
David Gilliam, John Powell
California Institute of Technology,
Jet Propulsion Laboratory

Matt Bishop
University of California at Davis

California Institute of Technology, Jet Propulsion Lab
Acknowledgement

- NOTE:
  - This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.
  - The work was sponsored by the NASA Office of Safety and Mission Assurance under the Software Assurance Research Program led by the NASA Software IV&V Facility.
  - This activity is managed locally at JPL through the Assurance and Technology Program Office.
Current Collaborators

- David Gilliam – Principle Investigator, JPL
- John Powell – JPL Software Engineer
- Matt Bishop – Associate Professor of Computer Science, University of California at Davis
Agenda

- Goal
- Problem
- Software Security Assessment Instrument (SSAI)
- Final Notes
Goal: Secure Computing

- Secure Computing Environment
  - Secure Systems
  - Secure Data
- Reduce security risk to the computing environment by mitigating vulnerabilities in the software development and maintenance life cycles
- Provide an instrument and tools to help reduce vulnerabilities and exposures in software
- To aid in complying with security requirements and best practices
Agenda

- Goal
- Problem
- Software Security Assessment Instrument (SSAI)
- Final Notes
***STOP: 0x000000D1 (0x00000000, 0xF73120AE, 0xC0000008, 0xC0000000)

A problem has been detected and Windows has been shut down to prevent damage to your computer.

If this is the first time you've seen this Stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

*** WXYZ.SYS - Address F73120AE base at C0000000, DateStamp 36b072a3

Kernel Debugger Using: COM2 (Port 0x2f8, Baud Rate 19200)
Beginning dump of physical memory
Physical memory dump complete. Contact your system administrator or technical support group.
Problem

- Lack of Experts: Brooks —
  - "No Silver Bullet" is still valid
    (IEEE Software Engineering, 1987)
- Poor Security Requirements
- Poor System Engineering
  - Leads to poor design, coding, and testing
- Cycle of Penetrate and Patch
- Piecemeal Approach to Security Assurance
Agenda

- Goal
- Problem
- Software Security Assessment Instrument (SSAI)
- Final Notes
Reducing Software Security Risk Through an Integrated Approach

- Software Vulnerabilities Expose IT Systems and Infrastructure to Security Risks
- Goal: Reduce Security Risk in Software and Protect IT Systems, Data, and Infrastructure
  - Security Training for System Engineers and Developers
  - Software Security Checklist for end-to-end life cycle
  - Software Security Assessment Instrument (SSAI)

SSAI Includes:
- Model-Based Verification
- Property-Based Testing
- Security Checklist
- Vulnerability Matrix
- Collection of security tools
Model Checking & The Flexible Modeling Framework

- MC with FMF Benefits Software Early in its Lifecycle
  - Earlier Discovery of Software Errors
  - Correction is easier / better / less expensive

- FMF must adapt to early lifecycle events
  - Modular model design allows easy extension of existing models
    - Multiple client scenarios for the server login example were quickly modeled and verified
    - The various client scenarios allows extensive off-nominal verification with ease
Model Checking &
The Flexible Modeling Framework (Cont.)

- Rapidly changing requirements and designs
  - Multiple design trade offs in login protocol were easily explored

- Varying levels of detail were defined for different arbitrary system parts
  - Multiple login failure propagation scenarios that are known but not formally defined for different systems using the login protocol
  - Model extensions are readily possible for many if not all of these scenarios
    - These scenarios may be developed quickly and adapted at will
    - These extensions can be cross tested against all client scenarios and protocol design trades with little additional effort.
Tester’s Assistant Specifications

- No access until hash verified

  Obtain user's name
  Obtain user's password
  Is password correct? {
    Generate user’s password hash
    Compare user’s password hash to hash stored for that user name
    If match, set UID to user's uid (granting access)
    If no match, set UID to ERROR (denying access)
  }
  If access granted {
    compare UID to the uid for which access is granted
    if match, all is well
    if no match, specification violated
  }
Server Login Model

- Obtain User Name
- Obtain User Password
- After User Name and Password Obtained
- Use hash to transform password for verification
- If verification fails then deny access
- If verification succeeds then grant access

```c
procute server(){
    do
        ::user_name == 0 && password == 0 ->
            login?user_name;
        ::user_name != 0 && password == 0 -
            >login?password;
        ::user_name != 0 && password != 0 -
            do
                ::user_name != password ->
                    if
                        :: hash_complete == 0->
                            password = password * -1;
                            hash_complete = 1;
                        ::hash_complete == 1->accessed!0; break;
                    fi
                ::user_name == password ->accessed!1;
            break:
    od
```
Client Login Models and Their Verification

channel login = [0] of {short};
...
active proctype good_client(){
  short usr_nm = 1;
  short pswd = -1;
  run server();
  login!usr_nm;
  login!pswd;
  accessed?accsd;
}

channel login = [0] of {short};
...
active proctype bad_client(){
  short usr_nm = 1;
  short pswd = -1;
  run server();
  login!usr_nm;
  login!-65535;
  accessed?accsd;
}

- **Property of Interest**
  - No access until hash verification is complete
    - accsd==0 ∪ hash_complete==1

- **Good Client**
  - Correct Login
    - Hash verification correlates user and password before access

- **Bad Client**
  - circumvents Login
    - Buffer handles data range of \(-2^{15}\) to \(2^{15}-1\) (short data type)
    - -65535 overflows buffer
      - Access granted before password hash execution
      - Hash function ignored
Property-Based Testing

- Property-based testing tool – Tester’s Assistant (Matt Bishop, UC Davis)
  - Perform code slicing on applications for properties for a known set of vulnerabilities
  - Test for vulnerabilities in code on the system or whenever the computing environment changes
  - Initially, checks software developed in JAVA and C
    - The goal is to have the tool check other programming and scripting languages as well (C++, Perl, ActiveX, etc.)
Example C Code

/* get user name */
if (fgets(stdin, uname, sizeof(uname)-1) == NULL)
    return(FAILED);

/* get user password */
typedpwd = getpass("Password: ");

/* now get information about user with that name */
if ((pw = getpwnam(uname)) != NULL){
    /* generate user's password hash */
    hashtp = crypt(pw->pw_passwd, typedpwd);
    /* compare this to stored hash; if match, grant access */
    if (strcmp(pw->pw_passwd, hashtp) == 0){
        /* match -- grant access */
        setuid(pw->pw_uid);
        return(SUCCESS);
    }
    /* didn't match -- fall through to deny access */
}
return(FAILED);
View process as sequence of states
- Care only about those related to properties
- State 1: name, hash, uid for user with login name obtained
- State 2: add in hash of password supplied by user
- State 3: add in equality of hash of password supplied by user and hash of user with login name
- State 3A (alternate expression of state 3): user has authenticated him/herself
- State 4: add in granting of access
/* if assert, enter state 1 */

**location func getpwnam(name) result pwent**

```
{ assert user_password(name, pwent->pw_passwd, pwent->pw_uid); }
```

/* if assert, transition from state 1 to state 2 */

**location func crypt(password,salt) result encryptpwd**

```
{ assert password_entered(encryptpwd); }
```

/* if assert, transition from state 2 to state 3 */

**location func strcmp(s1, s2) result 0**

```
{ assert equals(s1, s2); }
```

/* if in state 3, transition to state 3A */

`password_entered(pwd1) and user_password(name, pwd2, uid) and equal(pwd1, pwd2)`

```
{ assert authenticated(uid); }
```

/* if assert, transition from state 3A to state 4 */

**location func setuid(uid) result 1**

```
{ assert access_acquired(uid); }
```

/* invariant; must always hold */

`authenticated(uid) before access_acquired(uid)`

---

January 8, 2004

David Gilliam, John Powell (JPL), & Matt Bishop (UC Davis)
C Code and States

```c
if (fgets(stdin, uname, sizeof(uname)-1) == NULL)
    return(FAILED);

typedpwd = getpass("Password: ");
if ((pw = getpwnam(uname)) != NULL){
    /* if here, enter state 1 */
    hashtp = crypt(pw->pw->passwd, typedpwd);
    /* enter state 2 */
    if (strcmp(pw->pw->passwd, hashtp) == 0){
        /* if here, enter state 3 and then 3A */
        setuid(pw->pw->uid);
        /* enter state 4 */
        return(SUCCESS);
    }
}
return(FAILED);
```

January 8, 2004

David Gilliam, John Powell (JPL), & Matt Bishop (UC Davis)
Good login

user_password("me", "xyz", 917)
  In state 1
password_entered("xyz")
  In state 2
equals("xyz", "xyz")
  In state 3
authenticated(917)
  In state 3A
access_acquired(917)
  In state 4

Invariant holds

Bad login, but access granted due to programming error

user_password("me", "xyz", 917)
  In state 1
password_entered("abc")
  In state 2
equals("abd", "xyz")
  Fails, so do not enter state 3

access_acquired(917)
  In state 4

Invariant fails as state 3A never entered, as invariant require
Software Security Checklist (SSC)

- Two Phases
  - Phase 1:
    - Provide instrument to integrate security as a formal approach to the software life cycle
    - Requirements Driven
    - Pre-Requirements
      - Understand the Problem and Scope
    - Requirements Gathering and Elicitation
      - Be Aware of Applicable Requirements Documents
      - Provide Trace to External Requirements Docs
SSC (Cont.)

- Requirements Specification
- Design – Reviews
- Code – Inspection & Walkthroughs, Safe Routines, Bounds Checking, Access Control, etc.
- Testing – Property-Based Test, Fault Injection, et.al.
- Maintenance – Regression Test
Phase 2:

- External Release
  - Release Process
  - Areas for Protection:
    - Protect People
    - Protect ITAR and EAR
    - Protect Trade Secrets – Patents
    - Protect Organizational Resources
  - Considerations
    - Insecure Subsystem Calls
    - Embedded IP Addresses or Phone Numbers
Agenda

- Goal
- Problem
- Software Security Assessment Instrument (SSAI)
- Final Notes
Final Notes

- Protection of Resources and Data
- Maintenance Costs Decrease
- Integrates with Other Formal Disciplines
- End-to-End Life Cycle Process

January 8, 2004

David Gilliam, John Powell (JPL), & Matt Bishop (UC Davis)
Final Notes (Cont.)

- Current and Future Uses
  - Integrate with Deep Space Mission Systems (DSMS)
    - SSL Verification
    - Potential to Verify Space Communication Protocol Standard (SCPS) Implementations (i.e., IPN)
  - Formal Verification of Patchlink JAVA client
  - Verification Checklist for External Release of Software
  - Integrate with IT Security Risk Management Tool (Defect Detection and Prevention – DDP) for Software Life Cycle
Future Development Needs

- IT Security Risk Management Process Needed with Risk Mitigation Instruments
  - Extend Defect Detection and Prevention Tool (DDP) to Include Security Template and Risk Mitigations
  - Include Mitigation Tools

- Training and Experts Needed to Provide Tool Assistance

- Extend Modeling Framework to include Model Security Templates for Re-Use

- Extend PBT to Include More Languages (C++, C#, Perl Script, et.al.)
Final Notes (Cont.)

- Maintain Security Assessment Tool (SAT) Site at UC Davis
  - Location for submission and evaluation of Security Assessment Tools
  - Provide Pros and Cons and Uses of Tools with List of Alternatives
David Gilliam
JPL
400 Oak Grove Dr., MS 144-210
Pasadena, CA 91109
Phone: (818) 354-0900 FAX: (818) 393-1377
Email: david.p.gilliam@jpl.nasa.gov

John Powell
MS 125-233
Phone: (818) 393-1377
Email: john.d.powell@jpl.nasa.gov

Website: http://rssr.jpl.nasa.gov/
BAD

Questions?

GOOD

January 8, 2004

David Gilliam, John Powell (JPL), & Matt Bishop (UC Davis)