PRIVCHECK: Advanced Static Analysis Security Tool

COURSE: Compiler Construction

SUBMITTED TO: Sir Hassam Shakil

GROUP MEMBER:

NAME	**REG ID**
ABDUL SARIM KHAN	021-22-62527

Abstract

This report presents **PRIVCHECK**, a comprehensive static analysis system refactored to align with **Compiler Construction** principles. Originally conceived as a pattern-matching tool, the system now implements a full compiler front-end pipeline—**Lexical**, **Syntactic**, and **Semantic Analysis**—to detect high-privileged system command misuse. By moving beyond simple keyword detection to structural and logical verification, PRIVCHECK ensures that system scripts follow secure operational guidelines.

1. Project Scope & CC Requirements

Traditional pattern-matching tools lack the structural awareness to distinguish between benign text and high-risk operations. PRIVCHECK solves this by implementing a full compiler pipeline—Lexical, Syntactic, and Semantic analysis—to validate command hierarchies and logical intent. This transition ensures scripts are not only syntactically valid but also semantically secure.

2. Introduction:

2.1. Lexical Analysis

The lexer is the entry point of the compiler. It transforms the source script into a list of Token objects.

2.1.1. Tokenization Strategy

The lexer utilizes regular expressions to identify distinct categories:

- **SUDO:** Privilege escalation keywords.
- **FLAG:** Command switches (e.g., -rf, -la).
- **PATH:** Absolute or relative system paths.
- **PERM:** Numeric permission sets (e.g., 777).
- **IDENTIFIER:** Base command names.
- **UNKNOWN:** Malformed or unrecognized characters **(triggers "Red" UI state)**.

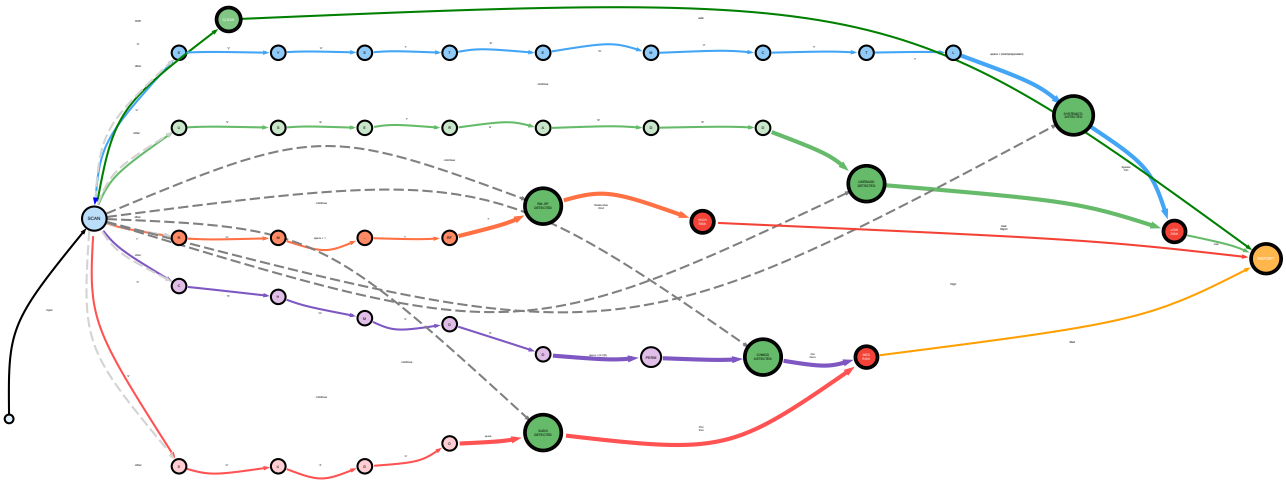

Figure 1- Lexical DFA

2.2.Syntactic Analysis

The Syntactic Analyzer enforces a structural standard based on our defined Context-Free Grammar (CFG).

2.2.1. Context-Free Grammar (CFG)

The grammar is defined as follows:

- **Script ->** {Command NEWLINE | NEWLINE}*

- **Command ->** [SUDO] IDENTIFIER {FLAG | PATH | PERM | ARG}*

2.2.2. Abstract Syntax Tree (AST)

The parser produces an AST for every command. This structure allows the compiler to understand the relationship between a command and its arguments.

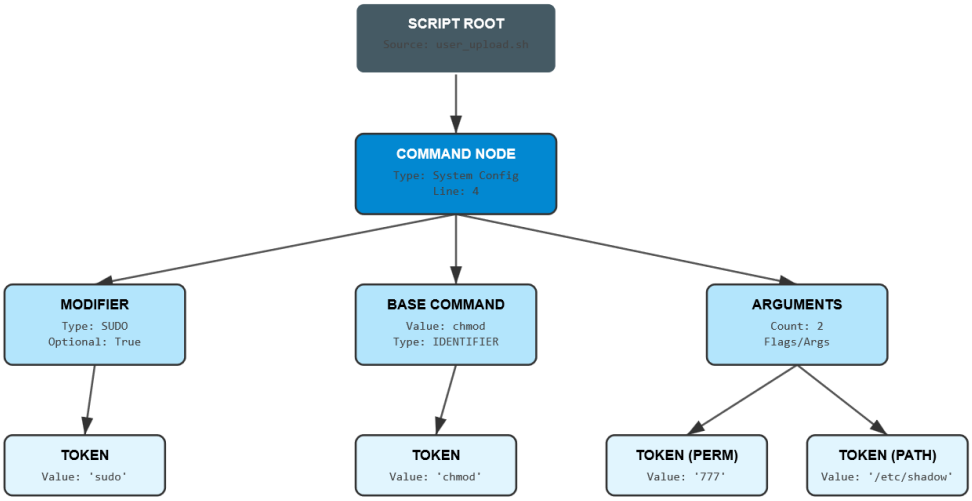

Figure 2- Abstract Syntax Tree

2.3. Semantic Analysis

Semantic analysis ensures the script is logically sound. While **rm -rf /** is syntactically perfect, it is semantically catastrophic.

2.3.1. Semantic Rules

- **Redundancy Check:** Flagging **sudo su** as redundant escalation.
- **Context-Aware Permissions:** Warning if **chmod 777** is applied to sensitive paths like **/etc or /root**.
- **Destructive Bounds:** Detecting recursive deletions targeted at the system root.

2.3.2. Annotated Semantic Tree

During this phase, the **AST** is "annotated" with risk metadata and warnings.

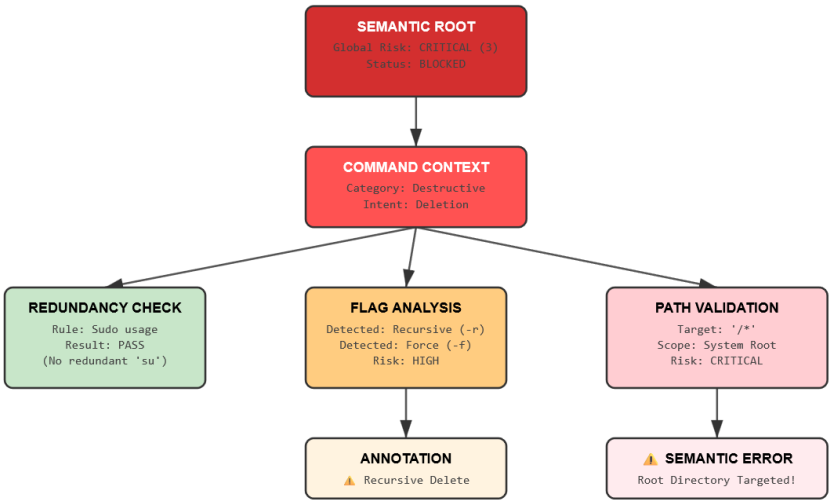

Figure 3- Annotated Semantic Tree

3. Implementation & System Flow

- **Lexer (lexer.py):** Strips comments and produces tokens.
- **Parser (parser_module.py):** Validates structure and builds the **AST**.
- **Semantic Engine (semantics.py):** Audits the **AST** against security rules.
- **Security Detector (detector.py):** Maps commands to threat categories.
- **UI (app.py):** Renders the dashboard and risk gauge.

4. Results & Discussion

4.1. Testing Results

- **Benign Scripts:** Successfully pass all three compiler stages, triggering the Winning Celebration animation.
- **Syntactic Errors:** Corrected misordered tokens **(e.g., -la ls)** are identified as errors.
- **Semantic Risks:** Logically dangerous commands are flagged even if they follow correct syntax.

4.2. CC Scope Alignment

The transition from a **DFA-based** scanner **(TOA)** to a full-front-end compiler **(CC)** ensures that the project provides a comprehensive audit. By analyzing the Context **(Semantics)** and Structure **(Syntax)**, **PRIVCHECK** offers far superior protection compared to simple string matching.

4.3. Screenshots

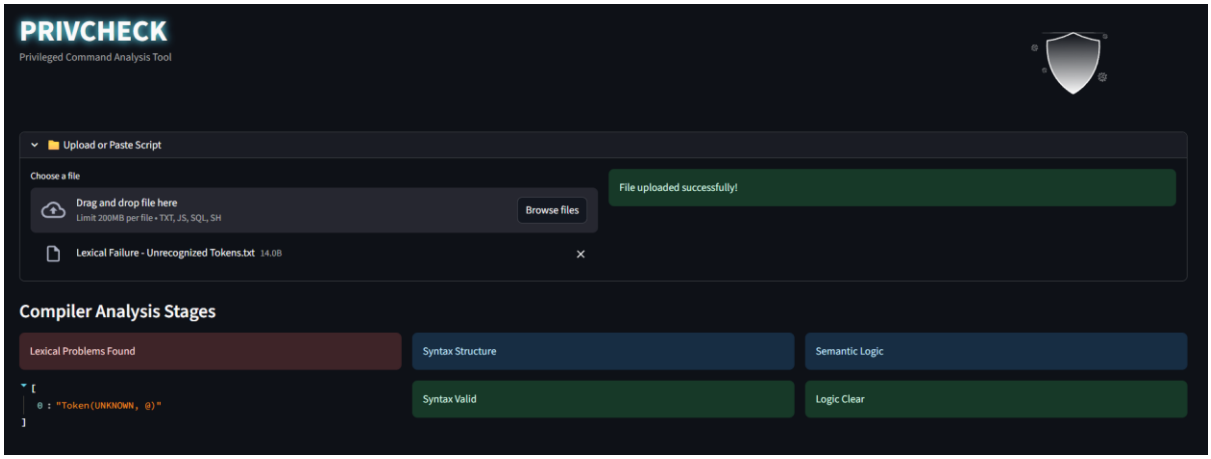

Figure 4- Lexical Error Detected on Script with Faulty Structure

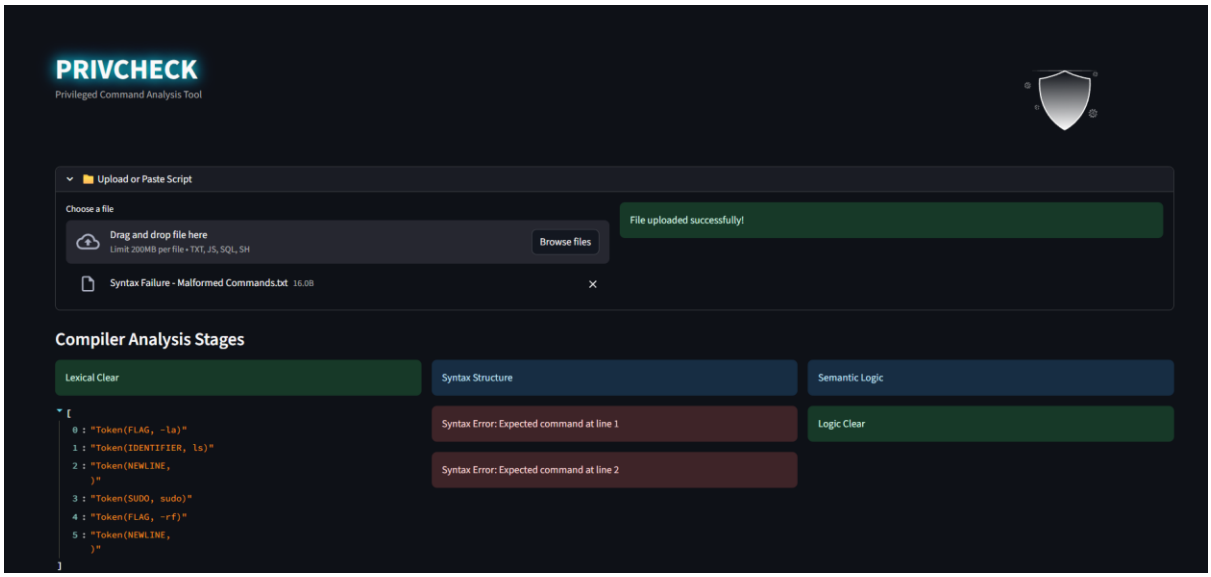

Figure 5- Syntax Error Detected on Script with Faulty Syntax

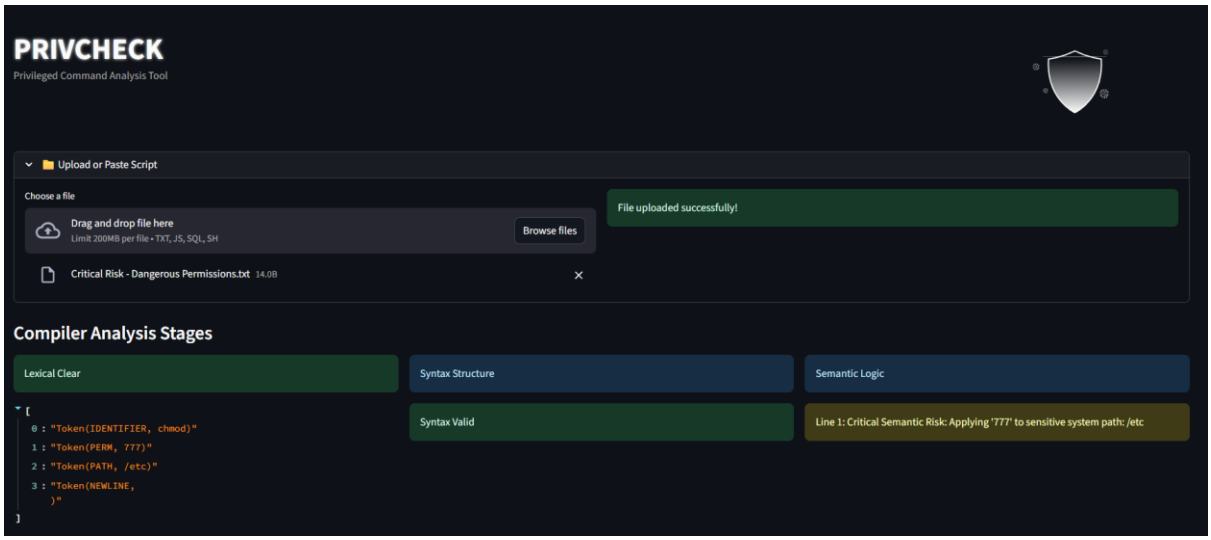

Figure 6- Semantic Error Detected on Malicious Script

5. Conclusion

PRIVCHECK demonstrates the practical application of Compiler Construction in cybersecurity. By implementing **Lexical**, **Syntactic**, and **Semantic** analyzers, we have created a tool that understands the structure and meaning of system code, effectively identifying risks before execution.

6. Appendix:

6.1. app.py (Main):

```
import streamlit as st
from streamlit_lottie import st_lottie
import json
import os
import plotly.graph_objects as go
from detector import detect_patterns, calculate_severity_and_tier
from parser_module import SyntaxParser
from lexer import tokenize
from semantics import SemanticAnalyzer

st.set_page_config(
    page_title="PRIVCHECK - Script Integrity Analysis",
    page_icon="🛡️",
    layout="wide",
    initial_sidebar_state="expanded"
)

def load_lottie_file(filepath: str):
    paths_to_try = [
        filepath,
        os.path.join("assets", filepath),
        os.path.join(os.path.dirname(__file__), filepath)
    ]
    for path in paths_to_try:
        try:
            with open(path, "r", encoding='utf-8') as f:
                return json.load(f)
        except:
            continue
    return None

def local_css(file_name):
    try:
        file_path = os.path.join(os.path.dirname(__file__), file_name)
        with open(file_path, encoding='utf-8') as f:
            st.markdown(f"<style>{f.read()}</style>", unsafe_allow_html=True)
    except Exception:
        pass

local_css("styles.css")


def animated_header():

    col1, col2 = st.columns([4, 1])

    with col1:
        st.markdown("""
        <div style="text-align: left; margin-bottom: 2rem;">
            <h1 class="glow" style="margin: 0; padding: 0;">PRIVCHECK</h1>
            <p class="subheader" style="margin: 0;">Privileged Command Analysis Tool</p>
        </div>
        """, unsafe_allow_html=True)

    with col2:
```

```
        shield_data = load_lottie_file("shield_animation.json")
        if shield_data:
            st_lottie(shield_data, height=150, key="shield")

def user_input():
    with st.expander("📁 Upload or Paste Script", expanded=True):
        col1, col2 = st.columns(2)
        with col1:
            uploaded_file = st.file_uploader("Choose a file", type=["txt", "js", "sql",
"sh"])
        with col2:
            if uploaded_file:
                script = uploaded_file.read().decode("utf-8")
                st.success("File uploaded successfully!")
                return script
            else:
                script = st.text_area("Or paste your script here", height=200)
                return script if script else None

def main():
    animated_header()
    script = user_input()

    if script:
        tokens = tokenize(script)
        parser = SyntaxParser(tokens)
        commands, syntax_errors = parser.parse()
        semantic_engine = SemanticAnalyzer()
        semantic_warnings = semantic_engine.analyze(commands)

        matches, found_types = detect_patterns(script)
        severity, tier = calculate_severity_and_tier(found_types)

        st.subheader("Compiler Analysis Stages")
        col1, col2, col3 = st.columns(3)

        with col1:
            lexical_errors = [t for t in tokens if t.type == 'UNKNOWN']

            if lexical_errors:
                st.error("Lexical Problems Found")
                st.write(lexical_errors)
            else:
                st.success("Lexical Clear")
                st.write(tokens[:10])

        with col2:
            st.info("Syntax Structure")
            if syntax_errors:
                for err in syntax_errors:
                    st.error(err)
            else:
                st.success("Syntax Valid")

        with col3:
            st.info("Semantic Logic")
            if semantic_warnings:
                for w in semantic_warnings:
                    st.warning(f"Line {w['line']}: {w['msg']}")
            else:
                st.success("Logic Clear")

        if not matches and not syntax_errors:
            st.balloons()
            celebration = load_lottie_file("celebration.json")
            if celebration:
                st_lottie(celebration, height=300, key="win_anim")

if __name__ == "__main__":
```

```
    main()
```

6.2. detector.py:

```
import re
from patterns import (
    PRIV_ESCALATION_PATTERNS,
    USER_MANAGEMENT_PATTERNS,
    FILE_PERMISSION_PATTERNS,
    SYSTEM_CONFIG_PATTERNS,
    DESTRUCTIVE_COMMAND_PATTERNS,
    INFO_COMMAND_PATTERNS,
    NETWORK_COMMAND_PATTERNS
)

def detect_patterns(script):
    matches = []
    found_types = set()

    cleaned_script = re.sub(r'//.*?$|/\*.*?\*/|#.*?$', '', script, flags=re.MULTILINE |
re.DOTALL)

    categories = {
        'Privilege Escalation': PRIV_ESCALATION_PATTERNS,
        'User Management': USER_MANAGEMENT_PATTERNS,
        'File Permission': FILE_PERMISSION_PATTERNS,
        'System Config': SYSTEM_CONFIG_PATTERNS,
        'Destructive Command': DESTRUCTIVE_COMMAND_PATTERNS,
        'Sensitive Info Access': INFO_COMMAND_PATTERNS,
        'Network Admin': NETWORK_COMMAND_PATTERNS
    }

    for category, patterns in categories.items():
        for pattern in patterns:
            for match in re.finditer(pattern, cleaned_script, re.IGNORECASE):
                line_start = cleaned_script.rfind('\n', 0, match.start()) + 1
                line_end = cleaned_script.find('\n', match.end())
                command = cleaned_script[line_start:line_end].strip()

                matches.append((category, command))
                found_types.add(category)

    return matches, found_types

def calculate_severity_and_tier(found_types):
    if not found_types:
        return 0, "No Privileged Commands"

    severity_map = {
        'Destructive Command': 3,
        'Privilege Escalation': 3,
        'File Permission': 2,
        'System Config': 2,
        'User Management': 2,
        'Network Admin': 1,
        'Sensitive Info Access': 1
    }

    max_severity = max(severity_map.get(t, 0) for t in found_types)

    tier_map = {
        0: "Clean",
        1: "Low Risk",
        2: "Medium Risk",
        3: "High Risk"
    }
```

```
    return max_severity, tier_map.get(max_severity, "Unknown")
```

6.3. lexer.py:

```
import re

class Token:
    def __init__(self, type, value, line):
        self.type = type
        self.value = value
        self.line = line

    def __repr__(self):
        return f"Token({self.type}, {self.value})"

def tokenize(script):
    tokens = []
    script = re.sub(r'//.*?$|/\*.*?\*/|#.*?$', '', script, flags=re.MULTILINE | re.DOTALL)

    lines = script.split('\n')
    for line_num, line in enumerate(lines, 1):
        if not line.strip(): continue

        parts = re.findall(r'(?:[^\s"\']|"(?:\\.|[^"])*"|\'(?:\\.|[^\'])*\')+', line)

        for part in parts:
            if part.lower() == 'sudo':
                tokens.append(Token('SUDO', part, line_num))
            elif part.startswith('-'):
                tokens.append(Token('FLAG', part, line_num))
            elif '/' in part or part.startswith('C:\\'):
                tokens.append(Token('PATH', part, line_num))
            elif re.match(r'^[0-7]{3,4}$', part):
                tokens.append(Token('PERM', part, line_num))
            elif re.match(r'^[a-zA-Z_][a-zA-Z0-9_]*$', part):
                tokens.append(Token('IDENTIFIER', part, line_num))
            else:
                tokens.append(Token('UNKNOWN', part, line_num))

        tokens.append(Token('NEWLINE', '\n', line_num))

    return tokens
```

6.4. parser_module.py:

```
class SyntaxParser:
    def __init__(self, tokens):
        self.tokens = tokens
        self.pos = 0
        self.errors = []

    def peek(self):
        return self.tokens[self.pos] if self.pos < len(self.tokens) else None

    def consume(self):
        token = self.peek()
        self.pos += 1
        return token

    def parse(self):
        commands = []
        while self.peek():
            if self.peek().type == 'NEWLINE':
                self.consume()
                continue
```

```
            cmd_node = {'sudo': False, 'base': None, 'flags': [], 'args': [], 'line': 0}

            if self.peek().type == 'SUDO':
                cmd_node['sudo'] = True
                cmd_node['line'] = self.peek().line
                self.consume()

            if self.peek() and self.peek().type == 'IDENTIFIER':
                cmd_node['base'] = self.consume().value
                if not cmd_node['line']: cmd_node['line'] = self.tokens[self.pos-1].line
            else:
                self.errors.append(f"Syntax Error: Expected command at line {self.peek().line
if self.peek() else 'end'}")
                self.consume()
                continue

            while self.peek() and self.peek().type != 'NEWLINE':
                token = self.consume()
                if token.type == 'FLAG': cmd_node['flags'].append(token.value)
                else: cmd_node['args'].append(token.value)

            commands.append(cmd_node)
        return commands, self.errors
```

6.5. patterns.py:

```
PRIV_ESCALATION_PATTERNS = [
    r"\bsudo\b",
    r"\bsu\s+root\b",
    r"\brunas\b",
    r"\bsetuid\b",
    r"\bdoas\b",
]

USER_MANAGEMENT_PATTERNS = [
    r"\badduser\b",
    r"\buseradd\b",
    r"\busermod\b",
    r"\bnet\s+user\b",
    r"\bnet\s+localgroup\b",
    r"\bdsadd\b",
    r"\bpasswd\b",
    r"\bgroupadd\b",
]

FILE_PERMISSION_PATTERNS = [
    r"\bchmod\s+[0-7]{3,4}\b",
    r"\bchown\b",
    r"\bicacls\b",
    r"\battrib\b",
    r"\bsetfacl\b",
    r"\btakeown\b",
]

SYSTEM_CONFIG_PATTERNS = [
    r"\bsystemctl\s+(enable|disable|start|stop|restart)\b",
    r"\bsc\s+config\b",
    r"\breg\s+(add|delete)\b",
    r"\bregedit\b",
    r"\bpowercfg\b",
    r"\bservices\.msc\b",
    r"\bupdate-rc\.d\b",
    r"\bchkconfig\b",
]

DESTRUCTIVE_COMMAND_PATTERNS = [
    r"\brm\s+-rf\s+[^\s]*",
```

```
    r"\bdel\s+/s\s+/q\b",
    r"\bmkfs\b",
    r"\bformat\s+\w:",
    r"\bshutdown\b",
    r"\breboot\b",
    r"\bdd\s+if=.*of=.*",
    r"\bmv\s+/system\b",
]

INFO_COMMAND_PATTERNS = [
    r"\bcat\s+/etc/shadow\b",
    r"\bcat\s+/etc/passwd\b",
    r"\btype\s+.*\.(pem|key)\b",
    r"\bsudo\s+-l\b",
    r"\bwhoami\s+/priv\b",
]

NETWORK_COMMAND_PATTERNS = [
    r"\biptables\b",
    r"\broute\b",
    r"\bnetstat\b",
    r"\bifconfig\b",
    r"\bip\s+route\b",
    r"\bnetsh\b",
]
```

6.6. semantics.py:

```
class SemanticAnalyzer:
    def __init__(self):
        self.warnings = []

    def analyze(self, commands):
        for cmd in commands:
            base = cmd['base'].lower()

            if cmd['sudo'] and base in ['su', 'doas']:
                self.warnings.append({
                    'line': cmd['line'],
                    'msg': f"Semantic Warning: Redundant privilege escalation. '{base}'
already switches users."
                })

            if base == 'chmod' and '777' in cmd['args']:
                target = next((a for a in cmd['args'] if '/' in a), "Unknown")
                if any(sys_path in target for sys_path in ['/etc', '/root', '/var/log']):
                    self.warnings.append({
                        'line': cmd['line'],
                        'msg': f"Critical Semantic Risk: Applying '777' to sensitive system
path: {target}"
                    })

            if base == 'rm' and '-rf' in cmd['flags']:
                if any(p in cmd['args'] for p in ['/', '/etc', '/*']):
                    self.warnings.append({
                        'line': cmd['line'],
                        'msg': "Severe Error: Recursive deletion targeted at system root!"
                    })
        return self.warnings
```

6.7. styles.css:

```
@keyframes glow {
    0% { text-shadow: 0 0 5px #fff; }
    50% { text-shadow: 0 0 20px #00ffea, 0 0 30px #0084ff; }
    100% { text-shadow: 0 0 5px #fff; }
```

```
}

@keyframes fadeIn {
    from { opacity: 0; }
    to { opacity: 1; }
}

@keyframes slideIn {
    from { transform: translateX(-20px); opacity: 0; }
    to { transform: translateX(0); opacity: 1; }
}

@keyframes bounce {
    0%, 20%, 50%, 80%, 100% {transform: translateY(0);}
    40% {transform: translateY(-10px);}
    60% {transform: translateY(-5px);}
}

/* Custom styling */
.header-container {
    text-align: left;
    margin-bottom: 2rem;
}

.glow {
    animation: glow 2s infinite;
    color: white;
    font-size: 3rem;
    font-weight: bold;
    text-align: left;
}

.subheader {
    color: #aaa;
    font-size: 1.2rem;
}

.fade-in {
    animation: fadeIn 1s ease-in;
}

.slide-in {
    animation: slideIn 0.5s ease-out forwards;
}

.bounce {
    animation: bounce 1s;
}

.pattern-card {
    background: rgba(255, 255, 255, 0.1);
    padding: 1rem;
    border-radius: 8px;
    margin-bottom: 0.5rem;
    border-left: 4px solid #0c8fcc !important;;
}

.pattern-type {
    font-weight: bold;
    color: #ff5555;
    margin-right: 1rem;
}

button:hover, .stButton:hover {
    background-color: #0c8fcc !important;
    color: white !important;
    border: 2px solid #0c8fcc !important; /* Blue border on hover */
}
.success-card {
```

```
    background: rgba(0, 255, 0, 0.1);
    padding: 1.5rem;
    border-radius: 8px;
    text-align: center;
    border: 1px solid #00ff00;
}

a:hover {
    color: #0c8fcc;  /* Blue color on hover */
}

.severity-box {
    background: rgba(255, 255, 255, 0.05);
    padding: 1.5rem;
    border-radius: 8px;
    margin-top: 1rem;
}

.command-content {
    padding: 8px;
    background-color: #3a3a3a;
    border-radius: 4px;
    margin: 5px 0;
}
.context-info {
    padding: 8px;
    background-color: #2a4a6a;
    border-radius: 4px;
    margin-top: 5px;
}
.pattern-card {
    transition: all 0.3s ease;
}
.pattern-card:hover {
    transform: translateY(-3px);
    box-shadow: 0 4px 8px rgba(0,0,0,0.2);
}
```