Pass
the SALT

Pluggable Authentication Module & Typestate & Rust

PROTECT

Eddie Billoir - PhD - Cybersecurity Architect
LeChatP
02/07/26

AIRBUS

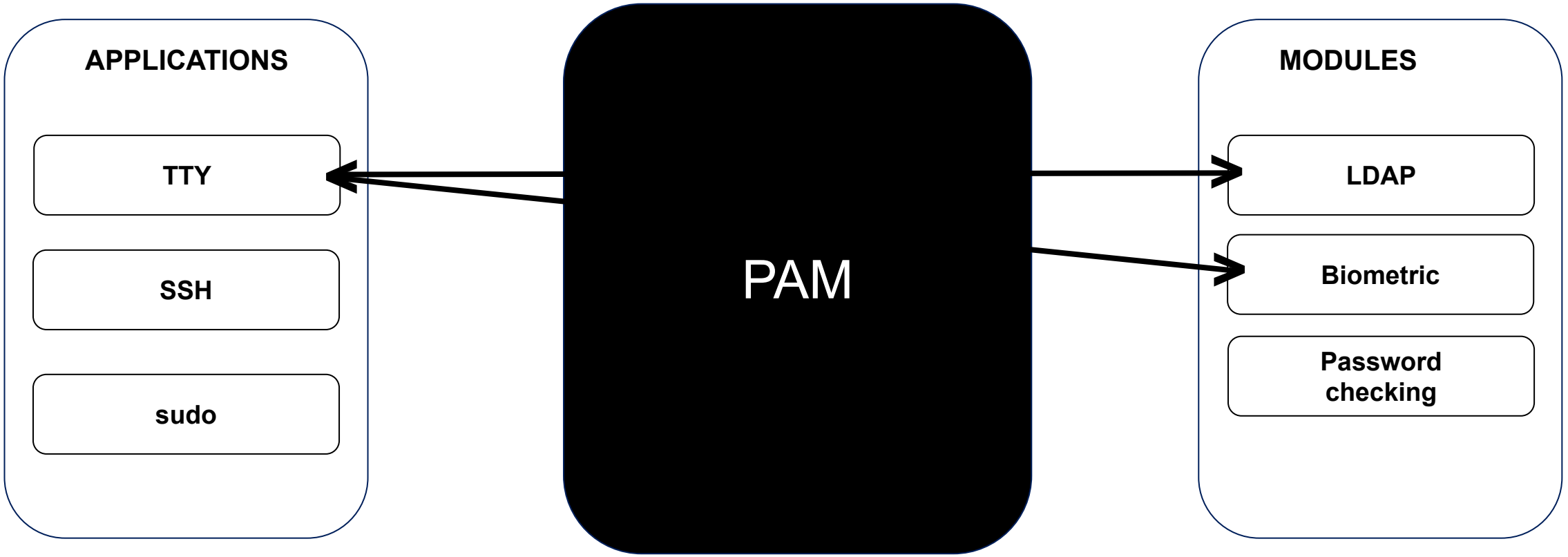
“It's the story of Pam from the TV series The Office who goes to a library, and BAM! that’s how PAM was created.”
– A Great Philosopher

AI generated image

AIRBUS

PAM as a blackbox.

Imagine PAM is a very special router…



Kind of modules for PAM

Authentication

Verifies identity,

tldr; **"Type the password"**

Account Management

Validates access rights,

tldr; **"Is this account allowed?"**

Password Update

Manages credential renewal,

tldr; **"Update your password"**

Session Configuration

Prepares session,

tldr; **"Set env vars"**

The PAM Stack

Sequential process

Configuration files define an Ordered List of modules.

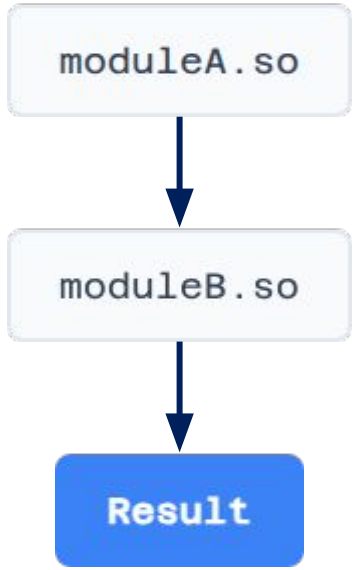
Authentication processes traverse this chain sequentially.

Discrete Decision

Each module returns a discrete Success or Failure status to the application.

Logical Synthesis

The final synthesis of the entire stack's results determines the access result.



Decision model of PAM

Required

Has to succeed. If it fails, the stack **continues** but denies access at the end.

Requisite

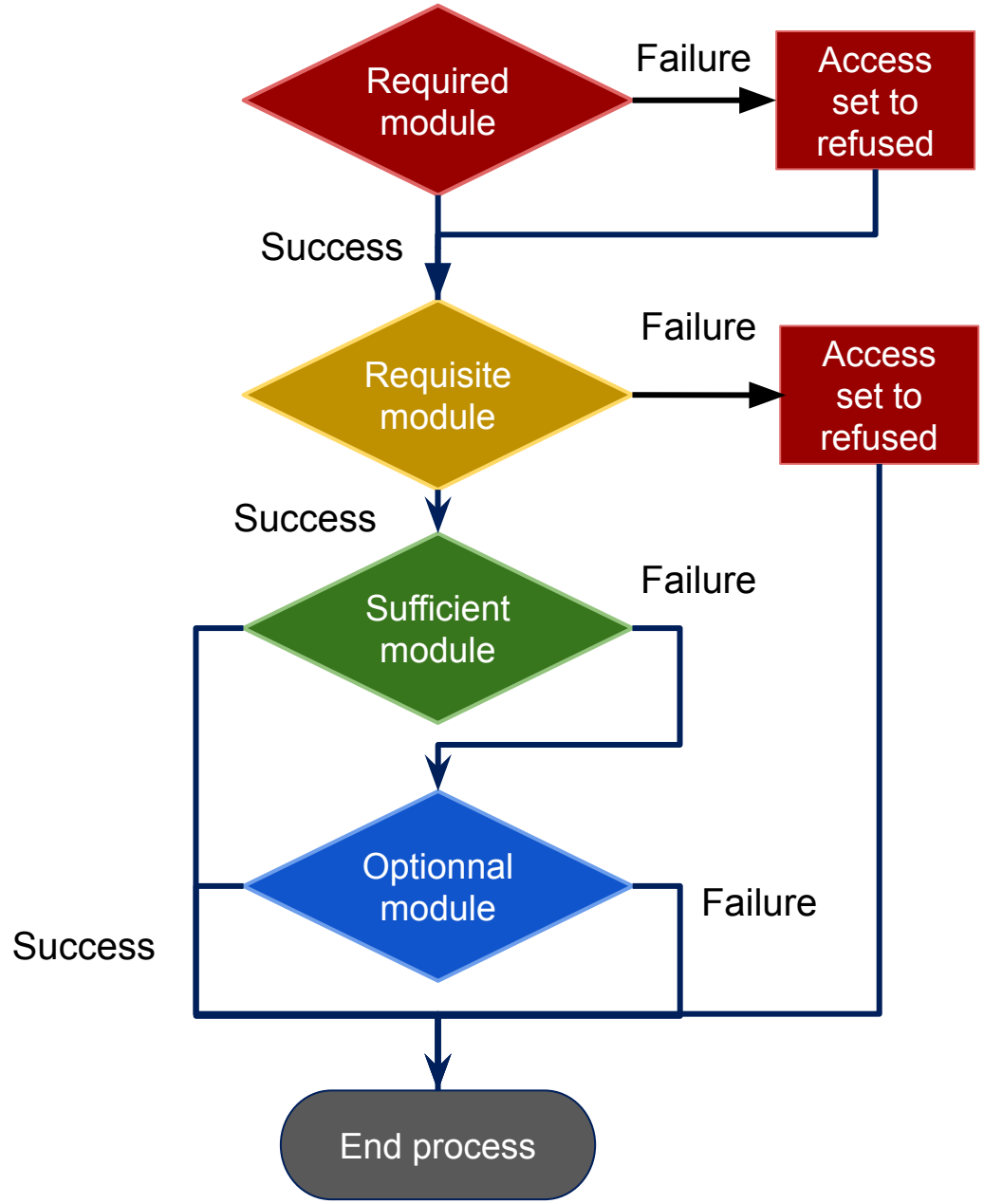
Blocking success. If this fails, stops **immediately** and denies access.

Sufficient

If it succeeds, access is granted immediately **ONLY** if no prior "Required" failed.

Optional

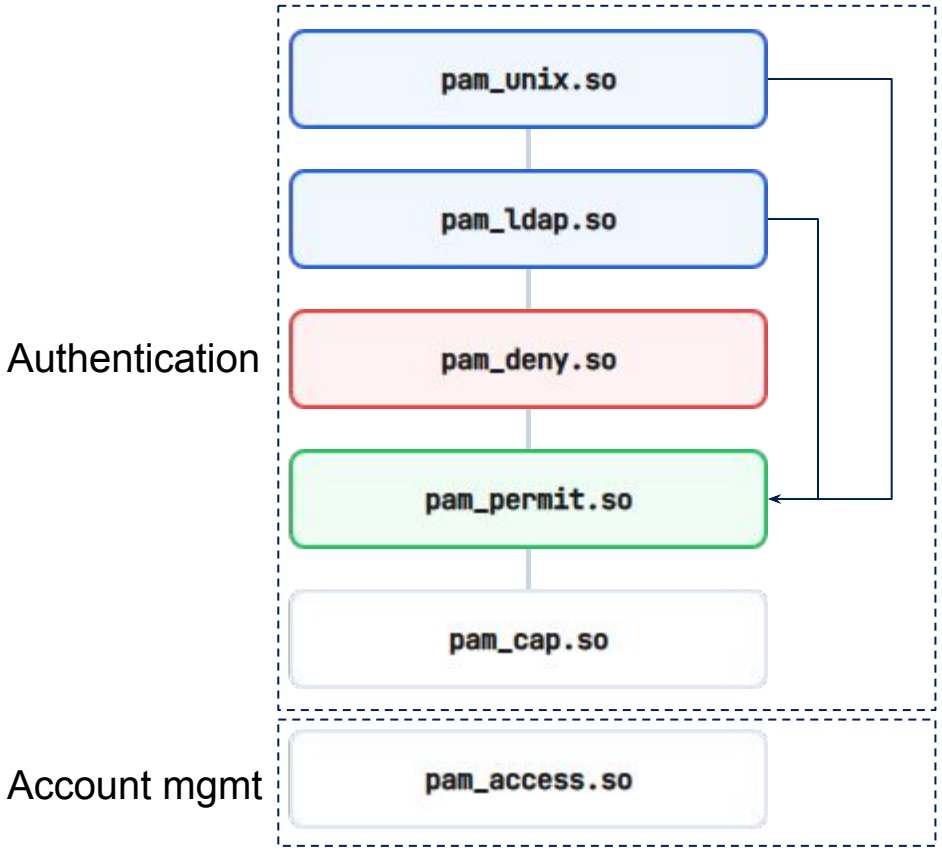
No direct impact module. Success or failure can be ignored or used as a supplementary check.



AIRBUS

A common PAM configuration

```
auth       [success=2]   pam_unix.so
auth       [success=1]   pam_ldap.so
auth       requisite     pam_deny.so
auth       required      pam_permit.so
auth       optional      pam_cap.so
account    required      pam_access.so
```



Now let's use the libpam lib…

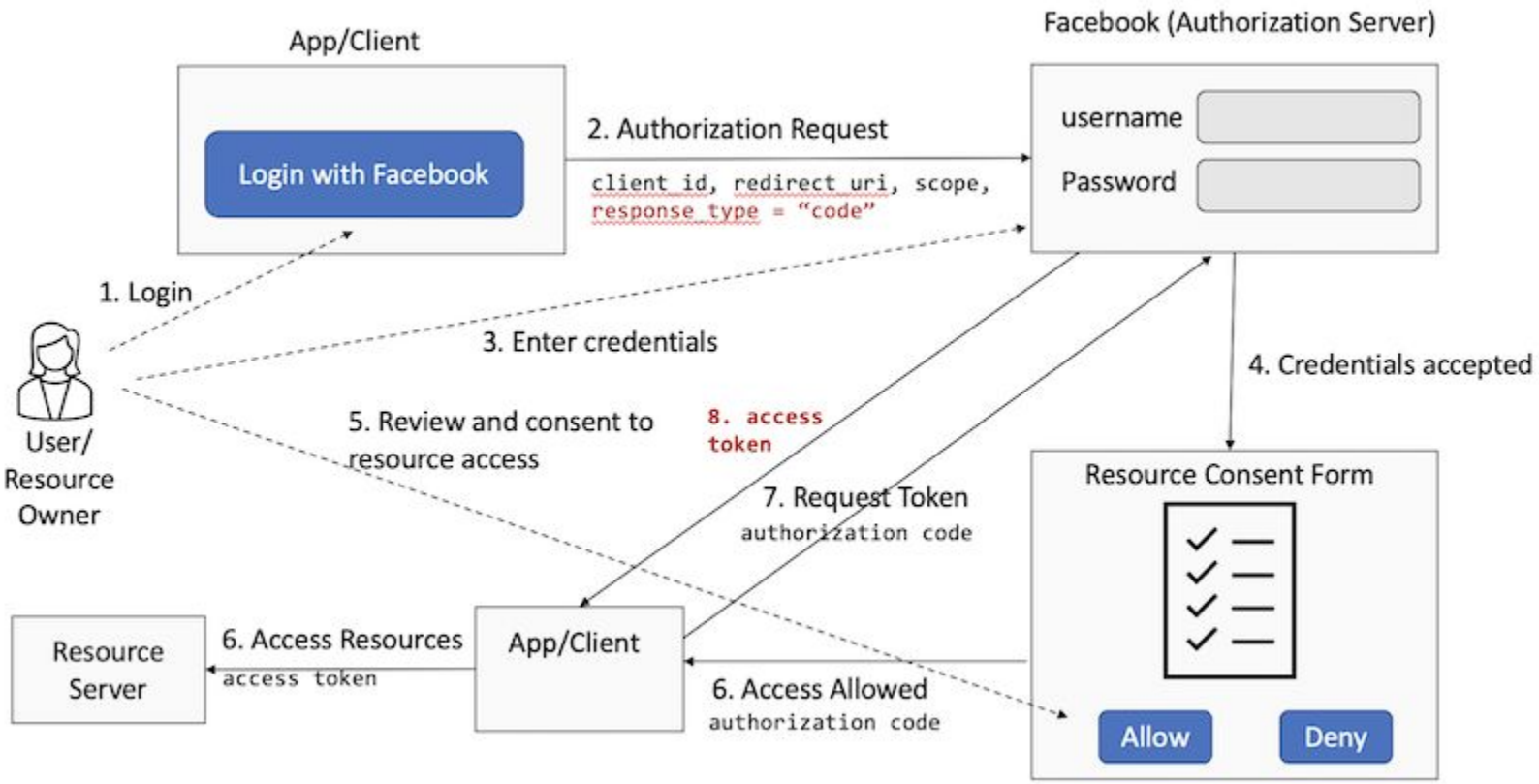
```
typedef struct pam_handle pam_handle_t;
struct pam_conv { int (*conv)(int, const struct pam_mess                    id *appdata; };

int pam_start(const char *svc, const char *usr, const st                    );
int pam_end(pam_handle_t *pamh, int status);
int pam_authenticate(pam_handle_t *pamh, int flags);
int pam_setcred(pam_handle_t *pamh, int flags);

int pam_acct_mgmt(pam_handle_t *pamh, int flags);
int pam_open_session(pam_handle_t *pamh, int flags);
int pam_close_session(pam_handle_t *pamh, int flags);
int pam_chauthtok(pam_handle_t *pamh, int flags);

int pam_set_item(pam_handle_t *pamh, int type, const voi
int pam_get_item(const pam_handle_t *pamh, int type, con
int pam_get_user(pam_handle_t *pamh, const char **user,
int pam_set_data(pam_handle_t *h, const char *n, void *d                    );
int pam_get_data(const pam_handle_t *h, const char *n, c

int pam_sm_authenticate(pam_handle_t *pamh, int flags, int argc, const char **argv);
int pam_sm_setcred(pam_handle_t *pamh, int flags, int argc, const char **argv);
int pam_sm_acct_mgmt(pam_handle_t *pamh, int flags, int argc, const char **argv);
int pam_sm_open_session(pam_handle_t *pamh, int flags, int argc, const char **argv);
int pam_sm_close_session(pam_handle_t *pamh, int flags, int argc, const char **argv);
int pam_sm_chauthtok(pam_handle_t *pamh, int flags, int argc, const char **argv);
```

AIRBUS

Now let's use the libpam lib...

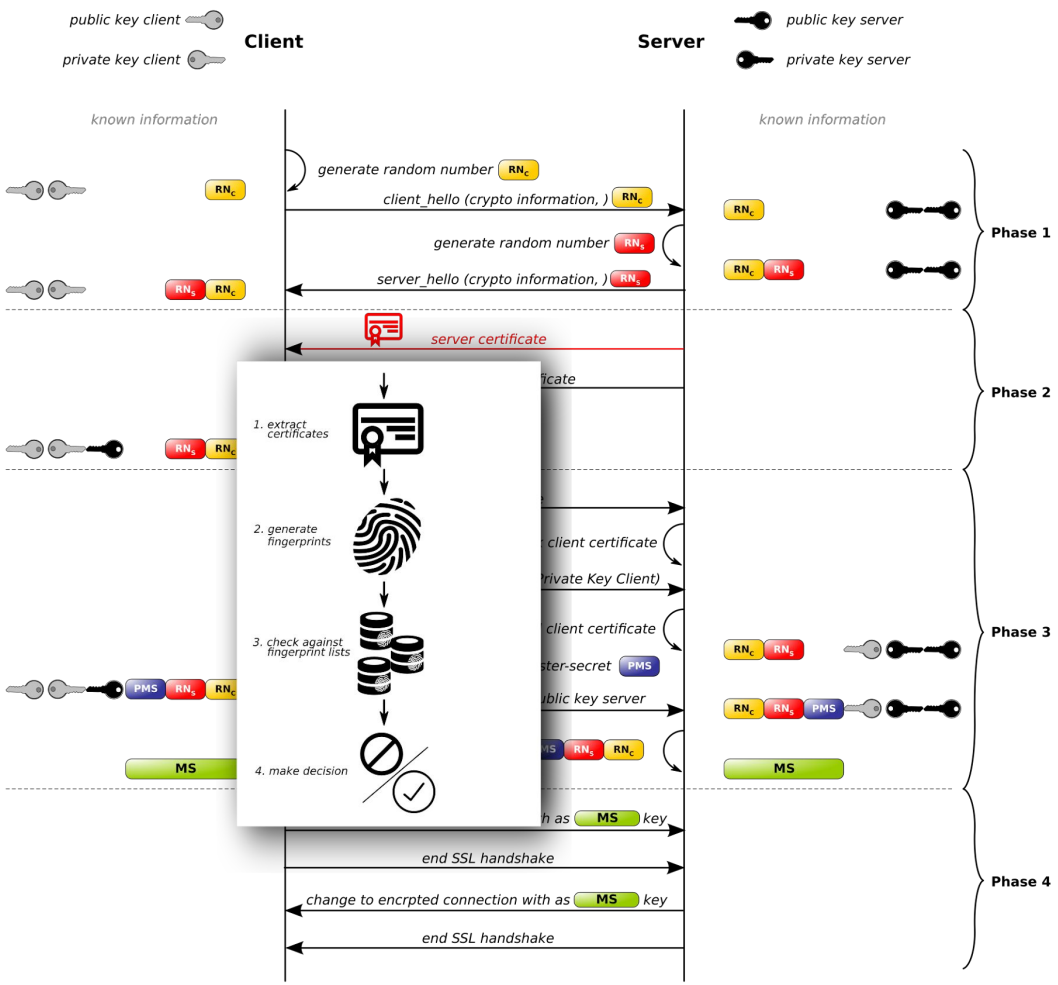
```c
#include <security/pam_appl.h>
#include <stdio.h>

extern struct pam_conv my_conv;

int login(const char *username) {
    pam_handle_t *pamh = NULL;
    int ret;
    ret = pam_start("my_service", username, &my_conv, &pamh);
    if (ret != PAM_SUCCESS) return 0;
    ret = pam_authenticate(pamh, 0);
    if (ret == PAM_SUCCESS) {
        /* forget to use pam_acct_mgmt(pamh, 0) */
        pam_open_session(pamh, 0);
        pam_end(pamh, PAM_SUCCESS);
        return 1;
    }
    printf("[ERROR] Auth failed.\n");
    pam_end(pamh, ret);
    return 0;
}
```

Some other examples



AIRBUS

Some other examples



CVE	Software	Missing / Incorrect X.509 Validation
CVE-2014-1266	Apple Secure Transport	The "goto fail" bug
CVE-2014-0092	GnuTLS	Accepting invalid certificates
CVE-2014-1959	GnuTLS	Failure to reject X.509 v1 certificates as intermediate CAs
CVE-2020-0601	Microsoft CryptoAPI	Incorrect validation of elliptic curve parameters in certificate chains
CVE-2009-2408	OpenSSL	Incomplete enforcement of BasicConstraints during certificate chain validation
CVE-2014-3504	Apache Serf	Improper hostname validation
CVE-2022-3602	OpenSSL	Improper processing of NameConstraints extension
CVE-2016-2101	OpenSSL	Incorrect certificate chain verification when using alternative chains
CVE-2021-3450	OpenSSL	Failure to properly enforce certificate authority constraints when the X509_V_FLAG_X509_STRICT flag was disabled

Let’s get Rusty

The … (highly objective and honest) … Advantages of Rust

- Memory Safety
- Strong Typing
- Avoid Undefined Behavior
- Zero-Cost Abstraction
- Modern Tooling
- Fearless Concurrency
- Comprehensive Compilation Errors
- Enforce locking rules

Enforcing the Typestate Pattern for Libpam

Define 4 States

`NotAuthenticated`

`Authenticated`

`Established`

`SessionActive`

Define our struct

```
struct PamSession<Handle, State> {
    transaction: Handle,
    _state: PhantomData<State>
}
```

```rust
impl<T: Transaction> PamSession<T, Authenticated> {
    /// Create a new session from an authenticated transaction
    pub fn new(transaction: T) -> Self {
        Self {
            transaction,
            _state: PhantomData,
        }
    }

    /// Establish credentials for the session
    /// This corresponds to pam_setcred(PAM_ESTABLISH_CRED)
    pub fn establish_credentials(mut self) -> Result<PamSession<T, CredentialsEstablished>> {
        unsafe { self.transaction.set_credentials(CredAction::Establish, BaseFlags::empty()) }?;
        Ok(PamSession {
            transaction: self.transaction,
            _state: PhantomData,
        })
    }

}
```

🍳 Nonstick, the rust crate about safe PAM implementation

thetorpedodog

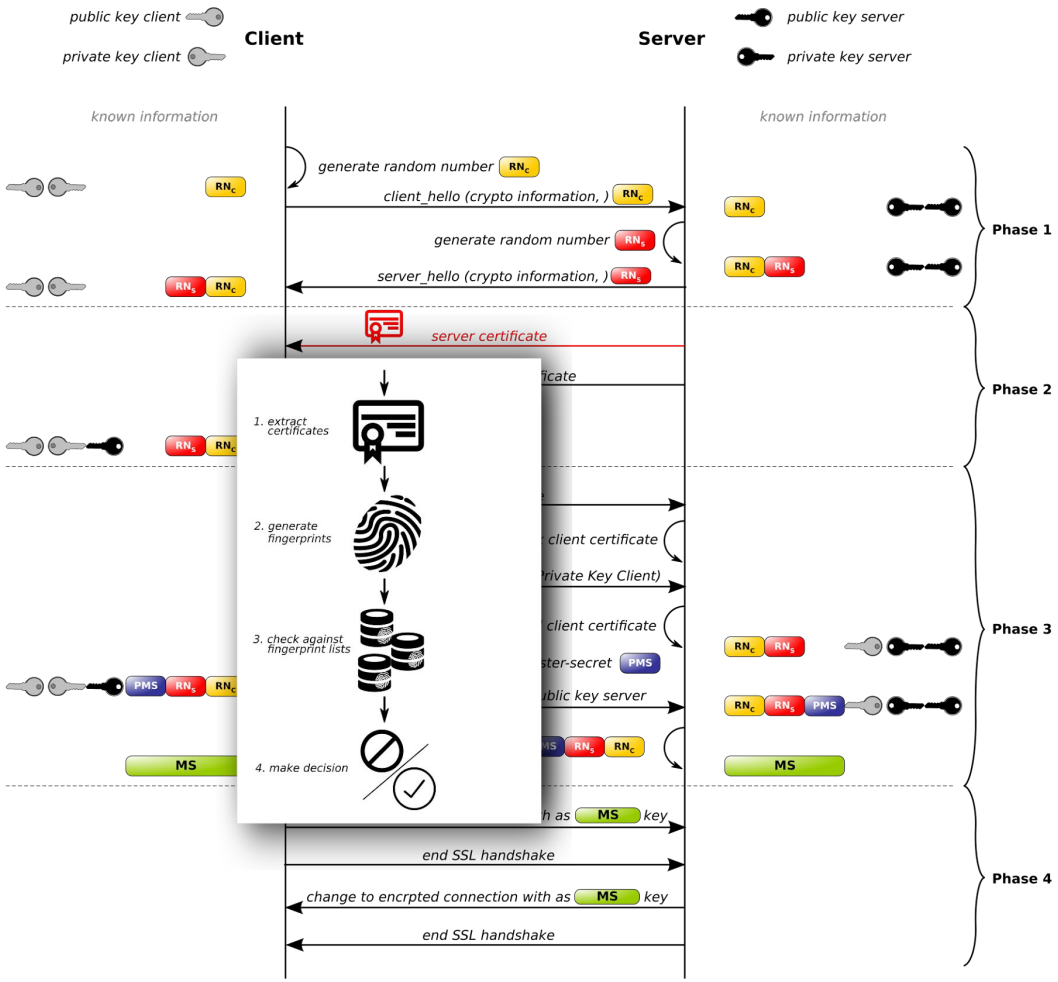
Use nonstick for interacting with PAM from both sides:

- PAM applications: call into PAM to authenticate users.
- PAM modules: write a backend that PAM uses for authentication.

It supports all known PAM implementations:

- Linux-PAM, used on most (all?) Linux distributions.
- OpenPAM, used on most BSDs, including Mac OS.
- Sun's PAM, used on Solaris and derivatives like Illumos.

Imagine a world where we can tRust



PROTECT

AIRBUS