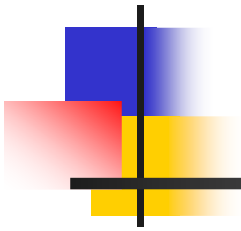
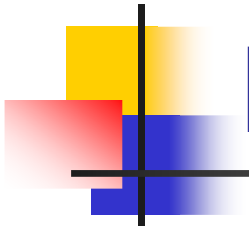


CS 155 – Computer & Network Security

Programming Project 2 – Spring `04

Priyank Patel
pkpatel@cs.stanford.edu

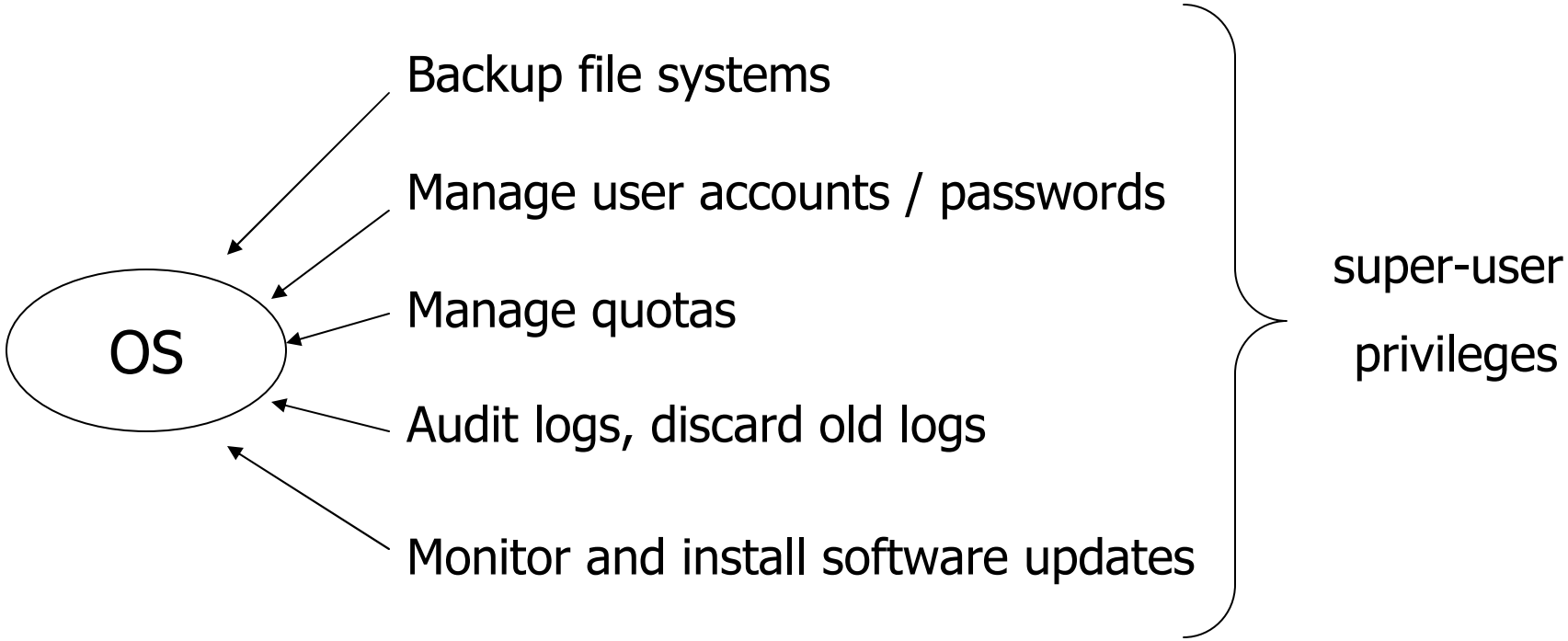


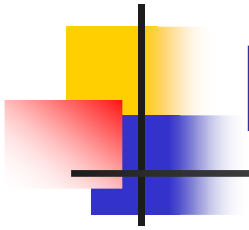
PP2 from 50,000 feet

- Extend an already existing shell
- Execute shell as setuid root
- Provide root privileges for
  ‘specific commands’ by ‘specific users’
- All other commands run at normal privileges
- Keep audit trails – logging

Motivation

A Unix Deployment



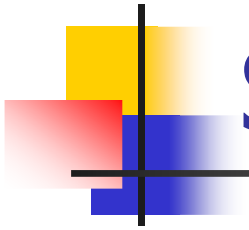


Problem…

- There is only one ‘root’
- … and several super-users

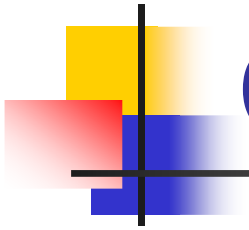
- Give ‘root’ to every admin
  - Each admin has more than required privs
  - Can modify/harm intentionally/accidentally

- Who is accountable if something screws up?



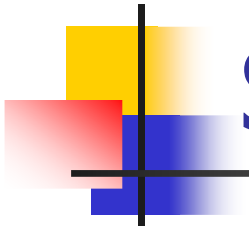
Solutions (a few)

- #1 – Sudo (super-user doer)
  - separate program which allows users to runs programs as root
  - configured by the root
- #2 – Operator Shell (Mike Neuman)
  - setuid root shell
  - decides if a particular user can access certain files or execute certain programs as root.



Our Solution

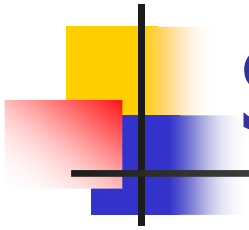
- Josh – Journaling Operator Shell
  - Similar idea to Neuman's operator shell
  - Runs as setuid root
  - Two config files :
    - /etc/josh_exec :
      - decide what programs can be executed as root by specific users
    - /etc/josh_access :
      - decide which users have access to which files



Starter code

- Build on the Old Shell (Gunnar Ritter)
- 847 lines of code
  [including the Copyright notice and comments]

- Implements the basic parsing and the fork-exec infrastructure.
- Work under *boxes* this time around as well.

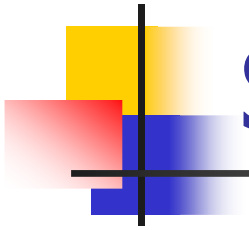


Step 1 : Secure the Perimeter

- Audit the current code – mitigate the effects of questionable code.
  - Eg : substvars() – buffer overflow,
    pcmd() – array overflow, etc.

- Current exec code in the forked child

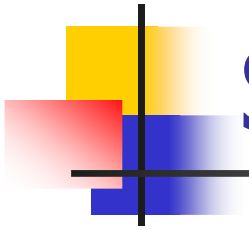
```
execv(args[0], args);
        /* try current directory first */
if (errno == ENOENT)
        execvp(args[0], args); /* try $PATH*/
```



Step 1 – (contd.)

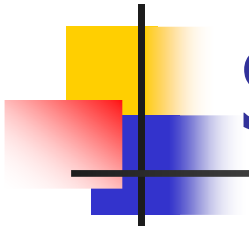
- execv – searches in the current directory
- execvp – uses $PATH to search for executable

- Change required
  - Try current directory only if
    - '.' in $PATH Eg : PATH=/bin:.:/sbin
    - Empty path Eg : PATH=/bin:/sbin::/usr/bin



Step 2 : Executables

- Config file : /etc/josh_exec
- Installed ‘root:root’ with perms ‘600’
- Josh aborts at startup if /etc/josh_exec is writable by any user other than root
- Contains entries of the form **`“userid:progpath”`**

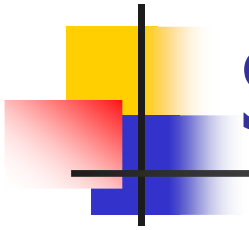


Step 2 : (contd.)

- Sample /etc/josh_exec

```
bob:/usr/kill
alice:/sbin/shutdown
```

- alice should be able to run ‘shutdown’ by specifying non-absolute pathnames
  - Eg : `alice:~$  ../../sbin/shutdown`

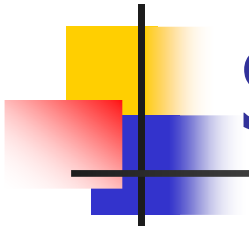


Step 3 : File Access

- Config file : /etc/josh_access
- Installed ‘root:root’ with perms ‘600’
- Josh startup behavior similar to josh_exec

- Contains entries of the form

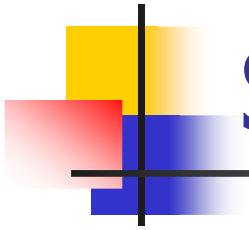
```
“userid:filepath:perms”
perms : [+-](r|w|rw)
```



Step 3 : (contd.)

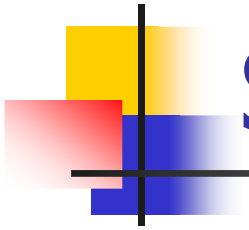
- ‘r’ : read, ‘w’ : write
- ‘+/-’ : grant/revoke permission

- Can revoke permission only if previously granted by Josh
- If user has access to a file through normal Unix perms, Josh should not negate them



Step 3 : (contd.)

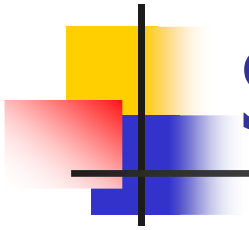
- Entries are cumulative
  - Eg. : alice:/abc:+r
    alice:/abc:+w
- Multiple positive and negative entries :
  last entry wins
- Permissions to the directories apply recursively
- Should not allow to create files in a directory even if ‘+w’ is given on the directory



Step 4 : Editor

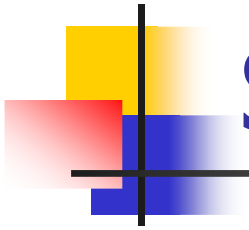
- What if alice has to modify the `/etc/rc.d` scripts?
- Can we add
  - `alice:/usr/bin/vi => josh_exec`
    `alice:/etc/rc.d:+w => josh_access`
- If vi is running as ‘root’, alice can modify any other file in the system –

  including `josh_exec` and `josh_access` !!



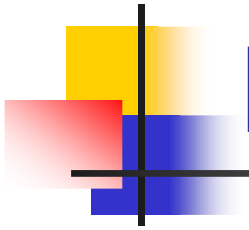
Step 4 : (contd.)

- Implement a shell built-in ‘edit’
- **`edit  /etc/rc.d/rc2.d/inetd_script`**
  - Copy `/etc/rc.d/rc2.d/inetd_script` to `/tmp`
  - Allow alice to modify the file in `/tmp`
  - On exit from `vi`, copy back the file from `/tmp` to the original location



Step 5 : Journaling

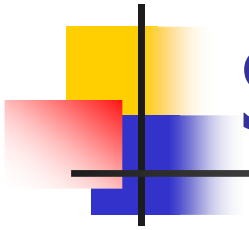
- Josh maintains 3 types of log events
  - OK : everything is fine
  - FAILED : exec call fails
    (non-executable / non-existent file)
  - DENIED : try to edit without sufficient privs
    in Unix AND under Josh
- Logging unit : single pipeline
  [the argument to ppipe()]



Extra-credit

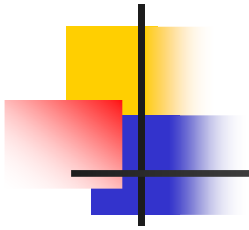
- Josh as a collaboration utility

- `alice` creates `josh_*` in home directory
- Grant access to other users over files which are readable and writable by `alice`
- Other users can access/execute with the same privileges as `alice`



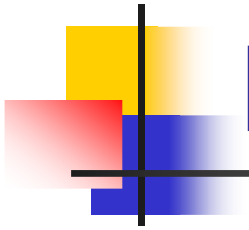
Suggestions

- Familiarize yourself with the Josh code
- Design your solution before diving in and plugging code
- Follow the principle of least privilege
- Make reasonable checks on user input



… Suggestions

- Read the references thoroughly
- The original shell terminates on several errors, its fine if your Josh does the same.
  Be sure to **fail-safe,** even if you have to **fail-stop**
- Document your design in the README
- Please start early



Finally…

- Don’t write target8 for project 1 in Spring`05!
- Have fun…