OVERVIEW

This section will seek to detail voting protocols deployed in actual elections, namely iVote protocol which was in March 2015 for the state election for New South Wales, Australia (Halderman & Teague, 2015).. Additionally, Voatz protocol will be discussed, which was used in 2018 and in state, federal as well as municipal elections in West Virginia, Utah, Denver and Oregon, and used in 2016 Utah Republican Convention and Massachusetts Democratic Convention (Specter et al., 2020). Furthermore, vulnerabilities and attacks on said protocols will be explained including, depending on the attack, its corresponding threat actor (e.g., malicious voter, malicious government). It is also important to understand that due to the sensitive nature of the data being used in the voting applications, the vulnerabilities discovered and discussed are limited to public data and replications of the original voting application.

VULNERABILITIES OF PROTOCOLS

Vulnerability of iVote Protocol

The protocol iVote as previously mentioned was used in state election in New South Wales, Australia and accounted for 5% of total votes for the entire election and was developed by Scytl partnered with New South Wales Electoral Commission (NSWEC) (Halderman & Teague, 2015). Security experiments and analysis were executed by a team of researchers named, J. Alex Halderman and Vanessa Teague from University of Michigan and University of Melbourne, respectively. They discovered two downgrade-to-export attacks that the iVote protocol is vulnerable to, namely FREAK attack (Factoring RSA Export Keys) and Logjam attack.

FREAK Attack

The scope of their experiments was limited to publicly available data including client-side HTML, CSS and JavaScript and due them invalid voters they were limited to the login page https://cvs.ivote.nsw.gov.au and the test website, https://practise.ivote.nsw.gov.au/ that the public could practice voting on. Also, both websites were inspected and had virtually the same client-side code. For secure web distribution, iVote, employs HTTPS. The SSL Test performed by Qualys SSL Labs certified that the principal iVote HTTPS server was secure against known vulnerabilities. An external web server on the other hand, https://ivote.piwikpro.com which also imports and executes JavaScript from the Piwik tool when iVote is loaded, was discovered to be insufficient in SSL settings, obtaining a 'F' grade. The server also supports 512-bit RSA and ephemeral Diffie-Hellman key exchange ciphersuites (Halderman & Teague, 2015).

The FREAK attack, which stands for Factoring RSA Export Keys, is a TLS vulnerability that was made public on March 3, 2015, less than two weeks before the election. Due to setup issues on the Piwik server, it was vulnerable to FREAK, and a network-based man-in-the-middle attacker (Beurdouche et al., 2017). FREAK exploits the flaw in export-grades of 512-bit RSA keys supplied by the TLS protocol, which is a legacy feature of 1990s-era US cryptography export limitations. An attacker could trick browsers into employing export-grade RSA (which has reduced cryptographic entropy), get the RSA private key by factoring the 512-bit public key, and modify the contents of the connection if a server supported it, which Piwik does. The attacker initiates the attack by intercepting the browser's TLS CLIENT HELLO message and sending a substitute message to the server pretending that the browser wants to use

export-grade RSA. In export-grade RSA modes, the server issues a temporary 512-bit RSA public key to the client and signs it with a node chosen by the client using the public key from the normal X.509 certificate. The client validates the certificate chain from the server's X.509 certificate to a trusted root certificate authority, then uses the temporary RSA key to encrypt session key information that will be used to safeguard the connection for the lifetime of the connection. The attacker's main objective in establishing a connection is to convince a voter's browser that they are ivote.piwikpro.com by utilizing the server's signature and an RSA public key (Halderman & Teague, 2015). Nadia Heninger demonstrated how to factor 512-bit RSA keys in 7 hours for under $100 using open-source tools and Amazon EC2 (Valenta et al., 2016).

After factoring the key, the attacker can intercept the user's connection, note the nonce of the client, and issue a request to the legitimate Piwik server with the same nonce, posing as a signature oracle. The Piwik server changes its temporary key every hour, making factoring the key difficult. However, by maintaining a long-lived TLS connection and repeatedly invoking client-initiated renegotiation, the server may be forced to utilize the same temporary RSA key for extended periods of time (Halderman & Teague, 2015). Several browsers namely Internet Explorer, Safari, and Chrome for Mac OS and Android were vulnerable to this attack until a patch was published March 10 and voting on iVote commenced March 16 so many users might still be browsing on vulnerable browser versions (Z. Durumeric et al., 2015).

Logjam Attack

All widely used browsers were vulnerable to the downgrade-to-export Logjam attack, which was launched against the ivote.piwikpro.com server. This attack targets ephemeral Diffie Hellman (DHE) ciphersuites and is enabled by a vulnerability in the TLS protocol rather than a client-side implementation issue . A man-in-the-middle attacker can compel browsers to utilize export-grade Diffie-Hellman with parameters that an attacker can break, gain session keys, and intercept or unilaterally modify the contents of the connection if the server supports it. The Logjam attack used open-source software to complete the precomputation step for three more popular 512-bit values of p, each occupying about a week of wall-clock time by using idle cycles on a cluster (Adrian et al., 2015).

After precomputation, the researchers were able to break individual key exchanges based on those values in roughly 90 seconds using a single 24-core machine. The same sort of attack would be possible against Piwik's system, allowing them to attack all iVote sessions from any browser for a set up-front cost for the precomputation. Another flaw was that the NSW web server delivered the iVote application using a secure HTTPS configuration, which then loaded extra JavaScript from an unsecured external server, ivote.piwikpro.com. An attacker who intercepted communications between the voter's browser and the PiwikPro server may alter this JavaScript, allowing them to insert arbitrary malicious code into the iVote application. A proof-of-concept demonstration was created to show how an attacker may control the iVote system by exploiting the FREAK or Logjam vulnerabilities. The attack made use of vulnerabilities in the Piwik server to substitute code loaded from ivote.piwikpro.com with malicious JavaScript that might alter the functioning of the iVote web application at will. The malicious code was introduced into critical components of the iVote client code, which used AngularJS to execute worker JavaScript threads that performed cryptographic operations. As these signals were delivered to the worker script that performed the encryption, the malware intercepted them and changed the intended vote to a different one. This altered the vote transmitted to the iVote server, exposing the voter's

intended vote and authentication credentials to the attacker's command-and-control server (Halderman & Teague, 2015).

Vulnerability & Attacks of Voatz Protocol

As previously mentioned, the Voatz protocol was used in 2018 for state, federal as well as municipal elections in multiple states and again in elections in 2016. But it is not without Its vulnerabilities. A team of researchers from Massachusetts Institute of Technology (MIT), namely Michael A. Specter, James Koppel and Daniel Weitzner ran a security analysis limited to a cleanroom environment and a replica of the original Voatz application and its infrastructure so not to affect actual Voatz server. They discussed that the vulnerabilities could be perpetrated by 3 types of adversaries:

1. An attacker that is controlling a user's device,
2. An attacker controlling Voatz's API server,
3. An attacker that can intercept network activity between the voter's device and the API server but has no further access.

(Specter et al., 2020)

An attacker with root capabilities can deactivate Voatz's host-based safeguards, allowing them to steal the user's vote, disclose her secret ballot, and exfiltrate her PIN and other authentication data. The Zimperium SDK, which is included with Voatz, is configured to detect debugging and other efforts to change the app, as well as collect intelligence on any malware it discovers. It would have recognized the researcher’s security analysis, blocked the app from functioning normally, and notified the API server of said activities by default. But a bypass can be accomplished by altering Zimperium's entry points to prevent the SDK from running. The Xposed Framework's hooking tools enabled the researcher to reroute control flow (Specter et al., 2020).It is also important to note that this was accomplished provided there is no out-of-band contact between Zimperium and Voatz, this assertion is corroborated as there is no evidence of this service in either Zimperium's documentation and the study of the app. The user's PIN and other login details are not saved in secure storage and instead flow through the application's memory, allowing a remote attacker to impersonate the user to Voatz's servers directly, even if the device is not connected to the internet. An attacker with root access to the device can take the PIN and the rest of Voatz's authentication credentials anonymously (Specter et al., 2020).

The researchers created a program that intercepts and logs every communication between the device and the server prior to encryption with SK_{aes} (which is a 256-bit symmetric key used in the TSL handshake between Voatz server and application) and data encryption and storage in the local database. This allows the researchers to view the user's raw PIN as well as other authentication data in plaintext. Offline attacks can exploit Voatz's database, as it only requires the user's PIN to unlock limited to exactly 8 numeric characters with unlimited attempt as inputting the PIN. This implies only 100,000,000 possible PINs. Additionally, Voatz does not allow PINs with three consecutive digits, removing 5% of PINs before beginning the exploit. This attack would result in quickly relearning the PIN, retrieving the user's PIN, login information, and vote history all at once. The researchers constructed a prototype of this technique and demonstrated that an attacker can brute force the key on a 3.1GHz 2017 MacBook Pro in about two days (Specter et al., 2020).

An attacker can employ a stealth UI modification attack to change the application to submit a desired vote while still displaying the same UI as if the app logged the user's contribution. This is due to a vulnerability of the application feature that allows for voter spoiling which enables a voter to cast a new vote that invalidates all prior votes (Specter et al., 2020). A variant of this attack affected the Estonia e-voting system with similar results (Springall et al., 2014).

The protocol is also vulnerable to server attacks, as the analysis indicated that no assurances are made against the API server actively changing the user's vote via a MITM (Man-In-The-Middle) attack. A salt is also required to unlock the database, which is stored in plaintext on disc in the application's shared preferences file. If the server conducts cryptographic procedures such as AES encryption, it may decrypt the user's vote before sending it to any external log via the SK_{aes} and convincingly re-encrypt any value provided to the log. Even if is not symmetric key available to the server, the application is vulnerable to a MITM attack, as access to the unencrypted TLS stream would have to be enabled if a for example a Hardware Security Module (HSM) is employed for cryptographic operations (Specter et al., 2020).

There is no public key authentication as part of the device handshake, and the device provides evidence to the effect that these interactions are ever registered on the blockchain. The server can terminate the connection before the HSM and arbitrarily impersonate the user's device by repeating the whole device handshake between Voatz server and application and all subsequent communication back to the blockchain via the HSM. The hypothetical HSM, which has the TLS keys necessary to terminate the connection and executes all cryptographic activities, is capable of launching attacks against the user. An attacker having access to the user's network activities, but no key material can determine how the user voted. The software specifically leaks the length of the plaintext, which can allow an attacker to identify, at the very least, who candidate the user voted for. The flaw originates from how a ballot is transmitted to the server after a user has finished making their choice. In a vote submission, the "choices" list containing users' choice, as well as the entirety of the metadata given by the server about that candidate. As a result, the length of the ciphertext varies greatly depending on the voter's choices. Attackers might also deduce the voter's preferences by evaluating the length of the packet that corresponds to the actual vote submission, with the remainder consisting of other protocol requests involved in vote casting and user maintenance. This issue is exacerbated by Voatz's extra encryption. Data is gzip-compressed at the application layer before being encrypted over TLS in Voatz's version, which may have provided some privacy if the compression alone was sufficient to mask the size discrepancies between plaintexts. However, because Voatz encrypts incoming data before the system applies gzip, this step is made insignificant, and the length of the final packet's ciphertext remains proportionate to the size of the plaintext (Specter et al., 2020).

The Voatz app is a privacy invading application that gathers information from users such as email, physical address, birth date, IP address, picture, device model, OS version, and language choice. It also asks for permission to read GPS data on the initial login. Voatz's use of third-party code comprises more than 22 libraries from 20 suppliers, including more than 22 from separate companies namely Jumio, Zimperium, Amazon AWS, Realm DB, Google Firebase / Crashlytics, gson, protobufs, zxking, Square OHTTP & Retrofit, Datatheorem's Trustkit. Overseas military voters have been reported to use the application, implying that information revealed about users might possibly supply rivals with knowledge on US military deployments. The user's IP address can convey location information, allowing organisations like Jumio, Crashlytics, and Zimperium to infer troop deployments (Specter et al., 2020).

Additionally, due to the application not requiring voters to re-enter their PIN upon login and does not notify users of re-voted or spoilt ballots, it is vulnerable to coercive attacks. If a voter becomes unconscious, an attacker with physical access to the device and the user's fingerprint might simply vote on their behalf. This susceptibility is especially crucial in instances of domestic violence (Matthews et al., 2017).

REFERENCES

Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J. A., Heninger, N., Springall, D., Thomé, E., Valenta, L., VanderSloot, B., Wustrow, E., Zanella-Béguelin, S., & Zimmermann, P. (2015). Imperfect forward secrecy: How diffie-hellman fails in practice. *Proceedings of the ACM Conference on Computer and Communications Security*, *2015-October*, 5–17. https://doi.org/10.1145/2810103.2813707

Beurdouche, B., Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Pironti, A., Strub, P. Y., & Zinzindohoue, J. K. (2017). A messy state of the union: Taming the composite state machines of TLS. *Communications of the ACM*, *60*(2), 99–107. https://doi.org/10.1145/3023357

Halderman, J. A., & Teague, V. (2015). The New South Wales iVote System: Security Failures and Verification Flaws in a Live Online Election. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9269*, 35–53. https://doi.org/10.1007/978-3-319-22270-7_3

Matthews, T., O'Leary, K., Turner, A., Sleeper, M., Woelfer, J. P., Shelton, M., Manthorne, C., Churchill, E. F., & Consolvo, S. (2017). *Stories from survivors: Privacy & security practices when coping with intimate partner abuse*. https://research.google/pubs/pub46080/

Specter, M. A., Koppel, J., Weitzner, D., Specter MIT, M. A., Koppel MIT, J., & Weitzner MIT, D. (2020). The Ballot is Busted Before the Blockchain: A Security Analysis of Voatz, the First Internet Voting Application Used in {U.S}. Federal Elections. In *arXiv*. https://www.usenix.org/conference/usenixsecurity20/presentation/zhou-jie

Springall, D., Finkenauer, T., Durumeric, Z., Kitcat, J., Hursti, H., MacAlpine, M., & Halderman, J. A. (2014). Security analysis of the estonian internet voting system. *Proceedings of the ACM Conference on Computer and Communications Security*, 703–715. https://doi.org/10.1145/2660267.2660315

Valenta, L., Cohney, S., Liao, A., Fried, J., Bodduluri, S., & Heninger, N. (2016). Factoring as a Service. *Financial Cryptography*, *9603 LNCS*, 321–338. https://doi.org/10.1007/978-3-662-54970-4_19

Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, & J. A. Halderman. (2015, March 3). *Tracking the FREAK Attack*. https://freakattack.com/