How video injection attacks can even challenge state-of-the-art Face Presentation Attack Detection Systems

Kévin CARTA
CLR Labs
La Ciotat, France

André HUYNH
CLR Labs
La Ciotat, France

Stéfane MOUILLE
CLR Labs
La Ciotat, France

Pr. Nadia EL MRABET
Ecole des Mines de St Etienne
Gardanne, France

Dr. Claude BARRAL
Bactech
Aix-en-Provence, France

Dr. Sébastien BRANGOULO
Unissey
Paris, France

ABSTRACT

Owing to the digitization of our society, recently rushed by COVID 19 pandemic, remote biometrics, particularly facial recognition, are emerging in plenty of applications that require authentication at distance. These sensitive applications, mostly used in banking, governmental, or insurance domains, are threatened by the well-known presentation attack threat but also by video injection attacks (modification of the images taken by the camera by the attacker), which are now more accessible to fraudsters as devices are not scrutinized and under their control. We will see in this paper that even a simple photo can fool state-of-the-art biometric attack detection systems. Thus, we will give recommendations to avoid the usage of video injection attacks by malicious applicants.

1. INTRODUCTION

Presentation Attack Detection (PAD) for Facial Recognition Systems (FRS) has received important attention from the biometric community since FRS cannot distinguish between bona fide and impostor presentations. The usage of Presentation Attack Instruments (PAI), whether they are simple such as face printed or displayed attacks or more elaborated such as 3D rigid or silicone face masks, for instance, to fool FRS has been demonstrated in many papers [7], [16], [10]. This awareness about presentation attacks has led to the creation of an international standard, the ISO/IEC JTC1 SC37 30107 [19], on PAD systems. Thanks to these researches and the establishment of security evaluation methodologies based on ISO/IEC 30107, the detection of presentation attacks has been improved constantly to allow FRS complemented with PAD systems to reach an acceptable level of security against this type of attack.

This work is being part of a Ph.D Thesis from Kevin Carta under Prof. Nadia El Mrabet with SAS laboratory at Ecole des Mines de Saint Etienne and CLR Labs

Nevertheless, presentation attacks are not the only existing type of attacks against FRS. Figure 1 describes all the possible types of attacks against a facial recognition system. Due to the digitization of our society, well helped by the COVID 19 pandemics, we have seen in the last couple of years the emergence of digital solutions using remote biometrics and particularly remote facial recognition. Remote biometrics is the usage of a biometric recognition at distance in a non-trusted environment (e.g., facial verification of user ID with a webcam and a passport for entering in a secured chat) in opposition to face-to-face biometrics which is used in a trusted environment (e.g., usage of a human supervised automated gate during border control). These types of solutions are commonly used for digital identity verification in domains of banking, governmental, or insurance applications. Such solutions, where the application requires a substantial level of security (e.g., open a bank account remotely), perform remote identity proofing thanks to facial recognition between the applicant's face and the photo of its ID document.

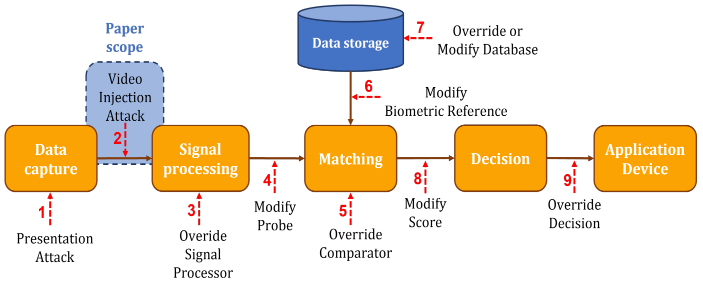


Fig. 1: Different types of attack on FRS[18]

ISBN: 978-1-950492-71-8
ISSN: 2771-5914

https://doi.org/10.54808/IMCIC2023.01.105

Remote FRS are of course vulnerable against presentation attacks like face-to-face FRS. Still, they are also vulnerable against video injection attacks, described as type 2 attacks in Figure 1. Video injection attacks consist of modifying the data taken by the camera (i.e. the biometric data capture system) with malicious data. This attack can consist in replacing the images taken by the camera or overwrite them, for example. Thus, as FRS is used remotely, it means that the attacker can use its device to build an attack on the facial recognition. Having control of the device (e.g., its smartphone or computer) allows the fraudster to change the original behavior of the device. For instance, the attacker can install a virtual camera on his computer which will grant him the opportunity to send chosen images and the targeted application will think that it is receiving images from a real webcam.

Video injection attacks are historically related to the domain of cybersecurity and penetration testing [12]. However, the European cybersecurity community and national security agencies, such as the French ANSSI (Agence Nationale de la Sécurité des Systèmes d'Informations) [1] or the German BSI (Bundesamt für Sicherheit in der Informationstechnik), are in line with the fact that mobile applications and web applications can't be considered as secured, and finding vulnerabilities is always a question of time. Thus, as these web and mobile applications can't be deemed fit to trusted environments, biometric developers need to implement biometric security features to prevent video injection attacks.

In this paper, the Face PAD solution from the French company Unissey will be taken as an example to see if state-of-the-art PAD solutions are resistant against video injection attacks. Unissey has developed a Face PAD system, based on Deep Learning, evaluated on its security level by two different laboratories with zero percent presentation attack success rate. Moreover, Unissey is one of the first PAD developers to consider this new threat as video injection attacks, and has developed countermeasures to mitigate this threat.

This paper primarily addresses the following question: "How vulnerable are state-of-the-art Face PAD systems to video injection attacks?". Specifically, we investigate the vulnerability of Unissey's PAD web-browser application to presentation attacks, (i.e.:simple, as well as more elaborated video injection attacks) on two different devices, computer and smartphone. This specific PAD use case has been chosen for this study because it is one of the top-level solutions from one of the first companies to develop security features preventing video injection attacks. Our study goal is focused on the following targets:

1) Assessing the performance of what seems a systematic FRS-vulnerability study linking video injection attacks with advanced PAD system.
2) Demonstrating the real need to develop a biometric-based countermeasure against emerging threats, as mobile applications and web applciations can't be considered as trusted environments because of the presence of Information Technology (IT) vulnerability that allows attackers to make innovative attacks such as video injection.

In Section 2 we provide brief reviews of other researchers' relevant works on presentation attacks and video injection attacks. The robustness of Unissey against presentation attacks and simple video injection attacks is described in Section 3. An efficient video injection attack against state-of-the-art PAD systems and the experimental results are presented in Section 4. Eventually, Section 5 summarises this study, the conclusions drawn from the results, and an outlook on future developments.

https://unissey.com/

Level 2 conformance with iBeta (USA) and CLR Labs (France)

2. STATE-OF-THE-ART

2-A. Presentation attacks and their detection

Most common attacks against FRS are known as presentation attacks. An attacker can fool the FRS simply by presenting a face artifact of a legitimate user, which can be easily created thanks to the availability of face images and videos of a person on the internet and social media. These attacks have attracted the attention of biometric experts and have even given birth to a specific international standard, the ISO/IEC 30107.

Presentation attacks on FRS are made with artifacts, or PAIs, which usually belong to one of these categories:

- Face printed [14]
- Face displayed on a screen [14]
- Paper face mask
- 3D face mask [15, 17, 8]

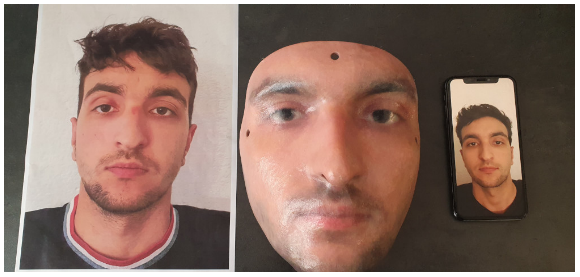


Fig. 2: Examples of face PAI

Of course, each artifact requires different expertise, cost, or even biometric sources (biometric traits of the victim needed to create the artifact, e.g., a photo or a face 3D scan), and obtaining each artifact does not imply the same level of difficulty. For instance, it is far easier to possess a printed photo of the victim's face than a 3D silicone mask of its face, which requires measurements of the victim's face and a manufacturing cost of several thousands of euros by a high qualified cinema makeup artist.

Many researchers took this approach for years to evidence the efficiency of these artifacts' attacks on FRS if the latter are not complemented with PAD [7, 16, 10].

Because of this threat, PAD development has been a prior topic for the biometric community. The improvement of Artificial Intelligence (AI) and many years of research have given rise to the emergence of plenty of methodologies to detect presentation attacks on FRS. Here are some existing, but non-exhaustive, examples in the literature:

- Liveness detection, or so-called "Passive PAD". Some tools can detect liveness in images without asking the user for any challenge. In [13], the authors show that it is possible to detect the blood pulse on the face with a classical camera. Unissey uses this technique to detect presentation attacks.
- Challenge-Response, or so-called "Active PAD". This tool consists in requesting a specific challenge to the user (e.g., turning the head right or eyes blinking) and detecting the relevant answer from the user (e.g., the user has turned his head right or has blinked eyes) [20, 22]
- Deep Learning algorithm trained with datasets of real faces and datasets of different kinds of face PAI (Unissey is an example of this kind of PAD solution). The algorithm is

then able to distinguish between a real face and a 3D rigid mask for instance [2, 3]

- Specific Hardware Materials. In some use cases, having specific hardware materials can help to detect artifacts. In [9], the authors show that having a specific camera, called "Time-of-Flight" camera, helps to get 3D information about a face. "Structured light" is another technique to obtain 3D information with an IR camera as described in [21].

2-B. Video injection attacks

Code injection is the act of introducing a malicious script into an application source code to modify its normal behavior. In cybersecurity, injection is particularly used by penetration testing laboratories, and hackers, to find vulnerabilities in a system. One of the most used injection attacks is SQL injection which consists of injecting code snippets to get information from a database fraudulently [5].

For facial recognition systems, the injection has an interest for the attacker if he can inject chosen images in the system so that the system uses these images instead of the ones taken by the camera: this is a video injection attack. In this case, the attacker does not only inject code to get information (e.g., a password) but does also inject data.

Since video injection attacks did not have a huge interest for attackers in the past, as biometrics were mainly used in trusted environments or under human scrutiny, the current emergence of remote biometrics opens for the attacker the choice of a selected device (enabling behavior modification) and makes video injection attack more accessible. Although the criticality of the emergence of threats linked to remote biometrics has been recently reported in general terms (e.g.: through accessible reviews), we are familiar with only two main focused research papers on this subject.

The first one [11] talks about the feasibility of video injection attacks on medical material. In this paper, the authors do not describe how to make a video injection attack but propose a blockchain framework to prevent false medical video injection in the health domain.

The second one deals with facial recognition. In [4], the authors describe a video injection attack onto a secured digital identity verification mobile application that uses facial recognition to verify that the applicant is the legitimate holder of the passport. The paper highlights the fact that video injection is a new threat against remote biometrics and particularly its usage in identity proofing solutions, as described in our introduction. However, this paper does not mention the name of the targeted mobile application and does not explain in detail how to implement the video injection attack on the Android device, which makes the attack unrepeatable.

To overcome the above limitations, the purpose of this article is to take a specific example used in the real-world market, here the Unissey PAD solution, and to illustrate how potential impostors can implement video injection attack to fool state-of-the-art biometric securities (i.e., PAD solutions) which can be integrated into sensitive applications featuring identity verification for banking or governmental services for instance.

3. ROBUSTNESS OF UNISSEY'S APPROACH

Structured Query Language (SQL): domain-specific language used in programming and designed for managing data

The security of an application using facial recognition relies on 3 components: PAD, the detection of any virtual device which can modify the original device behavior (e.g., a virtual camera), and the classical IT security features (e.g., source code obfuscation). Figure 3 highlights the different levels of adhesion surface to biometrics for these 3 security components.

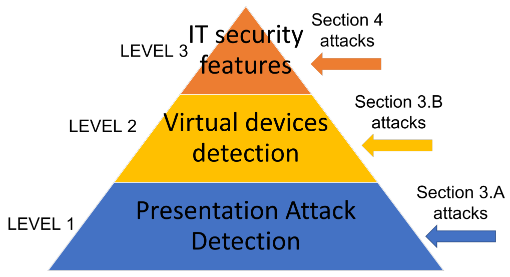


Fig. 3: Structure of security features in application using facial recognition

First of all, as described in the introduction, the choice of Unissey's PAD system for this paper has been made because Unissey's solution is a state-of-the-art system to prevent presentation attacks (the most evaluated on the market) and is one of the first PAD systems to implement countermeasures for video injection attacks using external resources. Having as a targeted application a solution that implements the first two main security level components, as described in Figure 3, will help this paper to highlight the feasibility and danger of video injection attacks.

Then, before explaining what are the different attacks that we have realized on the Unissey PAD system, it is essential to understand how the Unissey solution operates. This tool is a so-called "passive" PAD system, meaning that the user does not have to do anything during the process. The usage of the product consists in launching the camera and putting the face in a target on the screen, as seen on Figure 4. Then the system, within a single second, acquires a few images of the user's face and performs a presentation attack detection thanks to a deep learning algorithm trained with consequent datasets of real human faces and face artifacts.

Eventually, the Unissey PAD system application can be used via a web browser installed on a computer (with a webcam) or a mobile device.

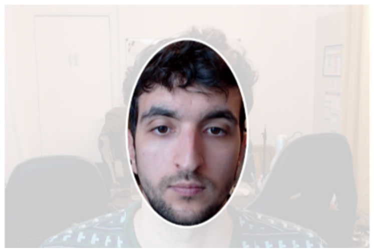


Fig. 4: Transaction on Unissey PAD system

3-A. Presentation attacks

As explained in the introduction, Unissey has been evaluated by two different laboratories on presentation attack detection on a substantial level and for both laboratories, 0% of attack has been successful. In order to check these results, we firstly experiment attempts to fool the Unissey PAD system by achieving presentation attacks.

Figure 5 shows the prepared 5 different presentation attacks used:

1) A face photo printed
2) A 3D rigid face mask
3) A face video displayed on a smartphone screen
4) A 3D latex face mask
5) A 3D silicone face mask

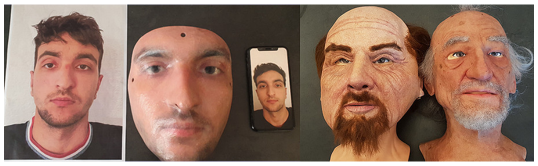


Fig. 5: PAI used against Unissey PAD system

Each attack has been presented 12 times and the environment has been changed for each presentation: each attack has been made on four different cameras (an embedded laptop camera, an iPhone 5S front camera, a Samsung Galaxy S10 front camera, and a Logitech C920 webcam) with 3 different lighting conditions (indoor with around 100lux illuminance, indoor with around 500lux illuminance and outdoor with around 1500lux illuminance). The attack results are presented in Table 1.

Attacks	Biometric source	Success rate
Photo printed	A photo	0/12 = 0%
Video displayed	A video	0/12 = 0%
A 3D rigid face mask	A 3D face scan	0/12 = 0%
A 3D latex face mask	Face measurements	0/12 = 0%
A 3D silicone face mask	Face measurements	0/12 = 0%

Table 1: Results for presentation attacks

No scenario has been able to trick the system regardless of the light environment or the quality of the camera, thus confirming Unissey's robustness against face presentation attacks, in line with the announced results of the two prior evaluations of this product.

3-B. Simple video injection attacks

The most straightforward video injection attack consists of using a tool able to simulate a real webcam's behaviour and send chosen images of a screen or a video file. Such a tool can be a virtual camera, which is a software able to be seen as a camera by a website or a computer application and which can use video file or replay a screen as a data source. Alternatively, the tool can be an external capture card, which is a hardware device able to take any data source via a HDMI cable (e.g.: a laptop screen) and the output of the device is seen as a camera by the computer used for the attack. Figure 6 is a diagram of how the video injection attack is achieved on a computer thanks to a virtual camera or an external capture card.

To check if the video injection was possible on the Unissey web-browser application, we used the simplest video data possible for our attack: a video of a victim's face without any modification or editing. To test the feasibility of video injection, we selected 5 different general public web-browsers:

- Google Chrome on Windows 10
- Edge on Windows 10
- Opera on Windows 10
- Mozilla Firefox on Linux Ubuntu
- Chromium on Linux Ubuntu

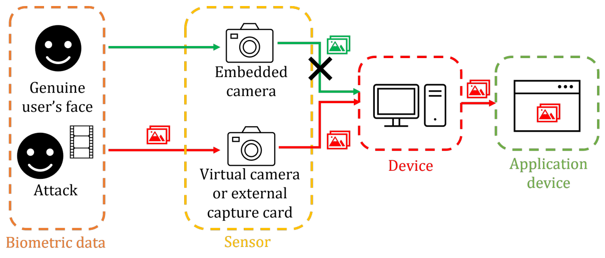
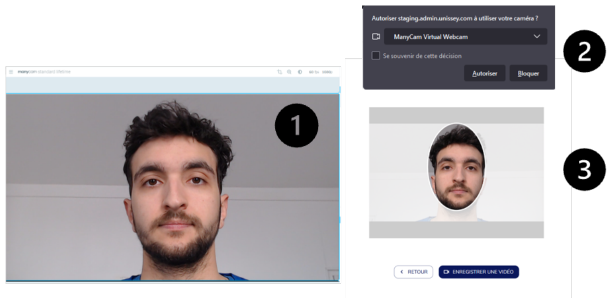


Fig. 6: Video injection attack with a virtual camera

Eventually, we used 4 different attack paths: 3 different virtual cameras and 1 external capture card.

- OBS virtual cam (virtual camera)
- Manycam (virtual camera), see Figure 7
- Akvcam (virtual camera)
- Elgato Camlink (external capture card)



Autoriser staging.admin.unissey.com à utiliser votre caméra ?

ManyCam Virtual Webcam

Se souvenir de cette décision

Autoriser Bloquer

Fig. 7: Attack stages with Manycam virtual camera

As previously described, we used a video of a face via each tool using each web browser (when it was possible) and we got the attack results presented in Table 2.

Web-browser(OS)	OBS	Manycam	Akvcam	Camlink
Chrome(Win 10)	Failed	Failed	Incompatible	Failed
Edge(Win 10)	Failed	Failed	Incompatible	Failed
Opera(Win 10)	Failed	Failed	Incompatible	Failed
Firefox(Linux)	Failed	Incompatible	Failed	Incompatible
Chromium(Linux)	Failed	Incompatible	Failed	Incompatible

Table 2: Results for simple injection attacks

As can be seen, no injection attacks fooled the Unissey PAD system which means that in each configuration, the system has detected a virtual camera. This is because Unissey has developed a virtual camera detection method in addition to its PAD system which is not the case for the majority of the PAD systems on the market. On the flip side, we can therefore, assume some (if not all) of our attacks would be effective on PAD systems with no embedded added virtual camera detection system.

However, we will see in Section 4 that using a virtual camera isn't the only way to make an injection, meaning the presence of an effective virtual camera detection is not sufficient to prevent video injection attacks.

4. AN EFFECTIVE ATTACK AGAINST STATE-OF-THE-ART PAD SYSTEMS

DISCLAIMER: *The vulnerability exploited here concerns all web browser Android applications. It means that even if Unissey has been taken as an example in this paper,* ***this vulnerability is***

exploitable for all web-browser PAD applications and is not a vulnerability of the Unissey PAD system. *It is worth noting that the French ANSSI has been the first to take into account video injection attack threat in their remote identity proofing using FRS certification scheme[1] and has led to the integration of the video injection threat in the ETSI remote identity proofing certification scheme [6].*

Using a virtual camera software or a capture card is difficult to set up on a mobile device. Yet, it doesn't mean that video injection attacks are not possible on such devices, it just means that the way to inject images isn't the same. We shall report in this section how to make a video injection attack in a mobile web browser application and discuss if this ends up fooling the Unissey PAD system. To experiment with this attack, we used a Samsung Galaxy S8 with Android 9.

4-A. Methodology of video injection on mobile web browser

Making a video injection attack on a mobile device isn't as simple as installing virtual camera software on a computer. Indeed, it requires from the attacker some knowledge in mobile penetration testing (i.e., knowledge on mobile hacking and mobile application reverse engineering). Making injection here will not consist in creating a new flow of images via a tool like a virtual camera, but it consists in modifying the image flow taken by the legitimate camera. In this attack, we will overwrite images taken by the camera with chosen images (the images of our attack) before they arrive in the targeted web browser mobile application. The attack is done during the communication between the camera and our mobile app as described in Figure 8.

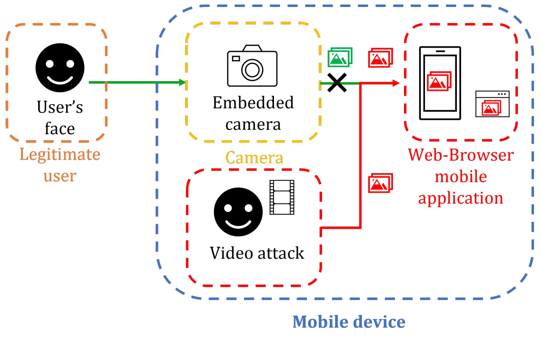


Fig. 8: Injection attack on a mobile device

To be able to inject images in mobile web browser applications, we need to have:

- **An Android mobile device.** In our paper, we have taken a Samsung Galaxy S8, but every Android device can be used. In theory, video injection attacks should be possible on IOS devices but we have not tested it.
- **A TWRP ROM image (binary image of the Android OS) for the Android Device with Magisk.** TWRP and Magisk allow the user to root its smartphone (the rooting process is explained in the next paragraph).
- **Odin.** Odin is software that permits to flash ROM image on an Android device.

https://twrp.me/
https://github.com/topjohnwu/Magisk
https://www.odinflash.com/

- **Jadx-Gui.** Jadx is an open-source Dex to Java decompiler. It allows anyone to get the source code of any apk file (format of an Android mobile application).
- **Frida.** Frida is an open-source toolkit that enables its user to inject script and to hook any function. The hooking process is an operation where function calls are intercepted by a program to modify their behavior. Frida is built on a server-client architecture. To operate, the smartphone has to be connected to a computer via USB. The user installs and executes the server-side of Frida into the smartphone and he has to install the client-side of Frida on his computer.
- **Android Debug Bridge (ADB).** It is a versatile command-line tool that lets you communicate with a device. The ADB command facilitates a variety of device actions, such as installing and debugging apps, and it provides access to a "super-user" shell, if the device is rooted, that you can use to run a variety of commands on a device.
- **A general public web browser mobile application.** The choice of the web browser is not important as the way to get the images from the mobile camera is the same for all the web browsers.

The first step to make the injection possible is to create a root of the smartphone device. Rooting consists of flashing a new ROM image on which the user will have "super-user" access. Having a "super-user" access means that the user has more permissions on the smartphone than he would have on a standard device. In our case, it will allow us to execute the Frida server in the smartphone (executing a program into an Android device is not permitted with standard access). However, note that Frida can be used without rooting (thanks to a tool called Frida gadget) but its usage is more complicated and isn't as powerful as using with rooting.

To root the Samsung Galaxy S8, we have first unlocked the OEM (Original Equipment Manufacturer) option in the developer parameters of the device. This action gives the user permission to flash a new ROM image on the device. Then, we have flashed the TWRP ROM image with Odin and install Magisk on our device. Note that if rooting can be seen as a complicated task, it is accessible to anyone as you can find plenty of ROM images and tutorials on the web for any Android device.

Once the smartphone is rooted, Frida can be downloaded and installed on the computer and in the device (generally, we store Frida in the directory /data/local/tmp).

Then, the most important step of the attack consists in finding the function used by the web browser application which calls the device camera. To find this function, we have to reverse engineer the mobile application, with Jadx, to obtain the source code. Once it is done we have find in the code how and where the camera is called. However, knowing about the internals of an Android camera operations can make this research far quicker since the attacker can search for a specific function name or object.

To get images from the camera, a mobile application has to call the Android camera API. Android camera API is the interface between a mobile application (whether it is a native OEM application or an installed application like our web browser application) and the hardware sensor of the camera. So, when it needs to open the camera to take a photo or a video, the mobile application calls the camera API and asks for images with different parameters like: pixel format, size of the image,

https://github.com/skylot/jadx
https://github.com/frida/frida

front or back camera, etc.

Android devices possess 2 different camera APIs: Camera1 and Camera2. Camera1 is the old camera API that Camera2 slowly replaces, but as some developers prefer the old API and old smartphones are not compatible with the new one, Camera1 is still used. In our research, we have voluntarily chosen an old version of our web browser application (a 2018 version) to select the Camera1 interface. Because the update of the web browser mobile application is not mandatory, the attacker has the opportunity to use an older version of his app. Making injection on the Camera1 interface is far simpler as images transit in clear as ByteArray between the camera and the mobile application. At the same time, Camera2 transfers images via buffer memory, making injected images preparation more laborious.

The mobile application asks for images through Camera1 API via the function *onPreviewFrame*. This function returns two objects: a ByteArray which is the image taken by the camera, and an object called "camera" which groups all parameters of the camera (front camera or back camera and size of image).

The video injection attacks consist in finding this particular function in the source code and making a hook of this function to overwrite the images. Once the function has been found, we have to use Frida to make the hook.

To use Frida, we have to execute frida-server in the smartphone with the ADB command line (go in /data/local/tmp and execute ./frida-server). Then the hook is made thanks to a JavaScript file which will be injected by Frida on the client-side with a specific command line (on our computer connected via USB to the mobile device). Successive hook steps are summarized in Figure 9.

In the script, we need first to find the context of our web browser mobile application. The context which allows Frida to attach the JavaScript file in the targeted mobile application. To do it we can use this code lines: *const ActivityThread = Java.use("android.app.ActivityThread");* and then *const Application = ActivityThread.currentApplication();*. Thanks to these lines, it will be able hook any function of our web browser mobile application with Frida when the application is opened. As we know where the *onPreviewFrame()* function is located, we have then to declare the Class where the function is called (e.g., *const VideoCaptureAndroid = Java.use("org.webrtc.videoengine.VideoCaptureAndroid");*) and to implement our hook of the function *onPreviewFrame()* (*VideoCaptureAndroid.onPreviewFrame.implementation = function(frameBytes, camera)*). Thanks to these steps, we are able to modify the behavior of the original function and thus to modify the object *frameBytes* which is the image taken by the camera.

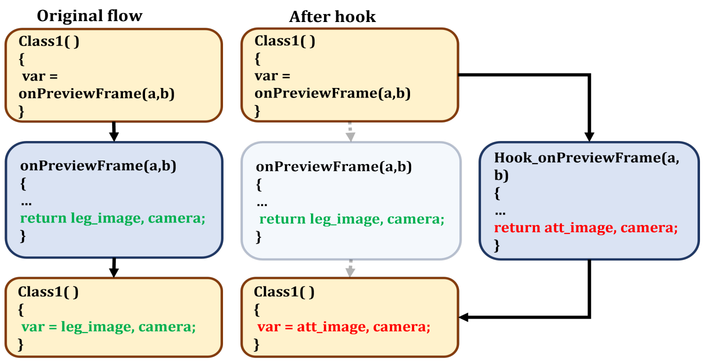

Original flow
After hook
Class1( ) { var = onPreviewFrame(a,b) }
onPreviewFrame(a,b) { ... return leg_image, camera; }
Class1( ) { var = leg_image, camera; }
Class1( ) { var = onPreviewFrame(a,b) }
onPreviewFrame(a,b) { ... return leg_image, camera; }
Hook_onPreviewFrame(a, b) { ... return att_image, camera; }
Class1( ) { var = att_image, camera; }

Fig. 9: Hook of onPreviewFrame()

Before replacing *frameBytes* with the images of our attack, we have to know the size of the images taken by the camera and the pixel format (i.e., how the image is encoded). In the source code of the web browser mobile application, the return value of the function *getBitsPerPixel* gives the nature of the format image. In our case, the value returned was 17 which corresponds to NV21 image format according to Android developer documentation. Then we can obtain the width and height of the image thanks to the commands *console.log("Width:" + camera.getParameters().getPreviewSize().width.value);* and *console.log("Height:" + camera.getParameters().getPreviewSize().height.value);* which for Unissey PAD system gave us a size of 1280x720 pixels.

Once we have these values, we have to decompose our video into images in raw format (with no compression) with color encoded in NV21 at size 1280x720 pixels. This can be done thanks to the tool Ffmpeg for instance.

Then, we have to store our images in the files directory of our web browser mobile application (e.g., /data/data/web-browser/files) and call them via our JavaScript file within the *onPreviewFrame()* implementation to overwrite in live the images taken by the camera.

We can share our injection JavaScript file on our e-mail address on demand to replay video injection and to avoid writing here all the needed code lines to make the video injection. Due to the sensitive nature of this file, we can't share it on our Github account.

4-B. Video injection attacks

Thanks to the work described in the last subsection, we are now able to inject any images on any website on which we would use the camera. Thus, we can inject any images in the Unissey PAD system which will think that it receives legitimate images from the smartphone camera.

The purpose of the attacks that we will consider would be to impersonate the identity of a specific victim. In other terms, we will consider that the attacker is an impostor. Performing a video attack requires a biometric source from the victim: this one can be a photo, a video, a face scan, etc. Of course, each biometric source does not require the same effort from the attacker: while a 3D scan of the face of the victim is barely impossible to obtain, finding a photo or a video is simple at the age of the internet and social media.

In this paper, we have considered 5 different attacks, 3 made with a simple photo of the victim's face and 2 made with a video of the victim's face.

Attack 1 — Simple Photo. The first attack consists in injecting a single, static, photo of the victim's face.
Attack 2 — Face Reenacted. Face reenactment consists in using a face image and reproducing face movements of a specific face video with the photo. In our attack, we have considered a face image and we have made the eyes blink. Many mobile applications or open-source can do face reenactment: we have used Avatarify.
Attack 3 — Low Quality Deepfake. A face deepfake video attack consists in putting the face of a subject A, the victim, on the face of subject B, the attacker, during a video. Thanks to this process we can recreate the movement of a face (subject B, the attacker) with another one (subject A, the victim). Our Low-Quality Deepfake

https://developer.android.com/
https://ffmpeg.org/
kevin.carta@cabinet-louis-reynaud.fr

attack is done with an Android mobile application named Reface App. To create the attack, the impostor gives to the application a video of his face and a photo of the victim's face and the app recreate a video in a few seconds which is the same as the one given by the attacker but his face has been replaced by the victim's one. This deepfake is qualified as ”low quality” because the result obtained is low resolution and got plenty of defaults due to the data scarcity of the biometric source (a single photo). Yet, in most FRS, even if the resolution is low and the result quite disappointing, the match between a legitimate photo of the victim's face and the deepfake is a success. Results are visible on Figure 10.

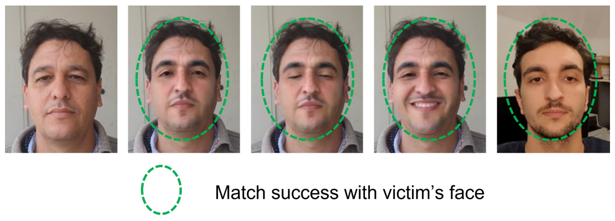


Fig. 10: On the left the attacker, in the middle 3 frames from the Low Quality Deepfake attack, on the right the victim

Attack 4 — Simple Video. The fourth attack consists in injecting a single video of the victim's face. In our paper, we have used video of faces registered during video conferences.

Attack 5 — High Quality Deepfake. High-quality deepfake is the same kind of attack as the low-quality deepfakes: the victim's face is put on the attacker's face within a video. But in this case, the biometric source of the victim's face is a video (we have used face's video registered during video conference). Moreover, to create these attacks, we have used an open-source deep-learning deepfake tool called DeepFaceLab with which we trained a specific model for each victim. Each model has required 3 full days of work with a computer equipped with an Nvidia RTX 3090 graphics card. Results are visible on Figure 11.

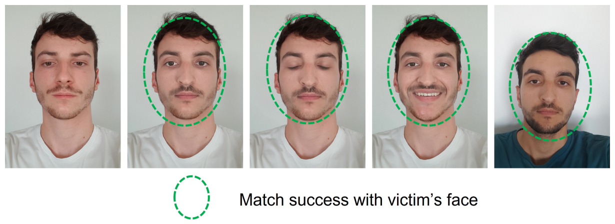


Fig. 11: On the left the attacker, in the middle 3 frames from the High Quality Deepfake attack, on the right the victim

4-C. Results

We have considered 4 different victims and 2 different attackers for each attack, as presented in Table 3. The attackers and victims have been associated according to their gender.

Once all the video attacks have been prepared, we have decomposed each of them with FFmpeg at 24 Frame Per Second (FPS) in .raw format with color encoded in NV21 format at size 1280x720 pixels according to the parameters notified during injection set up (Section 4.A). For each attack, we have obtained the results presented in Table 4.

	Age	Gender
Victim 1	25	Female
Victim 2	47	Female
Victim 3	21	Male
Victim 4	45	Male
Attacker 1	22	Female
Attacker 2	24	Male

Table 3: Victims and Attackers used for video injection attacks on mobile web-browser application

Attack	Victim 1	Victim 2	Victim 3	Victim 4
Simple Photo	Success	Success	Success	Success
Face Reenacted	Success	Success	Success	Success
Low Quality Deepfake	Success	Success	Success	Success
Simple Video	Success	Success	Success	Success
High Quality Deepfake	Success	Success	Success	Success

Table 4: Results of video injection attacks on mobile web-browser application

As can be seen from Table 4, all the prepared attacks have been able to fool the Unissey PAD system. This underlines the fact that once the injection method has been well set up, it is more complicated to detect it. It means that even if the PAD system is highly secured against presentation attacks with features such as virtual camera detection or emulator detection been set, injection attacks are still possible and even a single image photo allows an impostor to steal the victim's identity.

Eventually, we can understand here that the real vulnerability is exploited in the Camera API architecture which means that if Unissey can be fooled, as we have shown in this paper, it is likely also possible for each PAD system or Android mobile application using facial recognition.

But, because it is not possible to uniquely identify the camera in the current smartphone architecture, countermeasures have to be placed within the PAD system.

5. CONCLUSION AND FUTURE WORK

The development of remote FRS used by remote digital identity verification solutions paves the way for new attacks using mobile devices, which are more accessible for attackers because the mobile device is under the attacker's control with no scrutiny from any authority. We have seen that injection may require specific expertise in mobile hacking for PAD systems that develop virtual devices detection solutions (levels 1 and 2 according to Figure 3). However, as this virtual devices detection is not implemented by the majority of PAD systems on the market, video injection attacks are accessible to anyone for these systems.

It is of paramount importance for the biometric community to consider this new threat which can challenge the security of any system using remote facial recognition.

Since current device architecture can't make mobile and web applications trustworthy, the integration of randomness in image capture, to prevent the attackers from preparing in advance their video attacks, is a preliminary way to prevent injection. By doing this, developers would force attackers to use live methodologies to create video attacks, which would complexify the attackers' implementation.

In our upcoming work, we plan to focus on new countermeasures to overcome this new breach avenue facing face recognition. We will focus our interest on how to prevent injection, irrespective of the type of device used by the attacker to make a video injection attack.

6. REFERENCES

[1] ANSSI. "Référentiel d'exigences ANSSI - Prestataires de vérification d'identité à distance - version 1.1". In: (2021). URL: https://www.ssi.gouv.fr/uploads/2021/03/anssi-referentiel%5C_exigences-pvid-v1.1.pdf (visited on 03/23/2021).

[2] Sushil Bhattacharjee *et al.* "Recent Advances in Face Presentation Attack Detection". In: *Handbook of Biometric Anti-Spoofing: Presentation Attack Detection*. Ed. by Sébastien Marcel *et al.* Cham: Springer International Publishing, 2019, pp. 207–228.

[3] Ying Cai *et al.* "A fast and robust 3D face recognition approach based on deeply learned face representation". In: *Neurocomputing* 363 (2019), pp. 375–397. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.07.047. URL: https://www.sciencedirect.com/science/article/pii/S0925231219310215.

[4] K. Carta *et al.* "Video injection attacks on remote digital identity verification solution using face recognition". English. In: *IMCIC 2022 - 13th International Multi-Conference on Complexity, Informatics and Cybernetics, Proceedings*. Vol. 2. 2022, pp. 92–97. URL: www.scopus.com.

[5] Shreya Chowdhury *et al.* "A Comprehensive Survey for Detection and Prevention of SQL Injection". In: *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*. Vol. 1. 2021, pp. 434–437. DOI: 10.1109/ICACCS51430.2021.9442012.

[6] ETSI. "TS 119 461: Electronic Signatures and Infrastructures (ESI); Policy and security requirements for trust service components providing identity proofing of trust service subjects". In: (2021).

[7] Anas Husseis *et al.* "A Survey in Presentation Attack and Presentation Attack Detection". In: *2019 International Carnahan Conference on Security Technology (ICCST)*. 2019, pp. 1–13. DOI: 10.1109/CCST.2019.8888436.

[8] Shan Jia, Guodong Guo, and Zhengquan Xu. "A survey on 3D mask presentation attack detection and countermeasures". In: *Pattern Recognition* 98 (2020), p. 107032. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2019.107032. URL: https://www.sciencedirect.com/science/article/pii/S0031320319303358.

[9] Simon Meers and Koren Ward. "Face Recognition Using a Time-of-Flight Camera". In: *2009 Sixth International Conference on Computer Graphics, Imaging and Visualization*. 2009, pp. 377–382. DOI: 10.1109/CGIV.2009.44.

[10] Zuheng Ming *et al.* "A Survey On Anti-Spoofing Methods For Face Recognition with RGB Cameras of Generic Consumer Devices". In: (Oct. 2020).

[11] Ahmed Mohiuddin. "False Image Injection Prevention Using iChain". In: *Applied Sciences* 9 (Oct. 2019), pp. 1–11. DOI: 10.3390/app9204328.

[12] B. Mueller, S. Schleier, and J. Willemsen. *Mobile Security Testing Guide*. OWASP, 2019. Chap. Android Testing Guide.

[13] Xuesong Niu *et al.* "Continuous heart rate measurement from face: A robust rPPG approach with distribution learning". In: *2017 IEEE International Joint Conference on Biometrics (IJCB)*. 2017, pp. 642–650. DOI: 10.1109/BTAS.2017.8272752.

[14] K. Patel, H. Han, and A. K. Jain. "Secure Face Unlock: Spoof Detection on Smartphones". In: *IEEE Transactions on Information Forensics and Security* 11.10 (2016), pp. 2268–2283. DOI: 10.1109/TIFS.2016.2578288.

[15] R. Ramachandra *et al.* "Custom silicone Face Masks: Vulnerability of Commercial Face Recognition Systems Presentation Attack Detection". In: *2019 7th International Workshop on Biometrics and Forensics (IWBF)*. 2019, pp. 1–6. DOI: 10.1109/IWBF.2019.8739236.

[16] Raghavendra Ramachandra and Christoph Busch. "Presentation Attack Detection Methods for Face Recognition Systems: A Comprehensive Survey". In: 50.1 (2017). ISSN: 0360-0300. DOI: 10.1145/3038924. URL: https://doi.org/10.1145/3038924.

[17] Raghavendra Ramachandra *et al.* "Custom silicone Face Masks: Vulnerability of Commercial Face Recognition Systems amp; Presentation Attack Detection". In: (2019), pp. 1–6. DOI: 10.1109/IWBF.2019.8739236.

[18] N. K. Ratha, J. H. Connell, and R. M. Bolle. "Enhancing security and privacy in biometrics-based authentication systems". In: *IBM Systems Journal* 40.3 (2001), pp. 614–634. DOI: 10.1147/sj.403.0614.

[19] ISO/IEC JTC1 SC37. "ISO/IEC 30107-1:2016 Information technology - Biometric presentation attack detection - Part 1: Framework". In: (2016). URL: https://www.iso.org/standard/53227.html.

[20] A. K. Singh, P. Joshi, and G. C. Nandi. "Face recognition with liveness detection using eye and mouth movement". In: *2014 International Conference on Signal Propagation and Computer Technology (ICSPCT 2014)*. 2014, pp. 592–597. DOI: 10.1109/ICSPCT.2014.6884911.

[21] Shengtao Sun *et al.* "Anti-spoofing Face Recognition Using Infrared Structure Light". In: *Frontiers in Optics / Laser Science*. Optica Publishing Group, 2020, FW1F.3. DOI: 10.1364/FIO.2020.FW1F.3. URL: http://opg.optica.org/abstract.cfm?URI=FiO-2020-FW1F.3.

[22] Munalih Ahmad Syarif *et al.* "Challenge response interaction for biometric liveness establishment and template protection". In: *2016 14th Annual Conference on Privacy, Security and Trust (PST)*. 2016, pp. 698–701. DOI: 10.1109/PST.2016.7907025.