ALL THAT WE LET IN:
HACKING 30 MOBILE HEALTH APPS AND APIS

According to Mobius MD, there are now over 318,000 mHealth apps available in major app stores. Over 60 percent of people have downloaded an mHealth app, which is now more common of a smartphone activity than online banking, job searches, or accessing schoolwork or educational content (Pew Research, 2015). With the pandemic pushing more patients towards virtual visits with their family physician and mental health provider, hackers have begun shifting their attention to this new attack surface in search of protected healthcare information (PHI) which is now demanding more of a payout per record than credit card numbers on the dark web.

Summary

This paper details the results of a 6-month long vulnerability research campaign into the compromise of 30 mobile health apps and APIs to demonstrate a systemic lack of hardening of mHealth apps and APIs to sufficiently secure protected healthcare information (PHI).

Author Information

Alissa Valentina Knight
Partner
Knight Ink
1980 Festival Plaza Drive
Suite 300
Las Vegas, NV 89135
ak@knightinkmedia.com

Publication Information

This white paper is sponsored by Approov.

Initial Date of Publication:
February 03, 2021
Revision: 1.0

TABLE OF CONTENTS

06

- **Key Points**
- **Introduction**

15

- **The Rise of Mobile Health**

17

- **APIs and Microservices**

TABLE OF CONTENTS

20
- **API Security**
- Authenticating Requests
- Authorizing Requests

21
- **How APIs Are Breached**
- Broken Object Level Authorization

23
- **The Research**
- Tactics, Techniques, and Tools
- Company Profiles
- Mobile Applications

TABLE OF CONTENTS

28

- **The Findings**
- Mobile Applications
- APIs

30

- **The Solution**
- Understanding Synthetic Traffic
- Shift-Left Security
- Perform Penetration Testing
- Certificate Pinning

33

- **Peer Reviewed Evidence**
- Exhibit A
- Exhibit B
- Exhibit C
- Exhibit D
- Exhibit E
- Exhibit F
- Exhibit G
- Exhibit H
- Exhibit I
- Exhibit J
- Exhibit K
- Exhibit L
- Exhibit M
- Exhibit N
- Exhibit O

TABLE OF CONTENTS

46

- **Conclusion**

48

- **Sources**

50

- **About the Author**
- About Knight Ink

KEY POINTS

This section outlines the salient points from this paper. While it’s my hope you’ll read this paper in its entirety, this section attempts to summarize this paper’s key points and research findings but should not be considered all encompassing.

- This paper was written for chief information security officers (CISOs) and cybersecurity engineers who want to better understand the tactics, techniques, and procedures (TTPs) adversaries use to breach these APIs and mHealth apps that lead to PHI data leaks.
- Thirty mHealth apps and APIs were tested in this research through the cooperation of multiple companies who agreed to participate so long as none of the findings were attributed to their company.
- The purpose of this research is to prove through empirical data, the risk vulnerabilities in mHealth apps and APIs pose to mHealth companies and the PHI of patients they safeguard.
- Out of all thirty mHealth apps tested, 77% contained hardcoded API keys, some which don’t expire, and 7% even contained hardcoded usernames and passwords. 7% of the API keys belonged to third-party payment processors that warn about hard-coding their secret keys in plain text.
- According to Experian, a social security number will cost $1, a credit card up to $110, but full medical records can cost up to $1,000 per record. (Experian, 2017) [3].
- Out of the API endpoints tested, 100% of them were vulnerable to Broken Object Level Authorization (BOLA) attacks leading to unauthorized access to full patient records, downloadable lab results and x-ray images, blood work, allergies, and personally identifiable information (PII) including home addresses, family member data, birthdates, and social security numbers.
- Shift-left security should not be considered a panacea. Rather, combining shift-left security with *shield-right to* prevent synthetic traffic from reaching APIs.
- Generally speaking, APIs are an intermediary between applications that defines how they can talk to one another, decoupling the consuming application from the infrastructure.
- Some of the largest companies involved in this research range from $600 Million to $8 Billion in annual revenues with diluted earnings per share (EPS) of $.50 to $3.00 and NET Operating Cash Flow of $1 Billion to $3 Billion. The average number of employees for all companies tested was 15,000.
- The average number of downloads for each app tested was 772,619. Each app combined capabilities of allowing clinicians to review their schedule, patient lists, and patient charts.
- The findings demonstrate that the security standards required for compliance with US government FHIR/SMART standards merely represent a subset of the steps needed to secure mobile apps and the APIs which enable apps to retrieve data and interoperate with data resources and other applications.

KEY POINTS: STATISTICS

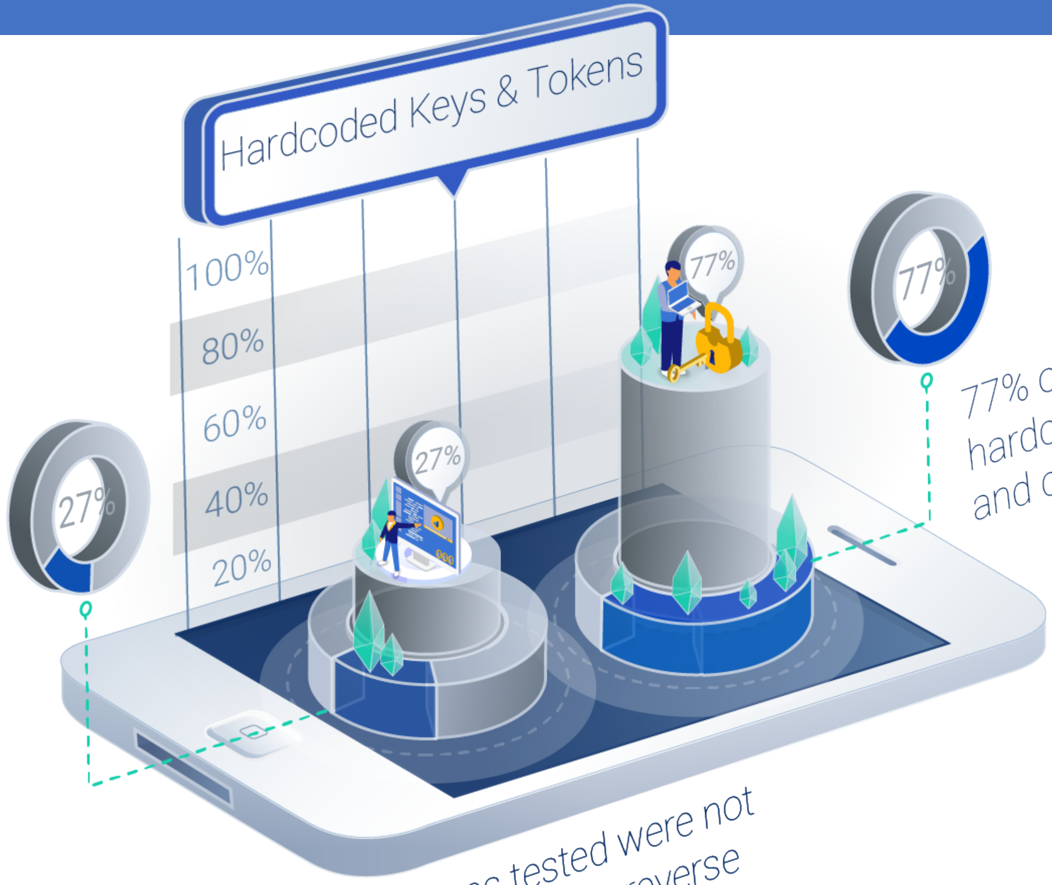
Mobile Apps

- 27% of the apps tested were not secured against reverse engineering through code obfuscation
- 77% of the apps tested contained hard-coded API keys, tokens, private keys, and hard-coded usernames and passwords (Exhibit D)
- 100% of the apps tested failed to implement certificate pinning, enabling me to be able to perform woman-in-the-middle attacks against the app (Exhibit J)
- 114 hardcoded API keys and tokens were found for authenticating with the mHealth company and third-party APIs
- API keys and tokens were discovered for Google, Branch.io, Braze, Tune, Optimizely, Cisco Umbrella, Microsoft App Center, Bugsnag, Contentful, Stripe, Amazon AWS, Radaee, Sendbird, AppsFlyer, Facebook, Vonage, SalesForce, Mparticle
- 7% of the apps tested contained hardcoded keys for third-party payment processors (Exhibit D)
- 63% of the apps contained hardcoded private keys

APIs

- 50% of the APIs tested allowed me to access the pathology, x-rays, and clinical results of other patients (Exhibit N, Exhibit O, Exhibit P)
- 50% of the APIs tested allowed me to access admissions records for patients being admitted into the hospital as in-patients that I shouldn’t have been able to access with my level of authorization
- 100% of the APIs tested were vulnerable to Broken Object Level Authorization (BOLA) vulnerabilities
- BOLA vulnerabilities in 100% of the APIs tested allowed me to view the personally identifiable information (PII) and protected healthcare information (PHI) for patients that were not assigned to my clinician account
- 50% of the APIs tested did not authenticate requests with tokens
- 50% of the records accessed contained names, social security numbers, addresses, birthdates, allergies, medications, and other sensitive data for patients
- A replay vulnerability allowed me to replay days-old FaceID unlock requests that allowed me to take over other users’ sessions

THE FACTS ON VULNERABILITIES IN MOBILE HEALTH APPS AND APIS



77% of the apps contained hardcoded API keys, tokens and credentials.

27% of apps tested were not obfuscated against reverse engineering.

SOURCES: KNIGHT INK, LLC
RESEARCHER: ALISSA V. KNIGHT
WEB: WWW.KNIGHTINKMEDIA.COM
EMAIL: INFO@KNIGHTINKMEDIA.COM

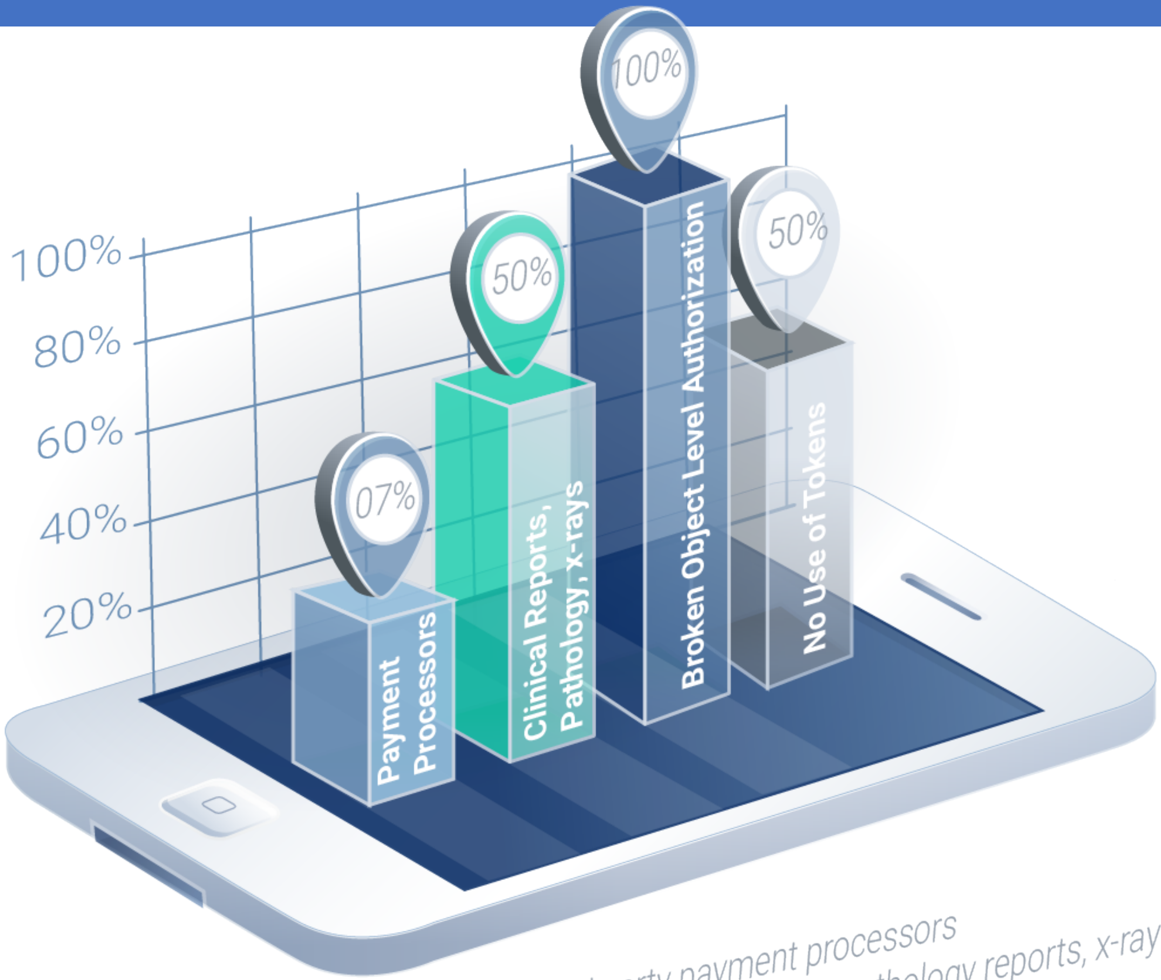
OFFICE:
1980 FESTIVAL PLAZA DR
SUITE 300,
LAS VEGAS, NV 89135

KNIGHTINK

RESEARCH SPONSORED BY:
approov

DOWNLOAD THE WHITE PAPER AT
WWW.APPROOV.IO/MHEALTH/HACKING

THE FACTS ON VULNERABILITIES IN MOBILE HEALTH APPS AND APIS



07% of apps contained hardcoded keys to 3rd party payment processors
50% of the APIs allowed unauthorized access to clinical reports, pathology reports, x-rays
100% of the APIs were vulnerable to broken object level authorization vulnerabilities
50% of the APIs did not implement tokens

SOURCES: KNIGHT INK, LLC
RESEARCHER: ALISSA V. KNIGHT
WEB: WWW.KNIGHTINKMEDIA.COM
EMAIL: INFO@KNIGHTINKMEDIA.COM

OFFICE:
1980 FESTIVAL PLAZA DR
SUITE 300,
LAS VEGAS, NV 89135

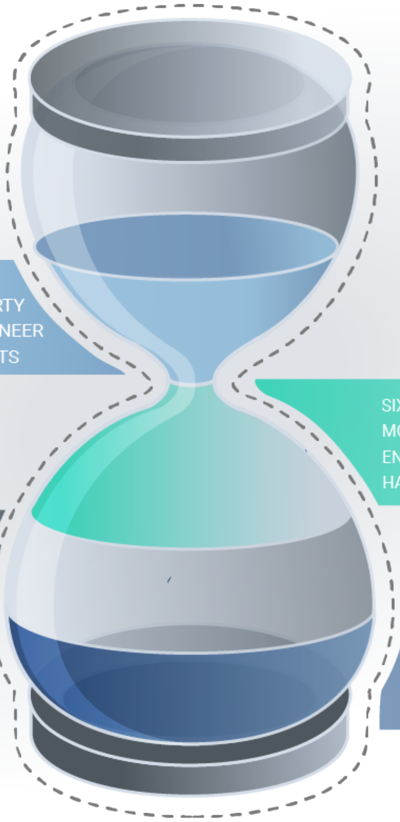
KNIGHTINK

RESEARCH SPONSORED BY:
approov

DOWNLOAD THE WHITE PAPER AT
WWW.APPROOV.IO/MHEALTH/HACKING

THE FACTS ON VULNERABILITIES IN

MOBILE HEALTH APPS AND APIS



05 APPROXIMATELY FIVE MINUTES WAS SPENT ON EACH OF THE THIRTY MHEALTH APPS TO REVERSE ENGINEER AND FIND HARDCODED API SECRETS

06 SIX MONTHS SPENT ANALYZING THIRTY MOBILE HEALTH APPS, REVERSE ENGINEERING THEM AND FINDING HARDCODED KEYS

01 IN LESS THAN ONE MINUTE BOLA VULNERABILITIES WERE DISCOVERED IN 100% OF THE APIS

02 100% OF THE APIS WERE COMPROMISED LEADING TO ACCESS TO PATIENT RECORDS AND PII IN TWO WEEKS

SOURCES: KNIGHT INK, LLC
RESEARCHER: ALISSA V. KNIGHT
WEB: WWW.KNIGHTINKMEDIA.COM
EMAIL: INFO@KNIGHTINKMEDIA.COM

OFFICE:
1980 FESTIVAL PLAZA DR
SUITE 300,
LAS VEGAS, NV 89135

KNIGHTINK

RESEARCH SPONSORED BY:
approov

DOWNLOAD THE WHITE PAPER AT
WWW.APPROOV.IO/MHEALTH/HACKING

INTRODUCTION

As of 2018, 84,000 app publishers were responsible for publishing approximately 318,000 mobile health (mHealth) apps available in the app marketplaces which have enjoyed annual downloads of over 3.7 Billion (Research2Guidance, 2017) [1]. This number has certainly increased since 2020, especially in this post-pandemic world where mHealth is quickly becoming the new norm for those needing to see their family physician or mental health professional without leaving their house. Just as a picture is worth a thousand words, numbers also speak volumes. According to a report published in April of 2020 by Reports and Data, the global mobile health market is forecasted to reach $311.98 Billion by 2027.

With so many mHealth apps collecting millions of protected healthcare information (PHI) records globally, a massive data lake of monetizable data is being generated. This growing attack surface is quickly drawing the attention of transnational crime syndicates wanting to lock-and-leak it in order to extort payments from its data owners and sell it to the highest bidder.

The shores of these massive data lakes are lined with APIs ready to serve it to requesting API consumers. However, how can mHealth companies know that the requestor is the legitimate mHealth app they created or synthetic requests generated by a tool?

This paper was written for chief information security officers (CISOs) and cybersecurity engineers who want to better understand the tactics, techniques, and procedures (TTPs) adversaries use to breach these APIs and mHealth apps that lead to PHI data leaks.

In 2019, I published a revealing report on the financial services and fintech industry after finding systemic vulnerabilities in 30 financial services and fintech mobile apps. [2] However, in that research, I stopped at the static code analysis and didn't target the APIs of the apps due to a lack of approvals.

In this year's research, I've downloaded 30 mHealth apps and partnered with some of the world's largest mHealth manufacturers in a coordinated partnership to perform penetration testing of their mobile apps and APIs. The purpose of this research is to prove through empirical data, the risk that vulnerabilities in mHealth apps and APIs pose to the business and the PHI of patients they safeguard.

As a CISO or security engineer for a mHealth company, your attack surface extends across two separate beachheads, the mHealth mobile app and every API endpoint it communicates with. Some organizations involved in this research had well over 1,000 APIs serving their applications. Whereas some CISOs enjoy the luxury of only having a perimeter that extends to the end of their ISPs demarcation point, your edge extends to every mobile device your mHealth app is installed on.

Out of all thirty mHealth apps tested, 77% contained hardcoded API keys, some of the tokens don’t expire, and 7% even contained hardcoded usernames and passwords. 7% of the exposed API keys belonged to third-party payment processors that warn about hard-coding their secret keys in plain text in their API documentation.

Out of the API endpoints tested, 100% of them were vulnerable to Broken Object Level Authorization (BOLA) attacks leading to unauthorized access to full patient records, downloadable lab results and x-ray images, blood work, allergies, and personally identifiable information (PII) including home addresses, family member data, birthdates, and social security numbers.

It's clear from the data collected, why there is such a high premium for PHI above the cost of credit card numbers on the dark web marketplaces. Simply put, it’s a lot easier for a bank to send you a new card because it’s been compromised or refund any fraudulent charges, but it’s a lot harder for someone to send you a new identity or invalidate what was your past medical history that is now for sale on the dark web.

According to Experian, a social security number will cost $1, a credit card up to $110, but full medical records can cost up to $1,000 per record. (Experian, 2017) [3].

Despite the attempted protections put into place on many of the apps, such as reliance solely on JWT tokens, geolocation binding, and checks if the app is being run on a rooted/jailbroken device, the attacks were still possible. All of the APIs tested failed to implement certificate pinning, allowing me to insert myself in the middle of the communication a la a woman-in-the-middle (WITM) attack after I logged in.

The type of attacks I performed in this report did not require a jailbroken phone and geolocation blocking was possible to get around as well. JWT tokens were not effective because of the attack vectors as well as the absence of certificate pinning, which will be explained later.

This report uncovers these vulnerabilities, how they were exploited, and the resulting evidence of these findings in what I anticipate will be the most controversial report on the healthcare industry ever published. If you’ve never been convinced because of a lack of empirical evidence, that security needed to shift-left in your organization, here is your proof. But shift-left security should not be considered a panacea. Rather, Shift-left security should not be considered a panacea. Rather, combining shift-left security with shield-right to prevent synthetic traffic from reaching APIs.

Because the PHI records in these tests were real, they have been redacted to protect the sensitivity of the data. If any of the screenshots in the annexes are blurry, it isn’t because of a low-resolution image. They were blurred intentionally in order to protect the identity of the patient’s record.

THE RISE OF MOBILE HEALTH

The terms telehealth, telemedicine, remote patient monitoring (RPM), and mobile health (mHealth) are mistakenly used interchangeably when in fact they all encompass different things. Much like machine learning is mistakenly conflated with artificial intelligence when in fact it's a subset of AI. mHealth is a subset of the broad category of health technologies, telehealth.

mHealth refers to the use of mobile technology to achieve improved health. As defined by the World Health Organization (WHO), mHealth refers to “use of mobile and wireless technologies to support the achievement of health objectives." [4]

In our new mobile economy where people prefer a cell phone or tablet over a laptop, mHealth represents a very specific type of telehealth driven by our new mobile app economy. Telehealth in the broader sense refers to the use of technology to improve healthcare outcomes, while mHealth refers to the specific use of mobile technology and apps for patients to acquire their own health information without the intervention of a clinician. I, however, subscribe to the more all-encompassing definition of mHealth since there has been no widespread agreement on just what mHealth encompasses. As described by InnovateMedTech, “mHealth refers to the practice of medicine and public health supported by mobile devices such as mobile phones, tablets, personal digital assistants and the wireless infrastructure. Within digital health, mHealth encompasses all applications of telecommunications and multimedia technologies for the delivery of healthcare and health information that caters to both clinicians and patients.” [5]

Accordingly, “there are different categories of mHealth apps in the market, to include symptom checkers, clinical records management, self monitoring, rehabilitation programs, prescription filing, communication with a patient’s doctor or mental health professional, and more.

With so many different data categories of patient data, it’s no wonder that almost half (48%) of consumers now prefer to take the control of their own health into their hands with mHealth apps using mobile devices and wearables as compared to just 16% in 2014 according to a 2018 report by Accenture. [5]

KNIGHTINK

Calling a doctor

APIS AND MICROSERVICES

Behind every mHealth app on a mobile device is a service translating each request sent to it from the app called an application programming interface (API). The API is responsible for processing every request from the mHealth app and either providing the results of the request to the app or inserting data from the app into the backend database.

Generally speaking, APIs are an intermediary between applications that defines how they can talk to one another -- decoupling the consuming application from the infrastructure.

I'll use the analogy of a restaurant to better explain what an API is. When you sit down and order food at a restaurant, you're making specific requests to the waiter on what it is you want based on the "options" in the menu. In our case, you can think of yourself as the mHealth app, the waiter (the API) and the menu is the supported API options on what you can request. The waiter then takes your order to the kitchen (the database) and retrieves your order from the chef, who then brings it back to you.

There are different API architectures, internal app-to-app and human-to-app also referred to as the "last mile." Human-to-app architectures can be considered many-to-one or many humans/apps to one service. The latter architecture was in scope of this research and not the former as internal server-to-server communication breaches assumes a beach head or foothold on the internal network. Additionally, internal microservice to other microservices or 3rd party services are relatively few-to-few which constrains the security problem compared to the human to service APIs.

KNIGHTINK

API SECURITY

In order to secure an API effectively, you want to implement authentication and authorization to ensure that whatever application or device is sending an API request has the right to do so (authentication) and is allowed to read or write the data (authorization).

Authenticating Requests

API requests can be authenticated a number of ways. Unfortunately, some authentication mechanisms are more secure than others and while this is widely known, insecure authentication methods such as the use of Basic Auth or API keys are still being used.

Types of authentication in APIs includes, API keys, a long string of random numbers and characters generated by the API endpoint that grants access to whomever passes it in the authorization header of the request; Basic Auth where a username and password are used to authenticate an individual; JSON Web Tokens (JWTs), and OAuth, which uses tokens instead of sharing credentials; OAuth2, which exchanges a username and password for a token; SMART, which is increasingly becoming an implementation of OAuth in healthcare, and OpenID Connect. There are also other methods of authentication, such as implementing MFA through third-party solutions.

Authorizing Requests

Think of authentication simply as proving who you are while authorization proves you're allowed to see the information being requested. Just because you're authenticated, it doesn't necessarily mean you should be able to see the data you're requesting. For example, requesting the patient records of another patient instead of your own.

One of the most common methods of implementing authorization in an API is using Auth0. Per Auth0 documentation, authorization can be determined through the use of policies and rules, which can be used with role-based access control (RBAC). Regardless of whether RBAC is used, requested access is transmitted to the API via scopes and granted access is returned in the issued Access Tokens. [13]

HOW APIS ARE BREACHED

Gartner predicts that by 2022, API abuses will become the most common attack employed against web applications. [7] The three most common tactics and techniques used by adversaries to breach APIs affect authentication, authorization, and availability.

For authentication attacks, adversaries combine dumped credentials from previous breaches in an account takeover (ATO) style attack referred to as credential stuffing where usernames and passwords are sent to the API until a successful authentication is established. This is also called brute forcing.

The next type of attacks against APIs affects authorization vulnerabilities. Simply authenticating an API with either a legitimate account or with an API key or token, doesn't mean that the individual is authorized to read or write the data. An example of an authorization vulnerability is number one on the OWASP API Top 10 list, Broken Object Level Authorization (BOLA). This vulnerability is also referred to as Insecure Direct Object Reference (IDOR) whereupon a successfully authenticated API request asks to read or write data that doesn't belong to the authenticated user. For example, a patient being able to request the patient records of another patient from the API endpoint.

The third type of attack affects availability of the endpoints using a Denial of Service (DoS) attack. A DoS attack renders the API endpoint unusable for legitimate requests by overloading the API endpoint with synthetic API requests in order to knock it offline. While some DoS and Distributed Denial of Service (DDoS) attacks are volumetric in nature (overwhelming the API endpoints with more requests than they can handle), many of the DoS attacks simply exploit bugs in the API endpoint that can also render it unable to respond to new requests.

In this research, all of the APIs tested were vulnerable to BOLA attacks, which is demonstrated in the findings section[1] of this report and accompanying video.

Broken Object Level Authorization

It should come as no surprise that the BOLA vulnerability made the 1st place position in the OWASP API Top 10 list. BOLA vulnerabilities have now become the most popular and widely seen method of API abuses in the wild.

Simply put, a BOLA vulnerability enables an adversary to substitute the ID of a resource with the ID of another. When the object ID can be directly called in the URI, it opens the endpoint up to ID enumeration that allows an adversary the ability to read objects that don't belong to them. These exposed references to internal implementation objects can point to anything, whether it's a file, directory, database record, or key.

For example:

Substituting patientID 1001 in 'GET /patients/1001/lab_results' with ID 2001 in 'GET /patients/2001/lab_results'

[1] All That We Let In: Hacking mHealth Apps and APIs: https://alissaknight.co/hacking_mhealth_trailer

THE RESEARCH

TRANSPARENT DEDUCTION OF RESULTS & DETAILED DESCRIPTION OF METHODS USED

THE RESEARCH

In this section, I present the company profiles of the mHealth companies who participated in this research with me without directly identifying the companies or their apps.

In addition to the company profiles, I also present the static code analysis results of the mobile apps and the exploitation of vulnerabilities found in the API endpoints.

Initially, I present the findings from the API endpoints not secured with Approov, the company that sponsored this research. Then, I demonstrate the efficacy of the Approov solution using the same tactics and techniques I used to exploit them.

Please note. While I understand the sensitivity of the empirical data I'm unveiling in this report and its potential impact on the healthcare industry, none of the findings are attributable to the actual companies who participated in the research.

Tactics, Techniques, and Tools

Each mHealth app was reverse engineered using Mobile Security Framework (MobSF), an open source security framework designed to automate the static and dynamic code analysis of mobile applications, supporting APK and IPA file formats as well as zipped source code.

MobSF is a layered framework of different tools, one of which is apktool. Apktool handles decompiling and decoding of the compiled sources in an APK file. Know that an APK file is nothing more than a compressed zip file of resources and assembled java code.

Every APK will contain three files at a minimum:

- **AndroidManifest.xml,** which defines permissions for the application;
- **classes.dex**, which contains all the Java class files; and
- **resources.arsc**, which contains the meta-information about the resources and nodes. [8]

Once imported into MobSF, I then used grep combined with a number of different regular expression (REGEX) patterns at the command line against the MobSF uploads directory to find hardcoded secrets.

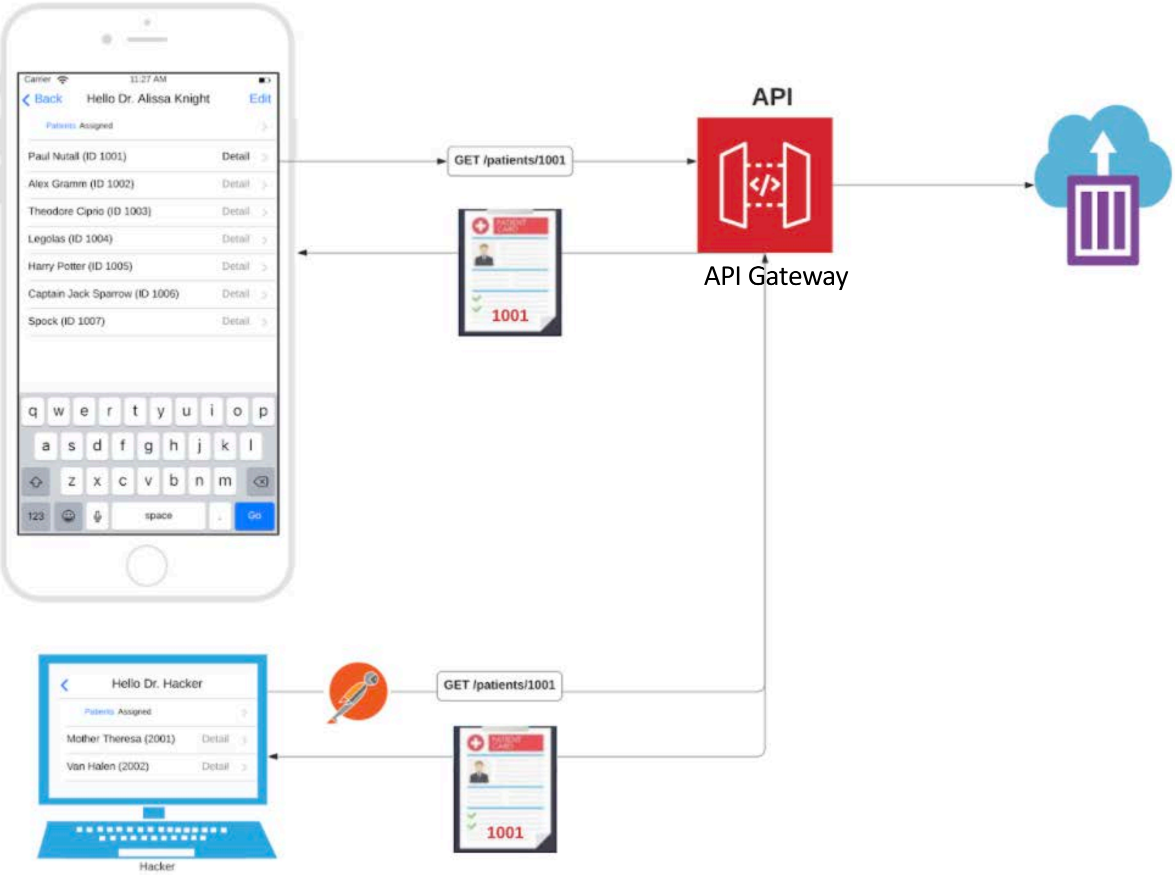
Once static code analysis was completed, I then targeted the APIs the mobile apps communicate with. In order to better understand how the APIs worked, I performed network interdiction with the apps using a stack of different tools, including Mitmproxy, Postman, and Burp Suite Pro that were configured with self-signed certificates in order to decrypt the SSL/TLS traffic used by the mobile apps. This was possible because the mHealth companies failed to implement pinning.

For penetration testing of the APIs, two separate applications were used. For creating custom API requests, I used Postman and for intercepting the mobile app traffic and replaying it to the APIs modified, I used Burp Suite Pro.

The perspective of the attacker in this research is access from the Internet requiring no internal access or beach head on the target network.

Figure 1 illustrates the attack lab setup/architecture.

Figure 1. API Attack Lab Architecture


Carrier
11:27 AM
Back
Hello Dr. Alissa Knight
Edit
Patients Assigned
Paul Nutall (ID 1001) Detail
Alex Gramm (ID 1002) Detail
Theodore Ciprio (ID 1003) Detail
Legolas (ID 1004) Detail
Harry Potter (ID 1005) Detail
Captain Jack Sparrow (ID 1006) Detail
Spock (ID 1007) Detail
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
Go
GET /patients/1001
1001
API
API Gateway
Hello Dr. Hacker
Patients Assigned
Mother Theresa (2001) Detail
Van Halen (2002) Detail
Hacker
GET /patients/1001
1001

Source: Knight Ink

Company Profiles

Numerous companies opened their mHealth APIs up for testing out of the 30 mobile apps tested.

The mHealth apps and APIs tested are leading suppliers of health care information technology ("HCIT") solutions and tech-enabled services. Participating company demographics include the United States, Europe, Asia, and South America. The mHealth apps combine clinical, financial, and administrative information management applications, including tools for managing electronic health records (EHRs) for patients and clinicians.

Some of the largest companies targeted in this research range from $600 Million to $8 Billion in annual revenues with diluted earnings per share (EPS) of $.50 to $3.00 and NET Operating Cash Flow of $1 Billion to $3 Billion. The average number of employees for all companies tested was 15,000.

Mobile Application Profiles

The average number of downloads for each app tested was 772,619. Each app combined capabilities of allowing clinicians to review their schedule, patient lists, and patient charts.

Additionally, some apps enabled clinicians to manage physician handoffs, access patient demographics and photos, and manage transfer of patient care between providers. Some apps allowed clinicians to review, add, and modify patient histories, problems, and allergies.

Clinician accounts could review radiology reports, clinical results, and pathology reports and were also able to create and edit prescribed medications and review prescription orders and other medication history for assigned patients as well as enable patients to connect with board-certified doctors for phone or video visits and request and fill prescriptions by their doctors.

Patient accounts were able to create and update their entire medical history as well as maintain a record of all medical information, clinical and pathology reports, x-rays, and other pertinent protected healthcare information (PHI) for the patient.

All of this data was accessible to me by exploiting BOLA vulnerabilities in the APIs.

Many of the apps included access to not just medical physicians, but also psychologists, psychiatrists, and other mental health therapists who were also able to prescribe medications including anti-psychotics and even integrated with Apple HealthKit allowing patients to take and record vital health information, such as blood pressure and heart rate.

In addition to mobile API testing, I also tested APIs built on the open Fast Healthcare Interoperability Resources (FHIR) (pronounced fire) standards framework.

FHIR was built by the Health Level Seven International (HL7) healthcare standards organization to provide for the exchange of electronic health records (EHR). These types of APIs are partner-facing, allowing third-party companies to develop mobile apps for their FHIR APIs.

The power behind FHIR is its ability to allow mHealth companies to expose discrete data elements as services via a RESTful API protocol acting as a translator between legacy healthcare systems and healthcare providers and individuals and their mobile devices built by third-party developers. One common integrator into FHIR APIs is the well-known personal health app, Apple Health. This EHR data can include patient and admissions records, diagnostic reports, and medications. [9]

THE FINDINGS

THE PRIMARY RESEARCH FINDINGS WITHOUT BIAS OR INTERPRETATION.

THE FINDINGS

Mobile Applications

This section details the findings of the static code analysis of the mobile apps without bias or interpretation. Any findings from the static code analysis were then carried-over and used in the API findings when possible. Examples of this information to complete the kill chain was the discovery of hard-coded credentials or API keys and tokens inside the mobile apps.

Findings Summary:

Mobile Apps

- 27% of the apps tested were not secured against reverse engineering through code obfuscation
- 77% of the apps tested contained hard-coded API keys, tokens, private keys, and hard-coded usernames and passwords (Exhibit D)
- 100% of the apps tested failed to implement certificate pinning, allowing me to perform woman-in-the-middle attacks against the app (Exhibit J)
- 114 hardcoded API keys and tokens were found for authenticating with the mHealth company and third-party APIs
- API keys and tokens were discovered for Google, Branch.io, Braze, Tune, Optimizely, Cisco Umbrella, Microsoft App Center, Bugsnag, Contentful, Stripe, Amazon AWS, Radaee, Sendbird, AppsFlyer, Facebook, Vonage, SalesForce, Mparticle
- 7% of the apps tested contained hardcoded keys for third-party payment processors (Exhibit D)
- 63% of the apps contained hardcoded private keys

APIs

- 50% of the APIs tested allowed me to access the pathology, x-rays, and clinical results of other patients (Exhibit N, Exhibit O, Exhibit P)
- 50% of the APIs tested allowed me to access admissions records for patients being admitted into the hospital as in-patients that I shouldn't have been able to access with my level of authorization
- 100% of the APIs tested were vulnerable to Broken Object Level Authorization (BOLA) vulnerabilities
- BOLA vulnerabilities in 100% of the APIs tested allowed me to view the personally identifiable information (PII) and protected healthcare information (PHI) for patients that were not assigned to my clinician account
- 50% of the APIs tested did not authenticate requests with tokens
- 50% of the records accessed contained names, social security numbers, addresses, birthdates, allergies, medications, and other sensitive data for patients
- A replay vulnerability allowed me to replay days-old FaceID unlock requests that allowed me to take over other users' sessions
- 100% of the accessible records included admissions records for the hospitals of all processed in-patients

THE SOLUTION

Organizations who hire developers to write code need to invest in secure code training for their developers. It's an absolute imperative that security shift-left in the software development life cycle instead of waiting until the code is pushed to the app store or placed into production. While many organizations invest in requiring security awareness training from all employees, separate training modules should be purchased for developers to help them write more secure code.

In addition to training, solutions which add security without changing development flows are most likely to be embraced by product teams. Tooling such as Approov's API threat protection is a SDK and cloud service providing frictionless enablement of their security solution. Approov ensures that traffic destined for the organization's API is indeed coming from the legitimate mobile app and not a third-party tool. This ensures synthetic traffic generated by account takeover (ATO) tools and other API clients, such as Postman or Burp Suite, are blocked.

One of the most common answers I got when asking organizations why they didn't implement certificate pinning is because of their fear that every deployed app will become bricked should a certificate expire. Approov provides easy administration for certificate management when its pinning is enabled, eliminating the concern over bricked apps when problems arise with a certificate.

When Approov was turned on against a target API, I wasn't even able to communicate with the API because the API requests weren't originating from the app compiled with their SDK. It's my opinion that the Approov solution is effective in preventing the types of attacks used in this research.

Code obfuscation can be implemented to prevent the discovery of keys and tokens after decompiling the apps, but it shouldn't be seen as a panacea as even obfuscated code can be deobfuscated. Keys and tokens can be discovered when using a WITM technique if pinning is not enabled as demonstrated in this research.

Understanding Synthetic Traffic

Synthetic traffic is traffic generated by a tool versus traffic generated by human interaction with the approved app. For example, an ATO tool designed to spray an API with stolen credentials, brute forcing the API until a successful login occurs. Human traffic is traffic generated by a human, such as a legitimate patient using the mHealth app to request their clinical reports or x-rays.

A mobile app compiled with Approov arms the app with a dynamic challenge-response integrity measurement protocol, which does not require secrets to be stored in the app -- an advanced DNA test of the app f you will.

The difference between Approov and traditional approaches is that Approov makes an absolute accept/reject decision, while these other tools 'guess' that, statistically-speaking, something isn't right. When guesses are wrong, legitimate users can be falsely rejected. AI can be added to make these tools learn and adapt, hopefully making their guesses a bit better and adaptive to changing conditions, but they are still statistical inferences which will not always be right.

Therefore, synthetic traffic can not only cause security concerns for confidentiality, integrity, and availability of an API, but also can cause significant costs to the organization in CPU and network bandwidth, especially when destined for cloud workloads.

Shift Left Security *and Shield Right*

In the parlance of DevOps and security, shifting security left is the concept of moving security to the earliest point in the development process. This can not only have positive outcomes from an IT risk management perspective, but also a cost perspective. According to the System Sciences Institute at IBM, finding vulnerabilities in the design phase of an application is six times cheaper than finding it in implementation. [10]

In order to implement cybersecurity controls into the agile DevOps continuous integration/continuous delivery (CI/CD) process, automation is a business imperative to ensure developers are able to address user requirements dynamically, release features incrementally, and deliver at a faster pace while maintaining confidentiality, integrity, and availability throughout the software development and deployment lifecyle. (Cloud Security Alliance, 2020). [11] However, as with all things covered in this section, this is just one layer in the multi-layered approach that should be taken to securing your APIs.

Perform Penetration Testing

You must penetration test your own apps and APIs before the app is placed into production. Retain penetration testers skilled in testing mobile applications (on all platforms your app is available on) and the APIs. Static and dynamic code analysis should be performed regularly along with the penetration testing.

Certificate Pinning

Pinning was introduced to address the threat of WITM attacks whereupon an unauthorized individual injects herself in the middle of two talking endpoints in an encrypted session.

Pinning tells the API it can only accept a specific public key from an app. Any other certificates are refused. By doing this, an adversary attempting to perform a WITM attack by presenting a self-signed certificate to both endpoints would fail, refusing to make the connection.

However, if done incorrectly or if you need to change keys, certificates, issuers, or your CA vendor, you must update and push out a new version of your client or all API requests will be blocked.

As a matter of fact, many vendors discourage the use of pinning because the potential to negatively impact availability far outweighs the perceived benefits. (Digicert, 2020) [12]

This is the primary reason a lot of organizations I’ve spoken to, who fell victim to my BOLA attacks, gave as as reason for why they didn’t implement pinning. The threat of bricking the app because of an expired certificate or other issue was greater than the perceived threat to them of a WITM attack occurring against their app.

Approov in Action

During the research, Approov was enabled on one of the hospital's that partnered with me in the research.

I used the same tactics and techniques used in this research to breach the target APIs. 100% of the techniques failed against the API as a result of me communicating from a script and not from a genuine app instance which had been registered with the solution. (Exhibit Q)

The efficacy of Approov's solution was measured based on the ability to breach the target APIs by attempting to send the same exact API requests that allowed unauthorized access to the same APIs without Approov enabled. The success rate of Approov in being able to detect and prevent the unauthorized API requests was 100%.

The tools used to employ these techniques to exploit the identified BOLA vulnerabilities were prevented from being able to successfully reach the API.

To proceed further, we temporarily disabled Approov pinning protection for my device. I then logged in with the WITM proxy turned off and was able to get an initial valid Approov token, which was only good for five minutes. This allowed me to check the API for vulnerabilities, such as BOLA. After that, when turning on Mitmproxy or Burp Suite, my tests failed, due to me receiving bad tokens with a status of either NO_NETWORK or MITM_DETECTED.

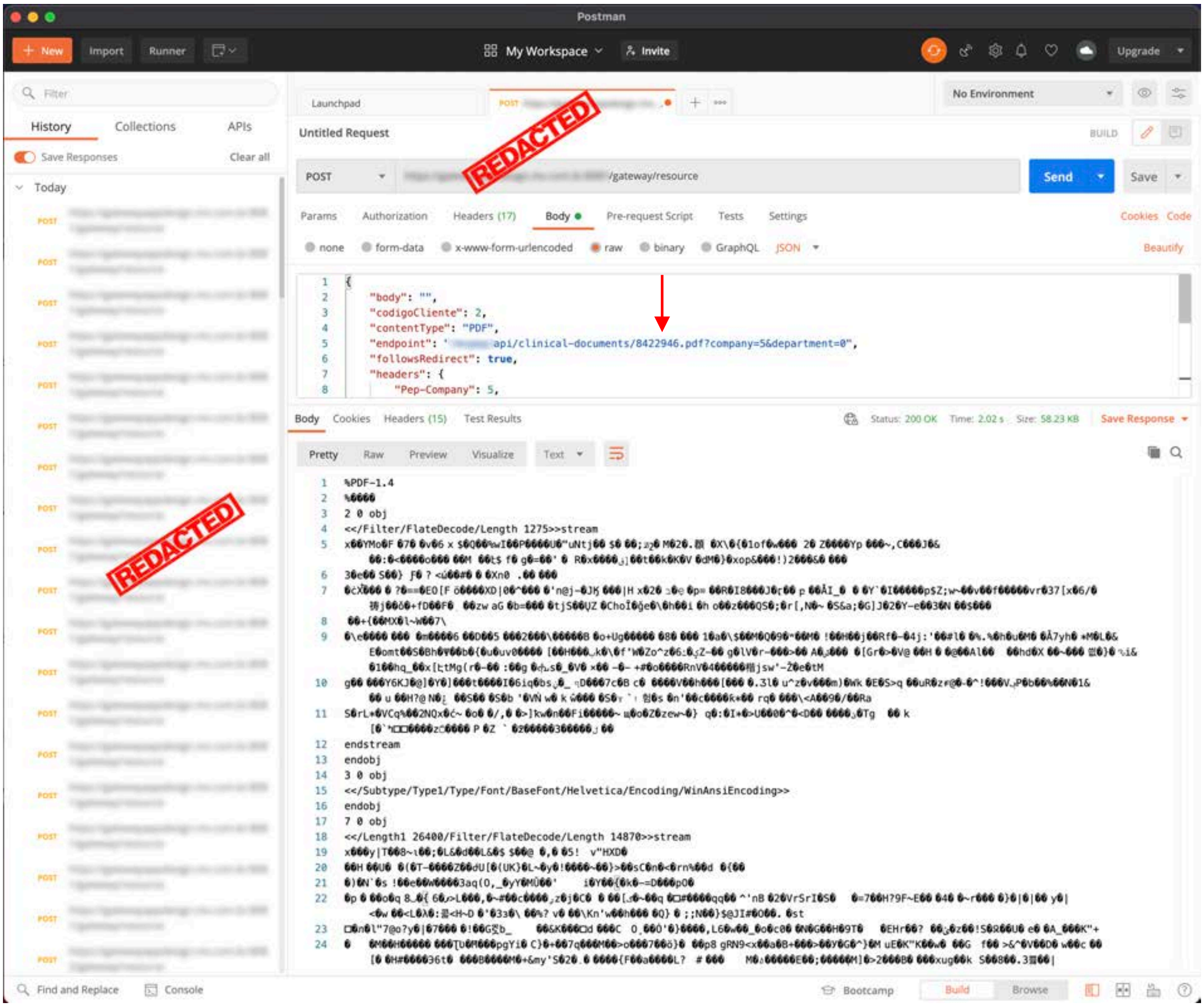
With pinning enabled, even if there was a BOLA weakness, Approov would prevent it from being exploited because a BOLA attack could only be made from an artificial source or a tampered app - neither of which would be 'approoved'.

While some Approov customers implement the error responses differently, the hospital who partnered with me in this research decided not to accept my API calls when they don't get a good status returned from an Approov token fetch.

PEER REVIEWED EVIDENCE

THE PRIMARY RESEARCH EVIDENCE

Exhibit A. Accessing the patient clinical and pathology reports for a patient not assigned to my clinician account through a BOLA vulnerability by changing the filename in the URI.



Postman
New
Import
Runner
My Workspace
Invite
Upgrade
Filter
History
Collections
APIs
Save Responses
Clear all
Today
Launchpad
No Environment
BUILD
Params
Authorization
Headers (17)
Body
Pre-request Script
Tests
Settings
Cookies
Code
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
Body
Cookies
Headers (15)
Test Results
Save Response
Pretty
Raw
Preview
Visualize
Text
Find and Replace
Console
Bootcamp
Build
Browse
REDACTED
REDACTED

Untitled Request

POST [redacted]/gateway/resource

Send

Save

```
{
    "body": "",
    "codigoCliente": 2,
    "contentType": "PDF",
    "endpoint": '[redacted]api/clinical-documents/8422946.pdf?company=5&department=0",
    "followsRedirect": true,
    "headers": {
        "Pep-Company": 5,
```

Status: 200 OK Time: 2.02 s Size: 58.23 KB

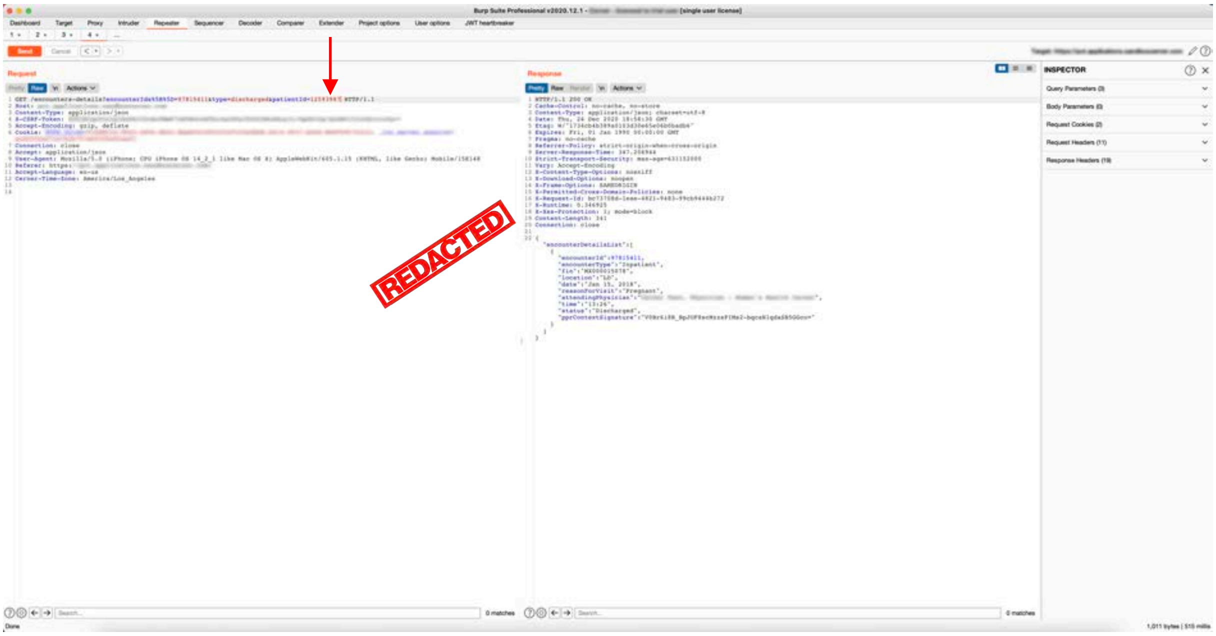
```
%PDF-1.4
2 0 obj
<</Filter/FlateDecode/Length 1275>>stream
endstream
endobj
3 0 obj
<</Subtype/Type1/Type/Font/BaseFont/Helvetica/Encoding/WinAnsiEncoding>>
endobj
7 0 obj
<</Length1 26400/Filter/FlateDecode/Length 14870>>stream
```

Source: Knight Ink

Evidence Explanation

Here, we see a modified API request sent to the API endpoint for a different filename in an attempt to access clinical reports for other patients. Because of the BOLA vulnerability, I was able to specify a different patient's report (8422946.pdf). Postman allows you to save results like this to a file to be viewed by an external application. I saved this result to a PDF and showed the redacted report in Exhibit N. Another mistake made by the developer was that the filenames of the reports were set to auto-increment allowing me to easily find other reports by simply incrementing or decrementing the numbers in the filename.

Exhibit B. Accessing the patient record of a patient not assigned to my clinician account by changing the patientID in the URI.


Burp Suite Professional v2020.12.1 - [redacted] [single user license]
Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Extender
Project options
User options
JWT heartbreaker
Send
Cancel
Request
Response
Pretty
Raw
Actions
INSPECTOR
Query Parameters (3)
Body Parameters (0)
Request Cookies (2)
Request Headers (11)
Response Headers (19)
REDACTED
Done

```
GET /encounters-details?[illegible]&type=discharged&patientId=[illegible] HTTP/1.1
Host: [redacted]
Content-Type: application/json
X-CSRF-Token: [redacted]
Accept-Encoding: gzip, deflate
Cookie: [redacted]
Connection: close
Accept: application/json
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 14_2_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Mobile/15E148
Referer: https:[redacted]
Accept-Language: en-us
Cerner-Time-Zone: America/Los_Angeles
```

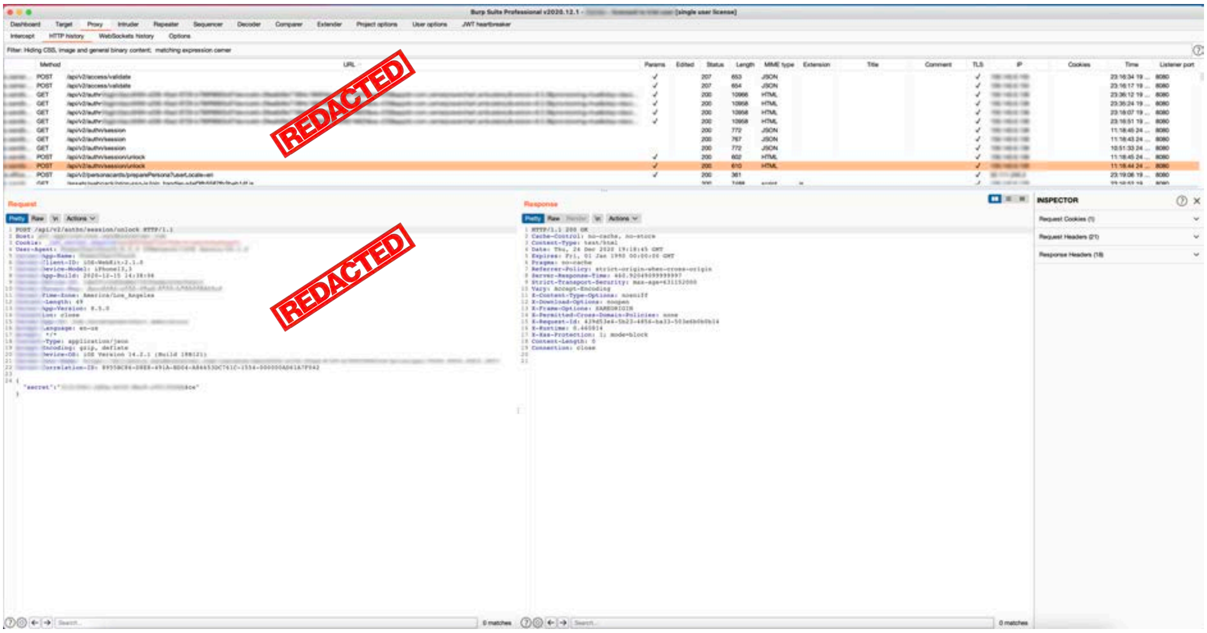
```
HTTP/1.1 200 OK
Cache-Control: no-cache, no-store
Content-Type: application/json; charset=utf-8
Date: Thu, 24 Dec 2020 18:56:30 GMT
Etag: W/"[illegible]"
Expires: Fri, 01 Jan 1990 00:00:00 GMT
Pragma: no-cache
Referrer-Policy: strict-origin-when-cross-origin
Server-Response-Time: [illegible]
Strict-Transport-Security: max-age=631152000
Vary: Accept-Encoding
X-Content-Type-Options: nosniff
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
X-Permitted-Cross-Domain-Policies: none
X-Request-Id: [illegible]
X-Runtime: [illegible]
X-Xss-Protection: 1; mode=block
Content-Length: 341
Connection: close

{
  "encounterDetailsList":[
    {
      "encounterId":97815411,
      "encounterType":"Inpatient",
      "fin":"[illegible]",
      "location":"LD",
      "date":"Jan 15, 2018",
      "reasonForVisit":"Pregnant",
      "attendingPhysician":"[redacted]",
      "time":"13:26",
      "status":"Discharged",
      "pprContextSignature":"[illegible]"
    }
  ]
}
```

Source: Knight Ink

Evidence Explanation

In this screenshot, I'm using Burp Suite to exploit another BOLA vulnerability in the patientID parameter to access other patient records not assigned to my clinician's account. The field (marked with a red arrow) can be modified to any number corresponding to a specific patient. The records from this endpoint contain admissions records for the hospital, providing a simple status of whether they are currently an in-patient or have been discharged and the reason for their visit.

Exhibit C. Replay attack allowing me to unlock a previously locked session days before by replaying the Apple FaceID authentication packet. SECRET parameter was the same across all packets. I was then able to replay all other API requests following that had session locks.

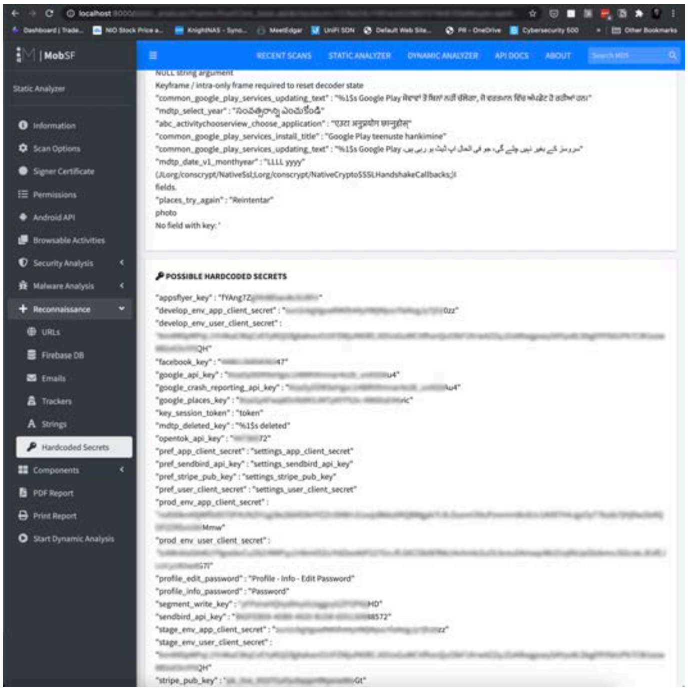


Source: Knight Ink

Evidence Explanation

This evidence provides proof of session takeover. I was able to replay the session unlock request to the API, unlocking a previously locked session. This request is generated by an iPhone when the user uses FaceID to unlock the phone. Upon closer look, I noticed the secret key in the unlock request is the same in every single unlock request.

Exhibit D. Hardcoded keys found by MobSF in mHealth apps after the decompiling process.


localhost:8000/[redacted]
Dashboard | Trade...
NIO Stock Price a...
KnightNAS - Syno...
MeetEdgar
UniFi SDN
Default Web Site...
PR - OneDrive
Cybersecurity 500
Other Bookmarks
MobSF
RECENT SCANS
STATIC ANALYZER
DYNAMIC ANALYZER
API DOCS
ABOUT
Search MD5
Static Analyzer
Information
Scan Options
Signer Certificate
Permissions
Android API
Browsable Activities
Security Analysis
Malware Analysis
Reconnaissance
URLs
Firebase DB
Emails
Trackers
Strings
Hardcoded Secrets
Components
PDF Report
Print Report
Start Dynamic Analysis

```
NULL string argument
Keyframe / intra-only frame required to reset decoder state
"common_google_play_services_updating_text" : "%1$s Google Play ਸੇਵਾਵਾਂ ਤੋਂ ਬਿਨਾਂ ਨਹੀਂ ਚੱਲੇਗਾ, ਜੋ ਵਰਤਮਾਨ ਵਿੱਚ ਅੱਪਡੇਟ ਹੋ ਰਹੀਆਂ ਹਨ।"
"mdtp_select_year" : "సంవత్సరాన్ని ఎంచుకోండి"
"abc_activitychooserview_choose_application" : "एउटा अनुप्रयोग छान्नुहोस्"
"common_google_play_services_install_title" : "Google Play teenuste hankimine"
"common_google_play_services_updating_text" : "%1$s Google Play سروسز کے بغیر نہیں چلے گی، جو فی الحال اپ ڈیٹ ہو رہی ہیں۔"
"mdtp_date_v1_monthyear" : "LLLL yyyy"
(JLorg/conscrypt/NativeSsl;Lorg/conscrypt/NativeCrypto$SSLHandshakeCallbacks;)I
fields.
"places_try_again" : "Reintentar"
photo
No field with key: '
```

POSSIBLE HARDCODED SECRETS

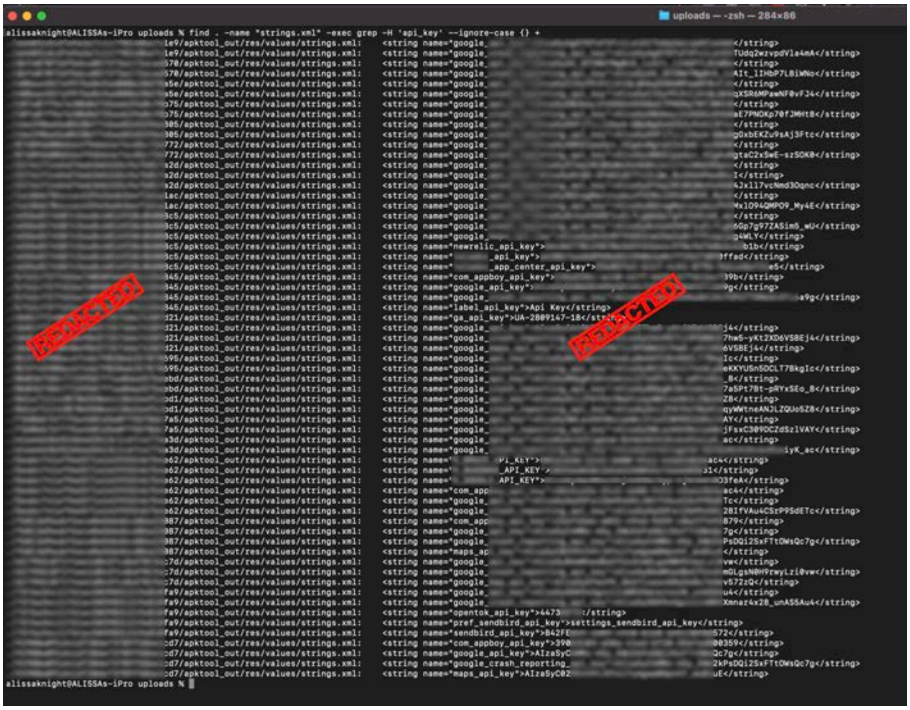
```
"appsflyer_key" : "fYAng7Z[redacted]"
"develop_env_app_client_secret" : "[redacted]0zz"
"develop_env_user_client_secret" :
"[redacted]QH"
"facebook_key" : "[redacted]47"
"google_api_key" : "[redacted]u4"
"google_crash_reporting_api_key" : "[redacted]Au4"
"google_places_key" : "[redacted]ric"
"key_session_token" : "token"
"mdtp_deleted_key" : "%1$s deleted"
"opentok_api_key" : "[redacted]72"
"pref_app_client_secret" : "settings_app_client_secret"
"pref_sendbird_api_key" : "settings_sendbird_api_key"
"pref_stripe_pub_key" : "settings_stripe_pub_key"
"pref_user_client_secret" : "settings_user_client_secret"
"prod_env_app_client_secret" :
"[redacted]Mmw"
"prod_env_user_client_secret" :
"[redacted]G7I"
"profile_edit_password" : "Profile - Info - Edit Password"
"profile_info_password" : "Password"
"segment_write_key" : "[redacted]HD"
"sendbird_api_key" : "[redacted]8572"
"stage_env_app_client_secret" : "[redacted]zz"
"stage_env_user_client_secret" :
"[redacted]QH"
"stripe_pub_key" : "[redacted]Gt"
```

Source: Knight Ink

Evidence Explanation

This is evidence of hardcoded keys found in one of the apps using MobSF. This shows what little sophistication is needed and how quickly an adversary can find hardcoded API keys and tokens in an app, even with a pretty graphical user interface (GUI).

Exhibit E. Hardcoded keys found in all mHealth apps after being decompiled in MobSF at the command line using 'grep' for keyword 'api_key'.



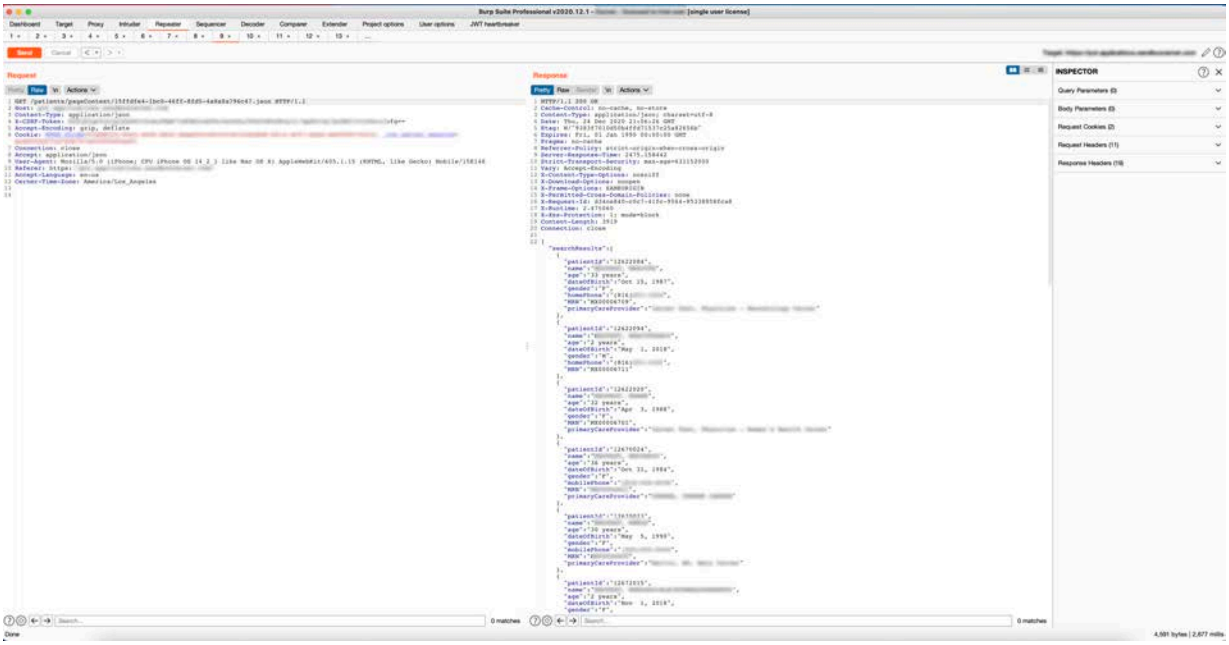
```
alissaknight@ALISSAs-iPro uploads % find . -name "strings.xml" -exec grep -H 'api_key' --ignore-case {} +
[redacted]
alissaknight@ALISSAs-iPro uploads %
```

Source: Knight Ink

Evidence Explanation

In this screenshot, I'm using different queries with grep to find keys and tokens across all of the apps. When an app is imported into MobSF, it places the decompiled app into its own sub-folder in the <MobSF_root>/uploads folder. This makes searching for specific keywords across all apps trivial as an alternative to using the GUI. Also, MobSF makes a best attempt at finding hardcoded keys. By using grep combined with regular expressions (REGEX), I was able to find other keys and tokens as well as credentials that MobSF missed.

Exhibit F. BOLA vulnerability found allowing me to see all patients admitted into the hospital.

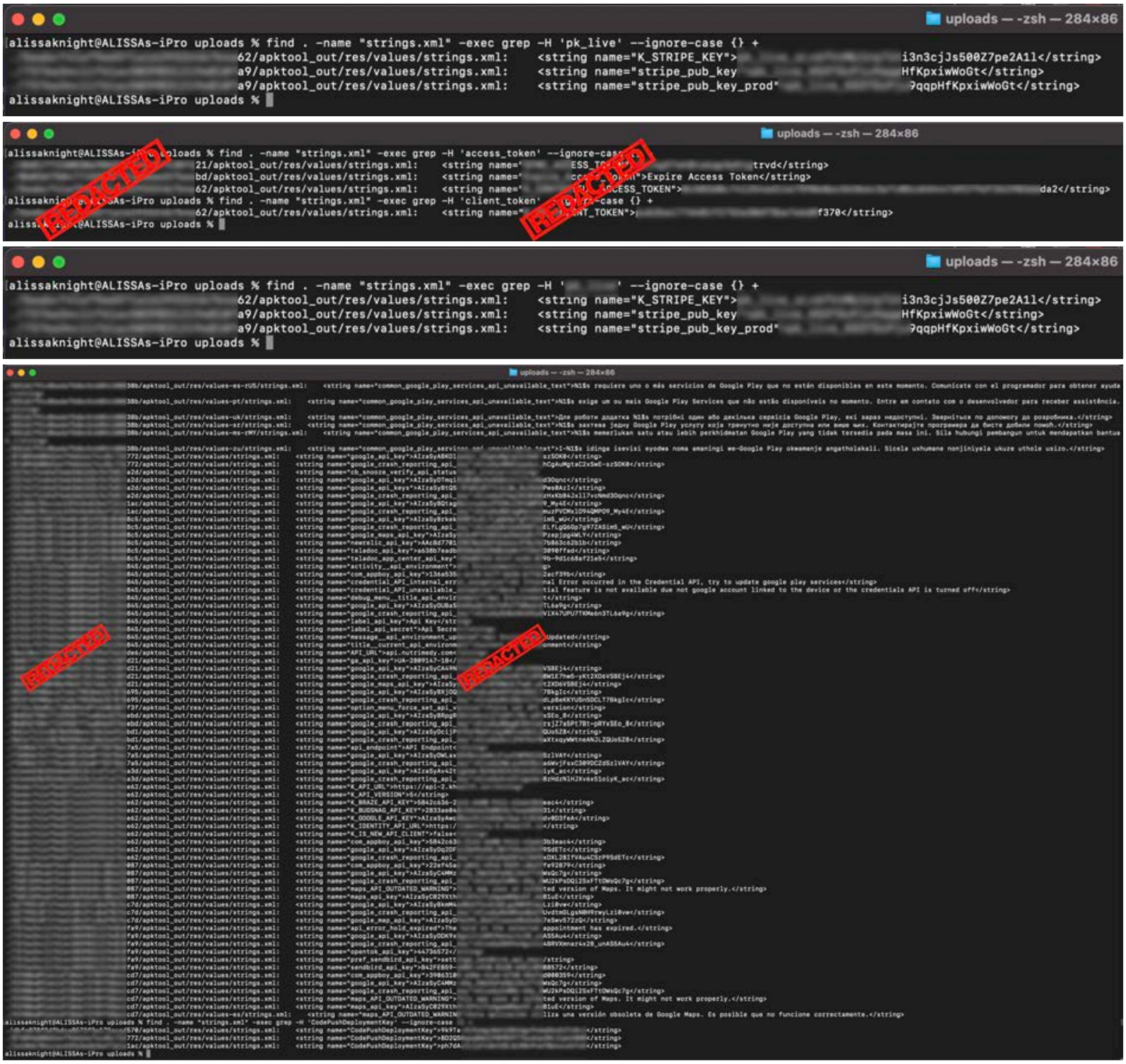

Burp Suite Professional v2020.12.1 - [redacted] [single user license]
Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Extender
Project options
User options
JWT heartbreaker
Send
Cancel
Request
Pretty
Raw
\n
Actions
Response
Render
INSPECTOR
Query Parameters (0)
Body Parameters (0)
Request Cookies (2)
Request Headers (11)
Response Headers (19)
0 matches
Search...
Done
4,501 bytes | 2,677 millis

Source: *Knight Ink*

Evidence Explanation

Here I've modified the API request and because of a BOLA vulnerability was able to pull up a list of all patients matching a specific query from the admissions records of the hospital.

Exhibit G. Hundreds of keys and tokens discovered using multiple 'grep' queries against all apps after being decompiled by MobSF.



```
alissaknight@ALISSAs-iPro uploads % find . -name "strings.xml" -exec grep -H 'pk_live' --ignore-case {} +
[redacted]62/apktool_out/res/values/strings.xml:    <string name="K_STRIPE_KEY">[redacted]i3n3cjJs500Z7pe2A1l</string>
[redacted]a9/apktool_out/res/values/strings.xml:    <string name="stripe_pub_key[redacted]HfKpxiwWoGt</string>
[redacted]a9/apktool_out/res/values/strings.xml:    <string name="stripe_pub_key_prod"[redacted]9qqpHfKpxiwWoGt</string>
alissaknight@ALISSAs-iPro uploads %
```

```
alissaknight@ALISSAs-[redacted]uploads % find . -name "strings.xml" -exec grep -H 'access_token' --ignore-cas[redacted]
[redacted]21/apktool_out/res/values/strings.xml:    <string name="[redacted]ESS_T[redacted]trvd</string>
[redacted]bd/apktool_out/res/values/strings.xml:    <string name="[redacted]">Expire Access Token</string>
[redacted]62/apktool_out/res/values/strings.xml:    <string name="[redacted]CCESS_TOKEN">[redacted]da2</string>
alissakni[redacted]As-iPro uploads % find . -name "strings.xml" -exec grep -H 'client_token'[redacted]-case {} +
[redacted]62/apktool_out/res/values/strings.xml:    <string name="[redacted]NT_TOKEN">[redacted]f370</string>
aliss[redacted]@ALISSAs-iPro uploads %
```

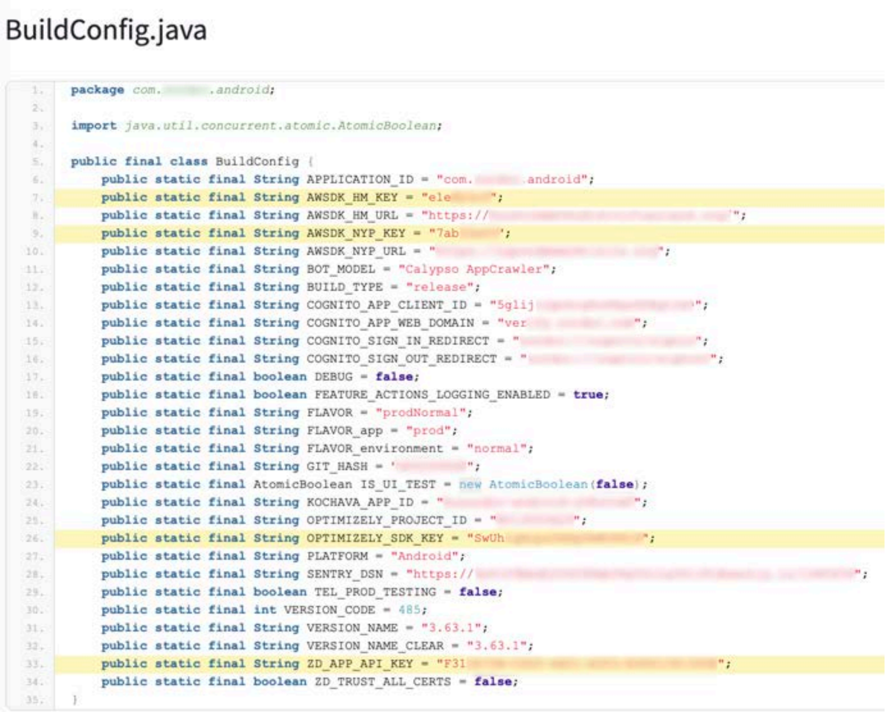
```
alissaknight@ALISSAs-iPro uploads % find . -name "strings.xml" -exec grep -H '[redacted]' --ignore-case {} +
[redacted]62/apktool_out/res/values/strings.xml:    <string name="K_STRIPE_KEY">[redacted]i3n3cjJs500Z7pe2A1l</string>
[redacted]a9/apktool_out/res/values/strings.xml:    <string name="stripe_pub_key[redacted]HfKpxiwWoGt</string>
[redacted]a9/apktool_out/res/values/strings.xml:    <string name="stripe_pub_key_prod"[redacted]9qqpHfKpxiwWoGt</string>
alissaknight@ALISSAs-iPro uploads %
```

Source: Knight Ink

Evidence Explanation

Here is more evidence of hardcoded keys and tokens found in the decompiled apps folder of MobSF. As you can see, keys for not only the mHealth company were found, but also keys for third-parties, such as payment processors and cloud service providers.

Exhibit H. More keys discovered hardcoded within specific files missed by MobSF.



BuildConfig.java

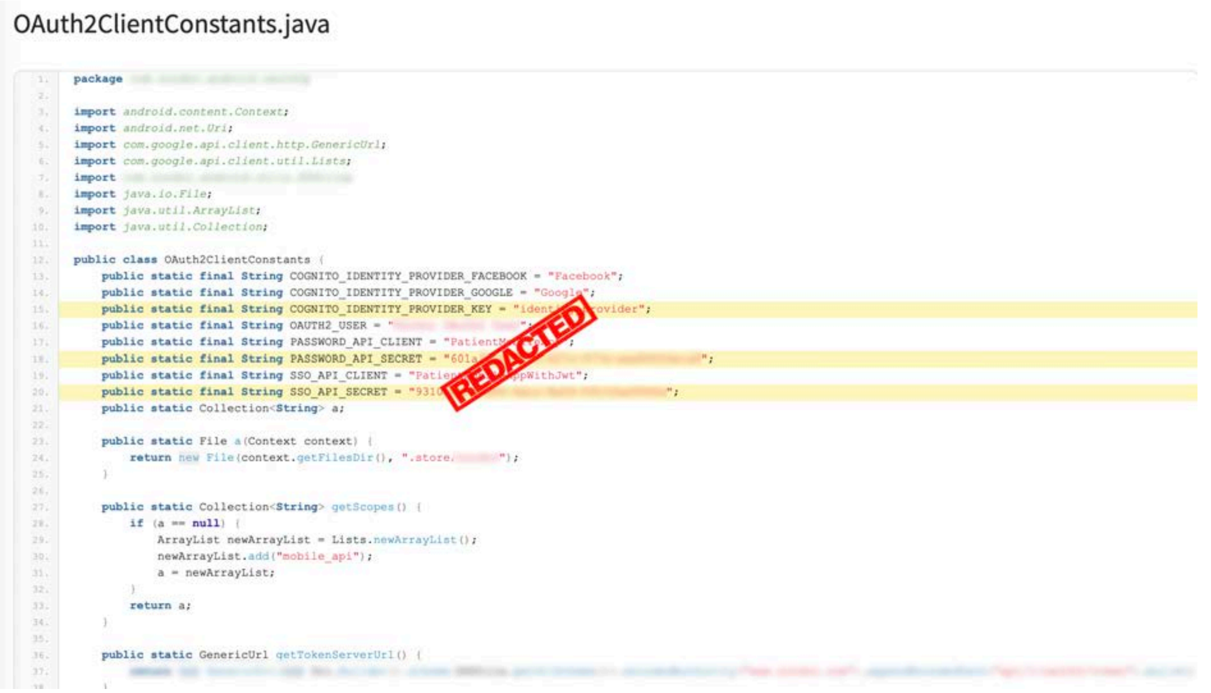
```
package com.[redacted].android;

import java.util.concurrent.atomic.AtomicBoolean;

public final class BuildConfig {
    public static final String APPLICATION_ID = "com.[redacted].android";
    public static final String AWSDK_HM_KEY = "ele[redacted]";
    public static final String AWSDK_HM_URL = "https://[redacted]";
    public static final String AWSDK_NYP_KEY = "7ab[redacted]';
    public static final String AWSDK_NYP_URL = "[redacted]";
    public static final String BOT_MODEL = "Calypso AppCrawler";
    public static final String BUILD_TYPE = "release";
    public static final String COGNITO_APP_CLIENT_ID = "5glij[redacted]";
    public static final String COGNITO_APP_WEB_DOMAIN = "ver[redacted]";
    public static final String COGNITO_SIGN_IN_REDIRECT = "[redacted]";
    public static final String COGNITO_SIGN_OUT_REDIRECT = "[redacted]";
    public static final boolean DEBUG = false;
    public static final boolean FEATURE_ACTIONS_LOGGING_ENABLED = true;
    public static final String FLAVOR = "prodNormal";
    public static final String FLAVOR_app = "prod";
    public static final String FLAVOR_environment = "normal";
    public static final String GIT_HASH = '[redacted]";
    public static final AtomicBoolean IS_UI_TEST = new AtomicBoolean(false);
    public static final String KOCHAVA_APP_ID = "[redacted]";
    public static final String OPTIMIZELY_PROJECT_ID = "[redacted]";
    public static final String OPTIMIZELY_SDK_KEY = "SwUh[redacted]";
    public static final String PLATFORM = "Android";
    public static final String SENTRY_DSN = "https://[redacted]";
    public static final boolean TEL_PROD_TESTING = false;
    public static final int VERSION_CODE = 485;
    public static final String VERSION_NAME = "3.63.1";
    public static final String VERSION_NAME_CLEAR = "3.63.1";
    public static final String ZD_APP_API_KEY = "F31[redacted]";
    public static final boolean ZD_TRUST_ALL_CERTS = false;
}
```

Source: Knight Ink

Evidence Explanation

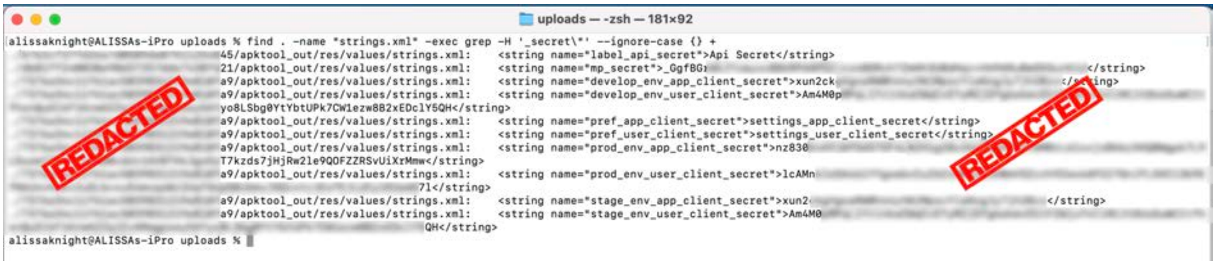
MobSF provides a separate screen for what it believes may be hardcoded keys and passwords found in files that don't show up in the hardcoded keys section of the page. This is a screenshot of one of the files found in one of the apps BuildConfig.java. This file contained numerous keys for third-party sites.

Exhibit I. Attempting different 'grep' queries to find more hardcoded keys and tokens missed by MobSF



OAuth2ClientConstants.java

```
package [redacted]

import android.content.Context;
import android.net.Uri;
import com.google.api.client.http.GenericUrl;
import com.google.api.client.util.Lists;
import [redacted]
import java.io.File;
import java.util.ArrayList;
import java.util.Collection;

public class OAuth2ClientConstants {
    public static final String COGNITO_IDENTITY_PROVIDER_FACEBOOK = "Facebook";
    public static final String COGNITO_IDENTITY_PROVIDER_GOOGLE = "Google";
    public static final String COGNITO_IDENTITY_PROVIDER_KEY = "ident[redacted]rovider";
    public static final String OAUTH2_USER = "[redacted]";
    public static final String PASSWORD_API_CLIENT = "PatientM[redacted]";
    public static final String PASSWORD_API_SECRET = "601a[redacted]";
    public static final String SSO_API_CLIENT = "Patie[redacted]ppWithJwt";
    public static final String SSO_API_SECRET = "9310[redacted]";
    public static Collection<String> a;

    public static File a(Context context) {
        return new File(context.getFilesDir(), ".store.[redacted]");
    }

    public static Collection<String> getScopes() {
        if (a == null) {
            ArrayList newArrayList = Lists.newArrayList();
            newArrayList.add("mobile_api");
            a = newArrayList;
        }
        return a;
    }

    public static GenericUrl getTokenServerUrl() {
        [redacted]
    }
```



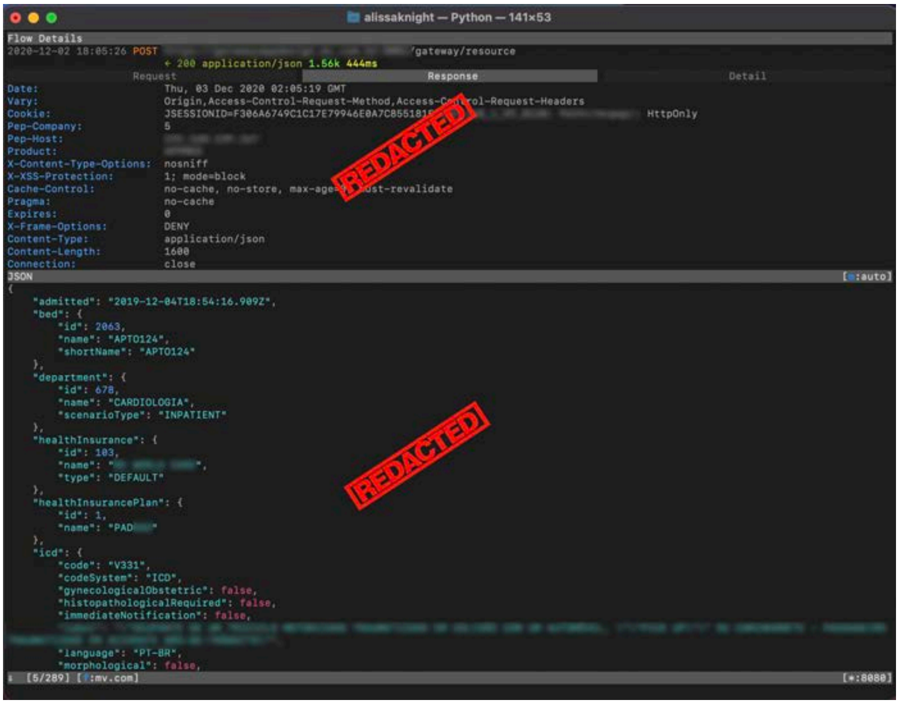
```
alissaknight@ALISSAs-iPro uploads % find . -name "strings.xml" -exec grep -H '_secret\"' --ignore-case {} +
[redacted]45/apktool_out/res/values/strings.xml:    <string name="label_api_secret">Api Secret</string>
[redacted]21/apktool_out/res/values/strings.xml:    <string name="mp_secret">_GgfBG[redacted]</string>
[redacted]a9/apktool_out/res/values/strings.xml:    <string name="develop_env_app_client_secret">xun2ck[redacted]</string>
[redacted]a9/apktool_out/res/values/strings.xml:    <string name="develop_env_user_client_secret">Am4M0p[redacted]
[redacted]yo8LSbg0YtYbtUPk7CW1ezw8B2xEDclY5QH</string>
[redacted]a9/apktool_out/res/values/strings.xml:    <string name="pref_app_client_secret">settings_app_client_secret</string>
[redacted]a9/apktool_out/res/values/strings.xml:    <string name="pref_user_client_secret">settings_user_client_secret</string>
[redacted]a9/apktool_out/res/values/strings.xml:    <string name="prod_env_app_client_secret">nz830[redacted]
[redacted]T7kzds7jHjRw2le9QOFZZRSvUiXrMmw</string>
[redacted]a9/apktool_out/res/values/strings.xml:    <string name="prod_env_user_client_secret">1cAMn[redacted]
[redacted]71</string>
[redacted]a9/apktool_out/res/values/strings.xml:    <string name="stage_env_app_client_secret">xun2[redacted]</string>
[redacted]a9/apktool_out/res/values/strings.xml:    <string name="stage_env_user_client_secret">Am4M0[redacted]
[redacted]QH</string>
alissaknight@ALISSAs-iPro uploads % ▌
```

Source: Knight Ink

Evidence Explanation

Here I'm attempting more grep queries with different keywords, this time with _secret. I've also provided a screenshot of another file OAuth2ClientConstants.java from the MobSF GUI of more API secrets it discovered in the file.

Exhibit J. Mitmproxy possible due to absence of certificate pinning with the API.



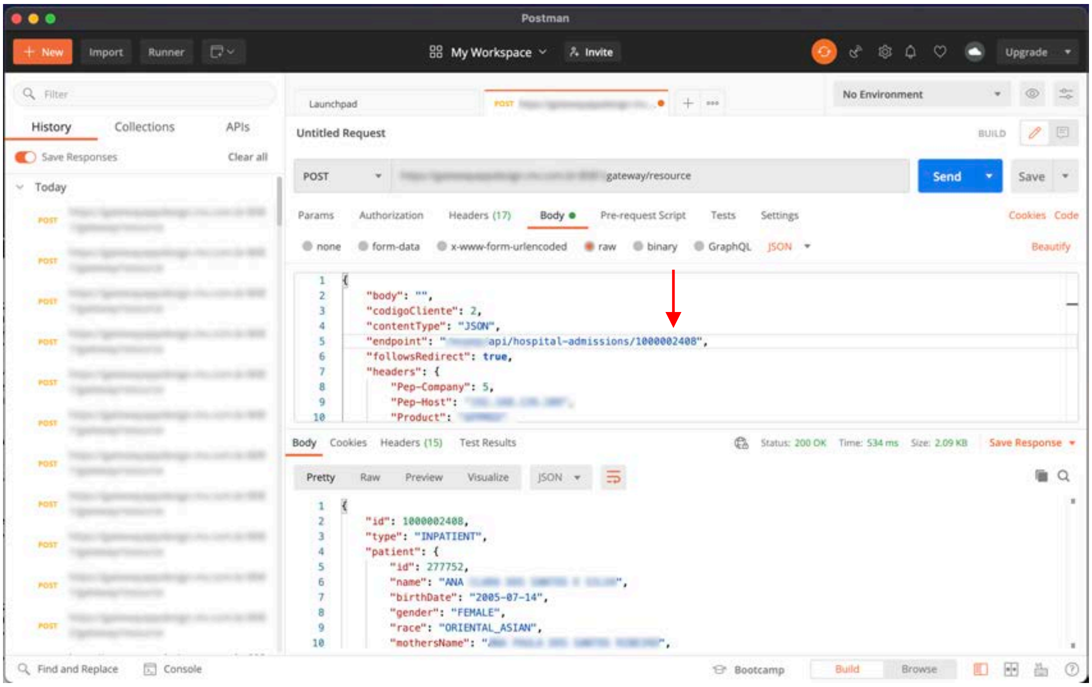
```
Flow Details
2020-12-02 18:05:26 POST [redacted]/gateway/resource
                         ← 200 application/json 1.56k 444ms
                    Request                    Response                    Detail
Date:                     Thu, 03 Dec 2020 02:05:19 GMT
Vary:                     Origin,Access-Control-Request-Method,Access-Co[illegible]rol-Request-Headers
Cookie:                   JSESSIONID=F306A6749C1C17E79946E0A7C855181[redacted] HttpOnly
Pep-Company:              5
Pep-Host:                 [redacted]
Product:                  [redacted]
X-Content-Type-Options:   nosniff
X-XSS-Protection:         1; mode=block
Cache-Control:            no-cache, no-store, max-age=[illegible]ust-revalidate
Pragma:                   no-cache
Expires:                  0
X-Frame-Options:          DENY
Content-Type:             application/json
Content-Length:           1600
Connection:               close
JSON                                                                         [ :auto]
{
    "admitted": "2019-12-04T18:54:16.909Z",
    "bed": {
        "id": 2063,
        "name": "APTO124",
        "shortName": "APTO124"
    },
    "department": {
        "id": 678,
        "name": "CARDIOLOGIA",
        "scenarioType": "INPATIENT"
    },
    "healthInsurance": {
        "id": 103,
        "name": "[redacted]",
        "type": "DEFAULT"
    },
    "healthInsurancePlan": {
        "id": 1,
        "name": "PAD[redacted]"
    },
    "icd": {
        "code": "V331",
        "codeSystem": "ICD",
        "gynecologicalObstetric": false,
        "histopathologicalRequired": false,
        "immediateNotification": false,
[redacted]
        "language": "PT-BR",
        "morphological": false,
  [5/289] [ :mv.com]                                                         [*:8080]
```

Source: *Knight Ink*

Evidence Explanation

In this screenshot, you see the Mitmproxy interface. Because of the absence of pinning on this API, I was able to perform a WITM attack allowing me to decrypt the SSL encrypted traffic from the app and better understand what queries the API supports. This type of reconnaissance is a common step I take as “intelligence collection” when I perform an API penetration test.

Exhibit K. BOLA vulnerability found allowing me to see any patient record not assigned to my clinician account.


Postman
New
Import
Runner
My Workspace
Invite
Upgrade
Filter
History
Collections
APIs
Save Responses
Clear all
Today
Launchpad
No Environment
Untitled Request
BUILD
Send
Save
Params
Authorization
Headers (17)
Body
Pre-request Script
Tests
Settings
Cookies
Code
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
Body
Cookies
Headers (15)
Test Results
Save Response
Pretty
Raw
Preview
Visualize
JSON
Find and Replace
Console
Bootcamp
Build
Browse

POST [redacted]gateway/resource

```
{
    "body": "",
    "codigoCliente": 2,
    "contentType": "JSON",
    "endpoint": "[redacted]api/hospital-admissions/1000002408",
    "followsRedirect": true,
    "headers": {
        "Pep-Company": 5,
        "Pep-Host": [redacted]
        "Product": [redacted]
```

Status: 200 OK Time: 534 ms Size: 2.09 KB

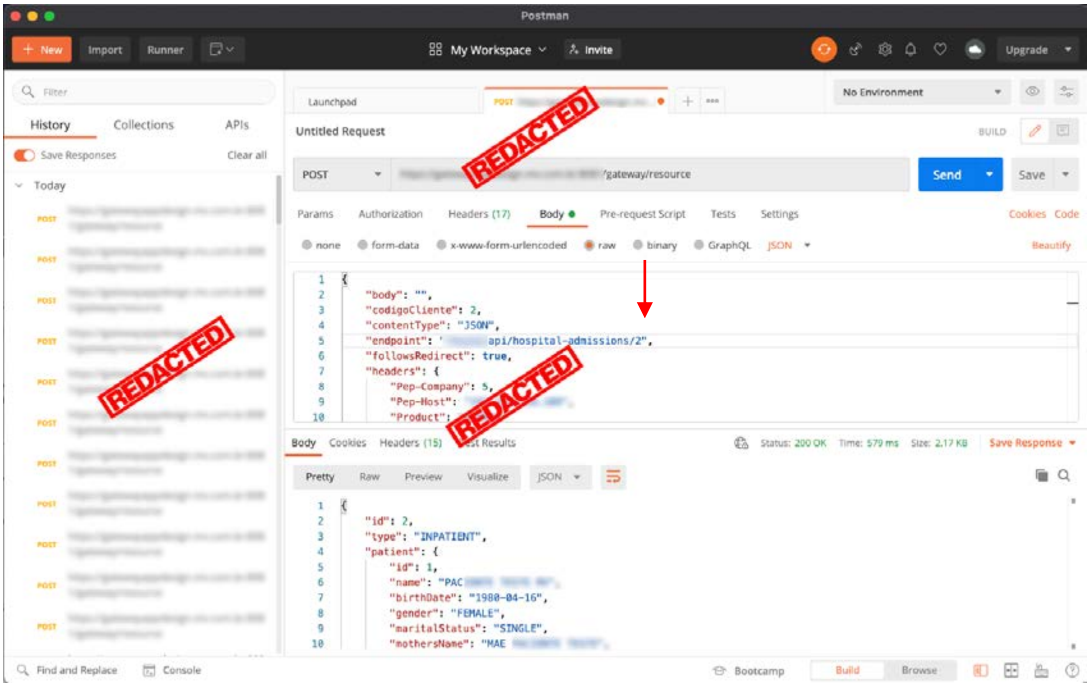
```
{
    "id": 1000002408,
    "type": "INPATIENT",
    "patient": {
        "id": 277752,
        "name": "ANA [redacted]",
        "birthDate": "2005-07-14",
        "gender": "FEMALE",
        "race": "ORIENTAL_ASIAN",
        "mothersName": "[redacted]",
```

Source: Knight Ink

Evidence Explanation

This is a screenshot of Postman where I'm able to create API requests from scratch, using parameters grabbed from the WITM technique used with Mitmproxy. This screenshot shows patient data for a patient who was admitted into the hospital and is currently staying, not yet discharged. Notice the information also contains PII for family members. What can't be seen here is if scrolling down, the information for the mother and other family members was also stored in the server.

Exhibit L. BOLA vulnerability found allowing me to see any patient record not assigned to my clinician account admitted by hospital admissions


Postman
New
Import
Runner
My Workspace
Invite
Upgrade
Filter
History
Collections
APIs
Save Responses
Clear all
Today
Launchpad
No Environment
Untitled Request
BUILD
REDACTED
REDACTED
REDACTED
Send
Save
Params
Authorization
Headers (17)
Body
Pre-request Script
Tests
Settings
Cookies
Code
Beautify
Cookies
Headers (15)
Save Response
Pretty
Raw
Preview
Visualize
Find and Replace
Console
Bootcamp
Build
Browse

POST [redacted]/gateway/resource

○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

```
{
    "body": "",
    "codigoCliente": 2,
    "contentType": "JSON",
    "endpoint": '[redacted]api/hospital-admissions/2",
    "followsRedirect": true,
    "headers": {
        "Pep-Company": 5,
        "Pep-Host": [redacted]
        "Product":
```

Status: 200 OK Time: 579 ms Size: 2.17 KB

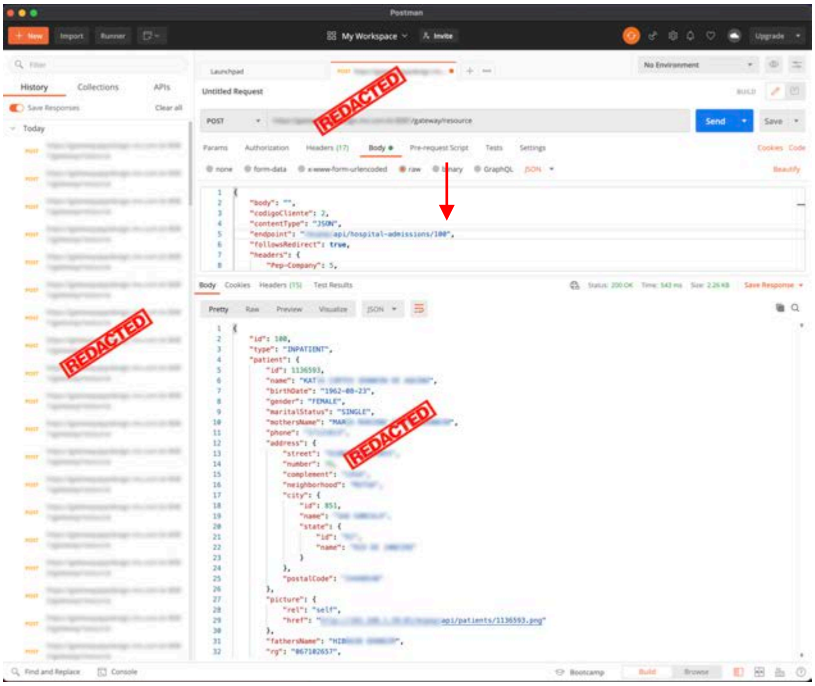
```
{
    "id": 2,
    "type": "INPATIENT",
    "patient": {
        "id": 1,
        "name": "PAC [redacted]
        "birthDate": "1980-04-16",
        "gender": "FEMALE",
        "maritalStatus": "SINGLE",
        "mothersName": "MAE [redacted]
```

Source: Knight Ink

Evidence Explanation

In this screenshot, I'm exploiting a BOLA vulnerability to pull up other patient records by simply modifying the patientID field. Notice the /2 on the endpoint query. Changing this number to any other number pulls up the corresponding patient in the database.

Exhibit M. BOLA vulnerability found allowing me to see any patient record not assigned to my clinician account admitted by hospital admissions.


Postman
New
Import
Runner
My Workspace
Invite
Upgrade
History
Collections
APIs
Save Responses
Clear all
Today
Launchpad
No Environment
Untitled Request
REDACTED
Send
Save
Params
Authorization
Headers (17)
Body
Pre-request Script
Tests
Settings
Cookies
Code
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
Body
Cookies
Headers (15)
Test Results
Status: 200 OK
Time: 543 ms
Size: 2.26 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
REDACTED
REDACTED
Find and Replace
Console
Bootcamp
Build
Browse

POST [redacted]/gateway/resource

```
{
    "body": "",
    "codigoCliente": 2,
    "contentType": "JSON",
    "endpoint": "[redacted]api/hospital-admissions/100",
    "followsRedirect": true,
    "headers": {
        "Pep-Company": 5,
```

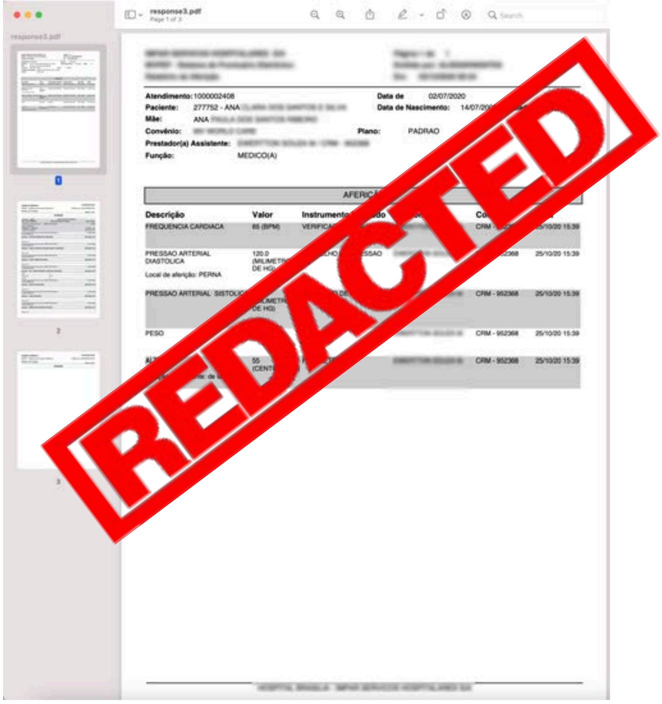
```
{
    "id": 100,
    "type": "INPATIENT",
    "patient": {
        "id": 1136593,
        "name": "KAT[redacted]",
        "birthDate": "1962-08-23",
        "gender": "FEMALE",
        "maritalStatus": "SINGLE",
        "mothersName": "MAR[redacted]",
        "phone": [redacted],
        "address": {
            "street": [redacted],
            "number": [redacted],
            "complement": [redacted],
            "neighborhood": [redacted],
            "city": {
                "id": 851,
                "name": [redacted],
                "state": {
                    "id": [redacted],
                    "name": [redacted]
                }
            },
            "postalCode": [redacted]
        },
        "picture": {
            "rel": "self",
            "href": "[redacted]api/patients/1136593.png"
        },
        "fathersName": "HIB[redacted]",
        "rg": "067102657",
```

Source: Knight Ink

Evidence Explanation

Another screenshot where I've modified the endpoint query to reflect patient 100 instead to pull up the PII for another patient admitted into the hospital.

Exhibit N. BOLA vulnerability allowing access to any PDF report for any patient not assigned to my clinician account.



response3.pdf
Page 1 of 3

[redacted]

Atendimento: 1000002408 Data de 02/07/2020
Paciente: 277752 - ANA [redacted] Data de Nascimento: 14/07/20
Mãe: ANA [redacted]
Convênio: [redacted] Plano: PADRAO
Prestador(a) Assistente: [redacted]
Função: MEDICO(A)

AFERIÇ

Descrição	Valor	Instrumento do		Co	
FREQUENCIA CARDIACA	85 (BPM)	VERIFICA	[redacted]	CRM -	25/10/20 15:39
PRESSAO ARTERIAL DIASTOLICA Local de aferição: PERNA	120.0 (MILIMETR DE HG)	LHO SSAO	[redacted]	52368	25/10/20 15:39
PRESSAO ARTERIAL SISTOLIC	LIMETR DE HG)	O DE	[redacted]	CRM - 952368	25/10/20 15:39
PESO			[redacted]	CRM - 952368	25/10/20 15:39
AL	55 (CENT	METR	[redacted]	CRM - 952368	25/10/20 15:39

[redacted]

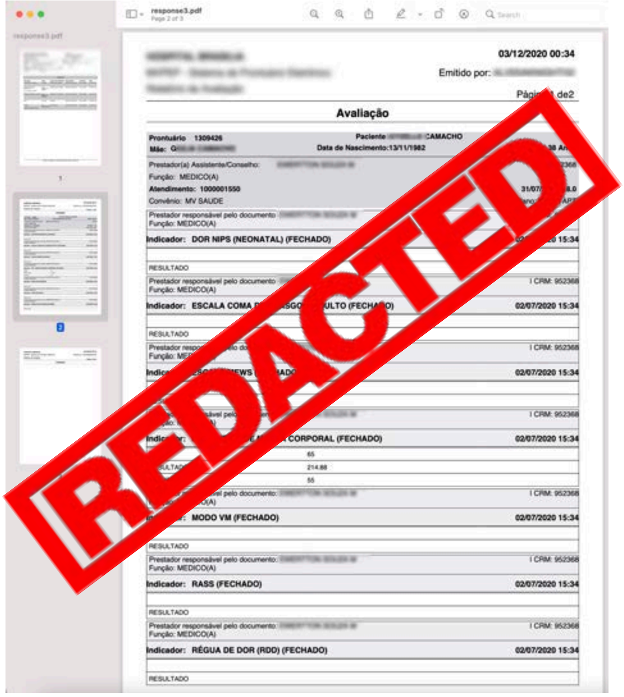
REDACTED

Source: *Knight Ink*

Evidence Explanation

This PDF was downloaded using the SAVE FILE feature in Postman. Once I had exploited a BOLA Vulnerability in the filename field, I was able to save any file sent back to me of any filename I specified, save it to a PDF, then view it within Preview on my Mac.

Exhibit O. BOLA vulnerability allowing access to any PDF report for any patient not assigned to my clinician account.



[redacted]

03/12/2020 00:34

Emitido por: [redacted]

Págin de2

Avaliação

Prontuário 1309426

Mãe: G [redacted]

Paciente [redacted] CAMACHO

Data de Nascimento:13/11/1982

Prestador(a) Assistente/Conselho: [redacted]

Função: MEDICO(A)

Atendimento: 1000001550

Convênio: MV SAUDE

Prestador responsável pelo documento [redacted]
Função: MEDICO(A)

Indicador: DOR NIPS (NEONATAL) (FECHADO)

RESULTADO

Prestador responsável pelo documento | CRM: 952368
Função: MEDICO(A)

Indicador: ESCALA COMA (FECH O) 02/07/2020 15:34

RESULTADO

| CRM: 952368

02/07/2020 15:34

| CRM: 952368

CORPORAL (FECHADO) 02/07/2020 15:34

65
214.88
55

| CRM: 952368

MODO VM (FECHADO) 02/07/2020 15:34

RESULTADO

Prestador responsável pelo documento: [redacted] | CRM: 952368
Função: MEDICO(A)

Indicador: RASS (FECHADO) 02/07/2020 15:34

RESULTADO

Prestador responsável pelo documento: [redacted] | CRM: 952368
Função: MEDICO(A)

Indicador: RÉGUA DE DOR (RDD) (FECHADO) 02/07/2020 15:34

RESULTADO

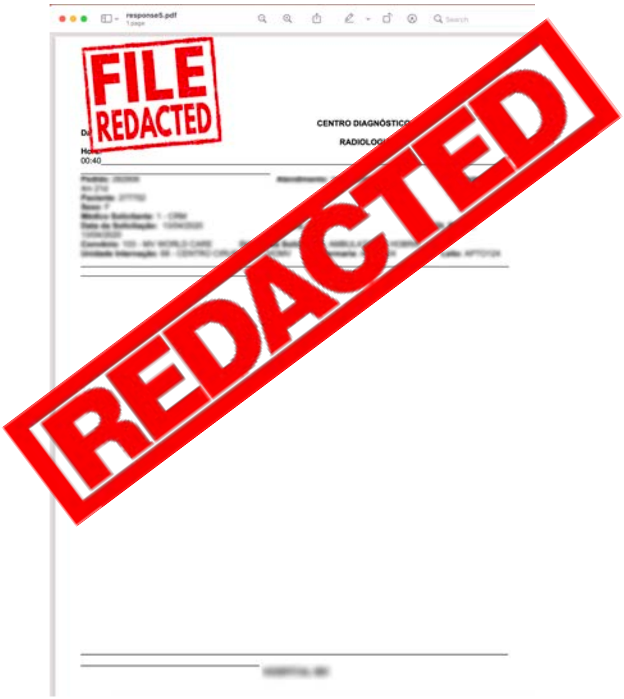
REDACTED

Source: Knight Ink

Evidence Explanation

This PDF was downloaded using the SAVE FILE feature in Postman. Once I had exploited a BOLA Vulnerability in the filename field, I was able to save any file sent back to me of any filename I specified, save it to a PDF, then view it within Preview on my Mac.

Exhibit P. BOLA vulnerability allowing access to any PDF report for any patient not assigned to my clinician account.

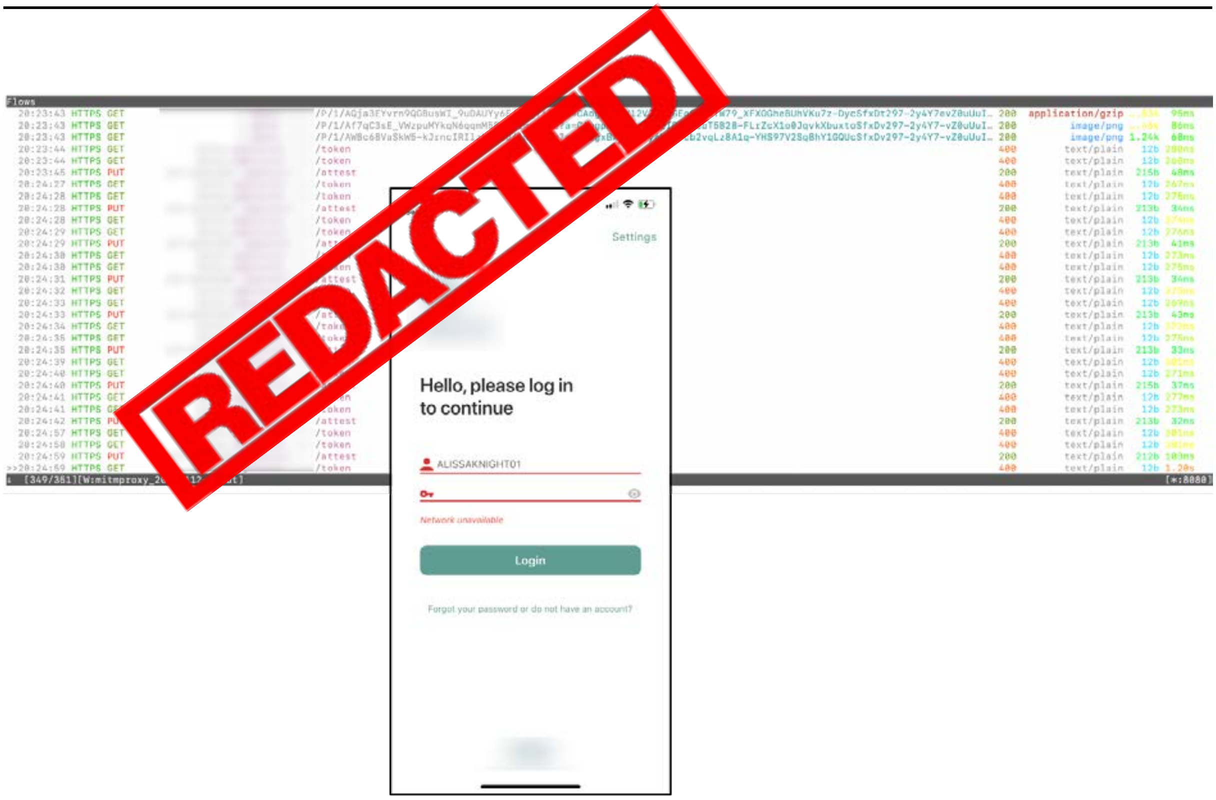

response5.pdf
FILE REDACTED
CENTRO DIAGNÓSTICO
REDACTED

Source: Knight Ink

Evidence Explanation

This PDF was downloaded using the SAVE FILE feature in Postman. Once I had exploited a BOLA Vulnerability in the filename field, I was able to save any file sent back to me of any filename I specified, save it to a PDF, then view it within Preview on my Mac.

Exhibit Q. Evidence of being unable to communicate with the API server after Approov was enabled on the target API.



REDACTED

[redacted]

Flows
Settings

Hello, please log in to continue

ALISSAKNIGHT01

Network unavailable

Login
Forgot your password or do not have an account?

Source: Knight Ink

Evidence Explanation

In this screenshot, you can see repeated 400 errors as my client made attempts to fetch a new valid Approov token (Error 400).

CONCLUSION

AUTHOR’S FINAL THOUGHTS

CONCLUSION

In 2019, I hacked a large European bank who partnered with me in the research and posted a video of it on YouTube. In that attack, I used a BOLA vulnerability that allowed me to specify any bank customer's ATM debit card number and account number to perform money transfers and change pin codes.

BOLA vulnerabilities are the most prevalent vulnerability I've found in APIs. In every single API I've breached, I exploited a BOLA vulnerability to gain unauthorized access to data, insert data into the backend database, or take control of the device or automobile.

The results from this research were produced from two weeks of API testing and 6 months of static code analysis. The number of hardcoded keys and tokens in these mHealth apps, not just to the mHealth company's APIs, but also third-party payment processors, was astonishing. Basic cybersecurity hygiene, such as not hard coding usernames and passwords in source code and authorizing all requests is an endemic problem in mHealth.

mHealth companies need to implement more of a zero-trust approach to the security of their apps and APIs, ensuring that just because someone is authenticated doesn't necessarily mean they are authorized to access the data. In some cases, tokens without a lifetime set were used to authenticate requests allowing for replay attacks of those API requests. Combining this with BOLA vulnerabilities allows anyone, with or without an account, to request PHI for any patient for an unlimited amount of time because of a lack of expiration periods on tokens in many of the apps.

mHealth companies must do better, especially when it comes to our most sensitive personal health information. The lack of security controls and insecure code writing left me stunned as I completed this research. There is a clear lack of static code analysis and penetration testing that would have mitigated many of the "low hanging fruit" issues I discovered.

While I'm a big proponent of shift-left security and practicing good cybersecurity hygiene, it's just one of the hardening steps that should be taken to secure your APIs. The Approov solution effectively stopped 100% of all my attacks against the APIs. Its efficacy lies in the fact that it determines if the traffic being ingested into the API is synthetic or human, prevents APIs from being traced and reverse engineered, implements channel hardening (dynamic pinning), and allows only authentic apps to make API calls. Had all of these companies implemented Approov, these attacks wouldn't have been possible. For more information on Approov, please contact the sponsors of this research at https://www.approov.io

A video containing the results of this research is available at https://alissaknight.co/hacking_mhealth_trailer

SOURCES

[1] Research 2 Guidance, & Nikolova, S. (2017). 84,000 health app publishers in 2017 – Newcomers differ in their go-to-market approach. https://research2guidance.com/84000-health-app-publishers-in-2017/

[2] Aite Group, & Knight, A. V. (2019). The Vulnerability Epidemic in Financial Services Mobile Apps. Digital.ai. https://info.digital.ai/aite-research-financial-mobile-apps.html

[3] Experian, & Stack, B. (2017, December). Here's How Much Your Personal Information Is Selling for on the Dark Web. Experian. https://www.experian.com/blogs/ask-experian/heres-how-much-your-personal-information-is-selling-for-on-the-dark-web/

[4] World Health Organization. (2011). mHealth New horizons for health through mobile technologies: Based on the findings of the second global survey on eHealth (ISBN 978 92 4 156425 0). http://www.who.int/goe/publications/goe_mhealth_web.pdf

[5] Innovatemedtec. (n.d.). mHealth. Retrieved December 28, 2020, from https://innovatemedtec.com/digital-health/mhealth

[6] Safavi, K., Accenture, & Kalis, B. (2020, August). Digital Health Consumer Survey 2020 | Accenture. Accenture. https://www.accenture.com/us-en/insights/health/leaders-make-recent-digital-health-gains-last?utm_source=newsletter&utm_medium=email&utm_campaign=newsletter_axiosvitals&stream=top-stories

[7] Gartner Research, Zumerle, D., D'Hoinne, J., & O'Neill, M. (2019, August). API Security: What You Need to Do to Protect Your APIs (No. G00404900). Gartner Research. https://www.gartner.com/en/documents/3956746/api-security-what-you-need-to-do-to-protect-your-apis

SOURCES

[8] Penetration Testing Lab. (2017, February 6). Reverse Engineering Android Applications. https://pentestlab.blog/2017/02/06/reverse-engineering-android-applications/

[9] Bresnick, J. (2017, July 17). 4 Basics to Know about the Role of FHIR in Interoperability. HealthITAnalytics. https://healthitanalytics.com/news/4-basics-to-know-about-the-role-of-fhir-in-interoperability

[10] Dawson, M., Burrell, D. N., Rahim, E., & Brewster, S. (2009, December). Integrating Software Assurance into the Software Development Life Cycle (SDLC). https://www.researchgate.net/publication/255965523_Integrating_Software_Assurance_into_the_Software_Development_Life_Cycle_SDLC

[11] Cloud Security Alliance. (2020, July). The Six Pillars of DevSecOps: Automation. https://cloudsecurityalliance.org/artifacts/devsecops-automation/

[12] Rowley, J. (2020, July 21). What is Certificate Pinning? DigiCert. https://www.digicert.com/dc/blog/certificate-pinning-what-is-certificate-pinning/

[13] Auth0. (n.d.). Auth0 Documentation: Authorization. Auth0 Documentation. Retrieved January 6, 2021, from https://auth0.com/docs/authorization

ABOUT THE AUTHOR

Alissa Knight is a partner at Knight Ink and blends influencer marketing, content creation in writing and video production, go-to market strategies, and strategic planning for telling brand stories at scale in cybersecurity.

She achieves this through ideation to execution of content strategy, storytelling, and execution of influencer marketing strategies that take cybersecurity buyers through a brand's custom curated journey to attract and retain them as long-term partners.

Alissa is a published author, having published the first book on hacking connected cars and is working on a new series of books into hacking and securing APIs and microservices.

ABOUT KNIGHT INK

Firm Overview

Knight Ink is a content strategy, creation, and influencer marketing agency founded for category leaders and challenger brands in cybersecurity to fill current gaps in content and community management. We help vendors create and distribute their stories to the market in the form of written and visual storytelling drawn from 20+ years of experience working with global brands in cybersecurity. Knight Ink balances pragmatism with thought leadership and community management that amplifies a brand's reach, breeds customer delight and loyalty, and delivers creative experiences in written and visual content in cybersecurity.

Amid a sea of monotony, we help cybersecurity vendors unfurl, ascertain, and unfetter truly distinct positioning that drives accretive growth through amplified reach and customer loyalty using written and visual experiences.

Knight Ink delivers written and visual content through a blue ocean strategy tailored to specific brands. Whether it's a firewall, network threat analytics solutions, endpoint detection and response, or any other technology, every brand must swim out of a red sea of competition clawing at each other for market share using commoditized features. We help our clients navigate to blue ocean where the lowest price or most features don't matter.

We work with our customers to create a content strategy built around their blue ocean then perform the tactical steps necessary to execute on that strategy through the creation of written and visual content assets unique to the company and its story for the individual customer personas created in the strategy setting.

Contact Us

Web: www.knightinkmedia.com
Phone: (702) 637-8297
Address: 1980 Festival Plaza Drive, Suite 300, Las Vegas, NV 89135

Knight Ink
1980 Festival Plaza Drive
Suite 300
Las Vegas, NV 89135
ak@knightinkmedia.com