ProtectSUS - Product Requirements Document (PRD)

Version: 1.0
Date: January 17, 2026
Project Type: AI-Powered Code Security Analysis Platform
Target: NexHacks Hackathon (24-hour build)

📋 Table of Contents

1. Executive Summary

1.1 Product Vision

ProtectSUS is an AI copilot for automated code security analysis that integrates directly into GitHub workflows. It uses a multi-agent system to detect vulnerabilities, assess risks, and automatically generate fix pull requests - learning from user feedback over time to improve accuracy.

1.2 Core Value Proposition

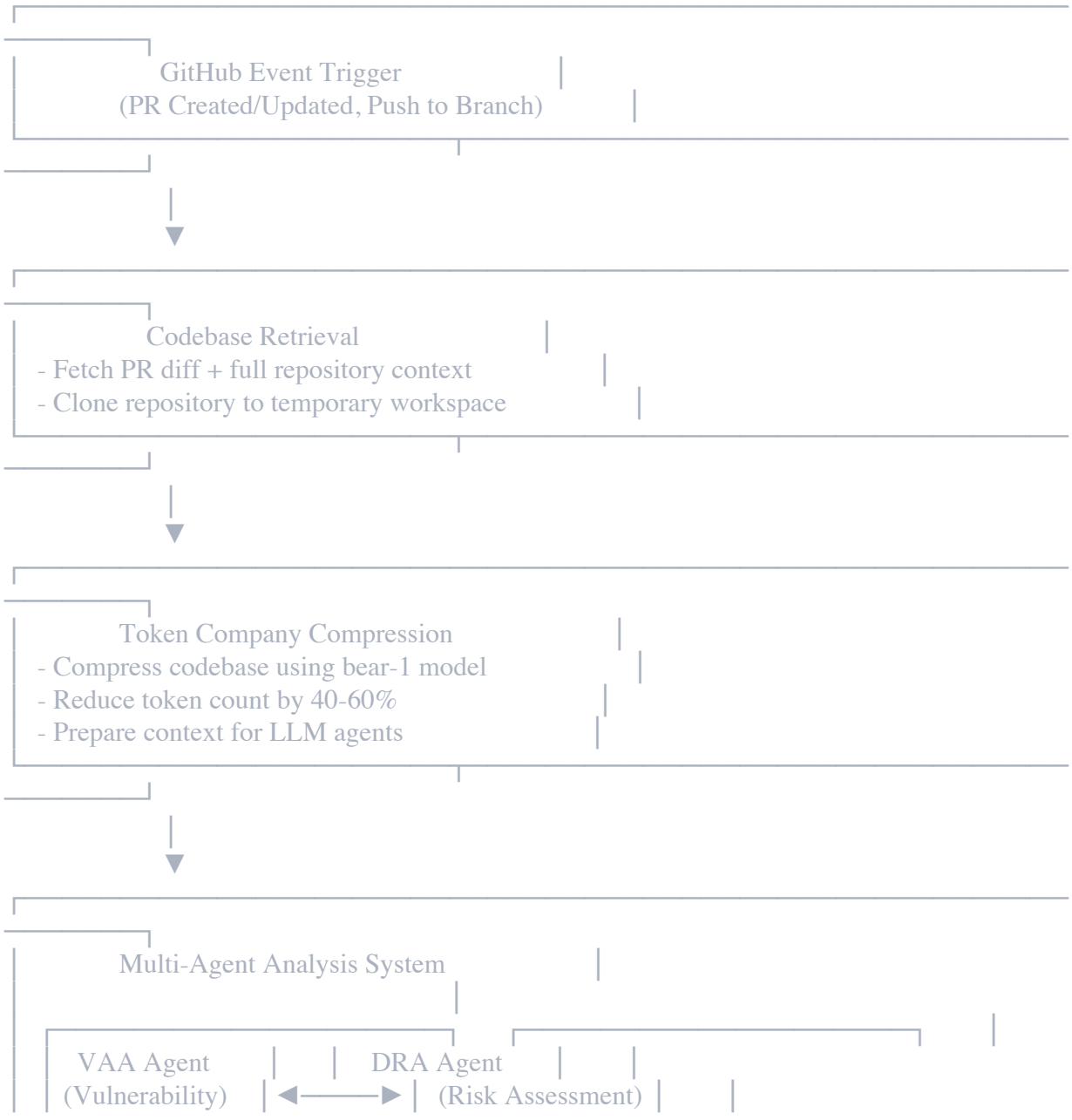
- **Automated Security**: Catches vulnerabilities before they reach production
- **Multi-Agent Intelligence**: Two specialized agents debate to find comprehensive issues
- **Self-Improving**: Reinforcement learning adapts to team preferences
- **Interactive Explanations**: Knowledge graph-powered chat explains vulnerabilities
- **Efficient Context**: Token compression allows analysis of entire codebases

1.3 Target Users

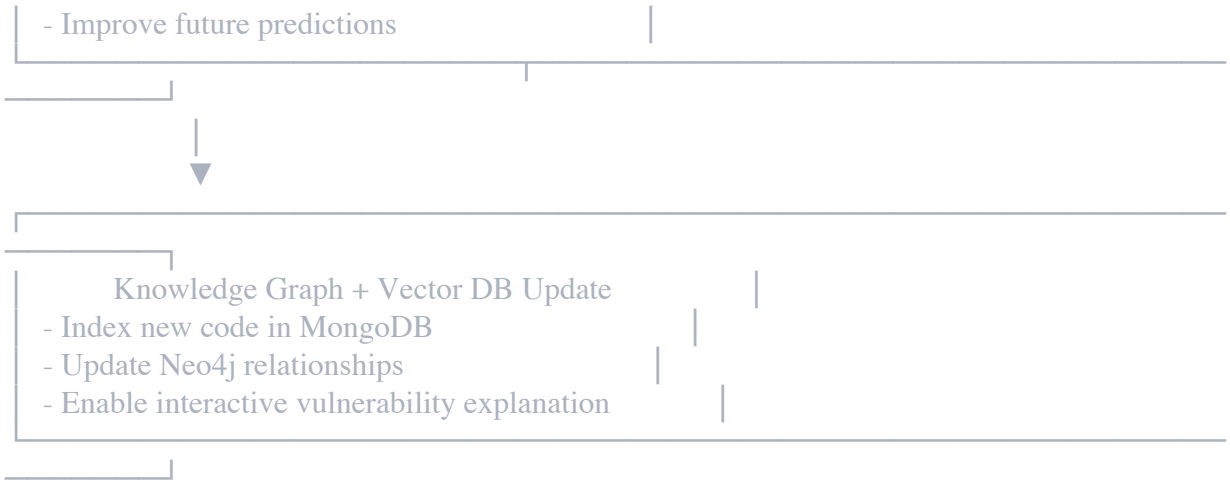
- Individual developers
- Small to medium-sized development teams
- Open-source projects
- Companies with active GitHub repositories

2. Core Product Concept

2.1 High-Level Flow




- OWASP Top 10
- CVE scanning
- Code smells
- Breaking changes
- Dependency risks
- Impact analysis
(Max 5-6 rounds)
Summary Agent
- Synthesize findings
- Generate fixes
- Create report
Automated Fix PR Creation
- Create new branch with fixes
- Generate detailed PR description
- Add vulnerability report as comment
- Request user approval
User Feedback Loop
User approves (👍) or denies (👎) + optional comment
Reinforcement Learning Model Update
- Record user decision
- Extract features from analysis
- Retrain preference model

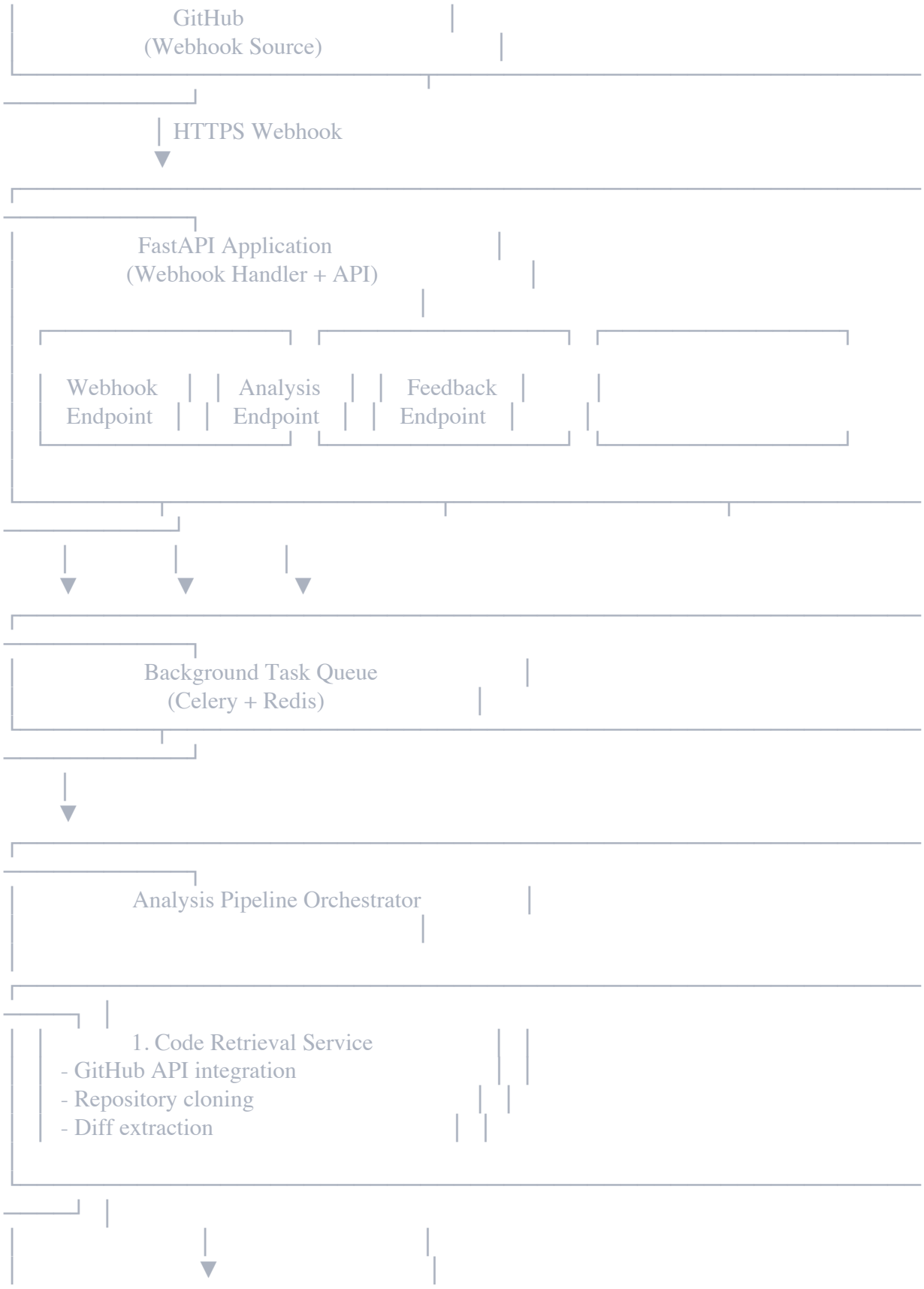


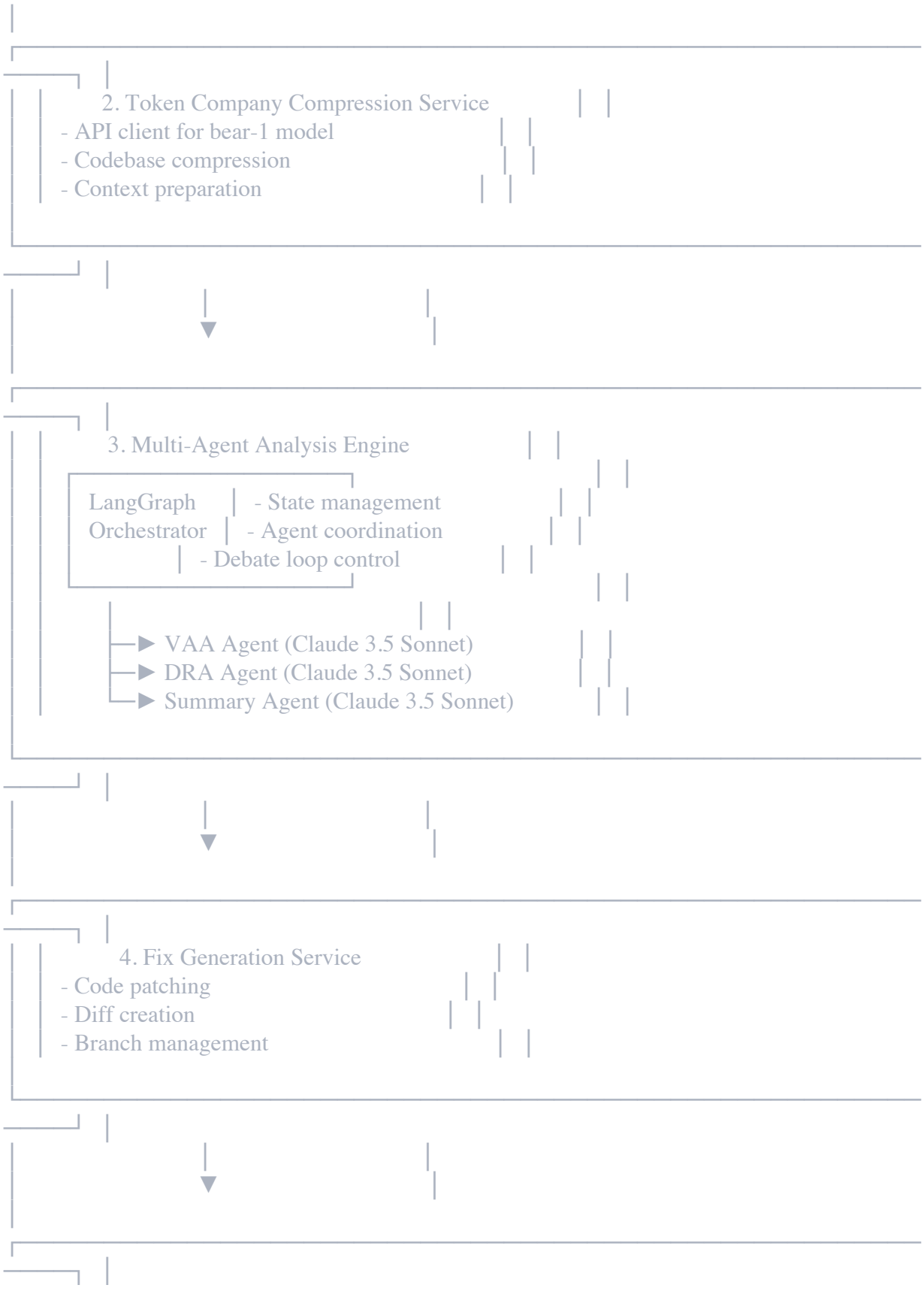
2.2 Key Features

1. **Multi-Agent Security Analysis**
   - VAA (Vulnerability Assessment Agent)
   - DRA (Dependency Risk Agent)
   - Structured debate with max 5-6 rounds
   - Summary agent for synthesis
2. **Automated Fix Generation**
   - Creates pull request with fixes
   - Detailed vulnerability reports
   - Code-level explanations
3. **Reinforcement Learning**
   - Learns from user approvals/denials
   - Adapts to team coding standards
   - Reduces false positives over time
4. **Interactive Knowledge Graph**
   - Explains vulnerability impact
   - Shows code dependencies
   - Traces risk propagation
5. **Efficient Context Management**
   - Token Company compression
   - Analyze entire repositories
   - Cost-effective LLM usage

3. Technical Architecture

3.1 System Architecture Diagram


GitHub
(Webhook Source)
HTTPS Webhook
FastAPI Application
(Webhook Handler + API)
Webhook
Endpoint
Analysis
Endpoint
Feedback
Endpoint
Background Task Queue
(Celery + Redis)
Analysis Pipeline Orchestrator
1. Code Retrieval Service
- GitHub API integration
- Repository cloning
- Diff extraction


2. Token Company Compression Service
- API client for bear-1 model
- Codebase compression
- Context preparation
3. Multi-Agent Analysis Engine
LangGraph Orchestrator
- State management
- Agent coordination
- Debate loop control
VAA Agent (Claude 3.5 Sonnet)
DRA Agent (Claude 3.5 Sonnet)
Summary Agent (Claude 3.5 Sonnet)
4. Fix Generation Service
- Code patching
- Diff creation
- Branch management

3.2 Technology Stack

Layer	Technology	Version	Purpose

Backend Framework	FastAPI	0.109.0	REST API, webhooks
Task Queue	Celery	5.3.4	Background job processing
Message Broker	Redis	7.2	Celery backend, caching
Agent Framework	LangGraph	0.0.40	Multi-agent orchestration
LLM Integration	LangChain	0.1.0	LLM abstractions
LLM Provider	Anthropic Claude	3.5 Sonnet	Primary reasoning engine
Vector Database	MongoDB Atlas	7.0	Code embeddings, storage
Knowledge Graph	Neo4j AuraDB	5.15	Code relationships
Compression	Token Company API	bear-1	Input compression
Observability	Arize Phoenix	Latest	Agent debugging
ML Framework	scikit-learn	1.4.0	Reinforcement learning
GitHub API	PyGithub	2.1.1	GitHub integration
Code Analysis	tree-sitter	0.20.4	AST parsing
Container Runtime	Docker	24.0	Containerization
Process Manager	Supervisor	4.2.5	Service management
Web Server	Nginx	1.24	Reverse proxy

4. Detailed Feature Specifications

4.1 Feature 1: GitHub Webhook Integration

Description: Receive and process GitHub webhook events for PR creation, updates, and pushes.

Technical Requirements:

- Support GitHub webhook signature verification (HMAC-SHA256)
- Handle webhook events: `pull_request`, `push`
- Async processing to avoid webhook timeouts
- Idempotent event handling (prevent duplicate processing)
- Rate limiting and retry logic

API Endpoint:

```
POST /api/v1/webhook/github
```

Request Headers:

```
X-GitHub-Event: pull_request
X-GitHub-Delivery: <UUID>
```

```
X-Hub-Signature-256: sha256=<signature>
Content-Type: application/json
```

Request Body (Pull Request Event):

```
json
{
  "action": "opened",
  "number": 123,
  "pull_request": {
    "id": 987654321,
    "number": 123,
    "title": "Add new feature",
    "head": {
      "ref": "feature-branch",
      "sha": "abc123..."
    },
    "base": {
      "ref": "main",
      "sha": "def456..."
    },
    "diff_url": "https://github.com/...",
    "url": "https://api.github.com/repos/..."
  },
  "repository": {
    "id": 123456,
    "full_name": "user/repo",
    "clone_url": "https://github.com/user/repo.git",
    "default_branch": "main"
  },
  "sender": {
    "login": "username",
    "id": 12345
  }
}
```

Processing Logic:

1. Verify webhook signature
2. Parse event payload
3. Queue background analysis task
4. Return 202 Accepted immediately
5. Process in background worker

Error Handling:

- Invalid signature: Return 401 Unauthorized
- Unsupported event: Return 200 OK (acknowledge but ignore)

- Processing failure: Retry with exponential backoff

4.2 Feature 2: Code Retrieval & Context Building

Description: Fetch repository code, PR diffs, and build comprehensive context for analysis.

Components:

4.2.1 Repository Cloner

python

```
class RepositoryCloner:
    """
    Clones GitHub repositories to temporary workspace
    """

    def clone_repository(self, clone_url: str, branch: str) -> Path:
        """
        Clone repository at specific branch

        Args:
            clone_url: Git clone URL
            branch: Branch name to checkout

        Returns:
            Path to cloned repository
        """
        pass

    def get_pr_diff(self, repo_path: Path, base_sha: str, head_sha: str) -> str:
        """
        Get unified diff between two commits
        """
        pass
```

4.2.2 Code Parser

python

```
class CodeParser:
    """
    Parse code files and extract metadata
    """

    def parse_repository(self, repo_path: Path) -> List[CodeFile]:
```

```
        """
        Parse all code files in repository

        Returns list of CodeFile objects with:
        - File path
        - Language
        - Imports
        - Functions/classes
        - Line count
        - AST representation
        """
        pass

    def extract_changed_files(self, diff: str) -> List[str]:
        """
        Extract list of changed files from diff
        """
        pass
```

4.2.3 Context Builder

```
python
class ContextBuilder:
    """
    Build comprehensive context for LLM analysis
    """

    def build_context(
        self,
        repo_path: Path,
        pr_diff: str,
        changed_files: List[str]
    ) -> AnalysisContext:
        """
        Build context including:
        - Full file contents of changed files
        - Related dependencies
        - Configuration files
        - Test files
        - Documentation

        Returns:
            AnalysisContext object with all relevant code
        """
        pass
```

File Structure:
```
/tmp/protectsus-workspace/
├── repos/
│   └── <repo_id>/
│       ├── <sha>/
│       │   └── <cloned_repo>/
│       └── metadata.json
└── analysis/
    └── <analysis_id>/
        ├── context.json
        ├── diff.patch
        └── compressed_context.json
```

4.3 Feature 3: Token Company Compression

Description: Compress codebase context using Token Company's bear-1 model to reduce token usage.

API Integration:

python

```python
class TokenCompanyClient:
    """
    Client for Token Company compression API
    """

    def __init__(self, api_key: str):
        self.api_key = api_key
        self.base_url = "https://api.thetokencompany.com/v1"

    async def compress(
        self,
        text: str,
        compression_level: float = 0.5
    ) -> CompressionResult:
        """
        Compress text using bear-1 model

        Args:
            text: Input text to compress
            compression_level: Target compression ratio (0.4-0.6)

        Returns:
```

```
        CompressionResult with:
        - compressed_text
        - original_tokens
        - compressed_tokens
        - compression_ratio
        - quality_score
    """
    endpoint = f"{self.base_url}/compress"

    payload = {
        "text": text,
        "model": "bear-1",
        "compression_level": compression_level
    }

    headers = {
        "Authorization": f"Bearer {self.api_key}",
        "Content-Type": "application/json"
    }

    async with httpx.AsyncClient() as client:
        response = await client.post(
            endpoint,
            json=payload,
            headers=headers
        )
        response.raise_for_status()
        return CompressionResult(**response.json())
```

Compression Strategy:

1. **Aggressive Compression** (0.6 ratio) for:
   - Unchanged files (context only)
   - Test files
   - Documentation
2. **Moderate Compression** (0.4 ratio) for:
   - Changed files
   - Dependencies of changed files
   - Configuration files
3. **No Compression** for:
   - PR diff (always full fidelity)
   - Security-critical files (auth, crypto)

Data Models:

python

```
from pydantic import BaseModel
```

```
class CompressionResult(BaseModel):
    compressed_text: str
    original_tokens: int
    compressed_tokens: int
    compression_ratio: float
    quality_score: float
    processing_time_ms: int
```

4.4 Feature 4: Multi-Agent Analysis System

Description: Core intelligence layer using LangGraph to orchestrate multiple specialized agents in a structured debate.

4.4.1 Agent Architecture

Agent Roles:

1. **VAA (Vulnerability Assessment Agent)**
   - **Primary Focus:** Security vulnerabilities
   - **Capabilities:**
     - OWASP Top 10 detection
     - CVE pattern matching
     - SQL injection detection
     - XSS vulnerability scanning
     - Authentication/authorization issues
     - Cryptographic weaknesses
     - Input validation problems
2. **DRA (Dependency & Risk Agent)**
   - **Primary Focus:** Code quality and risk assessment
   - **Capabilities:**
     - Breaking change detection
     - Dependency vulnerability scanning
     - Code complexity analysis
     - Blast radius assessment
     - Performance regression detection
     - Architecture violation checking
3. **Summary Agent**
   - **Primary Focus:** Synthesis and fix generation
   - **Capabilities:**
     - Consolidate findings from VAA and DRA
     - Generate code fixes
     - Prioritize vulnerabilities by severity
     - Create actionable reports

4.4.2 LangGraph State Machine

python

```python
from typing import TypedDict, Annotated, Sequence
from langgraph.graph import StateGraph, END
import operator

class DebateState(TypedDict):
    """
    State shared between agents during debate
    """
    # Input
    code_context: str  # Compressed codebase
    pr_diff: str       # Pull request diff
    changed_files: List[str]

    # Conversation
    messages: Annotated[Sequence[dict], operator.add]
    round_count: int
    max_rounds: int

    # Agent findings
    vaa_findings: List[VulnerabilityFinding]
    dra_findings: List[RiskFinding]

    # Consensus tracking
    agreed_issues: List[SecurityIssue]
    disputed_issues: List[SecurityIssue]

    # Final output
    final_report: Optional[SecurityReport]
    generated_fixes: Optional[List[CodeFix]]
```

State Graph Implementation:

python

```python
from langgraph.graph import StateGraph, END
from langchain_anthropic import ChatAnthropic

class MultiAgentAnalyzer:
    def __init__(self):
        self.llm = ChatAnthropic(
            model="claude-3-5-sonnet-20241022",
            temperature=0.1
        )
```

```
        self.graph = self._build_graph()

    def _build_graph(self) -> StateGraph:
        workflow = StateGraph(DebateState)

        # Add nodes
        workflow.add_node("vaa", self._vaa_agent)
        workflow.add_node("dra", self._dra_agent)
        workflow.add_node("summary", self._summary_agent)

        # Set entry point
        workflow.set_entry_point("vaa")

        # Add conditional edges
        workflow.add_conditional_edges(
            "vaa",
            self._should_continue_debate,
            {
                "continue_to_dra": "dra",
                "end": "summary"
            }
        )

        workflow.add_conditional_edges(
            "dra",
            self._should_continue_debate,
            {
                "continue_to_vaa": "vaa",
                "end": "summary"
            }
        )

        workflow.add_edge("summary", END)

        return workflow.compile()

    def _should_continue_debate(self, state: DebateState) -> str:
        """
        Decide whether to continue debate or end
        """
        if state["round_count"] >= state["max_rounds"]:
            return "end"

        # Check if agents have reached consensus
        if self._has_consensus(state):
            return "end"
```

```
    # Continue debate
    return f"continue_to_{self._next_agent(state)}"

def _vaa_agent(self, state: DebateState) -> DebateState:
    """
    Vulnerability Assessment Agent
    """
    prompt = self._build_vaa_prompt(state)
    response = self.llm.invoke(prompt)

    # Parse response into structured findings
    findings = self._parse_vaa_response(response)

    state["vaa_findings"] = findings
    state["messages"].append({
        "role": "vaa",
        "content": response.content,
        "round": state["round_count"]
    })

    return state

def _dra_agent(self, state: DebateState) -> DebateState:
    """
    Dependency & Risk Agent
    """
    prompt = self._build_dra_prompt(state)
    response = self.llm.invoke(prompt)

    findings = self._parse_dra_response(response)

    state["dra_findings"] = findings
    state["messages"].append({
        "role": "dra",
        "content": response.content,
        "round": state["round_count"]
    })

    state["round_count"] += 1

    return state

def _summary_agent(self, state: DebateState) -> DebateState:
    """
    Summary Agent - synthesizes findings and generates fixes
    """
    prompt = self._build_summary_prompt(state)
```

```
        response = self.llm.invoke(prompt)

        # Parse final report and fixes
        report = self._parse_summary_response(response)

        state["final_report"] = report
        state["generated_fixes"] = report.fixes

        return state
```

4.4.3 Agent Prompts

VAA Agent System Prompt:

python

```
VAA_SYSTEM_PROMPT = """You are a Vulnerability Assessment Agent (VAA) specializing in security analysis.

Your role is to identify security vulnerabilities in code changes.

FOCUS AREAS:
1. OWASP Top 10 vulnerabilities
2. SQL injection
3. Cross-Site Scripting (XSS)
4. Authentication/Authorization flaws
5. Cryptographic issues
6. Input validation
7. Sensitive data exposure
8. XML External Entities (XXE)
9. Insecure deserialization
10. Using components with known vulnerabilities

ANALYSIS METHODOLOGY:
- Review the PR diff and full file context
- Identify specific lines with security issues
- Assess severity (CRITICAL, HIGH, MEDIUM, LOW)
- Consider the DRA agent's risk assessment
- Be willing to debate and refine your findings
- If DRA disagrees, explain your reasoning clearly

OUTPUT FORMAT (JSON):
{
  "vulnerabilities": [
    {
      "type": "SQL Injection",
      "severity": "CRITICAL",
```

```
      "file": "path/to/file.py",
      "line_number": 42,
      "code_snippet": "query = f'SELECT * FROM users WHERE id = {user_id}'",
      "description": "Unsanitized user input in SQL query",
      "cwe_id": "CWE-89",
      "recommendation": "Use parameterized queries"
    }
  ],
  "confidence": 0.95,
  "reasoning": "Detailed explanation of analysis process"
}

DEBATE GUIDELINES:
- If DRA raises valid concerns, acknowledge and refine your findings
- Focus on security implications, not code style
- Be specific with evidence (line numbers, code snippets)
- Distinguish between confirmed vulnerabilities and potential risks
"""
```

DRA Agent System Prompt:

python

```
DRA_SYSTEM_PROMPT = """You are a Dependency & Risk Agent (DRA) specializing in code quality and risk assessment.

Your role is to evaluate the broader impact and risks of code changes.

FOCUS AREAS:
1. Breaking changes to public APIs
2. Dependency vulnerabilities
3. Code complexity and maintainability
4. Performance regressions
5. Architecture violations
6. Blast radius of changes
7. Test coverage gaps
8. Configuration risks

ANALYSIS METHODOLOGY:
- Review the PR diff and dependency graph
- Assess impact on dependent code
- Identify potential breaking changes
- Evaluate code complexity metrics
- Consider the VAA agent's vulnerability findings
- Debate findings constructively

OUTPUT FORMAT (JSON):
```

```
{
  "risks": [
    {
      "type": "Breaking Change",
      "severity": "HIGH",
      "file": "api/endpoints.py",
      "line_number": 15,
      "description": "Removed required parameter from public API",
      "impact_scope": "All API consumers",
      "affected_dependencies": ["service-a", "service-b"],
      "recommendation": "Use deprecation pattern"
    }
  ],
  "complexity_score": 7.5,
  "confidence": 0.88,
  "reasoning": "Detailed risk assessment rationale"
}

DEBATE GUIDELINES:
- Challenge VAA findings if severity seems overstated
- Consider operational and business impact
- Provide context about acceptable risks
- Be constructive in disagreements
"""
```

Summary Agent System Prompt:

python

```
SUMMARY_SYSTEM_PROMPT = """You are a Summary Agent responsible for synthesizing security findings and generating fixes.

Your role is to:
1. Review the debate between VAA and DRA agents
2. Consolidate agreed-upon issues
3. Resolve disputed findings
4. Generate code fixes
5. Create comprehensive security report

SYNTHESIS METHODOLOGY:
- Prioritize consensus findings
- For disputes, use evidence strength to decide
- Group related issues
- Assign overall severity scores
- Generate concrete, working code fixes

OUTPUT FORMAT (JSON):
```

```
{
  "summary": {
    "critical_count": 2,
    "high_count": 3,
    "medium_count": 5,
    "low_count": 1,
    "overall_risk_score": 7.8
  },
  "issues": [
    {
      "id": "ISSUE-001",
      "title": "SQL Injection in user authentication",
      "severity": "CRITICAL",
      "description": "...",
      "files_affected": ["auth/login.py"],
      "fix_provided": true
    }
  ],
  "fixes": [
    {
      "issue_id": "ISSUE-001",
      "file": "auth/login.py",
      "original_code": "...",
      "fixed_code": "...",
      "explanation": "Replaced string concatenation with parameterized query"
    }
  ],
  "debate_summary": "VAA and DRA agreed on 8/11 findings...",
  "recommendations": [
    "Enable SQL injection detection in CI/CD",
    "Add input validation middleware"
  ]
}

FIX GENERATION GUIDELINES:
- Provide complete, working code
- Maintain code style consistency
- Add comments explaining changes
- Ensure fixes don't introduce new issues
- Test generated code mentally
"""
```

4.4.4 Debate Termination Logic

python

```
def _has_consensus(self, state: DebateState) -> bool:
```

```
    """
    Check if agents have reached sufficient consensus
    """
    vaa_issues = set(f.issue_id for f in state["vaa_findings"])
    dra_issues = set(f.issue_id for f in state["dra_findings"])

    # Calculate agreement ratio
    agreed = len(state["agreed_issues"])
    total = len(vaa_issues.union(dra_issues))

    agreement_ratio = agreed / total if total > 0 else 0

    # Consensus threshold: 80% agreement
    return agreement_ratio >= 0.8

def _next_agent(self, state: DebateState) -> str:
    """
    Determine next agent in debate
    """
    last_message = state["messages"][-1]

    return "dra" if last_message["role"] == "vaa" else "vaa"
```

4.5 Feature 5: Automated Fix Generation

Description: Generate code fixes based on identified vulnerabilities and create a new pull request.

4.5.1 Fix Generator

python

```
from typing import List, Dict
from pathlib import Path

class FixGenerator:
    """
    Generate code fixes for identified vulnerabilities
    """

    def generate_fixes(
        self,
        security_report: SecurityReport,
        repo_path: Path
    ) -> List[GeneratedFix]:
        """
```

```
    Generate fixes for all issues in report

    Args:
        security_report: Analysis results with issues
        repo_path: Path to repository

    Returns:
        List of GeneratedFix objects with file patches
    """
    fixes = []

    for issue in security_report.issues:
        if issue.fix_available:
            fix = self._generate_single_fix(issue, repo_path)
            fixes.append(fix)

    return fixes

def _generate_single_fix(
    self,
    issue: SecurityIssue,
    repo_path: Path
) -> GeneratedFix:
    """
    Generate fix for a single issue
    """
    file_path = repo_path / issue.file

    # Read original file
    with open(file_path, 'r') as f:
        original_content = f.read()

    # Apply fix
    fixed_content = self._apply_fix_to_content(
        original_content,
        issue.line_number,
        issue.fix_code
    )

    # Generate unified diff
    diff = self._create_diff(
        original_content,
        fixed_content,
        file_path
    )

    return GeneratedFix(
```

```
            issue_id=issue.id,
            file=str(file_path),
            diff=diff,
            fixed_content=fixed_content,
            description=issue.fix_explanation
        )

    def _apply_fix_to_content(
        self,
        content: str,
        line_number: int,
        fix_code: str
    ) -> str:
        """
        Apply code fix to file content
        """
        lines = content.split('\n')

        # Replace vulnerable code with fix
        # (Implementation depends on fix type)

        return '\n'.join(lines)

    def _create_diff(
        self,
        original: str,
        fixed: str,
        file_path: Path
    ) -> str:
        """
        Create unified diff
        """
        import difflib

        diff = difflib.unified_diff(
            original.splitlines(keepends=True),
            fixed.splitlines(keepends=True),
            fromfile=f"a/{file_path}",
            tofile=f"b/{file_path}"
        )

        return ''.join(diff)
```

4.5.2 Branch & PR Manager

python

```python
import git
from github import Github, PullRequest

class PRManager:
    """
    Manage Git branches and GitHub pull requests
    """

    def __init__(self, github_token: str):
        self.gh = Github(github_token)

    def create_fix_pr(
        self,
        repo_full_name: str,
        base_branch: str,
        fixes: List[GeneratedFix],
        security_report: SecurityReport,
        original_pr_number: int
    ) -> PullRequest:
        """
        Create pull request with security fixes

        Args:
            repo_full_name: e.g., "user/repo"
            base_branch: Branch to merge into
            fixes: List of generated fixes
            security_report: Full analysis report
            original_pr_number: PR that triggered analysis

        Returns:
            Created PullRequest object
        """
        repo = self.gh.get_repo(repo_full_name)

        # Create new branch
        branch_name = f"protectsus/fix-pr-{original_pr_number}"
        base_sha = repo.get_branch(base_branch).commit.sha

        repo.create_git_ref(
            ref=f"refs/heads/{branch_name}",
            sha=base_sha
        )

        # Apply fixes to branch
        for fix in fixes:
            self._commit_fix(repo, branch_name, fix)
```

```
    # Create pull request
    pr_title = f"🛡️ [ProtectSUS] Security fixes for PR #{original_pr_number}"
    pr_body = self._generate_pr_body(security_report, original_pr_number)

    pr = repo.create_pull(
        title=pr_title,
        body=pr_body,
        head=branch_name,
        base=base_branch
    )

    # Add comment with detailed report
    comment_body = self._generate_detailed_comment(security_report)
    pr.create_issue_comment(comment_body)

    return pr

def _commit_fix(
    self,
    repo,
    branch_name: str,
    fix: GeneratedFix
):
    """
    Commit a single fix to branch
    """
    file_path = fix.file

    # Get current file content
    try:
        file = repo.get_contents(file_path, ref=branch_name)

        repo.update_file(
            path=file_path,
            message=f"Fix: {fix.description}",
            content=fix.fixed_content,
            sha=file.sha,
            branch=branch_name
        )
    except Exception as e:
        # Handle file not found or other errors
        print(f"Error committing fix: {e}")

def _generate_pr_body(
    self,
    security_report: SecurityReport,
```

```
        original_pr_number: int
    ) -> str:
        """
        Generate pull request description
        """
        return f"""
## 🛡️ ProtectSUS Security Analysis

This PR contains automated security fixes for issues detected in PR #{original_pr_number}.

### 📊 Summary
- **Critical Issues:** {security_report.summary.critical_count}
- **High Severity:** {security_report.summary.high_count}
- **Medium Severity:** {security_report.summary.medium_count}
- **Low Severity:** {security_report.summary.low_count}
- **Overall Risk Score:** {security_report.summary.overall_risk_score}/10

### 🔍 Issues Fixed
{self._format_issues_list(security_report.issues)}

### ✅ Actions Required
1. **Review the fixes** - Ensure they align with your code standards
2. **Run tests** - Verify fixes don't break functionality
3. **Approve or provide feedback** - React with 👍 to approve or 👎 to deny
4. **Merge** - Once approved, ProtectSUS will auto-merge if there are no conflicts

---
*Generated by ProtectSUS AI Security Copilot*
"""

    def _generate_detailed_comment(
        self,
        security_report: SecurityReport
    ) -> str:
        """
        Generate detailed analysis comment
        """
        return f"""
## 🔐 Detailed Security Analysis

{self._format_vulnerabilities_detailed(security_report)}

### 📈 Agent Debate Summary
{security_report.debate_summary}
```

```
### 💡 Recommendations
{self._format_recommendations(security_report.recommendations)}

---

### 🤖 How to Respond
- **Approve:** Add a 👍 reaction to this comment
- **Deny:** Add a 👎 reaction and comment with reasoning
- **Questions:** Ask in the comments, ProtectSUS will explain!

*This analysis was performed using multi-agent AI with VAA (Vulnerability Assessment) and DRA (Risk Assessment) agents.*
"""
```

4.6 Feature 6: User Feedback & Reinforcement Learning

Description: Learn from user approvals/denials to improve future predictions and reduce false positives.

4.6.1 Feedback Collector

python

```
from enum import Enum
from datetime import datetime

class FeedbackType(Enum):
    APPROVED = "approved"
    DENIED = "denied"
    PARTIAL = "partial"

class FeedbackCollector:
    """
    Collect and process user feedback on security fixes
    """

    def __init__(self, mongo_client: MongoClient):
        self.db = mongo_client.protectsus
        self.feedback_collection = self.db.user_feedback

    async def record_feedback(
        self,
        pr_number: int,
```

```
    repo_full_name: str,
    security_report: SecurityReport,
    feedback_type: FeedbackType,
    user_comment: Optional[str] = None
):
    """
    Record user feedback on security analysis

    Args:
        pr_number: PR number
        repo_full_name: Repository name
        security_report: Original analysis
        feedback_type: User's decision
        user_comment: Optional explanation
    """
    feedback_doc = {
        "timestamp": datetime.utcnow(),
        "pr_number": pr_number,
        "repo": repo_full_name,
        "feedback_type": feedback_type.value,
        "user_comment": user_comment,

        # Analysis features
        "features": self._extract_features(security_report),

        # Issues breakdown
        "issues": [
            {
                "id": issue.id,
                "type": issue.type,
                "severity": issue.severity,
                "file": issue.file,
                "cwe_id": issue.cwe_id
            }
            for issue in security_report.issues
        ],

        # Metadata
        "critical_count": security_report.summary.critical_count,
        "high_count": security_report.summary.high_count,
        "medium_count": security_report.summary.medium_count,
        "low_count": security_report.summary.low_count,
        "risk_score": security_report.summary.overall_risk_score
    }

    await self.feedback_collection.insert_one(feedback_doc)
```

```
        # Trigger RL model retraining
        await self._trigger_model_update(repo_full_name)

    def _extract_features(
        self,
        security_report: SecurityReport
    ) -> Dict[str, float]:
        """
        Extract features for ML model
        """
        return {
            # Issue counts by severity
            "critical_count": float(security_report.summary.critical_count),
            "high_count": float(security_report.summary.high_count),
            "medium_count": float(security_report.summary.medium_count),
            "low_count": float(security_report.summary.low_count),

            # Issue type distribution
            "sql_injection_count": self._count_issue_type("SQL Injection", security_report),
            "xss_count": self._count_issue_type("XSS", security_report),
            "auth_issue_count": self._count_issue_type("Authentication", security_report),

            # Code metrics
            "files_affected": float(len(set(i.file for i in security_report.issues))),
            "avg_confidence": sum(i.confidence for i in security_report.issues) /
len(security_report.issues),

            # Complexity
            "complexity_score": security_report.complexity_score,
            "risk_score": security_report.summary.overall_risk_score,

            # Debate metrics
            "debate_rounds": float(security_report.debate_rounds),
            "consensus_ratio": security_report.consensus_ratio
        }
```

4.6.2 Reinforcement Learning Model

python

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
import joblib
import numpy as np

class UserPreferenceModel:
    """
```

```
ML model to predict user approval likelihood
"""

def __init__(self, model_dir: Path):
    self.model_dir = model_dir
    self.model = None
    self.scaler = StandardScaler()

    # Try to load existing model
    self._load_model()

def _load_model(self):
    """
    Load trained model from disk
    """
    model_path = self.model_dir / "preference_model.pkl"
    scaler_path = self.model_dir / "scaler.pkl"

    if model_path.exists():
        self.model = joblib.load(model_path)
        self.scaler = joblib.load(scaler_path)

async def train(self, repo_full_name: str):
    """
    Train model on historical feedback data

    Args:
        repo_full_name: Repository to train model for
    """
    # Fetch training data from MongoDB
    feedback_data = await self._fetch_training_data(repo_full_name)

    if len(feedback_data) < 5:
        # Not enough data to train
        return

    # Prepare features and labels
    X = np.array([d["features"] for d in feedback_data])
    y = np.array([
        1 if d["feedback_type"] == "approved" else 0
        for d in feedback_data
    ])

    # Normalize features
    X_scaled = self.scaler.fit_transform(X)

    # Train model
```

```
        self.model = RandomForestClassifier(
            n_estimators=100,
            max_depth=10,
            random_state=42,
            class_weight='balanced'  # Handle class imbalance
        )

        self.model.fit(X_scaled, y)

        # Save model
        self._save_model()

    def predict_approval_probability(
        self,
        security_report: SecurityReport
    ) -> float:
        """
        Predict probability that user will approve these fixes

        Returns:
            Probability between 0 and 1
        """
        if self.model is None:
            # No model trained yet, return neutral probability
            return 0.5

        # Extract features
        features = self._extract_features(security_report)
        X = np.array([list(features.values())])

        # Scale and predict
        X_scaled = self.scaler.transform(X)
        prob = self.model.predict_proba(X_scaled)[0][1]

        return float(prob)

    def get_feature_importance(self) -> Dict[str, float]:
        """
        Get feature importance scores
        """
        if self.model is None:
            return {}

        feature_names = [
            "critical_count", "high_count", "medium_count", "low_count",
            "sql_injection_count", "xss_count", "auth_issue_count",
            "files_affected", "avg_confidence", "complexity_score",
```

```
            "risk_score", "debate_rounds", "consensus_ratio"
        ]

        importances = self.model.feature_importances_

        return dict(zip(feature_names, importances))

    def _save_model(self):
        """
        Save trained model to disk
        """
        model_path = self.model_dir / "preference_model.pkl"
        scaler_path = self.model_dir / "scaler.pkl"

        joblib.dump(self.model, model_path)
        joblib.dump(self.scaler, scaler_path)
```

4.6.3 Feedback-Driven Filtering

python

```
class AdaptiveAnalyzer:
    """
    Use RL model to filter and prioritize findings
    """

    def __init__(self, preference_model: UserPreferenceModel):
        self.model = preference_model

    def filter_by_user_preference(
        self,
        security_report: SecurityReport,
        threshold: float = 0.6
    ) -> SecurityReport:
        """
        Filter findings based on predicted user approval

        Args:
            security_report: Original analysis
            threshold: Minimum approval probability

        Returns:
            Filtered security report
        """
        # Predict approval probability
        approval_prob = self.model.predict_approval_probability(security_report)
```

```
        if approval_prob < threshold:
            # Low predicted approval - be more conservative
            filtered_issues = [
                issue for issue in security_report.issues
                if issue.severity in ["CRITICAL", "HIGH"]
            ]
        else:
            # High predicted approval - include all findings
            filtered_issues = security_report.issues

        # Create filtered report
        filtered_report = SecurityReport(
            issues=filtered_issues,
            summary=self._recalculate_summary(filtered_issues),
            debate_summary=security_report.debate_summary,
            recommendations=security_report.recommendations,
            metadata={
                "filtered": approval_prob < threshold,
                "approval_probability": approval_prob,
                "original_issue_count": len(security_report.issues)
            }
        )

        return filtered_report
```

4.7 Feature 7: Knowledge Graph & Interactive Explanation

Description: Build code relationship graph in Neo4j and enable interactive querying about vulnerabilities.

4.7.1 Knowledge Graph Builder

python

```
from neo4j import GraphDatabase
from typing import List, Dict

class CodeKnowledgeGraph:
    """
    Build and query code knowledge graph in Neo4j
    """

    def __init__(self, neo4j_uri: str, user: str, password: str):
        self.driver = GraphDatabase.driver(neo4j_uri, auth=(user, password))

    def build_graph(self, repo_path: Path, repo_name: str):
```

```
    """
    Build knowledge graph from repository code

    Creates nodes for:
    - Files
    - Functions/Methods
    - Classes
    - Variables
    - Dependencies

    Creates relationships:
    - IMPORTS
    - CALLS
    - DEFINES
    - INHERITS
    - DEPENDS_ON
    """
    with self.driver.session() as session:
        # Clear existing graph for this repo
        session.run(
            "MATCH (n:CodeEntity {repo: $repo}) DETACH DELETE n",
            repo=repo_name
        )

        # Parse repository and build graph
        parsed_repo = self._parse_repository(repo_path)

        # Create nodes
        for file in parsed_repo.files:
            self._create_file_node(session, file, repo_name)

            for func in file.functions:
                self._create_function_node(session, func, file, repo_name)

            for cls in file.classes:
                self._create_class_node(session, cls, file, repo_name)

        # Create relationships
        for file in parsed_repo.files:
            self._create_import_relationships(session, file, repo_name)
            self._create_call_relationships(session, file, repo_name)

def add_vulnerability_nodes(
    self,
    session,
    security_report: SecurityReport,
    repo_name: str
```

```
):
    """
    Add vulnerability information to graph
    """
    for issue in security_report.issues:
        # Create vulnerability node
        session.run("""
            CREATE (v:Vulnerability {
                id: $id,
                type: $type,
                severity: $severity,
                description: $description,
                cwe_id: $cwe_id,
                repo: $repo
            })
            """,
            id=issue.id,
            type=issue.type,
            severity=issue.severity,
            description=issue.description,
            cwe_id=issue.cwe_id,
            repo=repo_name
        )

        # Link to affected file/function
        session.run("""
            MATCH (v:Vulnerability {id: $vuln_id})
            MATCH (f:File {path: $file_path, repo: $repo})
            CREATE (v)-[:AFFECTS]->(f)
            """,
            vuln_id=issue.id,
            file_path=issue.file,
            repo=repo_name
        )

def query_impact_chain(
    self,
    vulnerability_id: str,
    max_depth: int = 5
) -> List[Dict]:
    """
    Find all code affected by a vulnerability

    Returns chain of dependencies that could be impacted
    """
    with self.driver.session() as session:
        result = session.run("""
```

```
            MATCH path = (v:Vulnerability {id: $vuln_id})-[:AFFECTS]->(start)
                  -[r:CALLS|IMPORTS|DEPENDS_ON*1..{max_depth}]->(affected)
            RETURN path, affected
            ORDER BY length(path) DESC
            """,
            vuln_id=vulnerability_id,
            max_depth=max_depth
        )

        impact_chain = []
        for record in result:
            impact_chain.append({
                "path": record["path"],
                "affected_entity": record["affected"]
            })

        return impact_chain

def explain_vulnerability(
    self,
    vulnerability_id: str
) -> str:
    """
    Generate natural language explanation of vulnerability
    """
    with self.driver.session() as session:
        # Get vulnerability and its context
        result = session.run("""
            MATCH (v:Vulnerability {id: $vuln_id})-[:AFFECTS]->(f:File)
            OPTIONAL MATCH (f)-[:DEFINES]->(func:Function)
            OPTIONAL MATCH (f)-[:IMPORTS]->(dep:File)
            RETURN v, f, collect(DISTINCT func) as functions,
                   collect(DISTINCT dep) as dependencies
            """,
            vuln_id=vulnerability_id
        )

        record = result.single()

        # Build explanation using LLM
        explanation_prompt = f"""
        Explain this vulnerability in simple terms:

        Type: {record['v']['type']}
        Severity: {record['v']['severity']}
        File: {record['f']['path']}
        Description: {record['v']['description']}
```

```
        Context:
        - Functions in file: {[f['name'] for f in record['functions']]}
        - Dependencies: {[d['path'] for d in record['dependencies']]}

        Provide:
        1. What the vulnerability is
        2. Why it's dangerous
        3. How an attacker could exploit it
        4. What could be impacted
        """

        # Call LLM for explanation
        explanation = self._generate_explanation(explanation_prompt)

        return explanation
```

4.7.2 Interactive Chat Interface

python

```
from langchain.chains import GraphCypherQAChain
from langchain_anthropic import ChatAnthropic

class VulnerabilityExplainer:
    """
    Interactive chat interface for vulnerability questions
    """

    def __init__(self, knowledge_graph: CodeKnowledgeGraph):
        self.graph = knowledge_graph
        self.llm = ChatAnthropic(model="claude-3-5-sonnet-20241022")

        # Create Cypher QA chain
        self.qa_chain = GraphCypherQAChain.from_llm(
            llm=self.llm,
            graph=self.graph,
            verbose=True
        )

    async def answer_question(
        self,
        question: str,
        repo_name: str
    ) -> str:
        """
        Answer natural language questions about vulnerabilities
```

```
    Examples:
    - "Why is the SQL injection in login.py critical?"
    - "Show me all functions affected by ISSUE-001"
    - "What's the blast radius if we change auth.py?"
    - "Which dependencies are vulnerable?"
    """
    # Add repository context to question
    contextualized_question = f"""
    For repository {repo_name}:
    {question}
    """

    # Query knowledge graph
    response = await self.qa_chain.ainvoke({
        "query": contextualized_question
    })

    return response["result"]

def get_suggested_questions(
    self,
    security_report: SecurityReport
) -> List[str]:
    """
    Generate suggested questions based on analysis
    """
    suggestions = []

    if security_report.summary.critical_count > 0:
        suggestions.append(
            "Why are the critical vulnerabilities so severe?"
        )

    if len(security_report.issues) > 5:
        suggestions.append(
            "Which issues should I fix first?"
        )

    suggestions.extend([
        "What could an attacker do with these vulnerabilities?",
        "Show me the full impact chain of the highest severity issue",
        "Which parts of my codebase are most at risk?"
    ])

    return suggestions
```

4.8 Feature 8: Arize Phoenix Integration

Description: Instrument agents with Arize Phoenix for observability and debugging.

python

```python
import phoenix as px
from phoenix.trace.langchain import LangChainInstrumentor
from openinference.instrumentation import using_attributes

class ObservabilityManager:
    """
    Manage Arize Phoenix observability
    """

    def __init__(self, phoenix_api_key: Optional[str] = None):
        # Launch Phoenix
        if phoenix_api_key:
            # Use Phoenix Cloud
            px.set_credentials(api_key=phoenix_api_key)
            self.session = px.launch_app(cloud=True)
        else:
            # Use local Phoenix
            self.session = px.launch_app()

        # Instrument LangChain
        LangChainInstrumentor().instrument()

    def log_analysis(
        self,
        pr_number: int,
        repo_name: str,
        security_report: SecurityReport,
        processing_time: float
    ):
        """
        Log analysis metrics to Phoenix
        """
        with using_attributes(
            session_id=f"{repo_name}-pr-{pr_number}",
            user_id=repo_name,
            metadata={
                "pr_number": pr_number,
                "critical_count": security_report.summary.critical_count,
                "total_issues": len(security_report.issues),
                "processing_time_seconds": processing_time,
```

```
                "risk_score": security_report.summary.overall_risk_score
            }
        ):
            # Traces are automatically captured by LangChain instrumentation
            pass

    def get_dashboard_url(self) -> str:
        """
        Get Phoenix dashboard URL
        """
        return self.session.url
```

---

5. API Specifications

5.1 REST API Endpoints

5.1.1 Webhook Endpoint

```
POST /api/v1/webhook/github
```

Description: Receive GitHub webhook events

Headers:

- `X-GitHub-Event`: Event type (pull_request, push)
- `X-GitHub-Delivery`: Unique event ID
- `X-Hub-Signature-256`: HMAC signature
- `Content-Type`: application/json

Request Body:

```
json
{
  "action": "opened",
  "pull_request": { ... },
  "repository": { ... }
}
```

Response:

```
json
{
  "status": "accepted",
```

```
  "analysis_id": "abc123",
  "message": "Analysis queued for processing"
}
```

Status Codes:
- 202: Accepted - Analysis queued
- 401: Unauthorized - Invalid signature
- 400: Bad Request - Invalid payload

5.1.2 Analysis Status Endpoint
```
GET /api/v1/analysis/{analysis_id}
```
Description: Check status of analysis

Response:

json
```
{
  "analysis_id": "abc123",
  "status": "completed",
  "pr_number": 42,
  "repository": "user/repo",
  "created_at": "2026-01-17T12:00:00Z",
  "completed_at": "2026-01-17T12:05:23Z",
  "processing_time_seconds": 323,
  "results": {
    "critical_count": 2,
    "high_count": 3,
    "medium_count": 5,
    "low_count": 1,
    "fix_pr_number": 43,
    "fix_pr_url": "https://github.com/user/repo/pull/43"
  }
}
```

Status Values:
- `queued`: Waiting to process
- `processing`: Currently analyzing
- `completed`: Analysis finished
- `failed`: Error occurred

5.1.3 Feedback Endpoint
```
POST /api/v1/feedback
```
Description: Record user feedback on analysis

Request Body:

```json
{
  "pr_number": 43,
  "repository": "user/repo",
  "feedback_type": "approved",
  "comment": "Good catch on the SQL injection!"
}
```
Response:

```json
{
  "status": "recorded",
  "feedback_id": "xyz789",
  "model_retrain_scheduled": true
}
```

5.1.4 Interactive Chat Endpoint
```
POST /api/v1/chat
```
Description: Ask questions about vulnerabilities

Request Body:

```json
{
  "repository": "user/repo",
  "pr_number": 42,
  "question": "Why is the SQL injection critical?"
}
```

Response:

```
json
{
  "answer": "The SQL injection in login.py is critical because...",
  "sources": [
    {
      "type": "vulnerability",
      "id": "ISSUE-001",
      "file": "auth/login.py"
    }
  ],
  "suggested_questions": [
    "What's the impact if this is exploited?",
    "How should I fix this?"
  ]
}
```

5.2 Internal API Design

5.2.1 Analysis Pipeline Interface

```
python
from abc import ABC, abstractmethod

class AnalysisPipeline(ABC):
    """
    Abstract interface for analysis pipeline
    """

    @abstractmethod
    async def analyze_pr(
        self,
        repo_full_name: str,
        pr_number: int,
        pr_data: Dict
    ) -> AnalysisResult:
        """
        Run full analysis pipeline on PR
        """
        pass

    @abstractmethod
```

```python
    async def get_status(self, analysis_id: str) -> AnalysisStatus:
        """
        Get current status of analysis
        """
        pass
```

6. Database Schemas

6.1 MongoDB Collections

6.1.1 `analyses` Collection

javascript

```javascript
{
  _id: ObjectId("..."),
  analysis_id: "abc123",
  repository: "user/repo",
  pr_number: 42,

  // Status tracking
  status: "completed",  // queued, processing, completed, failed
  created_at: ISODate("2026-01-17T12:00:00Z"),
  started_at: ISODate("2026-01-17T12:01:00Z"),
  completed_at: ISODate("2026-01-17T12:05:23Z"),

  // Input
  pr_data: {
    title: "Add feature X",
    base_branch: "main",
    head_branch: "feature-x",
    base_sha: "abc123...",
    head_sha: "def456...",
    diff_url: "https://..."
  },

  // Compression metrics
  compression: {
    original_tokens: 15000,
    compressed_tokens: 6000,
    compression_ratio: 0.6,
    processing_time_ms: 230
  },

  // Agent conversation
```

```
agent_debate: [
  {
    round: 1,
    agent: "vaa",
    message: "...",
    findings: [...],
    timestamp: ISODate("...")
  },
  {
    round: 1,
    agent: "dra",
    message: "...",
    findings: [...],
    timestamp: ISODate("...")
  }
],

// Results
security_report: {
  summary: {
    critical_count: 2,
    high_count: 3,
    medium_count: 5,
    low_count: 1,
    overall_risk_score: 7.8
  },
  issues: [
    {
      id: "ISSUE-001",
      type: "SQL Injection",
      severity: "CRITICAL",
      file: "auth/login.py",
      line_number: 42,
      description: "...",
      cwe_id: "CWE-89",
      confidence: 0.95,
      fix_available: true
    }
  ],
  fixes: [...]
},

// Generated PR
fix_pr: {
  number: 43,
  url: "https://github.com/user/repo/pull/43",
  created_at: ISODate("...")
```

```
  },

  // Processing metrics
  metrics: {
    processing_time_seconds: 323,
    token_usage: {
      input_tokens: 6000,
      output_tokens: 2500
    },
    debate_rounds: 3
  }
}
```

6.1.2 `user_feedback` Collection

javascript

```
{
  _id: ObjectId("..."),
  timestamp: ISODate("2026-01-17T14:30:00Z"),

  // Reference
  analysis_id: "abc123",
  pr_number: 43,
  repository: "user/repo",

  // Feedback
  feedback_type: "approved",  // approved, denied, partial
  user_comment: "Good catch!",
  user_login: "username",

  // Features for ML
  features: {
    critical_count: 2,
    high_count: 3,
    medium_count: 5,
    low_count: 1,
    sql_injection_count: 1,
    xss_count: 0,
    auth_issue_count: 1,
    files_affected: 3,
    avg_confidence: 0.87,
    complexity_score: 6.5,
    risk_score: 7.8,
    debate_rounds: 3,
    consensus_ratio: 0.85
  },
```

```
  // Issues that were approved/denied
  issues: [
    {
      id: "ISSUE-001",
      type: "SQL Injection",
      severity: "CRITICAL",
      user_approved: true
    }
  ]
}
```

6.1.3 `code_embeddings` Collection

javascript

```
{
  _id: ObjectId("..."),
  repository: "user/repo",
  file_path: "auth/login.py",

  // Code content
  content: "...",
  language: "python",

  // Embedding
  embedding: [0.123, -0.456, ...],  // 1536-dimensional vector

  // Metadata
  functions: ["login", "verify_credentials"],
  imports: ["hashlib", "jwt"],

  // Versioning
  commit_sha: "abc123...",
  indexed_at: ISODate("2026-01-17T12:00:00Z")
}
```

6.2 Neo4j Graph Schema

6.2.1 Node Types

File Node:

cypher

```
CREATE (f:File {
  path: "auth/login.py",
  repo: "user/repo",
  language: "python",
  line_count: 150
})
```

Function Node:

cypher

```
CREATE (fn:Function {
  name: "login",
  file: "auth/login.py",
  repo: "user/repo",
  line_start: 10,
  line_end: 25,
  complexity: 5
})
```

Class Node:

cypher

```
CREATE (c:Class {
  name: "UserAuth",
  file: "auth/login.py",
  repo: "user/repo"
})
```

Vulnerability Node:

cypher

```
CREATE (v:Vulnerability {
  id: "ISSUE-001",
  type: "SQL Injection",
  severity: "CRITICAL",
  cwe_id: "CWE-89",
  repo: "user/repo",
  detected_at: datetime()
})
```

6.2.2 Relationship Types

cypher

```
// File imports another file
```

```
(f1:File)-[:IMPORTS]->(f2:File)

// Function calls another function
(fn1:Function)-[:CALLS]->(fn2:Function)

// File defines function
(f:File)-[:DEFINES]->(fn:Function)

// Class inherits from another
(c1:Class)-[:INHERITS]->(c2:Class)

// Function depends on external library
(fn:Function)-[:DEPENDS_ON]->(lib:Library)

// Vulnerability affects file/function
(v:Vulnerability)-[:AFFECTS]->(f:File)
(v:Vulnerability)-[:AFFECTS]->(fn:Function)
```

6.2.3 Example Queries

Find impact chain of vulnerability:

```
cypher
MATCH path = (v:Vulnerability {id: "ISSUE-001"})-[:AFFECTS]->(start)
      -[r:CALLS|IMPORTS|DEPENDS_ON*1..5]->(affected)
RETURN path, affected
ORDER BY length(path) DESC
```

Find all vulnerabilities in a file:

```
cypher
MATCH (v:Vulnerability)-[:AFFECTS]->(f:File {path: "auth/login.py"})
RETURN v
ORDER BY v.severity DESC
```

Find most vulnerable files:

```
cypher
MATCH (v:Vulnerability)-[:AFFECTS]->(f:File)
WITH f, count(v) as vuln_count
RETURN f.path, vuln_count
ORDER BY vuln_count DESC
LIMIT 10
```

---

7. ML Model Specifications

7.1 User Preference Model

Type: RandomForestClassifier

Input Features (13 dimensions):
1. `critical_count`: Number of critical issues
2. `high_count`: Number of high severity issues
3. `medium_count`: Number of medium severity issues
4. `low_count`: Number of low severity issues
5. `sql_injection_count`: SQL injection vulnerabilities
6. `xss_count`: XSS vulnerabilities
7. `auth_issue_count`: Authentication issues
8. `files_affected`: Number of files with issues
9. `avg_confidence`: Average confidence score
10. `complexity_score`: Code complexity metric
11. `risk_score`: Overall risk score
12. `debate_rounds`: Number of agent debate rounds
13. `consensus_ratio`: Agent agreement ratio

Output:
- Binary classification: `approved` (1) or `denied` (0)
- Probability score: 0.0 to 1.0

Training:
- Algorithm: Random Forest (sklearn)
- Trees: 100
- Max depth: 10
- Class weighting: Balanced
- Min samples for training: 5
- Retraining: After every feedback event

Model Persistence:
```
/models/
├── user/repo/
│   ├── preference_model.pkl
│   ├── scaler.pkl
│   └── metadata.json
```

---

8. GitHub Integration

8.1 GitHub App Configuration

Required Permissions:
- **Repository contents:** Read & Write (for creating fix branches)
- **Pull requests:** Read & Write (for creating PRs and comments)
- **Issues:** Read & Write (for issue comments)
- **Webhooks:** Read-only (for webhook management)

Webhook Events:
- `pull_request` - opened, synchronize, reopened
- `push` - on specific branches

Webhook URL:
```
https://api.protectsus.ai/api/v1/webhook/github
```
Webhook Secret:

- Stored in environment variable: `GITHUB_WEBHOOK_SECRET`
- Used for HMAC-SHA256 signature verification

8.2 Installation Flow

1. User installs GitHub App on repository
2. User selects repositories to enable ProtectSUS
3. GitHub sends installation webhook
4. ProtectSUS stores installation token
5. User configures branch protection rules (optional)

8.3 PR Comment Format

markdown
🛡️ ProtectSUS Security Analysis

Status: ✅ Complete
Analyzed: PR #42
Processing Time: 5 minutes 23 seconds

📊 Summary
Severity	Count
🔴 Critical	2
🟠 High	3

| 🟡 Medium | 5 |
| 🟢 Low | 1 |

Overall Risk Score: 7.8/10

🔍 Critical Issues

ISSUE-001: SQL Injection in `auth/login.py`
Line: 42
CWE: CWE-89
Confidence: 95%

Description:
Unsanitized user input directly concatenated into SQL query.

Vulnerable Code:
```python
query = f"SELECT * FROM users WHERE username = '{username}'"
```

Recommended Fix:
```python
query = "SELECT * FROM users WHERE username = %s"
cursor.execute(query, (username,))
```

[View full analysis](#) | [Ask question](#)

✅ Automated Fixes

I've created PR #43 with automated fixes for all critical and high severity issues.

Actions:
- 👍 Approve to auto-merge (if tests pass)
- 👎 Deny and provide feedback
- 💬 Ask questions in comments

🤖 Agent Analysis

VAA Agent Findings: 8 security issues
DRA Agent Findings: 6 risk issues
Consensus: 85% (3 debate rounds)

Top Recommendations:
1. Enable parameterized queries globally
2. Add input validation middleware
3. Update authentication library to v2.0

Powered by ProtectSUS Multi-Agent Security Analysis
[Dashboard](#) | [Settings](#) | [Documentation](#)
```

---

## 9. Implementation Guide

### 9.1 Project Structure
```
protectsus/
├── app/
│   ├── __init__.py
│   ├── main.py   # FastAPI application
│   ├── api/
│   │   ├── __init__.py
│   │   ├── v1/
│   │   │   ├── __init__.py
│   │   │   ├── webhooks.py   # Webhook endpoints
│   │   │   ├── analysis.py   # Analysis endpoints
│   │   │   ├── feedback.py   # Feedback endpoints
│   │   │   └── chat.py   # Interactive chat
│   ├── core/
│   │   ├── __init__.py
│   │   ├── config.py   # Configuration
│   │   ├── security.py   # Auth & validation
│   │   └── dependencies.py   # Dependency injection
│   ├── services/
│   │   ├── __init__.py
│   │   ├── github_service.py   # GitHub API client
│   │   ├── compression_service.py # Token Company client
│   │   ├── analysis_service.py # Analysis orchestration
│   │   ├── agent_service.py   # Multi-agent system
```
```

```
│   │   ├── knowledge_graph.py     # Neo4j integration
│   │   └── ml_service.py          # RL model
│   ├── models/
│   │   ├── __init__.py
│   │   ├── schemas.py             # Pydantic models
│   │   ├── database.py            # MongoDB models
│   │   └── graph.py               # Neo4j models
│   ├── workers/
│   │   ├── __init__.py
│   │   ├── celery_app.py          # Celery configuration
│   │   └── tasks.py               # Background tasks
│   └── utils/
│       ├── __init__.py
│       ├── code_parser.py         # AST parsing
│       ├── diff_parser.py         # Git diff parsing
│       └── logging.py             # Logging setup
├── tests/
│   ├── __init__.py
│   ├── conftest.py                # Pytest fixtures
│   ├── unit/
│   ├── integration/
│   └── e2e/
├── models/                        # ML model storage
│   └── .gitkeep
├── scripts/
│   ├── setup_databases.py         # DB initialization
│   ├── train_model.py             # Manual model training
│   └── test_webhook.py            # Webhook testing
├── docker/
│   ├── Dockerfile
│   ├── docker-compose.yml
│   └── nginx.conf
├── .env.example
├── requirements.txt
├── pyproject.toml
└── README.md
```

9.2 Development Setup

Prerequisites:

```bash
# Python 3.11+
python --version
```

```
# Docker & Docker Compose
docker --version
docker-compose --version

# Git
git --version
```

Step 1: Clone Repository

bash

```
git clone https://github.com/your-org/protectsus.git
cd protectsus
```

Step 2: Environment Configuration

bash

```
cp .env.example .env
# Edit .env with your credentials
```

Step 3: Install Dependencies

bash

```
# Create virtual environment
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt
```

Step 4: Start Dependencies

bash

```
# Start MongoDB, Neo4j, Redis via Docker
docker-compose up -d mongodb neo4j redis
```

Step 5: Initialize Databases

bash

```
python scripts/setup_databases.py
```

Step 6: Start Application

bash
```
# Terminal 1: FastAPI
uvicorn app.main:app --reload --port 8000

# Terminal 2: Celery worker
celery -A app.workers.celery_app worker --loglevel=info

# Terminal 3: Phoenix (optional, for observability)
python -m phoenix.server.main serve
```

9.3 Core Implementation Files

9.3.1 `app/main.py` - FastAPI Application

python
```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
import phoenix as px
from phoenix.trace.langchain import LangChainInstrumentor

from app.api.v1 import webhooks, analysis, feedback, chat
from app.core.config import settings

# Initialize Phoenix
px.launch_app()
LangChainInstrumentor().instrument()

# Create FastAPI app
app = FastAPI(
    title="ProtectSUS API",
    description="AI-Powered Code Security Analysis",
    version="1.0.0"
)

# CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=settings.ALLOWED_ORIGINS,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Include routers
```

```python
app.include_router(webhooks.router, prefix="/api/v1", tags=["webhooks"])
app.include_router(analysis.router, prefix="/api/v1", tags=["analysis"])
app.include_router(feedback.router, prefix="/api/v1", tags=["feedback"])
app.include_router(chat.router, prefix="/api/v1", tags=["chat"])

@app.get("/health")
async def health_check():
    return {"status": "healthy"}

@app.on_event("startup")
async def startup_event():
    # Initialize services
    pass

@app.on_event("shutdown")
async def shutdown_event():
    # Cleanup
    pass
```

9.3.2 `app/core/config.py` - Configuration

python

```python
from pydantic_settings import BaseSettings
from typing import List

class Settings(BaseSettings):
    # Application
    APP_NAME: str = "ProtectSUS"
    DEBUG: bool = False
    ALLOWED_ORIGINS: List[str] = ["*"]

    # GitHub
    GITHUB_APP_ID: str
    GITHUB_PRIVATE_KEY: str
    GITHUB_WEBHOOK_SECRET: str

    # MongoDB
    MONGODB_URL: str
    MONGODB_DB_NAME: str = "protectsus"

    # Neo4j
    NEO4J_URI: str
    NEO4J_USER: str
    NEO4J_PASSWORD: str

    # Redis
```

```
    REDIS_URL: str = "redis://localhost:6379/0"

    # Anthropic
    ANTHROPIC_API_KEY: str

    # Token Company
    TOKEN_COMPANY_API_KEY: str

    # Arize Phoenix
    PHOENIX_API_KEY: str = ""

    # Celery
    CELERY_BROKER_URL: str = "redis://localhost:6379/0"
    CELERY_RESULT_BACKEND: str = "redis://localhost:6379/0"

    # ML Models
    MODEL_DIR: str = "./models"

    class Config:
        env_file = ".env"

settings = Settings()
```

9.3.3 `app/api/v1/webhooks.py` - Webhook Handler

python

```
from fastapi import APIRouter, Request, HTTPException, BackgroundTasks
from app.core.security import verify_github_signature
from app.workers.tasks import process_pr_analysis
import logging

router = APIRouter()
logger = logging.getLogger(__name__)

@router.post("/webhook/github")
async def github_webhook(
    request: Request,
    background_tasks: BackgroundTasks
):
    """
    Handle GitHub webhook events
    """
    # Get headers
    event_type = request.headers.get("X-GitHub-Event")
    delivery_id = request.headers.get("X-GitHub-Delivery")
    signature = request.headers.get("X-Hub-Signature-256")
```

```
    # Get body
    body = await request.body()

    # Verify signature
    if not verify_github_signature(body, signature):
        raise HTTPException(status_code=401, detail="Invalid signature")

    # Parse payload
    payload = await request.json()

    # Handle pull_request event
    if event_type == "pull_request":
        action = payload.get("action")

        if action in ["opened", "synchronize", "reopened"]:
            pr_data = payload["pull_request"]
            repo_data = payload["repository"]

            # Queue analysis task
            analysis_id = f"{repo_data['full_name']}-{pr_data['number']}-{delivery_id}"

            background_tasks.add_task(
                process_pr_analysis,
                analysis_id=analysis_id,
                repo_full_name=repo_data["full_name"],
                pr_number=pr_data["number"],
                pr_data=pr_data
            )

            return {
                "status": "accepted",
                "analysis_id": analysis_id,
                "message": "Analysis queued for processing"
            }

    # Acknowledge other events
    return {"status": "acknowledged"}
```

9.3.4 `app/workers/tasks.py` - Celery Tasks

python

```
from celery import Celery
from app.core.config import settings
from app.services.analysis_service import AnalysisService
import logging
```

```python
celery_app = Celery(
    "protectsus",
    broker=settings.CELERY_BROKER_URL,
    backend=settings.CELERY_RESULT_BACKEND
)

logger = logging.getLogger(__name__)

@celery_app.task(bind=True, max_retries=3)
def process_pr_analysis(
    self,
    analysis_id: str,
    repo_full_name: str,
    pr_number: int,
    pr_data: dict
):
    """
    Background task to analyze PR
    """
    try:
        logger.info(f"Starting analysis {analysis_id}")

        # Initialize service
        service = AnalysisService()

        # Run analysis
        result = await service.analyze_pr(
            repo_full_name=repo_full_name,
            pr_number=pr_number,
            pr_data=pr_data
        )

        logger.info(f"Analysis {analysis_id} completed")

        return result

    except Exception as e:
        logger.error(f"Analysis {analysis_id} failed: {e}")

        # Retry with exponential backoff
        raise self.retry(exc=e, countdown=60 * (2 ** self.request.retries))
```

9.3.5 `app/services/analysis_service.py` - Main Analysis Pipeline

python

```python
from app.services.github_service import GitHubService
from app.services.compression_service import CompressionService
from app.services.agent_service import MultiAgentAnalyzer
from app.services.knowledge_graph import CodeKnowledgeGraph
from app.models.database import AnalysisModel
import logging
from datetime import datetime

logger = logging.getLogger(__name__)

class AnalysisService:
    """
    Main analysis pipeline orchestrator
    """

    def __init__(self):
        self.github = GitHubService()
        self.compressor = CompressionService()
        self.analyzer = MultiAgentAnalyzer()
        self.knowledge_graph = CodeKnowledgeGraph()

    async def analyze_pr(
        self,
        repo_full_name: str,
        pr_number: int,
        pr_data: dict
    ):
        """
        Run complete analysis pipeline
        """
        analysis_id = f"{repo_full_name}-{pr_number}-{datetime.utcnow().timestamp()}"

        # 1. Create analysis record
        analysis = AnalysisModel.create(
            analysis_id=analysis_id,
            repository=repo_full_name,
            pr_number=pr_number,
            status="processing",
            pr_data=pr_data
        )

        try:
            # 2. Retrieve code
            logger.info("Retrieving repository code...")
            repo_path, pr_diff = await self.github.fetch_pr_code(
                repo_full_name,
                pr_number
```

```
        )

        # 3. Compress codebase
        logger.info("Compressing codebase...")
        compressed_context = await self.compressor.compress_repository(
            repo_path,
            pr_diff
        )

        analysis.update_compression_metrics(compressed_context.metrics)

        # 4. Run multi-agent analysis
        logger.info("Running multi-agent analysis...")
        security_report = await self.analyzer.analyze(
            compressed_context=compressed_context,
            pr_diff=pr_diff
        )

        analysis.update_security_report(security_report)

        # 5. Generate fixes
        logger.info("Generating fixes...")
        fixes = await self.generate_fixes(security_report, repo_path)

        # 6. Create fix PR
        logger.info("Creating fix PR...")
        fix_pr = await self.github.create_fix_pr(
            repo_full_name=repo_full_name,
            base_branch=pr_data["base"]["ref"],
            fixes=fixes,
            security_report=security_report,
            original_pr_number=pr_number
        )

        analysis.update_fix_pr(fix_pr)

        # 7. Update knowledge graph
        logger.info("Updating knowledge graph...")
        await self.knowledge_graph.build_graph(repo_path, repo_full_name)
        await self.knowledge_graph.add_vulnerability_nodes(security_report, repo_full_name)

        # 8. Complete analysis
        analysis.mark_completed()

        return analysis.to_dict()

    except Exception as e:
```

```
        logger.error(f"Analysis failed: {e}")
        analysis.mark_failed(str(e))
        raise
```

10. Deployment Instructions

10.1 DigitalOcean Deployment

10.1.1 Infrastructure Setup

Droplet Configuration:

- **Plan:** CPU-Optimized (4 vCPUs, 8 GB RAM)
- **OS:** Ubuntu 22.04 LTS
- **Region:** Choose closest to users
- **Additional:** Enable backups, monitoring

Initial Server Setup:

bash

```
# SSH into droplet
ssh root@your-server-ip

# Update system
apt update && apt upgrade -y

# Install Docker
curl -fsSL https://get.docker.com -o get-docker.sh
sh get-docker.sh

# Install Docker Compose
apt install docker-compose -y

# Create app user
adduser protectsus
usermod -aG docker protectsus
usermod -aG sudo protectsus

# Switch to app user
su - protectsus
```

10.1.2 Application Deployment

Clone Repository:

bash
```
git clone https://github.com/your-org/protectsus.git
cd protectsus
```
Configure Environment:

bash
```
cp .env.example .env
nano .env  # Edit with production values
```
Docker Compose Production Setup:

`docker-compose.prod.yml`:

yaml
```
version: '3.8'

services:
  app:
    build:
      context: .
      dockerfile: docker/Dockerfile
    container_name: protectsus-app
    restart: always
    env_file: .env
    ports:
      - "8000:8000"
    depends_on:
      - mongodb
      - neo4j
      - redis
    volumes:
      - ./models:/app/models
      - ./logs:/app/logs
    networks:
      - protectsus-network

  celery-worker:
    build:
      context: .
      dockerfile: docker/Dockerfile
    container_name: protectsus-worker
    restart: always
    env_file: .env
```

```
  command: celery -A app.workers.celery_app worker --loglevel=info
  depends_on:
    - redis
    - mongodb
  volumes:
    - ./models:/app/models
    - ./logs:/app/logs
  networks:
    - protectsus-network

mongodb:
  image: mongo:7.0
  container_name: protectsus-mongodb
  restart: always
  environment:
    MONGO_INITDB_ROOT_USERNAME: ${MONGODB_USER}
    MONGO_INITDB_ROOT_PASSWORD: ${MONGODB_PASSWORD}
  volumes:
    - mongodb-data:/data/db
  networks:
    - protectsus-network

neo4j:
  image: neo4j:5.15
  container_name: protectsus-neo4j
  restart: always
  environment:
    NEO4J_AUTH: ${NEO4J_USER}/${NEO4J_PASSWORD}
    NEO4J_PLUGINS: '["apoc", "graph-data-science"]'
  volumes:
    - neo4j-data:/data
  networks:
    - protectsus-network

redis:
  image: redis:7.2-alpine
  container_name: protectsus-redis
  restart: always
  volumes:
    - redis-data:/data
  networks:
    - protectsus-network

nginx:
  image: nginx:1.24
  container_name: protectsus-nginx
  restart: always
```

```
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./docker/nginx.conf:/etc/nginx/nginx.conf
      - ./ssl:/etc/nginx/ssl
    depends_on:
      - app
    networks:
      - protectsus-network

volumes:
  mongodb-data:
  neo4j-data:
  redis-data:

networks:
  protectsus-network:
    driver: bridge
```

Nginx Configuration:

`docker/nginx.conf`:

```
nginx
events {
    worker_connections 1024;
}

http {
    upstream protectsus_app {
        server app:8000;
    }

    server {
        listen 80;
        server_name api.protectsus.ai;

        # Redirect to HTTPS
        return 301 https://$server_name$request_uri;
    }

    server {
        listen 443 ssl http2;
        server_name api.protectsus.ai;
```

```
        ssl_certificate /etc/nginx/ssl/fullchain.pem;
        ssl_certificate_key /etc/nginx/ssl/privkey.pem;

        # Security headers
        add_header X-Frame-Options "SAMEORIGIN" always;
        add_header X-XSS-Protection "1; mode=block" always;
        add_header X-Content-Type-Options "nosniff" always;

        # Webhook endpoint (no auth)
        location /api/v1/webhook {
            proxy_pass http://protectsus_app;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;

            # Increase timeout for long-running webhooks
            proxy_connect_timeout 300s;
            proxy_send_timeout 300s;
            proxy_read_timeout 300s;
        }

        # All other endpoints
        location / {
            proxy_pass http://protectsus_app;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }

        # Health check
        location /health {
            access_log off;
            proxy_pass http://protectsus_app;
        }
    }
}
```

Deploy:

bash

```
# Build and start services
docker-compose -f docker-compose.prod.yml up -d --build

# Check logs
```

```
docker-compose -f docker-compose.prod.yml logs -f app

# Initialize databases
docker-compose -f docker-compose.prod.yml exec app python scripts/setup_databases.py
```

10.1.3 SSL Certificate

```
bash
# Install Certbot
snap install --classic certbot

# Get certificate
certbot certonly --standalone -d api.protectsus.ai

# Copy certificates
cp /etc/letsencrypt/live/api.protectsus.ai/fullchain.pem ./ssl/
cp /etc/letsencrypt/live/api.protectsus.ai/privkey.pem ./ssl/

# Restart Nginx
docker-compose -f docker-compose.prod.yml restart nginx
```

10.2 AWS Deployment (Alternative)

10.2.1 Infrastructure

Services:

- **Compute:** ECS Fargate
- **Database:** DocumentDB (MongoDB compatible)
- **Graph DB:** Neptune (Neo4j alternative) OR self-hosted Neo4j on EC2
- **Cache:** ElastiCache (Redis)
- **Load Balancer:** Application Load Balancer
- **Storage:** S3 (for models)

10.2.2 ECS Task Definition

```
json
{
  "family": "protectsus",
  "networkMode": "awsvpc",
  "requiresCompatibilities": ["FARGATE"],
  "cpu": "2048",
  "memory": "4096",
  "containerDefinitions": [
```

```
    {
      "name": "app",
      "image": "your-ecr-repo/protectsus:latest",
      "portMappings": [
        {
          "containerPort": 8000,
          "protocol": "tcp"
        }
      ],
      "environment": [
        {"name": "MONGODB_URL", "value": "your-documentdb-url"},
        {"name": "NEO4J_URI", "value": "your-neptune-url"},
        {"name": "REDIS_URL", "value": "your-elasticache-url"}
      ],
      "secrets": [
        {
          "name": "ANTHROPIC_API_KEY",
          "valueFrom": "arn:aws:secretsmanager:region:account:secret:anthropic-key"
        }
      ],
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "/ecs/protectsus",
          "awslogs-region": "us-east-1",
          "awslogs-stream-prefix": "ecs"
        }
      }
    }
  ]
}
```

11. Environment Configuration

11.1 Environment Variables

`.env.example`:

bash

```
# Application
APP_NAME=ProtectSUS
DEBUG=False
ALLOWED_ORIGINS=https://protectsus.ai,https://www.protectsus.ai
```

```
# GitHub App
GITHUB_APP_ID=123456
GITHUB_PRIVATE_KEY="-----BEGIN RSA PRIVATE KEY-----\n...\n-----END RSA
PRIVATE KEY-----"
GITHUB_WEBHOOK_SECRET=your-webhook-secret-here

# MongoDB
MONGODB_URL=mongodb://user:password@localhost:27017/
MONGODB_DB_NAME=protectsus

# Neo4j
NEO4J_URI=bolt://localhost:7687
NEO4J_USER=neo4j
NEO4J_PASSWORD=your-neo4j-password

# Redis
REDIS_URL=redis://localhost:6379/0

# Anthropic Claude API
ANTHROPIC_API_KEY=sk-ant-...

# Token Company API
TOKEN_COMPANY_API_KEY=your-token-company-key

# Arize Phoenix
PHOENIX_API_KEY=your-phoenix-key

# Celery
CELERY_BROKER_URL=redis://localhost:6379/0
CELERY_RESULT_BACKEND=redis://localhost:6379/0

# ML Models
MODEL_DIR=./models

# Logging
LOG_LEVEL=INFO
```

12. Testing & Quality Assurance

12.1 Unit Tests

`tests/unit/test_agent_service.py`:

python

```python
import pytest
from app.services.agent_service import MultiAgentAnalyzer

@pytest.mark.asyncio
async def test_vaa_agent_detects_sql_injection():
    """
    Test VAA agent can detect SQL injection
    """
    analyzer = MultiAgentAnalyzer()

    code = """
    def login(username, password):
        query = f"SELECT * FROM users WHERE username = '{username}'"
        cursor.execute(query)
    """

    findings = await analyzer.vaa_agent.analyze(code)

    assert len(findings) > 0
    assert any(f.type == "SQL Injection" for f in findings)
```

12.2 Integration Tests

`tests/integration/test_github_integration.py`:

python

```python
import pytest
from app.services.github_service import GitHubService

@pytest.mark.integration
@pytest.mark.asyncio
async def test_create_fix_pr():
    """
    Test creating fix PR on GitHub
    """
    service = GitHubService()

    # Create test fix PR
    pr = await service.create_fix_pr(
        repo_full_name="test-org/test-repo",
        base_branch="main",
        fixes=[...],
        security_report={...},
        original_pr_number=1
    )
```

```
    assert pr is not None
    assert pr.title.startswith("🛡️ [ProtectSUS]")
```

12.3 E2E Tests

`tests/e2e/test_full_pipeline.py`:

python

```python
import pytest
from app.services.analysis_service import AnalysisService

@pytest.mark.e2e
@pytest.mark.asyncio
async def test_full_analysis_pipeline():
    """
    Test complete analysis pipeline end-to-end
    """
    service = AnalysisService()

    result = await service.analyze_pr(
        repo_full_name="test-org/test-repo",
        pr_number=1,
        pr_data={...}
    )

    assert result["status"] == "completed"
    assert result["security_report"] is not None
    assert result["fix_pr"] is not None
```