COS 432: Information Security Homework 5: Electronic Voting Security Review

Jacopo Cesareo, Ian Davey, and Nick Jones

(jhug note: the original looked nicer, but I had to PDF -> Word -> PDF to make changes)

Overview

In this assignment we were tasked with assessing the security of another group's voting machine design.

Threat Model

We feel it is useful to restate the assumptions given in Homework 4:

- Election workers are perfectly trustworthy, skilled, and diligent
- Smartcards will not necessarily be inserted into the voting machine in the same order they were inserted into the encoder; in fact, some smartcards may never be inserted into the voting machine;
- The adversary can see all of your code, except for the two files encoderSecrets.h and machineSecrets.h;
- encoderSecrets.h can be #included in your encoder program and your createCard program, but not in any other program,
- machineSecrets.h can be #included in your machine program and your createCard program, but not in any other program,
- The adversary cannot read the persistent memory of any smartcard;
- The adversary can make his own smartcards that run any code he likes, and he can insert those cards into your devices whenever he wants
- The adversary can make his own devices that run any code he likes, and he can insert your smartcards or his own smartcards into his device(s) whenever he wants.

Summary of Voting Machine Design

Their system uses authentication challenges between the smart card and the encoder/voting machine in order to authenticate one component to another. When the card is created, the smart card receives both the encoder and the machine shared secrets. All cards receive the same shared secrets. Even though we do not believe it is a good idea to store secrets on the smart card, this design does not make the system insecure given the assumption that the persistent memory can't be directly read by an attacker.

The smart card has two different authentication procedures it uses depending on which machine it is interacting with. When interacting with the encoder, the smart card sends srand(time(NULL)) to the encoder. In this scenario, the smart card is authenticating the encoder to ensure that it is valid. The smart card waits for the encoder to send back the hashed value of the random number it sent. The smart card then checks to ensure that they match. When they match, the smart card enables itself by setting the variable canVote to 1.

When the smart card is inserted into the voting machine, a reverse of the above authentication happens. In this case, the voting machine sends a random number to the smart card. The smart card must reply with the correct answer in order to vote. Before replying, the card sets canVote to 0. In this system, the smart card is only allowed to answer the challenge once ever (when *canVote* == 1). When the smart card has been challenged , it will not answer again unless re-enabled using the encoder.

The system is secure because the smart card correctly disables itself even before answering the challenge sent by the voting machine. An attacker has no way to steal votes since the smart card is essentially "one time use".

Security of Random Number Generation

We noticed that the group used srand(time(NULL))and rand()to generate random numbers. While this method is not considered cryptographically secure, apparently it was suggested/approved by the TA. A more secure option would have been to read from /dev/urandom (or even /dev/random) and use a PRG with the result whenever randomness was needed, assuming both the card and voting machine are running something UNIX-based or otherwise have access to these facilities.

Storing Private Keys on Voting Smartcards

We noticed that the group also chose to store the private keys for their voting system on the smart card device. In their system, all smart cards recieve the same secret keys. There are not separate keys for each card. The secret keys are copied onto the smart card whenever it is created using the *createCard* program. After discussing this with the T.A., we were advised that the *createCard* program is assumed to be out of reach of an attacker.

We feel that in a real world situation it would be a very bad idea to store secret keys on the smart cards. Although reading keys off of memory may be very difficult to do, it is not completely impossible. If an attacker were ever able to compromise a single smart card, the attacker would be able to commit election fraud at any scale he wished. Specifically, if an attacker were ever able to somehow insert a malicous device into the *createCard* program, he could intercept the secret keys. Consider the following scenario:

Assume a malicious attacker creates a smart-card like device which is inserted into the *createCard* machine. The malicious device does not actually write the data that it is given to its persistent memory. Instead, the card transmits its memory to a notebook PC, giving the attacker a full copy of whatever would have been written to the card. In our specific scenario, since the private keys are being stored on the card, the attacker now has perfect knowledge of all shared secrets. This enables the attacker to vote at will.

Conclusion

In conclusion, we feel that this group's code is secure within the given assumptions of assignment four. However, we feel that if this system were to be implemented in a real world scenario, it would have several vulnerabilities. In a real world system, we feel that a more robust source of random numbers should be used and that secret keys should not be stored on the smart cards.