KNOM Review '17-02 Vol.20 No.02
https://doi.org/10.22670/knom.2017.20.2.11

명확한 프로토콜 사양 추출을 위한 프로토콜 리버스 엔지니어링 방법

구 영 훈◆, 심 규 석*, 박 지 태*, 채 병 민**, 문 호 원**, 김 명 섭°

A Method of Protocol Reverse Engineering for Clear Protocol Specification Extraction

Young-Hoon Goo◆, Kyu-Seok Shim*, Jee-Tae Park*, Byeong-Min Chae**, Ho-Won Moon**, Myung-Sup Kim°

요 약

인터넷 트래픽 발생량이 비대해지고, 새로운 응용 및 악성 행위가 지속적으로 출현함에 따라 이를 분석해야할 트래픽이 급증하고 있다. 이러한 환경 하에 발생하는 복잡 다양한 프로토콜 중 다수는 알려지지 않았거나 최소한으로 문서화되어 있는 비공개 프로토콜이다. 효율적인 네트워크 관리 및 보안을 위해 비공개 프로토콜의 구조 분석은 반드시 선행되어야 한다. 이를 위해 많은 프로토콜 리버스 엔지니어링 방법론이 제안되었지만, 적용하기에 각기 다른 단점이 존재한다. 이러한 단점을 해결하기 위해 본 논문에서는 네트워크 트레이스 분석 기반의 명확한 프로토콜 리버스 엔지니어링 시스템을 위한 방법론을 제안한다. 제안하는 방법론은 비공개 프로토콜의 직관적이며 정교한 Field Format, Message Format, Flow Format과 유한 상태 머신을 출력할 수 있다. 본 논문에서는 HTTP 프로토콜의 예를 통해 제안하는 방법론의 우수성을 설명한다.

Key Words : Protocol reverse engineering, Field Format, Message Format, Flow Format, Protocol State Machine, CSP Algorithm

ABSTRACT

Today, the amount of Internet traffic is increasing and new applications and malicious behavior continue to appear rapidly. Many of the complex and diverse protocols that occur under these situations, are unknown or proprietary protocols that are at least documented. For efficient network management and network security, analysis the structure of the unknown protocol must precede. Many protocol reverse engineering methods have been proposed for this purpose, but each of method has its own limitations. To address these drawbacks, in this paper proposes a method for clear protocol reverse engineering based on network traces analysis. The proposed method can extract highly intuitive and elaborative outputs which are Field Format, Message Format, Flow Format, and protocol state machine of the unknown protocol. We demonstrate the superiority of the method through an example of the HTTP protocol.

※이 논문은 2016년도 국방과학연구소와 한화시스템(주)의 재원을 받아 수행된 연구임(UE161105ED).
◆ First Author : Department of Computer and Information Science, Korea University, gyh0808@korea.ac.kr
° Corresponding Author : Department of Computer and Information Science, Korea University, tmskim@korea.ac.kr
* Department of Computer and Information Science, Korea University, {kusuk007, pjj5846}@korea.ac.kr
** Hanwha Systems, {byeongmin.chae, moon1000}@hanwha.com
논문번호 : KNOM2017-02-004, Received November 23, 2017; Revised December 5, 2017; Accepted December 10, 2017

Ⅰ. 서 론

IT 기술 발전으로 인터넷 트래픽 사용이 보편화, 다양화되고 인터넷 응용의 트래픽 발생량이 비대해짐에 따라 네트워크를 이용하는 응용 및 악성행위의 수도 급격히 증가하고 있다[1,2]. 이러한 환경 하에 발생하는 프로토콜 중 다수는 Skype 프로토콜, 바이러스 백신 도구가 사용하는 소프트웨어 업데이트 프로토콜과 같은 독점적인 프로토콜이거나 SCADA(Supervisory Control and Data Acquisition) 시스템의 프로토콜, Botnet의 C&C(Command and Contol) 프로토콜 등과 같은 최소한으로 문서화되어 있거나 전혀 알 수 없는 프로토콜이다[3,4]. 한편, 최근 3.20 사이버테러, 서울매트로 PC 해킹사고, 한수원 사이버테러 등 크고 작은 보안 사고들이 꾸준히 발생함에 따라 사이버 테러는 새로운 공동 관심사로 부상하고 있으며 주요 언론은 사이버범죄 피해가 2021년까지 미화 6조 달러에 달할 것으로 전망하고 있다[5,6]. 오늘날 비공개 프로토콜에 대한 구조분석 기술 확보가 시급한 시점이다.

비공개 네트워크 프로토콜의 사양을 추출하는 작업인 프로토콜 리버스 엔지니어링은 효율적인 네트워크 관리 및 보안 문제를 해결하기 위해 반드시 선행되어야 한다. 예를 들어, 비공개 프로토콜이 발생시키는 트래픽을 분류하여 신뢰성 있는 네트워크 사용 현황 파악, 확장 계획 수립 및 QoS 정책 설정에 활용이 가능하며 공격의 탐지 및 차단을 위한 방화벽 및 침입 탐지 시스템과 네트워크 취약성을 파악하기 위한 침투 시험 및 스마트 퍼저 시스템 구축에 유용한 정보를 제공할 수 있다.

기존의 다양한 연구에서 프로토콜 리버스 엔지니어링의 방법론을 제안하였지만, 프로토콜의 사양을 명확하게 추출하는 표준화된 방법론은 없으며, 각각의 방법에는 몇 가지 한계점이 존재한다. 전통적인 프로토콜 리버스 엔지니어링은 대부분 수동으로 수행되므로 시간이 오래 걸리며 오류가 발생하기 쉽다. 이를 해결하기 위한 많은 자동 프로토콜 리버스 엔지니어링 방법이 제안되었으나, 이 중 일부는 너무 많은 메시지 유형들을 출력하여 명확한 프로토콜 사양을 파악하기 어려우며, 일부는 프로토콜의 빈번한 값만을 메시지 유형들의 필드 형식으로 추출하기 때문에 완벽한 프로토콜 사양을 파악하기 어렵다. 또한, 대부분의 방법론들은 프로토콜 리버스 엔지니어링의 출력 중 일부만을 추출한다. 이상적인 프로토콜 리버스 엔지니어링은 프로토콜 syntax, semantics, FSM을 포함하여 명확한 프로토콜 구조를 추출하여야한다.

본 논문에서는 명확한 프로토콜 사양 추출을 위한 프로토콜 리버스 엔지니어링 방법을 제안한다. 본 방법론은 syntax를 Field Format, Message Format, Flow Format으로 구성하고 semantics를 추론하여 추출할 수 있으며 이를 기반으로 프로토콜 FSM을 추출한다. 이를 위해 본 연구진이 개발한 CSP(Contiguous Sequential Pattern) 알고리즘을 계층적으로 사용하여 syntax를 추출하며 재귀적으로 사용하여 하나의 필드 형식이 가질 수 있는 모든 값을 추출한다. 또한 Message Format을 구성하는 Field Format의 타입을 4종류로 정의하고 이를 통해 Message Format을 상세한 구조로 추출한다.

본 논문의 구성은 본 장의 서론에 이어 2장에서 관련 연구 및 해결하고자 하는 문제를 정의하고 3장에서 제안하는 프로토콜 리버스 엔지니어링 방법론에 대해 상세히 기술한다. 마지막으로 4장에서 결론 및 향후 연구를 기술하며 끝맺음한다.

Ⅱ. 관련 연구 및 문제 정의

2.1. 프로토콜 리버스 엔지니어링

프로토콜 리버스 엔지니어링은 알 수 없거나 문서화되지 않은 네트워크 프로토콜 및 파일 형식의 사양을 추출하는 것으로 일반적으로 OSI 7계층에서 알려지지 않은 응용 계층 프로토콜의 사양을 도출하는 과정을 말한다. 프로토콜 리버스 엔지니어링의 목표는 프로토콜을 구성하는 주요 3요소인 구문, 의미, 타이밍과 관련된 syntax, semantics, FSM 등을 포함한 상세한 구조를 추출하는 것으로 해당 비공개 프로토콜이 어떠한 유형의 메시지들을 갖고 있는지, 이러한 메시지 유형들은 어떠한 형식으로 구성되어 있으며, 어떠한 순서로 동작하는지를 나타낸다. 이를 위하여 네트워크를 통한 프로토콜의 메시지 교환을 모니터링하여 분석하거나 통신 종단점이 프로토콜의 메시지를 처리하는 방식을 분석할 수 있다.

2.2. 선행 연구의 분류 및 한계점

전통적인 프로토콜 리버스 엔지니어링 방법의 전통적인 접근 방식은 대개 수동적이다. 그 전형적인 예로 2005년 Borisov가 발표한 GAPA(Generic Application-level Protocol Analyzer)를 들 수 있다[7]. GAPA는 손으로 작성한 문법을 통해 네트워크 프로토콜의 사양을 확인하고 파싱할 수 있는 프레임워크이다. 그러나 수동적 프로토콜 리버스 엔지니어링은 모든 프로토콜의 요소를 정확하게 복구할 수 있지만 프로토콜 분석자의 작업 능력에 따라 출력 결과가 가변적이며 오류가 발생하기 쉽고 시간이 굉장히 많이 소모되는 비효율인 형태이므로 기하급수적으로 증가하는 응용의 속도에 대처할 수 없다. SAMBA 프로젝트가 Microsoft SMB(Server Message Block) 프로토콜을 보완 및 완성하는 데에는 12년이 걸렸으며[8], WINE(Wine Is Not an Emulator) 프로젝트의 경우, 처음으로 안정화된 API를 출시하는 데 15년이 걸렸다. 오늘날과 같은 고속의 대용량 네트워크 환경에서 고도로 지능화된 다양한 악성행위에 대처하기 위해서는 비공개 프로토콜 리버스 엔지니어링 기술의 자동화가 필수적이다.

자동 프로토콜 리버스 엔지니어링은 일반적으로 그림 1과 같이 실행 트레이스 분석 기반 방법 및 네트워크 트레이스 분석 기반 방법으로 나눌 수 있다.

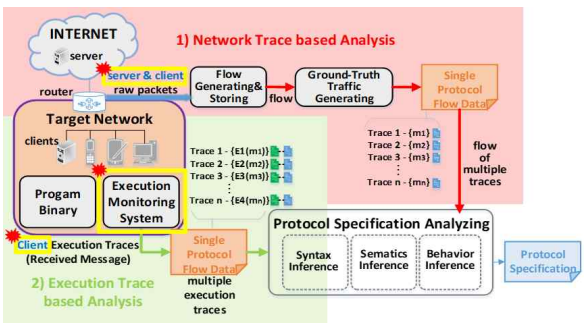

그림 1. 네트워크 트레이스 분석 기반 방법과 실행 트레이스 분석 기반 방법의 차이
Fig. 1. The difference of network traces analysis based method and execution traces analysis based method

실행 트레이스 분석 기반 방법은 Dynamic Taint Analysis를 통해 해당 프로토콜을 따르는 프로그램 바이너리를 모니터링하여 실행 명령, 메모리 사용, 시스템 콜, 특정 파일 시스템 접근 등을 기반으로 로깅한 각 실행 트레이스들을 입력으로 분석하는 방법이다. 선행 연구의 실행 트레이스 분석 기반 방법으로는 Caballero et al.이 제안한 Polyglot[9], Cui et al.의 Tupni[10], Comparetti et al.의 Prospex[11], Caballero et al.의 Dispatcher[12]가 있다. 이 방법은 실제 프로그램 바이너리가 실행되는 동안의 메시지 처리 과정을 분석하므로 프로토콜 구조 분석의 정확성은 향상될 수 있지만 비공개 프로토콜을 구현하는 프로그램 바이너리의 입수는 현실적으로 어렵다. 예를 들어 악성 Botnet들의 C&C 서버의 프로그램 바이너리는 외부 네트워크에 존재할 가능성이 크며 지속적인 공격의 성공을 위해 은닉하고 있기 때문이다. 또 다른 단점은 일반적으로 입력 메시지를 처리하는 동안 클라이언트의 프로그램 바이너리를 관찰하여 분석하기 때문에 수신된 메시지만을 분석한다. 그러나 완벽한 프로토콜의 구조 분석을 위해서는 송신 메시지 또한 분석하여야 하지만 대부분의 경우 서버의 프로그램 바이너리는 외부 네트워크에 존재하므로 현실적으로 불가능하다.

반면에 네트워크 트레이스 분석 기반 방법은 해당 프로토콜의 네트워크 패킷을 모니터링하여 캡처한 각 네트워크 트레이스들을 입력으로 분석하는 방법이다. 따라서 프로그램 바이너리를 실행하고 있는 호스트에 접근할 수 없는 환경에서도 분석이 가능하여 실용적이며, 타겟 네트워크와 외부 네트워크를 연결하는 최 앞단 라우터에서 발생하는 트래픽을 캡처하여 클라이언트와 서버 간의 송·수신 메시지를 모두 분석할 수 있다. 또한, 패킷 수집 및 단일 프로토콜별 트래픽 분류의 자동화가 가능하기 때문에 확장성이 있다. 선행 연구의 네트워크 트레이스 분석 기반 방법으로는 Beddoe et al.이 제안한 PIP[13], Maxim et al.의 Pext[14], Wang et al.의 Veritas[15], Bernudez et al.의 FieldHunter[16]가 있다. Leita et al.이 제안한 ScriptGen[17]과 Cui et al.의 RolePlayer[18]는 HoneyPot 시스템을 위한 패킷 Replay에 초점을 두었다. 본 논문에서 제안하는 방법론은 분석의 실용성 및 확장성 측면에서 유리한 네트워크 트레이스 분석 기반 방법을 사용한다.

2.3. 문제 정의

프로토콜 리버스 엔지니어링 엔지니어링의 출력은 크게 syntax, semantics, FSM 세 가지로 구성된다. syntax는 각 유형별 메시지의 형식을 나타낸다. 각 유형별 메시지의 형식은 이를 구성하는 필드들을 구분할 수 있는 구분자, offset 및 depth로 표현할 수 있는 위치 및 경계와 필드들의 순서 및 가지고 있는 값, 필드의 유형 등이 포함된다. 분석자의 편의에 따라 메시지 유형의 방향성, 구성하는 특정

필드 값의 빈도수와 같은 부수적인 정보를 추가할 수 있다. semantics는 메시지 유형을 구성하는 필드들이 내포하는 의미를 나타낸다. 선행 연구 방법의 대부분은 semantics를 추론하는 방법으로 자주 사용되는 필드의 semantics 유형들을 사전에 정의하고 이에 해당하는 필드들을 휴리스틱한 방법으로 식별하거나 특정 의미를 추출하지 않고 필드들의 의존성을 분석하여 필드들 간의 관계만을 추출한다. FSM은 메시지 유형들의 동작을 분석하고 발생 순서, 발생 조건, 메시지의 방향 등을 표현하기 위한 유한 오토마톤이다.

표 1. 입·출력에 따른 선행 연구의 방법론
Table 1. Previous Method according to input and output

Method	Year	Output		
		syntax	semantics	FSM
Execution Traces analysis based Method				
Polyglot	2003	○		
Tupni	2008	○		
Dispatcher	2012	○	○(weak)	
Prospex	2009			○
AutoFormat	2008	○		
ReFormat	2009	○		
Network Traces analysis based Method				
PIP	2004	○		
Biprominer	2011	○		
Discoverer	2007	○	○(weak)	
ScriptGen	2005			○
RolePlayer	2006	○	○(weak)	
ReverX	2011	○		○
AutoReEngine	2013	○		○
Pext	2007			○
Trifilo	2009			○
Veritas	2011	○		○
FieldHunter	2015	○	○(strong)	
Netzob	2014	○	○(weak)	○(manual)
Wasp	2016	○	○(weak)	

이상적인 프로토콜 리버스 엔지니어링을 위해서는 syntax, semantics, FSM 등을 포함하여 가능한 모든 정보를 추론하여야 하며, 정확한 프로토콜의 구조 분석을 위해 추출되는 syntax는 명확해야 한다. 그러나 선행 연구의 방법론에는 몇 가지 한계점이 존재한다.

첫째, 대부분의 선행 연구는 syntax, semantics, FSM 중 일부만을 추출한다. 표 1은 입·출력에 따른 선행 연구의 방법론 현황을 보여준다.

둘째, 선행 연구의 많은 방법론은 너무 많은 메시지 유형들을 추출하여 명확하지 않은 syntax를 출력한다. 이러한 유형의 syntax는 프로토콜의 각 개별 패킷을 상세하게 분석하는데 유용할 수 있으나 프로토콜의 구조를 파악하는데 있어서는 효과적이지 않다. 그림 2는 선행 방법론 중 하나를 사용하여 HTTP 프로토콜을 분석한 결과로 Group1과 Group2와 Group3는 같은 HTTP Request 메시지 유형이며 Group4와 Group5는 같은 HTTP Response 메시지 유형이며 Group4와 Group5는 같은 HTTP Response 메시지 유형임에도 불구하고 서로 다른 Group으로 분류되었다. 그림 2에서 빨간 줄은 값이

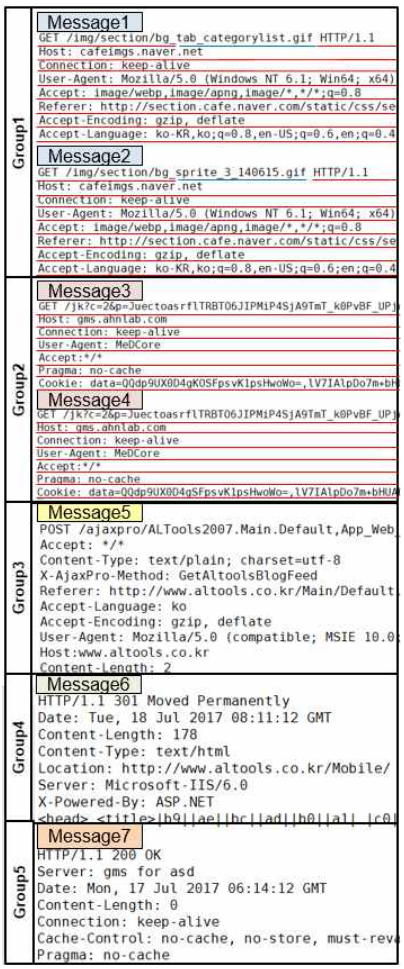

그림 2. 명확하지 않은 syntax 출력의 예시1
Fig. 2. The example1 of an unclear syntax as output

정적인 필드를 의미하며 파란 줄은 값이 동적인 필드를 의미한다. Group1, Group2와 같이 실제로 같은 필드임에도 불구하고 각 Group 내에서만 필드의 값이 정적인 필드와 동적인 필드를 분류하기 때문에 명확한 프로토콜 syntax 파악이 어렵다. 1000개의 HTTP 프로토콜 패킷을 입력으로 하여 Netzob

을 통해 프로토콜 리버스 엔지니어링을 수행하였을 경우, Similarity Percent를 50%로 설정하였을 때에는 324개의 메시지 유형이 출력되었으며, 25%로 설정하였을 때에는 225개의 메시지 유형이 출력되었다. 본 방법론은 이러한 문제를 해결하기 위해 Field Format, Message Format, Flow Format의 세 가지 형태의 syntax를 정의하고 본 연구진이 개발한 Hierarchical CSP 알고리즘을 이용하여 이를 추출한다.

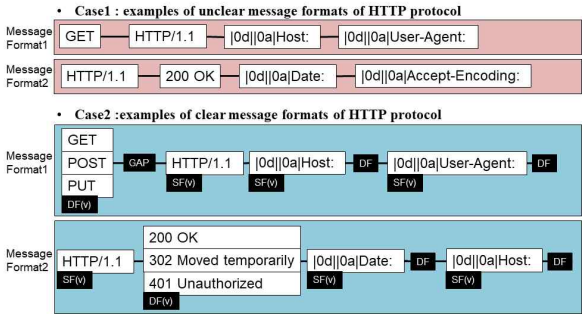

그림 3. 명확하지 않은 syntax 출력의 예시2
Fig. 3. The example2 of an unclear syntax as output

셋째, 선행 연구의 많은 방법론은 대부분 Zipf 법칙과 샤논 이론의 엔트로피 필터, 연관 규칙 마이닝의 Support, LDA의 출현 확률과 같이 특정 데이터의 빈도를 기반으로 필드의 값을 추출하기 때문에 그림 3의 case1과 같이 각 필드가 가질 수 있는 모든 값을 추출하지 않고 가장 빈번한 값만을 해당 필드의 값으로 추출한다. 제안하는 방법론은 본 연구진이 개발한 Recursive CSP 알고리즘을 이용하여 그림 3의 case2와 같이 각 필드가 가질 수 있는 모든 값을 추출한다.

추가적으로 제안하는 방법론은 메시지의 유형별로 값이 정적인 필드, 동적인 필드로서 추출되지 않은 데이터들을 분석하여 새로운 유형의 Field Format으로 분류하며 이를 통해 상세한 메시지 유형의 형식을 추출한다.

Ⅲ. 제안하는 프로토콜 리버스 엔지니어링 방법론

3.1. 용어 정의 및 방법론의 개관

Field Format은 SF(v), DF(v), DF, GAP 네 가지 유형으로 구성된다. (v)는 “value”를 의미하며 (v)가 표시된 Field Format은 해당 필드의 값을 예측할 수 있는 Field Format이다. 즉, SF(v)는 값이 정적인 필드이며 고정 길이 필드이고 값을 예측할 수 있으므로 값을 저장하여 Field Format을 출력한다. DF(v), DF, GAP은 모두 값이 동적인 필드이다. 이들은 고정 길이 필드일 수도 있고, 가변 길이 필드일 수도 있다. DF(v)는 값을 예측할 수 있는 필드이므로 가질 수 있는 모든 값을 저장하여 출력한다. DF와 GAP은 값이 너무 가변적이기 때문에 값을 예측할 수 없는 필드이다. DF와 GAP의 차이점은 DF의 경우 길이는 고정되어 있지 않더라도 어느 정도 예측이 가능하다는 점이다. 따라서 DF를 추출할 때에는 길이의 최소값 및 최대값을 추출하며 GAP은 발견될 수 있는 최소 offset과 최대 depth를 통해 메시지 유형에서의 위치만을 추출한다. SF(v)

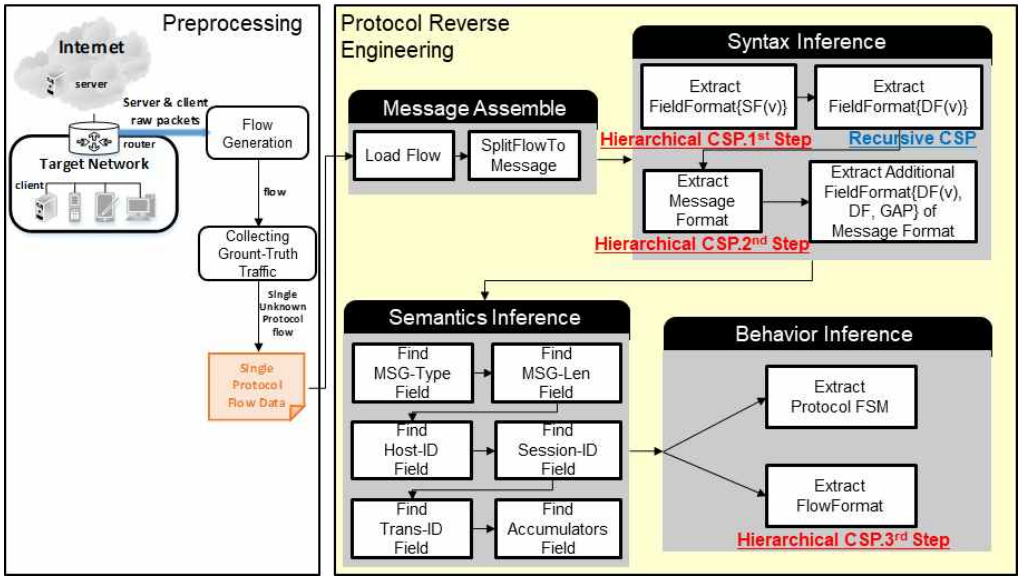

그림 4. 제안하는 방법론의 체계도
Fig. 4. The overview of proposed method

는 Hierarchical CSP 알고리즘의 1st step을 통해 추출되고, DF(v)는 Recursive CSP 알고리즘을 통해 추출된다. DF와 GAP은 Message Format 추출 후 각 Message Format에 해당하는 실제 메시지들에서 Message Format에 속하는 SF(v)와 DF(v) 사이에 해당하는 부분의 데이터를 분석하여 추출한다. Message Format과 Flow Format은 각각 Hierarchical CSP 알고리즘의 2nd step과 3rd step을 통해 추출된다.

Field Format은 프로토콜의 메시지들에서 공통적으로 발견되는 바이트스트림이다. Message Format은 특정 빈도수를 만족하는 하나의 동일한 메시지에서 연속적으로 나타나는 Field Format들의 시퀀스이다. Flow Format은 특정 빈도수를 만족하는 하나의 동일한 메시지에서 연속적으로 나타나는 Message Format의 시퀀스이다.

제안하는 방법론의 전체적인 구성은 그림 4와 같다. 본 방법론은 크게 Message Assemble, Syntax Inference, Semantics Inference, Behavior Inference의 4단계로 구성된다.

먼저 프로토콜 리버스 엔지니어링 수행을 위한 전처리 과정에서는 타겟 네트워크에서 패킷들을 수집하고 5-Tuple(출발지·목적지 IP, 출발지·목적지 port, 전송 계층 프로토콜)이 같은 패킷들을 시간 순으로 정렬하여 양방향 플로우 단위로 변환한다. 수집된 네트워크 트레이스는 단일의 프로토콜 트래픽이라 가정한다. Message Assemble 단계에서는 수집한 프로토콜의 플로우를 로드하고 메시지 단위로 분할한다. Syntax Inference 단계에서는 SF(v)와 DF(v)를 추출하고 이를 기반으로 Message Format을 추출한다. 그리고 각 Message Format에 DF와 GAP에 대한 정보를 추가한다. Semantics Inference 단계에서는 추출된 각 Message Format의 SF(v)와 DF(v)가 사전에 미리 정의한 semantics 유형에 해당하는지 식별한다. Behavior Inference 단계에서는 Flow Format과 FSM을 추출한다.

3.2. Contiguous Sequential Pattern 알고리즘

순차 패턴 알고리즘(Sequential Pattern Algorithm)[19]은 일종의 데이터 마이닝 기술로서 주어진 데이터베이스 내에서 일련의 값 혹은 특정 사건의 시계열 패턴을 추출한다. 이 기술은 연관 규칙 마이닝과 매우 유사하다. 둘 다 큰 데이터베이스 내에서 빈번한 패턴을 추출하는 알고리즘이다. 그러나 연관 규칙 마이닝의 목적은 동일한 트랜잭션 내에서 동시 패턴을 추출하는 것이지만, 순차 패턴 알고리즘은 서로 다른 트랜잭션으로부터 일정한 순서의 패턴을 추출하는 것이다. 이 알고리즘에서는 minimum support value라는 임계값이 매우 중요하다. support란 전체 시퀀스에 대한 타겟 서브 시퀀스를 갖는 시퀀스의 비율이다. 이 알고리즘은 사용자 정의한 minimum support value 값보다 높은 support 값을 갖는 서브 시퀀스를 추출하여 빈번한 서브 시퀀스를 추출할 수 있다.

본 연구진은 개발한 CSP(Contiguous Sequential Pattern) 알고리즘은 프로토콜 syntax 추출에 적합하도록 수정된 순차 패턴 알고리즘이다. 순차 패턴 알고리즘의 원래 버전은 시장의 구매 내역 데이터를 대상으로 순차적인 구매 패턴을 찾는다. 그러나 syntax 추론을 위해서는 단순히 메시지에서 발생되는 시계열 서브 시퀀스가 아닌, 메시지 내의 연속적으로 연결되어 있는 부분 시퀀스를 추출하여야한다. 예를 들어, Field Format의 값을 추출하기 위해서는 메시지에서 순차적으로 나타나는 문자들의 집합이 아니라 연속적으로 연결되어 있는 문자열을 추출해야 한다. 따라서 프로토콜 Syntax를 위한 CSP 알고리즘의 목적은 연속적으로 접해있는 순차 패턴을 추출하는 것이다. 이 알고리즘은 "특정 시퀀스가 빈번하다면 그 시퀀스의 부분 집합 역시 빈번하다"는 Apriori 속성을 기반으로 한다. 이 방법은 후보 시퀀스를 생성하고 자주 발생하는 시퀀스를 결정하기 위해 각 후보 시퀀스의 support 값을 확인한다. CSP 알고리즘은 이러한 기본 Apriori 알고리즘의 수정된 버전인 AprioriAll, AprioriTID, AprioriHash 등을 통합하고 개선하여 성능을 향상시킨다. 제안하는 방법론은 위에서 언급한 세 가지 유형의 Format을 추출하기 위해 그림 5와 같이 CSP를 계층적으로 사용한다. 이를 Hierarchical CSP라 정의한다. 각 step별로 사용된 CSP의 알고리즘은 동일하며 입력 데이터로 사용되는 시퀀스 집합, Length 1 서브 시퀀스 단위 및 support 계산 단위만 다르다.

Hierarchical CSP의 1st step은 메시지 시퀀스 데이터베이스 내에서 특정 빈도수를 만족하는 공통 부분 바이트스트림을 Field Format의 SF(v)로 추출한다. Hierarchical 2nd step은 하나의 동일한 메시지 시퀀스에서 나타나는 인접한 SF(v)의 연속이 특정 빈도수를 만족할 때 이를 추출한다. 이 SF(v)의 연속은 Message Format의 초기 골격이다. 추출된

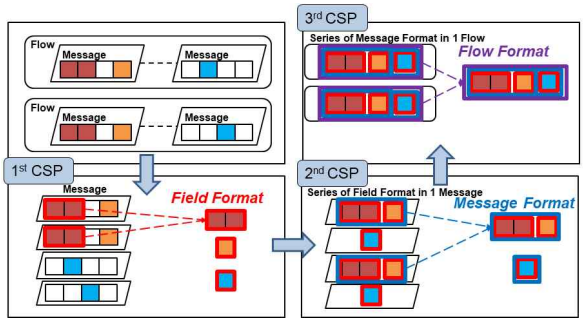

그림 5. Hierarchical CSP 알고리즘의 개념
Fig. 5. The Concept of Hierarchical CSP algorithm

Message Format은 그림 4의 "Extract Additional Field Format{DF(v), DF, GAP} of Message Format" 모듈과 Semantcis Inference 단계를 거쳐 완성된다. Hierarchical CSP의 3rd step은 하나의 동일한 플로우에 나타나는 인접한 Message Format의 연속이 특정 빈도수를 만족할 때 이를 Flow Format으로 추출한다.

$$SequenceSet=\{S_1,S_2,\cdots,S_n\} \quad (1)$$
$$S_i=\{Sequence_{ID},[I_1,I_2,\cdots,I_m] \quad (2)$$

수식 (1)은 CSP 알고리즘의 입력 중 하나인 시퀀스 집합을 나타낸다. 수식 (2)의 S_i는 하나의 시퀀스를 의미하며 시퀀스 ID와 일련의 Length 1 서브 시퀀스들로 구성되어 있다. 표 2는 Hierarchical CSP 알고리즘의 각 step별 차이점이다.

표 2. Hierarchical CSP 알고리즘의 step별 차이점
Table 2. The difference in each step of the Hierarchical CSP algorithm

Step	Transactions	Length-1 items	Support unit
1st CSP	req.[res.] msg.s	1bytes	# of req.[res.] msg.s who contain the candidate / # of total req.[res.] msg.s
2nd CSP	req.[res.] msg.s	Field Formats	# of req.[res.] msg.s who contain the candidate / # of total req.[res.] msg.s
3rd CSP	flows	Message Formats	# of flows who contain the candidate / # of total flows

Hierarchical CSP 알고리즘의 1st step과 2nd step에서는 request 메시지 시퀀스와 response 메시지 시퀀스 집합을 따로 수집하여 CSP를 수행시킨다. 이렇게 수행시킴으로써 request 메시지에만 해당하는 Field Format 및 Message Format과 response 메시지에만 해당하는 Field Format 및 Message Format을 보다 더 올바르게 추출할 수 있다. 예를 들어, HTTP request 메시지의 method 필드의 값 중 하나인 "GET"은 request 메시지 시퀀스 집합만 입력으로 하여 추출할 때 일반적으로 95% 이상의 support 값을 가지지만 request와 response의 구분 없이 전체 메시지 시퀀스 집합에서 추출할 때 support 값은 항상 50% 미만이다. 따라서 minimum support value를 만족하지 못하게 되어 추출을 하지 못하게 되며, 반대로 minimum support value를 낮추면 Noise 데이터들도 Field Format 혹은 Message Format으로 추출하게 된다. 그러나 Hierarchical CSP의 3rd step에서는 추출된 request, response Message Format기 교차되는 양방향 Flow Format을 추출하기 위해 데이터베이스로 전체 시퀀스 집합을 사용한다.

```
Input : SequenceSet, Min_Supp
Output : SubSequenceSet
01: foreach sequence S in SequenceSet do
02:   foreach item i in sequence S do
03:     L1 ← L1 ∪ i;
04:   end
05: end
06: k ← 2;
07: while Lk−1 ≠ ∅ do
08:   foreach candidate c in Lk−1 do
09:     supp ← calSupport(c, SequenceSet);
10:     if(supp < Min_Supp) then
11:       Lk−1 ← Lk−1 − c;
12:     end
13:   end
14:   Lk ← extractCandidate(Lk−1);
15:   k + +;
16: end
17: SubSequenceSet ←∪k Lk;
18: deleteSubset(SubSequenceSet);
19: return SubSequenceSet;
```

그림 6. CSP 알고리즘의 의사 코드
Fig. 6. Pseudo code of the CSP algorithm

그림 6은 CSP 알고리즘의 의사 코드 표현이다. 먼저, 모든 시퀀스로부터 가지고 있는 모든 Length 1 서브 시퀀스들을 추출하여 Length 1 서브 시퀀스 세트 L_1에 저장한다(그림 6, 1~5줄). Length 1 서브 시퀀스들로부터, k가 가장 긴 길이의 시퀀스에 도달하거나 추출되는 후보 서브 시퀀스들이 없을 때까지 length k를 증가시키면서 모든 길이의 length-k 후보 서브 시퀀스들을 추출한다(그림 6, 6~16줄). 이 반복 프로세스는 크게 두 부분으로 구성된다. 첫 번째 부분에서는 support 값을 계산하고 minimum support value 값을 충족하지 않는 후보 서브 시퀀스를 제거한다(그림 6, 8~13줄). 두 번째 부분에서

는 length k-1 후보 서브 시퀀스들 을 사용하여 Length k 후보 서브 시퀀스들을 추출합니다(그림 6, 14줄). 마지막 단계로 각 서브시퀀스 간의 포함 관계를 확인하고, Noise인 서브 시퀀스를 삭제한다(그림 6, 18줄).

3.3. Message Assemble

Message Assemble 단계에서는 프로토콜의 플로우가 로드되고 각 플로우들이 메시지로 분할된다. 두 호스트가 통신 할 때 데이터는 패킷 단위로 전송되므로 이를 응용 계층 수준 데이터 단위(ADU)인 메시지 단위로 재조립해야 한다. 본 연구진은 각 플로우의 패킷을 메시지 단위로 분할하는 방법론을 정의하였다. 전송 계층 프로토콜로 UDP를 사용하는 프로토콜의 경우 메시지의 단위는 하나의 패킷이라 정의하고, TCP를 사용하는 프로토콜의 경우 메시지의 단위는 동일한 방향의 연속적으로 접해있는 패킷 집합이라 정의한다. 그림 7은 Message Assemble 단계에서 패킷의 메시지 조립을 위한 의사 코드 표현이다.

```
Input : Flows consisting of packets
Output : Flows consisting of messages
01: foreach flow in Network traces do
02:   if(flow is TCP flow) then
03:     foreach packet in flow do
04:       if(packet is the first packet of flow) then
05:         M_temp = new Message;
06:         M_temp → insertPkt(packet);
07:       end
08:       else
09:         if(packet.direction == M_temp → direction) then
10:           M_temp → insertPkt(packet);
11:         end
12:         else
13:           flow → insertMsg(M_temp);   M_temp = NULL;
14:           M_temp = new Message;
15:           M_temp → insertPkt(packet);
16:         end
17:       end
18:       if(packet is the last packet of flow) then
19:         flow → insertMsg(M_temp);   M_temp = NULL;
20:       end
21:     end
22:   end
23:   else if(flow is UDP flow) then
24:     foreach packet in flow do
25:       M_temp = new Message;
26:       M_temp → insertPkt(packet);
27:       flow → insertMsg(M_temp);   M_temp = NULL;
28:     end
29:   end
30: end
```

그림 7. Message Assemble의 의사 코드
Fig. 7. Pseudo code of Message Assemble

3.4. Syntax Inference

Syntax Inference 단계의 목적은 상세한 Message Format을 위해 구성하는 모든 Field Format들이 SF(v), DF(v), DF 및 GAP으로 완전하게 분류된 Message Format을 추출하는 것이다. 그림 8은 Syntax Inference 단계의 개관이다.

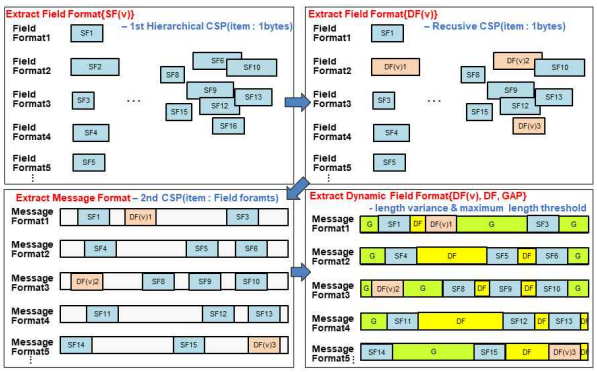

그림 8. Syntax Inference 단계의 개관
Fig. 8. The overview of Syntax Inference Phase

Syntax Inference의 첫 번째 모듈인 "Extract Field Format{SF(v)}"에서는 Hierarhcical CSP의 1st step을 사용하여 SF(v)를 추출한다.

두 번째 모듈인 "Extract Field Format{DF(v)}"에서는 추출된 SF(v) 중에서 DF(v)로 변환될 가능성이 있는 SF(v)를 선택한다. 이를 선택하기 위한 조건은 해당 SF(v)의 "support 값이 100%가 아니며 Position Variance가 충분히 낮다"라는 것이다. 본 연구진은 이 Position Variance의 임계값을 200으로 설정하였다. 그런 다음 선택된 각 SF(v)에 대해 각각 Recursive CSP를 수행한다.

Recursive CSP의 과정은 다음과 같다. 먼저, 초기 데이터베이스, 즉 메시지 시퀀스들 중 해당 SF(v)를 포함하지 않는 메시지 시퀀스들만을 따로 추출한다. 이 메시지 시퀀스들에 대하여 해당 SF(v)의 최소 offset과 최대 depth을 기준으로 모든 메시지 시퀀스들을 잘라 새로운 데이터베이스를 생성한다. 새로운 데이터베이스, 즉 SF(v)의 최소 offset과 최대 depth를 기준으로 잘린 메시지 시퀀스들을 입력으로 CSP를 수행한다. CSP를 통해 추출된 결과들 중 가장 높은 support 값을 갖는 결과를 해당 SF(v)의 필드 값 배열에 저장한다. 위의 과정을 더 이상 해당 SF(v)의 새로운 값이 추출되지 않을 때까지 반복한다. 이러한 SF(v)는 여러 개의 값을 가지는 Field Format이므로 DF(v)라 한다. 따라서 생

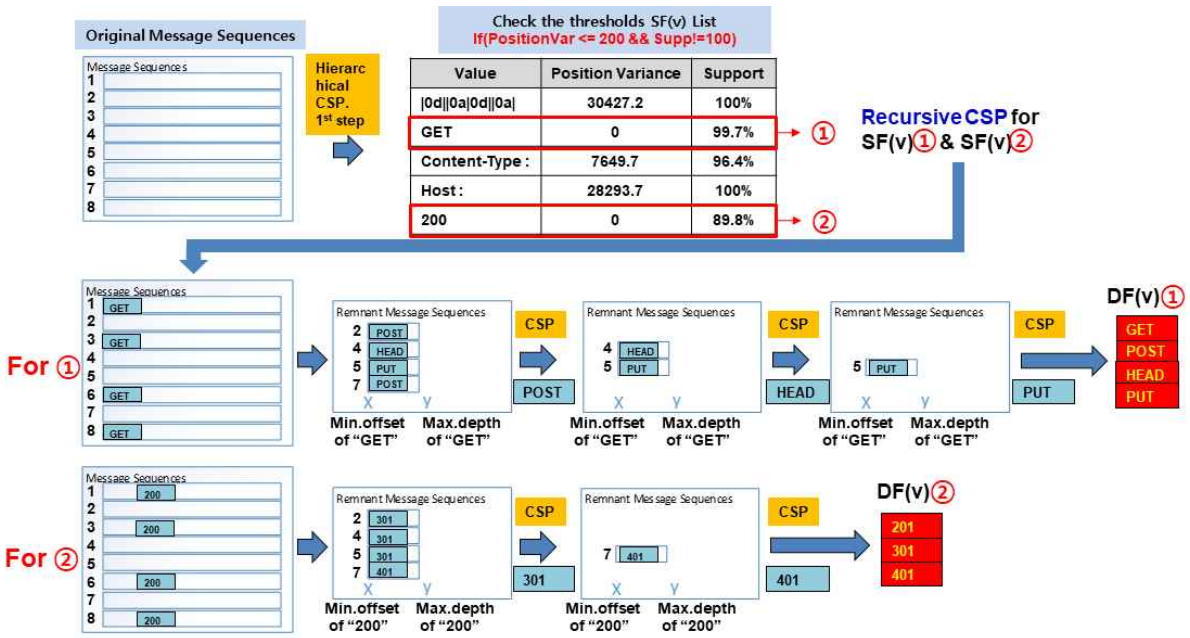

그림 9. Recursive CSP 알고리즘의 HTTP 프로토콜에 대한 예시
Fig. 9. The process of Recursive CSP by examplifying the HTTP protocol

성된 DF(v)에는 해당 Field Format이 가질 수 있는 모든 값을 저장하게 된다. 그림 9는 HTTP 프로토콜의 method 필드와 status code 필드를 예시로 한 Recursive CSP의 과정이다.

세 번째 모듈인 "Extract Message Format"에서는 Length 1 서브 시퀀스의 단위를 앞서 추출한 SF(v)와 DF(v)로 하여 Hierarchical CSP의 2nd step을 이용하여 Message Format을 추출한다.

네 번째 모듈인 "Extract Additional Field Format{DF(v), DF, GAP} of Message Format"에서는 앞서 추출한 각 Message Format을 구성하는 Field Format들의 사이를 DF나 GAP 혹은 DF(v)로 구분한다. 이 모듈에는 두 개의 임계값이 사용되며 모듈의 과정은 다음과 같다. 두 임계값의 첫 번째는 "이전 Field Format과 다음 Field Format 사이 부분의 Length Variance"에 대한 임계값이고 두 번째는 "이전 Field Format과 다음 Field Format 사이 부분의 Maximum Length"에 대한 임계값이다. 먼저 각 Message Format에서 구분하고자 하는 Field Format들의 사이를 설정한다. 그리고 원본 데이터베이스, 즉 전체 메시지 시퀀스에서 해당 Message Format에 해당하는 메시지 시퀀스들만을 선별한다. 선별한 모든 메시지 시퀀스들에서 분류하고자 설정한 Field Format 사이 부분들에 해당하는 데이터의 Maximum Length와 Length Variance를 계산한다. Length Variance가 사용자가 설정한 첫 번째 임계값보다 큰 경우 GAP으로 분류하고 그렇지 않은 경우 GAP이 아닌 Field Format으로 분류한다. 본 연구진은 첫 번째 Length Variance에 대한 임계값으로 5000을 사용하였다. GAP은 길이와 값이 매우 가변적인 Field Format을 의미한다. 다음으로, GAP이 아닌 Field Format의 경우 Maximum Length가 사용자가 설정한 두 번째 임계값보다 작으면 DF(v)로 분류하고, 그렇지 않으면 DF로 분류한다. 본 연구진은 두 번째 Maximum Length에 대한 임계값으로 25를 사용하였다. DF는 값은 매우 가변적이나 길이는 어느 정도 고정되어 있는 Field Format을 의미한다. DF(v)는 값과 길이를 어느 정도 고정 되어 있는 Field Format을 의미하므로 이 모듈에서 추출된 DF(v)에 설정한 Field Format 사이 부분의 값을 최대 10개까지 배열로 저장한다. 위 과정을 해당 Message Format의 모든 Field Format 사이에 대하여 수행하고 마찬가지로 모든 Message Format에 대하여 이를 수행한다. 그림 10에서 Part1과 Part3는 이전 Field Format과 다음 Field Format 사이의 Length Variance가 5000보다 작기 때문에 GAP이 아닌 Field Format으로 분류되며, 반면에 Part2는 Length Variance가 5000보다 크기 때문에 GAP으로 분류된다. 또한, Part1은 이전 Field Format과 다음 Field Format 사이의 Maximum Length가 5로 매우 작기 때문에 DF(v)로 분류되며, Part3은 이전 Field Format과 다음 Field Format 사

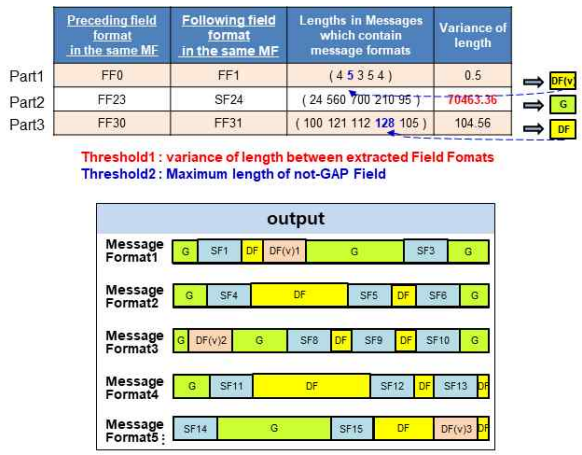

그림 10. DF(v), DF, GAP 추출의 예시
Fig. 10. The example of extracting DF(v), DF, and GAP

이의 Maximum Length가 128로 25보다 크므로 DF로 분류된다. 이 모듈을 완료하면 완전하게 구성된 Message Format을 얻을 수 있다.

3.5. Semantics Inference

Semantics Inference 단계에서는 6가지 유형의 semantics 유형을 미리 정의하고 모든 Field Format을 순회하면서 각 semantics 유형에 해당하는지 각각의 알고리즘을 통해 확인하여 해당 Field Format의 식별된 semantics를 저장한다. 본 알고리즘에서는 FieldHunter의 방법론을 빌려 각 Message Format을 구성하는 Field Format의 의미를 찾는다. 이는 표 1과 같이 여러 선행 연구들의 semantics 추출 방법론들 중 FieldHunter의 방법론이 가장 구체적이며 다양한 종류의 semantcis를 추출하기 때문이다. 본 알고리즘이 사용하는 semantics 추출 방법론과 FieldHunter의 방법론의 차이점은 FieldHunter의 경우, 메시지를 n-gram으로 먼저 분류하여 메시지의 모든 offset이 같은 n-gram을 가지고 semantics를 추론하지만, 본 연구에서 설계한 알고리즘은 Field Format과 Message Format을 모두 추출한 후에 특정 Message Format을 구성하는 특정 Field Format에 해당하는 데이터들만을 가지고 semantics를 추론하기 때문에 semantics 추출의 오차가 줄어든다. 또한, FieldHunter의 경우 인접한 n-gram이 Field Format으로 확장할 수 있는지의 확인 과정을 추가적으로 수행한다.

식별하는 6가지 semantics 유형은 MSG-Type, MSG-Len, Host-ID, Session-ID, Trans-ID, Accumulators이다. 이를 추론하기 위해 Syntax Inference의 네 번째 모듈에서와 같이 원본 데이터베이스에서 특정 Message Format의 특정 Field Format에 해당하는 데이터들만을 수집하고 각 6가지 semantics 유형이 해당하는 지를 판별한다. 따라서 하나의 Field Format은 여러 개의 semantics를 가질 수 있다. 이와 같은 과정을 모든 Message Format의 모든 SF(v)와 DF(v)에 대하여 수행한다.

1) MSG-Type : MSG-Type에 해당하는 Field Format은 정적 필드가 아니며 너무 가변적이지 않은 동적 필드이다. 이러한 동적 필드가 이 필드를 포함하는 메시지의 반대 방향 메시지에도 같은 위치에서 대응되는 semantics를 가진 Field Format이 있을 때 이를 MSG-Type으로 분류한다. 즉, MSG-Type에 해당하는 Field Format은 request / response와 같이 인과 관계가 있다. 이를 찾기 위한 알고리즘에서는 엔트로피 metric H(x)와 casuality meric I(q:r)/H(q)를 사용한다.

2) MSG-Len : MSG-Len에 해당하는 Field Format은 값으로 메시지의 길이를 갖고 있는 동적 필드다. 이를 찾기 위한 알고리즘은 Pearson 상관계수를 사용하여 각 메시지의 Field Format의 값과 해당 메시지의 길이가 선형 관계가 있음을 확인한다.

3) Host-ID : Host-ID에 해당하는 Field Format은 전자 메일 주소, 사용자 ID, 호스트 IP 주소와 같이 Source Address에 종속된 값을 갖는 동적 필드이다. 이를 찾기 위한 알고리즘은 categorical metric : R(x,y)=I(x;y)/H(x,y)를 사용한다.

4) Session-ID : Session-ID에 해당하는 Field Format은 세션, 즉 one-way-flow에 종속된 값을 갖는 동적 필드이다. cookie 또한 이 Session-ID에 해당한다. 이를 찾기 위한 알고리즘은 Host-ID에 해당하는 Field Format을 찾는 알고리즘과 마찬가지로 categorical metric을 사용한다.

5) Trans-ID : Trans-ID에 해당하는 Field Format은 Request 메시지와 Response 메시지 쌍인 트랜잭션에 종속된 값을 가지는 동적 필드이다. 또한 하나의 트랜잭션을 구성하는 Request 메시지와 Response 메시지의 Trans-ID는 같다. 이를 찾기 위한 알고리즘은 엔트로피 metric을 사용하고 대응되는 방향의 메시지에서의 필드의 값이 분석하고자 하는 필드의 값과 일치하는지를 확인한다.

6) Accumulators : Accumulators에 해당하는 Field Format은 시간의 흐름에 따른 연속적인 메시지들에서 해당하는 값이 어느 정도 일정한 증분치

를 가지고 증가하는 동적 필드이다. 이 알고리즘은 이 필드를 찾기 위해 너무 가변적이지 않은 증분이 있는지 각 증분들의 분산을 가지고 확인한다.

3.6. Behavior Inference

Behavior Inference에서는 앞서 추출한 Message Format으로 프로토콜 FSM과 Flow Format을 추출한다. 그림 11은 Behavior Inference의 개관이다.

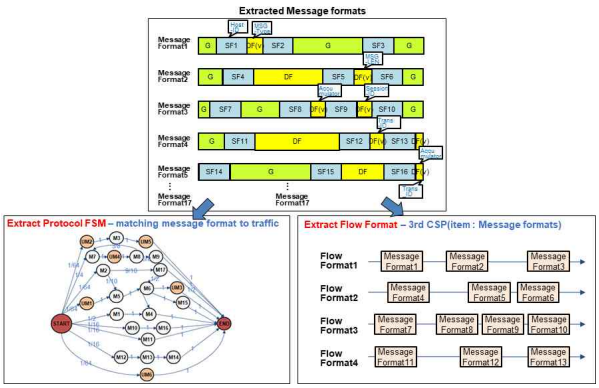

그림 11. Behavior Inference의 개관
Fig. 11. Overivew of Behavior Inference phase

1) Extract Protocol FSM 모듈 : 설계한 알고리즘에서 프로토콜 FSM의 각 state는 추출된 하나의 Message Format이며 동일한 유형의 메시지 집합을 의미한다. 이 모듈은 각 추출된 Message Format들을 입력 트래픽에 매칭시켜 각 state들에 해당하는 메시지 시퀀스의 변경을 찾아 FSM의 transition으로 추출한다. 이 과정에서 각 transition에 대한 매칭된 횟수를 기록하고 그림 11과 같이 각 state에서의 각 edge에 전이 확률을 계산하여 표시한다. 전이 확률은 하나의 state가 가지는 모든 transition과 해당 transition이 일치하는 횟수를 통해 계산된다. 이 전이 확률은 어떤 메시지가 가장 큰 가능성을 가지고 발생하는지 예측할 수 있기 때문에 패킷 Replay에 매우 유용하다. 모든 state와 transition을 추출한 뒤 FSM 각 경로의 최 앞단 state들 앞에 START state를 연결하고 최 말단 state들 뒤에 END state를 연결한다. 그림 11에서 START state에서 END state로 연결된 각 경로는 각각 하나의 플로우에 속해 있다. 추출된 FSM은 프로토콜의 각 메시지 유형이 어느 정도의 확률을 갖고 어떤 순서로 동작하는지 확인하는 데 도움이 된다.

2) Extract Flow Format 모듈 : 이 모듈에서는 Length 1 서브 시퀀스를 Message Format으로 하여 Hierarchical CSP의 3rd step을 사용하여 Flow Format을 추출한다. 추출된 Flow Format은 프로토콜의 주요 플로우 타입을 나타내므로 프로토콜에 대하여 더 자세한 정보를 줄 수 있다. 또한 Flow Format을 사용하여 프로토콜 FSM을 간결하게 만들 수 있다.

Ⅳ. 결론 및 향후 연구

본 논문에서는 명확한 프로토콜 리버스 엔지니어링을 위한 방법론을 제안하였다. 직관적이며 명확한 프로토콜 사양 추출을 위해 세 가지 유형의 Format과 네 가지 유형의 필드 형식을 정의하였으며 이를 위한 Hierarchical CSP와 Recursive CSP 알고리즘을 개발하고 프로토콜 사양을 추출하였다. 본 방법론의 우수성을 HTTP 프로토콜을 예시로 하여 설명하였다. 향후 연구로는 설계한 방법론을 개발하여 다양한 프로토콜에 적용하고 발전시킬 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. Institute of Commun. and Information Sciences, 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. Institute of Commun. and Information Sciences, 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”, Journal ACM Computing Survey, Vol. 48, Issue 3, No. 40, 2016

[4] W.-S. Jung, J.-H. Yun, S.-K. Kim, K.-S. Shim, S.-M. Kim, and M.-S. Kim, “Flow Whitelists Creation Reflecting the Diversity of Scada Networks”, J. Institute of Commun. and Information Sciences, Vol. 42, No. 6, pp. 1167-1174, Jan. 2017.

[5] S. Morgan, 5 stats of cyber security in 2017(2017), Retrieved Jun. 16, 2017, from http://www.itworld.co.kr

[6] M.-H. Kim, Major Security Vulnerabilities and Security Accident Trends in 2016(2017), Retrieved Jan. 4, 2017, from http://www.igloosec.co.kr

[7] N. Borisov, D. J. Nrumley, H. J. Wang, and C. Guo. "Generic Application-Level Protocol Analyzer and its Language" In Network and Distributed System Security Symposium(NDSS), San Diego, CA, Feb. 2007

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

[9] 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. on Computer and Commun. Security(CCS), 2007

[10] 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. on Computer and Commun. Security(CCS), 2008

[11] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda. "Prospex: Protocol Specification Extraction," In Proc. 30th IEEE Symposium on Security and Privacy(S&P), 2009

[12] J. Caballero and D. Song, "Automatic Protocol Reverse-Engineering: Message Format Extraction and Field Semantics Inference," Internatoinal Journal of Computer and Telecommun. Networking Vo. 57, Issue 2. Elsevier, pp 451-474, 2013

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

[14] M. Shevertalov, and S. Mancoridis. "A Reverse Engineering Tool for Extracting Protocols of Networked Applications", In 14th Working Conf. on Reverse Engineering(WCRE'07). pp229-238/ DOI:10.1109/WCRE. Jun. 2007

[15] Y. Wang, Z. Zhang, D. Yao, B. Qu, and L Guo, "Inferring Protocol State Machine From Network Traces: In Proc. of 9th Internatoinal Conf. Applied Criptograph and Network Security(ANCS). 2011.

[16] I. Bernudez, A. Tongaonkar, M. Iliofotou, M. Mellia, and M. Munafo. "Automatic Protocol Field Inference for Deeper Protocol Understanding," IFIP Networking Conference 2015.

[17] C. Leita, K. Mermoud, and M. Dacier. "ScriptGen: An Automated Script Generation Tool for HoneyD," In 21st Annual Computer Security Applications Conference (ACSAC'05), 200-214 DOI:10.1109/CSAC. 49.2005.

[18] W. Cui, V. Paxson, N. C. Wever, and R. H. Katz, "Protocol-Independent Adaptive Replay of Application Dialog," In Proc, 13th Symposium on Network and Distributed System Security(NDSS'06). 2006

[19] R. Agrawal, R. Srikant, "Mining Sequential Pattern", in Data Engineering, 1995. Proc. 11th Internatoinal Conf. on IEEE, pp 3-14. 1995

구 영 훈 (Young-Hoon Goo)

2016 고려대학교, 컴퓨터정보학과 학사

2016 ~ 현재 고려대학교 컴퓨터정보학과 석/박사통합과정

<관심분야> 네트워크 관리 및 보안, 트래픽 모니터링 및 분석

심 규 석 (Kyu-Seok Shim)

2014 고려대학교, 컴퓨터정보학과 학사

2016 고려대학교 컴퓨터정보학과 석사

2016 ~ 현재 고려대학교 컴퓨터정보학과 박사과정

<관심분야> 네트워크 관리 및 보안, 트래픽 모니터링 및 분석

박 지 태 (Jee-Tae Park)

2017. 08 고려대학교, 컴퓨터정보학과 학사
2017 ~ 현재 고려대학교 컴퓨터정보학과 석/박사통합과정
<관심분야> 네트워크 관리, 트래픽 모니터링 및 분석

채 병 민 (Byeong-Min Chae)

2007. 02 충남대학교, 물리학과 학사
2012. 02 충남대학교 컴퓨터공학과 석사
2007 ~ 2008 삼성전자 연구원
2008 ~ 현재 한화시스템 연구원
<관심분야> 네트워크 관리 및 보안, 트래픽 모니터링 및 분석

문 호 원 (Ho-Won Moon)

2000. 02 한양대학교, 수학과 학사
2000 ~ 2011 삼성전자 연구원
2011 ~ 현재 한화시스템 연구원
<관심분야> 네트워크 관리 및 보안, 트래픽 모니터링 및 분석, 라우팅

김 명 섭 (Myung-Sup Kim)

1998 포항공과대학교, 전자계산학과 학사
2000 포항공과대학교, 전자계산학과 석사
2004 포항공과대학교, 전자계산학과 박사
2006 Dept. of ECS, Univ of Toronto Canada
2006 ~ 현재 고려대학교 컴퓨터정보학과 교수
<관심분야> 네트워크 관리 및 보안, 트래픽 모니터링 및 분석, 멀티미디어 네트워크