자동화된 어플리케이션 레벨의 Signature 생성

박병철 [1], 원영준 [1], 김명섭 [2], 홍원기 [1]

[1] 포항공과대학교 컴퓨터공학과

[2] 고려대학교 컴퓨터공학과

[1]{fates, yjwon, jwkhong}@postech.ac.kr, [2]tmskim@korea.ac.kr

요 약

어플리케이션 트래픽의 정확한 분류는 네트워크 monitoring 과 분석 측면에서 매우 중요한 부분을 차지하고 있다. 가장 전통적인 트래픽 분류방법은 미리 정의된 well-known 포트와 비교하는 방법이 사용되었지만 이러한 분석 방법은 높은 정확도를 보장하지 못하고 있다. 대체 방법인 어플리케이션 레벨의 signature 비교 방법은 좀 더 높은 정확도를 제공하지만 신뢰도가 높은 signature 를 찾기 위해 트래픽에 대한 exhaustive search 가 요구된다. 사전에 어플리케이션의 프로토콜에 대한 정보를 알고 있다면 높은 정확도의 signature 를 보장할 수 있지만, 대부분의 인터넷 어플리케이션 프로토콜 정보는 공개가 되고 있지 않다. 본 논문에서는 어플리케이션 프로토콜에 대한 사전 조사 없이 선행적인 프로토콜 분석을 통한 signature 생성과 동일한 정확도의 어플리케이션 레벨의 signature 를 생성할 수 있는 효과적인 방법을 제시한다.

1. 서론

어플리케이션 트래픽 분류는 네트워크의 운용관리, 트래픽 엔지니어링, 과금, 그리고 QoS 관리 등을 위한 정보인 현재 네트워크 상태를 제공하기 위한 중요한 연구 분야이다. 최근 들어 웹(인터넷) storage 와 Peer-to-Peer (P2P) 파일 공유 어플리케이션들의 종류가 급격하게 증가하였고, 이들이 발생시키는 트래픽의 량이 전체 인터넷 트래픽에서 매우 많은 부분을 차지하고 있다[2]. 특히 최신의 P2P 어플리케이션들은 동적인 port 할당이나 relay 노드를 사용하는 것과 같은 방법들을 통해 detection 과 filtering 을 피하고 있다. 이러한 인터넷 트래픽의 변화로 인해 port 기반의 트래픽 분류와 같은 초기의 단순한 분류 방법들은 점차적으로 일정 수준 이상의 정확도를 보장하지 못하게 되었다. Moore 의 연구[4]에서도 port 기반의 분류 방법은 50~70%의 정확도를 넘지 못한다고 주장하고 있다.

기존의 분류 방법들[5]의 문제점들을 해결하기 위한 새로운 해결책으로 signature 기반의 분류 방법이 제안되었다. 어플리케이션 signature 는 어플리케이션의 종류를 구분 지을 수 있는, packet payload 데이터상의 고정적으로 나타나는 string 이나 hex 값의 패턴을 의미한다. 이러한 접근 방법은 Internet 웜 community (예: Intrusion Detection Systems)에서 먼저 사용 되었던 방법이지만 목표하는 바가 비정상 트래픽만을 detect 하는 것으로 트래픽을 각각의 종류별로 분류하는 것과는 차이가 있다. 어플리케이션 signature 는 네트워크 관리자나 보안 전문가들에 의해 manual 한 방법으로 만들어져 왔다. 이러한 방법은 공통적으로 나타나는 패턴을 추출하기 위해 프로토콜에 대한 조사나 packet payload 에 대한 정밀한 검사가 선행되어야만 한다. 따라서 새로운 어플리케이션이 등장하였을 때 signature 를 만들기 위해서는 상대적으로 많은 시간이 소요된다. 이러한 문제점들을 해결하기 위해서는 signature 를 생성하는 체계적인 시스템이 필요하다.

본 논문에서는 선행적인 프로토콜 조사 없이 자동적으로 어플리케이션 signature 를 생성하는 새로운 방법으로 구현된 semi-automatic signature 생성시스템을 소개하고 이 시스템의 효용성을 검증한다. 그리고 semi-automatic 시스템의 문제점인 사용자의 개입 없이 fully-automatic 한 방법으로 signature 를 생성할 수 있는 시스템의 가능성을 제시한다. 웜 signature 의 자동 생성에 관한 몇몇의 연구들이 존재하지만 웜과 정상적인 인터넷 트래픽의 차이점들 때문에 이들 연구에서 사용한 sliding window 알고리즘[9]을 적용하는 데에는 어려움이 따른다. 따라서 본 연구에서는 sliding windows 알고리즘과 전혀 다르고 linear time 에 동작할 수 있는 Longest Common Substring (LCS) 알고리즘[13]을 수정하여 적용하였다. 최근에 조사한 바로는 웜과 같은 비정상 트래픽이 아닌 일반적인 인터넷 트래픽에 대해 자동적으로 signature 를 생성하는 것에 관한 연구 시도는 없었다.

본 논문의 구성은 다음과 같다. 2 장에서는 관련 연구에 대해 기술하고 본 연구에서 목적으로 하는 문제의 범위를 정의한다. 3 장의 modified LCS 를 이용한 signature 생성에서는 자주 사용되는 웜

본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음 (IITA - 2006 - (C1090 - 0603 - 0045))

signature 와 어플리케이션 signature 의 포맷을 재정의 하고, LCS 의 modification 에 필요한 제약 사항들과 modified LCS 알고리즘을 실제 트래픽에 적용하는 방법을 설명한다. 또한 4 장에서는 modified LCS 알고리즘을 기반으로 실제 signature 를 생성하는 두 시스템에 대한 설명과 검증을 다룬다. 6 장에서 결론과 앞으로의 연구방향을 제시함으로써 본 논문을 마무리한다.

2. 문제의 범위 (Problem Domain)

이번 장에서는 기존에 존재하는 signature 기반의 트래픽 분류 연구에 대한 개략적인 소개를 하고, 웜 signature 생성에 사용되었던 접근 방법을 일반적인 어플리케이션 분류에 사용하기 어려운 이유에 대한 간략한 설명을 한다. 그리고 본 논문의 범위 내에서 signature 생성의 문제의 범위를 재정의한다.

2.1 관련 연구

다음에 언급되는 연구들은 signature 비교 방법을 기반으로 한 자신들만의 트래픽 분류 방법을 보여주고 있다. Gummadi 의 연구 [3]는 KaZaA 의 다운로드 트래픽의 분류를 위해 어플리케이션 레벨의 signature 를 사용하였지만, 생성된 signature 에 대해 정확도 측정과 같은 검증 과정을 제시하지 않았다. 물론 전체적인 접근 방법이 논리적이지만 트래픽 분류 결과의 정확성 에 대한 의문이 남는다. Sen 의 연구 [5]에서는 어플리케이션의 프로토콜을 분석하여 P2P 어플리케이션의 signature 를 생성하였다. 이러한 분석 방법은 signature 의 정확도를 높일 수 있지만 수동적인 방법으로 signature 를 생성해야 하므로 효율성이 떨어지고 실시간 시스템에 적용하기에는 부적절하다. 종합적으로 보았을 때에 이와 비슷한 연구들은 signature 비교 방법을 통한 어플리케이션 트래픽의 분석률을 제시하지만 signature 를 생성하는 방법이나 그 signature 자체의 정확도에 대한 검증을 중요하게 언급하지 않고 있다.

Signature 기반의 트래픽 분류는 웜 detection 을 위한 Intrusion Detection System (IDS) 들에서 주로 사용되어왔다[6]. 이 연구들은 웜 signature 를 자동으로 생성할 수 있는 시스템의 개발에 대해 기술하고 있는데, 이들 모두 공통적으로 port scanning 과 같은 웜의 대표적인 특성을 보이는 비정상 트래픽에 대한 signature 생성을 목적으로 한다. 이와 같은 목적이 일반적인 어플리케이션 signature 생성과 가장 큰 차이점은 각각의 어플리케이션들이 자신만의 유일한 signature 를 갖는 것과 달리 웜 signature 의 경우는 여러 개의 웜들이 하나의 signature 를 공유할 수 있다는 점이다. 이는 어떠한 의미에서 웜 signature 가 좀더 약한 제약사항을 갖는다고 볼 수 있다.

일반적으로 웜 signature 생성에 관련된 연구에서는 sliding-windows 알고리즘과 자신들이 정의 한 break point 를 사용하고 있다. Scheirer 의 연구 [10]에서는 이러한 break point 의 선정에 관한 설명을 기술하고 있는데, 이들 break point 는 웜들이 공통적으로 갖고 있는 특정 동작에 해당하는 instruction code 와 같은 hex 값들이다. 따라서 웜과 같은 비정상 트래픽과 특성이 다른 일반적인 인터넷 어플리케이션 트래픽 분류에 break point 를 사용하는 sliding-windows 알고리즘을 직접 적용하는 것에는 어려움이 따른다. 이러한 점들이 기존의 웜 signature 생성에 관한 연구와 본 연구 사이의 차이점을 시사한다.

Longest common subsequence problem (LCS) [13]은 모든 sequence 들에 (일반적으로 2 개) 공통적으로 나타나는 가장 긴 subsequence 를 찾는 문제이다. 이는 전통적인 컴퓨터과학 문제 중에 하나이며 생물정보학의 응용으로 주로 사용되어 왔다. 가장 주된 응용은 생명정보학에서의 DNA sequence matching 이다. DNA 는 각각 서로 다른 특정한 의미를 갖는 4 가지의 구성요소 (A, C, G, T)만으로 이루어 있는 반면 어플리케이션 트래픽은 이보다 더욱 복잡한 구성요소들로 이루어져 있으며 모든 구성 요소가 의미를 갖고 있는 것이 아니기 때문에 자동화된 어플리케이션 레벨의 signature 생성 시스템을 위해서 LCS 알고리즘을 수정/적용 하였다. LCS 의 동작에 대한 자세한 사항은 3.3 장에서 예제를 통해 기술한다.

2.2 문제 정의와 자동화

우리의 시스템은 어플리케이션의 signature 로 특정 어플리케이션의 packet stream 으로부터 하나 이상의 string 이나 hex 값이 이루는 sequence 들을 자동적으로 추출해낸다. 본 논문에서는 모든 어플리케이션들이 그들만의 독특하고 유일한 signature 를 갖는다는 가정하에 모든 네트워크 기반의 어플리케이션의 signature 를 생성하는 것을 목적으로 한다.

Modified LCS 알고리즘의 입력으로 signature 생성을 원하는 대상 application 에서 발생한 raw packet 들만을 수집하여 사용한다. 이러한 입력 데이터의 수집은 수동적인 방법과 트래픽 수집 agent 를 통해 이루어진다. 수동적인 방법의 경우에는 대상 어플리케이션을 독립적으로 실행시켜 data 를 수집하고 트래픽 수집 agent 를 사용하는 경우에는 agent 가 탑재된 컴퓨터에서 발생한 트래픽을 프로세스(어플리케이션) 별로 구분하여 저장하는 동작을 한다. 위의 두 가지 방법으로 수집된 데이터는 semi-automatic 시스템과 fully-automatic 시스템에서 각각 입력으로 사용된다. Signature 생성 시스템의 자동화 정도와 signature 의 정확도 사이에는 교환조건이 존재 하기 때문에 서로 다른 자동화의 정도를 갖는 시스템을 사용하며 두 시스템의 차이에 대해서는 4 장에서 더욱 자세히 설명을 한다.

3. Modified LCS 를 사용한 Signature 생성

이번 장에서는 여러 연구에서 사용된 signature 포맷에 대하여 기술하고 일반적인 인터넷 어플리케이션 signature 에 적합한 포맷을 제안한다. 또한 어플리케이션 signature 를 위한 제약 사항들과 modified LCS 알고리즘을 적용하여 signature 를 생성하는 동작에 대해 설명한다.

3.1 Signature 의 포맷

A. 웜 Signature 포맷

Newsome 의 연구 [15]에서는 polymorphic 웜들에 적합한 signature 포맷을 몇 가지로 분류하였다. Polymorphic 웜은 새로운 감염을 시도할 때마다 payload 의 내용을 변화시키는 웜을 의미한다. 이 연구에서 분류한 signature 들은 웜을 제외한 다른 트래픽에 적용되지 못한다. 이들 signature 는 아주 단순한 substring 의 sequence 로 이루어져있는데, 본 연구에서는 이를 바탕으로 일반적인 인터넷 어플리케이션에 적합한 조건들을 추가하여 새로운 포맷들을 정의 하였다. 이들에 대해서는 다음 장에서 자세히 소개되며, 아래의 signature 포맷들은 Newsome 의 연구에서 정의한 웜 signature 의 포맷들이다.

- **Conjunction signatures**: signature 는 substring (또는 token)들의 집합으로 이루어져있으며, token 들의 순서는 상관이 없다.

- **Token-subsequence signatures**: signature 는 특정 순서의 token 들의 집합으로 이루어진다.

- **Bayes signatures**: signature 는 각각 score 와 threshold 값을 갖는 token 들의 집합으로 이루어진다.

Brumle 의 연구 [14]역시 몇 가지의 웜 signature 의 포맷을 제안하고 있다. Turing machine signature, symbolic constraint signature, 그리고 regular expression signature 가 이에 해당한다. 하지만 이러한 signature 포맷으로 signature 를 기술할 경우 matching 효율이 낮아지기 때문에 signature 들을 모든 packet 들과 비교하는 경우에는 접합하지 못하다.

B. 인터넷 어플리케이션 signature 포맷

지금까지 진행되었던 signature 기반의 트래픽 분류에 관한 연구들에서는 자신들만의 signature 포맷을 정의하고 이에 맞는 signature 들을 생성하여 트래픽을 분류해왔다. 우리는 기존에 사용되어오던 signature 을 다음과 같이 몇 가지로 분류하였다.

- **Common string with fixed offset**: 이는 어플리케이션에서 발생하는 packet 의 payload 의 특정 위치에 반복적으로 나타나는 string 이나 hex 값을 어플리케이션 signature 로 정의 한다. 그리고 대부분의 경우는 offset 이 payload 의 시작점에 해당한다.

- **Common string with variable offset**: 이 포맷은 앞의 포맷과 유사하지만 반복적으로 나타나는 string 이나 hex 값이 고정된 위치가 아닌 다양한 위치에 나타날 수 있다는 점에서 다르다.

- **Sequence of common substrings**: 하나의 짧은 string 이나 hex 값은 정확도를 보장하지 못하기 때문에 이 포맷은 앞의 두 정의와 달리 하나의 string 이 아닌 여러 개의 substring 들이 일정한 순서를 갖고 payload 상에 나타나는 것을 signature 로 정의한다.

- **Behavioral signature**: 이는 어플리케이션 트래픽의 packet inter-arrival time 이나 최소/최대 packet 사이즈 등과 같은 특징들을 signature 로 정의한다. 본 논문에서는 packet payload 에 대한 조사를 통해 자동으로 signature 를 생성하는 것을 목적으로 하기 때문에 이 signature 포맷은 고려하지 않는다.

Karagiannis 는 자신의 논문 [10]에서 common string with fixed offset 포맷을 사용하였다. 하지만 이러한 포맷의 signature 를 사용할 경우에는 정확도가 상당히 떨어질 수 있다. 어플리케이션 프로토콜에 대한 조사가 없이 생성한 signature 이기 때문에 이러한 문제가 발생하게 되는데, 예를 들어 프로토콜의 signature 앞에 가변적인 길이를 가진 string 이 존재하는 경우에는 signature 의 offset 또한 변화할 수 있기 때문에 fixed offset 을 사용하는 경우에는 트래픽을 정확히 분류해 낼 수가 없다. 최근 많은 어플리케이션들이 편의성을 위해 HTTP 프로토콜을 이용하면서 많은 packet 들의 payload 에 'GET' 또는 'HTTP'라는 string 이 빈번이 나타난다. 하지만 이와 같은 string 들은 어플리케이션을 분류하는 역할을 하지 못한다. 따라서 앞에서 설명한 signature 포맷을 사용할 경우에는 이러한 string 들은 어플리케이션 레벨의 signature 로써 아무런 의미를 갖지 못한다. Sen [5]은 마지막에 언급했던 sequence of substrings 라는 signature 포맷을 이용하여 signature 의 정확도를 높이고 있다. 이러한 signature 포맷을 사용하면 'HTTP'와 같은 string 또한 signature 의 일부로써 의미를 갖게 된다.

전체적으로 살펴보면 behavioral signature 를 제외한 세가지 포맷 중 sequence of substrings 가 다른 나머지 두 개의 포맷을 포함하는 superset 이 되며 정확도가 높다. 따라서 앞으로 설명할 signature 생성을 위한 접근 방법에서는 sequence of substring 을 signature 포맷으로 삼는다.

3.2 어플리케이션 signature 를 위한 제약사항

LCS 를 사용하여 어플리케이션 signature 를 생

성하는 것은 DNA sequence matching 과는 차이점이 있다. 기본적으로 어플리케이션 signature 는 DNA 보다 더 많은 구성요소로 이루어지고 또한 구성요소 각각이 모두 의미를 갖는 것이 아니기 때문에 다음에 설명할 제약사항들이 포함되도록 LCS 알고리즘을 수정해야 한다. 이러한 제약사항들은 전체적인 signature 생성 시스템의 효율성을 증가시키고 signature 의 정확도를 증가시키는 것을 목적으로 한다.

A. Flow 를 이루는 packet 의 수

Flow 는 동일한 source IP, destination IP, source port, destination port, 그리고 프로토콜을 갖는 packet 들의 집합이다. [5]에서 signature 가 존재하는 packet 은 flow 초반의 몇 개의 packet 에 제한됨을 설명하였다. Flow 상에는 수많은 packet 들이 존재하는데, 이 모든 packet 에 대해 payload 를 조사할 필요가 없음을 의미한다. 적절한 하나의 flow 를 이루는 packet 의 수를 결정하는 것은 LCS 알고리즘을 이용하여 signature 를 생성할 때에 비교 횟수를 감소시켜 계산의 효율성을 증가시키기 위해 중요한 제약사항이다.

B. Substring 의 최소 길이

자동으로 생성된 signature 는 한 개 이상의 substring 들로 이루어지게 되는데, substring 의 길이는 정확한 트래픽 분류를 할 수 있는 신뢰할 수 있는 signature 로써의 중요도를 반영한다. 의미가 없는 substring 들이 signature 에 포함되는 것을 방지하기 위해 substring 의 최소 길이가 modified LCS 알고리즘의 제약사항로 고려되어야 한다. 예를 들어 하나의 character 가 common substring 중 하나로 결정 되었을 때에 이 character 가 fixed offset 에서 반복적으로 나타나지 않는 한, 실제 signature 에 포함시키기는 어렵다. 실제로 HTTP 프로토콜을 사용하는 어플리케이션들에서는 '/'와 같은 character 가 자주 쓰이는데, 본 제약조건을 통해 다른 string 들과의 조합이 아닌 이러한 character 하나만이 signature 에 포함되는 것을 방지한다.

C. Packet 크기의 비교

서로 비교되는 packet 들의 목적과 (예: signaling packet, download packet) 트래픽 특성이 비슷할 경우에 신뢰할 수 있는 signature 를 찾을 확률이 증가하게 된다. Packet 들 간의 유사성을 판단하기 위한 가장 기본적인 특성이 packet 의 크기 이다. 그림 1 은 세가지 P2P 어플리케이션들이 파일을 다운로드 시작할 때에 flow 상의 초기 100 개 packet 의 크기 분포를 나타내고 있다. 초기의 몇 개 packet 들은 connection handshake 나 다운로드 요청을 위한 packet 들이다. 이들의 크기는 실제 downloading packet 들보다 상대적으로 작다. 명확한 signature 는 초기의 signaling packet 에만 존재하므로 signature 를 생성하는 과정에서 크기가 작은 signaling packet 들과 크기가 큰 downloading packet 들을 서로 비교하는 과정은 필요가 없다. 결과적으로 효율성을 향상시키기 위해 비슷한 크기 범위 내에 있는 packet 들 사이에서만 비교가 이루어 지도록 하는 제약조건이 필요하다. 실제적인 예로 Gnutella 계열의 유명한 P2P 어플리케이션인 LimeWire 의 경우, connection control 이나 search 와 같은 signaling packet 의 크기는 평균적으로 390byte 인 것에 반해 downloading packet 의 size 는 1460byte 이다. 이러한 두 가지 종류의 트래픽들은 어플리케이션 내에서 수행하는 동작이 다르기 때문에 동일한 signature 를 포함할 확률이 적다.

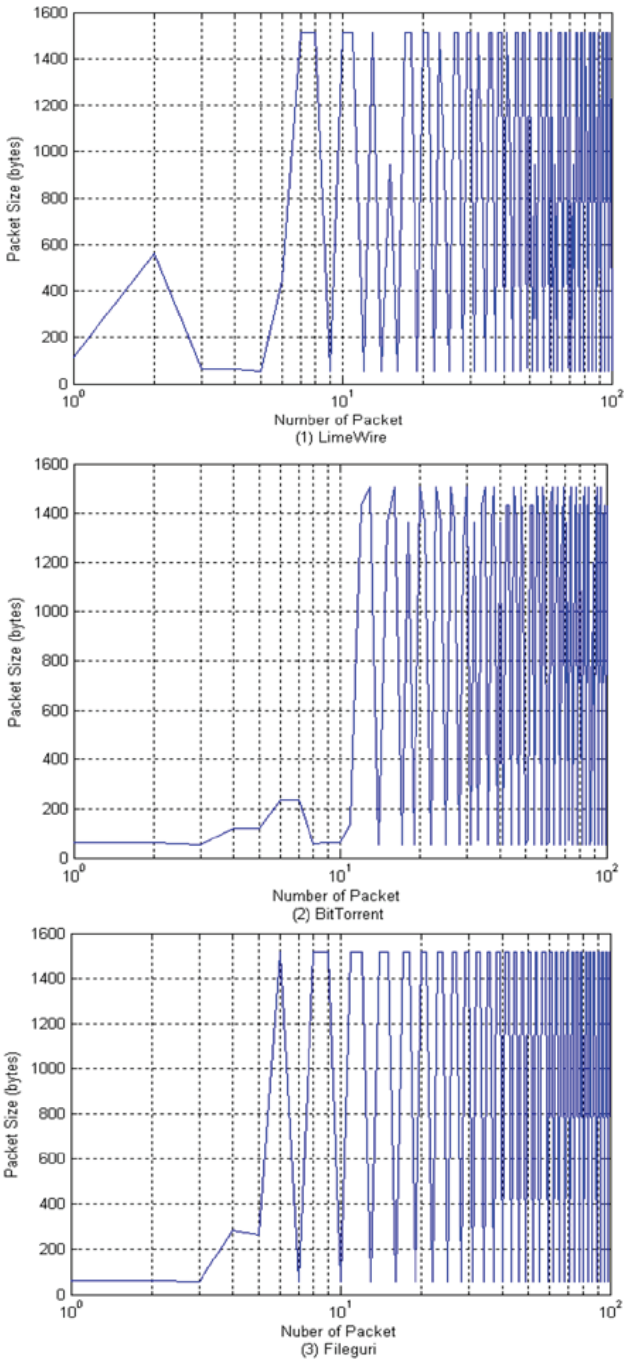

그림 1. 파일 다운로드를 시작할 때 생성되는 초기 100 개 packet 들의 크기 분포

3.3 Modified LCS 를 이용한 signature 생성

이번 장에서는 앞에서 설명한 제약사항들을 적용한 modified LCS 알고리즘을 통해 'sequence of common substrings' 포맷의 signature 를 생성하는 과정에 대해 설명한다. LCS 알고리즘의 두 개의 string 을 입력으로 사용하여 둘 사이의 longest common substring 을 찾아낸다. 여기에서 substring 은 두 입력 string 들에서 공통적으로 존재하는 작은 단위의 substring 을 의미하고 longest common subsequence (substring)는 두 입력 string 들에서 공통적으로 나타나는 substring 들이 동일한 순서를 보일 때에 이들 substring 들의 길이의 합이 최대가 되는 substring 들의 sequence 를 말한다. 이 때, substring 들은 연속적으로 나타나야만 할 필요는 없다.

입력 string 1 - XXX**AA**XX**BC**DXXX**EF**G**H**XX

입력 string 2 – YYYY**AA**Y**BC**YYYY**EFH**YYY

예를 들어 위의 두 개의 입력 string 이 주어졌을 때에 (bold 체는 common substring 을 의미), 두 string 간의 LCS 는 AA*BC*DF*H 와 같이 표현될 수 있다. substring 들은 각각의 입력 string 에서 서로 다른 위치에 존재할 수는 있지만 이들의 sequence 는 각 입력 string 에서 동일하게 유지 됨을 알 수 있다.

LCS 알고리즘에서는 substring 을 선택할 때에 first match 가 아닌 best match (longest length 를 갖는 match)가 되도록 선택한다. 이러한 선택의 기본 가정은 substring 의 길이가 길수록 두 입력 string 의 유사성이 높아질 수 있다는 것이다. 예를 들면, 다음과 같은 입력이 주어졌을 때에, first match 를 통해 substring 을 선택하면 LCS 는 A*A 형태를 갖게 된다.

입력 string 1 - **A**XXX**A**AXX

입력 string 2 – **AA**YYYYYY

하지만 LCS A*A 의 형태는 best match 로 선택한 LCS AA 보다 정확한 signature 가 될 수 없기 때문에 first match 가 아닌 best match 즉, substring 의 길이를 최대로 하는 LCS 를 생성하게 된다.

입력 string 1 - AXXX**AA**XX

입력 string 2 - **AA**YYYYYY

Modified LCS 의 입력으로는 서로 다른 두 개의 packet payload 상의 byte stream 이 사용되는데 각 packet 들은 같은 어플리케이션에서 발생한 두 개의 서로 다른 flow 에서 선택된다. 그림 2 는 두 개의 LimeWire flow 에서 packet 크기와 flow 를 이루는 packet 수에 관한 제약 조건을 적용하여 packet 을 선정한 후에 signature 를 생성하는 단계를 설명하고 있다. 각각의 byte stream 은 P2P 사용자가 Gnutella 프로토콜을 사용하는 Morpheus 와 LimeWire 서버에 접속하여 파일 다운로드를 시도할 때에 발생한 것이다. 그림에서 표현된 candidate signature 는 낮은 정확도를 갖는 완성되지 않은 signature 를 의미한다. 최종적인 signature 를 생성하는 과정에 대해서는 다음 장에서 자세히 설명한다.

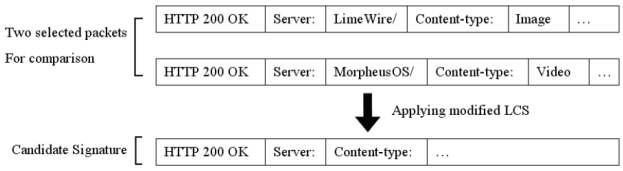

그림 2. LimeWire 에서 발생한 Packet 에 modified LCS 를 적용하는 예

두 개의 입력 byte stream 을 이용하여 LCS 를 구하면 **HTTP 200 OK Server:*e*Content-type:*e*** 과 같은 형태가 된다. 하지만 substring 'e'는 signature 로써 의미를 갖기에는 길이가 너무 짧기 때문에 이처럼 모호한 substring 을 substring 의 최소 길이에 관한 제약조건을 통해 제거한다. 이러한 과정을 거치면 **HTTP 200 OK Server:*Content-type:*** 와 같은 형태의 candidate signature 를 얻게 된다.

Payload 상의 string 은 모든 packet 에서 반복해서 나타나는 고정적인 string (예: HTTP)과 길이나 내용이 가변적인 string (예: 사용자 ID 또는 host name)으로 나누어 볼 수 있는데 signauture 에는 가변적인 string 이 포함되면 안되기 때문에 modified LCS 를 적용하여 이러한 string 들을 제거한다. 또한 'HTTP 200 OK'와 같은 substring 은 HTTP 프로토콜을 사용하는 어플리케이션에서는 자주 나타날 수 있기 때문에 그 자체로는 의미를 갖기 힘들지만 다른 substring 과 이루는 sequence 는 어플리케이션마다 서로 다른 특징이 있기 때문에 sequence 의 일부로써 의미 있는 substring 이 될 수 있다.

4. Signature 생성 시스템

이번 장에서는 3 장에서 설명한 modified LCS 알고리즘을 적용하여 최종적으로 어플리케이션 레벨의 signature 를 생성해내는 시스템들에 대하여 설명한다. 자동화의 정도에 따라 semi-automatic 시스템과 fully-automatic 시스템으로 나뉜다. Semi-automatic 시스템은 사용자의 개입이 일부 필요하지만 정확한 signature 를 생성한다. 이러한 system 에서 사용자의 개입이라는 단점을 보완하기 위해 fully-automatic 한 시스템을 개발하게 되었다. 이들 두 시스템간의 동작의 차이에 대해서는 다음에서 설명을 한다.

4.1 Semi-automatic 시스템

Semi-automatic signature 생성 시스템은 입력으로 사용될 데이터를 수집하는 과정에서 signature 생성을 원하는 어플리케이션을 독립적으로 실행시키고 트래픽을 capture 하는 사용자의 개입이 필요하다. 자동화 정도가 fully-automatic 시스템보다 낮지만 그보다 더 높은 정확도의 signature 를 생성하는 시스템이다.

A. 데이터의 수집

이 시스템의 입력으로 사용되는 데이터는 사용자에 의해 수동적으로 수집된다. Signature 를 생성하고자 하는 특정 어플리케이션을 독립적으로 실행시키고 해당 어플리케이션이 발생시키는 트래픽만을 선택적으로 수집한다. 특정 어플리케이션이 발생시키는 트래픽만을 수집하는 것은 다음에 설명할 트래픽 수집 agent 를 통해서도 가능하지만 이와 같은 방법을 사용하는 이유는 다음과 같다. 하나의 독립된 어플리케이션이 발생 시키는 트래픽은 어플리케이션의 동작에 따라 다른 특성을 띤다. 예를 들어 P2P 어플리케이션의 경우는 login, search, download 등과 같은 다양한 기능을 갖고 있는데 이러한 동작들이 수행될 때 발생되는 트래픽은 서로 다른 트래픽 패턴을 보인다. 만약 서로 다른 기능을 수행하는 트래픽을 입력으로 signature 를 생성할 경우에는 부정확한 signature 가 생성될 수 있으며 signature 가 전혀 생성되지 않는 경우가 생길 수 있다. 따라서 정확한 signature 를 생성할 수 있도록 수동적으로 어플리케이션에서 특정 동작을 할 때에 발생하는 트래픽을 선별적으로 수집하여 입력 데이터로 사용하는 것이다. 이 때에 수집된 트래픽은 동일한 5-tuple 정보를 갖는 flow 형태로 이루어진다.

B. Signature 의 생성

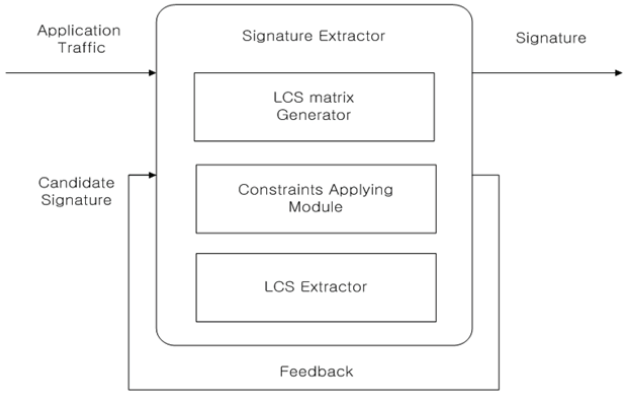

그림 3. Training model 을 이용한 최종 Signature 의 생성

최종적인 signature 는 candidate signature 에서 의미가 없는 string 들을 제거하는 과정을 통해 얻어진다. 그림 3 은 최종적인 signature 를 생성하기 위한 training model 을 설명하고 있다. 처음의 training 단계의 입력으로는 서로 두 개의 flow 가 사용된다. Signature extractor 는 modified LSC 알고리즘을 이용하여 candidate signature 를 생성하는 컴포넌트이다. 이 컴포넌트를 통해 처음으로 생성된 signature 는 두 개의 flow 만을 비교하여 생성된 것이기 때문에 정확성이 현저하게 떨어진다 (불필요한 string 들이 많이 포함되어있음). 이러한 문제를 해결하기 위해 위의 그림과 같은 training model 을 사용하여 training 이 반복 될 때마다 불필요한 string 들을 줄여나간다. 한번의 training 이 이루어질 때 생성되는 결과값은 전 단계의 결과값 보다 불필요한 string 들 (e.g IP address, user ID, host name, etc)이 줄어들지만 완벽한 signature 가 아니라면 다음 training 단계의 입력으로 재사용된다. 이와 같은 training 이 계속해서 반복될수록 불필요한 common substring 의 수는 감소하게 되며 점차 정확한 signature 에 가까워진다. 여기에서 반복적인 training 은 무한한 것이 아니라 특정 조건을 만족하게 되면 training 이 중지되며 최종 output 인 signature 를 얻을 수 있다. 특정한 종료 조건은 더 이상 candidate signature 에서 제거될 substring 이 존재하지 않는 것을 의미한다. Signature 생성의 단계를 다음과 같이 표현할 수 있다.

Candidate signature_1 = Sig(Input 1, Input 2)
Candidate signature_2 = Sig(Input 3, Candidate_signature_1)
…
Candidate signature_n = Sig(Input n+1, Candidate_signature_n-1)

만약 Candidate signature_n = Candidate signature_n-1 이라면 Candidate signature_n 은 최종 signature 가 된다.

C. 실험 결과

Modified LCS 알고리즘이 signature 생성에 있어서 유용함을 확인하기 위하여 세 가지의 어플리케이션들을 선정하여 각각의 signature 를 생성해보았다. 선정된 어플리케이션들은 대표적으로 많이 사용되는 P2P 어플리케이션 LimeWire, BitTorrent, 그리고 Fileguri 이다. LimeWire 는 Gnutella 프로토콜을 사용하는 대표적인 어플리케이션이며 BitTorrent 는 동시에 수많은 peer 들로부터 동시에 파일을 다운로드 할 수 있는 전세계적으로 매우 널리 사용되고 있는 어플리케이션이다. 두 어플리케이션은 다른 signature 기반의 트래픽 분류 연구 [5]에서도 사용되었던 어플리케이션이며, 마지막의 Fileguri 는 국내에서 많이 사용되는 어플리케이션으로 이 역시 다른 연구 [11]에서 사용되었다. Modified LCS 의 유용성을 판단하기 위해 다른 연구에서 수작업을 통해 생성한 signature 와 modified LCS 를 통해 생성된 signature 를 비교 분석하였다.

표 1 은 선정된 세가지 어플리케이션에 대해 각각 packet 분석, 프로토콜 분석, modified LCS 를 사용하여 생성한 signature 를 나타낸다. Packet 분석은

표 1. 세 가지 방법으로 생성된 signature 의 비교

	LimeWire	BitTorrent	Fileguri
Packet analysis	"GNUT"	"0x13Bit"	"Freechal"
Protocol analysis	'GET' or 'HTTP' followed "User-Agent: Limewire" or "UserAgent: Limewire" or "Server: Limewire"	"0x13BitTorrent protocol"	N/A
Modified LCS	Sequence of 24 substrings "GET", "uri-res", "urn", "sha", "HTTP", "HOST", "User-Agent", "LimeWire", "X-Queue", "X-Gnutella-Content-URN", "urn", "sha", "X-Alt", "Range", "bytes", "X-Node", "Chat", "X-Features", "fwalt", "browse", "chat", "queue", "X-Downloaded"	Sequence of 1 substring "0x13BitTorrent protocol"	Sequence of 6 substrings "GET", "HTTP", "Accept", "User-Agent", "Freechal", "P-Authentication"

어플리케이션의 프로토콜에 대한 사전 정보가 없이 raw packet 의 payload 를 직접 분석하여 signature 를 생성하는 방법을 의미하며 [2], 프로토콜 분석은 어플리케이션 프로토콜 정보를 분석하여 signature 를 생성하는 방법이다[2]. 따라서 프로토콜의 정보가 공개된 어플리케이션의 signature 를 생성한다면 정확한 프로토콜 정보를 확인하여 생성한 signature 가 가장 정확한 방법이라고 가정한다. 이러한 가정하에 각 방법들을 통해 생성한 signature 결과를 다음과 같이 정리하였다.

실제 실험에 사용한 modified LCS 알고리즘의 제약조건들은 다음과 같이 설정되었다. [5]와 그림 1 의 결과에 따라 flow 를 이루는 packet 의 수는 flow 상의 초기 10 개의 packet 으로 설정되었으며 substring 의 최소 길이는 3 으로 설정하였다. 마지막으로 비교될 packet 들의 크기 차이는 100byte 이하로 제안하였다. 이러한 제약조건들의 설정의 어플리케이션의 종류에 따라 조정이 가능하다.

LimeWire: packet 분석을 통해 생성된 signature 인 'GNUT' 는 다른 것들과 비교했을 때에 상대적으로 길이가 짧다. Signature 의 길이가 짧을수록 다른 어플리케이션의 signature 와 충돌이 될 가능성이 높아지므로 이러한 signature 를 이용하여 트래픽을 분류할 경우에는 정확도를 보장하기 어렵다. 프로토콜 분석을 통해 생성한 signature 와 modified LCS 가 만들어낸 signature 를 비교하면 modified LCS signature 가 다른 두 signature 들을 포함하고 있는 형태이다. 이러한 결과는 프로토콜 분석을 통해 signature 를 생성할 때 정확한 signature 가 될 최소한의 substring 들만을 선택하였기 때문이다. 실제로 'X-Gnutella-Content-URN'와 같은 string 은 LimeWire 의 프로토콜에 존재하는 값이다. Modified LCS 를 사용하면 이처럼 의도적으로 포함시키지 않은 string 들 또한 판단할 수 있다.

BitTorrent: packet 분석을 통해 생성된 signature 는 프로토콜 분석을 통해 생성한 signature 보다 길이가 짧지만 modified LCS signature 는 프로토콜 분석을 이용한 프로토콜과 동일한 signature 를 생성해 냈다.

Fileguri: 이 어플리케이션의 경우는 공개되지 않은 자체적인 프로토콜을 사용한다. 따라서 프로토콜 분석을 통한 signature 는 존재 하지 않는다. 나머지 두 signature 를 서로 비교하면 LimeWire 와 비슷하게 modified LCS signature 가 packet 분석을 통한 signature 를 포함하고 있는 형태이다.

다음으로 training model 의 효율성에 대해 분석을 한다. Modified LCS 알고리즘을 통해서 프로토콜이나 packet payload 에 대한 조사가 필요 없이 자동으로 signature 를 생성할 수 있지만 앞서서 설명한 것과 같이 반복을 통한 training 을 필요로 한다. 만약 training 을 위한 반복의 횟수가 많이 요구된다면 효율적이지 못한 것이 된다.

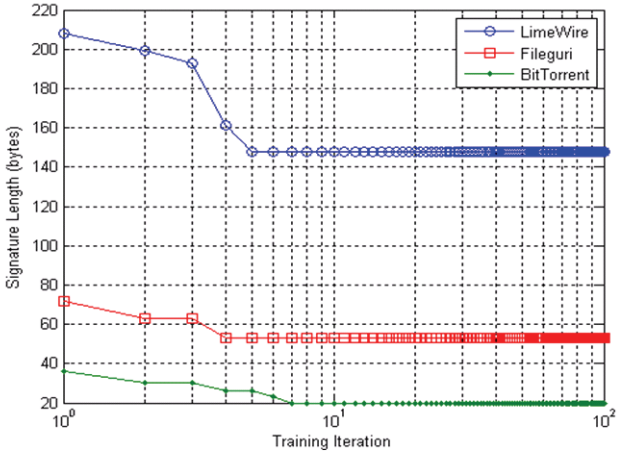

그림 4. Training 횟수에 따른 Candidate signature 길이 변화

그림 4 는 training 의 반복에 따른 candidate

signature 의 길이의 변화를 나타낸다. Training 의 횟수가 증가할수록 candidate signature 의 길이가 감소하는데, 이는 반복이 이루어질 때마다 불필요한 signature 들이 제거되는 것을 의미한다. Candidate signature 의 길이가 더 이상 줄지 않을 때에 training 과정이 종료되며 최종 signature 가 완성되는데, 그래프에서 확인할 수 있듯이 모든 경우에 있어서 10 회 이하의 training 을 통해 signature 가 생성되었다. 이는 적은 량의 입력 data 와 빠른 시간 내에 signature 를 생성해 낼 수 있음을 의미한다. 결과적으로 training 을 활용한 modified LCS 알고리즘은 수동적인 방법으로 signature 를 생성하는 것 보다 효율적으로 signature 를 생성할 수 있다.

4.1 Fully-automated 시스템

Semi-automated signature 생성 시스템은 자동으로 정확한 signature 를 생성할 수는 있지만 입력으로 사용할 데이터 수집에 있어서 사용자의 개입이 필요하다는 단점을 가지고 있다. 이러한 점을 보완하기 위해서 트래픽 수집을 자동적으로 수행할 수 있는 agent 를 탑재한 시스템이 요구되었다. 이 시스템은 기존에 사용되었던 modified LCS 알고리즘을 그대로 사용하면서 training 과정 대신 grouping 알고리즘을 통해서 signature 를 생성한다.

A. 데이터의 수집

Full-automated 시스템에는 input data 를 자동으로 수집할 수 있는 agent 가 탑재된다. Data 수집 agent 는 windows 기반으로 개발 되었으며 agent 가 탑재된 컴퓨터에서 발생하는 모든 트래픽을 어플리케이션 (또는 프로세스)별로 수집하는 기능을 수행한다. 이 때에 수집된 트래픽 역시 동일한 5-tupe 정보를 갖는 flow 형태로 이루어진다.

Agent 는 libpcap library 와 windows system call 을 이용하여 트래픽 capture 와 저장을 수행한다. Packet 이 capture 되었을 때 새로운 TCP 의 Syn flag 가 설정된 경우에는 새로운 세션이 맺어진 것으로 판단하고 어떠한 어플리케이션이 맺는 세션인지를 확인하기 위해 windows system call 을 통해 세션이 맺어질 때 사용한 네트워크 port 를 사용하고 있는 프로세스 정보를 운영체제로부터 받아오게 된다. 따라서 트래픽 수집 agent 는 자동적으로 어떠한 packet 이 어떠한 어플리케이션에서 발생한 것인지를 판단할 수 있고 트래픽을 어플리케이션 별로 수집한다.

이러한 agent 를 통해 트래픽을 수집하는 경우에는 특정 어플리케이션만을 독립적으로 실행시킬 필요가 없이 동시에 여러 어플리케이션의 트래픽을 수집할 수 있으므로 사용자가 수동으로 수집하는 방법에 비해 많은 량의 데이터를 빠르게 수집할 수 있다. 하지만 해당 트래픽이 어떠한 어플리케이션 동작 (e.g. login, search, download)을 위한 것인지를 판단하지 못하기 때문에 semi-automated 시스템의 training 과 다른 방법으로 signature 를 생성해야만 하는 문제가 따른다.

B. Signature 생성

Fully-automatic 시스템의 input 은 같은 어플리케이션에서 발생한 트래픽을 자동으로 수집한 것이지만 semi-automatic 시스템과 달리 하나의 어플리케이션에서 발생한 모든 종류의 트래픽이 섞여 있는 형태이다. 이러한 경우 입력으로 사용될 트래픽이 하나의 어플리케이션에서 발생한 것이지만 여러 종류의 트래픽 패턴이 존재하기 때문에 training model 을 그대로 사용한다면 signature 가 생성되지 않을 수 있다. 그리고 생성되어야 할 signature 또한 한 개가 아니라 어플리케이션이 가지고 있는 여러 동작의 종류에 해당되는 수가 생성되어야 한다.

Signature 를 생성하고자 하는 어플리케이션의 트래픽 데이터를 해당되는 동작 별로 구분 지을 수 있는 정확한 방법이 존재 하지 않기 때문에 modified LCS 에 의존하여 grouping 을 하는 방법을 사용하였다.

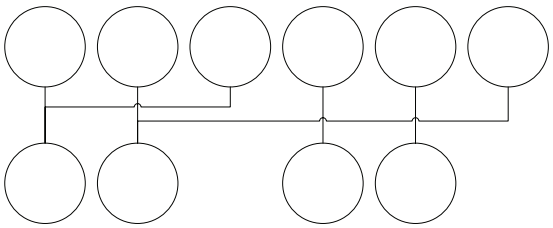

그림 5. Grouping 의 동작

그림 5 는 signature 생성을 위한 grouping 알고리즘의 동작을 설명하고 있다. 그림의 위쪽 줄에 표현된 flow 들이 시스템의 입력 값으로 사용되는 flow 들이며 'type'은 어플리케이션의 동작의 종류를 의미한다. 3 가지 종류의 서로 다른 동작을 하는 어플리케이션의 flow 6 개를 입력으로 사용한다. 그림의 아랫줄에 표현된 것은 입력으로 사용된 flow 들을 modified LCS 를 통해 grouping 한 후의 결과를 나타낸다. 여기서 하나의 group 은 여러 개의 입력 flow 들 가운데 LCS 가 가장 긴 (common substring 이 가장 많이 존재하는) 두 개의 flow 의 쌍으로 만들어진다. LCS 의 길이가 가장 길다는 것은 두 flow 간의 유사성이 가장 높다는 것으로 판단할 수 있다. Grouping 알고리즘을 통해서 나온 결과는 2 개의 동작 type 의 signature 이다. 'Type 1 signature'는 (flow 1, flow3)의 쌍에서 얻어진 signature 이며 'Type 2 signature'는 (flow 2, flow 6)의 쌍에서 얻어진 signature 이다. 나머지 쌍을 이루지 못한 flow 4 와 flow 5 는 결과에서 제외되게 된다. 그림 5 는 grouping 을 단순한 예제를 통해 설명한 것이지만 입력으로 사용되는 flow 가 많아지게 되면 더욱 복잡한 알고리즘이 필요하게 된다. N 개의 flow 가 입력 값으로 사용된다면 결과로 나올 수 있는 그룹 signature 는 최대 N/2 개가 되며 각각은 semi-automatic 시스템에서 한번의 training 단계를 거친

표 2. Grouping 알고리즘을 통해 생성한 signature

BitTorrent	Fileguri
0x13BitTorrent protocolex –BC (8)	HTTP Server Freechal User-Type Date Mon Feb GMT P-Authentication Content-Length Content-Type text plain path keyword extension file-count process-time download-waiting (4)
0x13BitTorrent protocolex (1)	
0x13BitTorrent protocol (2)	GET app asp HTTP Accept User-Agent Freechal Host www fileguri com Connection Keep-Alive (3)

것과 같은 형태를 띄게 된다. 따라서 이들을 optimize 할 필요가 있다.

Optimization 은 두 단계를 통해 이루어질 수가 있다. 첫 번째 단계는 어플리케이션의 동작에 따른 트래픽 별로 구분을 하는 것으로 어플리케이션 flow 들은 네트워크 port 가 할당된 range 별로 구분을 짓는 것이다. 어플리케이션이 랜덤한 방법으로 port 를 할당하지만 같은 동작을 수행할 때에는 비슷한 range 의 port 를 할당한다는 점에서 이러한 optimization 이 가능하다. 이 구분에 있어서는 100% 정확한 방법은 존재 하지 않는다. 다만 다음 단계에서 이루어질 optimization 을 좀더 단순화 시키기 위해 필요한 단계이다. 두 번째 단계에서는 grouping 의 결과로 생성된 n (< N/2)개의 signature 를 앞 단계에서 같은 동일한 port range 로 구분 지어진 group signature 끼리 다시 modified LCS 를 통하여 하나의 signature 를 만들어낸다. 이러한 경우 이상적인 결과는 어플리케이션의 여러 동작에 해당되는 signature 가 각각 생성되는 것이다. 만약 동작에 해당하는 트래픽을 구분 짓는 것이 완벽하지 않는다면 하나의 동작에 해당하는 signature 가 여러 개 존재 할 수 있지만 이들 모두를 signature 로 사용하여 트래픽을 분류할 수 있다.

C. 실험 결과

Modified LCS 와 grouping 알고리즘의 결과를 확인하기 위하여 training 모델에서 사용한 2 개의 어플리케이션의 signature 를 생성해보았다. 이 때 사용한 어플리케이션들은 BitTorrent 와 Fileguri 이다.

표 2 는 앞에서 설명한 optimization 과 grouping 알고리즘을 이용하여 두 어플리케이션의 signature 를 생성한 결과이다. 괄호 안의 숫자는 이러한 signature 를 공유하는 flow 쌍의 개수이다.

BitTorrent: 25 개의 flow 를 입력으로 이용해 생성한 signature 의 결과가 크게 3 가지 group 의 signature 로 종합되었다. Semi-automatic 시스템에서 생성한 signature 와 비교해 볼 때에, 3 가지 signature 모두 semi-automatic 시스템에서 생성한 signature 를 포함하고, 그 뒤에 다른 substring 이 붙어있는 형태이다. 더 이상의 optimization 이 불가능 할 경우에는 이들 3 개의 signature 모두를 이용하여 트래픽을 분류할 수 있다. Matching 비용이 단일 signature 보다 높지만 정확한 분류가 가능하다. 또한 BitTorrent 는 download 기능만을 가지고 있는 어플리케이션이므로 이들을 optimization 과정을 통해 정확한 signature 인 '0x13BitTorrent 프로토콜'이라는 정확한 signature 를 얻을 수도 있다.

Fileguri: 25 개의 flow 를 입력으로 사용하여 생성한 signature 의 결과가 크게 2 가지 group signature 로 종합되었다. Fileguri 의 signature 는 BitTorrent 와 달리 불필요한 string 들이 많이 포함되어있는 결과를 보였는데, 이는 Fileguri 가 HTTP 프로토콜을 사용하고 있기 때문에 signature 에 HTTP 에 자주 나타나는 common string 들 포함된 것이다. 또한 어플리케이션 내에 웹 브라우저가 포함되어 있어 단순한 html 파일의 내용이 signature 에 포함되었을 가능성 또한 있다. 하지만 packet 분석을 이용하여 생성한 signature 인 'Freechal'을 포함하고 있는 것을 확인할 수 있다.

두 어플리케이션에 fully-automatic 시스템을 적용해본 결과 HTTP 프로토콜을 사용하지 않는 어플리케이션의 경우 사용자의 개입 없이도 정확한 signature 를 생성할 수 있는 반면, HTTP 프로토콜을 사용하는 어플리케이션은 불필요한 string 들이 다수 포함됨을 확인하였다. 이러한 불필요한 string 들을 제외시키고 더욱 효과적으로 grouping 을 수행할 수 있도록 HTTP parser 를 시스템에 탑재시켜 HTTP 프로토콜을 사용하는 어플리케이션에 대해서도 정확한 signature 를 생성할 수 있도록 하는 연구가 진행중에 있다.

5. 결론 및 향후 과제

본 논문에서는 modified LCS 에 대해 설명하고 이를 기반으로 training model, grouping 알고리즘을 사용하여 자동적으로 어플리케이션 signature 를 생성할 수 있는 시스템들의 구현과 실험 결과를 제시하

였다. Semi-automatic 시스템은 어느 정도의 사용자 개입을 요구하는 대신에 매우 정확한 signature 를 효율적으로 생성할 수 있다. 이보다 자동화 정도를 높여 사용자의 개입이 없이 signature 를 생성할 수 있는 fully-automatic 시스템은 HTTP 프로토콜을 사용하지 않는 어플리케이션에 대해서 정확한 signature 를 생성하지만 다른 어플리케이션의 signature 는 상대적으로 완성도가 낮다.

이러한 문제를 해결하기 위해 HTTP parser 를 통한 filter 기능 추가에 대한 연구가 현재 진행 중이다. 또한 트래픽 수집 agent 에서 트래픽을 어플리케이션 종류별뿐만 아니라 수행하는 동작에 따라 수집할 수 있도록 한다면 semi-automatic 시스템에서 사용한 training 모델을 fully-automatic 하게 수정할 수 있기 때문에 이러한 기능의 data 수집 agent 의 관한 연구와 fully-automatic 시스템에서 다른 방법으로 정확한 signature 를 생성할 수 있는 효과적인 grouping 알고리즘과 optimization 알고리즘에 관한 연구도 향후 과제이다.

참고 문헌

[1] A. Gerber, J. Houle, H. Nguyen, M. Roughan, and S. Sen. “P2P The Gorilla in the Cable,” National Cable & Telecommunications Association (NCTA) 2003 National Show, Chicago, IL, June 2003.

[2] S. Sen and J. Wang. “Analyzing peer-to-peer traffic across large networks,” Proceedings of ACM SIGCOMM Internet Measurement Workshop, Marseilles, France, November 2002.

[3] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. “Measurement, modeling, and analysis of a peer-to-peer File-sharing workload,” Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP-19), Bolton Landing, NY USA ,October 2003.

[4] A. Moore and K. Papagiannaki. “Toward the Accurate Identification of Network Applications,” Passive and Active Measurements Workshop, Boston, MA, USA, March 31, April 1, 2005.

[5] Subhabrata Sen, Oliver Spatscheck, Dongmei Wang. “Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures,” New York, USA, WWW 2004 Conference.

[6] H. Kim, B. Karp. “Autograph: Toward automated, distributed worm signature detection,” Proceedings of the 13th Usenix Security Symposium, San Diego, CA, 2004.

[7] S. Singh, C. Estan, G. Varghese, S. Savage. “Automated worm Fingerprinting,” Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation, San Francisco, California ,2004.

[8] Sumeet Singh, Cristian Estan, George Varghese and Stefan Savage. “The EarlyBird System for Real-time Detection of Unknown worms”, UCSD Tech Report CS2003-0761, August 2003.

[9] W. Scheirer, M. Chuah. “Comparison of Three Sliding-Window Based worm Signature Generation Schemes,” , Lehigh University Technical Report LU-CSE-05-025.

[10] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and KC Claffy. “Transport layer identification of p2p traffic,” Internet Measurement Conference (IMC), Taormina, Sicily, Italy, 2004.

[11] Young J. Won, Byung-Chul Park, Hong-Taek Ju, Myung-Sup Kim, and James W. Hong. “A Hybrid Approach for Accurate Application Traffic Identification,” IEEE/IFIP E2EMON, Vancouver, April 2006, pp. 1-8.

[12] Tcpdump. http://www.tcpdump.org/.

[13] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, 2nd Edition, MIT Press, 2001.

[14] David Brumley, James Newsome, Dawn Song, Hao Wang, and Somesh Jha. “Towards Automatic Generation of Vulnerability Signatures,” In the Proceedings of the IEEE Symposium on Security and Privacy, May 2006.

[15] James Newsome, Brad Karp, Dawn Song. “Polygraph: Automatic Signature Generation for Polymorphic worms,” In IEEE Security and Privacy Symposium, California, USA , May 2005

[16] James Newsome and Dawn Song. “Dynamic Taint analysis: Automatic Detection, analysis, and Signature Generation of Exploit Attacks on Commodity Software,” In Network and Distributed Systems Security Symposium, San Diego, California, USA, 2005