https://doi.org/10.33778/kcsa.2022.22.3.087

시스템 기반 프로비넌스 그래프와 분석 기술 동향★

박 찬 일*

요 약

사이버 공격이 정교해지고 고도화된 APT 공격이 증가함에 따라 공격을 탐지하고 추적하기가 더 어려워졌다. 시스템 프로비넌스 그래프는 분석가들에게 공격의 기원을 밝히기 위한 기법을 제공한다. 사이버 공격에 대한 침투 기원을 밝히기 위해서 다양한 시스템 프로비넌스 그래프 기법이 연구되었다. 본 연구에서는 다양한 시스템 프로비넌스 그래프 기법을 조사하고 데이터 수집과 분석 방법에 관련해서 기술하였다. 또한 조사 결과를 바탕으로 향후 연구 방향을 제시해 본다.

A Survey on system-based provenance graph and analysis trends

Park Chanil*

ABSTRACT

Cyber attacks have become more difficult to detect and track as sophisticated and advanced APT attacks increase. System providence graphs provide analysts of cyber security with techniques to determine the origin of attacks. Various system provenance graph techniques have been studied to reveal the origin of penetration against cyber attacks. In this study, we investigated various system provenance graph techniques and described about data collection and analysis techniques. In addition, based on the results of our survey, we presented some future research directions.

Key words : Provenance Graph, ATP attack, Tracking, Origin of penetration

접수일(2022년 05월 25일), 수정일(1차: 2022년 08월 02일), 게재확정일(2022년09월14일)

★ 본 논문은 2022년 정부(방위사업청)의 재원으로 국방과학연구소의 지원으로 수행된 연구 결과임(912410301)

* 국방과학연구소(주저자/교신저자)

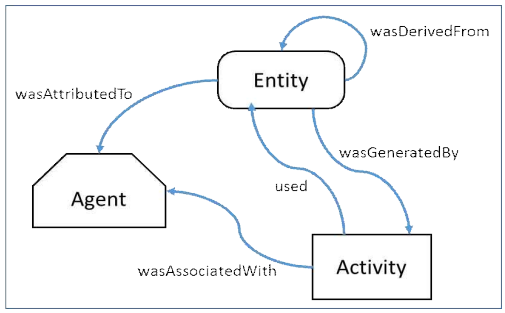
1. 서 론

최근에 사이버 공격이 고도화되고 APT 공격이 증가하면서 사이버 공격에 대한 분석이 더 어려워지고 있다. 특히, APT 공격은 오랜 시간 동안 지속해서 공격이 수행되고 있어 사후에 그 증거를 포착하기가 더 어렵다. 프로비넌스 그래프(provenance graph)는 분석가들에게 공격의 기원을 밝히기 위한 기법을 제공한다. 즉, 시스템에서 발생하는 로그 및 각종 이벤트 정보를 기록하였다가 추후 공격이 식별되었을 때 분석가들에게 공격의 시작점을 찾기 위한 증거를 제공한다. 하지만 평소에 많은 로그를 기록해야 하는 관계로 대용량 저장공간이 필요한 문제를 갖고 있다. 또한, 장시간 수행되는 프로그램의 경우 다른 많은 프로세스(process)와의 인과관계(causal dependencies)를 갖고 있어 인과관계 폭발(causal dependency explosion)과 같은 문제를 일으킨다. 특히, 대규모 조직에서는 각종 단말 PC와 서버에서 나오는 로그로 인해 복잡한 인과관계를 형성한다. 이는 분석가들이 자동화된 분석을 어렵게 하고 시스템에 영향을 끼치는 부담을 증가시키는 원인이 된다. 많은 연구에서 이러한 인과관계 폭발 및 저장공간에 대한 문제를 해결하고 분석가의 분석을 지원하기 위한 프로비넌스 그래프 시스템에 관련된 연구가 진행되어왔다. 프로비넌스 그래프를 위한 시스템은 크게 데이터 수집 부분, 데이터 저장 부분, 데이터 분석 부분 등 3가지 부분으로 구성된다. 본 연구에서는 프로비넌스 그래프를 위한 시스템의 각 부분별 연구를 조사하고 기술하고자 한다. 2장에서는 프로비넌스 그래프의 개념을 설명하고, 3장에서는 프로비넌스 그래프를 위한 데이터 수집 기술, 4장에서는 데이터 저장 관련 기술, 5장에서는 데이터 분석 관련 기술을 조사하였고 대표적인 기술을 간략히 설명하였다. 그리고 6장에서 결론을 기술하였다.

2. 프로비넌스 그래프

2.1 프로비넌스 그래프

프로비넌스 그래프는 데이터의 시작점 즉 출처를 확인하기 위해 설계된 구조로 데이터의 이력 정보 및 근원을 추적하기 위해서 Woodruff and Stonebraker[17]가 처음 제시하였다.



(그림 1) PROV Model Overview[15]

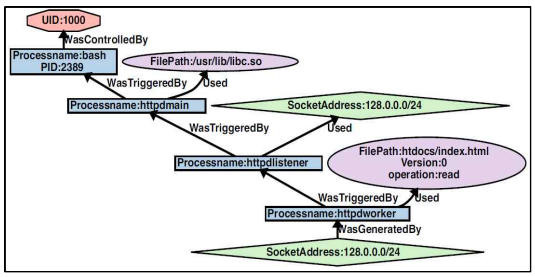
W3C(World Wide Web Consortium)[7] 에서 프로비넌스 그래프를 비순환 방향성 그래프로 정의하고 있다. 그래프 안에서 노드(node)는 데이터 즉 엔티티(entity), 데이터의 전송과 같은 활동(activity)하는 주체 또는 사람이나 조직을 나타내는 에이전트(agent)를 의미하고 엣지(edge)는 이런 노드들 사이의 관계를 의미한다[7].

W3C PROV(W3C provenance working group)[15]에서는 웹 및 다양한 정보 시스템에서 프로비넌스의 출판과 상호교환을 위해 데이터 모델을 만들었다. 이러한 프로비넌스 그래프는 다양한 분야에 응용되고 있는데 보안 분야에서는 APT 공격에 대응하는 방안의 하나로 시스템의 이벤트에 대한 출처를 추적하기 위해 많이 활용되고 있다. (그림 1)은 PROV 모델을 보여주고 있다. 엔티티(entity)는 물리적 실체 또는 개념적인 실체를 뜻하며 서로 다른 속성이 있다. 액티비티(activity)는 엔티티가 활동하는 방법과 그 속성을 의미하며 이전 엔티티의 활동으로 새로운 엔티티를 생성하기도 한다. 에이전트(agent)는 액티비티와 연관되어 있다.

2.2 시스템 프로비넌스 그래프

시스템 프로비넌스 그래프는 시스템 운영체제에서 발생하는 모든 이벤트와 관련하여 이벤트들의 인과관계를 그래프 모델을 이용하여 표현하는 그래프이다. 이를 통해 사이버 공격을 탐지한 순간부터 그 공격이 처음 시작된 기원을 추적할 수 있다. 이를 위해 시스템의 모든 객체(프로세스, 파일, 소켓 등)를 노드로 표시

하고 노드(node)들 사이의 모든 행위(operation)는 엣지(edge)로 표시한다.



(그림 2) 간단한 프로비넌스 그래프 샘플(아파치 웹서버가 싱글 사용자의 요청을 처리하고 있음)[16]

그래프 예시	그래프 설명
Process → File	파일 쓰기
Process ← File	파일 읽기
Process → Child Process	새로운 프로세스 생성
Process → Process	프로세스간 통신
Process → Socket	데이터 송신
Process ← Socket	데이터 수신

(그림 3) 일반적인 시스템 프로비넌스 그래프에서 이벤트 목록 및 표현 예시

(그림 2)는 간단한 시스템 프로비넌스 그래프의 예를 보여주고 있다. 시스템에서 수행되는 모든 행위는 다양한 도구를 이용하여 수집될 수 있다. 예를 들어 MS에서는 윈도우즈(windows) 시스템 개발자들이 개발 과정에서 디버깅 및 분석을 수행하고 시스템 레벨 이벤트를 추적하기 위해 ETW[9]라 프레임워크를 제공하고 있다. 리눅스(linux) 시스템에서는 별도로 Audit이라는 도구를 제공한다.

프로비넌스 그래프에서는 각 프로세스(process)와 프로세스, 프로세스와 파일(file), 프로세스와 소켓(socket) 등의 인과관계를 그 수행 동작에 따른 엣지로 연결한다. 각 프로세스가 파일을 읽고 쓰는 행위 및 자식 프로세스(child process)를 생성하는 행위, 프로세스가 통신을 위해 통신 소켓과 연결하는 행위 등 다양한 행위를 표시한다. (그림 3)은 일반적인 시스템 프로비넌스 그래프에서 사용하는 이벤트에 대한 그래프 표현 예시를 나타내고 있다. 이벤트 행위에는 항상 타임스탬프(timestamp)를 갖는다. 이는 이벤트가 수행된 순서를 나타낼 수 있다. 또한 행위를 나타내는 엣지에는 방향성을 갖는다. 이를 통해 데이터나 컨트롤에 대한 흐름을 나타낼 수 있다.

3. 데이터 수집 기술

시스템 프로비넌스 그래프를 생성하기 위한 가장 기본적인 일은 호스트 장비에서 로그 데이터를 수집하는 것이다. 따라서 다양한 연구에서 로그 수집을 위한 이벤트 데이터 수집 기법을 제시하고 있다. 데이터 수집 방법에는 크게 코스그레인드(coarse-grained) 수집과 파인그레인드(fine-grained) 수집의 방법이 있다[12].

코스그레인드 수집은 시스템 레벨(system level)에서 프로세스, 파일들에 대한 동작 등 주요 운영체제가 제공하는 이벤트 로그들을 수집한다. 리눅스 시스템의 경우는 Auditd, 윈도우즈 시스템의 경우 ETW[9]가 로그 수집기능을 제공한다. 또한 비츠(BITS)[19] 및 FUSE[8] 같은 이벤트 수집 도구들은 운영체제에서 제공하는 로그들을 효과적으로 수집하는 기능을 제공하고 있고 SPADE[20], CamFlow[21] 등의 연구에서 다양한 수집 모듈을 이용하여 로그를 수집하는 기법을 제시하였다. 하지만, 운영체제에서 제공하는 데이터는 운영체제의 동작 오류 및 보안 문제를 해결하기 위한 로그 기록을 제공하는 데 중심이 되고 있어 시스템 프로비넌스를 위한 로그로 사용하는 데는 정확성이나 및 제공하는 정보가 부족하다.

파인그레인드 수집은 운용체제의 커널단에서 각 프로세스 또는 파일의 동작을 수집한다. 이를 위해서는 별도로 커널단에서 수집을 위한 도구들이 필요하다. 파인그레인드 수집은 수집을 위한 기법에 따라 실행 유닛(exetution unit), 오염분석(taint analysis), 인과관계추론(cause inference) 등의 기법이 있을 수 있다. 실행 유닛 기법은 프로그램의 실행을 논리적인 단위로 나눠서 독립 유닛으로 로그를 관리하는 기법이다. 이를 통해 특정 프로세스에 집중되는 로그를 분리함으로써 의존성 폭발을 줄여줄 수 있다. 하지만 실행 유닛을 구분하기 위해서는 소스코드와 같은 추가적인 정보가

필요할 수도 있다. 오염분석 기법은 중요 데이터에 태그(tag)를 붙이고 이 정보가 퍼져나가는 것을 수집하는 기법이다. 분석가는 태그를 추적함으로써 보다 세부적인 분석을 할 수 있다. 하지만 태그를 위해 추가적인 오버헤드가 발생하기 때문에 태그 기반 분석에 한계가 있다. 인과관계 추론 기법은 로그 정보를 생성하는 인과성에 기반을 두고 수집하는 기법이다. 하지만

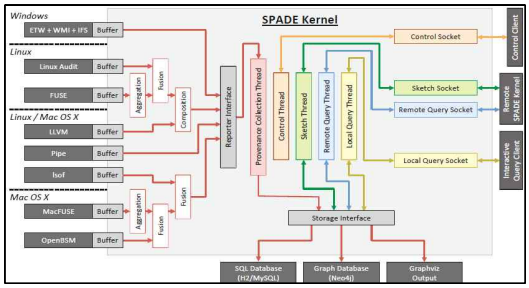
<표 1> 다양한 데이터 수집 기법 비교

수집기법		수집기술	정확도
Coarse Grained	System Level	SPADE	Low
		CamFlow	Low
Fine Grained	Execution Unit	BEEP	Mid
		LogGC	Mid
	Taint Analysis	RAIN	High
		RTAG	High
	Causality Inference	LDX	Mid
		Winnower	Mid

일반적인 인과관계만으로 정보를 수집하기에는 한계가 있어 실행 유닛 및 다양한 모델과 융합하여 수집하고 있다. <표 1>에서는 다양한 수집기법별로 수집기술에 관해서 정리하고 있다. 또한, 각 수집기술에 대해서 정확도를 제시하고 있다. 일반적인 시스템에서는 시스템 레벨에서 수집된 로그보다는 커널 기반의 정보를 수집하는 것이 더 세부적인 정보를 수집할 수 있어서 시스템 프로비넌스 그래프의 생성 및 분석에 있어 더 정확한 정보를 제공할 수 있다.

3.1 시스템 레벨 수집

SPADE[20]는 오픈소스 기반의 소프트웨어로서 데이터 기원 및 수집 관리를 위한 도구이다. SPADE는 그래프 기반의 도구로 공개 프로비넌스 모델(open provenance model)[35]을 표현하기 위해 기본적인 데이터 수집 모델로 버틱스(vertics)와 엣지를 사용하였다. SPADE는 리눅스, 안드로이드(android), 맥(mac) OS X 등 다양한 시스템에서 이벤트를 수집하기 위한 모듈을 제공한다. 선마이크로시스템즈(sun microsystems)에서는 시스템 레벨의 이벤트 추적을 위해서 BSM을 제시하였다. 이것은 오픈소스로서 OpenBSM으로 공개되었다. Mac OS X에서는 이러한 프레임워크를 도입하여 실시간으로 자체적인 이벤트를 추적하는 시스템을 제공하고 있다. (그림 4)에서는 SPADE 플랫폼을 보여주고 있다. 윈도우즈 시스템뿐이 아니라 리눅스, 맥 등 다양한 시스템에서 제공하는 로그 수집 도구를 활용한다.



(그림 4) SPADE 플랫폼[20]

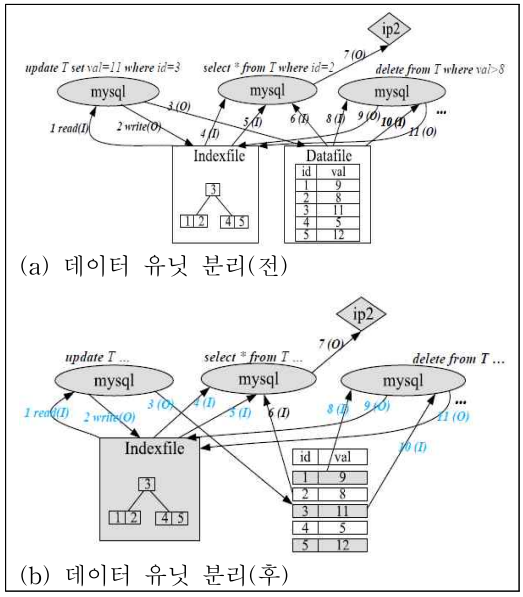
CamFlow[21]에서는 리눅스 OS에서의 수집을 위해 LSM(Linux Security Module) API를 사용한다. LSM은 보안 수행을 위해 2가지 타입을 구현했다. 첫째 커널 객체(object)가 할당되었을 때이고 두 번째는 커널 객체(object)가 액세스 되었을 때이다. 또한 CamFlow는 입력 socket_sock_ rcv_skb를 통해 패킷(packet)을 수집한다. CamFlow에서는 프로비넌스 그래프를 위해 패킷을 엔티티로 식별하였다.

3.2 실행 분할(Execution Partition) 수집

실질적인 환경에서 시스템 레벨에서 로그를 수집하는 것은 저장해야 할 로그가 엄청나게 많다는 것이다. 하루에 수 기가바이트씩 로그 용량이 늘어날 수 있다. 특히, 장시간 실행되어야 하는 서버 프로그램의 경우 다수의 입출력으로 의존선(dependency line)이 폭발적으로 늘어나 기원 분석을 복잡하게 만든다. 이러한 복잡한 분석을 해결하기 위해 BEEP 기법[5]이 제안되었다. BEEP 기법은 프로그램을 유닛(unit)이라는 논리적인 단위로 분리해서 처리한다. 유닛은 프로그램의 실행을 개별 요청을 다루는 논리적인 실행 루프를 기반으로 분할 한 것으로 이를 통해 의존성 폭발 없이 바이너리 프로그램의 로깅(logging)을 사용할 수 있다.

예를 들면 메일서버에서 이메일 요청이나 웹 브라우저에서 특정 URL 접근하거나 URL에서 수행되는 동작을 하나의 루프(loop)로 나눠서 독립 유닛으로 구분해서 로그를 수집한다. 이를 통해 시스템 프로비넌스 그래프의 각 노드에 연결된 엣지의 수를 감소시킬 수 있었다.

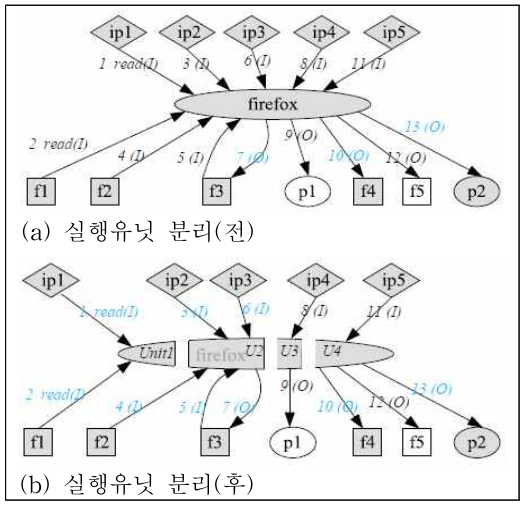
LogGC[22]는 의존성 폭발 문제를 해결하기 위해 가비지(garbage) 수집을 이용한 기법으로 제시되었다.



(a) 데이터 유닛 분리(전)

(b) 데이터 유닛 분리(후)

(그림 5) 데이터 유닛 분리[22]



(a) 실행유닛 분리(전)

(b) 실행유닛 분리(후)

(그림 6) 실행유닛 분리[22]

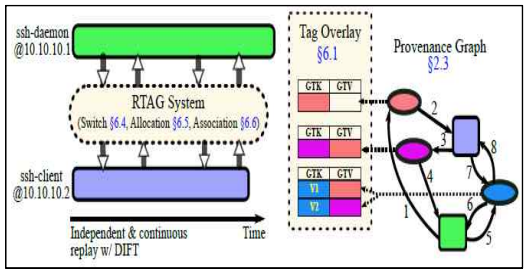
가비지 수집의 효율을 높이기 위해 LogGC에서는 실행 유닛(execution units) 기법과 데이터 유닛(data unit) 기법을 적용하였다. 실행 유닛은 하나의 프로세스를 개별의 요청을 다루는 논리적인 실행 유닛으로 나누어서 다룬다. 데이터 유닛은 파일에 대한 접근을 의존성이 없는 더 작은 데이터 유닛들로 나눈다. (그림 5)와 (그림 6)은 인터넷 웹 브라우저인 Firefox 프로그램 및 데이터베이스 도구인 MYSQL에서의 실행유닛 기법과 데이터 유닛 기법을 예시로 보여주고 있다. 실행 유닛 및 데이터 유닛 분리를 통해서 하나의 프로세스가 여러 노드로 분리됨으로써 불필요한 의존성 폭발과 연관을 줄여줄 수 있었다. 이후, ProTracer[23], UIScope[24] 등 다양한 연구에서 실행 유닛 기법을 적용한 연구가 수행되었다.

3.3 오염분석(Taint Analysis) 수집

다수의 호스트에서 일어나는 공격을 조사하면서 의존관계 폭발로 인해 중요 파일 데이터나 네트워크 정보, 메모리 정보 등이 무시되곤 한다. RAIN[25]에서는 의존성 폭발(dependency explosion)을 해결하기 위해서 DIFT(Dynamic Information Flow Tracking)[25] 기술을 이용하였다. DIFT 기술은 프로그램이 실행되면서 내부의 정보 흐름을 분석하는 기술이다. 이를 위해 프로그램의 흐름 중 중요 데이터에 태그(tag)를 붙이고 이 정보가 퍼져나가는 것을 수집한다. RAIN에서는 기존 DIFT에서 모든 프로세스에 대한 로그를 수집하는 대신에 프로세스 수를 줄이고 관련이 없는 프로세스를 제거하기 위해서 시스템 콜 수준의 도달도(reachability) 분석을 수행하였다. 하지만 하나의 호스트를 대상으로 정보를 수집하고 있어 다수의 호스트에서 일어나는 공격을 조사하기에는 부족함이 있다.

RTAG[26]에서는 다수의 호스트에서 일어나는 공격을 조사하기 위한 기법을 제시하였다. RTAG는 세 가지 기술을 제시하였는데, 첫째, record-and -replay 기술을 사용하였다. 이것은 서로 다른 데이터 흐름 태그 사이에 의존성을 제거한다. 둘째, RTAG는 시스템 콜 수준의 프로비넌스 그래프 정보를 활용한다. 마지막으로 RTAG는 네트워크 데이터 패킷에서 태그를 사용하여 호스트 사이의 데이터 흐름을 수집한다. 이를 통해 RTAG는 DIFT 기반의 분석에 사용된 메모리와

분석에 사용된 부담을 최대 90%까지 감소시킬 수 있었다. (그림 7)에서는 RTAG 모델에서 독립된 병렬 DIFT 동작을 보여주고 있다. 태그 스위치, 할당, 연계 기법 등을 활용하여 태그 의존성을 분리함으로써 오프라인 분석에서 각 프로세스가 독립적으로 동작할 수 있도록 하였다.

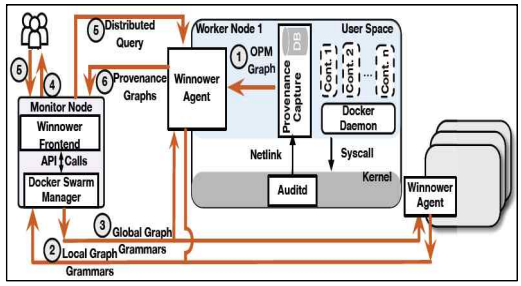


(그림 7) RTAG 독립된 병렬 DIFT[26]

3.4 인과관계 추론(causality inference) 수집

인과성 추론에 관한 다양한 연구가 수행되어 왔다. Kwon et al.은 새로운 인과관계 추론 기법을 제시하였고 이를 적용한 LDX[27]라는 인과관계 추론 엔진을 제시하였다. 이를 위해, 기존의 인과관계에 기초한 프로그램 의존성의 한계를 연구하였고 새롭게 조건법적(counterfactual) 인과관계 기법을 적용하였다. 하지만, 제시된 LDX 기법을 활용하기 위해서는 소스 코드에 대한 접근이 필요하다는 제약이 따른다.

지금까지 많은 침입탐지 도구가 제시되었지만, 대규모로 분산된 시스템에서 대량의 데이터를 중앙 노드에 보내는 문제나 많은 데이터에서 하나의 침입을 찾아내는 문제 등 침입을 탐지하는 것은 여전히 어려운 문제이다. Hassan et al.[16]은 이를 해결하기 위해 Winnower[16]라는 시스템을 제안하였다. 대규모 분산 시스템에서는 다수의 시스템을 운용하는 데 있어 높은 수준으로 유사한 로그를 생성한다. 왜냐하면 대규모 분산 시스템에서는 관리의 효율성을 위해 비슷한 환경을 설정하고 유사한 작업을 수행하고 있기 때문이다. 결과적으로 시스템별로 유사한 프로비넌스 그래프 구조가 생성된다. (그림 8)은 Winnower의 시스템 구조 및 워크플로우를 보여주고 있다. Winnower에서는 각 시스템에 공통된 프로비넌스 그래프 모델을 생성함으로써 불필요한 로그를 줄일 수 있다. 또한 시스템 사이에 공통으로 발생하는 이벤트에 대해서 신뢰수준(confidence level)을 나타내는 지표를 적용하여 한 번만 일어나는 이벤트에 낮은 값을 할당함으로써 공격 이벤트가 발생 되었을 경우 조기에 발견될 수 있도록 하였다.



(그림 8) Winnower 시스템 및 워크플로우[16]

4. 데이터 저장 기술

4.1 데이터 저장 모델

시스템 프로비넌스 그래프를 생성하기 위해서는 많은 데이터를 수집해야 한다. 기업이나 조직이 커짐에 따라 저장해야 할 데이터 용량은 기하급수적으로 늘어난다. 따라서 이러한 대용량 데이터를 저장할 모델도 중요한 문제이다. 기존의 데이터 모델에서는 대용량의 데이터 저장을 위해 오라클과 같은 DB 시스템을 사용하였다. 하지만 다양한 형태의 데이터 저장에 한계가 있어 최근에는 NoSQL 처리에 적합하고 빅데이터 처리에 강한 ElasticSearch[18]와 같은 빅데이터 DB를 많이 사용하고 있다. 또한 빅데이터를 이용한 시스템 프로비넌스 그래프에서는 데이터들이 노드와 엣지로 구성된 그래프 형태로 저장된다. 따라서 이러한 처리에 특화된 그래프(graph) DB들이 사용된다. 하지만 현존하는 그래프 DB들은 분석을 위해 그래프 전체를 메인 메모리에 올려야 하는 단점을 가지고 있다. 이러한 제약은 분석을 용이하게 하지만 많은 메모리가 필요하다. 특히 수많은 호스트에서 생성된 로그를 저장하는 시스템 프로비넌스 그래프에서는 분석을 위해 그래프 전체를 메인 메모리에 올리는 것은 큰 부담이다.

SLEUTH[4]에서는 그래프 DB의 저장 용량을 줄이기 위해 다양한 방법을 사용하였다. 기본적으로 시스

템 로그의 이벤트에 대한 식별자를 64bit 식별자 대신해서 32bit 식별자를 사용하였다. 또한, 이벤트의 주체와 객체를 표현하는 데이터 구조를 압축적으로 표현하였다. 예를 들어, 이벤트에 포함된 타임 스탬프의 경우 동일한 주체에서 발생한 경우 상대적인 시간으로 표현하였다. 다양한 기법을 통해 SLEUTH에서는 다수의 이벤트에 대해서 거의 6byte로 표현할 수 있었다.

UNICORN[3]에서는 지속적인 데이터 업데이트와 고정된 사이즈, 그리고 장기적인 그래프 데이터 구조를 위해 연속적인 그래프 히스토그램을 만드는 기법을 사용하였다. 이것은 시스템 실행의 전체적인 기록을 표현한다. 지속적 분석을 통해 시스템에서 발생하는 이벤트에 대해서 인과관계를 분석한다. 또한, 히스토그램에서 최신 이벤트와 인과관계가 없는 항목을 제거함으로써 점차 히스토그램의 영향을 제거한다.

그래프 DB는 그래프 형태의 데이터를 추적하는 데 유용하지만 많은 메모리를 필요로 하다는 한계가 있다. 그래프 DB만으로는 데이터 저장의 한계를 가지고 있다고 할 수 있다. 따라서, 그래프 DB에 핵심적인 추적 데이터를 저장하고 세부적인 데이터를 기존의 관계형 DB에 저장함으로써 메모리에 올라오는 데이터의 양을 줄이고 분석을 용이하게 할 수 있을 것이다.

4.2 데이터 저장 용량 감소 기술

사이버 공격이 점점 더 지능화되어 가면서 각 공격은 더 복잡해지고 탐지하기 어려워졌다. 따라서 사이버 공격에 대응하기 위해 더 많은 데이터를 분석할 필요가 생겼다. 하나의 APT 공격을 수행하는데 평균적으로 188일이 소요된다고 여겨진다[12]. 이처럼 장기간에 걸쳐 공격이 점진적으로 진행되기 때문에 탐지가 된 순간에는 시스템에 남아 있는 로그들이 많이 사라진 이후가 된다. 이는 분석가에게 공격의 진행 흔적을 찾는 것을 더 어렵게 만든다. 따라서 APT 공격과 같은 장기적인 공격을 분석하기 위해서는 평시에 시스템에서 발생하는 데이터를 수집하고 저장해 놓고 있어야 한다. 평균적으로 하나의 호스트에서 발생하는 이벤트 용량은 1.2GB/day를 초과하는 것도 있다[22]. 특히, 조직 및 기업이 커질수록 사용하는 호스트(서버 및 PC) 대수도 많아지고 따라서 수집하고 분석해야 할 데이터 양도 기하급수적으로 늘어간다. 예를 들어, 국방과 같은 환경에서는 수만 대의 호스트를 갖추고 있다. 단순 계산으로도 수천 테라바이트를 넘어가는 데이터를 저장하고 있어야 한다. 최근 빅데이터 분석 기술이 많이 발전했다고 하지만 대규모의 데이터 분석에는 상당한 시간과 인력이 필요하다. 또한, 대규모 조직의 경우 급격히 늘어가는 데이터의 수집 및 저장에 큰 비용이 소모된다. 따라서, 수집되는 데이터의 저장 용량을 줄이려는 방안이 필요하다. 일반적으로 저장공간을 확보하기 위해 데이터 용량을 줄이기 위해서는 압축 기술이 있을 수 있다. 하지만 압축 기술은 단순히 데이터를 보관하기 위한 용도가 우선시되고 시스템 프로비넌스 그래프와 같이 실시간적인 분석을 위해서는 적합하지 않다. 시스템 프로비넌스 그래프를 위한 데이터는 크게 노드와 엣지로 구성되어 있다. 따라서, 프로비넌스 그래프의 용량을 줄이기 위해 단순한 압축보다는 그래프에 사용되는 노드 또는 엣지를 줄이거나 이들의 데이터를 압축해서 표현하는 기술이 우선시 되어야 할 것이다.

Xu et al.[10]은 인과관계를 유지하면서 데이터를 줄이기 위한 기법을 제시하였다. 프로비넌스 그래프에서 인과관계 분석은 기본적인 기법이다. 따라서 최종 프로비넌스 그래프에서 인과관계를 분석하기 위한 능력을 유지해야 한다. 제시된 기법에서는 최종 객체(object)에서 쓰거나 읽기가 없어 의미 없이 반복되는 두 객체 사이의 읽기/쓰기를 제외시켰다[10]. 이를 통해 프로비넌스 그래프의 인과관계 분석은 유지하면서 최종 데이터양을 줄일 수 있었다. 하지만, 제시된 기법은 읽고/쓰는 것과 같은 접속 정보를 삭제함으로써 통계 정보를 생성하지 못한다는 단점이 있다[12].

Hossain et al[34]은 수집된 이벤트를 이용하여 생성하는 프로비넌스 그래프에서 엣지를 줄이는 기법에 대해서 제시하였다. 이를 위해 완전 의존성 보존(full dependence preservation, FD)와 소스 의존성 보존(source dependence preservation, SD) 기법을 제시하였다. 해당 기법을 통해 Xu et al보다 더 많은 엣지를 줄여서 표현하였다. 즉, Xu et al에서는 같은 노드쌍에 대해서 2개의 엣지를 줄이는 데 그쳤지만 Hossain et al[34]은 모든 엣지를 압축해서 표현할 수 있었다.

Chapman et al[11]. 은 프로비넌스 추적을 위한 데이터를 저장하기 위한 기법을 제시하였다. 다수의 호

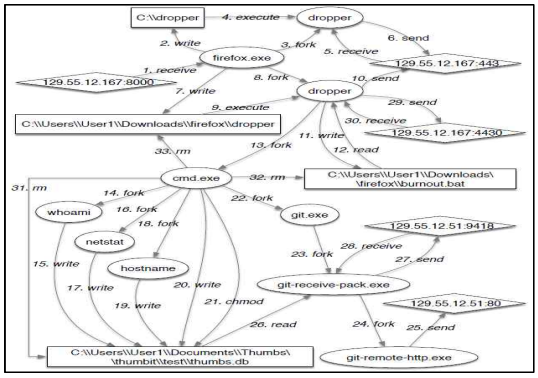
스트로 구성된 조직에서 많은 로그가 유사한 프로비넌스를 가지고 있다. 이러한 유사한 로그들을 하나만 저장하고 나머지는 삭제하는 프로비넌스 팩토라이제이션(factorization) 기법을 제시하였다.

5. 데이터 분석 기술

시스템 프로비넌스 그래프는 시스템에서 발생하는 이벤트 및 로그에 대해서 인과관계를 연결하는 그래프이다. 시스템 프로비넌스 그래프를 통해 사이버 공격에 대한 전체적인 흐름을 식별할 수 있다. King et al.[6]는 프로비넌스 그래프에서 백트래킹하기 위해 BackTracker를 제시하였다. BackTracker는 하나의 탐지 지점(예를 들면, 의심스러운 파일 또는 프로세스)에서 연속된 이벤트들을 추적하여 공격에 사용된 잠재적인 이벤트들을 자동으로 탐지하는 것이다. 기본적인 의존성 그래프에서 이벤트들의 연결된 체인을 나타낼 수 있다. 하지만, 이벤트 체인의 복잡성으로 인해 사이버 공격에 대한 시작점을 찾는데 제약이 되었다. Sitaraman et al.[2]은 이런 제한을 극복하기 위해 읽기와 쓰기 이벤트에서 오프셋(offset) 주소를 참조하는 기법을 제시하였다. 단순히 읽기와 쓰기를 수행한 파일명에서 읽거나 쓴 부분에 대한 오프셋 주소를 참조함으로써 이벤트들을 역으로 추적하는 범위를 축소할 수 있었다.

사이버 공격을 추적하기 위해서는 빠른 추적 수행과 높은 정확도가 요구된다. 하지만, 시스템 프로비넌스 그래프의 크기로 인해 많은 저장공간과 계산력이 요구된다. 지금까지 사이버 공격을 추적하기 위한 다양한 데이터 분석 연구가 수행되어 왔다. 연구된 기술들을 분류하면 다음과 같이 태그 기반 분석, 패턴 매칭 기반 분석, 추상화 그래프 기반 분석, 실행기반 분석 등으로

나눌 수 있다. 태그 기반 분석은 시스템에서 실행 기록을 태그로 저장하고 인과관계 추적을 위해 태그의 전파를 분석한다. 이를 통해 분석가가 우선순위 및 특정 대상을 집중할 수 있도록 한다. 패턴 매칭 기반 분석은 그래프에서 부분적인 패턴을 비교해서 의심스러운 행위를 추적하는 기법이다. 추상화 그래프 기반 분석은 기존의 로우 레벨 프로비넌스 그래프의 복잡성을 극복하기 위해 상위 레벨의 추상화된 그래프를 이용한 분석 기법이다. 실행기반 분석은 노드들의 인과관계 폭발을 막기 위해 분석 주체를 세분화하여 각 노드에 연결된 인과관계 폭발을 해결하기 위한 기법이다. 각 기법과 관련된 주요 연구는 다음과 같다.



(그림 9) 윈도우 시스템에 대한 공격의 프로비넌스 그래프[4]

Dataset	Entry Entities	Programs Executed	Key Files	Exit Points	Correctly Identified Entities	Incorrectly Identified Entities	Missed Entities
W-1	2	8	7	3	20	0	0
W-2	2	8	4	4	18	0	0
L-1	2	10	7	2	20	0	1
L-2	2	20	11	4	37	0	0
L-3	1	6	6	5	18	0	0
F-1	4	13	9	2	13	0	1
F-2	2	10	7	3	22	0	0
F-3	4	14	7	1	26	0	0
Total	19	89	58	24	174	0	2

(그림 10) 다양한 환경에서 APT 공격에 대한 SLEUTH 탐지 요약[4](W : 윈도우, L: 리눅스, F: FreeBSD 환경)

5.1 태그 기반 분석

태그 기반 분석은 시스템 로그를 기반으로 로그 이벤트의 주체(subject), 객체(object) 그리고 이벤트(event)를 식별하기 위해 태그 기반의 접근을 사용하는 기법이다. 이런 태그 기반 접근을 통해 분석을 위한 우선순위와 집중을 할 수 있게 해준다.

SLEUTH[4]는 플랫폼 중립적이고, 메인 메모리 기반의 의존성 그래프를 이용한 기술을 개발하였다. SLEUTH에서는 로그 기반의 의존성 그래프를 메인 메모리에서 처리될 수 있도록 효과적인 이벤트 저장과

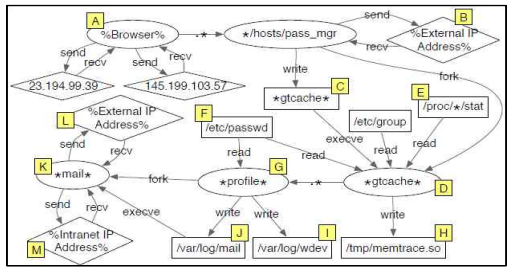
분석을 제공한다. 또한 태그 기반의 접근법을 사용하여 효과적으로 공격과 관련된 주체, 객체, 이벤트들을 식별할 수 있도록 하였다. SLEUTH에서는 그 역할에 2가지 태그를 사용하였다. 즉, 신뢰성(trustworthiness tags, t-tag) 태그와 기밀성(confid- entiality tags, c-tag)로 식별하였다. SLEUTH에서는 낮은 신뢰성을 갖는 노드가 높을 기밀성을 갖는 노드에 접근할 때 위협에 대한 알람을 생성한다. 이러한 태그의 분류는 분석가들이 집중하고 어떤 것에 우선순위를 둘지 식별할 수 있게 하였다. (그림 9)에서는 APT 공격에 대해서 윈도우 시스템의 로그 데이터를 기반으로 SLEUTH에서 자동으로 그려준 시스템 프로비넌스 그래프를 보여주고 있다. SLEUTH에서는 다양한 환경에서 APT 공격의 실험을 통해 효과를 APT 공격을 탐지하는 것을 보여주었다(그림 10). 하지만 태그 기반 접근법은 여전히 의존성 폭발의 문제를 일으킨다[14].

5.2 패턴 매칭 기반의 분석

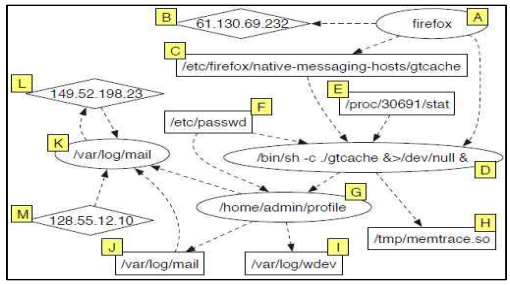
복잡한 시스템 프로비넌스 그래프에서 사이버 공격을 추적하기 위해서는 고성능의 컴퓨팅 파워가 요구된다. 하지만, 시스템 프로비넌스 그래프의 서브 그래프는 특정 악의적인 행위에 대한 특징을 구별할 수 있다. 따라서, 특정 행위에 대한 서브 그래프의 매칭을 통해 복잡한 시스템 프로비넌스 그래프의 분석을 효율적으로 처리할 수 있다.

UNICORN[3]은 프로비넌스를 기반으로 하는 APT 탐지 기술이다. UNICORN은 점진적으로 업데이트되고 고정된 크기의 그래프 데이터 구조를 갖추고 있다. 이를 위해 UNICORN에서는 그래프 스케치 기법(graph sketching technique)을 사용하였다. UNICORN에서는 실시간으로 프로비넌스 그래프를 입력받아 주기적으로 그래프의 특징을 히스토그램(histogram)으로 만든다. 이를 통해 고정된 크기의 그래프 스케치를 만들고 이들에 대해서 클러스터링하는 모델을 만든다.

POIROT[13]은 시스템에서 발생하는 모든 인과관계를 포함하기 위해서 커널 기반 로그를 이용한다. POIROT에서는 CTI(Cyber Threat Intelligence) 인과관계로부터 그려진 쿼리 그래프와 시스템 로그로 그려진 프로비넌스 그래프 사이의 유사성을 평가한다. 이를 위해 POIROT에서는 유사성 측정(similarity metric)을 사용한다. POIROT에서는 정렬에 대해서 노드 정렬(node alignment)과 그래프 정렬(graph alignment)을 사용하였고 이에 대한 측정값을 정리하였다. (그림 11)은 POITROT에서 쿼리 그래프 시나리오와 이를 통한 탐지 정렬 그래프를 보여주고 있다.



a) 리눅스 시스템에서 APT공격에 대한 쿼리 그래프



b) 리눅스 시스템에서 APT공격에 대한 탐지 그래프

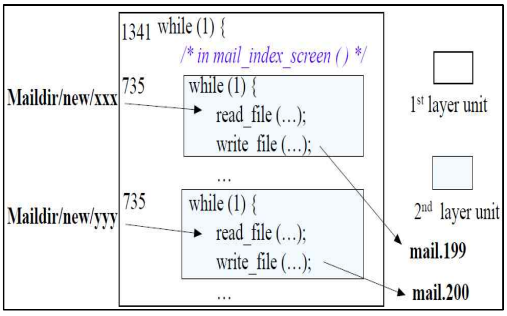
(그림 11) 리눅스 시스템에서 APT공격 대한 쿼리 그래프 및 및 탐지 그래프[13]

5.3 추상화 그래프 기반의 분석

사이버 공격을 탐지하기 위한 시스템 프로비넌스 그래프 분석에서 그래프의 복잡성은 분석을 더 힘들게 한다. 이를 해결하기 위해 HOLMES[14]는 호스트(host)에서 발생하는 로그 이벤트 사이의 인과관계인

프로비넌스 그래프 위에 이를 추상화한 HSG(High- level Scenario Graph)를 생성하였다. HSG는 각 로그 이벤트와 MITRE ATT&CK(R)[31]에서 제공하는 TTP와 매칭을 시켜서 상위 수준의 공격 흐름을 보여주고자 하였다. 또한, HOLMES에서는 시

스템 프로비넌스 그래프의 주체와 객체의 대상이 되는 노드들에 버전(version) 개념을 추가하였다. 즉 새로운 엣지가 기존에 존재하는 노드의 의존성에 변화를 준다면 새로운 엣지를 추가하면서 노드의 새로운 버전을 할당하였다. 버전화의 특징은 기존 분석의 변화 없이 많은 수의 로그 이벤트를 잘라내어 최적화할 수 있다는 것이다.



(그림 12) pine에 있는 유닛의 2 레이어(layers)[5]. 첫 번째 레이어는 메인 루프의 상호작용을 나타내고 2개의 두 번째 레이어는 내부 루프의 연속적인 상호작용을 나타낸다.

또한 HOLMES에서는 시스템 프로비넌스 그래프를 메인 메모리에 올릴 수 있도록 최적화를 하였다. HOLMES에서는 하나의 이벤트를 표시하는데 평균 5bytes 이하를 요구하도록 최적화하였다. HOLMES는 같은 커널 Audit data를 사용하였지만, 그래프 의존성을 메인 메모리에 올려서 사용하는 다른 접근을 시도하였다.

5.4 실행 분기 기반의 분석

실행 분기 기법은 프로그램의 실행을 하나의 통합된 노드로 식별하지 않고 특정 기능 및 주체를 중심으로 세분화하여 식별하기 위한 기법이다. 이를 통해 특정 노드에서 발생하는 이벤트의 의존성 폭발을 막고 분석을 효율적으로 할 수 있다. 지금까지 실행 분기를 적용한 다양한 연구가 진행되어 왔다. BEEP[5]는 더 세분화된 주체(subject)를 위해 유닛(unit)이라고 불리는 분석 기법을 사용한다. 유닛은 특정 주체를 처리하는 하나의 루프(loop)로 구성된 프로세스의 실행 세그멘트이다. 따라서 프로세스는 독립된 입력(input)을 갖는 다수의 유닛으로 분할될 수 있다. 분할을 통해 소스 노드는 한 개 또는 적은 수은 노드와 인과관계로 연결될 수 있고 이는 시스템 프로비넌스 그래프의 인과관계 폭발을 막을 수 있다. BEEP는 프로세스 단위의 실행 노드를 각 유닛 단위로 분할함으로써 소스 노드의 인과관계의 세분화를 줄여서 인과관계 폭발을 줄였다. 프로세스를 유닛 단위로 분할하기 위해 루프 단위로 분석하는 리버스 엔지니어링 기법을 제시하였다. (그림 12)는 리눅스에서 메일 접속 프로그램인 pine을 이용한 실행 유닛을 보여주고 있다.

PROPATROL[29]에서는 대규모 시스템에서 의존성 그래프가 폭발적으로 커지는 것을 막기 위해 활동을 구분하는 기법을 제시하였고 이를 위해 소스나 바이너리 계측(binary instrumentation)을 요구하지 않는다. 대신 인터넷을 사용하는 응용 프로그램에 대해서 실행 단계를 구분하는 것이다. 즉, PROPATROL은 장시간 수행된 프로세스를 더 작은 실행 활동 유닛으로 구분하여 분리한다. 이러한 구분은 일반적으로 입력 및 출력에 기반을 둔다. 예를 들면 구글 크롬 브라우저의 경우 각 실행 활동 유닛은 각 브라우저 창에 입력된 URL을 수행한 탭을 기준으로 구분될 수 있다. 일단 각 실행 활동 유닛으로 분류되고 나면 PROPATROL은 백트래킹을 통해 각 실행 활동 유닛에서 공격을 일으킨 시작점을 찾게 된다. 또한, 공격 시작점에서부터 전방향 추적을 통해 공격에 영향을 받은 지점(파일 또는 프로세스 등)을 찾을 수 있다.

6. 결론

지금까지 많은 연구에서 사이버 공격의 근원을 추적하기 위해 시스템 프로비넌스 그래프를 활용한 모델을 제시하였다. 프로비넌스 그래프는 사이버 공격을 탐지하고 추적하기 위해 다양한 시스템에서 적용되어 왔다. 본 연구에서는 시스템 프로비넌스 그래프를 위한 데이터 수집, 저장 및 분석 등에 관한 다수의 연구를 조사하고 정리하였다. 비록 많은 연구에서 프로비넌스 그래프를 생성하기 위한 기법을 제시하였지만, 여전히 대규모 조직에서 수집한 데이터에 따른 데이터 의존성 폭발 문제는 여전히 해결해야 할 부분으로 남

아 있다. 하나의 대안으로 제시한다면 실시간적으로 데이터를 분석하는 접근법을 제시할 수 있다. 많은 연구에서 수집된 데이터를 저장한 후 분석을 수행한다. 하지만 실시간 연결 분석이 가능하다면 시스템에 저장되어야 할 정보를 큰 폭으로 줄이고 연결 정보만을 저장함으로써 저장되어야 할 데이터를 줄일 수 있을 것으로 보인다. 또한 좀 더 세분화된 이벤트 분류 기법을 제시할 수 있다. 대규모 조직에서는 각 시스템에 유사한 이벤트가 식별된다면 저장되어야 할 불필요한 데이터를 줄일 수 있다. 하지만, 이러한 데이터 감소는 고도화된 사이버 공격을 추적하는데 필요한 정보를 놓칠 수 있는 단점을 가지고 있다. 향후, 데이터 저장을 줄이고 분석을 효율적으로 할 수 있는 새로운 연구가 필요해 보인다.

참고문헌

[1] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B. G. Chun, L.P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, “Taintdroid: an information-flow tracking system for real-time privacy monitoring on smartphones”, ACM Trans. Comput. Syst. (TOCS), 32(2), p.1 - 29, 2014.

[2] S. Sitaraman and S. Venkatesan, “Forensic Analysis of File System Intrusions using Improved Backtracking” Proceedings of the Third IEEE International Workshop on Information Assurance(IWIA ’05), pp. 154-163, 2005.

[3] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, “UNICORN: Runtime Provenance-Based Detector for Advanced Persistent Threats”, arXiv preprint arXiv:2001.01525, 2020.

[4] M. N. Hossain, S. M. Milajerdi, J. Wang, B. Eshete, R. Gjomemo, R. Sekar, S. Stoller, and V.N. Venkatakrishnan., “SLEUTH: real-time attack scenario reconstruction from COTS audit data”, 26th USENIX Security Symposium, USENIX Security, pp. 487 - 504, 2017.

[5] K. H. Lee, X. Zhang, and D. Xu, “High accuracy attack provenance via binary-based execution partition”, Proceedings of the 20th Annual Network and Distributed System Security Symposium, NDSS, pp. 16, 2013.

[6] S. T. King and P. M. Chen, “Backtracking intrusions”, Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, pp. 223 - 236, 2003.

[7] K. Belhajjame, R. B’Far, C. J. Cheney, Sam, S. Coppens, S. Cresswell, Y. Gil, P. Grothl, G. Klyne, T. Lebo, J. McCusker, S. Miles, J. Myers, S. Sahoo, and C. Tilmes, “Prov-DM: The PROV Data Model”, Technical Report. World WideWeb Consortium (W3C). https://www.w3.org/TR/prov-dm/, 2013.

[8] “Fuse”, http://fuse.sourceforce.net.

[9] “Event Tracing”, https://docs.microsoft.com/en-un/windows/win32/etw/event-tracing-portal.

[10] Z. Xu, Z. Wu, Z. Li, K. Jee, J. Rhee, X. Xiao, F. Xu, H. Wang, and G. Jiang, “High fidelity data reduction for big data security dependency analyses”, Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 504 - 516, 2016.

[11] A. P. Chapman, H.V. Jagadish, and P. Ramanan, “Efficient provenance storage”, Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 993 - 1006, 2008.

[12] Z. Li, Q. A. Chen, R. Yang, Y. Chen, and

W. Ruan, “Threat detection and investigation with system-level provenance graphs: A survey”, Computers & Security, Vol. 106, 2021.

[13] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V.N. Venkatakrishnan, “POIROT: Aligning Attack Behavior with Kernel Audit Records for Cyber Threat Hunting”, Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, pp. 1795 - 1812, 2019.

[14] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V.N. Venkatakrishnan, “HOLMES: real-time apt detection through correlation of suspicious information flows”, IEEE Symposium on Security and Privacy (SP), pp. 1137 - 1152, 2019.

[15] “PROV Model Overview”, https://en.wikipedia.org/wiki/PROV_(Provenance).

[16] W. U. Hassan, M. Lemay, N. Aguse, A. Bates, and T. Moyer, “Towards scalable cluster auditing through grammatical inference over provenance graphs”, Proceedings Network and Distributed System Security Symposium, 2018.

[17] A. Woodruff and M. Stonebraker, “Supporting fine-grained data lineage in a database visualization environment”, Proceedings 13th International Conference on Data Engineering, IEEE, pp. 91 - 102, 1997.

[18] “Elasitc, Elastic stack", https://www.elastic.co/, 2022.

[19] “Elastic, Filebeat", https://www.elastic.co/products/ beats/filebeat, 2022.

[20] A. Gehani and D. Tariq, Spade: “Support for provenance auditing in distributed environments”, Proceedings of the 13th International Middleware Conference. Springer, 2012.

[21] T. Pasquier, X. Han, and M. Goldstein, T. Moyer, D. Eyers, M. Seltzer, and J. Bacon, “Practical whole-system provenance capture”, Proceedings of the 2017 Symposium on Cloud Computing, Association for Computing Machinery, pp. 405 - 418, 2017.

[22] K. H. Lee, X. Zhang, and D. Xu, “LogGC: Garbage Collecting Audit Log”, In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, pp. 1005 - 1016, 2013.

[23] S. Ma, X. Zhang, and D. Xu, “ProTracer: towards practical provenance tracing by alternating between logging and tainting”, Internet Society, 2016.

[24] R. Yang, S. Ma, and H. Xu, “UISCOPE: Accurate, Instrumentation-free, and Visible Attack Investigation for GUI Applications”, Network and Distributed Systems Symposium.

[25] J. Y, S. Lee, E. Downing, W. Wang, M. Fazzini, T. Kim, A. Orso, and W. Lee, “RAIN: refinable attack investigation with on-demand inter-process information flow tracking”, Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security-CCS, ACM Press, pp. 377 - 390, 2017.

[26] J. Y, S. Lee, M. Fazzini, J. Allen, E. Downing, T. Kim, A. Orso, and W. Lee, “Enabling refinable cross-host attack investigation with efficient data flow tagging and tracking”, 27th USENIX Security Symposium, USENIX Association, pp. 1705 - 1722, 2018.

[27] Y. Kwon, D. Kim, W. N. Sumner, K. Kim, B. Saltaformaggio, X. Zhang, and D. Xu, “LDX: causality inference by light-weight dual execution”, In: Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, ACM, pp. 503 - 515, 2016.

[28] H. Irshad, G. Ciocarlie, A. Gehani, V. Yegneswaran, K. H. Lee, J. Patel, S. Jha, Y. Kwon, D. Xu, and X. Zhang, “TRACE:Enterprise-Wide Provenance Tracking for Real-Time APT Detection”, IEEE Transactions on Information Forensics and Security, Vol. 16, pp. 4363-4376, 2021.

[29] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. N. Venkatakrishnam, “ProPatrol: Attack Investigation via Extracted High-Level Tasks”, In International Conference on Information Systems Security, Springer, LNCS 11281, pp. 107-126, 2018.

[30] W. U. Hassan, A. Bates, and D. Marino, “Tactical Provenance Analysis for Endpoint Detection and Response Systems”, 2020 IEEE Symposium on Security and Privacy, pp. 1172-1189, 2020.

[31] “MITRE ATT&CK(R)”, https://attack.mitre.org, 2022.

[32] “Common Attack Pattern Enumeration and Classification”, https://capec.mitre.org, 2022.

[33] Y. Liu, M. Zhang, D. Li, K.Jee, Z. Li, Z.Wu, J. Rhee, P. Mitt, “Towards a Timely Causality Analysis for Enterprise Security”, Network and Distributed Systems Security Symposium ‘18, 2018.

[34] M. N. Hossain, J. Wang, R. Sekar, and S. D. Stoller, “Dependence-Preserving Data Compaction for Scalable Forensic Analysis”, USENIX Security Symposium, pp. 1723-1740, 2018.

[35] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. MIles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, J. V. Bussche, “The Open Provenance Model Core Specification”, Future Generation Computer Systems, 2010.

〔저자소개〕

박찬일(Chanil Park)
2001년 KAIST 수학과 석사
2011년 KAIST 전산학과 박사
현재 국방과학연구소 책임연구원
관심분야: IT융합 보안, 사이버 보안, 암호 프로토콜,
email : chanil@add.re.kr