

Secure Design Guidelines

John Slankas
CSC 515

The Protection of Information in Computer Systems

JEROME H. SALTZER, SENIOR MEMBER, IEEE, AND MICHAEL D. SCHROEDER, MEMBER, IEEE

Invited Paper

Abstract—This tutorial paper explores the mechanics of protecting computer-stored information from unauthorized use or modification. It concentrates on those architectural structures—whether hardware or software—that are necessary to support information protection. The paper develops in three main sections. Section I describes desired functions, design principles, and examples of elementary protection and authentication mechanisms. Any reader familiar with computers should find the first section to be reasonably accessible. Section II requires some familiarity with descriptor-based computer architecture. It examines in depth the principles of modern protection architectures and the relation between capability systems and access control list systems, and ends with a brief analysis of protected subsystems and protected objects. The reader who is dismayed by either the prerequisites or the level of detail in the second section may wish to skip to Section III, which reviews the state of the art and current research projects and provides suggestions for further reading.

GLOSSARY

THE FOLLOWING glossary provides, for reference, brief definitions for several terms as used in this paper in the context of protecting information in computers.

Authorize

To grant a principal access to certain information.

Capability

In a computer system, an unforgeable ticket, which when presented can be taken as incontestable proof that the presenter is authorized to have access to the object named in the ticket.

Certify

To check the accuracy, correctness, and completeness of a security or protection mechanism.

Complete isolation

A protection system that separates principals into compartments between which no flow of information or control is possible.

Confinement

Allowing a borrowed program to have access to data, while ensuring that the program cannot release the information.

Descriptor

A protected value which is (or leads to) the physical address of some protected object.

1. Securing the Weakest Link

- Attackers are more likely to attack a weak spot in a software system than to penetrate a heavily fortified component.



– Attackers will:

- Target the endpoints of communication (e.g., servers) rather than try to crack the cryptographic algorithm.
- Try to break through the applications visible through the firewall rather than crack the firewall.

– “Security is only as good as its weakest link, and people are the weakest link in the chain.”

- Bruce Schneier, *Secrets and Lies*, 2000

- Insider threat
- Social engineering



A Phishing Attack

From: "Nancy Betts" <nancy_betts@ncsu.edu>
Date: November 21, 2014 at 3:38:24 PM EST
To: prtelang@ncsu.edu
Subject: Library Account Access
Reply-To: nancy_betts@ncsu.edu

Dear User,

Your access to your library account is expiring soon and it won't be accessible for you. You must reactivate your account in order to continue to have access to this service. For this purpose, click the web address below or copy and paste it into your web browser. After logging in, your access is reactivated and you will be redirected to your library profile.

https://myaccount.lib.ncsu.edu/login_tGg0pfpya90JA4fwnDaxZtGg0pHdtHdtHTD8yNceuh3IPfYNI9Te8HFY90wCnYJHdtHdtkiya90PsoYiH8nYZHTD8ymgo4D8yNe8HPsmHTqo9Ob1qJSMjMm4D8yNe8HFYnYZtGg0pfp90layn066dy1qJSMjMm4D8yNe2ztViybT/

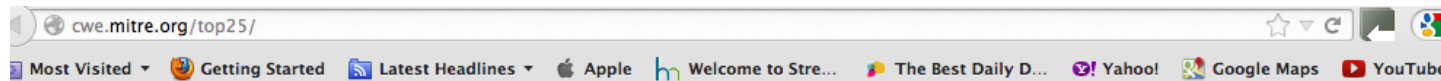
If you are not able to login, please contact Access Services Manager at nancy_betts@ncsu.edu.

Sincerely,

Nancy Betts
Access Services Manager
Access & Delivery Services
NCSU Libraries
North Carolina State University
(919) 515-4967
nancy_betts@ncsu.edu

Weakest Link: Defense

- Use common hacker tools on your system (e.g. nmap, wireshark, webscarab, zap)
- Be familiar with annual SANS Top 25 and OWASP Top 10 most dangerous programming list
- Do risk analysis (e.g. threat modeling) to analyze “biggest bang for the buck” for your security resources.



ome > CWE/SANS Top 25 2011

CWE List

Full Dictionary View

Development View

Research View

Reports

About

2011 CWE/SANS Top 25 Most Dangerous Software Errors

Copyright © 2011

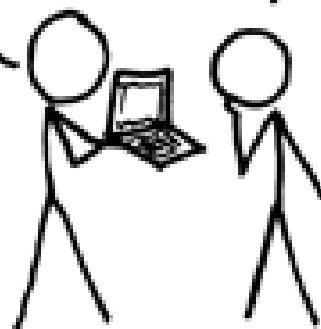
<http://cwe.mitre.org/top25/>

A CRYPTO NERD'S
IMAGINATION:

HIS LAPTOP'S ENCRYPTED.
LET'S BUILD A MILLION-DOLLAR
CLUSTER TO CRACK IT.

BLAST! OUR
EVIL PLAN
IS FOILED!

NO GOOD! IT'S
4096-BIT RSA!



WHAT WOULD
ACTUALLY HAPPEN:

HIS LAPTOP'S ENCRYPTED.
DRUG HIM AND HIT HIM WITH
THIS \$5 WRENCH UNTIL
HE TELLS US THE PASSWORD.

GOT IT.




Other Weak Links

- 3rd Party Vendors
 - 2013 Target Breach: HVAC Company
- Weak Passwords
- Implementation Vulnerabilities
 - Is the cryptographic function used correctly?
 - Software configured correctly?

2. Least Privilege

- Follow the “Need to know” or “Need to use” rule
- A subject (user/other system) should be given only those privileges it needs to complete its task.
- The function (or role) of a user (as opposed its identity) should control the assignment of rights.
- Two desirable effects:
 - security impact of a security incident or corruption of the component will be minimized; and
 - the security analysis of the component will be simplified (that is, if you have to do forensic analysis on a breach, the analysis will be easier).

Execution with Unnecessary Privileges



```
<?php
$host = 'localhost';
$userID = 'root';
$password = 'password';
$db = mysql_connect($host, $userID, $password) or die ('Error connecting to mysql');
$name = 'testdatabase';
mysql_select_db($name);
$sql="SELECT * FROM theTable";
$result=mysql_query($sql);
?>
```

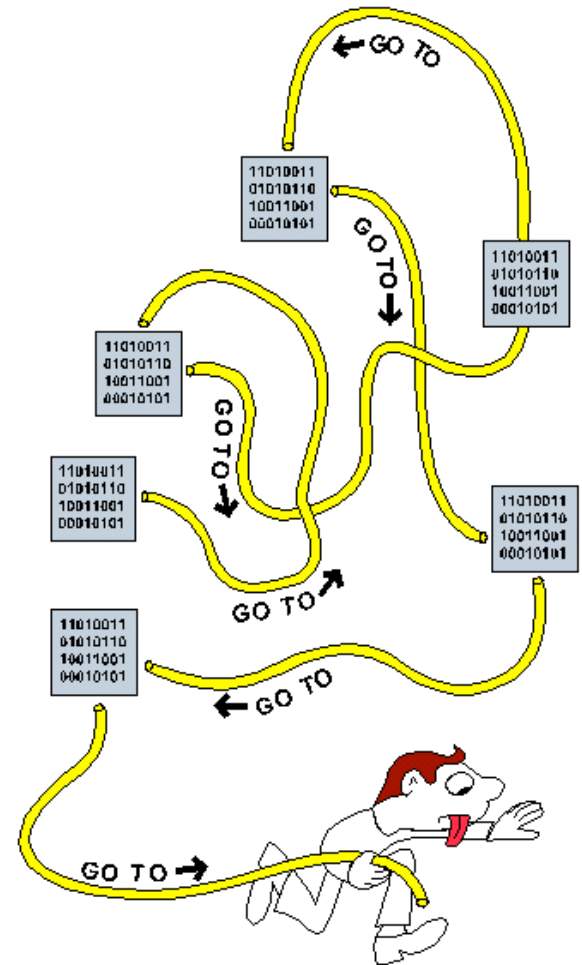
3. Separation of Privilege

- System should not grant permission based upon a single condition.
- Sensitive operations should require the cooperation of more than one check.
 - Two signatures on a large check.
 - UNIX users cannot change from their account to the root account unless:
 - User knows the root password
 - User is in the “wheel” group (group with GID 0)

4. Economy of Mechanism

- Security mechanisms (a.k.a security checks, security functionality) should be as simple as possible.
 - KISS (Keep it Simple and Small)
- Complex interactions make verifying security of systems more difficult.
- Isolate, consolidate, and minimize security controls.

From Computer Desktop Encyclopedia
© 1998 The Computer Language Co. Inc.

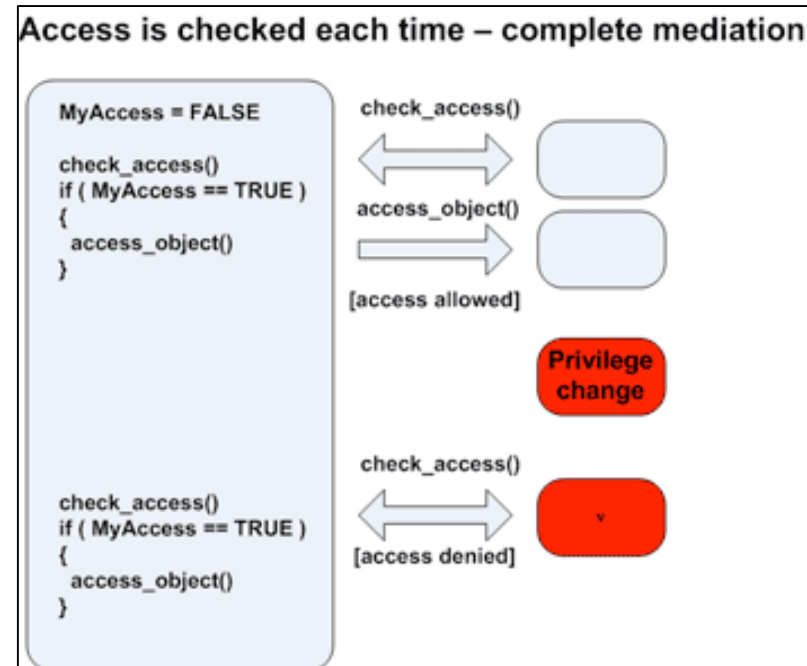


5. Least Common Mechanism

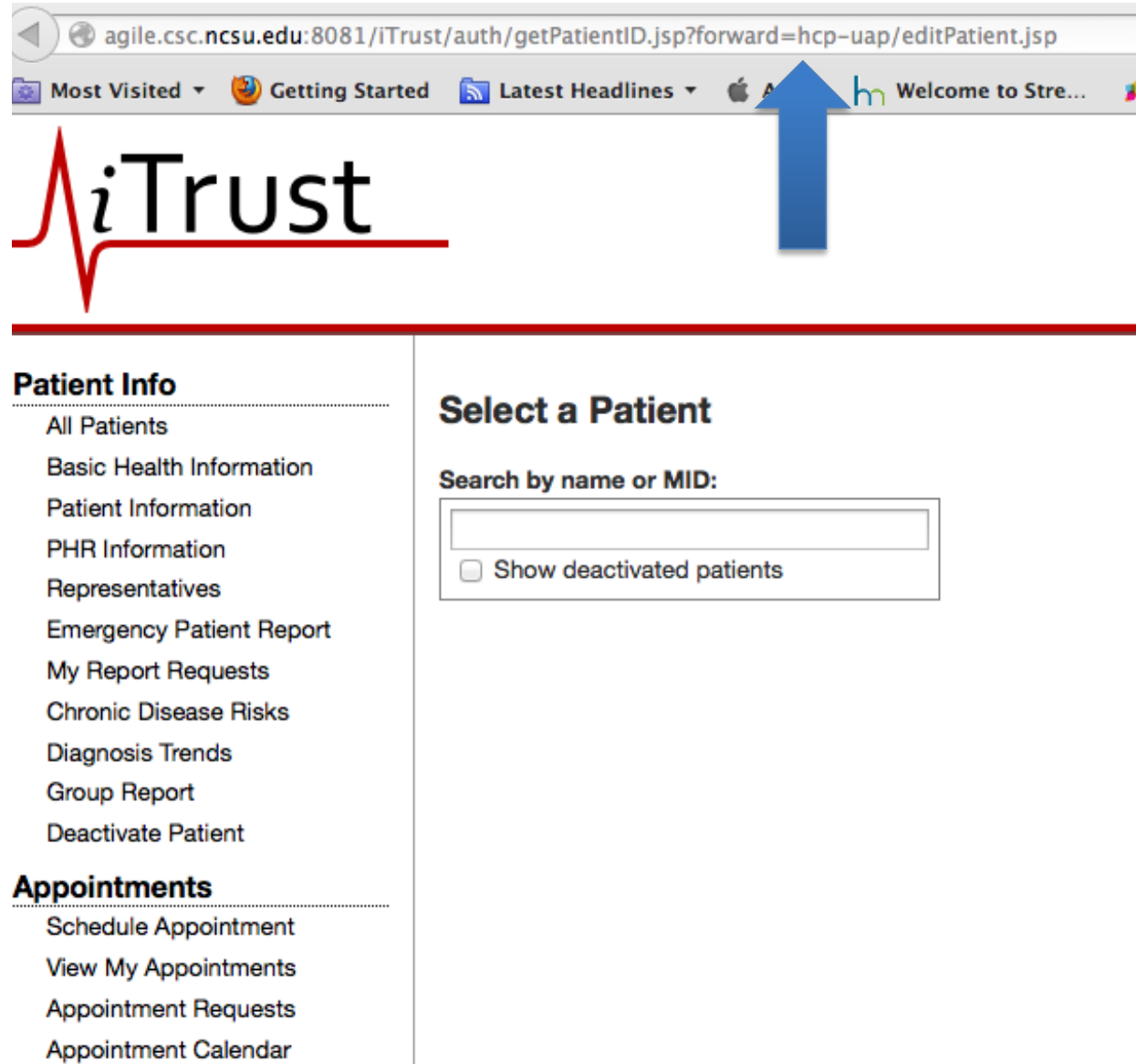
- Mechanisms used to access different resources should not be shared.
- If multiple components in the system require the same function or mechanism, the function or mechanism should be factored into a single mechanism that can be used by all of them.
- Separate machines, separate networks, virtual machines can help fulfill this principle and avoid cross-contamination.

6. Complete Mediation

- All accesses to entities are checked to ensure they are allowed, irrespective of who is accessing what. Check should ensure attempted access do not violate security properties.
- Caching permissions can increase the performance of a system, but at the cost of allowing secured objects to be accessed.
- Easier to implement and evaluate if principle of economy of mechanism has been followed.



Important complete mediation



agile.csc.ncsu.edu:8081/iTrust/auth/getPatientID.jsp?forward=hcp-uap/editPatient.jsp

Most Visited Getting Started Latest Headlines Apple App Store h Welcome to Stre...

iTrust

Patient Info

- All Patients
- Basic Health Information
- Patient Information
- PHR Information
- Representatives
- Emergency Patient Report
- My Report Requests
- Chronic Disease Risks
- Diagnosis Trends
- Group Report
- Deactivate Patient

Select a Patient

Search by name or MID:

☐ Show deactivated patients

Appointments

- Schedule Appointment
- View My Appointments
- Appointment Requests
- Appointment Calendar

7. Secure by Default

- Assumptions:
 - Users are likely not educated enough to know about security.
 - Users may not think they care about security (until there is a problem)
- A system's default settings should be as secure as possible. For example:
 - No open network ports
 - No default passwords (e.g. to the database)
- All security functionality should be enabled by default
- All optional features which entail any security risk should be disabled by default.
- Operations that reduce security should be relatively hidden (e.g., in an “advanced” preference pane) and should make the risks readily apparent.

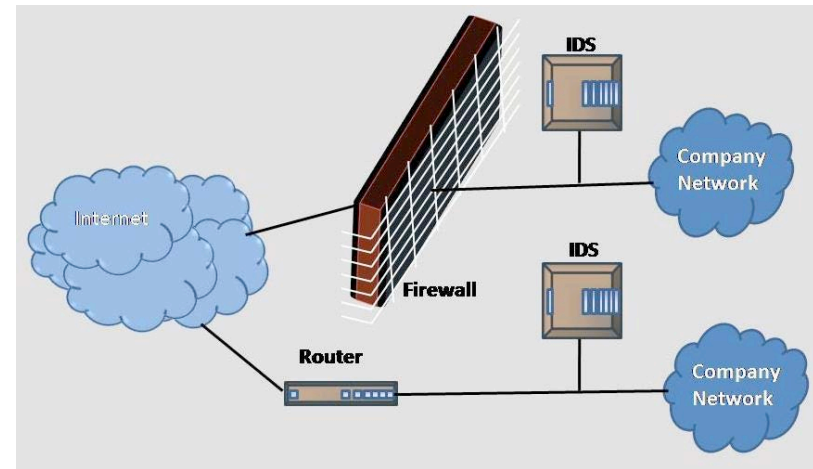


Users



8. Detect Intrusions

- Fact: Attackers will try to get in.
- You need to build in to your system:
 - the capability to log security-relevant events
 - the procedures to ensure the logs are proactively monitored regularly and protected
 - procedures to properly respond to an intrusion once detected



9. Don't reinvent the wheel



Encryption

Hibernate

Input Validation Frameworks

10. Don't allow modification or access without a trace

- Users should not be able to add, edit or delete data without the action being logged.
- Non-repudiation: assurance that someone cannot deny an action they took.
- Logging can be used for forensics after a breach has occurred.
- But proactive log analysis should be performed to identify potentially anomalous access (e.g. a human resource employee looking at the salaries of too many peers one day).
- Don't log sensitive data (social security numbers, credit card numbers) or the log file will be a source of sensitive data.
- Log files should be read only with restrictive access and backed up often.
- Common problem: Log files are for debugging only.

11. Quiet Your Error Messages

- Attackers can cause system errors intentionally to gather information about the system.
- Error messages should be minimalistic, without giving details on the failure.

Summary

- Knowledge of these principles is important.
- Build these principles into your architecture, design, and implementation
- Technically, tacking on security after an attack is difficult
- A “penetrate and patch” strategy for managing security is costly and damaging to your reputation.



The Rugged Manifesto

I am rugged and, more importantly, my code is rugged.

I recognize that software has become a foundation of our modern world.

I recognize the awesome responsibility that comes with this foundational role.

I recognize that my code will be used in ways I cannot anticipate, in ways it was not designed, and for longer than it was ever intended.

I recognize that my code will be attacked by talented and persistent adversaries who threaten our physical, economic and national security.

I recognize these things – and I choose to be rugged.

I am rugged because I refuse to be a source of vulnerability or weakness.

I am rugged because I assure my code will support its mission.

I am rugged because my code can face these challenges and persist in spite of them.

I am rugged, not because it is easy, but because it is necessary and I am up for the challenge.