FIREAXE: The DHS Secure Design Competition Pilot

[Extended Abstract]

Yevgeniy Vorobeychik, Michael Z. Lee, Adam Anderson, Mitch Adair, William Atkins, Alan Berryhill, Dominic Chen, Ben Cook, Jeremy Erickson, Steve Hurd, Ron Olsberg, Lyndon Pierson, and Owen Redwood[*]

ABSTRACT

Application security is a crucial problem in today's technological society. Currently, there does not exist a place for discovering, learning, and testing secure design principles. FIREAXE is the pilot competition that attempts to fill this gap. Two teams, one in New Mexico, and one in California, participated in this trial run. We successfully show that a secure design competition is feasible and useful for teaching and guiding students to implement more secure software.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security

Keywords

Cybersecurity competition

1. INTRODUCTION

FIREAXE is the trial run of a project, sponsored by the Science and Technology Directorate of the Department of Homeland Security, designed to test and explore new secure design principles. This document discusses the methods attempted and lessons learned, as well as future directions and competition deployment. In a world where security is so vital to preserving the stability and safety of our critical infrastructure, the failure of our educational system to train security-minded engineers is both a huge disservice to the next generation and a dangerous risk to our national safety.

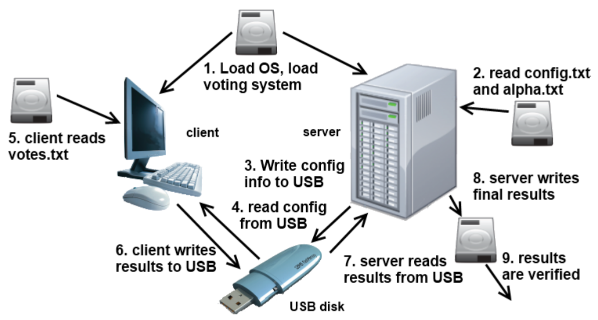
Secure design is a difficult concept to teach; while one can certainly present the theory, genuine understanding is usually not gained until the system has been attacked. Only then do the flaws of an implementation become truly apparent. Clearly, we prefer not to wait for an attack to discover design flaws, and would like any design to be security-minded from the start. This is so for any application, but especially true for critical infrastructure, where consequences of security failures could be catastrophic. The key open questions are: a) how do we ensure that software engineers internalize security principles which they may (at best) know only in theory and b) how do we create an environment where secure design principles, as well as securely designed systems, will naturally emerge? We submit that a secure design competition is the most naturally way to address both of these questions. Moreover, unlike typical cybersecurity competitions such as Tracer FIRE, CSAW, and ideas described in the DESSEC report, a crucial feature of our competition is that it involves *both* the design of a secure system, and an attack on it.

In a secure design competition that we describe, two teams compete by first jointly and independently designing a stylized electronic voting system (i.e., one stripped of much of the complexity that real voting systems necessarily possess, but having many of the relevant features from real systems), and then red teaming the opposing system. This process of "design then red team" is repeated twice to allow each team to address security failures exploited by its counterpart.

2. COMPETITION STRUCTURE

The FIREAXE Secure Design Competition pitted together two teams of three summer interns each. One team was situated at Sandia's Albuquerque, New Mexico site (we call it the NM team), while the other was located at Sandia's Livermore, California site (we call it the CA team). In addition, the competition organizers served as the "white team" tasked with mentoring, answering competition questions, clarifying the rules (and, in several cases, modifying these as necessary), and scoring the teams. The focus of the competition was on designing a stylized *electronic voting system (EVS)* following a provided specification and deployed on a pair of general-purpose PCs. In a nutshell, the EVS consisted of a server (analogous to the election management system) and a client (the actual voting station). The server reads the elec-

[*]Affiliations, in author order, are: Sandia National Labs, University of Texas at Austin, University of Nebraska at Omaha, University of Texas at Dallas, Sandia, University of California at Berkeley, Arizona State University, Sandia, Sandia, Sandia, Sandia, Sandia, Florida State University, Sandia. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.



1. Load OS, load voting system
2. read config.txt and alpha.txt
5. client reads votes.txt
client
server
3. Write config info to USB
4. read config from USB
8. server writes final results
6. client writes results to USB
7. server reads results from USB
9. results are verified
USB disk

Figure 1: High level EVS overview

tion configuration information (provided as a collection of text files on the server machine), and conveys it to the client via a USB. The client first reads the configuration from the USB, and subsequently reads in the votes (provided as a text file on the client machine). Finally, the client conveys the information about votes to the server using the same USB. Figure 2 offers a schematic representation of the voting process. In addition to designing the actual EVS application, the teams were allowed to choose and configure their own OS platform for deploying the EVS, with the constraint that it was based on Linux.

The competition proceeded in two rounds, with each round divided into two phases. The first phase involved each team independently developing a voting system. At the end of the first phase, both teams submitted their entire EVS platform (OS, together with the EVS executables) for both the client and the server (e.g., as an iso or a dd image), EVS deployment instructions, and the source code for the actual EVS client and server application. Each team's submission was then passed on to its counterpart, signaling the start of phase two. In the second phase, each team is tasked with generating attack vectors against the EVS of the other. In the first round, attacks were constrained to be executed as scripts from command line; thus, "terminal attacks", i.e., attacks involving special keyboard keys (such as control) were disallowed. In the second round, the attack surface was expanded to allow attacks involving an arbitrary sequence of ten strings to be entered at the keyboard, allowing in effect execution of both arbitrary scripts, as well as special keys. To ensure that the red teaming phase proceeded in congruence with the spirit of the competition, we imposed some ground rules on the attackers which effectively ruled out attacks on the evaluation structure itself, and additionally required that each attack vector be approved by the white team.

Upon the completion of the second phase, each team submitted its attack vectors, and the evaluation and scoring commenced. We structured the evaluation into five *scenarios* which provided the means for simulating penetration and subversion attacks by presenting an attack entry point:

- The **Baseline** scenario evaluated the EVS of each team without deploying any attacks. Effectively, this scenario evaluated whether the EVS complied with the specification.
- The **Client Breach** scenario executed the attacks on the client PC *as the EVS user* prior to running the client application.
- The **Server Breach** scenario executed the attacks on the server PC *as the EVS user* prior to running the server application.
- The **Client Subversion** scenario executed the attacks on the client PC *as root* prior to running the client application.
- The **Malicious USB** scenario placed attack files onto a USB drive to be used for running the EVS.

Each scenario was scored separately for each EVS, and the total scores were added together to give the final score for each team in the corresponding round.

After the first round was completed, each team received the attack vector submissions targeting its EVS. At this point, the second round proceeded exactly as the first.

3. SECURE DESIGN PRINCIPLES

One of the goals of this project is to identify secure design principles and observe how the competitors operationalize them in practice. Below we list some of these principles [1]:

- **Reduce the attack surface.** The smaller the attack surface, the easier it is to verify or prove correctness.
- **Use existing tools.** Do not re-implement preexisting components unless there is a good reason. A failure to understand certain nuances in important functionality could result in problems down the road. Worse yet, a flawed implementation could result in unexpected weaknesses.
- **Enforce policies at the lowest level.** In many cases, the "lower" the level of implementation, the more difficult it is for an attacker to gain access.
- **Defense in depth.** Never assume that the outermost layers of security will stop all attackers. Always have additional defensive layers outside of your known attack model.
- **Prevent easy access.** Although security through obscurity is generally frowned upon, an effort should be made to prevent attackers from easily analyzing or otherwise exploring a system. At the same time, do not depend on obscurity to stymie all attackers.

Many of these principles seem obvious, but nevertheless they are commonly overlooked in practice. In Section 4, we show how the individual teams applied these principles (not always consciously) to their EVS design.

4. EVS DESIGN

This electronic voting system (EVS) specification is a simplified representation of how a real EVS might behave. The voting configuration is stored on a central server, which is distributed to the client voting machine over a USB disk. The client simulates the actual voting process by reading a log of the voting session from a local file system. The client then sends the results back to the central server over USB, at which point the server runs a final tally and writes the results out to disk.

In addition to designing the EVS application, both teams were also allowed to customize a Linux OS, for example, by customizing the kernel. We now describe how each team designed the EVS and its environment.

4.1 Customizing the Kernel

Both teams created a customized version of the Linux kernel. For the NM team, the kernel customization provided the security anchor. In doing so, they ultimately succeeded in protecting themselves against a root-level (client subversion) attack by the CA team. Rather than creating custom modifications, both teams used readily available tools to harden the kernel. For example, both teams applied the grsecurity patches to the kernel to provide additional security restrictions and modified the default kernel configuration to remove much of the unneeded functionality. The NM team also packaged the root file system directly into the kernel image to prevent an adversary from rebuilding the image with a modified file system.

The kernel embodied several of the secure design principles. Both teams manipulated the kernel options to remove unnecessary functionality, greatly reducing the attack surface. Similarly, both teams hardened the kernel using existing grsecurity patches. Many runtime policies were enforced at the kernel level, and the NM team severely restricted the capabilities even at the root user level (the CA team opted not to do so). Moreover, the NM team made it difficult to access and modify their system out-of-band by packing the file system into the kernel as a compressed image.

4.2 Customizing the Userspace

In addition to securing the kernel, both teams attempted to harden user-level programs and, in particular, the EVS user shell. Both teams created a custom EVS user shell that was only able to execute the client/server code, but do nothing else. This alone made attacks that use this shell as a penetration point almost impossible to execute. Indeed, in round two, the CA team submitted no attacks for client/server breach scenarios, and the NM team's attacks amounted to denial-of-service that exploited the specific (and easily repaired) interface quirks of the CA system combined with poorly chosen remapping of keyboard keys. The NM team also substantially hardened the root user, removing all but the few default executables that the specification explicitly required in order to effectively score the competition. In order to enable additional customization and, moreover, make it more difficult for an attacker to reverse engineer the development environment, the NM team used BusyBox (a common embedded systems package) instead of GNU coreutils, for their development platform. The NM team also used uClibc as a small C runtime library and polarSSL for as a cryptography library. These components are usually found on embedded systems and are easy to pare-down and customize. The CA team, in contrast, used standard python libraries to develop their EVS.

In the second round, both teams deployed some form of access control at the EVS user level. The NM team made the EVS application itself password protected, making reverse engineering and attacks more difficult on its opponent. The CA team created an audio CAPTCHA that requires a user to listen to a sequence of beeps and enter the correct number in order to successfully log in as the EVS user. As it turned out, the audio CAPTCHA used by the CA team was exploited in a denial-of-service attack by the NM team.

The userspace libraries and runtime draw from several design principles. By severely restricting the user shells (and, in the case of the NM team, the root shell), both teams greatly reduced the attack surface. Moreover, all of the libraries and development environments were built using existing tools.

4.3 EVS

Teams were free to program their EVS application in any language they desired. The NM team created versions in Python, C++, and Java, but ultimately settled on C for the ability to harden the binary through the toolchain and to remove almost all library dependencies. The CA team wrote their EVS in Python.

Both teams used cryptographic tools to enforce integrity on the USB-based communication between the EVS client and server. The NM team used Diffie-Hellman key exchange, implemented in PolarSSL. The CA team used Diffie-Hellman key exchange to create a shared secret which was then used as pseudo-random key material. The key material was cyclically XORed with the data in a manner similar to a one-time pad. The data was usually short enough to never repeat the shared key, thus ensuring secrecy.

One of the interesting design decisions is the different uses of cryptography. In the kernel, the most critical use of cryptography is for verification. Thus, the NM team used RSA public key verification for this purpose (the CA team did not do verification at the kernel level), but they deliberately do not include a secret key inside the kernel to prevent the opposing team from re-signing their own binaries. A few weaknesses do exist in the NM scheme because the BusyBox executable actually depends on the execution path. However, this is difficult to exploit due to the read-only nature of the file system. In the electronic voting application, cryptography was used by both teams to verify message integrity but not secrecy.

4.4 The CA "Red Pill"

Arguably the most difficult scenario to defend against in the competition is the client-subversion attack, where the malicious code is run with root privileges. After being bested at this vector in the first round of the competition, the CA team attempted to exploit the human factor in an attempt to protect their EVS from the same attack happening a second time. The NM team's attack involved replacing the shell that the default user logs into and executes the client and server from. Assuming (incorrectly, as it turned out) that the NM team would reuse the same attack vector in the second round, the CA team introduced a "red pill" to prevent it. The red pill determined at boot time if the OS is running in a virtual machine or physical hardware by examining the contents of /dev/mem for known strings. If it is determined that the OS is on physical hardware, it made the shell along with some other important files immutable with the `chattr` command. However, if it is determined that the OS is running inside a virtual machine, it leaves the system unaltered in hopes of enticing the opposition to reuse their attack from round one.

5. ATTACK SCENARIOS

As mentioned earlier, there were several restrictions placed on the attackers, not the least of which is the requirement that the attack plans be approved by the white team. One of the main motivations for restricting attacks is to ensure that red teaming efforts are focused on attacking the system as designed by the counterparts, rather than flaws in the competition setup itself.

5.1 Round One Attacks

5.1.1 NM Team's Attacks

Since the CA team used a custom shell for the EVS user that had no script interpreter, the NM team could not deploy any attacks for the client/server breach scenarios. They did, however, have successful attacks for client subversion and malicious USB scenarios:

Client Subversion: The NM team's root level attack involved a simple replacement of the custom shell which then launched a substitute client binary. This was straightforward to implement as the CA system did not protect against a malicious root user.

Malicious USB: The NM team created a passive injection attack that tricked the CA team's submission into accepting an empty input message.

5.1.2 CA Team's Attacks

In round one, the NM team offered a full shell for the EVS user, which was subsequently exploited by the CA team during the red teaming phase.

Client Attacks: The CA used a replay attack. The NM round one submission did not use a randomized shared secret as part of their integrity check to protect the files as they moved from the client to the server. Thus, a given set of input files (sequence of votes and election configuration) produce the same set of output files that are communicated to the server. Thus, the CA team replayed an election engineered using the NM system offline, waited until the USB was unmounted by the client, and then overwrote its contents with the files artificially generated which would pass the integrity test at the server. Because the round one NM EVS submission had the complete shell for the regular EVS user, the CA team exploited the fact that certain obscure sh flags (sh -nv) allow one to copy the replay files onto the USB (even though no copy command was available in the user shell). Additionally, since mount and umount were enabled, the CA team was able to run the client breach attack in the client subversion scenario.

Server Breach: In this scenario the CA team executed a similar replay attack that was used in the other scenarios: they used sh to copy the replay files generated offline onto the USB once it was inserted into the server.

Malicious USB: The CA team exploited a bug in the NM team's EVS code by filling the USB prior to its use in EVS. In this case, the NM team creates empty files on the USB, which appeared to the server as if no one had voted. The result was that all candidates were declared as winners.

5.2 Round Two Attacks

5.2.1 NM Team's Attacks

For the second round, the New Mexico team came up with attacks for all scenarios, although in all cases except client subversion these were denial-of-service attacks.

Client/Server Breach: The NM team abused the audio CAPTCHA used by the CA system. Because the CAPTCHA takes a non-trivial amount of time to play, in combination with the decision by the CA team to remap filtered keys to the "enter" key, the NM team's attack involved pressing the "[" key for 5 seconds, which results in a 15 minute delay while the CA system processes the failed CAPTCHA answers together with repeated new queries. Since a short amount of time spent to enter an attack string causes a nearly arbitrary system delay, this was an effective denial-of-service attack.

Client Subversion: This time around, the CA system's root shell was better locked down and prevented write access to a lot of directories. However, because `mount` is required for the root shell, it is also possible to abuse it by mounting temporary file systems on top of protected directories. In particular, the NM team mounted a writable tmpfs on top of the home directory allowing arbitrary write access to supposedly trusted paths.

Malicious USB: The CA round two submission still did not verify the integrity of the message, but instead tried to empty the USB stick before use. However, due to implementation oversights, there were a number of ways to prevent this operation from succeeding, such as creating a symlink to a directory, which resulted in a denial-of-service attack (the attack was recognized as such by the CA team).

5.2.2 CA Team's Attacks

The CA team was able to produce only one successful attack (which resulted in a detected DoS) involving the malicious USB scenario. For this attack, the CA team created a folder structure on the USB that the recursive delete code couldn't delete. It would open too many directories and die due to the lack of file descriptors. The CA team also attempted a client subversion attack. However, it was unable to deploy it because the NM system in round two included two highly restrictive features of the root shell: 1) one could only execute or view files on a pre-generated white list which included only the election configuration files, output files, and the actual client/server binaries, and 2) configuration files were restricted to only contain alphanumeric characters, effectively preventing the CA team from executing scripts.

6. SCORING

The competition was scored as follows. During each scenario, the EVS system was executed following a specified sequence of commands. At the end, the server created output files containing election results, which were then compared to the "golden copies". If the output files matched the golden copies perfectly, the EVS received a score of 4 points for that scenario, with the red team receiving no points. If these were incorrect, the EVS team received 0 and the red team received 2 points. Additionally, we treated denial-of-service (DoS) attacks as a special case, and allowed an EVS to receive points merely for detecting that it has been a target of a DoS. An attack was considered a DoS if it resulting in the EVS not producing any output files with election results. If a DoS attack was correctly recognized by the EVS, it received 1 point, and the red team would receive 0; otherwise, the scenario was scored as a successful attack.

7. CONCLUSION

From our experiences, we believe that this pilot Secure Design competition is largely a success. Not only is it an effective way to teach people at all levels, from students to senior developers, about the concepts and importance of secure design, but it also provides a safe environment for both novel design and creative offense without impacting critical infrastructure.

8. REFERENCES

[1] Terry V. Benzel, Cynthia E. Irvine, Timothy E. Levin, Ganesha Bhaskara, Thuy D. Nguyen, and Paul C. Clark. Design principles for security. Technical report, Naval Postgraduate School, 2005.