Lecture for February 19, 2016

ECS 235A

UC Davis

Matt Bishop

Presentations for Monday, February 22

- Francesco Capponi:
  – Questioner: Calvin Li

  "Securing the Software-Defined Network Control Layer"
- Chaitrali Joshi:
  – Questioner: Sandeep Rasoori

  "Addressing the Challenge of IP Spoofing"

Presentations for Wednesday, February 24

- Mark Crompton:
  - Questioner: Yuan-Yu Chen

  "A Diagnosis-Based Intrusion Detection Approach"
- Apoorva Rangaraju:
  - Questioner: Francesco Capponi

  "Reinforcement Learning Algorithms for Adaptive Cyber Defense Against Heartbleed"

Execution-Based Mechanisms

- Detect and stop flows of information that violate policy
  - Done at run time, not compile time
- Obvious approach: check explicit flows
  - Problem: assume for security, $\underline{x} \le \underline{y}$

```
if x = 1 then y := a;
```

  - When $x \neq 1$, $\underline{x}$ = High, $\underline{y}$ = Low, $\underline{a}$ = Low, appears okay—but implicit flow violates condition!

Fenton's Data Mark Machine

- Each variable has an associated class
- Program counter (PC) has one too
- Idea: branches are assignments to PC, so you can treat implicit flows as explicit flows
- Stack-based machine, so everything done in terms of pushing onto and popping from a program stack

Instruction Description

- *skip* means instruction not executed
- *push*(x, $\underline{x}$) means push variable x and its security class $\underline{x}$ onto program stack
- *pop*(x, $\underline{x}$) means pop top value and security class from program stack, assign them to variable x and its security class $\underline{x}$ respectively

Instructions

- $x := x + 1$ (increment)
  - Same as:

    if $\underline{PC} \le \underline{x}$ then $x := x + 1$ else *skip*
- if $x = 0$ then goto n else $x := x - 1$ (branch and save PC on stack)
  - Same as:

```
if x = 0 then begin
  push(PC, PC); PC := lub{PC, x}; PC := n;
end else if PC ≤ x then
  x := x - 1
else
  skip;
```

More Instructions

- `if′` x `= 0 then goto` n `else` x `:=` x `– 1` (branch without saving PC on stack)
  – Same as:

    `if` x `= 0 then`

    `if` $\underline{x} \leq \underline{PC}$ `then` PC `:=` n `else` *skip*

    `else`

    `if` $\underline{PC} \leq \underline{x}$ `then` x `:=` x `- 1 else skip`

More Instructions

- `return` (go to just after last *if*)
  - Same as:

    `pop(`*`PC`*`, `<u>*`PC`*</u>`);`
- `halt` (stop)
  - Same as:

    `if` *`program stack empty`* `then` *`halt`*
  - Note stack empty to prevent user obtaining information from it after halting

Example Program

```
1   if x = 0 then goto 4 else x := x - 1
2   if z = 0 then goto 6 else z := z - 1
3   halt
4   z := z - 1
5   return
6   y := y - 1
7   return
```

- Initially $x = 0$ or $x = 1$, $y = 0$, $z = 0$
- Program copies value of x to y

Example Execution

x	y	z	PC	$\underline{PC}$	*stack*	*check*
1	0	0	1	Low	—	
0	0	0	2	Low	—	Low $\leq \underline{x}$
0	0	0	6	$\underline{z}$	(3, Low)	
0	1	0	7	$\underline{z}$	(3, Low)	$\underline{PC} \leq \underline{y}$
0	1	0	3	Low	—	

Handling Errors

- Ignore statement that causes error, but continue execution
  - If aborted or a visible exception taken, user could deduce information
  - Means errors cannot be reported unless user has clearance at least equal to that of the information causing the error

Variable Classes

- Up to now, classes fixed
  - Check relationships on assignment, etc.
- Consider variable classes
  - Fenton's Data Mark Machine does this for $\underline{PC}$
  - On assignment of form $y := f(x_1, \ldots, x_n)$, $\underline{y}$ changed to lub$\{ \underline{x}_1, \ldots, \underline{x}_n \}$
  - Need to consider implicit flows, also

Example Program

```
(* Copy value from x to y
 * Initially, x is 0 or 1 *)
proc copy(x: int class { x };
                 var y: int class { y })
var z: int class variable { Low };
begin
  y := 0;
  z := 0;
  if x = 0 then z := 1;
  if z = 0 then y := 1;
end;
```

- $\underline{z}$ changes when z assigned to
- Assume $\underline{y} < \underline{x}$

Analysis of Example

- $x = 0$
  - `z := 0` sets $\underline{z}$ to Low
  - `if x = 0 then z := 1` sets z to 1 and $\underline{z}$ to $\underline{x}$
  - So on exit, $y = 0$
- $x = 1$
  - `z := 0` sets $\underline{z}$ to Low
  - `if z = 0 then y := 1` sets y to 1 and checks that lub{Low, $\underline{z}$} $\leq \underline{y}$
  - So on exit, $y = 1$
- Information flowed from $\underline{x}$ to $\underline{y}$ even though $\underline{y} < \underline{x}$

Handling This (1)

- Fenton's Data Mark Machine detects implicit flows violating certification rules

Handling This (2)

- Raise class of variables assigned to in conditionals even when branch not taken
- Also, verify information flow requirements even when branch not taken
- Example:
  - In `if x = 0 then z := 1`, z raised to x whether or not $x = 0$
  - Certification check in next statement, that $\underline{z} \leq \underline{y}$, fails, as $\underline{z} = \underline{x}$ from previous statement, and $\underline{y} \leq \underline{x}$

Handling This (3)

- Change classes only when explicit flows occur, but *all* flows (implicit as well as explicit) force certification checks
- Example
  - When $x = 0$, first "if" sets $\underline{z}$ to Low then checks $\underline{x} \le \underline{z}$
  - When $x = 1$, first "if" checks that $\underline{x} \le \underline{z}$
  - This holds if and only if $\underline{x}$ = Low
    - Not possible as $\underline{y} < \underline{x}$ = Low and there is no such class

The Confinement Problem

- What is the problem?
- Isolation: virtual machines, sandboxes
- Detecting covert channels
- Analyzing covert channels
- Mitigating covert channels

Overview

- The confinement problem
- Isolating entities
  - Virtual machines
  - Sandboxes
- Covert channels
  - Detecting them
  - Analyzing them
  - Mitigating them

Example Problem

- Server balances bank accounts for clients
- Server security issues:
  - Record correctly who used it
  - Send *only* balancing info to client
- Client security issues:
  - Log use correctly
  - Do not save or retransmit data client sends

Generalization

- Client sends request, data to server
- Server performs some function on data
- Server returns result to client
- Access controls:
  - Server must ensure the resources it accesses on behalf of client include *only* resources client is authorized to access
  - Server must ensure it does not reveal client's data to any entity not authorized to see the client's data

Confinement Problem

- Problem of preventing a server from leaking information that the user of the service considers confidential

Total Isolation

- Process cannot communicate with any other process
- Process cannot be observed

Impossible for this process to leak information

  – Not practical as process uses observable resources such as CPU, secondary storage, networks, etc.

Example

- 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

Covert Channel

- A path of communication not designed to be used for communication
- In example, file system is a (storage) covert channel

Rule of Transitive Confinement

- If *p* is confined to prevent leaking, and it invokes *q*, then *q* must be similarly confined to prevent leaking
- Rule: if a confined process invokes a second process, the second process must be as confined as the first

Lipner’s Notes

- All processes can obtain rough idea of time
  - Read system clock or wall clock time
  - Determine number of instructions executed
- All processes can manipulate time
  - Wait some interval of wall clock time
  - Execute a set number of instructions, then block

Kocher's Attack

- This computes $x = a^z \bmod n$, where $z = z_0 \ldots z_{k-1}$

```
x := 1; atmp := a;
for i := 0 to k–1 do begin
  if zi = 1 then
     x := (x * atmp) mod n;
  atmp := (atmp * atmp) mod n;
end
result := x;
```

- Length of run time related to number of 1 bits in z

Isolation

- Present process with environment that appears to be a computer running only those processes being isolated
  - Process cannot access underlying computer system, any process(es) or resource(s) not part of that environment
  - A *virtual machine*
- Run process in environment that analyzes actions to determine if they leak information
  - Alters the interface between process(es) and computer

Virtual Machine

- Program that simulates hardware of a machine
  - Machine may be an existing, physical one or an abstract one
- Why?
  - Existing OSes do not need to be modified
    - Run under VMM, which enforces security policy
    - Effectively, VMM is a security kernel

VMM as Security Kernel

- VMM deals with subjects (the VMs)
  - Knows nothing about the processes within the VM
- VMM applies security checks to subjects
  - By transitivity, these controls apply to processes on VMs
- Thus, satisfies rule of transitive confinement

Example 1: KVM/370

- KVM/370 is security-enhanced version of VM/370 VMM
  - Goal: prevent communications between VMs of different security classes
  - Like VM/370, provides VMs with minidisks, sharing some portions of those disks
  - Unlike VM/370, mediates access to shared areas to limit communication in accordance with security policy

Example 2: VAX/VMM

- Can run either VMS or Ultrix
- 4 privilege levels for VM system
  - VM user, VM supervisor, VM executive, VM kernel (both physical executive)
- VMM runs in physical kernel mode
  - Only it can access certain resources
- VMM subjects: users and VMs

Example 2

- VMM has flat file system for itself
  - Rest of disk partitioned among VMs
  - VMs can use any file system structure
    - Each VM has its own set of file systems
  - Subjects, objects have security, integrity classes
    - Called *access classes*
  - VMM has sophisticated auditing mechanism

Problem

- Physical resources shared
  - System CPU, disks, etc.
- May share logical resources
  - Depends on how system is implemented
- Allows covert channels

Sandboxes

- An environment in which actions are restricted in accordance with security policy
  - Limit execution environment as needed
    - Program not modified
    - Libraries, kernel modified to restrict actions
  - Modify program to check, restrict actions
    - Like dynamic debuggers, profilers

Examples Limiting Environment

- Java virtual machine
  - Security manager limits access of downloaded programs as policy dictates
- Sidewinder firewall
  - Type enforcement limits access
  - Policy fixed in kernel by vendor
- Domain Type Enforcement
  - Enforcement mechanism for DTEL
  - Kernel enforces sandbox defined by system administrator

Modifying Programs

- Add breakpoints or special instructions to source, binary code
  - On trap or execution of special instructions, analyze state of process
- Variant: *software fault isolation*
  - Add instructions checking memory accesses, other security issues
  - Any attempt to violate policy causes trap

Example: Janus

- Implements sandbox in which system calls checked
  - *Framework* does runtime checking
  - *Modules* determine which accesses allowed
- Configuration file
  - Instructs loading of modules
  - Also lists constraints

Configuration File

```
# basic module
basic

# define subprocess environment variables
putenv IFS=”\t\n “ PATH=/sbin:/bin:/usr/bin TZ=PST8PDT

# deny access to everything except files under /usr
path deny read,write *
path allow read,write /usr/*
# allow subprocess to read files in library directories
# needed for dynamic loading
path allow read /lib/* /usr/lib/* /usr/local/lib/*
# needed so child can execute programs
path allow read,exec /sbin/* /bin/* /usr/bin/*
```

How It Works

- Framework builds list of relevant system calls
  - Then marks each with allowed, disallowed actions
- When monitored system call executed
  - Framework checks arguments, validates that call is allowed for those arguments
    - If not, returns failure
    - Otherwise, give control back to child, so normal system call proceeds

Use

- Reading MIME Mail: fear is user sets mail reader to display attachment using Postscript engine
  - Has mechanism to execute system-level commands
  - Embed a file deletion command in attachment …
- Janus configured to disallow execution of any subcommands by Postscript engine
  - Above attempt fails

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