Don't Call Us We'll Call You: McAfee ATR Finds Vulnerability in Agora Video SDK

Douglas McKee | FEB 17, 2021 | 16 MIN READ

The McAfee Advanced Threat Research (ATR) team is committed to uncovering security issues in both software and hardware to help developers provide safer products for businesses and consumers. We recently investigated and published several findings on a personal robot called "temi", which can be read about in detail here. A byproduct of our robotic research was a deeper dive into a video calling software development kit (SDK) created by *Agora.io*. Agora's SDKs are used for voice and video communication in applications across multiple platforms. Several of the most popular mobile applications utilizing the vulnerable SDK included social apps such as eHarmony, Plenty of Fish, MeetMe and Skout, and healthcare apps such as Talkspace, Practo and Dr. First's Backline. In early 2020, our research into the Agora Video SDK led to the discovery of sensitive information sent unencrypted over the network. This flaw, CVE-2020-25605, may have allowed an attacker to spy on ongoing private video and audio calls. At the time of writing, McAfee is unaware of any instances of this vulnerability being exploited in the wild. We reported this research to *Agora.io* on April 20, 2020 and the company, as of December 17th, 2020 released a new SDK, version 3.2.1, which mitigated the vulnerability and eliminated the corresponding threat to users.

Encryption has increasingly become the new standard for communication; often even in cases where data privacy is not explicitly sensitive. For example, all modern browsers have begun to migrate to newer standards (HTTP/2) which enforce encryption by default, a complete change from just a few years ago where a significant amount of browsing traffic was sent in clear text and could be viewed by any interested party. While the need to protect truly sensitive information such as financial data, health records, and other personally identifiable information (PII) has long been standardized, consumers are increasingly expecting privacy and encryption for all web traffic and applications. Furthermore, when encryption is an option provided by a vendor, it must be easy for developers to implement, adequately protect all session information including setup and teardown, and still meet the developers' many use cases. These core concepts are what led us to the findings discussed in this blog.

To boldly go where no one has gone before

As part of our analysis of the temi ecosystem, the team reviewed the Android application that pairs with the temi robot. During this analysis, a hardcoded key was discovered in the app.

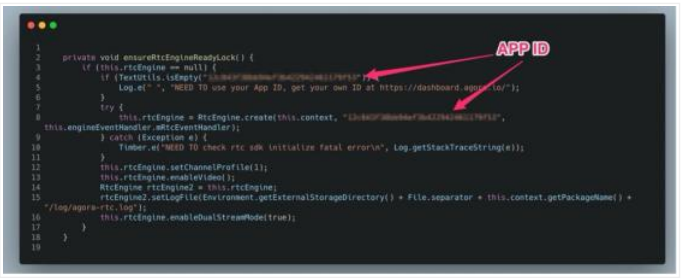


Figure 1: Application ID hardcoded in temi phone app

This raised the question: What is this key and what is it used for? Thanks to the detailed logging provided by the developers, we had a place to start, https://dashboard.agora.io.

What is Agora?

According to the website – "Agora provides the SDKs and building blocks to enable a wide range of real-time engagement possibilities" In the context of our initial robot project, it simply provides the technology required to make audio and video calls. In a broader context, Agora is used for a variety of applications including social, retail, gaming and education, among others.

Agora allows anyone to create an account and download its SDKs for testing from its website, which also provides extensive documentation. Its GitHub repositories also provide detailed sample projects on how to use the product. This is amazing for developers, but also very useful to security researchers and hackers. Using the logging comments from the above code we can look at the documentation to understand what an App ID is and what it is used for.


App ID

After signing up at Console, multiple projects can be created. Each project will be assigned a unique App ID. Anyone with your App ID can use it on any Agora SDK. Hence, it is prudent to safeguard the App IDs.

Figure 2: Agora documentation about App ID

The last two sentences of this documentation really grabbed our attention. We had found a key which is hardcoded into an Android application that "anyone can use on any Agora SDK" and is "prudent to safeguard".

Agora provides several different SDKs with different functionality. Since we encountered Agora through use of the video SDK, we decided to focus on only this SDK for the rest of this research. Simulating the mindset of an attacker, we began to investigate what this App ID or key could be used for. Furthermore, in the context of the video SDK, the question evolved into whether an attacker could interact with this video and audio traffic.

"They've done studies, you know. 60% of the time, encrypting works every time."

Since Agora provides sample projects and allows for free developer accounts, the best way to understand what potential attack vectors exists is to use these tools. Examining the GitHub example projects and the following associated documentation, we can learn exactly what is needed and how a normal user is connected to a video call.

```
    }
}
```

Figure 3: Sample project initializeEngine function

Here we see in the example code the App ID being used to create a new "RtcEngine" object. As can be seen in the documentation, creating an RtcEngine is the foundation and first step needed to create any video call.

Creates an RtcEngine instance.

Unless otherwise specified, all the methods provided by the RtcEngine class are executed asynchronously. Agora recommends calling these methods in the same thread.

Note

- You must create an RtcEngine instance before calling any other method.
- You can create an RtcEngine instance either by calling this method or by calling create2. The difference between create2 and this method is that create2 enables you to specify the connection area.
- The Agora RTC Native SDK supports creating only one RtcEngine instance for an app for now.

Figure 4: Agora documentation on RtcEngine

If we continue to examine the example code and look at the documentation's steps for connecting to a call, we come to a function named "joinChannel".

```
private void joinChannel() {
    // 1. Users can only see each other after they join the
    // same channel successfully using the same app id.
    // 2. One token is only valid for the channel name that
    // you use to generate this token.
    String token = getString(R.string.agora_access_token);
    if (TextUtils.isEmpty(token) || TextUtils.equals(token, "#YOUR ACCESS TOKEN#")) {
        token = null; // default, no token
    }
    mRtcEngine.joinChannel(token, "demoChannel1", "Extra Optional Data", 0);
}
```

Figure 5: Agora sample program joinChannel function

This function is responsible for connecting an end user to a call. In the example code, there are four parameters, three of which are hardcoded and one of which can be set to null. Without doing too much more digging, it appears that while Agora lists the App ID as important, it is not the only component needed to join a video call. An attacker would also require the values passed to the joinChannel API in order to join a call. If we assume that these values are only hardcoded for the purpose of a demo application, how would an attacker obtain the other necessary values? Code is an awesome resource, but when it comes to a network conversation the truth is in the packets. By running this example code and capturing traffic with Wireshark, we can further our understanding of how this system works.

```
35 2.852141    192.168.5.102                    54.223.118.211
▸ Frame 35: 147 bytes on wire (1176 bits), 147 bytes captured (1176 bits) on interface 0
▸ Ethernet II, Src: BizlinkK_df:07:1c (9c:eb:e8:df:07:1c), Dst: Ubiquiti_2b:02:f9 (b4:fb:e4:2b:02:f9)
▸ Internet Protocol Version 4, Src: 192.168.5.102, Dst: 54.223.118.211
▸ User Datagram Protocol, Src Port: 55322, Dst Port: 8000
▾ Data (105 bytes)
    Data: 690000002a001413287c72010000200038303737396134...
    [Length: 105]
0000  b4 fb e4 2b 02 f9 9c eb  e8 df 07 1c 08 00 45 00   ...+.... ......E.
0010  00 85 2c ea 00 00 40 11  d9 bd c0 a8 05 66 36 df   ..,...@. .....f6.
0020  76 d3 d8 1a 1f 40 00 71  2b c1 69 00 00 00 2a 00   v....@.q +.i...*.
0030  14 13 28 7c 72 01 00 00  20 00 38 30 37 37 39 39   ..(|r... .807799
0040  61 34 30 62 36 38 34 35  64 37 61 63 63 37 35 66   a40b6845 d7acc75f
0050  65 62 30 66 61 33 34 61  66 38 0c 00 64 65 6d 6f   eb0fa34a f8..demo
0060  43 68 61 6e 6e 65 6c 31  13 00 45 78 74 72 61 20   Channel1 ..Extra
0070  4f 70 74 69 6f 6e 61 6c  20 44 61 74 61 02 00 07   Optional  Data...
0080  00 00 00 03 00 34 35 30  09 00 00 00 01 00 30 00   .....450 ......0.
0090  00 00 00                                           ...
```

Figure 6: Wireshark capture of Agora traffic

When looking at the traffic, what immediately stands out is the values passed to joinChannel in the example code above. They are sent in plaintext across the network, in addition to the App ID needed to initiate the RtcEngine. Considering this is an example app, it is important to understand the difference between a test scenario and a production scenario. As noted in the code in Figure 5, the "token" parameter is being set to null. What is a token in this context, and would that affect the security of these parameters? We can use the Agora documentation to understand that a token is designed to be randomly generated and provide more security for a session.

- `token` : Pass a token that identifies the role and privilege of the user. You can set it as one of the following values:
  - `NULL`.
  - A temporary token generated in Console. A temporary token is valid for 24 hours. For details, see Get a Temporary Token.
  - A token generated at the server. This applies to scenarios with high-security requirements. For details, see Generate a token from Your Server.

Figure 7: Agora documentation regarding tokens

With tokens being an option, it is important we see what the traffic looks like in a scenario where the token parameter is not null. The use of a token is hopefully what we find in production or at least is what is recommended in production by Agora.

Use a token for authentication

Token is a dynamic key generated by App ID, App Certificate, user ID, channel name, token expiration timestamp, and other information. For scenarios requiring high-security, such as the production environment, Agora recommends using a token for authentication.

Figure 8: Agora documentation on token authentication

Running the example application again, this time using a token and capturing traffic, we can discover that the token is also a non-issue for an attacker.

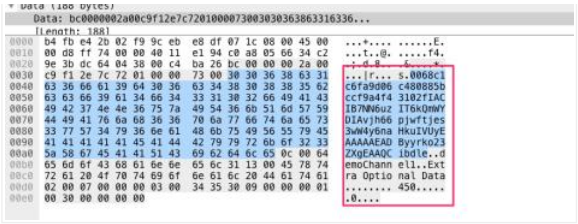


Figure 9: Wireshark capture of Agora call with tokens

The token is sent in plaintext just like the other parameters! You may have noticed this capture does not show the App ID; in this case the App ID is still sent in plaintext, in a different packet.

The Spy Who Loved Me

Information being sent in plaintext across the network to initiate a video call is one thing, but can this actually be used by an attacker to spy on a user? Will the call support a third party? Will the user be notified of a new connection? In order to answer these questions, we can use the example applications provided by Agora to run some tests. The diagram below shows the scenario we are going to try and create for testing.

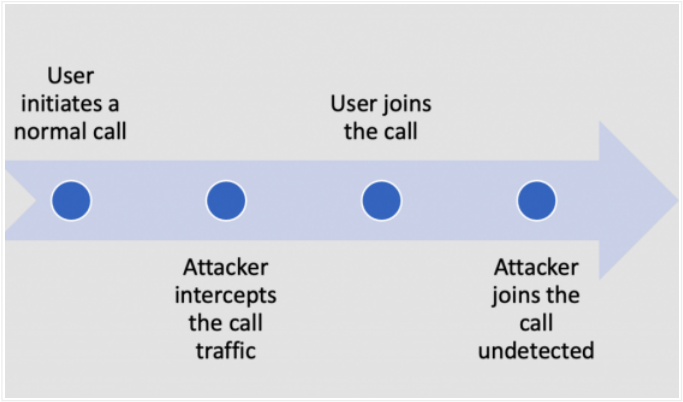
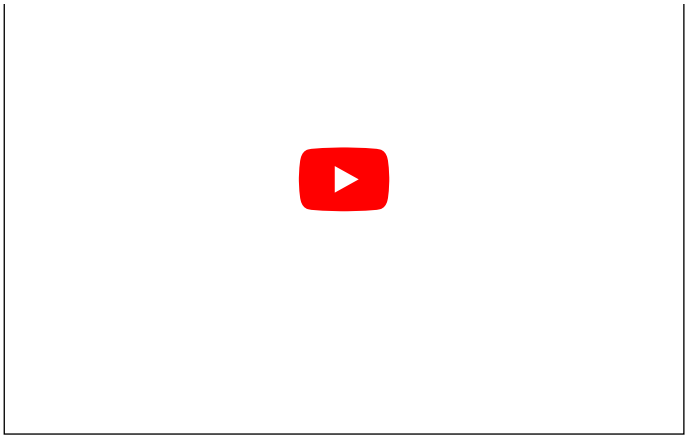


Figure 10: Agora attack scenario

The first step for an attacker sitting on the network is to be able to identify the proper network traffic containing the sensitive information. With this information from the network packets, they can then attempt to join the call that is in progress. Using a Python framework called Scapy, we built a network layer in less than 50 lines of code to help easily identify the traffic the attacker cares about. This was done by reviewing the video call traffic and reverse engineering the protocol. Like a lot of reversing, this is done by using context clues and a lot of "guess and check". Strings help us identify the use for certain fields and provide clues to what the fields around them might be. In some cases, fields are still unknown; however, this is normal. An attacker or researcher only needs to decipher enough of a packet to make an attack possible.

Figure 11: Agora Scapy filter

The above code identifies, parses, and displays in human readable form, only the important information in the three main packet types discovered in the Agora traffic. From here we can automate the scenario outlined in Figure 10 using Python with a few modifications to the example apps. It totally works! The demo video below shows an attacker sniffing network traffic to gather the call information and then launching their own Agora video application to join the call, completely unnoticed by normal users.



Besides using a token, were there any other security measures available to developers that might have mitigated the impact of this vulnerability? Per Agora's documentation, the developer has the option to encrypt a video call. We also tested this, and the App ID, Channel Name, and Token are still sent in plaintext when the call is encrypted. An attacker can still get these values; however, they cannot view the video or hear the audio of the call. Despite this, the attacker can still utilize the App ID to host their own calls at the cost of the app developer. We will discuss in the next section why, even though encryption is available, it is not widely adopted, making this mitigation largely impractical.

You talkin' to me? Are you talkin' to me?

This research started with the discovery of an App ID hardcoded into "temi", the personal robot we were researching. Agora documents clearly on its website that this App ID should be "kept safe", which we discovered as a vulnerability while researching temi. However, further examination shows that even if temi had kept this value safe, it is sent in cleartext over the network along with all the other values needed to join the call. So how does this impact other consumers of the Agora SDK outside of temi?

Agora's website claims – "Agora's interactive voice, video, and messaging SDKs are embedded into mobile, web and desktop applications across more than 1.7 billion devices globally." To ensure our comparisons would be reasonable, we looked only at other Android applications which used the video SDK. We also focused on apps with a very large install base.

To give a quick cross-section of apps on Google Play that utilize Agora, let's focus on MeetMe, Skout, Nimo TV, and, of course, temi. Google Play reports the number of installs at 50 million+ each for MeetMe and Skout, 10 million+ for Nimo TV and 1000+ for temi. By examining the applications' decompiled code, we can easily determine all four of these applications have hardcoded App IDs and do not enable encryption. We see very similar code as below in all the applications:

Figure 12: joinChannel function from Android apps not using encryption

The most important line here is the call to setEncryptionSecret which is being passed the "null" parameter. We can reference the Agora documentation to understand more about this call.

Figure 13: Agora documentation on setEncryptionSecret

Even though the encryption functions are being called, the application developers are actually *disabling* the encryption based on this documentation. A developer paying close attention might ask if this is the only place the API is being called. Could it be set elsewhere? The team searched the decompiled code for all the applications reviewed and was unable to find any other instance of the API call, leading us to believe this is the only call being made. With this in mind, the cleartext traffic has a greater impact that goes well beyond a personal robot application to millions – potentially billions – of users. Without encryption enabled and the setup information passed in cleartext, an attacker can spy on a very large range of users.

Although Agora should not have been sending this sensitive information unencrypted, if it offers an encrypted traffic option, why is it not being used? This would at least protect the video and audio of the call, even though an attacker is able to join. While it is impossible to be certain, one reason might be because the Agora encryption options require a pre-shared key, which can be seen in its example applications posted on GitHub. The Agora SDK itself did not provide any secure way to generate or communicate the pre-shared key needed for the phone call, and therefore this was left up to the developers. Many calling models used in applications want to give the user the ability to call anyone without prior contact. This is difficult to implement into a video SDK post-release since a built-in mechanism for key sharing was not included. It is also worth noting that, generally, the speed and quality of a video call is harder to maintain while using encryption. These may be a few of the reasons why these application developers have chosen to not use the encryption for the video and audio. Now that the

Agora has published a list of best practices for all its developers and partners here, which does include use of encryption when possible. We generally recommend following vendors' security best practices as they are designed to apply to the product or service directly. In this case, the vulnerability would still exist; however, its effectiveness would be extremely limited if the published best practices were followed. Although we have found Agora's recommendations were largely not being adopted, Agora has been actively communicating with customers throughout the vulnerability disclosure process to ensure its recommended processes and procedures are widely implemented.

Yo, Adrian, we did it. We did it.

Privacy is always a top concern for consumers, but also remains an enticing threat vector for attackers. If we look at the two biggest apps we investigated (MeetMe and Skout), both are rated for mature audiences (17+) to "meet new people" and both advertise over 100 million users. MeetMe also mentions "flirting" on the Google Play store and its website has testimonies about people meeting the "love of their life". Although they are not explicitly advertised as dating apps, it would be reasonable to draw the conclusion that it is at least one of their functions. In the world of online dating, a breach of security or the ability to spy on calls could lead to blackmail or harassment by an attacker. Other Agora developer applications with smaller customer bases, such as the temi robot, are used in numerous industries such as hospitals, where the ability to spy on conversations could lead to the leak of sensitive medical information.

One goal of the McAfee Advanced Threat Research team is to identify and illuminate a broad spectrum of threats in today's complex and constantly evolving landscape. As per McAfee's responsible disclosure policy, McAfee ATR informed Agora as soon as the vulnerability was confirmed to be exploitable. Agora was very receptive and responsive to receiving this information and further advanced its current security capabilities by providing developers with an SDK option (version 3.2.1) to encrypt the initial call setup information, mitigating this vulnerability. We have tested this new SDK and can confirm it fully mitigates CVE-2020-25605. At the time of writing, McAfee is unaware of any instances of this vulnerability being exploited in the wild, which demonstrates another powerful success story of mitigating an issue which may have affected millions of users before it is used for malicious purposes. Our partnership with Agora resulted in the release of a more secure SDK which has empowered developers across multiple companies to produce more secure video calling applications. We strongly recommend any development team which uses the Agora SDK to upgrade to the latest version, follow Agora's outlined best practices, and implement full encryption wherever possible.

Vulnerability Details

CVE: CVE-2020-25605
CVSSv3 Rating: 7.5/6.7
CVSS String: AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N/E:P/RL:O/RC:C
CVE Description: Cleartext transmission of sensitive information in Agora Video SDK prior to 3.1 allows a remote attacker to obtain access to audio and video of any ongoing Agora video call through observation of cleartext network traffic.

Douglas McKee

Douglas McKee is a Principal Engineer and Senior Security Researcher for the Advanced Threat Research team, focused on finding new vulnerabilities in both software and hardware. Douglas has an extensive...

More from McAfee Labs

Fake Security App Found Abuses Japanese Payment System

Authored by SangRyol Ryu and Yukihiro Okutomi McAfee's Mobile Research team recently analyzed new malware targeting mobile...

NOV 30, 2022 | 5 MIN READ

Threat Actors Taking Advantage of FTX Bankruptcy

Authored by Oliver Devane It hasn't taken malicious actors long to take advantage of the recent bankruptcy...

NOV 15, 2022 | 2 MIN READ

Microsoft's Edge over Popups (and Google Chrome)

Following up on our previous blog, How to Stop the Popups, McAfee Labs saw a sharp decrease...

NOV 15, 2022 | 2 MIN READ

Don't Get Caught Offsides with These World Cup Scams

Authored by: Christy Crimmins and Oliver Devane Football (or Soccer as we call it in the U.S.)...

NOV 09, 2022 | 6 MIN READ

New Malicious Clicker found in apps installed by 20M+ users

Authored by SangRyol Ryu Cybercriminals are always after illegal advertising revenue. As we have previously reported, we...

OCT 19, 2022 | 6 MIN READ

Malicious Cookie St... Chrome Extensions ... Million Users

Authored by Oliver Devan... Chole September 9, 2022 ... the original publication o...

AUG 29, 2022 | 8 MIN READ

McAfee

Corporate Headquarters
6220 America Center Drive
San Jose, CA 95002 USA

Products

McAfee® Total Protection
McAfee® Gamer Security
McAfee® Identity Monitoring Service
McAfee® Security Scan Plus
McAfee® WebAdvisor
McAfee® Techmaster Concierge
McAfee® Virus Removal Service

Resources

Antivirus
Free Downloads
Parental Controls
Malware
Firewall
Blogs
Activate Retail Card
Threat Center
McAfee Enterprise

Support

Consumer Support
FAQs
Renewals
Support Community

About

About McAfee
Careers
Contact Us
Newsroom
Investors
Legal Terms
Your Privacy Choices
System Requirements
Sitemap