Ensuring System Security Using Data Flow Analysis
Agenda

- Overview of projects
- Dynamic Data Flow Analysis (DDFA) details
- Process Coloring (PC) details
- DDFA+PC demonstration
Project Overview

- Researching software and system security for Intelligence Advanced Research Projects Activity (IARPA)
- Two separately funded projects
  - Dynamic Data Flow Analysis for Improving Software Security
    - SwRI and UT Austin
  - Process Coloring: An Information Flow-Preserving Approach to Malware Investigation
    - Purdue
- Projects started June 2007
- Projects scheduled to finish December 2008
Project Collaborators

- **Dynamic Data Flow Analysis (DDFA):**
  Jeremy Price, Mark Brooks, Steve Cook, Arif Kasim  
  *Southwest Research Institute*
  Calvin Lin, Walter Chang  
  *Department of Computer Science*
  *University of Texas at Austin*

- **Process Coloring:**
  Eugene Spafford, Dongyan Xu, Ryan Riley  
  *Department of Computer Science and*
  *Center for Education and Research in Information Assurance and Security (CERIAS)*
  *Purdue University*
  Xuxian Jiang  
  *Department of Computer Science*
  *North Carolina State University*
Project Roles

- **UT Austin**
  - Research and implementation of DDFA core
    - pointer analysis, policy language, compiler infrastructure

- **SwRI**
  - Technology transfer
  - Hardening of DDFA core
  - Collaboration and integration efforts

- **Purdue**
  - Research and implementation of PC core
    - Xen changes, Linux kernel changes, flow infrastructure
• Create secure computing environments from commodity software

• Most commodity software not designed with security in mind
What Does “Secure” Mean?

- Some answers are easy to define

Example: We want our programs to exhibit memory safety
- Memory safety: It’s only possible to read and write memory as intended by the programmer
- Thus, no buffer overflows, no dangling pointers, no overwriting of the stack

- Other answers are domain-specific
  - SQL-injection
  - Cross-site scripting
  - Information leakage . . .
“Secure” is Also Context-Specific

- Whom do you trust?
- What is the threat model?
- What is your environment?
- Many other possible assumptions
- We can’t expect commodity software to be secure
Do We Need Commodity Software?

- Custom software has two costs
  - Fixed cost of writing the software
  - Recurring cost of maintaining and evolving software to keep up with the latest tools, libraries, and standards
Dynamic Data Flow Analysis
What are we trying to do?

- Our approach uses a complementary combination of static and dynamic data flow analysis to enhance programs to enforce a specified security policy.

- Utilize trusted compiler technologies to apply user defined security policy.

- Automate the application of security to existing C source code.
  - Separation of concerns.
Research Goals

- Minimize the impact to software development
- Keep program runtime and size overhead as low as possible
- Support multi-level security and other complex scenarios
- Extensibility for future threats
Our Approach With DDFA

Security policy separate from source code

Static dataflow analysis minimizes instrumentation

2. DDFA Compiler processes the Security Policy.
3. Enhanced C Code is generated.
5. Enhanced Executable is produced.
6. DDFA Runtime Engine executes the Enhanced Executable.

OWASP Meeting

10/22/2008
What is Data Flow Analysis?

- One type of static analysis
- Derives information about the **dynamic** behavior of a program by **statically** examining the code
- At the heart of many compiler transformations
Does variable x contain an odd or an even value?

- Lattice tells us how to merge values
Data Flow Analysis Example II

- Do we need the value of variable x?
  Will we ever use the value of x in the future?

Dataflow analysis can operate on abstract entities
Security Analysis as Data Flow

- Example:

  ```c
  int sock;
  char buffer[100];
  sock = socket(AF_INET, SOCK_STREAM, 0);
  read(sock, buffer, 100);
  execl(buffer);
  ```

- Vulnerability: executes any remote command
  - What if this program runs as root?
  - Requirement:
    - Data from an Internet socket should not specify a program to execute

- Typestate analysis:
  - Attach tags to objects to reflect object state
  - Keep tags updated during execution
Deploying DDFA

- Trusted program is instrumented version of the untrusted program
  - Performs Dynamic Data Flow Analysis
  - Enforces typestate security policy
Benefits of Broadway Approach

- Deep analysis encapsulated in compiler
- Specifications written by security expert
  - Can be re-used many times
  - eg. Annotate Standard C library once, can be applied to many C programs
- System administrator simply invokes the compiler
int vs, vb;
v_s = Tainted;
v_b = Tainted;
if (v_b!=Tainted)
{
    execl(buffer);
}
DDFA works in three stages:

- **Introduction**
  - Associates property values to memory objects as they are introduced into a program

- **Propagation**
  - Tracks the flow of memory objects and their property values throughout the program

- **Violation**
  - Identifies if a violation occurs at runtime based on the memory objects’ property values, which static analysis alone is not able to do
Example 1
Format String Vulnerability

Introduction

Hacker introduces malformed printf() format string via web

DDFA marks data entering from the web as “Tainted”

Propagation

int sock;
char buf[100];
sock = socket(AF_INET, ...);
recv(sock, buf, 100, 0);
...
buf2 = strdup(buf);

DDFA tracks the flow of this “Tainted” data throughout the execution

Violation

printf(buf2);

Tainted string arrives at printf() statement

DDFA flags a runtime violation, preventing the vulnerability from being exploited by the hacker

OWASP Meeting 10/22/2008
Example 2
Role Based Access Control

Introduction

Beetle Bailey logs on to Missile system to perform safety checks
DDFA registers him to the system as “grunt” level

Propagation

ac_level = authenticate();
...
safety_check();

Violation

Beetle accidentally attempts to invoke launch()
DDFA flags a runtime violation, preventing missile from being launched

DDFA tracks the flow of all Beetle’s activities throughout the missile system application

10/22/2008
Benefits of DDFA

- Application dataflow is tracked at compile and run time
  - Very low runtime overhead (many cases < 1%)
    - Leverages semantic information from policy
  - Complements other security approaches (e.g. Procees Coloring)
  - Configurable error mitigation at run time (e.g. fight through)

- Policy is separate from the source code
  - Removes security concerns when developing new applications
    - Including 3rd party and open-source development
  - Can secure existing legacy applications
  - Requires one additional step in an automated build process
Benefits of DDFA (cont)

- Generality and expressiveness of policy language
  - Can simultaneously defend against multiple security vulnerabilities (i.e. policy elements are composable)
    - (e.g. format string, file disclosure)
  - Possible to solve problems affecting memory-safe languages like Java and C#
    - SQL injection
    - Cross-site scripting
  - Applicable to areas that depend upon semantic information
    - Role based access control
    - Privacy concerns that involve data flow
  - Agile to combat new/future threats
What is Process Coloring?

- Propagating and logging provenance information (“colors”) along OS-level information flows for malware detection and sensitive data protection
- System level inter-process data flow tracking
- User can define what colors are associated in the system
How does PC work?

- Colors are propagated using custom code in the Linux kernel

- Color propagation is logged via the Xen hypervisor
  - Policy decisions can be made based on log info

- Reads to and writes from file descriptors are monitored for color propagation
PC Usage Scenario:
Server-Side Malware

**Capability 1:**
PC malware alert

“No shell process should have the color of Apache”

**Capability 2:**
Color-based identification of malware break-in point

**Capability 3:**
Color-based log partition for contamination analysis

PC Usage Scenario: Client-Side Malware Attack

Client-Side Malware Attack

Web Browser

Agobot

Agobot

Tax files

PC malware alert

“Web browser and tax colors should never mix”

Demo at: http://friends.cs.purdue.edu/projects pc/files/sinkfile.avi
“Living Lab” VM: End User’s View

From: John Smith <smith@something.com>

To: rileyrd@purdue.edu

Subject: Here is a file...

Body: Here is a file...

Attachment: f1040.pdf (438K)
“Living Lab” VM: Administrator’s View

```
ryan@ilovny: /tmp

File Edit View Terminal Tabs Help

**type=IW dst=8538 pid=3609 p_oc="10" f_oc="10"**
**type=ID src=8538 name="/\[8538\]"**
**type=F src=P3565 dst=P3610 p_oc=""**
**type=F src=P3608 dst=P3611 p_oc=""**
**type=IW dst=8387 pid=3609 p_oc="10" f_oc="10"**
**type=ID src=8387 name="/\[8387\]"**
**type=F src=P3481 dst=P3612 p_oc=""**
**type=F src=P3481 dst=P3613 p_oc=""**
**type=IW dst=8546 pid=3561 p_oc="10" f_oc="10"**
**type=ID src=8546 name="/\[8546\]"**
**type=F src=P3561 dst=P3614 p_oc="10"**
**type=F src=P3561 dst=P3615 p_oc="10"**
**type=F src=P3561 dst=P3616 p_oc="10"**
**type=F src=P3481 dst=P3617 p_oc=""**
**type=F src=P3481 dst=P3618 p_oc=""**
**type=IR src=393524 pid=3561 f_oc="40" p_oc="10,40"**
**type=ID src=393524 name="/home/user/Desktop/docs/f1040.pdf"**
**type=IW dst=8385 pid=3561 p_oc="10,40" f_oc="10,40"**
**type=ID src=8385 name="/\[8385\]"**
**type=F src=P3481 dst=P3619 p_oc=""**
**type=F src=P3481 dst=P3620 p_oc=""**
**type=F src=P3481 dst=P3621 p_oc=""**
```
Evaluation Metrics – Efficiency

- System Call Overhead
- Shell Scripts (8 concurrent)
- Process Creation
- Pipe Throughput
- Execl Throughput
- Dhrystone 2 using register variables
- Arithmetic Test (type = double)

Index Score (larger is better)
Evaluation with Malware
(Agobot, P2P bot...)

File Edit View Terminal Tabs Help

type=ID src=393376 name="/home/user/Desktop/docs/finances.txt"
type=F src=P3297 dst=P3350 p_oc=""
type=E pid=3350 name="bash"
type=IR src=393261 pid=3348 f_oc="20" p_oc="40,20"
type=ID src=/home/user/repo/git/ryan.c"
type=F src=P3297 dst=P3351 p_oc=""
type=E pid=3351 name="bash"
type=F src=P3351 dst=P3352 p_oc=""
type=E pid=3352 name="vi"
type=E pid=3353 name="sh"
type=IR src=393342 pid=3348 f_oc="1" p_oc="40,20,1"
type=ID src=393342 name="/home/user/.Trash/dsn07/-codeinj (copy).pdf"
type=IR src=627318 pid=3348 f_oc="14" p_oc="40,20,1,14"
type=ID src=627318 name="/home/user/pics/dirl/dir2/dir3/img_9995.jpg"
type=IR src=627317 pid=3348 f_oc="13" p_oc="40,20,1,14,13"
type=ID src=627317 name="/home/user/pics/dirl/dir2/img_9993.1pg"
type=IR src=627316 pid=3348 f_oc="12" p_oc="40,20,1,14,13,12"
type=ID src=627316 name="/home/user/pics/dirl/img_9992.1pg"
type=IR src=393615 pid=3348 f_oc="10" p_oc="40,20,1,14,13,12,10"
type=ID src=393615 name="/home/user/agobot/agobot3"
type=IR src=527243 pid=3348 f_oc="11" p_oc="40,20,1,14,13,12,10,11"
type=ID src=/usr/lib/openoffice/program/soffice.bin"
Putting it all together
A Motivating Scenario

"Sensitive file should never leave this computer"
PC or DDFA Alone Cannot Solve It

- **PC**
  - 😞 Process-level information flow treating processes as blackboxes
  - 😞 Overly conservative color tainting
  - 😊 Color tainting across processes

- **DDFA**
  - 😞 Language-level information flow confined within one process
  - 😞 Not aware of colors across the system
  - 😊 Fine-grain data flow tracking within a process
Example: *Without “PC+DDFA” Integration*

- **File 1**
- **File 2**
- **Process**
- **New file**

OWASP Meeting

10/22/2008
Example: *With “PC+DDFA” Integration*

```
fetch_color(file1)
fecnh_color(file2)
push_color(new_file)
```

**File 1**

**File 2**

**Process (w/ DDFA)**

**New file**

**Process Coloring (Operating System level)**
Example Scenario Tasks

- **SWRI+UTexas**
  - Making DDFA color-aware
  - Instrumenting a real-world file manager **PCManFM** with DDFA capability

- **Purdue**
  - Implementing `fetch_color()` and `push_color()` in PC
  - Testing instrumented **PCManFM** in living lab VM

- **Integration Meeting**
  - September 8th, 2008 SwRI visited Purdue
Sensitive Financial Information
Flow Graphs

- Process Coloring *Without DDFA*

```
/home/user/pictures/img_9992.jpg

/home/user/pictures/img_9993.jpg

/home/user/pictures/img_9995.jpg
```

3 Colors – Bad.
• Process Coloring *With* DDFA

1 Color – Good.
Neither DDFA nor PC solves the entire system level security problem

DDFA = intra-process data flow tracking

PC = inter-process data flow tracking

Together DDFA and PC implement policy-driven system data flow tracking