Security Attacks & Defenses

Emin Gun Sirer

Outline

- Attack Nomenclature
  - Trojan horses, login spoofers, logic bombs, trap doors, viruses, worms, buffer overflows, DoS, protocol attacks, etc.
- Defense mechanisms
  - Firewalls, virus scanners, integrity checkers, intrusion detection
- Mobile code
  - Software fault isolation
  - Safe interpreters
  - Language-based protection
  - PCC

Trojan Horses

- A malicious program disguised as an innocent one
- Login spoofers are a specialized class of Trojan horses
  - Can be circumvented by requiring an operation that unprivileged programs cannot perform
  - E.g. Start login sequence with a key combination user programs cannot catch, CTRL+ALT+DEL on Windows

Logic Bombs and Trapdoors

- Hidden, out-of-spec code to go off in the future when certain conditions are met
  - In one case, program checked payroll records for two consecutive periods
- Attacked company has the option of calling the police or hiring the perpetrator as a "consultant"
  - Sometimes propagated by companies to ensure steady income stream in the future
- A classic trapdoor attack by Ken Thompson, "Trusting Trust"
  - Attack code places a trapdoor in a system utility
  - Attack code places a trapdoor in the system compiler to go off when recompiling the utility
  - Attack code places a trapdoor in the compiler to go off when recompiling the compiler

Viruses and Worms

- Viruses: passive code attached to other programs
  - E.g. a program that modifies MS Word
- Worms: code that actively replicates itself and does not depend on the execution of another program to spread
  - E.g. the Internet worm
- Buffer overflow
  - C string libraries are hard to use correctly, easy to allow outsiders to write outside string bounds
  - Most OS code is written in C, ergo many systems have vulnerabilities
  - If a string is stored on the stack, someone can modify the behavior of a program by going off the end of the string and changing a return address stored on stack

Denial of service

- Client sends a legitimate-looking request for service to a service provider
- Service provider commits the necessary resources to provide the service
  - Ports, buffer space, bandwidth
- The resources are wasted, legitimate users get diminished service
  - Usually launched from many computers controlled by attackers
- Possible whenever the cost to ask for service is far cheaper than the cost of providing it
  - Challenge-response mechanism, selective packet tagging

Other Attacks

- Protocol Attacks
  - Attacks on vulnerabilities in security protocols
  - Often based on a formal, abstract model of the security protocol and its implementation
  - E.g. 802.11b security
- Brute force attacks

Security Enforcement

- Need tools to reduce the exposure of systems to security attacks
- Firewall: a router that restricts network traffic to those flows that fit a security policy
  - E.g. "no incoming mail except to mailhost," "no fingerd," "no TCP unless initiated internally"
- Firewall protects against bad packets
  - Instead of protecting *every* machine on the network, need only protect one firewall on the perimeter
- Many attacks are at a higher level than bad packets

Virus Scanners

- Scan the static program images on disk to check if they contain viruses
- Viruses have well-known signatures and modes of behavior
  - A virus could encrypt the malicious code, but needs an unencrypted section to decrypt it – look for decryptor
  - A virus could mutate the decryption engine to avoid discovery – perform fuzzy search for polymorphic viruses
- Many public databases contain information on virus behavior, scanners compare what's on the disk against the database
  - Performance an issue, not effective for worms
- E.g. McAffee

Integrity Checkers

- Instead of looking for viruses, look for change
  - Compute a checksum for every program on disk
  - Encrypt the checksums, store on disk
  - Recompute checksums, compare
- It's ok for some files and directories to change
  - A policy language can specify what is ok what is not
  - In general, difficult to differentiate benign changes from malicious ones
  - Can lead to false alarms
- E.g. Tripwire

Lures

- Place a dedicated machine on the network
- Populate it with synthetic users and data
  - Make sure it looks exciting
- Raise a red flag as soon as someone gets into the dedicated machine
  - Well-publicized cases involving crackers with KGB ties
- Pros
  - Early warning system
- Cons
  - Can be a stepping stone to other machines
  - Requires management and administration
  - Legality not clear, can be considered entrapment

Intrusion Detection

- Examine the behavior of programs, alert someone if they are "not behaving well"
  - Difficult to define
  - Some schemes require specifying the range of system calls a program may perform
  - Some schemes use machine learning techniques to derive a profile from a known-to-be-good system
  - Some schemes use static program analysis to determine the range of behavior possible
- In the limit, encompasses all of machine learning
  - Simple schemes can be effective, esp. against worms
  - False alarms

Summary of attacks and defenses

- Many different types of attacks possible
  - Some clever, most not
- Standard techniques, i.e. secure OS design with secure reference monitors, can fail
- Can reduce risks and exposure with firewalls
- Can locate security breaches with virus scanners, signature checkers, intrusion detection tools
  - Emerging field with many opportunities

Mobile Code

- Shipping computation from one host to another is a very useful paradigm
  - Applets: programs can be more compact than equivalent data, can interact with user with low latency
    - Can be used for complex GUIs, page description languages, etc.
  - Agents: program acting on behalf of a user, can interact with its environment with low latency
    - Can be used for data collection (e.g. price comparison), load-balancing, long-lived computing tasks
  - Servlets, ASPs: code submitted by clients that would like to run in the context of a larger software system
    - Web servers, rent-a-server, database systems, etc.

Problems

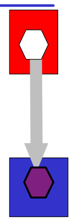
- Mobile code is invaluable in building extensible systems
- But in general, running code provided by someone else poses a security risk
- Could place every extension in a separate hardware address space
  - The code could perform any read, write, jump operation and the MMU would catch any missteps
  - The OS could catch every system call and direct through a reference monitor
  - BUT, the extension code typically must run in the same protection domain as the base system to be useful

Mobile Code Protection

- Can we place extension code in the same address space as the base system, yet remain secure ?
  - Imagine how an app can modify the paging policy the OS uses for its pages
- Many techniques have been proposed
  - SFI
  - Safe interpreters
  - Language-based protection
  - PCC

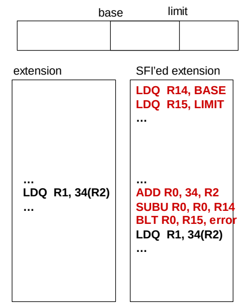
SFI

- Control what the application can do by managing the instruction stream
- Software fault isolation (SFI)
  - Assign a range of contiguous addresses to each extension
  - Rewrite the extension's code segment, inserting checks before every read, write and jump to ensure that it is legitimate
- Checks can be cheap
  - Need only recompute address and perform range check, 3-7 instructions



SFI Loads and Stores

- Every load and store is preceded by the check that the hardware would have done
- Dedicate two general purpose registers to hold the base and limit
- Modern processors have extra stall cycles during which the checks can be performed



SFI control flow

- An extension should only be able to jump to well-defined entry points in the system
- Restrict control flow to indirect jumps off of a table

SFI

- Hard to share data
  - Must still be copied from one extension's memory range into another's
- Performance problems
  - The checks extract a high penalty
- Hard to scale to large numbers of extensions

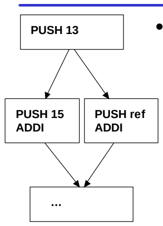
Safe Interpreters

- Restrict code to an interpreted language
  - E.g. telescript, python, perl, tcl, etc…
- The application must go through interpreter for execution
  - The interpreter can enforce security checks at any instruction, the application does not have direct access to hardware
- Slow

Language-based Typesafety

- Constrain the vocabulary of the extensions to a subset of safe instruction sequences
  - Force the programmer to use a language that cannot express unsafe operations
- Many instances
  - Imperative: Java, Modula-3, Limbo
  - Functional: ML, O'caml, Haskell
  - Domain-specific: BPF
- Use a *verifier* to check statically that extensions will not violate safety at runtime

Verification



- Verifier is a specialized theorem-prover
  - System safety depends on axioms such as "thou shalt not create arbitrary pointers through pointer arithmetic"
  - Verifier simulates all possible executions of the program, making conservative assumptions
  - Checks for violation of safety axioms

PCC

- Proof-carrying code
- Extension peresents a certificate that it is safe w.r.t. a safety policy
  - Certificate is a proof in first-order logic
  - The proof is linked to the code
  - The recipient evaluates the proof to check if the safety condition holds over the program
- Details beyond scope of this OS course
  - See courses by Prof. Morisett and Prof. Kozen