Making Linux Protection Mechanisms Egalitarian with UserFS

Taesoo Kim

and

Nickolai Zeldovich

MIT CSAIL

Overview:
How to build secure applications?

- *Simple* **in principle**:
  - **reduce privileges** of application components
  - **enforce policy** at lower level (e.g. OS kernel)

- *Difficult* **in practice** (unless root user):
  - **cannot** *create* new principals
  - **cannot** *reduce* privileges

This Talk:

How to help programmers to **reduce privileges** and **enforce security policy** in Linux?

by **allocating** and **managing** UIDs

Today’s Unix-like OS

- **UID** is **not a real user’s identity** anymore

  (instead, also use UID as **a protection principal**)

  i.e. nobody, www-data, wheelfs, etc.

- **Existing protection mechanisms** are using UID as a security principal

  i.e. filesystem permission

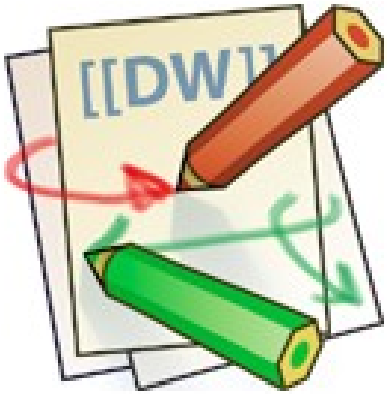
Running example: DokuWiki

[[dokuwiki]]

Show pagesource | Old revisions

Trace: » start » login » dokuwiki

DokuWiki



DokuWiki is a standards compliant, simple to use Wiki, mainly ai
developer teams, workgroups and small companies. It has a simple
readable outside the Wiki and eases the creation of structured te
required.

Read the DokuWiki Manual to unleash the full power of DokuWiki.

Download

DokuWiki is available at http://www.splitbrain.org/go/dokuwiki

Example: Security model of DokuWiki

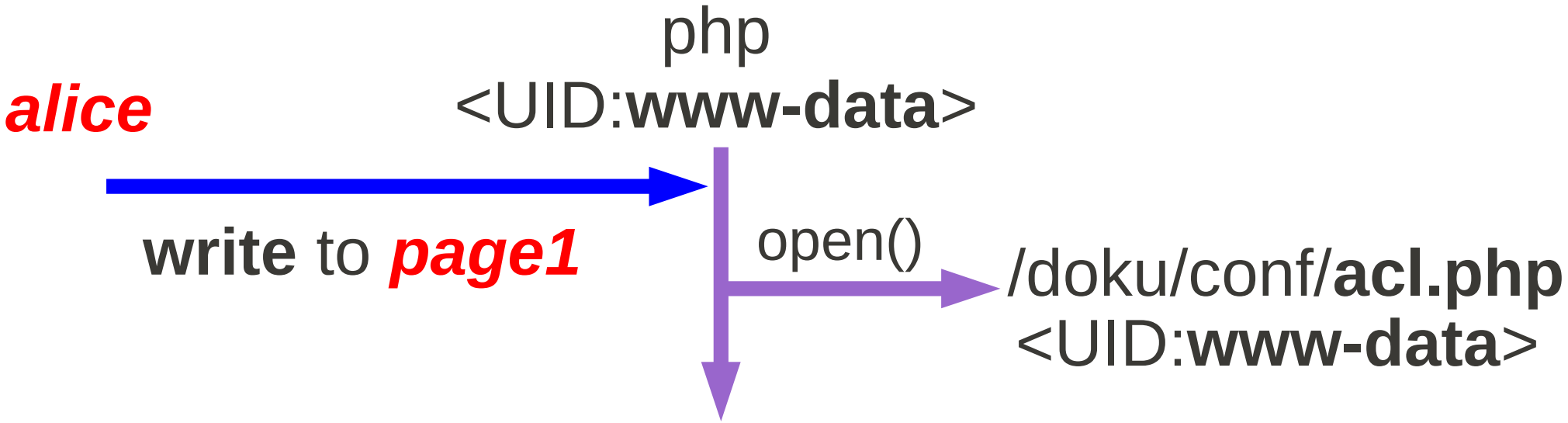
- **PHP** based Wiki
- Run as a **single UID**

- Main features
  1) **Wiki users**
  2) Saving **each page** as **a file**
  3) **ACL** on each page

Example: Run DokuWiki

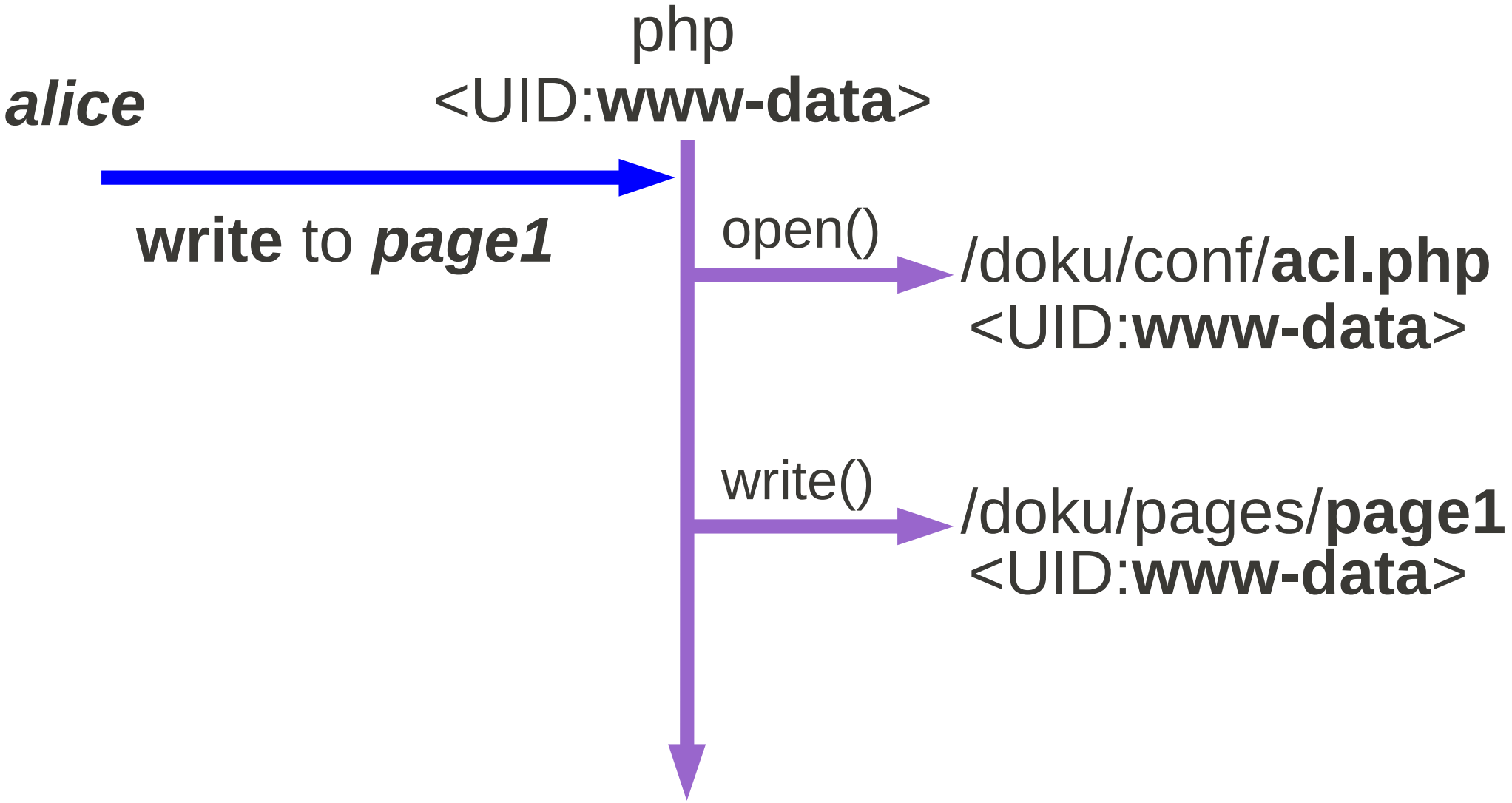
php
<UID:**www-data**>

Example: ***Alice*** write to the ***page1***

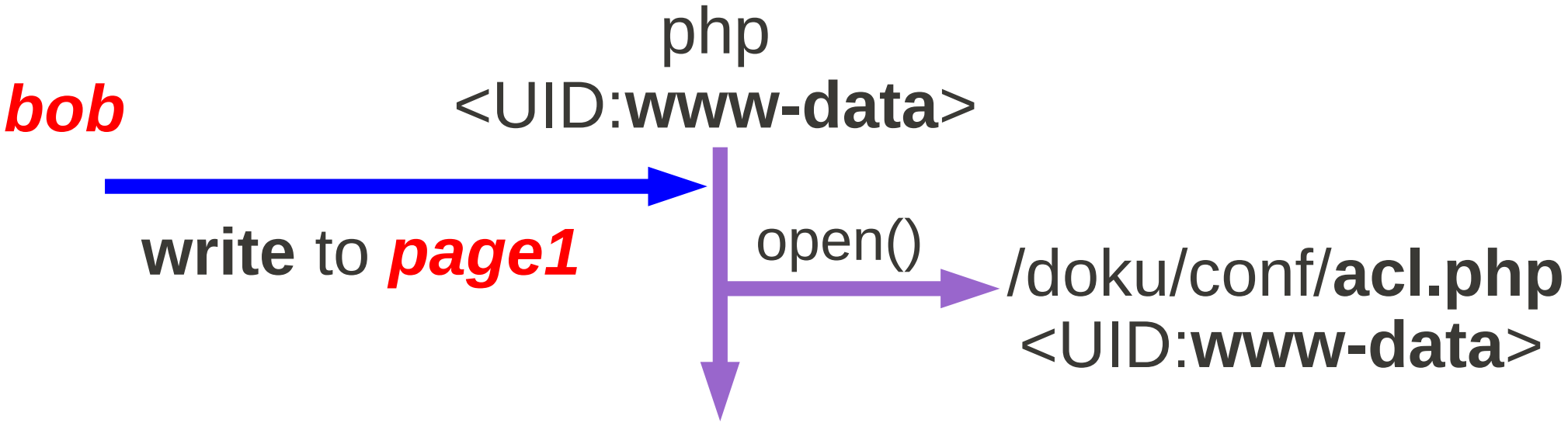


ACL of DokuWiki Pages	
/doku/pages/**page1**	**Alice**: r/w **Bob**:r/-
/doku/pages/**page2**	**Alice: r**/- **Bob**: r/w

Example: ***Alice*** write to the ***page1***

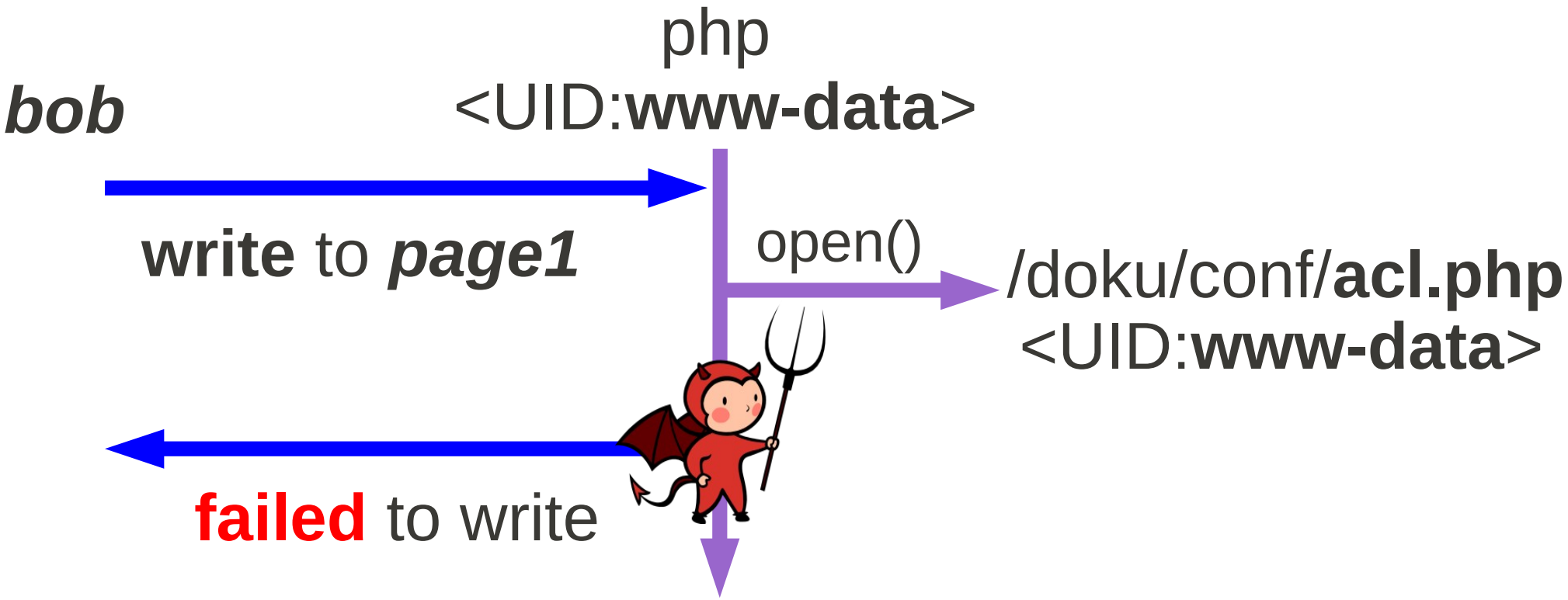


Example: ***Bob*** write to the ***page1***

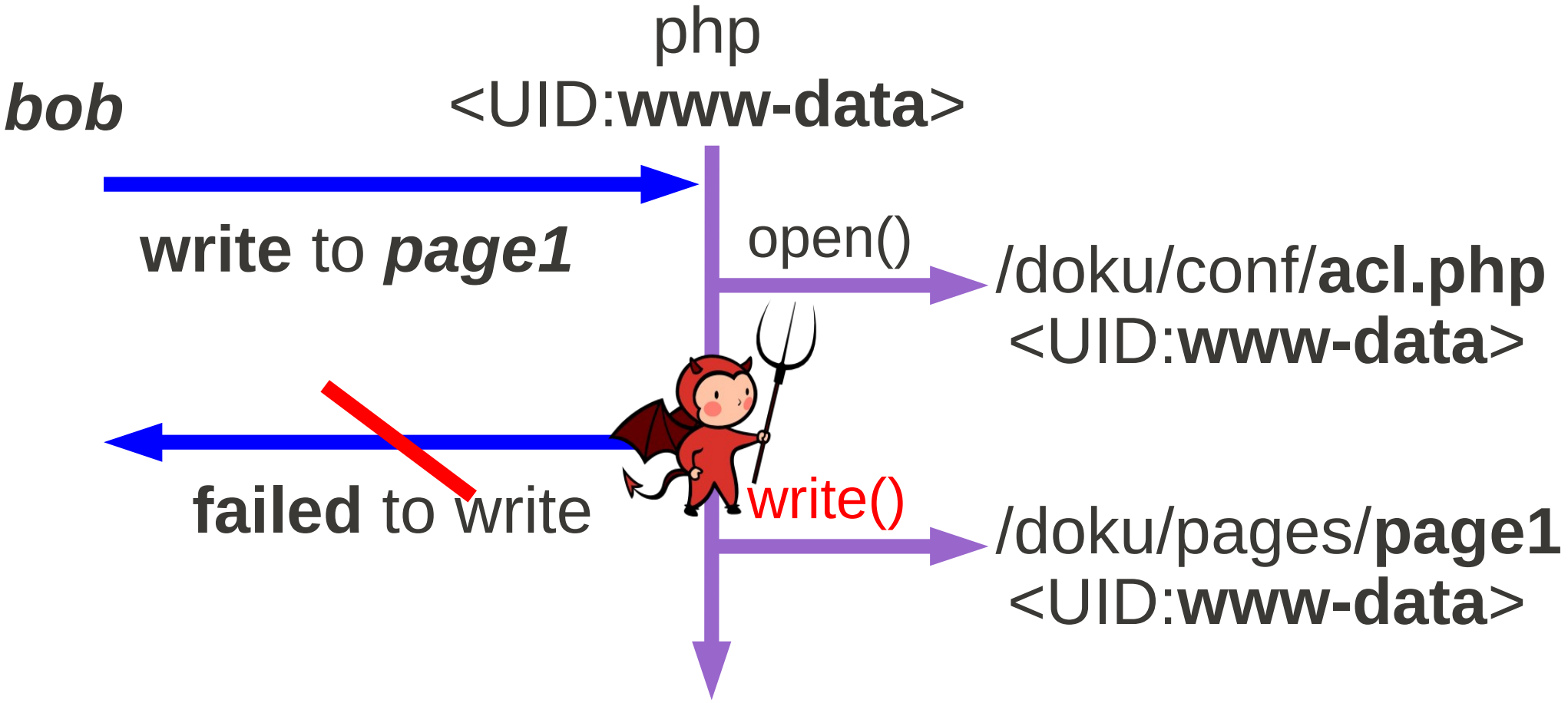


ACL of DokuWiki Pages	
/doku/pages/**page1**	**Alice**: r/w **Bob**:r/-
/doku/pages/**page2**	**Alice: r**/- **Bob**: r/w

Example: ***Bob*** write to the ***page1***



Example: Vulnerability when checking ACL



Example: Vulnerability when checking ACL

The ACL check happens ***40 times***
in DokuWiki's code:
New, *potentially-buggy* code in every app.

bob

.php
<UID:**www-data**>

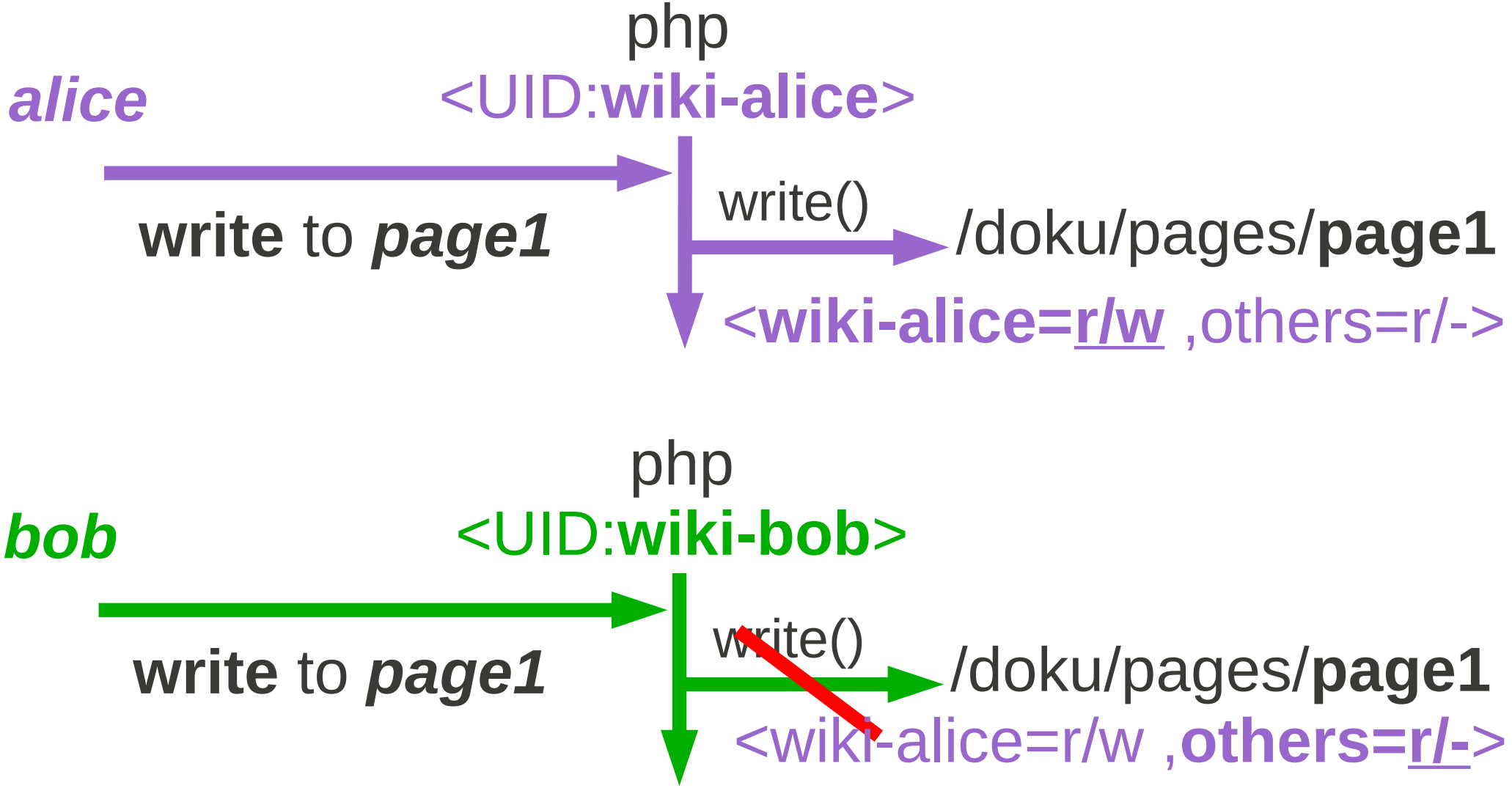
failed to write

write()

/doku/pages/**page1**
<UID:**www-data**>

CVE-2010-0288:
Insufficient Permission Check

***Strawman*: Running php with different UID**



Problem: Privilege separation is difficult in Unix

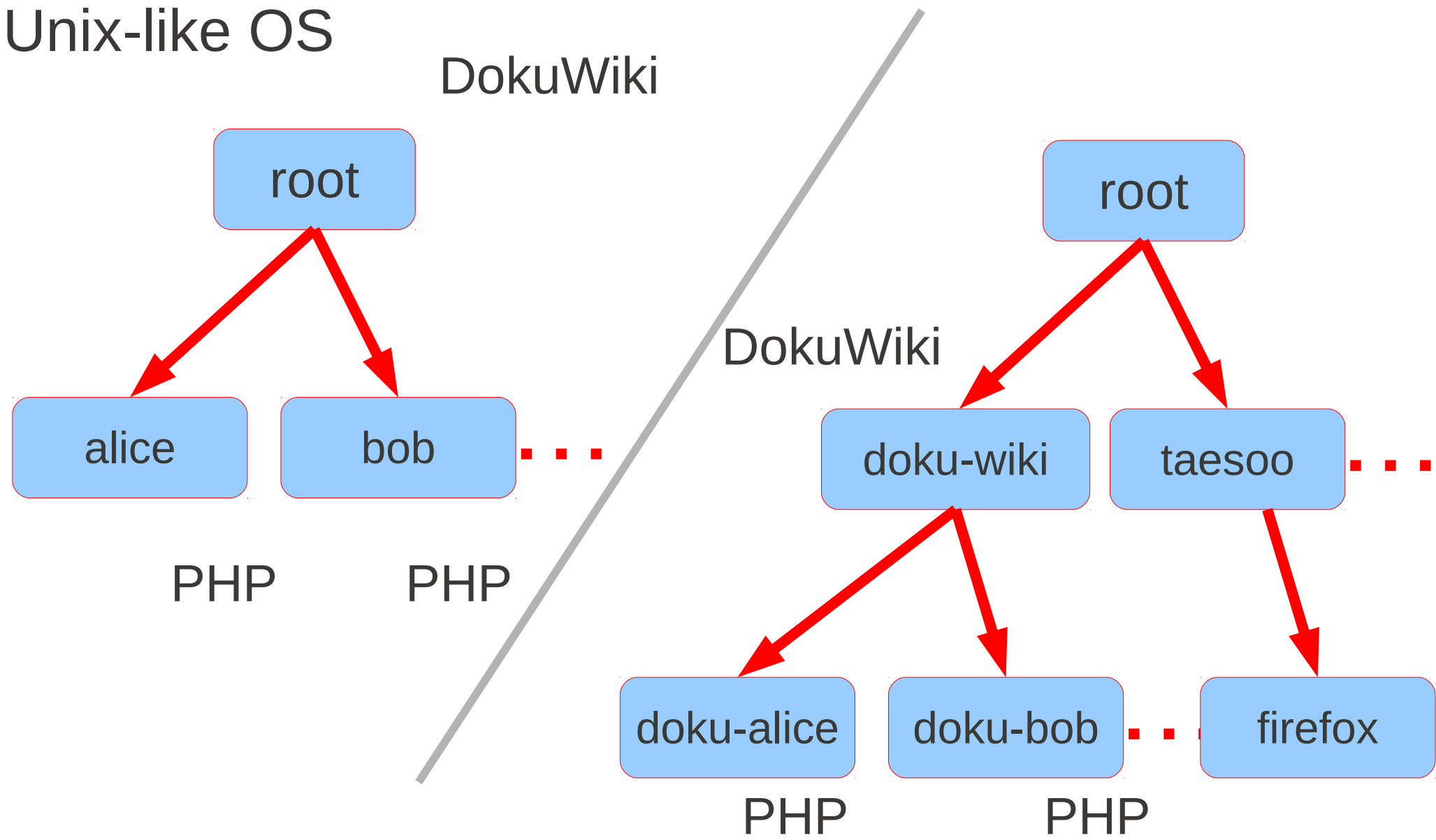
- Applications **cannot**
  - **allocate new UIDs** (e.g. adduser)
  - **switch current UID** (e.g. setuid)

  without **root privilege**
- **Ironically**,

  To **reduce** privilege, it requires **root** privilege

- Running DokuWiki as **root** is a **security disaster**

Problem: Privilege separation is difficult in Unix



Goal of this work

Allowing **any** application to use these **protection mechanisms without root privilege**

- create **a new principal**
- reuse **existing protection mechanisms**
  - use **chroot** and **firewall** mechanisms

Outline

- Overview
- **Design**

- Example
- Implementation
- Evaluation
- Limitation
- Related work
- Conclusion

Design: UID allocation

- **Strawman**: pick a **previously unused** UID
- Challenges
  - **who** can call setuid()?
  - How to **reuse** UIDs?
  - How to make UIDs **persistent**?

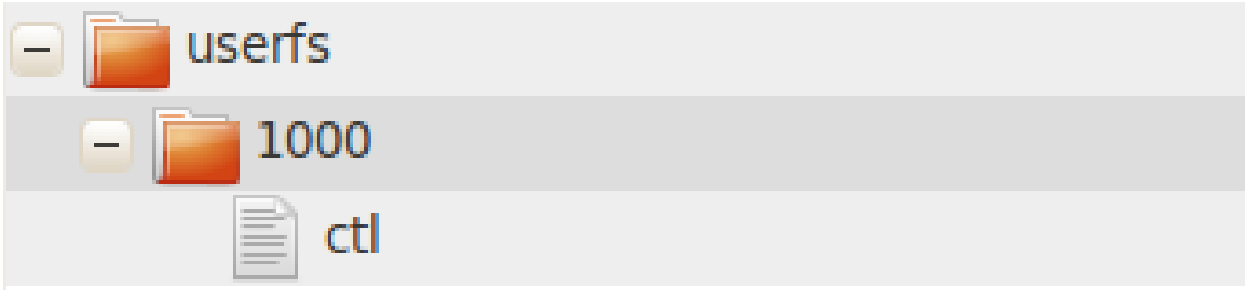
Challenge: Who can **call setuid()?**

- Current Linux
  - **Root** can switch to **any** UID with *setuid()*
  - **Non-root cannot** switch to new UID with *setuid()*
- Ideal system requirements
  - Need to **represent** privilege of each UID
  - Need to **specify** who can access each UID
  - Need to **pass** privilege between processes

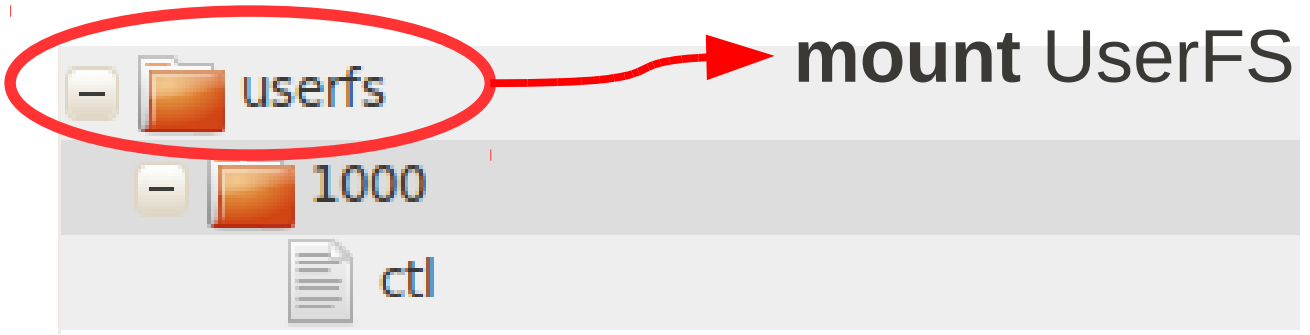
Key Idea: UserFS

- Maintaining **UIDs** as **files** in /proc-like **filesystem**
  - **Representing Privileges**
    - each UID is represented by **a file**
  - **Delegating Privileges**
    - **change** permissions on the file
    - **send** the file descriptor via FD passing
  - **Accountability**
    - track allocated UIDs of each user in **a directory**

Representing UIDs



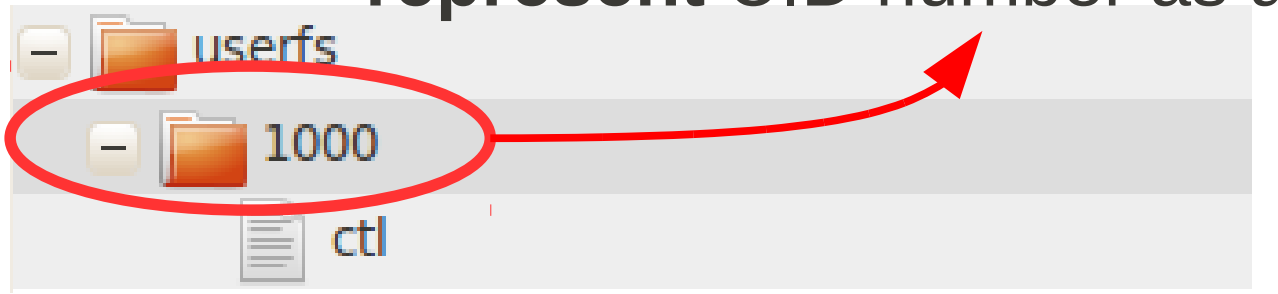
Representing UIDs



mount UserFS at /userfs

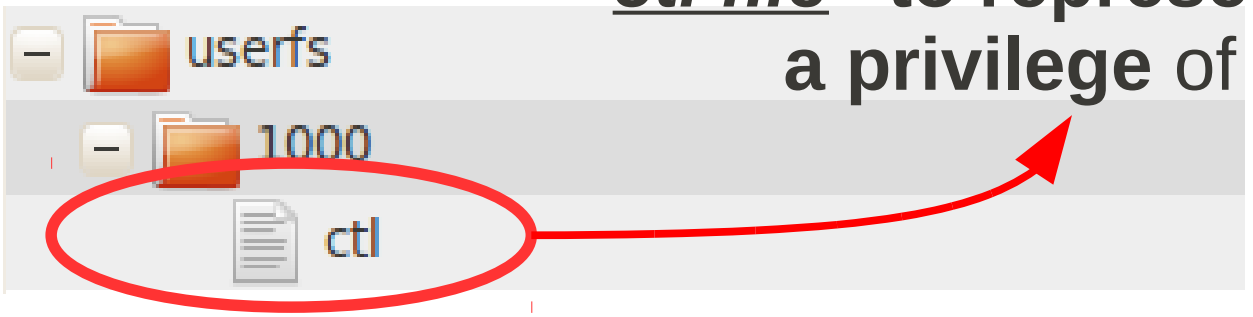
Representing UIDs

represent UID number as a directory

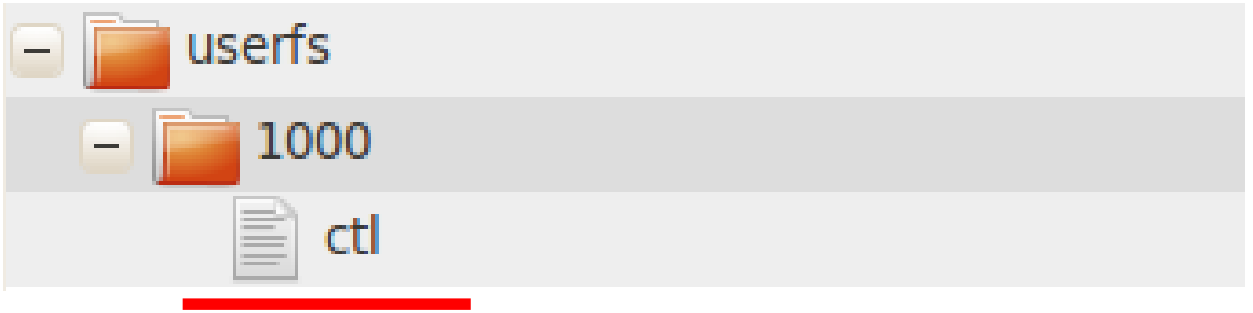


Representing UIDs

“*ctl file*” to represent a privilege of each UID



Representing Privileges



- Each UID has only one ***ctl file***
- Any process having the **file descriptor of the *ctl***
  - can **change** current **UID** e.g. *setuid()*
  - can **pass** it through **Unix domain socket** e.g. *send()*
  - can **deallocate UID** by **deleting** the ***ctl file*** e.g. unlink()

Challenge: How to **reuse** UIDs?

- **Ideally, unique ID** to every principal

- **Problem**:
  - Linux use **32-bit** UID
  - **Reuse** previously allocated UID

- **Solution**:
  - Introduce **64-bit #gen**
  - Use **#gen** to detect unwanted UID reuse

Challenge: How to make UIDs **persistent**?

- For each UID, keep track of:
  - **#gen**
  - permissions of ***ctl file***
  - **creator's** UID

in **persistent database**

Managing UIDs

File system	***UserFS***
Add a file	**Allocate** a UID
Delete a file	**Deallocate** a UID
Open a file	**Gain** the privilege of UID
Change permission	**Delegate** a privilege

Example: Using a *Ufile*

fd=**open**(**/userfs/1000/ctl**)

1) Setuid

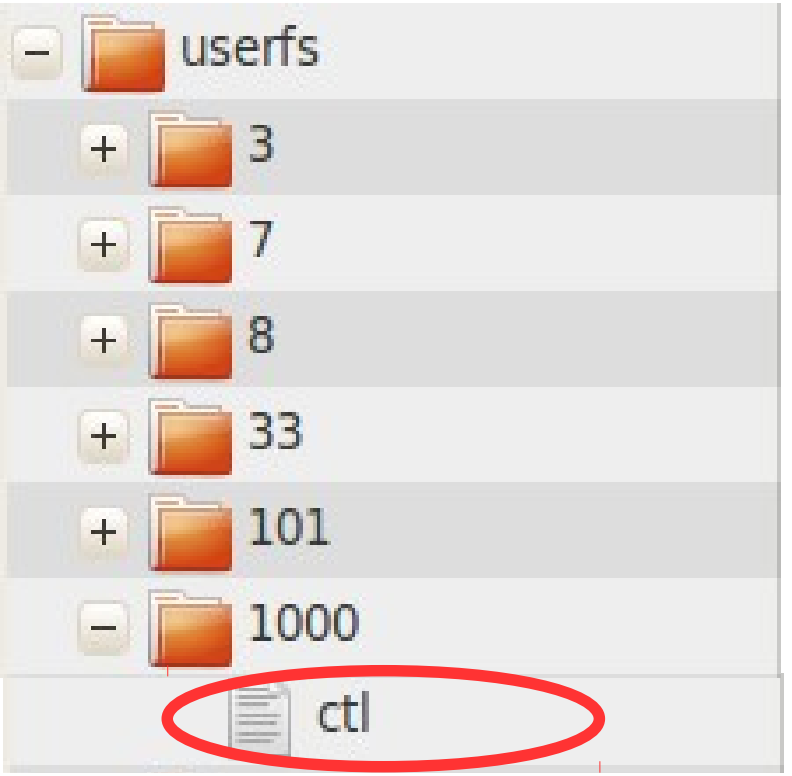
ioctl(fd, **IOCTL_SETUID**)

2) UID Allocation

ioctl(fd, **IOCTL_ALLOC**, **2000**)

3) Privilege Delegation

sendmsg(receiver-socket, **fd**)



Outline

- Overview
- Design
- **Example**

- Implementation
- Evaluation
- Limitation
- Related work
- Conclusion

Example: Security model of *UserFS-aware* DokuWiki

Key idea:

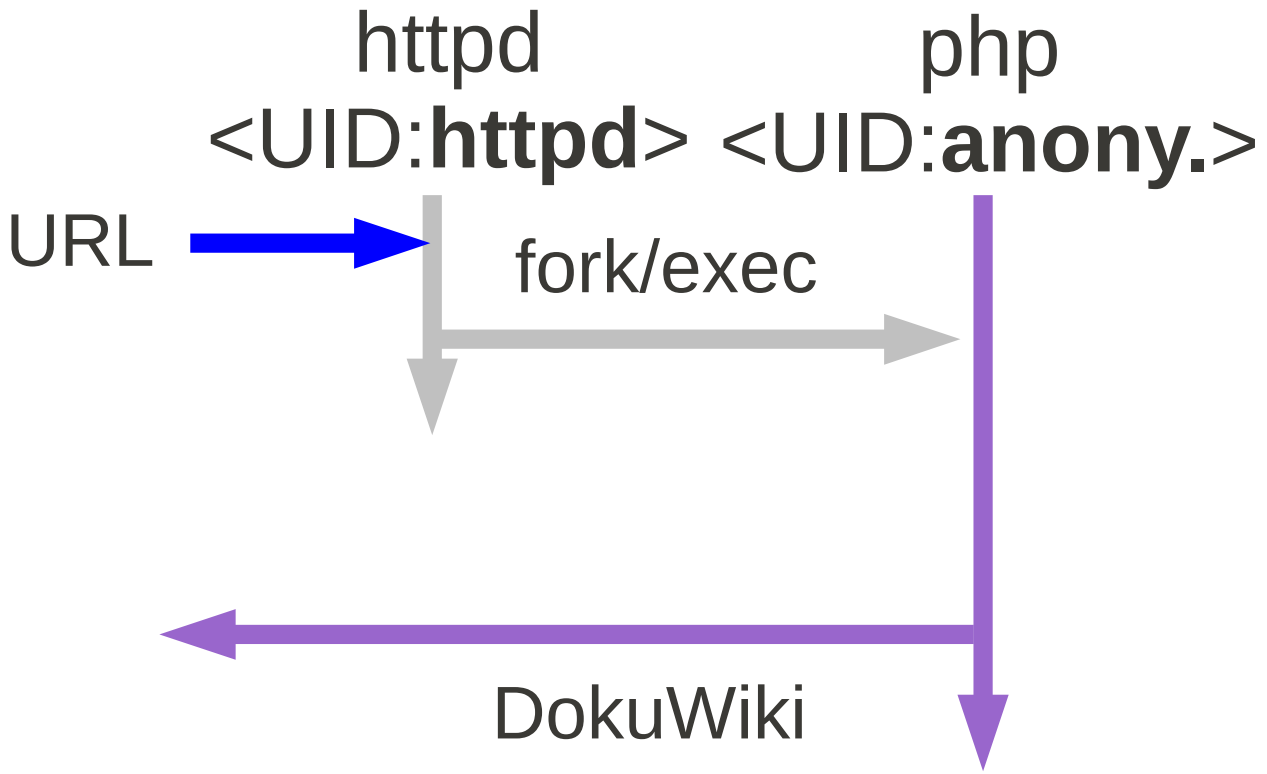
Allocate **UID** for **each Wiki user!**

- Authenticate users with non-root daemon
- Use UID Sandboxing
- Reuse well-tested ACL of filesystem

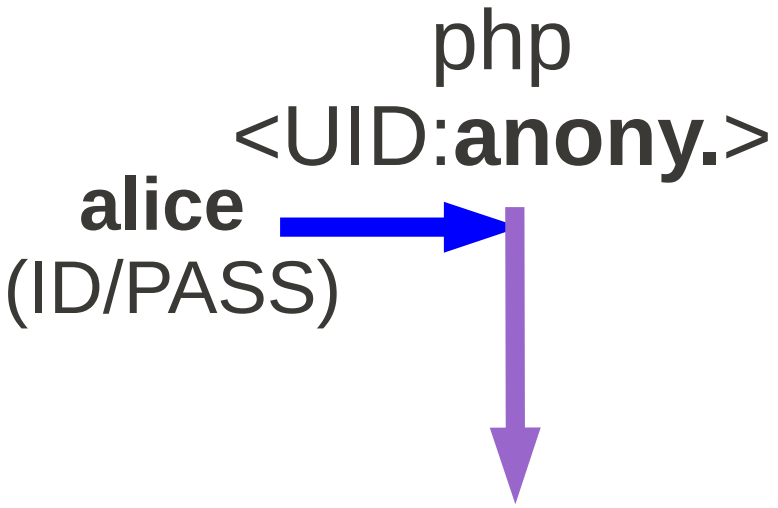
Example: Authenticating users with non-root daemon

- Allocate new **doku-admin** UID (Wiki admin)
- When a new user **signs up**
  - **doku-admin** will **allocate a UID** for the user
  - **doku-admin** will gain **read permission** on ***ctl file***
- When a user **logs in**
  - **login-mgr** (setuid to **doku-admin**) check id/passwd
  - open the ***ctl file*** of the Wiki user
  - **send** it through Unix domain socket

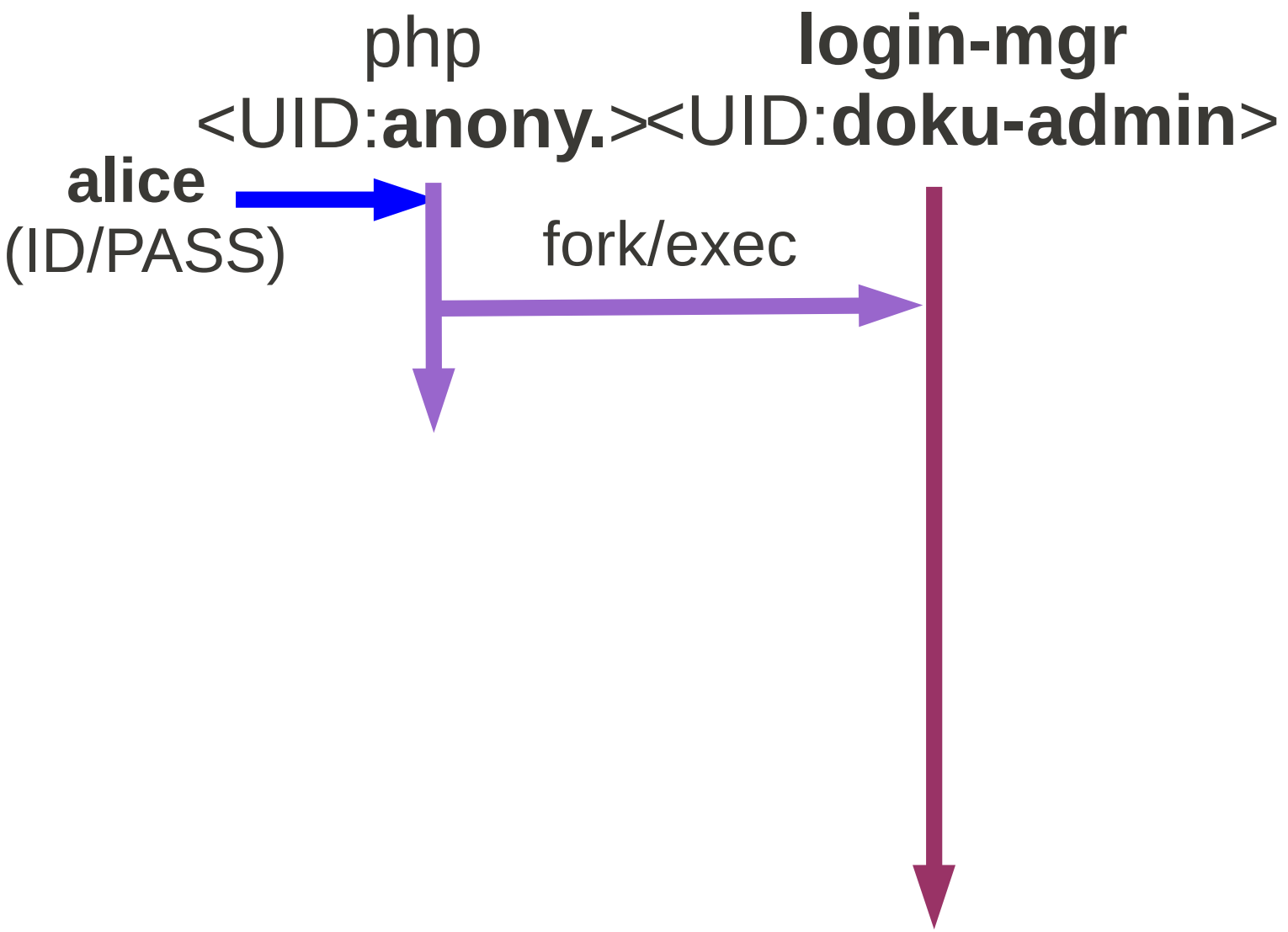
Example: Servicing DokuWiki with anonymous UID



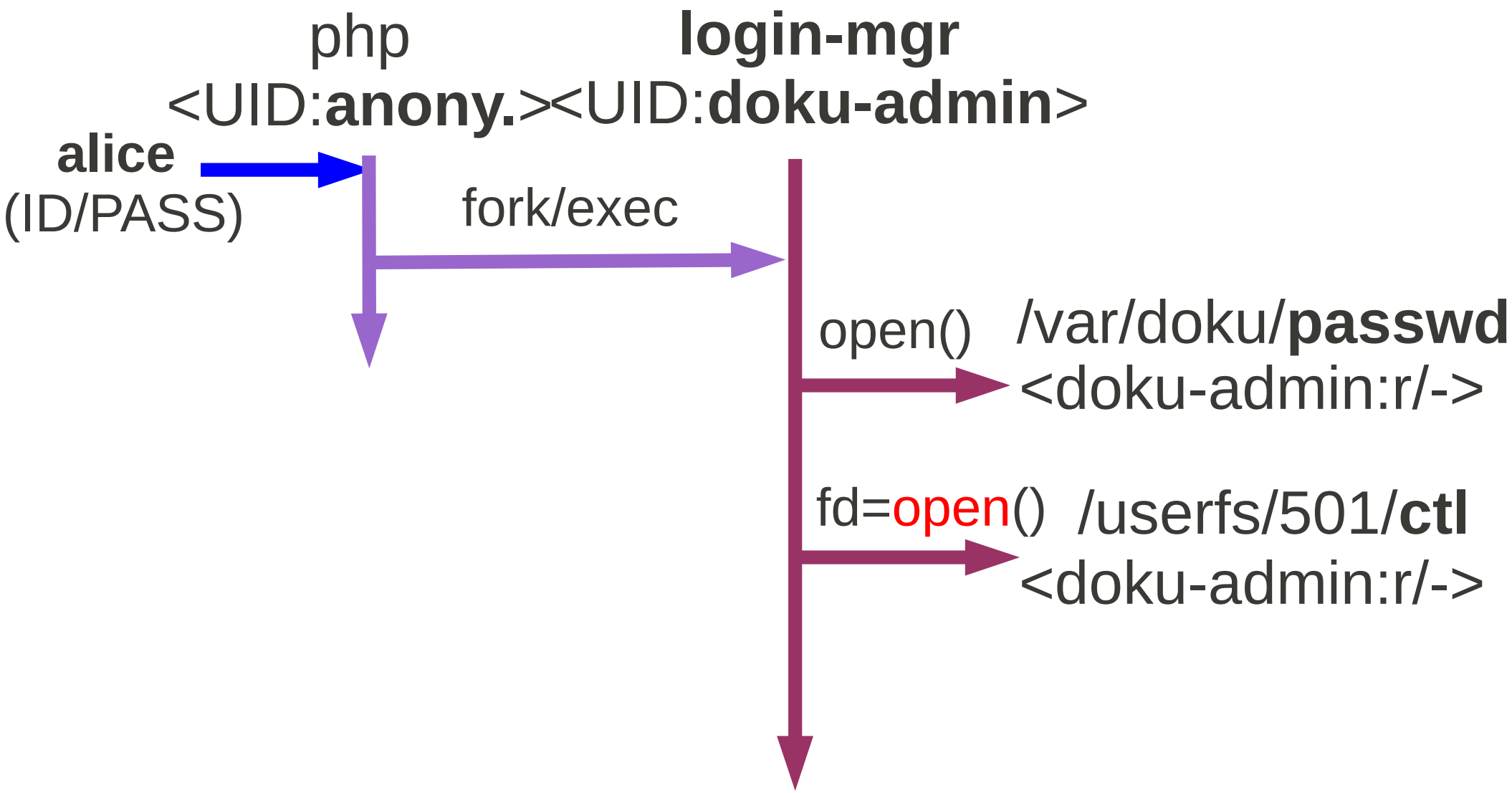
Example: Authenticating users with non-root daemon



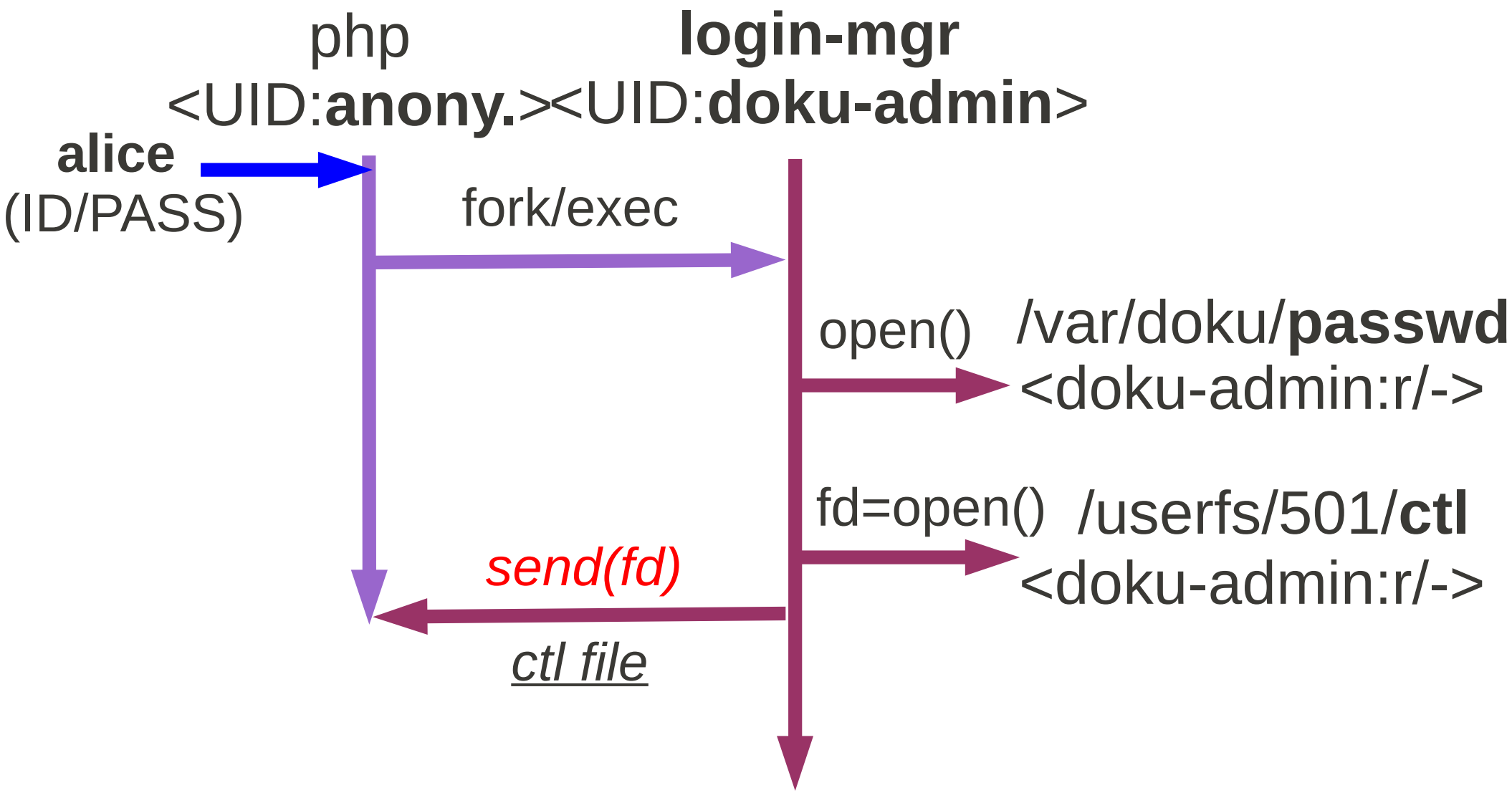
Example: Authenticating users with non-root daemon



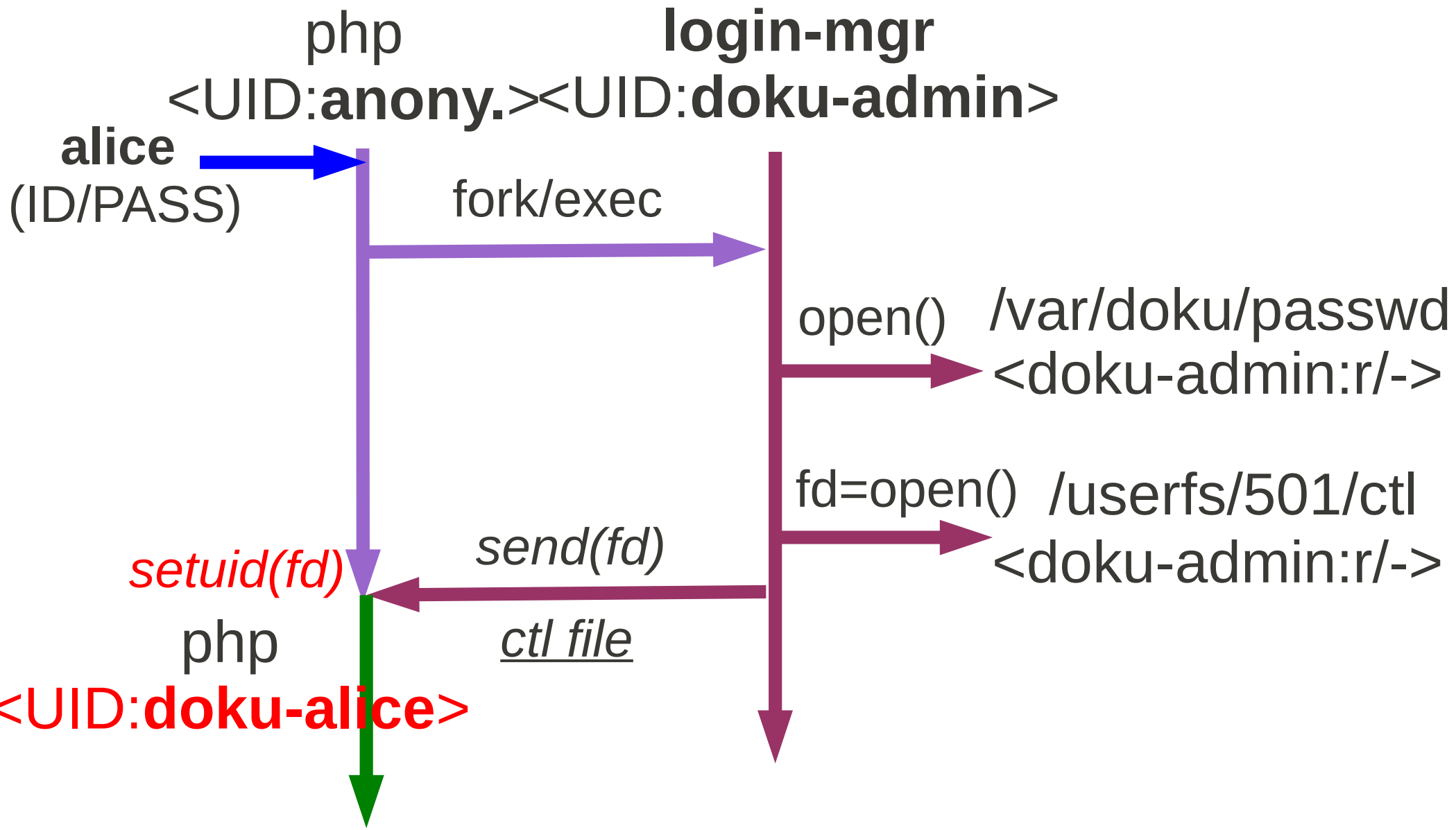
Example: Authenticating users with non-root daemon



Example: Authenticating users with non-root daemon



Example: Authenticating users with non-root daemon



Example: UID Sandboxing

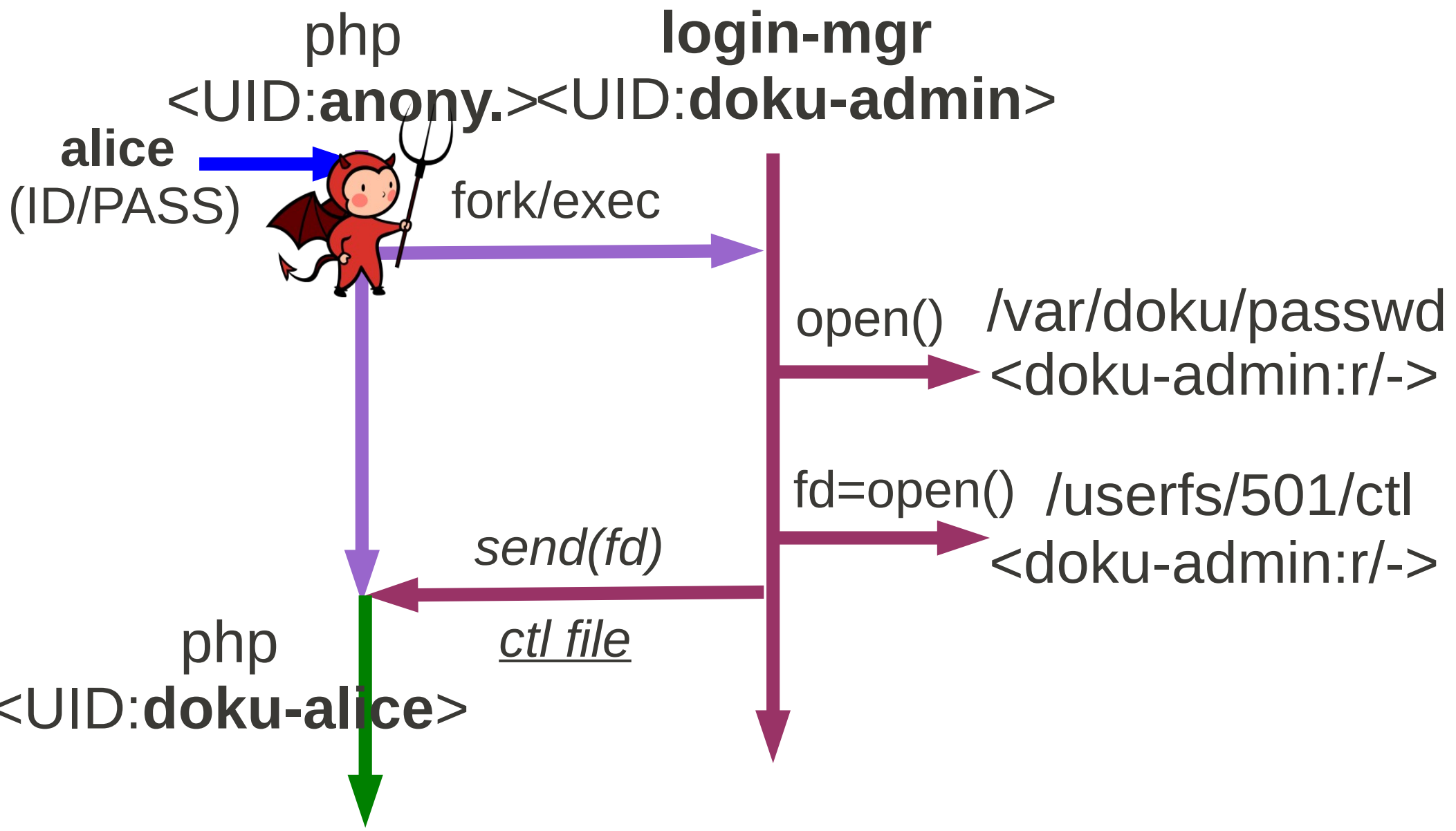
- **Initially,** launch PHP with **anonymous** UID

- After a Wiki user logins

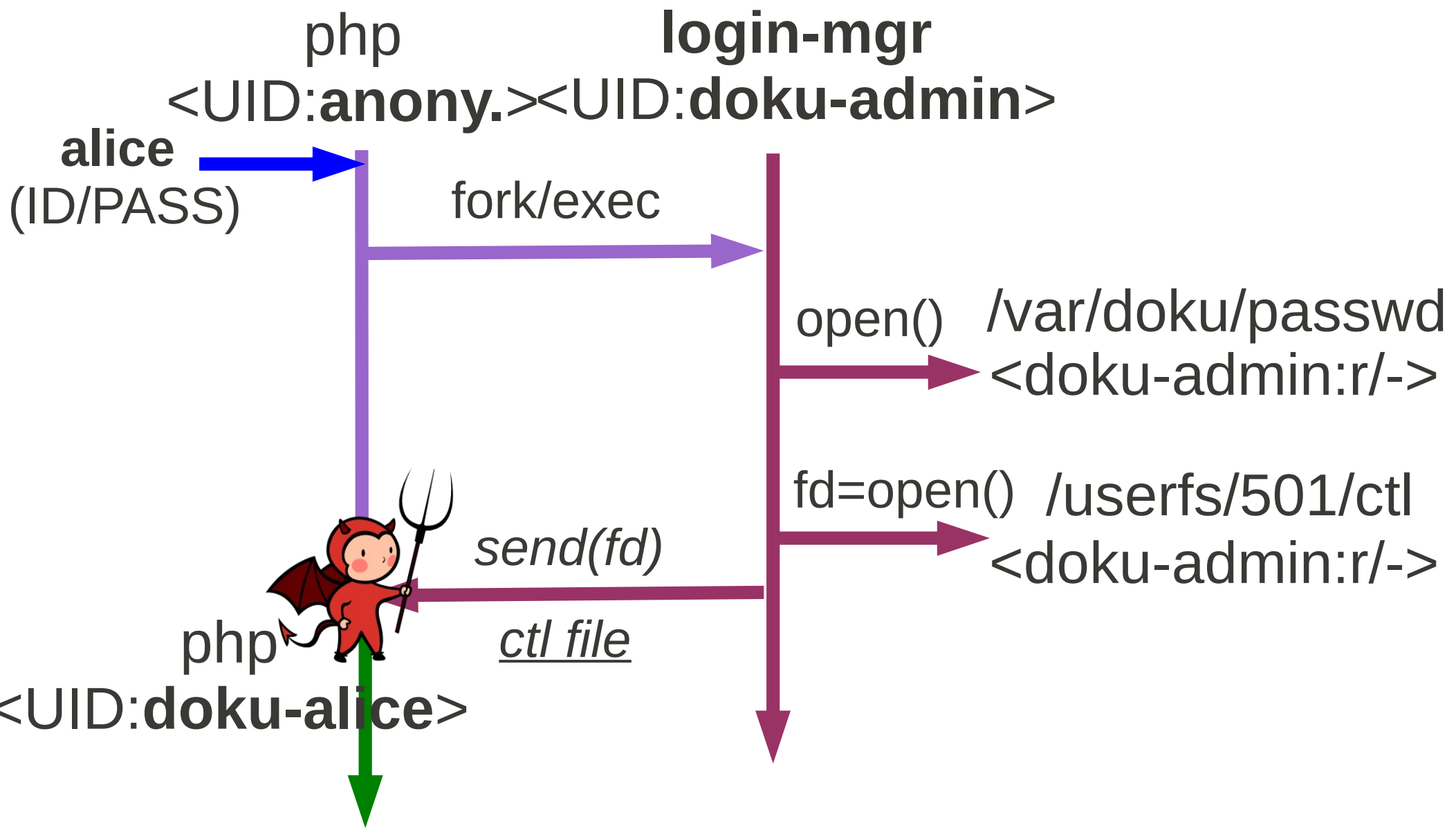
  change UID of PHP to **Wiki user's UID**

  - **login-mgr** will send the **file descriptor** of ***ctl file***
  - **receive** the file descriptor of the Wiki user
  - call **setuid()** with the received file descriptor

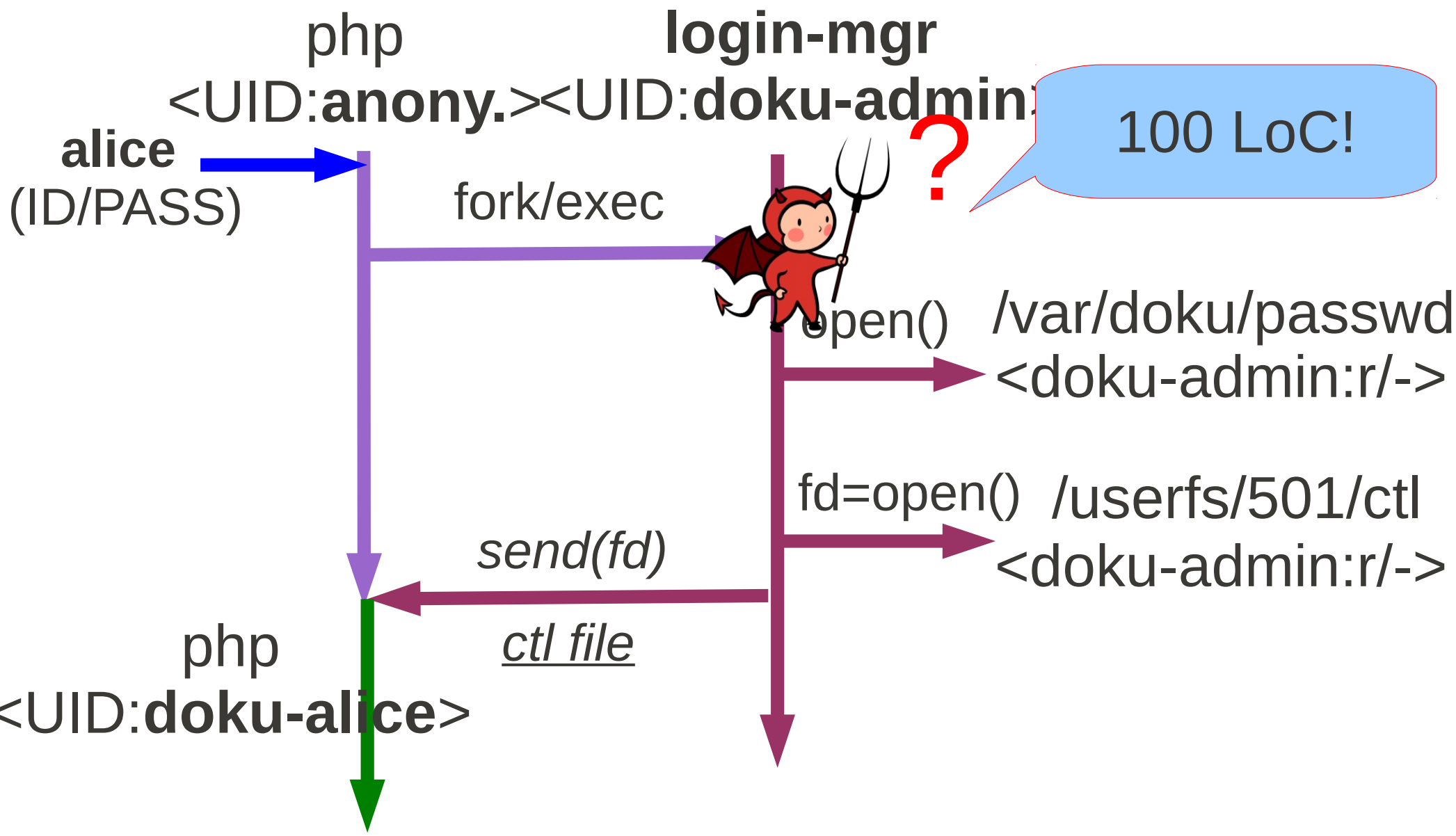
Example: UID Sandboxing



Example: UID Sandboxing



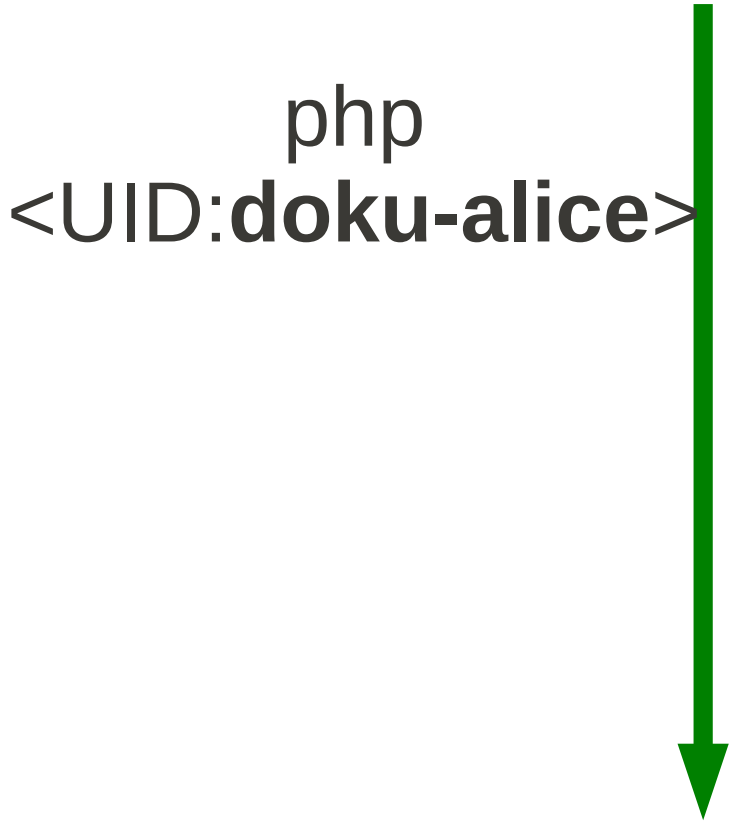
Example: UID Sandboxing



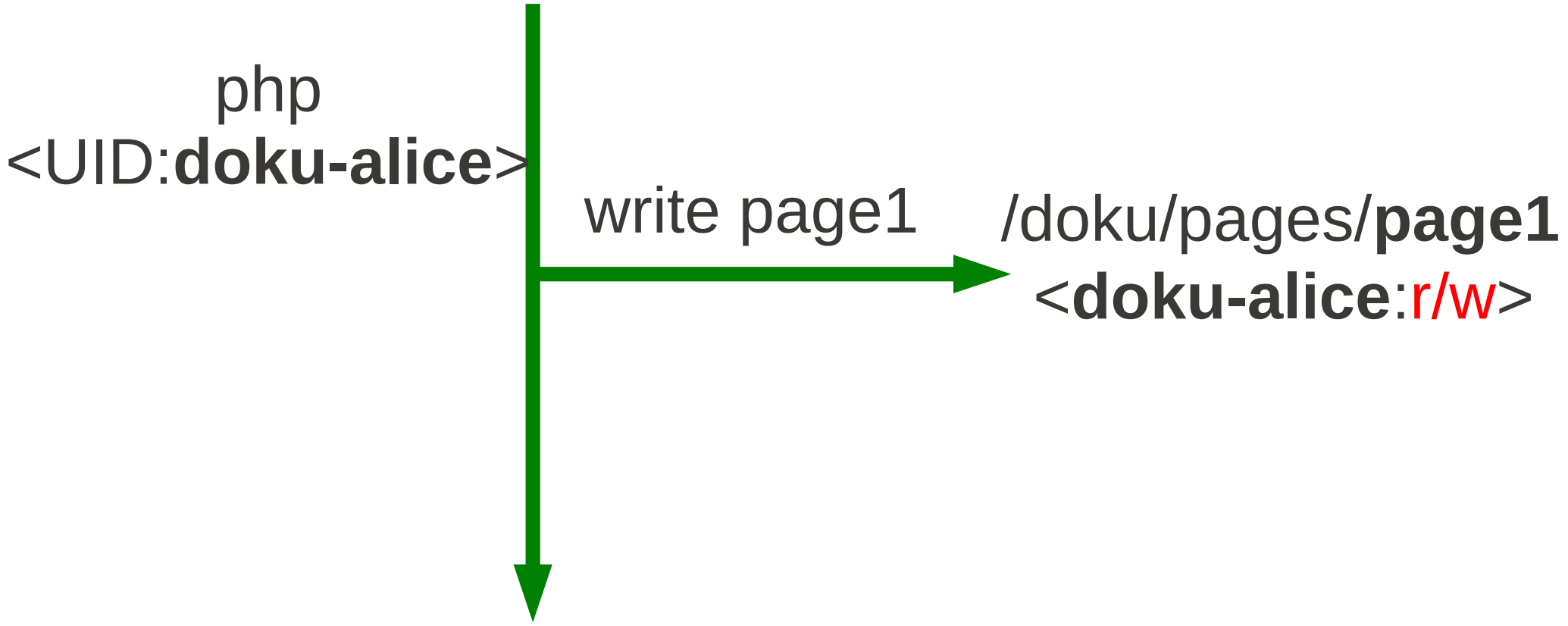
Example: Reusing well-tested ACL of filesystem

- **Save** each **Wiki page** as **a file** with owner's **UID**
- Align **ACL** of Wiki page to **the file permission**
- **OS** will enforce **security policy**

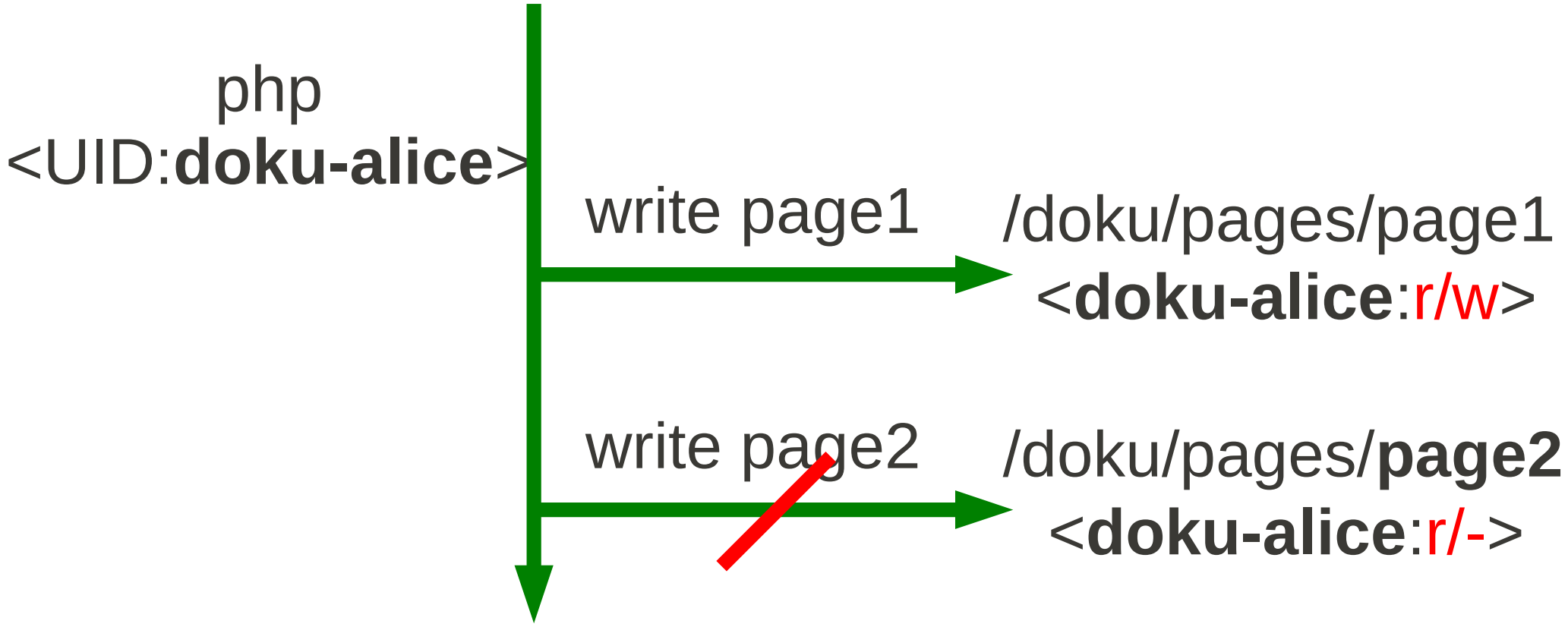
Example: Reusing well-tested ACL of filesystem



Example: Reusing well-tested ACL of filesystem



Example: Reusing well-tested ACL of filesystem



Implementation

- A single kernel module on Linux 2.6.31
  - Using **Linux Security Module** (LSM)
    ex) file_permission, inode_setattr, socket_send/recvmsg
  - Using **Netfilter** (NF)
    ex) NF_INET_LOCAL_IN/OUT
  - Using **Virtual File System** (VFS)
- **Minimal** changes of the Linux kernel
  - < **3,000** LoC Kernel Module
  - < **1,500** LoC Library

Implementing Generation Number

- Keeping system-wide **64-bit #gen**
- **Storing** #gen in ext. attributes for *setuid* binaries

  by hooking *inode_setattr()* of LSM
- **Checking** #gen when executing *setuid* binaries

  by hooking *file_permission()* of LSM

Implementing Database

- Maintaing **/etc/userfs/*** per UID
  - #gen
  - permission of *ctl file*
  - creator’s UID/GID
- **Lazily** update the database
- **mount.userfs** constructs /userfs after booting

Evaluation questions

- How **easy** is it to use UserFS?
  - Modified **5 applications**, minimal code changes
- What kinds of **security problems** can it prevent?
  - Catches **5/6 attacks** on one of the apps, DokuWiki
- What is the **performance overhead**?
  - Minimal overhead, see the paper

Applying UserFS to existing applications

Apps	LoC	Security enhancement
FTP Server	30 (+100 login-mgr)	Avoid root privilege
Chromium Browser	1	UID Sandboxing
DokuWiki	40 (+150 login-mgr)	Avoid root privilege UID Sandboxing Reuse OS protection mechanism
Cmdline Tool (su and bash)	15 (+60 bash)	Easier to switch privileges
Subsh (shell tools)	150	Easier to reduce privileges

Applying UserFS to existing applications

Apps	LoC	Security enhancement
FTP Server	30 (+100 login-mgr)	Avoid root privilege
Chromium Browser	1	UID Sandboxing
DokuWiki	40 (	
Cmdline Tool (su and bash)	15 (	
Subsh (shell tools)	150	Easier to reduce privileges

By changing **fork() -> ufork(),**
Provide **UID** for each **renderer** process

Vulnerabilities prevented

Attack Vectors	CVE
Directory Traversal	CVE-2010-0287
Insufficient Permission Check	CVE-2010-0288
Cross Site Request Forgery	CVE-2010-0289
PHP Code Upload	CVE-2006-4675
PHP Code Injection	CVE-2006-4674 CVE-2009-1960

- Prevent **5 out of 6 vulnerabilities**

  : not intended to prevent **Cross Site Req. Forgery**

- Application can **rely on OS** to **enforce policy**

  : or can even **get rid of manual ACL check** routine

Evaluation of DokuWiki

LoC of modification

Login-mgr	150 LoC
DokuWiki	40 LoC

Fetching a wiki page

Without UserFS	45 ms
With UserFS	61 ms

- **40 LoC** changes on DokuWiki (+150 LoC Login-mgr)
  - excluding 530 LoC UserFS PHP extension
- **35%** performance overhead with **extra security checks**
  - invoking **login-mgr** in every request
  - could avoid overhead with **long-running daemon**

Limitation

- UID gen# only tracked for setuid binaries
  - Reused UID owner can look at old UID's files
  - Applications should **clean up** sensitive files when deallocating UIDs
- **GID allocation** not implemented in prototype
  - Can **emulate** this by creating shared UID
- **Future work**: allow a process to have multiple concurrent UIDs (generalization of Unix GIDs)

Related Work

- **UID** sandboxing

  e.g. Android, Qmail

- System call **interposition**

  e.g. Ostia

- **New protection mechanisms** for Linux/Unix

  e.g. Flume, SELinux
  e.g. Capsicum – doesn't reuse file permission checks, would be a good complement to UserFS

- New **OS**

  e.g. HiStar, ServiceOS, KeyKOS, VSTa ...

Conclusion

- **Key idea**:

  Representing **UID as files** in **/proc-like** filesystem

- **Anyone**

  can create a **new protection principa**l

  can reuse **exiting protection mechanisms**

  without losing **compatibility**

- The **first system** to provide **egalitarian OS protection mechanisms** for Linux

  ex) UID allocation, chroot and firewall