COMS E6998-9: Software Security and Exploitation

Lecture 8: Fail Secure; DoS Prevention; Evaluating Components for Security

Hugh Thompson, Ph.D.
hthompson@cs.columbia.edu
Failing Securely and Denial of Service Prevention
Overview

• We’ll begin this section by looking at three of the most important concepts in computer security:
  – Defense in depth
  – Compartmentalization
  – Principle of least privilege

• Applying these three principles can greatly reduce (and sometimes eliminate) the impact of a vulnerability
Defense in depth - intro
Compartmentalization

- Closely related to defense in depth is compartmentalization
- The idea is to separate processes, applications and functionality so that a compromise of one system doesn’t mean a compromise of them all
- Partitioning can happen at the
  - Network level: Firewalls, routers, etc.
  - Operating system level: ACLs, containers, VMs, etc.
  - Application level: forked processes, exception handling, etc.
Least Privilege

• An application or process should only be given the permissions it needs to complete a task; no more and no less

• Some immediate consequences:
  – Big applications that have to do certain tasks with elevated privileges (root, administrator) should be broken up into components
  – Many of the applications that run as administrator have a very small amount of code that needs that permission
  – You shouldn’t just assign a high permission level to make something work; understand what permissions are needed and why
Security and Exception Handling

• One of the biggest software security sins is not planning for failure

• Environmental failures, bugs and exceptions will occur: it’s how they’re handled that makes the difference

• When writing exception handlers consider:
  – What data needs to be saved (what was the application doing)
  – What data needs to be purged
  – Degrade functionality and security commensurately (eg. Don’t just stop logging if you get a disk full message)
  – Calls to APIs can fail so check return values

© Hugh Thompson 2010
Error Handling Tips

- Have a clear understanding of the API you are calling
  - Does it set the error code variable to give more information? *(errno, SetLastError() )*
  - Does it return an error value?
- Carefully write error handlers
  - Use try/catch to handle exceptions
  - Return values can be handled with if statements
  - Where relevant use information like GetLastError() inside your error handling code to better understand the error**

**WARNING:** Make sure your API actually sets an error code before making decisions based on it

- Test your error handler
  - Most error handlers are woefully under-tested
  - Use fault injection techniques to execute them
Beware of disclosing too much information

• The natural developer tendency is to be as detailed as possible in an error message for debugging
• Detail can give attackers a HUGE advantage
• Common offenses:
  – ODBC Errors – Disclose SQL query strings, settings, implementation details, etc.
  – Login failure – Disclose whether the Username or Password was wrong
• Remember, generic error messages to a user is good
  – Can also include an incident number where details are stored in a secure log
Patchability

• Vulnerabilities are inevitable so the system must be designed so that it is patchable with minimal user impact

• Consider how patches:
  – Will be delivered to users
  – Can be applied to systems
  – Will impact the system (reboot, re-image, downtime, etc.)
  – Can be distributed simply (consider patch size)
  – Can be distributed if the system is being actively attacked
  – Can be authenticated as from the vendor
Race Conditions

• Also called Time Of Check To Time Of Use (TOCTTOU) vulnerabilities

• Happens because we assume that a sequence of actions are continuous and that no resources are modified in the middle

© Hugh Thompson 2010
Understanding DoS weaknesses

• Motivations of attackers may be simply to interrupt service
• Security is also about ensuring that a user cannot deny other users legitimate access to functionality and data
• Some of the most difficult attacks to prevent against
• May include starving: memory, disk space, bandwidth, CPU
• Consider carefully:
  – How the application receives data
  – If safeguards (such as login attempt lockout) can be leveraged by an attacker
  – That point loads can and will occur
  – How the system should respond under heavy load
Throttling for DoS prevention

• The reason that many actual outages occur is poor failure planning
• One technique to deal with DoS attacks is throttling
• The idea is to degrade service selectively under attack as opposed to cutting it off
• Remember: Attackers may use a reactive defense against you
Authentication and Authorization
To begin, a few definitions

• *Identification* is the act of professing that you are something or someone

• *Authentication* is the act of proving authenticity (or proving identity)

• *Authorization* is associating privileges with that one entity has on another entity (such as privileges a user has over a file)
Authentication

• There are several ways to authenticate:
  – Something you are
  – Something you have
  – Something you know
  – *Something you do (often considered to be a subset of *something you are*)

• Using any \( n \) of the above is described as \( n \)-factor authentication
  – E.g. Using an ATM card with a PIN
Authorization - Types of Access Control

- Mandatory Access Control – Access control based on the sensitivity of the object
- Discretionary Access Control – Access control based on the discretion of the object owner
- Role-Based Access Control – Access control based on an entities role
Authentication Technologies

- Basic authentication
- Digest authentication
- X.509 certificates
- Kerberos
- SSL/TLS
- *LDAP and Active Directory
OS Security Models - Windows

• Uses Access Control Lists (ACLs) to protect resources by restricting what can be done with them
• Discretionary ACLs (DACLs) define what can be done to objects
• System ACLs (SACLs) determine what to log when a resource is accessed
• DACLs and SACLs are composed of a list of zero or more Access Control Entries (ACE)
Windows Warnings

• Analyze business logic to set ACLs appropriately
  – Formalize access control requirements in a specification
  – Ensure the resources are deployed, implemented, and tested to meet specifications

• Never use a NULL DACL
  – This defaults to everyone, full control

• Be careful with ACE order – it matters
  – Specify deny permissions before allow

• Be careful with ACL inheritance
OS Security Models – Linux/Unix

• Permissions are defined for: USER (u), GROUP (g) and OTHERS (o)
• Permissions can be set for: READ (r), WRITE (w) or EXECUTE (x)
• Some flavors of Linux/Unix support more granular ACLs
• Setuid allows users to run executables with temporarily elevated privileges in order to perform specific tasks (e.g. passwd)
Unix/Linux tips and warnings

• Chroot() system call enables you to “change the filesystem root” to a specific directory and confine an application

• Beware the symbolic link – many UNIX exploits take advantage of insecure temporary file creation
  – Result: Bait and Switch
3rd Party and OS Component Security
Understanding the “Weakest Link” Principle

- Software is only as secure as its weakest component
- Whenever something is added to a system we inherit both its utility and its vulnerabilities
- A heroic security effort on one component can be negated by adding a weaker component
- Need to focus on addressing risk broadly

© Hugh Thompson 2010
Inherited risks from 3rd party components

• Several risk come from external components:
  – Coding flaws
  – Mismatch in handling data (not sanitizing memory, etc.)
  – Updates and patches that could become a product liability
  – A broader audience of folks looking for vulnerabilities
  – Licensing issues
  – Maintenance issues
Evaluation: Security Quality versus Quality of Security Service

• When we evaluate a component for security, we must also look carefully at its supplier.
• Need to ensure that the software is at a comparable level in terms of security quality.
• Need to ensure that lifecycle practices include security too because you are essentially signing up for a “service.”
Questions to start asking vendors or component providers about security

• Do you have a dedicated team to assess and respond to security vulnerability reports in your products?
• What is your vulnerability response process?
• What process improvements have you made as a result of vulnerabilities reported in your software?
• What is your patch release strategy?
• What training does your development and testing organizations receive on security?
• What level is this product certified to: EAL, CC, ...?
Some Questions about Security Showing to consider in RFPs

- How is security involved in your SDLC?
- What percentage of your dev and test team is focused on security?
- Does your company monitor the latest attack trends in the underground community and consider how those trends may affect your software?
- Do you offer organizations secure implementation guidance?
- Do you patch all currently supported and vulnerable versions of your applications / platforms at the same time?
- What are the terms and period of your security support agreement?
- Does your development team perform regular audits
Creating a safety net around other people’s code

- Be suspicious of data received
- Remember that an external component may not handle data the same way you do
- Updates may change the way the component works or the bounds of data that is returned
  - Particularly a concern for web services
- Enforce assumptions about data in your own code – especially the 3\textsuperscript{rd} party code is beyond your control
- Remember that external API calls can fail and create error handlers accordingly