Introduction

Basic security properties

• Confidentiality: to prevent unauthorized disclosure of the information
• Integrity: to prevent unauthorized modification of the information
• Availability: to guarantee access to information
• Authentication: to prove the claimed identity can be Data or Entity authentication

Auxiliary security properties

• Non repudiation: to prevent false denial of performed actions
• Authorisation: ”What Alice can do”
• Auditing: to securely record evidence of performed actions
• Attack-tolerance: ability to provide some degree of service after failures or attacks
• Disaster Recovery: ability to recover a safe state
• Key-recovery, key-escrow, etc…
• Digital Forensics

Security mechanisms

• Random Numbers (e.g. for Initialization Vectors)
• Pseudo Random Numbers
• Encryption/Decryption
• Hash functions
• Hash chain (inverted)
• Message integrity code (MIC)
• Message authentication code (MAC and HMAC)
• Digital signatures
    – Non repudiation
• Key exchange (establishment) protocols
• Key distribution protocols
• Time stamping

Types of attacker

- Insiders
- outsiders

Types of attack

• Passive: the attacker can only read any information
– Tempest (signal intelligence)
– Packet Sniffing
• Active: the attacker can read, modify,

generate, destroy any information

1. Overview

Computer Security: protection afforded to an automated information system in order to attain the applicable objectives of preserving the integrity, availability and confidentiality of information system resources (includes hardware, software, firmware, information/data, and telecommunications).

Computer Security Challenges
➢ not simple
➢ must consider potential attacks
➢ procedures used counter-intuitive
➢ involve algorithms and secret info
➢ must decide where to deploy mechanisms
➢ battle of wits between attacker / admin
➢ not perceived on benefit until fails
➢ requires regular monitoring
➢ too often an after-thought
➢ regarded as impediment to using system

Vulnerabilities and Attacks
➢ system resource: with vulnerabilities may
    ●be corrupted (loss of integrity)
    ●become leaky (loss of confidentiality)
    ●become unavailable (loss of availability)
➢ attacks are threats carried out and may be
    ●passive
    ●active
    ●insider
    ●outsider

Countermeasures
➢means used to deal with security attacks
● prevent
● detect
● recover
➢may result in new vulnerabilities
➢will have residual vulnerability
➢goal is to minimize risk, given constraints

Threat Consequences
➢unauthorized disclosure
●exposure, interception, inference, intrusion
➢deception
●masquerade, falsification, repudiation
➢disruption

●incapacitation, corruption, obstruction
➢usurpation
●misappropriation, misuse

Network Security Attacks
➢ classify as passive or active
➢ **passive** attacks are eavesdropping
● release of message contents
● traffic analysis
● are hard to detect so aim to prevent
➢ **active** attacks modify/fake data
● masquerade
● replay
● modification
● denial of service
● hard to prevent so aim to detect

Security Functional Requirements
➢ technical measures:
● access control; identification & authentication; system & communication protection; system & information
integrity
➢ management controls and procedures
● awareness & training; audit & accountability;
certification, accreditation, & security assessments;
contingency planning; maintenance; physical &
environmental protection; planning; personnel
security; risk assessment; systems & services
acquisition
➢ overlapping technical and management:
● configuration management; incident response; media
protection

X.800 Security Architecture
➢ X.800, Security Architecture for OSI
➢ systematic way of defining requirements for security and characterizing approaches to satisfying them
➢ defines:
● security attacks - compromise security
● security mechanism - act to detect, prevent, recover from attack
● security service - counter security attacks

Computer Security Strategy
➢specification/policy
●What is the security scheme supposed to do?
●codify in policy and procedures
➢implementation/mechanisms
●how does it do it?

●prevention, detection, response, recovery
➢correctness/assurance
●Does it really work?
●assurance, evaluation

2. User authentication

fundamental security building block
● basis of access control & user accountability
➢ is the process of verifying an identity claimed by or for a system entity
➢ has two steps:
● identification - specify identifier
● verification - bind entity (person) and identifier
➢ distinct from message authentication

Means of User Authentication
➢four means of authenticating user's identity
➢based on something the individual
● knows - e.g. password, PIN
● possesses - e.g. key, token, smartcard
● is (static biometrics) - e.g. fingerprint, retina
● does (dynamic biometrics) - e.g. voice, sign
➢can use alone or combined
➢all can provide user authentication
➢all have issues

Password Authentication
➢widely used user authentication method
● user provides name/login and password
● system compares password with that saved for specified login
➢authenticates ID of user logging and
● that the user is authorized to access system
● determines the user’s privileges
● is used in discretionary access control

Password Vulnerabilities
➢offline dictionary attack
➢specific account attack
➢popular password attack
➢password guessing against single user
➢workstation hijacking
➢exploiting user mistakes
➢exploiting multiple password use
➢electronic monitoring

Countermeasures
➢stop unauthorized access to password file

➢intrusion detection measures
➢account lockout mechanisms
➢policies against using common passwords but rather hard to guess passwords
➢training & enforcement of policies
➢automatic workstation logout
➢encrypted network links

Use of Hashed Passwords

UNIX Implementation
➢original scheme
● 8 character password form 56-bit key
● 12-bit salt used to modify DES encryption into a
one-way hash function
● 0 value repeatedly encrypted 25 times
● output translated to 11 character sequence
➢now regarded as woefully insecure
● e.g. supercomputer, 50 million tests, 80 min
➢sometimes still used for compatibility

Improved Implementations
➢have other, stronger, hash/salt variants
➢many systems now use MD5
● with 48-bit salt
● password length is unlimited
● is hashed with 1000 times inner loop
● produces 128-bit hash
➢OpenBSD uses Blowfish block cipher based hash algorithm called Bcrypt
● uses 128-bit salt to create 192-bit hash value

Password Cracking
➢dictionary attacks
● try each word then obvious variants in large dictionary against hash in password file
➢rainbow table attacks
● precompute tables of hash values for all salts
● a mammoth table of hash values
● e.g. 1.4GB table cracks 99.9% of alphanumeric Windows passwords in 13.8 secs
● not feasible if larger salt values used
A rainbow table attack is a type of password cracking attack that involves using precomputed tables to reverse cryptographic hash functions. The objective is to find the original plaintext password from its hash value. Rainbow table attacks exploit the fact that many users choose weak passwords, and even if passwords are hashed, common hash functions can be quickly reversed using these precomputed tables.

Password Choices
➢users may pick short passwords
● e.g. 3% were 3 chars or less, easily guessed
● system can reject choices that are too short

➢users may pick guessable passwords
● so crackers use lists of likely passwords
● e.g. one study of 14000 encrypted passwords guessed nearly 1/4 of them
● would take about 1 hour on fastest systems compute all variants, and only need 1 break!

Password File Access Control
➢can block offline guessing attacks by denying access to encrypted passwords
● make available only to privileged users
● often using a separate shadow password file
➢still have vulnerabilities
● exploit O/S bug
● accident with permissions making it readable
● users with same password on other systems
● access from unprotected backup media
● sniff passwords in unprotected network traffic

Using Better Passwords
➢clearly have problems with passwords
➢goal to eliminate guessable passwords
➢whilst still easy for user to remember
➢techniques:
● user education
● computer-generated passwords
● reactive password checking
● proactive password checking

Proactive Password Checking
➢ rule enforcement plus user advice, e.g.
● 8+ chars, upper/lower/numeric/punctuation
● may not suffice
➢ password cracker
● time and space issues
➢ Markov Model
● generates guessable passwords
● hence reject any password it might generate
➢ Bloom Filter
● use to build table based on dictionary using hashes
● check desired password against this table

Token Authentication
➢object user possesses to authenticate, e.g.
● embossed card
● magnetic stripe card
● memory card
● smartcard

Memory Card
➢store but do not process data
➢magnetic stripe card, e.g. bank card

➢electronic memory card
➢used alone for physical access
➢with password/PIN for computer use
➢drawbacks of memory cards include:
● need special reader
● loss of token issues
● user dissatisfaction

Smartcard
➢ credit-card like
➢ has own processor, memory, I/O ports
● wired or wireless access by reader
● may have crypto co-processor
● ROM, EEPROM, RAM memory
➢ executes protocol to authenticate with reader/computer
➢ also have USB dongles

Biometric Authentication
➢authenticate user based on one of their physical characteristics

Biometric Accuracy
➢never get identical templates
➢problems of false match / false non-match
➢can plot characteristic curve
➢pick threshold balancing error rates

Remote User Authentication
➢authentication over network more complex
● problems of eavesdropping, replay
➢generally use challenge-response
● user sends identity
● host responds with random number
● user computes f(r,h(P)) and sends back
● host compares value from user with own computed value, if match user authenticated
➢protects against a number of attacks

Authentication Security Issues
➢client attacks
➢host attacks
➢eavesdropping
➢replay
➢trojan horse
➢denial-of-service

3. Denial of Service

denial of service (DoS) an action that prevents or impairs the authorized use of networks, systems, or applications by exhausting resources such as central processing units (CPU), memory, bandwidth, and disk space

➢ attacks
- network bandwidth
- system resources
- application resources

➢ have been an issue for some time

Classic Denial of Service Attacks
➢ can use simple flooding ping
➢ from higher capacity link to lower
➢ causing loss of traffic
➢ source of flood traffic easily identified

Source Address Spoofing
➢ use forged source addresses
    - given sufficient privilege to “raw sockets”
    - easy to create

➢ generate large volumes of packets
➢ directed at target
➢ with different, random, source addresses
➢ cause same congestion
➢ responses are scattered across Internet
➢ real source is much harder to identify

SYN Spoofing
➢ other common attack
➢ attacks ability of a server to respond to future connection requests
➢ overflowing tables used to manage them
➢ hence an attack on system resource
➢ attacker often uses either
    - random source addresses
    - or that of an overloaded server
    - to block return of (most) reset packets

➢ has much lower traffic volume
    - attacker can be on a much lower capacity link

Types of Flooding Attacks
➢ classified based on network protocol used
➢ ICMP Flood
- uses ICMP packets, eg echo request
- typically allowed through, some required

➢ UDP Flood
- alternative uses UDP packets to some port

➢ TCP SYN Flood
● use TCP SYN (connection request) packets
● but for volume attack

Distributed Denial of Service Attacks
➢ have limited volume if single source used
➢ multiple systems allow much higher traffic volumes to form a Distributed Denial of Service (DDoS) Attack
➢ often compromised PC's / workstations
● zombies with backdoor programs installed
● forming a botnet
➢ e.g. Tribe Flood Network (TFN), TFN2K

Reflection Attacks
➢ use normal behavior of network
➢ attacker sends packet with spoofed source address being that of target to a server
➢ server response is directed at target
➢ if send many requests to multiple servers, response can flood target
➢ various protocols e.g. UDP or TCP/SYN
➢ ideally want response larger than request
➢ prevent if block source spoofed packets
➢ further variation creates a self-contained loop between intermediary and target
➢ fairly easy to filter and block

DNS Amplification Attacks
➢ use DNS requests with spoofed source address being the target
➢ exploit DNS behavior to convert a small request to a much larger response
● 60 byte request to 512 - 4000 byte response
➢ attacker sends requests to multiple well connected servers, which flood target
● need only moderate flow of request packets
● DNS servers will also be loaded

DoS Attack Defenses
➢ high traffic volumes may be legitimate
● result of high publicity, e.g. "slash-dotted"
● or to a very popular site, e.g. Olympics etc
➢ or legitimate traffic created by an attacker
➢ three lines of defense against (D)DoS:
● attack prevention and preemption
● attack detection and filtering
● attack source traceback and identification

Attack Prevention
➢ block spoofed source addresses
● on routers as close to source as possible
● still far too rarely implemented
➢ rate controls in upstream distribution nets
● on specific packets types

● e.g. some ICMP, some UDP, TCP/SYN
➢ use modified TCP connection handling
● use SYN cookies when table full
● or selective or random drop when table full
➢ block IP directed broadcasts
➢ block suspicious services & combinations
➢ manage application attacks with “puzzles” to distinguish legitimate human requests
➢ good general system security practices
➢ use mirrored and replicated servers when high-performance and reliability required

Responding to Attacks
➢ need good incident response plan
● with contacts for ISP
● needed to impose traffic filtering upstream
● details of response process
➢ have standard filters
➢ ideally have network monitors and IDS
● to detect and notify abnormal traffic patterns
➢ identify type of attack
● capture and analyze packets
● design filters to block attack traffic upstream
● or identify and correct system/application bug
➢ have ISP trace packet flow back to source
● may be difficult and time consuming
● necessary if legal action desired
➢ implement contingency plan
➢ update incident response plan

4. Intrusion Detection

Intruders
➢ significant issue hostile/unwanted trespass
● from benign to serious
➢ user trespass
● unauthorized logon, privilege abuse
➢ software trespass
● virus, worm, or trojan horse
➢ classes of intruders:
● masquerader, misfeasor, clandestine user

Examples of Intrusion
➢ remote root compromise
➢ web server defacement
➢ guessing / cracking passwords
➢ copying viewing sensitive data / databases
➢ running a packet sniffer
➢ distributing pirated software

➢ using an unsecured connection to access net
➢ impersonating a user to reset password
➢ using an unattended workstation

Hackers
➢ motivated by thrill of access and status
● hacking community a strong meritocracy
● status is determined by level of competence
➢ benign intruders might be tolerable
● do consume resources and may slow performance
● can't know in advance whether benign or malign
➢ IDS / IPS / VPNs can help counter
➢ awareness led to establishment of CERTs
● collect / disseminate vulnerability info / responses

Hacker Behavior Example
1. select target using IP lookup tools
2. map network for accessible services
3. identify potentially vulnerable services
4. brute force (guess) passwords
5. install remote administration tool
6. wait for admin to log on and capture password
7. use password to access remainder of network

Criminal Enterprise
➢ organized groups of hackers now a threat
● corporation / government / loosely affiliated gangs
● typically young
● common target credit cards on e-commerce server
➢ criminal hackers usually have specific targets
➢ once penetrated act quickly and get out
➢ IDS / IPS help but less effective
➢ sensitive data needs strong protection

Criminal Enterprise Behavior
1. act quickly and precisely to make their activities harder to detect
2. exploit perimeter via vulnerable ports
3. use trojan horses (hidden software) to leave back doors for re-entry
4. use sniffers to capture passwords
5. make few or no mistakes.

Insider Attacks
➢ among most difficult to detect and prevent
➢ employees have access & systems knowledge
➢ may be motivated by revenge / entitlement
● when employment terminated
● taking customer data when move to competitor
➢ IDS / IPS may help but also need:

● least privilege, monitor logs, strong authentication,
termination process to block access & mirror data
Insider Behavior Example
1. create network accounts for themselves and their friends
2. access accounts and applications they wouldn't normally use for their daily jobs
3. e-mail former and prospective employers
4. conduct furtive instant-messaging chats
5. visit web sites that cater to disgruntled employees, such as f'dcompany.com
6. perform large downloads and file copying
7. access the network during off hours

Intrusion Techniques
➢ objective to gain access or increase privileges
➢ initial attacks often exploit system or software vulnerabilities to execute code to get backdoor
● e.g. buffer overflow
➢ or to gain protected information
● e.g. password guessing or acquisition

Security Intrusion
a security event, or combination of multiple security events, that constitutes a security incident in which an intruder gains, or attempts to gain, access to a system
(or system resource) without having authorization to do so.
Intrusion Detection
a security service that monitors and analyzes system events for the purpose of finding, and providing real-time or near real-time warning of attempts to access system resources in an unauthorized manner.

Intrusion Detection Systems
➢ classify intrusion detection systems (IDSs) as:
● Host-based IDS: monitor single host activity
● Network-based IDS: monitor network traffic
➢ logical components:
● sensors - collect data
● analyzers - determine if intrusion has occurred
● user interface - manage / view IDS

IDS Principles
➢ assume intruder behavior differs from legitimate users
- expect overlap as shown
- observe deviations from past history
- problems of:
  - false positives
  - false negatives
  - must compromise

Host-Based IDS
➢ specialized software to monitor system activity to detect suspicious behavior

● primary purpose is to detect intrusions, log suspicious events, and send alerts
● can detect both external and internal intrusions
➢ two approaches, often used in combination:
● anomaly detection - defines normal/expected behavior
• threshold detection
• profile based
● signature detection - defines proper behavior

Audit Records
➢ a fundamental tool for intrusion detection
➢ two variants:
● native audit records - provided by O/S
    • always available but may not be optimum
● detection-specific audit records - IDS specific
    • additional overhead but specific to IDS task
    • often log individual elementary actions
    • e.g. may contain fields for: subject, action, object, exception-condition, resource-usage, time-stamp

Anomaly Detection
➢ threshold detection
● checks excessive event occurrences over time
● alone a crude and ineffective intruder detector
● must determine both thresholds and time intervals
➢ profile based
● characterize past behavior of users / groups
● then detect significant deviations
● based on analysis of audit records
• gather metrics: counter, measures, interval timer, resource utilization
• analyze: mean and standard deviation, multivariate, markov process, time series, operational model

Signature Detection
➢ observe events on system and applying a set of rules to decide if intruder
➢ Approaches (partial overlap):
● rule-based anomaly detection
    • analyze historical audit records for expected behavior, then match with current behavior
● rule-based penetration identification
    • rules identify known penetrations / weaknesses
    • often by analyzing attack scripts from Internet
    • supplemented with rules from security experts

Distributed Host-Based IDS

Network-Based IDS
➢ network-based IDS (NIDS)
● monitor traffic at selected points on a network

● in (near) real time to detect intrusion patterns
● may examine network, transport and/or application level protocol activity directed toward systems
➢ comprises a number of sensors
● inline (possibly as part of other net device)
● passive (monitors copy of traffic)

Intrusion Detection Techniques
➢ signature detection
● at application, transport, network layers; unexpected application services, policy violations
➢ anomaly detection
● of denial of service attacks, scanning, worms
➢ when potential violation detected sensor sends an alert and logs information
● used by analysis module to refine intrusion detection parameters and algorithms
● by security admin to improve protection

Distributed Adaptive Intrusion Detection

Honeypots
➢ are decoy systems
● filled with fabricated info
● instrumented with monitors / event loggers
● divert and hold attacker to collect activity info
● without exposing production systems
➢ initially were single systems
➢ more recently are/emulate entire networks

5. Firewalls and Intrusion Prevention Systems

➢ effective means of protecting LANs
➢ internet connectivity is essential
● for organization and individuals
● but creates a threat
➢ Solution could be to
● secure workstations and servers
➢ also use firewall as perimeter defense
● single choke point to impose security

Firewall Capabilities & Limits
➢ capabilities:
● defines a single choke point
● provides a location for monitoring security events
● convenient platform for some Internet functions such as NAT, usage monitoring, IPSEC VPNs
➢ limitations:

● cannot protect against attacks bypassing firewall
● may not protect fully against internal threats
● improperly secure wireless LAN
● laptop, PDA, portable storage device infected outside then used inside
Types of Firewalls:
- Packet filtering firewall
- Stateful inspection firewall
- Application proxy firewall
- Circuit-level firewall

Packet Filtering Firewall
➢ applies rules to packets in/out of firewall
➢ based on information in packet header
● src/dest IP address & port, IP protocol, interface
➢ typically a list of rules of matches on fields
● if match rule says if forward or discard packet
➢ two default policies:
● discard - prohibit unless expressly permitted
• more conservative, controlled, visible to users
● forward - permit unless expressly prohibited
• easier to manage/use but less secure

Packet Filter Rules: block/allow, port/ip

Packet Filter Weaknesses
➢ weaknesses
● cannot prevent attack on application bugs
● limited logging functionality
● do not support advanced user authentication
● vulnerable to attacks on TCP/IP protocol bugs
● improper configuration can lead to breaches
➢ attacks
● IP address spoofing, source route attacks, tiny fragment attacks

Stateful Inspection Firewall
➢ reviews packet header information but also keeps info on TCP connections
● typically have low, “known” port numbers for server
● and high, dynamically assigned client port n.
● simple packet filter must allow all return high port numbered packets back in
● stateful inspection packet firewall tightens rules for TCP traffic using a directory of TCP connections
● only allow incoming traffic to high-numbered ports for packets matching an entry in this directory
● may also track TCP seq numbers as well

Application-Level Gateway
➢ acts as a relay of application-level traffic
● user contacts gateway with remote host name

● authenticates themselves
● gateway contacts application on remote host and relays TCP segments between server and user
➢ must have proxy code for each application
● may restrict application features supported
➢ more secure than packet filters
➢ but have higher overheads

Circuit-Level Gateway
➢ sets up two TCP connections, to an inside user and to an outside host
➢ relays TCP segments from one connection to the other without examining contents
● hence independent of application logic
● just determines whether relay is permitted
➢ typically used when inside users trusted
● may use application-level gateway inbound and circuit-level gateway outbound
● hence lower overheads

Firewall Basing
➢ several options for locating firewall:
➢ bastion host
➢ individual host-based firewall
➢ personal firewall

Bastion Hosts
➢ critical strongpoint in network
➢ hosts application/circuit-level gateways
➢ common characteristics:
● runs secure O/S, only essential services
● may require user auth to access proxy or host
● each proxy can restrict features, hosts accessed
● each proxy small, simple, checked for security
● each proxy is independent, non-privileged
● limited disk use, hence read-only code

Host-Based Firewalls
➢ used to secure individual host
➢ available in/add-on for many O/S
➢ filter packet flows
➢ often used on servers
➢ advantages:
● taylored filter rules for specific host needs
● protection from both internal / external attacks
● additional layer of protection to org firewall

Personal Firewall
➢ controls traffic flow to/from PC/workstation
➢ for both home or corporate use
➢ may be software module on PC

➢ or in home cable/DSL router/gateway
➢ typically much less complex
➢ primary role to deny unauthorized access
➢ may also monitor outgoing traffic to detect/block worm/malware activity
Firewall Locations: Internal DMZ network, internal protected network, external firewall, internal firewall
Virtual Private Networks: firewall with IPsec, secure IP packet
Distributed Firewalls: Internal DMZ network, internal protected network, external firewall, internal firewall, external DMZ network, host-resident firewall

Firewall Topologies
➢ host-resident firewall
➢ screening router
➢ single bastion inline
➢ single bastion T
➢ double bastion inline
➢ double bastion T
➢ distributed firewall configuration

Intrusion Prevention Systems (IPS)
➢ recent addition to security products which
● inline net/host-based IDS that can block traffic
● functional addition to firewall that adds IDS capabilities
➢ can block traffic like a firewall
➢ using IDS algorithms
➢ may be network or host based

Host-Based IPS
➢ identifies attacks using both:
● signature techniques
• malicious application packets
● anomaly detection techniques
• behavior patterns that indicate malware
➢ can be tailored to the specific platform
● e.g. general purpose, web/database server specific
➢ can also sandbox applets to monitor behavior
➢ may give desktop file, registry, I/O protection

Network-Based IPS
➢ inline NIDS that can discard packets or terminate TCP connections
➢ uses signature and anomaly detection
➢ may provide flow data protection
● monitoring full application flow content
➢ can identify malicious packets using:
● pattern matching, stateful matching, protocol anomaly, traffic anomaly, statistical anomaly

Unified Threat Management Products

6. Buffer Overflow

➢ a very common attack mechanism
● from 1988 Morris Worm to Code Red, Slammer, Sasser and many others
➢ prevention techniques known
➢ still of major concern due to
● legacy of widely deployed buggy code
● continued careless programming techniques

Buffer Overflow Basics
➢ caused by programming error
➢ allows more data to be stored than capacity available in a fixed sized buffer
● buffer can be on stack, heap, global data
➢ overwriting adjacent memory locations
● corruption of program data
● unexpected transfer of control
● memory access violation
● execution of code chosen by attacker

Memory is a flat sequence of bytes. That's it. Each byte is identified by an address.
Via memory protection, areas of memory can be marked as readable, writable, executable.

INTERPRETATIONS OF MEMORY
Types do not exist in memory. They are just abstractions that define how a certain range of bytes is interpreted.
Example: integers (and pointers) are little-endian on x86.
67 45 23 01 <-> 0x01234567
Example: C arrays are a contiguous sequence of elements.

Buffer Overflow Attacks
➢ to exploit a buffer overflow an attacker
● must identify a buffer overflow vulnerability in some program
• inspection, tracing execution, fuzzing tools
● understand how buffer is stored in memory and determine potential for corruption

A Little Programming Language History
➢ at machine level all data an array of bytes
● interpretation depends on instructions used
➢ modern high-level languages have a strong notion of type and valid operations
● not vulnerable to buffer overflows
● does incur overhead, some limits on use
➢ C and related languages have high-level control structures, but allow direct access to memory
● hence are vulnerable to buffer overflow
● have a large legacy of widely used, unsafe, and hence vulnerable code

Stack Buffer Overflow
➢ occurs when buffer is located on stack
● used by Morris Worm
● “Smashing the Stack” paper popularized it
➢ have local variables below saved frame pointer and return address
● hence overflow of a local buffer can potentially overwrite these key control items
➢ attacker overwrites return address with address of desired code
● program, system library or loaded in buffer, ROP

The x86 stack is a fundamental component of the x86 architecture used in most modern computer systems. It follows a Last In, First Out (LIFO) structure, meaning that the last data pushed onto the stack is the first to be popped off. Here's a brief explanation of how the x86 stack is structured:

1. **Stack Pointer (ESP):** The stack is managed by a special register known as the Stack Pointer (ESP). It holds the memory address of the top of the stack.
2. **Push and Pop Operations:** The x86 architecture provides instructions for pushing data onto the stack (`PUSH`) and popping data off the stack (`POP`). These operations adjust the stack pointer accordingly.
3. **Memory Growth Direction:** The x86 stack typically grows downward in memory. This means that when items are pushed onto the stack, the stack pointer decreases, and when items are popped, the stack pointer increases.
4. **Stack Frames:** The stack is often organized into frames, especially in the context of function calls. When a function is called, a new stack frame is created, including space for local variables, return addresses, and other information.
5. **Function Prologue and Epilogue:** In x86 assembly, a function prologue is the code at the beginning of a function that sets up the stack frame, and an epilogue is the code at the end that tears down the stack frame. This involves adjusting the stack pointer appropriately.
6. **Local Variables and Parameters:** Local variables and function parameters are often stored on the stack within the current function's stack frame.

A **stack overflow** occurs when a computer program's call stack, a region of memory used for managing function calls and local variables, exceeds its allocated size. In simpler terms, when a program uses more space in the stack than is available, it overflows, leading to unpredictable behavior and potential crashes. This often happens due to a recursive function or a function with insufficient exit conditions, causing an endless loop that consumes the stack space until it's exhausted. Stack overflow is a common programming issue that developers need to address to ensure the stability and reliability of their software.

Shellcode
➢ code supplied by attacker
● often saved in buffer being overflowed
● traditionally transferred control to a shell
➢ machine code
● specific to processor and operating system
● traditionally needed good assembly language skills to create
● more recently have automated sites/tools

Shellcode Development
➢ shellcode must
● marshall argument for execve() and call it
● include all code to invoke system function
● be position-independent
● not contain NULLs (C string terminator)

More Stack Overflow Variants
➢ target program can be:
● a trusted system utility
● network service daemon
● commonly used library code, e.g. image
➢ shellcode functions
● spawn shell
● create listener to launch shell on connect
● create reverse connection to attacker
● flush firewall rules
● break out of chroot environment

Buffer Overflow Defenses
➢ buffer overflows are widely exploited
➢ large amount of vulnerable code in use
● despite cause and countermeasures known
➢ two broad defense approaches
● compile-time - harden new programs
● run-time - handle attacks on existing programs

Compile-Time Defenses:
Programming Language
➢ use a modern high-level languages with strong typing
● not vulnerable to buffer overflow
● compiler enforces range checks and permissible operations on variables
➢ do have cost in resource use
➢ and restrictions on access to hardware
● so still need some code in C-like languages

Safe Coding Techniques
➢ if using potentially unsafe languages, e.g., C
➢ programmer must explicitly write safe code
● by design with new code
● after code review of existing code, e.g., OpenBSD
➢ buffer overflow safety a subset of general safe coding techniques (Ch 12)
● allow for graceful failure
● checking have sufficient space in any buffer

Language Extension, Safe Libraries
➢ have proposals for safety extensions to C
● performance penalties

● must compile programs with special compiler
➢ have several safer standard library variants
● new functions, e.g. strlcpy()
● safer re-implementation of standard functions as a dynamic library, e.g. Libsafe, no recompile needed!

Stack Protection
➢ add function entry and exit code to check stack for signs of corruption
➢ use random canary
● e.g. Stackguard GCC extension
● check for overwrite between local variables and saved frame pointer and return address
● abort program if change found
● issues: recompilation, debugger support might be ruined
➢ or save & check safe copy of return address
● e.g. Stackshield, Return Address Defender (RAD)

Run-Time Defenses:
Non Executable Address Space
➢ use virtual memory support to make some regions of memory non-executable
● e.g. stack, heap, global data
● need h/w support in processor memory management unit (MMU)
● long existed on SPARC/Solaris systems
● recent on x86 Linux/Unix/Windows systems
➢ issues: need support for executable stack code
● need special provisions to allow execution (java runtime, nested functions, …)

Address Space Randomization
➢ manipulate location of key data structures
● stack, heap, global data
● using random shift for each process
● have large address range on modern systems means negligible impact, but very difficult to predict
➢ also randomize location of heap buffers
➢ and location of standard library functions (used for attacks)

Guard Pages
➢ place guard pages between critical regions of memory
● flagged in MMU as illegal addresses
● any access aborts process
➢ can even place between stack frames and heap buffers
● at execution time and space cost

Other Overflow Attacks
➢ have a range of other attack variants
● stack overflow variants
● heap overflow
● global data overflow

● format string overflow
● integer overflow
➢ more likely to be discovered in future
➢ some cannot be prevented except by coding to prevent originally

Replacement Stack Frame
➢ stack overflow variant that just rewrites buffer and saved frame pointer
● so return occurs but to dummy frame
● return of calling function controlled by attacker
● used when have limited buffer overflow
➢ limitations
● must know exact address of buffer
● calling function executes with dummy frame

Return to System Call
➢ stack overflow variant that replaces return address with standard library function
● response to non-executable stack defenses
● attacker constructs suitable parameters on stack above return address
● function returns and library function executes
• e.g. system(“shell commands”)
● attacker may need exact buffer address
● can even chain two library calls

Heap Overflow
➢ also attack buffer located in heap
● typically located above program code
● memory requested by programs to use in dynamic data structures, e.g. linked lists
➢ no return address
● hence no easy transfer of control
● may have function pointers can exploit
● or manipulate management data structures (images processing)
➢ defenses: non executable or random heap

Global Data Overflow
➢ can attack buffer located in global data
● may be located above program code
● if has function pointer and vulnerable buffer
● or adjacent process management tables
● aim to overwrite function pointer later called
➢ defenses: non executable or random global data region, move function pointers, guard pages

7. Operating System Security

OS Security
➢As a hardening process that includes:
● planning installation, configuration, update, and maintenance.
● for OS and key applications
➢For OS in general / Linux / Windows
➢Small number of basic hardening measure
● Can prevent a large proportion of attacks seen in recent years
•Australian report: implementing just the top four of the “top 35 Mitigation strategies” would have prevented over 70% of the targeted cyber intrusions investigated in 2009
•1) patch OS
•2) patch 3rd party apps
•3) restrict admin privileges to users who need them
•4) white-list approved applications
➢Build/deploy system should be planned process to counter threats
● Assess risks and plan the system deployment
● Secure underlying OS and then apps
● Ensure critical content is secured
● Ensure using network protection mech.
● Ensure using processes to maintain security

OS Security: planning
➢Purpose of the system, apps and services
➢Categories of users, privileges
➢How users are authenticated
➢How stored info are managed
➢What access the system has to information in other hosts
➢Who will administer the system (and how, e.g. remotely?)
➢Additional security measures required
● Firewalls, antivirus, logging

OS Hardening
➢ Install and patch the OS
➢ Harden and configure OS such that:
● Remove unnecessary services/apps/protocols
● Configure users/groups/permissions
● Configure resource controls
➢ Install/configure additional security control:
● Antivirus, host-based firewall, IDS, if needed
➢ Test security
● to ensure steps taken address needs

OS Security Maintenance

➢ Monitoring/analyzing logging information
➢ Regular backups
➢ Recovery from security compromises
➢ Regularly testing system security
➢ Using appropriate sw to patch/update critical software

Linux Security Model
➢Linux's traditional security model is:
● people or processes with "root" privileges can do anything
● other accounts can do much less
➢hence attacker's want to get root privileges
➢can run robust, secure Linux systems
➢crux of problem is:
● use of **Discretionary Access Controls** (DAC)

Patch Management
➢installed server applications must be:
● configured securely
● kept up to date with security patches
➢have tools to automatically download and install security updates
● e.g. up2date, YaST, apt-get, pacman…
● note: should not run automatic updates on change-controlled systems without testing

Network Access Controls
➢network a key attack vector to secure
➢TCP wrappers a key tool to check access
● originally tcpd inetd wrapper daemon
● before allowing connection to service checks (in order)
    •if requesting host explicitly in hosts.allow is ok
    •if requesting host explicitly in hosts.deny is blocked
    •if not in either is ok
● checks on service, source IP, username
● now often directly part of apps using lib wrappers
➢ also have the very powerful netfilter Linux kernel native firewall mechanism
● and iptables front end
➢ as useful on firewalls, servers, desktops
➢ do have automated rule generators
➢ typically for "personal" firewall use will:
● allow incoming requests to specified services
● block all other inbound service requests
● allow all outbound (locally-originating) requests
➢ if need greater security, manually config

File System Security
➢in Linux everything as a file
● e.g. memory, device-drivers, named pipes, and other system resources
● hence why filesystem security is so important
➢I/O to devices is via a "special" file

● e.g. /dev/cdrom
➢have other special files like named pipes
● a conduit between processes / programs

User Management
➢guiding principles in user-account security:
● need care setting file / directory permissions
● use groups to differentiate between roles
● use extreme care in granting / using root privs
➢commands: chmod, useradd/mod/del, groupadd/mod/del, passwd, chage
➢info in files /etc/passwd & /etc/group
➢manage user's group memberships
➢set appropriate password ages

Running As Unprivileged User/Group
➢every process "runs as" some user
➢extremely important this user is not root
● since any bug can compromise entire system
➢may need root privileges, e.g. bind port
● have root parent perform privileged function but main service from unprivileged child
➢user/group used should be dedicated
● easier to identify source of log messages

Logging
➢effective logging a key resource
➢Linux logs using **syslogd** or **Syslog-NG**
● receive log data from a variety of sources
● sorts by facility (category) and severity
● writes log messages to local/remote log files
➢Syslog-NG preferable because it has:
● variety of log-data sources / destinations
● much more flexible "rules engine" to configure
● can log via TCP which can be encrypted
➢should check and customized defaults

Log Management
➢balance number of log files used
● size of few to finding info in many
➢manage size of log files
● must rotate log files and delete old copies
● typically use logrotate utility run by cron
● to manage both system and application logs
➢must also configure application logging

Application Security
➢this is a large topic
➢many security features are implemented in similar ways across different applications
➢will review issues such as:

● running as unprivileged user/group
● running in chroot jail
● modularity
● (encryption)
● (logging)

Running in chroot Jail
➢chroot confines a process to a subset of /
● maps a virtual "/" to some other directory
● useful if have a daemon that should only access a portion of the file system, e.g. FTP
● directories outside the chroot jail aren't visible or reachable at all
➢contains effects of compromised daemon
➢complex to configure and troubleshoot
● must mirror portions of system in chroot jail

Modularity
➢applications running as a single, large, multipurpose process can be:
● more difficult to run as an unprivileged user
● harder to locate / fix security bugs in source
● harder to disable unnecessary functionality
➢hence modularity a highly prized feature
● providing a much smaller attack surface

Windows Security Architecture
➢Security Reference Monitor (SRM)
● a kernel-mode component that performs access checks, generates audit log entries, and manipulates user rights (privileges)
➢Local Security Authority (LSA)
● responsible for enforcing local security policy
➢Security Account Manager (SAM)
● a database that stores user accounts and local users and groups security information
● local logins perform lookup against SAM DB
➢Active Directory (AD)
● Microsoft's LDAP directory
● all Windows clients can use AD to perform security operations including account logon
● authenticate using AD when the user logs on using a domain rather than local account
● user's credential information is sent securely across the network to be verified by AD
➢WinLogon (local) and NetLogon (net) handle login requests

Windows Privileges
➢are systemwide permissions assigned to user accounts
● e.g. backup computer, or change system time
➢some are deemed "dangerous" such as:
● act as part of operating system privilege
● debug programs privilege
● backup files and directories privilege
➢others are deemed "benign" such as
● bypass traverse checking privilege

Windows System Hardening
➢ process of shoring up defenses, reducing exposed functionality, disabling features
● known as attack surface reduction
● use 80/20 rule on features
● not always achievable
e.g. requiring RPC authentication in XP SP2
e.g. strip mobile code support on servers
➢ servers easier to harden:
1. are used for very specific and controlled purposes
2. perceive server users are administrators with better computer configuration skills than typical users

Stripping Privileges
➢strip privileges from an account soon after an application starts
● e.g. Index server process runs as system to access all disk volumes
● but then sheds any unneeded privileges as soon as possible
● using AdjustTokenPrivileges
➢Windows Vista can define privileges required by a service
● using ChangeServiceConfig2

8. Access Control

Access Control
➢ “The prevention of unauthorized use of a resource, including the prevention of use of a resource in an unauthorized manner“
➢ central element of computer security
➢ assume have users and groups
● authenticate to system
● assigned access rights to certain resources on system

Access Control Principles:
User → authentication function → access control function → system resources
|
v
Security administrator <— Authorization database

Access Control Policies: discretionary access control policy, mandatory access control policy, role-based access control policy

Access Control Requirements
➢reliable input
➢fine and coarse specifications
➢least privilege
➢separation of duty
➢open and closed policies
➢policy combinations, conflict resolution
➢administrative policies

Access Control Elements
➢subject - entity that can access objects
● a process representing user/application
● often have 3 classes: owner, group, world
➢object - access controlled resource
● e.g. files, directories, records, programs etc
● number/type depend on environment
➢access right - way in which subject accesses an object
● e.g. read, write, execute, delete, create, search

Discretionary Access Control
➢often provided using an access matrix
● lists subjects (users) in one dimension (rows)
● lists objects (files) in the other dimension (columns)
● each entry specifies access rights of the specified subject to that object
➢access matrix is often sparse
➢**can be decomposed by either row or column**

Access Control Structures

		OBJECTS			
		File 1	File 2	File 3	File 4
SUBJECTS	User A	Own Read Write		Own Read Write	
	User B	Read Write	Own Read Write	Write	Read
	User C	Read Write	Read		Own Read Write

(a) Access matrix

Access Control Model

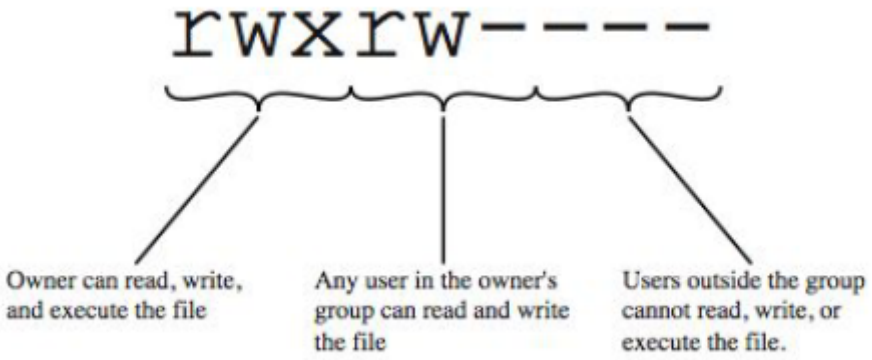
Access Control Function

Protection Domains
➢set of objects with associated access rights
➢in access matrix view, each row defines a protection domain
● but not necessarily just a user
● may be a limited subset of user's rights
● applied to a more restricted process
➢may be static or dynamic

UNIX File Concepts
➢UNIX files administered using inodes
● control structure with key info on file

•attributes, permissions of a single file
● may have several names for same inode
● have inode table / list for all files on a disk
•copied to memory when disk mounted
➢directories form a hierarchical tree
● may contain files or other directories
● are a file of names and inode numbers

UNIX File Access Control
➢ “set user ID”(SetUID) or “set group ID”(SetGID)
● system temporarily uses rights of the file owner / group in addition to the real user’s rights when making access control decisions
● enables privileged programs to access files / resources not generally accessible
➢ sticky bit: on directory limits rename/move/delete to owner
➢ superuser: is exempt from usual access control restrictions



Owner, users in owner’s group, users outside owner’s group

UNIX Access Control Lists
➢modern UNIX systems support ACLs
➢can specify any number of additional users / groups and associated rwx (read, write, execute) permissions
➢ACLs are optional extensions to std perms
➢group perms also set max ACL perms
➢when access is required
● select most appropriate ACL
•owner, named users, owning / named groups, others
● check if have sufficient permissions for access

Role-Based Access Control

The NIST RBAC Model refers to the Role-Based Access Control Model developed by the National Institute of Standards and Technology (NIST). RBAC, or Role-Based Access Control, is a security model that restricts system access to authorized users based on their assigned roles and responsibilities within an organization.

In the NIST RBAC Model, access permissions are tied to roles, and users are assigned to these roles based on their job functions or responsibilities. This approach simplifies access management and enhances security by ensuring that users have the necessary permissions to perform their tasks, without granting unnecessary access.

9. Malicious Software

Malicious Software
➢programs exploiting system vulnerabilities
➢known as malicious software or malware
● program fragments that need a host program
• e.g. viruses, logic bombs, and backdoors
● independent self-contained programs
• e.g. worms, bots
● replicating or not
➢sophisticated threat to computer systems

Malware Terminology
➢ Virus
➢ Worm
➢ Logic bomb
➢ Trojan horse
➢ Backdoor (trapdoor)
➢ Mobile code
➢ Auto-rooter Kit (virus generator)
➢ Spammer and Flooder programs
➢ Keyloggers
➢ Rootkit
➢ Zombie, bot

Malicious Software (other classifications)
➢Propagation
● Infected content – e.g., viruses
● Vulnerability exploit – e.g., worms
● Social engineering – e.g., spam and trojans
➢Payload action
● System corruption
● Attack agent – e.g., bots
● Information theft – e.g., keyloggers and spyware
● Stealthing – e.g., backdoors and rootkits

Viruses
➢ piece of software that infects programs
● modifying them to include a copy of the virus
● so it executes secretly when host program is run
➢ specific to operating system and hardware
● taking advantage of their details and weaknesses

➢ a typical virus goes through phases of:
● dormant
● propagation
● triggering
● execution

Virus Structure
➢components:
● infection mechanism - enables replication
● trigger - event that makes payload activate
● payload - what it does, malicious or benign
➢prepended / postpended / embedded
➢when infected program invoked, executes virus code then original program code
➢can block initial infection (difficult)
➢or propagation (with access controls)

Compression Virus
A "compression virus" typically refers to a type of computer virus that uses compression techniques to infect and conceal its presence in files or data. Compression is the process of reducing the size of files by encoding them in a more efficient way. In the context of a compression virus, the malicious code is often embedded within compressed files or data. The virus infects a file by embedding its code within the file's data, then to avoid detection and make the infection less noticeable, the virus may use compression techniques to reduce the size of the infected file. Compression makes it harder for traditional antivirus programs to detect the presence of malicious code. Once the infected file is executed or opened, the virus activates its payload. This payload could include various malicious actions, such as further spreading the infection, compromising system security, or damaging files.

Virus Classification
➢By target
● boot sector
● file infector
● macro virus
➢By concealing strategy
● encrypted virus
● stealth virus
● polymorphic virus
● metamorphic virus

Macro Virus
➢became very common in mid-1990s since
● platform independent
● infect documents
● easily spread
➢exploit macro capability of office apps
● executable program embedded in office doc
● often a form of Basic
➢more recent releases include protection

➢recognized by many anti-virus programs

E-Mail Viruses/Worms
➢more recent development
➢e.g. Melissa
● exploits MS Word macro in attached doc
● if attachment opened, macro activates
● sends email to all on users address list
● and does local damage
➢then saw versions triggered reading email
➢hence much faster propagation

Malware Countermeasures
➢prevention - ideal solution but difficult
● Policy / awareness / vulnerability mitigation
➢realistically need:
● detection
● identification
● removal
➢if detect but can't identify or remove, must discard and replace infected program
➢Where to detect: host / perimeter / distributed

Anti-Virus Evolution
➢virus & antivirus tech have both evolved
➢early viruses simple code, easily removed
➢as become more complex, so must the countermeasures
➢generations
● first - signature scanners
● second - heuristics
● third - identify actions
● fourth - combination packages

Generic Decryption
➢runs executable files through GD scanner:
● CPU emulator to interpret instructions
● virus scanner to check known virus signatures
● emulation control module to manage process
➢lets virus decrypt itself in interpreter
➢periodically scan for virus signatures
➢issue is long to interpret and scan
● tradeoff chance of detection vs time delay

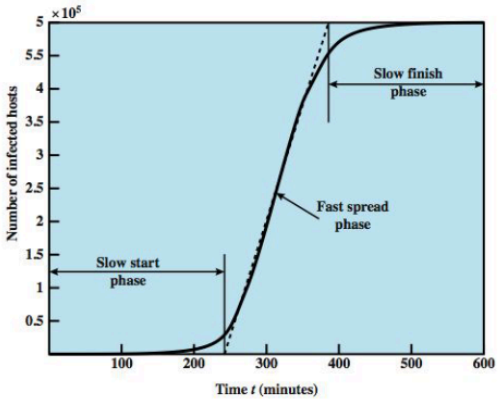
Digital Immune System
The term "Digital Immune System" refers to a concept analogous to the biological immune system but applied to the digital or cyber realm. It involves a set of technologies, processes, and security measures designed to detect, prevent, respond to, and recover from cyber threats and attacks on digital systems.

Behavior-Blocking Software
Behavior-blocking software refers to a type of security application designed to identify and prevent malicious activities on a computer or network by analyzing the behavior of software and applications. Instead of relying on known signatures of malware, behavior-blocking software focuses on monitoring the actions and behavior of programs in real-time to detect and thwart potential threats

Worms
➢ replicating program that propagates over net
● using email, remote exec, remote login
➢ has phases like a virus:
● dormant, propagation, triggering, execution
● propagation phase: searches for other systems, connects to it, copies self to it and runs
➢ may disguise itself as a system process
➢ concept seen in Brunner's "Shockwave Rider"
➢ implemented by Xerox Palo Alto labs in 1980's

Morris Worm
➢one of best know worms
➢released by Robert Morris in 1988
➢various attacks on UNIX systems
● cracking password file to use login/password to logon to other systems
● exploiting a bug in the finger protocol
● exploiting a bug in sendmail
➢if succeed have remote shell access
● sent bootstrap program to copy worm over

Worm Propagation Model



Recent Worm Attacks
➢ Code Red
● July 2001 exploiting MS IIS bug
● probes random IP address, does DDoS attack
● consumes significant net capacity when active
➢ Code Red II variant includes backdoor
➢ SQL Slammer

● early 2003, attacks MS SQL Server
● compact and very rapid spread
➢ Mydoom
● mass-mailing email worm that appeared in 2004
● installed remote access backdoor in infected systems
➢ Stuxnet – target (Iranian/nuclear?) industrial control system

Worm Technology
➢multiplatform
➢multi-exploit
➢ultrafast spreading
➢polymorphic
➢metamorphic
➢transport vehicles
➢zero-day exploit

Worm Countermeasures
➢overlaps with anti-virus techniques
➢once worm on system A/V can detect
➢worms also cause significant net activity
➢worm defense approaches include:
● signature-based worm scan filtering
● filter-based worm containment
● payload-classification-based worm containment
● threshold random walk scan detection
● rate limiting and rate halting

Proactive Worm Containment
Proactive worm containment refers to the preemptive measures and strategies implemented to detect, control, and limit the spread of computer worms before they can cause widespread damage. Worms are self-replicating malware that can propagate across networks and systems without user intervention. Proactive containment aims to curb the impact of worm infections by identifying and neutralizing them early in their lifecycle.

Network Based Worm Defense
Network-Based Worm Defense refers to a set of security measures and strategies implemented at the network level to detect, prevent, and mitigate the spread of computer worms. Worms are a type of malware that can self-replicate and spread across networks, often causing disruptions and compromising the security of systems. Network-based defense mechanisms are specifically designed to identify and counteract worm threats before they can proliferate within an organization's network.

Bots
➢program taking over other computers
➢to launch hard to trace attacks
➢if coordinated form a botnet
➢characteristics:
● remote control facility

•via IRC/HTTP etc
● spreading mechanism
•attack software, vulnerability, scanning strategy
➢various counter-measures applicable

Rootkits
➢ set of programs installed for admin access
➢ malicious and stealthy changes to host O/S
➢ may hide its existence
● subverting report mechanisms on processes, files, registry entries etc
➢ may be:
● persistent or memory-based
● user or kernel mode
➢ installed by user via trojan or intruder on system
➢ range of countermeasures needed

Rootkit System Table Mods

Rootkit System Table Mods refer to unauthorized modifications made by a rootkit to the system tables within an operating system. A rootkit is a type of malicious software that is designed to gain unauthorized access to a computer system and evade detection. System tables are critical components of an operating system that store essential information about the system's configuration, processes, and resources.

When a rootkit successfully infiltrates a system, it often seeks to maintain a persistent and stealthy presence. One way it accomplishes this is by modifying system tables.

10. Software Security

Software Security
➢ many vulnerabilities result from poor programming practices
● cf. Open Web Application Security Top Ten include 5 software related flaws
➢ often from insufficient checking / validation of program input (e.g., buffer overflow)
➢ awareness of issues is critical

Software Quality vs Security
➢ software quality and reliability (in general)
● accidental failure of program
● from theoretically random unanticipated input
● improve using structured design and testing
● not how many bugs, but how often triggered
➢ software security is related
● but attacker chooses input distribution, specifically targeting buggy code to exploit
● triggered by often very unlikely inputs
● which common tests don’t identify

Defensive Programming

➢ a form of defensive design to ensure continued function of software despite unforeseen usage
➢ requires attention to all aspects of program execution, environment, data processed
➢ also called secure programming
➢ assume nothing, check all potential errors
➢ rather than just focusing on solving task
➢ must validate all assumptions

Abstract Program Model
Correctly anticipating, checking and handling all possible errors will certainly increase the amount of code (...time/money) needed in…
=> **software security must be a design goal** (see the Internet/NDN case)

Security by Design
➢ security and reliability common design goals in most engineering disciplines
● society not tolerant of bridge/plane etc failures
➢ software development not as mature
● much higher failure levels tolerated
➢ despite having a number of software development and quality standards
● main focus is general development lifecycle
● increasingly identify security as a key goal

Handling Program Input
➢ incorrect handling a very common failing
➢ input is any source of data from outside
● data read from keyboard, file, network
● also execution environment, config data
➢ must identify all data sources
➢ and explicitly validate assumptions on size and type of values before use

Input Size & Buffer Overflow
➢ often have assumptions about buffer size
● e.g., that user input is only a line of text
● size buffer accordingly but fail to verify size
● resulting in buffer overflow
➢ testing may not identify vulnerability
● since focus on “normal, expected” inputs
➢ safe coding treats all input as dangerous
● hence must process so as to protect program

Interpretation of Input
➢ program input may be binary or text
● binary interpretation depends on encoding and is usually application specific
● text encoded in a character set e.g. ASCII
● internationalization has increased variety
● also need to validate interpretation before use
• e.g. filename, URL, email address, identifier
➢ failure to validate may result in an exploitable vulnerability

Injection Attacks
➢ flaws relating to invalid input handling which then influences program execution
● often when passed as a parameter to a helper program or other utility or subsystem
➢ most often occurs in scripting languages
● encourage reuse of other programs / modules
● often seen in web CGI scripts

SQL Injection
➢ another widely exploited injection attack
➢ when input used in SQL query to database
● similar to command injection
● SQL meta-characters are the concern
● must check and validate input for these

Cross Site Scripting Attacks
➢ attacks where input from one user is later output to another user
➢ XSS commonly seen in scripted web apps
● with script code included in output to browser
● any supported script, e.g. Javascript, ActiveX
● assumed to come from application on site
➢ XSS reflection
● malicious code supplied to site
● subsequently displayed to other users

Cross-Site Scripting (XSS) is a type of security vulnerability in web applications where an attacker injects malicious scripts into trusted websites. These scripts are then executed by unsuspecting users, often in their web browsers. XSS attacks occur when an application includes untrusted data from a user in a web page without proper validation or escaping. There are different types of XSS attacks, including stored, reflected, and DOM-based XSS. The goal of XSS attacks is typically to steal sensitive information, manipulate user sessions, or deface websites. Preventive measures include input validation, output encoding, and implementing secure coding practices.

XSS Example
➢ cf. guestbooks, wikis, blogs etc.
➢ where comment includes script code
● e.g., to collect cookie details of viewing users
➢ need to validate data supplied
● including handling various possible encodings
➢ attacks both input and output handling

Handling Input
● Validating Input Syntax
● Considering Alternate Encodings
● Validating Numeric Input
● Input Fuzzing (involves providing unexpected, random, or malformed data as input to a program, application, or system to discover vulnerabilities, bugs, or weaknesses)

Writing Safe Program Code

- Correct Use of Memory
- Race Conditions in Shared Memory
- Interacting with O/S
- System Calls and Standard Library Functions
- Safe Temporary Files