CSC8204
Secure Software Development Coursework 2024

Name: UJWAL GOWDA KRISHNA MURTHY

Student Number:240716473

Part A Applied Risk Analysis:

Below is a sample solution approach. The final deliverable should be adapted, refined, and formatted as appropriate. The following is based on the methodology discussed in McGraw's "Software Security: Building Security In" and the NIST guidelines referenced in the lecture materials. Page references to the Tokened report and McGraw's principles are illustrative.

A. Business Goals (Praxis High Integrity Systems / Tokeneer Project)

1. **Business Goals**

G1: Establish Praxis as a leading provider of high-assurance, formally verified secure software solutions.
G2: Show compliance with high government and security standards-for example, NSA-to gain more authoritative credibility.
G3: Increase brand reputation to attract high-value future contracts for long-term client engagement.
G4: Enhanced market positioning by demonstrating capability to provide cost-effective, reliable, and secure systems.

Ranking According to NIST Business Goal Classification

Using a perspective that emphasizes reputation, compliance, and growth:

1. G3 (Brand Reputation/Market Credibility)
2. G2 (Regulatory/Standards Compliance)
3. G1 (High-Integrity Technical Demonstration)
4. G4 (Market Expansion/Profitability)

Risk ID	Business Risk Description	Indicator	Likelihood (NIST)*	Impact (NIST)*	Severity (LxI)	Rationale
BR1	Failure to meet the security claims leading to reputational damage and loss of trust in Praxis's capabilities.	Negative external evaluations, security weaknesses identified post-release.	(Medium)	(High)	(High)	Reputation is critical; a high-profile failure reduces future contracts.
BR2	Non-compliance with sponsor/government standards leading to project rejection or financial penalty.	Audit findings, compliance checks fail	(Low)	(High)	(Medium)	Compliance is an absolute requirement; a single failure is severe.
BR3	Delays or cost overruns that undermine profitability and erode client confidence.	Schedule slips, budget reports	2 (Medium)	3 (Medium)	6 (Medium)	Timely delivery and budget control are essential for sustained business growth.

NIST likelihood and impact scale assumed: 1=Low, 2=Medium-Low, 3=Medium, 4=High, 5=Very High.

C. Technical Risks (from Tokeneer Report and Artefacts)
Based on McGraw's software security touchpoints and the "10 Best Practices" from McGraw, e.g., principles of secure design, careful specification, threat modelling, we identify five technical risks. Each stems from potential gaps or assumptions in the Tokeneer documentation and process.

Tech Risk ID	Technical Risk Description	Relevant Security Principle/Touchpoint	Likelihood	Impact	Existing Controls (From Tokeneer)	Rationale

TR1	**Inadequate cryptographic key management could expose authentication secrets.**	**Principle: Manage cryptographic keys properly; Touchpoint: Architecture Risk Analysis**	**1 (Low)**	**4 (High)**	**Formal verification of crypto routines, strict coding standards**	**Key management is critical to system integrity and confidentiality.**
TR2	**Potential incomplete input validation could allow unexpected data input paths.**	**Principle: Validate all inputs; Touchpoint: Code Review**	**1 (Low)**	**3 (Medium)**	**Formal methods & proofs reduce this but do not fully eliminate environmental assumptions**	**Input validation is a common attack vector; ensuring no oversight is essential.**
TR3	**Incorrect environmental assumptions in formal specifications might fail under real-world conditions.**	**Principle: Understand the Environment; Touchpoint: Requirements Analysis**	**2 (Medium)**	**4 (High)**	**Rigorous spec verification, but environment changes might not be fully captured**	**Even perfect code is vulnerable if assumptions about the environment are flawed.**
TR4	**Incomplete threat modeling could miss emerging attack vectors.**	**Principle: Know Your Threats; Touchpoint: Architecture Risk Analysis / Pen Testing**	**2 (Medium)**	**3 (Medium)**	**Formal design review, partial threat assessments**	**Threat modeling is never exhaustive; new threats evolve.**
TR5	**Dependencies on external components not verified with the same rigor may introduce vulnerabilities.**	**Principle: Secure the weakest link; Touchpoint: Supply Chain Security Review**	**1 (Low)**	**3 (Medium)**	**Strict software process controls, but external code may not be equally assured**	**External interfaces can introduce unverified code or assumptions.**

D. Risk Synthesis

Business Goal G3 (reputation) and BR1 (reputation damage) are closely related to TR1 (in case of failure of key management, the system cannot be trusted) and TR3 (environment assumptions failure may lead to operational faults causing harm to reputation).

Business Goal G2 (compliance) and BR2 (non-compliance) are related to TR1 (cryptographic compliance), TR4 (missed threats leading to non-compliance), and TR5 (unverified external components could fail compliance checks).

Timely Delivery can be affected if BR3 has to address missed threats or flawed assumptions through costly rework caused by TR3 or TR4.

Tech Risk ID	Mitigation Approach	Justification

TR1	**Implement Hardware Security Modules (HSMs) and regularly review key management protocols; adhere to NIST-recommended cryptographic standards.**	**Ensures strong, compliant key management and mitigates a high-impact compromise.**
TR2	**Implement runtime input sanitation and fuzz testing in addition to formal methods; integrate secure coding standards (e.g., CERT).**	**Complements formal verification with dynamic checks to catch real-world input anomalies.**
TR3	**Conduct scenario-based testing and extended operational environment simulations; involve domain experts to refine assumptions.**	**Prevents specification drift and ensures correctness in real deployments.**
TR4	**Regularly update threat models using current threat intelligence; engage external penetration testing teams periodically.**	**Staying current with evolving threats ensures ongoing compliance and brand protection.**
TR5	**Introduce strict supplier vetting procedures, code provenance checks, and sandboxing for external components.**	**Ensures that third-party dependencies meet the same assurance level, reducing supply-chain risk.**

Rationale Summary :

This applied risk analysis uses the methodology by McGraw to tie together business and technical risks systematically. The selected risks also correspond to the focus of the Tokeneer project on formal methods, security claims, and following high integrity standards.

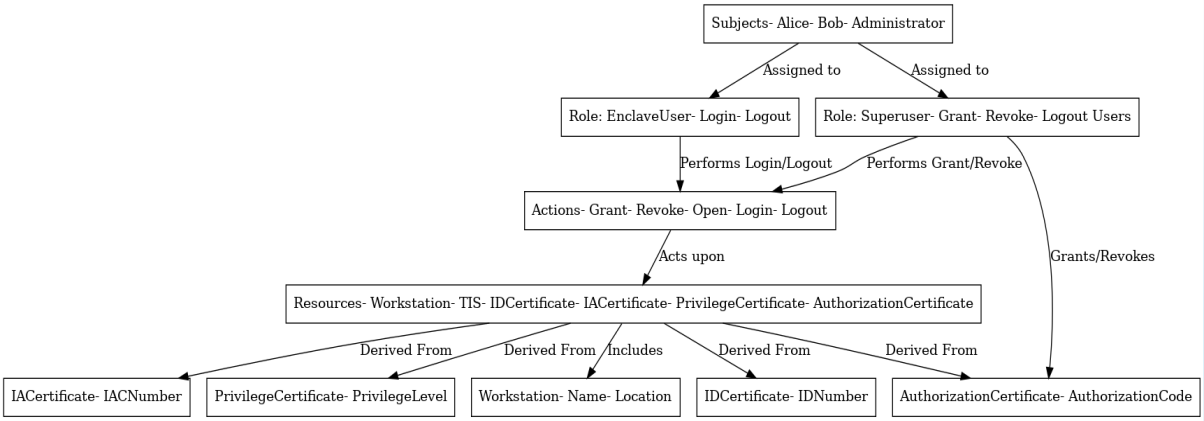
Choice of Business Risks: The three main business risks are reputational damage, non-compliance, and project overruns. They were chosen because they go to the heart of Praxis's strategic aims: having a trusted brand (BR1), maintaining stringent regulatory standards (BR2), and ensuring profitable and timely delivery (BR3).

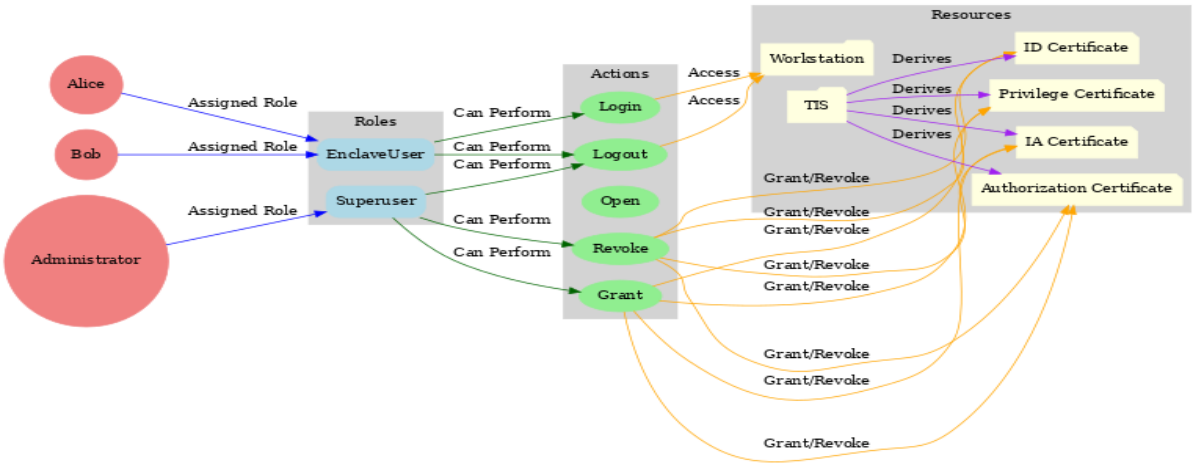
Choice of Technical Risks: TR1, cryptographic key management; TR2, input validation; TR3, environmental assumption correctness; TR4, completeness of threat modeling; and TR5, external dependencies, were selected because they may potentially invalidate security claims and compliance. They represent the most plausible vulnerabilities in a formal verification context where correctness is assumed but not guaranteed without complementary controls.

Mitigations: Each mitigation will align with one or more of McGraw's 10 best practice principles. Examples include the following: cryptographic protocol hardening (TR1), and validation of inputs (TR2) are core secure design practices. Environmental testing, TR3, and refresh of threat modeling, TR4, ensure that the formal model reflects real-world evolution. Vetting of external dependencies, TR5, secures the supply chain, one of the most important-and neglected-domains.

This coherent approach ensures that the technical controls directly support the key business goals of reputation, compliance, and growth under rigorous conditions, as illustrated by the Tokeneer project.

Part B Secure UML Design:





Rationale
Design Goals: The aim of this UML class diagram is to model a security policy for the Tokeneer ID station, TIS, using SecureUML concepts. We follow the Model-Driven Security approach to specify Roles, Resources, Actions, Permissions, and Assignments such that RBAC-based authentication and authorization are enforced. Certificate handling and workstation access are managed as stipulated.
Main Points from the Requirements:
1.Subjects and Roles:
.Entities: Alice, Bob, Administrator.

.Roles: Enclave User, Superuser.
.We assign Alice and Bob to the Enclave User role, Administrator to the Superuser role.
2.Resources:
.Workstation: Users can login / logout here.
.TIS: The central system that maintains and enforces certificate handling.
.Certificates: ID Certificate, IACertificate, PrivilegeCertificate, AuthorizationCertificate. Those are resources that can be granted or revoked. By having an Authorization Certificate it is possible to log in.
3.Actions:
.Grant, Revoke: Certificate handling.
.Login, Logout: Controlling the Workstation.
.Open: The possibility of opening a resource, for example, a workstation session.
4.Policy Rules:
.Any user holding an Authorization Certificate is permitted to log in to a Workstation.
.Certificates are created/enacted by the TIS.
.A Superuser is able to assign/revoke any certificate and log out any user.
.An EnclaveUser may log in/log out a Workstation only if they possess an AuthorizationCertificate.
.Superuser can also issue these certificates via the TIS.
SecureUML Mapping:
.Role classes represent the roles in the system.
.Permission classes represent allowed actions on resources by roles.
.Subject-Role Assignment defines which subjects belong to which roles.
.Action and Resource classes represent what can be done and to what.
Rationale for the Design:
We show Enclave User and Superuser as specializations of a Role class.
Subjects (Alice, Bob, Administrator) are instances assigned to these roles.
Permissions link a Role to a Resource and the allowed Actions. For example, the Superuser role has permissions for Grant and Revoke certificates (IACertificate, IDCertificate etc.) against the TIS Resource.
The Enclave User role is granted permission to Login and Logout on a Workstation resource but only if the user can present an Authorization Certificate; this can be modelled as a constraint of the model (e.g. a note or OCL constraint).The Superuser role is also granted permission to perform a Logout of users on Workstations, so they can manage sessions.

Part C Formal Modelling

```
// Enumerated types for Clearance and Privileges
enum CLEARANCE_CLASS {
  // Levels of clearance, ordered by sensitivity
  unmarked,        // No specific clearance assigned
  unclassified,    // Unrestricted and general information
  restricted,      // Sensitive but lower level
  confidential,    // Moderately sensitive information
  secret,          // Highly sensitive information
  topSecret        // The most sensitive level of clearance
}
enum PRIVILEGE {
  // Privilege levels for users of the system
  user,            // Regular user with no administrative privileges
  guard,           // User responsible for physical security
  auditManager,    // Responsible for reviewing and archiving logs
  securityOfficer  // High-level user with full administrative access
}
enum ADMINOP {
  // Administrative operations available in the system
  overrideLock,    // Override a locked state
  archiveLog,      // Archive system logs
  updateData,      // Update sensitive data
  shutdownOp       // Shut down the system
}
// Clearance class
class Clearance {
  var clearance: CLEARANCE_CLASS  // The clearance level for this object
  // Constructor to initialize clearance level to 'unmarked'
  constructor() ensures clearance == CLEARANCE_CLASS.unmarked {
    clearance := CLEARANCE_CLASS.unmarked; // Default clearance level
  }
  // Ghost function to return the Clearance object with the lower level
  ghost function minClearance(c1: Clearance, c2: Clearance): Clearance
    reads c1, c2
  {
    // Compare clearance levels and return the lower one
    if c1.clearance <= c2.clearance then c1 else c2
  }
}

// Function to return available operations for a privilege
```

```
function availableOps(p: PRIVILEGE): set<ADMINOP> {
  match p {
    case PRIVILEGE.guard => {ADMINOP.overrideLock}            // Guard can only override locks
    case PRIVILEGE.auditManager => {ADMINOP.archiveLog}       // Audit Manager can archive logs
    case PRIVILEGE.securityOfficer => {ADMINOP.updateData, ADMINOP.shutdownOp}  // Security Officer has broader permissions
    case PRIVILEGE.user => {}                                 // Regular users have no admin operations
  }
}

// Token class
class Token {
  var tokenID: int                    // Unique identifier for the token
  var idCert: Certificate             // ID Certificate associated with the token
  var privilegeCert: Certificate      // Privilege Certificate
  var iAndACert: Certificate          // Integrity and Authentication Certificate
  var authCert: optional<Certificate>   // Optional Authorization Certificate

  // Predicate to check if the token is valid
  predicate ValidToken() {
    idCert != null &&                                 // ID Certificate must exist
    privilegeCert != null &&                          // Privilege Certificate must exist
    iAndACert != null &&                              // I&A Certificate must exist
    privilegeCert.tokenID == tokenID &&               // Privilege Cert must reference the correct Token ID
    privilegeCert.idCertID == idCert.id &&            // Privilege Cert must reference the correct ID Cert
    iAndACert.tokenID == tokenID &&                   // I&A Cert must reference the correct Token ID
    iAndACert.idCertID == idCert.id                   // I&A Cert must reference the correct ID Cert
  }

  // Predicate to check if the token has a valid Authorization Certificate
  predicate TokenWithValidAuth() {
    authCert? &&                                      // Authorization Certificate must exist
    authCert!.tokenID == tokenID &&                   // Auth Cert must reference the correct Token ID
    authCert!.idCertID == idCert.id                   // Auth Cert must reference the correct ID Cert
  }

  // Predicate to check if the token is current (valid at a specific time)
  predicate CurrentToken(now: TIME) {
    ValidToken() &&                                   // Token must be valid
    idCert.validityPeriod.contains(now) &&            // ID Cert must be valid at the current time
    privilegeCert.validityPeriod.contains(now) &&     // Privilege Cert must be valid at the current time
    iAndACert.validityPeriod.contains(now)            // I&A Cert must be valid at the current time
  }
}

// Abstract Certificate type
trait Certificate {
  var id: int                   // Unique identifier for the certificate
  var tokenID: int              // Token ID referenced by the certificate
  var idCertID: int             // ID Certificate ID referenced by the certificate
  var validityPeriod: set<TIME>     // Time period during which the certificate is valid
}

// Helper type for optional values
type optional<T> = ts: set<T> | |ts| <= 1  // Either empty or contains a single value

type TIME = nat  // Time is represented as a natural number
```

Part D:

1.Sums Method
Specification:
The method ensures that:

Code:

```
method Sums(x: int, y: int) returns (m: int, n: int)
  ensures m > n
{
  var a: int;
  m := x;
  n := y;
  a := 2 * m + n;
  n := n - 1;
  m := a;
}
```

Specification:

- Postcondition (ensures): `m > n`

We need to find the weakest precondition on `x` and `y` that ensures `m > n` at the end.

Step-by-step derivation:

1. **Final assignment:** `m := a`

   The postcondition after `m := a` must be `m > n`. Right before the assignment, we substitute `m` by `a` in the postcondition:

   Postcondition: `m > n`
   After substitution (for the line `m := a`):
   `a > n`

   Therefore, immediately before the line `m := a`, we need:

   a>na > na>n

2. **Previous assignment:** `n := n - 1`

Before this assignment, let the old value of `n` be `n_old`. After `n := n - 1`, we have `n = n_old - 1`.

We want `a > n` to hold after this line. Thus:

a>n ⟹ a>(nold−1)a > n \implies a > (n_old - 1)a>n⟹a>(nold−1)

Since `n` refers to the value before the assignment on the left side of the implication, we rewrite the condition before `n := n - 1` as:

a>n−1a > n - 1a>n−1

Hence, immediately before `n := n - 1`:

a>n−1a > n - 1a>n−1

3. **Previous assignment:** `a := 2 * m + n`

   We now have `a > n - 1` as the required condition after `a` is assigned. To find the condition before `a := 2*m + n`, we substitute `a` with `2*m + n`:

   From `a > n - 1`:

   2∗m+n>n−12*m + n > n - 12∗m+n>n−1

   Simplifying:

   2∗m+n>n−1 ⟹ 2∗m>−1 ⟹ m>−122*m + n > n - 1 \implies 2*m > -1 \implies m > -\frac{1}{2}2∗m+n>n−1⟹2∗m>−1⟹m>−21

   Thus, before `a := 2*m + n`, we need:

   m>−12m > -\frac{1}{2}m>−21

4. **Assignments to m and n from x and y:** `m := x; n := y`

   Initially, after `m := x` and `n := y`, we have `m = x` and `n = y`.

   Substituting `m = x` into `m > -\frac{1}{2}`, we get:

   x>−12x > -\frac{1}{2}x>−21

   This is the condition on the inputs `x` and `y` that ensures the final postcondition will hold. Notice that `y` does not appear in the final precondition due to how the inequalities simplified.

Final weakest precondition for the Sums method:

x>−12x > -\tfrac{1}{2}x>−21

If we assume `x` and `y` are integers, this effectively means:

x≥0x \geq 0x≥0

Thus, the required precondition is that `x` must be greater than -0.5 (and hence nonnegative if integers are expected).

2) Update Alarms Method :

CODE :

```
datatype ALARM = silent | alarming
method UpdateAlarms(doorAlarm: ALARM, auditAlarm: ALARM) returns (alarm:
ALARM)
  ensures (alarm == alarming) <==>
        (doorAlarm == alarming) || (auditAlarm == alarming)
{
  if doorAlarm == alarming || auditAlarm == alarming
  {
    alarm := alarming;
  }
  else
  {
    alarm := silent;
  }
}
```

Specification:

Postcondition (ensures): (alarm == alarming) <==> (doorAlarm == alarming || auditAlarm == alarming)

We need to verify that the final value of alarm matches the given equivalence. We want to see if any precondition is required.

Step-by-step reasoning:

The condition inside the if: doorAlarm == alarming || auditAlarm == alarming

If doorAlarm == alarming || auditAlarm == alarming is true, then alarm is set to alarming.
After this, alarm == alarming is true, which matches the right-hand side of the equivalence, since (doorAlarm == alarming || auditAlarm == alarming) is true. Thus the ensures condition holds.

If doorAlarm == alarming || auditAlarm == alarming is false, then alarm is set to silent.
After this, alarm == alarming is false, and since (doorAlarm == alarming || auditAlarm == alarming) is also false in this branch, the equivalence (alarm == alarming) <==> (doorAlarm == alarming || auditAlarm == alarming) holds again (false <==> false).

No restrictions on inputs:

The postcondition is directly enforced by the code's logic. There are no conditions on doorAlarm or auditAlarm that need to be assumed to prove the correctness of the ensures clause. The program works correctly for all possible values of doorAlarm and auditAlarm.

Final weakest precondition for UpdateAlarms method:

true\text{true}true

No precondition is needed; the method is correct for all inputs.