Some Fundamental Concepts

- **Authentication** (who are you)
  - identifying principals (users / programs)
- **Authorization** (what are you allowed to do)
  - determining what access users and programs have to things
- **Auditing** (what happened)
  - record what users and programs are doing for later analysis / prosecution

The Local Case: Unix Password File

- Encrypt passwords with passwords

David’s password, “allison,” is encrypted using itself as the key and stored in that form.

- Password supplied by user is encrypted with itself as key, and result compared to stored result.
- “No problem if someone steals the file”

The Dictionary Attack

- Encrypt many (all) possible password strings offline, and store results in a dictionary
  - I may not be able to invert any particular password, but the odds are very high I can invert one or more

- 26 letters used, 7 letters long
  - 8 billion passwords (33 bits)
  - Generating 100,000/second requires 22 hours

- But most people’s passwords are not random sequences of letters!
  - girlfriend/boyfriend/spouse/dog’s name/words in the dictionary

Dictionary attacks have traditionally been incredibly easy

Making it harder

- Using symbols and numbers and longer passwords
  - 95 characters, 14 characters long
  - $10^{7}$ passwords = 91 bits
  - Checking 100,000/second breaks in $10^{14}$ years

- Require frequent changing of passwords
  - guards against loaning it out, writing it down, etc.

Do longer passwords work?

- People can’t remember 14-character strings of random characters
- People write down difficult passwords
- People give out passwords to strangers
- Passwords can show up on disk
- If you are forced to change your password periodically, you probably choose an even dumber one
  - “feb04” “mar04” “apr04”
- How do we handle this in CSE?
Countermeasure to the Dictionary Attack:
Salt
• Unix (1979): salted passwords
  – The salt is just a random number from a large space
  Encryption is computed after affixing a number to the password. Thwarts pre-computed dictionary attacks

Okay, are we done? Problem solved?

Attack Models
• Besides the problems already mentioned that obviously remain (people give out their passwords / write them down / key loggers / ...), there may be other clever attacks that we haven’t thought of
• Attack Model: when reasoning about the security of a mechanism, we need typically need to carefully describe what kinds of attacks we’re thinking of
  – helps us reason about what vulnerabilities still remain

Example 1: Login spoofers
• Login spoofers are a specialized class of Trojan horses
  – Attacker runs a program that presents a screen identical to the login screen and walks away from the machine
  – Victim types password and gets a message saying “password incorrect, try again”
• 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

Example 2: Cool password attack
• VMS (early 80’s) password checking flaw
  – password checking algorithm:
    for (I=0; I<password.length(); I++) {
      if password[I] == supplied_password[I]
        return false;
    }
  return true;
  – can you see the problem?
  • hint: think about virtual memory…
  • another hint: think about page faults…
  • final hint: who controls where in memory supplied_password lives?

Distributed Authentication (Single Domain)

Aside on Encryption
• Encryption: takes a key and plaintext and creates ciphertext: \( \{M\}_k = C \)
• Decryption: takes ciphertext and a key and recovers plaintext: \( \{C\}_k = M \)
• Symmetric algorithms (aka secret-key aka shared secret algorithms):
  – \( k_1 = k_2 \) (or can get \( k_2 \) from \( k_1 \))
• Public-Key Algorithms
  – decryption key \( k_2 \) cannot be calculated from encryption key \( k_1 \)
  – encryption key can be made public!
  – encryption key = “public key”, decryption key = “private key”
• Computational requirements:
  – Deriving \( M \) from \( \{M\}_k \) is “really hard”
  – Computing \( \{M\}_k \) and \( \{C\}_k \) is efficient
Kerberos

Alice

Request for TGT

Client Key DB

Authentication Server

Ticket Granting Service

Print Server (B)

At this point Alice and the server have a shared secret:

\[ \text{Data}K_{st} \]

Trust Relationships

- Both Alice and the server must trust the Kerberos servers ("trusted third party")
- This architecture is essentially what Microsoft passport is:

Distributed Authentication at World Scale

- Bill Gates wants to login to his Citibank account to move $10 from savings to checking
- Both Bill and Citibank are worried:
  - Citibank:
    - How do I know that I’m talking with Bill?
    - Does Bill have $10 in his savings account?
    - …
  - Bill:
    - How do I know that I’m talking with Citibank?

Authentication Solutions

- Citibank authenticating Bill
  - This is just a client accessing a server. Citibank can use shared secrets.
  - Bill has to use some secret communicated out-of-band (e.g., ATM PIN number) to create a shared secret for online access.
- Bill authenticating Citibank
  - Could shared secret work for the bank to authenticate itself to the client?
  - …
  - In the end, we rely on a trusted third party (just like Kerberos, but implemented differently)
Why not this?

Citibank

Bill, {N, Login}KBill

{N, KS, Password?}KCiti

{N, Password}KS

Public Key Encryption

• Key pairs, $K_{Public} / K_{Private}$
  
  $\{ M \} K_{Public} \equiv \{ M \} K_{Private} \equiv M$

  • Each key is the decryption key for the other used as an encryption key

  • It is computationally infeasible to deduce $K_{Private}$ from $K_{Public}$
  
  • You can distribute $K_{Public}$ freely

• $\{ M \} K_{Public}$ can be decrypted only by the holder of the private key

• $\{ M \} K_{Private}$ can be created only by the holder of the private key
  
  – “Signing”

Authentication by Certificate: Basic Idea

Citibank

Documents

Digital Certificate

w/ $K_{CitiPublic}$

Get login

Password?

Password

Client

TTP on $K_{TTPPublic}$

• Much more is need for this to actually work
  
  • E.g., what keeps yegg.com from copying the certificate?

  • Why not have the client contact the TTP directly to obtain the certificate at the outset?

  • Why might you “want” to contact the TTP in any case?

Client/Server Communication: ssl (tls)

Citibank

Hello, N$_{Client}$

Hello, N$_{Server}$

{Pre-master}K$_{ServerPublic}$

TTP on $K_{TTPPublic}$

{Finished}K$_{Session}$

Notes:

1. Master/session key determined independently by both client and server as:

   $F(N_{client}, N_{server}, \text{Pre-master})$

2. I’ve taken some liberties to simplify the explanation…

   (cf. CSE 461)

The Larger Security Problem

• Integrity
  
  My data should be protected against modification by malicious parties
  
  – “Modification” includes deletion

• Privacy
  
  My data should not be disclosed without my consent

• Both issues have become much more complicated in the last decade
  
  – Attackers exploit bugs/weaknesses accessible through the net

Spyware

• Software that is installed that collects information and reports it to third party
  
  – key logger, adware, browser hijacker, …

• Installed one of two ways
  
  – piggybacked on software you choose to download

  – “drive-by” download

  – your web browser has vulnerabilities

  – web server can exploit by sending you bad web content

• Estimates
  
  – majority (50-90%) of Internet-connected PCs have it

  – 1 in 20 executables on the Web have it

  – about 0.5% of Web pages attack you with drive-by-downloads
kingsofchaos.com
• A benign web site for an online game
  – earns revenue from ad networks by showing banners
  – but, it relinquishes control of the ad content

banner ad from
adworldnetwork.com
(a legitimate ad network)
inline javascript loads
HTML from ad provider

Incident
• kingsofchaos.com was given this "ad content"
  <script type="text/javascript">document.write(
"<p>
of of of of ...etc.
</p>

• This "ad" ultimately:
  – bombarded the user with pop-up ads
  – hijacked the user’s homepage
  – exploited an IE vulnerability to install spyware

What’s going on?
• The advertiser was an ex-email-spammer
• His goal:
  – force users to see ads from his servers
  – draw revenue from ad “affiliate programs”
    • Apparently earned several millions of dollars
• Why did he use spyware?
  – control PC and show ads even when not on the Web

Principle of Least Privilege
• Figure out exactly which capabilities a program
  needs to run, and grant it only those
  – start out by granting none
  – run program, and see where it breaks
  – add new privileges as needed.
• Unix: concept of root is not a good example of this
  – some programs need root just to get a small privilege
    • e.g., FTP daemon requires root
      – to listen on network port < 1024
    – to change between user identities after authentication
  – but root also lets you read any file in filesystem

Principle of Complete Mediation
• Check every access to every object
  – in rare cases, can get away with less (caching)
    • but only if sure nothing relevant in environment has changed...and there is a lot that’s relevant!
• A TLB caches access control information
  – page table entry protection bits
  – is this a violation of the principle?

Modern security problems
• Confinement
  – How do I run code that I don’t trust?
  – e.g., RealPlayer, Flash
  – How do I restrict the data it can communicate?
  – What if trusted code has bugs?
  – e.g., Internet Explorer
• Solutions
  – Restricted contexts – let the user divide their identity
  – ActiveX – make code write identify self
  – Java – use a virtual machine that intercepts all calls
  – Binary rewriting – modify the program to force it to be safe
Restricted contexts

- Role-based access control (RBAC)
  - Add extra identity information to a process
  - e.g., both username and program name (mikesw:navigator)
  - Use both identities for access checks
  - add extra security checks at system calls that use program name
  - add extra ACLs on objects that grant/deny access to the program
  - Allows users to sub-class themselves for less-trusted programs
- chroot
- Browse in a VMWare machine

ActiveX

- All code comes with a public-key signature
- Code indicates what privileges it needs
- Web browser verifies certificate
- Once verified, code is completely trusted

ActiveX Diagram:

- Signature/Certificate
- Permissions
- Code
- Written by HackerNet
- Signed by VerifySign
- Let JavaScript call this

Java / C#

- All problems are solved by a layer of indirection
  - All code runs on a virtual machine
  - Virtual machine tracks security permissions
  - Allows fancier access control models - allows stack walking
- Interposition using language VM doesn’t work for other languages
- Virtual machines can be used with all languages
  - Run virtual machine for hardware
  - Inspect stack to determine subject for access checks

Binary rewriting

- Goal: enforce code safety by embedding checks in the code
- Solution:
  - Compute a mask of accessible addresses
  - Replace system calls with calls to special code

Original Code:

```
lw    $a0, 14($s4)   and $t6, $a0, 0x001ffe0
jal   ($s5)          lw $a0, 14($t6)
move $a0, $v0        and $t6, $a0, 0x001ffe0
jal  $printf         jal ($t6)
                   move $a0, $v0
                   jal $sti_printf
```

Rewritten Code:

```
and $t6, $s4, 0x001fff0
lw $a0, 14($t6)          and $t6, $s5, 0x001fff0
jal ($t6)                move $a0, $v0
move $a0, $v0            jal $sfi_printf
```