The confused deputy problem (Or why capabilities might have been invented)

Berlin, 21st November 2014
A very powerful program - a program that can delete all your files - a program that can scan your email for interesting tidbits. Many of us spend hours running this program. What is this program?

A computer game --and every other program we use

The only rights the game really needs to do its job are the ability to write in its window and to receive UI events directed at its window. Yet -- like every other program we execute -- it runs with a lot more rights than that. It runs with all of our authority.
Power is dangerous. While the game probably doesn’t do any of those things, if it became corrupted by a virus, it could

The less power we give to a program, the less harm it can do when it runs. In general, giving every program we run access to all our files is dangerous, because if any of those programs get compromised, our files would be lost.

The solution is obvious: we should only grant a program the rights it needs to do its job, and no more. The same goes for every other application we run. The goal of object capabilities is to make that easier.
Content

- Introduction
- Analysis of the confused deputy problem
- Examples
- Ambient authority
- Solution-capabilities
- Advantages
- Discussion
Introduction

- Deputy

Confidence trick

Definitions:

Programs that take actions on the behalf of other programs are deputies and need appropriate permissions for their duties.

An attempt to defraud a victim after first gaining his/her confidence in order to use him/her as a confused deputy (in this context)

The confused deputy | P.Jagannatha | Computer Security Seminar
The Confused Deputy Problem

**Definition:** The confused deputy problem occurs when one process tricks another process to do an action it doesn’t have permissions to do.

Example: A compiler was given a permission to write in a directory. The user compiled a program and specified some very special filename for the output log. He/She was able to overwrite some files he/she did not have access to.
Other Examples

- **Cross-site Scripting:**
The Samy (computer worm) used this vulnerability to turn the browser’s authenticated MySpace session into a confused deputy making it post an executable copy of the worm as a MySpace message which was then viewed and executed by friends of the infected user.

- **Cross-site request forgery:**
  Web browser (confused deputy) was used to perform sensitive actions against a web application.

- **Clickjacking:**
The user (confused deputy) thought he/she was harmlessly browsing a website (an attacker-controlled website) but he/she is in fact tricked into performing sensitive actions on another website.
Analysis of The Confused Deputy Problem

- The compiler runs with authority from two sources
  - the invoker
  - the system admin (who installed the compiler and controls billing and other info)

- It is the deputy of two masters

- There is no way to tell which master the deputy is serving when performing a write
The Confused Deputy Problem

System Admin

SYSX/FORT $OUTPUT Compiler Program

User

SYSX (Dir)
  a)FORT
  b)STAT
  c)BILL

Write to the bill file

SYSX/BILL

Write to output file

$Output
Ambient authority

“cp” must run with all of the user’s authority:

```
$ cp foo.txt bar.txt
```

“cat” needs no authority other than what’s given to it:

```
$ cat < foo.txt > bar.txt
```

Authority: ability to affect the rest of the world.
Ambient authority: authority that’s available even if you don’t ask for it.
Capabilities

- Stored by rows
- Capabilities is stored for each user
- Capability systems are set up so that the only way you can side-effect the rest of the world is by sending a message via some capability. That means that the only way to influence the outside world is by holding a capability that will let you do so. In other words, in capability systems, capabilities are the sole carriers of authority.
How the Capability Approach Solves the Confused Deputy Problem

- Invoker must pass in a capability for $OUTPUT$, which is stored in slot 3.
- Writing to output uses the capability in slot 3.
- Invoker cannot pass a capability he doesn’t have.
The Confused Deputy Solution

Home Files Directory (SYSX)

Compiler

(SYSX)STAT

(SYSX)BILL

"write debugging info to the write-stream I have opened for you"

Compiler User's Directories

Compiler User
Advantages of capabilities

- When a subject holds a capability for an object, it knows it has access to that object

- Capabilities allow finer-grained treatment of subjects
  - E.g., at the process level rather than the user level

- Capabilities allow easier delegation
  - With capabilities, can delegate a subset of the rights you have

- Better at enforcing “principle of least privilege”
  - Provide access to minimal resources, to the minimal set of subjects
Outlook

• Ambient authority should be avoided

• By default, components should come to life with no initial authority

• Help programmer avoid confused deputy bugs

• Make it easy to build security boundaries
Discussion

Can confused deputy problems be eliminated completely?
Literature


