Scytl

Innovating Democracy

KRD

eValg2013

Audit Technical Report - Scytl clarifications

Version 1.2

08/11/2013

Scytl - Secure Electronic Voting

Revision chart

Version	*Date*	*Primary Author(s)*	*Reviewed by*	*Description*
1.0	29-10-2013	SG, JC, CT, DK, DA	JP, GDS, DA, LB	Initial reviewed document
1.1	07-11-2013	CTL, BS	DK	Complementary information (par 2.1).
1.2	08-11-2013	DK, CTL, BS	DK, GDS, PS, LB	Internal review remarks

INDEX

1 Introduction

Following the Norwegian Technical Report written by Tor E. Bjørstad - Oslo, 2013-08-06 - (referenced as the “Technical Report” throughout this document) of the source code audit of the Norwegian electronic voting system (requested by the Ministry of Local Government and Regional Development), the present document has been produced by SCYTL to provide additional clarification.

The Technical Report mainly presents a list of detailed findings relative to specific observations regarding the Norwegian Voting system source code. For each of these findings, the review document provides detailed answers and justifications. The main conclusion is that none of the findings of the source code audit represent security risks, as we explain in detail for each item.

Nevertheless, due to the continuous improvement of the solution, most of these findings have already been taken into account and the corrective actions are already planned in the project. The others will be addressed in future development for a general refactoring of the solution.

2 General comments

2.1 About software quality

The Technical Report is focused on the analysis of the cryptography used in the Java implementation of the electronic voting system. However, due to the fact that some subjective impressions regarding the overall quality of the e-voting project were stated, we have included additional comments relative to software quality.

From the beginning of the project in 2010, the system has evolved around the development of a well understood robust and modular next generation cryptographic protocol. Therefore, the electronic voting solution represents a large and complex project, with strict security requirements. Some of the impressions given in the Technical Report (i.e. relative to the code duplication or the difficult identification of the classes used in production) are in most cases correct and are a consequence of the complexity of such a large-scale system. Aware of this risk, the objective, during the development of the 2013 project, has been to improve the quality of the solution. This has been done by:

- maintaining the whole integrity;
- avoiding the refactor of the whole solution due to the time constrains and new features requested.

For that reason, improvement activities have been performed during the project such as the implementation of specific refactors, deletion of dead code,… It enhanced the previous software quality level, guarantying at any moment the integrity of the solution in parallel with the new developments.

For the 2013 project we used Sonar, an external application to perform static and dynamic Java code analysis based on industry well known standards. Sonar is the core platform for monitoring the metrics from findbugs, checkstyle, PMD, CPD and other tools. The rules and alarms used by these different tools have been customized, in accordance to the project objectives. The goal for the code used in the last election was to fix all the blocker issues as well as reducing the critical ones in a meaningful way, while not adding any violation for the new implemented code. Nightly automatic warnings were raised to cater for the case of the defined rules not being respected to allow an immediate fix. It is important to consider that fixing a violation usually implies a refactoring of the code in order to reach the quality standards. We enhanced and refactored the code on regular basis, to improve the previous rules compliance. The metrics results of this tool have been provided weekly to the Ministry of Local Government and Regional Development.

Regarding the unit tests of the application, we can use the following information as an indicator of the average level of coverage in real scenario projects, in order to evaluate the current and expected status of the eVoting system: Sonar collects a great deal of open source projects and provides information about their coverage and other metrics (http://nemo.sonarqube.org/dashboard/index/MASTER_PROJECT).

Currently we have information of 204 open source projects from different Forges, data from over 10 million lines of code. Their average test coverage is 26.7%. There is history of the projects since 2010 where the average was 19.0% showing a steady increment trend of the coverage. Current coverage of eVoting is 37% which is 11% more than average.

The objective is to keep the coverage on higher levels, but we should not lose focus of the real objective, which is deliver quality code. Coverage does not necessary guaranty the quality of the code, so it should not be blindly followed. Also, increasing code coverage has a real development cost (make code testable, abstract, dependency injection, mock objects, write test cases) and there are also situations where tests might not achieve a sufficient return-of-investment depending to the complexity, criticality, frequency of specific code use and other variables.

A good strategy in order to get the most of the effort of improving code coverage is to focus on specific cryptographic or other critical modules and provide higher quality standards on those. This strategy is one of our current improvements.

The Technical Reports comments the apparent “no-use” of the automated testing. This statement is incorrect. In fact, it is one of the most important improvements made during the 2013 project development: the extension of the continuous integration environment (CI). In addition to the existing automated unit and integration tests, this environment has been also be configured for the end-to-end tests, where all modules interact with each other, the deployment is automatically done in a Production-like environment and tests cases are written by QA engineers using Behavior-driven development (BDD). This has been a major achievement as it allows guaranteeing a stable version throughout the implementation phase. Consequently, this Continuous Delivery (CD) platform gives a repeatable and reliable process for releasing/deploying software.

QA environments were also improved by means of Private Cloud (OpenNebula) & Automatic Deployment (Puppet), providing the following advantages: self-service, consistent and replicable QA environments, automatic installation of all applications (3rd party + proprietary) and a centralized configuration management under version control.

Regarding the documentation, as reported, the high level documentation is of an excellent quality. However, the low level documentation is not enough to make the understanding of the source code for external reviewers easier. Therefore, the proposed recommendation of the Technical Report relevant to the link between the high level architecture and the low level implementation has been taken into account for the future, in addition to the identification of the modules of the source code that are not used for production (i.e. the code and tools implemented and used only for testing purposes). These improvements of the documentation will facilitate a better understanding of the low level of the system in the future.

2.2 About security

The Technical Report describes detailed findings that the auditor identified during the review of the Norwegian Voting system source code. The most remarkable findings are related to few mistakes when using default keysizes from old algorithms in new ones (e.g., SHA2-MAC function seed by a 192bits key instead of 256bit). However, they did not represent any security issue when evaluated in the context where these algorithms were used. In the next iteration of the voting system these inconsistencies will be solved. Also, the algorithms will be revised to avoid inconsistences and provide better homogeneity (e.g., to use always SHA2 algorithm instead of SHA1 or AES intead of 3DES).

3 Items detailed explanation

In the following subsections, we will provide our feedback to each of the finding summarized in Appendix A (List of findings) of the audit report.

3.1 [2.3] - Old versions of third party security libraries

Regarding the detected vulnerability, the library Spring Security core is only used in Authentication service code (v 3.0.0.RELEASE), in fact it is referenced in pom file but it has never been used through the code. We only use spring-security-web (v 3.0.3.RELEASE, which should be updated) and the vulnerability mentioned would be present only if we had really used spring-security-core (more details at http://support.springsource.com/security/cve-2011-2894)

This library not being used (spring-security-core) has been removed from pom.xml for the future. Update of existing 3rd party libraries will be done as part of future proposed improvements.

3.2 [3.5] - Use of SecureRandom with default RNG and provider

The finding does not represent any issue in the voting platform since the use of the default provider is secure in the Norwegian voting system infrastructure.The implementation is designed for Linux systems and therefore, it takes the default Linux system provider for RNG, the NativePRNG which:

- In case of not receiving a seed, the first time it is called it is automatically seeded with system entropy (/dev/random).

- When enough new entropy is available, it is reseeded again (automatically).

3.3 [4.1.1.1] – CryptoFactory

This finding is not representing currently any security issue to the voting platform. The places where the default values are used, despite they are not the same for the protection of the votes, are kept there for compatibility:

- The voter credential and electoral board share certificates are signed with the algorithm "Sha1withRSA": this is still broadly used since most of the validators of digital certificates do not support signature based on sha2. In any case, since the use of voter certificates is limited to the election, this will be changed to SHA2.
- The RSA shares stored in PIN protected smartcards are ciphered with the default algorithm "DESede/ECB/PKCS5Padding": In detail, the ciphered shares and the symmetric key are respectively stored in the public and private areas of the card. This approach was used for backward compatibility with former cards where the private area was too small to accommodate the shares. The system will be changed to store the shares in clear in the

private area for current models of smartcards (with larger private areas). For older models, where the private part is too small, the encryption of the shares will be changed to AES.

3.4 [4.1.2.4] – BCPKCS7Envelope

This finding is related to dead code not used in the Norwegian project. The code will be eliminated to prevent misunderstandings.

3.5 [4.1.3.1] – BigNumber

BigNumber is a wrapper over Java's BigInteger, used to provide some improved functionalities over the latter. For methods common to the BigInteger, BigNumber behaves the same, and so does regarding the initialization of a SecureRandom passed in the constructor.

3.6 [4.1.3.1] – BigNumber

This finding is not an issue in the Norwegian voting system since the implementation of timing side channel attacks are not feasible in practice in this system.

In a scheme where an entity signs input messages, in order to obtain the signing private key through a timing attack, an attacker needs:

- Control of the input to be signed: influence the content of the message.
- Accurate timing of the signing time.
- A big number of signatures over different messages.

For the first part, it has to be taken into account that in the eVoting protocol most of the signatures are performed over data computed by the components themselves, not over input messages which come from other components.

For the second part, most components perform several operations over input data, before performing a digital signature on the output data generated. Therefore, guessing the exact timing corresponding to the signature is infeasible. Also, online components are behind a load balancer, communications are subject to network delays, and components may have different workloads. Therefore the response time will vary according to things other than the signature timing.

Regarding the third part, the eVoting protocol implies that there are certain rules that have to be accomplished by input messages in one component, so that it does its computations and produces a digitally signed output. There are few occasions where a component could be seen as a "signing service", so that a large number of signatures may be obtained from it. In such cases, the digital signatures are performed over data that cannot be influenced by an attacker.

3.7 [4.1.4.2] – ShamirShadowManager

This finding does not represent a security risk as explained below. However the recommendation from auditor will be considered and the correct range of the Shamir specification will be implemented to fulfil with the original secret sharing scheme.

In Shamir Secret Sharing Scheme, a secret “s” is divided in a number “n” of smaller secrets, which we will call shares from now on, in a way that if a threshold “t” of shares are put together then the secret “s” can be reconstructed whereas if “t-1” shares or less are put together no information about the secret “s” is learnt.
The scheme uses what is known as polynomial interpolation with random polynomials. There is a security proof that says that if “t-1” or less shares are put together, then there is literally no information about “s” leaked. This is, every possible secret “s” is equally probable.

The mistake in the code causes the polynomials used in the polynomial interpolation to not be as random as they should be. This implies that every possible secret “s” is not equally probable, which means that there are secrets which are more probable than others.
However, an in-depth analysis of the vulnerability shows that, although there are secrets which are more probable than others, the difference of probabilities is negligibly small. This means that when given “t-1” or less shares the probability that “s” has the most probable value is really close to the probability that “s” has the least probable value.
This analysis shows that the information leaked by “t-1” or less shares is negligibly small, which means that it doesn’t imply any significant security risk.

3.8 [4.1.4.3] – FileConnection

This is a finding over test code for making easier the testing process. In production code the shares are always stored in smartcards. Therefore, there is not any security issue.

3.9 [4.1.4.3] – TokenConnection

As mentioned in Section 3.3, in the next revision of the code, an AES encryption approach will be used in case the shares need to be encrypted.

3.10 [4.2.1.1] – AsymmetricCipher

This finding does not pose any security issue since it is related to test code.

3.11 [4.2.1.3] – JCEEnvelope

Any potential issues related to this finding do not pose a real security issue in the Norwegian voting system since the JCEEnvelope function is used in the Key Management System (KMS) environment.

In IND-CCA2, the adversary is given access to the public key, and to a decryption oracle, which is queried by the adversary in order to get information about the decryption of arbitrary ciphertexts.
The use of the JCEEnvelope encrypt/decrypt functions is limited to some functionalities in the offline components, and to decrypting configuration data received during configuration in the online components. The online components, which are the ones more exposed to a chosen ciphertext/adaptive chosen ciphertext attack because they can be reached from _the outside _ (uncontrolled environments), use the JCEEnvelope functions in order to decrypt some properties files provided during configuration from the eConfig machines.
The processing of these encrypted files is limited to those received from the eConfig machines, since the use of RBAC tokens and Administration Board signatures over such files are verified. Therefore, the only entity with access to a decryption oracle is the eConfig machine, which is the only potential IND-CCA2 attacker. Since such machines are trusted (from the point of view that they are part of the KMS environment, and are in charge of sending the configuration to the platform components), there is no possible IND-CCA2 attack because there is no IND-CCA2 attacker.

3.12 [4.2.1.5] – SymmetricCipher

This finding does not represent an issue since only applies to test code.

The SymmetricCipher class is used in com.scytl.evote.protocol.managers.rcmanager.encryptedCode and com.scytl.evote.protocol.managers.rcmanager.encryptedCodeColl in order to encrypt the short return codes with the long return codes, to be stored in the RCG database.

There are two constructors, one using the symmetric encryption algorithm by default, and another one which receives the encryption algorithm as an input parameter. The second one is only used in tests, so the first one is the one used in production. The following are the default constants:

```
private static final String CIPHER_ALGORITHM_DEFAULT = "AES/CBC/PKCS5Padding";
private static final String PROVIDER_NAME_DEFAULT = "SunJCE";
```

Therefore, the encrypt/decrypt methods will always be used with a block cipher.

3.13 [4.2.3.1] – BlockElGamalEngine

This issue does not pose any issue in Norwegian voting system since auditor rationale does not apply to it.

Audit report says that BlockElGamalEngine is in charge of dividing a bytearray in blocks, so that each block is individually encrypted. However, the BlockElGamalEngine does not do that, but manages just one block. Therefore, there is no purpose in discussing the possible chaining of multiple blocks.

The BlockElGamalEngine implements a primitive in charge of processing one block, like an AES engine would do: a higher level function has to be in charge of chaining multiple blocks processed by the underlying engine.

Also, this code is not actually used in production, but the one that receives BigIntegers to encrypt/decrypt instead of bytearrays.
Finally, in a higher level, multiple options that may be individually encrypted are "linked" by Schnorr Signature and RSA Signature, so that they are not malleable at all.

3.14 [4.2.6.2] – RCGCrypto

This finding does not pose any security issue for the Norwegian system, since it is used to select a random number between a small set of numbers. Therefore, initializing the PRNG with a timestamp does not give any advantage to an attacker.

If we look at the potential vulnerability of using Date() as the seed of the PRNG, which is used to decide if the RCG verifies of not the proofs generated by the VCS (1/8 times):

- In order for the VCS to guess which proof will be verified (so that it is not caught by the RCG), the VCS can either directly guess the number generated by the SecureRandom(new Date()), or the value of Date(), since the number can then be directly computed from such value.
- Using the first strategy, VCS has a probability of 1/8 to make a correct guess. This has been specified in the protocol reference and accepted as a probability low enough of a VCS cheating not to be caught for one vote.
- Using the second strategy, considering that new Date() returns the date with milliseconds accuracy, the VCS should be able to guess the value returned by new Date() at the RCG, with a margin of error of 10ms, in order to have roughly the same probability of winning than in the first strategy. Although VCS and RCG are synchronized, the exact time (within a margin of 10ms) when the RCG will process the VCS request depends on many environmental factors, such as load balancing, network traffic, the fact that there are several RCG that may attend the petition of the VCS, and other similar issues.

Therefore, it can be considered that the use of new Date() for computing the number does not gives an extra advantage to a malicious VCS, and so this does not result in a vulnerability in the application.

3.15 [4.2.6.3] – VCSCrypto

This issue does not pose any issue in the Norwegian voting system since it is implemented for testing purposes. Test data misses digital signatures to prevent being confused with production data.

The method partiallyDecryptVoteWithoutSigning is used to verify that the correct return codes have been uploaded to the database. To this end, an alternative path than the usual is used to generate the return codes from test votes. In the alternative path it was specified that no signatures were used, just as an additional control to ensure that a test vote will never be considered as a vote to be counted.
Test for signing keys is done with other service checks.

3.16 [4.2.6.4] - HardcodedElection- ManagementService

This finding does not pose any issue in the Norwegian system since it is related to test code.

The passwords and PKCS#12 identifiers defined in the HardcodedElectionManagementService class are not used in the production environment. Methods getMXSManagerP12() and getAuditP12(final String string) are not used in production, while the others methods affected are overridden by the RemoteElectionManagementServiceImpl class in the production code.

3.17 [4.2.6.5] – CredentialsGeneratorJCE

This finding it is not considered and issue for the Norwegian voting system but a requirement of the original specifications of the system.

Voter certificates are not intended to be anonymous. It is defined that the common name is the hash of the voter's SSN.

3.18 [4.2.6.5] – CredentialsGeneratorJCE

This finding is not considered an issue in the Norwegian voting system but a required update to support the use of PKCS#12 compliant key containers in Javascript implementations (i.e., the voting client).

CredentialsGenerator uses the BouncyCastle provider to generate the PKCS#12 containers that store the voters' signing keys. However, PKCS#12's key containers created with BouncyCastle are not fully compatible with the PKCS#12 code implemented in the Forge JavaScript library. This issue is not present when PKCS#12 containers are created using the SUNJCE provider instead of BouncyCastle one. Therefore CredentialsGeneratorJCE new class was generated in order to use SUNJCE provider instead.

3.19 [4.2.8.2] – HashedReturnCode

This finding is not considered an issue in the Norwegian voting system and was implemented that way in purpose (space saving), evaluating that does not represent a security issue.

In this case the hash is done over the Return Codes, and it is used in the RCG database in order to locate the linked encrypted short codes. The hash is truncated to 160 bits due to space requirements and the potential issue is to have a collision.

However, this collision cannot happen since it requires at the time of generation, the generation of 2^80 Return Codes to have a probability of 50% of finding a collision in their truncated hashes. Note that the collision should be found among the Return Codes belonging to a specific voter and election type (referendum, parliamentary/sami/municipal/county). The number of Return Codes for one voter is estimated to be much lower. For example for a parliamentary election with 10 parties, where the largest party has 20 candidates, 451 Return Codes are generated.

3.20 [4.2.9] - com.scytl.evote.protocol.signers package

As mentioned by the auditor, there are no security issues in the Norwegian voting platform. Comments will be considered for next version.

3.21 [4.2.10.2] – GeneratorSelector

This finding does not pose an issue in the Norwegian Voting platform.

This GeneratorSelector class receives the ElGamal parameters p,q and generates a corresponding generator of the required order. The generator is generated accordingly to the size of the parameters p, q, and not to the size specified in the input.

However, we will consider in the next iteration to check the size of p and q against the size specified, at least to see if it matches.

3.22 [4.3.1.1] – CryptographicConstants

This finding does not pose a security issue, since the key length is still strong enough and any practical attack requires a high level of interaction with the system. Furthermore, the HMAC key affected has a short period of life, so it may be infeasible to find such value before it is actually published at the end of the block.

In any case, the inconsistency on the key length will be corrected.

3.23 [4.3.2.1] – UtilsRBAC

This finding does not pose a high security issue if proper passwords are used.

The RBAC tokens and the symmetric key generation for VCS and RCG are the parts that have a higher impact due to the security issue described. Nevertheless, the risk and impact can be reduced or is low. For the RBAC tokens protection the mitigations given can reduce the risk. And for the symmetric key generation the risk is not high if good security practises were followed when the passwords were chosen.

However, this code will be improved in the next version.

3.24 [4.3.3.1] – TemporalDigestedFile

This finding does not represent any issue, since the use of the hash function is just to check if the file has been updated while processed. If so, the contents are read again. Therefore, it is not used for security purposes.

3.25 [4.4.1] - com.scytl.slogger package

Our system internally expects to have UTF-8 encoding when converting String to byte[], and the System requirements are defined specified this locale for every server. Therefore the server infrastructure was configured in consonance and this finding did not pose any security issues. Improvements on the secure logger library will be done as part of future proposed development.

3.26 [4.4.1.7] - SignedEncryptedHMac- EventProcessor

This finding does not represent an issue, since the deletion of logs from bottom to up will be detected when checking the periodicity of the checkpoints.

The secure logger makes a checkpoint (digital signature) every specific number of logs or time after last checkpoint. The elimination of logs from bottom will eliminate these checkpoints and therefore, this attack can be detected.

3.27 [4.4.1.8&9] - SignatureEventValidator / HMAC validators

This finding does not pose a security issue in the logs, but a missing validation process of the checkpoint periodicity in the immutable logs. Furthermore, the voting system also makes a parallel register of the logs in a central server, so it is also possible to retrieve the correct log information.

The manipulation of logs from the bottom will be detected, since this manipulation will not be validated without the checkpoint (i.e., without access to the private key).

However, in the next release the validator will be updated with the missing verification of the checkpoint integrity.

3.28 [4.4.2.1] – AuditSecureFileAppender

This finding does not pose any security issue.

In the voting system the code devoted to use the Secure Logger, in the class AuditSecureFileAppender, there is an extension of the already existing class SignatureEventProcessor. The new implementation of the class overrides the method "sign" and allows the usage of a different cryptographic provider. The reason of this is to allow the usage of the HSMs in the secure logger when they are available.

3.29 [4.4.2.2] – RemoteConnector

This finding is related to old code not currently used in the voting system. It will be deleted to prevent additional confusion.

3.30 [4.5.1.1] – Utils

This finding is related to test code and therefore, it does not pose any security issue.

3.31 [4.5.2.1] – InteractiveKms

This finding is related to code not in use.

The InteractiveKms code is no longer used in the project since it was completely removed in KMS from version 3.2.5 and above.

It will be eliminated to prevent any confusion.

3.32 [4.5.2.7] – SymmKeyGenerationCommand

The risk of this finding is limited, since only affects the controlled environment in case a weak password is selected.

The next iteration of the voting system will implement a random initialization vector when encrypting and the standard recommended number of iterations (at least 100000) in the PBKDF.

3.33 [4.5.2.2] – CAGenerationCommand

This finding is known and it is mitigated using physical and procedural means.

3.34 [4.5.2.3] –RSAGenerationCommand

This finding is known and it is mitigated using physical and procedural means.

3.35 [4.5.2.8] –VoterCredGenerationCommand

This finding is known and part of the design of the cryptographic protocol used by the voting system. It does not represent a security issue since the identity of the voter does not depend on the voting system key containers.

The voter identity is ensured by the identity token provided by IDPorten, not by the voter ability to open the PKCS#12.

Access to the PKCS#12 keystores is restricted to the Auth Service during the voting phase. The Auth Service cannot take advantage of the access to such keystores in order to perform ballot box stuffing, since it should be able to get a MinID identity token for each fake vote.

The requests to MinID identity tokens are alerted through the voters by SMS. The validity of a MinID identity token is checked by several components of the eVoting platform (VCS, RCG, Cleansing).

4 Conclusion

As demonstrated in the findings described in the Technical Report, these do not have an impact on the security of the system. Even if the software quality could be enhanced, the software development mechanisms introduced, like the continuous integration/delivery for automated testing and the code metrics improvements, allowed ensuring the integrity of the solution.

The Technical Report confirms the need to continue the improvement process already initiated during the 2013 project. They represent important suggestions and recommendations to complement our future plans and solutions. The solution has been used in two elections, successfully providing a stable voting process protocol. Currently, action plans are in progress to take into account the improvements of the technical aspects for the whole solution. Clearly, all the observations that apply to the eVoting solution will be a part of this process.

Innovating Democracy