# Program Structure II

# More Architecture — More on Email Security

- We want to secure email
- Generally, that requires crypto, which in turn requires protecting keys
- How shall we do that?

# Standard Techniques

- Encrypt the private key with a user-typed passphrase
- Use special-purpose crypto hardware
- The latter is rarely available; we need to use the former, at least in some cases

# Where are Decryption and Signing Done?

- Gateway machine?
- End-user's machine?

# Signing at the Gateway

- Tempting target
- Hard for user to supply the key or the passphrase
- How does the gateway *know* who sent the mail?
- Best for *organizational* signatures

# Decrypting at the Gateway

- Again, how are keys supplied?
- When is decryption done?
- Is the mail stored internally in the clear?

# Signing Every Message

- Suppose we want to sign every message
- Do we prompt users for a passphrase on each email sent?
- Rather annoying — can we cache passphrases?

# (Why Sign Everything?)

- Principle?
- Prevent false attribution?
- Anti-spam?

# Caching Keys

- If we cache keys, they're exposed to bugs in the mailer
- How risky are mailers?
- (How big are they?)

# Some Mailer Sizes

| *Mailer* | *KLOC* |
|---|---|
| Thunderbird | 6000 |
| Evolution | 2500 |
| (extras) | 2200 |
| Claws-Mail | 840 |
| Pine | 530 |
| Mutt | 288 |

Numbers are *very* imprecise. All of these mailers require many libraries, especially the GUI mailers. (GTK+ is about 3,000,000 lines of code.)

# (Why are Mailers So Big?)

- Mail formats are complex
  - MIME
  - Multilingual
  - GUIs
- HTML rendering
- Other stuff bundled in (calendar, vCard, etc)
- Frequently include an editor

# Why are Mailers Insecure?

- Size—security hole rates go up as the square of the code size
- Accept untrusted input
- Plenty of room for user error

# Entrust our Keys to Mailers?

- They're big and complicated
- They interact with lots of other programs
- They have long histories of security problems
- Handing them keys doesn't sound like a great idea. . .

# Outboard Key Manager

- Should we have a separate application to handle keys?
- How big are such applications?
- Can we trust them?

# Key Managers

| *Component* | *KLOC* |
|---|---|
| GNOME Keyring | 150 |
| GNOME Keyring Manager | 97 |
| GPG | 520 |
| GPG2 | 737 |
| pinentry | 55 |

These aren't exactly tiny, either. . .

# Bug Rates

- How many bugs per 1,000 lines of code?
- Hard to measure
- Different types of software have different rates
- We can't count bugs that aren't found!
- That said...

| *Component* | *Bugs/KLOC* |
|---|---|
| Linux 2.6 Kernel | `.17` |
| Commercial code | 20–30 |

  (Is that bug rate for Linux believable?)
- But — Microsoft claims that Vista and its components have had fewer security bugs than the open source competition

# Managing the Key Manager

- The mailer still tells the key manager what to decrypt or sign
- If the mailer is buggy, it can fool the key manager
- You don't know what's *really* being signed or decrypted
- (This all applies to crypto hardware solutions, too)

# Pure Outboard Solution?

- Save inbound mail; manually decrypt it
- (Hand-carry it to an offline decryption machine?)
- Edit outbound mail separately; manually sign, then paste that into mailer buffer
- (Hand-carry it from an offline encryption and signing machine?)
- Does this work?

# It's Too Inconvenient

- Most users *won't* put up with this
- Result: very few signed messages
- Result: reluctance to receive inbound encrypted messages
- Does this give us worse security?

# What Do We Do?

- There are no perfect solutions
- How disciplined are the users?
- How important is secure email?
- Can you have separate grades of keys?
- Who is your enemy?

# Outboard Keys

- Despite the risks, outboard keys are still better
- Still simpler than the mailer
- Less risk of key theft
- Easier to add (secure) audit trail

# Windows Vista and IE

- Web browsers have also been problematic
- Historically, Internet Explorer has been bad, but it's been improving
- (IE 6 was *horrid*)
- (These days, Firefox seems to have twice as many security bugs as IE.)
- IE 7 on Vista was a lot better; its successors are better still
- Why?

# Protected Mode

- Run web browser with fewer privileges (exception: trusted sites can have full privileges)
- Compromise of the browser does not result in compromise of (most) user files
- (Plus — very rigorous development process, with a lot of emphasis on security)

# Components

- User Account Control (UAC)
- Mandatory Integrity Control (MIC)
- User Interface Privilege Isolation (UIPI)

# User Account Control

- Eliminate need to log in as Administrator
- Even Administrator can run most applications without privilege — they changed the privilege requirements for some operations
- Privilege can be raised as needed, with password entry. (Will users make that decision correctly?)
- Users have found UAC *very* annoying

# Mandatory Integrity Control

- Low-privilege processes cannot write to protected files
- Available levels: low, medium, high
- Similar to MAC

# Bell-Lapdula and MIC

- Recall how Bell-Lapadula confidentiality mechanisms could be used for integrity protection, by reversing labels
- MIC uses half of it: it's really "no write down"
- MIC does not provide confidentiality protection

# Privilege is Inherited

- The privilege level of a process is inherited by its children
- Children spawned by protected mode IE also run at Low privilege
- This blocks attacks by ActiveX, VBScript, etc.

# Virtualization

- A lot of existing code wants to write files (cache, temporary files, cookies, history, registry, etc.)
- A shim layer virtualizes these functions
- Files to be modified in Low mode are copied to the Low area; the changes are made only to the copies

# Why Virtualization?

- Legacy code and legacy design patterns
- Older programs were not intended to be sandboxed like this
- Virtualization layer makes it easy to convert

# Gaining Privilege

- Sometimes, Low processes need to do things requiring privilege
- Special *broker* processes will perform such operations on request
- Brokers ask user consent before proceeding
- Is that reliable?

# Trusting the User?

- Users can be tricked
- Many of today's dialog boxes are useless
- From a W3C glossary Wiki:

  *Dialog box: A window in which resides a button labeled "OK" and a variety of text and other content that users ignore.*

# Users Don't Like It

- Some older applications break
- These were probably insecure to begin with
- But people are used to them
- Windows 7 has cut down on the prompts — but some say that makes it less secure. Must security be annoying?

# Lack of Confidentiality Protection

- Low mode malware can still read your files
- It appears possible for Low mode applications to export data
- But — full Bell-Lapadula confidentiality control is impractical
- Cookies are a special case — prevent (some) cross-site scripting attacks

# User Interface Privilege Isolation

- Prevents Low mode processes for sending certain messages to higher-mode processes
- Blocks “shatter attack” (inject code into another process via Windows messages)
- In essence, ACL for message-passing

# What Has Microsoft Done?

- Separated Internet Explorer from Windows Explorer (i.e., restored the distinction between net and desktop)
- (In the antitrust trial in 1998, Microsoft claimed they couldn't separate the two.)
- Used OS access controls to isolate browser
- Added more access controls
- *Structural separation*

# Does it Work?

- IE7 on Vista is immune to the `.ani` file (animated cursor) attack (see `http://www.microsoft.com/technet/security/bulletin/MS07-017.mspx`)
- More precisely, the attack code couldn't escape the Low mode jail
- Human interface attacks may still be an issue
- Other delivery mechanisms for `.ani` still work

# Firefox vs. Chrome

- Chrome has a higher rate of security bugs reported than Firefox does
- (May reflect different amounts of attention)
- But—critical and high priority bug rates in Chrome are *much* lower (and falling) than in Firefox
- Is this because of the privilege separation architecture in Chrome? It still has holes, but they're not nearly as serious.
- Firefox does not use privilege separation.

# Securing a Browser

- User interface runs with normal privileges
- Retrieving and rendering pages done with low privileges
- What about separation between sites?

# Process Separation

- Firefox runs as one process
- Chrome and IE 8 use a process per tab
- ☞ Good for monitoring and controlling resource consumption
- Experimental Gazelle browser uses separate protection domains for each web site contacted
  - Protects against improper information flow between web sites
  - Matches browser's "same origin" principal
  - In other words: implement browser security semantics via OS security mechanisms

# Summary

- Structural separation helps
- It's not a panacea
- There are still challenging user interface issues
- Backwards compatibility is a problem