->Options选项中通过设置匹配和替换规则来改



添加规则后，刷新页面（必须启用Burp的内置规则"Require non-cached response"或执行强制刷新），这样，数据就不再通过WebSockets发送了。

这很好，但是，如果您使用的应用程序已经提供了优先于我们的新默认值的传输选项呢？在这种情况下，可以修改匹配和替换规则。下面给出的规则，将适用于socket.io库的



为了便于复制粘贴，下面给出对应的字符串：

```
this\.transports=.*?\.transports\|\|\["polling","websocket"]
```

```
this.transports=["polling"]
```

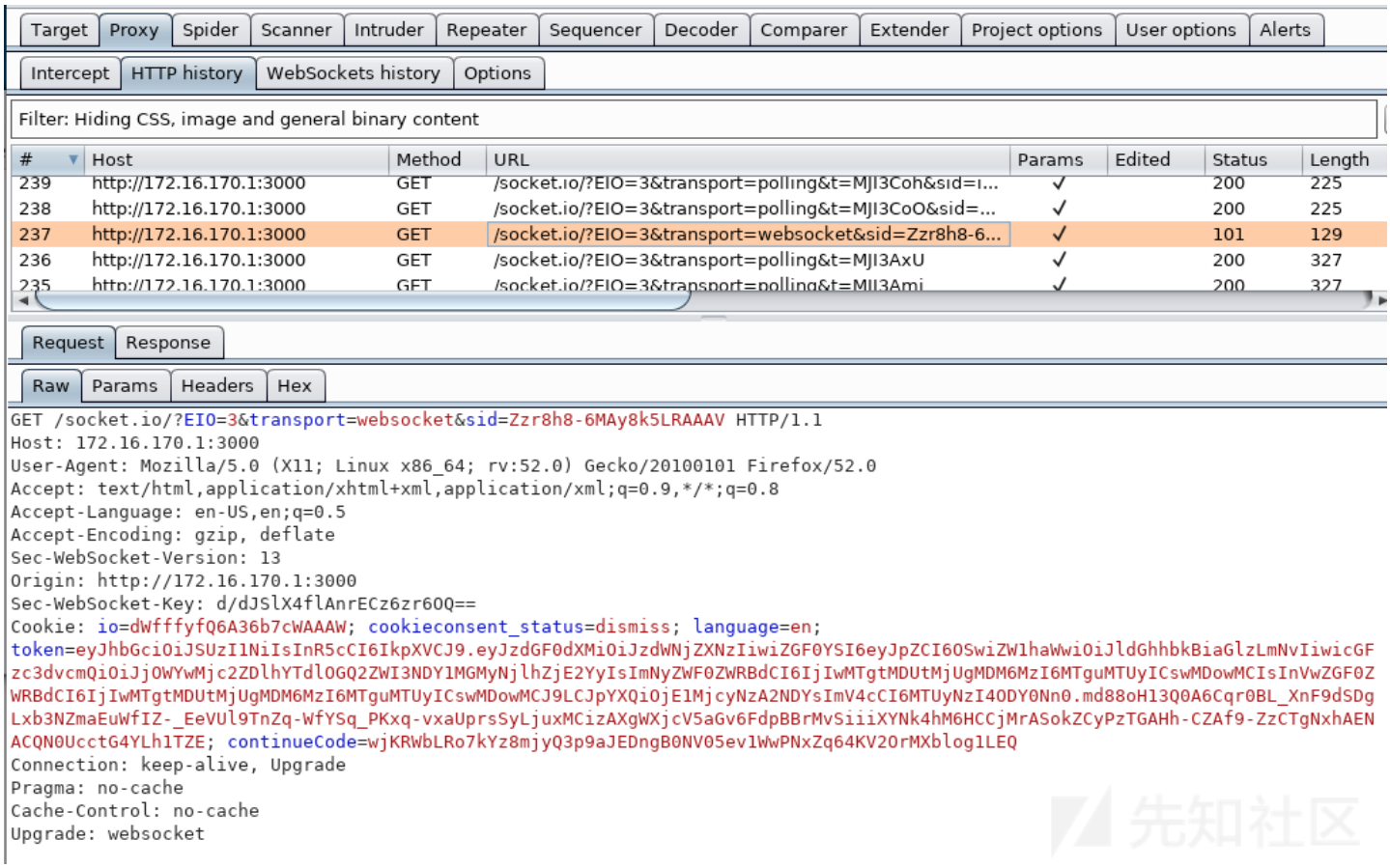
请务必将其设置为正则表达式匹配。

方法2：阻止WebSockets升级

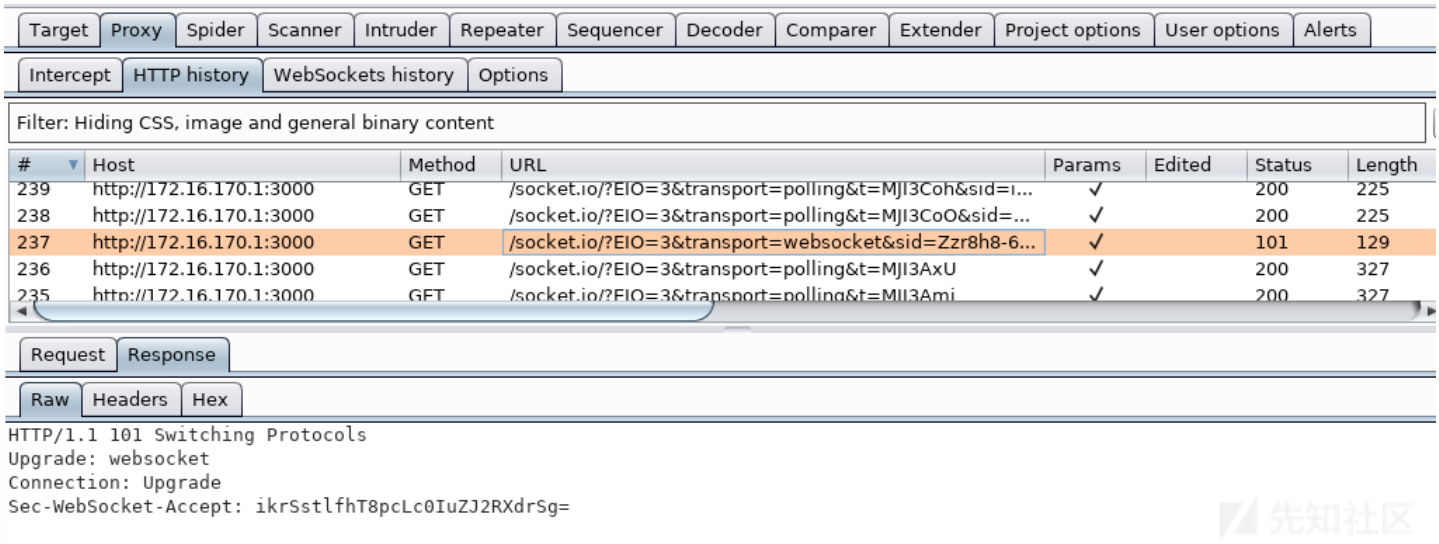
需要说明的是，方法1只能用于socket.io，不过，如果经过适当修改的话，也可以用于其他客户端库。但是，下面介绍的方法将更加通用，因为它是以WebSockets协议本身

经过一番研究后，我发现WebSockets会首先通过HTTP进行通信，以便与服务器进行相关的协商，然后将连接"升级"为WebSocket方式。为此，需要完成的重要工作包括：

1）客户端发送带有WebSocket某些特殊头部的升级请求。



2）服务器将返回状态码101 Switching Protocols，以及WebSocket的某些特殊头部。



3）通信转换为WebSockets方式，之后，就看不到用于该特殊会话的HTTP请求了。

WebSockets RFC文档的第4.1节提供了如何中断这个工作流程的相关线索。

以下内容节选自https://tools.ietf.org/html/rfc6455#section-4.1页面：

1.如果从服务器收到的状态代码不是101，那么客户端仍然根据HTTP
[RFC2616]过程来处理响应。特别是，如果客户端收到401状态代码，则可能进行身份验证；服务器可使用3xx状态代码重定向客户端（但客户端不需要遵循它们），等等。否

2.如果响应中缺少头部字段Upgrade，或头部字段Upgrade包含的值与大小写敏感的ASCII字符串"WebSocket"不完全匹配的话，客户端必须废除这个WebSocket
连接。

3.如果响应中缺少头部字段Connection，或头部字段Connection包含的值与大小写敏感的ASCII字符串"Upgrade"不完全匹配的话，客户端必须废除这个WebSocket连接。

4.如果响应中缺少Sec-WebSocket-Accept头部字段，或Sec-WebSocket-Accept头部字段的值并非是由Sec-WebSocket-Key（作为字符串，未经base64解码）与字符串"
，客户端必须废除这个WebSocket连接。

5.如果响应中包括Sec-WebSocket-Extensions头部字段，并且该头部字段要求使用的扩展并没有出现在客户端的握手消息中（服务器指示的扩展并非是客户端所请求的），

通过考察上面"必须废除"连接的条件，我想出了如下文所示的一套替换规则，这些规则可以在上述五种情形下都会令客户端废除WebSocket连接。

Enabled	Item	Match	Replace	Type	Comment
☑	Response header	HTTP/1.1 101 Switching Protocols	HTTP/1.1 200 OK	Literal	Disable Websockets
☑	Response header	Upgrade: websocket		Literal	Disable Websockets
☑	Response header	Connection: upgrade	Connection: close	Literal	Disable Websockets
☑	Response header	^Sec-WebSocket-.*$		Regex	Disable Websockets

一旦这些规则就位，所有WebSocket升级请求都会失败。由于socket.io默认情况下无法使用HTTP，因此，这些规则完全可以达到理想的效果。当然，某些特殊实现或其他原

Request | Original response | Auto-modified response

Raw | Params | Headers | Hex

```
GET /socket.io/?EIO=3&transport=websocket&sid=iUTykeQQumxFJgEJAABL HTTP/1.1
Host: 172.16.170.1:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Sec-WebSocket-Version: 13
Origin: http://172.16.170.1:3000
Sec-WebSocket-Key: Ozl4AEQ/doHJSt/rn+39VQ==
Cookie: io=iUTykeQQumxFJgEJAABL; cookieconsent_status=dismiss; language=en;
token=eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJ
OGQ2ZWI3NDY1MGMyNjlhZjE2YyIsImNyZWF0ZWRBdCI6IjIwMTgtMDUtMjUgMDM6MzI6MTguMTUyICswMD
iOjE1MjcyNzA2NDYsImV4cCI6MTUyNzI4ODY0Nn0.md88oH13Q0A6Cqr0BL_XnF9dSDgLxb3NZmaEuWfIz
6HCCjMrASokZCyPzTGAHh-CZAf9-ZzCTgNxhAENACQN0UcctG4YLh1TZE; continueCode=5VO3DL3k6v
Connection: keep-alive, Upgrade
Pragma: no-cache
Cache-Control: no-cache
Upgrade: websocket
```

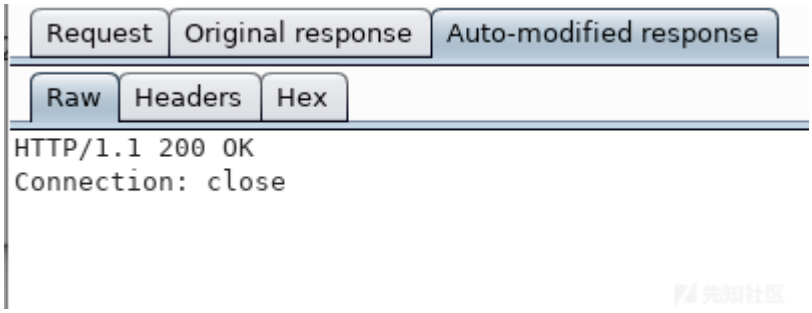
原始响应看起来就是这种情况，并且会导致客户端和服务器转而使用WebSockets进行通信。

Request | Original response | Auto-modified response

Raw | Headers | Hex

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: UDhmOR2JM0wMNdm6NhE4KKucdD4=
```

实际上，客户端收到的响应虽然来自服务器，但是却已经进行了相应的修改，按照RFC的规定，它会废弃该WebSockets连接。



我在测试中遇到的一件事是，在使用这些匹配和替换规则后，客户端在重试WebSockets连接时非常持久，并在我的HTTP历史记录中产生了大量不必要的流量。如果您正在

将Burp Repeater用作Socket.io客户端

由于我们已经迫使通信流量通过HTTP进行传输，而非通过WebSockets进行传输，所以，现在可以添加自定义的匹配和替换规则，将其应用于过去通过WebSockets传输的

接下来，我们可以完成更进一步的修改，从而为使用Repeater、Intruder和Scanner等工具铺平道路。当然，这些修改都是针对socket.io库的。

不过，当我们重发socket.io使用的HTTP请求时，还面临两个问题。

1. 每个请求都有一个会话号，任何无效请求都将导致服务器终止该会话。
2. 每个请求的主体都有一个计算字段，用来表示消息的长度。如果这个值不正确的话，服务器会将其视为无效请求并终止会话。

以下是应用程序使用的几个示例URL。

```
/socket.io/?EIO=3&transport=polling&t=MJJR2dr
```

```
/socket.io/?EIO=3&transport=polling&t=MJJZbUa&sid=iUTykeQQumxFJgEJAABL
```

URL中的“sid”参数表示到服务器的单个连接流。如果发送了无效消息（这种情况在攻击过程中很常见），那么服务器将关闭整个会话，这样的话，我们就不得不重启新会话。

给定请求的主体中含有一个字段，其中存放有效载荷的字节数。这类似于“Content-Length”HTTP头部，只不过该字段的值近针对socket.io的有效载荷而已。例如，如果您要

下面是一个消息体的示例。该示例取自Juice-Shop应用程序中的响应，不过请求的格式是完全相同的。注意，这里的“215”表示“：”之后的有效载荷的长度。

215:42[“challenge solved”,{“key”:”zeroStarsChallenge”,”name”:”Zero Stars”,”challenge”:”Zero Stars (Give a devastating zero-star feedback to the store.)”,”flag”:”e958569c4a12e3b97f38bd05cac3f0e5a1b17142”,”hidden”:false}]

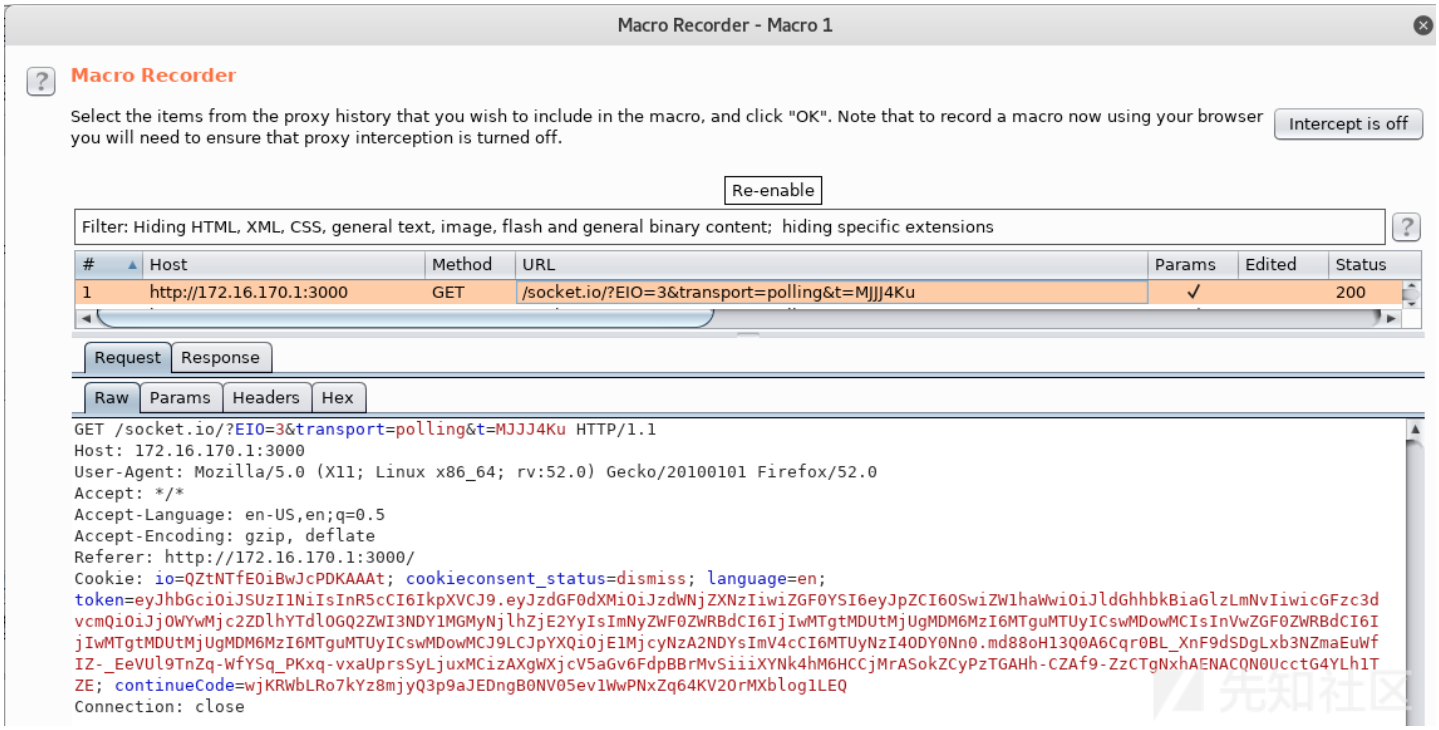
宏指令

我们可以使用Burp宏指令来解决第一个问题。基本上，每当Burp发现服务器拒绝消息时，宏指令都会自动建立新会话，并用有效的“sid”更新原始请求。要想使用宏指令，请
options->Sessions->Macros->Add选项，这样就可以创建新的宏指令了。

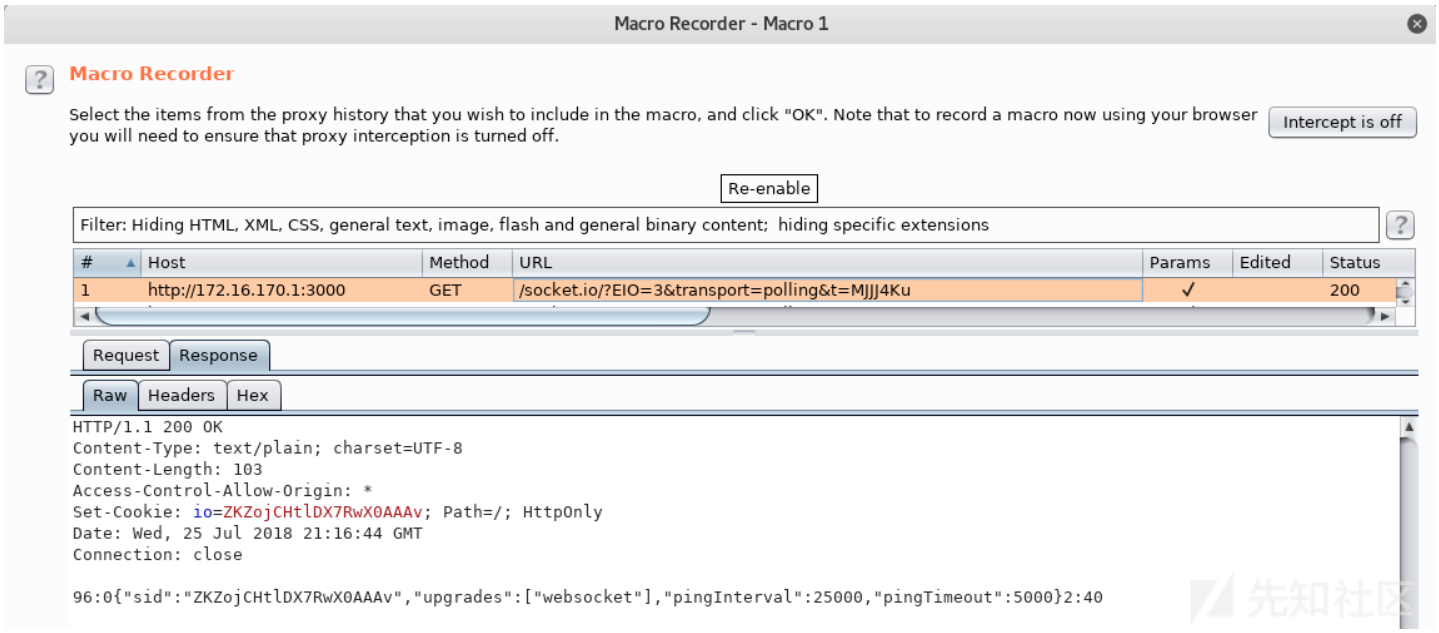
为了获取建立新会话的URL，只需省略“sid”参数即可，例如：

```
/socket.io/?EIO=3&transport=polling&t=MJJJ4Ku
```

我发现，“t”的值并不重要，所以我没有理会它。

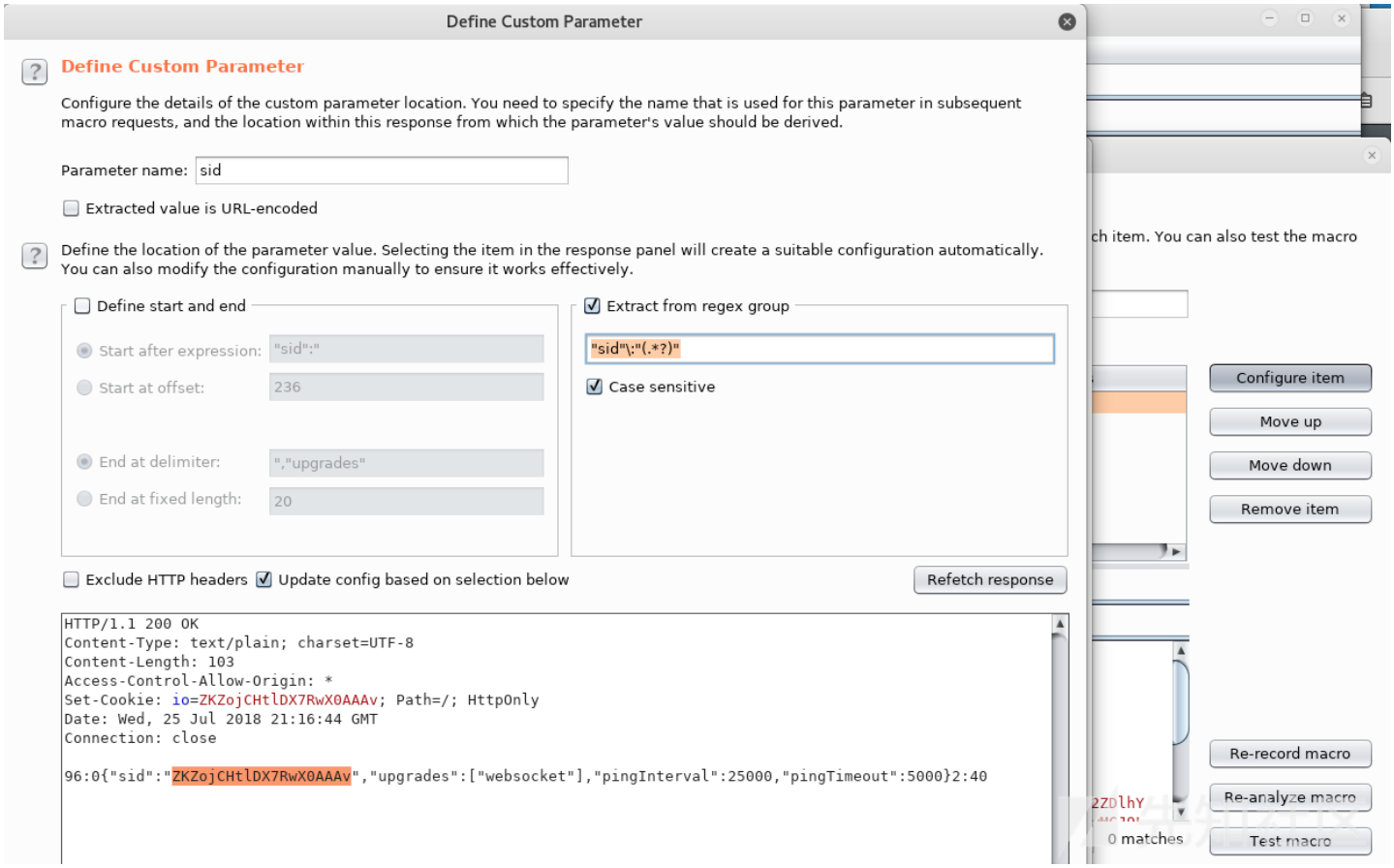


服务器响应包含了一个供我们使用的全新“sid”值。

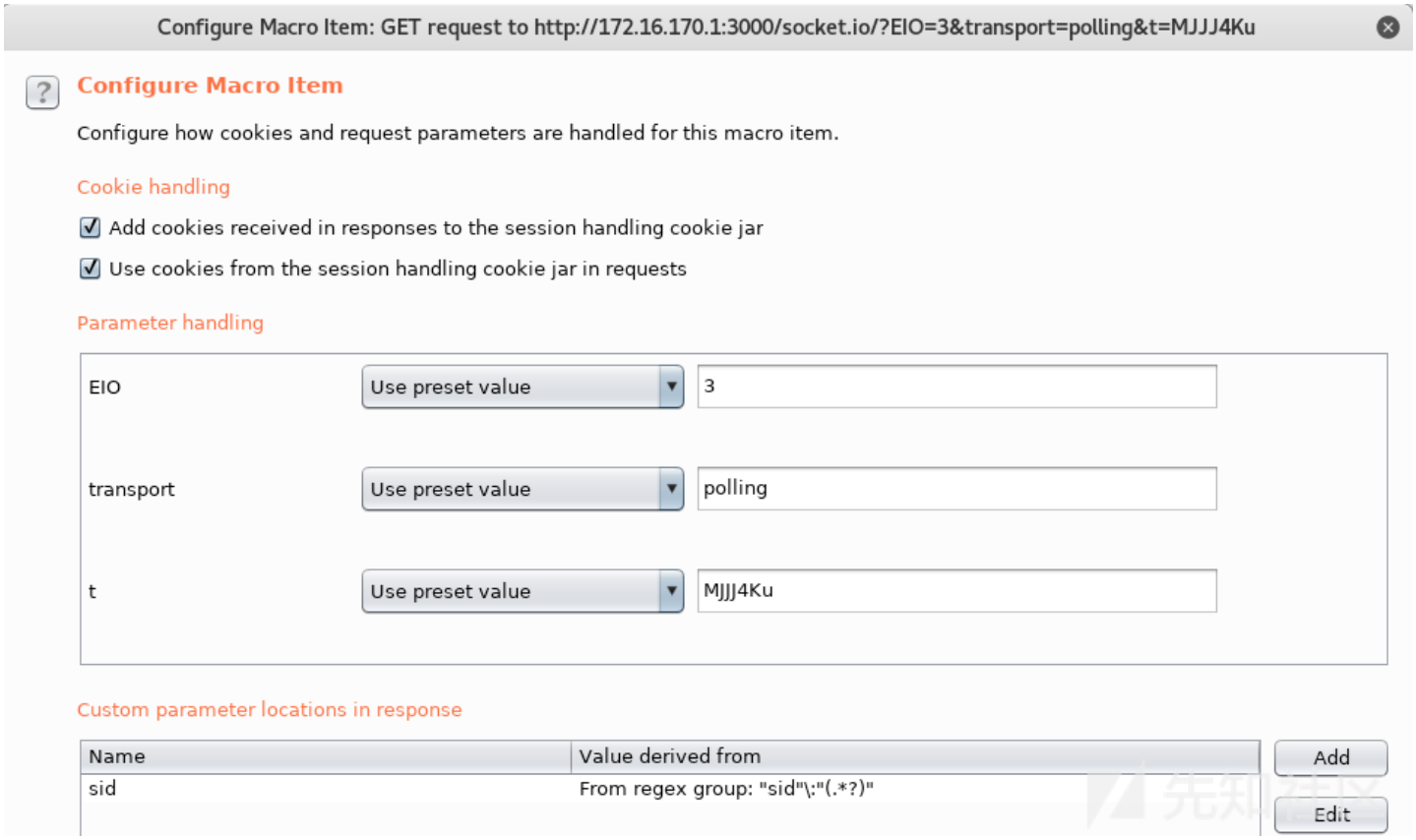


接下来，单击“Configure item”按钮，并将参数名称命名为“sid”。然后，选择“Extract from regex group”选项，并使用如下所示的正则表达式。

```
"sid"\:"(.*?)"
```



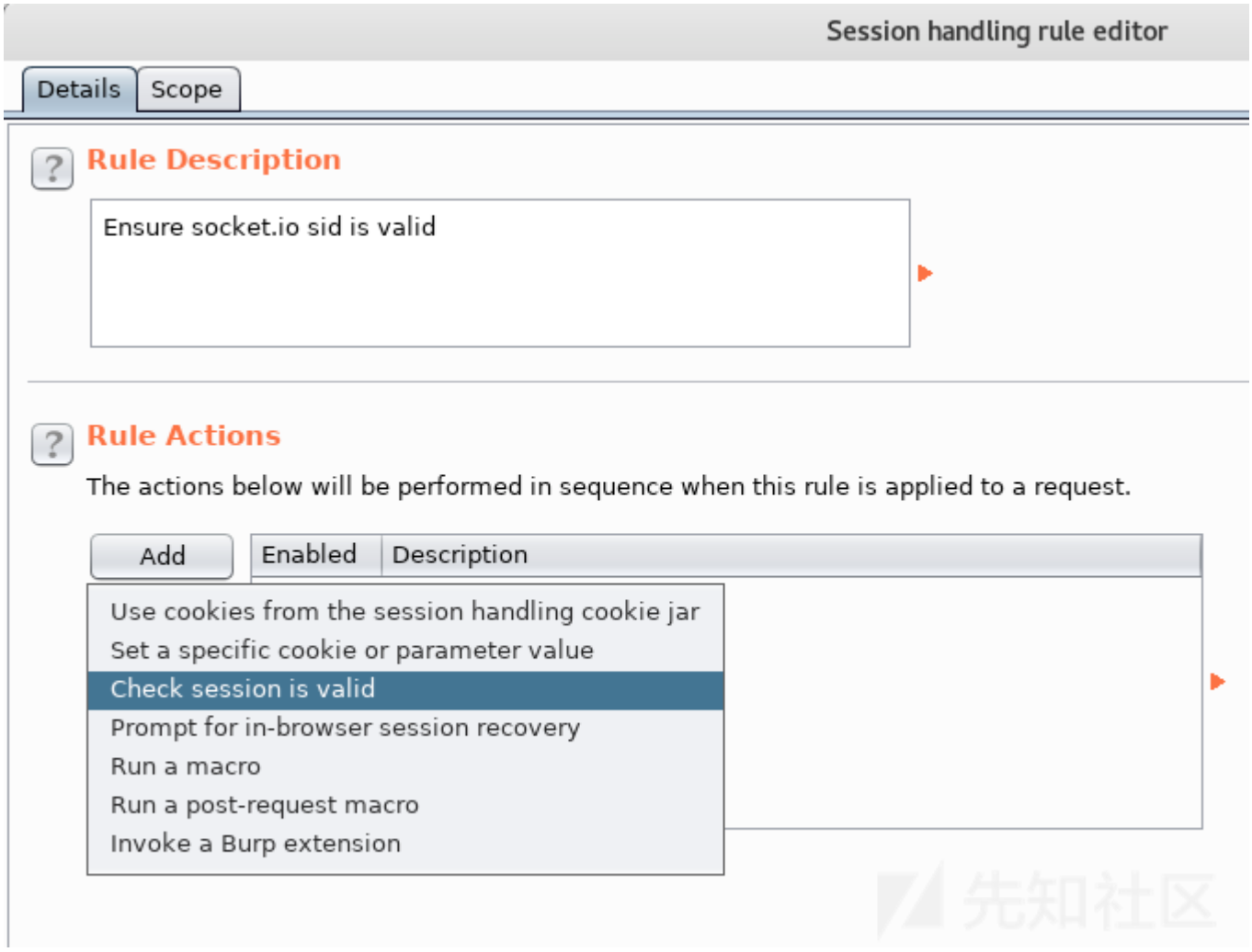
这时，配置窗口应如下所示：



会话处理规则

现在，我们已经建好了一个宏指令，接下来要做的是，设法触发它。这时候，Burp会话处理规则就派上用场了。为此，请选择Project options->Sessions->Session Handling Rules->Add菜单项。

为“Check session is valid”创建新的规则动作



对新的规则动作进行配置，具体如下：

Session handling action editor - Ensure socket.io sid is valid

This action checks whether the session is valid, by either issuing the current request or running a predefine
inspecting the response. If the session is invalid, you can optionally perform a further action to obtain a ne

Make request(s) to validate session:

- (•) Issue current request
- ( ) Run macro:

Add | Edit | New socket.io sid

- [ ] Validate session only every 10 requests

Inspect response to determine session validity:

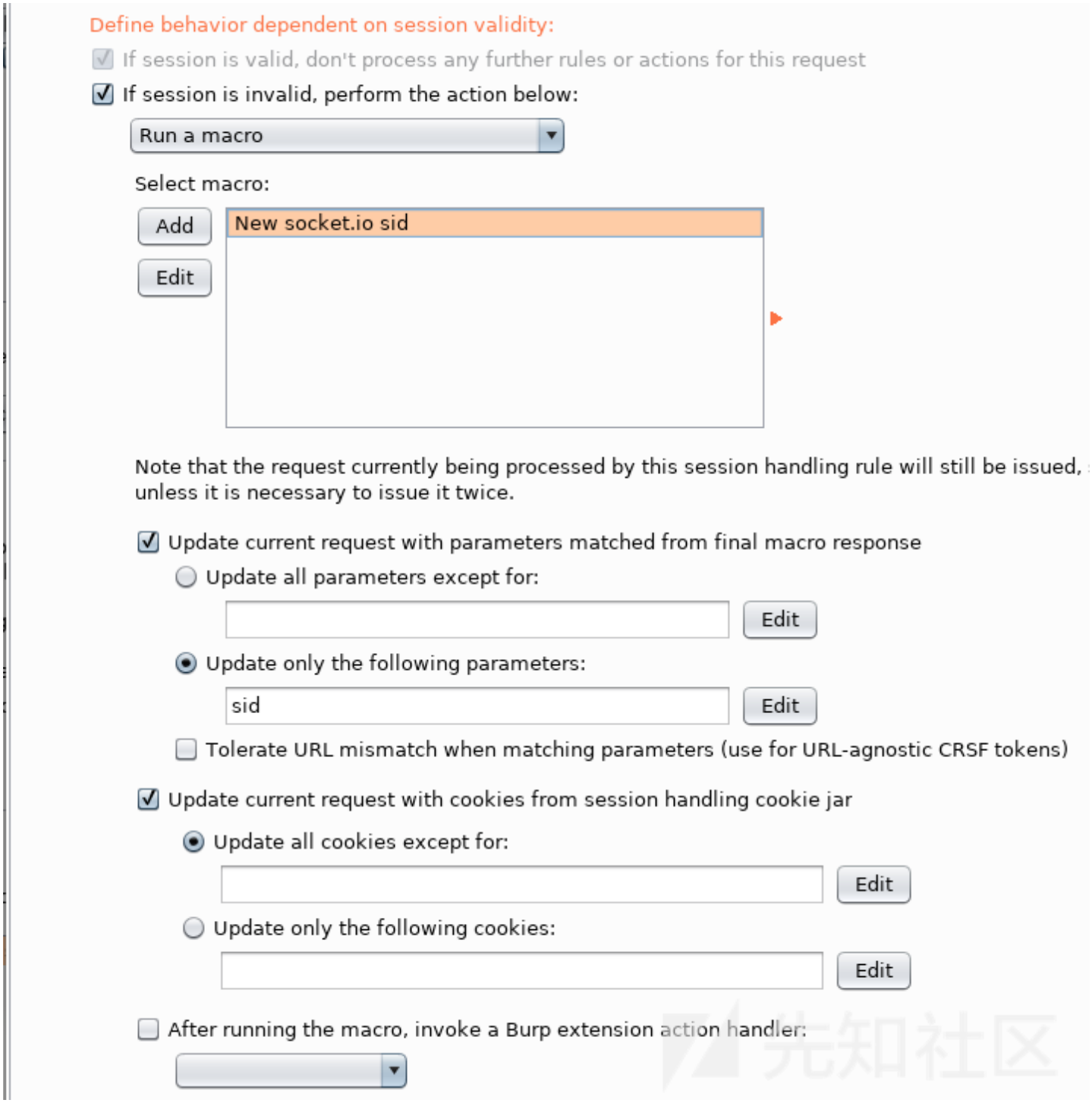
Location(s):
- [ ] HTTP headers
- [x] Response body
- [ ] URL of redirection target

Look for expression: Session ID unknown

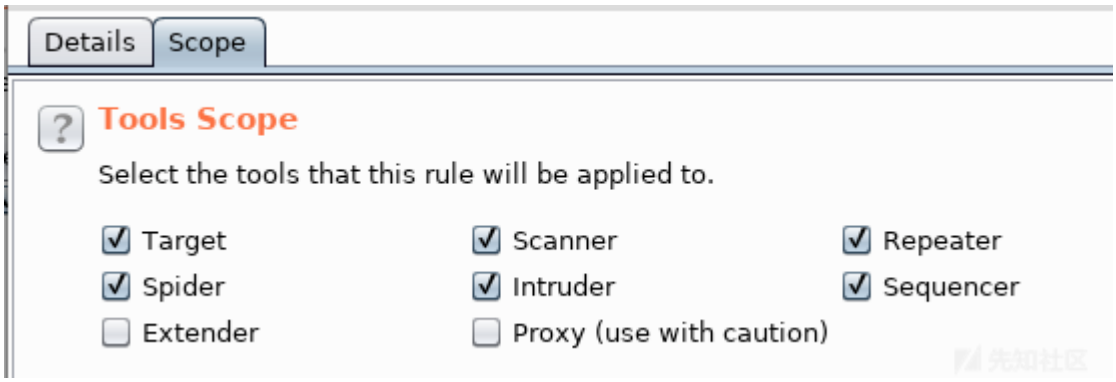
Match type:
- (•) Literal string
- ( ) Regular expression

Case-sensitivity:
- ( ) Sensitive
- (•) Insensitive

Match indicates:
- (•) Invalid session
- ( ) Valid session

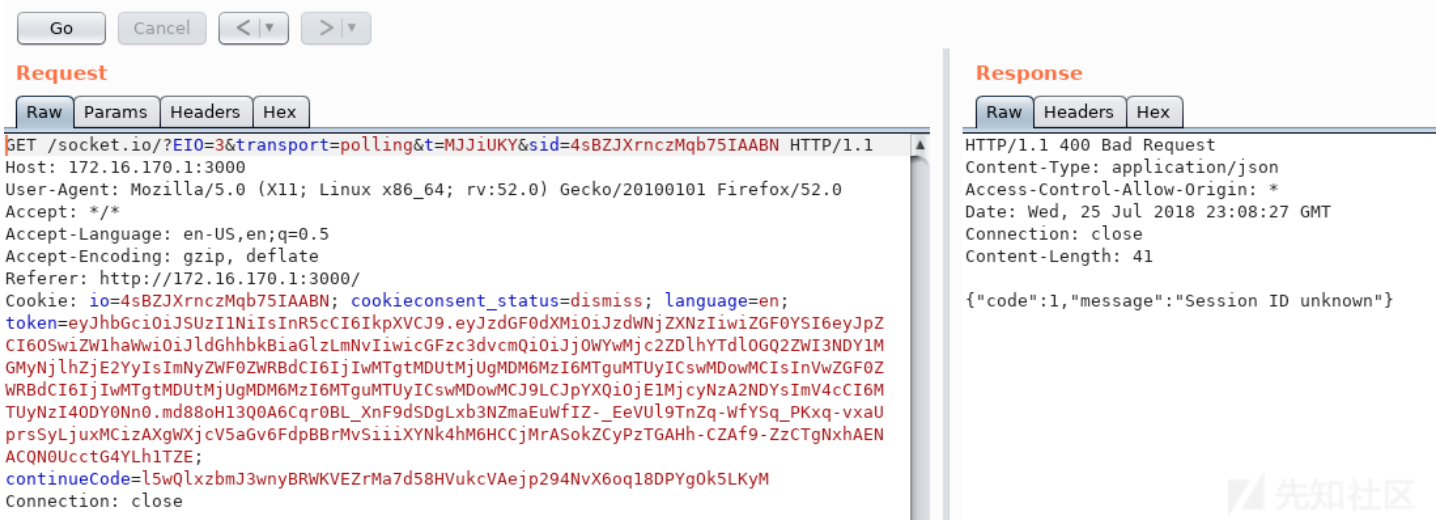


最后，在弄好新的规则动作后，还要修改规则的应用范围，也就是说，处决定要将该规则应用到哪些地方。我建议至少将其应用于Repeater上面，这样，我们就可以手动方

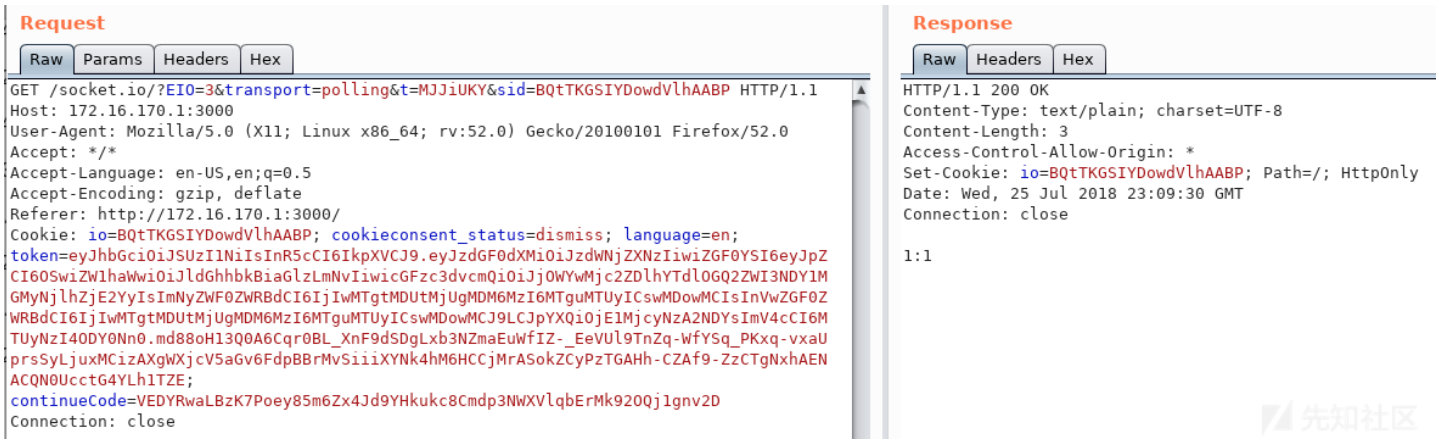


我的规则应用范围的配置如下所示。当然，您也可以让应用范围更加具体，但下面的选项应该能够适用于大多数情况。

这里是在没有会话处理规则的情况下发出的请求。



这里是在会话处理规则生效后发出的同一请求。请注意，会话处理规则"透明地"更新了cookie以及请求中"sid"参数的值。



结束语

最后说明一点，在处理上面讨论的第2个问题的时候，由于无法使用Burp的Scanner和Intruder插件，于是，我修改了一个现有的Burp插件，这似乎可以完成这项工作，不过

上一篇：XML外部实体注入小结 下一篇：一个传真接管你的网络：含有传真功能...

1. 0 条回复
   - 动动手指，沙发就是你的了！