CSE 509
Course Summary
Cryptography Basics

- Algorithm Vs Key
- Symmetric key ciphers (DES, AES, ...)
  - Block vs stream ciphers
- Public key techniques (RSA, ...)
- Hash functions (MD5, SHA, ...)
- Random number generation
- Applications
  - Encryption (Block vs Stream Ciphers)
  - Key generation
  - Authentication
  - Digital signatures
  - Certificates
User Authentication

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

- Password-based authentication
  - Storing encrypted passwords, Offline/online Dictionary attacks
  - Ease of remembering Vs guessing
  - Password theft and trusted path
  - Variants and Improvements
    - Master password (ssh, browsers, …)
    - One-time passwords
    - Multi-factor authentication
    - Visual passwords
Authentication: Biometrics

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

- **Discretionary Access Control**
  - Access control matrix
  - ACLs
    - UNIX permission model
  - Capabilities
    - Limited use in OSes

- **Trojan Horse and Mandatory Access Control**
  - MLS: Bell-La Padula, Biba models
    - Benefits and drawbacks
  - Domain and Type Enforcement
    - Benefits and drawbacks
    - SELinux
  - Clark-Wilson policy
  - Chinese wall policy

- **Delegation and trust management**
Principles of Secure System Design

- Least privilege
- Fail-safe defaults (default deny)
- Economy of mechanism (simplicity => assurance)
- Complete mediation (look out for ways in which an access control mechanism may be bypassed)
- Open design (no security by obscurity)
- Separation of privilege (similar to separation of duty)
- Least common mechanism (avoid unnecessary sharing)
- Psychological acceptability (onerous security requirements will be actively subverted by users)
Software Vulnerabilities: Memory Errors

- Memory corruption exploits
  - Stack-smashing
  - Heap overflows
  - Format-string bugs
  - Integer overflows

- Exploit defenses
  - Canaries
  - Separating control data from other data
  - Randomization
    - Address-space (absolute or relative address)
    - Data-space
    - Instruction-space

- Preventing memory errors
  - Definition of memory error
  - Spatial vs Temporal Errors
  - Spatial error defenses
    - “Smart” pointers
    - Out-of-band metadata
    - Jones/Kelly
    - CRED
  - Temporal errors
    - Can be addressed using garbage collection (where feasible)
Injection Vulnerabilities

- **Example attacks**
  - SQL injection
  - Command injection
  - XSS
  - Path traversal
  - Format string bugs
  - Memory corruption/code injection attacks

- **Defenses**
  - Static taint analysis
  - Runtime fine-grained taint-tracking
  - Taint-aware policy enforcement
More Software Vulnerabilities …

- **Browser attacks**
  - XSS
  - CSRF

- **CWE-25**

- **File-name based attacks**
  - Symlink attacks
  - TOCTTOU attacks
    - How to succeed in races …
Program Transformations for Security

- General idea
  - Maintain additional metadata, check policies using this

- Source-to-source transformations
  - Guarding techniques
  - Absolute and Relative-address randomization
  - Full memory error detection
  - Fine-grained taint-tracking
Program Transformation on Binaries

- Key challenges compared to source code
- Static rewriting
  - disassembly techniques and challenges
  - rewriting challenges
- Dynamic translation
  - Dynamo Rio, Valgrind, Qemu, Pin, ...
  - How it achieves speed
- Applications
  - Program shepherding
  - Taint-tracking
- Issues and limitations
Static Analysis for Vulnerability Detection

- Techniques to identify potential bugs and vulnerabilities
- 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
      - Access check or locking on some program paths, but not others
Static Analysis

❖ Usually require source code
  ❖ Binary code analysis limited by absence of type/bounds information, as well as higher level control structures

❖ 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
  ❖ Practical issues: FPs and FNs, scalability, range of properties that can be supported, ...
Dynamic Analysis

- Manual testing
- Random testing ("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 crashes
    - Using a debugger or other means, hackers can find additional information to turn the crash into an exploit
- Coverage-guided fuzzing
- Manually assisted fuzz testing
  - In many cases, random inputs don't work, as they get discarded very early
    - Most of the code is not exercised
  - Better to ensure that some parts of input are valid, so as to traverse more program paths
    - Remaining parts of input can be fuzzed
Symbolic Execution

- “Intelligent” approach that chooses inputs to ensure more coverage
  - Often based on some form of symbolic execution
    - Variables left unbound
    - As conditions are tested, constraints on unbound inputs are gathered, depending on whether “then” or “else” clause is taken
    - When multiple conditions are present on the value of a variables, use a constraint solving procedure to narrow down the range
  - Key challenges
    - Range of constraints that can be handled
    - state-space explosion
    - Many approaches choose to bind variables to concrete values when faced with these problems

- Penetration testing
  - Just another name for dynamic vulnerability testing
Malicious Code

- Current threat environment: Profit-driven crime

- Types
  - Viruses
  - Worms
  - Spam
  - Phishing
  - Botnets
  - Rootkits
  - Spyware
  - DDoS
  - Extortion
  - Cyberwar
Malicious code: Stealth Techniques

- **Stealth and Obfuscation**
  - Behavioral obfuscation
    - Anti-virtualization and anti-analysis techniques
    - Trigger-driven
  - Code obfuscation
    - Control-flow obfuscation
    - Data obfuscation
    - Encryption and packing
    - Polymorphism
    - Metamorphism
Untrusted code defense

- Untrusted code implies strong adversary, requires correspondingly strong defenses
  - Mechanisms
    - System-call interception
      - Techniques and trade-offs
    - Inline-reference monitors
      - Issues, challenges
      - Software-based fault-isolation
    - RISC and CISC
      - Control-flow integrity

- Defenses
  - Sandboxing (confinement policies)
    - Policies are hard to write!
    - Indirect attacks!
    - Example: Native Client
  - Isolation
    - Virtual machines
      - VMware, Xen, KVM, Qemu
    - One-way isolation
      - With copy-on-write
    - Complete isolation
      - Smart phones
  - Information flow mediation
    - Vista (one-way)
    - MLS (two-way)
Isolating untrusted code: Virtual machines

- Process Vs Namespace Vs System virtualization
- Type I and Type II VMMs
- Paravirtualization Vs full virtualization
- Implementation techniques
  - Binary translation, paravirtualization, hardware-assisted virtualization
- Memory virtualization
- Security applications
  - Honeypots, sandboxes, malware analysis, high-assurance Vms
  - Protection from compromised OS
Untrusted code: Java, Javascript and Browser Security

- **Safe languages**
  - Java
  - Type safety, byte-code verification, loader checks
  - Sandbox model, stack inspection, doPrivileged
  - Javascript
  - Type safe language, better integration with browser, security based on removing OS access

- **Browser security**
  - HTTP protocol (GET/POST), cookies, authentication
  - HTML forms, parameters, server-side processing
  - Same origin policy, application to scripts, frames, network reads; Ajax and XmlHttpRequests
  - Reflected and persistent XSS; XSS vectors and defenses
  - Other injections (HTTP headers, ...)
  - CSRF and defenses
Side-channel attacks and physical security

- **Covert channels**
  - Intentionally embedded
  - Implicit flows, timing, steganographic techniques, ...

- **Side channel attacks**
  - Timing analysis, power monitoring
  - Differential fault analysis
  - Emanations (keyboard, power, screen/camera, shock sensor)
  - Remanence

- **Physical layer attacks and tamper resistance**
  - Transmit info by file name or metadata (e.g., timestamp)
    - Information retrieved by checking file presence or stat
      - No need to read the file (or have read permissions on the file)
  - “Port-knocking”
    - Transmit info by probing network ports in a certain sequence
  - tcp acks or retransmissions, packet fragmentation, ...
Side-channel attacks and physical security

- **Covert channels**
  - Timing, implicit flows, DNS requests, ...

- **Side-channels**
  - Execution time
Intrusion Detection

- **Network intrusions**
  - Protocol attacks (Teardrop, Synflood, Smurf, …)
  - DDoS
  - Botnets
  - Reflection attacks
  - Worms

- **Attack stages**
  - Probing
  - DoS
  - Privilege escalation
Intrusion Detection

- False positives and negatives

- Observation points:
  - Host-based Vs Network intrusion detection
    ▪ Benefits and drawbacks

- Techniques
  - Anomaly detection
  - Misuse detection
  - Specification-based detection

- Algorithms
  - Pattern-matching
  - Machine learning
Host-based Intrusion Detection

- **Models**
  - Strings, finite state automata, PDA, ...
  - FSA based technique (using system calls)

- **Evasion: Mimicry attacks**

- **Dataflow Vs Controlflow models**