NSWEC-10

DemTech Group

DEMTECHGROUP
DEMOCRACY, TECHNOLOGY & TRUST

Ribegade 19 st th
2100 Copenhagen
Denmark
CVR: 35527052

Review of the Revised iVote 2021 System

David Hook and Carsten Schürmann

July 2021

Final Report

Branch: master Commit: cbff0245cd06f7c0bf20a23d03313fe484e25e2d

Table of Contents

1 Executive Summary

We have reviewed the version 1.8.5 (06/2021) of the iVote source code. Overall, our analysis has shown that most of the code has not undergone major changes since the release in 2019, the update to the mixing component being the most notable exception. While this would suggest the risk of a major malfunction of the iVote system during deployment is relatively low due to stability of the code base, the recent changes to the configuration files and dependencies may have introduced new risks that can, at this point in time, only be mitigated through rigorous, systematic, and extensive testing.

Remaining Areas of Concern

1. *Remaining Known Vulnerabilities.* We have carefully followed the ongoing effort of the Vendor to reduce the number of known vulnerabilities of the code base. The remaining vulnerabilities are carefully documented and accompanied by a rationale. Overall the rationales make sense, assuming the Vendor assertions related to them can be accepted.
2. *Trusted Build Issues.* The final code drop appeared to have three forms, what was given to us, what was provided to the Commission by the Vendor, and what was described in the static code analysis documentation the Vendor provided us with. With assistance from the iVote Team at the NSWEC, we believe we have worked out that the codebase we were given was a superset of what was deployed as it appears some duplicate and unused packages are present in the trusted build source. If we continue to assume this premise is correct and also ignore the version numbers and some discrepancies in module naming for the static analysis report, the static analysis report also appears to match up. As this is the case it seems likely but not certain, that what we were presented with is the same as what NSWEC has deployed, barring some last minute patches.
3. *Test Quality.* Despite several requests, the Vendor failed to provide annotated coverage reports for system testing, and also details of any security or "fuzz" testing. We have been provided with partial coverage summaries with percentages, but these are not enough to allow for the recognition of dead code in the system which would be a useful indicator of quality and risk in the system. We say risk as occasionally otherwise unused code can feature in attacks built around feeding unexpected input into a system. This risk is usually checked for using what is commonly referred to as "fuzz" testing.

4. *Documentation Quality.* The documentation of the iVote system is imprecise, incomplete, and in many parts out of date.
5. *Static Code Analysis.* The Vendor provided us with a detailed SpotBugs report of the final build. This report is noteworthy, because it consists of over 500 pages, listing common programming problems identified in the iVote source code, many of which are considered harmful. While we regard it as a positive that, over the course of the review, the Vendor appears to have integrated the reporting into their development process, there are a number of issues highlighted, in particular which may indicate concurrency problems, that the report details.

Recommendations

1. We strongly recommend the Commission should carefully check that they are fully satisfied with the explanations provided and ask the Vendor for a more details, if required. We say this as some explanations indicate the Vendor is assuming other mitigations are in place. While this does not discount the rationale offered, in the interests of safety and security, it is important that Commission staff understand what the implications of any rationale may be.
2. We strongly recommend the Commission review the test plans provided by the Vendor and ensure those plus testing by the Commission ensure that they take account of edge conditions, unexpected events, and how the system behaves under high loads. This is a remote voting service and we are in the middle of a pandemic - it is highly likely the system will see higher than usual load. While some of the concurrency issues reported in SpotBugs are likely to be false positives, load testing will also increase the likelihood of finding any real concurrency issues.
3. Ongoing, we strongly recommend the Commission should devise and implement a strategy to avoid maintenance problems such as the ones outlined in this and previous reports. This would need to include, the local building of the iVote system from Vendor sources to ensure accurate review is possible, and to minimise any opportunities for supply-chain attacks on the system via manipulation of dependencies or software. It should also include, at a minimum: the continued use of automated static code analysis, the automated generation of annotated coverage reports for system tests, continuous monitoring of third-party dependencies for CVEs and other security issues, and finally, on going maintenance of documentation to reflect design and software changes. Finally, there should be be an explicit commitment by the Vendor to take action in regard to any of the aforementioned should an issue be flagged, and that such checks and reports will be generated regularly, even if the code is not under active development.

2 Scope and Review Methodology

This report is a review of the iVote version 1.8.5 (06/2021). It supersedes previous analysis which concerned iVote version 1.8 (03/2021). The source code of iVote 1.8.5 was delivered in form of a 1.8GB compressed file entitled "nswec-trusted-build.tar.gz". The documentation provided by NSWEC and the Vendor comprises several documents [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], of which we rely heavily on [8] that contains the description of the voting protocol. We use the other documents only as reference. The documentation was not updated from iVote 1.8 to 1.8.5, but the Vendor provided an additional 500 page document [17] that summarizes all remaining software deficiencies produced by the Vendor's software quality assurance tools, which we also review in this report. Two other reports [15, 16] were shared by the Vendor in response to earlier versions of this draft. In addition, we made use of four relevant scientific papers [11, 12, 13, 14].

[1] NSW Electoral Commission. iVote Project – Security Threat Model. Internal Report, July 2018.

[2] NSW Electoral Commission. iVote Voting System, Election configuration file specification . Internal release, April 16 2018. Version 0.1.

[3] NSW Electoral Commission. iVote Voting System, iVote Voting System – Cryptolib Architecture. Internal release, April 30 2018. Version 1.0.

[4] NSW Electoral Commission. iVote Voting System, iVote Voting System – Secure Logger Architecture. Internal release, April 30 2018. Version 1.0.

[5] NSW Electoral Commission. iVote Voting System, iVote Voting System – Solution Architecture. Internal release, April 30 2018. Version 1.0.

[6] NSW Electoral Commission. iVote Voting System, Specification of Scytl Online Voting. Internal release, April 30 2018. Version 1.0.

[7] NSW Electoral Commission. Security controls and risk reduction. Internal Report, January 2018.

[8] NSW Electoral Commission. iVote Voting System, Voting Protocol Description. Version 1.2, February 2020.

[9] NSW Electoral Commission. iVote Voting System, Interface specifications. Internal release, April 10 2021. Version 1.0.

[10] Scytl R+S Department. Scytl PRNG, May 5 2021. V 2.0.

[11] Rolf Haenni. Swiss Post Public Intrusion Test - Generating Random Group Elements (Best Practice). Technical report, Bern University of Applied Sciences, March 12 2019.

[12] Rolf Haenni. Swiss Post Public Intrusion Test - Undetectable Attack Against Vote Integrity and Secrecy. Technical report, Bern University of Applied Sciences, March 12 2019.

[13] J. Alex Halderman and Vanessa Teague. The New South Wales iVote System: Security Failures and Verification Flaws in a Live Online Election. Washington, D.C., August 2015. USENIX Association.

[14] Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. Ceci n'est pas une preuve. The use of trapdoor commitments in Bayer-Groth proofs and the implications for the verifiabilty of the Scytl-Swiss Post Internet voting system. Technical report, Open Privacy Research Society, UCLouvain, The University of Melbourne, March 12 2019.

[15] Scytl. Naked SQL and list of CVE libraries under analysis. Internal note, July 17, 2021.

[16] Scytl. Scytl's answers to the auditor's questions/recommendations. Internal note, July 17, 2021.

[17] Spotbugs. Spotbugs report, based on data provided by Scytl. HTML formatted XML data, July 2021.

Because of the sheer size of the source code, we carefully scoped the functional matching task: In scope were all modules that touch, store, cleanse, mix, and decrypt the vote, including the govlab modules and the JavaScript client, but excluding the secure logging system [4]. Outside the scope were, in general, all other modules except for the functionality necessary to follow the vote. I.e. we do not review the cryptolib [3] exhaustively, but only parts, for example to create and verify Schnorr proofs, encrypt and decrypt votes, and create and verify digital signatures. Requirements for establishing authenticated channels, for example using certificate chains, are not described in the specification [8], and are therefore outside the scope of this review.

Furthermore, outside the scope of this report were are all parts of the implementation for which no design documents or other documents were provided. This includes some of the standard modules of the Vendor voting solution, but also

configuration functionality related to iVote deployment and third-party libraries, such as inVote, RabbitMQ, Spring, and Hibernate.

Due to the lack of good code scanning tools for JavaScript, no automated code scanning analysis for JavaScript APIs was conducted.

Some information which affects the ability to properly assess the quality of the software has been unavailable. The Vendor has not provided meaningful coverage information, despite request. The absence of information contributes to a lack of transparency that makes it impossible to asses the overall quality of the software.

3 Functional Matching

3.1 The Quality of the Documents Provided

We expected that the files included in the source code drop would give us sufficient information (1) to serve as design documents and (2) to provide us with enough information on how the different software modules interact. Unfortunately, this was not the case. When we reviewed the Interface specification [9], we found it to be under-specified and incorrect. For example, Figure 1 in [9] gives a good overview over eleven modules that define iVote. The document explains some of the interfaces, but not all. In the figure, interfaces for the voter portal and the verification app are depicted, but they are not explained in the body of the document. Some of the arrows in the figure carry different identifications than those used in the body of the document and some were not explained at all. In addition, there is no direct matching between Figure 1 and the implementation.

For some modules, no documentation was provided, for example the inVote system, an integrated solution developed and provided by the Vendor for organization public and private elections. When asking the Vendor for additional information, we were informed that it was unavailable. From the logs supplied with the source code, we learned that inVote's development originated around 2006, but modules for iVote were only added in November 2019.

Recommendation: The Vendor should supply additional documentation for inVote.

3.2 Verifiability Analysis

In this analysis, we traced the the vote throughout the source code, starting with how an empty virtual ballot paper (VBP) is generated, supplied to the authenticated user when requested through the JavaScript client, how it is completed by the voter, how evidence in the form of a Schnorr proof is generated, the signature are generated, the vote is returned to the castvote module in the inVote system, how the the voter requests the vote for verification, how the ballot is then forwarded to the cleanser, the mixer, and eventually decrypted. Overall, we found, that the iVote system implements the specification faithfully. However, we

have identified several observations that we believe can help the NSWEC and the Vendor to improve the implementation.

3.2.1 Complexity

The complexity of the source code is owed in part to an apparent issue with the build and dependency management system at the Vendor, and also in part because iVote is built on top of several different frameworks that in turn depend on several third-party libraries. The Spring framework provides all functionality with respect to secure message passing, the inVote framework the functionality for implementing a particular voting protocol. We found that compared to the prior version, the complexity of the iVote system was greatly improved. The 1.8.5 build of iVote shows substantial progress in reducing complexity, previously we were given 105 modules, this has now been reduced to 72. But there is still duplication.

nsw-ivapi.min.js:

This file appears in the nsw_voter_portal_0_15_26. It is apparently used in the web client for voters and is just over 126,000 lines long. Unfortunately we were not able to confirm how it was constructed as, while the Vendor did provide some instructions, the instructions failed as the building process appears to rely on access to internal Vendor systems.

Our concern here is two fold, we are not entirely sure what has gone into the file's construction and the file does appear to contain some duplication, for example, the Fortuna random number generator appears at least twice, possibly more times. It is difficult to review JavaScript at the best of times, but the duplication makes it impossible to be sure what code will be executed.

While we would expect the file to be obfuscated and compressed for download to clients, it is important to keep in mind those steps are reversible and the code is essentially published to the general public, including those who might not be well intentioned. The code provides an access path into the inVote system, so it is important that the client side API should be reviewable.

Recommendation: The build of the nsw-ivapi.min.js should be simplified to make it possible for reviewers to generate a file without duplication.

The nswec_govlab module:

This module contains the source code of the inVote framework that is a central part of the iVote system.

```
nswec_govlab_1_3_0-RC2
nswec_govlab_1_3_1
```

We observe that two Java files, and multiple POM files are different. A POM file captures the dependencies of modules on other modules and/or libraries. Two files are updated,

```
NSWImportVotersAndVotes.java
CsvManagerJdbcDao.java
```

and the differences in both cases appear to be meaningful. Which govlab module is in use?

The cryptolib module:

The library that implements the basic cryptographic operations, cryptolib,

```
cryptolib_2_4_1
cryptolib_2_7_2
```

show substantial differences. The only module depending on 2.7.2 is the secure logger. Assuming that 2.7.2 is more recent than 2.4.1, we wonder why doesn't the entire iVote system use the latest release? Is the 2.4.1 release still maintained?

The jbasis_cryptolib module:

Also the jbasis_crypto library appears at different versions and levels of maturity.

```
jbasis_crypto_4_1_0
jbasis_crypto_4_2_1
jbasis_crypto_4_2_1_3
jbasis_crypto_4.3.1
```

In this case jbasis_crypto_4_1_0 and jbasis_crypto_4_2_1 appear largely the same, while a diff shows substantial differences they appear to be due to a change of formatting. jbasis_crypto_4_2_1_3 has a couple of differences from jbasis_crypto_4_2_1 and also includes git conflict messages in the source code indicating a failed merge. The differences between 4_2_1 and 4_2_1_3 appear to be in the XML parsing. jbasis_crypto_4.3.1 appears to add some use of generics with further formatting changes and a change to the method for outputting certificate PEM files but appears to be missing the XML changes from 4_2_1_3. Each module appears to be used at least once. In post election maintenance the modules need to properly reviewed and the divergence that seems to have happened in 4_2_1_3 resolved.

The p7_cms module:

These modules are duplicates that appear at different version numbers and levels of maturity.

```
p7_cms_1_2_0
p7_cms_1_5_1
p7_cms_1_5_1_1
```

1.2.0 is substantially different from 1.5.1, 1.5.1.1 uses the latest version of the Bouncy Castle crypto library. It is also concerning the 1.5.1 is still in use with invote_plugin_counting_tally_1_4_3_1, invote_plugin_counting_mixing_1_4_2_1 and invote_plugin_counting_cleansing_1_4_2_1 as it overrides the choice of Bouncy Castle to use bcmail 1.55 which is well out of date. This last issue is also concerning as the resolution of the transitive dependency on bcmail 1.55 during building may override the parent dependency on BC 1.68 resulting in the use of bcprov 1.55 which is also subject to a number of CVEs. 1.2.0 is well out of date and should not be present at all.

The scytl_math module:

```
scytl_math_1_0_1
scytl_math_1_1_0
```

1.1.0 adds two new methods to BigIntegers class. We could not identify any other meaningful changes, which seems to suggest that 1.0.1 is unnecessary. Why 1.0.1 is present?

The nsw_commons module:

```
nsw_commons_lib_1_7_2
nsw_commons_lib_1_7_4
```

Differences appear to be related to the POM files. The Java source files are different only in the context of the copyright notice which has been updated from 2020 to 2021.

The maven dependency and generic modules:

```
maven_dependencies_1_2_0
maven_dependencies.2.1.0
maven_dependencies.2.2.1
maven_dependencies.2.2.1.1

maven_generic_conf_1_5_4
maven_generic_conf_1_5_5
maven_generic_conf_1_5_6
maven_generic_conf_2_0_0
maven_generic_conf_2_0_2
```

Version 2.1.0 does not appear to be used anywhere, but is included in the trusted build. maven-dependencies 2.2.1.1 appears to be widely used. Related to the maven dependencies, `mixing_1_0_0_1` refers to maven dependency 1.2.0 - while this should be a cause for concern it appears the POM file

in `mixing_1_0_0_1` overrides all the imports so the maven-dependencies file is actually ignored.

From a software engineering point of view, even the remaining complexity should be considered harmful. It provides a soft target to criticize the iVote system on and depending on the criticisms made could be very difficult to defend against.

Recommendation: The source code should be further simplified.

3.2.2 Explicit Erasure of Votes

We identified a piece of code, which we classify as dangerous: (class AuthorizedElectionVotersDaoImpl, line 29 ff)

```
public void deauthorizeAllElectionEventMinusElection(final Voter
    voter, final Election election) {
     StringBuilder sql = new StringBuilder();
     sql.append("delete from authorized_election_voters where
        voter_id = :voter ")
         .append(" and election_id <> :election ").append(
         " and election_id in (select id from elections where
            election_event_id = :electionEvent) ");

  getSession().createSQLQuery(sql.toString()).setString(VOTER_PARAM,
     voter.getId())
        .setString(ELECTION_PARAM, election.getId())
        .setString(ELECTION_EVENT_PARAM,
           election.getElectionEvent().getId()).executeUpdate();
```

First and foremost, this call is a direct SQL call, which is executed within a method that is under the Spring framework's control. It is not clear, how transaction management is handled. In general, when dealing with concurrency in programming, race conditions can have awful side effects, for example, the Spring framework may want to roll back a transaction, but because the SQL was executed this way it cannot be or may not be correct. The code assumes that someone else has already created a transaction. The consequence is an inconsistent database.

Second, in terms of verifiability such implementation should be avoided. Retroactively, there is no easy way to explain if a vote was deleted or not. It could be the case that there is sufficient evidence stored in the SQL server's log file, but this information is typically not considered verifiable.

Third, this functionality can be easily misused by someone with access to the server iVote system, and executed, for example, after the deadline for vote verification expired. Executing similar functionality, for example compiled into a separate module, would allow the attacker to remove votes selectively from the database to sway a tight election.

Fourth, getSession() returns the current session on the thread if it exists. It may actually return null if a session is not established, in which case the SQL command will fail. Another potential issue, assuming this code is actually under a transaction, is if the current transaction in the session which is being borrowed is rolled-back. If this happens it will also roll-back the delete which will again cause consistency problems. It is difficult for us to assess how serious this problem is as substantial analysis work would be required to identify both the possible transactions in the system and the dependent classes and operations making up those transactions. There may be no risk here, we only flag this as a potential issue.

We counted several similar occurrences of naked SQL in the following files that delete information from SQL-databases using the “DELETE FROM” or “SELECT FROM” SQL-syntax. It seems that several cases may be necessary to provide the necessary infrastructure for the Spring framework to operate properly, but in all other cases the naked SQL code seems to be doing real deletes. The latter are of concern.

Recommendation: The Vendor has responded [15] that all naked SQL calls can be regarded as safe or as dead code. If the code is dead code it should be eliminated by the Vendor in future. For future review, where the code is not dead code, the dependent classes making up each transaction should be documented allowing a fuller analysis of the approach used.

3.2.3 Key Generation and Randomness

Key generation in the minified JavaScript, nsw-ivapi.min.js, appears to be relying on a Fortuna implementation for providing the underlying randomness for keys. The JavaScript modules supporting this are 3rd party ones, but do come from recognised implementations of the algorithms they purport to represent. The Vendor has provided some quality analysis and documentation concerning entropy collection [10] however it does appear to be for an earlier version. A look at the code in the JavaScript reveals that the entropy collection is based on code that pre-dates the previous version of iVote so there is some confidence the analysis presented in the out-of-date document is as relevant as the Vendor claims and there is some evidence in the JavaScript that it is been applied.

We say some evidence as the JavaScript is unstructured, contains repeated code, and is over 120000 lines. Ideally there would be no repetition and it would be possible to properly examine the components making up the script. We did request assistance in this matter from the Vendor and we were recently provided with some instructions as to how to do this, but unfortunately it was not realised in time that the scripts backing those instructions relied on access to internal Vendor repositories and they failed.

Recommendation: In future, it must be possible to build the minified JavaScript as part of the review process. The Vendor should investigate the causes of duplication and eliminate them where possible.

3.2.4 Unused Code

We found unused and potentially harmful code all over the source code of the iVote system.

In com.scytl.invote.plugin.counting.mixing.basic, line 29ff., which is a highly non-verifiable mix of the ballotbox. Although this code most likely will never be executed when running iVote in production environment, its mere presence is worrisome. This code was highlighted in our draft commentary on 1.8, it is still present in 1.8.5.

```
public MixingBasicOutput execute(MixingBasicInput
   input) {
  List<byte[]> block = input.getMixingBlock();
  Random random = input.getRandom();
  Collections.shuffle(block, random);
  Collections.shuffle(block, random);
  return new MixingBasicOutput(block);
}
```

And there are other places like this. For example, the iVote source code defines two kinds of functionalities, one is called “homomorphic” and the other “verifiablemixing”. We do not believe that the modules for “homomorphic” are used in iVote, however, they are still there.

In SecureMessageStoreService.java, line 55, the variable

```
  private boolean onlyCanVoteOneElectionIntoElectionEvent =
     false;
```

is hardcoded, so why is it there at all?

Recommendation: The source code must be refactored and all unused modules and functionality removed. Any production build should only be based on the cleaned code.

3.2.5 Missing Contracts and Invariants

Related to this, we observe that many modules in the iVote source code do not provide comments about what they implement, and if they do, they don't do it in a structured way. This absence adds to the complexity of the source code, which becomes very unwieldy for a third party to review and analyze. This is clearly not good practice, and definitely not state of the art. At the very least, methods and functions should provide contracts[1] that define clearly what are the input arguments to a function, which invariants have to hold, for example, $n > 0$ or k is a valid key, what are the output arguments, and what are the relations between input and output arguments. Such contracts allow a reviewer to check the correctness of and implementation of a method and its call sites, and allow the programmer to program better code. Contracts are state of the art, and they greatly improve code quality.

Without contracts, it is impossible to comment on the quality of the code for many methods. For example, at several places in the code, where html and xml information is prepared (ReceiptMessageService.java, line 37ff.), it is not clear if the inputs are properly sanitized.

Recommendation: The Vendor should add contracts to all methods that define the iVote system.

3.2.6 Passwords

The method loginAdvanced in ivapi.js, line 344ff. does require that username(ivote number), [password], and pin, are passed in transformed form. Passwords are not transformed, since TransformAuthentication inherits its methods from the NoTransformAuthentication, which means that in iVote, the argument password will always be set to `"password"`. It appears that the Vendor has used their standard modules, and customized some of them to use pin as a third argument, leaving the password argument unused. So far so good. The problem, however, is that other standard modules might not be modified for the use in iVote. After all, the size of the source code and the many unused modules present in the source code are a strong indication that this might be the case. There is a concern that other parts of the unused functionality may be activated using

[1] Bertrand Meyer, *Applying "Design by Contract"*, IEEE Computer, October 1992.

iVote-number and password `"password"` alone (without the pin), which would allow an attacker to gain access and cause havoc.

Following up on this observation is outside the scope of this review. However, if this suspicion proves true then iVote is vulnerable to outside attack. It is possible that there is no issue here, however, while this could be regarded as speculation, faced with a situation where things could be "okay" or "not okay", we feel obliged to flag the issue for the Vendor to respond to. The uncertainty around this issue would be greatly reduced, and possibly eliminated, if all unnecessary functionality was removed from the iVote source code.

Recommendation: All unnecessary functionality must be removed from the source code.

3.2.7 Hardcoded Passwords

We have identified a place in iVote's configuration file, which looks like hardcoded passwords. It is unclear, if these are the passwords are used in the production system. We flag this for further investigation. The files are govlab_1_3_1/rest-portal/src/main/config/env.properties, line 35ff. and govlab_1_3_2-RC2/rest-portal/src/main/config/env.properties, line 35ff. Redacted examples are:

```
auth_server_password=[redacted]
message_server_password=[redacted]
portal_secure_log_sign_password=[redacted]
portal_secure_log_cipher_password=[redacted]
gdpr_key_store_password=[redacted]
gdpr_hmac_key_password=[redacted]
gdpr_hmac_alias=[redacted]
gdpr_aes_key_password=[redacted]
gdpr_aes_alias=[redacted]
```

If these passwords are not used, then they should be removed from the code. If they are used, they should also be removed from the source code and stored elsewhere in a secure file.

Recommendation: The hardcoded passwords should be removed from the source code.

3.2.8 Quality of the Specification

Although the specification describes the steps to understand the cryptographic protocol implemented in the iVote system, it is also extremely unspecific when it comes to details, including ballot structure, validity checking, security policies, database use, etc. This generality gives the Vendor considerable freedom in implementing and reusing existing functionality. This lack of specificity may well have contributed to the issues apparent with bloat in the implementation that is under review. We demonstrate this using a few examples.

1. To get a better feel of what is missing, ivapi.js, line 865, for example, describes functionality of downloading additional content, getAdditionalImage, getAudio, validateBallot, but this content is not described in the specification [8].

2. In AnswerEncoderVM.js, line 53, where ballot preferences are being reordered. These kind of operations are not captured in the specification, but should be.

3. In the specification it is not mentioned that it is checked that prior to decryption that the randomnessID matches as well. In fact, the specification does not even mention the existence of a field randomnessId. Of course one can argue that if the randomness doesn't match, the verification will fail anyway, however, this is an indication that the specification should be refined.

4. The implementation generates several files during cleansing, including a DistrictsOutputfile, a ManifestFile, a ElectionEventPublishFile, and Auditable File. None of these files are described in the specification. This indicates that there is a difference between implementation and specification.

Recommendation: In future, NSWEC should consider creating in-house capacity for refining specifications into proper design documents.

4 Static Analysis

4.1 Trusted Build

We observe that the production version of the iVote system was delivered to NSWEC two weeks before a usable “trusted build” script was completed by the Vendor that we could use to compile the iVote source code on our own machines. This implies that the Vendor’s build process relies very much on their own development environment and cannot easily be exported to third parties. But this also means that there is no guarantee that the source code and libraries under review are really the source code and libraries used to build the production system, and that they are complete. For example, the module invote-audit-tool-invote-audit-tool-1.4.1 was not included in the original drop, but shared with us later.

In our assessment, if things remain as they are, NSWEC will not have access to the source code that was used to build the iVote system, nor can obtain a copy from the Vendor and convince others that this is the correct version. We consider this inherently problematic and a unnecessary security risk and flag it as an issue.

The separation of the software distribution process and the creation of the trusted build reduces confidence in our ability as reviewers to determine whether what we are looking at is actually what has been delivered. This confidence is further eroded by the continuing presence of duplication, particularly in areas of the code that are security relevant and admissions by the Vendor that constructing the trusted build is problematic. In some cases duplicated libraries appear to have meaningful differences which would suggest bug fixing and that the older version is unpatched. It is impossible for us to determine whether an out-of-date version of a Vendor library has been deployed to part of the delivered system where the lack of a patch will be meaningful and result in a critical failure.

The separation has also serious security implications: A supply-chain attack on the Vendor or a Maven repository provider would allow a stealthy adversary to attack the binary distribution of iVote without the Vendor, the reviewers, or the NSWEC ever knowing about it before it is too late. In such a hypothetical supply-chain attack, the adversary leverages previously obtained access to the vendors’ computer systems to augment the build process of iVote unnoticed

by adding triggers or malware, such as backdoors, ransomware, or other dangerous software into the compiled system. This kind of attack is not explicitly considered in the materials under review [1, 7], but clearly possible considering the Solarwinds attack in late 2020. It could be easily mitigated by ensuring the NSWEC is given the original source and artifacts to build the system and then passes the actual source and artifacts to reviewers for review.

Recommendation: NSWEC should have access to the original source, for example, a git repository, and should be able to build the production iVote system from scratch.

4.2 Analysis of SLOCcount Report

SLOCcount is a simple tool for doing analysis and cost estimation of the effective lines of code in software. We compare the SLOCcount report is much improved, because of the removal of duplicate files and unnecessary modules. For a full report, consult Appendix A.

Current (iVote 1.8.5):

```
javascript:  593941 (45.44%)
java:        427715 (32.73%)
xml:         227204 (17.38%)
ansic:        31923 (2.44%)
python:       19782 (1.51%)
cpp:           4907 (0.38%)
sh:             946 (0.07%)
lisp:           264 (0.02%)
cs:             211 (0.02%)
perl:            99 (0.01%)
```

Previous (iVote 1.8):

```
javascript: 1530825 (64.10%)
java:        520115 (21.78%)
xml:         261152 (10.94%)
ansic:        33194 (1.39%)
python:       30699 (1.29%)
cpp:           8994 (0.38%)
sh:            2203 (0.09%)
```

```
cs:             420 (0.02%)
perl:           295 (0.01%)
lisp:           264 (0.01%)
```

There is still some obvious duplication. A look at the full SLOCcount report shows (see Appendix A):

```
0           online_voting_logging_1.3.6.1 (none)
0           rabbitmq_spring_webapp_2_0_0 (none)
```

The reason for the zero line count is that the contents of online_voting_logging_1.3.6.1 is identical to the contents of logging_1_3_6_1 and rabbitmq_spring_webapp_2_0_0 is identical to rabbitmq_producer_2_0_0. It is not clear why the duplicate packages would be present.

4.3 Analysis of Test Rail Report

Test Rail report appears to be consistent with code updates. The last meaningful code update appears to have been on June 18, 2021 in nswec_govlab_1_3_1.

4.4 SpotBugs Static Analysis

The Vendor provided a SpotBugs static analysis report, which we have made available in a separate document [17] due to its size (approx. 512 pages). The SpotBugs analysis still indicates some code quality issues, some of which may be important.

What follows is a sample of what was found.

RestoreElectionEventTaskParams

There are 2 classes with this name, the one reported is under: nswec_govlab_1_3_1/rest-services-bo

The error reported is PREDICTABLE RANDOM, the code lines of code concerned are:

```
this.fileElectionEventName = "ELECTION_EVENT_RESTORE"
    .concat(String.valueOf((int) (Math.floor((Math.random() * 10000000) + 1))))
```

```
        .concat(".xml");
    this.fileCandidateListName = "CANDIDATE_LIST_RESTORE"
        .concat(String.valueOf((int) (Math.floor((Math.random() * 10000000) + 1))))
        .concat(".xml");
```

It appears that the code is trying to generate some random names and avoid a possible clash by using the java.util.Random. We are assuming the need for randomness is because the class is being used in a multi-threaded fashion and that a clash between two uses may cause failure or invalid behaviour.

We would point out that the above could be done safely using a counter based on classes from the Java concurrency API and that as Random is based on a 48 bit progression, the chance of a clash is far higher than should be acceptable in an application being used to run an election. We are unable to ascertain what the implications of a clash would really be.

ApplicationInfoVO

This class appears under nswec_govlab_1_3_1/rest-services-bo/.

The error report is EL_EXPOSE_REP. The code concerned is:

```
/**
 * @return Returns the currentTime.
 */
public Date getCurrentTime() {
  return currentTime;
}

/**
 * @param currentTime The currentTime to set.
 */
public void setCurrentTime(final Date currentTime) {
  this.currentTime = currentTime;
}
```

In Java, java.util.Date is actually a mutable object. In both cases the caller to the above methods could change the value in current time, consequently changing what is in ApplicationInfoVO. It is unclear what the implications of an error like this would be, but it could be easily avoided by creating a new Date object to hold the value on either the set or the get method.

SuspendedElection

This class appears under nswec_govlab_1_3_1/repositories-bo/.

The error report is EQ_CHECK_FOR_OPERAND_NOT_COMPATIBLE_WITH_THIS. The code concerned is:

```
/**
 * @see java.lang.Object#equals(java.lang.Object)
 */
@Override
public boolean equals(final Object obj) {
  boolean equals = false;

  if (obj instanceof SimpleElection) {
    equals = getId().equals(((SimpleElection) obj).getId());
  } else {
    super.equals(obj);
  }

  return equals;
}
```

SimpleElection is the wrong type and this code cannot possibly be correct. Use of super.equals() defaults to object identity, or at least it would if the return value from super.equals() was not ignored. The code will actually fail to return true if an object is compared with itself. It is not clear what the implications of this error are.

RenewAuthTokenController

This class appears under nswec_govlab_1_3_1/repositories-bo/.

The error report is SERVLET_PARAMETER. The code concerned is:

```
ElectoralScope es = ElectoralScopeBuilder.getInstance()
    .withInstitutionId(request.getParameter(MessageConstants.INSTITUTION_ID))
    .withElectionEventId(request.getParameter(MessageConstants.ELECTION_EVENT_ID))
    .withElectionId(request.getParameter(MessageConstants.ELECTION_ID)).build();
```

The issue reported here is the servlet parameters are passed straight in without validation. It is not clear the parameters are validated else which is a particular concern as the ElectoralScope object is referenced in a number of the Dao objects.

As already mentioned, this is only a sample, of the information contained within the SpotBugs report. With the exception of the SuspendedElection bug, all the above reports appear multiple times in the 512 pages making up the current SpotBugs report [17] based on the XML provided by the Vendor on the 4th June 2021 and we have confirmed the issues still appear to be present. Of particular concern are reports related to possible concurrency issues and servlet parameter validation. Tweaking servlet parameters to cause chaos is a common approach employed by people looking to attack a system.

Recommendation: We would recommend the NSWEC review the SpotBugs report with the Vendor, paying particular attention to concurrency issues and invalidated servlet parameters, and patch where possible. Testing should also be done to with faulty, invalid, and out of range servlet parameters to ensure the system deals with them gracefully.

Appendix A: SLOCcount Report

```
SLOC  Directory   SLOC-by-Language (Sorted)
304585  lib_parent_grunt_0_5_1 javascript=247635,ansic=29665,python=19266,
                    cpp=4907,xml=2132,sh=505,lisp=264,cs=211
180654  nswec_govlab_1_3_0-RC2 java=122132,xml=57755,javascript=753,sh=14
174095  nsw_customization_1_4_4 javascript=173949,sh=146
104461  nswec_olv_credential_manager_1_7_8 xml=96592,java=7869
78878   nsw_voter_portal_0_15_26 javascript=78878
66573   cryptolib_2_4_1 java=56235,javascript=6002,xml=4336
64115   cryptolib_2_7_2 java=55158,javascript=6133,xml=2824
22735   nsw_commons_lib_1_7_2 java=19445,xml=3290
22734   nsw_commons_lib_1_7_4 java=19445,xml=3289
22730   nswec_govlab_new_backoffice_frontend_1_2_1 javascript=22662,xml=68
19354   nsw_converter_1_7_3 xml=15047,java=4307
19135   invote_receipt_admin_1_2_1 java=15756,xml=3379
19040   js_forge_0_6_45_4 javascript=18919,xml=121
16370   invote_domain_model_1.4.2.1 java=15471,xml=899
15425   lge_converter_1_0_3 xml=9394,java=6031
15193   nswec_govlab_javascript_client_api_1_3_6 javascript=14771,xml=422
15009   mixing_1_0_0_1  java=14482,xml=527
14656   js_forge_0_6_8  javascript=14569,xml=87
9884    crypto_resources_maven_plugin_1_1_0_1 java=8786,xml=1098
9048    invote_plugin_counting_tally_1_4_3_1 java=8462,xml=586
7734    secure_logger_5_4_0 java=7182,xml=552
7526    maven_generic_conf_1_5_4 java=5815,xml=1711
6000    nsw_results_1_7_5 java=5596,xml=404
5994    jbasis_crypto_4.3.1 java=4766,xml=1175,javascript=53
5209    invote_plugin_counting_cleansing_1_4_2_1 java=4509,xml=700
5088    nswec_govlab_receipts_frontend_1_2_0 javascript=5020,xml=68
5037    invote-protocol-common-pom-1.8.3.1 java=4655,xml=382
4796    invote_plugin_import_election_event_1_4_3_1 java=4318,xml=478
4442    certificate_validation_1_5_3 java=2838,xml=1604
4166    invote_plugin_castvote_1_4_3_1 java=3634,xml=532
3379    nswec_event_consumers_1_7_3 java=2145,xml=1234
3200    invote_crypto_base_1_10_0_1 java=2994,xml=206
3199    nswec_govlab_1_3_1 xml=2435,java=764
2844    jbasis_crypto_4_1_0 java=2741,xml=103
2797    invote_plugin_counting_decrypt_1_4_2_1 java=2420,xml=377
2777    invote_plugin_api_1_4_3_1 java=2655,xml=122
2667    jwt_springsecurity_1_2_5_1 java=2353,xml=314
2587    shares_3_2_0    java=2506,xml=81
2504    nswec_credential_manager_fe_1_0_9 javascript=2434,xml=70
2389    invote_plugin_counting_mixing_1_4_2_1 java=2062,xml=327
2357    scytl_math_1_0_1 java=1178,ansic=621,xml=558
2034    nswec_crypto_resources_generator_1_8_1 xml=2034
2019    rabbitmq_producer_2_0_0 java=1658,xml=361
1598    scytl_gmp_6_1_1 ansic=1564,xml=34
1527    nswec_receipts_portal_frontend_portal_1_7_1 javascript=1463,xml=64
1301    detectjvm_3_0_19 javascript=596,xml=479,java=226
1130    jbasis_crypto_4_2_1_3 java=1015,xml=115
1111    logging_1_3_6_1 java=822,xml=289
1075    nsw_splunk_1_7_3 xml=1075
1020    lib_jwt_1_0_2_1 java=912,xml=108
984     invote_plugin_security_model_config_castvote_verifiable-mixing-1_1_2_1
xml=984
976     maven_dependencies.2.1.0 xml=976
976     maven_dependencies.2.2.1 xml=976
976     maven_dependencies.2.2.1.1 xml=976
954     p7_cms_1_5_1    java=776,xml=178
943     filerule_maven_plugin_2_0 java=786,xml=157
931     p7_cms_1_2_0    java=726,xml=205
890     jbasis_crypto_4_2_1 java=787,xml=103
```

```
772     scytl_math_1_1_0 java=506,xml=193,ansic=73
700     nswec_ansibles_1_8_7_1 python=516,xml=184
555     aux_nswec_rabbitmq_spring_webapp_1_0_0 sh=281,xml=149,perl=99,java=26
522     property2json_maven_plugin_1_5_0 java=477,xml=45
516     maven_generic_conf_1_5_6 xml=516
504     maven_generic_conf_1_5_5 xml=504
409     invote_plugin_manager_1_4_1_1 java=288,xml=121
296     maven_generic_conf_2_0_2 xml=296
279     maven_generic_conf_2_0_0 xml=279
256     maven_dependencies_1_2_0 xml=256
191     js_forge_0_6_8_1 javascript=104,xml=87
181     p7_cms_1_5_1_1  xml=181
0       online_voting_logging_1.3.6.1 (none)
0       rabbitmq_spring_webapp_2_0_0 (none)


Totals grouped by language (dominant language first):
javascript:    593941 (45.44%)
java:          427715 (32.73%)
xml:           227204 (17.38%)
ansic:          31923 (2.44%)
python:         19782 (1.51%)
cpp:             4907 (0.38%)
sh:               946 (0.07%)
lisp:             264 (0.02%)
cs:               211 (0.02%)
perl:              99 (0.01%)



Total Physical Source Lines of Code (SLOC)                = 1,306,992
Development Effort Estimate, Person-Years (Person-Months) = 374.21 (4,490.53)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 5.09 (61.07)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 73.53
Total Estimated Cost to Develop                           = $ 50,550,819
 (average salary = $56,286/year, overhead = 2.40).
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
SLOCCount is Open Source Software/Free Software, licensed under the GNU GPL.
SLOCCount comes with ABSOLUTELY NO WARRANTY, and you are welcome to
redistribute it under certain conditions as specified by the GNU GPL license;
see the documentation for details.
Please credit this data as "generated using David A. Wheeler's 'SLOCCount'."
```