Network Security Expert System with Backward and Forward chaining

Abstract:
This report presents an in-depth exploration of an inference engine designed to diagnose and mitigate network attacks, a critical concern in the realm of cybersecurity. The engine operates within the specialized domain of network security, employing both backward and forward chaining methodologies. Backward chaining is utilized to diagnose network attacks by tracing symptoms to their root causes, while forward chaining is applied to devise and implement mitigation strategies once an attack is identified. The implementation of the system is detailed, covering the methodologies, algorithms, and coding practices adopted. Furthermore, the report provides a comprehensive analysis of the program, including enhancements made for improved efficiency and effectiveness.

1. Problem and Domain

This report addresses the challenge of rapidly detecting and responding to network attacks using an advanced inference engine.

1.1 Problem Description: The challenge involves using the inference engine to detect diverse network attacks. These attacks include, but are not limited to, ransomware, viruses, trojan horses, malware, DDoS attacks, and SQL injections. Each attack type is characterized by unique network behaviors and anomalies. The engine utilizes a set of rules, where each rule corresponds to different combinations of symptoms, such as unauthorized access, file changes, unusual traffic patterns, and external network activities, to identify the specific type of network attack.

1.2 Domain: The domain of the inference engine is network security, with a focus on automating attack detection and response. By analyzing network data, the engine applies backward chaining to deduce the type of attack from observed symptoms. Forward chaining is then employed to recommend or implement mitigation strategies. This dual approach allows the system to address a wide range of cyber threats effectively and efficiently.

2. Methodology

2.1 Design Philosophy: The engine is designed with a focus on modularity and accessibility, ensuring ease of maintenance and adaptability. The knowledge bases were built as complete separate components of the inference engine so keep the program as dynamic as possible. This design choice ensures that any expansion or modification of the knowledge base can be implemented without impacting the overall functionality of the engine

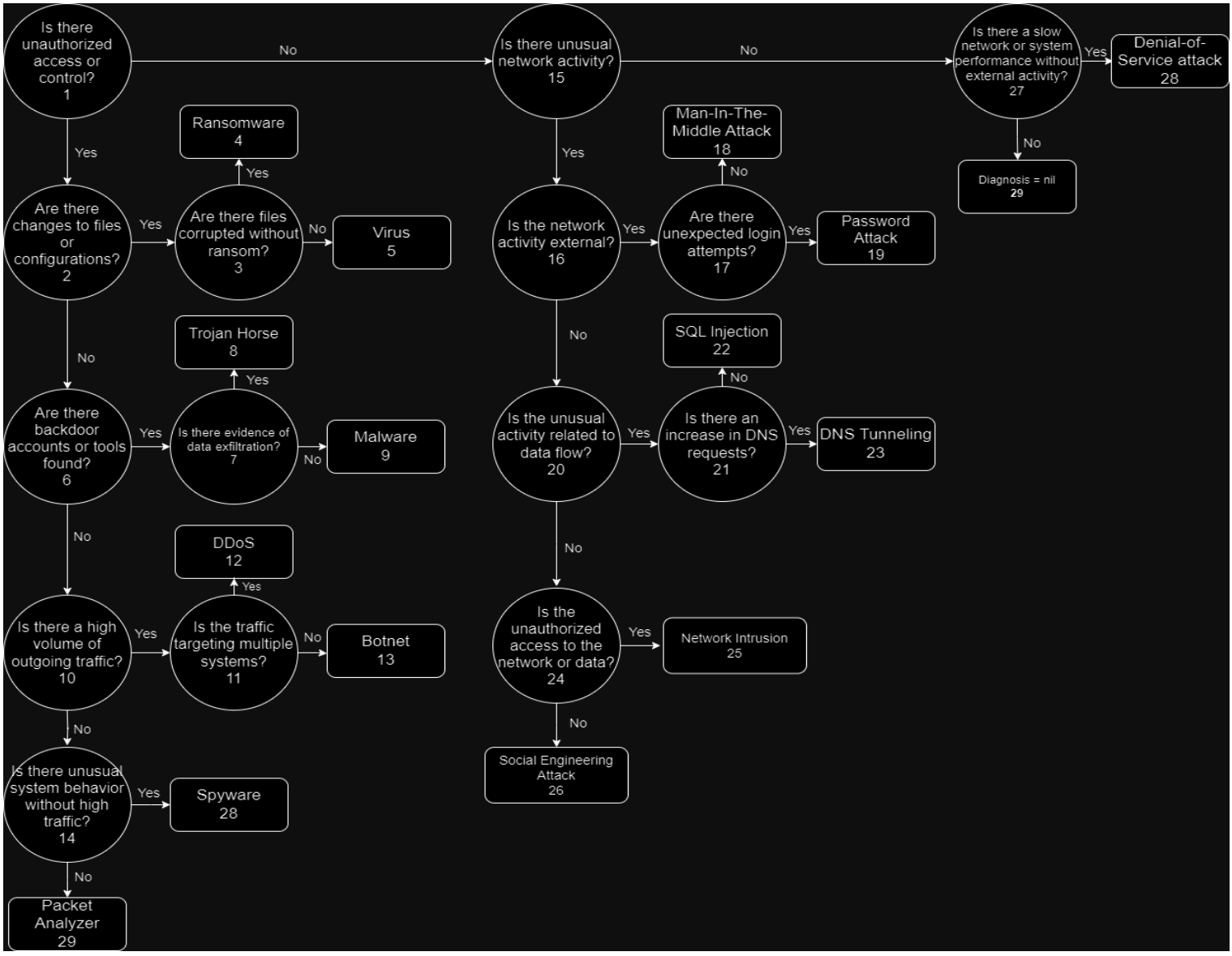
2.2 Knowledge Base Structure: The structure of the knowledge base is deliberately separated from the core algorithms of the inference engine. It is designed for efficient parsing, facilitating the identification of conclusion lists, rules, and other critical

elements. Both backward and forward chaining components share a compact and well-organized format, with rules distinctly numbered and associated with relevant clauses and conclusions, streamlining the engine's operational process.

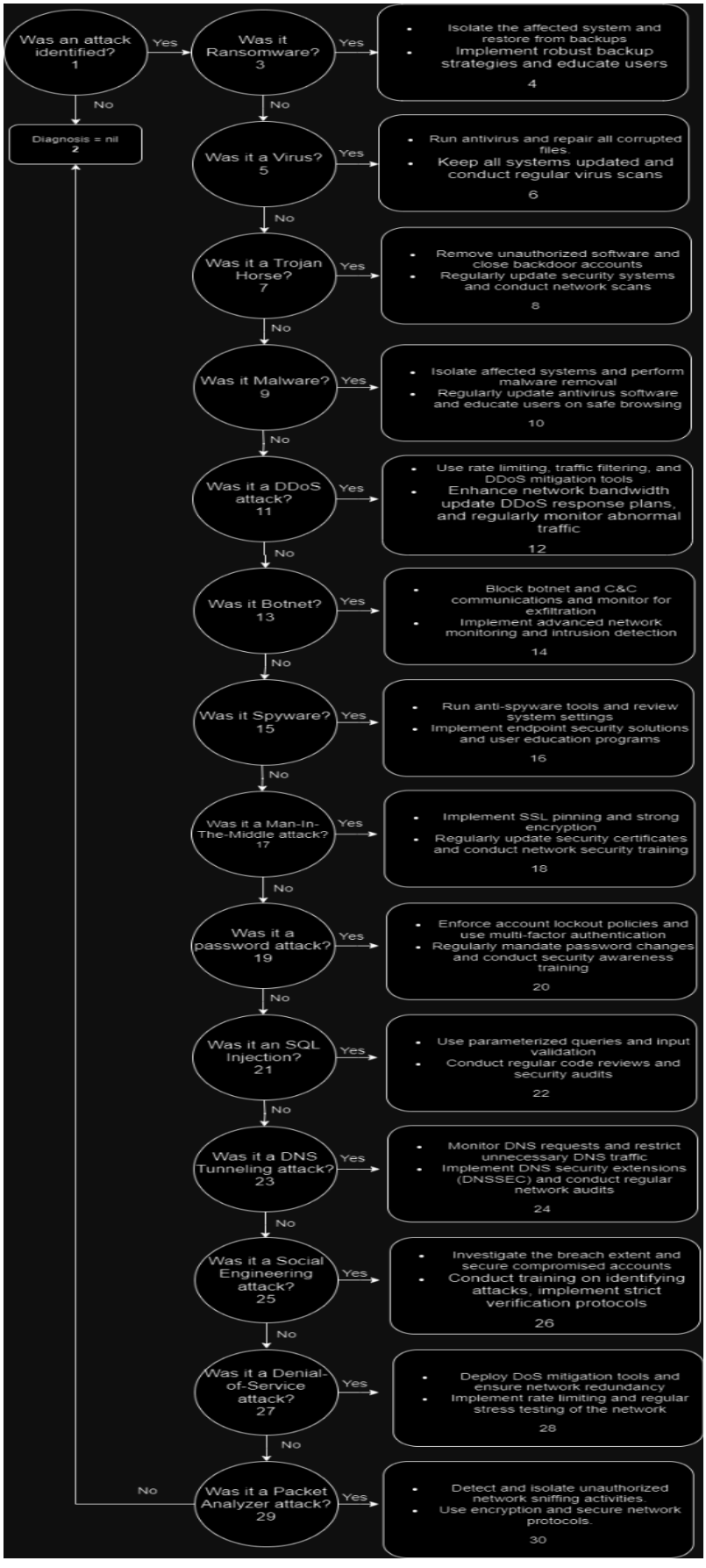
2.3 Decision Tree Conversion: The knowledge bases mirror the decision trees' rules but in a textual format. This transformation ensures that users engaging with the engine can trace the decision-making process parallel to the decision trees, achieving consistent outcomes.

3. Decision Trees

3.1 Backward Chaining Decision Tree



3.2 Forward Chaining Decision Tree



4. Rules

4.1 Backward Chaining Rule Base:

Rule 15: Ransomware Detection

Description:

- Check if there is unauthorized access or control.
- Check if there are changes to files or configurations.
- Check if files are corrupted without a request for ransom.
- If all these conditions are met, it indicates ransomware.

Pseudocode:

```
if unauthorized_access_or_control and file_changes_or_configurations
and files_corrupted_without_ransom:
   conclude("Ransomware")
```

Rule 30: Virus Detection

Description:

- Confirm unauthorized access or control.
- Confirm changes to files or configurations.
- Ensure there are no files corrupted without a request for ransom.
- These signs point to a virus.

Pseudocode:

```
if unauthorized_access_or_control and file_changes_or_configurations
and not files_corrupted_without_ransom:
   conclude("Virus")
```

Rule 45: Trojan Horse Detection

Description:

- Detect unauthorized access or control.
- Ensure no changes to files or configurations.
- Look for backdoor accounts or tools on the system.
- Search for evidence of data exfiltration.
- These factors together suggest a trojan horse.

Pseudocode:

```
if unauthorized_access_or_control and not file_changes_or_configurations
and backdoor_accounts_or_tools_found and
evidence_of_data_exfiltration:
   conclude("Trojan_Horse")
```

Rule 60: Malware Detection

Description:

- Identify unauthorized access or control.
- Verify no changes to files or configurations.
- Find backdoor accounts or tools present.
- Ensure no evidence of data exfiltration.
- This combination of factors indicates malware.

Pseudocode:

```
if unauthorized_access_or_control and not file_changes_or_configurations
and backdoor_accounts_or_tools_found and not
evidence_of_data_exfiltration:
   conclude("Malware")
```

Rule 75: DDoS Attack Detection

Description:

- Ascertain unauthorized access or control.
- Confirm no changes to files or configurations.
- Ensure no backdoor accounts or tools are found.
- Check for a high volume of outgoing traffic.
- Verify if the traffic is targeting multiple systems.
- Together, these conditions signify a DDoS attack.

Pseudocode:

```
if unauthorized_access_or_control and not file_changes_or_configurations
and not backdoor_accounts_or_tools_found and
high_volume_of_outgoing_traffic and traffic_targeting_multiple_systems:
   conclude("DDoS_Attack")
```

Rule 90: Botnet Activity Detection

Description:

- Check for unauthorized access or control.
- Make sure there are no file changes or configurations.
- Confirm no backdoor accounts or tools are found.
- Look for high volumes of outgoing traffic.
- Ensure the traffic isn't targeting multiple systems.
- This pattern of activity typically indicates botnet involvement.

Pseudocode:

```
if unauthorized_access_or_control and not file_changes_or_configurations
and not backdoor_accounts_or_tools_found and
```

```
high_volume_of_outgoing_traffic and not
traffic_targeting_multiple_systems:
   conclude("Botnet_Activity")
```

Rule 105: Spyware Detection
Description:

- Identify unauthorized access or control.
- Verify no changes to files or configurations.
- Ensure no backdoor accounts or tools found.
- Confirm there is no high volume of outgoing traffic.
- Look for unusual system behavior without high traffic volume.
- These conditions usually point to spyware.

Pseudocode:

```
if unauthorized_access_or_control and not file_changes_or_configurations
and not backdoor_accounts_or_tools_found and not
high_volume_of_outgoing_traffic and
unusual_system_behavior_without_high_traffic:
   conclude("Spyware")
```

Rule 120: Packet Analyzer Detection
Description:

- Confirm unauthorized access or control.
- Check for no changes in files or configurations.
- Ensure no backdoor accounts or tools are present.
- Verify no high volume of outgoing traffic.
- Confirm no unusual system behavior without high traffic.
- This scenario suggests the use of a packet analyzer.

Pseudocode:

```
if unauthorized_access_or_control and not file_changes_or_configurations
and not backdoor_accounts_or_tools_found and not
high_volume_of_outgoing_traffic and not
unusual_system_behavior_without_high_traffic:
   conclude("Packet_Analyzer")
```

Rule 135: Password Attack Detection
Description:

- Make sure there's no unauthorized access or control.
- Identify unusual network activity.
- Check if the network activity is external.
- Look for unexpected login attempts.

- These signs typically indicate a password attack.

Pseudocode:

```
if not unauthorized_access_or_control and unusual_network_activity and network_activity_external and unexpected_login_attempts:
    conclude("Password_Attack")
```

Rule 150: Man in the Middle Attack Detection

Description:

- Ensure no unauthorized access or control.
- Detect unusual network activity.
- Confirm the network activity is external.
- Verify there are no unexpected login attempts.
- These conditions often suggest a man-in-the-middle attack.

Pseudocode:

```
if not unauthorized_access_or_control and unusual_network_activity and network_activity_external and not unexpected_login_attempts:
    conclude("Man_in_the_Middle_Attack")
```

Rule 165: DNS Tunneling Detection

Description:

- Verify no unauthorized access or control.
- Look for unusual network activity.
- Ensure the network activity is not external.
- Identify unusual activity related to data flow.
- Check for an increase in DNS requests.
- This pattern is indicative of DNS tunneling.

Pseudocode:

```
if not unauthorized_access_or_control and unusual_network_activity and not network_activity_external and unusual_activity_related_to_data_flow and increase_in_dns_requests:
    conclude("DNS_Tunneling")
```

Rule 180: SQL Injection Detection

Description:

- Confirm no unauthorized access or control.
- Detect unusual network activity.
- Ensure network activity is not external.
- Look for unusual activity related to data flow.
- Verify there is no increase in DNS requests.

- These signs often point to an SQL injection.

Pseudocode:

```
if not unauthorized_access_or_control and unusual_network_activity and
not network_activity_external and unusual_activity_related_to_data_flow
and not increase_in_dns_requests:
    conclude("SQL_Injection")
```

Rule 195: Network Intrusion Detection

Description:

- Ensure no unauthorized access or control.
- Identify unusual network activity.
- Confirm the network activity is not external.
- Check for unauthorized access to the network or data.
- This combination of factors typically indicates a network intrusion.

Pseudocode:

```
if not unauthorized_access_or_control and unusual_network_activity and
not network_activity_external and not
unusual_activity_related_to_data_flow and
unauthorized_access_to_network_or_data:
    conclude("Network_Intrusion")
```

Rule 210: Social Engineering Attack Detection

Description:

- Confirm no unauthorized access or control.
- Detect unusual network activity.
- Verify the network activity is not external.
- Make sure there is no unusual activity related to data flow.
- Confirm no unauthorized access to the network or data.
- These conditions usually suggest a social engineering attack.

Pseudocode:

```
if not unauthorized_access_or_control and unusual_network_activity and
not network_activity_external and not
unusual_activity_related_to_data_flow and not
unauthorized_access_to_network_or_data:
    conclude("Social_Engineering_Attack")
```

Rule 225: Denial of Service Attack Detection

Description:

- Verify no unauthorized access or control.
- Ensure there is no unusual network activity.
- Check for slow network or system performance without external activity.

- These signs often indicate a denial of service attack.

Pseudocode:

```
if not unauthorized_access_or_control and not unusual_network_activity
and slow_network_or_system_performance_without_external_activity:
    conclude("Denial_of_Service_Attack")
```

Rule 240: Nil (No Attack Detected)

Description:

- Ensure no unauthorized access or control.
- Confirm no unusual network activity.
- Verify no slow network or system performance without external activity.
- If all these are true, it indicates no attack (Nil).

Pseudocode:

```
if not unauthorized_access_or_control and not unusual_network_activity
and not
slow_network_or_system_performance_without_external_activity:
    conclude("Nil")
```

4.2 Forward Chaining Rule Base:

Rule 1: No Attack Identified (Nil)

Description:

- Check if no attack was identified.
- If true, it indicates no attack (Nil).

Pseudocode:

```
if not attack_was_identified:
    conclude("Nil")
```

Rule 2: Ransomware Detection

Description:

- Confirm an attack was identified.
- Check if the attack was ransomware.
- If true, it indicates ransomware.

Pseudocode:

```
if attack_was_identified and attack_was_ransomware:
    conclude("Ransomware")
```

Rule 3: Virus Detection

Description:

- Confirm an attack was identified.
- Ensure it was not ransomware.

- Check if the attack was a virus.
- If true, it indicates a virus.

Pseudocode:

```
if attack_was_identified and not attack_was_ransomware and
attack_was_virus:
   conclude("Virus")
```

Rule 4: Trojan Horse Detection

Description:

- Confirm an attack was identified.
- Ensure it was not ransomware or a virus.
- Check if the attack was a trojan horse.
- If true, it indicates a trojan horse.

Pseudocode:

```
if attack_was_identified and not attack_was_ransomware and not
attack_was_virus and attack_was_trojan_horse:
   conclude("Trojan_Horse")
```

Rule 5: Malware Detection

Description:

- Confirm an attack was identified.
- Ensure it was not ransomware, a virus, or a trojan horse.
- Check if the attack was malware.
- If true, it indicates malware.

Pseudocode:

```
if attack_was_identified and not attack_was_ransomware and not
attack_was_virus and not attack_was_trojan_horse and
attack_was_malware:
   conclude("Malware")
```

Rule 6: DDoS Attack Detection

Description:

- Confirm an attack was identified.
- Ensure it was not ransomware, a virus, a trojan horse, or malware.
- Check if the attack was a DDoS attack.
- If true, it indicates a DDoS attack.

Pseudocode:

```
if attack_was_identified and not attack_was_ransomware and not
attack_was_virus and not attack_was_trojan_horse and not
attack_was_malware and attack_was_ddos:
```

conclude("DDoS")

Rule 7: Botnet Detection
Description:

- Confirm an attack was identified.
- Ensure it was not ransomware, a virus, a trojan horse, malware, or a DDoS attack.
- Check if the attack was related to a botnet.
- If true, it indicates botnet activity.

Pseudocode:

```
if attack_was_identified and not attack_was_ransomware and not attack_was_virus and not attack_was_trojan_horse and not attack_was_malware and not attack_was_ddos and attack_was_botnet:
    conclude("Botnet")
```

Rule 8: Spyware Detection
Description:

- Confirm an attack was identified.
- Ensure it was not ransomware, a virus, a trojan horse, malware, a DDoS attack, or botnet activity.
- Check if the attack was spyware.
- If true, it indicates spyware.

Pseudocode:

```
if attack_was_identified and not attack_was_ransomware and not attack_was_virus and not attack_was_trojan_horse and not attack_was_malware and not attack_was_ddos and not attack_was_botnet and attack_was_spyware:
    conclude("Spyware")
```

Rule 9: Man in the Middle Attack Detection
Description:

- Confirm an attack was identified.
- Ensure it was not identified as ransomware, a virus, a trojan horse, malware, a DDoS attack, botnet activity, or spyware.
- Check if the attack was a man-in-the-middle attack.
- If true, it indicates a man-in-the-middle attack.

Pseudocode:

```
if attack_was_identified and not attack_was_ransomware and not attack_was_virus and not attack_was_trojan_horse and not
```

attack_was_malware and not attack_was_ddos and not attack_was_botnet and not attack_was_spyware and attack_was_man_in_middle:
conclude("Man_in_the_Middle")

Rule 10: Password Attack Detection
Description:

- Confirm an attack was identified.
- Ensure it was not ransomware, a virus, a trojan horse, malware, a DDoS attack, botnet activity, spyware, or a man-in-the-middle attack.
- Check if the attack was a password attack.
- If true, it indicates a password attack.

Pseudocode:

if attack_was_identified and not attack_was_ransomware and not attack_was_virus and not attack_was_trojan_horse and not attack_was_malware and not attack_was_ddos and not attack_was_botnet and not attack_was_spyware and not attack_was_man_in_middle and attack_was_password_attack:
conclude("Password_Attack")

Rule 11: SQL Injection Detection
Description:

- Confirm an attack was identified.
- Ensure it was not ransomware, a virus, a trojan horse, malware, a DDoS attack, botnet activity, spyware, a man-in-the-middle attack, or a password attack.
- Check if the attack was an SQL injection.
- If true, it indicates an SQL injection.

Pseudocode:

if attack_was_identified and not attack_was_ransomware and not attack_was_virus and not attack_was_trojan_horse and not attack_was_malware and not attack_was_ddos and not attack_was_botnet and not attack_was_spyware and not attack_was_man_in_middle and not attack_was_password_attack and attack_was_sql_injection:
conclude("SQL_Injection")

Rule 12: DNS Tunneling Detection
Description:

- Confirm an attack was identified.

- Ensure it was not ransomware, a virus, a trojan horse, malware, a DDoS attack, botnet activity, spyware, a man-in-the-middle attack, a password attack, or an SQL injection.
- Check if the attack was DNS tunneling.
- If true, it indicates DNS tunneling.

Pseudocode:

```
if attack_was_identified and not attack_was_ransomware and not attack_was_virus and not attack_was_trojan_horse and not attack_was_malware and not attack_was_ddos and not attack_was_botnet and not attack_was_spyware and not attack_was_man_in_middle and not attack_was_password_attack and not attack_was_sql_injection and attack_was_dns_tunneling:
    conclude("DNS_Tunneling")
```

Rule 13: Network Intrusion Detection

Description:

- Confirm an attack was identified.
- Ensure it was not ransomware, a virus, a trojan horse, malware, a DDoS attack, botnet activity, spyware, a man-in-the-middle attack, a password attack, an SQL injection, or DNS tunneling.
- Check if the attack was a network intrusion.
- If true, it indicates a network intrusion.

Pseudocode:

```
if attack_was_identified and not attack_was_ransomware and not attack_was_virus and not attack_was_trojan_horse and not attack_was_malware and not attack_was_ddos and not attack_was_botnet and not attack_was_spyware and not attack_was_man_in_middle and not attack_was_password_attack and not attack_was_sql_injection and not attack_was_dns_tunneling and attack_was_network_intrusion:
    conclude("Network_Intrusion")
```

Rule 14: Social Engineering Attack Detection

Description:

- Confirm an attack was identified.
- Ensure it was not ransomware, a virus, a trojan horse, malware, a DDoS attack, botnet activity, spyware, a man-in-the-middle attack, a password attack, an SQL injection, DNS tunneling, or network intrusion.
- Check if the attack was a social engineering attack.
- If true, it indicates a social engineering attack.

Pseudocode:

```
if attack_was_identified and not attack_was_ransomware and not
attack_was_virus and not attack_was_trojan_horse and not
attack_was_malware and not attack_was_ddos and not attack_was_botnet
and not attack_was_spyware and not attack_was_man_in_middle and not
attack_was_password_attack and not attack_was_sql_injection and not
attack_was_dns_tunneling and not attack_was_network_intrusion and
attack_was_social_engineering:
  conclude("Social_Engineering")
```

Rule 15: Denial of Service Attack Detection

Description:

- Confirm an attack was identified.
- Ensure it was not ransomware, a virus, a trojan horse, malware, a DDoS attack, botnet activity, spyware, a man-in-the-middle attack, a password attack, an SQL injection, DNS tunneling, network intrusion, or a social engineering attack.
- Check if the attack was a denial of service attack.
- If true, it indicates a denial of service attack.

Pseudocode:

```
if attack_was_identified and not attack_was_ransomware and not
attack_was_virus and not attack_was_trojan_horse and not
attack_was_malware and not attack_was_ddos and not attack_was_botnet
and not attack_was_spyware and not attack_was_man_in_middle and not
attack_was_password_attack and not attack_was_sql_injection and not
attack_was_dns_tunneling and not attack_was_network_intrusion and not
attack_was_social_engineering and attack_was_denial_of_service:
  conclude("Denial_of_Service")
```

Rule 16: Packet Analyzer Detection

Description:

- Confirm an attack was identified.
- Ensure it was not ransomware, a virus, a trojan horse, malware, a DDoS attack, botnet activity, spyware, a man-in-the-middle attack, a password attack, an SQL injection, DNS tunneling, network intrusion, a social engineering attack, or a denial of service attack.
- Check if the attack was related to a packet analyzer.
- If true, it indicates the use of a packet analyzer.

Pseudocode:

```
if attack_was_identified and not attack_was_ransomware and not
attack_was_virus and not attack_was_trojan_horse and not
attack_was_malware and not attack_was_ddos and not attack_was_botnet
and not attack_was_spyware and not attack_was_man_in_middle and not
attack_was_password_attack and not attack_was_sql_injection and not
attack_was_dns_tunneling and not attack_was_network_intrusion and not
attack_was_social_engineering and not attack_was_denial_of_service and
attack_was_packet_analyzer:
  conclude("Packet_Analyzer")
```

Rule 17: No Specific Attack Identified (Nil)

Description:

- Confirm an attack was identified.
- Ensure it was not ransomware, a virus, a trojan horse, malware, a DDoS attack, botnet activity, spyware, a man-in-the-middle attack, a password attack, an SQL injection, DNS tunneling, network intrusion, a social engineering attack, a denial of service attack, or a packet analyzer.
- If all these are false, it indicates no specific attack (Nil).

Pseudocode:

```
if attack_was_identified and not attack_was_ransomware and not
attack_was_virus and not attack_was_trojan_horse and not
attack_was_malware and not attack_was_ddos and not attack_was_botnet
and not attack_was_spyware and not attack_was_man_in_middle and not
attack_was_password_attack and not attack_was_sql_injection and not
attack_was_dns_tunneling and not attack_was_network_intrusion and not
attack_was_social_engineering and not attack_was_denial_of_service and
not attack_was_packet_analyzer:
  conclude("Nil")
```

5. Implementation

5.1 Program Initialization: Upon invoking the driver function for the backward chaining portion of the inference engine, there are multiple data structures that are initialized that will ultimately be tasked with storing key component knowledge of the system.

5.1.1 Knowledge Base Data: At the start of the program, data structures to hold the knowledge base are instantiated. We employ a struct to ensure efficient and consistent access to data, pivotal for the program's decision-making process. This structure facilitates the creation of a vector of clauseVariable struct objects, which provides direct access to all clauses and their associated rules. Additionally, a map

is used for the rules, enabling efficient identification of the relevant rule for a given clause number. A vector is also in place to maintain the list of conclusions.

5.1.2 Knowledge Base Parsing: Following the initialization of these data structures, the system employs a parser, defined in parser.cpp, to process the knowledge base text file. This parser populates the backward chaining data structures with essential information, such as rules and clauses, necessary for the engine's functionality.

5.2 Backward Chaining Process: The backward chaining process is a systematic approach, beginning with the extraction of conclusions from the parsed rules. A stack is used to manage these conclusions, and a set tracks the rules already evaluated. The engine iteratively examines each conclusion, searching for corresponding rules. Upon finding a matching rule, the engine calculates the necessary clause number and gathers relevant clauses for that rule. These clauses are then processed in reverse order to align with the backward chaining methodology. The engine solicits user input for uninstantiated clauses, marking them as instantiated post-response. It then determines the required and actual boolean values for each clause, representing the rule's conditions and the user's responses. A comparison of these boolean values identifies if the conditions of a rule are met. If so, the attack is diagnosed, setting the attack type and concluding the diagnosis. The process continues until all conclusions are evaluated, ensuring a thorough examination for potential network attacks. Once an attack is identified, a struct of type Attack is returned from the main function which contains an attack and if it was identified.

5.3 Forward Chaining Process: The forward chaining process in the inference engine starts once it receives the AttackResult struct, which contains the information about the attack type and its identification status. The process begins with the initialization of various data structures, including arrays for rules, variables, and conclusions, populated using the parsed content from the forward chaining knowledge base file. This setup forms the backbone of the engine's decision-making framework. The engine then adapts the attack type from the AttackResult into a format consistent with the variables in the knowledge base and acknowledges whether an attack was identified. It employs a function, Search_CVL, to locate the clause number associated with the given attack type, instantiating the corresponding variable in the process. This clause number is then translated into a rule number through a simple division, linking the attack type to a specific rule in the knowledge base.

Subsequently, the engine employs the validate_Ri function, which checks the conditions of the identified rule against the instantiated variables. This determines whether the conditions stipulated by the rule are met. If they are, the function fetches the index of the relevant conclusion from the Conclusions array, representing the final inference about the

attack type. Finally, the engine concludes the forward chaining process by returning this conclusion, which provides a detailed description of the inferred attack type.

6. Source Code

6.1 File Structure (Implementation Simplified):

1. Main Execution (main.cpp): Initiates the program, orchestrating the backward and forward chaining processes.
2. Backward Chaining (BackwardChaining.cpp): Invoked by `main.cpp`, it utilizes functions from `parsefile.cpp` to process the knowledge base stored in `kbbc.txt`.
3. Forward Chaining (ForwardChaining.cpp): After backward chaining completion, `main.cpp` inputs the identified attack into this module, which uses `kbfc.txt` for its knowledge base.
4. Header Files (.h): Used throughout for readability and modularity, defining the structure of classes and functions.

6.2 main.h:

```
#ifndef CS4346_PROJECT_1_MAIN_H
#define CS4346_PROJECT_1_MAIN_H

#include "backwardChaining.h"
#include "forwardChaining.h"

#endif //CS4346_PROJECT_1_MAIN_H
```

6.3 main.cpp:

```
#include <iostream>
#include "main.h"
#include "timing.h"



int main() {
    // starts the backward chaining algo
    start_total_timer(); // Start the total program timer
    resume_timing();
    AttackResult result = Attacks_BW();
    if(result.attackWasIdentified){
        std::cout << "Attack identified is: " << result.typeOfAttack << std::endl;
    } else {
        std::cout << "No attack found." << std::endl;
    }

    Conclusions finalConclusion = forwardChaining(result);
```

```
    // Output the conclusion and description
    std::cout << "Conclusion: " << finalConclusion.conclusion << std::endl;
    std::cout << "Description: " << finalConclusion.description << std::endl;
    pause_timing();
    end_total_timer(); // End the total program timer
    print_execution_time(); // Print out both execution times

    return 0;
}
```

6.4 data.h:

```
#ifndef DATA_H
#define DATA_H
#include <string>
#include <vector>
#include <iostream>
#include <optional>
struct ClauseVariable
{
    std::string name;
    std::vector<int> clauseNumbers;
    std::string meaning;
    bool instantiated = false;
    std::optional<bool> response;
};
struct Rule
{
    int ruleNumber;
    std::string conclusion;
};
// values initially assume no attack was found
struct AttackResult
{
    std::string typeOfAttack = "None";
    bool attackWasIdentified = false;
};
// Structure for the rules
struct Rules {
    std::vector<std::string> conditions;
    std::vector<int> clauseNum;
    std::string conclusion;
};
// Structure for the variables
struct Variables {
    std::string variable;
```

```
    std::string type;
    std::string instantiated;
    int clauseNum;
};
// Structure for the conclusions
struct Conclusions {
    std::string conclusion;
    std::string description;
};
#endif
```

6.5 parser.h:

```
#ifndef PARSER_H
#define PARSER_H
#include <fstream>
#include <sstream>
#include <iostream>
#include <stdexcept>
#include <string>
#include <vector>
#include <unordered_map>
#include <optional>
#include <algorithm>
#include <map>
#include "data.h"
void parseFile(const std::string &filename, std::map<int, std::string> &rules,
std::vector<ClauseVariable> &clauses);
std::string trim(const std::string &str);
std::string removeCommas(const std::string &str);
#endif
```

6.6 parser.cpp:

```
#include "parser.h"
// Trims whitespace from both ends of a given string
std::string trim(const std::string &str) {
    size_t first = str.find_first_not_of(' ');
    // Return original string if no non-space character is found
    if (std::string::npos == first) {
        return str;
    }
    size_t last = str.find_last_not_of(' ');
    // Extracts a substring from 'first' to 'last', inclusive
    return str.substr(first, (last - first + 1));
}

// Removes all commas from a given string
std::string removeCommas(const std::string &str) {
```

```
    std::string result;
    // Reserve space upfront to avoid multiple reallocations
    result.reserve(str.size());
    for (char ch : str) {
        if (ch != ',') {
            result += ch; // Append character if it's not a comma
        }
    }
    return result;
}
// Parses a file to extract rules and clause variables
void parseFile(const std::string &filename, std::map<int, std::string> &rules,
std::vector<ClauseVariable> &clauses) {
    std::ifstream file(filename);
    std::string line;
    int lastRuleNumber = -1;  // To keep track of the last rule number
    while (std::getline(file, line)) {
        if (line.find("Rule ") != std::string::npos) {
            std::istringstream iss(line);
            std::string tmp;
            iss >> tmp >> tmp; // Extracting and skipping the "Rule" part, getting
the rule number
            try {
                int ruleNumber = std::stoi(tmp);
                rules[ruleNumber] = ""; // Adding a new rule with the extracted
number
                lastRuleNumber = ruleNumber; // Updating last rule number
            } catch (const std::invalid_argument &ia) {
                std::cerr << "Invalid argument: " << ia.what() << '\n';
                continue;
            }
        } else if (line.find("Clause: ") != std::string::npos) {
            size_t clausePos = line.find("ClauseNumber: ");
            size_t meaningPos = line.find(", clauseMeaning: ");
            std::string clauseName = removeCommas(trim(line.substr(8, clausePos -
9)));
            std::string clauseNumStr = trim(line.substr(clausePos + 14, meaningPos
- (clausePos + 14)));
            std::string clauseMeaning = "";
            if (meaningPos != std::string::npos) {
                clauseMeaning = trim(line.substr(meaningPos + 17));
            }
            try {
                int clauseNumber = std::stoi(clauseNumStr);
                auto foundClause = std::find_if(clauses.begin(), clauses.end(),
                                                [&](const ClauseVariable &c) {
return c.name == clauseName; });
```

```
                if (foundClause != clauses.end()) {
                    // Clause already exists, so add this clause number to its
vector
                    foundClause->clauseNumbers.push_back(clauseNumber);
                } else {
                    // New clause: determine if it's negated and whether it's
instantiated
                    bool isNegated = !clauseName.empty() && clauseName[0] == '!';
                    bool instantiated = isNegated;
                    std::optional<bool> response = isNegated ?
std::optional<bool>(false) : std::nullopt;

                    clauses.push_back({clauseName, {clauseNumber}, clauseMeaning,
instantiated, response});
                }
            } catch (const std::invalid_argument &ia) {
                std::cerr << "Invalid argument: " << ia.what() << '\n';
                continue;
            }
        } else if (line.find("Conclusion: ") != std::string::npos) {
            // Sets the conclusion of the last rule in the vector
            if (!rules.empty()) {
                rules[lastRuleNumber] = line.substr(12); // Sets the conclusion of
the last rule
            }
        }
    }
}
```

6.7 backwardChaining.h:

```
#ifndef BACKWARDCHAINING_H
#define BACKWARDCHAINING_H
#include <string>
#include <vector>
#include <iostream>
#include <fstream>
#include <sstream>
#include <algorithm>
#include <cctype>
#include <limits>
#include <stack>
#include <set>
#include <map>
#include <iomanip>
#include <chrono>
#include <ctime>
```

```
#include "data.h"
#include "parser.h"
// Prototypes for output functions
void testingOutput(const std::string text);
std::string getCurrentTime();
void printRules(const std::map<int, std::string>& rules);
void printClauses(std::vector<std::reference_wrapper<ClauseVariable>> &clauses);
void printBoolComparison(const std::vector<bool>& requiredClauseBools, const
std::vector<bool>& actualClauseBools);
void printConclusions(std::vector<std::string> &conclusions);
void getUserInput(ClauseVariable &tempClause);
std::vector<bool>
determineRealClauseBools(std::vector<std::reference_wrapper<ClauseVariable>>
&currentRuleClauses);
std::vector<bool>
determineRequiredClauseBools(std::vector<std::reference_wrapper<ClauseVariable>>
&currentRuleClauses);
bool compareClauseBools(std::vector<bool> &requiredBools, std::vector<bool>
&actualBools);
void printClauseNumbers(const std::vector<int> &clauseNumbers);
// Prototypes for backward chaining algorithm functions
AttackResult Attacks_BW();
std::vector<std::string> getConclusions(const std::map<int, std::string>& rules);
std::string trim(std::string &str);
int calcClauseNumber(int ruleNumber);
#endif // BACKWARDCHAININGFUNCTIONS_H
```

6.8 backwardChaining.cpp:

```
#include "backwardChaining.h"
#include "timing.h"

// test output functions
std::string getCurrentTime() {
    auto now = std::chrono::system_clock::now();
    std::time_t currentTime = std::chrono::system_clock::to_time_t(now);
    std::tm* ptm = std::localtime(&currentTime);
    char buffer[32];
    // Format: YYYY-MM-DD HH:mm:ss
    std::strftime(buffer, 32, "%Y-%m-%d %H:%M:%S", ptm);
    return std::string(buffer);
    // this is a test
}
void testingOutput(const std::string text) {
    // The symbols for the border can be changed to anything you like
    const std::string border(60, '*');
    const std::string timeStr = getCurrentTime();
```

```
    std::cout << "\n" << border << std::endl;
    std::cout << "* [" << timeStr << "] - " << text << std::endl;
    std::cout << border << std::endl;
}
void printClauses(std::vector<std::reference_wrapper<ClauseVariable>>& clauses) {
    std::cout << "----------------- Clauses -----------------\n";
    for (const auto& clauseRef : clauses) {
        const ClauseVariable& clause = clauseRef.get();
        std::cout << "Clause Name: " << clause.name << "\n"
                  << "Clause Numbers: ";
        for (auto num : clause.clauseNumbers) {
            std::cout << num << " ";
        }
        std::cout << "\n-----------------------------------------\n";
    }
}
void printConclusions(const std::vector<std::string>& conclusions) {
    std::cout << "--------------- Conclusions ---------------\n";
    for (const auto& conclusion : conclusions) {
        std::cout << "- " << conclusion << "\n";
    }
    std::cout << "-----------------------------------------\n";
}
void printClauseNumbers(const std::vector<int>& clauseNumbers) {
    std::cout << "----------- Clause Numbers -----------\n";
    for (const auto& num : clauseNumbers) {
        std::cout << num << " ";
    }
    std::cout << "\n--------------------------------------\n";
}
void printRules(const std::map<int, std::string>& rules) {
    std::cout << "----------------- Rules -----------------\n";
    for (const auto& rulePair : rules) {
        std::cout << "Rule Number: " << std::left << std::setw(4) << rulePair.first
                  << "Conclusion: " << rulePair.second << "\n";
    }
    std::cout << "---------------------------------------\n";
}
void printBoolComparison(const std::vector<bool>& requiredClauseBools, const
std::vector<bool>& actualClauseBools) {
    if (requiredClauseBools.size() != actualClauseBools.size()) {
        std::cerr << "Error: Vectors are not the same size.\n";
        return;
    }

    std::cout << "------- Clause Boolean Comparison -------\n"
              << std::left << std::setw(10) << "Index"
```

```
              << std::setw(20) << "Required Clause"
              << "Actual Clause\n"
              << "----------------------------------------\n";

    for (size_t i = 0; i < requiredClauseBools.size(); ++i) {
        std::cout << std::left << std::setw(10) << i
                  << std::setw(20) << (requiredClauseBools[i] ? "True" : "False")
                  << (actualClauseBools[i] ? "True" : "False") << "\n";
    }
    std::cout << "----------------------------------------\n";
}

// main
AttackResult Attacks_BW(){ // Parses knowledge base and extracts the data, keeping
it seperate from inference engine
    std::map<int, std::string> rules; // hashing
    std::vector<ClauseVariable> clauses;
    std::vector<std::string> conclusions;
    parseFile("kbbc.txt", rules, clauses);
    conclusions = getConclusions(rules); // this contains all conclusion variables
used for the engine
    AttackResult attack;
    std::stack<std::string> conclusionStack;
    std::set<int> checkedRules;
    // loop that will go through all conclusions (goal) until one is found for the
user (type of attack)
    testingOutput("Beginning Backward Chaining procedure...");
    for (auto &conclusion : conclusions){
        conclusionStack.push(conclusion);
        // parse through rules until it finds a rule with the same conclusion as
the conclusion we're using as goal
        for (const auto &rulePair : rules) { // iterates through the rules hash map
            if (checkedRules.find(rulePair.first) != checkedRules.end())
{continue;} // Rule already checked
            if (trim(rulePair.second) == trim(conclusion)){ // rule we want that
has the conclusion were using for the goal
                testingOutput("Current Goal: " + conclusion + ", located at rule
number: " + std::to_string(rulePair.first));
                int clauseNumber = calcClauseNumber(rulePair.first);
                std::vector<std::reference_wrapper<ClauseVariable>>
currentRuleClauses;
                bool clauseFound = false;
                for (auto &clause : clauses){ // Find clauses for the current rule
                    // Check if the clause contains the current clauseNumber
                    if (std::find(clause.clauseNumbers.begin(),
clause.clauseNumbers.end(), clauseNumber) != clause.clauseNumbers.end()){
                        currentRuleClauses.push_back(clause); // Add reference to
```

```
the clause
                        clauseFound = true;
                    }
                    else{if (clauseFound) {break;}}
                    if (clauseFound){ // Increment clauseNumber if a matching
clause was found
                        clauseNumber++;
                        clauseFound = false; // Reset clauseFound for the next
iteration
                    }
                }
                std::reverse(currentRuleClauses.begin(), currentRuleClauses.end());
                // Skip to the next rule as no clauses were found for the current
rule
                if (currentRuleClauses.empty()) {continue;}
                testingOutput("Clauses for rule " +
std::to_string(rulePair.first));
                printClauses(currentRuleClauses);
                // iterating through the currentRuleClauses to instantiate and
determine clause responses(yes this is the attack/no not this one) based on user
input
                for (auto &clauseRef : currentRuleClauses){
                    ClauseVariable &clause = clauseRef.get(); // Get the original
clause reference
                    bool clauseIsMet = false;
                    if (!clause.instantiated){
                        pause_timing();
                        getUserInput(clause);
                        resume_timing();
                        clause.instantiated = true;
                    }
                    else {continue;} // if the clause is instaniated, go to next
clause
                }
                std::vector<bool> requiredClauseBools =
determineRequiredClauseBools(currentRuleClauses); // bools required for conclusion
of rule to be used
                std::vector<bool> actualClauseBools =
determineRealClauseBools(currentRuleClauses); // bools of users responses
(yes=t,no=f)
                // compare the bools to see if what the user gave matches what is
needed for the rule to provide the conclusion, if it doesn't keep going through the
rules until it's found
                testingOutput("Required bools vs. Users answers for conclusion to
be met:");
                printBoolComparison(requiredClauseBools, actualClauseBools);
                if (compareClauseBools(requiredClauseBools, actualClauseBools)){ //
```

```
attack is identified, return attack
                    attack.attackWasIdentified = true;
                    attack.typeOfAttack = rulePair.second;
                    return attack;
                }
                checkedRules.insert(rulePair.first);
            } // after loop, system is out of clauses for that rule.
        } conclusionStack.pop();
    }
    return attack;
}
std::vector<std::string> getConclusions(const std::map<int, std::string>& rules) {
    std::vector<std::string> conclusions;
    conclusions.reserve(rules.size()); // Reserving memory for known map size
    for (const auto& rulePair : rules) {
        conclusions.push_back(rulePair.second); // Adding the conclusion from each
rule
    }
    return conclusions;
}
std::string trim(std::string& str) {
    auto start = std::find_if_not(str.begin(), str.end(), ::isspace);
    auto end = std::find_if_not(str.rbegin(), str.rend(), ::isspace).base();
    return (start < end) ? std::string(start, end) : "";
}
int calcClauseNumber(int ruleNumber){
    return static_cast<int>((static_cast<double>(ruleNumber) / 10.0 - 1) * 4) + 1;
}
void getUserInput(ClauseVariable &tempClause){
    std::string userResponse;
    while (true){
        std::cout << tempClause.meaning << "\n: ";
        getline(std::cin, userResponse);
        for (auto &c : userResponse) {c = std::tolower(c);}
        if (userResponse == "yes"){
            tempClause.response = true;
            break;
        }
        else if (userResponse == "no"){
            tempClause.response = false;
            break;
        }
        else {std::cout << "Invalid response. Please answer with 'yes' or
'no'.\n";}
    }
}
std::vector<bool>
```

```
determineRequiredClauseBools(std::vector<std::reference_wrapper<ClauseVariable>>
&currentRuleClauses){
    std::vector<bool> requiredBools;
    for (auto &clauseRef : currentRuleClauses){
        ClauseVariable &clause = clauseRef.get(); // Get the actual ClauseVariable
        requiredBools.push_back(clause.name[0] != '!');
    }
    return requiredBools;
}
std::vector<bool>
determineRealClauseBools(std::vector<std::reference_wrapper<ClauseVariable>>
&currentRuleClauses){
    std::vector<bool> realBools;
    for (auto &clauseRef : currentRuleClauses){
        ClauseVariable &clause = clauseRef.get(); // Get the actual ClauseVariable
        realBools.push_back(clause.response.value_or(false));
    }
    return realBools;
}
bool compareClauseBools(std::vector<bool> &requiredClauseBools, std::vector<bool>
&actualClauseBools){
    if (requiredClauseBools.size() != actualClauseBools.size()) {return false;}
    return std::equal(requiredClauseBools.begin(), requiredClauseBools.end(),
actualClauseBools.begin());
}
```

6.9 forwardChaining.h:

```
#ifndef FORWARD_CHAINING_
#define FORWARD_CHAINING_

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <sstream>
#include "data.h"

// Function prototypes
void initializeArr(std::string array[]);
void populateRules(Rules rule[], std::string array[]);
void populateVariables(Variables var[], std::string array[]);
void populateConclusions(Conclusions con[], std::string array[]);
int Search_CVL(Variables var[], std::string type);
int Clause_To_Rule(int clauseNum);
void identifyCheck(std::string identified, Variables var[]);
int validate_Ri(int ruleNum, Rules rules[], Variables var[], Conclusions
conclusion[]);
```

```
Conclusions forwardChaining(AttackResult);

#endif // FORWARD_CHAINING_
```

6.10 forwardChaining.cpp:

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <sstream>
#include "forwardChaining.h"

//puts all info into an array organized by chunk
void initializeArr(std::string array[]){
    std::ifstream inputFile("kbfc.txt"); // Open file
    if (!inputFile) {
        std::cout << "Error opening file." << std::endl;
        return;
    }
    std::string line;
    std::string paragraph;
    int counter = 0;

    while(std::getline(inputFile, line)){
        if(!line.empty()){
            paragraph += line + "\n";
        }
        else{
            array[counter] = paragraph;
            paragraph = "";
            counter++;
        }
    }

    // Add the last paragraph if it exists
    if (!paragraph.empty()) {
        array[counter] = paragraph;
    }

    inputFile.close();
}
//populated the rules structure array with each rule
void populateRules(Rules rule[], std::string array[]){
  for(int i = 0; i < 17; i++){
    std::string paragraph = array[i];
    // Parse the paragraph
    std::istringstream iss(paragraph);
```

```
    std::string line;

    while (std::getline(iss, line)) {
      if(line.find("Condition:") != std::string::npos){
        int pos1 = line.find("Condition: ") + 11;
        int length = line.find(',') - pos1;
        std::string condition = line.substr(pos1, length);
        rule[i].conditions.push_back(condition);
        if(i != 16){
          pos1 = line.find("Clause: ") + 8;
          length = line.find(',') - pos1;
          std::string clause = line.substr(pos1, length);
          int num = std::stoi(clause);
          rule[i].clauseNum.push_back(num);
        }
        }else if(line.find("Conclusion: ") != std::string::npos){
          int pos1 = line.find("Conclusion: ") + 12;
          int length = line.length() - pos1;
          std::string conclusion = line.substr(pos1, length);
          rule[i].conclusion = conclusion;
        }
    }
  }
}

//populates the variables structure array with each variable
void populateVariables(Variables var[], std::string array[]){
  //counter for var[] array
  int counter = 0;
  //loops through array starting at the first variable chunk to last
  for(int i = 16; i < 32; i++){
    std::string paragraph = array[i];
    // Parse the paragraph
    std::istringstream iss(paragraph);
    std::string line;
    while (std::getline(iss, line)) {
      if(line.find("Variable:") != std::string::npos){
        int pos1 = line.find("Variable: ") + 10;
        int length = line.length() - pos1;
        std::string variable = line.substr(pos1, length);
        var[counter].variable = variable;
      }else if(line.find("Type:") != std::string::npos){
        int pos1 = line.find("Type: ") + 6;
        int length = line.length() - pos1;
        std::string type = line.substr(pos1, length);
        var[counter].type = type;
      }else if(line.find("Instantiated:") != std::string::npos){
```

```
        int pos1 = line.find("Instantiated: ") + 13;
        int length = line.length() - pos1;
        std::string instantiated = line.substr(pos1, length);
        var[counter].instantiated = instantiated;
      }else if(line.find("Clause Number:") != std::string::npos){
        int pos1 = line.find("Clause Number: ") + 15;
        int length = line.length() - pos1;
        std::string clause = line.substr(pos1, length);
        int num = std::stoi(clause);
        var[counter].clauseNum = num;
      }
    }
    counter++;
  }
}

void populateConclusions(Conclusions con[], std::string array[]){
    int counter = 0;

    for(int i = 32; i < 49; i++){  // Assuming 17 rules in total
        std::string paragraph = array[i];
        std::istringstream iss(paragraph);
        std::string line;
        bool foundConclusion = false;

        // Reset description for each new conclusion
        con[counter].description = "";

        while (std::getline(iss, line)){
            if(line.find("Conclusion:") != std::string::npos){
                int pos1 = line.find("Conclusion: ") + 12;
                con[counter].conclusion = line.substr(pos1);
                foundConclusion = true;
            } else if (foundConclusion) {
                // Append to description only after finding the conclusion
                con[counter].description += line + "\n";
            }
        }

        if (!foundConclusion) {
            // Handle case where no conclusion is found in the paragraph
            con[counter].conclusion = "No conclusion found";
        }

        counter++;
    }
    /*
```

```
    // Optional: Print all conclusions and descriptions for debugging
    for (int j = 0; j < counter; j++) {
        std::cout << "Conclusion: " << con[j].conclusion << std::endl;
        std::cout << "Description: " << con[j].description << std::endl;
    }*/
}


//searches for the clause number, returns clause number or -1 if not found
int Search_CVL(Variables var[], std::string type){
  //std::cout << type << std::endl;
  for(int i = 0; i < 17; i++){
    // std::cout << "type: " + type + ", var[i]: " + var[i].variable << std::endl;
    if(type == var[i].variable){
      var[i].instantiated = "true";
      //std::cout << "variable found" << std::endl;
      return var[i].clauseNum;
    }
  }

  //if goes past loop, variable not found, nil clause = 4
  return 4;
}

int Clause_To_Rule(int clauseNum){
  int ruleNum = clauseNum / 4;

  return ruleNum;
}

void identifyCheck(bool identified, Variables var[]){
  if(identified){
    var[0].instantiated = "true"; // attack_was_identified is true if identified is
passed in
  }
}

int validate_Ri(int ruleNum, Rules rules[], Variables var[], Conclusions
conclusion[], AttackResult result){
  //denial o f service, NIL

  int i = ruleNum - 1;
  int size = rules[i].conditions.size();
   //std::cout << rules[i].conditions.size() << std::endl;
  int validated = true;
    for(int i = 0; i < 17; i++){
      //std::cout << conclusion[i].conclusion << " " << result.typeOfAttack <<
```

```
std::endl;
      if( conclusion[i].conclusion  == result.typeOfAttack){
        //std::cout << conclusion[i].conclusion << std::endl;
        return i;
      }
    }
  return -1;
}

Conclusions forwardChaining(AttackResult result){
  //initialize variables for testing
  std::string type = "attack_was_" + result.typeOfAttack;
  bool identified = result.attackWasIdentified;
  std::string array[48];
  initializeArr(array);

  Rules rules[17];
  populateRules(rules, array);

  Variables variables[16];
  populateVariables(variables, array);

  Conclusions conclusion[16];
  populateConclusions(conclusion, array);

  int Ci = Search_CVL(variables, type);

  int Ri = Clause_To_Rule(Ci);

  //check if attack was identified
  identifyCheck(identified, variables);

  //returns the index of the conclution
  int conclusionIndex = validate_Ri(Ri, rules, variables, conclusion, result);

  if(conclusionIndex == -1){
    std::cout << "couldn't find conclusion index in validate_Ri" << std::endl;
  }

  return conclusion[conclusionIndex];
}
```

6.11 kbbc.txt:

```
Rules:
Rule 15
Clause: unauthorized_access_or_control, ClauseNumber: 3, clauseMeaning: Is there
```

```
unauthorized access or control?
Clause: file_changes_or_configurations, ClauseNumber: 4, clauseMeaning: Are there
changes to files or configurations?
Clause: files_corrupted_without_ransom, ClauseNumber: 5, clauseMeaning: Are there
files corrupted without a request for ransom?
Conclusion: ransomware

Rule 30
Clause: unauthorized_access_or_control, ClauseNumber: 9
Clause: file_changes_or_configurations, ClauseNumber: 10
Clause: !files_corrupted_without_ransom, ClauseNumber: 11
Conclusion: virus

Rule 45
Clause: unauthorized_access_or_control, ClauseNumber: 15
Clause: !file_changes_or_configurations, ClauseNumber: 16
Clause: backdoor_accounts_or_tools_found, ClauseNumber: 17, clauseMeaning: Are
there backdoor accounts or tools found on the system?
Clause: evidence_of_data_exfiltration, ClauseNumber: 18, clauseMeaning: Is there
evidence of data exfiltration?
Conclusion: trojan_horse

Rule 60
Clause: unauthorized_access_or_control, ClauseNumber: 21
Clause: !file_changes_or_configurations, ClauseNumber: 22
Clause: backdoor_accounts_or_tools_found, ClauseNumber: 23
Clause: !evidence_of_data_exfiltration, ClauseNumber: 24
Conclusion: malware

Rule 75
Clause: unauthorized_access_or_control, ClauseNumber: 27
Clause: !file_changes_or_configurations, ClauseNumber: 28
Clause: !backdoor_accounts_or_tools_found, ClauseNumber: 29
Clause: high_volume_of_outgoing_traffic, ClauseNumber: 30, clauseMeaning: Is there
a high volume of outgoing traffic?
Clause: traffic_targeting_multiple_systems, ClauseNumber: 31, clauseMeaning: Is the
traffic targeting multiple systems?
Conclusion: ddos_attack

Rule 90
Clause: unauthorized_access_or_control, ClauseNumber: 33
Clause: !file_changes_or_configurations, ClauseNumber: 34
Clause: !backdoor_accounts_or_tools_found, ClauseNumber: 35
Clause: high_volume_of_outgoing_traffic, ClauseNumber: 36
Clause: !traffic_targeting_multiple_systems, ClauseNumber: 37
Conclusion: botnet_activity
```

```
Rule 105
Clause: unauthorized_access_or_control, ClauseNumber: 39
Clause: !file_changes_or_configurations, ClauseNumber: 40
Clause: !backdoor_accounts_or_tools_found, ClauseNumber: 41
Clause: !high_volume_of_outgoing_traffic, ClauseNumber: 42
Clause: unsual_system_behavior_without_high_traffic, ClauseNumber: 43,
clauseMeaning: Is there unusual system behavior without a high volume of traffic?
Conclusion: spyware

Rule 120
Clause: unauthorized_access_or_control, ClauseNumber: 45
Clause: !file_changes_or_configurations, ClauseNumber: 46
Clause: !backdoor_accounts_or_tools_found, ClauseNumber: 47
Clause: !high_volume_of_outgoing_traffic, ClauseNumber: 48
Clause: !unsual_system_behavior_without_high_traffic, ClauseNumber: 49
Conclusion: packet_analyzer

Rule 135
Clause: !unauthorized_access_or_control, ClauseNumber: 51
Clause: unusual_network_activity, ClauseNumber: 52, clauseMeaning: Is there unusual
network activity?
Clause: network_activity_external, ClauseNumber: 53, clauseMeaning: Is the network
activity external?
Clause: unexpected_login_attempts, ClauseNumber: 54, clauseMeaning: Are there
unexpected login attempts?
Conclusion: password_attack

Rule 150
Clause: !unauthorized_access_or_control, ClauseNumber: 57
Clause: unusual_network_activity, ClauseNumber: 58
Clause: network_activity_external, ClauseNumber: 59
Clause: !unexpected_login_attempts, ClauseNumber: 60
Conclusion: man_in_the_middle_attack

Rule 165
Clause: !unauthorized_access_or_control, ClauseNumber: 63
Clause: unusual_network_activity, ClauseNumber: 64
Clause: !network_activity_external, ClauseNumber: 65
Clause: unusual_activity_related_to_data_flow, ClauseNumber: 66, clauseMeaning: Is
the unusual activity related to data flow?
Clause: increase_in_dns_requests, ClauseNumber: 67, clauseMeaning: Is there an
increase in DNS requests?
Conclusion: dns_tunneling

Rule 180
Clause: !unauthorized_access_or_control, ClauseNumber: 69
Clause: unusual_network_activity, ClauseNumber: 70
```

```
Clause: !network_activity_external, ClauseNumber: 71
Clause: unusual_activity_related_to_data_flow, ClauseNumber: 72, clauseMeaning: Is
the unusual activity related to data flow?
Clause: !increase_in_dns_requests, ClauseNumber: 73, clauseMeaning: Is there an
increase in DNS requests?
Conclusion: sql_injection

Rule 195
Clause: !unauthorized_access_or_control, ClauseNumber: 75
Clause: unusual_network_activity, ClauseNumber: 76
Clause: !network_activity_external, ClauseNumber: 77
Clause: !unusual_activity_related_to_data_flow, ClauseNumber: 78
Clause: unauthorized_access_to_network_or_data, ClauseNumber: 79, clauseMeaning: Is
the unauthorized access to the data or network?
Conclusion: network_intrusion

Rule 210
Clause: !unauthorized_access_or_control, ClauseNumber: 81
Clause: unusual_network_activity, ClauseNumber: 82
Clause: !network_activity_external, ClauseNumber: 83
Clause: !unusual_activity_related_to_data_flow, ClauseNumber: 84
Clause: !unauthorized_access_to_network_or_data, ClauseNumber: 85
Conclusion: social_engineering_attack

Rule 225
Clause: !unauthorized_access_or_control, ClauseNumber: 87
Clause: !unusual_network_activity, ClauseNumber: 88
Clause: slow_network_or_system_performance_without_external_activity, ClauseNumber:
89, clauseMeaning: Is there a slow network or system performance without external
activity?
Conclusion: denial_of_service_attack

Rule 240
Clause: !unauthorized_access_or_control, ClauseNumber: 93
Clause: !unusual_network_activity, ClauseNumber: 94
Clause: !slow_network_or_system_performance_without_external_activity,
ClauseNumber: 95
Conclusion: nil
```

6.12 kbfc.txt:

```
Rule 1
Condition: !attack_was_identified, Clause: 4
```

```
Conclusion: nil

Rule 2
Condition: attack_was_identified, Clause: 4
Condition: attack_was_ransomware, Clause: 8
Conclusion: ransomware

Rule 3
Condition: attack_was_identified, Clause: 4
Condition: !attack_was_ransomware, Clause: 8
Condition: attack_was_virus, Clause: 12
Conclusion: virus

Rule 4
Condition: attack_was_identified, Clause: 4
Condition: !attack_was_ransomware, Clause: 8
Condition: !attack_was_virus, Clause: 12
Condition: attack_was_trojan_horse, Clause: 16
Conclusion: trojan_horse

Rule 5
Condition: attack_was_identified, Clause: 4
Condition: !attack_was_ransomware, Clause: 8
Condition: !attack_was_virus, Clause: 12
Condition: !attack_was_trojan_horse, Clause: 16
Condition: attack_was_malware, Clause: 20
Conclusion: malware

Rule 6
Condition: attack_was_identified, Clause: 4
Condition: !attack_was_ransomware, Clause: 8
Condition: !attack_was_virus, Clause: 12
Condition: !attack_was_trojan_horse, Clause: 16
Condition: !attack_was_malware, Clause: 20
Condition: attack_was_ddos, Clause: 24
Conclusion: ddos

Rule 7
Condition: attack_was_identified, Clause: 4
Condition: !attack_was_ransomware, Clause: 8
Condition: !attack_was_virus, Clause: 12
Condition: !attack_was_trojan_horse, Clause: 16
Condition: !attack_was_malware, Clause: 20
Condition: !attack_was_ddos, Clause: 24
Condition: attack_was_botnet, Clause: 28
Conclusion: botnet
```

```
Rule 8
Condition: attack_was_identified, Clause: 4
Condition: !attack_was_ransomware, Clause: 8
Condition: !attack_was_virus, Clause: 12
Condition: !attack_was_trojan_horse, Clause: 16
Condition: !attack_was_malware, Clause: 20
Condition: !attack_was_ddos, Clause: 24
Condition: !attack_was_botnet, Clause: 28
Condition: attack_was_spyware, Clause: 32
Conclusion: spyware

Rule 9
Condition: attack_was_identified, Clause: 4
Condition: !attack_was_ransomware, Clause: 8
Condition: !attack_was_virus, Clause: 12
Condition: !attack_was_trojan_horse, Clause: 16
Condition: !attack_was_malware, Clause: 20
Condition: !attack_was_ddos, Clause: 24
Condition: !attack_was_botnet, Clause: 28
Condition: !attack_was_spyware, Clause: 32
Condition: attack_was_man_in_middle, Clause: 36
Conclusion: man_in_the_middle

Rule 10
Condition: attack_was_identified, Clause: 4
Condition: !attack_was_ransomware, Clause: 8
Condition: !attack_was_virus, Clause: 12
Condition: !attack_was_trojan_horse, Clause: 16
Condition: !attack_was_malware, Clause: 20
Condition: !attack_was_ddos, Clause: 24
Condition: !attack_was_botnet, Clause: 28
Condition: !attack_was_spyware, Clause: 32
Condition: !attack_was_man_in_middle, Clause: 36
Condition: attack_was_password_attack, Clause: 40
Conclusion: password_attack

Rule 11
Condition: attack_was_identified, Clause: 4
Condition: !attack_was_ransomware, Clause: 8
Condition: !attack_was_virus, Clause: 12
Condition: !attack_was_trojan_horse, Clause: 16
Condition: !attack_was_malware, Clause: 20
Condition: !attack_was_ddos, Clause: 24
Condition: !attack_was_botnet, Clause: 28
Condition: !attack_was_spyware, Clause: 32
Condition: !attack_was_man_in_middle, Clause: 36
Condition: !attack_was_password_attack, Clause: 40
```

```
Condition: attack_was_sql_injection, Clause: 44
Conclusion: sql_injection

Rule 12
Condition: attack_was_identified, Clause: 4
Condition: !attack_was_ransomware, Clause: 8
Condition: !attack_was_virus, Clause: 12
Condition: !attack_was_trojan_horse, Clause: 16
Condition: !attack_was_malware, Clause: 20
Condition: !attack_was_ddos, Clause: 24
Condition: !attack_was_botnet, Clause: 28
Condition: !attack_was_spyware, Clause: 32
Condition: !attack_was_man_in_middle, Clause: 36
Condition: !attack_was_password_attack, Clause: 40
Condition: !attack_was_sql_injection, Clause: 44
Condition: attack_was_dns_tunneling, Clause: 48
Conclusion: dns_tunneling

Rule 13
Condition: attack_was_identified, Clause: 4
Condition: !attack_was_ransomware, Clause: 8
Condition: !attack_was_virus, Clause: 12
Condition: !attack_was_trojan_horse, Clause: 16
Condition: !attack_was_malware, Clause: 20
Condition: !attack_was_ddos, Clause: 24
Condition: !attack_was_botnet, Clause: 28
Condition: !attack_was_spyware, Clause: 32
Condition: !attack_was_man_in_middle, Clause: 36
Condition: !attack_was_password_attack, Clause: 40
Condition: !attack_was_sql_injection, Clause: 44
Condition: !attack_was_dns_tunneling, Clause: 48
Condition: attack_was_network_intrusion, Clause: 52
Conclusion: network_intrusion

Rule 14
Condition: attack_was_identified, Clause: 4
Condition: !attack_was_ransomware, Clause: 8
Condition: !attack_was_virus, Clause: 12
Condition: !attack_was_trojan_horse, Clause: 16
Condition: !attack_was_malware, Clause: 20
Condition: !attack_was_ddos, Clause: 24
Condition: !attack_was_botnet, Clause: 28
Condition: !attack_was_spyware, Clause: 32
Condition: !attack_was_man_in_middle, Clause: 36
Condition: !attack_was_password_attack, Clause: 40
Condition: !attack_was_sql_injection, Clause: 44
Condition: !attack_was_dns_tunneling, Clause: 48
```

```
Condition: !attack_was_network_intrusion, Clause: 52
Condition: attack_was_social_engineering, Clause: 56
Conclusion: social_engineering

Rule 15
Condition: attack_was_identified, Clause: 4
Condition: !attack_was_ransomware, Clause: 8
Condition: !attack_was_virus, Clause: 12
Condition: !attack_was_trojan_horse, Clause: 16
Condition: !attack_was_malware, Clause: 20
Condition: !attack_was_ddos, Clause: 24
Condition: !attack_was_botnet, Clause: 28
Condition: !attack_was_spyware, Clause: 32
Condition: !attack_was_man_in_middle, Clause: 36
Condition: !attack_was_password_attack, Clause: 40
Condition: !attack_was_sql_injection, Clause: 44
Condition: !attack_was_dns_tunneling, Clause: 48
Condition: !attack_was_network_intrusion, Clause: 52
Condition: !attack_was_social_engineering, Clause: 56
Condition: attack_was_denial_of_service, Clause: 60
Conclusion: denial_of_service

Rule 16
Condition: attack_was_identified, Clause: 4
Condition: !attack_was_ransomware, Clause: 8
Condition: !attack_was_virus, Clause: 12
Condition: !attack_was_trojan_horse, Clause: 16
Condition: !attack_was_malware, Clause: 20
Condition: !attack_was_ddos, Clause: 24
Condition: !attack_was_botnet, Clause: 28
Condition: !attack_was_spyware, Clause: 32
Condition: !attack_was_man_in_middle, Clause: 36
Condition: !attack_was_password_attack, Clause: 40
Condition: !attack_was_sql_injection, Clause: 44
Condition: !attack_was_dns_tunneling, Clause: 48
Condition: !attack_was_network_intrusion, Clause: 52
Condition: !attack_was_social_engineering, Clause: 56
Condition: !attack_was_denial_of_service, Clause: 60
Condition: attack_was_packet_analyzer, Clause: 64
Conclusion: packet_analyzer

Rule 17
Condition: attack_was_identified
Condition: !attack_was_ransomware
Condition: !attack_was_virus
Condition: !attack_was_trojan_horse
Condition: !attack_was_malware
```

```
Condition: !attack_was_ddos
Condition: !attack_was_botnet
Condition: !attack_was_spyware
Condition: !attack_was_man_in_middle
Condition: !attack_was_password_attack
Condition: !attack_was_sql_injection
Condition: !attack_was_dns_tunneling
Condition: !attack_was_network_intrusion
Condition: !attack_was_social_engineering
Condition: !attack_was_denial_of_service
Condition: !attack_was_packet_analyzer
Conclusion: nil

Variable: attack_was_identified
Type: boolean
Instantiated: false
Clause Number: 4

Variable: attack_was_ransomware
Type: boolean
Instantiated: false
Clause Number: 8

Variable: attack_was_virus
Type: boolean
Instantiated: false
Clause Number: 12

Variable: attack_was_trojan_horse
Type: boolean
Instantiated: false
Clause Number: 16

Variable: attack_was_malware
Type: boolean
Instantiated: false
Clause Number: 20

Variable: attack_was_ddos_attack
Type: boolean
Instantiated: false
Clause Number: 24

Variable: attack_was_botnet_activity
Type: boolean
Instantiated: false
Clause Number: 28
```

```
Variable: attack_was_spyware
Type: boolean
Instantiated: false
Clause Number: 32

Variable: attack_was_man_in_middle_attack
Type: boolean
Instantiated: false
Clause Number: 36

Variable: attack_was_password_attack
Type: boolean
Instantiated: false
Clause Number: 40

Variable: attack_was_sql_injection
Type: boolean
Instantiated: false
Clause Number: 44

Variable: attack_was_dns_tunneling
Type: boolean
Instantiated: false
Clause Number: 48

Variable: attack_was_network_intrusion
Type: boolean
Instantiated: false
Clause Number: 52

Variable: attack_was_social_engineering_attack
Type: boolean
Instantiated: false
Clause Number: 56

Variable: attack_was_denial_of_service_attack
Type: boolean
Instantiated: false
Clause Number: 60

Variable: attack_was_packet_analyzer
Type: boolean
Instantiated: false
Clause Number: 64

Conclusion: ransomware
```

How to Stop Attack: Immediately isolate the affected systems to prevent spread. Restore data from backups after ensuring no remnants of the malware remain.
How to Prevent Attack: Regular backups, staff training on phishing and malicious emails, and updated antivirus programs are crucial.

Conclusion: virus
How to Stop Attack: Use updated antivirus software to identify and remove the virus. Repair or restore any damaged files.
How to Prevent Attack: Keep all systems and antivirus software updated. Regularly scan for and remove potential threats.

Conclusion: trojan_horse
How to Stop Attack: Identify and remove the Trojan. Reinforce network security by closing any backdoors created by the Trojan.
How to Prevent Attack: Avoid downloading software from untrusted sources. Keep security software and system patches up-to-date.

Conclusion: malware
How to Stop Attack: Isolate infected systems, run malware scans, and remove malicious files. Review and tighten security controls.
How to Prevent Attack: Maintain updated antivirus solutions and perform regular system scans. Educate users about safe browsing habits.

Conclusion: ddos_attack
How to Stop Attack: Deploy DDoS mitigation techniques like rate limiting and scrubbing. Reroute traffic if necessary.
How to Prevent Attack: Invest in robust DDoS protection services and plan for excess bandwidth. Regularly monitor network traffic patterns.

Conclusion: botnet_activity
How to Stop Attack: Identify and block botnet-related network traffic. Cleanse affected systems and monitor for further activity.
How to Prevent Attack: Employ advanced network monitoring and intrusion detection systems. Train staff to recognize signs of botnet activity.

Conclusion: spyware
How to Stop Attack: Employ anti-spyware tools to detect and remove spyware. Review and reset compromised security settings.
How to Prevent Attack: Use comprehensive endpoint protection and conduct regular security training for users.

Conclusion: packet_analyzer
How to Stop Attack: Detect and isolate unauthorized network sniffing activities.
How to Prevent Attack: Use encryption and secure network protocols.

Conclusion: man_in_the_middle_attack
How to Stop Attack: Encrypt all data transmissions. Use HTTPS and SSL/TLS protocols

to secure web communications.
How to Prevent Attack: Employ strong encryption for data in transit. Educate users on the importance of secure connections.

Conclusion: password_attack
How to Stop Attack: Immediately reset compromised passwords. Implement multi-factor authentication for added security.
How to Prevent Attack: Use strong, unique passwords and change them regularly. Educate users on secure password practices.

Conclusion: sql_injection
How to Stop Attack: Immediately patch vulnerable SQL elements. Use prepared statements and parameterized queries for database access.
How to Prevent Attack: Conduct code reviews to identify vulnerabilities. Educate developers about secure coding practices.

Conclusion: dns_tunneling
How to Stop Attack: Monitor and analyze DNS traffic for anomalies. Block unauthorized DNS queries and tunneling activities.
How to Prevent Attack: Implement DNS security measures like DNSSEC. Regularly review and update DNS firewall settings.

Conclusion: network_intrusion
How to Stop Attack: Aquire a VPN and turn it on, reset the router, and possible contact network provider for a change in IP address depending on the scope of the attack.
How to Prevent Attack: Continue use of a VPN for further browsing and connect to websites that are SSL certified.

Conclusion: social_engineering_attack
How to Stop Attack: Educate employees to recognize and report attempts. Review and update security protocols.
How to Prevent Attack: Conduct regular training on social engineering tactics. Foster a culture of security awareness in the organization.

Conclusion: denial_of_service_attack
How to Stop Attack: Implement network hardware capable of handling unexpected traffic spikes. Use cloud-based DDoS mitigation services.
How to Prevent Attack: Prepare a response plan for potential attacks. Increase network redundancy and distribute network resources.

Conclusion: nil
Default: No attack found.

7. Program Run

```
************************************************************
* [2024-02-20 23:06:01] - Beginning Backward Chaining procedure...
************************************************************

************************************************************
* [2024-02-20 23:06:01] - Current Goal: ransomware, located at rule number: 15
************************************************************

************************************************************
* [2024-02-20 23:06:01] - Clauses for rule 15
************************************************************
----------------- Clauses -----------------
Clause Name: files_corrupted_without_ransom
Clause Numbers: 5
-----------------------------------------
Clause Name: file_changes_or_configurations
Clause Numbers: 4 10
-----------------------------------------
Clause Name: unauthorized_access_or_control
Clause Numbers: 3 9 15 21 27 33 39 45
-----------------------------------------
Are there files corrupted without a request for ransom?
: no
Are there changes to files or configurations?
: yes
Is there unauthorized access or control?
: no

************************************************************
* [2024-02-20 23:06:20] - Required bools vs. Users answers for conclusion to be
met:
************************************************************
------- Clause Boolean Comparison -------
Index     Required Clause     Actual Clause
-----------------------------------------
0         True                False
1         True                True
2         True                False
-----------------------------------------

************************************************************
* [2024-02-20 23:06:20] - Current Goal: virus, located at rule number: 30
************************************************************

************************************************************
* [2024-02-20 23:06:20] - Clauses for rule 30
```

```
************************************************************
---------------- Clauses ----------------
Clause Name: !files_corrupted_without_ransom
Clause Numbers: 11
----------------------------------------
Clause Name: file_changes_or_configurations
Clause Numbers: 4 10
----------------------------------------
Clause Name: unauthorized_access_or_control
Clause Numbers: 3 9 15 21 27 33 39 45
----------------------------------------

************************************************************
* [2024-02-20 23:06:20] - Required bools vs. Users answers for conclusion to be
met:
************************************************************
------- Clause Boolean Comparison -------
Index     Required Clause     Actual Clause
--------------------------------------
0         False               False
1         True                True
2         True                False
--------------------------------------

************************************************************
* [2024-02-20 23:06:20] - Current Goal: trojan_horse, located at rule number: 45
************************************************************

************************************************************
* [2024-02-20 23:06:20] - Clauses for rule 45
************************************************************
---------------- Clauses ----------------
Clause Name: evidence_of_data_exfiltration
Clause Numbers: 18
----------------------------------------
Clause Name: backdoor_accounts_or_tools_found
Clause Numbers: 17 23
----------------------------------------
Clause Name: !file_changes_or_configurations
Clause Numbers: 16 22 28 34 40 46
----------------------------------------
Clause Name: unauthorized_access_or_control
Clause Numbers: 3 9 15 21 27 33 39 45
----------------------------------------
Is there evidence of data exfiltration?
: no
Are there backdoor accounts or tools found on the system?
```

```
: yes

************************************************************
* [2024-02-20 23:06:24] - Required bools vs. Users answers for conclusion to be
met:
************************************************************
------- Clause Boolean Comparison -------
Index     Required Clause     Actual Clause
----------------------------------------
0         True                False
1         True                True
2         False               False
3         True                False
----------------------------------------

************************************************************
* [2024-02-20 23:06:24] - Current Goal: malware, located at rule number: 60
************************************************************

************************************************************
* [2024-02-20 23:06:24] - Clauses for rule 60
************************************************************
----------------- Clauses -----------------
Clause Name: !evidence_of_data_exfiltration
Clause Numbers: 24
----------------------------------------
Clause Name: backdoor_accounts_or_tools_found
Clause Numbers: 17 23
----------------------------------------
Clause Name: !file_changes_or_configurations
Clause Numbers: 16 22 28 34 40 46
----------------------------------------
Clause Name: unauthorized_access_or_control
Clause Numbers: 3 9 15 21 27 33 39 45
----------------------------------------

************************************************************
* [2024-02-20 23:06:24] - Required bools vs. Users answers for conclusion to be
met:
************************************************************
------- Clause Boolean Comparison -------
Index     Required Clause     Actual Clause
----------------------------------------
0         False               False
1         True                True
2         False               False
3         True                False
```

```
---------------------------------------

************************************************************
* [2024-02-20 23:06:24] - Current Goal: ddos_attack, located at rule number: 75
************************************************************

************************************************************
* [2024-02-20 23:06:24] - Clauses for rule 75
************************************************************
----------------- Clauses -----------------
Clause Name: traffic_targeting_multiple_systems
Clause Numbers: 31
-----------------------------------------
Clause Name: high_volume_of_outgoing_traffic
Clause Numbers: 30 36
-----------------------------------------
Clause Name: !backdoor_accounts_or_tools_found
Clause Numbers: 29 35 41 47
-----------------------------------------
Clause Name: !file_changes_or_configurations
Clause Numbers: 16 22 28 34 40 46
-----------------------------------------
Clause Name: unauthorized_access_or_control
Clause Numbers: 3 9 15 21 27 33 39 45
-----------------------------------------
Is the traffic targeting multiple systems?
: no
Is there a high volume of outgoing traffic?
: yes

************************************************************
* [2024-02-20 23:06:28] - Required bools vs. Users answers for conclusion to be
met:
************************************************************
------- Clause Boolean Comparison -------
Index     Required Clause     Actual Clause
-------------------------------------
0         True                False
1         True                True
2         False               False
3         False               False
4         True                False
-------------------------------------

************************************************************
* [2024-02-20 23:06:28] - Current Goal: botnet_activity, located at rule number: 90
************************************************************
```

```
************************************************************
* [2024-02-20 23:06:28] - Clauses for rule 90
************************************************************
---------------- Clauses ----------------
Clause Name: !traffic_targeting_multiple_systems
Clause Numbers: 37
-----------------------------------------
Clause Name: high_volume_of_outgoing_traffic
Clause Numbers: 30 36
-----------------------------------------
Clause Name: !backdoor_accounts_or_tools_found
Clause Numbers: 29 35 41 47
-----------------------------------------
Clause Name: !file_changes_or_configurations
Clause Numbers: 16 22 28 34 40 46
-----------------------------------------
Clause Name: unauthorized_access_or_control
Clause Numbers: 3 9 15 21 27 33 39 45
-----------------------------------------

************************************************************
* [2024-02-20 23:06:28] - Required bools vs. Users answers for conclusion to be
met:
************************************************************
------- Clause Boolean Comparison -------
Index     Required Clause     Actual Clause
-------------------------------------
0         False               False
1         True                True
2         False               False
3         False               False
4         True                False
-------------------------------------

************************************************************
* [2024-02-20 23:06:28] - Current Goal: spyware, located at rule number: 105
************************************************************

************************************************************
* [2024-02-20 23:06:28] - Clauses for rule 105
************************************************************
---------------- Clauses ----------------
Clause Name: unsual_system_behavior_without_high_traffic
Clause Numbers: 43
-----------------------------------------
Clause Name: !high_volume_of_outgoing_traffic
```

```
Clause Numbers: 42 48
----------------------------------------
Clause Name: !backdoor_accounts_or_tools_found
Clause Numbers: 29 35 41 47
----------------------------------------
Clause Name: !file_changes_or_configurations
Clause Numbers: 16 22 28 34 40 46
----------------------------------------
Clause Name: unauthorized_access_or_control
Clause Numbers: 3 9 15 21 27 33 39 45
----------------------------------------
Is there unusual system behavior without a high volume of traffic?
: yes

************************************************************
* [2024-02-20 23:06:30] - Required bools vs. Users answers for conclusion to be
met:
************************************************************
------- Clause Boolean Comparison -------
Index     Required Clause     Actual Clause
----------------------------------------
0         True                True
1         False               False
2         False               False
3         False               False
4         True                False
----------------------------------------

************************************************************
* [2024-02-20 23:06:30] - Current Goal: packet_analyzer, located at rule number:
120
************************************************************

************************************************************
* [2024-02-20 23:06:30] - Clauses for rule 120
************************************************************
----------------- Clauses -----------------
Clause Name: !unsual_system_behavior_without_high_traffic
Clause Numbers: 49
----------------------------------------
Clause Name: !high_volume_of_outgoing_traffic
Clause Numbers: 42 48
----------------------------------------
Clause Name: !backdoor_accounts_or_tools_found
Clause Numbers: 29 35 41 47
----------------------------------------
Clause Name: !file_changes_or_configurations
```

```
Clause Numbers: 16 22 28 34 40 46
----------------------------------------
Clause Name: unauthorized_access_or_control
Clause Numbers: 3 9 15 21 27 33 39 45
----------------------------------------

************************************************************
* [2024-02-20 23:06:30] - Required bools vs. Users answers for conclusion to be
met:
************************************************************
------- Clause Boolean Comparison -------
Index     Required Clause     Actual Clause
----------------------------------------
0         False               False
1         False               False
2         False               False
3         False               False
4         True                False
----------------------------------------

************************************************************
* [2024-02-20 23:06:30] - Current Goal: password_attack, located at rule number:
135
************************************************************

************************************************************
* [2024-02-20 23:06:30] - Clauses for rule 135
************************************************************
----------------- Clauses -----------------
Clause Name: unexpected_login_attempts
Clause Numbers: 54
----------------------------------------
Clause Name: network_activity_external
Clause Numbers: 53 59
----------------------------------------
Clause Name: unusual_network_activity
Clause Numbers: 52 58 64 70 76 82
----------------------------------------
Clause Name: !unauthorized_access_or_control
Clause Numbers: 51 57 63 69 75 81 87 93
----------------------------------------
Are there unexpected login attempts?
: no
Is the network activity external?
: no
Is there unusual network activity?
: yes
```

```
************************************************************
* [2024-02-20 23:06:49] - Required bools vs. Users answers for conclusion to be
met:
************************************************************
------- Clause Boolean Comparison -------
Index     Required Clause     Actual Clause
----------------------------------------
0         True                False
1         True                False
2         True                True
3         False               False
----------------------------------------

************************************************************
* [2024-02-20 23:06:49] - Current Goal: man_in_the_middle_attack, located at rule
number: 150
************************************************************

************************************************************
* [2024-02-20 23:06:49] - Clauses for rule 150
************************************************************
----------------- Clauses -----------------
Clause Name: !unexpected_login_attempts
Clause Numbers: 60
----------------------------------------
Clause Name: network_activity_external
Clause Numbers: 53 59
----------------------------------------
Clause Name: unusual_network_activity
Clause Numbers: 52 58 64 70 76 82
----------------------------------------
Clause Name: !unauthorized_access_or_control
Clause Numbers: 51 57 63 69 75 81 87 93
----------------------------------------

************************************************************
* [2024-02-20 23:06:49] - Required bools vs. Users answers for conclusion to be
met:
************************************************************
------- Clause Boolean Comparison -------
Index     Required Clause     Actual Clause
----------------------------------------
0         False               False
1         True                False
2         True                True
3         False               False
```

```
---------------------------------------

************************************************************
* [2024-02-20 23:06:49] - Current Goal: dns_tunneling, located at rule number: 165
************************************************************

************************************************************
* [2024-02-20 23:06:49] - Clauses for rule 165
************************************************************
----------------- Clauses -----------------
Clause Name: increase_in_dns_requests
Clause Numbers: 67
-----------------------------------------
Clause Name: unusual_activity_related_to_data_flow
Clause Numbers: 66 72
-----------------------------------------
Clause Name: !network_activity_external
Clause Numbers: 65 71 77 83
-----------------------------------------
Clause Name: unusual_network_activity
Clause Numbers: 52 58 64 70 76 82
-----------------------------------------
Clause Name: !unauthorized_access_or_control
Clause Numbers: 51 57 63 69 75 81 87 93
-----------------------------------------
Is there an increase in DNS requests?
: yes
Is the unusual activity related to data flow?
: yes

************************************************************
* [2024-02-20 23:06:54] - Required bools vs. Users answers for conclusion to be
met:
************************************************************
------- Clause Boolean Comparison -------
Index     Required Clause     Actual Clause
-----------------------------------------
0         True                True
1         True                True
2         False               False
3         True                True
```

8. Analysis of Program, Modifications, and Enhancements

In the development of the improved inference engine, significant modifications were made to the original erroneous program to enhance its functionality, maintainability, and user interaction.

8.1 Backward Chaining Analysis

8.1.1 Enhanced Data Structure Management: The new implementation introduces sophisticated data structure management. The utilization of std::map for rules and std::vector for clauses and conclusions optimizes data storage and retrieval. Unlike the original version, this approach allows for more efficient parsing and handling of the knowledge base, leading to quicker and more accurate rule matching and conclusion derivation.

8.1.2 Streamlined Backward Chaining Process: The backward chaining process has been significantly streamlined. The use of a std::stack for managing conclusions and a std::set for tracking checked rules ensures that the engine efficiently navigates through potential conclusions without redundancy. This refined process contrasts sharply with the original program, which lacked an effective system for managing the flow and evaluation of rules and conclusions.

8.1.3 Improved Clause Processing: The clause processing mechanism has been overhauled for better performance and user interaction. The function getUserInput solicits and validates user responses, enhancing user engagement and accuracy in clause evaluation. The functions determineRequiredClauseBools and determineRealClauseBools manage the boolean logic required for clause validation, a feature not as firmly implemented in the original code.

8.1.4 Optimization and Memory Management: The new program exhibits improved memory management, evident in the use of reserve for pre-allocating memory in vectors, contributing to faster execution, as well as utilizing pass by reference instead of by value. This refined memory handling was absent in the original program, making the updated engine more efficient in resource utilization.

8.1.5 Enhanced Readability and Maintainability: To further improve the maintainability and simplicity of the code, a header file has been introduced in the new implementation. This modular approach, with a clear separation of the program's logic into distinct functions and files, greatly enhances readability and ease of maintenance. In contrast, the original erroneous program was less structured, making it more challenging to comprehend and modify.

8.2 Forward Chaining Analysis

8.2.1 File Handling and Data Organization:

In forwardChaining.cpp, data is read from a file using std::ifstream inputFile("kbfc.txt"); (line 10), which is more efficient and flexible than hard-coding data.The data is organized into an array, as shown in the

initializeArr function (lines 8-42). This approach is more structured compared to the unorganized data handling in the erroneous code.

8.2.2 Modular Design:

Functions like populateRules (lines 43-72), populateVariables (lines 74-110), and populateConclusions (lines 112-134) indicate a modular approach. Each function handles a specific part of the process, unlike the monolithic structure of the erroneous code.

8.2.3 Use of Standard Library and Data Structures:

The use of std::string and std::vector (lines 1-5) is more efficient and safer compared to plain arrays and C-style strings. For example, std::istringstream (lines 48, 82, 119) is used for parsing strings efficiently.

8.2.4 Error Handling:

Basic error handling is included, such as checking if the file opened correctly (lines 12-14). This is a significant improvement in terms of robustness compared to the lack of error handling in the erroneous code.

8.2.5 String Parsing and Processing:

The program uses std::istringstream and std::getline for parsing strings (lines 48-70, 82-108, 119-132), which is more efficient and less error-prone than manual string manipulation in the erroneous code.

8.2.6 Improved Variable Naming and Comments:

Variable names and comments in forwardChaining.cpp are more descriptive, aiding in code readability. For instance, populateRules clearly indicates its purpose, and lines like 52-56 show clear and readable string processing.

9. Analysis of the Results

Run Number	Network Attack Type	Result (Detected/Not Detected	Memory Usage (KB)	Execution Time Excluding User Input (Milliseconds)	Execution Time Including User Input (Seconds)	Modifications Impact
1	Ransomware	Detected	4.994 KB	4.6946 ms	2508.17 ms	Utilizing std::map for optimized data retrieval, this code segment contributes to the notably lower memory usage and rapid execution times in ransomware detection scenarios.
2	Malware	Detected	11.27 KB	10.2705 ms	2558.45 ms	The improved clause processing mechanism in the function determineRealClauseBools enhances the accuracy of malware detection, with a tolerable increase in memory usage. (boolean comparison aspect)
3	DDoS	Detected	14.17 KB	13.0374 ms	3612.84 ms	Improved memory management techniques, such as pre-allocating memory and using references, have significantly boosted the execution speed, allowing for quick DDoS attack detection.
4	Social Engineering Attack	Detected	34.20 KB	33.6022 ms	6799.87 ms	The use of std::set for tracking checked rules minimizes redundancy in the backward chaining process, yielding a faster and more efficient detection of social engineering attacks.
5	Denial Of Service	Detected	41.54 KB	39.5036 ms	7463.21 ms	By employing a std::stack to manage conclusions, the system can efficiently navigate through potential outcomes, which is vital in the complex and memory-intensive process of detecting Denial of Service attacks.

10. Conclusion

This report delves into the development and operation of an inference engine designed for cybersecurity, specifically focusing on diagnosing and mitigating network attacks. The engine innovatively integrates both backward and forward chaining methods. Backward chaining is crucial for identifying the root causes of network attacks by analyzing symptoms, while forward chaining is employed to develop and implement strategies to counter identified threats. The detailed implementation of this system, encompassing methodologies, algorithms, and coding practices, is explored in the report. Furthermore, it examines the enhancements made to the program to boost its efficiency and effectiveness.

During the process of creating this inference engine, I engaged in the process of constructing decision trees and deriving rules from these trees, which was a huge step in developing the intelligence of the system. This experience was not just a deep dive into the technical aspects of intelligent systems but also an enlightening journey into the mechanics of how individual decision-making components can work together to form a comprehensive program. It emphasized the significance of meticulous planning and strategic rule formulation, ensuring the system's precision and efficiency.

11. References

11.1 Cyber Security Network Attacks

- "Top 20 Most Common Types of Cyber Attacks." *Fortinet*, www.fortinet.com/resources/cyberglossary/types-of-cyber-attacks. Accessed 20 Feb. 2024.

12. Contributions

12.1 Logan Falkenberg:

- Created decision trees/backward chaining inference engine

12.2 Alyssa Hewlett:

- Created forward chaining inference engine

12.3 Jonathan Sibbett:

- Co-created forward/backward inference engines