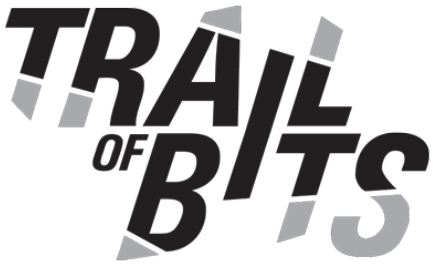

Voatz

Security Assessment
Volume I of II: Technical Findings

March 12, 2020

Prepared For:
Bradley Tusk | *Tusk Philanthropies*
btusk@tuskholdings.com

Nimit Sawhney | *Voatz*
ns@voatz.com

Aileen Kim | *Tusk Philanthropies*
akim@tuskstrategies.com

Sheila Nix | *Tusk Philanthropies*
sheila@tuskholdings.com

Prepared By:
Stefan Edwards | *Trail of Bits*
stefan.edwards@trailofbits.com

JP Smith | *Trail of Bits*
jp@trailofbits.com

Dan Guido | *Trail of Bits*
dan@trailofbits.com

Evan Sultanik | *Trail of Bits*
evan.sultanik@trailofbits.com

Changelog:

February 21, 2020:	Final report delivered
March 2, 2020:	Added fix log
March 11, 2020:	Added additional responses from Voatz

Executive Summary 5

Project Dashboard 8

Engagement Goals 11

Client-Side Questions 11

Backend Questions 11

Communications Protocol Questions 12

Procedural Questions 12

Timeline of Asset Discovery and Delivery 13

Coverage 17

Security Properties and Questions 19

Client-Side 19

Does the client-side security in this system provide reasonable tamper protection? 19

Is sensitive information stored on the client safe? 19

Does the Voatz app use or collect any location data? If so, why? 19

Does the Voatz app have appropriate security controls to ensure a user who leaves their device unattended is not further compromised? 20

Is the AndroidManifest.xml configuration sufficient? Does it allow unencrypted traffic or device backups where it should not? 20

Is SIM swapping sufficient to steal a voter's account? 20

Backend 20

Are requests appropriately time-throttled at the API endpoints? 20

Can resources be exhausted by forcing the backend to store large strings/data? 20

Does the system properly block devices so other devices on the same network will not be blocked? 21

Exactly how is the authentication data associated with the unique ID later assigned to each voter? Is the mapping predictable? 21

How does the system prevent voters from being identified by the approximate time of their vote, i.e., the time at which their ballot was recorded on the blockchain? 21

How does Voatz ensure that multiple voters do not vote from the same phone? 21

Do all cryptographic functions use cryptographically secure sources of randomness? 21

Are nonces chosen properly for Voatz' AES/GCM implementation? 22

Is the nimsim.com domain properly registered to prohibit malicious transfer? 22

How are Voatz employees prevented from looking up a specific voter's ballot? 22

How does the system handle arbitrarily large write-ins? 22

What if a voter's voting data is too large to fit in a QR code receipt? 22

Are elements reflected on the administrative web interface susceptible to cross-site scripting attacks? 22
Communications Protocol 23
Does each client use certificate pinning to communicate with the Voatz backend? 23
Does the Voatz backend use certificate pinning to communicate with third party APIs (e.g., Jumio)? 23
Is SSL configured securely? 23
Is sensitive information in requests encrypted? 23
Are the encryption schemes used in communication sufficient? 23
Procedural 24
Can voting data be de-anonymized? If so, how? 24
Can a user trigger a ban for an account/device that is not their own? 24
How does Voatz prevent double-voting? 24
Does the system properly handle two devices that both try to register as the same voter? 25
Are spoiled ballots appropriately spoiled? Can a user ever force a spoiled ballot to be counted? 25
If a voter's “anonymous code” on their ballot/receipt is compromised, can it allow the attacker to overwrite or invalidate their vote? Could an attacker vote in place of the original voter? What remediation exists here? 25
Is Voatz “E2E-V”? 26
Can a voter independently verify that their ballot receipt is valid? 26
Does Voatz satisfy Smyth, et al.’s notion of verifiability? Can the public independently validate that their votes were tallied correctly? 26
Are Voatz votes fungible between elections? 26
When a voter requests a receipt, what unique ID does the voter use to identify their ballot? 26
How does Voatz implement the “mixnet” anonymization described in the Voatz FAQ? 26

Recommendations Summary 27
Short Term 27
Long Term 32

Findings Summary 35

Backend Findings 39
1. Device IDs not validated against inner request device IDs 39
2. Amazon admin password is hardcoded in source file 41
3. Non-Anonymous ballot receipts are encrypted with AES-CBC using hardcoded key and IV 42
4. Secrets are stored in environment variables sourced from bash script 43

5. API for the onboarding workflow prohibits partitioning cloud resources for concurrent elections 44
6. Receipt and affidavit filename collisions 45
7. A voter can unregister another voter's device 46
8. Input keying material for AES GCM encoding is sent to Graylog 48
9. Voatz backend SSL key has a subdomain wildcard 49
10. Clients can specify their own audit token 50
11. Test parameters in the registration APIs can bypass SMS verification 51
12. QR code receipt generation will fail for large non-anonymous ballots 52
13. Session token validation ignores idle timeout 53
14. Receipt encryption is weak and can leak confidential information 54
15. Insufficient device ID validation on backend 55
16. Potential resource exhaustion via logging/storage of unsanitized data 57
17. Resource exhaustion via specially crafted Zimperium threats 58
18. Zimperium checks on the backend are a blacklist, not a whitelist 60
19. AES-GCM key/nonce/tag encryption system breaks authenticity 61
20. Unauthenticated ECDH is vulnerable to key compromise impersonation 62
21. AES-GCM keys, nonces, and "tag"s are encrypted using AES-ECB 63
22. Voatz API server lacks OCSP stapling 64
23. Empty ballots are not recorded in Hyperledger 65
24. Database root credentials stored in git 66
25. Signed voter affidavits are sent to an administrative email 67
26. AES-GCM AAD usage is nonstandard 68
27. Session cookie expiration offset is a hardcoded literal 69

Android Findings 71

28. Encrypted application data is trivially brute-forceable 71
29. PBDKF2 provides insufficient security margin for PIN codes 73
30. Third-party apps can capture the Android client screen and read screenshots taken from the client 74
31. Android release build signing key password and keystore password stored in git 75
32. A malicious website can read from the Android client's internal storage 76
33. Insufficient Android deviceId construction 77
34. Android client does not use the SafetyNet Attestation API 78
35. Android client does not use the SafetyNet Verify Apps API 79
36. Certificate pinning is only configured for the main Voatz domain 80
37. No explicit verification of the Android Security Provider 81
38. Jumio Netverify API credentials stored in git 82
39. Google Services API key stored in git 83
40. A malicious website may be able to execute JavaScript within the Android client 84

iOS Findings **85**
41. The iOS client does not disable custom keyboards 85
42. The iOS client does not use system-managed login input fields 86
43. iOS client keychain items are not excluded from iCloud and iTunes backups 88
44. Cryptographic credentials are not generated in the iOS Secure Enclave 89
45. iOS client disables App Transport Security (ATS) 90
46. iOS client is vulnerable to object substitution attacks 92
47. An iOS user can lose their registration 93
48. iOS client is susceptible to URI scheme hijacking 94

A. Vulnerability Classifications **95**

B. Review of Prior Security Assessments **97**
1. July 2018 97
2. October 2018 97
3. December 2018 98
4. October 2019 98
The MIT Report 98
B.1 Side-channel information leak 100
B.2 Voter disenfranchisement via network disruption 100
B.3 On-device security circumvention 101
B.4 GUI modification and data exfiltration 101
B.5 PIN cracking 102
B.6 Server compromise 102

C. Insufficient validation of encrypted API requests **103**

D. Verifiability and Voatz **107**
End-to-end verifiability 107
Verifiability notions for e-voting protocols 107

E. Fix Log **109**
Finding status 109
Detailed fix log 113
Unaddressed findings and unverified fixes 116

Executive Summary

On December 18th, 2019, Tusk Philanthropies and Voatz engaged Trail of Bits to review the security of the Voatz mobile voting platform. Trail of Bits conducted this assessment over the course of twelve (12) person-weeks with five (5) engineers working from commit hash 3443f4a of the Voatz Core Server repository, commit hash 07d1adb of the Voatz Android Client, commit hash d8436c1 of the Voatz iOS client, and commit hash 69d7a8b of the Voatz Administrative Web Interface.

To the best of our knowledge, this is the first "white-box" assessment of the Voatz system, and the first assessment to include in its scope the discovery of Voatz Core Server and backend software vulnerabilities. Our report and any conclusions drawn from it are only meant to reflect the security of the Voatz solution, not mobile voting in general. Review of election proceedings, both prior and current, was not in-scope for this assessment.

This report is divided into two volumes:

1. The security assessment's technical findings
2. A threat model containing architectural and operational findings

The assessment was scheduled to take place from January 21 through February 14, 2020, but ultimately stretched to February 21, 2020 due to a combination of delays in receiving code and assets, the unexpected complexity and size of the system, and the associated reporting effort. The Voatz system has over two dozen components in its architecture. Trail of Bits' engineers made their best effort to manually inspect each piece of code; however, this required each engineer to analyze, on average, almost 3,000 pure lines of code across 35 files per day of the assessment in order to achieve minimal coverage. Trail of Bits was only provided a backend for live testing on the second-to-last scheduled day of the assessment, and was asked not to attack or maliciously alter the instance in such a way that it would deny service to other concurrent audits sharing it. Therefore, almost all of the findings in this report are the result of manual analysis of the codebase. A detailed timeline of asset discovery and furnishment is also provided.

The assessment resulted in forty-eight (48) findings, of which a third are high severity, another quarter medium severity, and the remainder a combination of low, undetermined, and informational severity. The high-severity findings are related to:

- Cryptography, *e.g.*, improper use of cryptographic algorithms, as well as *ad hoc* cryptographic protocols
- Data exposure, *e.g.,* sensitive credentials available to Voatz developers and personally identifiable information that can be leaked to attackers, and

- Data validation, *e.g.*, a family of findings related to reliance on unvalidated data provided by the clients.

The use of the Hyperledger Fabric blockchain mimics the functionality of a distributed database with auditability. The assessed version of Voatz no longer uses any custom chaincode or smart contracts; all data validation and business logic are executed off-chain in the Scala codebase of the Voatz Core Server. Several high-risk findings were the result of data validation issues and confused deputies in the Core Server that could allow one voter to masquerade as another before even touching the blockchain.

Storing voting data on a blockchain maintains an auditable record to prevent fraud, but this comes at the expense of both privacy and increased attack surface. Clients do not connect directly to the blockchain themselves, and are therefore unable to independently verify that their votes were properly recorded. Anyone with administrative access to the Voatz backend servers will have enough information to fully reconstruct the entire election, deanonymize votes, deny votes, alter votes, and invalidate audit trails.

Other e-voting systems attempt to achieve the best of both worlds—cryptographic authentication, validation, and nonrepudiation *as well as* provable privacy—by using exotic cryptographic schemes like zero-knowledge proofs and forms of secure multiparty computation. However, these, like proof-of-authority blockchains, are nascent technologies that are exceedingly hard to implement correctly, as was recently demonstrated by the failure of Swiss Post's e-voting experiment. Throughout this engagement, Trail of Bits has provided assistance to Voatz in navigating this complex trade space to mitigate the risks presented by voting systems in general and, if possible, avoid issues that have plagued other experimental voting systems.

Voatz' code, both in the backend and mobile clients, is written intelligibly and with a clear understanding of software engineering principles. The code is free of almost all the common security foibles like cryptographically insecure random number generation, HTTP GET information leakage, and improper web request sanitization. However, it is clear that the Voatz codebase is the product of years of fast-paced development. It lacks test coverage and documentation. Logical checks for specific elections are hard-coded into both the backend and clients. Infrastructure is provisioned manually, without the aid of infrastructure-as-code tools. The code contains vestigial features that are slated to be deleted but have not yet been (TOB-VOATZ-009). Validation and cryptographic code are duplicated and reimplemented across the codebase, often erroneously (TOB-VOATZ-014). Mobile clients neglect to use recent API features of Android and iOS (TOB-VOATZ-034 and TOB-VOATZ-042). Sensitive API credentials are stored in the git repositories (TOB-VOATZ-001). Many of its cryptographic protocols are nonstandard (TOB-VOATZ-012).

There is a great deal of uncertainty and public speculation about Voatz' implementation and security properties. Therefore, we sought to investigate a series of questions that

would address the overall security posture, guarantees, and behavior of the Voatz system. The answers to these questions are in the Security Properties and Questions section.

Voatz should immediately address all of the recommendations in the "Short Term" section of our recommendations summary, especially those related to high-severity issues. High priority should be given to remediating data sanitization of device IDs, improper use of cryptography, and overreliance on the authenticity and honesty of client implementations. Operationally, the system is also in dire need of infrastructure management automation. Overall, it seems that Voatz is struggling to manage a codebase of its size while concurrently, manually managing election pilots. We hope that this assessment will improve the overall security posture of the Voatz system, but there is still a great deal of work to be done to achieve that goal.

Update: On February 27, 2020, Trail of Bits reviewed fixes proposed by Voatz for the issues presented in this report. Eight (8) issues were addressed, and forty (40) issues remain partially or fully unfixed. See a detailed review of the current status of each issue in Appendix E: Fix Log.

Project Dashboard

Application Summary

Name	Voatz Core Server
Version	Git Commit 3443f4aa878719fb60a2bfb358954715158d8af1 Branches: `develop`
Type	Scala
Platforms	*nix
Number of Source Files	961
Lines of Code	71k
Lines of Comments	13k

Name	Voatz Android Client
Version	Git Commit 07d1adba25a471dc460c8e5f37151488cb1e8102 Branches: `develop`
Type	Java, Kotlin
Platforms	Android
Number of Source Files	338
Lines of Code	26k
Lines of Comments	2k

Name	Voatz iOS Client
Version	Git Commit d8436c1065eadb6e9bcd73ae225c79d604bdf16b Branches: `development`
Type	Swift
Platforms	iOS
Number of Source Files	410
Lines of Code	41k
Lines of Comments	8k

Name	Voatz Administrative Web Interface
Version	Git Commit 69d7a8bbcb38269bd2c553ed48f22763a6079e6f Branches: `develop`
Type	Angular, TypeScript
Platforms	Web
Number of Source Files	353
Lines of Code	30k
Lines of Comments	2k

Name	Voatz Auditing Web Portal
Version	Unknown; given access to a live instance, but never received source code.
Type	Unknown
Platforms	Web
Number of Source Files	Unknown
Lines of Code	Unknown
Lines of Comments	Unknown

Engagement Summary

Dates	January 21 through February 21, 2020
Method	White-box
Consultants Engaged	5
Level of Effort	12 person-weeks

Vulnerability Summary

Total High-Severity Issues	16	■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■
Total Medium-Severity Issues	12	■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■
Total Low-Severity Issues	10	■ ■ ■ ■ ■ ■ ■ ■ ■ ■
Total Undetermined-Severity Issues	6	■ ■ ■ ■ ■ ■
Total Informational-Severity Issues	4	■ ■ ■ ■
Total	48	

Category Breakdown

Access Controls	3	■ ■ ■
Configuration	6	■ ■ ■ ■ ■ ■
Cryptography	12	■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■
Data Exposure	12	■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■
Data Validation	9	■ ■ ■ ■ ■ ■ ■ ■ ■
Denial of Service	2	■ ■
Patching	1	■
Session Management	3	■ ■ ■
Total	48	

Engagement Goals

The engagement was scoped to provide a security assessment of Voatz' entire backend infrastructure, mobile clients, administrative web interface, and election auditing web portal.

There is a great deal of uncertainty and public speculation about Voatz' implementation and security properties. For example, researchers from Lawrence Livermore National Laboratory, the University of South Carolina, Citizens for Better Elections, Free & Fair, and the US Vote Foundation enumerated a series of questions about the security of Voatz in May, 2019. More recently, researchers at MIT discovered vulnerabilities in the Voatz Android client and speculated about—but had no proof of—related issues in the backend implementation (see Appendix B for a detailed discussion of MIT's findings).

Therefore, we sought to address a series of questions falling into four categories: Client-Side, Backend, Communications Protocol, and Procedural. The answers to these questions are in the Security Properties and Questions section.

Client-Side Questions

- Does the client-side security in this system provide reasonable tamper protection?
- Is sensitive information stored on the client safe?
- Does the Voatz app use or collect any location data? If so, why?
- Does the Voatz app have appropriate security controls to ensure a user who leaves their device unattended is not further compromised?
- Is the AndroidManifest.xml configuration sufficient? Does it allow unencrypted traffic or device backups where it should not?
- Is SIM swapping sufficient to steal a voter's account?

Backend Questions

- Are requests appropriately time-throttled at the API endpoints?
- Can resources be exhausted by forcing the backend to store large strings/data?
- Does the system properly block devices so other devices on the same network won't be blocked?
- Exactly how is the authentication data associated with the unique ID later assigned to each voter? Is the mapping predictable?
- How does the system prevent voters from being identified by the approximate time of their vote, *i.e.,* the time at which their ballot was recorded on the blockchain?
- How does Voatz ensure that multiple voters do not vote from the same phone?
- Do all cryptographic functions use cryptographically secure sources of randomness?
- Are nonces chosen properly for their AES/GCM implementation?

- Is the nimsim.com domain properly registered to prohibit malicious transfer?
- How are Voatz employees prevented from looking up a specific voter's ballot?
- How does the system handle arbitrarily large write-ins?
- What if a voter's voting data is too large to fit in a QR code receipt?
- Are elements reflected on the administrative web interface susceptible to cross-site scripting attacks?

Communications Protocol Questions

- Does each client use certificate pinning to communicate with the Voatz backend?
- Does the Voatz backend use certificate pinning to communicate with third party APIs (*e.g.*, Jumio)?
- Is SSL configured securely?
- Is sensitive information in requests encrypted?
- Are the encryption schemes used in communication sufficient?

Procedural Questions

- Can voting data be de-anonymized? If so, how?
- Can a user trigger a ban for an account/device that is not their own?
- How does Voatz prevent double-voting?
- Does the system properly handle two devices that both try to register as the same voter?
- Are spoiled ballots appropriately spoiled? Can a user ever force a spoiled ballot to be counted?
- If a user's "anonymous code" at the bottom of their ballot/receipt is compromised, can it allow the attacker to overwrite or invalidate the user's vote? Could the attacker vote in place of that user?
- Is Voatz "E2E-V"?
- Can a voter independently verify that their ballot receipt is valid?
- Does Voatz satisfy Smyth et al.'s notion of verifiability? Can the public independently validate that their votes were tallied correctly?
- Are Voatz votes fungible between elections?
- When a voter requests a receipt, what unique ID does the voter use to identify their ballot?
- How does Voatz implement the "mixnet" anonymization described in the Voatz FAQ?

Timeline of Asset Discovery and Delivery

Asset	Date Requested	Date Delivered	Version
System Documentation	01/09/2020	01/21/2020	
Prior Audit Reports (Redacted)	01/09/2020	01/21/2020	
Core Server Backend for Live Testing	01/09/2020	02/13/2020	This backend is shared among concurrent audits and therefore cannot be used for testing attacks that could result in corruption or denial of service
CoreServer Source Code	01/09/2020	01/23/2020	Commit 3443f4a from 01/22/2020 Given access solely to the `develop` branch Note that on February 6, at the end of the third week of our assessment, we discovered that 883 files were deleted from the Core Server git repository immediately before it was delivered to us. The majority of these were JSON and SQL files used to seed the databases. However, the deleted files also included sensitive API tokens/secrets/credentials and documentation that would have been useful earlier in the assessment.
Admin Web UI Source Code	01/09/2020	01/22/2020	Commit 69d7a8b from 01/22/2020 Given access solely to the `develop` branch

Android Client Source Code	01/09/2020	01/22/2020	Commit 07d1adb from 01/21/2020 Given access solely to the `develop` branch
iOS Client Source Code	01/09/2020	01/22/2020	Commit d8436c1 from 01/21/2020 Given access solely to the `development` branch
Audit Portal Source Code	01/09/2020	Not Furnished	
Cryptographic Protocol Documentation	01/27/2020	01/27/2020	`Confidential-VMA-InitialHandshake-111219-0635-353.pdf` SHA256 2ba9b4102198636e95f54f9e2d5478040494d67e6700d9ef96b6040440c59d43
Sample CoreServer Apache Configuration	01/28/2020	02/04/2020	`Virtualhost.071.voatzapi-alpha1.conf` SHA256 b6ea48f5f31d470f218ee42479fc1c1a6a015b0bd0cee3a350e9059cb363b507
NIST 800-60 Catalog of System Components	01/28/2020	Not Furnished	
Hyperledger Chaincode/ Smart Contract Source Code	01/29/2020	01/29/2020	Voatz no longer uses any chaincode or smart contracts.
SSH Configs	01/29/2020	Not Furnished	
AWS Configs	01/29/2020	Not Furnished	

Apache Configs	01/29/2020	Not Furnished	
Access Credentials for the Audit Application	01/29/2020	02/10/2020	
ACL and Bucket Policy for S3 Bucket `vinc2018` in us-east-1	01/30/2020	02/14/2020	
ACLs /IAMs/ Profiles for All Cloud Assets	01/30/2020	02/14/2020	Additional policy furnished for the “cpe” S3 bucket, but nothing else.
Firebase Security Rules	01/30/2020	02/12/2020	`firebase-securityrules.txt` SHA256 bb16213809bcc3f910812e970285987deda327aadc0cbb6d1440b8a5d78b4cf6
Example `./config/application-*` `MainConfig` file for the backend	01/30/2020	Not Furnished	
iOS `.ipa`	02/02/2020	02/05/2020	SHA256 5a61d052acb7bd7f42c24e3c15cc183b673a896155e6f06351de0e10a2476e0b
Android `.apk`	02/02/2020	02/04/2020	SHA256 ed8af86865d9d4886db400f7df768e805d378fb956955b1a42258ad8c0821219
Anonymized MIT Security Report Summary (Appendix B)	N/A	02/06/2020	SHA256 c98d1da408dc083e134833e46fde848d4bfcec2c9a0572ed5edfde8fd6697f60

Original DHS CISA HIRT Assessment Report	02/20/2020	02/20/2020	SHA256 39d1a3cbb8ded2efb2f83d9d9434a8bd57ecfb161cb284606bd68fa8b44b07e1

Coverage

Trail of Bits was provided over 168,000 lines of source code across approximately 2,100 files, not including white space, comments, configuration files, and documentation. These source code files were distributed across four git repositories, implemented in several different programming languages, and interoperable with each other via REST APIs. The system is unusually complex, with over two dozen components in its architecture. Trail of Bits' engineers made their best effort to manually inspect each piece of code; however, this required each engineer to analyze, on average, almost 3,000 pure lines of code across 35 files per day of the assessment in order to achieve minimal coverage. The quantity of findings discovered during this assessment, combined with the complexity of the system, leads us to believe that other vulnerabilities are latent. Therefore, our main focus was on the Core Server codebase, as it provides a common interface and attack surface for all of the other components, followed by the mobile clients.

Trail of Bits was not given a backend for live testing of malicious attack vectors. The backend instance to which we were eventually given access was only provided on the second-to-last scheduled day of the assessment. This severely limited our ability to test the mobile applications in a production-like setting, test attack vectors, and confirm exploit scenarios on a live system. Therefore, all of the findings in this report are the result of static analysis of the codebase.

Voatz Core Server. This includes the backend API that handles onboarding, vote submission, and interactions with MySQL, MongoDB, and Hyperledger. The Core Server codebase also includes the Receipt Service responsible for generating, storing (S3), and delivering (Sendgrid) receipts. Trail of Bits was only given a single Apache configuration file for the Core Server and no other configs. We did not have access to any of the other system components necessary to run the backend (*i.e.*, MySQL, MongoDB, Hyperledger, Zimperium, Sendgrid, Twilio, Names, Jumio, S3, etc.). Therefore, Trail of Bits' analysis of the core server was limited to manual analysis of the Scala codebase.

Voatz Android & iOS Clients. Manual analysis of the codebase and automated static analysis using Data Theorem's App Secure and MobSF.

Voatz Administrative Web Interface. Manual analysis of the codebase. Without a sufficient backend to run the admin web portal, and with no running instance to test on, our ability to test for certain classes of vulnerability, (such as cross-site scripting attacks [XSS]), was limited.

Voatz Audit Web Portal. We performed manual, client-side testing of a live instance, but had no access to source code. Our testing was limited to non-harmful attacks since this live instance was being used for an active audit.

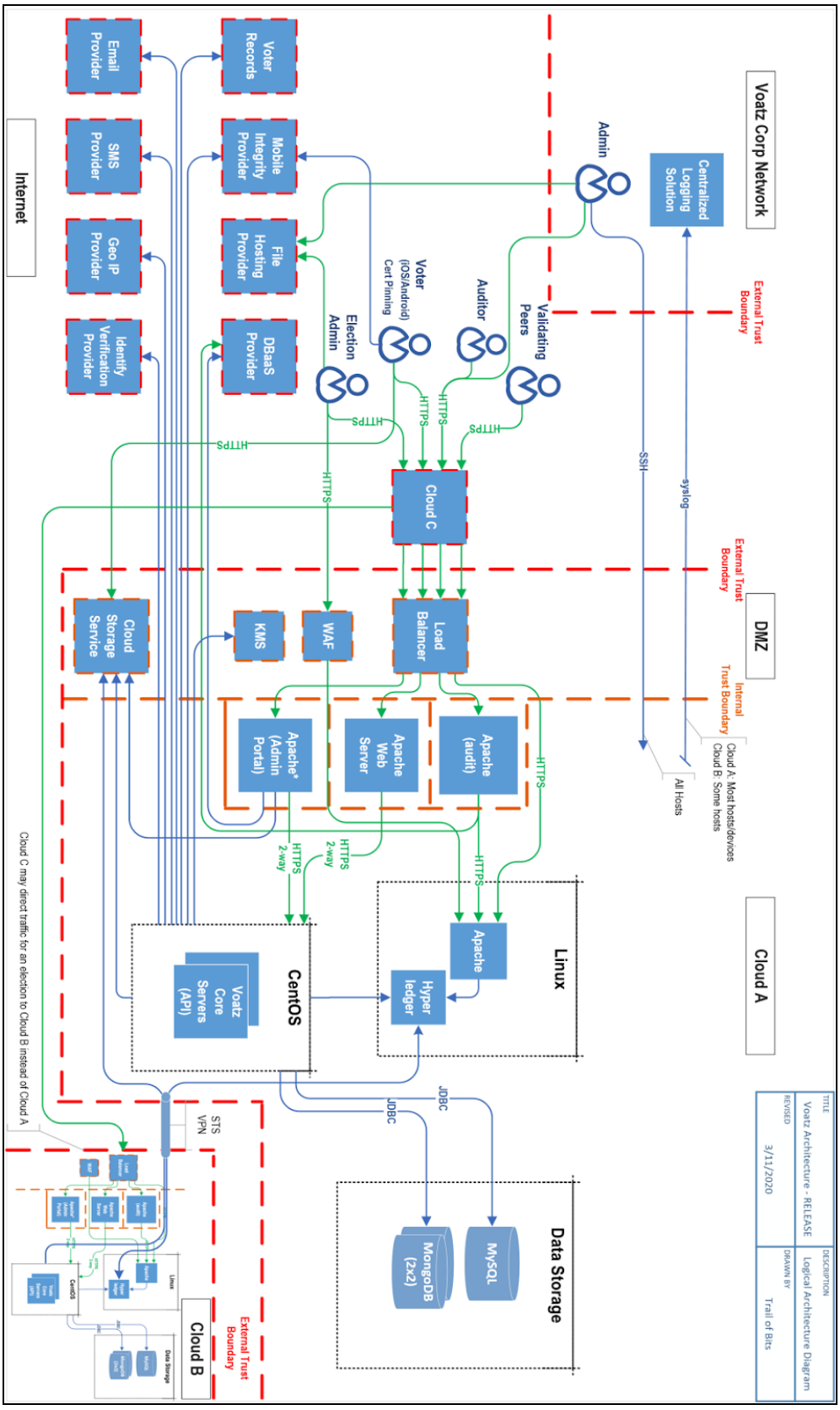

Figure 1: Dataflow diagram of the Voatz system generated as a result of this engagement

Security Properties and Questions

The following is a summary of the security properties tested during this assessment, as well as security-related questions that have been answered.

Client-Side

Does the client-side security in this system provide reasonable tamper protection?

Except for Zimperium, a commercial library that uses proprietary checks to detect tampering, there is no tamper protection. Zimperium can be circumvented (see TOB-VOATZ-029 and B.3). Voatz also uses a complex, custom cryptographic handshake protocol. However, in a matter of weeks, MIT researchers successfully reverse-engineered the protocol[1] (see Appendix B).

Is sensitive information stored on the client safe?

No, several reasonable measures could have been implemented on both the Android and iOS clients to better protect the sensitive information stored on them. For example, third-party apps can surreptitiously take screenshots of sensitive information on the Android app (see TOB-VOATZ-032). Similarly, the iOS client does not disable custom keyboards that can (and often do) record and exfiltrate keystrokes (see TOB-VOATZ-040). Additionally, Voatz data stored in the iOS keychain is not excluded from backup to iCloud or iTunes (see TOB-VOATZ-043).

The Android app does not use the Google SafetyNet APIs (see TOB-VOATZ-037 and 045), and it does not explicitly verify that the newest version of the Android security provider is running (see TOB-VOATZ-034). The clients are also vulnerable to PIN number brute force attacks (see TOB-VOATZ-048 and B.5). Generally, Voatz lacks protection against malicious applications that could access sensitive Voatz information, except for that provided by Zimperium, which can be disabled (see B.3).

Does the Voatz app use or collect any location data? If so, why?

Yes, the mobile clients force the voter to enable location services while the app is running. During new user onboarding, the voter's latitude and longitude are sent to the server, and the server requires these fields to be non-empty in order for the voter to be registered. These coordinates are resolved via the GeoNames service and saved to MongoDB. User

[1] Note that the protocol has been modified since the MIT assessment; it no longer chooses the 57th generated keypair. However, the chosen key is still deterministic, so the generation of 100 keypairs provides no additional security. Therefore, the handshake protocol remains functionally identical to the version reverse-engineered and analyzed by MIT.

locations for most (if not all) other API requests are also logged to MongoDB. These are presumably used for auditing suspicious activity. However, accurate location reporting is predicated on the assumption that the voter has not maliciously tampered with their client. An attacker with the capability of passive network introspection can determine the location of a voter's client, which is potentially sufficient to de-anonymize their vote (*e.g.*, if a ballot is cast from home).

Does the Voatz app have appropriate security controls to ensure a user who leaves their device unattended is not further compromised?

No, sensitive data is trivially recoverable from the Android client and additional, related controls can be circumvented (see TOB-VOATZ-048, TOB-VOATZ-018, and B.5).

Is the AndroidManifest.xml configuration sufficient? Does it allow unencrypted traffic or device backups where it should not?

The `AndroidManifest.xml` prohibits cleartext traffic (HTTP) and device backups. It does request permission for the `CALL_PHONE` privilege, which appears to be related to a feature that lets the user call a phone number for live assistance. `network_security_config.xml` only pins the certificate for the Voatz domain and not any of the domains of its third-party services (see TOB-VOATZ-026).

Is SIM swapping sufficient to steal a voter's account?

Yes, SIM swapping/SS7 attacks are sufficient to steal a voter's account. However, an attacker must either 1) have access to the voter's email, or 2) have knowledge of the Voatz API and how to circumvent email-based side-channel verification (see TOB-VOATZ-022).

Backend

Are requests appropriately time-throttled at the API endpoints?

Customer and Organization authentication and onboarding are at least partially throttled, both by IP and device ID. Additional protections may be provided by Cloudflare.

Can resources be exhausted by forcing the backend to store large strings/data?

We could not test this on a live system since we were not provided with a backend that could be maliciously attacked in this way. However, there do not appear to be any protections that would prevent an attacker from storing or logging excessively large fields in a way that could lead to resource exhaustion. This could take the form of database storage exhaustion, bloating of logs, slow writing to disk, forcing a long regex query, etc.

Does the system properly block devices so other devices on the same network will not be blocked?

There is IP blacklisting, so a malicious client could block its entire (potentially shared) IP.

Exactly how is the authentication data associated with the unique ID later assigned to each voter? Is the mapping predictable?

For authentication within the API, voters are uniquely identified by the device ID reported by their mobile client—currently the device ID reported by their mobile OS. A modified or custom client can choose whichever ID it likes. Much (but not all) of Voatz' device ID processing code sanitizes device IDs, removing symbols, so there is the potential for ID collisions (see TOB-VOATZ-022). When a user re-registers, they can reset their device ID. In the re-registration workflow, each user is uniquely identified by their mobile phone number. The "Anonymous IDs" used for ballots and auditing are generated after a voter has registered. Vestigial code exists to generate audit tokens from hashed personally identifiable information from the client, and stores this data alongside a voter's customer ID in MySQL. However, the current implementation securely randomly generates audit tokens on the backend and sends this to the client. The backend does not store the audit token at all, and has no way of validating that an audit token is one that was officially generated by the backend. Therefore, Anonymous ID collisions can also be forced (see TOB-VOATZ-046).

How does the system prevent voters from being identified by the approximate time of their vote, *i.e.*, the time at which their ballot was recorded on the blockchain?

The Voatz system does not appear to have any mitigation for this type of de-anonymization. The Voatz FAQ talks about a mixnet for anonymizing votes, but we found no evidence of a mixnet in the code.

How does Voatz ensure that multiple voters do not vote from the same phone?

Votes are uniquely identified by a device ID, which is specified by `Settings.Secure.ANDROID_ID` on Android and `identifierForVendor.uuidString` on iOS. However, this device ID will change on iOS if the Voatz app is deleted and then reinstalled (see TOB-VOATZ-007). This allows two voters to vote on the same iPhone.

Do all cryptographic functions use cryptographically secure sources of randomness?

Yes, they all appear to properly use `java.util.SecureRandom` on the backend.

Are nonces chosen properly for Voatz' AES/GCM implementation?

No, see TOB-VOATZ-011 and TOB-VOATZ-024.

Is the nimsim.com domain properly registered to prohibit malicious transfer?

Yes, the domain registration lists the client delete, transfer, update, and renew prohibited Extensible Provisioning Protocol (EPP) status codes. The domain is registered at GoDaddy and we did not verify whether 2FA or other mitigations were in place.

How are Voatz employees prevented from looking up a specific voter's ballot?

Anyone with administrative access to a subset of MongoDB, S3, Graylog, and Hyperledger will have sufficient information to de-anonymize votes. Currently, only two designated Voatz team members have access to the credentials required to access these production instances, and they are only accessible from whitelisted IPs in the Voatz offices.

How does the system handle arbitrarily large write-ins?

There do not appear to be protections or limitations against arbitrarily large write-ins. QR-code receipt generation will fail on sufficiently large ballots (see TOB-VOATZ-009). However, QR-code receipts are only used in "non-anonymous" elections, which have not occurred since 2018, and are being phased out by Voatz. The PDF receipt generation does not appear to be similarly susceptible. A malicious client could potentially deny service to the server by posting sufficiently large API requests; however, this was not tested by Trail of Bits and could possibly be thwarted by Voatz' IP-based and temporal throttling.

What if a voter's voting data is too large to fit in a QR code receipt?

QR-code receipt generation will fail (see TOB-VOATZ-009).

Are elements reflected on the administrative web interface susceptible to cross-site scripting attacks?

Many request fields are not sanitized prior to being stored in the database. As a result, the burden of data validation and sanitization falls on the systems that read the fields, including the administrative web interface, audit portal, and receipt service. Although a review of the `adminwebui` did not identify any problematic reflection of these client-controlled fields, we cannot rule out the existence of such vulnerabilities. Trail of Bits was never provided with a copy of the audit portal source code, so we cannot comment on its resilience to this class of attack.

Communications Protocol

Does each client use certificate pinning to communicate with the Voatz backend?

TrustKit forces pinning, but only to the main Voatz domain (see TOB-VOATZ-026). Also, neither voatz.com nor voatzapi.nimsim.com are configured with OCSP Stapling (see TOB-VOATZ-033). This will eventually become a requirement for apps submitted to the iOS app store.

Does the Voatz backend use certificate pinning to communicate with third party APIs (*e.g.*, Jumio)?

No, Voatz only cert-pins the core Voatz API server (see TOB-VOATZ-026).

Is SSL configured securely?

Not entirely. All of the API endpoints we tested are configured with secure TLS 1.2 cipher suites. Voatz claims that all TLS traffic is terminated at their Apache instances. However, the TLS cipher suites returned by the Voatz servers appear to be different from the ones specified in the sole Apache config furnished to Trail of Bits. This suggests that either the Apache config is different from the one used in production, or that TLS is being terminated somewhere before Apache (*e.g.*, Cloudflare).

Moreover, all of the cloud servers use a shared SSL certificate with a subdomain wildcard, which could allow an attacker to commandeer the Voatz SSL private key (see TOB-VOATZ-028). The original Department of Homeland Security CISA HIRT assessment report from October of 2019 notes that Voatz maintains unpatched honeypots; we were not given access to these servers, but would recommend that they use distinct SSL credentials.

Is sensitive information in requests encrypted?

Yes, but with an *ad hoc* scheme and cryptographic handshake protocol. All API requests are made using HTTP POSTs over TLS 1.2. Some unique identifiers such as the caller's device ID are duplicated both inside and outside of the encrypted data, which can lead to confused deputy issues since the two are not validated against each other (see TOB-VOATZ-014 and Appendix C).

Are the encryption schemes used in communication sufficient?

Sensitive API calls are encrypted using an *ad hoc* scheme. The cryptography used throughout the system is non-standard (*e.g.,* TOB-VOATZ-012). Encrypted requests can often be posted with an arbitrary device ID specified by the client (TOB-VOATZ-014).

Procedural

Can voting data be de-anonymized? If so, how?

Yes, but not necessarily retroactively. The "Anonymous ID" on ballot receipts is a securely randomly generated "audit token" produced by the backend and sent to the client. The backend does not store these audit tokens. When a client submits a vote to the backend, the API request is partially authenticated by the client's customer ID and device ID, both of which are linked to the voter's personally identifiable information. Therefore, a Voatz administrator with access to the backend server can observe these API requests in real time and deanonymize votes. If a log is kept of client connections, this could also be retroactively temporally correlated with votes that were added to the blockchain.

Since the backend does not store a mapping of audit tokens to ballots, there is no way for votes to be directly retroactively de-anonymized, other than using temporal analysis.

Earlier versions of Voatz used a base64 encoded string containing a voter's personally identifiable information, including their name, mobile number, phone number, state, date of birth, and audit token. This was stored in MySQL. This vestigial implementation still exists in the codebase, but does not appear to be used in current elections.

Can a user trigger a ban for an account/device that is not their own?

Yes, see TOB-VOATZ-014 and TOB-VOATZ-022.

How does Voatz prevent double-voting?

At the beginning of an election, Voatz is seeded with a voter roll of all voters eligible for voting via Voatz, provided by the voting precinct. After a voter casts their ballot, Voatz records this state in MongoDB, preventing the user from voting again, since this ballot record is tied to the voter roll in addition to the customer and device IDs. However, these protections only prevent voters from double-voting through the Voatz app; there is no protection within the blockchain (*e.g.,* chaincode) that prevents double-voting.

Therefore, anyone who can modify MongoDB can permit a user to double-vote. Similarly, anyone with access credentials to Hyperledger can record arbitrary votes on the blockchain. It is incumbent upon the precinct to detect instances of double-voting involving a voter casting a ballot *both* through Voatz *and* via vote-by-mail.

Does the system properly handle two devices that both try to register as the same voter?

Yes. Currently, registering voters are uniquely identified by the combination of their email address and mobile phone number, both of which are provided by the voting precinct from voter registration data. The only way a user can register is if they provide an email address and mobile phone number that matches the data provided by the precinct. If a second device successfully registers with the same data, the previously registered device will be unregistered.

Are spoiled ballots appropriately spoiled? Can a user ever force a spoiled ballot to be counted?

Voatz has no automated capability to spoil a ballot. The protocol for spoiling a ballot is for the precinct to manually inform Voatz that a voter wishes their ballot to be spoiled. A Voatz administrator will manually reset the voter's account in MongoDB, allowing the voter to re-register from scratch, producing a new anonymous ID and, ultimately, a second ballot. From an auditor's perspective, the spoiled ballot and valid ballot will appear as two distinct ballots from different voters. It is incumbent on the precinct, not Voatz, to properly account for and discard the first, spoiled ballot.

If a voter's "anonymous code" on their ballot/receipt is compromised, can it allow the attacker to overwrite or invalidate their vote? Could an attacker vote in place of the original voter? What remediation exists here?

The "anonymous code" on each ballot/receipt is the user's "audit token" generated by the Core Server. This is a base64-encoded string of 64 random bytes concatenated with a timestamp and base64-encoded again. We discovered that a lack of sufficient data validation resulted in a high-severity vulnerability in which the client itself can specify an arbitrary audit token to certain API endpoints, including the endpoint for casting a ballot (TOB-VOATZ-046). Therefore, if a malicious user with a custom client knows another voter's anonymous code, their ballots can be crafted to have the same anonymous code. This will *not* cause one vote to overwrite or invalidate another, but it may cause confusion and cast doubt on the integrity of the election during an audit.

A malicious user with a custom client can also use that same device ID to re-register with the system. However, the attacker would need to spoil the user's previous ballot by first passing another identity check. We discovered one vulnerability in which knowledge of a voter's device ID allows an attacker to unregister a voter before they cast their ballot, but it would not allow the attacker to overwrite or invalidate a user's pre-existing vote (TOB-VOATZ-022).

Is Voatz "E2E-V"?

No, see Appendix D: Verifiability and Voatz.

Can a voter independently verify that their ballot receipt is valid?

No.

Does Voatz satisfy Smyth, *et al.*'s notion of verifiability? Can the public independently validate that their votes were tallied correctly?

No, it fails the notions of *Universal Verifiability* and *Eligibility Verifiability.* The public can neither validate that all votes were tallied correctly nor can they validate that all votes were cast by authorized voters. Even for Voatz auditors, it fails the notion of *Eligibility Verifiability* since auditors only have access to anonymized votes and must trust that the Voatz system properly vetted the identities and eligibility of the voters.

Are Voatz votes fungible between elections?

We were unable to discover an attack vector that would allow this. An exploit that would allow fungibility between elections would likely necessitate voting infrastructure that is reused between elections. Voatz claims that it manually instantiates new infrastructure between elections. We were unable to confirm that there are no shared components (*e.g.*, S3 buckets or folders in their consumer cloud file hosting provider). The name of the S3 bucket is hard-coded into the source code and suggests that a single S3 bucket may have been continuously used between elections since 2018, however, we can neither confirm this nor, if true, validate whether the bucket was properly expunged between elections.

When a voter requests a receipt, what unique ID does the voter use to identify their ballot?

Their device ID. Ballot receipts are also uploaded to an Amazon S3 bucket, unencrypted, with a filename containing the voter's audit token reported from the client (see TOB-VOATZ-020).

How does Voatz implement the "mixnet" anonymization described in the Voatz FAQ?

There does not appear to be, nor is there mention of, a mixnet in the code provided to Trail of Bits. The core server has the capability to deanonymize all traffic, including ballots.

Recommendations Summary

This section aggregates all the recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Short Term

❑ **Rebase the git repositories.** Remove all sensitive API tokens and secrets. TOB-VOATZ-001, 002, 016, 017, and 047

❑ **Check whether credentials that were once stored in git are still active.** If so, revoke them and generate new ones. TOB-VOATZ-001 and 002

❑ **Replace ECDH and the system's *ad hoc* PAKE**. Use Noise or a TLS 1.3 handshake instead. These are authenticated and prevent key compromise impersonation. TOB-VOATZ-004

❑ **Refactor the session cookie expiration offsets.** Ensure that they are derived from a singular definition and can be uniformly refactored across the codebase. TOB-VOATZ-005

❑ **Remove all past voting data encrypted with a hardcoded key and IV.** Deprecate and remove the code for non-anonymous elections. Send receipts only over ephemeral channels with a unique channel key, determined dynamically. TOB-VOATZ-006

❑ **Document that an iOS voter can be un-registered by deleting and reinstalling the Voatz app.** Voatz has indicated that this is intended behavior, so voters should be made aware of their necessity to re-enroll after a reinstall. TOB-VOATZ-007

❑ **Check for the `null` case when obtaining the `ANDROID_ID`.** Document the effects of a factory reset operation on the Android `deviceId`, and provide clear instructions for how voters can remediate the problem if it occurs. TOB-VOATZ-008

❑ **Log all QR code generation failures.** Alert the voter on failure. TOB-VOATZ-009

❑ **Ensure that all backend infrastructure is re-provisioned between elections and not shared between elections/clients.** Each server should also receive its own unique SSL certificate and credentials. Use infrastructure-as-code tools like Ansible and Terraform to automate and manage provisioning. TOB-VOATZ-010 and 028

❑ **Remove all use of AES-ECB from the codebase.** ECB mode is famously insecure and has neither semantic security nor authentication properties. Replace it with AES-GCM. TOB-VOATZ-011

❑ **Remove the "tag" parameter from all AES-GCM usage.** This AAD usage is nonstandard and potentially dangerous. Additionally, remove code updating the AAD with a null buffer. TOB-VOATZ-012

❑ **Switch to a purpose-built secret management system.** Use Vault or Keywhiz. Rotate all credentials stored with the previous system. TOB-VOATZ-016

❑ **Refactor the handling of encrypted requests.** Ensure the `deviceId` used to index the shared secret used for encryption is validated against the device ID used in the underlying request. Alternatively, consider removing device IDs from the inner request to preserve bandwidth and alleviate any device ID conflicts. This will address the family of issues enumerated in TOB-VOATZ-014 and Appendix C

❑ **Do not use PDF encryption.** This will greatly enhance the security of receipts at rest. Replace the use of PDF encryption with age. TOB-VOATZ-015

❑ **Re-enable the session token idle timeout.** The check is currently commented out. TOB-VOATZ-018

❑ **Validate the `deviceId` to ensure it is of good form.** Block any users who attempt to register with an invalid device ID. These checks should account for case-sensitivity and the format of the identifier after the "`and-`" and "`ios-`" prefixes. TOB-VOATZ-019

❑ **Move all device ID sanitization and escaping as early as possible in the onboarding process.** Should happen when a device ID is first set. This can avoid device ID collisions. TOB-VOATZ-020

❑ **Ensure that the "admin notice inbox" is large enough to thwart any attempted spam overrun.** If the inbox is controlled by a precinct, they must be made aware of the necessary protections for this email address. TOB-VOATZ-021

❑ **Refactor the re-registration workflow.** Always require a second factor of authentication. It should also not rely on any information provided by the client. This can prevent one client from un-registering another. TOB-VOATZ-022

❑ **Remove log messages containing cryptographic keying material.** Ensure that production instances are configured to run at the highest log level for which the necessary auditing information is still recorded. TOB-VOATZ-023

❑ **Remove all use of AES-ECB from the Voatz system.** Instead of multiple levels of AES with encrypted keys and nonces, simply have one standard AEAD construction. TOB-VOATZ-024

❑ **Properly configure PBDKF2.** Update the number of iterations to 50,000. This surpasses the latest recommendation from NIST and is required given the low entropy of user PINs. TOB-VOATZ-025

❑ **Pin certificates for all third-party APIs.** This can be accomplished using TrustKit, and will prevent man-in-the-middle attacks. TOB-VOATZ-026

❑ **Improve documentation and training materials for auditors.** Be sure to inform auditors that a vote cast with an empty ballot will not appear in Hyperledger. TOB-VOATZ-027

❑ **Use different SSL credentials for each `nimsim.com` subdomain.** Ensure that out-of-date servers are either patched or decommissioned. In the current configuration, if a single machine is compromised, an attacker can masquerade as the entire domain. TOB-VOATZ-028

❑ **Switch to a server-side anti-tamper check that whitelists clients.** For example, ensure that the client *both* was not tagged as a threat by Zimperium *and* attested to Zimperium in the first place. This, like all anti-tamper protections, is not foolproof. However, it will at least require an attacker to perform the additional step of spoofing a valid Zimperium attestation rather than simply allowing them to gain access once Zimperium is bypassed. TOB-VOATZ-029

❑ **Ensure that only known `threatId`s are stored in the database.** A list of known threats is already tracked by the Voatz backend to determine if a user should be blocked from authentication. If, instead, Voatz wishes to store all threats so a new `threatId` could be added to the list in the future *and* have any previously reported threats ban the user retroactively, then additional data validation should be performed on these fields. TOB-VOATZ-030

❑ **Protect all sensitive windows within the App by enabling the `FLAG_SECURE` flag.** This will prevent malicious third-party apps from recording the Voatz app. This also prevents screenshots of sensitive information. TOB-VOATZ-032

❑ **Update all Voatz servers and mobile clients to enable support for OCSP Stapling.** This will prevent attacks in the event of an SSL certificate revocation. TOB-VOATZ-033

❑ **Ensure that the Android Security Provider is up-to-date.** On every app startup, run `ProviderInstaller.installIfNeeded()` provided by Google Play services. If the Security Provider remains out of date or an error occurs, this method will throw an exception and Voatz can decline to run. TOB-VOATZ-034

❑ **Prevent all WebViews from reading from internal storage.** Explicitly set the `setAllowFileAccess` method to `false`. TOB-VOATZ-035

❑ **Explicitly prevent all WebViews from executing JavaScript.** Set the `setJavaScriptEnabled` method to `false`. TOB-VOATZ-036

❑ **Use the Google SafetyNet Attestation API.** This will supplement Zimperium in assessing the integrity and safety of the user's device. TOB-VOATZ-037

❑ **Mitigate iOS object substitution attacks.** Migrate all classes that use `NSCoding` to `NSSecureCoding`. TOB-VOATZ-038

❑ **Mitigate iOS URI scheme hijacking attacks.** Confirm that the `voatz://` URI scheme is not used for messaging, and document the code to ensure that it never will be. TOB-VOATZ-039

❑ **Disable third-party keyboards within the Voatz iOS client.** This will help prevent leaking of sensitive data entered by the user. Add the `application:shouldAllowExtensionPointIdentifier:` method within the Voatz client's `UIApplicationDelegate` TOB-VOATZ-040

❑ **Use the Secure Enclave when performing any cryptographic operation on an iOS device.** This avoids revealing sensitive credentials in memory to the application processor. TOB-VOATZ-041

❑ **Use iOS-managed login input fields.** Use the `UITextContentType` property introduced in iOS 12 to identify username and password fields. This will allow automated password generation and management. TOB-VOATZ-042

❑ **Ensure that sensitive keychain items are not stored in iCloud and iTunes backups.** Explicitly set a `ThisDeviceOnly` accessibility class (such as `kSecAttrAccessibleWhenUnlockedThisDeviceOnly`) for all keychain items. This will prevent both Apple, Inc. and attackers from accessing sensitive voter information. TOB-VOATZ-043

❑ **Precisely define the exceptions to ATS that are required.** Every required exception reduces the security of the Voatz app. Minimize your exposure by providing the narrowest possible exceptions. TOB-VOATZ-044

❑ **Use the SafetyNet Attestation API to assess the integrity and safety of the user's device.** Identify and address any server-side issues that require ATS exceptions. Configure the Attestation API to use the `basicIntegrity` parameter to support devices that have not passed CTS certification. TOB-VOATZ-037

❑ **Use the SafetyNet Verify Apps API to ensure this feature is enabled and that harmful apps are not installed on user devices.** This supplements malicious app detection by Zimperium. TOB-VOATZ-045

❑ **Validate that the audit tokens provided in client requests are associated with the voter's client ID.** This will prevent voters from using duplicate audit tokens and help protect the integrity of the election. TOB-VOATZ-046

❑ **Remove all test code from production.** Ensure that clients can neither accidentally nor intentionally trigger test code. TOB-VOATZ-047

❑ **Encrypt the salt for user PINs and check their device encryption status.** Further protecting the salt and ensuring whole-disk encryption will help mitigate data exposure risks of the encrypted database on Android clients. TOB-VOATZ-048

Long Term

❑ **Integrate a tool like truffleHog into your git hooks.** This will help prevent sensitive information from being committed to the repository. TOB-VOATZ-001, 002, 016, and 017

❑ **Avoid designing any kind of transport encryption.** Use standardized and integrated frameworks such as Wireguard or TLS 1.3. TOB-VOATZ-004

❑ **Review the use of hardcoded literals throughout the codebase.** Ensure that significant variables do not make use of repetitively hardcoded literals, but instead derive from well-defined constants, configuration-based variables, or otherwise uniform definitions. TOB-VOATZ-005

❑ **Ensure that no encryption uses hardcoded credentials.** Use of AES-CBC should be deprecated in favor of AES-GCM or another AEAD construction. TOB-VOATZ-006

❑ **Review all unique identifiers for users.** Ensure that the identifiers cannot collide with one another. Ensure users are made well aware of cases in which these identifiers could change and affect their voting experience. TOB-VOATZ-008

❑ **Provide a more robust means for delivering non-anonymous election receipts to voters.** For example, return *multiple* QR codes, if necessary. Alternatively, deprecate this feature and remove all code supporting non-anonymous elections. TOB-VOATZ-009

❑ **Revise the onboarding workflow to provide an election identifier from the very first step.** This should allow provisioning of a completely independent backend cloud infrastructure for each election. TOB-VOATZ-010

❑ **Add a cryptographic analyzer to Voatz' continuous integration process.** An analyzer like Cryptosense can automatically detect the use of insecure algorithms. TOB-VOATZ-011

❑ **Carefully audit all cryptographic primitives.** Check that their use conforms to their specification. TOB-VOATZ-012

❑ **Ensure that no secrets are stored in either code or environment variables.** These methods are prone to leakage. The use of a secret manager is preferable. TOB-VOATZ-013

❑ **Review data validation surrounding requests on the Voatz backend.** Ensure a user cannot submit requests for any accounts/devices but their own, and that malformed data cannot be provided to the server to invoke resource exhaustion. TOB-VOATZ-014, 046, 047, and Appendix C

❑ **Carefully audit all symmetric encryption used in the Voatz system.** Check for known vulnerabilities. TOB-VOATZ-015

❑ **Improve unit test coverage to test idle timeouts.** TOB-VOATZ-018

❑ **Review all API request fields to ensure sufficient data validation is performed.** In cases where malformed data is provided, consider the strength of the evidence as an indicator that the user is malicious and should be blocked. TOB-VOATZ-019

❑ **Transition to unique identifiers that are assigned by the backend.** The clients should not be able to specify their own unique IDs. TOB-VOATZ-020

❑ **Transition to a different method for archiving signed affidavits that is not prone to denial of service.** Email is both hard to secure and prone to various forms of denial of service. TOB-VOATZ-021

❑ **Improve unit test coverage to test all edge cases relating to data provided in the API request.** In general, it should never be assumed that a client is not maliciously modified. TOB-VOATZ-022

❑ **Perform a comprehensive audit of the log messages used within the system.** Ensure that they do not contain any sensitive information. TOB-VOATZ-023

❑ **Standardize all symmetric encryption in the Voatz system.** Use a single AEAD construction. Remove any use of symmetric primitives other than this construction. TOB-VOATZ-024

❑ **Replace PBKDF2.** Other key derivation functions, such as Argon2id and scrypt, are stronger and more difficult to misconfigure. TOB-VOATZ-025

❑ **Audit all network calls made by the application.** Maintain a list of domains accessed. For each domain, ensure calls are only made using TLS with certificate pinning. TOB-VOATZ-026

❑ **Store all ballot oval states in Hyperledger.** This will prevent confusion caused by any empty ballots that are cast, and will provide a richer audit trail on the blockchain. TOB-VOATZ-027

❑ **Ensure that the security of Voatz is not predicated on the authenticity of the Voter's client.** There is no foolproof way to ensure that a client communicating with the Voatz backend is authentic or has not been tampered with. TOB-VOATZ-029

❑ **Review all requests to ensure proper data validation is performed.** Fields should be limited in length to prevent resource exhaustion attacks. Ensuring proper form will also prevent attacks from incorrect handling of malformed data. TOB-VOATZ-030 and 031

❑ **Ensure that developer documentation is updated to include screen capture and recording as potential threats for data exposure.** TOB-VOATZ-032

❑ **Perform certificate revocation exercises.** This will ensure the system's protections are sufficient and train the Voatz staff on how to react to a compromised SSL credential. TOB-VOATZ-033

❑ **Continue adding anti-tamper and security update protections to the Voatz clients.** TOB-VOATZ-034 and 037

❑ **Add tests to ensure malicious websites cannot read from the Android client's internal storage.** This is currently possible via WebViews. TOB-VOATZ-035

❑ **Add tests to ensure malicious websites cannot execute arbitrary JavaScript within the Android client.** This is *not* currently possible, but if the default behavior of WebViews ever changes, it could become possible. TOB-VOATZ-036

❑ **Require an affirmative `ctsProfileMatch` result which indicates that the user is in possession of a device that passed CTS certification.** Devices without a CTS certification possess unknown security risks and are likely to be compromised. TOB-VOATZ-037

❑ **Transition from iOS URI Schemes to the newer Universal Links.** "Universal Links,"introduced in iOS 9, allows apps to register web domains that are solely owned by the app. TOB-VOATZ-039

❑ **Stay abreast of changes to iOS that might permit data exfiltration from the Voatz client.** TOB-VOATZ-040

❑ **Stay abreast of new cryptographic features added to the iOS SDK.** TOB-VOATZ-041 and 042

❑ **Empirically validate that no sensitive data is stored to a backup of the Voatz iOS application.** Consider uniform usage of a wrapper, such as Square's Valet, to simplify storing and retrieving data from the keychain. TOB-VOATZ-043

❑ **Require network encryption that meets the minimum standards of ATS.** Identify and address any server-side issues that require ATS exceptions. TOB-VOATZ-044

❑ **Do not require clients to submit data like audit tokens in requests.** Instead, perform a database lookup of their data. TOB-VOATZ-046

❑ **Replace the local data encryption with a system based on the Android StrongBox.** This will ensure that local encryption is tied to the user's device and conducted with the strongest methods available. TOB-VOATZ-048

Findings Summary

#	Title	Type	Severity
1	Device IDs not validated against inner request device IDs	Data Validation	High
2	Amazon admin password is hardcoded in source file	Data Exposure	High
3	Non-anonymous ballot receipts are encrypted with AES-CBC using hardcoded key and IV	Cryptography	High
4	Secrets are stored in environment variables sourced from bash script	Data Exposure	High
5	API for the onboarding workflow prohibits partitioning cloud resources for concurrent elections	Configuration	High
6	Receipt and affidavit filename collisions	Data Validation	High
7	A voter can unregister another voter's device	Access Controls	High
8	Input keying material for AES GCM encoding is sent to Graylog	Data Exposure	High
9	Voatz backend SSL key has a subdomain wildcard	Configuration	High
10	Clients can specify their own audit token	Data Validation	High
11	Test parameters in the registration APIs can bypass SMS verification	Data Validation	High
12	QR code receipt generation will fail for large non-anonymous ballots	Data Validation	Medium
13	Session token validation ignores idle timeout	Session Management	Medium

14	Receipt encryption is weak and can leak confidential information	Cryptography	Medium
15	Insufficient device ID validation on backend	Data Validation	Medium
16	Resource exhaustion via logging/storage of unsanitized data	Denial of Service	Medium
17	Potential resource exhaustion via specially crafted Zimperium threats	Denial of Service	Medium
18	Zimperium checks on the backend are a blacklist, not a whitelist	Access Controls	Medium
19	AES-GCM key/nonce/tag encryption system breaks authenticity	Access Controls	Medium
20	Unauthenticated ECDH is vulnerable to key compromise impersonation	Cryptography	Medium
21	AES-GCM keys, nonces, and “tag”s are encrypted using AES-ECB	Cryptography	Medium
22	Voatz API server lacks OCSP stapling	Cryptography	Medium
23	Empty ballots are not recorded in Hyperledger	Data Validation	Low
24	Database root credentials stored in git	Data Exposure	Undetermined
25	Signed voter affidavits are sent to an administrative email	Data Exposure	Undetermined
26	AES-GCM AAD usage is non-standard	Cryptography	Undetermined
27	Session cookie expiration offset is a hardcoded literal	Configuration	Informational
28	Encrypted application data is trivially brute-forceable	Cryptography	High

29	PBDKF2 provides insufficient security margin for PIN codes	Cryptography	High
30	Third-party apps can capture the Android client screen and read screenshots taken from the client	Data Exposure	High
31	Android release build signing key password and keystore password stored in git	Data Exposure	High
32	A malicious website can read from the Android client's internal storage	Data Exposure	High
33	Insufficient Android device ID construction	Session Management	Low
34	Android client does not use the SafetyNet Attestation API	Configuration	Low
35	Android client does not use the SafetyNet Verify Apps API	Configuration	Low
36	Certificate pinning is only configured for the main Voatz domain	Cryptography	Low
37	No explicit verification of the Android Security Provider	Patching	Low
38	Jumio Netverify API credentials stored in git	Data Exposure	Undetermined
39	Google Services API key stored in git	Data Exposure	Undetermined
40	A malicious website may be able to execute JavaScript within the Android client	Access Controls	Informational
41	The iOS client does not disable custom keyboards	Data Exposure	Medium
42	The iOS client does not use system-managed login input fields	Configuration	Low

43	iOS client keychain items are not excluded from iCloud and iTunes backups	Data Exposure	Low
44	Cryptographic credentials are not generated in the iOS Secure Enclave	Cryptography	Low
45	iOS client disables Apple Transport Security (ATS)	Cryptography	Low
46	iOS client is vulnerable to object substitution attacks	Data Validation	Undetermined
47	An iOS user can lose their registration	Session Management	Informational
48	iOS client is susceptible to URI scheme hijacking	Data Validation	Informational

Backend Findings

1. Device IDs not validated against inner request device IDs

Severity: High
Type: Data Validation
Target: `<various API endpoint handlers>`

Difficulty: Low
Finding ID: TOB-VOATZ-014

Description
There is insufficient data validation when processing encrypted requests on the backend server. This allows a user to send requests with another user's device ID, bypassing protections from the per-device shared secret encryption scheme.

The structure of an encrypted API request generally has two fields: `deviceId` and `request`. The device ID is used to look up a per-device encryption key to secure communications between the client and the server, and to decrypt the underlying `request` data.

```
case class ApiEncryptedThreatDetectedRequest(deviceId: String, request: String)
```

Figure TOB-VOATZ-014.1: The structure of an encrypted request used to report client-side threats detected on the device (`customer.scala#L194`).

Often, the underlying request data will also have a `deviceId` field.

```
 case class ApiThreatDetectedRequest(deviceId: String, customerId: Option[Int], threatId:
String, threatName: Option[String], threatSummary: Option[String], threatType:
Option[String],
   threatSeverity: Option[String])
```

Figure TOB-VOATZ-014.2: The underlying request to report client-side threats detected on the device (`customer.scala#L192–L193`).

In many cases, Voatz fails to validate that the device ID in the underlying request matches the device ID used to encrypt the request. This means an attacker can use their own device ID to encrypt requests which are actually targeting another user/device.

The consequences of this issue vary per operation. In the example above, a user could create an `ApiThreatDetectedRequest` to blacklist another device ID from authentication by using their own device ID to encrypt the outer `ApiEncryptedThreatDetectedRequest`.

In other cases, such as an `ApiEncryptedCustomerAuthenticateRequest`, the Voatz backend will validate the session cookie against the outer device ID used for encryption, but use the device ID in the inner request to perform an action. In this case, a user can authenticate to their account while forcing the Voatz backend to pass an invalid device ID to the `makeTransactionRequest` function call. The same consequence can be observed during handling of the `ApiEncryptedCustomerLogoutRequest`.

Appendix C contains a complete list of the encrypted API requests that are affected by similar data validation issues.

Exploit Scenario
Bob is using Voatz to vote in a local election. Alice, an attacker on his network who also uses Voatz to vote locally, wants to blacklist him from voting. Alice sends an `ApiEncryptedThreatDetectedRequest` to the Voatz backend, providing Bob's device ID in the inner request, but encrypting the request with her own device ID. The Voatz backend will accept this request and blacklist Bob's device from future authentication.

Recommendation
Short term, refactor the handling of encrypted requests to ensure the device ID used to index the shared secret for encryption is validated against the device ID in the underlying request. Alternatively, consider removing device ID from the inner request to preserve bandwidth and alleviate any device ID conflicts.

Long term, review data validation surrounding requests on the Voatz backend to ensure a user cannot submit requests for any accounts/devices but their own, and that malformed data cannot be provided to the server to invoke resource exhaustion.

2. Amazon admin password is hardcoded in source file

Severity: High
Type: Data Exposure
Target: `AmazonTestOtpUtility.scala`

Difficulty: Low
Finding ID: TOB-VOATZ-017

Description
Amazon credentials for an account with the username “admin” can be found in the Scala source tree. This allows anyone with source access to control the entire deployment environment of the Voatz application. The password also has insufficient entropy.

Exploit Scenario
An attacker gains source access to the Voatz codebase, discovers these credentials, and stealthily installs rootkits on all deployed Voatz servers. They can then observe any individual's voting patterns and tamper with the actual votes cast.

Recommendation
Short term, conduct a thorough investigation into whether this key could be misused. Remove it from the source file and rotate it. Replace it with a stronger password generated by a password manager.

Long term, implement the kind of secret management process described in TOB-VOATZ-012, and audit all new code for secrets.

3. Non-Anonymous ballot receipts are encrypted with AES-CBC using hardcoded key and IV

Severity: High
Type: Cryptography
Target: `CustomerVoteInfoAsync.scala`

Difficulty: Low
Finding ID: TOB-VOATZ-006

Description
In order to give voters a receipt of their voting choices, Voatz takes their voting data, encrypts it with AES in CBC mode, creates a QR code of this data, and uploads it to Amazon S3. Voatz then shares a link to this QR code with the voter, who decrypts the information contained. Notably, the URLs are highly predictable, and could allow for trivial enumeration.

This encryption uses a hard-coded static key and IV found in the core server code as well as the Android and iOS application code. This means that anyone capable of downloading a Voatz mobile application can decrypt these receipts and see the highly confidential voting data contained. It appears the particular key and IV used are also copied from a Stack Overflow answer.

Note that QR code receipts are currently only generated if the event name is “ElectionNA” or “ElectionNAMulti” (*i.e.*, a non-anonymous election). Voatz claims that non-anonymous ballot elections have not been used since 2018 and all code to support them will be removed from the system.

Exploit Scenario
During an election, an attacker extracts the static key and IV from a Voatz Android application, then crawls the S3 bucket used for voting receipts. They then have access to all voting data from that election. Moreover, voters’ signatures are uploaded to the same S3 bucket, allowing time-based correlation between voters’ names and their ballots.

Recommendation
Immediately conduct a thorough investigation of whether past voting data is accessible via this vector, and take down any data found. Send receipts only over ephemeral channels with a unique channel key determined dynamically.

Long term, ensure no encryption uses hardcoded credentials. AES-CBC should be deprecated in favor of AES-GCM or another AEAD construction.

4. Secrets are stored in environment variables sourced from bash script

Severity: High
Type: Data Exposure
Target: `startup.sh`

Difficulty: Medium
Finding ID: TOB-VOATZ-013

Description
Many of Voatz core services' secrets (but not all) are loaded as environment variables at startup. Specifically, an (optionally) AES-CBC encrypted file containing bash source to assign a number of variables is (optionally) decrypted and then sourced in the startup script. The variables in the first file are used to assemble a second bash script, which is then also executed. The second bash script should then be deleted.

This loading process has several issues. First, any modification of the persistent file trivially leads to code injection. AES-CBC has no anti-malleability guarantees, and cannot prevent this issue. Secondly, should the application shut down unexpectedly, it is possible that secrets will remain in another plaintext file. The standard shutdown process (as documented in the provided shutdown script) apparently seems to be running `kill -9`, so this is certainly a possibility.

Additionally, even if the persistent file is always well-behaved, keeping secrets in environment variables is a well-known antipattern. Environment variables are commonly captured by all manner of debugging and logging information, can be accessed from procfs, and are passed down to all child processes. Environment variables are in no way an acceptable substitute for a real secret-management system such as Vault or Keywhiz.

Exploit Scenarios
- An attacker with local disk access modifies the encrypted secrets file to gain arbitrary code execution on the Voatz server.
- A logging service transports environment variables over HTTP, allowing anyone with local access to read Voatz secrets.
- An improper shutdown or server misconfiguration leaves secrets unencrypted on the Voatz server, allowing any attacker who can read files to access all keys.

Recommendation
Short term, switch to a purpose-built secret-management system such as Vault or Keywhiz. Rotate all credentials stored with the previous system.

Long term, ensure that no secrets are stored in either code or environment variables, and use the secret manager instead.

5. API for the onboarding workflow prohibits partitioning cloud resources for concurrent elections

Severity: High
Type: Configuration
Target: `Cryptography.scala`

Difficulty: High
Finding ID: TOB-VOATZ-010

Description
The first API call a mobile client makes after the cryptographic handshake registers it with an email address, phone number, and device ID using the customer's/pre-registered REST endpoint. This call does not include any information about the election for which the customer is pre-registering, and therefore must be routed to a single, centralized core server. Backend infrastructure (S3, Mongo, Hyperledger, *etc.*) *could be* duplicated for each election, but there do not appear to be provisions for that in the backend code. Therefore, a unique Voatz client must be built and distributed for each election, preconfigured with the backend instance specific to that election. This is why voters are first taken to a landing page for each election, where they are directed to download an election-specific client through Apple TestFlight or Google Play Beta.

Voatz has indicated that it plans to support multiple elections from a single app distributed through the public iOS and Android app stores. However, this will necessitate changes to the underlying protocol.

Exploit Scenario
An attacker compromises an election's landing page, changing the download link to install a modified version of the client created by the attacker.

Recommendation
Short term, ensure that all backend infrastructure is re-provisioned between elections and not shared between elections/clients. Use infrastructure-as-code tools like Ansible and Terraform to automate and manage provisioning.

Long term, revise the onboarding workflow to provide an election identifier from the very first step so a completely independent backend cloud infrastructure can be provisioned for each election.

6. Receipt and affidavit filename collisions

Severity: High
Type: Data Validation
Target: `ReceiptHelper.scala`

Difficulty: High
Finding ID: TOB-VOATZ-020

Description
The Core Server deterministically chooses a filename for receipts and affidavits before they are uploaded to S3. These filenames are identified based upon the voter's anonymous IDs (audit tokens). In constructing the filename, they are sanitized by removing special characters:

```
val deviceIdEsc = Pattern.compile(
    "[\\{\\}\\(\\)\\[\\]\\.\\+\\*\\?\\^\\$\\\\\\\\|\\=\\/\\@\\;]")
    .matcher(deviceId).replaceAll("")
```

Figure TOB-VOATZ-020.1: Escaping special characters in the receipt and affidavit filename generation (`ReceiptHelper.scala#L16`).

If two anonymous IDs are the same except for the existence of special characters, the receipt filenames uploaded to S3 will conflict.

Exploit Scenario
Alice modifies her mobile app or writes her own custom client (see Appendix B, Claim 3). The bespoke client registers for an election using the same audit token as Bob, plus a reserved character (*e.g.*, by exploiting a vulnerability like TOB-VOATZ-046). Alice's affidavit and receipt will overwrite Bob's.

Note that the S3 bucket policy can affect the result of this exploit. For example, the bucket might have versioning enabled, in which case Bob's files will not be permanently overwritten. We were not given access to any of the production S3 bucket policies, so we cannot assess this.

Recommendation
Short term, move all device ID sanitization and escaping as early as possible in the onboarding process, when a device ID is first set. Perform a check to ensure that the sanitized ID does not collide with a preexisting ID associated with any other voter.

Long term, transition to unique identifiers that are assigned by the backend, not by the clients.

7. A voter can unregister another voter's device

Severity: High
Type: Access Controls
Target: `CustomerMongoDaoAsync.scala`

Difficulty: High
Finding ID: TOB-VOATZ-022

Description

The re-registration workflow is designed to address the situation in which a voter's device ID changes (*e.g.*, due to a lost device). Unlike most other post-onboarding API endpoints that use the voter's device ID as a unique identifier, re-registration instead relies on the voter's mobile phone number, since a change in the device ID likely prompted the re-registration in the first place. The last step in the re-registration process calls the `reRegisterAsFuture` function, which receives the mobile phone number from the client as an HTTP parameter. Another optional HTTP parameter provided by the client, `useSideChannel`, can be used to disable the email account recovery validation.

```
val useSideChannel = request.useSideChannel.getOrElse(false)
val (sideChannelOk, sideChannelErr, sideChannelEmail) = if(useSideChannel){
  ⋮ // Elided E-mail verification code that can be avoided
}else{
  (true, "", None)
}

if(maxReqExceededByMob){
  ⋮ // Elided error handling code
}else if(maxReqExceededByDevice){
  ⋮ // Elided error handling code
}else if(maxReqExceededByIp){
  ⋮ // Elided error handling code
}else if(!sideChannelOk){
  ⋮ // Elided error handling code
}else{
  val preRegResult = await(CustomerPreRegisterAsync.preRegister(request.mobileNumber,
request.emailAddress.getOrElse(""), request.deviceId, ipAddress,
OffsetDateTime.now().toString()))
  CustomerPreRegisterReadAsync.create(preRegResult)
  (preRegResult, None, sideChannelEmail, None)
}
```

Figure TOB-VOATZ-022.1: Reliance on the re-registration request parameters to use email validation and to specify the mobile number (`CustomerMongoDaoAsync.scala#L1224–L1277`*).*

Exploit Scenario
Alice and Bob both register with Voatz on their respective devices. Alice, using a custom or modified client, sends a re-registration API request using her own information, but with Bob's mobile number and the `useSideChannel` flag set to `false`. Bob's registration will be reset and he will no longer be able to log in from his device.

Recommendation
Short term, ensure that a second factor of authentication is always necessary for re-registration.

Long term, refactor the re-registration workflow so it does not rely on any information provided by the client. Improve unit test coverage to test all edge cases relating to data provided in the API request. In general, it should never be assumed that a client is not maliciously modified.

8. Input keying material for AES GCM encoding is sent to Graylog

Severity: High
Type: Data Exposure
Target: `Aes128GcmEncoding.scala`

Difficulty: High
Finding ID: TOB-VOATZ-023

Description
On line 103 of `Aes128GcmEncoding.scala`, the input keying material for AES is sent to Graylog at the `trace` level. We were not given the configuration files necessary to determine whether logging would be configured to expose this keying material on a production instance. The cryptographic key is used for securing requests to and from the receipt service.

Exploit Scenario
An attacker gains read access to Graylog and is able to compromise all AES GCM encrypted requests to and responses from the receipt service. This can enable a passive attacker with network access to de-anonymize ballot receipts.

Recommendation
Short term, remove the log message. Ensure that production instances are configured to run at the highest log level for which the necessary auditing information is still recorded.

Long term, perform a comprehensive audit of the log messages used within the system to ensure they do not contain any sensitive information.

9. Voatz backend SSL key has a subdomain wildcard

Severity: High | Difficulty: High
Type: Configuration | Finding ID: TOB-VOATZ-028
Target: Voatz backend servers hosted in the `nimsim.com` domain

Description
The Voatz backend servers are hosted on the `nimsim.com` domain. All servers hosted at this domain appear to use the same SSL certificate with a wildcard matching any subdomain: `*.nimsim.com`. This implies that each server has a duplicate copy of the domain's SSL private key. Therefore, the private key to the entire Voatz backend domain—to which the clients are pinned (see TOB-VOATZ-026)—is only as secure as the weakest server on the domain. Voatz server provisioning and management are currently performed manually, so it is likely that some infrastructure will be left unpatched over time.

Exploit Scenario
A Voatz development server is not decommissioned, and/or is left unpatched. Alice exploits the server and gains access to the private key for `*.nimsim.com`, allowing her to passively snoop on and decrypt encrypted network traffic, man-in-the-middle connections, and potentially instantiate her own backend server masquerading as a legitimate Voatz server.

Recommendation
Short term, use different SSL credentials for each `nimsim.com` subdomain. Ensure that out-of-date servers are either patched or decommissioned.

Long term, transition to automated infrastructure provisioning and management, *e.g.*, with infrastructure-as-code tools like Ansible and Terraform. Let's Encrypt, via an ACME-compliant certificate provisioning tool like certbot, can automate the acquisition and distribution of TLS certificates on Voatz infrastructure.

References
- Certbot instructions for CentOS 6 and Apache HTTPD

10. Clients can specify their own audit token

Severity: High
Type: Data Validation
Target: `CustomerApiWorkerAsync.scala`

Difficulty: High
Finding ID: TOB-VOATZ-046

Description
Audit tokens are generated in the Voatz core server and sent to the client. The API request for submitting a vote accepts the voter's audit token from the client without validation. There is a check to ensure that the submitted audit token is valid; however, there is no check to ensure that the audit token is associated with the voter. Therefore, an attacker with access to another voter's audit token can cast their vote with the same token, causing confusion during auditing and calling the integrity of the entire election into question.

Appendix C provides more instances of API endpoints that lack sufficient validation and may result in similar exploits.

We did not have access to the audit portal source code, so it is unclear how it might react to an audit token collision. Possibly it would cause the same Hyperledger data to be associated with each ballot receipt, or it might cause one of the votes to disappear. This vulnerability may also result in a denial of service on the audit portal, since a malicious client could submit an arbitrarily large audit token.

Exploit Scenario
Alice knows Bob's audit token (*e.g.*, by observing his ballot receipt or colluding with him). She uses a modified client to submit her vote with the same audit token as Bob. During an audit, Eve notices that Alice and Bob's votes both use the same token, calling the integrity of the election into question.

Recommendation
Short term, validate that the audit tokens provided in client requests are associated with the voter's client ID.

Long term, do not require clients to submit data like audit tokens in requests. Instead, perform a database lookup of their data.

11. Test parameters in the registration APIs can bypass SMS verification

Severity: High
Type: Data Validation
Target: `CustomerApiWorkerAsync.scala`

Difficulty: High
Finding ID: TOB-VOATZ-047

Description
Both the `preRegisterCustomer` and `reRegisterCustomer` API endpoints accept an optional "test" HTTP parameter. When included in a registration request, it bypasses the normal SMS verification code that uses Twilio and instead uses Amazon OTP. The code suggests that this is an experimental feature of Voatz.

In the version of the codebase assessed during this engagement, the credentials for interacting with the Amazon OTP server are hard-coded into `AmazonTestOtpUtility.scala`. However, on February 21 in git commit 5f5938a, the hard-coded credentials were removed (although not rebased from the git history) and instead read from MongoDB using `getAmazonTestOtpSettings`. The logic inside of `reRegisterCustomer` was also changed such that if the client sets the `test` parameter to `true` *and* the MongoDB instance does *not* include any Amazon OTP settings (as is likely to be the case in production), then no SMS verification will be sent.

Appendix C provides more instances of API endpoints that lack sufficient validation and may result in similar exploits.

Exploit Scenario
Alice knows Bob's mobile number, which she uses to initiate the re-registration process (see TOB-VOATZ-007). Using a modified client, she sets the `test` parameter to `true`, preventing the SMS verification message from being sent to Bob. If Alice has access to Bob's email and also knows Bob's previous OTP verification code, she can gain control of his account.

Recommendation
Short term, remove the test code from production. Rebase the git repository to remove API secrets from the history.

Long term, revise the API to remove any reliance on inputs provided by clients.

12. QR code receipt generation will fail for large non-anonymous ballots

Severity: Medium
Type: Data Validation
Target: `CustomerVoteInfoAsync.scala`

Difficulty: Low
Finding ID: TOB-VOATZ-009

Description
Voting information is encrypted (see TOB-VOATZ-006) and encoded in a QR code before being emailed to the voter and uploaded to an Amazon S3 bucket. The message encoded in the QR code is:

```
"For your records, here is how you voted on MMMM dd, yyyy during the
$eventName event:|$voteData"
```

This is encrypted with AES256-CBC using PKCS7 padding and an IV, resulting in a ciphertext length of:

$$\left(\frac{len(plaintext)}{16} + 2\right) \cdot 16 \text{ bytes.}$$

Another two bytes ("+9") are finally prepended to the ciphertext before it is rendered as a QR code. The QR Code is generated using the *Zebra Crossing* (ZXing) library, which defaults to ECC level "L." At this level, a maximum-sized QR Code (177 by 177) can store at most 2,953 bytes. Therefore, the following invariant must hold:

$$\left(\frac{76 + len(\$eventName) + len(\$voteData)}{16} + 2\right) \cdot 16 + 2 \leq 2953.$$

A QR code will not be generated if the length of the event name plus the length of the vote data is greater than 2,843 bytes.

Note that QR code receipts are only generated if the event name is "ElectionNA" or "ElectionNAMulti" (*i.e.*, non-anonymous ballot elections). Voatz claims that non-anonymous ballot elections have not been used since 2018 and all code to support them will be removed from the system.

Exploit Scenario
Alice submits a vote for an election with a very large ballot (*e.g.*, a simultaneous federal, state, and municipal election). She enters write-ins for each candidate, resulting in over ~2KiB of ballot data. The QR Code generation for her ballot will fail, she will not receive a QR code email receipt, and the QR Code will not be uploaded to S3.

Recommendation
Short term, log any such QR code generation failure events, and alert the voter.

Long term, provide a more robust means for delivering receipts to voters (*e.g.*, by returning multiple QR Codes, if necessary).

13. Session token validation ignores idle timeout

Severity: Medium
Type: Session Management
Target: `SessionAuthenticatorAsync.scala`

Difficulty: Low
Finding ID: TOB-VOATZ-018

Description
The code to reject authentication when a session token has exceeded its idle timeout is commented out. In fact, the `MaxIdleTime` variable is unused in the code, despite appearing in the configuration.

This finding is partially remediated in practice, since session information is kept in Memcached, which will delete the session on logout. (This happens on idle timeout and when the app transitions from the foreground to the background.) However, this requires the client to initiate an API call to `logout`; if the client crashes or the device loses Internet connectivity, the session will stay active since there is no equivalent timeout on the backend.

```
if (csrfToken != csrfTokenOpt.get) {
  Left(AuthenticationFailedRejection(CredentialsRejected, List()))
}
/*
else if(idleTime > config.HttpConfig.Authentication.MaxIdleTime){
 Left(AuthenticationFailedRejection(CredentialsRejected, List()))
}
*/
else {
 Right(HttpApiSession(sessionCookie, csrfToken, customerId, lastUse))
}
```

Figure TOB-VOATZ-018.1: Commented-out token timeout check (`SessionAuthenticatorAsync.scala#L115–L125`).

Exploit Scenario
Alice gains access to Bob's session token and credentials (*e.g.,* with physical access to Bob's phone, and by terminating its network connection before it can call logout). She can then wait an arbitrary amount of time for an election to start and she will still be able to vote on his behalf.

Recommendation
Short term, re-enable the timeout check.

Long term, improve unit test coverage to address these scenarios.

14. Receipt encryption is weak and can leak confidential information

Severity: Medium
Type: Cryptography
Target: `ReceiptBuilder.scala`

Difficulty: Low
Finding ID: TOB-VOATZ-015

Description
Receipt PDFs are encrypted using standard PDF encryption in AES 128 mode, which is effectively AES-128 in CBC mode with some limited usage of AES-128 in ECB mode. This encryption is quite poorly designed; almost all PDF readers allow an attacker with the ability to modify PDF receipts to leak plaintext information. Additionally, the length of the encrypted PDF can be used as a side-channel to leak information about the plaintext without requiring modification.

Müller, *et al.*'s 2019 work *Practical Decryption exFiltration: Breaking PDF Encryption* details methods for recovering the entire plaintext of encrypted PDFs in most PDF reader software. They also detail exfiltration methods for extracting the plaintext once it has been recovered. While many PDF readers found to be vulnerable have been patched, there are still many unpatched systems, and many PDF readers have never been assessed for this vulnerability.

Exploit Scenario
An attacker gains access to modify an encrypted PDF receipt or receipts. They quickly decrypt all sensitive information contained without needing to know the password.

Recommendation
Short term, do not use PDF encryption. Replace the use of PDF encryption with the age tool.

Long term, carefully audit all symmetric encryption used in the Voatz system for known vulnerabilities.

References
- Practical Decryption exFiltration: Breaking PDF Encryption

15. Insufficient device ID validation on backend

Severity: Medium | Difficulty: Medium
Type: Data Validation | Finding ID: TOB-VOATZ-019
Target: `CustomerValidationsAsync.scala, OrganizationMongoDaoAsync.scala`

Description
Device IDs are constructed on the client side in one of two ways, depending on whether the platform is iOS or Android. There is insufficient validation of the device ID format on the backend which allows clients to bypass Android licensing and device model checks, report as using neither an iOS or Android device, and unnecessarily forwards the burden of data validation onto external services such as the administrative web interface and audit portal, which may reflect this field.

There are three instances of device ID checks on the backend. The first instance has been commented out:

```
    }/*else if((deviceIdFromReq.startsWith("and-") || !deviceIdFromReq.startsWith("ios-"))
&& !androidWhitelistOpt.isDefined){
      (false, s"Sorry, your device currently does not have access to the Voatz platform.
Thanks for your interest, we will be in touch as soon as your device can be granted
access.", None)}*/
```

Figure TOB-VOATZ-019.1: Commented-out device ID validation (`CustomerValidationsAsync.scala#L111–L112`).

During AndroidLvl checks, the device ID is checked to see if the device is an Android device. If the device ID does not start with "`and-`" (case-insensitive), these checks will be skipped. This check is insufficient because it does not ensure the format of the device ID past this prepended prefix, nor is it case-sensitive, allowing device IDs such as "`aNd-`" or "`AnD-FakeDeviceId`," neither of which should ever be constructed by the client:

```
 private def isAndroidLvlCheckOk(deviceIdFromReq: String, androidLvlNonceOpt:
Option[Int])(implicit db: MongoDatabase) = async{
   if (deviceIdFromReq.matches("(?i)^and-.*")) {
```

Figure TOB-VOATZ-019.2: Insufficient device ID validation during `AndroidLvl` checks (`CustomerValidationsAsync.scala#L157–L158`).

In the final instance of device ID format validation when checking a device model, the device ID is checked for the prefix "`and-`" (case-sensitive). If this prefix is not found, the device model checks will be skipped. This means an attacker can prevent additional checks with a specially crafted device ID:

```
 def checkDeviceModel(organizationSnap: OrganizationSnapshot, deviceProfileOpt:
Option[DeviceProfile])(implicit db: MongoDatabase): Future[(Boolean, String)] = async{
   if(organizationSnap.deviceModelCheck.getOrElse(false)){
     if(deviceProfileOpt.isDefined){
       if(deviceProfileOpt.get.deviceId.isDefined){
```

```
        if(deviceProfileOpt.get.deviceId.get.startsWith("and-")){
```

Figure TOB-VOATZ-019.3: Insufficient device ID validation during organization device model checks (`OrganizationMongoDaoAsync.scala#L428–L432`*).*

The device ID is a client-constructed identifier which should follow an explicit format. The backend should validate the device ID is of good form, or block the user.

The use of a device ID has implications for the End-to-End Verifiability (E2E-V) of the Voatz system since this parameter identifies voters. See Appendix D: Verifiability and Voatz for further discussion.

Exploit Scenario
An attacker is writing a script that simulates a client, performs the initial handshake, and can invoke subsequent encrypted requests. Realizing that the device ID is insufficiently validated, the attacker no longer has to research how to construct a valid device ID, and can easily construct one that bypasses additional device model checks.

Recommendation
Short term, validate the device ID to ensure it is of good form. Block any users who attempt to register with an invalid device ID. These checks should account for case-sensitivity and the format of the identifier after the "`and-`" and "`ios-`" prefixes.

Long term, review all API request fields to ensure sufficient data validation is performed. In cases where malformed data is provided, consider the strength of the evidence as an indicator that the user is malicious and should be blocked. Avoid reliance on client-supplied values for security of the system.

16. Potential resource exhaustion via logging/storage of unsanitized data

Severity: Medium | Difficulty: Medium
Type: Denial of Service | Finding ID: TOB-VOATZ-031
Target: `CustomerApiWorkerAsync.scala, CustomerMongoDaoAsync.scala`

Description
Request handlers throughout the Voatz backend do not properly sanitize fields, which are later passed through loggers or stored in a database.

```
val logErr = s"EncryptedUpdateCustomerWithIdv req failed: customerId in request
$reqCustomerId does not match customerId $devCustomerId connected to deviceId $deviceId"
log.error(logErr)
```

Figure TOB-VOATZ-031.1: Logging of unsanitized, client-controlled variables (`CustomerApiWorkerAsync.scala#L2524-L2525`).

Assuming the hosting provider uses a logging provider that writes to disk or otherwise performs an expensive operation with the logged data, this could be used to exhaust resources. Similarly, storing many of these variables to the database provides an attack vector similar to TOB-VOATZ-030.

Furthermore, an attacker can construct a field (such as `deviceId`, displayed above) so the logged error seems to say the field's content was part of the encapsulating message. This could be used in a phishing attempt against a developer reviewing logs.

Exploit Scenario
Eve, an attacker, performs a handshake and establishes a connection with the Voatz backend. Afterwards, she sends many requests which trigger an error that concatenates a client-controlled field. As a result, she is able to force the Voatz backend to create very large logs, which could lead to a potential resource exhaustion, depending on the logging provider used.

Recommendation
Short term, ensure that appropriate data validation is performed on client-controlled fields before operating on them.

Long term, review all requests to ensure proper data validation is performed. Fields should be limited in length to prevent resource exhaustion attacks. Ensuring proper form will also prevent any additional attacks from incorrect handling of malformed data.

17. Resource exhaustion via specially crafted Zimperium threats

Severity: Medium | Difficulty: Medium
Type: Denial of Service | Finding ID: TOB-VOATZ-030
Target: `CustomerApiWorkerAsync.scala, CustomerMongoDaoAsync.scala`

Description
The backend currently stores all previously unreported Zimperium-reported threats for a given device ID and `threatId`. However, neither the device ID nor `threatId` are validated, allowing users to spam the backend with specially crafted threat-detected requests which will be stored in the database. This could lead to resource exhaustion.

```
 def createThreatDetection(request: ApiThreatDetectedRequest, ipAddress: String)(implicit
db: MongoDatabase) = async{
   val threatSnapOpt = await(ThreatDetectionAsync.getByDeviceIdAndThreatId(request.deviceId,
request.threatId))
   if(!threatSnapOpt.isDefined){
     await(ThreatDetectionAsync.create(request.deviceId, request.customerId,
request.threatId, request.threatName, request.threatSummary, request.threatType,
       request.threatSeverity, ipAddress))
   }else{
     threatSnapOpt
   }
 }
```

Figure TOB-VOATZ-030.1: Creation of a new detected threat in the database if the device ID and `threatId` *don't already exist (*`CustomerMongoDaoAsync.scala#L2341-L2349`*).*

Assuming the hosting provider sets an upper bound on the size of the database, an attacker may spam these types of requests to ensure the maximum database size is reached. Alternatively, queries for any reported threats against a device may become very expensive, causing resource exhaustion during the authentication process.

It is also worth noting that fields in the request such as `threatName` or `threatSummary` could be set to very large strings. This may cause resource exhaustion when fetching from the database, rendering the data client-side or otherwise displaying such fields on any audit/administrator portal, which may allow review of these threats in the future.

Exploit Scenario
Eve, an attacker, performs a handshake and establishes a connection with the Voatz backend. Afterward, she sends many Zimperium threat-detected requests to the server which are large in size, and use a unique `threatId` each time, knowing it will expand the database and cause threat-related queries to slow down. As a result, the server now must spend more time querying the database for threats, and if a threat pertains to a given device, it must relay data which could be much larger than the request that triggered it.

Recommendation
Short term, ensure that only threats with known `threatId`s are stored in the database. A list of concerning `threatId`s is already maintained by the Voatz backend to determine if a

reported threat should result in the user being blocked from authentication. The Voatz backend should cross-reference this list before storing a reported threat.

Long term, review all requests to ensure proper data validation is performed. Fields should not be allowed to exceed a given length to perform resource exhaustion attacks. Ensuring proper form will also prevent any additional attacks from incorrect handling of malformed data.

18. Zimperium checks on the backend are a blacklist, not a whitelist

Severity: Medium
Difficulty: Medium
Type: Access Controls
Finding ID: TOB-VOATZ-029
Target: `CustomerMongoDaoAsync.scala`

Description
The `wasThreatDetectedOnDevice` function in `CustomerMongoDaoAsync.scala` is used by `CustomerValidationsAsync.scala` to test whether a client's authentication request is valid. It first checks whether the client's IP address is whitelisted and, if not, whether its device ID is present in a list of detected threats reported by Zimperium. This acts as a blacklist of devices that failed the Zimperium anti-tamper checks. However, since it is not implemented as a whitelist, a client that has been modified to remove Zimperium (see Appendix B Finding B.3) will not be present in the list of threats and will therefore pass the authentication validation.

Exploit Scenario
Alice modifies the Voatz client to bypass its anti-tamper checks. The client can then immediately communicate with the Voatz Core Servers, since it will have never failed a Zimperium check.

Recommendation
Short term, switch to a server-side check that whitelists clients, *e.g.*, by ensuring that the client *both* was not tagged as a threat by Zimperium *and* attested to Zimperium in the first place. This, like all anti-tamper protections, is not foolproof. However, it will at least require an attacker to perform the additional step of spoofing a valid Zimperium attestation rather than simply gaining access once Zimperium is bypassed.

Long term, ensure that the security of Voatz is not predicated on the authenticity of the Voter's client.

19. AES-GCM key/nonce/tag encryption system breaks authenticity

Severity: Medium
Type: Cryptography
Target: `Cryptography.scala`

Difficulty: High
Finding ID: TOB-VOATZ-024

Description
AES encryption in the Voatz system happens in two parts: first, an AES-GCM key, nonce, and "tag" are encrypted using AES-ECB (*q.v.* TOB-VOATZ-011). Then, these parameters are used to decrypt the actual data. However, this defeats the authenticity guarantees that AES-GCM provides. Because AES-ECB is malleable, the key and nonce can be arbitrarily modified. This allows for the creation of a new key and nonce such that the saved ciphertext decrypts to an arbitrary plaintext. This attack is best detailed in Dodis *et al*.'s 2018 work *Fast Message Franking: From Invisible Salamanders to Encryptment*.

Exploit Scenario
Alice learns Bob's shared secret. She can then modify the AES-ECB encrypted key, nonce, and tag such that Bob's encrypted data decrypts to a message of her choosing (so long as it is the same length as the original message). AES-GCM's authenticity guarantees do nothing, as the checks they perform are a function of the key used.

Recommendation
Short term, remove all use of AES-ECB from the Voatz system. Instead of multiple levels of AES with encrypted keys and nonces, simply have one standard AEAD construction.

Long term, standardize all symmetric encryption in the Voatz system to a single AEAD construction. Remove any use of symmetric primitives other than this construction.

References
- Fast Message Franking: From Invisible Salamanders to Encryptment

20. Unauthenticated ECDH is vulnerable to key compromise impersonation

Severity: Medium
Difficulty: High
Type: Cryptography
Finding ID: TOB-VOATZ-004
Target: `Cryptography.scala`

Description
Unlike protocols such as TLS and Wiregard, Voatz' use of ECDH does not authenticate handshakes, and is therefore vulnerable to Key Compromise Impersonation (KCI) attacks. An attacker with access to a voter's private key can impersonate the Voatz server without *either* party being able to detect the deception. To be explicit, this issue refers to the ECDH implementation in Scala found in the Core Server codebase, not that performed in the TLS handshake.

Exploit Scenario
Bob wishes to cast a vote using Voatz. Alice remotely compromises Bob's phone (*e.g.*, via a known vulnerability in Bob's mobile operating system, or via phishing), gaining access to his private key. Alice can then man-in-the middle communication between Bob and Voatz, altering messages without either end detecting. Notably, logging public keys cannot help expose such an attack because from Bob's perspective, Alice and Voatz have identical public keys.

For example, Alice can masquerade as the Voatz server, accept Bob's vote, discard the vote, and Bob will be unaware of his disenfranchisement. Alice can then use the private key to submit a *different* vote to the *real* Voatz server. This does not require any modification to the Voatz mobile application.

Notably, there is something of an *ad hoc* password-authenticated key exchange implementation making use of the user's device ID, a large list of fake keys, and the ECDH scheme described above. The security properties of this scheme are undetermined, and the use of the device ID specifically leads to the issues described in TOB-VOATZ-014. We recommend removing this scheme with the upgrade to a more standardized AKE as described below.

Recommendation
Short term, replace ECDH and the system's *ad hoc* PAKE with Noise or a TLS 1.3 handshake. These are authenticated, and prevent key compromise impersonation.

Long term, avoid designing any kind of transport encryption. Use standardized and integrated frameworks such as Wireguard or TLS 1.3.

References
- Key Compromise Impersonation attacks (KCI)

21. AES-GCM keys, nonces, and “tag”s are encrypted using AES-ECB

Severity: Medium
Type: Cryptography
Target: `Cryptography.scala`

Difficulty: High
Finding ID: TOB-VOATZ-011

Description
In Cryptography.scala, the `encryptKeyNonceTag` function is used to encrypt secrets used for AES-GCM encryption. It does this by invoking `cipher.getInstance("AES")`, which returns a cipher using AES in the famously insecure ECB mode. This mode has no semantic security or authentication properties.

Exploit Scenario
Alice has compromised the Voatz system in such a way that she has discovered some key/nonce/tag triples. She can unambiguously associate those with their encrypted and stored versions, since AES-ECB lacks semantic security. She can also undetectably modify stored entries such that they decrypt to a triple of her choosing, since AES-ECB is trivially malleable.

Recommendation
Short term, remove all use of AES-ECB from the codebase, replacing it with AES-GCM.

Long term, add a cryptographic analyzer such as Cryptosense to Voatz’ continuous integration process to automatically detect use of insecure algorithms.

References
- Why shouldn’t I use ECB mode?

22. Voatz API server lacks OCSP stapling

Severity: Medium
Type: Cryptography
Target: Voatz Core Server and Clients

Difficulty: High
Finding ID: TOB-VOATZ-033

Description
The Voatz Core Server does not return its SSL certificate's revocation status via *OCSP Stapling*. This feature provides clients with the ability to detect whether the server's SSL certificate has been revoked.

Apple recommends that OCSP Stapling should be implemented on all mobile endpoints. This implies that OCSP Stapling will become a requirement for iOS Apps on the App Store.

Exploit Scenario
Alice gains access to a Voatz server using its shared wildcard SSL certificate (see TOB-VOATZ-028), compromising the certificate's private key. Even if Voatz revokes the compromised certificate, clients will continue to allow connections to any server with the certificate—even ones hosted by Alice—because there is no OCSP Stapling.

Recommendation
Short term, update all Voatz servers and mobile clients to enable support for OCSP Stapling.

Long term, perform certificate revocation exercises to ensure that the protections are sufficient, as well as to train Voatz staff on how to react to a compromised SSL credential.

References
- Apple WWDC 2017: Your Apps and Evolving Network Security Standards
- Apache SSL/TLS Strong Encryption: OCSP Stapling

23. Empty ballots are not recorded in Hyperledger

Severity: Low
Type: Data Validation
Target: Voatz Core Server and Audit Portal

Difficulty: Low
Finding ID: TOB-VOATZ-027

Description
Each “oval” (ballot selection) is stored in a block in the Hyperledger blockchain. If a ballot is recorded in which the voter did not select any candidates, nothing will be saved to Hyperledger. The voter’s ballot will still be recorded in MongoDB and MySQL, a paper ballot generated, and a correct receipt PDF E-mailed. However, there will be no record of the vote in Hyperledger.

Exploit Scenario
At least one voter submits a ballot with no ovals filled. During the auditing phase, an auditor cannot validate that the ballot was legitimate since there are no corresponding blocks in Hyperledger.

Recommendation
Short term, improve documentation and training materials for auditors to inform them of this edge case.

Long term, store all ballot oval states in Hyperledger.

24. Database root credentials stored in git

Severity: Undetermined
Type: Data Exposure
Target: `deploy/secrets.txt`

Difficulty: Low
Finding ID: TOB-VOATZ-016

Description
The MongoDB and MySQL passwords were both added to git in commit `2809e13385bbaeba6c0a45cfabbc4f272f775526#diff-e1870fde547e5595490d9d9000dbe1bb`. Both databases appear to have the same, relatively weak, password. Additionally, both databases appear to run as the root user. This password can also be found in `build.sbt`.

Exploit Scenario
An attacker compromises a Voatz employee with access to git but without authorized access to the Voatz database servers. The attacker searches git and finds credentials to access these resources, then gains root on the MongoDB and MySQL database servers.

Recommendation
Short term, use unique passwords per application. Do not store secrets in source code. Do not run databases as root.

Long term, ensure all credentials are controlled by a dedicated secret management application as described in TOB-VOATZ-013.

25. Signed voter affidavits are sent to an administrative email

Severity: Undetermined
Type: Data Exposure
Target: `ReceiptRouter.scala`

Difficulty: Medium
Finding ID: TOB-VOATZ-021

Description
Once an affidavit is signed and processed by the voter, a copy is E-mailed to them and BCC'd to an "admin notice inbox" E-mail address specified in the client config, defaulting to [redacted]@voatz.org.

```
val bccAddress: EmailAddress = EmailAddress(
  request.event.eventData.voteReceiptConfiguration.get
    .adminNoticeInboxes.tail.headOption.getOrElse("[redacted]@voatz.org")
)
```

Figure TOB-VOATZ-021.1: Signed affidavits are BCC'd to an administrator E-mail address (`ReceiptRouter.scala#L252–255`).

The purpose of this delivery method and destination is unclear, hence the undetermined severity of this finding. However, if the affidavits are used for auditing the election (*e.g.*, to ensure that all voters properly signed the affidavit), then this will lead to a high severity exploit, as follows.

Exploit Scenario
Alice writes a script to deny service to the [redacted]@voatz.org email address by filling its inbox with spam. Once the inbox is full, legitimate affidavit emails will bounce. An audit of the election will make it appear as if ballots were cast where the voter did not sign the affidavit.

Recommendation
Short term, ensure that the "admin notice inbox" is large enough to thwart any attempted spam overrun.

Long term, transition to a different method for archiving signed affidavits that is not prone to denial of service.

26. AES-GCM AAD usage is nonstandard

Severity: Undetermined
Type: Cryptography
Target: `Cryptography.scala`

Difficulty: High
Finding ID: TOB-VOATZ-012

Description
The usage of AES-GCM throughout the Voatz codebase is not standard. In addition to the typical key and IV inputs to the cipher, there is pervasive use of a third input, called “tag”. This “tag” is added to the Authenticated Additional Data (AAD), but not used for any other purpose. In some usage there is no “tag” provided, and instead a null buffer is used.

This does not appear to add any security benefit, but it does allow unambiguous association of key/nonce/tag triples with ciphertexts.

The severity of this finding is Undetermined because the security implications of the nonstandard tag are not fully understood.

Exploit Scenario
Alice compromises a key. Since the AAD is not encrypted, Alice can use the nonstandard tag to match the key to ciphertexts for which it was used *without* needing to perform any brute-force decryption operations.

Recommendation
Short term, remove the “tag” parameter from Voatz code. Additionally, remove code updating the AAD with a null buffer.

Long term, carefully audit all cryptographic primitives for use conforming to their specification.

27. Session cookie expiration offset is a hardcoded literal

Severity: Informational | Difficulty: Low
Type: Configuration | Finding ID: TOB-VOATZ-005
Target: `CustomerRoutesAsync.scala, OrganizationRoutesAsync.scala`

Description
During customer and organization authentication methods, the Voatz backend sets the user's session cookie expiry date as an offset from the current timestamp. These offsets are supplied as a hardcoded integer literal.

```
                setCookie(HttpCookie(SessionCookie, session.sessionCookie, httpOnly = true,
secure = config.HttpConfig.UseHttps,
                  expires = Some(DateTime.now.+(36000001)), domain = domainVal, path =
Some("/"))) {
                  complete(AuthenticateResultWithNextKey(nextKey, customerSnapshot,
votedEventIdLastUseTsPairs))
                }
```

Figure TOB-VOATZ-005.1: Cookie expiration setting during in `authenticateCustomer` (`CustomerRoutesAsync.scala#L318-L321`).

```
                setCookie(HttpCookie(SessionCookie, payload.sessionCookie, httpOnly = true,
secure = config.HttpConfig.UseHttps,
                  expires = Some(DateTime.now.+(432000001)), domain = domainVal, path =
Some("/"))) {
                  complete(orgSnapshot)
                }
```

Figure TOB-VOATZ-005.2: Cookie expiration setting in `extendedOrgKeyLogin` (`OrganizationRoutesAsync.scala#L959-L962`).

As seen above, there are two different hardcoded cookie expiration offsets are used, `36000001` and `432000001`. These are used in various places throughout these two files.

In the event these cookie expiration offsets are found to be insufficient, multiple instances of hardcoded literals existing as such would increase the possibility of developer-error during any refactoring.

Exploit Scenario
Bob is a developer of Voatz. Upon review, Bob realizes the cookie expiration dates set are insufficient and wishes to refactor them. He attempts to refactor all instances of the undesirable timestamp offset, but misses an instance. This results in a disjoint in session cookie expiration times when performing different authentication operations.

Recommendation
Short term, refactor the session cookie expiration offsets so that they are derived from a singular definition and can be uniformly refactored across the codebase.

Long term, review the use of hardcoded literals throughout the codebase. Ensure that significant variables do not make use of repetitively hardcoded literals, but instead derive from well-defined constants, configuration-based variables, or otherwise uniform definitions.

Android Findings

28. Encrypted application data is trivially brute-forceable

Severity: High
Type: Cryptography
Target: Voatz Android Client

Difficulty: Low
Finding ID: TOB-VOATZ-048

Description
The Android client creates a local database that stores sensitive identifiers and the user's voting history. This database is encrypted with the user's 8-digit PIN; however, it can be trivially brute-forced. An attacker gaining access to this database file would see the user's past votes and have the means to impersonate the user in an election.

The Voatz Android client attempts to protect its local data through the following process:

1. Request the user create an 8-digit PIN code to protect their data
2. Use PBKDF2 with 1,000 iterations and an 8-character salt to convert the PIN to a key
3. Provide the key to the Realm.io encryptionKey parameter
4. Store the encrypted Realm database and the unencrypted salt on the filesystem

First, this process is sabotaged by the extremely low entropy of 8-digit numeric PIN codes, for which there are only 99,999,999 possible options. By comparison, an 8-character alphanumeric password has 218 trillion possible options.

The Android client uses PBKDF2 ("PBKDF2WithHmacSHA1") to slow down attempts to brute force the user's 8-digit PIN; however, commonly available laptop computers can guess ~100,000 PBKDF2 keys per second. Therefore, it only requires, at most, 15 minutes to fully exhaust the keyspace of the 8-digit PIN and successfully decrypt this file.

Second, this cryptographic system does not tie the encrypted database to the Android device. It is possible to extract the database from the Android device and crack it on a different, faster device, like a laptop computer or specialized password cracking system.

The Realm database stores a uniquely identifying audit token, the past history of votes, and various notifications and configuration information for the Voatz app. In particular, knowing the audit token allows an attacker to:

1. De-anonymize votes given access to the audit portal, Hyperledger, S3, or the voter's ballot receipt
2. Submit their own vote with the same audit token (via TOB-VOATZ-046), causing discrepancies during an audit or even causing the first voter's ballot to be discarded (via TOB-VOATZ-020). The attacker must already be registered as a voter, and can only exploit this vulnerability once, wasting their own vote in the process. This is

because the backend does not prevent double-voting via audit tokens; it uses the customer ID, which is uniquely tied to the device ID that made the API request.

This vulnerability has a similar effect to the PIN-cracking finding of the MIT report (see B.5), but requires less technical expertise and device access.

Exploit Scenario
Alice leaves her phone unattended at a bar, and Bob extracts the encrypted Voatz database from it. On his home computer, Bob writes a script that tries PIN codes one at a time from 00000000 to 99999999, derives a key with PBKDF2 for each, then checks if decryption is successful. Bob gains access to Alice's Voatz data within 15 minutes and impersonates her in an election.

Recommendation
Short term, encrypt the salt with the Android keystore. This will complicate attempts to crack the database, but the gain is marginal and attacks will remain easy. Use the getStorageEncryptionStatus() method to check whether a user's Android device is encrypted, and do not let them vote if it is not. This will reduce the likelihood that others can extract data from a user's phone. See also TOB-VOATZ-025 for a discussion of the KDF.

Long term, replace this cryptographic system with one based on the Android StrongBox. The Android StrongBox Keymaster facilitates generating and using keys in a built-in hardware security module where they cannot be easily extracted by an attacker. This will ensure that password cracking attempts must occur on the device rather than on a remote system.

References
- Android Developer Documentation: Android keystore system
- DevicePolicyManager: getStorageEncryptionStatus()

29. PBDKF2 provides insufficient security margin for PIN codes

Severity: High
Type: Cryptography
Target: `PinCrypto.kt`

Difficulty: Low
Finding ID: TOB-VOATZ-025

Description
In the Android client, PBKDF2 with 1,000 iterations is used as a key derivation function (KDF) to generate a cryptographic key from the user's PIN. In this configuration, PBKDF2 provides an insufficient security margin due to the extremely low entropy of 8-digit PIN codes.

There are 99,999,999 possible options for 8-digit PIN codes, and common laptop computers can guess ~100,000 PBKDF2 keys (with 1,000 iterations) per second. Therefore, it only requires ~15 minutes to crack the user's PIN.

Exploit Scenario
An attacker compromises data encrypted with a user PIN derived from PBKDF2. The attacker fully exhausts the possible PIN codes in a short time period.

Recommendation
Short term, follow NIST 800-163 guidelines for setting the iteration count of PBKDF2. This will reduce the rate at which an attacker can guess user PINs.

> *... the computation required for key derivation by legitimate users also increases with the number of iterations. The user may perceive this increase, for example, in the time required for authentication, or in the time to access the protected data on the storage medium. There is an obvious tradeoff: larger iteration counts make attacks more costly, but hurt performance for the authorized user.* ***The number of iterations should be set as high as can be tolerated for the environment, while maintaining acceptable performance.***

Long-term, replace PBKDF2 with Argon2id or scrypt. These modern KDFs are *memory-hard* and therefore will frustrate attempts at parallelization and brute-force password cracking to a greater degree than PBKDF2. To comply with NIST 800-163, Argon2id is based on AES, and scrypt is based on SHA-256, making either one acceptable for use in the absence of strict FIPS 140 requirements.

References
- NIST Authenticator and Verifier Requirements
- Wikipedia: Argon2 and scrypt

30. Third-party apps can capture the Android client screen and read screenshots taken from the client

Severity: High
Type: Data Exposure
Target: Voatz Android Client

Difficulty: Medium
Finding ID: TOB-VOATZ-032

Description
The `android.media.projection` API was introduced in Android 5.0. It allows any third-party app on the phone to perform a screen capture of other running apps, including the Voatz client. Such a third-party app can capture everything on the device's screen, even sensitive activity such as password keystrokes. Third-party apps may continue recording the screen even after the user terminates/closes the app, but not after a reboot.

The Voatz client can prevent this behavior by enabling the `FLAG_SECURE` flag. Screenshots taken by the user are, by default, stored on the phone's SD card where they are accessible to any other application. The `FLAG_SECURE` flag also has the added benefit of preventing screenshots.

Exploit Scenario
Alice writes a malicious app that Bob installs. Alice's app surreptitiously records and exfiltrates a recording of Bob's use of the Voatz app as he is entering his sensitive information and ballot choices.

Recommendation
Short term, protect all sensitive windows within the Voatz app by enabling the `FLAG_SECURE` flag. This will prevent malicious third-party apps from recording usage of the Voatz app and taking screenshots of sensitive information.

Long term, ensure that developer documentation is updated to include screen capture and recording as potential threats for data exposure.

31. Android release build signing key password and keystore password stored in git

Severity: High
Type: Data Exposure
Target: `app/build.gradle`

Difficulty: High
Finding ID: TOB-VOATZ-003

Description
The authentication key password and keystore password used for the `androidvma` code signing key are stored in plaintext in the `build.gradle` file currently on the master branch of the `android` repository. This key appears to be used for code signing release builds of the Android application.

Although the passwords may only be used to authenticate use of the actual key, it is considered bad practice to hardcode and store such authenticating credentials in the git repository because an attacker with access to the `androidkeystore` can authenticate and use these keys.

Exploit Scenario
Voatz developer Bob has his machine compromised by an attacker, Alice. Knowing that Bob has code signing keys in the Android keystore on his machine, Alice is able to use the keystore credentials to authenticate and perform a code signing operation.

Recommendation
Short term, remove these passwords from the git repository and repository history. Instead, integrate them into the CI/CD pipeline accordingly.

Long term, assess the storage of sensitive credentials in order to minimize the capability of a well-positioned attacker.

32. A malicious website can read from the Android client's internal storage

Severity: High
Type: Data Exposure
Target: `WebViewExtension.kt`

Difficulty: High
Finding ID: TOB-VOATZ-035

Description
The Android client uses a WebView to, for example, access the Voatz "contact us" webpage during the signup process. By default, Android WebViews allow the webpage to access the app's local storage.

Exploit Scenario
Alice compromises a web page accessed from the Voatz Android client's WebView. She can leverage this access to exfiltrate all of the internal storage of the Voatz Android client.

Recommendation
Short term, explicitly set the `setAllowFileAccess` method to `false`:

```
webview.getSettings().setAllowFileAccess(false);
```

Long term, add tests to ensure that malicious websites cannot read the client's internal storage via a WebView.

33. Insufficient Android deviceId construction

Severity: Low
Type: Session Management
Target: `ContextExtension.kt`

Difficulty: Low
Finding ID: TOB-VOATZ-008

Description
The Android client uses `Settings.Secure.ANDROID_ID` to construct a `deviceId` for uniquely identifying their device. It should not be assumed that this identifier is persistent, since factory-reset operations are known to reset this ID.

Furthermore, it has been noted that this function may return `null`. It should not be assumed any specific Android model will not return `null` under specific conditions or in a future update. However, the Android client lacks a `null` check to validate the obtained `ANDROID_ID` before it is prepended with "`and-`" to construct the `deviceId`.

Exploit Scenario
Alice downloads the Voatz app and attempts to complete the onboarding process. Unfortunately, Alice's device returns an `ANDROID_ID` of `null`, causing her `deviceId` to be constructed as "`and-null,`" which the Voatz backend will currently fail to validate as it is not a valid `deviceId`. This means Alice can continue to use the application despite the lack of a valid `deviceId`.

Recommendation
Short term, check for the `null` case when obtaining the `ANDROID_ID`, document the effects of a factory reset operation on the Android `deviceId`, and provide clear instructions for how voters can remediate the problem if it occurs.

Long term, review all unique identifiers for users and ensure they cannot collide with one another. Ensure users are made wellaware of cases in which these identifiers could change and affect their voting experience.

34. Android client does not use the SafetyNet Attestation API

Severity: Low
Type: Configuration
Target: Voatz Android Client

Difficulty: High
Finding ID: TOB-VOATZ-037

Description
The SafetyNet Attestation API is not checked by the Voatz Android client.

Google Play provides the SafetyNet Attestation API for assessing the safety of the device that apps are running on. The API uses software and hardware information to provide a cryptographically signed attestation about the overall integrity of the device. This can provide an additional line of anti-tamper defense in conjunction with Zimperium.

The SafetyNet Attestation API is capable of handling devices that have passed Compatibility Test Suite (CTS) certification and devices that have not (via the `basicIntegrity` parameter).

Exploit Scenario
Alice is using a phone that has been rooted and has malware on it. The modifications to her device would be detected by Google through the SafetyNet Attestation API; however, the Voatz app does not check it before allowing Alice to vote.

Recommendation
Short term, use the SafetyNet Attestation API to assess the integrity and safety of the user's device. Configure the Attestation API to use the `basicIntegrity` parameter to support devices that have not passed CTS certification.

Long term, require an affirmative `ctsProfileMatch` result which indicates that the user is in possession of a device that passed CTS certification. Devices without a CTS certification possess unknown security risks and increase the likelihood that the device has been compromised.

References
- Android Developer Documentation: SafetyNet Attestation API
- Inside Android's SafetyNet Attestation: Attack and Defense

35. Android client does not use the SafetyNet Verify Apps API

Severity: Low
Type: Configuration
Target: Voatz Android Client

Difficulty: High
Finding ID: TOB-VOATZ-045

Description
The SafetyNet Verify Apps API is not checked by the Voatz Android client.

Google Play provides the SafetyNet Verify Apps API to check whether there are potentially harmful apps on a user's device. Google monitors and profiles the behavior of Android apps, and informs users of potentially harmful apps via the Verify Apps feature. Users are notified and encouraged to remove the app. However, they are free to disable this feature and free to ignore these warnings. The SafetyNet Verify Apps API can tell Voatz whether this feature is enabled and whether any such apps remain on the user's device. This can provide an additional line of defense in conjunction with Zimperium.

Exploit Scenario
Alice has unknowingly installed a malicious application on her Android device that is detected by Google SafetyNet but not by Zimperium. She ignores the warnings to uninstall the app because it includes a game she enjoys. Alice uses the Voatz app to participate in an election. The malicious app abuses available Android Intents and access to phone storage to manipulate or record her actions with the Voatz app.

Recommendation
Short term, use the SafetyNet Verify Apps API to require that this feature be enabled for all Voatz users, and ensure that known, harmful apps are not installed on their devices.

Long term, stay updated on new security features in Android and continue adding relevant safety protections to the Voatz mobile clients.

References
- Android Developer Documentation: SafetyNet Verify Apps API
- App security best practices

36. Certificate pinning is only configured for the main Voatz domain

Severity: Low
Type: Cryptography
Target: `network_security_config.xml`

Difficulty: High
Finding ID: TOB-VOATZ-026

Description
Voatz uses TrustKit to force certificate pinning. However, the only certificates pinned are those of the primary Voatz API domain, not any of the third-party services used. This means an attacker could still potentially man-in-the-middle calls to APIs other than the primary one used.

Exploit Scenario
An attacker performs a man-in-the-middle attack against calls to the Jumio identity verification service to steal user PII. The included certificate pinning is ineffective, because the calls are not to the Voatz domain.

Recommendation
Short term, pin certificates for all third-party APIs in the TrustKit configuration.

Long term, audit all network calls made by the application and maintain a list of domains accessed. For each domain, ensure calls are only made using TLS with certificate pinning.

37. No explicit verification of the Android Security Provider

Severity: Low
Type: Patching
Target: Voatz Android Client

Difficulty: High
Finding ID: TOB-VOATZ-034

Description
The Voatz Android client does not explicitly check whether it is running on a device that has an up-to-date Android Security Provider.

The Security Provider is responsible for providing secure network communications, such as SSL/TLS. Running Voatz on a device with an outdated Security Provider exposes it to network attacks. For example, it can allow an attacker on the network to decrypt and compromise Voatz' SSL/TLS traffic.

Zimperium's library may provide some or all of these checks; however, that library must be included and configured appropriately to do so, and there is no guarantee the library will be included with Voatz in the future (see the December 2018 security review by ShiftState).

Exploit Scenario
A new vulnerability discovered in Android can be exploited to produce a man-in-the-middle attack (similar to CVE-2014-0224). Bob has not upgraded his phone to include the latest version of the Android Security Provider to mitigate this vulnerability, and his SSL traffic to the Voatz API server can be snooped and modified.

Recommendation
Short term, on every app startup, run `ProviderInstaller.installIfNeeded()` supplied by Google Play services. This method will ensure that the Android Security Provider is up to date. If the Security Provider remains out of date or an error occurs, this method will throw an exception and Voatz should decline to run.

Long term, continue adding anti-tamper and security update protections to the Voatz clients.

References
- Android Developer Documentation: ProviderInstaller.installIfNeeded()
- Update your security provider to protect against SSL exploits

38. Jumio Netverify API credentials stored in git

Severity: Undetermined
Type: Data Exposure
Target: `IDVStartActivity.java`

Difficulty: Low
Finding ID: TOB-VOATZ-001

Description
The Jumio Netverify API token and secret were added to the Android client repository in commit `d1e9a0c0e8bb26ab27d39876f4b2210c0d235d9f`. Specifically, they were included on lines 48–50 of:

```
app/src/main/java/voatz/nimsim/com/voatz/
ui/registration/IDVStartActivity.java
```

While this file was later deleted in commit `764826e43dd48891cd047ac93b7ddcc5c4f61113`, the credentials still exist in the git history.

The severity of this finding is Undetermined because it is unclear whether the API credentials are still valid.

Exploit Scenario
Alice is given access to the Voatz Android git repository (*e.g.*, as a new employee of Voatz, or as an employee of a subcontractor working with Voatz). Alice discovers the Netverify API credentials, which allows her to use them for her own purposes, potentially incurring usage fees billed to Voatz. She can also use the Netverify Retrieval and Delete APIs to exfiltrate sensitive voter data and delete voter data, compromising the auditability of the election.

Recommendation
Short term, rebase the Android git repository to remove this API token and secret. Check whether the credentials are still active and, if so, revoke them and generate new ones.

Long term, integrate a tool like truffleHog into your git hooks to prevent sensitive information from being committed to the repository in the first place.

References
- TruffleHog

39. Google Services API key stored in git

Severity: Undetermined
Type: Data Exposure
Target: `google-services.json`

Difficulty: Low
Finding ID: TOB-VOATZ-002

Description
Voatz' Google Services API key was added to the Android client repository in commit `d1e9a0c0e8bb26ab27d39876f4b2210c0d235d9f`. Specifically, it was included on lines 22–25 of:

```
app/google-services.json
```

Although this file was later deleted in commit `764826e43dd48891cd047ac93b7ddcc5c4f61113`, the credentials still exist in the Git history.

The severity of this finding is Undetermined because it is unclear whether the API key is still valid, and the restrictions placed on this specific key are unknown.

Exploit Scenario
Alice is given access to the Voatz Android Git repository (*e.g.*, as a new employee of Voatz, or as an employee of a subcontractor working with Voatz). Alice discovers the Google Services API key and is able to interact with Google Services as if she were Voatz—potentially incurring usage fees billed to Voatz and accessing Voatz cloud data.

Recommendation
Short term, rebase the Android git repository to remove this API key. Check whether the key is still active and, if so, revoke it and generate a new one.

Long term, integrate a tool like truffleHog into your Git hooks to prevent sensitive information from being committed to the repository in the first place.

References
- TruffleHog

40. A malicious website may be able to execute JavaScript within the Android client

Severity: Informational
Type: Access Controls
Target: `WebViewExtension.kt`

Difficulty: High
Finding ID: TOB-VOATZ-036

Description
The Android client uses a WebView to, for example, access the Voatz “contact us” webpage during the signup process. By default, Android WebViews disable JavaScript, but it is a good idea to explicitly disable it.

Exploit Scenario
A future update to Android changes the default JavaScript behavior of WebViews, allowing an attacker with control of a webpage loaded from the WebView to run malicious code within the Voatz Android client.

Recommendation
Short term, explicitly set the `setJavaScriptEnabled` method to `false`:

```
webview.getSettings().setJavaScriptEnabled(false);
```

Long term, add tests to ensure that malicious websites cannot execute JavaScript via a WebView.

iOS Findings

41. The iOS client does not disable custom keyboards

Severity: Medium
Type: Data Exposure
Target: `AppDelegate.swift`

Difficulty: Medium
Finding ID: TOB-VOATZ-040

Description
The Voatz iOS client does not disable custom keyboards. Since iOS 8, users have been able to install custom keyboards that can be used in any app, replacing the system's default keyboard. Custom keyboards can—and very frequently do—log and exfiltrate the data they enter.

Custom keyboards are not enabled when the user types into a “Secure” field (such as password fields) but they can potentially log all the user’s keystrokes in regular fields, such as those used for the voter’s personal information or write-in candidates. Voatz does not use system-managed input fields for username and password entry (TOB-VOATZ-042)’ therefore, those fields would get logged by custom keyboards.

Exploit Scenario
Alice creates a custom keyboard that Bob uses. Alice’s keyboard can silently exfiltrate all of Bob’s keystrokes in the Voatz app.

Recommendation
Short term, disable third-party keyboards within the Voatz iOS client to prevent leakage of sensitive data entered by the user. This can be achieved by implementing the `application:shouldAllowExtensionPointIdentifier:` method within the Voatz client’s `UIApplicationDelegate`.

Long term, stay abreast of changes to iOS that might permit data exfiltration from the Voatz client.

References
- Apple Developer Documentation: UIApplicationDelegate

42. The iOS client does not use system-managed login input fields

Severity: Low
Type: Configuration
Target: Voatz iOS Client

Difficulty: Low
Finding ID: TOB-VOATZ-042

Description
The Voatz iOS client does not specify text fields marked as username and password input fields. Since iOS 12, the iOS SDK has included text field properties to automate the process of password generation and credential entry, offering to auto-generate strong passwords and save them in the system keychain or a password manager. This could be used for the current PIN entry field.

Furthermore, identifying these fields as login input fields may help prevent entered text from being misused by iOS. Text entered into fields that lack these identifiers may be sent to a spellcheck service, added to an auto-complete dictionary, or otherwise cached in a way that increases their risk of exposure.

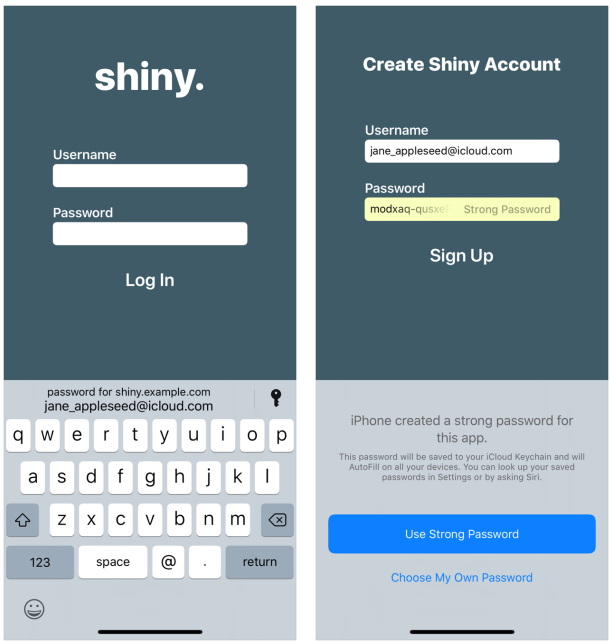

Figure TOB-VOATZ-042.1: iOS offers to generate strong passwords for identified login fields.

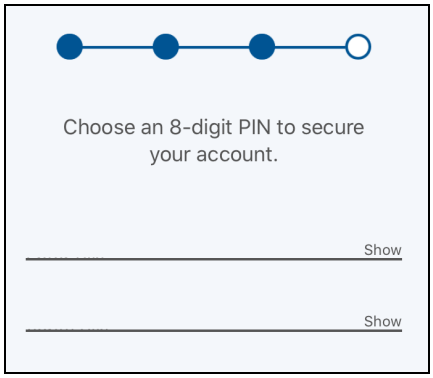

Figure TOB-VOATZ-042.2: Current Voatz PIN entry field without this feature.

Exploit Scenario
Bob installs Voatz and cannot use a machine-generated PIN since Voatz does not use system-managed login input fields on iOS. Bob chooses an insecure PIN code, and it is cached into an auto-complete dictionary by iOS.

Recommendation
Short term, use the `UITextContentType` property introduced in iOS 12 to identify username and password fields, allowing automated password generation and management.

Long term, stay abreast of new security features added to the iOS SDK.

References
- Apple Developer Document: textContentType
- About the Password AutoFill Workflow

43. iOS client keychain items are not excluded from iCloud and iTunes backups

Severity: Low
Type: Data Exposure
Target: Voatz iOS Client

Difficulty: High
Finding ID: TOB-VOATZ-043

Description
The Voatz iOS client does not prohibit its keychain items from being saved to an iTunes backup or uploaded to iCloud. Both Apple, Inc. and any attacker with access to a voter's iTunes or iCloud backup will have access to a voter's private data.

Exploit Scenario
Alice gains physical access to Bob's phone, and knows his passcode. She initiates a backup of Bob's phone to iTunes from which she is able to extract all of Voatz' sensitive keychain data. Alternatively, Mallory identifies voter email addresses, then uses a previously disclosed password database to guess their current iCloud passwords. She retrieves iCloud backups that contain sensitive Voatz keychain data from a large number of users.

Recommendation
Short term, explicitly set a `ThisDeviceOnly` accessibility class (such as `kSecAttrAccessibleWhenUnlockedThisDeviceOnly`) for all keychain items. This should prevent keychain data from being migrated to iTunes and iCloud backups.

Long term, empirically validate that no sensitive data is stored to a backup of the Voatz iOS application. Consider uniform usage of a wrapper, such as Square's Valet, to simplify storage and retrieval of data from the keychain.

References
- Apple Developer Documentation: Keychain Services
- Square Valet

44. Cryptographic credentials are not generated in the iOS Secure Enclave

Severity: Low
Type: Cryptography
Target: `CryptoExportImportManager.swift`

Difficulty: High
Finding ID: TOB-VOATZ-041

Description
The Voatz iOS client does not use the Secure Enclave API to securely generate its keys.

iOS 13 provides an API that generates and stores cryptographic credentials inside the Secure Enclave. This means that cryptographic credentials never actually leave the Secure Enclave and are therefore never stored in plaintext in memory. This feature is available on all iOS devices with an A7 chip or newer.

Exploit Scenario
Mallory compromises the iOS device of a prospective voter. She uses her access to read the voter's cryptographic credentials from memory and is able to communicate with the Voatz API server directly on behalf of the voter.

Recommendation
Short term, use the Secure Enclave when performing any cryptographic operation on the device to avoid revealing sensitive credentials in memory to the application processor.

Long term, stay abreast of new cryptographic features added to the iOS SDK.

References
- Apple Developer Documentation: Storing Keys in the Secure Enclave
- Apple Platform Security: Secure Enclave Overview

45. iOS client disables App Transport Security (ATS)

Severity: Low
Type: Cryptography
Target: Voatz iOS Client

Difficulty: High
Finding ID: TOB-VOATZ-044

Description
App Transport Security is disabled by the Voatz iOS app.

Apple platforms include a network security feature called App Transport Security (ATS) that improves the use of encryption and integrity protections for network communications. It does this by requiring that network connections are secured by Transport Layer Security (TLS) with stronger-than-default certificates and ciphers. ATS blocks connections that fail to meet minimum security requirements.

```
<key>NSAppTransportSecurity</key>
<dict>
→   <key>NSAllowsArbitraryLoads</key>
→   <true/>
</dict>
```

Figure TOB-VOATZ-044.1: Apple Transport Security is fully disabled by the Voatz iOS client.

By default, all TLS connections on iOS check that the server certificate:

- Has an intact digital signature
- Is not expired
- Has a name that matches the server's DNS name
- Is signed by a certificate chain ending in a valid Certificate Authority

ATS requires these checks, and provides additional checks:

- The server certificate must be signed with an RSA key of at least 2048 bits or an ECC key of at least 256 bits
- The server certificate must use SHA-2 with a digest length of at least 256 bits
- The connection must use TLS protocol version 1.2 or later
- Data must be exchanged using AES-128 or AES-256
- The link must support perfect forward secrecy (PFS) through an Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) key exchange

Zimperium's library may provide some or all of these checks; however, that library must be included and configured appropriately to do so, and there is no guarantee the library will be included with Voatz in the future.

Exploit Scenario

Alice uses the Voatz app in an election, and her preferences are sent to Voatz via network communications encrypted with an outdated version of TLS and weak ciphers. Bob is a network administrator at an intermediate routing point with access to Alice's network traffic. He uses an active attack against the outdated version of TLS to decrypt Alice's traffic, or collects it for future decryption via the Logjam weakness of the server (for example).

Recommendation
Short term, precisely define the ATS exceptions required for the Voatz app. Configure ATS exceptions only when needed, use the narrowest possible exception available, and upgrade Voatz servers to meet the requirements imposed by ATS.

Long term, remove all exceptions. All network communications should meet the minimum requirements imposed by ATS.

References
- Apple Developer Document: NSAllowsArbitraryLoads
- Preventing Insecure Network Connections
- RFC7457: Summarizing Known Attacks on Transport Layer Security (TLS)

46. iOS client is vulnerable to object substitution attacks

Severity: Undetermined
Type: Data Validation
Target: Voatz iOS Client

Difficulty: High
Finding ID: TOB-VOATZ-038

Description
The insecure `NSCoding` protocol is used throughout the Voatz iOS client codebase, and in its `ZDetection` and `AWSCore` dependencies. `NSCoding` is designed to allow serialization and deserialization of code objects. However, this protocol does not verify the type of object upon deserialization. Thus, it is vulnerable to object substitution attacks.

A maliciously crafted payload deserialized via the `NSCoding` protocol can result in execution of attacker-controlled code. Apple provides the `NSSecureCoding` protocol, which is robust to this type of attack. `NSSecureCoding` protects against object substitution attacks by requiring the programmer to declare the expected type of object before deserialization completes. Thus, if an invalid object is deserialized, the error can be handled safely.

The severity of this finding is Undetermined because it is unclear whether there are any available attack vectors that can exploit the vulnerability.

Exploit Scenario
Alice gains control of a resource that is loaded into the iOS client via the `NSCoding` protocol. This allows her to instantiate the object as whichever class she chooses.

Recommendation
Short term, migrate all classes that use `NSCoding` to `NSSecureCoding`.

Long term, ensure all input data is validated before it is used, especially when dealing with data that becomes executable.

References
- NSSecureCoding: Everything you need to know about NSSecureCoding
- Apple Developer Documentation: NSSecureCoding

47. An iOS user can lose their registration

Severity: Informational
Type: Session Management
Target: `AppState.swift`

Difficulty: Low
Finding ID: TOB-VOATZ-007

Description
The iOS client uses `identifierForVendor.uuidString` as a device ID for uniquely identifying their device. This identifier is guaranteed to change if the user deletes and subsequently reinstalls the Voatz app.

Exploit Scenario
Alice downloads the Voatz app and completes the onboarding process. She deletes and subsequently reinstalls the Voatz app some time after onboarding but before voting. When the app is reinstalled, she receives a new device ID. From the perspective of the Voatz Core Server API, it will appear as if she is connecting from a completely new device.

We were not able to test this scenario on a live Voatz instance because, at the time of its discovery, we did not have access to a dedicated API backend for testing. It may be the case that Voatz has a mitigation for this scenario; however, it is unclear whether this would involve a manual mitigation (*e.g.*, having to contact Voatz support for an account reset).

Recommendation
Voatz has indicated that this is intended behavior; any changes to a device ID should necessitate a re-registration.

Short term, document this behavior in the iOS application, and provide clear instructions for how users can remediate the problem if it occurs.

48. iOS client is susceptible to URI scheme hijacking

Severity: Informational
Type: Data Validation
Target: Voatz iOS Client

Difficulty: High
Finding ID: TOB-VOATZ-039

Description
The Voatz iOS client defines the `voatz://` URI scheme for receiving messages from other apps on the device. URI schemes can be hijacked by another app if the malicious app registers the same scheme and is also installed on the device. Consequently, a rogue app could receive messages sent via URI schemes intended for Voatz.

The severity of this finding is Informational, since it does not appear that Voatz is currently using messages sent via its URI scheme.

Exploit Scenario
A future refactor to Voatz makes use of its URI scheme to accept OAuth tokens or credentials sent via email or SMS. Alice creates a malicious app using the same `voatz://` URI scheme and coerces Bob to install it. When Bob receives his credential, Alice's app receives it instead of Voatz.

Recommendation
Short term, confirm that the `voatz://` URI scheme is not used for messaging, and document the code to ensure that it never shall be.

Long term, transition to “Universal Links” introduced in iOS 9. These allow apps to register web domains that are solely owned by the app.

References
- Apple Developer Documentation: Support Universal Links

A. Vulnerability Classifications

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices, or software
Cryptography	Related to protecting the privacy or integrity of data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing system failure
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Timing	Related to race conditions, locking, or order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth
Undetermined	The extent of the risk was not determined during this engagement
Low	The risk is relatively small or is not a risk the customer has indicated is important
Medium	Individual user's information is at risk, exploitation would be bad for client's reputation, moderate financial impact, possible legal

	implications for client
High	Large numbers of users, very bad for client's reputation, or serious legal or financial implications

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploit was not determined during this engagement
Low	Commonly exploited, public tools exist or can be scripted that exploit this flaw
Medium	Attackers must write an exploit, or need an in-depth knowledge of a complex system
High	The attacker must have privileged insider access to the system, may need to know extremely complex technical details, or must discover other weaknesses in order to exploit this issue

B. Review of Prior Security Assessments

Trail of Bits was given access to the redacted reports resulting from four of Voatz' prior security reviews.

The National Cybersecurity Center (NCC) publicly released a fifth assessment report in August 2019. NCC is a private entity, distinct from and unrelated to the Department of Homeland Security's National Cybersecurity and Communications Integration Center (NCCIC). NCC also has no relation to the *NCC Group*, an international information assurance firm founded seventeen years prior to NCC. NCC does not employ any technical security experts. Therefore, the content of the NCC report functions more like a user acceptance test, validating *not* that the system is *secure,* but rather that its features and operation meet the needs of the user.

During the course of Voatz' engagement with Trail of Bits, a sixth "black-box" assessment was independently performed by MIT researchers, focusing on the Voatz Android mobile client. The MIT assessment had neither sanction nor assistance from Voatz.

To the best of our knowledge, no assessment prior to ours has been scoped to include the discovery of Voatz core server and backend software vulnerabilities. Trail of Bits has performed the first system-wide, "white-box" assessment of Voatz. Also, while this report is intended for *both* technical *and* non-technical audiences, the previous technical reports provided summaries targeting only engineers.

The rest of this section discusses the chronology, methodology, findings, and results of the four previous technical assessments as well as the unsanctioned MIT assessment.

1. July 2018

In July 2018, REDACTED SECURITY VENDOR reviewed the security of the Voatz iOS and Android mobile applications. Their report does not indicate the level of effort for the project. This security review appears to have been conducted as a "black-box" audit, *i.e.*, without source code access. It appears that only the mobile clients were in scope, and the servers and cloud infrastructure were not.

Four low-severity configuration issues were identified, including issues with password policies, registration, brute force protections, and cookie settings.

2. October 2018

In October 2018, TLDR Security broadly reviewed the application, cloud, mobile, and corporate security of Voatz. Their report does not indicate the level of effort for the project.

The assessment provided high-level security hygiene. It was scoped to cover a shallow analysis of the system architecture (*e.g.*, cloud asset policies and configurations), manual analysis of user and data workflows, and threat mitigation planning (*e.g.*, access control policies).

Four high-severity issues were discovered in S3 bucket permissions, server monitoring, corporate device control, and password management; two medium-severity issues in DMARC controls and cloud configuration; and one low-severity issue related to denial-of-service mitigations. Their report also reviewed the Hyperledger smart contracts, and found three medium-severity issues in the TLS configuration, a dependence on mobile clients for security, limited peer diversity, and one low-severity issue related to code quality.

3. December 2018

In December 2018, ShiftState Security conducted a post-election security review of Voatz to determine whether best practices were followed during the 2018 General Election in West Virginia, when Voatz was used by 144 military personnel stationed overseas. Their report does not indicate the level of effort for the project. Application and source code review were not in scope for this assessment.

Issues with *ad hoc* and limited logging, unmanaged servers, and opportunities for denial of service were discovered during the review. It was also revealed that Zimperium's anti-mobile malware solution was not enabled during the pilot.

4. October 2019

In October 2019, the DHS Cybersecurity and Infrastructure Agency (CISA) conducted a one-week assessment of Voatz' servers and logs for signs of existing compromise. This included Voatz' internal network at their corporate headquarters and their cloud resources in both Amazon Web Services and Microsoft Azure. No source code was reviewed.

Issues with unmonitored PowerShell scripting, unmanaged local accounts, limited software control, *ad hoc* logging, and over-permissioned and weakly managed cloud accounts were discovered. No signs of existing compromise were found.

The MIT Report

On February 5th, 2020, Trail of Bits was given an anonymized, summary report of security issues in the Voatz Android mobile application externally reported to the DHS CISA. Six vulnerabilities were described, primarily related to the Android mobile application (version 1.1.60, circa September 24, 2019). Trail of Bits began verifying the issues and provided an initial evaluation confirming the presence of the described vulnerabilities to Voatz on February 11th.

On February 13th, Specter, Koppel, and Weitzner published *The Ballot is Busted Before the Blockchain: A Security Analysis of Voatz, the First Internet Voting Application Used in U.S. Federal Elections*, revealing themselves as the originators of the report to DHS. On the same day, Voatz released a blog post, *Voatz Response to Researchers' Flawed Report*, refuting the report. On the following day, February 14th, the MIT researchers released an FAQ about their paper.

Voatz presented three objections to the MIT report:

Objection 1

> *The researchers were analyzing an Android version of the Voatz mobile voting app that was at least 27 versions old at the time of their disclosure and not used in an election.*

The version of the app assessed by the MIT researchers was from late September 2019, approximately four months before they started their assessment. In our review, we did not identify any security relevant changes in the codebase between September 2019 and the code delivered at the start of this engagement other than: 1) minor changes to Zimperium; and 2) a minor change in the cryptographic handshake protocol. Neither change substantively affects MIT's claims.

Objection 2

> *As the researchers admitted, the outdated app was never connected to the Voatz servers, which are hosted on Amazon AWS and Microsoft Azure. This means that they were unable to register, unable to pass the layers of identity checks to impersonate a legitimate voter, unable to receive a legitimate ballot, and unable to submit any legitimate votes or change any voter data.*

This is correct—the modified client never connected to Voatz infrastructure. The MIT paper made no claims regarding registration, onboarding, ballot processing, and backend vote integrity.

Objection 3

> *In the absence of trying to access the Voatz servers, the researchers fabricated an imagined version of the Voatz servers, hypothesized how they worked, and then made assumptions about the interactions between the system components that are simply false. This flawed approach invalidates any claims about their ability to compromise the overall system. In short, to make claims about a backend server without any evidence or connection to the server negates any degree of credibility on behalf of the researchers.*

Developing a mock server in instances where connecting to a production server might result in legal action is a standard practice in vulnerability research. It is also a standard practice in software testing. The MIT findings are focused within the Android client and do not rely on intimate knowledge of the Voatz servers.

The remainder of this section outlines the primary claims covered in the MIT paper, whether we can confirm their existence, and what mitigations existed or have since been added to address the vulnerabilities.

B.1 Side-channel information leak

Claim: A passive observer can determine the ballot entries of a voter solely by the size of their encrypted vote submission message.

Status: Voatz claims that the clients have been modified to include padding before the ballot data is transmitted. However, we were unable to find this feature in the codebase. Padding does occur within the backend, however. It may be the case that it was added to clients in a feature branch that has not yet been merged into the development branch, and therefore was not provided to us.

Likelihood: Moderate. The ballot submission message data, in addition to the identifiers of the voter's ballot choices, also includes the ballot statements and descriptions for the choices. Ballot submissions will almost certainly vary in size in predictable ways depending on the voter's choices. An attacker exploiting this vulnerability must have control over a node in the network route between the voter and Voatz. Under normal circumstances, this is unlikely. However, Voatz is intended to be used by overseas voters in which network infrastructure is likely controlled by foreign governments.

Recommendation: Ensure that all vote submission messages are exactly the same size.

B.2 Voter disenfranchisement via network disruption

Claim: An active network participant (*e.g.*, one with control over any node in the route from the voter to the Voatz API server) can choose to drop a user's messages to the Voatz server. Moreover, the mechanism described in B.1 can be exploited to selectively drop only ballots that contain certain votes.

Status: Confirmed. There is no mechanism that would prevent this attack.

Likelihood: High. An attacker exploiting this vulnerability must have control over a node in the network route between the voter and Voatz. Under normal circumstances, this is unlikely. However, Voatz is intended to be used by overseas voters in which network infrastructure is likely controlled by foreign governments. If exploited, a voter would be aware that they were disenfranchised since they would not receive a ballot receipt (unless

this exploit is combined with an attack against the transport layer security of the system plus an attack against the underlying cryptographic protocol). However, an attacker with knowledge of voters' E-mail addresses can craft or copy a valid ballot and E-mail it to disenfranchised voters.

Recommendation: Stand up redundant API endpoints in different IP ranges and geographic regions to make it harder to exploit this vulnerability at scale. Devise a way for voters to independently verify the validity of their ballots.

B.3 On-device security circumvention

Claim: The libraries used for threat detection in the mobile clients can be disabled on rooted devices, allowing the clients to be run on unsupported devices as well as with modified versions of the client.

Status: Confirmed. We were able to build a version of the Android application with threat detection disabled. There does not appear to have been any additional mitigations added since version 1.1.60. See finding TOB-VOATZ-29.

Likelihood: Moderate. An adversary with sufficient resources could release a modified version of the app to the public (*e.g.,* through *ad hoc* distribution), or remotely modify a legitimately installed version of the app if they have root access to the device.

Recommendation: In general, there is no way to prevent modified clients from interacting with the system, or a sufficiently advanced adversary from reverse-engineering the communication protocol and writing a custom client. Ensure that the security of the Voatz protocol does not rely on the assumption that the official, unmodified clients are being run.

B.4 GUI modification and data exfiltration

Claim: On a rooted device, and with Zimperium disabled (B.3), it is trivial to change the user interface of the Voatz application to, for example, make the software vote for a candidate not chosen by the user.

Status: Confirmed. If an attacker has control of a rooted device, they can modify *any* application arbitrarily.

Likelihood: Moderate. Any vote modified by a malicious client would also be detectable by the voter given his or her receipt from the server.

Recommendation: Continue striving to employ state-of-the-art tamper detection technology. Educate voters that there is no foolproof way to secure a mobile device to prevent tampering.

B.5 PIN cracking

Claim: An attacker with access to the Voatz app's storage (*e.g.*, on a rooted device) can trivially compromise a user's Voatz PIN, even if the Voatz app is not running.

Status: Confirmed. See TOB-VOATZ-048.

Likelihood: High. Related vulnerabilities such as TOB-VOATZ-048 allow an attacker to compromise the Voatz database, revealing the user's voting history, and potentially allowing the attacker to vote on behalf of the user.

Recommendation: Allow and encourage users to enter a password with greater entropy. Allow and encourage users to provide a second factor of authentication necessary for each login. Store all sensitive information within the Android keystore.

B.6 Server compromise

Claim: The anonymous researchers who submitted the report to DHS speculate (but have no proof) that anyone with access to the API server can alter, expose, or discard any user's vote. They also observe that there is no evidence of any blockchain verification code in the client.

Status: Confirmed, on all accounts. However, in order to alter a vote that has already been cast, the attacker would also need to have control over the Hyperledger Fabric blockchain. The credentials for accessing the blockchain are stored on the API server. An attacker who can modify the software running in the API server can alter, expose, or discard any user's vote. The clients do not interact with the blockchain directly, so there is no blockchain verification code in the client.

Likelihood: The API server presents the largest target for a sufficiently advanced adversary, such as a nation-state. The API server is a single target that would allow the attacker to affect all votes in an election. However, there are several other targets that could wreak havoc on an election. For example, an attacker with control over Voatz' consumer cloud file hosting provider or the audit portal server could inject false data to call the legitimacy of the vote into question.

Recommendation: Provide a cryptographic means for users to sign their own ballots in the client and subsequently, independently verify their digitally signed ballots have been recorded on the blockchain—*without* ever giving the backend access to the voter's signing credentials.

C. Insufficient validation of encrypted API requests

This appendix serves as a glossary of encrypted API request types, and the observed consequences of insufficient data validation. This not only includes instances when the device ID provided in the outer encapsulating packet is different from any specified in the inner request data (as in TOB-VOATZ-014), but also fields that are wholly controlled by the client. We have prioritized these findings based upon the potential for undefined and potentially malicious behavior as a result of fields that are accepted from the client without validation.

The table below illustrates the priority in which each "`ApiEncrypted`"-prefixed request type should be reviewed for relevant fixes:

- **High:** Requests can be sent with modified data to produce some significant result.
- **Moderate:** Data validation does not exist or is ineffective, but this omission was not exploitable in a meaningful way, or not fully assessed.
- **None:** This request type was not observed to include client-specified fields that lack validation.
- **Unknown:** This request type was not reviewed. Any other impact indicates at least a partial review.

Definition	Priority	Notes
`ApiEncryptedCustomerOidProfileCreateRequest`	High	Impact not fully assessed, although one can bypass session validation by setting the outer device ID to one that matches the session cookie. This appears to allow spoofing of the inner device ID sent to the later transactions, among other fields.
`ApiEncryptedPerformOrgIdvRequest`	High	Impact not fully assessed, although one can bypass session validation by setting the outer device ID to the one that matches the session cookie. This appears to allow spoofing of the inner device ID sent to the later transactions, among other fields.
`ApiEncryptedAnonCustomerCreateRequest`	High	Can spoof various fields (*e.g.,* spoofing `deviceProfile` to create an anonymous customer with an arbitrary device ID).
`ApiEncryptedCustomerAuthenticateRequest`	High	If `customerId` matches the outer device ID, one can supply a "bad" inner device ID which will get passed to the authentication transaction.

`ApiEncryptedVoteAsTreeRequest`	High	If an outer device ID is associated with a session cookie and inner `customerId`, other fields can be spoofed in the later transaction.
`ApiEncryptedVoteAsStringRequest`	High	If an outer device ID is associated with a session cookie and inner device ID, other fields can be spoofed in the later transaction.
`ApiEncryptedCustomerLogoutRequest`	High	If `customerId` matches the outer device ID, one can supply a "bad" inner device ID which will get passed to the logout transaction.
`ApiEncryptedThreatDetectedRequest`	High	Does not validate inner device ID to outer device ID. An attacker can pass an arbitrary inner device ID they want to report a threat on, and ban from authentication.
`ApiEncryptedCustomerPreRegisterRequest`	Moderate	Inner device ID is not validated against outer device ID. Further impact not determined.
`ApiEncryptedCustomerVerifyOtpRequest`	Moderate	Can re-encrypt this request for another user, and additional data validation can likely be performed against the `preRegisterId` or other fields.
`ApiEncryptedCustomerCompleteProvisioningRequest`	Moderate	Outer device ID is not validated against inner `customerId`. This API endpoint requires further investigation, as it may allow an attacker to provision another user's account with an incorrect mobile number.
`ApiEncryptedCustomerReregisterRequest`	Moderate	Inner device ID is not validated against outer device ID. Other fields can be spoofed, which may lead to interesting behaviour. See TOB-VOATZ-022 for additional impact analysis.
`ApiEncryptedCustomerReverifyOtpRequest`	Moderate	Can re-encrypt this request for another user; additional data validation could be performed here against `preRegisterId` or other fields.
`ApiEncryptedControlNumberIssueRequest`	Unknown	
`ApiEncryptedControlNumberReissueRequest`	Unknown	
`ApiEncryptedGetVoteCountVMARequest`	Unknown	

`ApiEncryptedGetVoteCountVMARequest`	None	Not useful; session cookie needs to correspond to the phone number you request.
`ApiEncryptedMessageListLast10ByCategory`	None	Not useful; session cookie needs to correspond to the phone number you request.
`ApiEncryptedEventListByOrganizationRequest`	None	Session cookie is used to pull `customerId`, which checks privileges.
`ApiEncryptedCustomerOidProfileGetByCustomerIdRequest`	None	Can bypass session cookie auth using outer device ID, but inner device ID needs to be associated with the outer device ID.
`ApiEncryptedCustomerVoteRequest`	None	`customerId` is validated against session cookie. Only the vote data is encrypted.
`ApiEncryptedCustomerGetRequest`	None	Can bypass session cookie auth using outer device ID, but inner `customerId` needs to be associated with the outer device ID. `customerId` is the only target field here.
`ApiEncryptedGetNetVerifyCredentials`	None	device ID is the only field here.
`ApiEncryptedAndroidLvlGetNonceRequest`	None	Validates inner/outer device IDs in the `isMaxConsecutiveAndroidLvlNonceReqValid` call.
`ApiEncryptedAndroidLvlCheckResponseRequest`	None	Validates inner/outer device ID in the `isMaxConsecutiveAndroidLvlCheckReqValid` call.
`ApiEncryptedGetIapInfoRequest`	None	Validates inner/outer device ID in the `isMaxConsecutiveZiapConfigReqValid` call.
`ApiEncryptedGetVoteMetadataRequest`	None	Uses device ID on outer packet to validate session and obtain the corresponding `customerId` to act on.
`ApiEncryptedEventListByCustomerRequest`	None	One can bypass session validation by setting the outer device ID to the one that matches the session cookie, and using a related `customerId`, but there are no other privileged fields.
`ApiEncryptedCustomerUpdateRequest`	None	If `customerId` matches the outer device ID, and date-of-birth and other related checks can be passed, it appears one can modify other fields, but nothing otherwise privileged.

`ApiEncryptedCustomer UpdateWithIdvRequest`	None	If `customerId` matches the outer device ID, and date-of-birth and other related checks can be passed, it appears you can modify other fields, but nothing otherwise privileged.
`ApiEncryptedCustomer GetBasicIdvStatus Request`	None	Appropriately checks that the `customerId` (the only target field) is associated with the outer device ID.
`ApiEncryptedGetReceipt CodeRequest`	None	Appropriately checks that the `customerId` (the only target field) is associated with the session cookie for the outer device ID.
`ApiEncryptedGet LegislatorUrlRequest`	None	Appropriately checks that the `customerId` (the only target field) is associated with the session cookie for the outer device ID.
`ApiEncryptedSubmitVote ImageRequest`	None	Appropriately checks that the `customerId` (the only target field) is associated with the session cookie for the outer device ID.
`ApiEncryptedGetOrg AffiliationsRequest`	None	Appropriately checks that the `customerId` (the only target field) is associated with the session cookie for the outer device ID.
`ApiEncryptedGetVoted EventIdsRequest`	None	Appropriately checks that the `customerId` (the only target field) is associated with the session cookie for the outer device ID.
`ApiEncryptedGetAll DemographicsRequest`	None	Does not contain sensitive fields that require privileged access.
`ApiEncryptedOrgIdv ProfileGetRequest`	None	Does not contain sensitive fields that require privileged access.
`ApiEncryptedGet DatafileMetaData Request`	None	Does not contain sensitive fields that require privileged access.
`ApiEncryptedGetResults ForEventList`	None	Does not contain sensitive fields that require privileged access.

D. Verifiability and Voatz

This appendix describes several notions of "verifiability" from the e-voting literature and how they apply (or do not apply) to the Voatz system.

End-to-end verifiability

The first notion of verifiability is from *Securing the Vote: Protecting American Democracy*, published by the National Academy of Science, Engineering, and Medicine. In this work, blockchain voting is discussed as a possibility for future voting. The authors discuss End-to-End Verifiable (E2E-V) systems as necessary for blockchain voting. They cite a 2015 report from the U.S. Vote Foundation stating that any electronic voting system must be E2E-V, and echo its claims.

E2E-V systems allow voters to cast encrypted ballots such that ballot counts are verifiable to anyone, but individual voters' preferences are not revealed. Additionally, all voters should be able to verify their ballot was counted correctly, and the system tabulating votes must be transparent and publicly available. The authors also note that simply because a system is E2E-V, it is not necessarily secure or suitable for use.

Voatz is not E2E-V. Ballots do not protect voter identities, as they are identified by voters' device IDs (see TOB-VOATZ-019). On most mobile phones, these IDs are available to any app running on the voter's phone and are known to be collected *en masse* by advertising companies. Ballots can be de-anonymized with access to the Voatz backend databases, blockchain, and logs. Votes are simply base64-encoded, not strongly encrypted. Also, vote totals, tabulation software, and auditing capabilities are not publicly available.

In *Securing the Vote*, the issue of coercion resistance is also discussed. Ideally, vote tabulation software would allow users to verify their vote was *counted correctly* without offering a receipt listing whom they specifically voted for. This is because such receipts could easily be used to coerce voters to vote for a particular candidate. Voatz is not coercion-resistant, as voters receive an explicit receipt showing which candidates they voted for.

Verifiability notions for e-voting protocols

In 2016, Cortier, *et al.,* published a systemization of knowledge on verifiability in e-voting. They identified three primary definitions of verifiability: individual verifiability, universal verifiability, and eligibility verifiability. They claim that viable e-voting systems must satisfy all three properties.

Individual verifiability refers to the ability of individuals to check that their vote was tabulated. As mentioned above, it may be preferable that individuals can only check that their vote was *counted* without an explicit receipt. Voatz satisfies the tabulation property, albeit by providing an explicit receipt.

Universal verifiability refers to the ability of any observer to verify that the outcome of the election is in accordance with all submitted votes. Voatz again satisfies this property, as choice IDs are simply base64-encoded, so an auditor can easily decode them and ensure the sum is as advertised. Although only auditors have access to this information, this is similar to a traditional election setup.

Eligibility verifiability refers to the ability of any observer to check that only eligible individuals cast votes in a given election. Voatz does not provide any such guarantees, and while some *ad hoc* verification could be performed with records of Jumio calls, no formal system is in place to ensure this property holds. Again, only auditors have access to the most relevant information. As a corollary, in Voatz it is impossible to verify whether any individual voters voted more than once in a given election.

E. Fix Log

Trail of Bits performed a retest of the Voatz system on February 27, 2020. Voatz provided fixes and supporting documentation for the findings in this security assessment report. Each finding was re-examined and verified by Trail of Bits.

Some of Voatz' modifications were made in feature branches that were not merged into master branch, likely due to time limitations. The fixes were not tested for functional correctness, and we could not verify the fixes in any deployed system.

Voatz addressed eight (8) issues and partially addressed six (6) issues. The remaining thirty-four (34) issues either remain unfixed or their fixes were not verifiable by Trail of Bits.

Finding status

#	Title	Severity	Status
1	Device IDs not validated against inner request device IDs	High	Not Fixed
2	Amazon admin password is hardcoded in source file	High	Partial Fix
3	Non-anonymous ballot receipts are encrypted with AES-CBC using hardcoded key and IV	High	Fixed
4	Secrets are stored in environment variables sourced from bash script	High	Not Fixed
5	API for the onboarding workflow prohibits partitioning cloud resources for concurrent elections	High	Not Fixed
6	Receipt and affidavit filename collisions	High	Not Fixed
7	A voter can unregister another voter's device	High	Fixed
8	Input keying material for AES GCM encoding is sent to Graylog	High	Fixed

9	Voatz backend SSL key has a subdomain wildcard	High	Not Fixed
10	Clients can specify their own audit token	High	Not Fixed
11	Test parameters in the registration APIs can bypass SMS verification	High	Not Fixed
12	QR code receipt generation will fail for large non-anonymous ballots	Medium	Fixed
13	Session token validation ignores idle timeout	Medium	Not Fixed
14	Receipt encryption is weak and can leak confidential information	Medium	Not Fixed
15	Insufficient device ID validation on backend	Medium	Not Fixed
16	Potential resource exhaustion via logging/storage of unsanitized data	Medium	Not Fixed
17	Resource exhaustion via specially-crafted Zimperium threats	Medium	Partial Fix
18	Zimperium checks on the backend are a blacklist, not a whitelist	Medium	Partial Fix
19	AES-GCM key/nonce/tag encryption system breaks authenticity	Medium	Partial Fix
20	Unauthenticated ECDH is vVulnerable to key compromise impersonation	Medium	Not Fixed
21	AES-GCM keys, nonces, and “tag”s are encrypted using AES-ECB	Medium	Fixed
22	Voatz API server lacks OCSP stapling	Medium	Not Fixed
23	Empty ballots are not recorded in Hyperledger	Low	Not Fixed

24	Database root credentials stored in git	Undetermined	Not Fixed
25	Signed voter affidavits are sent to an administrative email	Undetermined	Not Fixed
26	AES-GCM AAD usage is non-standard	Undetermined	Not Fixed
27	Session cookie expiration offset is a hardcoded literal	Informational	Not Fixed
28	Encrypted application data is trivially brute forceable	High	Not Fixed
29	PBDKF2 provides insufficient security margin for PIN codes	High	Partial Fix
30	Third-party apps can capture the Android client screen and read screenshots taken from the client	High	Fixed
31	Android release build signing key password and keystore password stored in git	High	Not Fixed
32	A malicious website can read from the Android client's internal storage	High	Fixed
33	Insufficient Android device ID construction	Low	Partial Fix
34	Android client does not use the SafetyNet Attestation API	Low	Not Fixed
35	Android client does not use the SafetyNet Verify Apps API	Low	Not Fixed
36	Certificate pinning is only configured for the main Voatz domain	Low	Not Fixed
37	No explicit verification of the Android Security Provider	Low	Not Fixed
38	Jumio Netverify API credentials stored in git	Undetermined	Not Fixed

39	Google Services API key stored in git	Undetermined	Not Fixed
40	A malicious website may be able to execute JavaScript within the Android client	Informational	Fixed
41	The iOS client does not disable custom keyboards	Medium	Not Fixed
42	The iOS client does not use system-managed login input fields	Low	Not Fixed
43	iOS client keychain items are not excluded from iCloud and iTunes backups	Low	Not Fixed
44	Cryptographic credentials are not generated in the iOS secure enclave	Low	Not Fixed
45	iOS client disables Apple Transport Security (ATS)	Undetermined	Not Fixed
46	iOS client is vulnerable to object substitution attacks	Undetermined	Not Fixed
47	An iOS user can lose their registration	Informational	Not Fixed
48	iOS client is susceptible to URI scheme hijacking	Informational	Not Fixed

Detailed fix log

This section includes brief descriptions of fixes implemented by Voatz after the end of this assessment that Trail of Bits was able to review.

TOB-VOATZ-006: Non-anonymous ballot receipts are encrypted with AES-CBC using hardcoded key and IV (High)
Fixed. The code related to non-anonymous events will no longer be used and has been removed from the codebase.

TOB-VOATZ-008: Insufficient Android device ID construction (Low)
Partial fix. Android device IDs now default to a UUID if the version returned from the OS is `null`. This approach does not prevent a user from being unregistered on device reset, however, Voatz has indicated that this is intended behavior.

TOB-VOATZ-009: QR code receipt generation will fail for large non-anonymous ballots (Medium)
Fixed. The code related to non-anonymous events will no longer be used and has been removed from the codebase.

TOB-VOATZ-011: AES-GCM keys, nonces, and "tag"s are encrypted using AES-ECB (Medium)
Fixed. AES-ECB has been replaced with AES-GCM.

TOB-VOATZ-017: Amazon admin password is hardcoded in source file (High)
Partial fix. The Amazon test OTP settings have been removed from `AmazonTestOtpUtility.scala`. However, they were moved from the codebase to MongoDB rather than to a secure location such as a secret vault or hardware security module.

This refactor also contributed to a new finding, TOB-VOATZ-047, that can allow an attacker to bypass SMS verification during pre- and re-registration.

Finally, the password was removed from the HEAD of the development branch. However, in the mirror of the git repository provided to Trail of Bits, the credentials still exist in the git history. If not already performed upstream, the git repository should be rebased to remove the credentials from the history. Trail of Bits has no way to independently confirm whether the admin password has been rotated.

TOB-VOATZ-022: A voter can unregister another voter's device (High)
Fixed. `CustomerMongoDaoAsync.scala` was modified on February 21 in git commit ce70626 to always require a user to authenticate via email during re-registration.

However, the modification to `CustomerMongoDaoAsync.scala` to accommodate this refactor resulted in a new finding, TOB-VOATZ-047, that can allow an attacker to bypass SMS verification during pre- and re-registration.

TOB-VOATZ-023: Input keying material for AES-GCM encoding is sent to Graylog (High)
Fixed. This has been removed from `AesGcmEncoding.scala`.

TOB-VOATZ-024: AES-GCM key/nonce/tag encryption system breaks authenticity (Medium)
Partial fix. The AES-GCM key, nonce, and tag are now encrypted using AES-GCM, both within the Core Server and Android client. However, source code updates for other system components were not provided, so we cannot confirm a fix for the entire system.

TOB-VOATZ-025: PBDKF2 provides insufficient security margin for PIN codes (High)
Partial fix. PBKDF2 was reconfigured to use 10,000 iterations. However, given the low entropy of the PIN codes, this is still insufficient. We recommend performing an experiment to determine the highest iteration count that is computationally feasible in the system. Ultimately, we recommend transitioning to a modern KDF like Argon2id or scrypt.

TOB-VOATZ-029: Zimperium checks on the backend are a blacklist, not a whitelist (Medium)
Partial fix. `CustomerMongoDaoAsync.scala` was modified on February 11 in git commit 1430323 to include a call to the Zimperium API during new user registration. This ensures that the user's device has attested to Zimperium at least once. However, this does not prevent an attacker from running an unmodified version of Voatz once to attest to Zimperium, and then proceeding to register with a modified version of Voatz with Zimperium disabled.

TOB-VOATZ-030: Resource exhaustion via specially-crafted Zimperium threats (Medium)
Partial fix. `CustomerMongoDaoAsync.scala` was modified on February 9 in git commit 58afe9d to include verification of threats. However, although the `threatId` check is case-insensitive, when retrieving a snapshot from the `threatId` the comparison appears to be case-sensitive. If this is correct, an attacker can supply different case variants for `threatId` and achieve duplicate requests. The API endpoint also still lacks verification of the inner *vs.* outer request layer device IDs. This allows an attacker to send repeated requests with different device IDds to achieve a similar resource exhaustion to enumerating `threatIds`. See finding TOB-VOATZ-014.

TOB-VOATZ-032: Third-party apps can capture the Android client screen and read screenshots (High)
Fixed. FLAG_SECURE was set on relevant Android windows.

TOB-VOATZ-035: A malicious website can read from the Android client's internal storage (High)
Fixed. File access was disabled from the Voatz Android client's WebView.

TOB-VOATZ-036: A malicious website may be able to execute JavaScript (Informational)
Fixed. JavaScript support was explicitly disabled in the Voatz Android client's WebView.

Unaddressed findings and unverified fixes

This section includes brief descriptions of:

1. findings that remain unfixed;
2. findings whose risk has been accepted by Voatz; and/or
3. fixes that cannot be independently verified by Trail of Bits (*e.g.*, due to a lack of access to code or infrastructure updates).

On March 11th, we received additional information concerning Voatz's plans to address these issues in the future. We reproduce that information here. All responses from Voatz are included as italicized quotes. Where applicable, we provide justification for instances in which independent verification was impossible.

TOB-VOATZ-001: Jumio Netverify API credentials stored in git (Undetermined)

> *These were a few years old and have since been rotated and are no longer stored in git.*

The API credentials were removed from the `HEAD` of the development branch. However, in the mirror of the git repository provided to Trail of Bits, the credentials still exist in the git history. If not already performed upstream, the git repository should be rebased to remove the credentials from the history. Trail of Bits has no way to independently confirm whether the credentials have been rotated.

TOB-VOATZ-002: Google Services API key stored in git (Undetermined)

> *Voatz accepts the risk presented here and believes the frequent key rotation, other controls in place provide sufficient safeguards in the short term.*

TOB-VOATZ-003: Android release build signing key password and keystore password stored in git (High)

> *Voatz accepts the risk presented here and believes the frequent key rotation, other controls in place provide sufficient safeguards in the short term.*

TOB-VOATZ-004: Unauthenticated ECDH is Vulnerable to key compromise impersonation (Medium)

> *This is being addressed as part of the noise protocol implementation in an upcoming release.*

TOB-VOATZ-005: Session cookie expiration offset is a hardcoded literal (Informational)

Voatz accepts the risk here and plans to re-evaluate this in a future release.

TOB-VOATZ-008: Insufficient Android device ID construction (Low)

The reregistration upon device reset is a mandatory part of the user workflow and users are advised about the same via help messages, other tutorials.

We were unable to independently verify the existence of modified help messages or tutorials that indicate this behavior.

TOB-VOATZ-010: API for the onboarding workflow prohibits partitioning cloud resources for concurrent elections (High)

Given the nature of the current election pilots, Voatz is comfortable with the current approach and plans to address scalability challenges as part of the version 2 of its platform.

TOB-VOATZ-012: AES-GCM AAD usage is nonstandard (Undetermined)

Voatz is comfortable with its AAD usage and believes that it aids its security protocols.

TOB-VOATZ-013: Secrets are stored in environment variables sourced from bash script (High)

Voatz accepts the risk presented here and believes that its internal controls provide sufficient safeguards to prevent misuse.

TOB-VOATZ-014: Device IDs not validated against inner request device IDs (High)

For endpoints that are sensitive, Voatz believes that the additional session level checks in place in the code provide sufficient protection against such a threat. For endpoints that are not sensitive, Voatz accepts the risk presented here and believes its layered security protocols will detect misuse.

TOB-VOATZ-015: Receipt encryption is weak and can leak confidential information (Medium)

Voatz is working on addressing this as part of its upcoming release.

TOB-VOATZ-016: Database root credentials stored in git (Undetermined)

These were old local test credentials from a few years ago and are no longer used.

The database credentials were removed from the `HEAD` of the development branch. However, in the mirror of the git repository provided to Trail of Bits, the credentials still exist in the git history. If not already performed upstream, the git repository should be rebased to remove the credentials from the history. Trail of Bits has no way to independently confirm whether the credentials are still used.

TOB-VOATZ-018: Session token validation ignores idle timeout (Medium)

> *A memcached session entry is ejected when the TTL or TimeToLive expires. This setting is configurable server side and is used to control the duration of a valid session. This renders the MaxIdleTime setting superfluous and it is for this reason the code reading the MaxIdleTime setting was commented out.*

This will likely mitigate the issue. However, Trail of Bits was not furnished with a copy of the memcached configuration file and can therefore not confirm that it is configured in this way.

TOB-VOATZ-019: Insufficient device ID validation on backend (Medium)

> *Voatz is working on adding good-form validation as part of its upcoming release.*

TOB-VOATZ-020: Receipt and affidavit filename collisions (High)

> *This is mitigated by ensuring single-use audit tokens that cannot be reused again.*

The code provided to Trail of Bits does not appear to have any reference to “single-use audit tokens”. In the assessed version of the code, the backend does not verify audit tokens on ballot submission.

TOB-VOATZ-021: Signed voter affidavits are sent to an administrative email (Undetermined)

> *This finding is not relevant. Firstly, this is required per the legal guidelines of the election jurisdictions. See sample affidavit for reference. Secondly, the destination email is provided by the jurisdiction. The email address in your snippet is just a placeholder. Thirdly, our pilot jurisdictions already allow eligible absentee voters to return ballots via email or efax (~remember Voatz is an additional method that is being piloted) and have their own practices, procedures in terms of handling spam, etc. Lastly, Voatz servers send these emails from a whitelisted email address using a dedicated IP address and using a service that is protected using DMARC, DKIM, SPF.*

TOB-VOATZ-025: PBDKF2 provides insufficient security margin for PIN codes (High)

This has been partially addressed. The remainder is being addressed as part of the updates for TOB-VOATZ-048.

TOB-VOATZ-026: Certificate pinning is only configured for the main Voatz domain (Low)

Voatz accepts the risk here and is addressing this as part of an upcoming release.

TOB-VOATZ-027: Empty ballots are not recorded in Hyperledger (Low)

Voatz has addressed this in the audit documentation.

Trail of Bits was not furnished with the updated audit documentation.

TOB-VOATZ-028: Voatz backend SSL key has a subdomain wildcard (High)

Voatz accepts the risk presented here and plans to re-evaluate this at a later time.

TOB-VOATZ-029: Zimperium checks on the backend are a blacklist, not a whitelist (Medium)

Voatz believes that its 3-way off channel check will detect attempts to bypass Zimperium as such a check is not visible to an attacker.

Zimperium attestation checks added after the assessment are only during registration and re-registration. We did not see any code that would detect a client that had removed Zimperium after registration.

TOB-VOATZ-030: Resource exhaustion via specially-crafted Zimperium threats (Medium)

Voatz accepts the risk presented here and believes its layered security protocols will detect this early and stop the misuse.

TOB-VOATZ-033: Voatz API server lacks OCSP stapling (Medium)

This has been enabled on the relevant Voatz servers.

Trail of Bits was not furnished with a list of servers that have been updated, and therefore cannot independently verify that they have been updated. Also, while iOS supports OCSP stapling by default, the Android client will need to be updated to support it.

TOB-VOATZ-034 (No explicit verification of Android security provider)

This is addressed via the Zimperium integration.

Trail of Bits could not independently verify that Zimperium's proprietary anti-tamper checks explicitly verify the Android security provider. We recommend an additional check be included in the event that Zimperium is ever disabled (*e.g.*, as occurred during the 2018 West Virginia pilot election), intentionally or unintentionally.

TOB-VOATZ-037: Android client does not use the SafetyNet Attestation API (Low)

This is being added as part of the upcoming release.

TOB-VOATZ-040: The iOS client does not disable custom keyboards (Medium)

Voatz accepts the risk here and plans to address this in an upcoming release.

TOB-VOATZ-045: Android client does not use the SafetyNet Verify Apps API (Low)

This is being added as part of the upcoming release.

TOB-VOATZ-046: Clients can specify their own audit token (High)

Voatz accepts the risk presented here and believes the other controls in place (such as single use audit tokens) in the system provide sufficient safeguards to prevent misuse.

As is discussed in the response to TOB-VOATZ-020 above, the code provided to Trail of Bits does not appear to have any reference to "single-use audit tokens".

TOB-VOATZ-047: Test parameters in the registration APIs can bypass SMS verification (High)

This test code has been removed from the repository.

This change would fully fix the issue. However, these changes do not appear to have been pushed to the git repository to which Trail of Bits was given access. Therefore, we cannot independently confirm that the fix is correct.

TOB-VOATZ-048: Encrypted application data is trivially brute-forceable (High)

Voatz is enhancing this functionality as part of its upcoming release. The salt is actually stored in an encrypted shared preferences file so there is partial mitigation in place already.

We observed that the salt is stored in the app's shared preferences, which are not encrypted at rest.