https://doi.org/10.7840/kics.2019.44.5.904

통계적 분석 기반 비공개 프로토콜의 구분자 및 정적필드 추출 방법

이 민 섭◆, 심 규 석*, 구 영 훈*, 백 의 준*, 채 병 민**, 문 호 원**, 김 명 섭°

Identification of Delimiter and Static Field in Protocol Reverse Engineering Using Statistic Analysis

Min-Seob Lee◆, Kyu-Seok Shim*, Young-Hoon Goo*, Ui-Jun Baek*, Byeong-Min Chae**, Ho-Won Moon**, Myung-Sup Kim°

요 약

오늘날의 네트워크 환경은 고속화 및 대용량화가 이루어지면서 개별 네트워크 환경에서 전 세계가 하나의 네트워크로 연결된 환경으로 변모하고 있다. 이에 따라서 매우 다양하고 복잡한 통신들이 이루어지고 있으며 특정 통신들은 비공개 프로토콜을 사용하고 있다. 이러한 비공개 프로토콜은 상용화되지 않고 폐쇄적인 성격을 띄고 있으며 프로토콜의 사양이 문서로 나타나 있지 않다. 비공개 프로토콜이 악의적인 행위에 사용되지 않는다면 네트워크 보안에 문제가 되지 않지만 악의적인 행위에 사용되는 비공개 프로토콜들은 프로토콜의 사양을 알 수 없기에 방지나 대처가 어렵다. 이런 문제점을 해결하기 위해서 비공개 프로토콜의 사양을 추출하는 프로토콜 리버스 엔지니어링과 관련된 수많은 연구가 현재까지 진행되고 있다. 본 논문은 프로토콜의 메시지 포맷, 플로우 포맷의 사양을 추출하는데 있어서 토대가 되는 필드를 통계적 분석을 기반으로 명확하게 추출하는 방법을 제안한다.

키워드 : 프로토콜 리버스 엔지니어링, 비공개 프로토콜, 구분자, 필드, 프로토콜 구조 분석
Key Words : Protocol Reverse Engineering, Private Protocol, Delimiter, Field, Protocol Structure Analysis

ABSTRACT

Today's network environment has become faster and larger, and as a result, the individual network environment is changing to an environment where the world is connected to a single network. Thus, the various and complicated communications are being made and specific communications are using a private protocol. These private protocols are generally unused and close, and the protocol specifications are not documented. If the private protocol is not used for malicious behavior, it is not a problem for the network security. However, the private protocol used for malicious behavior is difficult to detect or deal with malicious behavior because the specification of the protocol is unknown To solve these problems, many studies related to protocol reverse engineering for extracting specifications of a private protocol have been conducted. In this paper, we propose a method to explicitly extract fields based on statistical analysis for extracting protocol message format and flow format.

※ 이 논문은 2016년도 국방과학연구소와 한화시스템(주)의 재원을 받아 수행된 연구(UE161105ED)와 2018년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(NRF-2018R1D1A1B07045742)
◆ First Author : Department of Computer and Information Science, Korea University, chenlima2@korea.ac.kr, 학생회원
° Corresponding Author : Department of Computer and Information Science, Korea University, tmskim@korea.ac.kr, 종신회원
* Department of Computer and Information Science, Korea University, {kusuk007, gyh0808, pb1069}@korea.ac.kr, 학생회원
** Hanwha Systems, {byeonmin.chae, moon1000}@hanwha.com
논문번호 : 201812-394-B-RN, Received December 21, 2018; Revised March 11, 2019; Accepted March 13, 2019

Ⅰ. 서 론

오늘날의 네트워크 환경은 고속화 및 대용량화가 이루어지면서 개별 네트워크 환경에서 전 세계가 하나의 네트워크로 연결된 환경으로 변모하고 있다. 이러한 변화된 네트워크 환경 때문에 매우 다양하고 복잡한 통신들이 이루어지기 시작했으며 상용화된 프로토콜을 사용하는 통신 이외에도 비공개 프로토콜을 사용하는 새로운 응용이나 악성행위들이 지속적으로 발생하고 있다. 비공개 프로토콜은 알려져 있지 않거나 프로토콜의 사양이 문서화 되어있지 않은 프로토콜로써[1,2] 이러한 비공개 프로토콜을 사용하는 악성행위[3] 같은 경우에는 비공개 프로토콜의 특성 상 트래픽 분석을 하더라도 그 내용을 정확하게 파악하기 힘들기 때문에 탐지나 대처하기가 매우 어렵다. 이러한 문제점을 해결하기 위해서 특정 비공개 프로토콜의 사양을 추출하는 것을 목표로 하는 학문인 프로토콜 리버스 엔지니어링 분야가 꾸준히 연구되어 왔다. 갈수록 복잡해지고 새로운 유형의 사이버 공격이 발생하는 현대 사회의 네트워크 환경에서는 비공개 프로토콜을 사용하는 악성행위들에 대처하기 위해서 비공개 프로토콜을 분석하여 구조를 추출하는 프로토콜 리버스 엔지니어링은 필수적으로 진행되어야 하는 학문이다.

프로토콜 리버스 엔지니어링은 크게 두 가지 방식으로 나눌 수 있다. 첫 번째 방식은 수동적인 프로토콜 리버스 엔지니어링 방법이다.[4,5] 이 방식은 과거부터 진행되어 오던 전통적인 프로토콜 리버스 엔지니어링 방식으로써 분석자가 직접 수동으로 비공개 프로토콜을 분석하기 때문에 시간이 많이 소요되며, 분석자의 능력에 따라 오류가 발생하기 쉽기 때문에 오늘날에 발생하는 비공개 프로토콜처럼 복잡하고 다양한 비공개 프로토콜을 분석하는 방법으로써는 적절하지 않다. 두 번째 방식은 자동적인 프로토콜 리버스 엔지니어링 방법이다. 자동적인 프로토콜 리버스 엔지니어링은 기존의 수동적인 프로토콜 리버스 엔지니어링을 보완하기 위해서 고안되었다. 현재 자동적인 프로토콜 리버스 엔지니어링 방법은 명확한 방법이 정해져 있지 않고 다양한 연구결과를 통해 여러 가지 프로토콜 리버스 엔지니어링 방법들이 제안되었다. 자동적인 프로토콜 리버스 엔지니어링 방법은 두 가지 방법으로 나눌 수 있다[6].

첫 번째 방법은 실행 트레이스 기반 분석 방법이다.[7-10] 실행 트레이스 기반 분석 방법은 특정 비공개 프로토콜을 사용하는 프로그램 바이너리의 실행을 모니터링 하여 추출한 실행 트레이스를 분석 프로그램의 입력으로 사용한다. 이 방법은 비공개 프로토콜의 사양을 명확하게 추출할 수 있는 장점이 있다. 하지만 해당 비공개 프로토콜을 사용하는 프로그램 바이너리에 접근할 수 있는 경우에만 사용이 가능한 방법이다. 현실적으로 비공개 프로토콜을 사용하는 프로그램 바이너리에 접근 하는 것은 불가능하기 때문에 본 논문에서 사용할 방법으로는 적절하지 않다.

두 번째 방법은 네트워크 트레이스 기반 분석 방법이다.[11-14] 네트워크 트레이스 기반 분석 방법은 특정 비공개 프로토콜의 네트워크 트래픽을 캡처하여 네트워크 트레이스를 구성하고 이 네트워크 트레이스를 분석 프로그램의 입력으로 사용하여 비공개 프로토콜을 분석하는 방법이다. 이 방법은 실행 트레이스 기반 분석 방법과 다르게 특정 비공개 프로토콜을 사용하는 프로그램 바이너리에 접근을 하지 않아도 라우터에서 발생하는 트래픽을 캡처하여 분석을 할 수 있기 때문에 본 논문에서는 해당 방법을 사용한다.

네트워크 트레이스 기반 분석 방법은 하향식 방식과 상향식 방식으로 분류할 수 있다. 하향식 방식은 전체적인 구조에서 상세한 구조를 구분해 나가나는 방식이기 때문에 너무 대략적인 구조만 추출할 수 있는 단점이 있다. 상향식 방식은 네트워크 트래픽을 구성하는 의미를 가지는 가장 작은 단위인 필드를 먼저 추출하고 필드의 시퀀스로 이루어진 메시지 포맷, 메시지의 시퀀스로 이루어진 플로우 포맷을 순차적으로 추출하기 때문에 하향식 방식보다 좀 더 상세한 구조를 추출할 수 있다. 따라서 본 논문에서는 상향식 방식에서 메시지 포맷과 플로우 포맷을 추출하는 과정에서 가장 중요한 필드를 명확하게 추출하는 방법을 제안한다.

Ⅱ. 관련 연구

네트워크 트레이스 기반 분석 방법을 사용한 선행연구에는 다양한 연구들이 존재하나 본 논문에서는 가장 유망한 분석 방법으로 주목 받고 있는 J. Z. Luo et al. 이 제안한 AutoReEngine[15], Ignacio Bermudez et al. 이 제안한 FieldHunter[16] 두 가지 방법론만 언급한다.

2.1 AutoReEngine

AutoReEngine은 단일 비공개 프로토콜에 대한 네트워크 트래픽을 입력으로 사용한다. AutoReEngine은 데이터 전처리를 수행하는 Data Pre-Processing, 프로토콜의 정적필드를 추출하는 Protocol Keyword Extraction, 정적필드의 시퀀스로 이루어진 메시지 포맷을 추출하는 Message Format Extraction, 메시지 포

맷의 시퀀스로 이루어진 플로우 포맷을 추출하는 State Machine Inference 총 4가지 단계로 구성되어있다. AutoReEngine에서 핵심적으로 사용하는 기법은 순차 패턴 마이닝 기법중 하나인 Apriori 알고리즘으로써 입력으로 사용된 비공개 프로토콜에서 특정 Threshold 이상으로 빈번하게 발생하는 문자열 후보를 추출한다. 그 후에 문자열 후보들 중에서 메시지와 라인에서 비교적 고정적인 위치에 나타나는 문자열 후보들만 최종 필드로 추출한다.

2.2 FieldHunter

FieldHunter는 AutoReEngine과 마찬가지로 단일 비공개 프로토콜에 대한 네트워크 트래픽을 입력으로 사용하며 구분자 필드와 구분자 필드로 구분되는 필드들을 추출하는 Field Extractor, 필드가 가지는 의미를 추출하는 Field Type Inference Engine 총 2가지 단계로 구성되어있다. FieldHunter는 텍스트 프로토콜인 경우에는 알파벳이나 숫자가 아닌 길이가 1 혹은 2이며 가장 빈번하게 나타나는 값을 구분자 필드로 추출한다. 그 다음 추출한 구분자 필드로 메시지를 구성하는 필드들을 구분하고 구분한 필드들에서 key-value를 구분하는 구분자를 추출한다.

2.3 문제 정의

앞서 언급한 두 가지 방법론들은 각각의 한계점이 존재한다. AutoReEngine의 경우 Apriori 알고리즘에 적용하는 support 단위로 Session, Site-Specific Session set을 사용한다. Session은 플로우와 같은 의미로써 5-tuple(Source IP Address, Destination IP Address, Source Port, Destination Port, L4 Protocol)이 같은 패킷들의 집합을 의미한다. Site-Specific Session set는 하나의 서버와 해당 서버랑 통신하는 클라이언트 간에 연결을 맺고 있는 모든 플로우들을 원소로 하는 집합을 의미한다. 이 두 가지 support 단위를 사용할 경우 두 단위 모두에서 빈번하게 나타나는 문자열들을 필드로써 추출하게 되는데 플로우에서 한 번만 발생하는 키워드의 경우 필드로 추출되어야함에도 불구하고 support를 만족하지 못하기 때문에 필드로 추출하지 못하는 한계점이 존재한다.

FieldHunter는 앞에서 언급하였듯이 필드들을 구분하는 구분자 필드를 먼저 추출한 다음에 추출한 구분자 필드를 사용하여 필드들을 구분하는 작업을 수행한다. 이럴 경우에 구분자 필드가 존재하지 않는 프로토콜은 필드를 추출할 수 없는 한계점이 존재한다.

따라서 본 논문에서는 미리 정의한 구분자 필드 후보들에 대하여 구분자 필드가 존재하는지에 대한 알고리즘을 적용하여 구분자 필드의 존재여부를 우선적으로 검사하여 추출하고 구분자 필드의 존재여부와 상관없이 Apriori 알고리즘에 3가지 support를 적용하여 정교하게 Static Field를 추출하는 방법을 제안한다. 본 논문에서 추출하고자 하는 Static Field는 메시지에서 비슷한 위치에 발생하며 일정한 Length값을 가지고 프로토콜을 구성하는 Flow, Message, Server_set에서 빈번하게 나타나야한다는 속성을 기반으로 한다.

Ⅲ. 본 론

3.1 System Overview

본 논문에서 제안하는 시스템은 단일 프로토콜 트래픽을 입력으로 사용하여 구분자와 필드를 추출하는 시스템이다. 그림 1은 제안하는 시스템의 전체적인 흐름도이다.

먼저, 트래픽이 입력되면 Message Assemble 단계를 시작하며 이 단계에서는 입력으로 사용된 비공개 프로토콜 트래픽파일을 플로우 형태로 읽어들인 후 해당 프로토콜이 TCP 프로토콜일 경우에는 하나의 TCP 세그먼트로, UDP 프로토콜일 경우 하나의 패킷을 메시지로 정의하여 플로우를 메시지들로 재조립한다.

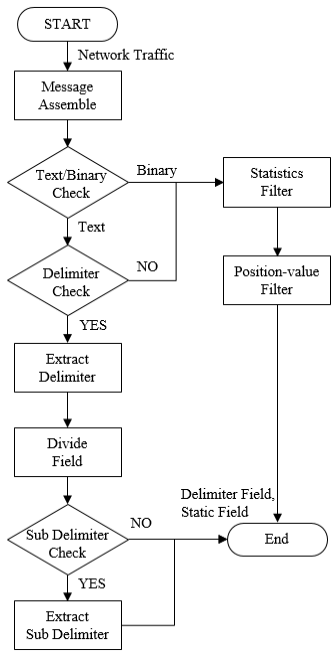

그림 1. 제안하는 구분자 및 정적필드 추출 시스템 흐름도
Fig. 1. Flow Chart of Delimiter and Static Field Extraction System

Message Assemble 단계를 마치면 입력된 트래픽이 Text 기반 프로토콜인지 Binary 기반 프로토콜인지를 검사하고 Binary 기반 프로토콜일 시 Statistics Filter 단계로 넘어간다. Text 기반 프로토콜이면 구분자 필드를 추출하기 위해 구분자가 존재하는지 검사한다. 만약 구분자가 존재하는 Text 기반 프로토콜이라면 Extract Delimiter 단계로 넘어간다.

Extract Delimiter 단계에서는 미리 정의한 다섯 가지 구분자 필드 후보(|0d|0a|, space, ;, =, :)에 대하여 구분자 필드를 추출한다. 현재 상용화되어있고 구분자가 존재하는 Text 기반 프로토콜들의 경우 위 다섯 가지를 구분자로 사용하고 있으며 비공개 프로토콜도 비슷한 경향을 보일 거라는 통찰을 기반으로 한다. 또한 모든 1~2바이트 문자를 다 검사할 수도 있으나 미리 정의한 다섯 가지 후보를 제외하면 구분자가 될 수 있는 가능성이 매우 희박하고 수행시간이 길어져 비효율적이기 때문에 본 논문에서는 위 다섯 가지 구분자 필드 후보에 대하여 구분자 필드를 추출한다. 또한 본 연구진은 추출한 구분자 필드를 기반으로 메시지 내 필드들을 구분하는 모듈과 구분된 필드들에서 key-value, value-value를 구분하는 Sub-Delimiter를 추출하는 모듈을 설계하였으며 현재 개발 중에 있다. Text 기반 프로토콜이지만 구분자가 존재하지 않거나 Binary 기반 프로토콜이라면 통계적 분석 방법을 사용하는 Statistics Filter 단계와 Position-value Filter 단계를 통해 Static Field만을 추출한다. Statistics Filter 단계에서는 순차패턴 마이닝 기법 중 하나인 Apriori 알고리즘에 세 가지 support를 적용하여 Static Field 후보를 추출한다. Position-value Filter 단계에서는 앞 단계에서 추출한 Static Field의 후보들에 대하여 각각 Position-value를 계산하고 특정 Threshold 이상의 값을 가지는 후보들을 삭제하여 최종적으로 Static Field를 추출한다.

3.2 Message Assemble

본 단계에서는 위에서 설명한 것처럼 비공개 프로토콜 트래픽이 플로우 형태로 로드되고 각 플로우들을 L4 Protocol에 따라서 각기 다른 메시지 형태로 분할한다. 임의의 두 노드가 통신할 때 데이터는 패킷 단위로 송·수신되기 때문에 응용 계층 수준 데이터 단위(ADU)인 메시지 단위로 재조립한다. 그림 2와 같이 TCP 프로토콜은 스트림 기반의 연결지향 프로토콜이므로 메시지의 단위는 동일한 방향의 연속적으로 접해있는 패킷들의 집합으로 정의하고 UDP 프로토콜은 하나의 패킷을 메시지의 단위로 정의한다.

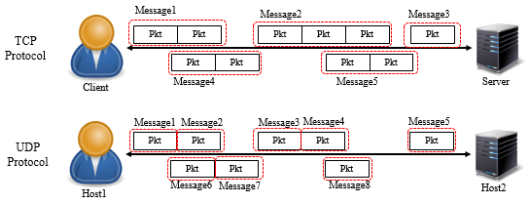

그림 2. Message Assemble 개념
Fig. 2. Concept of Message Assemble

3.3 Extract Delimiter

본 단계에서는 구분자 필드에 대한 두 가지 속성을 기반으로 구분자 필드를 추출한다. 첫 번째 속성은 구분자 필드가 모든 메시지에서 비슷한 횟수로 발생한다는 속성이다. 특정 프로토콜은 대부분 비슷한 유형의 메시지로 구성되어 있기 때문에 각 메시지에서 구분자 필드가 발생하는 횟수도 비슷하게 발생한다는 특징이 있다. 두 번째 속성은 구분자 필드는 메시지 내에서 고르게 분포한다는 속성이다. 구분자 필드의 역할은 메시지 내에서 필드를 구분하는 역할이기 때문에 메시지 내에서 균등하게 분포되어 있어야하며 한쪽에 치우쳐서 나타난다면 구분자 필드라고 정의하기 어렵다. 따라서 본 논문에서는 구분자 필드가 가지는 이 두 가지 속성에 초점을 맞춰서 구분자 필드를 추출한다. 그림 3은 본 단계의 의사코드이다.

우선 미리 정의한 구분자 필드 후보들에 대하여 각

```
Input : Flows consisting of packets
Output : Delimiter Field

01: foreach Delimiter Field Candidates do
02:     foreach Messages do
03:         Calculate Offset;
04:         Calculate Occurrence count in each Message;
05:     end
06: end
07: foreach Delimiter Field Candidates do
08:     Calculate Variance(Occurrence count);
09: end
10: foreach Delimiter Field Candidates do
11:     if(Variance of Occurrence count > Threshold)
12:         delete Delimiter Field Candidates;
13:     else
14:         foreach Messages do
15:             Calculate Average of Offset;
16:             Calculate Variance(Average of Offset);
17:         end
18: end
19: foreach Delimiter Field Candidates do
20:     if(Variance of Offset is Maximum)
21:         Extract to Delimiter Field;
22: end
```

그림 3. Extract Delimiter 단계 의사 코드
Fig. 3. Pseudo code for Extract Delimiter Stage

메시지에서의 Offset과 발생 횟수를 구한다. 각각의 구분자 필드 후보는 메시지 개수만큼 발생 횟수가 존재하고 발생 횟수에 대한 분산 값을 계산한다. 이 분산 값이 크다면 해당 구분자 필드 후보는 메시지마다 매우 상이한 빈도로 나타난다는 의미이다. 따라서 해당 구분자 필드 후보는 위에서 설명한 구분자 필드가 가지는 속성에 위배되므로 구분자 필드 후보에서 삭제한다. 그 다음은 각각의 구분자 필드 후보에서 각각의 메시지마다 Offset의 평균을 계산하고 평균값들에 대하여 분산 값을 계산한다. 이 분산 값이 크다면 해당 구분자 필드 후보는 메시지 내에서 매우 균등하게 나타난다는 것을 의미하고 분산 값이 작다면 메시지 내에서 어느 한쪽에 치우쳐서 나타난다는 것을 의미한다. 본 논문에서는 남아있는 구분자 필드 후보들에서 가장 큰 분산 값을 가지는 구분자 필드 후보를 해당 프로토콜의 구분자로 추출한다.

3.4 Statistics Filter

본 단계에서는 순차패턴 마이닝 기법 중 하나인 Apriori 알고리즘에 3가지 support를 적용하여 빈번하게 발생하는 문자열들을 필드 후보로써 추출한다. 그림 4에 도시한 Apriori 알고리즘은 "특정 시퀀스가 빈번하다면 그 서브시퀀스들도 역시 빈번하고 특정 시퀀스가 빈번하지 않다면 그 시퀀스를 포함하는 모든 시퀀스도 역시 빈번하지 않다"는 Apriori 속성에 기반을 둔다. 이 알고리즘은 각 level별로 접근하여 빈번하게 발생하는 length k-1 아이템으로부터 length k 아이템 후보를 생성하고 아이템이 존재하는 데이터 내에서 해당 아이템 후보가 얼마나 빈번하게 나타나는지를 나타내는 support를 계산한다. 만약 이 아이템 후보가 빈번하게 나타나지 않는다면 이 아이템으로부터 파생되는 모든 시퀀스 역시 빈번하지 않을 것이므로, 후보에서 삭제하고 만들어진 length k 아이템을 다시 length k-1 아이템으로 적용하여 length k 아이템이 만들어지지 않을 때까지 수행한다.

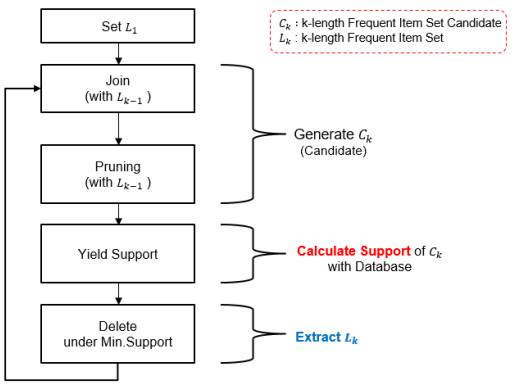

그림 4. Apriori 알고리즘 순서도
Fig. 4. Flowchart of Apriori Algorithm

본 논문에서는 Apriori 알고리즘에 support를 적용하는 방법을 개선하여 빈번하게 나타나는 문자열들을 추출함과 더불어 플로우에서 한번만 발생하는 로그인 키워드 같은 필드들을 추출한다. 그림 5는 본 논문에서 사용하는 3가지 support에 대한 설명이다.

Flow support는 해당 필드 후보가 플로우에서 발생하는 빈도를 의미한다. 간단한 예를 들면, "Login"이라는 키워드가 플로우 5개에서 모두 나타난다면 이 키워드의 Flow support는 100%(5/5)이다. Message support도 마찬가지로 해당 필드 후보가 메시지에서 발생하는 빈도를 의미한다. Server support은 해당 필드 후보가 Server_set에서 발생하는 빈도를 의미하며 Server_set은 앞서 관련연구에서 언급한 AutoReEngine에서 사용하는 Site-Specific Session set과 같은 개념이다.

본 논문에서는 전통적인 Apriori 알고리즘과는 다르게 Apriori 알고리즘에 세 가지 support를 적용한다. 전통적인 Apriori 알고리즘은 특정 아이템이 하나의 데이터셋에서 빈번하게 발생하는지를 검사하여 필터링을 하지만 본 논문에서 제안하는 방법은 세 가지 support를 새로운 방법으로 적용한다. 그림 6은 본 논문에서 Apriori 알고리즘을 기반으로 개발한 필드 추출 알고리즘의 의사 코드 이다.

본 논문에서 제안하는 방법은 우선 Server support가 50% 미만인 문자열을 필드 후보에서 삭제한다. 이 support는 특정 서버와 통신할 때만 발생하는 키워드를 제거하는 역할을 한다. 간단한 예를 들면, Server A와 통신할 때 발생하는 "abcd"라는 문자열이 Server B, C, D와 통신할 때는 발생하지 않는다면 이 문자열의 Server support는 25%이며 필드로 추출되지 않는다. 이 support를 사용하는 이유는 본 논문에서 추출하고자 하는 Static Field는 특정 서버에 종속되지 않는

$$flow_{supp} = \frac{number\ of\ flows\ containing\ item}{total\ number\ of\ flows}$$

$$message_{supp} = \frac{number\ of\ messages\ containing\ item}{total\ number\ of\ messages}$$

$$Server_{supp} = \frac{number\ of\ Server\_set\ containing\ item}{total\ number\ of\ Server\_set}$$

그림 5. 제안하는 방법의 세 가지 support 단위
Fig. 5. Three Support unit used for Proposed Method

```
Input : All Messages, All Flows, All Server_set, L1
Output : Static Field Candidates

 Lk : frequent itemset of size k
 Ck : candidate itemset of size k
01: for(k=2; Lk-1 ! = ϕ; k++)
02:    make Ck from Lk-1
03:    foreach c ∈ Ck do
04:      m=Calculate Message support
05:      f=Calculate Flow support
06:      s=Calculate Server support
07:      if(s < 50)
08:        continue
09:      if(f ≥ 98)
10:        if(m == number of flow / number of message)
11:          Lk= insert(c)
12:          continue
13:        endif
14:      endif
15:      if(m ≥65 && f ≥50)
16:        Lk= insert(c)
17:    end
18: end
```

그림 6. 필드 추출 알고리즘
Fig. 6. Field Extraction Algorithm

값이어야 하기 때문에 50%미만의 Server support를 가지는 문자열은 필드 후보에서 삭제한다.

그 다음으로는 Flow support와 Message support를 같이 적용한다. 우선 플로우에서 한번만 나타나지만 꼭 필드로 추출되어야 하는 로그인 키워드 같은 필드를 추출하기 위해서 Flow support가 98% 이상 나타나는 필드 후보를 찾는다. 그 후에 해당 필드 후보의 Message support가 플로우 개수만큼 나타나는지를 검사한다. 플로우에서 한번 씩만 나타는 필드라면 플로우 개수만큼 발생할 것이고 Message support가 플로우 개수와 동일하므로 이 알고리즘으로 플로우에서 한번 나타나는 필드를 추출할 수 있다.

마지막으로는 Flow support와 Message support를 동시에 만족하는 문자열을 필드 후보로 추출한다. 본 논문에서 추출하고자 하는 Static Field는 해당 프로토콜의 특징을 잘 나타내는 필드이기 때문에 메시지와 플로우 모두에서 빈번하게 나타나야 한다. 따라서 Flow support와 Message support를 동시에 적용하여 Flow와 메시지 모두에서 빈번하게 나타나는 문자열을 필드 후보로 추출한다.

3.5 Position-value Filter

본 단계에서는 앞 단계의 출력인 필드 후보들의 위치정보를 사용해서 메시지 내에서 비교적 고정적인 위치에 발생하는 필드 후보들을 최종 Static Field로 추출

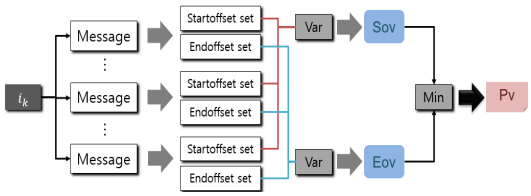

그림 7. Position-value Filter 단계 개요
Fig. 7. Overview of Position-value Filter Stage

한다. 그림 7은 Position-vlaue Filter를 요약한 그림이다.

모든 k길이의 필드 후보 i_k에 대하여 메시지마다 Startoffset set과 Endoffset set을 구성한다. Startoffset set은 메시지 시작 부분을 기준으로 한 필드 후보가 나타나는 메시지 내 offset 집합이며 Endoffset set은 메시지 끝부분을 기준으로 한 필드 후보가 나타나는 메시지 내 offset 집합이다. Startoffset set과 Endoffset set을 구한 다음 필드 후보가 나타나는 모든 메시지에 대하여 Startoffset set의 원소들만의 분산 값(Sov)을 구하고 Endoffset set의 원소들만의 분산 값(Eov)을 구한다. Sov라는 값은 해당 필드 후보가 메시지 시작부분을 기준으로 얼마나 고정적인 위치에 발생하는지 나타내는 지표이며 Eov라는 값은 해당 필드 후보가 메시지 끝 부분을 기준으로 얼마나 고정적인 위치에 발생하는지 나타내는 지표이다. Sov와 Eov중에 최솟값을 Pv 값으로 선정한다. Pv라는 값은 해당 필드 후보가 메시지 내에서 어떻게 분포되어 있는지 나타나는 최종 평가지표이며 Pv가 작으면 작을수록 해당 필드 후보는 메시지 내에서 매우 고정적인 위치에 나타난다는 것을 의미한다. 따라서 Pv가 특정 Threshold 미만의 값을 가진다면 해당 필드 후보를 최종 Static Field로 추출한다.

Ⅳ. 실 험

본 장에서는 본 논문에서 제안한 방법론의 타당성을 검증하는 실험 및 결과를 설명한다. 비공개 프로토콜을 실험에 사용한다면 정확한 검증 결과를 도출할 수 있지만 비공개 프로토콜을 수집하는 것은 현실적으로 불가능하기 때문에 가독성이 좋고 현재 상용화되어있는 Text 기반 프로토콜인 HTTP 프로토콜에 대하여 두 가지 트래픽 셋을 구성하고 실험을 진행하였다. 표 1은 실험에 사용한 트래픽의 정보이다.

또한 본 논문에서 제안한 방법론이 얼마나 Static Field를 잘 추출했는지 정량적으로 평가하기 위해서 3

표 1. 실험 트래픽 정보
Table 1. Traffic Information

Traffic Info	Flow	Pkt	Message	Byte
Set1	359	3841	1189	4674813
Set2	216	3391	859	4298405

가지 평가지표[17]를 정의하였다. 4가지 평가지표는 ConcisenessTF, ConcisenessEF, Correctness이다. 평가지표를 계산하기 위해 HTTP 프로토콜에 대한 정답지를 구성하였다. HTTP 프로토콜에서 Static Field로 추출해야 할 부분으로 Method, Version, Status code, Phrase, Header Name들을 정답이라고 정의하고 이를 True Field(f)라고 한다. 예를 들면, GET, HTTP/1.1 등이 있다. 그리고 True Field를 클러스터링 한 것을 True Field Format(TF)이라고 정의한다. 또한 본 방법론에서 추출해 낸 Static Field는 Extracted Field(EF)라고 정의한다.

4.1 평가지표

그림 8은 평가지표를 계산하기 위해 필요한 용어들을 설명하기 위한 그림이다. TFE는 앞서 정의한 TF중에서 본방법론에서 추출해 낸 EF에 포함되어 있는 TF를 의미한다. TFN은 TF중에서 EF에 포함되지 않는 TFF를 의미한다.

간단한 예를 들면, Content-Type, HTTP/1.1이라는 TFF가 있고 |0d||0a|Content-Type이라는 본 방법론에서 추출해 낸 EF가 있을때 Content-Type은 EF에 포함되는 TF 이므로 TFE가 되고 HTTP/1.1은 EF에 포함되지 않으므로 TFN이 된다. EFT는 EF중에서 TF를 포함하고 있는 EF를 의미하며 EFV는 EF중에서 TF를 포함하고 있지 않는 EF를 의미한다. True Field는 2가지로 나눌 수 있는데 TFE에 속하는 True Field를 f_e, TFN에 속하는 True Field를 f_n이라고 정의한다. 아래 수식은 각 평가지표에 대한 수식이다.

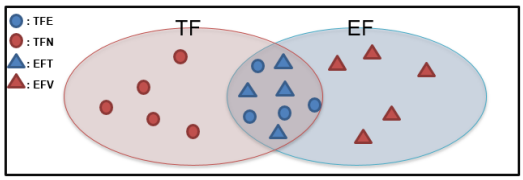

그림 8. 평가지표 용어의 개념
Fig. 8. Concept of Terminology used for Evaluation Metrics

$$Conciseness\,TF = \frac{n(TFE)}{n(TF)} \quad (1)$$

$$Conciseness\,EF = \frac{n(EFT)}{n(EF)} \quad (2)$$

$$Correctness = \frac{n(f_e)}{n(f)} \quad (3)$$

ConcisenessTF는 TF들 중 추출한 TFE의 비율을 의미한다. 이 평가지표는 입력으로 사용된 트래픽에서 추출할 수 있는 모든 정답 중에서 얼마나 많은 정답을 추출했는지 평가하는 지표이다.

ConcisenessEF는 EF들 중 정답인 EFT의 비율을 의미한다. 이 평가지표는 추출한 필드들 중에서 얼마나 많은 정답을 추출했는지 평가하는 지표이다.

Correctness는 입력으로 사용된 트래픽에서 추출할 수 있는 모든 True Field 중에서 얼마나 많은 True Field를 추출했는지 평가하는 지표이다.

4.2 실험 결과

표 2는 Extract Delimiter 단계에서 구분자 필드를 추출하기 위해 필요한 Threshold를 찾기 위해서 HTTP set1 트래픽에 대하여 실험한 결과이다. HTTP 프로토콜은 |0d||0a|가 구분자로 추출되어야하기 때문에 발생 횟수에 대한 분산 값의 Threshold는 14, Offset 평균의 분산 값 Threshold는 0.2로 설정하고 HTTP set2 트래픽에 적용해서 실험을 진행하였다. 표 3을 보면 |0d||0a|

표 2. HTTP Traffic set1 실험 결과
Table 2. HTTP Traffic set1 Experiment Result

	|0d||0a|	space	;	=	:
variance of Occurrence count	14.891	471.503	53.038	81.960	120.071
variance of Offset Average	0.219	0.119	0.113	0.096	0.070

표 3. HTTP Traffic set2 실험 결과
Table 3. HTTP Traffic set2 Experiment Result

	|0d||0a|	space	;	=	:
variance of Occurrence count	5.863	1077.381	18.891	50.275	95.819
variance of Offset Average	0.286	0.141	0.136	0.112	0.084

표 4 TF와 EF
Table 4. TF and EF

TF	
HTTP/1.1	HTTP/1.1|0d||0a|Host:
Date	|0d||0a|Date:Tue, 10 Apr 2018 07:0
GET	GET /
200	|0d||0a|Server:
Server	|0d||0a|Content-Type:
Host	|0d||0a|Referer:http://
OK	HTTP/1.1 200 OK|0d||0a|

가 구분자로 잘 추출된 것을 볼 수 있다.

표 4는 본 논문에서 정답으로 설정한 TF와 추출한 EF 중에서 support가 높은 순으로 정리한 결과이다.

그림 9는 실험 결과에 대한 평가지표를 나타낸 그래프이다. 그래프에 나타난 바와 같이 ConcisenessTF는 약 25% 정도의 수치를, CocisenessEF와 Correctness는 각각 88%, 79%의 준수한 수치를 보였다. 결과를 분석해 보면 본 논문에서 제안하는 방법론으로 추출한 필드들 중의 약 88%는 정답이며, 찾아야할 필드들 중 약 80% 가까이 추출해 냈다는 것을 알 수 있다. 하지만 통계적 기반 분석 방법이기 때문에 낮은 빈도로 발생하는 필드들을 추출하지 못해서 추출해야 할 정답을 약 25% 정도 밖에 추출하지 못하였다. 하지만 미리 정의한 구분자 필드 5개(|0d||0a|, space, ;, =, :) 중에 HTTP 프로토콜에서 구분자로 판단되는 |0d||0a|를 성공적으로 추출했기 때문에 추출된 구분자 필드를 사용하여 좀 더 정교하게 필드를 추출한다면 성능을 대폭 개선될 여지가 있다고 판단된다.

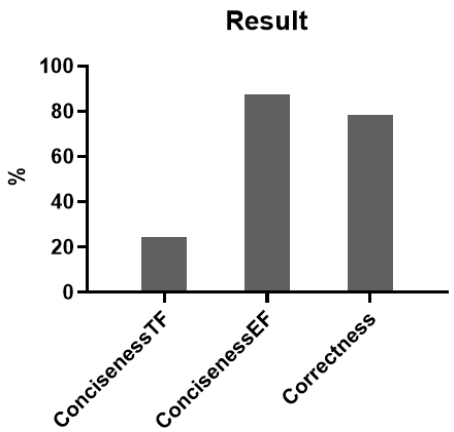

그림 9. 실험 결과
Fig. 9. Experimental Results

V. 결 론

본 논문에서는 미리 정의한 구분자 필드 후보들에 대하여 구분자 필드의 존재여부를 우선적으로 검사해서 구분자 필드를 추출하고 구분자 필드가 존재하지 않더라도 통계적 분석 방법인 Statistics Filter, Position-value Filter 단계를 수행해서 정교하게 Static Field를 추출하는 방법을 제안하였다. 비공개 프로토콜 트래픽을 수집할 수 없기에 현재 상용화되어있고 가독성이 좋은 HTTP 프로토콜에 대하여 실험을 진행하였고, 결과로써 구분자 필드와 Static Field를 잘 추출하였다. 하지만 본 방법론은 암호화된 프로토콜에 대해서는 분석을 할 수 없고 입력 트래픽에 대해서 모든 정답을 추출하지 못하는 한계점이 있다. 향후 연구로써 추출한 구분자 필드를 사용하여 좀 더 정교하게 필드와 Sub-Delimiter를 추출하는 연구를 진행할 계획이다. 추가적으로, 본 연구를 토대로 더 많은 종류의 프로토콜 트래픽에 대하여 실험을 진행해서 최적의 Threshold 값을 찾는 연구를 진행할 계획이다.

References

[1] K.-S. Shim, Y.-H. Goo, S.-H. Lee, B. D. Sija, and M.-S. Kim, “Automatic payload signature update system for classification of recent network applications,” *J. Inst. Commun. and Inf. Sci.*, vol. 42, no. 1, pp. 1-10, Jan. 2017.

[2] Y.-H. Goo, S.-O. Choi, S.-K. Lee, S.-M. Kim, and M.-S. Kim, “A method for tracking the source of cascading cyber attack traffic using network traffic analysis,” *J. Inst. Commun. and Inf. Sci.*, vol. 41, no. 12, pp. 1-9, Dec. 2016.

[3] J. Narayan, S. K. Shukla, and T. C. Clancy, “A survey of automatic protocol reverse engineering tools,” *J. ACM Comput. Survey*, vol. 48, no. 40, 2016.

[4] N. Borisov, D. J. Nrumley, H. J. Wang, and C. Guo, “Generic application-level protocol analyzer and its language,” in *Network and Distrib. Syst. Secur. Symp.*(NDSS), San Diego, CA, Feb. 2007.

[5] A. Tridgell, *How Samba Was Written*(2003), Retrieved Aug. 17, 2017, from https://www.samba.org/ftp/tridge/misc/french_cafe.txt

[6] Y.-H. Goo, K.-S. Shim, J.-T. Park, B.-M.

Chae, H.-W. Moon, and M.-S. Kim, "A method of protocol reverse engineering for clear protocol specification extraction," *KNOM Rev.*, vol. 20, no. 2, pp. 11-23, Dec. 2017.

[7] J. Caballero, H. Yin, Z. Liang, and D. Song, "Polyglot: Automatic extraction of protocol message format using dynamic binary analysis," in *Proc. 14th Acm Conf. Computer and Commun. Secur.*(CCS), pp. 317-329, Alexandria, Virginia, USA, 2007.

[8] W. Cui, M. Peinado, K. Chen, H. J. Wang, and L. Irun-Briz, "Tupni: Automatic reverse engineering of input formats," in *Proc. 15th ACM Conf. Computer and Commun. Secur.* (CCS), pp. 391-402, Alexandria, Virginia, USA, Oct. 2008.

[9] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda, "Prospex: Protocol specification extraction," in *Proc. 30th IEEE Symp. Secur. and Privacy*(S&P), pp. 110-125, May 2009.

[10] J. Caballero and D. Song, "Automatic protocol reverse-engineering: Message format extraction and field semantics inference," *Int. J. Comput. and Telecommun. Netw.*, vol. 57, no. 2, pp. 451-474, 2013.

[11] M. Beddoe, *The Protocol Informatics Project* (2004) Retrieved Jun. 4, 2017, from http://www.4tphi.net/~awalters/PI/PI.html

[12] M. Shevertalov and S. Mancoridis, "A reverse engineering tool for extracting protocols of networked applications," in *14th Working Conf. on Reverse Eng.*(WCRE'07), pp. 229-238, DOI:10.1109/WCRE. Jun. 2007.

[13] Y. Wang, Z. Zhang, D. Yao, B. Qu, and L Guo, "Inferring protocol state machine from network traces," in *Proc. 9th Int. Conf. Applied Criptograph and Netw. Secur.*(ANCS), 2011.

[14] C. Leita, K. Mermoud, and M. Dacier, "Automatic protocol field inference for deeper protocol understanding," *IFIP Netw. Conf.*, pp. 1-5, Toulouse, France, May 2015.

[15] J.-Z. Luo and S.-Z. Yu, "Position-based automatic reverse engineering of network protocols," *J. Netw. and Comput. Appl.*, vol. 36, no. 3, pp. 1070-1077, 2013.

[16] I. Bermudez, A. Tongaonkar, M. Iliofotou, M. Mellia, and M. M. Munafo, "Automatic protocol field inference for deeper protocol understanding," in *IFIP Netw. 2015*, pp. 1-9, Toulouse, France, May 2015.

[17] Y.-H. Goo, K.-S. Shim, M.-S. Lee, J.-T. Park, and M.-S. Kim, "A study of evaluation and validation method for protocol reverse engineering," in *Proc. KICS Winter Conf.*, pp. 746-747, Jeju, Korea, Jan. 2018.

이 민 섭 (Min-Seob Lee)

2018년 : 고려대학교 컴퓨터정보학과 학사

2018년~현재 : 고려대학교 컴퓨터정보학과 석사과정

<관심분야> 네트워크 관리 및 보안, 트래픽 모니터링 및 분석

[ORCID:0000-0003-4854-9521]

심 규 석 (Kyu-Seok Shim)

2014년 : 고려대학교, 컴퓨터정보학과 학사과정

2016년 : 고려대학교 컴퓨터정보학과 석사과정

2016년~현재 : 고려대학교 컴퓨터정보학과 박사과정

<관심분야> 네트워크 관리 및 보안, 트래픽 모니터링 및 분석

[ORCID:0000-0002-3317-7000]

구 영 훈 (Young-Hoon Goo)

2016년 : 고려대학교 컴퓨터정보학과 학사
2016년~현재 : 고려대학교 컴퓨터정보학과 석/박사통합과정
<관심분야> 네트워크 관리 및 보안, 트래픽 모니터링 및 분석
[ORCID:0000-0002-3013-7011]

백 의 준 (Ui-Jun Baek)

2018년 : 고려대학교 컴퓨터정보학과 학사
2018년 ~현재 : 고려대학교 컴퓨터정보학과 석사과정
<관심분야> 네트워크 관리 및 보안, 트래픽 모니터링 및 분석
[ORCID:0000-0002-4358-7839]

채 병 민 (Byeong-Min Chae)

2007년 : 충남대학교 물리학과 학사
2012년 : 충남대학교 컴퓨터공학과 석사
2007년~2008년 : 삼성전자 연구원
2008년~현재 : 한화시스템 연구원
<관심분야> 네트워크 관리 및 보안, 트래픽 모니터링 및 분석
[ORCID:0000-0003-3299-6693]

문 호 원 (Ho-Won Moon)

2000년 : 한양대학교 수학과 학사
2000년~2011년 : 삼성전자 연구원
2011년~현재 : 한화시스템 연구원
<관심분야> 네트워크 관리 및 보안, 트래픽 모니터링 및 분석, 라우팅
[ORCID:0000-0002-5668-7265]

김 명 섭 (Myung-Sup Kim)

1998년 : 포항공과대학교 전자계산 학과 학사
2000년 : 포항공과대학교 전자계산 학과 석사
2004년 : 포항공과대학교 전자계산 학과 박사
2006년 : Dept. of ECS, Univ of Toronto Canada
2006년~현재 : 고려대학교 컴퓨터정보학과 교수
<관심분야> 네트워크 관리 및 보안, 트래픽 모니터링 및 분석, 멀티미디어 네트워크
[ORCID:0000-0002-3809-2057]