

Design Category:
Trust with Reluctance

Trust with Reluctance (TwR)

- **Whole system security** depends on the **secure operation of its parts**
 - These parts are **trusted**
- So: **Improve security by reducing the need trust**
 - By using a **better design**
 - By using a **better implementation process**
 - By **not making unnecessary assumptions**
 - If you use third party code, how do you know what it does?
 - If you are not a crypto expert, why do you think you can design/ implement your own crypto algorithm?
- **Categories:** Prevention and mitigation

TwR: Small TCB

- Keep the **TCB small** (and simple) to **reduce overall susceptibility to compromise**
 - The trusted computing base (TCB) comprises the system components that *must* work correctly to ensure security
 - **Category:** Prevention
- **Example: Operating system kernels**
 - Kernels enforce security policies, but are often millions of lines of code
 - Compromise in a device driver compromises security overall
 - Better: Minimize size of kernel to reduce trusted components
 - Device drivers moved outside of kernel in micro-kernel designs

Failure: Large TCB

- **Security software** is part of the TCB
- But as it grows in size and complexity, it becomes vulnerable itself, and can be bypassed



Additional security layers often create vulnerabilities...

October 2010 vulnerability watchlist

Vulnerability Title	Fix Avail?	Date Added
XXXXXXXXXXXX Local Privilege Escalation Vulnerability	No	8/25/2010
XXXXXXXXXXXX Denial of Service Vulnerability	Yes	8/24/2010
XXXXXXXXXXXX Buffer Overflow Vulnerability	No	8/20/2010
XXXXXXXXXXXX Sanitization Bypass Weakness	No	8/18/2010
XXXXXXXXXXXX Security Bypass Vulnerability	No	8/17/2010
XXXXXXXXXXXX Multiple Security Vulnerabilities	Yes	8/16/2010
XXXXXXXXXXXX Remote Code Execution Vulnerability	No	8/16/2010
XXXXXXXXXXXX Use-After-Free Memory Corruption Vulnerability	No	8/12/2010
XXXXXXXXXXXX Remote Code Execution Vulnerability	No	8/10/2010
XXXXXXXXXXXX Multiple Buffer Overflow Vulnerabilities	No	8/10/2010
XXXXXXXXXXXX Stack Buffer Overflow Vulnerability	Yes	8/10/2010
XXXXXXXXXXXX Security-Bypass Vulnerability	No	8/10/2010
XXXXXXXXXXXX Multiple Security Vulnerabilities	No	8/10/2010
XXXXXXXXXXXX Buffer Overflow Vulnerability	No	7/29/2010
XXXXXXXXXXXX Remote Privilege Escalation Vulnerability	No	7/28/2010
XXXXXXXXXXXX Cross Site Request Forgery Vulnerability	No	7/26/2010
XXXXXXXXXXXX Multiple Denial Of Service Vulnerabilities	No	7/22/2010

6 of the vulnerabilities are in security software

Color Code Key: Vendor Replied – Fix in development Awaiting Vendor Reply/Confirmation Awaiting CC/S/A use validation

Approved for Public Release, Distribution Unlimited

<http://www.darpa.mil/WorkArea/DownloadAsset.aspx?id=2147484449>

TwR: Least Privilege

- Don't give a part of the system more privileges than it needs to do its job ("*need to know*")
 - **Category:** Mitigation
- **Example:** Attenuate delegations
 - Mail program delegates to editor for authoring mails
 - `vi`, `emacs`
 - But many editors permit escaping to a command shell to run arbitrary programs: too much privilege!
 - Better Design: Use a restricted editor (`pico`)

Lesson: Trust is Transitive

- **If you trust something, you trust what it trusts**
 - *This trust can be misplaced*
- **Previous e-mail client example**
 - Mailer delegates to an arbitrary editor
 - The editor permits running arbitrary code
 - Hence the mailer permits running arbitrary code

Rule: Input validation

- Input validation is a **kind of least privilege**
 - You are **trusting a subsystem** only **under certain circumstances**
 - *Validate that those circumstances hold*
- Several **examples** so far:
 - Trust a given function *if* the range of its parameters is limited (e.g., within the length of a buffer)
 - Trust a client form field *if* it contains no `<script>` tags (and other code-interpretable strings)
 - Trust a YAML-encoded string *if* it contains no code

TwR: Promote Privacy

- A good overall system goal is to **restrict flow of sensitive data** as much as possible
 - Doing so promotes privacy by *reducing trust/privilege*
 - **Category:** Mitigation
- **Example:** A student admission system receives (sensitive) letters of recommendation as PDF files
 - A typical design would allow reviewers to download these files for viewing on their local computers
 - But then compromise of these computers leaks private information
 - Better: PDFs only viewable in browser; no data downloaded to client machine.

TwR: Compartmentalization

- **Isolate a system component** in a compartment, or **sandbox**, reducing its privilege by making certain interactions impossible
 - **Category:** Prevention and Mitigation
- **Example:** Disconnect student records database from the Internet
 - Grant access only be direct terminals
- **Example:** **Seccomp** system call in Linux
 - Enables compartments for untrusted code

SecComp

- Linux system call enabled since 2.6.12 (2005)
 - Affected process can subsequently **only perform read, write, exit, and sigreturn system calls**
 - No support for `open` call: Can only use already-open file descriptors
 - **Isolates a process by limiting possible interactions**
- Follow-on work produced **seccomp-bpf**
 - **Limit process to policy-specific set of system calls**, subject to a policy handled by the kernel
 - Policy akin to *Berkeley Packet Filters (BPF)*
 - Used by *Chrome*, *OpenSSH*, *vsftpd*, and others

Idea: Isolate Flash Player

- Receive .swf code, save it
- Call `fork` to create a new process
- In the new process, open the file
- Call `exec` to run Flash player
- Call `seccomp-bpf` to compartmentalize

