[VolantMQ/volantmq [891 stars]](https://github.com/VolantMQ/volantmq)

注：我们已向厂商通报此安全问题及修复建议

#### 0x01 攻击场景与测试

考虑IoT应用的共享场景，即智能家居系统使用 MQTT 协议进行物联网设备和用户管理，其中有两个用户角色。管理员，也就是房主  
可以授权其他普通用户（例如，Airbnb 客人）访问他的智能家居设备的权利。普通用户的访问权限可能会被撤销和到期。我们  
认为管理员和设备是良性的，而客人可能是恶意的，会尽可能地去试图未授权访问设备（越权或是维持被撤销的权限）。

* **攻击场景**

首先，攻击者登记入住，因此目前攻击者拥有主题“A”的“写权限”，可以暂时控制受害设备。受害设备订阅此主题。  
  
  
1. 设备使用“CleanStart = False”（一个持久化session状态而常常使用的CONNECT flag）连接broker并订阅主题“A”。  
  
2. 攻击者通过关闭路由器等方式，使得设备持续掉线。  
  
3. 攻击者随后向主题“A”发布“QoS 1/2 消息”（该消息会被保存在设备session的队列中）。  
  
4. 攻击者的权限被管理员或设备所有者撤销。  
  
5. 设备重新连接“Clean Start = False”（例如下一个租户打开路由器或设备），然后它会收到攻击者发布的有害消息。

* **漏洞危害**

首先我们的恶意消息可以是一条延时执行的指令（例如在xx时间打开门锁），所以不容易被下一任租客发现；其次攻击的目标可以远离路由器（例如，配套的车库大门门锁），也难以被受害租客发现。相当于给设备留下了一个定时炸弹。

#### 0x02 漏洞测试步骤

* **测试环境**

**VolantMQ**: 0.4.0

**mqtt client**: 任意客户端即可 (paho.mqtt)

**访问控制插件**: 官方插件[http auth]([VolantMQ / vlplugin / Auth / http · GitLab](https://gitlab.com/VolantMQ/vlplugin/auth/http))（由于golang更新已不再支持plugin模块，因此这个插件目前无法使用），也可修改VolantMQ内置的auth测试插件 (见附录auth.go, 替换cmd/volantmq/auth.go)，由于漏洞的原理为broker的permission check位置不当 (或没有进行足够的检查)，而无关于permission check本身的正确与否，因此无论权限检查插件使用何种机制 (使用http请求授权服务器、使用database存储ACL等)，漏洞本身都是存在的。

配置测试用户：

admin: 拥有所有权限

user1(attacker): 拥有publish权限

若使用[http auth]([VolantMQ / vlplugin / Auth / http · GitLab](https://gitlab.com/VolantMQ/vlplugin/auth/http))或是附录中的auth.go，则仅需简单写一个http服务 (见附录app.py)，在broker请求/acl页面获取用户是否拥有进行敏感操作的权限时，回复"allow" (代表拥有权限)/"xxxxx"即可。

from flask import Flask, request, render\_template, session, jsonify  
from flask\_cors import CORS, cross\_origin  
import json  
import time as mytime  
from datetime import \*  
  
app = Flask(\_\_name\_\_)  
cors = CORS(app)  
  
  
@app.route('/acl', methods=['GET'])  
def Start():  
 user = request.args.get('user')  
 resp = "deny"  
 if(user == "admin"):  
 resp = "allow"  
 elif(user == "user1"):  
 resp = "allow"  
 return resp  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(host='0.0.0.0', debug=True, port=80)

* **测试步骤**

1. 设备使用“CleanStart = False”（一个持久化session状态而常常使用的CONNECT flag）连接broker并订阅主题“test”。

$ mosquitto\_sub -u admin -P admin -t "test" -c

1. 攻击者通过关闭路由器等方式，使得设备持续掉线(这里可以使用断开客户端与broker之间的连接进行模拟)。
2. 攻击者随后向主题“A”发布“QoS 1/2 消息”（该消息会被保存在设备session的队列中）。

$ mosquitto\_pub -u user1 -P pass1 -t "test" -m "bad" -q 2

1. 攻击者的权限被管理员或设备所有者撤销。

若使用auth.go进行访问控制，则可手动控制auth server的访问控制配置来进行测试，例如当撤销attacker全新啊时，修改web服务代码app.py中的回复为deny：

from flask import Flask, request, render\_template, session, jsonify  
from flask\_cors import CORS, cross\_origin  
import json  
import time as mytime  
from datetime import \*  
  
app = Flask(\_\_name\_\_)  
cors = CORS(app)  
  
  
@app.route('/acl', methods=['GET'])  
def Start():  
 user = request.args.get('user')  
 resp = "deny"  
 if(user == "admin"):  
 resp = "allow"  
 elif(user == "user1"):  
 resp = "deny"  
 return resp

1. 设备重新连接“Clean Start = False”（例如下一个租户打开路由器或设备），然后它会收到攻击者发布的有害消息。

$ mosquitto\_sub -u admin-user -P admin-password -t "test" -c

#### 0x03 漏洞原理分析

1. 当broker向订阅者投递普通消息/retained message时，即没有检查发布者权限（仅在收到PUBLISH报文时检查，在收到PUBREL报文开始投递时没有检查），也没有检查订阅者是否拥有接收消息的权限

connection\session.go: 93

// SignalPublish process PUBLISH packet from client  
func (s \*session) SignalPublish(pkt \*mqttp.Publish) error {  
 pkt.SetPublishID(s.subscriber.Hash())  
  
 // [MQTT-3.3.1.3]  
 if pkt.Retain() {  
 if err := s.messenger.Retain(pkt); err != nil {  
 s.log.Error("Error retaining message", zap.String("clientId", s.id), zap.Error(err))  
 }  
 }  
  
 if err := s.messenger.Publish(pkt); err != nil {  
 s.log.Error("Couldn't publish", zap.String("clientId", s.id), zap.Error(err))  
 }  
  
 return nil  
}

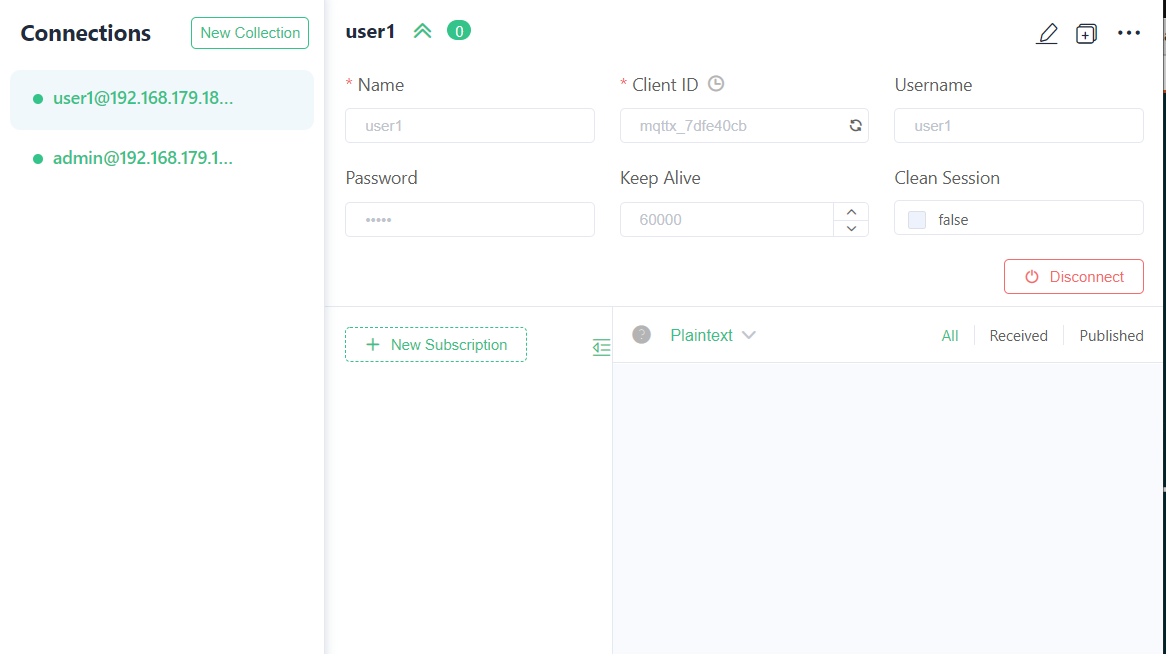
#### 0x04 漏洞效果

##### 测试前配置

测试用的账号：admin和user1

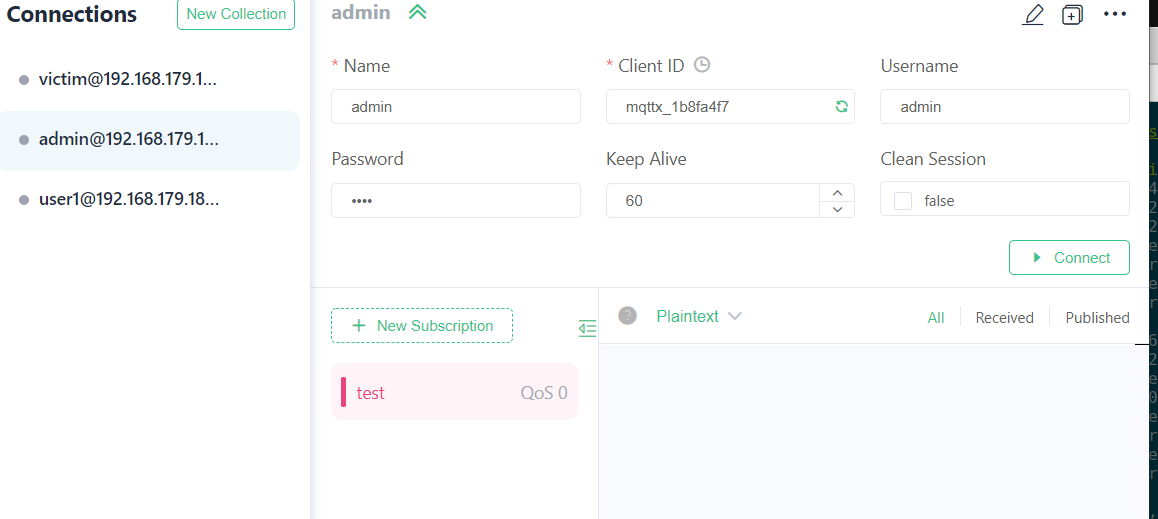
目前user1拥有test topic的发布权限

#app.py  
@app.route('/acl', methods=['GET'])  
def Start():  
 user = request.args.get('user')  
 resp = "deny"  
 if(user == "admin"):  
 resp = "allow"  
 elif(user == "user1"):  
 resp = "allow"  
 return resp

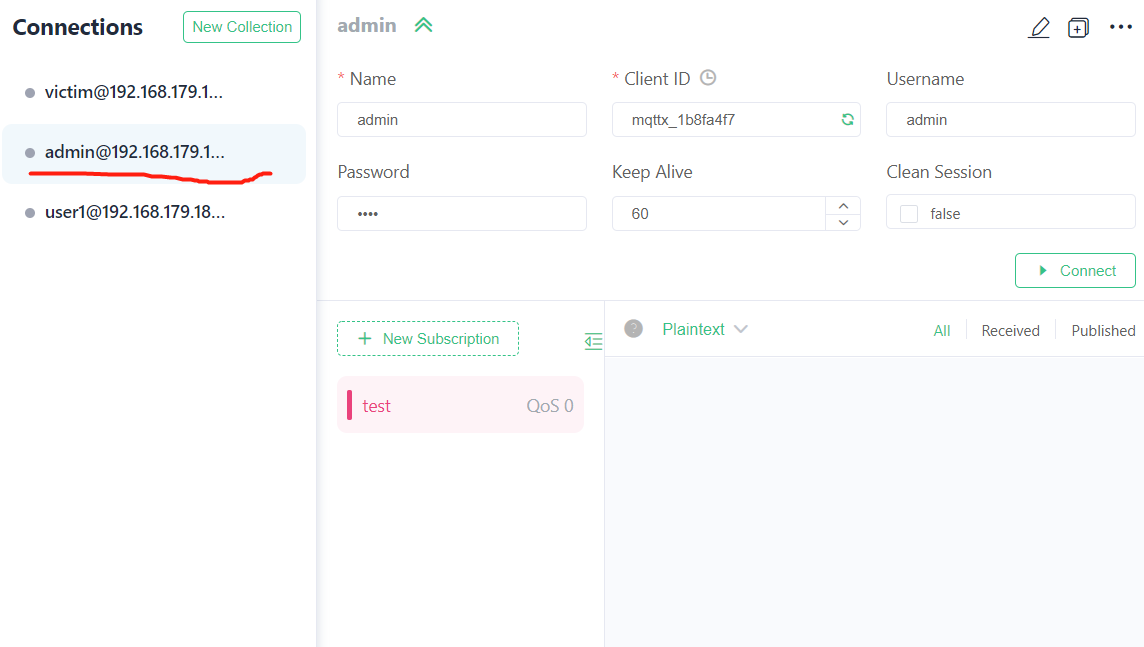


##### 测试流程

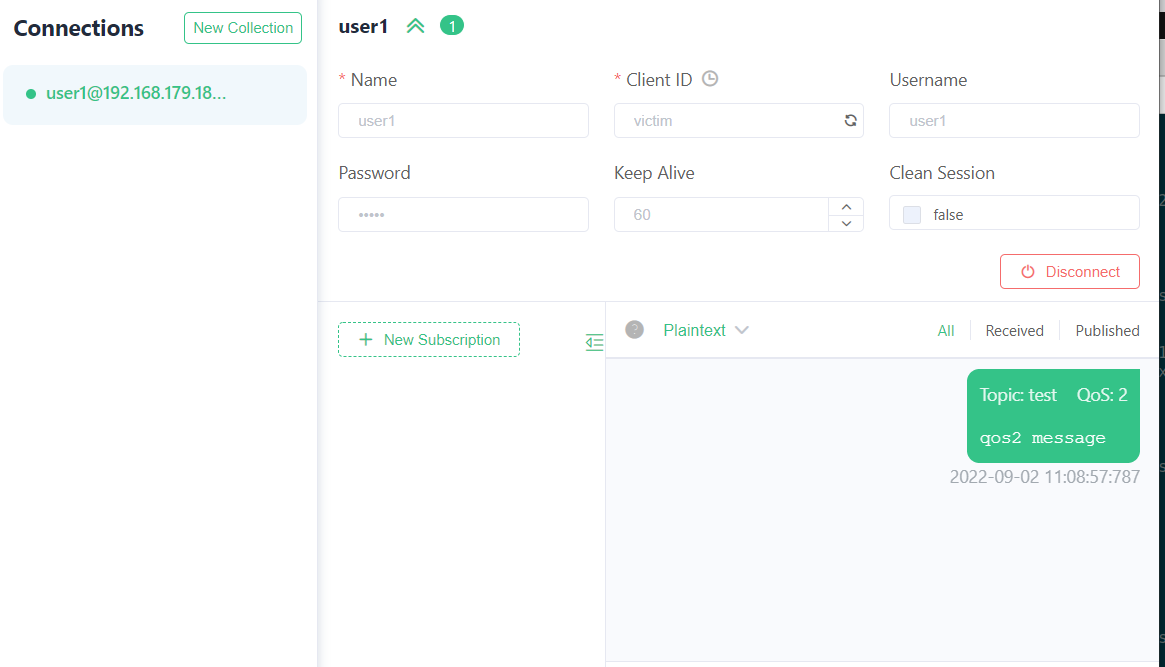
1. 使用MQTTX客户端，设备(为了测试方便使用admin账号)使用“CleanStart = False”（一个持久化session状态而常常使用的CONNECT flag）连接broker并订阅主题“test”。



1. 攻击者通过关闭路由器等方式，使得设备持续掉线(这里可以使用断开客户端与broker之间的连接进行模拟)。



1. 攻击者随后向主题“A”发布“QoS 1/2 消息”（该消息会被保存在设备session的队列中）。



1. 攻击者的权限被管理员或设备所有者撤销。

若使用auth.go进行访问控制，则可手动控制auth server的访问控制配置来进行测试，例如当撤销attacker全新啊时，修改web服务代码app.py中的回复为deny：

from flask import Flask, request, render\_template, session, jsonify  
from flask\_cors import CORS, cross\_origin  
import json  
import time as mytime  
from datetime import \*  
  
app = Flask(\_\_name\_\_)  
cors = CORS(app)  
  
  
@app.route('/acl', methods=['GET'])  
def Start():  
 user = request.args.get('user')  
 resp = "deny"  
 if(user == "admin"):  
 resp = "allow"  
 elif(user == "user1"):  
 resp = "deny"  
 return resp

1. 设备重新连接“Clean Start = False”（例如下一个租户打开路由器或设备），然后它会收到攻击者发布的有害消息。

