

# Using Instructional Operating System to Teach Computer Security Courses



---

Wenliang (Kevin) Du  
EECS Department  
Syracuse University

Sponsored by NSF CCLI Program



# Course Objectives

---

- Teaching security principles and Technologies
  - Access control, Authentication, security policy
  - Encryption, key management, basic crypto
  - Principle of least privilege
- First-hand experience with
  - Security mechanisms
  - Vulnerabilities
- Design & Implementation security mechanisms
- Analysis & Testing for security



# Course Projects (Labs)

---

- Labs are important for computer security education
- Many course projects exist
  - Ad hoc approaches
  - Lack of a systematic approach
  - Scope of most approaches is narrow
  - Other “old” fields (OS, Network, Compilers) are not like this



# Overview

---

- Course projects based on Minix
  - **iSYS**: Instructional SYStem for security
  - **iLAN**: Instructional LAN for security
- A survey of the existing course projects for computer security



# Learning from Other Fields

---

- Operating System Courses
  - Instructional OS: **Minix, Nachos, Xinu.**
  - Examples: scheduling, inter-process communication, file system, paging system.
- Compiler Courses
  - Instructional compilers and languages
- Networking Courses
  - Also using instructional OS
  - Example: IP/ICMP/UDP implementation.



# What did I learned?

---

- I have learned:
  - The base system is always functioning
  - Each project adds a new functionality to the base system or replaces a functionality
  - It is NOT a toy system, and it is NOT so complicated (some of the instructional OS has been used in some embedded systems)
- Can we do the same for computer security courses?
  - Examples: Access Control Mechanisms.



# Why Instructional OS

---

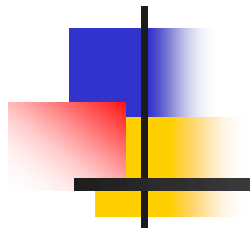
- Why not use a real operating system?
  - Large and complex
  - Our time limitation
  - Our mission: teach fundamentals
- Instructional operating systems
  - Small size, manageable within a semester.
  - Easier to install, modify, compile, and debug, compared to production OSes.



# Outline

---

- iSYS environment setup
  - Minix Instructional Operating System
  - Running environment
- iSYS labs
  - Observation Labs
  - Design Labs
  - Vulnerability Labs



# Environment Setup



# Selecting Instructional OSes

---

- We have studied
  - Nachos
  - Xinu
  - Minix
- iSYS lab design is OS independent
  - Can be built upon any of them



# Minix Operating System

---

- Open source
- Many documentations
- POSIX-compliant Unix
- Modern modular micro kernel architecture
  - File system and TCP/IP are not in kernel
- Small
  - Minix Version 3: < 3800 lines of kernel code



# How to run Minix

---

- On Native machines
  - Need dedicated machines
  - Inconvenient
- Emulator
- Virtual Machine



# Emulator

---

- Simulates a complete Intel x86 computer
  - Simulate every single machine instruction.
- Bochs
  - x86 PC emulator.
  - Runs on many platforms, including x86, PPC, Alpha, Sun, and MIPS
- Advantage: portability
- Disadvantage: slowdown factor is about 100



# Virtual Machine

---

- Virtualization:
  - “Simulating” x86 instructions on an x86 machine
  - Directly run most of the native machine instructions.
- VM Software
  - Vmware and VirtualPC
  - Plex86: open source
- Advantage: speed
- Disadvantage: portability



# Our experience

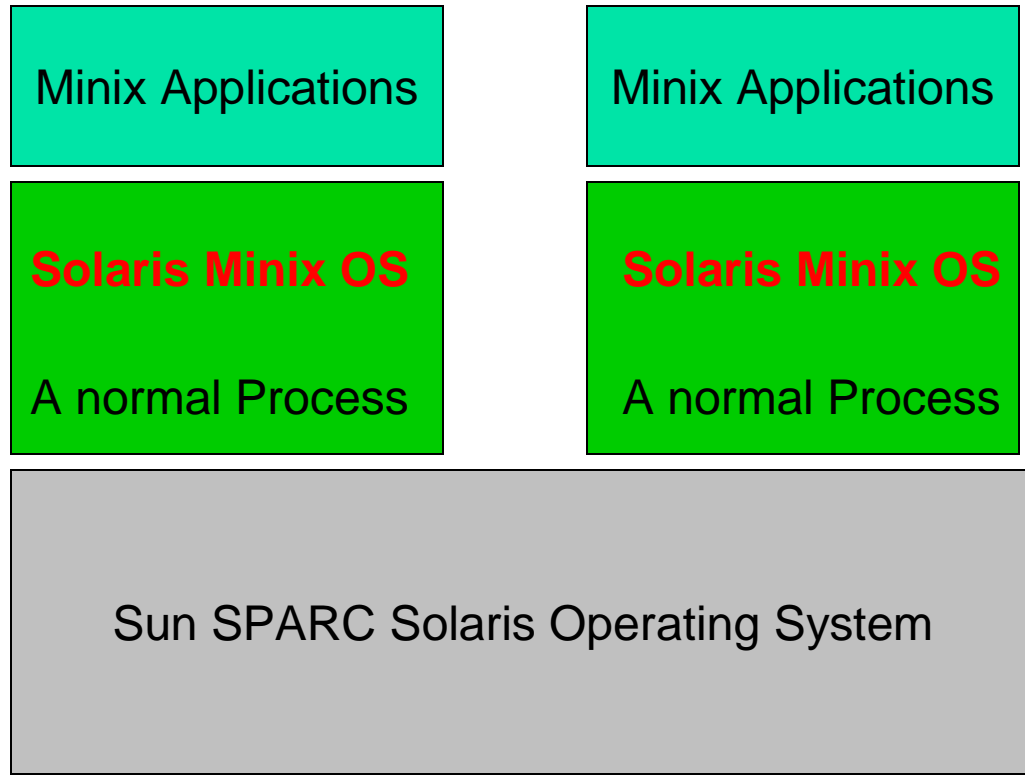
---

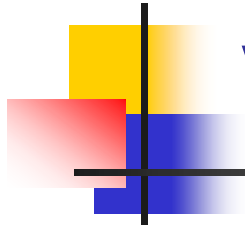
- We tried two approaches
  - Emulator: Using Solaris Minix (SMX)
  - Virtual Machine: Vmware
  
- The most important thing is:
  - Get Minix to run!
  - Get networking to work!



# SMX Approach

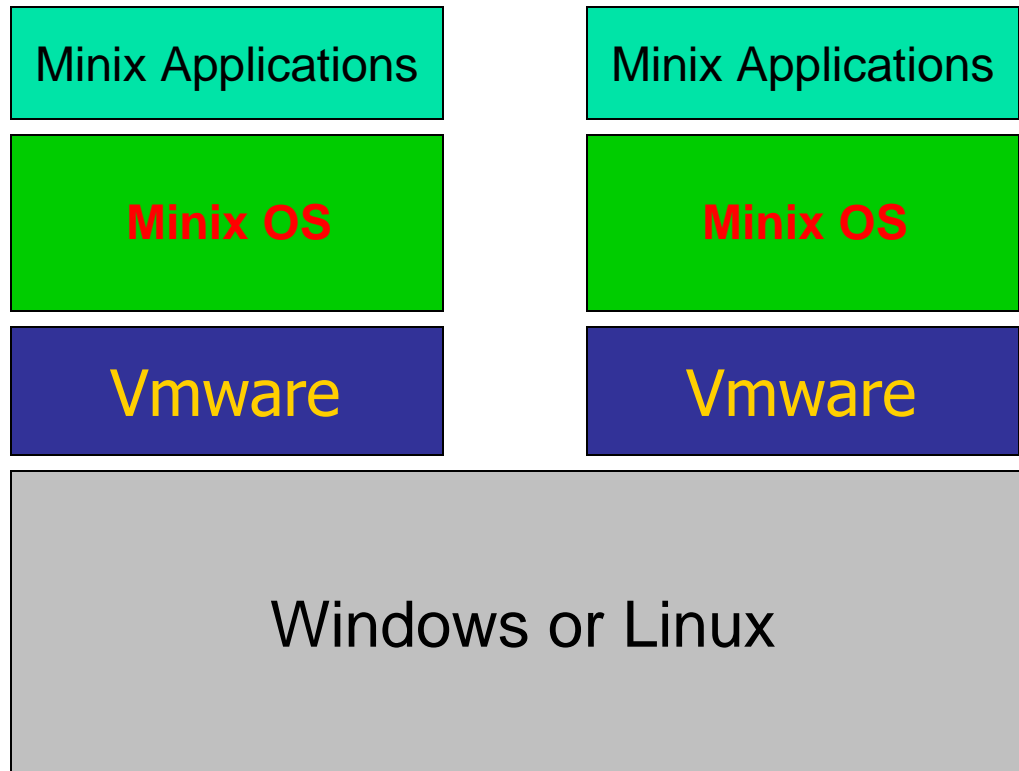
---

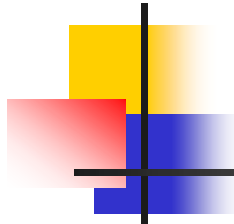




# Vmware Approach

---





# Labs

---

- We have developed a pool of labs
  - Cover a wide range of security concepts
  - An instructor can choose a subset
- Small and Focused Labs
  - Cover a single security concept
  - e.g. access control
- Comprehensive Labs
  - Cover several security concepts
  - e.g., encrypted file system



# Three Types of Labs

---

- Observation Lab
  - Play with security mechanisms
  - Evaluate system's security
- Design/Implementation Lab
  - Security mechanisms
  - Systems with security mechanisms
- Vulnerability Lab
  - Finding vulnerabilities



# Observation Labs

---

- Normally does not involve coding
- Focus on gaining experience
- Tasks include
  - Use security mechanism
  - Read source code
  - Read documentation
  - Make minor change to security mechanism

# Vulnerabilities Labs

**Real-World Vulnerabilities**

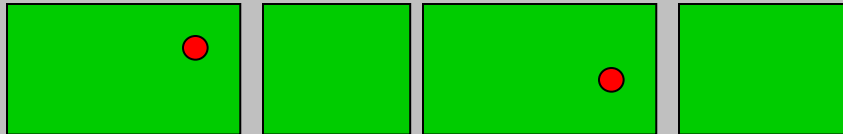
Fault Injection

Students' Tasks:

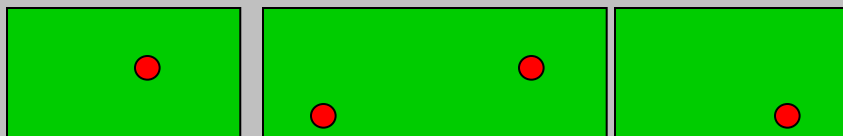
1. Find out those vulnerabilities
2. Exploit the vulnerabilities
3. Fix the vulnerabilities

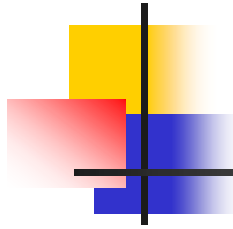
**Minix OS**

User  
Space





Kernel  
Space

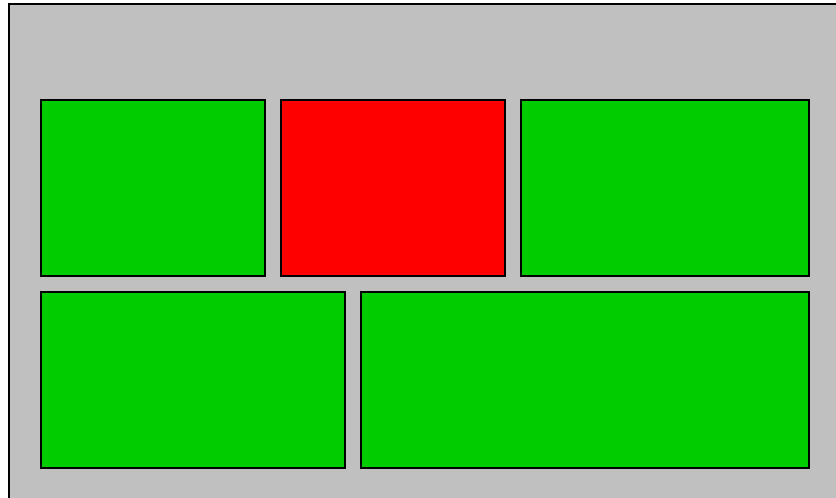




# Design/Implementation Labs

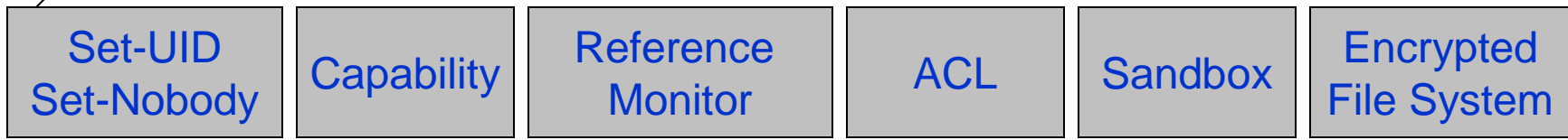
A Security Mechanism

-  Existing Components
-  Students' Tasks



Properties of this design:

- Focused
- Each lab takes 2-3 weeks

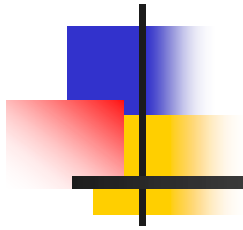




# Design/Implementation Labs

---

- Privileges
  - Set-UID
- Access Control
  - Access Control List, Capability, MAC
  - Reference Monitor
  - Sandbox
- Authentication
- Comprehensive Labs
  - Encrypted File System
  - IPSec



# Set-UID Lab



# Set-UID Lab

---

- Set-UID
  - Access control is based on effective user id
  - Effective user id  $\neq$  Real user id
  - Turn on Set-UID bit: `chmod 4755 exec_file`
  - Escalate a user's privileges
- Objectives
  - Understand Set-UID concept
  - Understand why we need it
  - Understand its danger
  - Think about how to improve it



# Set-UID: Lab Description

---

- Play with Set-UID programs
  - Why should `passwd` and `su` be setuid?
  - What happens if they are not?
- Read Minix source code
  - How is set-uid implemented
  - How does Set-UID affects access control?
  - How to disable Set-UID?
- Think about the following
  - What is the danger of Set-UID
  - Is it a good design? Why? Design an improvement



# Set-UID: Set-Nobody

---

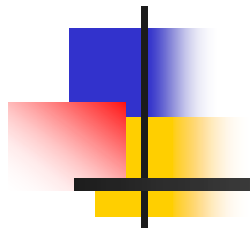
- Opposite of Set-UID
  - **Set-UID** escalates a process's privilege
  - **Set-Nobody** restricts a process's privilege
  - Set the effective user to "nobody"
- Lab description
  - Implement Set-Nobody mechanism
  - Analyze whether it is still dangerous



# Set-UID: Experience

---

- Simple and Focused project
  - Warm-up for the more difficult labs
  - Skills: C programming, kernel code reading, recompile source code and security analysis.
  - Take 1-2 weeks
  - Most students like it



# Access Control List Lab



# Access Control List Lab

---

- Objectives
  - Understand how Access Control works in Minix
  - Understand how ACL works in Minix
  - Extend Minix's ACL



# ACL: Lab Description

---

- Abbreviated ACL
  - Minix's ACL is based on "owner", "group", and "others".
- Full ACL
  - Define permissions for individual users
- Lab Tasks
  - Implement full ACL for Minix



# ACL: Design Issues

---

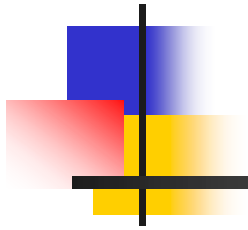
- Where to store ACL?
  - I-node
  - Unused field in I-node
- ACL policies
  - Types: allow, deny, group, etc.
- Utilities
  - `setacl` and `getacl`



# ACL: Experience

---

- A simple project
- The involved coding is not much
- Challenging parts
  - I-node data structure
  - Writing new system calls



# Capability Lab



# Capability Lab

---

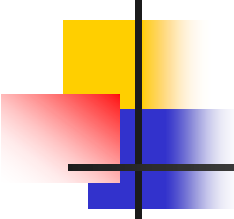
- Capability
  - One type of access control
  - Different from access control list
  - Like tokens
- Objectives
  - Understand capability
  - How capability-based system works
  - Applications of capability



# Capability: Lab Description

---

- Allow user to restrict its own privilege
  - Useful for running untrusted code
  - Useful for Set-UID programs
  - Can't be achieved using ACL
  - Use capability
- We define the following capabilities
  - File-Reading, File-Writing, File-Deleting, and File-Execution
  - Networking capabilities.

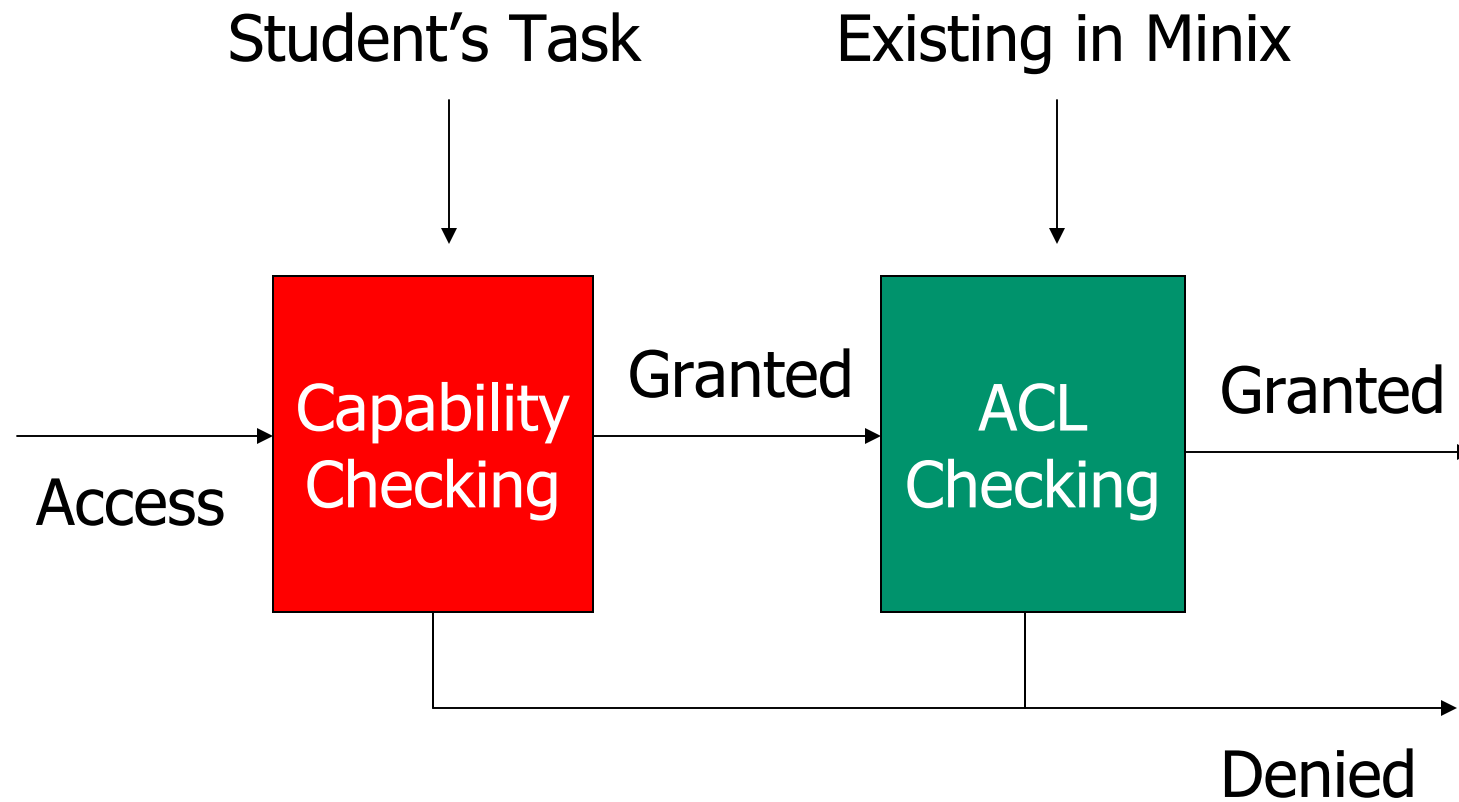


# Capability + Set-UID

---

- Improve Set-UID
  - Set-UID has one capability: **root**
  - Divide **root** capability to many capabilities
  - A program carries those that are needed
  - Can reduce risk

# Capability: Lab Tasks

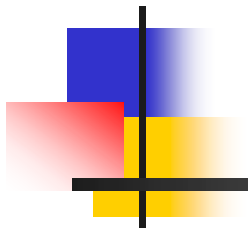




# Capability: Design Issues

---

- The capability system architecture
- How to represent capabilities?
- How to securely store them?
  - Study the file descriptor as an example
- How to initialize the capabilities of a process?
- A process can control its own capabilities
  - Deleting, Disabling, Enabling, Copying, Revocation



# Reference Monitor



# Reference Monitor Lab

---

- RM is an important concept for computer security practitioners
- Properties of RM
  - **Always invoked**: every access is mediated.
  - **Tamperproof**: impossible to bypass.
  - **Small** enough to be subject to analysis and test



# RM: Lab Objectives

---

- Understand the Reference Monitor concept
- See how Reference Monitor works
- Evaluate the Reference Monitor.



# RM: Project Tasks

---

- Develop security policies for Minix
- Find out where the RM is and how RM works in Minix
- Does Minix's RM enforce all the policies you developed?
- How are the 3 properties of RM satisfied?
- Is Minix's RM design good or bad?



# RM: Testing & Improving

---

- Testing Reference Monitor
  - Students are given a modified RM with injected flaws
  - Black-box and White-box testing
- Improving Reference Monitor



# Encrypted File System



# Encrypted File System Lab

---

- Encrypted File System
  - Computer can be physically stolen
  - Protecting removable file system
- Objectives of the Lab
  - Understand and implement EFS
  - Need to combine the knowledge of encryption/decryption, key management, authentication, access control, and security in OS kernels and file systems.

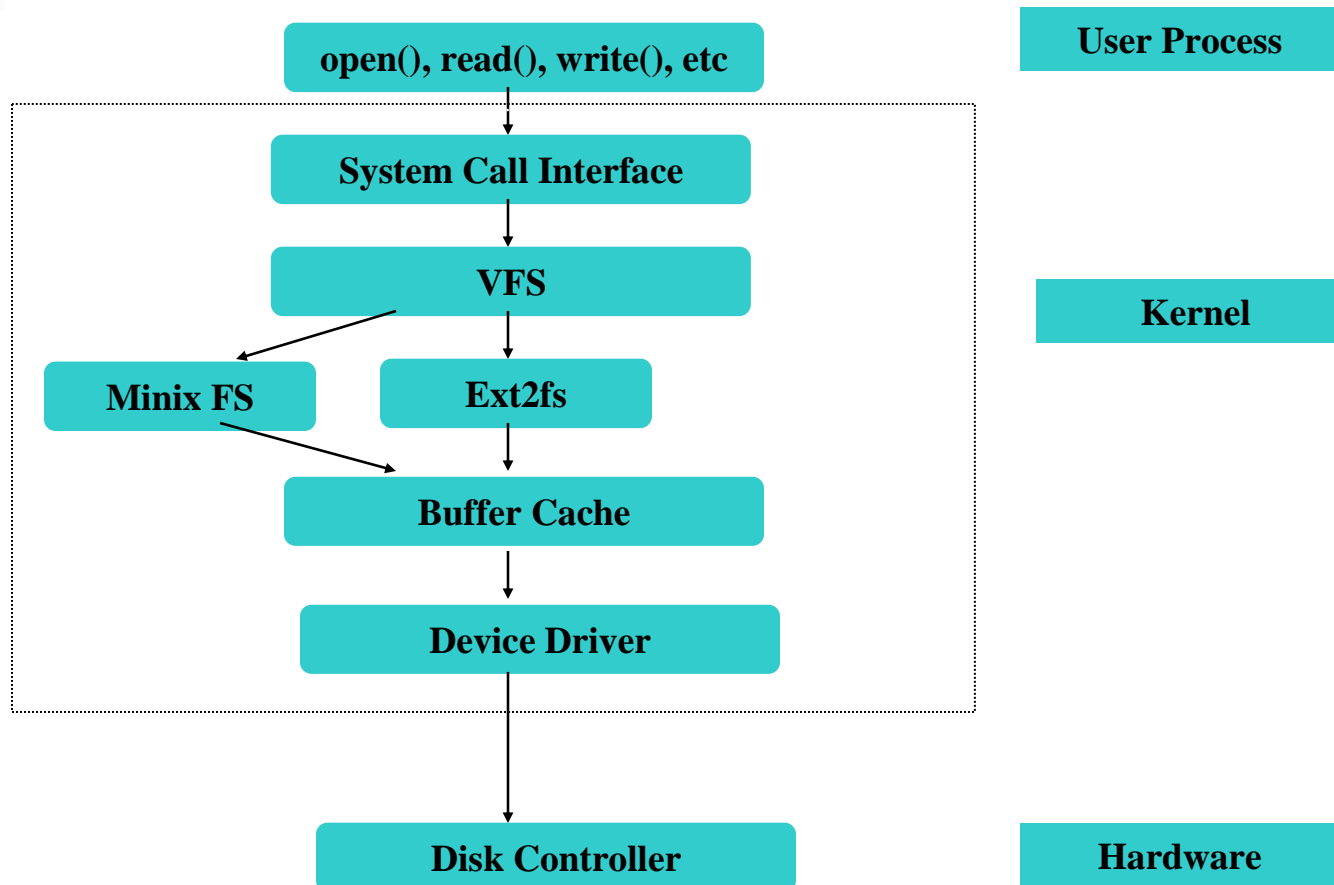


# EFS: Lab Tasks

---

- Add a layer of encryption to the existing Minix file system
- Encrypt and decrypt files on the fly
- Encryption should be transparent
- Keys must be secured

# EFS: Kernel Architecture



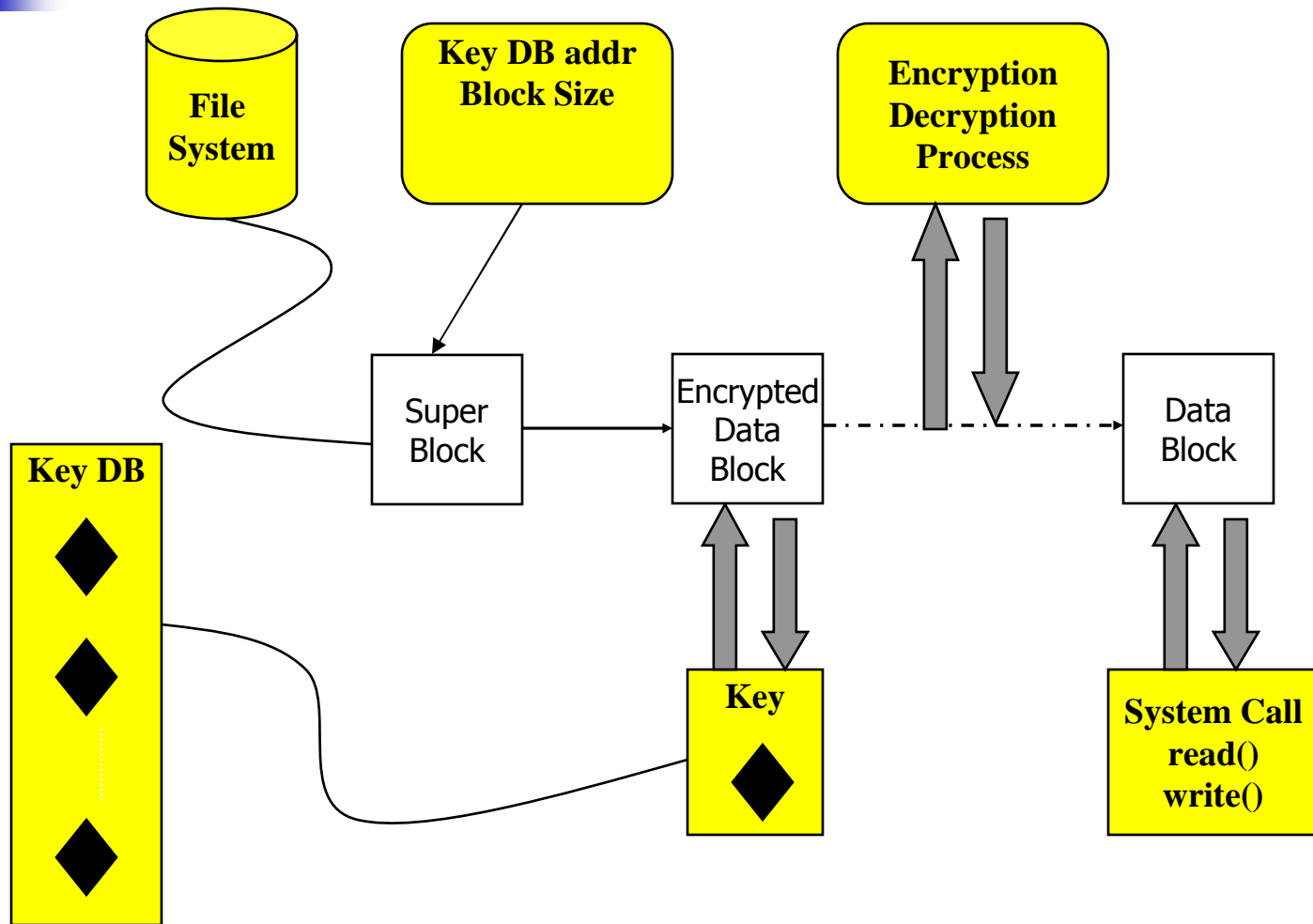


# EFS: Design Issues

---

- File encryption and decryption
  - On-the-fly encryption/decryption
  - Buffering, large files, etc.
- User transparency
- Key management
  - Where to store the keys
  - How to store the keys
- Authentication
- Change of file ownership

# EFS: Architecture





# EFS: Modules

---

- Encryption module
  - Encrypt data chunk in `read()` & `write()`
- Key Management module
  - Change mount/umount
  - Create new system calls for add/del key
  - Allocate double-direct data link in super block for keys



# EFS: Work Load

---

- New system calls
  - 80 lines of code
- Encryption/decryption functions
  - 100 lines of code
- Key management:
  - 200 lines of code
  - 5-7 files



# EFS: Experience

---

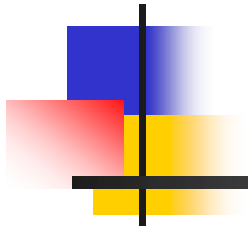
- Help student understand file system
- Customize project for students who do not have sufficient background
- User-space implementation v.s. Kernel-space implementation
- Grading



# EFS: Simplified Version

---

- EFS can be simplified into three sub-projects (for undergraduates)
  - Use encryption algorithms for application
  - Create the corresponding system calls
  - Deals with the key management issues (how to user super block for key DB)



# IPSec Lab



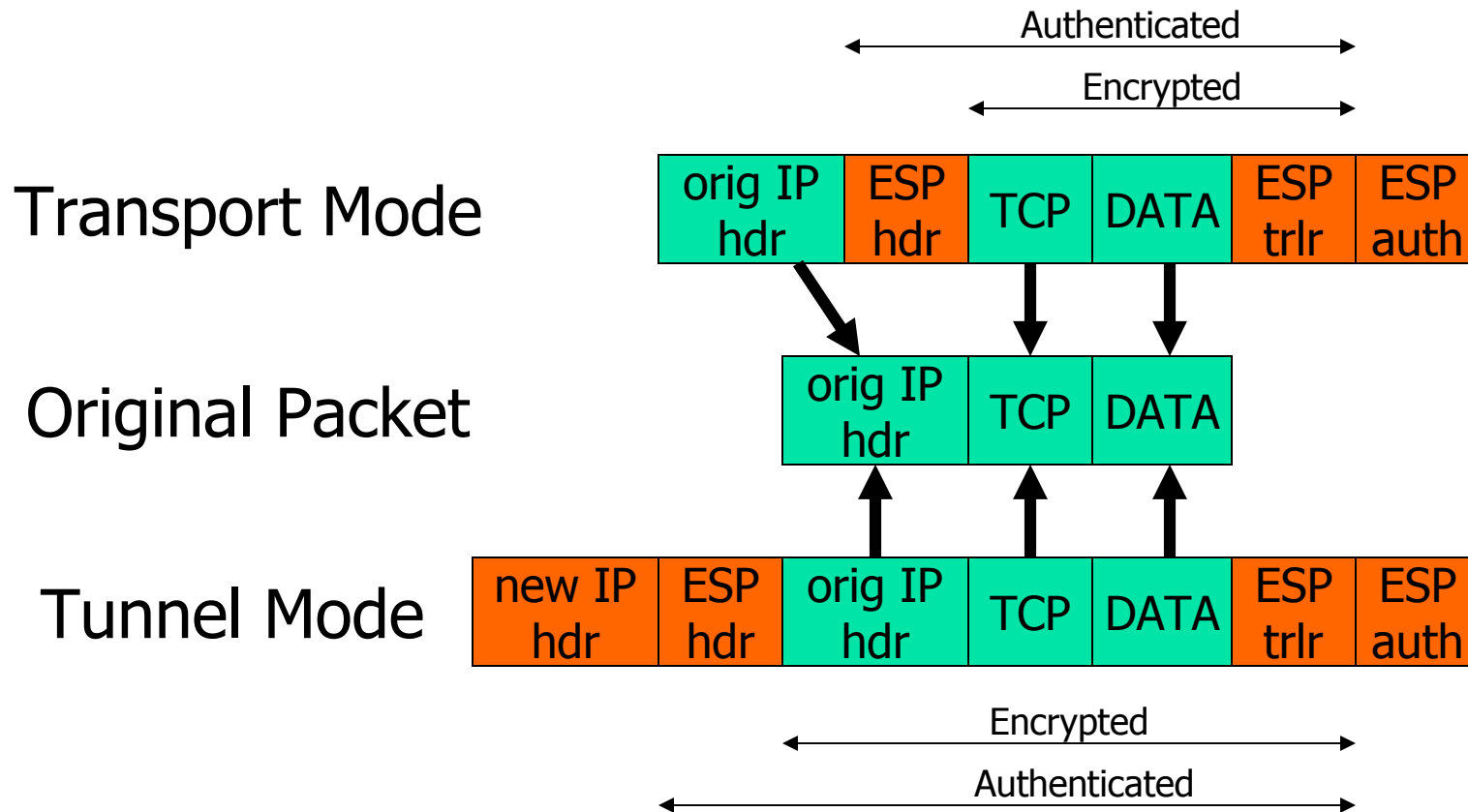
# IPSec Lab

---

- IPSec
  - A suite of protocols for securing network connections
  - Implemented in the IP stack
- Objectives of the lab
  - Learn IPSec protocol, understand how it is implemented
  - Apply comprehensive knowledge: networking, encryption/decryption, key management, access control, authentication, and security in OS kernels etc.

# IPSec: IPSec headers

- IP AH and IP ESP operate in two mode:





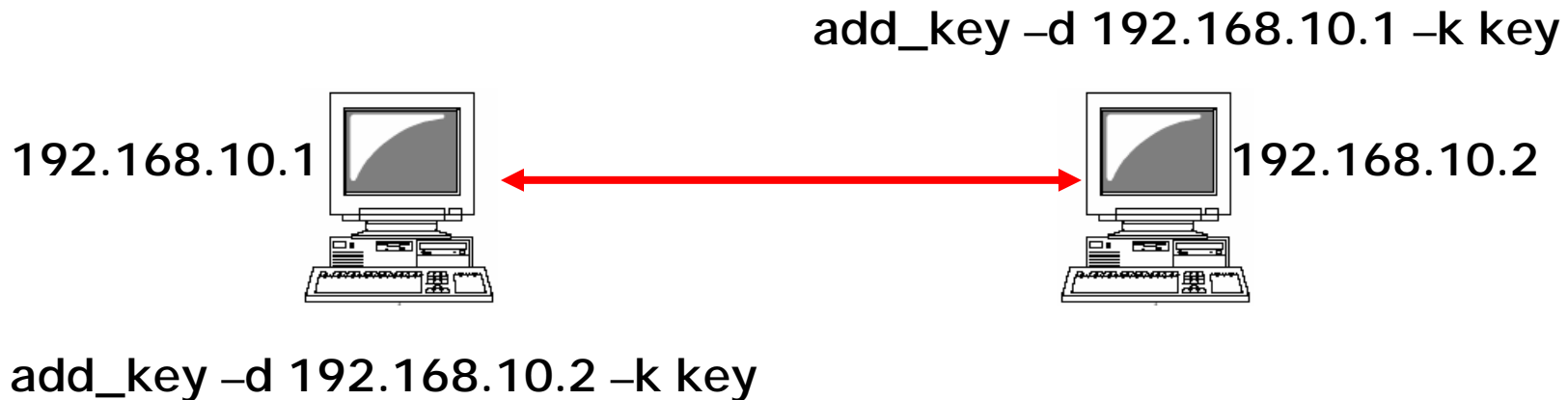
# IPSec: Lab Requirements

---

- Project Requirement
  - Implement the ESP tunnel mode in Minix
  - Use the implemented IPSec to build VPN (Virtual Private Network)
  - 6 weeks
- Project Simplification
  - Keys are manually set (no need to implement the complicated IKE key exchange protocol)
  - Interoperability with other OSes is optional

# IPSec: Encryption Keys

- Simplification: keys are manually set



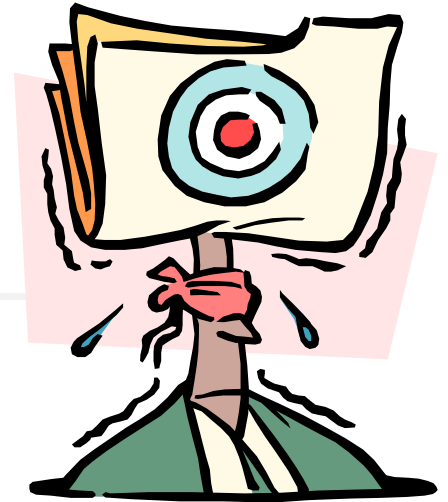


# IPSec: Where to start?

---

- Tracing how packet traverses the IP stack.
- 16,000 lines code in TCP/IP, but ...
- `ip_read.c` & `ip_write.c`: processing incoming and outgoing packets
- `add_route.c` & `pr_routes.c`: good example on how to set keys in kernel using `ioctl()` system call

# IPSec: Design Issues



- How to handle large IP packets?
- Compatibility issues
- How to manage keys? Where to save the keys?
- Will IPSec affect routing?
- Padding for encryption and HMAC



# IPSec: Workload

---

- Read about 2500 lines code in 7 files related to IP and system calls
- AES and HMAC code are given
- IPSec module: **400** lines code for ESP functionalities
- Key management module: around **300** lines code
- TA finished in 3 weeks



# IPSec: Our Experience

---

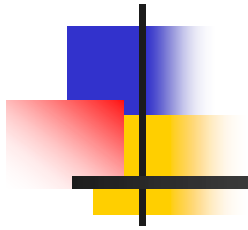
- Most challenging parts:
  - Understanding the data flow
  - Data structure in TCP/IP stack
- How to minimize the time on these parts
  - Develop helping materials
  - Give a lecture on these parts
  - Use web sites



# IPSec: Extension

---

- Compatible with commercial OS
- Expand functionalities to VPN
- Key exchange protocols



---

# Sandbox Lab



# Sandbox

---

- Sandbox provides a safe place for running untrusted programs
- *chroot*( ) changes the root directory of a process
  - Only root has permission to call it
  - We inject a vulnerability by removing this constraint
    - Modify access control policy
    - Let *chroot* program be set-UID





# Sandbox: Lab Tasks

---

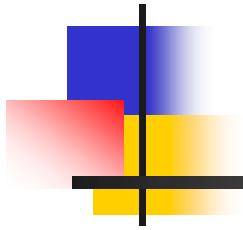
- Read source code
  - `chroot.c`, `su.c` & `stadir.c`
  - Find the vulnerability
  - Implement attack to obtain a root shell
- Design/Implement a solution
  - Normal users can still call `chroot( )`



# Sandbox: Experience

---

- Attack is difficult without hints
  - Hint 1: run `/bin/login` and login as root
  - Hint 2: tell students the passwd part
- Students gives various solutions
  - Good solution is non-trivial
  - We ask students to design a solution and analyze their solutions
  - We did not ask student to implement



# Vulnerability Lab



# Vulnerability Lab

---

- Objectives
  - Gain first-hand experience on software vulnerabilities
  - Understand how a seemingly-not-so-harmful flaw can cause security breaches
  - Practice vulnerability analysis and testing skills



# Vulnerability: Approach

---

- Collect vulnerabilities from real OSes
- Port them to Minix
  - Fault injection
- Currently we have 8 vulnerabilities
- Will develop more



# Vulnerability Types

---

- In kernel space
  - Vulnerabilities are flaws in the kernel
  - System calls
  - File descriptors
  - Kernel buffers
- In user space
  - Set-UID programs
  - Environment variables
  - Symbolic links



# stdio Vulnerability

---

- File descriptors 0, 1, 2: standard devices.

```
/* Your program */  
close(2);  
system(a set-UID program)
```

```
/* The set-UID program */  
fd = open("/usr/adm/syslog"); // fd=2  
...  
printf(stderr, "Error Message"); // stderr=2
```



# Coredump Vulnerability

---

- Coredump allows for the state of a machine to be saved at crash time
  - A `core` file is created by the OS during crash
- **Vulnerability:** if a core file already exists, overwrite it.
- Student's tasks:
  - Exploit this vulnerability
  - Fix the problem.



# Race Condition Vulnerability

---

- Context-switch can happen between “check” and “use”
- **Vulnerability:** The “check” result might be invalid after context switch
- Student’s tasks:
  - “at” program has a vulnerability
  - To make attack easier, we intentionally enlarged the window between `check` and `use`.



# "su" vulnerability

---

- **Vulnerability**
  - `su` is a set-uid program
  - If `/etc/passwd` file can not be opened, system launches a root shell for user to solve problem
- **Student's tasks**
  - Read `su.c` and `open.c`
  - Exploit the vulnerability



# “lpr” Temp File Vulnerability

---

- Some set-uid programs create temporary files in running time
  - Usually temp file does not exist, so create one
  - **Vulnerability**: If the temp file exists, open it (incorrect use of `open( )` system call)
  - **Vulnerability**: temp file's name is predictable



# Experience

---

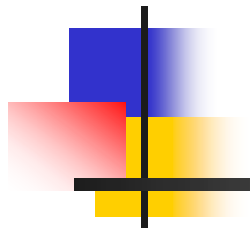
- Compared to design/implementation labs, this lab is easier
  - No need for programming
- Students had a lot of fun
- Hints need to be provided



# More Labs (Under Development)

---

- MAC: Mandatory Access Control
  - Ideas from SELinux
- 80386 Protection Mode
  - Find out ring labels
  - Access control in the protection mode.
- iLAN labs
  - Firewalls, Intrusion Detection System,
  - Syn-cookie, VPN, etc.



# Summary of iSYS Labs



# Current Status

---

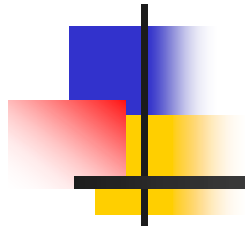
- Have been experimenting with these labs since 2002.
  - Existing labs have been updated
  - New labs were added every year
- Results are encouraging.
  - Students' positive feedbacks.
  - Industry recruiters are interested.
- Developed related lectures.
- On-going development:
  - Extend **iSYS** to network security courses (**iLAN**).



# Lessons

---

- Helping Materials
  - Students spend most of time figuring out how things work in Minix.
  - Helpful materials can reduce this time.
- How things work
  - File System: system calls, I-nodes, data structure.
  - Process: system calls, process table.
  - Network Stack: how data flows in the stack.



# Review of Other Course Projects



# Classification

---

- Analysis & Evaluation
- Design & Implementation
- Vulnerability
- Research



# Analysis & Evaluation

---



# System Analysis

---

- Analyze a popular tool for vulnerabilities
  - Google Toolbar
  - Microsoft Desktop Search
- How much private information is leaked?
- How does this compare to running Kazaa?



# Analyze Virtual Machines

---

- Evaluate the security of the VMWare virtual machine against malicious attempt to harm the host OS
- Explore better way to structure the virtual machine implementation
  - Isolate the security-critical functionality and make the TCB simpler and easier-to-verify



# Analyze Vulnerabilities

---

- Build tools to analyze and improve the security of a computer
  - Select the goal
  - Determine how to measure success or failure
  - Design & implement the tools
  - Analyze its effectiveness and see whether the goal is met



# Security of Network Protocols

---

- Analyze a network protocol for the presence of security flaws
  - 802.11i wireless security
  - Secure multicast and group key management
  - Secure location verification for mobile devices
  - Secure routing in ad-hoc networks



# Design & Implementation

---



# Secure Instant Messaging

---

- Implement SIM program:
  - Account
  - Messaging
  - Buddy list
  - Conferencing
- Security goals
  - Authentication, confidentiality and integrity
- Denial of Service (DoS) resistance



# Resource Bounds

---

- Use proof-carrying code techniques to ensure that malicious code never exceeds a fixed resource bound
  - Insert checks to a global timer wherever we cannot prove a satisfactory upper bound on the running time of the program



# Create a Sandbox

---

- Devise a scenario in which you wish to place attackers in a sandbox.
- Design and implement a sandbox in Linux



# OS fingerprint detector

---

- Explore the “hallmark” characteristics of a variety of OSes
- Write a fingerprint detector



# Linux Security Modules

---

- Understand how ACLs, MLS are patched in kernel
- Add special purpose modules
  - Support the *privilege separation* policies.



# Login authentication

---

- Modern Unix systems support *pluggable authentication modules* (PAM).
- Write a PAM that uses
  - Smart card
  - Palm Pilot
  - Some other interesting techniques.



# Malicious code detection

---

- Parse a program and statically detect if it will misbehave
- Use a long list of patterns to match the flaw in software



# Statical Analysis

---

- Using an existing tool to detect security problems
  - CQUAL and FindBugs,
- Write a new tool to detect new kinds of security problems



# Code Obfuscation

---

- Build an obfuscation system
- Find some obfuscated code and unobfuscate it
- Create a dataflow/control flow tool
  - Study how Office XP detects changes in hardware or detects if it's been copied



# Dynamic analysis

---

- Use a compiler hack, an object-code rewriting hack, or a Java bytecode rewriting hack to detect buggy program behaviors
- Design a tool to check various buggy program behaviors
  - Buffer overflows
  - Common C pointer mishandling issues
  - Like [Purify](#)



# Software Protection

---

- Design and implement a tool to prevent or contain execution of malicious code
- Evaluate usefulness against various types of attacks



# Privacy Protection

---

- Study the strengths and weaknesses of an existing privacy protection scheme
- Propose and implement a new tool for protecting privacy
  - Implement an existing privacy-preserving data mining scheme
  - Propose a new privacy-preserving data mining scheme

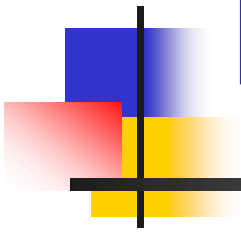


# Secure Email System

---

- Use symmetric-key and public-key techniques to develop a secure email system
- Program have three main functions:
  - A mini-database utility to keep track of certificates that you have acquired from the web site
  - A method to send encrypted and signed email
  - A method to verify and decrypt received email

# Vulnerability: Attack & Detection





# Detect Vulnerabilities

---

- Detect vulnerabilities in programs
  - Boon and Mops, from Berkeley
- Build tools to automate the process of reviewing security-critical source code
  - Use runtime testing, static analysis, model checking, formal verification etc. to detect any interesting classes of security holes



# Firefox Vulnerability Search

---

- Microsoft's Honey Monkey project
  - Identify many sites that exploit Internet Explorer vulnerabilities
  - Discover zero-day vulnerability
- Try the same experiment with the open source browser Firefox



# Preventing Casting Bugs

---

- `typedef unsigned short uid_t;`  
`void dowork(uid_t u);`

```
main()
{
    int x = read_from_network();
    // Squish root (it's not safe to execute dowork()
    // with uid 0)
    if (x==0)
        exit(1);
    dowork(x); // 65536 will be treated as 0
}
```



# Linux System Security (1)

---

- File Security

- Telnet to the machine assigned to your group
- Try to find as many bugs related to file permissions and fix them

- Password Security

- Try to crack the passwords of the users in your machine
- Write a report on your findings



# Linux System Security (2)

---

- Internet Security
  - How to gain access to a computer
  - Report on how secure your system is
  - Describe solutions for the problems you find
- COPS
  - Analyze COPS
  - How useful this tool is for administrator



# Research

---



# Smart Card

---

- Design the security functions for a smart card
  - Make it tamper-proof and hack-proof.
  - Define what approaches an attacker use
  - How each approach could be foiled



# Active Defense

---

- Design a trace-back system
  - Track an attacker back through the Internet
  - Locate the attacker's bases of operation and identity
  - What legal/ethical impediments might there be
  - Estimate performance costs of mechanism
  - How would attackers seek to avoid your trace
  - How to counter their attempts.



# Configure for Forensics

---

- Design a forensic data collector and attack-anticipation software functions
  - Can be executed before, during, and after the attack
  - Attack can be rapidly visible to administrators
  - System collects and analyzes forensic data, identify the attacker and determine the extent of damage



# Key Escrow

---

- Why need key escrow?
- Design a key escrow system
- What attacks could undermine the integrity of the system
- How does escrow system defeat those attacks?



# Key management

---

- "KeyChain" store all other cryptographic keys in a single box
- Design a general-purpose OS mechanism for handling all these different forms of key storage



# Play & Improvement

---



# Intrusion detection

---

- Existing open source Intrusion detection system
  - Snort, Bro, Tripwire and Systrace.
- Explore various techniques used, including some experimental ones
- Modify an existing intrusion detection tool to meet a threat for which the tool was not intended to apply



# Intrusion prevention

---

- Various techniques on Intrusion Prevention
  - Stackguard and Libsafe
  - [PaX/grsecurity](#), [OpenWall](#) and [Program Shepherding](#)
- How to apply these techniques?



# Security auditing

---

- Audit a under-scrutinized open-source package that is security-critical.
  - How to re-structure or re-implement it to make it more robust?
  - What tools make auditing task easier?
  - How effective are existing tools?



# Privilege separation

---

- Sandbox only allows controlled sharing or limited interaction
- How to securely allow this limited interaction in some application context of interest
- Pick an application and investigate how to apply privilege separation techniques to reduce the size of the TCB.



# Side Channel Attacks

---

- Java and other mobile code
  - Attackers can run code on target machines to measure timings or memory operations by observing scheduling or swapping decisions
- Investigate how to mount side channel attacks within the constraints imposed by Java or other widely-deployed mobile code system



# Stamp out SPAM

---

- Devise a best-of-breed technique for detecting and eliminating SPAM
- Determine how attackers create messages that would not be detected as SPAM
- Make your system to learn incrementally
  - When attackers became more sophisticated, or observed your SPAM killer, that they could not use that knowledge effectively to deter your detector



# Voting security

---

- Play with a voting system or machine
- Try to spot problems in the system
- Use cryptographic protocol verification to solve the problem discovered in a real voting system.



# Password Cracking

---

- Study how passwords are created, maintained, and checked by NT and Unix
- Select a cracking tool for Windows NT and Unix
- Crack weak passwords in manageable time (2 – 10 hours)



# Summary

---

- iSYS/iLAN Labs using Minix OS
  - Feel free to use them
  - You are welcome to contribute new labs
  - Create a repository for iSYS/iLAN labs
  - <http://www.cis.syr.edu/~wedu/SCIENS/seed/>