Design

Secure Application Development

Modules 12, 13

Konstantin Beznosov

What Do you Already Know?

- What principles of designing secure systems do you already know?
- What anti-principles do you know?

Outline

- Overview
- Principles of designing secure systems
- Principles of designing usable security

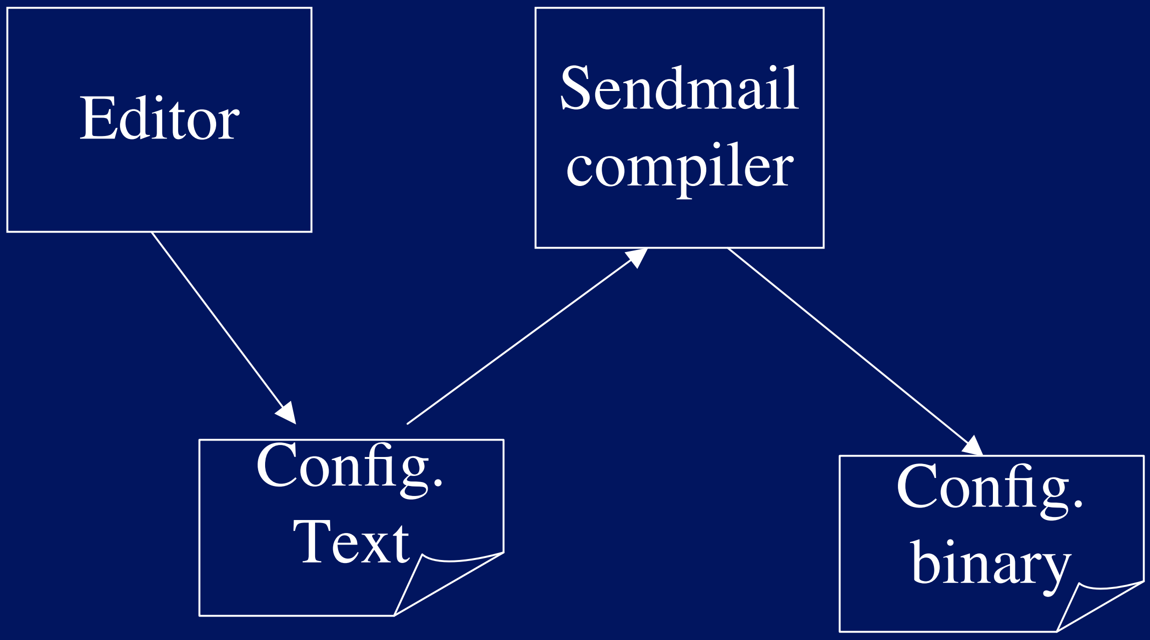
Principles of Designing Secure Systems

Design Principles Outline

- Overview
- Principles
  1. Least Privilege
  2. Fail-Safe Defaults
  3. Economy of Mechanism
  4. Complete Mediation
  5. Open Design
  6. Separation of Privilege
  7. Least Common Mechanism
  8. Psychological Acceptability
  9. Defense in Depth
  10. Question Assumptions

J. Saltzer and M. Schroeder "The Protection of Information in Computer Systems" 1975

Introductory Example: Sendmail



Overarching Goals

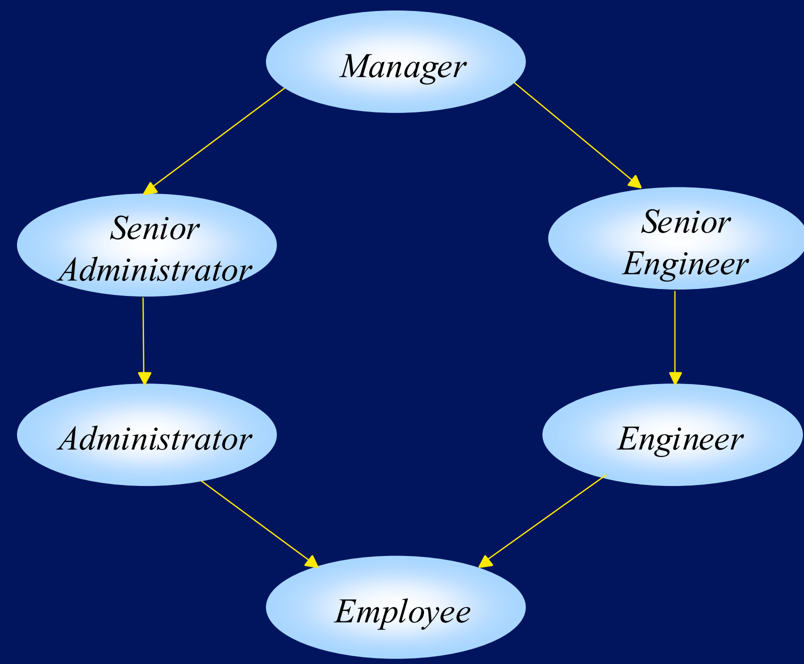
- Simplicity
  - Less to go wrong
  - Fewer possible inconsistencies
  - Easy to understand
- Restriction
  - Minimize access
    - “need to know” policy
  - Inhibit communication to minimize abuse of the channels

Example 1: Privileges in Operating Systems

- Until Windows NT, all privileges for everybody
- Separate admin (a.k.a., root) account on Windows and Unix
  - Ways to switch between accounts

Example 2: RBAC

Differentiation between assigned and activated roles



Principle 1: Least Privilege

Every program and every user of the system should operate using the least set of privileges necessary to complete the job

- Rights added as needed, discarded after use
- Limits the possible damage
- Unintentional, unwanted, or improper uses of privilege are less likely to occur
- Guides design of protection domains

Example 3: Temporary Upgrade of the Priveleges in MacOS

- sudo command on MacOS
- admin account authentication

Principle 2: Fail-Safe Defaults

Base access decisions on permission rather than exclusion.

suggested by E. Glaser in 1965

- Default action is to deny access
- If action fails, system as secure as when action began

Principle 3: Economy of Mechanism

Keep the design as simple and small as possible.

- KISS Principle
- Rationale?
- Essential for analysis
- Simpler means less can go wrong
  - And when errors occur, they are easier to understand and fix

Example 4: .rhosts mechanism abused by 1988 Internet Worm

Access to one account opened unchecked access to other accounts on different hosts

Principle 4: Complete Mediation

Every access to every object must be checked for authority.

- If permissions change after, may get unauthorized access

Example 5: Multiple reads after one check

- Process rights checked at file opening
- No checks are done at each read/write operation
- Time-of-check to time-of-use

Kerckhoff's Principle

"The security of a cryptosystem must not depend on keeping secret the crypto-algorithm. The security depends only on keeping secret the key"

Auguste Kerckhoff von Nieuwenhof

Dutch linguist

1883

Principle 5: Open Design

Security should not depend on secrecy of design or implementation

P. Baran, 1965

- "Security through obscurity"
- Does not apply to information such as passwords or cryptographic keys

Example 6: Content Scrambling System

- DVD content
  - SecretEcrypt(Movie,K_T)
  - SecretEcrypt(K_T,K_D)
  - Hash(K_D)
  - SecretEcrypt(K_D,K_{p1})
  - …
  - SecretEcrypt(K_D,K_{pn})
- 1999
  - Norwegian group derived SecretKey by using K_{Pi}
  - Plaintiff's lawyers included CSS source code in the filed declaration
  - The declaration got out on the internet

Example 7:
Getting *root* Access in BSD Unix

Two-conditions for getting root access

- Knowledge of *root* password
- Group *wheel* membership

Principle 6: Separation of Privilege

Require multiple conditions to grant permission

R. Needham, 1973

- Similar to separation of duty
- Another example:
  - Two-factor authentication

Principle 7:
Least Common Mechanism

Mechanisms should not be shared

- Information can flow along shared channels in uncontrollable way
- Covert channels
- Isolation
  - Virtual machines
  - Sandboxes

Example 8: Switching between user accounts

- Windows NT -- pain in a neck
- Windows 2000/XP -- “Run as …”
- Unix -- “su” or “sudo”

Principle 8: Psychological Acceptability

Security mechanisms should not add to difficulty of accessing resource

- Hide complexity introduced by security mechanisms
- Ease of installation, configuration, use
- Human factors critical here

Example 9: Windows Server 2003

Potential problem	Mechanism	Practice
Buffer overflow	defensive programming	check preconditions
Even if it were vulnerable	IIS 6.0 is not up by default	no extra function-ty
Even if IIS were running	default URL length 16 KB	conservative limits
Even if the buffer were large	the process crashes	fail-safe
Even if the vulnerability were expl.	Low privileged account	least privileged

Principle 9: Defense in Depth

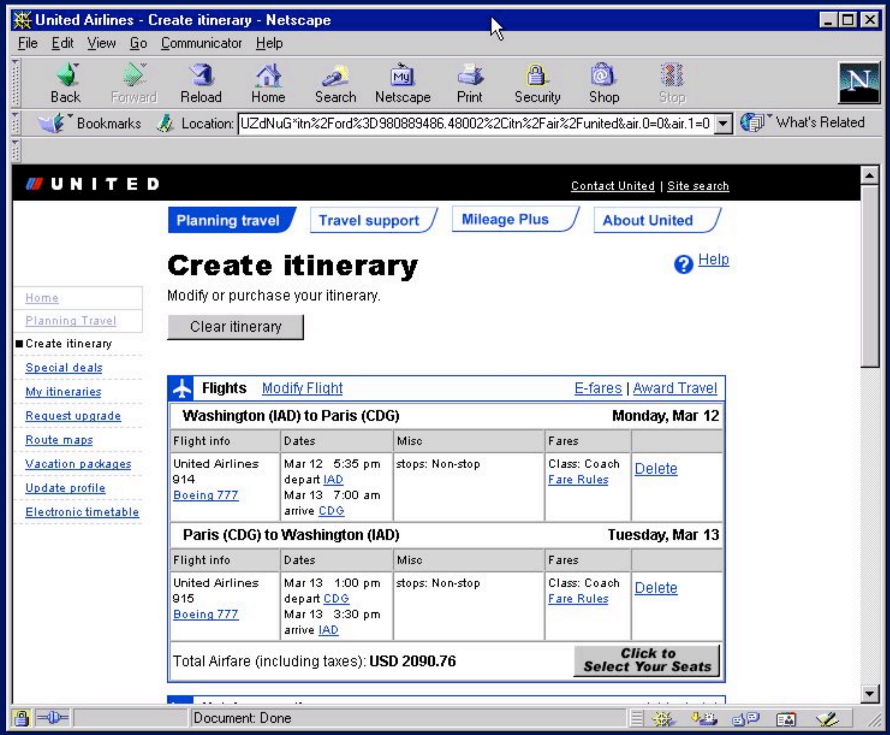
Layer your defenses

Example 10: Assumtpions, Assumptions, …

- *ident*
- *finger* protocol

Example 11: Assuming Honest Client

- Web server application requires browser to validate input
- Attack plan:
  - Remove the client from the communications loop and talk directly to the server
  - Leverage incorrect trust model, trusting the client



Principle 10: Question Assumptions

Frequently re-examine all the assumptions about the threat agents, assets, and especially the environment of the system

Summary

- Overarching Goals
- Principles
  1. Least Privilege
  2. Fail-Safe Defaults
  3. Economy of Mechanism
  4. Complete Mediation
  5. Open Design
  6. Separation of Privilege
  7. Least Common Mechanism
  8. Psychological Acceptability
  9. Defense in depth
  10. Question assumptions

Principles of Designing Usable Security

Food for Thought

"Humans are incapable of securely storing high-quality cryptographic keys, and they have unacceptable speed and accuracy when performing cryptographic operations.

(They are also large, expensive to maintain, difficult to manage, and they pollute the environment.

It is astonishing that these devices continue to be manufactured and deployed.

But they are sufficiently pervasive that we must design our protocols around their limitations.)"

Charlie Kaufman, Radia Perlman, Mike Speciner in "Network Security: Private Communication in a Public World"

What's More Important:

The correctness of security functions/mechanisms,

or

the correct use of them?

Outline

- Principles of secure interaction design
- Five lessons about usable security

Usability and Security Tradeoffs

- A computer is secure from a particular user's perspective if the user can depend on it and its software to behave as the user expects.

- Acceptable security is a requirement for usability.

- Acceptable usability is a requirement for security.

Secure Interaction Design

Ka-Ping Yee
with Norm Hardy, Mark Miller, Chip Morningstar,
Kragen Sitaker, Marc Stiegler, Dean Tribble, and Miriam Walker

Basic Concepts

ACTOR-ABILITY MODEL

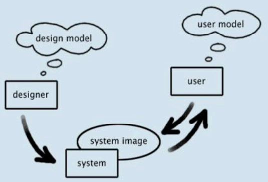
At any point in time, the user's model contains a set of **actors** in the system and a set of **potential actions** for each actor. For a system to be secure, the actual abilities of any actor must never come to exceed the bounds in the user model.

actors $A = \{A_0, A_1, \ldots, A_n\}$
perceived abilities $P = \{P_0, P_1, \ldots, P_n\}$
real abilities $R = \{R_0, R_1, \ldots, R_n\}$

$P_0 \subseteq R_0$
$P_i \supseteq R_i$ for $i > 0$

SYSTEM IMAGE

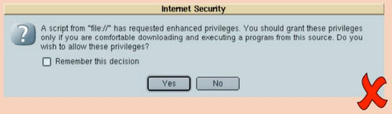
The actors, actions, and objects in the user's mental model are derived from observing the **system image**, not from knowledge of its internal design.



USERS AND USER AGENTS

The software system intended to serve and protect the interests of the user is the **user agent**. On a stand-alone PC, this is the operating system shell, through which the user interacts with an arena of entities such as files and programs. On a networked PC, a second level of user agent represents the user's interests in a larger arena of interacting computers.

Fundamental Principles

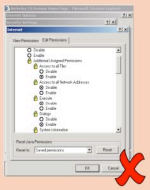


PATH OF LEAST RESISTANCE

The natural way to do any task should also be the secure way.

APPROPRIATE BOUNDARIES

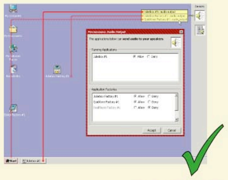
The interface should expose distinctions between objects and between actions along boundaries that matter to the user.



Actor-Ability State

VISIBILITY

The interface should allow the user to easily review any active authority relationships that would affect security-relevant decisions.

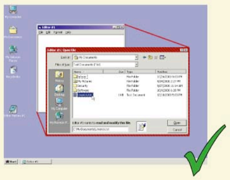


REVOCABILITY

The interface should allow the user to easily revoke authorities that the user has granted, wherever revocation is possible.

EXPLICIT AUTHORITY

A user's authorities must only be provided to other actors as a result of an explicit action that is understood by the user to imply granting.



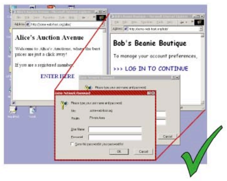
EXPECTED ABILITY

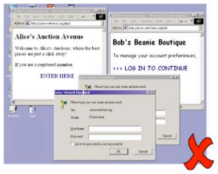
The interface must not generate the impression that it is possible to do something that cannot actually be done.

Input and Output

TRUSTED PATH

The interface must provide an unspoofable and faithful communication channel between the user and any entity trusted to manipulate authorities on the user's behalf.





IDENTIFIABILITY

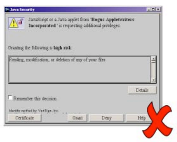
The interface should enforce that distinct objects and distinct actions have unspoofably identifiable and distinguishable representations.

EXPRESSIVENESS

The interface should provide enough expressive power to (a) describe a safe security policy without undue difficulty and (b) allow users to express security policies in terms that fit their goals.

CLARITY

The effect of any security-relevant action must be clearly apparent to the user before the action is taken.

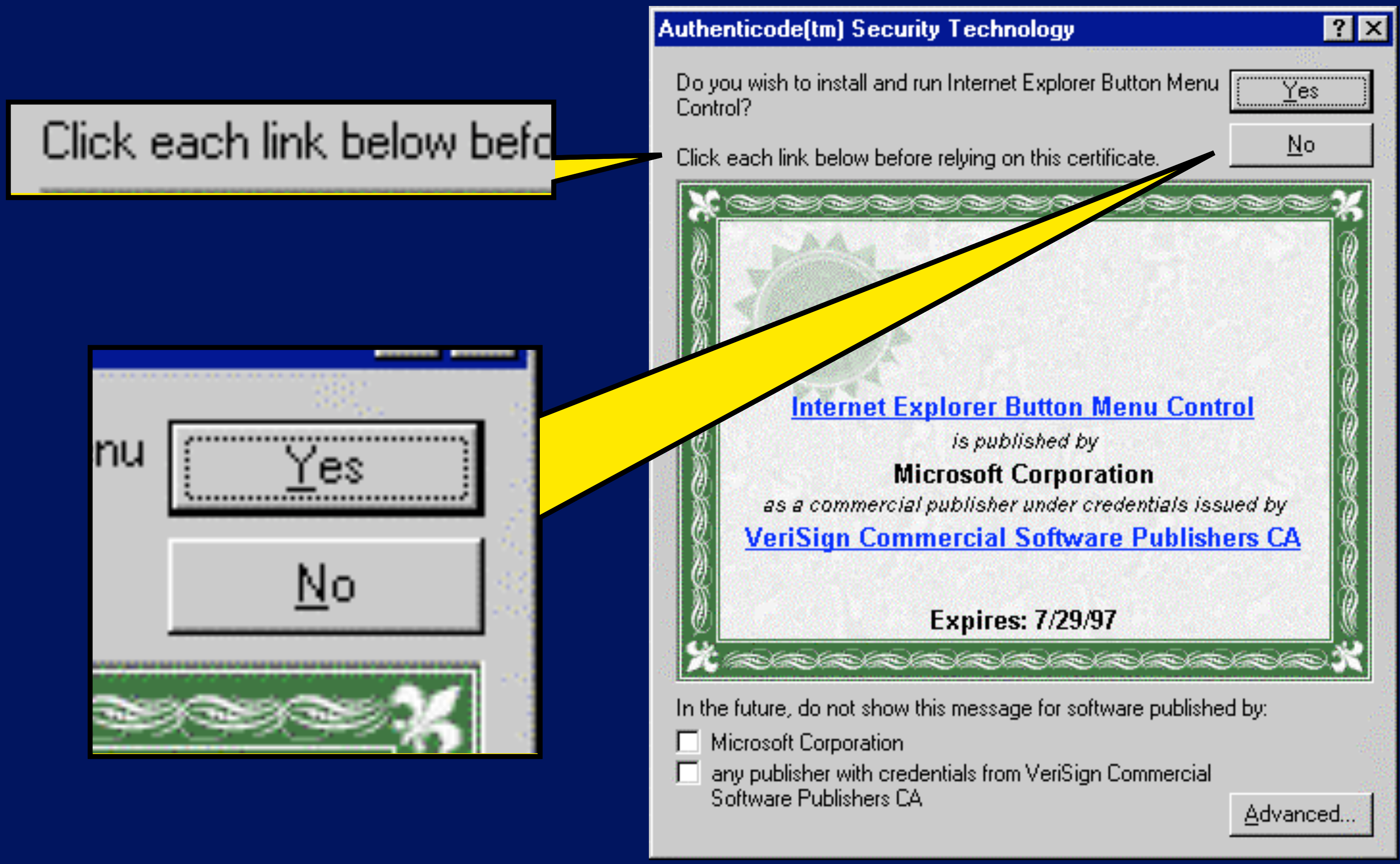


Principle 1:
Path of Least Resistance

To the greatest extent possible,
the natural way to do a task should be
the secure way.

Example 1: Least resistance

-



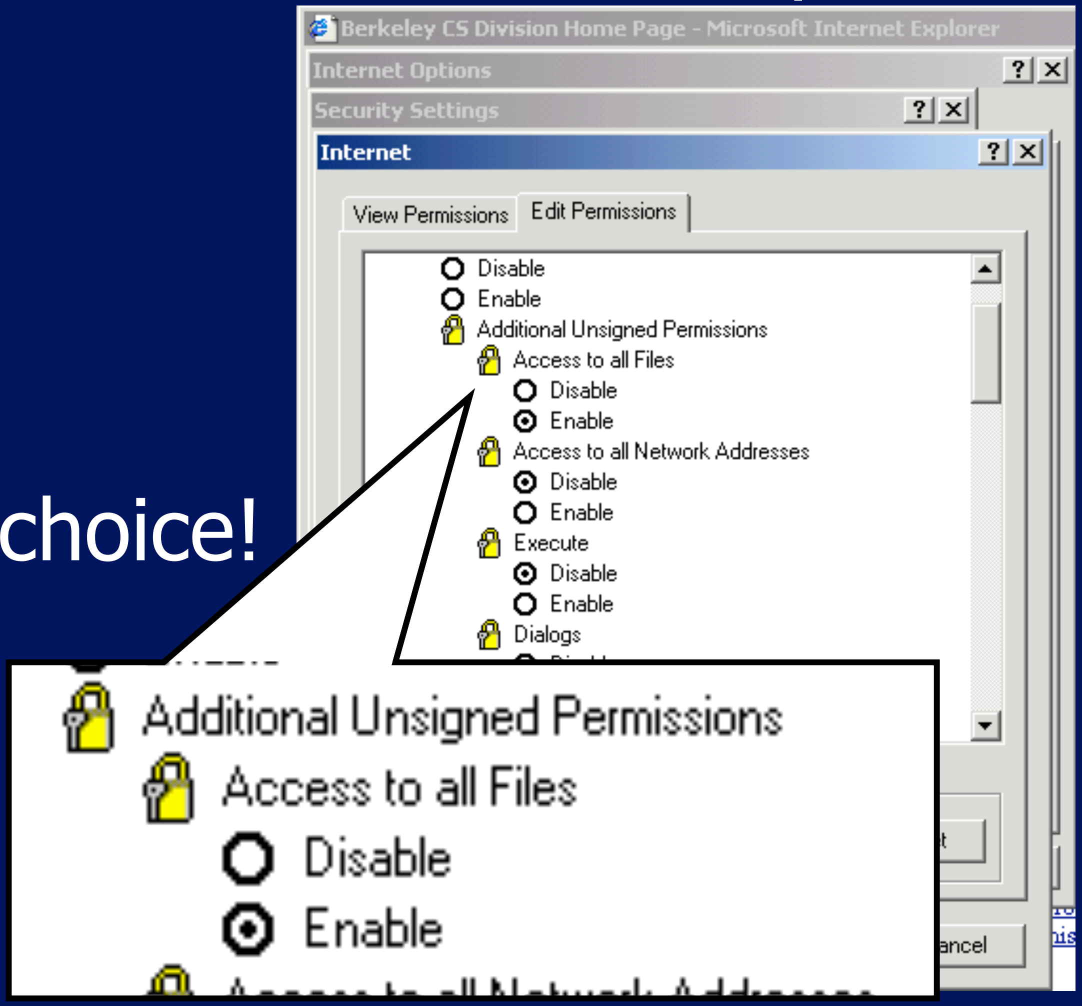
Principle 2: Appropriate Boundaries

The interface should expose, and the system should enforce, distinctions between objects and between actions that matter to the user.

I.e., any boundary that could have meaningful security implications to the user should be visible, and those that do not should not be visible.

Example 2: Bad boundaries

- A real dialog window in Internet Explorer:



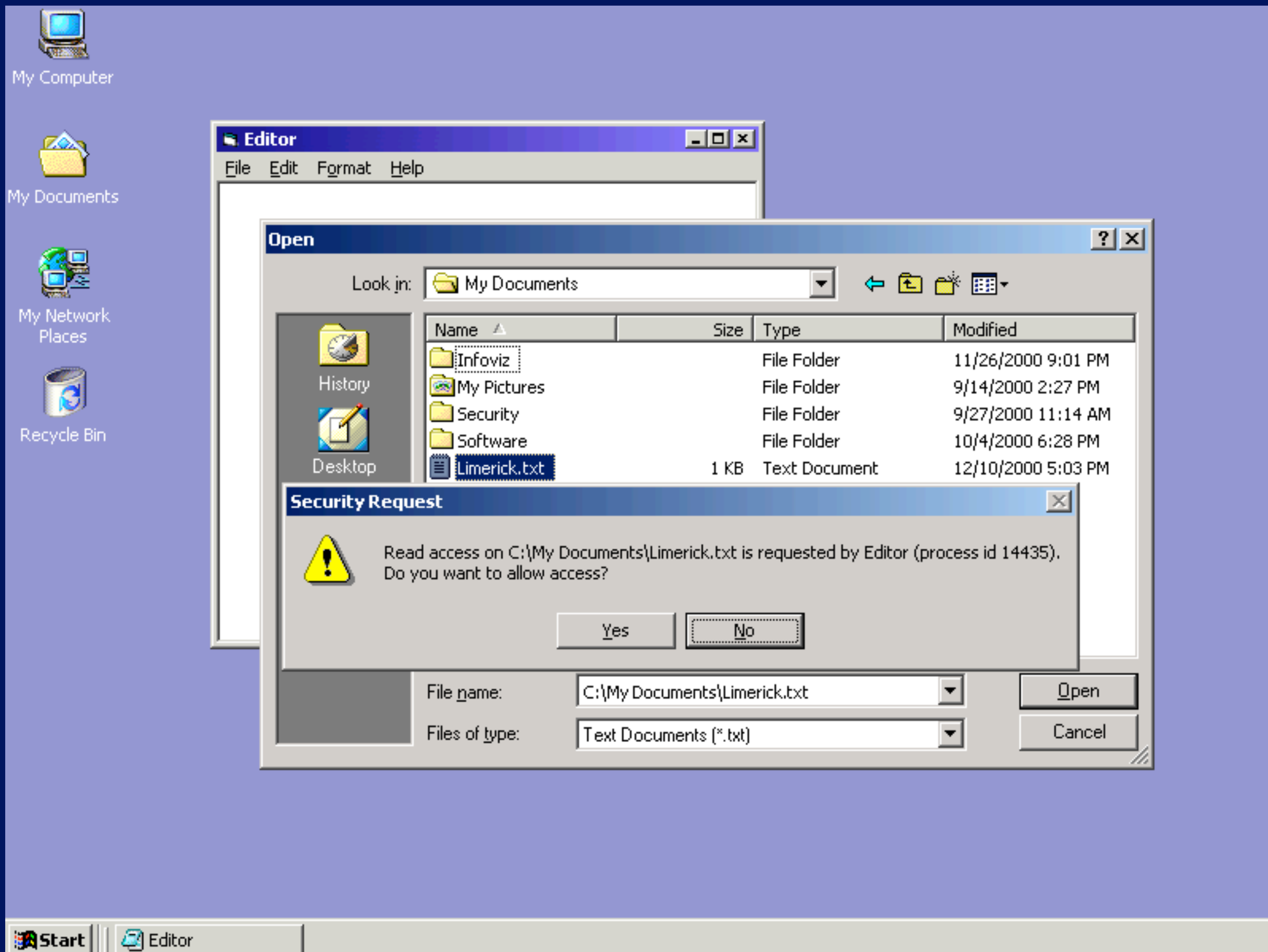
- User is forced to make an all-or-nothing choice!

Principle 3: Explicit Authorization

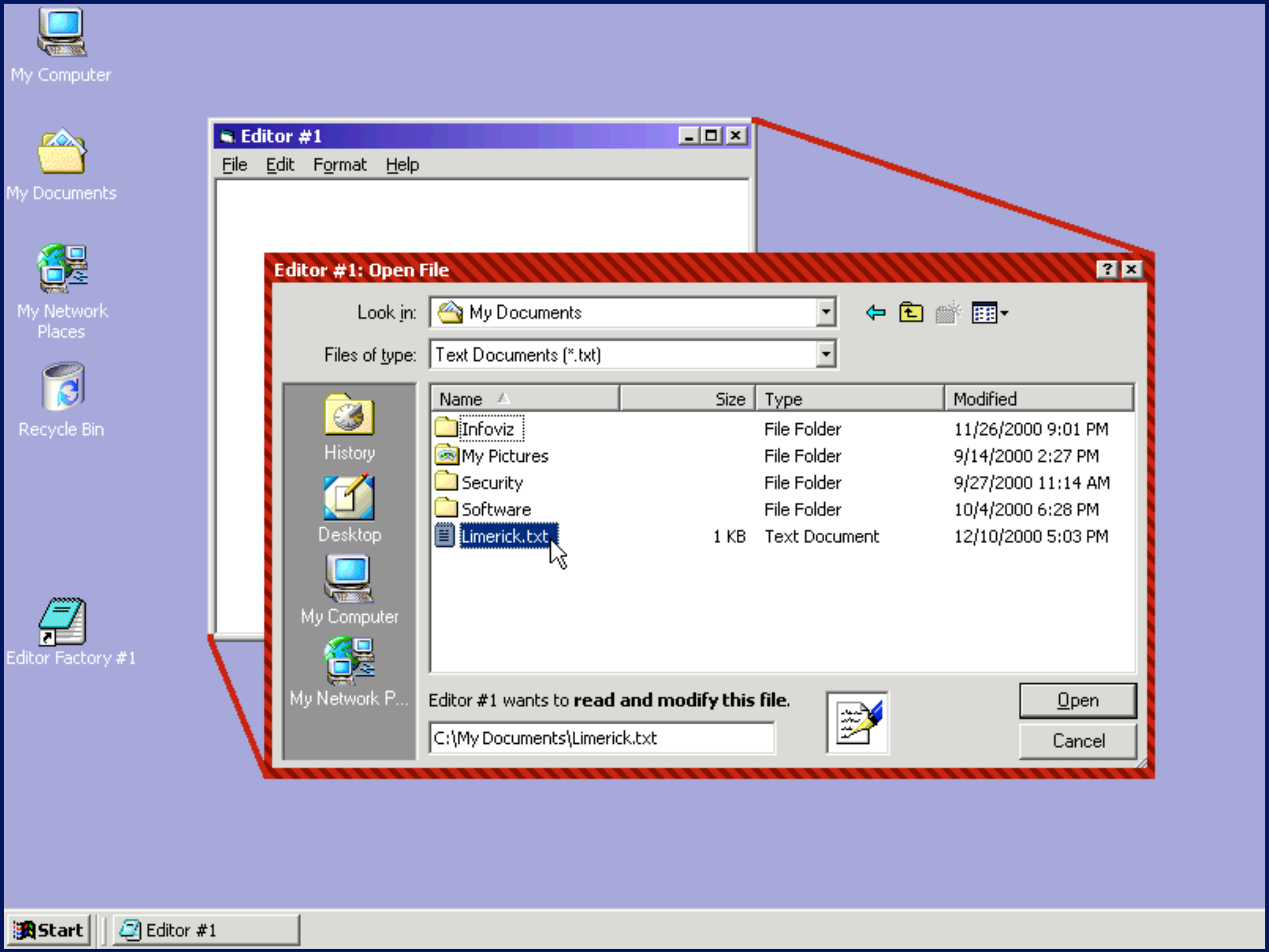
A user's authorities must only be provided to other actors as a result of an explicit action that is understood to imply granting.

- Conflicts with Least Resistance?
- Authorizes the increase of privileges
- Combining designation with authorization

Example 3: When do we ask?



Example 3: When do we ask?



Principle 4: Visibility

The interface should allow the user to easily review any active authorizations that would affect security-relevant decisions.

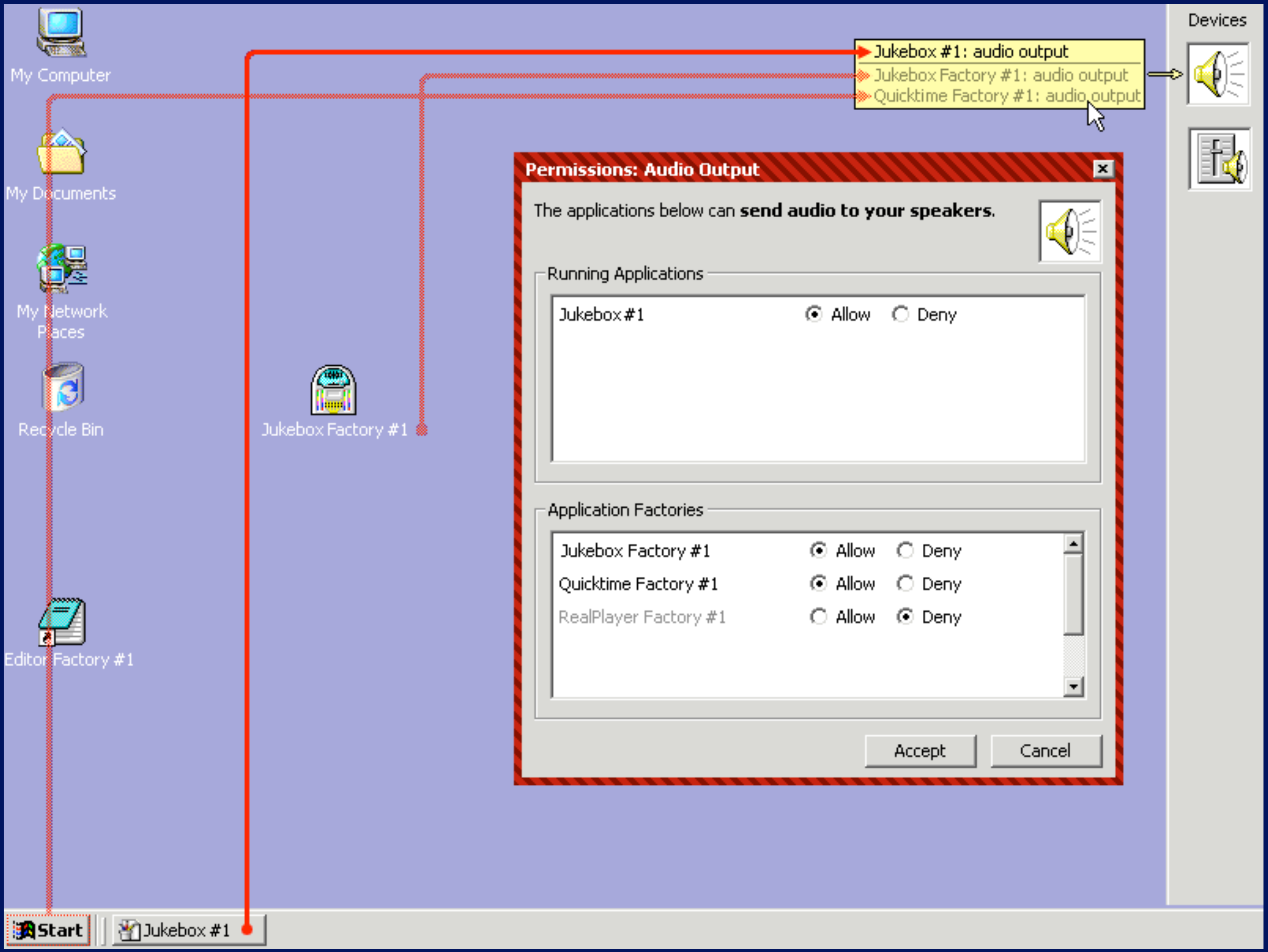
Example 4: What do we show?

Not this:

```
  7:09am  up 117 days,  6:02,  1 user,  load average: 0.17, 0.23, 0.23
110 processes: 109 sleeping, 1 running, 0 zombie, 0 stopped
CPU states:  7.6% user,  4.5% system,  0.0% nice, 87.8% idle
Mem:   512888K av,  496952K used,   15936K free,      60K shrd,   29728K buff
Swap: 1052216K av,  146360K used,  905856K free                 181484K cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
24733 root      18   0  2556 2556   488 S     6.0  0.4   1:42 chargen
25184 ping      16   0   996  996   748 R     3.9  0.1   0:01 top
24276 root       9   0  1888 1864  1484 S     0.7  0.3   0:04 sshd
23519 apache    10   0 21792  13M  8080 S     0.1  2.6   0:23 httpd
23520 apache    10   0 21456  12M  8076 S     0.1  2.5   0:20 httpd
    1 root       8   0   188  148   148 S     0.0  0.0   0:25 init
    2 root       9   0     0    0     0 SW    0.0  0.0   0:00 keventd
    3 root       9   0     0    0     0 SW    0.0  0.0   0:00 kapm-idled
    4 root      19  19     0    0     0 SWN   0.0  0.0   0:33 ksoftirqd_CPU0
    5 root       9   0     0    0     0 SW    0.0  0.0  94:12 kswapd
    6 root       9   0     0    0     0 SW    0.0  0.0   0:02 kreclaimd
    7 root       9   0     0    0     0 SW    0.0  0.0   0:08 bdflush
    8 root       9   0     0    0     0 SW    0.0  0.0   0:15 kupdated
    9 root      -1 -20     0    0     0 SW<   0.0  0.0   0:00 mdrecoveryd
  654 root       9   0   348  288   288 S     0.0  0.0   2:41 syslogd
  659 root       9   0   852  120   120 S     0.0  0.0   0:06 klogd
  744 root       9   0  1988 1988  1728 S     0.0  0.3   0:07 ntpd
  757 daemon     9   0   172  116   116 S     0.0  0.0   0:00 atd
  786 root       9   0   360  232   200 S     0.0  0.0   0:03 sshd
  807 root       8   0   476  336   292 S     0.0  0.0   0:56 xinetd
  866 root       8   0   396  332   312 S     0.0  0.0   0:34 crond
  915 root       9   0  2076  476   476 S     0.0  0.0   0:25 miniserv.pl
  919 root       9   0   108   48    48 S     0.0  0.0   0:00 mingetty
  920 root       9   0   108   48    48 S     0.0  0.0   0:00 mingetty
```

Example 4: What do we show?



Principle 5: Identifiability

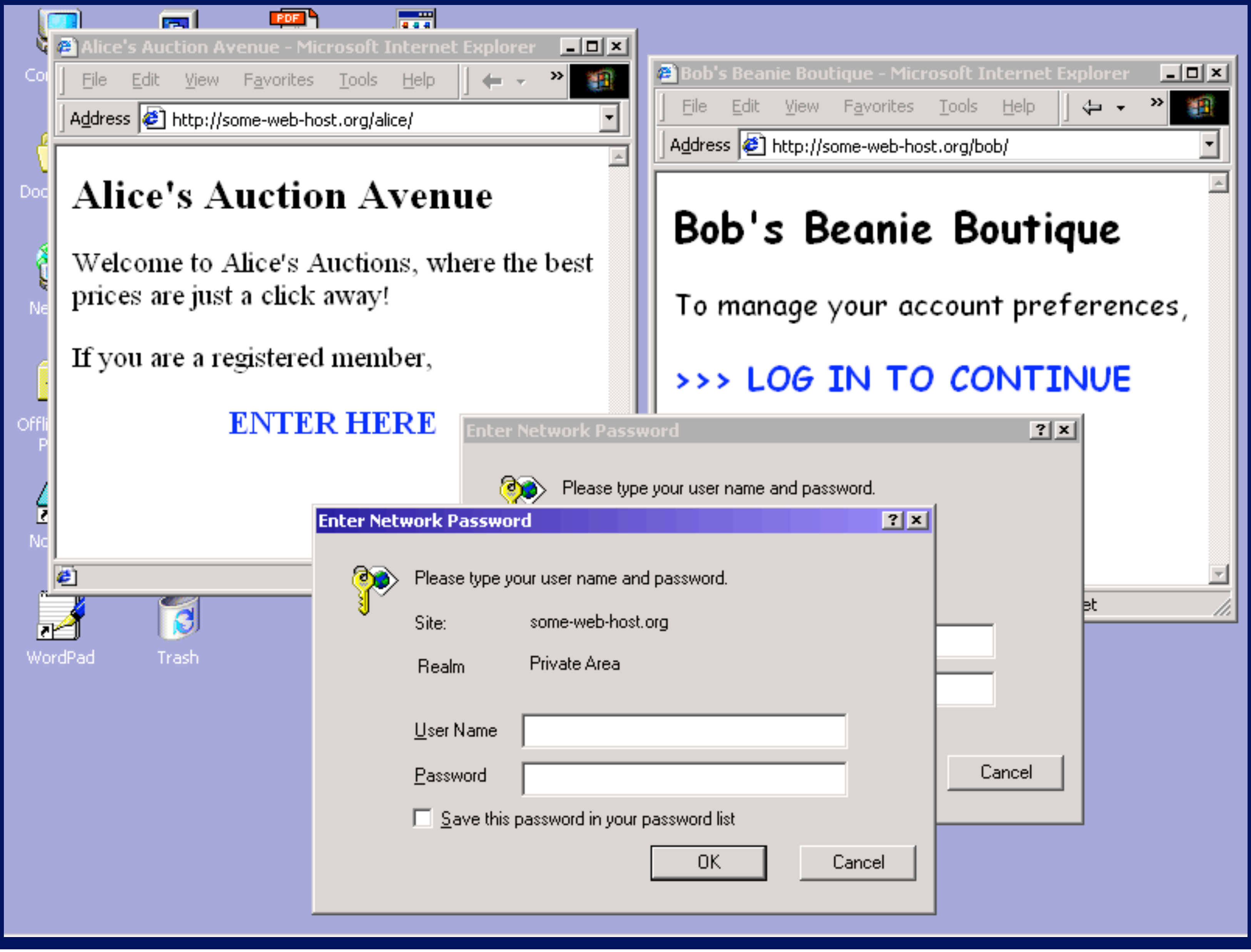
The interface should enforce that distinct objects and distinct actions have unspoofably identifiable and distinguishable representations.

two aspects

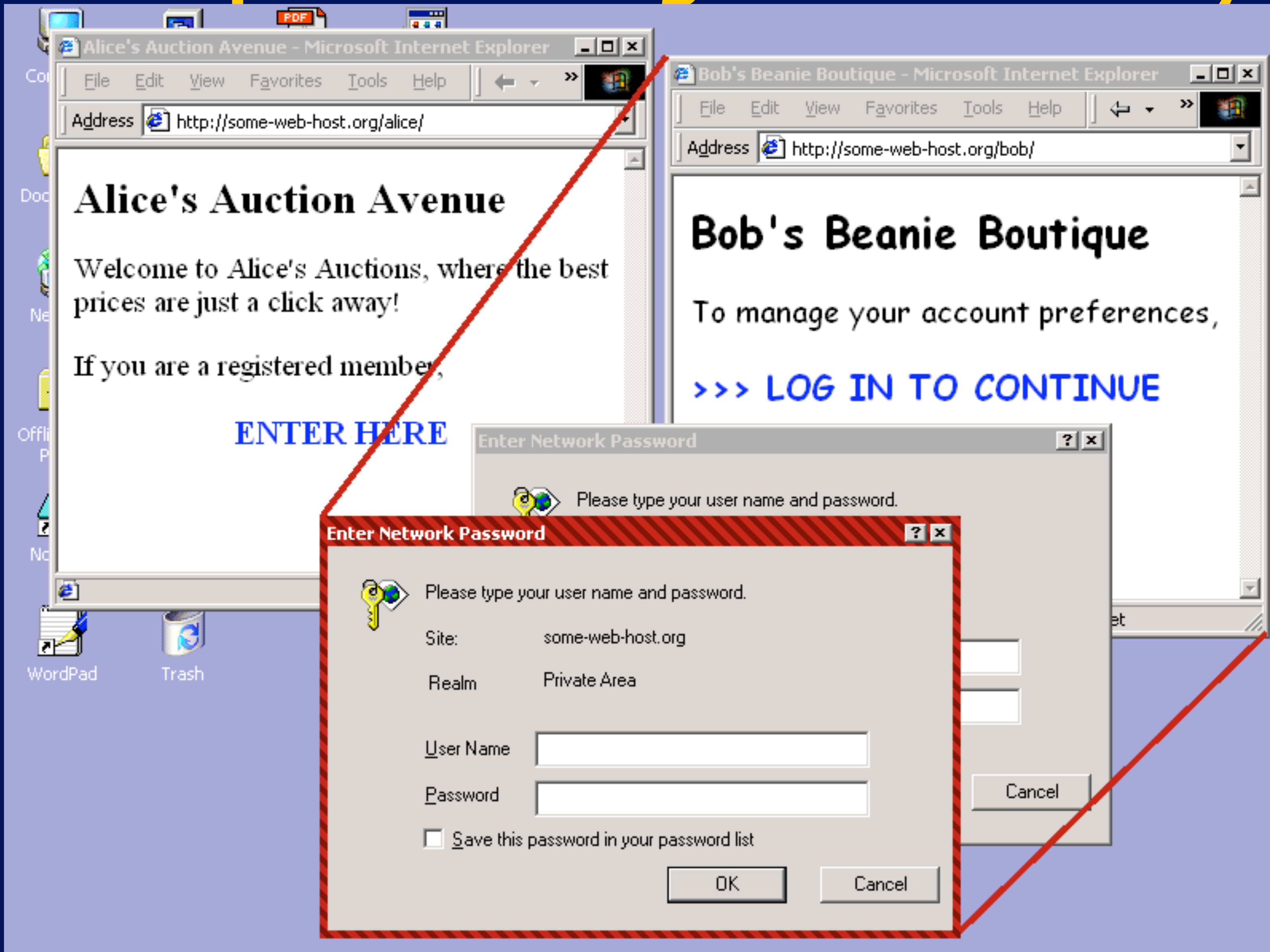
- Continuity: the same thing should appear the same
- Discriminability: different things should appear different

- *perceived* vs. *be* different/same

Example 5: Violating identifiability



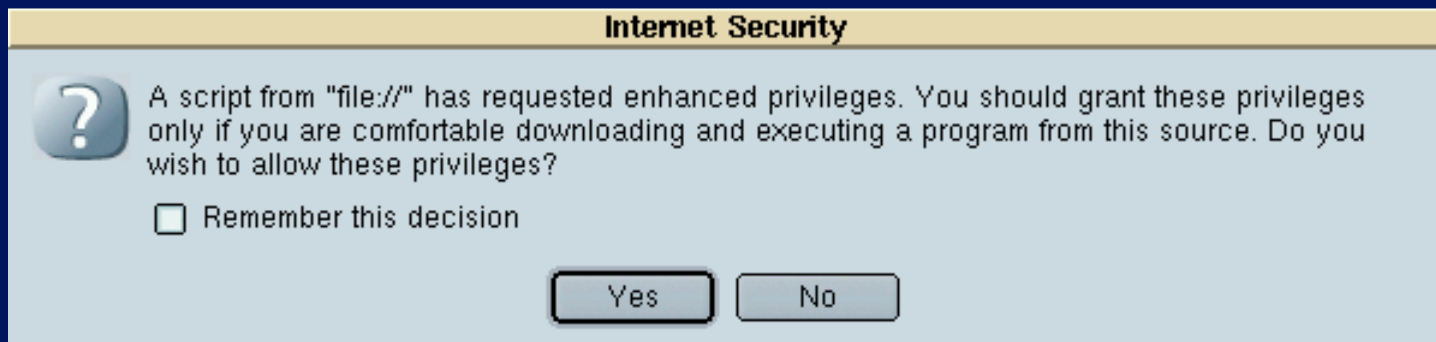
Example 5: Fixing identifiability



Principle 6: Clarity

The effect of any security-relevant action must be apparent before the action is taken.

Example 6: Violating Clarity



What program?
What privileges?
How long?
Remember this decision?

What source?
What purpose?
How to revoke?
What decision?

Might as well click “Yes”: it’s the default.

Principle 7: Expressiveness

In order for the security policy enforced by the system to be useful, we must be able to express a **safe policy**, and we must be able to express the **policy we want**.

Example 7: Unix File Permissions

```
-rw-r--r--    1 konstant  konstant    89418 18 Oct 13:57 Berry 2002 painpaper.pdf
-rwxr--r--    1 konstant  konstant  3639577  8 Oct 17:32 MarineAquarium206_OSX.dmg
drwxrwxrwx    3 konstant  konstant      102 17 Oct 08:11 My Great DVD.dvdproj
-rw-r--r--    1 konstant  konstant    50536 18 Oct 13:57 Shaw 2001.pdf
drwxr-xr-x  267 konstant  konstant     9078 25 Nov 11:33 downloads
-rw-r--r--    1 konstant  konstant     9204 29 Aug 14:29 konstantin_beznosov_thumbnail.jpg
-rw-r--r--    1 konstant  konstant   158195 18 Oct 13:57 shaw 2002 SE rsrch.pdf
-rw-r--r--    1 konstant  konstant   255671 18 Oct 13:57 shaw 2003 -icse03.pdf
-rw-r--r--    1 konstant  konstant     5318  9 Oct 23:16 sidney_fels.jpg
-rw-r--r--    1 konstant  konstant      139 22 Nov 13:09 wcsf-notes.rtf
```

Usable Security Principles Summary

In order to use a system safely, a user needs to have confidence in all of the following statements:

1. Things don't become unsafe all by themselves. (Explicit Authorization)
2. I can know whether things are safe. (Visibility)
3. I can make things safer. (Revocability)
4. I don't choose to make things unsafe. (Path of Least Resistance)
5. I know what I can do within the system. (Expected Ability)
6. I can distinguish the things that matter to me. (Appropriate Boundaries)
7. I can tell the system what I want. (Expressiveness)
8. I know what I'm telling the system to do. (Clarity)
9. The system protects me from being fooled. (Identifiability, Trusted Path)

Lessons learned about usable security

1. You cannot retrofit usable security
   - Adding explanatory dialogs to a confusing system makes it more confusing
2. Tools are not solutions
   - They are just Lego™ blocks
3. Mind the upper layers
   - Application-level security design allows intentional, implicit, application-specific security
4. Keep your users satisfied
   - Put your users' needs first
   - Evaluate your solution on the target audience
5. Think locally, act locally
   - Don't assume support from global infrastructure (e.g., PKI)
   - If a generic security tool can be used independently of application, the user(s) must deal with it directly