CSE 509 SYSTEMS SECURITY
List of main topics and concepts

R. Sekar
Stony Brook University

May 6, 2020

Contents

Part I

Foundations

Chapter 1

Cryptographic Foundations

(Lecture Topic #7)
Slides: http://seclab.cs.sunysb.edu/sekar/cse509/ln/crypto.pdf
Notes: http://seclab.cs.sunysb.edu/sekar/cse509/ln/crypto_notes.pdf

- Communications vs System security
- Security concerns
- Algorithm Vs Key, Plaintext vs ciphertext, cryptography vs steganography
- Symmetric key ciphers (AES)
- Block vs stream ciphers
- Public key techniques (RSA)
  - Public vs private key, Encryption vs signing
- Hash functions (MD5, SHA)
- Random number generation and their use
- Encryption of bulk data
- Authentication, session keys
- Digital signatures
- Certificates

Chapter 2

Identification and Authentication

(Lecture Topic #8)
Slides: http://seclab.cs.sunysb.edu/sekar/cse509/ln/auth.pdf
Notes: http://seclab.cs.sunysb.edu/sekar/cse509/ln/auth_notes.pdf

- Something you know (secret), have (badge, smartcard) or are (biometrics)

2.1 Passwords

2.1.1 Attacks and Defenses

- Password theft
  - Storing one-way hash to avoid direct thefts
    - But reuse of reuse of passwords across sites can allow a cracked password to be reused on other sites
  - Password theft at source (violates trusted path)
    - phishing
      - defenses: site keys, security questions, SSL, password managers
    - keyboard sniffers, ...
- Offline password cracking
  - Brute-force attacks
  - Dictionary attacks
  - Slowing down offline attacks using salt values
  - Speeding up offline attacks using GPUs
- Online cracking and defenses
  - Limit number and/or increase delays
  - CAPTCHAs
- Ease of remembering Vs guessing
  - Password rules and expiry dates

2.2 Variants and Improvements to Passwords

- Master password (ssh, browsers)
- One-time password schemes
- Challenge-response schemes

  - SSH
  - SSL
- Multi-factor authentication
  - Example: SecureID
- Visual passwords

2.3 Biometrics

- Metrics: Fraud, insult rates
- Techniques
  - Handwritten signatures
  - Fingerprint
  - Iris
  - Face
  - Voice
  - Speech
- Use in identification Vs authentication

Chapter 3

OS Security and Access Control

Lecture Topic #12
Slides: http://seclab.cs.sunysb.edu/sekar/cse509/ln/ac.pdf
Notes: http://seclab.cs.sunysb.edu/sekar/cse509/ln/policynotes.pdf

- Terminology and concepts

3.1 Discretionary Access Control (DAC)

3.1.1 Access Control and Capabilities

- Access control matrix
- ACLs
- Capabilities
  - Motivation: the "confused deputy" problem
  - Requirements
  - Advantages, drawbacks and current state of deployment

3.1.2 UNIX

- permissions model
- resources
- userids (effective, real, saved)
- Other details: groups, umask, sticky bit, ACLs, locks

3.2 Mandatory Access Control

- Motivation
  - Carelessness of owner
  - Trojan-Horse problem

3.2.1 Multi-level security (MLS)

- Labels
- Policies
  - Bell-La Padula (confidentiality)
  - Biba (integrity)
  - LOMAC (integrity)
- Benefits
  - "*" property: policy governs not only the first access to data, but also all future uses.
- Drawbacks
  - Label creep
  - Policy exceptions require unlimited trust

3.2.2 Domain and Type Enforcement (DTE)

- Approach overview
- Benefits: Supports notion of limited trust
- Drawbacks:
  - Policies tend to get very complex; in practice, one policy per application
  - No guarantee about future use of accessed data
- Relationship to SELinux
  - SELinux combines DTE and MLS, but usually it is DTE that is enabled

3.3 Other Policies

- POSIX capabilities don't closely follow the definition of OS capabilities discussed above.
  - Purpose
  - Implementation in Linux
- Commercial policies (as opposed to MLS developed in the context of military systems)
  - Examples: Clark-Wilson policy, Chinese wall policy

3.4 Policies suitable for Untrusted Code

- Sandboxing
  - chroot jails
  - Seccomp and Seccomp-bpf
  - userid-based confinement
  - Most are strong enough against malicious code
- Isolation
  - one-way isolation
  - two-way isolation (e.g., Virtual machines)
  - information flow containment

Chapter 4

Virtual Machines

Lecture Topic #10
Slides: http://seclab.cs.sunysb.edu/sekar/cse509/ln/VM.pdf

4.1 Key concepts

- Virtualization
  - What resources can/should be virtualized?
- VMM (aka hypervisor)
- VM

4.2 Types of virtualization (Process, Namespace, System)

- Process virtualization
- Namespace virtualization
- ("Whole") system virtualization
  - Type I ("bare metal") vs type II ("hosted")
  - Requires virtualizing
    * CPU
    * Memory
    * I/O (aka device or peripheral virtualization)

4.3 System Virtualization

4.3.1 CPU virtualization

- Sensitive vs Privileged instructions
- Criteria for efficient virtualization (Popek-Goldberg requirement)
  - Can use "trap and emulate" approach
- Coping with violation
  - binary translation
    * feasible because user-level does not need translation, but still some performance overhead
  - paravirtualization
    * high performance but requires OS changes (can't be applied by end-users to closed-source OSes)

- Violation of this criteria in original x86
- recent instruction set extensions to rectify this problem

4.3.2 Memory virtualization

- overhead for guest physical to host physical address translation
- cope using nested pagetable support

4.3.3 I/O virtualization

- Guest device driver code execution actions have to be "interpreted"
- Major overheads unless custom drivers are used by guest OS.
- Even with custom drivers, significant overhead

4.4 Security applications of virtualization

- Untrusted code execution (inconvenient)
- Malware analysis
- Honeypot systems
- Coping with compromised OSes (Rootkit detection etc.)
- High-assurance VMs (locked-down OSes with specialized functions)
- Attacks on VMs
  - Antivirtualization: exploit performance divergence between CPU and I/O-intensive code.
  - VMM escape attacks

4.5 Security Challenges

- In cloud environment, co-tenancy makes attacks too easy unless strong isolation is provided
- Even with strong isolation, co-tenancy increases the effectivess of side-channel attacks
- Denial of service is difficult to prevent in all cases, but is less of a problem in practice.

4.6 Docker

- Uses namespace virtualization, plus cgroups for resource accounting and limiting
- Sharing part of host file system with containers can be dangerous
- Docker daemon is another target for attacks
- Should use Linux capabilities to restrict root user within containers

Chapter 5

Principles of Secure System Design

- Principles recommended by Saltzer and Shroeder
- Principles mentioned but not recommended
- Ability to recognize instances where one of these principles has been applied, and explain the application (e.g., recognize that complete mediation or trusted path principles have been violated)

Part II

Malware and Untrusted Code

Chapter 6

Malicious code

Lecture Topic #8
Slides: http://seclab.cs.sunysb.edu/sekar/cse509/ln/malware.pdf
Notes: http://seclab.cs.sunysb.edu/sekar/cse509/ln/malwarenotes.pdf

- Current threat environment
  - Profit-driven crime
  - (state-sponsored) industrial espionage
  - intelligence operations and cyberwar
  - censorship and political control

6.1 Malware types and goals

- Viruses
- Worms
- Rootkits
- Spyware
- Spam
- Phishing
- DDoS
- Extortion
- Botnets

6.2 Code obfuscation

- Code encryption (aka "packing")
- Control-flow obfuscation techniques
- Data obfuscation
- Polymorphism
- Metamorphism

6.3 Behavioral obfuscation

- Anti-virtualization and anti-analysis techniques
- Trigger-based (act benign until given a trigger, e.g., when a magic cookie is received from the Internet)

6.4 Differences between untrusted and benign code

- Defenses for benign code are rarely suitable for untrusted code. Untrusted code could be malware, which implies a strong adversary. It requires correspondingly strong defenses, e.g.,
  - System-call interception
  - Inline-reference monitors
  - Software-based fault-isolation
  - Control-flow integrity
  - ...

Chapter 7

Securing Untrusted Code

7.1 Reference monitors (RM)

- Inline Vs external RMs

7.2 System-call based (External) RMs

- Motivation
  - Advantages of defining RM based on system call interface
  - Drawbacks
  - Relative merits/drawbacks/similarities of LSM framework

7.2.1 System-call interception approaches

  - Outside monitored process address space
    * In-kernel
    * user-space
  - Within monitored process
    * library-interception based
  - Security and performance impact of each approach
  - Hybrid approaches
    * Use existing kernel mechanisms for security (e.g., Seccomp or userids)
    * Use in-process helper library to forward requests to a helper process that implements the RM functionality
    * After granting access, the RM transfers file descriptors back to the monitored process so that read/writes don't have to be intercepted.

7.3 Inline RMs (IRMs)

- Motivation
- Challenge
  - protecting IRM from malicious code that runs within the same process address space
- Fault isolation

7.3.1 Software-based fault isolation (SFI)

- Motivation
- Challenges
  - Security (protection against subversion of inline checks)
  - Performance
- Approach for achieving SFI
  - Address sandboxing
  - Need for compiler support
  - Need for verifying safety of compiler-generated code
- SFI for CISC (specifically x86)
  - CISC Challenges (variable-length instructions, too few registers)
  - Solution (PittSFIeld)

7.3.2 Control-flow integrity

- SFI techniques have to worry about control-flow bypassing IRM checks, so SFI requires CFI
- Better to build CFI first and layer SFI on top
- Approaches for realizing CFI
  - Focus on Approach 2. (You can ignore approach 1 for the exam.)
- Exploit mitigation using CFI

7.3.3 Native Client (NaCl)

- Motivation and context
- Relationship to PittSFIeld and CFI

Chapter 8

Browser security and Web-related Vulnerabilities

Lecture Topic #15
Slides: http://seclab.cs.sunysb.edu/sekar/cse509/ln/websec.pdf

8.1 HTTP protocol

- HTTP overview
  - Request and response
  - HTTP headers
  - Stateless nature
- HTTP request types
  - GET
    - Parameters and how they are encoded
  - POST
  - When to use GET and when to use POST
- Coping with statelessness
  - Cookies and how they are used
    - Session maintenance using session cookies
      - session cookie is simply a handle to access client-specific data maintained on the server side
    - Lifetime
  - Other ways to maintain state: (hidden) form fields
- Authentication
  - HTTP authentication (uses HTTP headers)
  - Cookie-based authentication
  - Hidden form-field based
  - Need to use TLS (previously SSL) to transmit sensitive info (passwords)

8.2 Web Browsers

- Frames:
  - Basic unit of isolation in browsers.
  - Each browser tab is a frame; additional frames can be created using the "frame" and "iframe" HTML-tags
  - Each frame is associated with a domain — it is the domain name component of the URL from which the frame's contents were fetched.
  - All data associated with a frame "belongs" to the domain associated with that frame.
    * web page content (including Javascript code),
    * cookies
    * web server response
- Javascript:
  - Type-safe general-purpose language that limits access to local resources on the browser's host machine
  - Browser provides an API for Javascript programs to interact with the browser in a safe way.
  - Scripts can construct web pages on the fly by
    * querying page content using the DOM (Document Object Model) API
    * navigating the page and updating content (also using DOM API)
  - Scripts can modify behavior of typical user actions (e.g., clicks, form submission, etc.)
  - Scripts can spontaneously (i.e., without any user action) submit a form
- Hyper links and form submission:
  - When the user clicks a link (which may submit a form), the current frame is destroyed. (The same holds for submit operations by a script.)
  - All cookies associated with the domain of the URL clicked (or typed in the URL bar or submitted by a script) are automatically sent to that server.
  - The response is rendered in a new frame associated with that domain.
- XMLHttpRequest (XHR):
  - Key difference: XHR does not destroy the current frame. Instead, data is returned to the script that calls this API.
  - Introduced to support Web 2.0 applications that continue to fetch and render content from the domain associated with current frame.

8.3 Security concerns and SOP (Same Origin Policy)

- Threat model
  - User does not trust the web server
    * Prevent server from accessing local resources (files or other devices)
    * restrict code execution to a safe language (JavaScript) in a sandboxed environment.
  - Web server does not trust user
    * User may forge sensitive data: either don't send such data to user, or protect it cryptographically.
  - Web servers for different sites do not trust each other
    * Requires strong isolation of content from different web sites
- SOP
  - Browser partitions its data based on DNS domains of web sites
  - Each web site can only access data belonging to its domain

  - Policy enforced at the granularity of frames
- Security concerns
  - Confidentiality:
    * SOP within the browser, to prevent one domain from accessing another domain's data
    * SSL/TLS over the network
  - Integrity:
    * SOP prevents one domain from writing another domain's data.
    * SSL/TLS provides integrity over the network.
    * Server should cryptographically integrity protect sensitive data that shouldn't be modified by browser user.
    * Gaps:
      · By default, cross-origin GET/POST (but not XHR) requests permitted
      · By default, all browser cookies associated with the target domain are automatically included
    * By default, can include Javascript from different domain
      · Runs with the privileges of the including frame
    * By default, DNS uses unencrypted communication
      · A problem for non-https connections; SSL/TLS prevents forging of certificates even if DNS is compromised

8.4 Common vulnerabilities and exploits

8.4.1 CSRF

- How and why it works
- Limitations ("blind" attack)
- Example attacks
- Defenses
  - Referrer header: why it works and its limitations
  - authentication tokens

8.4.2 Clickjacking

8.4.3 Reflected XSS

- Underlying vulnerabilities
- Exploitation techniques
- Example attacks
- Mitigation techniques

8.4.4 Persistent XSS

- Differences from reflected XSS; relative advantages and drawbacks
- Underlying vulnerabilities
- Exploitation techniques
- Mitigation
- XSS vs CSRF: XSS is strictly more powerful

Part III

Software Vulnerabilities and Defenses

Chapter 9

Memory Corruption Vulnerabilities and Defenses

(Lecture Topic #2)
Slides: http://seclab.cs.sunysb.edu/sekar/cse509/ln/memerr-rand.pdf
Notes: http://seclab.cs.sunysb.edu/sekar/cse509/ln/memerrnotes.pdf

9.1 Common Vulnerabilities and Exploits

9.1.1 Stack-smashing

- Attack and its variants
- Defenses
  - Canaries
    - Weaknesses
  - Shadow stack (save second copy of return address)
    - Weakness: compatibility issues
  - Propolice
    - Protecting RA + BP + local variables
  - Safe stack (no arrays or pointers to objects on safe stack)

9.1.2 Heap overflows

- Defenses
- Canaries
- check pointers before using them
- Separating control data from other data

9.1.3 Format-string bugs

9.1.4 Integer overflows

- Reasons
  - Assignment between variables of different widths
  - comparison or assignment between signed and unsigned
  - arithmetic overflow
- Exploits
  - Typically, integer overflows enable buffer overflow despite bounds checking

9.2 Components of a memory corruption exploit

- Type of memory error
  - Spatial error
  - Temporal error (use-after-free)
- Corruption targets
  - code pointers (e.g., RA, function pointers, exception or signal handlers, GOT entries)
  - data pointers
  - non-pointer data
- Payload
  - Injected code
  - Existing code
    - return-to-libc
    - ROP (Return-oriented programming) gadgets
  - Security-critical data

9.3 Generalized memory exploit defenses

- Target one of the above three components of exploits
- Block memory errors
  - Using metadata on object boundaries (e.g., CRED)
  - Using metadata on all pointers (e.g., SoftBounds)
  - Partial solutions (e.g., AddressSanitizer)
- Protect high-value corruption targets
  - e.g., various stack and heap overflow defenses
- Disrupt corruption effect
  - Address-space randomization to break pointer corruption attacks
  - Data-space randomization extends this protection to all data.
- Block payload execution
  - NX (aka DEP aka W $\oplus$ X)
  - control-flow integrity and related ROP defenses

9.4 Advances in Exploit Techniques

- ROP to overcome NX
- Brute-force attacks on randomization
- Heapspray (also against randomization)
- Target incomplete randomization
- Use memory leaks to leak pointers

Chapter 10

Input Validation Vulnerabilities and Defenses

10.1 Common Input Validation Vulnerabilities

10.1.1 SQL injection

10.1.2 Command injection

10.1.3 Script injection

10.1.4 Format string attacks

10.1.5 XSS

10.1.6 Directory/path traversal

10.1.7 Memory corruption exploits

10.2 Taint-tracking based exploit mitigation

10.2.1 Unified view of input validation vulnerabilities

- input and output
  - untrusted input exerts *undue influence* on outputs that control access to security-sensitive data or resources
- How taint information can distinguish attacks from legitimate behavior
- Importance of byte-granularity taint
  - marking a variable as "tainted" or "untainted" does not help. The content matters at a finer granularity.

10.2.2 Byte-granularity taint-tracking

- Shadow memory based technique
  - For each memory location M, a shadow location M' stores the taint of the data byte stored at M.
  - Each memory copy (including variable assignments) is accompanied by a copy of the corresponding shadow memory locations.
- Implicit flows
  - Control dependence
  - Negative control dependence
  - Access using tainted array index or pointer

- Implementation techniques
  - Source-to-source transformation
  - binary instrumentation
  - direct comparison of input and output (taint inference)
- Performance

10.2.3 Taint-based policies for exploit detection

- Application-specific policies (e.g., using regular expressions)
- Application-independent (e.g., lexical confinement)

Chapter 11

File name related vulnerabilities

- Symlink attacks
- TOCTTOU attacks
- Techniques for improving odds of winning in races
- Avoiding filename related pitfalls

Chapter 12

Vulnerability Taxonomies and Secure Coding

12.1 Taxonomies: CVE and CWE

- CVEs
- CWE top 25
- CWE classification
- OWASP top 10
- Secure coding practices and principles

Chapter 13

Vulnerability Analysis

Lecture Topic #13
Slides: http://seclab.cs.sunysb.edu/sekar/cse509/ln/static.pdf

- Key challenges
  - False positives and negatives
  - Range of properties
  - Scalability
    - How long does it take?
    - How much memory does it need? (State-space explosion)
- Approaches
  - Manual testing
  - Static analysis and verification
  - Dynamic Analysis
    - Symbolic execution
    - Fuzz testing

13.1 Static analysis and verification

- Techniques to identify potential bugs and vulnerabilities
- Difficulties:
  - Requires a model of what is good behavior, or bad behavior
  - "Good behaviors" are typically application specific, and hard to come by
  - "Bad behaviors" can be somewhat more generic
    - Common software vulnerabilities
    - Buffer overflow, SQL injection, ...
    - Inconsistencies (e.g., access check or locking on some program paths, but not others)
  - Usually require access to source code. (Binary-only approaches are typically far too inaccurate.)
- Most program properties are undecidable
  - Static analysis has to approximate in order to terminate.
  - Approximation means that analysis can be sound or complete, but not both.
  - Sound: Guaranteed to find all vulnerabilities
  - Complete: No false positives

13.2 Fuzz testing

- Vulnerabilities often arise due to insufficient testing and optimistic assumptions about input
- This means that incorrect inputs will cause unexpected behaviors
- Random input will typically cause a crash
  - Using a debugger or other means, hackers can find additional information to turn the crash into an exploit!
- Techniques
  - Coverage-guided fuzzing
  - Manually assisted fuzz testing
- Challenges
  - In many cases, random inputs don't work, as they get discarded very early. Most of the code is not exercised.
  - Coverage feedback mitigates this to some extent, but the search is still largely blind
- Tools and state of art

13.3 Dynamic Symbolic Execution

(aka Directed random testing)

- DSE approach
- Examples
- Benefits and weaknesses

13.4 Penetration testing

- Just another name for dynamic vulnerability testing
- Part-manual, part-automated
- Blind: Looks for "typical" bugs
  - Usually, much more limited than coverage-guided fuzzing

Chapter 14

Program Transformations for Security

Lecture Topic #11
Slides: http://seclab.cs.sunysb.edu/sekar/cse509/ln/binrewr.pdf
Notes: http://seclab.cs.sunysb.edu/sekar/cse509/ln/dynTrans_notes.pdf

14.1 Approach

- Maintain additional metadata, check policies using this metadata
- Use automated source-to-source or binary transformations to introduce code for metadata maintenance and policy checking
- In the context of a security-enhancing transformation, we are realizing an IRM using the transformation

14.2 Examples of Security-enhancing Transformations

- All exploit-specific guarding techniques
- Absolute and Relative-address randomization
- Full memory error detection
- Byte-granularity taint-tracking
- SFI, CFI, NaCl
- Note: Some of these transformations don't satisfy all of the requirements for RMs — this is because they target benign code rather than untrusted/malicious code.

14.3 Binary instrumentation (aka rewriting)

- Motivation and benefits
- Approaches
  - Static binary instrumentation (SBI)
    - instrument whole binary before a run
  - Dynamic binary instrumentation (DBI)
    - instrument at runtime
    - one basic block (BB) at a time
    - before the first execution of the BB
  - Relative benefits and drawbacks of SBI and DBI

14.3.1 Static binary instrumentation (SBI)

- Disassembly
  - Linear sweep
  - Recursive disassembly
  - Challenges of static disassembly
  - Effects of optimization on the ease of disassembling binary code
- Code movement and insertion
  - Difficulties and work-arounds
  - Limitations

14.3.2 Dynamic binary instrumentation (DBI)

- Tools using DBI (Pin, DynamoRIO, Valgrind, QEMU, ...)
- Benefits
  - Just-in-time disassembly avoids all challenges posed by static disassembly.
  - No limitation on introducing new code
- Drawbacks
  - Performance (although it is much better than what you might expect)
  - Large and complex runtime infrastructure needed within the instrumented application to support DBI
- DynamioRIO:
  - Overview of structure and operation
  - Application: Program shepherding
  - API for writing instrumentation applications
- Applicability of SBI Vs DBI

Part IV

Advanced Topics

14.4 Miscellaneous Attacks

Lecture Topic #17
Slides: http://seclab.cs.sunysb.edu/sekar/cse360/ln/summ.pdf

Chapter 15

Covert Channels

- Types of covert/side channels
- Reverse-engineering cryptographic keys from algorithm runtime, and possible ways to protect against such attacks.
- Reverse engineering keys via hardware fault injection.
- Tamper resistance
- Emanations: types of emanations, and the feasibility of exploiting them.

Chapter 16

Network-Based Attacks

Attack types

- Probing/Reconnaissance
- Denial-of-Service
- Privilege escalation
- Stringing these types together
  - remote to user, vs user to root

DoS attacks

- need for "amplification effect" in the network
- role of software vulnerabilities in amplifying attacker's capabilities
  - examples of historical vulnerabilities
  - key mitigation techniques.
- role of exceptions and other rarely-encountered scenarios in DoS attacks.
- hiding the true source of DoS.

DDoS

- what is it? how is it carried out?
- how is it used?

Self-propagating attacks

- viruses
- worms
- types of worms
  - scanning strategies

Defenses

Relative importance of:

- recovery preparedness (file system backups!)
- periphery control (firewalls)
- patching
- accountability mechanisms (e.g., auditing, privilege separation).
- antivirus and antimalware software
- locking down configurations and preferences
- deployment of exploit mitigation measures
- vulnerability analysis and penetration testing
- IDS

Chapter 17

Intrusion Detection

The slides and notes for this topic are included in the previous section.

17.1 IDS Overview

- IDS Assumptions and role
- Metrics (TP, TN, FP, FN, Precision, Recall)
  - applicability and interpretation
- Entities monitored (users, applications, network packets)
- Approaches (anomaly/misuse/specification-based detection)
- Algorithms (pattern-matching, machine-learning, ...)

17.2 Network Vs Host-Based IDS

- Network IDS
  - strengths
    - deployment ease, ease of prevention
    - portability/reusability as NIDS rely on standardized network protocols — "signatures" can be shared.
  - weaknesses
    - very low-level of abstraction
      - difficult to observe (encryption)and/or interpret
    - typically, high FPs and FNs
- Host-based IDS
  - Host-based vs Network IDS: differences and relative strengths/weaknesses
  - Events to monitor (system calls, audit logs, application logs, file system content, user interactions)

17.3 System call anomaly detection

- Models of normal behavior (N-grams, HMMs, FSA, PDA, ...)
- Techniques for model construction and their strengths/weaknesses
  - Static analysis
  - machine-learning

- N-grams: strengths (simplicity) and weaknesses (lack of generalization, short window to capture sequence information)
- FSA
  - Benefits (capture richer information about program behavior) and drawbacks (potentially more difficult to learn)
  - Using program counter to simplify learning
- FSA model construction approach
  - Using ptrace (or other system call interception mechanisms) to extract system call and stack trace
  - Coping with ASLR
  - Building FSA from traces
  - Performance: convergence, false positives, ...

17.4 Mimicry attacks and Mitigations

- Additional challenges posed by FSA and other techniques that inspect call stack
- Learning system call arguments to defeat these attacks
  - Weakness of algorithms based on system call names alone
  - Classes of attacks that can be detected with and without system call argument information