Securing the User's Work Environment

Nicholas P. Cardo

Sterling Software, Inc.

Numerical Aerospace Simulation Facility

NASA Ames Research Center

Moffett Field, CA USA

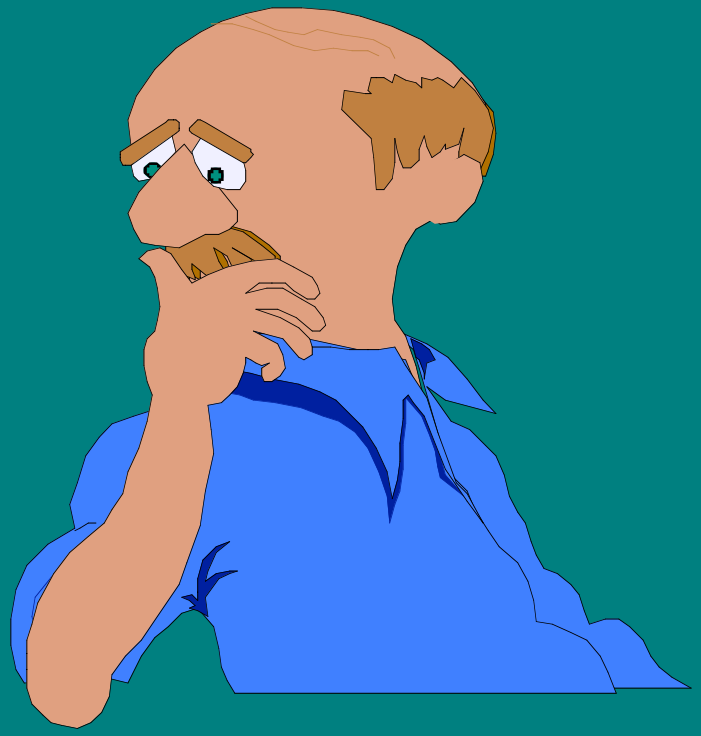
cardo@nas.nasa.gov

Table of Contents

- The Problems
- Developing Applications
- Useful Utilities
- Summary

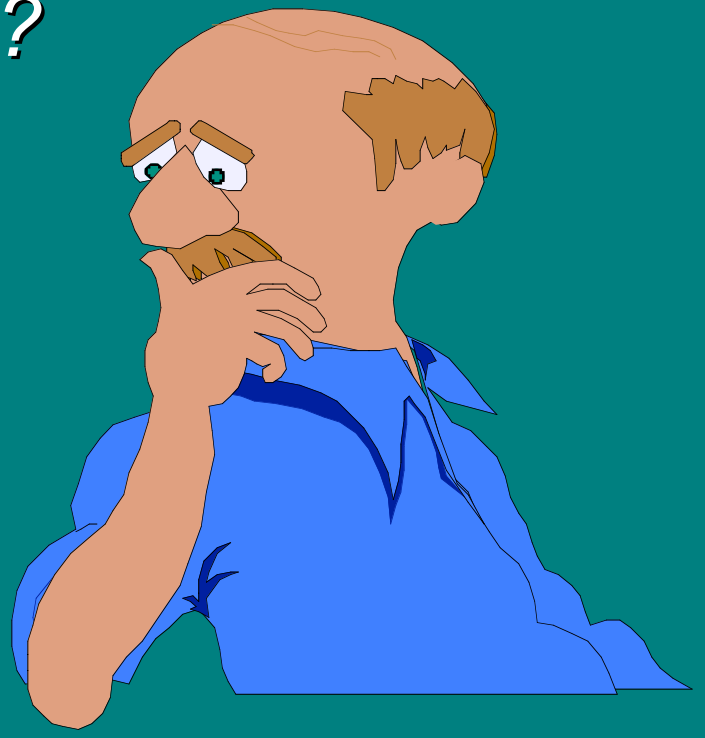
Problem: File Ownership

- Is gid valid for user?
- Is uid a valid user?
- Quota scamming!



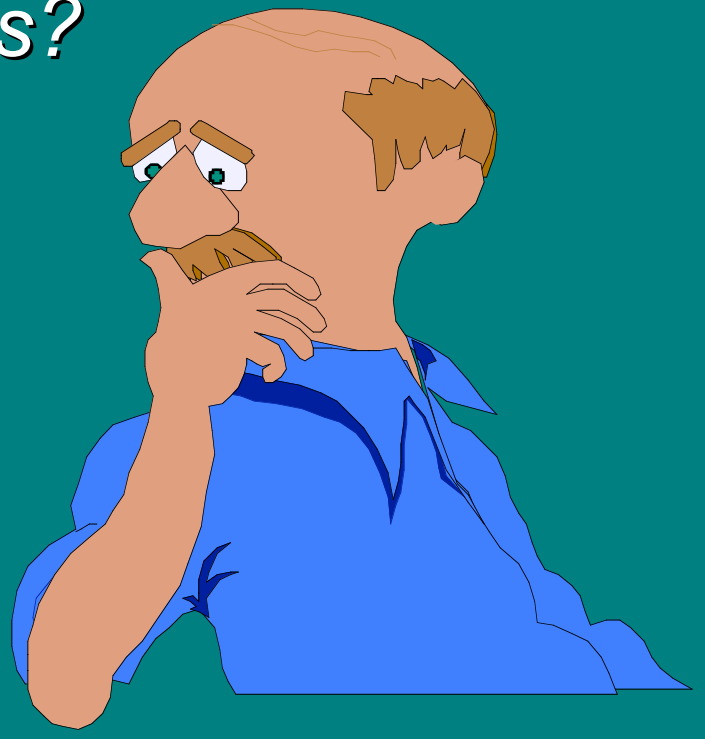
Problem: Home Directories

- Directory mode OK?
- Proper uid ownership?
- Proper gid ownership?



Problem: Home Filesystems

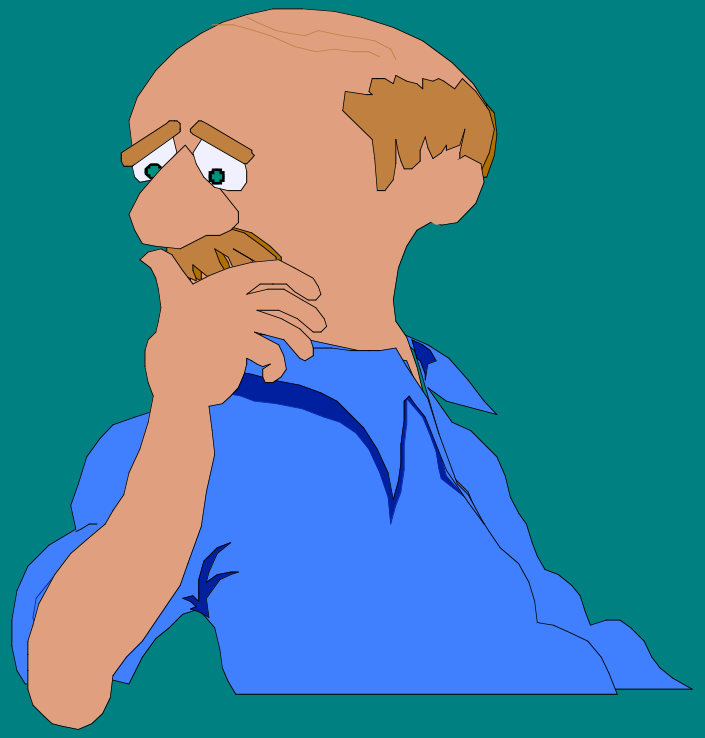
- Does each user have a login directory?
- Does each directory have a valid user?
- Any non-directory files?



Problem: .rhosts

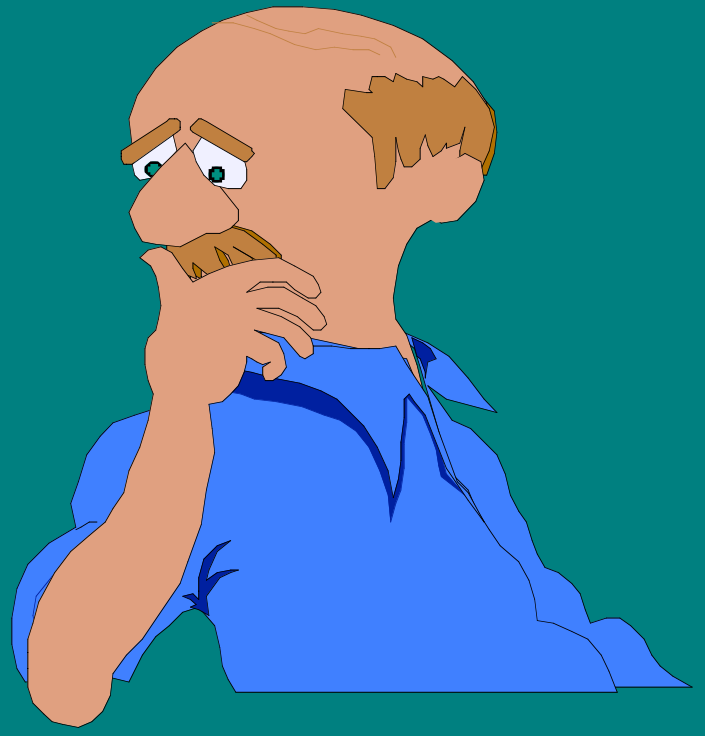
Need I say more?

- *File ownership*
- *File mode*
- *Well formed entries*
- *Account Sharing*
- *Valid hosts*



Problem: Secure SHell (SSH)

- $HOME/.shosts
- $HOME/.ssh/authorized_keys



Developing Applications

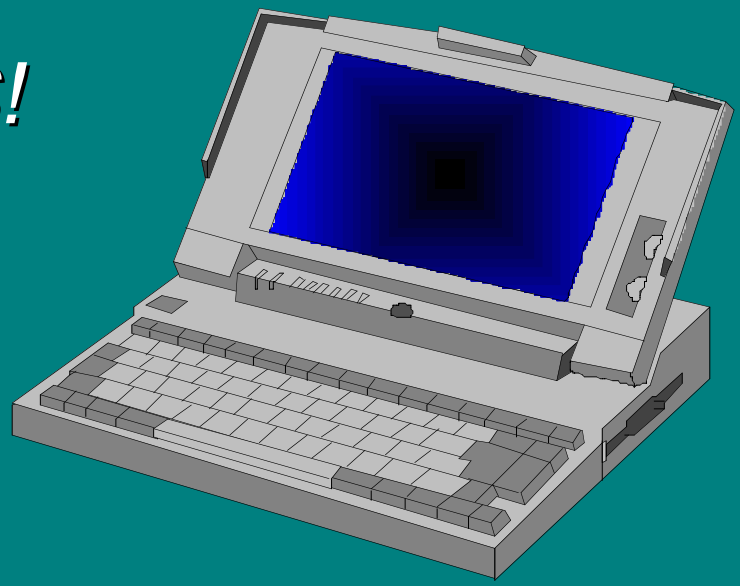
Goals:

- *Speed*
- *Flexibility*
- *Portability*

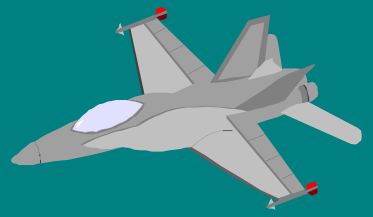
Don’t forget accuracy!

Portability

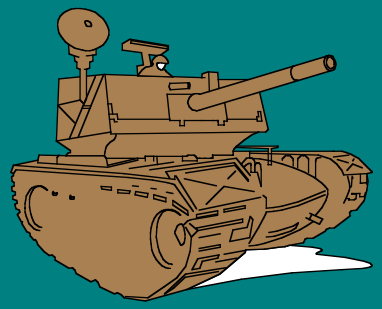
- *No more UNICOS!*
- *No more traditional CRAYs!*
- *Migration towards Scalable Node Architecture*
- *IRIX is **NOT** UNICOS!*



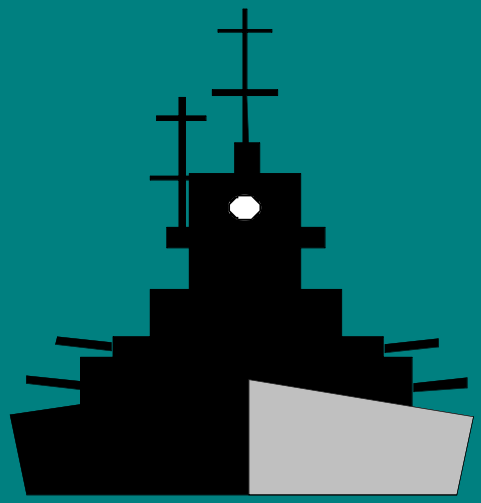
Scripts versus Programs



- Scripts are in most cases slower than programs.
- Scripts rely on programs for information
- Information must be reformated



And the battle begins...



The stat Structure

st_dev	*Device number*
st_ino	*Inode number*
st_mode	*File mode*
st_nlink	*File link count*
st_uid	*User ID ownership*
st_gid	*Group ID ownership*
st_size	*File size*
st_atime	*Last access timestamp*
st_mtime	*Last modify timestamp*
st_ctime	*Last inode update timestamp*

ftw/nftw

File Tree Walk

Provides a simple way for programs to traverse directory trees. Has directory traversal controls. Automatically provides **stat** *structure and pathname to YOUR custom function.*

Sample: Find setuid Programs

```
main
{
    nftw(argv[optind],scantree,32,
        FTW_PHYS|FTW_MOUNT);
}

int scantree(path,sb,code,fcode)
char *path;
struct stat sb;
int   code;
struct FTW *fcode;
```

Sample: Find setuid Programs

```
Switch(code) {
  case FTW_DNR:
  case FTW_NS:
    fprintf(stderr,
      ”can’t read %s\n”,path);
  case FTW_D:
  case FTW_DP:
  case FTW_SL:
    return(0);
  case FTW_F:
    break;
}
```

Sample: Find setuid Programs

```
If (sb->st_mode & S_ISUID)
  print("%s %d %d %o\n",
    path,
    sb->st_uid,
    sb->st_ctime,
    sb->st_mode);
  return(0);
}
```

Reading passwd and group

- getpwent()
- getpwnam()
- getpwuid()

- *getgrent()*
- *getgrnam()*
- *getgrgid()*

Could be multiple lines per group in the group file.

Utilities: homeck

Validate all user's login directory. Check uid and gid ownership as well as the directory's mode. Check for missing login directories. Look for stray files and directories on the home filesystems.

Utilities: remotck

Using the passwd file for a reference, examine each users .rhosts and .shosts files. Validate that each entry is properly formed and that the hosts listed within really exist. Also check the .ssh/authorized_keys files for potentially bad entries. What for account sharing!

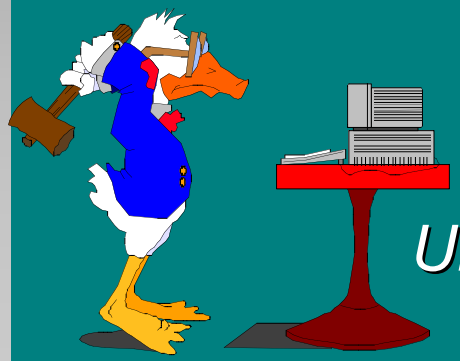
Utilities: scandir

Validate proper ownership for all files in every users directory tree. Look for root owned files and setuid files. Look for dangling softlinks.

Summary

Both IRIX and UNICOS support the necessary library functions and system calls to allow for powerful and portable security utilities to be developed.

The responsibility for finding and correcting these problems falls on both the system/security administrator and the user.



No system is 100% safe...

Unless it's powered off and never powered on...

When All Else Fails