

# Using Minix to Teach Computer Security Courses

Wenliang Du and Sankara Narayanan  
Department of Electrical Engineering and Computer Science  
Syracuse University, 121 Link Hall, Syracuse, NY 13244  
Email: {wedu,sasankar}@ecs.syr.edu

## ABSTRACT

To address national needs for computer security education, many universities have incorporated computer and security courses into their undergraduate and graduate curricula. In these courses, students learn how to design, implement, analyze, test, and operate a system or a network to achieve security. Pedagogical research has shown that effective laboratory exercises are critically important to the success of these types of courses. However, such effective laboratories do not exist in computer security education.

Intrigued by the successful practice in operating system and network courses education, we adopted a similar practice, i.e., building our laboratories based on an instructional operating system. We use `Minix` operating system as the lab basis, and each lab exercise requires students to add a different security mechanism to the system. Benefited from the instructional operating system, we can design our lab exercises in such a way that makes it easy for students to focus on one or a few specific security concepts while doing each exercise. The similar approach has proved to be effective in teaching operating system and network courses, but it has not yet been used in teaching computer security courses.

## General Terms

Security, Design

## Keywords

Computer security, operating system

## 1. INTRODUCTION

The high priority that information security education warrants has been recognized since the early 1990's. In 2001, Gene Spafford of Purdue University's Center for Education and Research in Information Assurance and Security, testified before Congress that "to ensure safe computing, the security (and other desirable properties) must be designed in from the start. To do that, we need to be sure all of our students understand the many concerns of security, privacy, integrity, and reliability" [11].

To address these needs, many universities have incorporated computer and information security courses into their undergraduate and graduate curricula; in many, computer security and network security are the two core courses. These courses teach students how to design, implement, analyze, test, and operate a system or a network with the goal of making them secure. For these types of courses, pedagogical research has shown that students' learning is enhanced if they can engage in a significant amount of hands-on exercises. Therefore, effective laboratory exercises (or course projects) are critically important to the success of our computer security education.

Traditional courses, such as operating systems or compiler, have effective laboratory exercises, the result of twenty years maturation. In contrast, laboratory designs in security education courses are still embryonic. A variety of approaches are currently used; three of the most frequently used designs are the followings: (1) the *free-style* approach where instructors allow students to pick whatever security-related topics they like for the course project; (2) the *dedicated-computing-environment* approach, where students conduct security implementation, analysis and testing [6, 7] in a contained environment; (3) the *build-it-from-scratch* approach, where students are required to build a secure system from scratch [9].

Free-style design projects are effective for creative students who enjoy the freedom it affords; however, most students become frustrated with this strategy because of the difficulty in finding an interesting topic. With the dedicated-environment approach, projects can be very interesting, but the logistical burdens of the laboratory - obtaining, setting up, and managing the computing environment - are considerable for instructors. In addition, course size is constrained by the size of the dedicated environment. The third design approach requires that students spend considerable time on activities irrelevant to security issues, but essential for a meaningful system.

Having understood the drawbacks of the existing approaches, we propose to develop a new approach for the laboratories of computer security courses. We want our labs to have the following properties: (1) It should be well designed, and each laboratory should clearly lay out its goals and tasks. (2) The lab does not need special computing environment or super-user privilege. Students with normal user accounts should be able to carry out the lab assignments. (3) Most importantly, the lab provides an infrastructure to students, so they do not need to implement everything from scratch. Their implementation should be part of a much more complex system, they can immediately see how their implementation behaves without having to build the rest of the complex system.

Such a laboratory design does not exist in computer security education, but similar laboratory designs exist in other more traditional and mature courses, such as operating systems (OS), compilers, and

networks. In OS courses, a widely adopted successful practice is to use an instructional OS (e.g. `Minix` [12], `Nachos` [2], and `Xinu` [3]) as the framework and ask students to write significant portions of each major piece of a modern OS. The compiler and network courses adopted a similar approach. Inspired by the success of the instructional OS strategy, we adapt it to our computer security courses. Specifically, we will provide students with a system as the framework, and then ask them to implement significant portions of each fundamental *security-relevant functionalities* for a system. Although there are a number of instructional systems for OS courses, to our knowledge, this approach has not yet been applied in computer and information security courses.

We identified `Minix` instructional operating system as our basis for three reasons: first, `Minix` is very complete comparing to other unix-style instructional OS; second, `Minix` can run in `Solaris` operating system as a normal process, hence needs no special privilege; third, `Minix` operating system is particularly flexible, easy to use, and easy to understand.

We have designed a number of laboratory assignments based on `Minix`. These assignments cover topics ranging from the design and implementation of security mechanisms to the analysis and testing of a system for security purpose. Each assignment can be considered as adding/modifying security mechanisms to `Minix`. To finish each task, students just need to focus on those security mechanisms, and they are not bothered by non-relevant issues. For example, while teaching the Discretionary Access Control (DAC) concept, we will give students a modified version of the `Minix` system without the DAC mechanism. Students will need to write programs to design and implement DAC for the system. However, since the file system (the system that uses DAC) is already in place, students can immediately see how their DAC implementation affect the system. This makes it possible for students to stay focus on the DAC concept.

## 2. THE COMPUTER SECURITY COURSE

### 2.1 Scope of the course

Our department offers two graduate courses in security: one is the computer security, and the other is the network security. The computer security course focuses on the concepts, principles, and techniques for system security, such as *encryption algorithms*, *authentication*, *access control*, *privilege*, *vulnerabilities*, *system protection*, etc. Currently, our proposed approach is target at the computer security course, although our future plan includes extending this approach for the network security course.

### 2.2 Pedagogical Approach

Learning theories, principles and techniques of computer security is not enough to understand system security. Students must be able to put what they have learned into use. We use the “learning by doing” approach. “Learning by doing” forces students to digest the information they learned in class to the degree where they can instruct the computer how to apply it. Active learning such as this has a higher chance of having a lasting effect on students than if the students passively listen to lectures without reinforcement [8].

More specifically, we try to use the `Minix` OS as our base system to develop assignments that can give students the hands-on experience with those theories taught in class. For example, when teaching `Set-UID` concept of `Unix`, we developed an assignment to enable students to play with it, to know why it is needed, and to learn how it is implemented.

We have developed two types of assignments: small assignments and comprehensive assignments. Each small assignment focuses

on one specific concept, such as `Set-UID` and access control. These types of assignments are usually small; they do not need much programming, and take only one week or two. Therefore, we can have several small projects to cover a variety of concepts in system security.

Being able to deal with each individual concept is not enough, students need to learn how to put them together. To this end, we have developed comprehensive assignments, which cover many concepts in one assignment. They are usually the appropriate candidates for final projects.

### 2.3 Course Prerequisites

Because this course focus on system security, we require students to have appropriate system background. All students taking the course are expected to have taken the graduate operating systems. They should be proficient in C programming.

## 3. LABORATORY SETUP

### 3.1 Introducing `Minix`

`Minix` is very similar to `Unix` operating system, and its name came from “mini `Unix`”. The structure and commands in both operating system are almost the same, but unlike `Unix`, `Minix` is intended for being used as an instructional operating system. Therefore, `Minix` has been kept relatively small and simple; it only has about 15,000 lines of code, and the installation is also easy. Because of this any first time user with decent background in `Unix` can understand the concepts of this operating system easily.

`Minix` operating system has a very high modular structure, which not only makes it easy to understand the concepts of an operating system, but also makes it easy for any user to add or modify codes in the needed parts. Because of the modularity, for small projects, students just need to change very few lines of code in the appropriate places.

### 3.2 Laboratories Setup

`Minix` was originally developed as a real operating system, running directly on Intel x86 machines. Later, `Minix` was ported to `Solaris` environment, where `Minix` can run on top of the `Solaris` operating system. This ported version makes it possible to run `Minix` without owning a real machine. We use this ported version in our laboratory. All of the laboratory exercises will be conducted in `SUN Solaris` environment using C language. Except for giving students more disk space (100 Megabytes) to store the files of `Minix` system, `Minix` poses no special requirements on the general `Solaris` computing environment.

## 4. THE ASSIGNMENTS

We have designed seven lab assignments based on `Minix`. However, depending on the students’ familiarity with `Unix` and their proficiency in C programming, instructors might want to choose a subset of it. Currently, we are still developing more assignments, and we will also solicit contributions from other people. Our goal is to create a pool of lab assignments, so different instructors can choose the subset that is closely relevant to their syllabi.

### 4.1 Preparation

We use this assignment to let students get familiar with the `Minix` operating system, such as installing and compiling the `Minix` OS, conducting simple administration tasks (such as adding/removing users), and learning to use/modify some common utilities. More importantly, we want students to be able to understand and modify the kernel of the OS. Since students will only deal with the

part of the kernel that are related to the security, they do not need to know the entire kernel; instead, they just need to know system calls, file systems, the data structure of `i-node` and `process` table. They do not need to worry about process scheduling, memory management, etc. Since most of the above knowledge and skills were taught in our graduate operating system course (the prerequisite for this course), students should be able to achieve the above goals within two to three weeks.

The following is a list of sample tasks we used, but each instructor can choose their own tasks to achieve the same goals:

1. Compile and install `Minix`, then add three user accounts to your system.
2. Change the password verification procedure, such that a user is blocked for 15 minutes after three failed trials.
3. Implement system calls to enable users to print out attributes in `i-node` and `process` table. Appropriate security checking should be implemented to make sure a user cannot read other users' information.

From our experience, this assignment is crucial to the success of the subsequent assignments. Some students who overlooked this assignment find themselves in deep trouble later. In fact, when we used the proposed approach at the first time, we did not give students this assignment because we thought it was not necessary. As a result, students later spent a great deal of time in figuring out how to achieve the tasks in this assignment. Most of the students told us that they spent 80% of their time to figure out the system, and once they knew how things work, it did not take them long to finish the required tasks.

Therefore, when we use the approach again, we used several lectures to teach students the necessary materials, and ask the TA to devote significant amount of time to help the students finish this assignment. The improvement is substantial.

## 4.2 Set-UID Programs

Set-UID is an important security feature in Unix operating system; it is also a good example to show students how privileges are managed in a system, and what problems a system could have if privileges are not handled properly. We use this project to let students become familiar with the Set-UID concept, its implementation, and potential problems.

Students need to finish the following tasks: (1) Figure out why `passwd`, `chsh`, `su` commands need to be Set-UID programs, and what will happen if they are not. (2) Students are given a binary code for `passwd` program, which contains a number of security flaws injected by the instructor. Students need to identify those flaws, and exploit them to gain the root privilege. (3) Read the OS source codes of `Minix`, and figure out how Set-UID is implemented in the system. (4) Modify the OS source code to disable the Set-UID mechanism.

The experience that the students gained with this project was really instrumental to their course work also. Some students gained significant knowledge as to how the Set-UID programs work. They manipulated the `passwd` program with the security flaws and found out the flaws in them. They also knew and understood, the importance of the Set-UID programs. They manipulated the OS source codes and understood how Set-UID program is implemented in `Minix`.

## 4.3 Access Control

Access control is an important security mechanism implemented in many systems, and there are different types of access control:

Discretionary Access Control (DAC) and Mandatory Access Control (MAC). The goal of this project is two-fold: (1) to get first-hand experience with DAC and MAC, and (2) to be able to implement DAC and MAC.

Students are given a version of `Minix` with no access control mechanism. They need to implement all (or some) of the following access control mechanisms: (1) Abbreviated ACL: the access control is based on three classes: owner, group, and others. (2) Full ACL: the access control can be based on individual users. (3) MAC: design and implement a simple MAC access control mechanism for `Minix`.

This was a bit more challenging project to students compared to the Set-UID project. The students showed interest towards the implementation of the DAC and MAC concepts which had been taught in the computer security course. The students learned that access control allows an user to specify access permissions on a per user (principal) basis, rather than the current owner-group-other protection method. They learned to manipulate the normal `Minix` file permissions like read, write, execute. They even had to know how the `Open` system call works in `Minix`. Many students really liked the project and found it interesting.

## 4.4 Capability

Capability is another important concept in computer security. The goal of this project is to let students be familiar with the capability concept. We achieve this by asking them to implement a simplified capability mechanism.

Students are supposed to implement a simple capability mechanism in `Minix`. Such a mechanism should support the following: (1) Permission granting based on capability. (2) Capability copying: a process should be able to copy its capabilities to another process. (3) Capability amplifying/reduction: a process should be able to amplify or reduce its current capabilities. For example, a process can temporarily remove its own setuid capability, but later can add it back. Of course, a process cannot add a new capability to itself if it does not already own that capability. (4) Capability revocation: the root should be able to revoke capabilities from current and future processes.

This project enhanced the students' understanding of the capability concept. At the beginning, most of the students had trouble mapping the capability concept to the real system because what they had learned in the classroom is abstract and their daily use of computer systems gives them barely any first-hand experience with the capability. We did not tell the students how the capability should be implemented, we ask them to design their own capability mechanisms. This requires them to figure out how the capabilities should be represented in the system, where to store the capabilities, how the system can use the capability to conduct access control, etc. Once students have figured out all of these issues, the implementation becomes relatively easy; therefore the amount of coding for this project is not significant, and students are able to accomplish the task within two weeks. Had it not been for `Minix`, students would need to spend a lot of time implementing a meaningful system where the effect of the capability can be demonstrated.

We also encouraged students to be more creative and to design and achieve some other features beyond the basic requirements. Students were highly motivated, some implemented a more generic capability-based access control mechanism than the required one, and some allow new capabilities to be defined by the superuser. Students later commented that "now we really understand the capability".

## 4.5 Vulnerability Analysis

The first goal of this lab is to let students gain first-hand experience on software vulnerabilities, to be familiar with a list of common security flaws, to understand how a seemingly-not-so-harmful flaw in a program can become a great risk to the system. The second goal of this lab is to give students opportunities to practice their vulnerability analysis and testing skills. Students learn a number of methodologies from the class, such as vulnerability hypothesis, penetration testing methodology, code inspection technique, and blackbox and whitebox testing [10]. They need to practice using these methodologies in this lab.

The students are given a version of the `Minix` operating system that is full of injected vulnerabilities. These vulnerabilities simulate system flaws caused by incorrect design, implementation, and configuration. The students are also given some hints, such as a list of possible vulnerabilities, the possible locations of the vulnerable programs, etc. Their task is to find those vulnerabilities.

With this project, students were exposed to some of the most common vulnerabilities, such as buffer overflow, race condition, security holes in the access control mechanisms, and information leaking. Because we can arbitrarily modify the operating system, injecting these vulnerabilities are not a difficult task for us. We plan to create a collection of vulnerable security components for `Minix` to cover a variety of vulnerabilities drawn from the reality.

## 4.6 Sandboxing

A sandbox is an environment in which the actions of a process are restricted according to a security policy [1]. Such restriction protects the system from untrusted applications. Sandboxing is an important concept in computer security. We want students to understand such a concept by studying an existing sandbox system and by implementing a sandbox for `Minix`. In addition to the sandboxing concept, this project also includes a number of other concepts related to security, such as access control, system calls, isolation, and security policies. Therefore, this lab is considered as a comprehensive project.

`Janus` [5] is a well-known sandbox that restricts a process's invocation of system calls. It is well documented. In this lab assignment, students need to learn from these documentations how `Janus` is designed and how it works. Then students need to implement a simplified version of `Janus` for `Minix`.

Students can learn a number of security concepts from this project:

- System call interposition: system call interposition techniques are widely used by many systems for enhancing security. There are several ways to achieve this, some using user-space utilities, some requiring kernel modification. Students got to choose which technique to use, and by doing this, they understand the pros and cons of each different technique.
- Security policies: a sandbox needs to enforce some security policies. Student need to define what policies should be enforced by the sandbox? how to define their security policies, how flexible the policies should be? and how they can be enforced? Those are important decisions that any access control mechanism should make.
- Security problems: Based on several years' experience with `Janus`, Garfinkel pointed out that there are many traps and pitfalls in implementing a sandbox, including race conditions, relative-path problem, symbolic-link problem, etc. [4]. This project gives students the first-hand experience with these security problems, and gives them the opportunities to attempt to solve those problems.

- Usability: the sandbox should be compatible with and transparent to the application programs. Moreover, it should be configurable so different application programs can have different sandbox.

## 4.7 Encrypted File System (EFS)

Traditional file system does not encrypt the files that are stored on a disk, so if the disk is stolen, contents of those files can be recovered. An EFS solves this problem by encrypting all files on the disk, such that only users who knows the encryption keys can access the files. The encryption/decryption operations should be transparent to users. Designing and implementing such a system require students to combine together the knowledge about encryption, key management, authentication, access control, and security in OS kernels and file systems. Therefore this project is meant to be a comprehensive project. We suggest giving this project as a final project after most of the relevant concepts have already been covered in class.

`Minix` operating system already has a complete file system, so students do not need to worry about building a file system at the first place; they can build the EFS upon the existing file system. As we mentioned before that `Minix` is developed for instructional uses, its file system is reasonably easy to understand by students. Students are able to design their EFS's once they understand how the file system works.

This project is not just about the encryption, it actually covers a variety of security-related concepts:

- Using encryption and hashing algorithms. Since we do not focus on how an encryption algorithm is implemented, we just provide students with the encryption codes (e.g. AES) downloaded from the Internet. Via using these algorithms, students can gain more understanding on those concepts (e.g., block cipher, padding, etc.) that they have learned in class.
- Key management: how to change/revoke keys? how and where to store the secret keys? how to protect the keys? These are some of the important issues in this project. Students come up with different designs. For example, regarding the key storage problem, some students store the key (encrypted) in a file, and some store it in the i-node of the encrypted file. We also found out that some students mistakenly save the plain-text key on the disk, and it defeats the whole purpose of the EFS.
- Authentication: how to decide whether a user can access the encrypted file system or not? This part of the project not only teach students the authentication purpose, more importantly, it teaches students an important lesson about the trade-off between the usability and the security. Some student's projects require users to authenticate themselves each time they access a file in EFS; some conduct just one authentication when the users mount the EFS (the best implementation in our opinion); some conduct the authentication during the login. During their demos, we point out the advantages and disadvantages of their designs, so they understand why their designs are good/bad.
- User transparency: how to make the security mechanisms transparent to users? An important feature of EFS is that the encryption/decryption should be transparent to users. This basically requires the encryption/decryption to be conducted on the fly while users are reading/writing a file. This is quite a challenge in this project because students need to understand how the system calls (such as `open()`, `read()`,

`write()` of the file system work, and how to modify them without affecting the behaviors of those system calls.

- Security analysis: is there a way to by-pass the implemented security mechanisms? what are the potential vulnerabilities?

Because students come from a variety of background, some of them found this project extremely difficult. In the future, we plan to design a multi-level requirement for this project, such that students with different backgrounds can find a level of requirement that they can achieve. For example, after realizing this problem, we told some of the students that they can implement their projects without satisfying the “user transparency” requirement; of course, they will receive lower grades than those whose implementations satisfy the requirement.

## 5. OTHER LESSONS LEARNED

We did a teaching experiment in the 2002 spring semester when we taught the computer security course (CSE 785). At that time, we decided to use the original `Minix` operating system and asked students to modify its kernel to add certain specific security mechanisms into the system. We only give them one project for the whole semester because modifying an OS seems to be a daunting job for most of the students. The students liked the project very much and were highly motivated. At the end of the semester, the students provided a number of useful suggestions. For example, many students noted, “most of our time was spent on figuring out how such an operating system work, if somebody or some documentation can explain that to us, we could have done four or five different projects of this type instead of doing one during the whole semester”. This observation shapes the goal of our design: we want students to implement a project within two to four weeks using our proposed instructional environment.

When we taught teach CSE 785 again in Spring 2003, we provided students with sufficient information about how `Minix` works, and we added a lecture to introduce `Minix`. As a result, students had gotten familiar with `Minix` within the first three weeks, and were ready for the projects we had designed for them. The same degree of familiarity took my previous students half of a semester due to the lack of information. Moreover, we were able to assign four projects in one semester, including a complicated one, the encrypted file system project. Here are some additional lessons we have learned during the last two years:

1. Preparation: The preparation part is extremely important. If students fail this part, they will spend enormously more time on the subsequent projects. This is very clear when we compare the performance of the students in our 2003 course with that of the students in 2002. We plan to integrate the materials related to `Minix` into the lecture, so students can be prepared better.
2. Background Knowledge: We also realized that some students in the class does not know the basics of `Unix` operating system because they have been using Windows most of the time. This brings some challenges because these students do not know how to set up the `PATH` environment variable, how to search for a file, etc. We plan to develop materials to help students get over this first obstacle.

## 6. CONCLUSION AND FUTURE WORK

We have described a laboratory design for the computer security course. Our design is based on `Minix`, an instructional operating

system. We have described a series of lab assignments based on `Minix`. The experience we have obtained is very encouraging, and students in our class have shown great interests in the course and the projects.

We will continue developing and experimenting with our approach. We will also conduct a full evaluation of the effectiveness of using `Minix` in computer security courses. More importantly, we will work on making this laboratory approach easy to be adopted by other people. This requires us to provide detailed documentations, instructions, and a pool of different projects covering a wide range of security concepts.

Furthermore, because `Minix` has a full implementation of TCP/IP protocols, we are thinking about extending our course projects to cover some of the security concepts related to network security, such as firewall, network sniffing, and TCP/IP protocol vulnerabilities.

## 7. REFERENCES

- [1] M. Bishop. *Computer Security: Art and Science*. Addison-Wesley, 2002.
- [2] W. A. Christopher, S. J. Procter, and T. E. Anderson. The nachos instructional operating system. In *Proceedings of the Winter 1993 USENIX Conference*, pages 481–489, San Diego, CA, USA, January, 25-29 1993. Available at <http://http.cs.berkeley.edu/~tea/nachos>.
- [3] D. Comer. *Operating System Design: the XINU Approach*. Prentice Hall, 1984.
- [4] T. Garfinkel. Traps and pitfalls: Practical problems in in system call interposition based security tools. In *Proceedings of the Network and Distributed Systems Security Symposium*, February 2003.
- [5] I. Goldberg, D. Wagner, R. Thomas, and E. Brewer. A secure environment for untrusted helper applications: Confining the wily hacker. In *Proceedings of the 6th USENIX Security Symposium*, pages 1–13, 1996.
- [6] J. M. D. Hill, C. A. C. Jr., J. W. Humphries, and U. W. Pooch. Using an isolated network laboratory to teach advanced networks and security. In *Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education*, pages 36–40, Charlotte, NC, USA, February 2001.
- [7] J. Mayo and P. Kearns. A secure unrestricted advanced systems laboratory. In *Proceedings of the 30th SIGCSE Technical Symposium on Computer Science Education*, pages 165–169, New Orleans, USA, March 24-28 1999.
- [8] C. Meyers and T. B. Jones. *Promoting Active Learning: Strategies for the College Classroom*. Jossey-Bass, San Francisco, CA, 1993.
- [9] W. G. Mitchener and A. Vahdat. A chat room assignment for teaching network security. In *Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education*, pages 31–35, Charlotte, NC, USA, February 2001.
- [10] C. Pfleeger, S. Pfleeger, and M. Theofanos. A methodology for penetration testing. *Computers and Security*, 8(7):613–620, 1989.
- [11] E. H. Spafford. February 1997 testimony before the united states house of representatives’ subcommittee on technology, computer and network security. Available at <http://www.house.gov/science/hearing.htm>, 2000.
- [12] A. Tanenbaum. *Operating Systems: Design and Implementation*. Prentice Hall, 2nd edition, 1996.