May 24: Confinement

- Confinement, non-VM isolation
  - Program modification
  - Covert channels
Compiling

• Compiler enforces or validates constraints
  – Type-safe language enforces them
  – Certifying compiler validates them
Type Safety

• Java is type-safe
  – Compiler enforces correct usage of types

• C is not type-safe
  – Need to add semantics to make it safe

• Example: CCured imposes type safety on C
  – Adds code to C programs so pointers point to 0 or objects of right type
  – Handles dynamic pointers, too
  – Impacts performance
Certifying Compiler

- Generates proof that program satisfies specific security properties
  - Before execution, proof is validated
- Example: Touchstone validates type-safe subset of C
  - Checks all array references
Touchstone

- Analyzes functions, annotating code with loop invariants, preconditions, postconditions
- It then generates validation code
  - Predicate for each function holds iff postconditions hold
- Theorem prover verifies proof automatically
  - Uses inference rules about array bounds
- Performance impact of 30% to 150% on standard C benchmarks
Loading

• Load libraries that apply confinement constraints
  – Sandboxing that is embedded in process rather than a separate process

• Aurasium (Android) prevents apps exfiltrating sensitive data
  – Two parts: tool, modified libraries
Aurasium

- Tool inserts code to enforce given policies when app uses Android resources
  - Like SMS messaging
- Modified standard C libraries determine if system call should be blocked based on policy
- Problem: most apps signed
  - Verify signature, then modify app and resign with Aurasium’s own certificate
- On test, re0packed over 99% of apps known to be malicious; negligible performance impact
Sandboxes, VMs, and TCB

• Sandboxes, VMs part of trusted computing bases
  – Failure: less protection than security officers, users believe
  – “False sense of security”

• Must ensure confinement mechanism correctly implements desired security policy
Covert Channels

- Shared resources as communication paths
- *Covert storage channel* uses attribute of shared resource
  - Disk space, message size, etc.
- *Covert timing channel* uses temporal or ordering relationship among accesses to shared resource
  - Regulating CPU usage, order of reads on disk
Example Storage Channel

- Processes $p$, $q$ not allowed to communicate
  - But they share a file system!
- Communications protocol:
  - $p$ sends a bit by creating a file called 0 or 1, then a second file called $send$
    - $p$ waits until $send$ is deleted before repeating to send another bit
  - $q$ waits until file $send$ exists, then looks for file 0 or 1; whichever exists is the bit
    - $q$ then deletes 0, 1, and $send$ and waits until $send$ is recreated before repeating to read another bit
Example Timing Channel

• System has two VMs
  – Sending machine $S$, receiving machine $R$

• To send:
  – For 0, $S$ immediately relinquishes CPU
    • For example, run a process that instantly blocks
  – For 1, $S$ uses full quantum
    • For example, run a CPU-intensive process

• $R$ measures how quickly it gets CPU
  – Uses real-time clock to measure intervals between access to shared resource (CPU)
Example Covert Channel

- Uses ordering of events; does not use clock
- Two VMs sharing disk cylinders 100 to 200
  - SCAN algorithm schedules disk accesses
  - One VM is High (H), other is Low (L)
- Idea: L will issue requests for blocks on cylinders 139 and 161 to be read
  - If read as 139, then 161, it’s a 1 bit
  - If read as 161, then 139, it’s a 0 bit
How It Works

- \( L \) issues read for data on cylinder 150
  - Relinquishes CPU when done; arm now at 150
- \( H \) runs, issues read for data on cylinder 140
  - Relinquishes CPU when done; arm now at 140
- \( L \) runs, issues read for data on cylinders 139 and 161
  - Due to SCAN, reads 139 first, then 161
  - This corresponds to a 1
- To send a 0, \( H \) would have issued read for data on cylinder 160
Analysis

• Timing or storage?
  – Usual definition ⇒ storage (no timer, clock)

• Modify example to include timer
  – $L$ uses this to determine how long requests take to complete
  – Time to seek to 139 < time to seek to 161 ⇒ 1; otherwise, 0

• Channel works same way
  – Suggests it’s a timing channel; hence our definition
Noisy vs. Noiseless

- **Noiseless**: covert channel uses resource available only to sender, receiver
- **Noisy**: covert channel uses resource available to others as well as to sender, receiver
  - Idea is that others can contribute extraneous information that receiver must filter out to “read” sender’s communication
Key Properties

- **Existence**: the covert channel can be used to send/receive information
- **Bandwidth**: the rate at which information can be sent along the channel
- **Goal of analysis**: establish these properties for each channel
  - If you can eliminate the channel, great!
  - If not, reduce bandwidth as much as possible
Step #1: Detection

- Manner in which resource is shared controls who can send, receive using that resource
  - Shared Resource Matrix Methodology
  - Information flow analysis
  - Covert flow trees
SRMM

- Shared Resource Matrix Methodology
- Goal: identify shared channels, how they are shared
- Steps:
  - Identify all shared resources, their visible attributes [rows]
  - Determine operations that reference (read), modify (write) resource [columns]
  - Contents of matrix show how operation accesses the resource
Example

• Multilevel security model

• File attributes:
  – existence, owner, label, size

• File manipulation operations:
  – read, write, delete, create
  – create succeeds if file does not exist; gets creator as owner, creator’s label
  – others require file exists, appropriate labels

• Subjects:
  – High, Low
## Shared Resource Matrix

<table>
<thead>
<tr>
<th></th>
<th>read</th>
<th>write</th>
<th>delete</th>
<th>create</th>
</tr>
</thead>
<tbody>
<tr>
<td>existence</td>
<td>R</td>
<td>R</td>
<td>R, M</td>
<td>R, M</td>
</tr>
<tr>
<td>owner</td>
<td></td>
<td></td>
<td>R</td>
<td>M</td>
</tr>
<tr>
<td>label</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>M</td>
</tr>
<tr>
<td>size</td>
<td>R</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
</tbody>
</table>
Covert Storage Channel

- Properties that must hold for covert storage channel:
  1. Sending, receiving processes have access to same attribute of shared object;
  2. Sender can modify that attribute;
  3. Receiver can reference that attribute; and
  4. Mechanism for starting processes, properly sequencing their accesses to resource.
Example

- Consider attributes with both R, M in rows
- Let High be sender, Low receiver
- create operation both references, modifies existence attribute
  - Low can use this due to semantics of create
- Need to arrange for proper sequencing accesses to existence attribute of file (shared resource)
Use of Channel

- 3 files: ready, done, 1bit
- Low creates ready at High level
- High checks that file exists
  - If so, to send 1, it creates 1bit; to send 0, skip
  - Delete ready, create done at High level
- Low tries to create done at High level
  - On failure, High is done
  - Low tries to create 1bit at level High
- Low deletes done, creates ready at High level
Covert Timing Channel

• Properties that must hold for covert timing channel:
  1. Sending, receiving processes have access to same *attribute* of shared object;
  2. Sender, receiver have access to a time reference (wall clock, timer, event ordering, …);
  3. Sender can control timing of detection of change to that attribute by receiver; and
  4. Mechanism for starting processes, properly sequencing their accesses to resource
Example

- Revisit variant of KVM/370 channel
  - Sender, receiver can access ordering of requests by disk arm scheduler (attribute)
  - Sender, receiver have access to the ordering of the requests (time reference)
  - High can control ordering of requests of Low process by issuing cylinder numbers to position arm appropriately (timing of detection of change)
  - So whether channel can be exploited depends on whether there is a mechanism to (1) start sender, receiver and (2) sequence requests as desired
Uses of SRM Methodology

- Applicable at many stages of software life cycle model
  - Flexbility is its strength

- Used to analyze Secure Ada Target
  - Participants manually constructed SRM from flow analysis of SAT model
  - Took transitive closure
  - Found 2 covert channels
    - One used assigned level attribute, another assigned type attribute
Summary

- Methodology comprehensive but incomplete
  - How to identify shared resources?
  - What operations access them and how?
- Incompleteness a benefit
  - Allows use at different stages of software engineering life cycle
- Incompleteness a problem
  - Makes use of methodology sensitive to particular stage of software development