The Evolution of Secure Operating Systems

Trent Jaeger

Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University
Operating Systems

- Make computer systems easier to use
- Improve productivity and collaboration
Operating Systems

- Users make mistakes that may affect others
- Users may have malicious intentions
Need for Security

- The need for operating systems to enforce security requirements was recognized from the advent of multi-user operating systems
Need for Security

- The need for operating systems to enforce security requirements was recognized from the advent of multi-user operating systems


  - “Of considerable concern is the issue of privacy. Experience has shown that privacy and security are sensitive issues in a multi-user system where terminals are anonymously remote.”
Questions

- So, were we done? No, still several difficult questions to address, including

  - (1) What does security mean?
    - Policy: What access can be allowed to sensitive data while still protecting its secrecy and integrity?

  - (2) How do we enforce security effectively?
    - Mechanism: What should be the requirements of a security mechanism to enforce security policies correctly?

  - (3) How do we validate correctness in enforcement?
    - Validation: What methods are necessary to validate the correctness requirements for enforcing a security policy?
Evolution of Secure OS

• In this talk, I will review the evolution of the design of secure operating systems with respect to these questions
  • **Phase 1**: The (Early) Multics Experience (to 1977)
  • **Phase 2**: The Security Kernel Experience (to early 90s)
  • **Phase 3**: Recent and Future Directions (from 90s)
In this talk, I will review the evolution of the design of secure operating systems with respect to these questions.

**Phase 1: The (Early) Multics Experience**
- **Archaen** – “the formation of continents and life started to form”

**Phase 2: The Security Kernel Experience**
- **Proterozoic** – “from the appearance of oxygen in Earth's atmosphere to just before the proliferation of complex life”

**Phase 3: Recent and Future Directions**
- **Phanerozoic** – “starts with the rapid emergence of a number of life forms”
Multics Project (to 1977)

- Importantly, the Multics project explored all three big questions
  - And made important contributions to each
Multics Project (to 1977)

• Importantly, the Multics project explored all three big questions
  ‣ And made important contributions to each

• What does security (policy) mean?
  ‣ Security has to protect secrecy and integrity even when adversaries control processes → mandatory access control
Multics Project (to 1977)

• Importantly, the Multics project explored all three big questions
  ‣ And made important contributions to each

• What does security (policy) mean?
  ‣ Security has to protect secrecy and integrity even when adversaries control processes → mandatory access control

• What does enforcement mean?
  ‣ Enforcement mechanisms must satisfy the reference monitor concept
Multics Project (to 1977)

• Importantly, the Multics project explored all three big questions
  ‣ And made important contributions to each

• What does security (policy) mean?
  ‣ Security has to protect secrecy and integrity even when adversaries control processes \(\rightarrow\) mandatory access control

• What does enforcement mean?
  ‣ Enforcement mechanisms must satisfy the reference monitor concept

• What does validation require?
  ‣ Small code base; design for security; formal verification
Security Policy in Multics

- Found that “security” is more than “protection” (Lampson)

- Protection makes trust assumptions about a user’s processes
  - Your processes are not out to get you
  - Your processes can resist determined attacks
Security Policy in Multics

• Found that “security” is more than “protection” (Lampson)
• Protection makes trust assumptions about a user’s processes
  ‣ Your processes are not out to get you
  ‣ Your processes can resist determined attacks
Mandatory Access Control

- Multics introduced **mandatory access control (MAC)** to prevent such attacks
  - Mandatory – System-defined administration of policies
  - Access control – Information flow or MLS (e.g., Bell-La Padula, Biba)

- User programs are not authorized to
  - Read/Write to data to unauthorized files or processes
  - Or change the access control policy

- Prevents Trojan horses (malware) and compromised programs from violating expected data security
  - No permissions that leak data or depend on less trusted data
MAC Challenges

• Information flow is an ideal
  ‣ Real systems often require operations that violate information flow
    • E.g., services that manage secrets must still be able to reply to client requests
  • Resulting in implementation artifacts outside model
    ‣ Trusted readers/writers or guards that can violate information flow
    ‣ Ad hoc declassifers (secrecy) and endorsers (integrity) to change information flow policies
• Trust in these artifacts requires validation
  ‣ Need (mostly) automated support to fill these gaps
Enforcement in Multics

- Found that enforcement itself must be **systematic and secured**
  - Which OS operations should be protected by authorization checks?
  - How do we know that authorization checks are performed correctly?
  - How do we know that authorization checks and the policy enforced cannot be modified?

- Clearly, an informal approach to the enforcement of policies is insufficient
Reference Monitor

• The Anderson report (USAF 1972) proposed the reference monitor concept to provide
  ‣ “Explicit control must be established over each user’s (programs) access to any system resource which is shared with any other user or system program.”

• Reference Monitor Concept requirements:
  ‣ The reference validation mechanism must be tamperproof
  ‣ The reference validation mechanism must always be invoked (complete mediation over security-sensitive operations)
  ‣ The reference validation mechanism must be small enough to be subject to analysis and tests, the completeness of which can be assured (validation)
Enforcement in Multics

• Found that enforcement itself must be systematic and secured
  ‣ Which OS operations should be protected by authorization checks? Complete mediation over security-sensitive operations
  ‣ How do we know that authorization checks are performed correctly? Validation
  ‣ How do we know that authorization checks and the policy enforced cannot be modified? Tamperproof

• Provides guidance over how to build correct enforcement mechanisms
  ‣ But, some further work is necessary to make these ideas precise
Enforcement in Multics

- Tamperproofing
  - Protection rings
  - Kernel in ring 0
  - Gates protecting kernel entry and exit

- Complete mediation
  - Resources modeled as “segments”
  - Control all segment operations (ACLs, MLS, ring brackets)

- Validation
  - Come back to this
Karger-Schell Analysis

• Demonstrated the importance of following the reference monitor concept
  ‣ Flaws in Tamperproofing
    • Untrusted “master mode” code run outside Ring 0 for performance
  ‣ Flaws in Complete Mediation
    • Failure to mediate some indirect memory accesses

• However, these were both flaws in implementation, not design, that would have been alleviated by following the reference monitor concept correctly
Validation in Multics

- Challenges were seen for validating Multics (circa 1977)
  - Size of the code base – 54 SLOC
    - Although the Multics Final Report suggests that the kernel size can be reduced by approximately half
  - How to do formal validation on a kernel?
    - To this point techniques had not been developed
- Ultimately, the Multics design formed the basis for the B2 assurance level of the Orange Book
  - Security policy model clearly defined and formally documented (B2)
  - Satisfies reference monitor requirements (B3)
A number of projects emerged to address the challenge of validating secure operating systems, which came to be called security kernels. To address three main challenges:

- Reduce size and complexity of operating systems and utility software
- Define security enforced by the OS internal controls
- Validate the correctness of the implemented security controls

July 1983, IEEE Computer
Security Kernel Approach

- **Security Kernel Design**: Ames, Gasser, and Schell
- **Basic Principles**
  - A formally defined security model
    - Complete, mandatory, and validated for security requirements
  - Faithful implementation
    - Transfer model to design incrementally and formally
- **While addressing practical considerations**
  - Extracting security relevant functionality from OS at large
  - Formal specification and validation methods
Security Kernel Approach

- From model to implementation
What techniques are necessary to formally assure a kernel implementation satisfies a security model?

- “verification has turned out to be more difficult than we expected”

Goal: correctness

- Techniques not ready to prove correctness

Approaches (at this time)

- Compare kernel security to information flows allowed
- Specification and implementation correspondence
VMM Security Kernel

• **Choices** in bringing security kernel OS to market
  ‣ High-assurance version of existing OS
    • But, would trail the standard product development lifecycle
  ‣ Custom, high-assurance OS
    • Lack application and ecosystem support

• **Alternative**: high-assurance virtual machine monitor (VMM)
    • VMM security kernel layers under commercial OSes
    • To support multiple OSes and versions
VAX/SVS Project

• Project Successes
  ‣ System was piloted in 1989 – “reasonably successful”
  ‣ “A VMM Security Kernel for the VAX Architecture” was lead paper and Best Paper Award winner at the 1990 IEEE Symposium on Security and Privacy
  ‣ Comprehensive effort for A1 assurance applying formal methods for system design, test, maintenance, and cover channels

• Nonetheless, the project was cancelled in 1990
  ‣ Lack of customers – export controls did not help
  ‣ Lack of features – e.g., no Ethernet support
VAX/SVS Project

• Other issues that may have had an impact
  ‣ Drivers are in the VMM security kernel
    • Direct Memory Access was not controlled
    • All new device drivers must be fully assured
  ‣ Multi-user and privileged VMs
    • Achieving A1 assurance in practice requires tracking individual users, but no visibility into VMs
  ‣ Assembly code
    • About 11K SLOC of the VMM security kernel was implemented in assembly
Recent and Future Work

- Significant advances since then
  - Hardware support for security
    - E.g., IOMMU
  - Software architectures for security
    - E.g., Decentralized information flow control
  - Program analysis for security
    - E.g., Validation and retrofitting of security in programs
  - Formal methods for security
    - E.g., seL4
- These advances address several prior limitations
A variety of hardware features have been explored and even introduced in production hardware

- **IOMMU**
  - Control direct memory access for devices
  - Critical for drivers in security kernels

- **Native virtualization**
  - For VMM security kernels

- **Features for specific security requirements**
  - ARM TrustZone, Intel SGX, Intel MPX, Intel PT, etc.

- **Research systems**: CHERI
Software Architectures

- Information flow control in systems and software has been explored broadly - **Decentralized information flow control**
  - Systems that enforce information flow comprehensively and flexibly
    - All processes are controlled or within the model
  - Compiler-based methods for validating information flow control enforcement in programs
    - Declassification/endorsement as first-order principles in languages
  - Applications of such methods to real systems (in research)
    - Android-Weir (USENIX 2016), OpenStack-Pileus (ACSAC 2016), Web Applications-DATS (ASPLOS 2018), Fabric/Mobile Fabric (Oakland 2012)
Program Analysis

- In addition to information flow, program analysis methods for security examine a number of other challenges
  
  - Automated Privilege Separation
    - Enable programmers to decompose their monolithic programs into components to satisfy information flow
  
  - Program Specialization
    - Remove code unnecessary for particular deployment
  
  - Automated Hook Placement
    - Identify where to place security checks into existing programs
  
  - How to utilize such techniques in concert to “design for security”?
Take Away

- The importance of enforcing security in operating systems has been long recognized
- **Multics** examined the dimensions of what to enforce (policy) how to enforce (mechanism), and need for validation
- **Security kernel** projects explore how to validate real systems based on security designs converted to implementations
- **Recent and future** work shows promise of overcoming some of the major challenges that have held back prior work
- With the availability of a formally verified core kernel, there is an opportunity to develop secure operating environment
Thank You!

• Questions?
• E-mail: tjaeger@cse.psu.edu
Thank You!

- Questions?
- E-mail: tjaeger@cse.psu.edu

About SYNTHESIS

This volume is a printed version of a work that appears in the Synthesis Digital Library of Engineering and Computer Science. Synthesis Lectures provide concise, authoritative presentations of important research and development efforts, published quickly in digital and print formats. For more information, visit www.morganclaypool.com

Series Editor: Ravi Sandhu, University of Texas at San Antonio

SYNTHESIS LECTURES ON INFORMATION SECURITY, PRIVACY, AND TRUST

Operating System Security

Trent Jaeger, The Pennsylvania State University

Operating systems provide the fundamental mechanisms for executing computer programs. Since the 1960s, operating systems designers have explored how to build "secure" operating systems—operating systems whose mechanisms protect the systems against motivated adversaries. Recently, the importance of ensuring such security has become a mainstream issue for all operating systems. In this book, we examine past research that defined the requirements for a secure operating system and research that implemented example systems that met those requirements. We also survey work on systems that have been adopted into mainstream operating systems, systems that have been designed to be secure but have not been deployed into mainstream operating systems, and work on extending mainstream systems that have not been designed to be secure. We conclude the book with a survey of implementation challenges that we are still exploring to this day. History of operating systems design shows us how achieving the necessary security is challenging. Recent security attacks on mainstream systems often result in implementation challenges that we are still exploring to this day. History of operating systems design shows us how achieving the necessary security is challenging. Recent security attacks on mainstream systems often result in implementation challenges that we are still exploring to this day. History of operating systems design shows us how achieving the necessary security is challenging. Recent security attacks on mainstream systems often result in implementation challenges that we are still exploring to this day. History of operating systems design shows us how achieving the necessary security is challenging. Recent security attacks on mainstream systems often result in implementation challenges that we are still exploring to this day. History of operating systems design shows us how achieving the necessary security is challenging. Recent security attacks on mainstream systems often result in implementation challenges that we are still exploring to this day. History of operating systems design shows us how achieving the necessary security is challenging. Recent security attacks on mainstream systems often result in implementation challenges that we are still exploring to this day. History of operating systems design shows us how achieving the necessary security is challenging. Recent security attacks on mainstream systems often result in implementation challenges that we are still exploring to this day. History of operating systems design shows us how achieving the necessary security is challenging. Recent security attacks on mainstream systems often result in implementation challenges that we are still exploring to this day.