Vulnerability Assessment and Secure Coding Practices for Middleware

Barton P. Miller
James A. Kupsch
Computer Sciences Department
University of Wisconsin
bart@cs.wisc.edu

Elisa Heymann
Computer Architecture and Operating Systems Department
Universitat Autònoma de Barcelona
elisa@cs.wisc.edu

EPIC Systems
March 11, 2010

Roadmap

› Part 1: Vulnerability Assessment Process
› Part 2: Secure Coding Practices
Key Issues for Security

› Need independent assessment
  - Software engineers have long known that testing groups must be independent of development groups
› Need an assessment process that is NOT based solely on known vulnerabilities
  - Such approaches will not find new types and variations of attacks

Key Issues for Security

› Automated Analysis Tools have Serious Limitations
  - While they help find some local errors, they
    • MISS significant vulnerabilities (false negatives)
    • Produce voluminous reports (false positives)
› Programmers must be security-aware
  - Designing for security and the use of secure practices and standards does not guarantee security
Addressing these Issues

› We must evaluate the security of our code
  - The vulnerabilities are there and we want to find them first
› Assessment isn’t cheap
  - Automated tools create an illusion of security
› You can’t take shortcuts
  - Even if the development team is good at testing, they can’t do an effective assessment of their own code

Addressing these Issues

› First Principles Vulnerability Assessment (FPVA)
  - A strategy that focuses on critical resources
  - A strategy that is not based solely on known vulnerabilities
› We need to integrate assessment and remediation into the software development process
  - We have to be prepared to respond to the vulnerabilities we find
Goal of FPVA

› Understand a software system to focus search for security problems
› Find vulnerabilities
› Make the software more secure

"A vulnerability is a defect or weakness in system security procedures, design, implementation, or internal controls that can be exercised and result in a security breach or violation of security policy."
- Gary McGraw, Software Security

i.e., a bad thing

First Principles Vulnerability Assessment

Step 1: Architectural Analysis
Step 2: Resource Identification
Step 3: Trust & Privilege Analysis
Step 4: Component Evaluation
Step 5: Dissemination of Results
Studied Systems

Condor, University of Wisconsin
Batch queuing workload management system

SRB, SDSC
Storage Resource Broker - data grid

MyProxy, NCSA
Credential Management System

glExec, Nikhef
Identity mapping service

CrossBroker, Universitat Autònoma de Barcelona
Resource Manager for Parallel and Interactive Applications

Gratia Condor Probe, NCSA
Feeds Condor Usage into Gratia Accounting System

Condor Quill, University of Wisconsin

Studied Systems

Wireshark (in progress)
Network Protocol Analyzer

Condor Privilege Separation, University of Wisconsin (in progress)
Restricted Identity Switching Module

VOMS Admin, Instituto Nazionale di Fisica Nucleare (in progress)
Virtual Organization Management Service

Condor Privilege Separation, University of Wisconsin (in progress)
Restricted Identity Switching Module

VOMS Admin, Instituto Nazionale di Fisica Nucleare (in progress)
Virtual Organization Management Service
First Principles Vulnerability Assessment
Understanding the System

Step 1: Architectural Analysis
- Functionality and structure of the system major components (modules, threads, processes), communication channels
- Interactions among components and with users

User Supplied Data
- All attacks ultimately arise from attacker (user) communicated data
- Attack surface: Interfaces available to the attacker
- If not, your system is malware: mere installation causes a security violation
- Important to know where the system gets user supplied data
- What data can users inject into the system
Step 1: Architectural Analysis

› Create a detailed big picture view of the system
› Document and diagram
  - What processes/hosts exist and their function
  - How users interact with them
  - How executables interact with each other

Step 1: Architectural Analysis
External Services Used

› How are external programs used
› External services
  - Database (DBMS, LDAP, DNS, …)
  - Web server
  - Application server
  - Other
› External executables launched
  - Signs in the code: `popen` `system` `exec`*
Step 1: Architectural Analysis
Process Communication Channels

» What exists between...
  - Servers
  - Client and server
  - Client/Server and external programs
    - DBMS
    - DNS
    - LDAP
    - Kerberos
    - File services: NFS AFS ftp http ...
    - Helper applications

» Shows interaction between components

Step 1: Architectural Analysis
Communication Methods

» OS provides a large variety of communication methods
  - Command line
  - Files
  - Creating processes
  - IPC
  - Environment
  - Sockets
  - Signals
  - Directories
  - Symbolic links
  - Pipes
  - FIFO’s or named pipes
  - System V IPC
  - Memory mapped files
Step 1: Architectural Analysis

Command Line

- Null-terminated array of strings passed to a starting process from its parent
- Convention is that `argv[0]` is the path to executable file
- Signs in code
  - C/C++: `argc argv`
  - Perl: `$0 @ARGV`
  - Sh: `$0 $1 $2... $# $@ $*`
  - Csh: `$0 argv`

Sockets

- Creates a communication path
  - processes on same host
  - between hosts using protocols such as TCP/IP
- Can be stream or message based
- Signs in code
  - C/C++: `socket bind connect listen accept socketpair send sendto sendmsg recv recvfrom recvmsg getpeername getsockname setsockopt getsockopt shutdown`
Step 1: Architectural Analysis

IPC

- Inter- and Intra-host communication methods
- Some can pass file descriptors between processes
- Signs in code:
  - Pipes: `pipe`, `mkfifo`
  - SysV Message Q: `msgget` `msgctl` `msgsnd` `msgrcv`
  - SysV Semaphore: `semget` `shmctl` `semop`
  - SysV Shared Mem: `shmget` `shmctl` `shmat` `shmdt`
  - Memory mapped files: `mmap`

Condor

Condor submit host

1. fork

2. schedd

3. submit job ClassAd

4. job ClassAd

5. Negotiator cycle

6. Report match

7. claim host

8. fork

9. establish channel

10. start job

Condor execute host

1. fork

2. machine ClassAd

3. Negotiator cycle

4. Report match

5. Negotiator cycle

6. Report match

7. claim host

8. fork

9. establish channel

10. start job

Stork server host

1. fork

2. condor & root

3. user

OS privileges

- condor & root
- user
Step 1: Architectural Analysis

- SRB
- SRB client process
- SRB server host
- SRB master
- SRB client host
- SRB agent
- MCAT host
- MCAT PostgreSQL

First Principles Vulnerability Assessment
Understanding the System

Step 2: Resource Identification
- Key resources accessed by each component
- Operations allowed on those resources
Step 2: Resource Analysis

- A resource is an object that is useful to a user of the system and is controlled by the system
  - Data
    - files
    - DBMS
    - memory
  - Physical entities
    - Disk space
    - CPU cycles
    - Network bandwidth
    - Attached devices (sensors, controllers)

Step 2: Resource Identification

Documenting Resources

- What resources exist in the system
- What executables-hosts control the resource
- What operations are allowed
- What does an attacker gaining access to the resource imply
Step 2: Resource Identification

Files

› Represented by a path
› File descriptors represent files in program
  - From opening or creating a file
  - Inherited from parent process
› Contents can be data, configuration, executable code, library code, scripts
› Signs in code:
  - C/C++: `open creat fopen dlopen` *stream*

Step 2: Resource Identification

Standard File Descriptors

› Convention is creating process opens file descriptors 0, 1 and 2 for use by the created process to be used as standard in, out, and err
› Functions and libraries often implicitly use these and expect them to be opened
› Signs in code
  - C/C++: `stdin stdout stderr STDIN_FILENO STDOUT_FILENO STDERR_FILENO getchar gets scanf printf vprintf vscanf cin cout cerr`
Step 2: Resource Identification

Directories

› List of named file system objects
› Operations:
  - Get list of entries
  - Create entry
  - Rename entry
  - Delete entry
› Entries have metadata like type, size, and owner
› Signs in code:
  - C/C++: opendir readdir closedir creat open(with O_CREAT) fdopen mkdir mknod symlink link unlink remove rename rmdir

Step 2: Resource Identification

Symbolic Links

› File system object that contains a path (referent)
› When evaluating a path the operating system follows the referent in the link
› Operations:
  - Create symbolic link (can’t modify)
  - Read referent
› Signs in code:
  - C/C++: implicitly in any function taking a path, symlink readlink
Step 2: Resource Identification

(a) Common Resources on All Condor Hosts
- generic Condor daemon
- Condor Binaries & Libraries
- Condor Config
- Operational Data & Run-time Config Files
- Operational Log Files
- OS privileges:
  - condor
  - root
  - user

(b) Unique Condor Checkpoint Server Resources
- ckpt_server
- Checkpoint Directory
- Send and Receive Checkpoints (with Standard Universe Jobs)

(c) Unique Condor Execute Resources
- User Job
- starter
- execute
- Job Execution Directories
- System Call Forwarding and Remove I/O (with Standard Universe Jobs)

(d) Unique Condor Submit Resources
- shadow
- User’s Files

Step 2: Resource Identification

SRB
- OS privileges:
  - srb
  - postgresql
  - user

SRB server host
- SRB master & agents
- SRB config files
- SRB data Store 1
- SRB data Store 2
- SRB tape storage

SRB client host
- SRB client process
- client home dir & config files

MCAT host
- MCAT PostgreSQL
- db config files
- db data store
First Principles Vulnerability Assessment
Understanding the System

Step 3: Trust & Privilege Analysis
- How components are protected and who can access them
- Privilege level at which each component runs
- Trust delegation

Step 3: Trust & Privilege Analysis

› Process Attributes
- What user/group is the process started as
- Is the process setuid/setgid
- Any unusual process attributes
- uid/gid switching
Step 3: Trust & Privilege Analysis

- Privilege is the authorization for a user to perform an operation on a resource
  - What privileges exist in the system
  - Do they map appropriately to operations on resources
  - Are they fine grained enough
  - How are they enforced

Step 3: Trust & Privilege Analysis

- Authentication
  - Is it performed correctly and securely
  - If an attacker can authenticate as another user they gain their privileges
Step 3: Trust & Privilege Analysis
External Privilege Systems

› System used: OS, DBMS, ...
› Accounts and privileges used
› Purpose of each account
› Does the program use external privileges to enforce its privilege model
› Are minimal privileges used
› Use of root or admin accounts require special attention

Step 3: Trust & Privilege Analysis
Trust

› An executable trusts another when
  - It relies on a behavior in the other
  - Doesn't or can't verify the behavior
› Implicit trust
  - The operating system
  - Process with root privilege on the same host
    • they can do anything
  - Processes with same uid on the same host
    • they can do anything to each other
  - All the code in your executable including libraries
Step 3: Trust & Privilege Analysis
Bad trust

› Not validating data from another trust domain for proper form (form, length, range)
› Bad assumptions
  - User supplied data is in proper form
  - Data passed through client is unchanged

More Bad Trust

› Bad assumptions (cont.)
  - Client validated data
    • Client can be rewritten or replaced
    • Good to validate on the client, but server validation is required
› Not validating data from trusted processes
  - Allows an attack to spread
  - Not defense in depth
First Principles Vulnerability Assessment
Search for Vulnerabilities

Step 4: Component Evaluation
- Examine critical components in depth
- Guide search using:
  - Diagrams from steps 1-3
  - Knowledge of vulnerabilities
- Helped by Automated scanning tools (!)

Step 4: Component Evaluation
Categories of Vulnerabilities

- **Design Flaws**
  - Problems inherent in the design
  - Hard to automate discovery
- **Implementation Bugs**
  - Improper use of the programming language, or of a library API
  - Localized in the code
- **Operational vulnerabilities**
  - Configuration or environment
- **Social Engineering**
  - Valid users tricked into attacking
Step 4: Component Evaluation  
Many Types of Vulnerabilities

- Buffer overflows
- Injection attacks
  - Command injection (in a shell)
  - Format string attacks (in printf/scanf)
- SQL injection
- Cross-site scripting or XSS (in HTML)
- Directory traversal
- Integer vulnerabilities
- Race conditions
- Not properly dropping privilege
- Insecure permissions
- Denial of service
- Information leaks
- Lack of integrity checks
- Lack of authentication
- Lack of authorization

Step 4: Component Evaluation  
Focusing the Search

- It’s impossible to completely analyze a system for vulnerabilities
- From critical resources and try to think of ways an attack can be realized
- From vulnerabilities can occur in the code to resources
- Look for similar problems to prior security problems
Step 4: Component Evaluation
Process Configuration

› How is an executable configured
  - Configuration file
  - Hard coded
  - Other

› What can be configured
  - How does it affect the application
  - Often reveals functional and architectural information

Step 4: Component Evaluation
Communication Methods

› OS provides a large variety of communication methods
  - Command line
  - Files
  - Creating processes
  - IPC
    • Pipes
    • FIFO's or named pipes
    • System V IPC
    • Memory mapped files
  - Environment
  - Sockets
  - Signals
  - Directories
  - Symbolic links
First Principles Vulnerability Assessment
Taking Actions

Step 5: Dissemination of Results
- Report vulnerabilities
- Interaction with developers
- Disclosure of vulnerabilities

Step 5: Dissemination of Results
Vulnerability Report

- One report per vulnerability
- Provide enough information for developers to reproduce and suggest mitigations
- Written so that a few sections can be removed and the abstracted report is still useful to users without revealing too much information to easily create an attack.
Step 5: Dissemination of Results

Vulnerability Report Items

- Summary
- Affected version(s) and platform
- Fixed version(s)
- Availability - is it known or being exploited
- Access required - what type of access does an attacker require: local/remote host? Authenticated? Special privileges?
- Effort required (low/med/high) - what type of skill and what is the probability of success
Step 5: Dissemination of Results

Vulnerability Report Items

› **Impact/Consequences** (low/med/high) - how does it affect the system: minor information leak is low, gaining root access on the host is high

› **Only in full report**
  - **Full details** - full description of vulnerability and how to exploit it
  - **Cause** - root problem that allows it
  - **Proposed fix** - proposal to eliminate problem
  - **Actual fix** - how it was fixed

Step 5: Dissemination of Results

Vulnerability Disclosure Process

› Disclose vulnerability reports to developers
› Allow developers to mitigate problems in a release

Now here’s the really hard part:
› Publish abstract disclosures in cooperation with developers. When?
› Publish full disclosures in cooperation with developers. When?
Summary of Results
First Principles Vulnerability Assessment

Technique has been extremely successful
- found critical problems
- helped groups redesign software
- changed their development practices and release cycle management

First Principles Vulnerability Assessment (FPVA) white paper:

Our Work -- Summary

Assess: We continue to assess new software systems

Train: We present tutorials and white papers, and continue to develop new educational materials

Research: Our results provide the foundation for new research to make FPVA less labor-intensive and improve quality of automated code analysis