# 确保设备与物联网平台之间的通信

## 介绍

随着物联网（IoT）逐渐成为许多行业的热门话题，越来越多的设备进入市场，这些设备连接起来并在众多用户之间共享。 在这个快速增长的空间中，成千上万的东西正在连接。 因此，很容易忽视正在连接的设备的安全方面。 随着对物联网僵尸网络新兴的DDoS攻击，组织选择具有适当身份和安全功能的物联网解决方案非常重要。 我们来看看WSO2物联网服务器（一种开源的物联网平台）如何满足这个互联世界中的安全需求。

## WSO2物联网平台

WSO2物联网平台是一款企业级开源平台，能够管理移动和物联网设备。 WSO2 IoT平台继承了WSO2产品的卓越API管理，身份管理和分析功能，其中IoT为设备增加了设备管理功能。 在物联网平台中，设备管理是通过两个重要实体之间的通信完成的; 即物联网服务器和物联网代理。

### WSO2 IoT服务器

IoT服务器提供一套完整的REST API来成功管理设备，设备制造商可以编写自己的设备类型并将这些设备类型插入IoT服务器以管理这些设备。 这些设备将与IoT服务器进行通信以获取自己的信息

### WSO2 IoT代理

对于所有这些通信的发生，设备中必须有一个知道如何联系服务器，注册，发送设备传感器数据并处理服务器发送的任何命令的实体。 这由安装在设备上的IoT代理处理。 基于设备平台，可以使用任何适当的编程语言编写代理程序，代理程序将使用IoT服务器公开的设备管理API。 让我们仔细研究物联网服务器中发生的不同类型的通信

## 联通模式

在WSO2 IoT Server中发生的所有通信都可以分解为两个主要区域，如图1所示：

1. 设备到服务器
2. 服务器到应用程序或外部系统

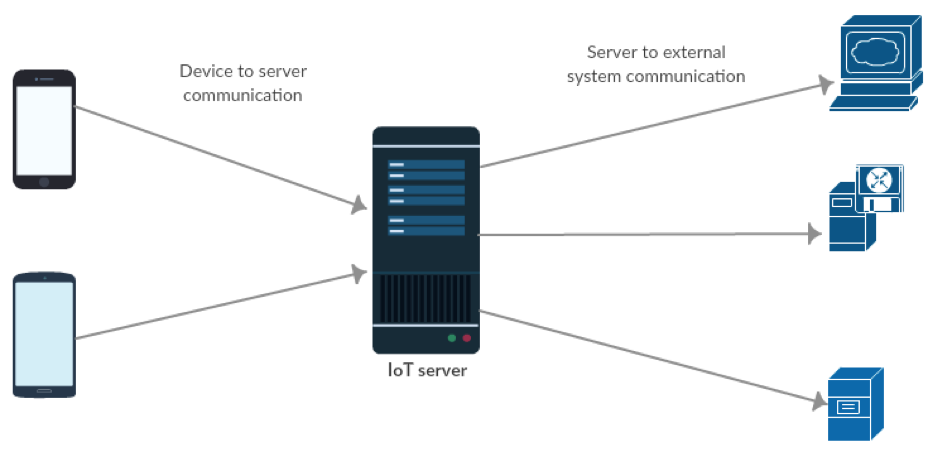


图1

WSO2 IoT服务器在必要时与外部系统（如LDAP或数据库）通信，并且安全地完成此通信。 但是，本文的重点是讨论可用于保护设备与服务器之间通信的安全机制。 因此，让我们来看看WSO2 IoT服务器如何处理设备 - 服务器通信并以安全的方式进行处理。

### 设备和服务器通信

物联网的核心在于设备或物联网平台的连通性和连通性。设备连接到WSO2 IoT服务器以实时发送新事件或检查设备上需要执行的操作和策略。

今天，资源越来越多地被暴露为API，随着这些API的兴起，窃取企业数据和滥用资源的企图正在增加。在物联网领域，这些设备需要不断或定期与服务器进行通信，而这些连接通常是通过公共网络使用诸如MQTT，HTTP或XMPP等协议完成的。无论通信方式如何，它们都必须具有额外的安全保护层，以保护公司数据被盗。

由于设备到服务器通信发生在公共网络上，因此保护用于传输数据的通道至关重要。为了确保通道安全，在设备和服务器之间使用SSL / TLS非常重要，并确保只有预期的各方才能读取正在传输的消息。除了传输层的基本保护之外，服务器和设备必须相互识别和验证以确保只有授权设备/用户才能访问受保护资源。这是适当的身份验证发挥作用的地方。在本文中，我们将看看两种认证形式：

1. 基于令牌的认证
2. 基于证书的认证

#### 基于令牌的身份验证

令牌可以被认为是由认证服务器创建的字符串，可用于识别设备及其访问受保护资源的权限。通常，在该流程中，设备可以发送一些秘密凭证，该凭证将由服务器识别，并且在验证凭证之后，设备可以被赋予可用于访问受保护资源的令牌。一种这样的行业标准协议是OAuth。

OAuth是基于令牌的授权标准，通常与认证手段一起使用以确保资源安全。访问OAuth服务器进行授权的任何请求都必须先由身份验证服务器进行身份验证。如果用户被认证并被授权，则发布令牌。这允许设备根据给予令牌的许可与API进行通信。

访问受保护的资源时，授权提出请求的设备很重要。 OAuth有利于这个确切的要求。 OAuth的另一个重要特性是其简单的设计，使得低功率物联网设备能够使用最低处理能力和资源进行身份验证和授权。 WSO2 API管理器保护设备用于通信的后端API。一些由IoT服务器管理的设备依靠OAuth与后端进行安全通信。 WSO2 IoT Server能够管理开箱即用的Android设备，并且可以使用名为设备管理代理的代理来完成。该代理将从IoT服务器接收要在设备上执行的命令。设备监控的另一种形式是通过使用名为Android感应代理的代理完成的，该代理监控Android设备的传感器，并将这些值报告给IoT服务器，以便分析这些值并在设备上执行任何操作。在这两种代理中，保护设备和服务器之间的通信至关重要，因为它们包含必须保持安全的公司数据。这是OAuth来救援的地方。我们来看看Android设备如何使用OAuth安全地与IoT服务器进行通信，如图2所示。

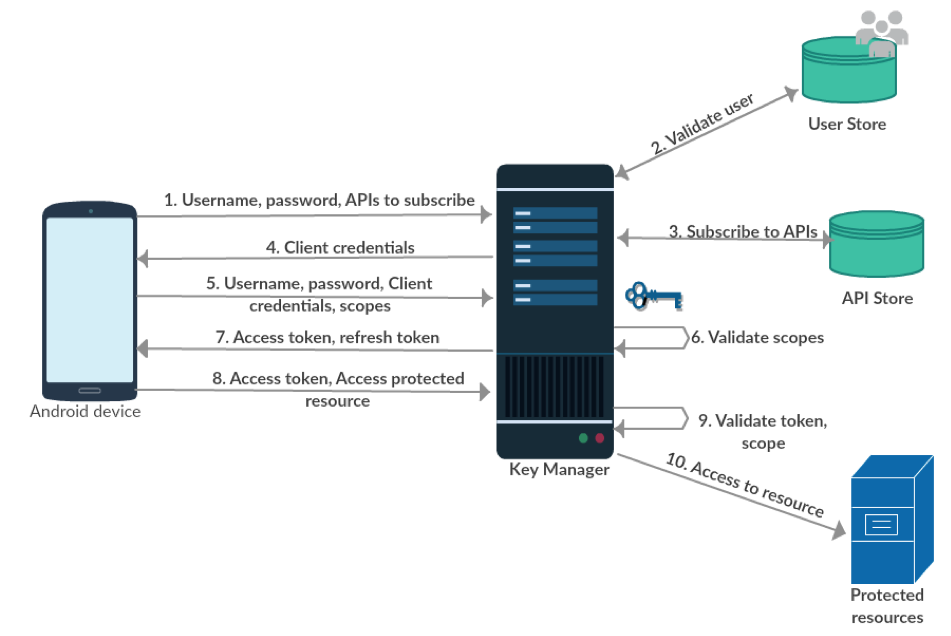


图2

1. 首先，设备需要有一个特殊的证书集[3]，称为客户端证书来生成OAuth令牌。为此，设备首先联系动态客户端注册[1] API以及它需要订阅的API列表。
2. 在API密钥管理器中，用户凭证通过连接到相关用户存储进行验证。
3. 如果用户证书有效，则用户通过调用API存储来订阅所请求的API。
4. 用户订阅后，将生成客户端凭据并将其发送回设备。
5. 接下来，该设备连同范围[6]，客户端凭证和用户凭证一起发出请求以获取访问令牌。
6. 密钥管理器根据定义的用户存储验证登录凭证。除了验证用户凭证之外，所请求的范围也被验证。每个用户都有预定义的角色及其关联的权限。这些权限与API范围相对应，当用户使用一组范围请求令牌时，密钥管理器会检查用户是否具有所请求范围的必要权限。
7. 令牌对发送到包含访问令牌的设备，以访问允许的API以及刷新令牌，以在访问令牌到期时刷新访问令牌。
8. 一旦设备具有访问令牌，它就可以调用可由步骤5和6中请求的范围访问的受保护的API。
9. 当访问API的请求与访问令牌一起被接收时，密钥管理器将检查访问令牌是否有效，并且还将检查访问令牌是否具有访问所请求的API的相关许可。如果是这样，访问如果提供。

上面的用例展示了一个简单而安全的OAuth实现，用于保护后端API和资源。 由于动态客户端证书生成[1]，消除了在注册之前将客户端凭证存储在设备中的需求。 由于每台设备都有一个单独的客户端凭证，而不是共享相同的客户端凭证，即使一台设备的客户端凭证受到损害，其余设备也不会受到影响。

#### 基于证书的身份验证

与使用令牌进行身份验证类似，可以在身份验证流程中使用证书来唯一标识设备和服务器。在通信中使用SSL时，设备可以识别和验证服务器。基于证书的身份验证的主要目标是为每个设备创建一个证书，并在服务器中使用它来识别和验证每个设备。因此，使用SSL和基于证书的身份验证，可以创建相互身份验证，其中服务器可以使用设备身份证书来识别设备，并且设备可以使用SSL证书来识别服务器。这提供了一个非常安全的通信通道，非常适合企业数据通信。实施基于证书的认证的最常见方式之一是通过使用SCEP协议;让我们考虑它是什么以及它为什么重要。

简单证书注册协议（SCEP）[2]是一种功能强大的协议，允许设备即时生成设备身份证书。首先，设备使用认证服务器进行认证。一旦通过身份验证，服务器允许设备发送证书签名请求（CSR），该请求将由服务器的CA证书签名。这允许服务器信任设备的证书，因为它是由设备签名的，并且设备将信任服务器，因为服务器证书已预先安装在设备上。 SCEP适用于物联网设备，因为设备身份可以随时为每个设备动态生成。由于服务器和设备之间存在相互信任，所以通信通道变得非常安全。

WSO2 IoT Server也支持SCEP，因此便于基于证书的身份验证。平台（如iOS）主动依靠WSO2 IoT Server中的SCEP功能来注册和验证身份。我们来看看SCEP是如何工作的以及在IoT Server中如何实施SCEP，特别是在iOS设备管理的情况下。 SCEP成立以解决向设备（如路由器）提供设备特定证书的可扩展性问题。该协议的最终目标是将设备特定证书传递给可用于验证和授权用户的设备。让我们来看看SCEP沟通的流程以获得更好的理解（图3）。

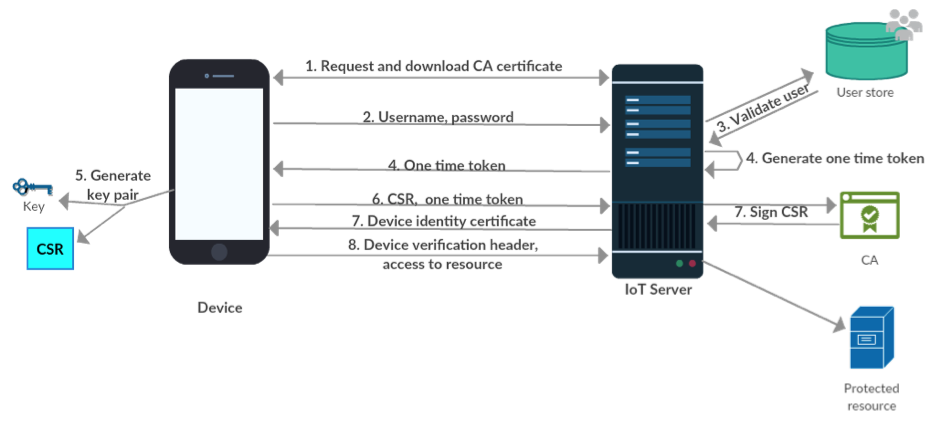


图3

1. 注册阶段的设备需要下载并安装IoT Server的自签名根CA证书。
2. 用户首先在设备上输入用户凭证（用户名和密码），然后将其传递给WSO2 IoT Server。
3. WSO2 IoT Server验证用户凭证。
4. 如果凭证有效，服务器将生成一次性令牌并将返回给设备。服务器期望令牌在将来与服务器进行通信时使用。
5. 该设备生成一个私钥和相应的证书签名请求（CSR）。
6. 设备将CSR与一次性令牌一起传递给服务器。
7. WSO2 IoT Server使用内置CA服务器设施签署CSR并向设备颁发证书。请注意，除了颁发证书之外，服务器还会存储针对其所属用户生成的证书以备将来参考。
8. 此时，设备具有IoT服务器信任的密钥对（公共和私有），并且由于在步骤1中CA证书安装在设备上，因此设备也可以信任服务器。在此之后，设备可以发送一个验证头，并使用其私钥签名，验证头可用于验证服务器以识别设备。

基于以上解释，在该过程结束时，设备具有身份证书以向服务器信任的loT服务器标识自身。 与服务器通信时，设备身份证书作为标头发送以标识设备，而消息内容通常由设备标识的私钥签名。 由于服务器具有相应的公钥，因此服务器可以验证设备是否实际签署了内容。 现在该设备可以安全地调用受保护的API。

##### SCEP的最佳实践

需要注意的是，由于SCEP被视为未经身份验证的协议，因此在签署证书时不会进行其他身份验证。 为了克服这个问题，提供了一个额外的安全层，当提交CSR时，用户凭证首先传递给服务器并进行验证。 服务器将首先验证用户证书是否有效，如果是，则证书将被签署并颁发。 为了跟踪请求，发出挑战令牌的证书也将与证书相关联，证书将在请求发送时进行验证。

除OAuth和SCEP外，WSO2 IoT Server还支持基本身份验证和开箱即用的相互TLS身份验证。 由于认证逻辑的可扩展性，组织可以建立自己的认证方案来保护与服务器的通信。

### 添加您的自定义安全方案

在某些情况下，开箱验证机制不足以满足自定义要求。在这种情况下，物联网服务器为您自己的身份验证方法中的插件提供足够的可扩展性很重要。 WSO2 IoT Server提供了这种可扩展性，这让您可以编写自己的逻辑，使您可以连接到外部系统，以对输入到服务器的身份验证请求进行身份验证。例如，让我们设想一种情况，即我们有一个定制设备，该设备始终将身份验证标头值附加到发送给IoT服务器的请求。为了验证设备的真实性，我们需要将此自定义标头值发送给设备制造商提供的Web服务。在这种情况下，我们可以编写一个扩展，它将位于IoT服务器和设备之间，并通过与外部系统通信进行标题验证。我们来看看如何编写自定义身份验证器并将其插入IoT服务器。

#### 如何插入你自己的认证方案？

设备向IoT服务器后端发出的任何请求都首先通过API网关。 在API网关中，有多个执行不同任务的处理程序[5]，并且这些处理程序在请求流经时依次逐一执行。 其中一个处理程序是在默认消息流下执行OAuth身份验证的APIAuthenticationHandler处理程序。 在编写自定义处理程序之后，我们需要用我们的自定义处理程序替换APIAuthenticationHandler，以便我们的身份验证逻辑在其他核心API管理任务（比如API调节和使用统计信息收集）之前发生。 图4展示了自定义处理程序之前的处理程序之间的流程。

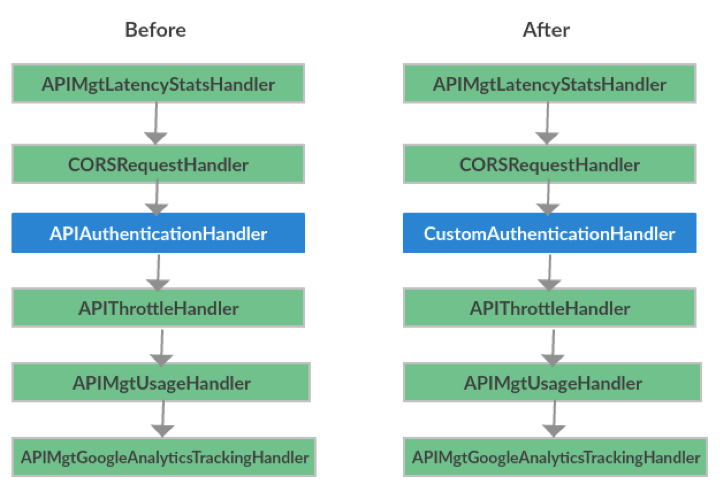


图4

现在我们已经知道处理程序之间的消息是如何流动的，让我们看看整个流程如何，例如，验证自定义标头以验证设备时（图5）。

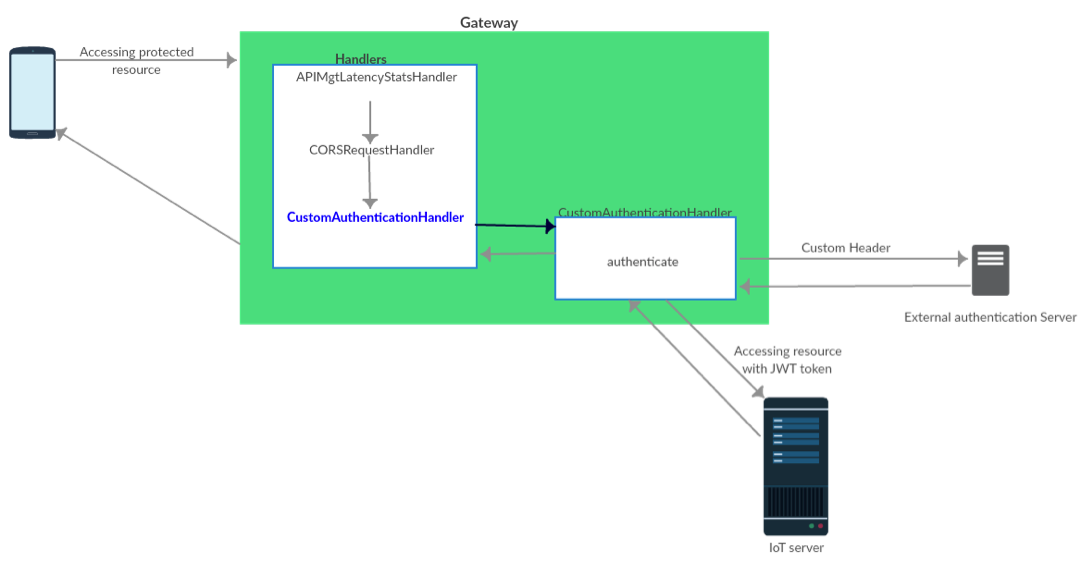


图5

1. 设备向服务器发出请求以访问安全资源，并且它将首先进入API网关。发送此请求时，设备可以添加需要验证的自定义标头以验证设备。
2. API网关包含我们编写的自定义处理程序，其中包含用于验证设备的自定义逻辑。首先，处理程序需要与需要保护的API进行交互。
3. 使用时，处理程序中的自定义逻辑将验证自定义标题。例如，处理程序可以将此设备发送的自定义报头值转发给外部系统并验证设备。
4. 在此之后，刚刚通过身份验证的用户的详细信息需要传递给IoT Server后端，以便正确处理原始请求。为此，我们将创建一个JWT令牌。
5. 将原始请求与刚刚创建的JWT令牌一起传递给WSO2 IoT Server。
6. WSO2 IoT Server验证JWT令牌并提供对原始请求的访问。

您可以在这里找到自定义身份验证处理程序的示例实现[4]。 让我们按照上面的步骤通过这个示例来更清楚地理解代码。 请注意，在此项目中，我们的自定义处理程序逻辑写入CustomAuthenticationHandler类中，其包名称为org.wso2.carbon.authenticator。

1. 如我们的第一步所述，设备向服务器发送一个自定义报头值，并由网关拦截。
2. 将自定义逻辑（自定义处理程序）放入网关中，并在要保护的API中使用处理程序。 为此，首先在/ repository / deployment / server / synapse-configs / default / api目录中找到与要修改的API相关的API定义，转到handles标记，插入<handler class =“org .wso2.carbon.authenticator.CustomAuthenticationHandler“/>标记，并移除<handler class =”org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler“/>处理程序。
3. 当自定义处理程序使用API时，将执行自定义逻辑。 我们来看一下自定义处理程序的结构。 每当请求进入处理程序时，都会执行handleRequest方法。 此方法获取MessageContext对象，其中包含设备的请求详细信息。 在这个例子中，我们提取标题值，如下所示：

public boolean handleRequest(MessageContext messageContext) {

if (log.isDebugEnabled()) {

log.debug("CustomAuthenticationHandler handleRequest

called");

}

return authenticate(messageContext);

}

private boolean authenticate(MessageContext synCtx) {

Map headers = getTransportHeaders(synCtx);

String authHeader = (String) headers.get(CUSTOM\_AUTHENTICATION\_HEADER);

// These values must be passed by the user and these variables has to be populated.

String tenantDomain = "carbon.super";// This can be the teant you created for example marketing or sales

String username = "admin";

// Fill here with the custom authentication logic as you wish. For example call an external system to

// validate a value that comes in the CUSTOM\_AUTHENTICATION\_HEADER. After authenticating, the information

// about the user needs to be passed to the backend, therefore a JWT token is generated.

//if (authHeader.startsWith("custom\_authenticated\_value")) {

try {

headers.put(X\_JWT\_ASSERTION, AuthenticationUtils.createJWTToken(username, tenantDomain));

return true;

} catch (CustomAuthenticatorException e) {

log.error("Error while attempting to authenticate the request.", e);

return false;

}

//}

//return false;

}

从请求中提取必要的细节后，您可以根据需要验证发送的详细信息。

验证成功后，您可以返回此方法的结果。 另外，验证成功后，您需要以JWT令牌的形式向请求添加有关已验证用户的详细信息，以便IoT服务器获得有关用户的详细信息，并知道该请求已通过验证。 这是通过调用AuthenticationUtils.createJWTToken实用程序的方法完成的，它需要将用户名和租户域传递给它。 由于util类中的代码不需要修改，因此我们不会关注这一点。

1. WSO2 IoT Server现在可以处理请求并提供对所需资源的访问。

#### 如何支持多因素认证？

可以将多因素身份验证视为可与现有身份验证机制相结合的附加安全层。多因素的重要性在于，如果一个因素受到影响，附加层将保证安全资源不会受到影响。登录到不同的服务时，通常使用多因素身份验证，例如用户必须提供用户凭据的电子邮件。此外，他们可以选择通过将他们的移动号码与服务器将一次性令牌发送到用户的电话的系统相结合来获得额外的安全层，并且用户必须输入该令牌来完成认证。

我们还可以通过前面提到的自定义身份验证处理程序实现多因素身份验证。在这里，客户端首先发送第一组凭据以验证用户身份并进行验证。因此，可以写入自定义逻辑来生成并存储一次性令牌。令牌通过设备（例如SMS网关）传递给设备。设备然后可以将此令牌转发回服务器，并且自定义处理程序根据其存储的值验证令牌。由于认证机制的可扩展性，您可以根据需要实施自定义逻辑并与任何外部系统进行链接。

## 概要

总结讨论，很明显WSO2 IoT Server具有强大的物联网安全机制，可以保护端到端的通信，并且允许组织通过提供的扩展点来定制他们的功能。 这些功能使组织能够在这个快速增长的行业中拥有非常安全和可扩展的物联网基础架构。

## 参考

[1] <https://tools.ietf.org/html/rfc7591>

[2] <https://www.ietf.org/id/draft-gutmann-scep-06.txt>

[3] <https://tools.ietf.org/html/rfc6749#section-2>

[4] <https://github.com/inoshperera/custom-authentication-handler/>

[5] <https://docs.wso2.com/display/AM200/Writing+Custom+Handlers#WritingCustomHandlers-Introducinghandlers>

[6] <https://docs.wso2.com/display/IoTS310/Device+Management+API+Scopes>