# 6.857 Project: Automating COVIDPass Daily Attestation

Irin Ghosh (irin@mit.edu)

Amanda Horne (ahorne@mit.edu)

Alex Martirosian (amartiro@mit.edu)

Julia Wang (jwlw2022@mit.edu)

May 20, 2021

## Contents

**Abstract**

Our main goals are to demonstrate a liveness attack and conduct security analysis on COVIDPass and its vulnerabilities. We wrote a Python script using Selenium that can automate MIT COVIDPass daily attestation answers and submit them every 24 hours. To combat adversarial attacks and make the system secure, IS&T and the contracted creators of COVIDPass have many effective countermeasures in place such as digital signatures in the form of MIT certificates, Touchstone login, and Duo authentication. The COVIDPass system already does an excellent job of upholding confidentiality, integrity, and authentication. However, the community will benefit from tighter security because there is a lot of sensitive information stored in the database. Furthermore, the attestation form doesn't have liveness confirmation and questions are presented in the same order every time. To improve security and authentication, we recommend adding countermeasures including CAPTCHA and question reordering. IS&T already has features that counteract the trade-off between security and usability. Humans can easily adapt to question reordering with indications that they answered a question incorrectly, but random ordering is difficult for a machine to adapt to.

# 1 Introduction

During the pandemic, it is important for people living and working on the MIT campus to fill out attestation surveys and get tested for COVID twice a week to gain access to campus rooms and resources. The on-campus MIT community is required to download the Atlas app, present a barcode in the app twice a week when getting tested, and fill out a COVIDPass attestation answering questions related to possible symptoms and contact tracing.

The attestation must be completed once every 24 hours for those living on-campus, and prior to every campus visit for those off-campus. The app shows the places where testing is going on and the wait times of the queues and the places on campus that can be accessed by the user by tapping their MIT ID card. The central database gets updated with each user's attestation, test results, and access information every 30 minutes.

The motivation for automating the COVIDPass attestation form is so that a user will constantly maintain campus access and not need to manually upload their data and potentially forget to fill it out one day and lose building access. Students frequently get locked out of their dorms, potentially late at night, and it is a nuisance to fill out the form and then wait 30 minutes for the building access to update.

We successfully implemented an automation script that submits the daily attestation form every 24 hours. We analyzed this exploitation script and the COVIDPass system to identify vulnerabilities and propose additional countermeasures. Our security analysis suggests methods by which we can improve the overall health of the MIT community through effective COVID-19 infection monitoring and control.

# 2 Permission and Responsible Disclosure

We received permission from IS&T to implement the automated attestation script and conduct a thorough security analysis of COVIDPass's vulnerabilities. At the request of IS&T specialists, we presented our findings and advisory to them on May 14, 2021 prior to our class presentation on May 17, 2021. IS&T confirmed that everything we present in this paper is safe to share with the MIT community.

# 3 Related Work

## 3.1 Other Automation Attempts

This is not the first time the idea has been raised in the MIT community. There have been other MIT community members (who will remain anonymous) who have studied the potential for automating the attestation forms. Our goal is to discover and exploit vulnerabilities with a dummy account and finally inform IS&T about those vulnerabilities and suggest additional security measures to mitigate those vulnerabilities.

## 3.2 Selenium for Web Scraping

A relevant tool in our implementation was the Selenium graphics library (Section 4.2). The main reason why Selenium is a popular library for web scraping is that it supports writing tests in multiple programming languages. It supports multiple browsers, including Chrome, Firefox, and Internet Explorer [7]. Selenium's main purpose is functional testing, but it is also good for web scraping. For effective functional testing, it mimics what a human would do in a browser. Selenium uses three parts: browsers, a driver for each browser, and the proper package depending on the programming language used [7].

For our project, we used Python to implement an automated attestation script. Selenium was helpful for clicking the appropriate buttons to fill out and submit the form.

## 3.3 Atlas Security Analysis

The campus attestation form submission relies on Atlas user account frameworks. Gibson, Hutchinson, Millis, and Quintero wrote a report on analyzing the security of Atlas.mit.edu for their 6.857 final project in Spring 2019 [8]. They analyzed potential attacks and provided recommendations to IS&T for improvements to the Atlas platform. They explored a threat model that focused on a malicious actor who has access to a valid MIT Kerberos account and access to Atlas. Gibson et. al. advised IS&T to stop leaking vendor software versions, properly validate user input, update all domains within the Atlas ecosystem, and include support for editing profile information through Atlas [8]. Analogously, we provide recommendations to improve the security of COVIDPass.

# 4 Implementation

We wrote a script, fully authenticated for Atlas users, that can automate daily attestation answers and submit them every 24 hours. Duo authentication needs to be done with a secondary device such as a phone every 30 days.

## 4.1 Materials

The necessary materials include a computer (which we borrowed from IS&T), a phone for Duo authentication, and a remote server such as an Amazon Web Services (AWS) Elastic Compute Cloud (EC2) instance.

## 4.2 Script

We successfully coded a Python script that is automated and can run the attestation every $x$ seconds (user chooses integer $x$) as long as the laptop stays open. There were two different methods that we tried: an HTTP POST request and a graphical approach.

The first method involved inspecting the HTML and Javascript elements on Chrome in hopes of gathering the appropriate data to replicate the same HTTP POST request that goes through when users submit the form. We were able to find the body of the HTTP request which contained the JSON representing the series of YES and NO answers that you clicked on the attestation. We also noticed that there was a bearer token that was included in the header of the POST request. We were successfully able to replicate this request using the bearer token as the authorization token. However, one hour into running the script, that bearer token changed. Unfortunately, we didn't figure out a way to automatically grab the new bearer tokens each time, so we tried a different approach instead.

Our second approach, which we successfully implemented, was graphical. We used Selenium, which is a Python library that does web scraping. With the help of Selenium, we were able to program our script to automatically click on the proper graphical elements of the COVIDPass website. We placed this Selenium library call inside a helper function that would automatically open up the Chrome browser every $x$ seconds and then close the browser after the Selenium helper function was done running. The final sequence of events would be

1. Open the Chrome browser.
2. Run the Selenium code.
3. Close the Chrome browser.

For our in-class presentation, we demonstrated this working twice with a 5 second delay in between each round. If a deceitful student actually wanted use this script, they would just have to change the 5 second delay to be 23 hours and 30 minutes to avoid flagging any admins who might notice an exorbitant amount of attestations per day compared to the average user.

By using this script, users would be able to bypass the need to manually do the attestation every day, and thus not be thinking about whether they actually have symptoms or not. All they would need to do is answer their phone for Duo Factor Authentication every 30 days because of the use of their MIT certificate in Chrome.

If you would like to watch a demo video of our exploitation, you can do so at this YouTube link: `https://www.youtube.com/watch?v=3rZe74NG2VE&t=1s`

## 4.3 AWS Server Hosting

Although we were successful with implementing the automation script, there was still room to take it to the next level, though we ultimately had trouble with this part of the project. To successfully automate the attestation form for days on end, the script would need to run constantly on the user's local machine. For example, a user could set up the form to automatically submit every 23 hours and

30 minutes. Though this is viable, it would increase the burden on a machine belonging to a typical student. Ideally, the automation script would be run remotely instead. Given its ubiquity and wide range of different services, we chose to try to use Amazon Web Services (AWS) for this part of our task.

There were two general approaches that we considered to run the script remotely using AWS:

1. Host the script locally and run the terminal command on a remote server.
2. Simultaneously host and run the script remotely.

The first option was preferable because it is more convenient to keep the code on a local machine. However, it was difficult to find services that offered to run locally hosted files without first hosting them on a remote server, and we did not end up finding anything that was viable and did not necessitate considerably changing our implementation approach.

As for the second option, it came with another set of challenges. While there were several options via AWS that could host and run a script remotely, such as creating an EC2 instance that a user could SSH into, there was an important security obstacle. To open the attestation form on a specific machine, it is necessary for the machine to possess MIT certificates (or to use two-factor authentication, but that defeats the purpose of our script, as automating that type of login is beyond the scope of our project). Although Amazon smoothly manages integration of its own potential certificates throughout its web services, it makes it difficult to use third-party certificates on its servers.

# 5 Security Analysis

To analyze the security of COVIDPass, we first generated a security policy to comprehensively outline the scope of the system. We provide the relevant permissions and responsibilities for each actor. Then we list the security goals of COVIDPass.

## 5.1 Security Policy

**Security Policy Assumptions:**
We made the following assumptions when outlining the security policy below.

- The COVIDPass roster is accurate and up-to-date at all times.
- Everyone uses Touchstone and Duo authentication to login.
- Building access is updated at precisely :00 and :30 every hour.

We list the COVIDPass permissions and responsibilities for each type of actor in the MIT community.

1. **Administration (IS&T)**
    - Permissions: View/modify attestation submissions, attestation forms and push updates to COVIDPass, electronic test results, and users in the COVIDPass system.
    - Responsibilities: Ensure the availability of the COVIDPass system and correctness of the data within the COVIDPass system.
2. **Supervisors**
    - Permissions: Designate students/faculty to have access to specific buildings (under the supervisor's purview).
    - Responsibilities: Maintain the list of students/faculty to have COVIDPass access for specific buildings.

3. **On-Campus Students/Faculty/Staff**
    - Permissions: View accessible buildings and learning spaces, in-person classes, labs they work in. Receive COVID test reminders and results. Submit COVID Daily Attestation. Receive reminders to submit COVID Daily Attestation.
    - Responsibilities: Get tested twice a week. Follow the rules outlined by administration. Fill out the attestation form at least once every 24 hours.
4. **Off-Campus Students/Faculty/Staff**
    - Permissions: Sign up for COVIDPass to gain campus access to specific buildings. If campus access is granted, the permissions are the same as those for on-campus students/faculty. If campus access is not granted, they only have permission to get one COVID test per week using the same app functionality.
    - Responsibilities: If campus access is granted, the responsibilities are the same as those for on-campus students/faculty.
5. **Bystanders (anyone outside the MIT Community)**
    - No permissions/responsibilities granted.

### 5.2 Security Objectives

To address all aspects of security, the COVIDPass system aims to maintain the following principles:

- **Confidentiality:** Protect user information and prevent other users and adversaries who do not have HIPAA permission from accessing the user information.
- **Integrity:** Record and convey accurate user information, and ensure that users who use the system are properly authenticated.
- **Availability:** Update the database every 30 minutes and give building access to appropriate supervisors and users.

### 5.3 Authentication

The COVIDPass system requires an MIT certificate, Touchstone, and Duo authentication for users to login and fill out the attestation form. The system uses Kerberos and MIT ID numbers as unique identifiers. Ideally, additional hash IDs would be employed within the COVIDPass database to anonymize user data. We suggest an additional security measure in the form of hash functions applied to the body of the JSON request.

## 6 Vulnerabilities

COVIDPass is vulnerable to passive and active adversaries. Passive adversaries could obtain leaked information such as ID numbers, birthdays, dorms, and test results. Active adversaries could attempt automate the attestation form or hack the back-end of the COVIDPass system to obtain unauthorized building access or tamper with confidential information.

### 6.1 Adversarial Attacks

We consider standard web technologies and their generic vulnerabilities and plausible attacks. The work-at-home model during COVID-19 and virtualization of classes and work has increased the vulnerability of unsecured WiFi and poor access controls [2].

Contact-tracing apps are especially vulnerable to adversarial attacks in the form of phishing schemes. Hackers often misrepresent official tracing accounts via email. Moreover, in their perspective, widely used apps contain a lot of sensitive data with individual names and addresses as well as insights from contacts and location-tracking. In the context of COVIDPass, hackers with malicious intent could potentially change building access permissions, extract sensitive information, or modify the attestation form.

In summary, an adversary may attempt to:

- Gain unauthorized access to MIT buildings.
- Obtain COVIDPass user information.
- Tamper with the attestation form itself.

An adversary could be an administrator, student, bystander, or remote non-user who attempts to hack the back-end of the COVIDPass system.

### 6.2 Liveness

*Liveness* is the property of having a living, authorized user [6]. Most protocols require active responses, but if an adversary had a playback that could play over and over, they could trick the system. Our implementation proves that it is easy to fake liveness with an automated script. This lack of liveness would harm the accuracy of the COVIDPass attestation form results if a user felt symptoms but let the script automatically report that they were perfectly healthy.

Students, faculty, and staff are incentivized to fill out the daily attestation form to avoid getting locked out of dorms and buildings that they have permission to enter. Forgetting to submit the form means waiting up to 30 minutes for the database to update and renew their building access. Therefore, users may be motivated to write to write an automation script like we did (see Section 4) to bypass liveness.

### 6.3 Building Access

The building access vulnerabilities range from using an automated script to sustain building access to hacking the back-end to gain unauthorized building access. Our implementation in Section 4 demonstrates how a user would automate attestation responses for consistent building access. As for hacking the back-end, it is difficult to say how hard it would be without knowing the code and architecture of COVIDPass. If an adversary did successfully access and modify the back-end code, they would have access to a lot of sensitive information including MIT ID number, birthday, test results, and living group information. Therefore, it is critical for IS&T to make COVIDPass as secure and difficult to hack as possible.

## 7 Countermeasures

We analyze four types of countermeasures against adversaries hacking COVIDPass.

1. Hash Functions
2. Digital Signatures
3. CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart)
4. Altering Questions or Order of Questions

## 7.1 Hash Functions

Cryptographic hash functions are deterministic, one-way, collision-resistant, and pseudo-random. We can add an additional layer to Touchstone and Duo authentication by requiring users to enter a password that is verified with a hash computation. Secure hashes are excellent for verifying message integrity. Comparing message digests (hash digests over the message) calculated before, and after, transmission can determine whether any changes have been made to the COVIDPass attestation form.

MD5, SHA-1, or SHA-2 hash digests are sometimes published on websites or forums to allow verification of integrity for downloaded files. This practice establishes a chain of trust as long as the hashes are posted on a trusted site – usually the originating site – authenticated by HTTPS. Using a cryptographic hash and a chain of trust detects malicious changes to the file. Other error-detecting codes such as cyclic redundancy checks only prevent against non-malicious alterations of the file made by others [3].

Specifically, one of the above mentioned hash algorithms could be used on the body of the request sent with the form submission data by hashing a string representation of the JSON object in the body. Adversaries who would want to manipulate form data by intercepting the request would not be able to do so, and thus the integrity of this data is maintained.

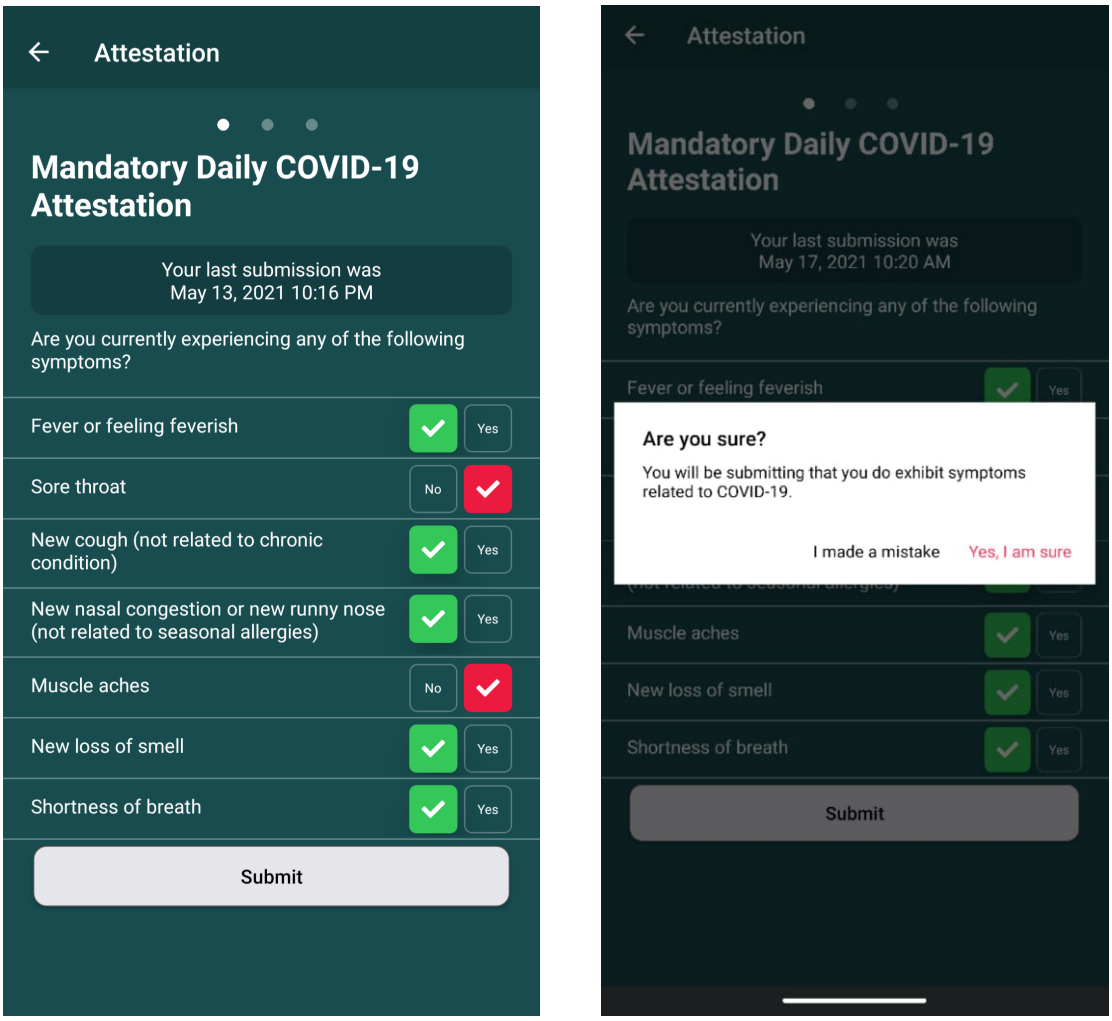
## 7.2 Digital Signatures

Digital signatures are a countermeasure against forgers. When a someone electronically signs a document, the signature is created using the signer's private key. The mathematical algorithm acts like a cipher, creating data matching the signed document, called a hash, and encrypting that data. The resulting encrypted data is the digital signature. MIT certificates act as digital signatures for authentication.

## 7.3 CAPTCHA

Another potential countermeasure is CAPTCHA, which stands for "Completely Automated Public Turing test to tell Computers and Humans Apart" [4]. CAPTCHA is a program which can generate and grade the tests that AI programs (including itself) have trouble passing. CAPTCHA is secure if none of the computer program should be able to pass the tests generated by it even with functional knowledge of the CAPTCHA. The effectiveness of CAPTCHA of a given strength is determined by how frequently the guesses of CAPTCHA can be tested by an attacker.

The universal structure for a segmentation attack is framed to analyze security of CAPTCHA. Segmentation attacks extract characters from image along with its position in image. Segmentation attacks are applicable to most of characters (i.e. connected, overlapped, and disconnected characters) present in CAPTCHA images. If IS&T added CAPTCHA authentication that was secure against segmentation attacks, then this would help ensure the liveness of users and integrity of attestation responses [5].




(a) (b)

Figure 1: (a) Sample attestation form with wrong responses. The red buttons are clearly noticeable to the users.
(b) Warning message shown when user attempts to submit a response that indicates they have COVID-19 symptoms.

### 7.4 Altering Questions

The final countermeasure we propose concerns the actual contents of the app. Though not explicitly a vulnerability, one of the major weaknesses of the application is that the questions and ordering are deterministic and always the same; specifically, the *correct* sequence of answers (those that indicate good health) is the same at every instance of the attestation form. At this point, most students have memorized the sequence of nine NOs and two YESes that indicate that they are healthy, and this specific, known sequence made it fairly simple for us to write a script to automate the answers.

We propose changing up the questions and correct answer sequence in every instance of the attestation form using two different methods.

1. Mechanism (1): Altering the order of the questions on a randomized basis
2. Mechanism (2): Changing the wording to reflect the same meaning but flip the correct answer (i.e. alternate using *"Do you agree to wear a face covering or mask while engaging in activities on campus"* and *"Will you engage in activities on campus without wearing a face covering or mask"* for the same question).

Using mechanism (1) alone will mean that any automation script will have to base its answers on the text of the question directly to the left hand side instead of the pre-recorded answer sequence, which makes it a bit harder for an adversary to succeed. Adding mechanism (2) as well makes it even more challenging for the adversary, since they will have to account for analyzing different wording for each question to determine the proper YES/NO answer. Overall, both of these mechanisms reduce the number of repetitive user actions that are easy to automate.

However, though it heightens security, question reordering may hinder the user experience, as students could get annoyed at being locked out because they filled the form incorrectly due to muscle memory. Although question reordering is inconvenient for users, we believe that the trade-off between usability and security is worth it, especially since there are methods of thwarting attacks that can be still user-friendly.

In fact, IS&T already has robust checks in place. As shown in Figure 1, IS&T would color the button in red so that users are alerted if they answer any questions incorrectly when they fill the form out manually. For a student filling out the form manually, this would be an easy way to fix the answer and submit the correct response, yet an automated script would not register that change and thus be more likely to get the answers incorrect.

Another check that IS&T has implemented is a pop-up window: if you try to submit the form with an answer that indicates you have symptoms, the form will give you a warning as shown in the screenshot. For any user that missed the glaring red button, this would allow them to check their answers again and easily retry submitting. An automated script might find this warning to be an extra obstacle, or it might actually find it useful and use a brute-force tactic to try many different YES/NO sequence combinations until the submission was accepted without a warning. This method could be countered by adding a limit on the number of times a user could retry their submission, which effectively satisfies user experience, probably takes minimal effort to configure, and implements an important security barrier.

## 8 Advisory to IS&T

Based on our successful implementation of an automated script, we want to recommend some changes to promote the security of COVIDPass. To make COVIDPass more secure, we recommend adding CAPTCHA and changing the order of the attestation questions. CAPTCHA will make it difficult for

the automated script to submit the form, and question reordering will confuse the script and make it difficult to select the correct answers. Since our implementation approach used Selenium to select the pixels corresponding to the proper HTML buttons, the script would have to undergo trial and error to figure out the correct healthy sequence. Thanks to the button coloring and warning message discussed in subsection 6.4, question reordering is easy for people to adapt to, but it is hard for the script to guess the order of the questions and answers.

We concede that MIT certificates provide a high level of security. We could not find a workaround for running the automated script on an AWS server with certificates in place. Therefore, COVIDPass is already a very secure system given its user-friendliness and functionality for keeping the MIT community safe and healthy.

# 9 Contributions

Amanda and Irin implemented the Python script on the loaner laptop that we borrowed from IS&T (Section 4). Alex researched AWS remote server hosting (Subsection 4.3) and the implications of question reordering (Subsection 7.4). Julia analyzed the security and vulnerabilities (Sections 5 and 6). We all contributed equally to brainstorming potential countermeasures and how to implement them (Section 7), creating and giving the presentation, and writing this paper.

# 10 Conclusion

Accordingly, we have successfully demonstrated that it is possible to write an script that will automatically complete the required daily attestation form for building access. Though the barrier of MIT certificates made it challenging to run the script on a remote server, the automation script could still be run on a local machine for days on end at times within the user's control. This benign attack defies the goal of the COVIDPass application to get live, manual data from real users.

MIT certificates (a variant of digital signatures), Touchstone login, and Duo authentication provide secure authentication. As a result, COVIDPass is well-crafted compared to many contact-tracing and pandemic-monitoring apps, yet there are still many vulnerabilities and plausible attacks. To maintain integrity of the attestation form data, we propose applying hash functions with salts. As for the primary goal of our exploration, ensuring the liveness of the form data, we suggest using CAPTCHA and altering the question order and phrasing as possible countermeasures to make it considerably harder for an adversary to automate filling out the attestation form. We hope that our advisory will help IS&T improve the confidentiality, integrity, and availability of COVIDPass.

# 11 Acknowledgements

We would like to thank Professor Ron Rivest and Professor Yael Kalai; the 6.857 TAs Andrés Fabrega, Billy Woltz, and Deep Gupta; and the LAs Kyle Hogan and Mike Specter. We are also very thankful toward IS&T specialists Olu Brown, Dudley Kirkpatrick, and Juan Heyns.

# References

[1] IS&T Help Desk. Accessed 25 March 2021. `https://ist.mit.edu/help`

[2] "The Future of Hacking: COVID-19 shifting the way hackers work and who they target." https://www.securitymagazine.com/articles/93086-the-future-of-hacking-covid-19-shifting-the-way-hackers-work-and-who-they-target

[3] Wikipedia. "Cryptographic Hash Function." https://en.wikipedia.org/wiki/Cryptographic_hash_function

[4] "CAPTCHA Official Site." http://www.captcha.net

[5] "CAPTCHA Security Analysis." https://www.researchgate.net/publication/287060062_Security_analysis_of_CAPTCHA

[6] Thales Group. https://www.thalesgroup.com/en/markets/digital-identity-and-security/government/inspired/liveness-detection

[7] "Web Scraping With Selenium: DIY or Buy?." https://oxylabs.io/blog/selenium-web-scraping

[8] "Atlas.mit.edu Security Analysis." http://courses.csail.mit.edu/6.857/2019/project/4-Gibson-Hutchinson-Millis-Quintero.pdf