**Intro to Security**

**Three main concepts:**

* Confidentiality
  + Data confidentiality: Assures confidential information is not disclosed to unauthorized individuals
  + Privacy: Assures that individual control or influence what information may be collected and stored
* Integrity
  + Data Integrity: assures that information and programs are changed only in a specified and authorized manner
  + System integrity: Assures that a system performs its operations in unimpaired manner
* Availability: assure that systems works promptly and service is not denied to authorized users

**Security policy:**

Security Policy: A set of rules and practices that specify or regulate how a system or organization provides security services to protect sensitive and critical system resources.

**Cryptography**

* Symmetric encryption uses same key to encrypt and decrypt. Used from 1900-1950, used in war, codebook: Enigma
* Asymmetric Encryption uses public key to encrypt, secret key to decrypt. Started in 1970s by Merkle, Hellman, and Diffe in 1976 and Shamir, Rivest and Adleman in 1978.
  + Asymmetric encryption foundation of E-commerce
* Euler totient function, Setting up an RSA system involves choosing large prime numbers *p* and *q*, computing *n* = *pq* and *k* = *φ*(*n*), and finding two numbers *e* and *d* such that *ed* ≡ 1 (mod *k*). The numbers *n* and *e* (the "encryption key") are released to the public, and *d* (the "decryption key") is kept private. A message, represented by an integer *m*, where 0 < *m* < *n*, is encrypted by computing *S* = *me* (mod *n*). It is decrypted by computing *t* = *Sd* (mod *n*). Euler's Theorem can be used to show that if 0 < *t* < *n*, then *t* = *m*. The security of an RSA system would be compromised if the number *n* could be factored or if *φ*(*n*) could be computed without factoring *n*
* **Fermat's little theorem** states that if *p* is a [prime number](https://en.wikipedia.org/wiki/Prime_number), then for any [integer](https://en.wikipedia.org/wiki/Integer) *a*, the number *ap* − *a* is an integer multiple of *p*. In the notation of [modular arithmetic](https://en.wikipedia.org/wiki/Modular_arithmetic), this is expressed as
* Typical symmetric algorithms include 3DES and AES with key lengths varying between 128 and 256 bits, and a typical asymmetric algorithm is RSA with a key length between 1,024 and 4,096 bits.
* Message Authentication checks integrity and authenticity of message
* Hash Function Requirements• applied to any size data• H produces a fixed-length output.• H(x) is relatively easy to compute for any given x• one-way property • computationally infeasible to find xsuch that H(x) = h• weak collision resistance • computationally infeasible to find y≠ xsuch thaH(y) = H(x)• strong collision resistance • computationally infeasible to find any pair (x, y) such that H(x) = H(y)26
* Hash Functions• two attack approaches• cryptanalysis•11 exploit logical weakness in algorithm• brute-force attack• trial many inputs• strength proportional to size of hash code (2n/2)• SHA most widely used hash algorithm• SHA-1 gives 160-bit hash• more recent SHA-256, SHA-384, SHA-512 provide improved size and security27Hash Functions• two attack approaches• cryptanalysis• exploit logical weakness in algorithm• brute-force attack• trial many inputs• strength proportional to size of hash code (2n/2)• SHA most widely used hash algorithm• SHA-1 gives 160-bit hash• more recent SHA-256, SHA-384, SHA-512 provide improved size and securit
* A **digital envelope** is a secure electronic data container that is used to protect a message through encryption and data authentication. A **digital envelope** allows users to encrypt data with the speed of secret key encryption and the convenience and security of public key encryption.
* In cryptography, a **collision attack** on a [cryptographic hash](https://en.wikipedia.org/wiki/Cryptographic_hash) tries to find two inputs producing the same hash value, i.e. a [hash collision](https://en.wikipedia.org/wiki/Hash_collision). This is in contrast to a [preimage attack](https://en.wikipedia.org/wiki/Preimage_attack) where a specific target hash value is specified.
* **The**[**purpose of homomorphic encryption**](https://eprint.iacr.org/2015/1192.pdf)**is to allow computation on encrypted data. Thus data can remain confidential while it is processed, enabling useful tasks to be accomplished with data residing in untrusted environments. In a world of distributed computation and heterogeneous networking this is a hugely valuable capability.**
* Select primes: p=17, q=112.Compute n=pq=17×11=1873.Compute φ(n)=(p–1)(q-1)=16×10=1604.Select e:gcd(e,160)=1 and e<160♣choose e=75.Determine d: de=1 mod 160 and d<160 ♣d=23since 23×7=161=10×160+16.Publish public key KU={7,187}7.Keep secret private key KR={23,187}

**User 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 privileges5
* slide 9Storing Passwords• Instead of user password, store Hash(password)• When user enters a password, compute its hash and compare with the entry in the password file• System does not store actual passwords• Cannot go from hash to password• ... except by guessing the password• Hash function H must have some properties• Given H(password), hard to find any string X such that H(X)=H(password) -why?slide 9Storing Passwords• Instead of user password, store Hash(password)• When user enters a password, compute its hash and compare with the entry in the password file• System does not store actual passwords• Cannot go from hash to password• ... except by guessing the password• Hash function H must have some properties• Given H(password), hard to find any string X such that H(X)=H(password) -why?
* Two steps:• identification: specify identifier• verification: bind entity (person) and identifier
* A cryptographic **salt** is made up of random bits added to each password instance before its hashing. Without salt, attacker can pre-compute hashes of all common passwords once• Same hash function on all UNIX machines; identical passwords hash to identical values• One table of hash values works for all password files• With salt, attacker must compute hashes of all common passwords for each possible salt value• With 12-bit random salt, the same password can hash to 4096 different hash values

**Access Control**

* User-oriented security policy (based on ID of requestor)•Discretionary because an entity has rights to enable another entity to access a resource• General approach as used in operating systems and database management systems is that of an access matrix• Lists subjects in one dimension (rows)• Lists objects in the other dimension (columns)• Each matrix entry specifies access rights of the specified subject to that object Intrusion Detection/Prevention
* Access rights stored with objects• ACL may contain default (public) entries• If users not explicitelly listed in ACL – default rights (e.g., read only)• Elements of ACL include individual users as well as groups of users• ACLs are convinient when desired to determine which subjects have which access rights to particular resource• Not convinient for determining the access rights of a particular user • UNIX and Windows use ACLs to protect files/processesACL requires subjects to be authenticatedbefore access to aparticular object!
* ACL is files->users, Capabilities is Users->files, windows and Unix us ACL
* Traditional DAC systems define the access rights of individual users and groups of users• In many organizations (in industry), the user do not own the information for which they are allowed access• Rather, the coporation is the actual owner of system objects• Access control is often based on employee job functions (roles) rather than data ownership• E.g. roles in a hospital: doctor, nurse, pharmacists,... • RBAC is based on the roles that users assume in a corporation/organization(rather than the user’s ID)• RBAC systems asign access rights to roles • And users are assigend to different rolesRBAC23
* • Authorization management• RBAC breaks authorization task into two independent parts: one which assigns users to roles and one which assigns rights for objects to roles• User’s change more frequently than roles, easy revocation of rights• Hierarchical roles•Least privilege• Roles allow a user to sign on with the leastprivilege required for the particular task at hand• Users with powerful roles do not need to exercisethem until those privileges actually needed•Separation of duties• No single principle should be given enough privileges to misuse the system on their own• E.g. two-person operation: 1st any authorized user, 2nd any authorized user different from the 1st (example: banks)Advantages of RBAC28

Misuse detection (a.k.a. signature-based)•Anomaly detection (a.k.a. statistical-based) MisuseDetectionIntrusion Patterns activities pattern matching intrusion Can’t detect new attacksExample: if(src\_ip == dst\_ip) then“land attack”

Anomaly Detectionactivity measures0102030405060708090CPUProcessSizenormal profileabnormalprobable intrusionRelatively high false positive rate •Anomalies can just be new normal activities.•Anomalies caused by other element faults•E.g., router failure or misconfiguration, P2P misconfigurationAny problem ?

* Host-Based IDSs•Using OS auditing mechanisms– E.G., BSM on Solaris: logs all direct or indirect events generated by a user–strace for system calls made by a program (Linux)•Monitoring user activities– E.G., analyze shell commands•Problems: user dependent– Have to install IDS on all user machines !– Ineffective for large scale attacks
* Network IDSs•Deploying sensors at strategic locations– E.G., Packet sniffing via tcpdump at routers•Inspecting network traffic – Watch for violations of protocols and unusual connection patterns•Monitoring user activities– Look into the data portions of the packets for malicious code•May be easily defeated by encryption– Data portions and some header information can be encrypted– The decryption engine may still be there, especially for exploit

**Firmware Security**

* Software is often designed for OS, while Firmware is for a piece of hardware(like a single chip).•Software is often stored in user accessible memory while Firmware is located in an inaccessible storage embedded in the hardware.•Software can be replaced without much hassle while replacing firmware is often difficult.•Software can be very big while firmware are usually very small.
* Extreme persistence•Stealth•Bypass software (OS or VMM) based security•Unfettered access to hardware•OS independence•Making the system unbootable (bricking)

**Malware/Anit-Malware**

* Malware registers its own function (i.e hook) into the target location (hook site)
* Later, data in the hook site is loaded into the EIP, and the execution is redirected into malware’s own function
* Worm A worm is self-replicating software designed to spread through the networkνTypically, exploit security flaws in widely used servicesνCan cause enormous damage ωLaunch DDOS attacks, install bot networks ωAccess sensitive informationωCause confusion by corrupting the sensitive informationWorm vs VirusνA virus is code embedded in a file or programνWorms are self-contained and may spread autonomously
* Morris worm• One of the first computer worms distributed via the Internet• Released by Robert Morris in 1988• Affected 6,000 computers; cost $10-$100 M• 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
* State of worm technology• Multiplatform: not limited to Windows• Multi-exploit: Web servers, emails, file sharing ...• Ultrafast spreading: do a scan to find vulnerable hosts• Polymorphic: each copy has a new code• Metamorphic: change appearance/behavior• Transport vehicles (e.g., for DDoS)• Zero-day exploit of unknown vulnerability (to achieve max surprise/distribution)
* **Ransomware** is a type of malicious software, or malware, designed to deny access to a computer system or data until a ransom is paid.
* Detectors use technology from 15 years ago: signature-based detection.• Even the most effective malware detectors fail to detect more than 30% of malware seen in the wild.• Traditional malware detectors are based on syntactic signatures• Malware producers can easily generate malware variants capable of evading existing signatures
* Behavior-based malware detection• Detect high-level actions that financially motivate malware development & distribution• keystroke logging• data leaking• proxying• program download and execute• Monitor execution of the program using an emulator• Lowest level events in behavior specifications are system calls• Malicious behaviors are described as sequences of essential actions

**Buffer Overflow Vulnerability**