Software Foundations of Security & Privacy
15316 Spring 2017
Lecture 15:
Capabilities

Matt Fredrikson, Jean Yang
mfredrik@cs.cmu.edu

March 21, 2017

Today's Lecture

Confused Deputy Problem

```
$ cc prog.c -o prog
```

Confused Deputy Problem

```
$ cc prog.c -o prog
```

- Server wants to charge people for time spent compiling

Confused Deputy Problem

```
$ cc prog.c -o prog
```

- Server wants to charge people for time spent compiling
- Stores usage log in `/var/log/charges`

Confused Deputy Problem

```
$ cc prog.c -o prog
```

- Server wants to charge people for time spent compiling
- Stores usage log in `/var/log/charges`
- Obviously, `/var/log/charges` is only writeable by `cc`

Confused Deputy Problem

```
$ cc prog.c -o prog
```

- Server wants to charge people for time spent compiling
- Stores usage log in `/var/log/charges`
- Obviously, `/var/log/charges` is only writeable by `cc`
- Can we compile for free?

Confused Deputy Problem

```
$ cc prog.c -o prog
```

- Server wants to charge people for time spent compiling
- Stores usage log in `/var/log/charges`
- Obviously, `/var/log/charges` is only writeable by `cc`
- Can we compile for free?

```
$ cc prog.c -o prog
$ cc foo.c -o /var/log/charges
```

Confused Deputies

What went wrong?

Confused Deputies

What went wrong?

- `cc` needs authority to write all files under control of the user

Confused Deputies

What went wrong?

- `cc` needs authority to write all files under control of the user
- It *also* needs authority to write to the charge log

Confused Deputies

What went wrong?

- `cc` needs authority to write all files under control of the user
- It *also* needs authority to write to the charge log
- Admin "deputizes" `cc` with authority on `/var/log/charges`

Confused Deputies

What went wrong?

- `cc` needs authority to write all files under control of the user
- It *also* needs authority to write to the charge log
- Admin "deputizes" `cc` with authority on `/var/log/charges`
- We tricked `cc` into using its power for evil

Confused Deputies

What went wrong?

- `cc` needs authority to write all files under control of the user
- It *also* needs authority to write to the charge log
- Admin "deputizes" `cc` with authority on `/var/log/charges`
- We tricked `cc` into using its power for evil

Hence, `cc` is a **confused deputy**

Ambient authority

Authority

The **authority** of a program is the set of ways in which it can cause changes to external system state.

Ambient authority

Authority

The **authority** of a program is the set of ways in which it can cause changes to external system state.

Ambient authority (jargon)

Authority posessed by a program at a particular time that it did not request, and does not *necessarily* need.

Example

`cp` must run with all the user's filesystem authority

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

Example from David Wagner

Example

`cp` must run with all the user's filesystem authority

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

`cat` only needs the authority you give it

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

Example from David Wagner

Example

`cp` must run with all the user's filesystem authority

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

`cat` only needs the authority you give it

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

`cat` requires far less ambient authority than `cp`

Example from David Wagner

Example

`cp` must run with all the user's filesystem authority

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

`cat` only needs the authority you give it

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

`cat` requires far less ambient authority than `cp`

Really, this is about **least privelege**

Example from David Wagner

Back to the confused deputy

```
let main (s: string) (d: string) =
  let src = read_contents s in
  let obj = compile src in
  let _ = write_charge "/var/log/charges" in
  write_contents d obj
```

Back to the confused deputy

```
let main (s: string) (d: string) =
  let src = read_contents s in
  let obj = compile src in
  let _ = write_charge "/var/log/charges" in
  write_contents d obj
```

Source of the problem:

- ▶

Unconfusing the deputy

```
let charge_fd = open_out "/var/log/charges"

let main (s: in_channel) (d: out_channel) =
  let src = read_contents s in
  let obj = compile src in
  let _ = write_charge charge_fd in
  write_contents d obj
```

Unconfusing the deputy

```
let charge_fd = open_out "/var/log/charges"

let main (s: in_channel) (d: out_channel) =
  let src = read_contents s in
  let obj = compile src in
  let _ = write_charge charge_fd in
  write_contents d obj
```

What did we do here?

Unconfusing the deputy

```
let charge_fd = open_out "/var/log/charges"

let main (s: in_channel) (d: out_channel) =
  let src = read_contents s in
  let obj = compile src in
  let _ = write_charge charge_fd in
  write_contents d obj
```

What did we do here?

- Can only access files **already opened by the user**

Unconfusing the deputy

```
let charge_fd = open_out "/var/log/charges"

let main (s: in_channel) (d: out_channel) =
  let src = read_contents s in
  let obj = compile src in
  let _ = write_charge charge_fd in
  write_contents d obj
```

What did we do here?

- Can only access files **already opened by the user**
- Aside from log, this is all the compiler needs

Unconfusing the deputy

```
let charge_fd = open_out "/var/log/charges"

let main (s: in_channel) (d: out_channel) =
  let src = read_contents s in
  let obj = compile src in
  let _ = write_charge charge_fd in
  write_contents d obj
```

What did we do here?

- Can only access files **already opened by the user**
- Aside from log, this is all the compiler needs
- Access to charge log is baked into the source

Unconfusing the deputy

```
let charge_fd = open_out "/var/log/charges"

let main (s: in_channel) (d: out_channel) =
  let src = read_contents s in
  let obj = compile src in
  let _ = write_charge charge_fd in
  write_contents d obj
```

What did we do here?

- Can only access files **already opened by the user**
- Aside from log, this is all the compiler needs
- Access to charge log is baked into the source
- No more confusion

Capabilities

This solution utilizes **capabilities** to achieve least privelege

Capabilities

This solution utilizes **capabilities** to achieve least privelege

Capability model equates **designation** and **authority**

Capabilities

This solution utilizes **capabilities** to achieve least privelege

Capability model equates **designation** and **authority**

- Designation: which resource to operate on

Capabilities

This solution utilizes **capabilities** to achieve least privelege

Capability model equates **designation** and **authority**

- Designation: which resource to operate on
- Authority: which operations are allowed on the resource

Capabilities

This solution utilizes **capabilities** to achieve least privelege

Capability model equates **designation** and **authority**

- Designation: which resource to operate on
- Authority: which operations are allowed on the resource

In the previous example, file pointers:

Capabilities

This solution utilizes **capabilities** to achieve least privelege

Capability model equates **designation** and **authority**

- Designation: which resource to operate on
- Authority: which operations are allowed on the resource

In the previous example, file pointers:

- Designate a particular file (e.g., `fopen('`**prog.c**`', 'r')`)

Capabilities

This solution utilizes **capabilities** to achieve least privelege

Capability model equates **designation** and **authority**

- Designation: which resource to operate on
- Authority: which operations are allowed on the resource

In the previous example, file pointers:

- Designate a particular file (e.g., `fopen('`**`prog.c`**`', 'r')`)
- Authorize rights (e.g., `fopen('prog.c', '`**`r`**`')`)

Capabilities in the abstract

Is this just an ACL by a different name?

Implementing capabilities: systems

Implementing capabilities: crypto

Object capabilities

Object capabilities: confinement

Object capabilities: revocation

Language support for capabilities

Example: capabilities in Java

Example: capabilities in OCaml

Reasoning about authority

Evolving capabilities

Bounding authority

Modular reasoning

Example: tamper resistent logging

Coding with discipline

Recursive authority reduction

Immutability