UNIVERSIDADE ESTADUAL DE CAMPINAS
Cidade Universitária Zeferino Vaz
Campinas - SP

Cooperação Unicamp-TSE

Para Inspeção do Código-Fonte da Urna Eletrônica

Relatório Final

12 de agosto de 2022

1. Preâmbulo

O presente relatório final de atividades diz respeito à cooperação estabelecida entre o Tribunal Superior Eleitoral e a Universidade Estadual de Campinas, por meio de Termo de Adesão assinado pelo reitor desta Universidade em sete de abril de 2022. Em virtude da baixíssima disponibilidade de tempo das pessoas capacitadas a trabalhar neste tema, os trabalhos foram efetivamente iniciados dois meses mais tarde, após a montagem de um laboratório com acesso restrito, contendo o servidor que abrigou o código-fonte da urna. Em virtude dessas restrições, decidiu-se dar atenção a certos tópicos específicos, considerados sensíveis. As análises foram feitas apenas por meio de exame dos códigos-fonte, sem compilação e execução dos mesmos, já que não havia tempo hábil para a montagem de um laboratório completo com urnas e sistemas de apuração. Um exame crítico completo do código-fonte da urna necessitaria do aporte de milhares de horas de trabalho de uma equipe multidisciplinar; desta forma, a cobertura da análise realizada pela presente equipe deve ser considerada, para quaisquer fins, amostral. Uma vez que é difícil determinar quantitativamente a cobertura da nossa análise, procuramos deixar explícitos os diretórios e arquivos examinados.

1.1 Sobre o objeto da nossa análise

Os tópicos que foram objeto da nossa análise são os seguintes: (i) a votação paralela realizada em uma parcela das urnas; (ii) o algoritmo empregado para armazenar votos no Registro Digital de Votos; (iii) o app Vota, de controle da urna; (iv) o emprego recomendável de rotinas criptográficas. Tratamos de cada um desses aspectos em seções separadas deste documento.

1.2 Sobre a versão do código examinado

Nosso trabalho foi feito sobre a versão Timbu do código-fonte, que foi a disponibilizada inicialmente pelo TSE, com a seguinte especificação:

- **Gedai-UE v 6.12.1.0**
- **UENUX v 8.13.1.0**
- **Pada-UE v 2.6.4.0**
- **Sorteio e Votação, ambos v 5.7.1.0**

Estamos trabalhando, presentemente, num adendo a este relatório, com os mesmos tópicos sob análise, cobrindo uma nova versão do código-fonte.

1.3 Sobre a equipe

A equipe que participou deste trabalho foi formada por alunos, professores, servidores e colaboradores da Unicamp, todos signatários de termo de sigilo junto ao TSE. O professor Ricardo Dahab, diretor-geral de TIC da Unicamp, foi designado pelo reitor da Universidade para coordenar os trabalhos. É ele quem assina este relatório, em nome da equipe.

2. Análise da votação paralela

Resumidamente, a votação paralela consiste na simulação da votação com urnas oficiais alimentadas com as listas oficiais de candidatos e de eleitores.

Durante as reuniões para delimitação de escopo do trabalho, um possível cenário de ataque foi especulado, na forma da existência de código malicioso – ou mero erro programático – na urna, com o efeito de alterar votos individuais durante a eleição real. Porém, tal comportamento não poderia se manifestar durante a votação paralela, sob pena de ser descoberto. Portanto, tal código inadequado necessitaria ser acompanhado de outros códigos (ou por excepcional alinhamento de erros programáticos) que detectam, indiretamente, se a urna encontra-se em votação paralela e alteram seu comportamento. Tal detecção deve ocorrer por meio de particularidades da votação paralela que não se espera que ocorram numa votação real; por exemplo, uma das possibilidades aventadas foi uma alta proporção de eleitores sem o uso de identificação biométrica, ou a maior frequência de votos sendo inseridos do que em condições normais.

Assim,, foi realizada uma revisão dos códigos (por meio de análise estática) relacionados ao ciclo de habilitação do eleitor nos arquivos em *uenux/src/app/vota/*, em particular,

- *cmostraeleitorvotando.cpp,*
- *cmostraeleitorvotando.h*,
- *ccontrolareconhecimento.cpp,*
- *ccontrolareconhecimento.h,*
- *cnomeeleitor.cpp,*
- *cnomeeleitor.h,*
- *cpedetitulo.cpp*,
- *cpedetitulo.h*,
- *cpedeanonascimentosembiometria.cpp,* e

- *cpedeanonascimentosembiometria.h,*

além de outros com funções comuns a esses, em busca de código potencialmente malicioso que pudesse corresponder a um ataque, ou erro, como descrito. Entretanto, nenhum código que pudesse auxiliar na implementação de um ataque ou na ativação de um erro nestes moldes foi localizado.

3. Análise do algoritmo para registro do voto

Uma das formas de verificação dos votos depositados na urna eletrônica é o chamado Registro Digital de Votos (RDV), em que cada voto é armazenado num arquivo. Se a ordem cronológica em que os eleitores votaram puder ser correlacionada com a posição que os votos ocupam no RDV, cria-se a possibilidade de um ataque que violaria o sigilo do voto. Como é virtualmente impossível evitar que um observador externo registre a ordem em que os eleitores votaram, é crucial que o algoritmo que determina a posição de cada voto no RDV não permita correlação alguma desta posição com a ordem de inserção dos votos.

Mediante análise de arquivos presentes no diretório *unenux/src/app/comum/dados*, em particular,

- *crdvvota.cpp*,
- *md/rdv/cvotoscargos.cpp*,
- *md/rdv/cvoto.h,* e
- *md/rdv/cvoto.cpp*,

constatou-se que o algoritmo usado essencialmente implementa o algoritmo clássico de ordenação por inserção. Cada voto, em cada um dos cargos disponíveis na eleição corrente, é armazenado em um vetor específico para aquele cargo, em uma posição escolhida de forma a manter o vetor ordenado a cada inserção; tal posição é determinada no método *CRdvPosicionadorVota::GetPosicaoInsercao(),* em *crdvvota.cpp*.

A função de comparação *bool operator<(const CVoto & lhs, const CVoto & rhs)*, em *md/rdv/cvoto.cpp*, codifica uma ordem lexicográfica primariamente sobre o tipo de voto – definido pela *enum ETipo* em *md/rdv/cvoto.h*, contendo opções como legenda, nominal, branco, nulo, etc, a qual, em virtude de ser do tipo *enum*, possui uma representação numérica associada que induz uma noção de ordem sobre os tipos de voto – e secundariamente sobre o número digitado. Em princípio, o uso de um algoritmo estável de ordenação, como é o caso da ordenação por inserção, poderia preservar a sequência cronológica de inserção dos votos; entretanto, o fato da ordem lexicográfica definida ser uma ordem total sobre os dados armazenados afasta tal possibilidade. Com efeito, para um dado conjunto de votos, todas as permutações possíveis (induzidas pela sequência de comparecimento dos eleitores) são mapeadas para uma permutação única possível, que é a ordem lexicográfica (como anteriormente definida) de tais votos. Portanto, concluímos que a forma de inserção dos votos no RDV não compromete o sigilo do voto.

4. Análise de alguns aspectos relacionados a rotinas criptográficas

4.1 Possíveis maus-usos em chamadas de bibliotecas criptográficas

O mau-uso de bibliotecas criptográficas por programadores não-especialistas em Criptografia é fenômeno comum e pode acarretar não só problemas de compatibilidade, que causam erros em funcionalidades de aplicações, mas também introduzir vulnerabilidades nem sempre visíveis num exame superficial do código.

Entre os diversos tipos de maus-usos de bibliotecas criptográficas está o emprego de métodos criptográficos que caíram em desuso pela comunidade, por uma razão ou outra. Com o objetivo de encontrar a possível ocorrência desses métodos obsoletos, os seguintes diretórios do código-fonte foram analisados:

- *uenux-kernel/src/kernel/linux*/crypto/assimetric_keys* e
- *uenux/src/app/verificador*.

No primeiro diretório é possível encontrar referências a algoritmos obsoletos, por não serem mais considerados seguros, entre eles MD5 e SHA1. Entretanto, tais ocorrências não são compiladas, e portanto não fazem parte do código binário da urna, mas estão presentes no código por serem parte de software legado. Outros diretórios não foram analisados por questões de tempo (exceto os mencionados na próxima seção relativa ao app Vota); logo, não é possível afirmar o mesmo para os outros diretórios. A seguir são apresentadas as ocorrências e os respectivos arquivos de código-fonte presentes no diretório *uenux-kernel/src/kernel/linux*/crypto/assimetric_keys*:

1. */assym_tpm.c:*
    - SHA1 utilizado para MAC (ainda é seguro para essa aplicação específica);
    - PKCS1v15 (RSA) utilizado com *padding* como algoritmo padrão;
    - Referências a algoritmos inseguros como o MD5 e SHA1.
    - Nota: não é compilado.
2. */mscode_parser.c*:
    - Na função *mscode_note_digest_algo*: Referências a algoritmos inseguros como o, MD4, MD5 e SHA1.
    - Nota: não é utilizado na urna, já que é parte de códigos binários para MS-Windows.
3. */pkcs7_parser.c*:
    - Na função *pkcs7_sig_note_digest_algo*: Referências a algoritmos de hash inseguros como o MD4, MD5 e SHA1. De fato, utiliza-se o ECDSA com o SHA256/384/512 para assinatura, que são considerados seguros.
4. */verify_elffile.c*:

    - Aqui se observa que apenas o SHA512 e o RSA com *padding* para certificados PKCS7 podem ser utilizados para assinaturas. De fato, o algoritmo utilizado é o ECDSA P521 com SHA512, considerado seguro.
5. */x509_cert_parser.c*:
    - A função *x509_note_pkey_algo* aceita certificados x.509 com algoritmos inseguros como *md4WithRSAEncryption*, *sha1WithRSAEncryption*, além de outros.

No caso do diretório *uenux/src/app/verificador*, em que está o código-fonte do app VERIFICADOR, que verifica os arquivos estáticos da urna, foram constatadas apenas duas ocorrências de funções que utilizam métodos criptográficos. Em ambos os casos, os algoritmos utilizados ainda não são considerados obsoletos. A seguir, as ocorrências:

1. */TSE/verificador.cpp*:
    - A função *verificaAssinaturas* verifica as assinaturas com o SHA-512 nos certificados dos arquivos estáticos da urna. Aqui, o algoritmo utilizado para encriptação assimétrica é o RSA-PKCS1 com *padding* seguro.
    - Função *verificaArquivo*: verificação de hash feita também com o SHA-512, considerado seguro.

Essas foram algumas das ocorrências de uso de algoritmos criptográficos verificados. Nem todas têm relação direta com a aplicação executada na urna e as demais ocorrências localizadas em outros diretórios necessitam ser igualmente verificadas. Uma sugestão final é a de que, apesar de ainda não ser considerado obsoleto, substitua-se o algoritmo SHA-512 por algoritmos mais modernos em versões futuras do código-fonte da urna. Além disso, seria prudente, como medida saneadora, remover completamente qualquer menção a métodos criptográficos obsoletos do código-fonte.

4.2 Considerações sobre alguns diretórios do app Vota, e em particular sobre rotinas criptográficas

A análise nos diretórios dentro de uenux/src/app/vota teve como objetivo compreender melhor as funções e fluxos do sistema para habilitação, monitoramento e verificação de identidade do eleitor. Nestes arquivos, nada foi encontrado que parecesse problemático, mas isso não significa que nada exista. Nossa análise se limitou à leitura e estudo do código, sem testes rigorosos ou tentativas de ataque, o que não constitui prova experimental nem matemática de que a implementação seja, de fato, segura. O que pode ser afirmado é que, dentro do contexto da análise e da limitação de tempo, nada suspeito ou incorreto foi encontrado nesta revisão do código.

Em particular foram analisados os diretórios a seguir.

- *uenux/src/app/vota/comum*
- *uenux/src/app/vota/log*
- *uenux/src/app/vota/main*
- *uenux/src/app/vota/monitor*
- *uenux/src/app/vota/doc*

No arquivo *uenux/src/app/vota/main/vota.cpp*, é possível observar a inicialização de algumas funcionalidades criptográficas, que estão definidas e implementadas nos arquivos a seguir.

- *libecourna/src/api/security/ihash.hpp* e *.cpp* (funções de hash)
- *libecourna/src/api/security/irng.hpp* e *.cpp* (geração de números aleatórios)
- *libecourna/src/api/security/isymmetriccipherfactory.hpp* e *.cpp* (cifras simétricas)
- *uenux/src/api/hwil/ipkcs11.h* e *.cpp*

Sugerimos a análise aprofundada desses arquivos em trabalho futuro, pois:

- é preciso garantir uma implementação segura das diversas cifras simétricas, funções de hash e do gerador de números aleatórios;
- é preciso garantir o uso de primitivas atualizadas, e a substituição de primitivas antigas consideradas vulneráveis.

Foi encontrada, no arquivo *ipkcs11.h*, uma função *GenerateKey()* que, de acordo com a documentação, faz uso de um algoritmo chamado GUARANA para gerar uma chave simétrica que será usada posteriormente. Este algoritmo, conforme nos foi esclarecido pela equipe de TI do TSE, é um algoritmo classificado/confidencial. Dessa forma, não foi possível encontrar especificações a respeito dele ou publicações de avaliações de sua segurança. Idealmente, seria importante existir uma avaliação cuidadosa do uso e da segurança desse algoritmo, caso já não tenha sido conduzida.

Caso o GUARANA seja utilizado de forma conjunta, em camada, com algoritmo simétrico reconhecidamente seguro, a análise de segurança do referido algoritmo proprietário é dispensável, sendo apenas necessário garantir que (i) a função seja bijetora de forma que ela jamais altere o texto em claro, (ii) a composição com o outro algoritmo não constitua involução matemática.

Ainda sobre o GUARANA, nos foi informado pela equipe de TI do TSE que, embora a aplicação Vota inicie a interface PKCS11 do hardware de segurança, ele não utiliza as funções relativas ao GUARANA. Esse algoritmo somente é empregado no processo de inseminação da chave kernel no hardware de segurança, conforme descrito em https://sol.sbc.org.br/index.php/wte/article/view/14039.

5. Sugestão relativa ao aplicativo utilizado para os eleitores lerem o boletim de urna após o fim do horário de votação

Embora não tenha sido um dos objetos da presente análise, o aplicativo Boletim na Mão, disponibilizado pelo TSE para que os eleitores possam ler e decodificar o QR-code presente nos boletins de urna, fazemos, a seguir, uma sugestão a partir de uma perspectiva de usuário.

Segundo informações fornecidas em reunião realizada entre membros da equipe de TI do TSE e membros da equipe da Unicamp, o aplicativo de celular fornecido pelo TSE para os eleitores coletarem o resultado de cada urna (QR-code impresso no boletim de urna) necessitaria de acesso à internet para poder complementar as informaçẽos do boletim com chaves públicas de validação de assinatura, informações sobre a configuração do processo eleitoral (nomes das eleições e dos cargos, etc), siglas de partidos e nomes de candidatos. O motivo para isso é que o QR-code não contém todas essas informações, mas apenas algumas poucas, como o número do candidato e o número de votos recebidos.

Considerando que estamos em um momento de grandes discussões sobre a confiabilidade da urna e dos resultados por ela apurados, parece-nos recomendável que o aplicativo que lê e mostra os resultados do boletim de urna não tenha nenhuma necessidade de fazer conexão com o TSE para poder mostrar os resultados.

Em que pese a expectativa de número elevado de candidatos, acima dos mais de 28 mil das eleições gerais de 2018, se considerarmos que 5KB de informação por candidato permitiria a exibição de fotos, nomes e filiação e resultaria em armazenamento total de apenas 140MB (tamanho absolutamente compatível com os aplicativos bastante utilizados pelos brasileiros, a exemplo de WhatsApp 113MB, Netflix 108MB, LinkedIn 299MB), parece-nos viável uma estratégia de embutir no próprio aplicativo os dados básicos dos candidatos, da eleição e chaves públicas de verificação de assinaturas. Caso seja inviável, ou muito caro e complexo, embutir todos esses dados no aplicativo, ainda assim seria melhor, na nossa percepção, que o mesmo não se conectasse à internet, e mostrasse apenas os dados obtidos a partir do boletim de urna.

Acreditamos que seria mais transparente e menos sujeito a ações de desinformação se os eleitores pudessem ter a percepção de que os resultados que acabaram de ler com o aplicativo no celular vieram de uma única fonte, qual seja o QR-code do boletim de urna. Como benefício adicional, os brasileiros residentes em zonas sem conectividade poderiam também realizar a consulta de forma assíncrona.

6. Conclusão

Em que pese a cobertura limitada pelo tempo, dentre os tópicos examinados, que consideramos sensíveis, nada foi encontrado que possa colocar em dúvida a integridade e confiabilidade do código-fonte da urna eletrônica brasileira nos aspectos que compõem o objeto do presente trabalho:

I. na votação paralela, nenhum trecho de código foi encontrado que pudesse ser usado para detectar um contexto de votação fora do rito oficial;
II. no algoritmo para armazenamento de votos no Registro Digital de Votos, concluímos que não há correlação entre a ordem em que eleitores depositam seus votos e a ordem dos votos no RDV;
III. sobre possíveis maus-usos de rotinas criptográficas, nada foi encontrado no código que indicasse o real emprego de forma incorreta ou de métodos criptográficos obsoletos;

IV. no app Vota, de controle da urna, nada suspeito ou fora de padrões usuais de codificação foi encontrado.

Não obstante a ausência de achados negativos, algumas sugestões de melhorias foram apontadas, no intuito de emprestar maior transparência ao processo eleitoral.

Uma palavra final é sempre necessária em análises que tenham como objetivo mostrar a *inexistência* de alguma característica, seja positiva ou negativa, e em especial no caso de software: a menos que seja possível realizar um exame exaustivo do código, acompanhado de testes em condições reais de uso, as conclusões serão sempre parciais. Assim é com a presente análise. No entanto, tendo sido feita em condições favoráveis de acesso irrestrito ao código, traz consigo uma razoável carga de transparência e confiabilidade.

Prof. Ricardo Dahab
(pela equipe)