J. Inf. Technol. Appl. Manag. 25(2): 147~159, June 2018
https://doi.org/10.21219/jitam.2018.25.2.147

ISSN 1598-6284 (Print)
ISSN 2508-1209 (Online)

3-tier 시스템 환경에서 비 에이전트 방식의 데이터베이스 사용자 식별 방안

한정상* · 신동천**

A Non-Agent based Identification Scheme for Identifying Database Users in 3-tier System Environments

Jung Sang Han* · Dong Cheon Shin**

Abstract

The changes of internet environment have made services through web application server (WAS) popular. Accordingly, technical difficulties in identifying users who access databases through WAS were incurred. In order to solve these problems, many companies adopt an agent-based approach for identifying users by installing additional software on WAS. However, this approach must submit to some disadvantages in terms of cost, maintenance, and development process. In this paper, we devise an non-agent based approach for identifying database users in 3-tier environments.

Keywords : User Identification, Web Application Server, 3-tier, Non-Agent Approach

Received : 2018. 06. 07. Revised : 2018. 07. 02. Final Acceptance : 2018. 07. 04.

* Dept. of Industrial Convergence Security, Graduate School of Chung-Ang University, e-mail : onetop.hjs@gmail.com
** Corresponding Author, Dept. of Industrial Security, Chung-Ang University, 84 Heukseok-ro, Dongak-gu, Seoul, 06974, Korea; Tel : +82-2-820-5151, e-mail : dcshin@cau.ac.kr

1. 서 론

방대한 자료를 효율적으로 처리하고 관리하기 위하여 데이터베이스 관리 시스템의 활용과 중요성은 점점 더 중요해지고 있다. 이에 따라, 포털 사이트나 기업 내 데이터베이스에 저장된 중요 데이터의 비인가자 혹은 내부자에 의한 유출이나 도난의 위험성이 증가되고 있으며 실제 개인정보의 대량 유출 사건이 끊임없이 발생하고 있어 데이터베이스 보안에 대한 요구사항이 매우 높아지고 있다[송유진 외 1인, 2009]. 한국데이터진흥원에서 발간하는 데이터산업 백서에 따르면 데이터베이스 보안 관리 솔루션 시장이 규모를 조사하기 시작한 2010년 이후 꾸준히 성장세를 보이며 2015년 약 1,021억 원의 시장규모를 기록하고 있다[한국데이터진흥원, 2016].

Multi-tier 구조는 시스템을 모듈단위로 나누어 구성하기 때문에 손쉬운 수정만을 통해 다양한 서비스를 쉽게 지원할 수 있다는 장점을 갖고 있다[Wayne, 1995]. 아울러 분산 시스템의 증가와 인터넷의 확산으로 인하여 클라이언트/서버 환경에서 웹 환경으로 전환되면서 웹 어플리케이션 서버를 통한 데이터베이스 접근이 늘어나고 있어 잠재적인 공격의 위협으로부터 개인정보를 저장하고 있는 데이터베이스를 보호하기 위한 대책이 필요하다[백종일 외 1인, 2009).

데이터베이스 접근 통제에 있어서 웹 어플리케이션 서버는 구조상의 약점을 가지고 있다. 웹 어플리케이션 서버는 사용자 요청에 따라 데이터베이스에 정보조회를 요청하고 그 결과를 사용자에게 보여 주어야하기 때문에 웹 어플리케이션 서버로부터의 요청을 모두 허용 할 수밖에 없다[김희석, 2014]. 또한 어떠한 사용자가 웹 어플리케이션 서버를 통하여 데이터베이스를 접근할 경우 데이터베이스 측에서는 모든 요청이 웹 어플리케이션 서버로부터 요청되어 웹 어플리케이션 서버 뒤에 가려진 사용자를 확인할 수 없기 때문에 접근 통제 정책을 적용할 수 없는 기술적인 문제가 발생할 수 있다[금융결제원, 2007]. 즉 웹 어플리케이션 서버를 통해 접근한 사용자에 대한 신원확인이 어렵기 때문에 적절한 접근 제어 정책을 적용시킬 수 없으며 치밀한 모니터링도 불가능 하게 된다. 이러한 문제를 해결하기 위하여 3-tier 환경에서도 접근 제어 및 모니터링을 위하여 사용자를 식별할 수 있는 기술 개발이 필요하다.

3-tier 환경에서 데이터베이스 접근 사용자에 대한 완벽한 식별은 통제를 위한 필수 선행 과정이다. 통제하기 위해서는 통제 받는 대상이 무엇인지 혹은 누구인지 확실히 알아야 한다. 최근 국내 솔루션 업계에서 웹 어플리케이션 서버가 포함된 3-tier 환경에서 데이터베이스 사용자를 식별하기 위한 기술현황에 따르면 웹 어플리케이션 서버에 추가적인 소프트웨어 즉 에이전트를 설치하는 방법으로 서비스를 제공하고 있다. 그러나 에이전트 방식은 정확한 식별이 가능하다는 장점이 있지만 운영과 관리의 어려움이 존재하며 비용적인 측면에서도 효율적이지 않다. 따라서 에이전트 방식만으로는 다양한 엔터프라이즈 환경에서 요구하는 필요를 완벽하게 채울 수 없다. 따라서 3-tier 환경에서 에이전트를 설치할 필요가 없이 데이터베이스 사용자를 식별할 수 있는 비(非)에이전트 방안에 대하여 연구가 필요하다.

본 논문에서는 에이전트를 설치하지 않고도 사용자를 식별할 수 있는 방안을 제안한다. 이를 위해 웹 어플리케이션 서버에 설치된 에이전트로 부터 정보를 받지 않는 상황에서 사용자를 식별하기 위해서 시스템에서 발생하는 네트워크 패킷의 정보를 활용한다. 웹 어플리케이션 서버에 설치된 에이전트의 정보를 받지 않는 상황에서 사용자를 식별하기 위해서는 시스템에서 발생하는

네트워크 패킷의 정보를 활용할 수밖에 없다. 본 논문에서는 네트워크 패킷 분석을 기반으로 시간 차이 값이라는 식별요소를 도출하여 최대한의 정확도를 가지고 사용자 식별을 수행할 수 있는 방안을 제안한다.

본 논문의 구성은 다음과 같다. 제 2장에서는 3-tier 환경에서 비 에이전트 방식의 사용자 식별에 대해 논한다. 제 3장에서는 3-tier 환경에서 발생되는 TCP 패킷의 매커니즘을 활용하여 네트워크 패킷 탐지순서 기반 식별 과정을 분석하고 사용자 식별을 위한 식별요소를 도출한다. 제 4장에서는 가상의 3-tier 환경을 구축하여 도출된 식별 요소가 제대로 동작 할 수 있는지 효과성을 검토하기 위한 실험을 한다. 제 5장에서는 결론을 맺는다.

2. 관련 연구

2016년도에 발간된 데이터산업백서에 따르면 국내 데이터베이스(DB) 보안 솔루션의 시장 점유율은 조사가 시작된 2012년도부터 2015년까지 국내 기업이 90% 이상의 점유율을 기록하여 대부분의 기업에서는 국내 솔루션을 도입하고 있는 것으로 나타났다[데이터산업백서, 2016].
<Table 1>은 2017년도 하반기를 기준으로 한 국내 DB 접근 제어 솔루션의 시장 점유율이며 전체 시장의 60%를 점유하고 있는 피앤피시큐어에 이어 웨어벨리가 20%의 점유율을 기록하였으며 상위 두 업체가 전체의 80%를 점유하고 있다. 이 수치는 한국기업데이터 및 나이스 신용평가정보와 전자공시시스템을 참고하여 취합한 것이다.

〈Table 1〉 Market Share of DB Access Control Solution

(Unit : %)

Company	Market share (DB access control)
PNPSECURE	60
WAREVALLEY	20
SINSIWAY	8.5
SOMANSA	8.5
BANET	2.86

현재 3-tier 식별 기술 개발이 진행되고 있는 추세 파악을 위해 DB 접근 제어 솔루션을 개발하는 5개의 회사의 솔루션들의 기능을 조사하고 분석한 결과를 <Table 2>는 보여준다[피앤시큐어, 웨어벨리, 신시웨이, 소만사, 바넷정보기술]. 피앤피시큐어의 DBSAFER와 웨어밸리의 Chakra Max가 Agent 방식의 3-tier 식별 기술을 확장 기능으로 제공하고 있었으며 다른 솔루션에서는 아직 3-tier 식별 기술이 적용되지 않고 있다. 상위 두 회사의 경우 웹 어플리케이션 서버에 에이전트를 설치하여 식별하는 에이전트 방식을 적용하고 있다.

〈Table 2〉 3-tier DB User Identification Technology

(Order by Market share)

Company	Solution Name	3-tier User identification
PNPSECURE	DBSAFER	WAS Agent
WAREVALLEY	Chakra Max	WAS Agnet
SINSIWAY	PETRA	none
SOMANSA	DB-i	none
BANET	MIDDLEMAN	none

에이전트 없는 방식의 3-tier 환경에서의 DB 사용자 식별 기술이 개발 된다면 여러 측면에서 그 장점을 활용할 수 있을 것으로 기대된다. 관련 연구와 해외의 자료에 따르면 에이전트 없는 방식은 에이전트를 설치하는 방식에서 얻을 수 없는 장점들이 존재하며 가장 큰 장점은 유지 관리의 편의성과 비용 그리고 운영과 개발의 작업 효율성이었다. 에이전트 없는 방식은 특히 유지 보수 및 관리의 효율성에서 특별한 강점을 보인다. 추가적인 에이전트가 설치될 경우 유지 에이전트에

대한 관리와 필요시 업그레이드가 요구되기 때문에 그만큼 업무량이 증가하게 된다[Stephen, 2013]. 또한 이미 사용되고 있는 시스템에 변경을 가하지 않기 때문에 특히 핵심 기능을 수행하고 있는 서버의 경우에는 에이전트설치로 인한 위험을 줄일 수 있으며, 에이전트가 업데이트 될 때마다 수천대의 서버를 업데이트해야 하는 IT 부서의 업무량을 줄일 수 있는 것이다[Mark, 2017].

비용적인 측면에서도 에이전트를 설치한다는 것은 업데이트가 될 때마다 추가 비용이 발생하게 되며 에이전트를 설치하는 방법에서는 비용문제를 피할 수 없다. 또한 개발과정에서 수많은 버전이 발생하게 되며 이는 더 많은 버그가 발생하게 되고 버그를 고치는 일이 그만큼 많아지게 된다는 의미이다. 반대로 버전 수가 줄어든다는 것은 개발이 편해지고 그만큼 더 높은 수준의 지원을 받게 되며 원하는 새로운 기술을 더욱 빠르게 제공받을 수 있게 된다[UpGuard, 2016].

기존의 식별기술은 에이전트 방식으로 서버에 직접 설치된 소프트웨어를 통하여 더욱 넓은 범위의 정보를 활용하여 강력한 통제의 기능을 제공할 수 있다는 장점이 있다. 그러나 추가적인 에이전트를 설치하는 비용과 운용 및 개발 과정의 비효율성을 감수해야만 한다. 그러나 추가적인 에이전트를 설치하지 않고도 사용자를 식별할 수 있는 기술이 개발 된다면 비에이전트 방식의 장점을 가지고 더욱 다양한 엔터프라이즈 환경에서 사용자 식별 기술을 도입할 수 있게 될 것으로 생각한다.

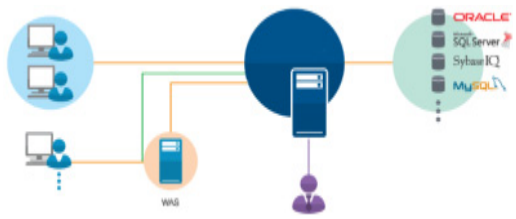
비에이전트 방식에 있어서 사용자를 식별하기 위해서는 네트워크 트레픽 스니핑 방식을 적용할 수밖에 없다. 하지만 네트워크 트래픽을 취득하는데 패킷에 대한 손실은 네트워크 계층에서부터 어플리케이션 계층 수준의 문제로 인하여 발생할 수밖에 없으며 패킷 유실을 완전히 없애기 위해서는 하드웨어에서부터 소프트웨어에 이르기까지 신중하게 구성되어야 한다고 설명하고 있다[Daniel, 2012]. <Table 3>은 두 방식의 장단점을 비교한 것을 보여 주고 있다.

3. 네트워크 탐지에서 사용자 식별 문제

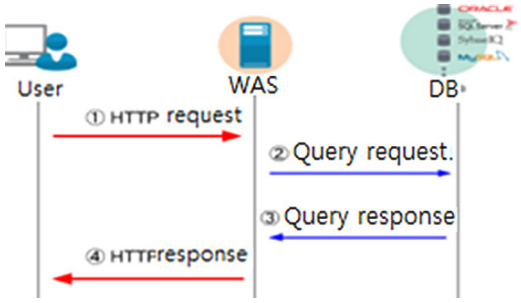
기본적으로 에이전트 없는 방식의 사용자 식별 방안은 구성된 시스템에서 발생하는 패킷을 분석하여 식별한다. 따라서 웹 어플리케이션 서버에 접근하는 사용자의 요청 데이터를 확인하기 위해서 기존의 3-tier의 구성에서 네트워크 구성을 물리적 또는 논리적으로 변경시켜야 한다. 사용자 식별의 기능을 수행하는 식별시스템은 게이트웨이의 위치에서 발생되는 패킷을 모두 확인할 수 있어야한다. <Figure 1>과 같이 사용자가 웹 어플리케이션 서버에 요청하는 HTTP 패킷을 트래픽

〈Table 3〉 Agent vs. Agentless Method

	Agent	Agentless
Advantages	• Lots of information available • All access routes can be controlled	• Low cost • Efficiency of management • No system changes • Accelerating new technology development • Efficiency in operation
Disadvantages	• System Shutdown Risks • Server performance degradation • Difficulty with version control • Difficulty of operation • High cost	• Less information available • Increase network traffic • Decrease detection rate by sniffing method



〈Figure 1〉 Network Configuration for Network Packet Analysis



〈Figure 2〉 Order of Packets Generated

검사 서버가 확인할 수 있도록 구성하는 것이 네트워크 패킷 탐지 순서 기반 식별을 위한 기본적인 조건이다.

기본적으로 네트워크의 연결은 TCP 연결이며 TCP는 신뢰성을 기반으로 한 연결로 받은 패킷에 대하여 수신 여부를 확인하는 패킷을 보내게 되어있다. 네트워크 패킷의 탐지 순서를 기반으로 하는 식별은 페이지를 요청하고 응답받는 TCP 연결의 특징을 이용하는 것이다. 따라서 웹 어플리케이션 서버와 사용자 사이에서 발생하는 HTTP 페이지에 대한 요청과 응답을 이용한 것이다. 사용자가 웹 어플리케이션 서버를 통해 DB를 접근하는 일련의 과정은 <Figure 2>와 같다.

<Figure 1>에서 게이트웨이는 <Figure 2>와 같이 시간 순으로 사용자의-웹 어플리케이션 서버-DB간의 요청과 응답을 확인하는 것이 가능하게 된다. 게이트웨이에 위치한 DB 사용자 식별 프로세스는 위의 순서만 따르게 된다면 간단하게 웹 어플리케이션 서버를 경유한 사용자를 식별해 낼 수 있게 된다. ①, ②, ③, ④의 순서는 절대 변하지 않으며 ①과 ④ 사이의 ②, ③(DB 쿼리)은 ①번을 보낸 사용자가 발생시킨 것이 확실하다. ①번 패킷은 HTTP 요청 패킷으로 사용자의 IP를 포함하여 다양한 식별 정보를 헤더에 포함하고 있기 때문에 결론적으로 DB 쿼리를 발생시킨 사용자의 정보를 확인할 수 있는 것이다.

<Figure 3>은 실제 사용자(IP : 192.168.3.43)가 웹 어플리케이션 서버(192.168.3.40)를 통해 DB에 접근하였을 때 탐지한 시간 순으로 정렬된 패킷이다. ①번 패킷에는 192.168.3.43이라는 사용자의 IP가 기록되어 있으며 ④번 패킷은 웹 어플리케이션 서버에 요청한 페이지가 필요에 따라 DB 조회를 마치고 페이지를 완성하여 사용자에게 응답을 완료하였다는 패킷이다. 따라서 DB 쿼리를 보낸 것은 웹 어플리케이션 서버(IP : 192.168.3.40)지만 DB 접근 시 발생하는 패킷 순서로 볼 때 ② 쿼리는 192.168.3.43의 사용자가 발생시킨 것이므로 DB 사용자 식별이 가능하다.

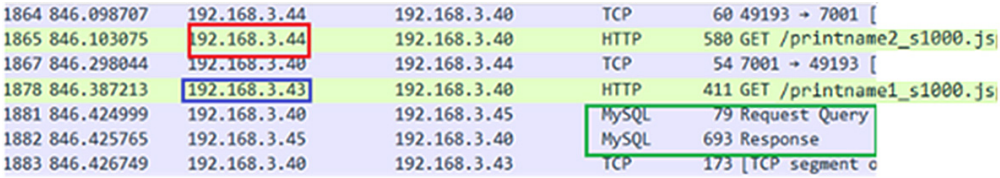
Time	Source	Destination	Protocol	Length	Info
3683.095835	192.168.3.40	192.168.3.43	TCP	66	7001 → 49637 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MS
3683.096066	192.168.3.43	192.168.3.40	TCP	60	49637 → 7001 [ACK] Seq=1 Ack=1 Win=65536 Len=0
3683.097031	192.168.3.43	192.168.3.40	HTTP	579	GET /printname.jsp HTTP/1.1 ① HTTP request
3683.100217	192.168.3.40	192.168.3.45	MySQL	79	Request Query ② Query request
3683.100704	192.168.3.45	192.168.3.40	MySQL	693	Response ③ Query response
3683.101543	192.168.3.40	192.168.3.43	TCP	173	[TCP segment of a reassembled PDU]
3683.101651	192.168.3.40	192.168.3.43	TCP	1514	[TCP segment of a reassembled PDU]
3683.101891	192.168.3.43	192.168.3.40	TCP	60	49637 → 7001 [ACK] Seq=526 Ack=1580 Win=65536 Len=0
3683.101900	192.168.3.40	192.168.3.43	HTTP	2895	HTTP/1.1 200 OK (text/html) ④ HTTP response
3683.102062	192.168.3.43	192.168.3.40	TCP	60	49637 → 7001 [ACK] Seq=526 Ack=4421 Win=65536 Len=0

〈Figure 3〉 Network Packet Detection Sequence-Based

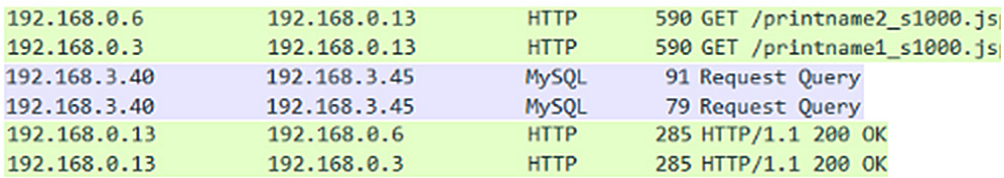
위와 같은 매우 간단하고 단순한 환경에서는 사용자 식별이 명확하지만 특정 상황에서는 사용자를 식별하는데 어려움이 발생한다. 웹 어플리케이션 서버를 통해 접속하는 두 사용자가 같은 시간대에 작업을 하고 있을 경우 동시에 HTTP 요청이 일어나게 될 수 있다. 즉, 한 사용자에 의하여 ①번 패킷이 발생하였고 아직 작업이 완료되지 않아 ④번 패킷이 전달되기 이전에 또 다른 사용자로부터 ①번 패킷이 들어오게 되는 경우이다. 이 경우에 발생되는 ②번 패킷은 어떤 사용자로부터 요청된 것인지 식별할 수 없게 된다.

<Figure 4>는 실제로 발생한 상황을 보여주고 있다. 사용자 1(IP : 192.168.3.44)과 사용자 2(IP : 192.168.3.43)가 비슷한 시각에 웹 어플리케이션 서버에 페이지를 요청한다. 두 사용자는 순서의 뒤바뀜을 눈으로 쉽게 확인하기 위해서 약간 다른 페이지를 요청한다. printname1_s1000.jsp는 79바이트의 쿼리를 생성하고 printname2_s1000.jsp는 91바이트의 쿼리를 생성하도록 설정하였다. 사용자 1(IP : 192.168.3.44)은 웹 어플리케이션 서버에 먼저 페이지를 요청한 후 사용자 2(IP : 192.168.3.43)는 아직 사용자 1에 대한 처리가 끝나기 전에, 거의 동시에 페이지를 요청한 경우이다. 이런 경우 그 이후 웹 어플리케이션 서버가 발생시킨 DB 쿼리(MySQL)들은 어느 사용자에 대한 요청인지 확정할 수 없다.

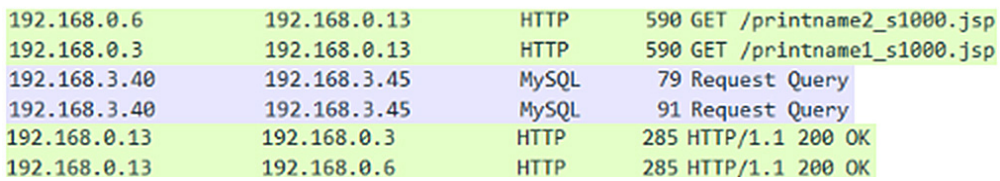
<Figure 5>와 <Figure 6>을 통해 HTTP 요청의 순서로 쿼리 발생 순서를 판단 할 수 없다는 것을 알 수 있다. 즉 HTTP 요청 순서는 동일하지만 응답순서는 서로 다르다는 것을 알 수 있다. 따라서 어떤 HTTP 요청 페이지가 어떤 DB 쿼리를 발생시켰는지 식별할 수 있는 방안이 필요함을 알 수 있다.



〈Figure 4〉 The Overlap of HTTP Requests



〈Figure 5〉 Processed Sequentially



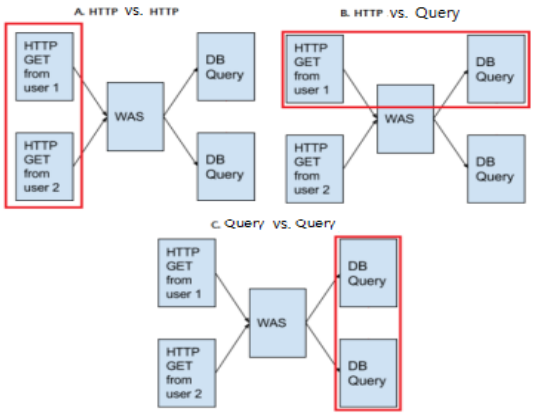
〈Figure 6〉 Processed Reversely

4. 사용자 식별 요소 도출

4.1 사용자 식별 요소

3-tier 환경에서 발생하는 패킷을 비교 및 분석하여 패킷사이에 서로 구별할 수 있는 요소를 도출한다. 사용자 식별에 관여하는 패킷은 사용자에 의하여 발생한 HTTP 요청 패킷과 웹 어플리케이션 서버에 의하여 발생한 DB 쿼리 요청 패킷이다. 따라서 HTTP 요청 패킷과 DB 쿼리 요청 패킷 간의 비교 분석을 한다.

<Figure 7>은 식별요소를 도출하기 위하여 발생된 패킷을 비교한 내용을 보여 주고 있다. Figure과 같이 발생된 패킷에 대하여 세 가지 경우의 비교가 가능하다. 각각의 경우를 비교하고 확인한 내용은 다음과 같다.



〈Figure 7〉 Comparison of Packets for Derivation of Identification Elements

A : 서로 구별이 가능하다. 각각의 필드가 사용자의 네트워크 인터페이스 정보로 채워져 있기 때문에 어떤 사용자가 발생시킨 패킷인지 알 수 있다.

B : 연관 시킬 수 있는 정보를 확인할 수 없다. HTTP 요청 패킷에 있는 정보 어떤 것도 DB 쿼리에 포함되어 있지 않다. 이는 웹 어플리케이션 서버가 DB 서버로 부터 필요한 정보를 얻는 과정에서 사용자에 대한 컨트롤을 수행하지 않는 것을 의미한다.

C : 서로 구별이 불가능하다. 어떤 사용자가 요청하든 웹 어플리케이션 서버는 완전히 같은 패킷을 발생시킨다. 웹 어플리케이션 서버의 작동 원리에 의하여 패킷의 고유한 식별 값이 다를 수 있을 것으로 예상하였지만 완전히 동일한 쿼리 패킷을 생성함을 확인하였다.

이상의 분석결과로부터 웹 어플리케이션 서버는 사용자의 그 어떤 정보도 DB 쿼리 패킷에 포함시키지 않으므로 사용자 식별을 위해 HTTP 요청 패킷의 정보를 이용하는 것 이외에는 다른 방안이 없음을 알 수 있다. 뿐만 아니라, 패킷들의 헤더의 필드에 기록된 값을 이용하여 간단한 비교를 통한 매칭은 불가능하다는 것을 말하고 있다. 이는 에이전트 없는 식별 방법을 개발하는데 있어서 상당히 어려운 상황임을 의미한다. 따라서 IP주소 혹은 포트 번호와 같이 고정되고 고유한 값을 활용할 수 없기 때문에 유일하게 식별 가능한 HTTP 요청 패킷과 그와 관련된 정보로 식별 가능한 값을 만들어 낼 수밖에 없게 된다. 본 논문에서는 패킷 간의 시간 차이를 고유한 값으로 만들어 사용자를 식별하는 방안을 제안한다.

HTTP 요청 패킷으로부터 요청하는 페이지의 이름을 확인할 수 있기 때문에 요청하는 페이지별 평균 시간 차이 값을 미리 가지고 있다면 서버는 이 값을 바탕으로 DB 쿼리가 어떤 사용자에 의하여 요청된 것인지 식별이 가능하게 된다. 이 시간 차이 값의 평균은 네트워크 패킷 탐지순서기반 식별 법에서 식별 불가능한 상황을 해결하기 위한 식별 요소가 된다. 비 에이전트 방식에서는 결국 네트워크 패킷 정보를 가지고 식별을 해야 하는데 한 가지의 요소로만 완벽한 식별을 하는 것은 거의 불가능하다. 이는 패킷 간에 구별 가능한 식별 요소를 찾기 어렵기 때문이다. 본 논문에서는 시간 차이 값이라는 요소를 가지고 실제로 식별 가능성을 분석한다.

192.168.0.6	192.168.0.13	HTTP	590 GET /printname2_s1000.jsp
192.168.0.3	192.168.0.13	HTTP	590 GET /printname1_s1000.jsp

〈Figure 8〉 HTTP Request Packets

식별 요소로 도출한 시간 차이 값은 HTTP 요청이 웹 어플리케이션 서버에 도착한 시간으로부터 DB 쿼리를 발생시킬 때까지의 시간의 차이를 말한다. 시간 차이 값은 요청된 HTTP 페이지의 구성과 관계가 있다. 웹 어플리케이션 서버는 사용자가 요청한 페이지를 소스 코드대로 페이지를 작성한 후 사용자에게 전달하게 된다. 이 과정에서 페이지마다 일정한 시간 차이가 존재하게 된다는 것을 알 수 있었다. 결국 이러한 시간 차이는 DB를 접근하는 페이지 별로 고유한 값이 될 수 있으며 식별요소가 될 수 있다.

<Figure 8>은 사용자가 웹 어플리케이션 서버에 보내는 패킷이다. GET 메소드를 통하여 각각 원하는 페이지의 이름을 보낸다. 사용자 1(IP : 192.168.0.6)의 경우 printname2_s1000.jsp를 요청하고 있고 사용자 2(IP : 192.168.0.3)는 printname1_s1000.jsp를 요청하고 있다. 두 페이지는 서로 다른 소스코드와 페이지 구성을 가지고 있으며 웹 어플리케이션 서버가 요청을 받아 페이지를 작성하는 과정에서 DB 쿼리를 요청하기까지 서로 다른 시간 차이 값을 갖고 있음을 활용하는 것이 바로 시간 차이 값을 이용한 방안이다. 따라서 시간 차이 값을 이용하기 위해서는 미리 특정 페이지에 대한 평균적인 시간 차이 값을 가지고 있어야 한다.

특정 페이지에 대한 평균적인 시간 차이 값을 확보하기 위해 두 페이지의 시간 차이 값을 충분히 여러 번 측정하고 그 평균을 계산한 결과를 <Figure 9>는 보여 주고 있다. printname1_s1000. jsp는 HTTP 요청 페이지가 웹 어플리케이션 서버에 도착한 후 웹 어플리케이션 서버가 DB 쿼리를 생성할 때 가지 걸리는 시간이 평균 0.94786초라는 뜻이다. 마찬가지로 printname2_s1000 .jsp는 평균 1.001963초가 걸린다는 것을 확인할 수 있다.

Unit : Sec

Printname1_s1000.jsp

HTTP R - Query R		Time V
7.389807	-6.48724	0.902567
4.014922	-3.11241	0.90251
2.125007	-1.22173	0.90328
9.796837	-8.89954	0.897294
⋮		
1.108775	-0.19888	0.909897
9.156271	-8.2543	0.901976
7.562729	-6.65431	0.908422
Average		0.904786

Unit : Sec

Printname2_s1000.jsp

HTTP R - Query R		Time V
10.717737	-9.70983	1.007908
8.466509	-7.46218	1.004329
6.2114983	-5.21453	0.996972
4.043312	-3.04513	0.998184
⋮		
5.058721	-4.05347	1.005249
2.980599	-1.98126	0.999339
10.606356	-9.59782	1.00854
Average		1.001963

〈Figure 9〉 Average Time Difference Value of Two Pages

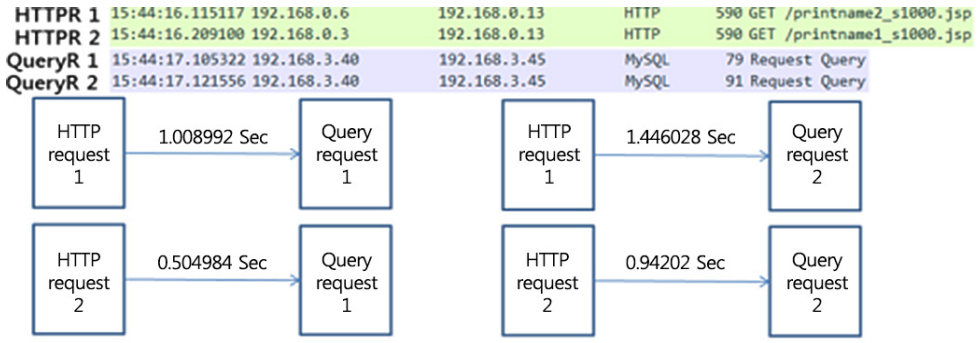
4.2 사용자 식별

식별 요소로 도출된 시간 차이 값이 사용자를 식별할 수 있음을 검증하기 위해 실제 발생된 상황에 적용한다. HTTP 요청순과 동일하게 DB 쿼리가 발생된 경우와 그 순서가 뒤바뀐 경우로 나누어 적용한다. 똑같은 쿼리를 발생시키면 제대로 매칭을 하였는지 확인할 수 없으므로 printname1_s1000.jsp는 79바이트의 패킷을 발생 시키고 printname2_s1000.jsp는 91바이트의 쿼리를 발생시키도록 조정한다.

4.2.1 경우 1-순서가 바뀌지 않은 경우

두 사용자가 발생시킨 HTTP 요청패킷에 대하여 웹 어플리케이션 서버가 발생시킨 DB 쿼리를 가지고 모든 경우의 차이를 구한다. 아직까지는 쿼리 순서가 순차적으로 발생 되었는지 뒤바뀌었는지 알 수 없는 상황에서 시간 차이 값을 활용하여 패킷의 짝을 찾는다. 계산 결과는 다음과 같다(<Figure 10> 참조).

HTTPR 1-QueryR 1 : 1.008992초
HTTPR 1-QueryR 2 : 1.446028초
HTTPR 2-QueryR 1 : 0.504984초
HTTPR 2-QueryR 2 : 0.94202초



〈Figure 10〉 Case 1 - Calculate Time Difference between Packets

Time Value	Average1 0.904786	Average2 1.001963
1.008992	0.10421	0.00703
0.504984	0.399802	0.496979
1.446028	0.54124	0.44406
0.94202	0.03723	0.059943

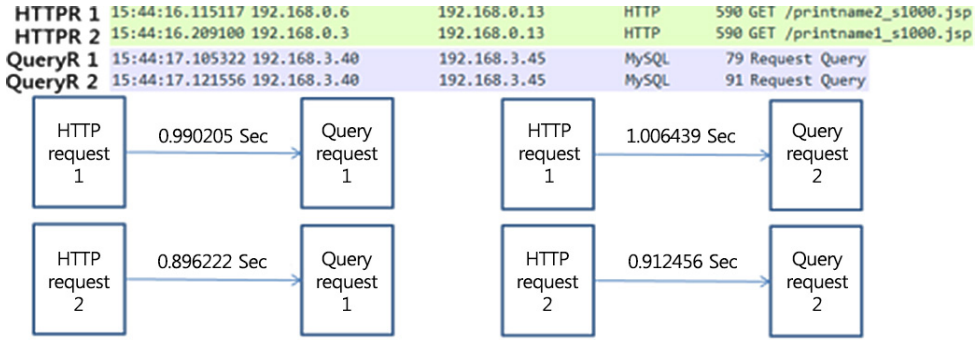
〈Figure 11〉 Case 1-Calculate the Difference between Time Difference Value and Average Value

계산된 결과와 <Figure 9>에서 구한 페이지의 시간 차이 값 평균을 비교하여 가장 근사한 값과 매칭 시킨다. <Figure 11>은 앞서 구한 네 개의 시간 차이 값과 이미 구한 평균값과의 차이를 보여주고 있다. HTTPR 1-QueryR 1의 시간 차이 값이 평균과의 차이에서 0.00703초로 최솟값이며 이는 printname2_s1000.jsp의 평균값과 가장 가까운 것을 알 수 있다. 따라서 HTTPR 1-QueryR 1에 해당하는 것이 printname2_s1000. jsp이며 HTTPR 2-QueryR 2에 해당하는 것이 printname1_s1000. jsp임을 알 수 있다. 결과적으로 printname1_s1000. jsp는 QueryR 2를 발생시켰고 printna me2_s1000. jsp는 QueryR 1을 발생시킨 것이다. 이처럼 순서가 바뀌지 않은 것을 판단할 수 있기 때문에 사용자를 성공적으로 식별할 수 있다.

4.2.2 경우 2-순서가 뒤바뀐 경우

경우 1과 마찬가지로 두 사용자가 발생시킨 HTTP 요청패킷에 대하여 웹 어플리케이션 서버가 발생시킨 DB 쿼리를 가지고 모든 경우의 차이를 구한다. 계산 결과는 다음과 같다(<Figure 12> 참조).

HTTPR 1-QueryR 1 : 0.990205초
HTTPR 1-QueryR 2 : 1.006439초
HTTPR 2-QueryR 1 : 0.896222초
HTTPR 2-QueryR 2 : 0.912546초



〈Figure 12〉 Case 2-Calculate Time Difference between Packets

계산된 결과와 <Figure 9>에서 구한 페이지의 시간 차이 값 평균을 비교하여 가장 근사한 값과 매칭 시킨다. <Figure 13>은 앞서 구한 네 개의 시간 차이 값과 이미 구한 평균값과의 차이를 보여 주고 있다. HTTPR 1-QueryR 2의 시간 차이 값이 0.00448초로 최솟값이며 이는 printname2_s1000.jsp의 평균값과 가장 가까운 것을 알 수 있다. 따라서 HTTPR 1-QueryR 2에 해당하는 것이 printname2_s1000. jsp이며 HTTPR 2-QueryR 1에 해당하는 것이 printname1_s1000. jsp임을 알 수 있다. 결과적으로 printname1_s1000. jsp은 QueryR 1을 발생시켰고 printname2_s1000. jsp는 QueryR 2를 발생시킨 것이다. 이처럼 순서가 뒤 바뀐 것을 판단할 수 있기 때문에 사용자를 성공적으로 식별할 수 있다.

Time Value	Average1 0.904786	Average2 1.001963
0.990205	0.08542	0.011758
0.896222	0.008564	0.105741
1.006439	0.10165	0.00448
0.912456	0.00767	0.089507

〈Figure 13〉 Case 2-Calculate the Difference between Time Difference Value and Average Value

4.3 사용자 식별 방안 효과성 실험

본 논문에서 실험하고자 하는 환경은 사용자, 웹 어플리케이션 서버, DB 서버의 구축을 가상 환경에서 테스트베드를 구성한다. 가상 환경은 Oracle VirtualBox 5.1.22 버전을 사용하였으며 웹 어플리케이션 서버, 사용자용, DB 서버용 시스템을 구성하였다. 각각의 시스템 구성을 정리한 내용은 <Table 4>와 같다.

사용자 식별 프로그램은 C언어로 시간 차이 값을 이용한 식별 방안을 그대로 구현한다. 이 프로그램의 목적은 사용자 요청이 겹치는 상황에서 시간 차이 값을 가지고 식별 불가능을 해결할 수 있는지 측정하기 위한 것이다. 실험은 동시요청을 두 사용자로 제한한다. 두 사용자의 겹침 현상을 일정 수 의도적으로 발생시키고 작성된 프로그램으로 계산하도록 하며 두 페이지의 평균 시간 차이 값의 차이를 변수로 설정하여 이 시간 값 차이에 대한 매칭 성공 확률을 측정하도록 한다. 프로그램의 대한 간략한 설명은 아래와 같다.

〈Table 4〉 Test Environment Information

Tier	OS	Application
WAS	Windows 7 64bit	Oracle Weblogic 12.1.3
User	Windows 7 64bit	chrome
DB Server	CentOS 6.5 64bit	MySql 5.6.35

변수 선언 및 정의 부분에서는 기본적으로 파일 입출력과 시간 데이터를 저장하기 위한 변수, 프로그램에서 사용하는 다양한 카운터 등이 있다. 또한 실험의 편의를 위하여 실험에 사용하려는 페이지의 평균 시간 차이 값을 미리 입력해 놓는다.

시간 값 입력에서는 HTTP요청 패킷과 DB 쿼리 요청 패킷의 시간 값을 각각 시간 순으로 입력 받도록 한다. 또한 HTTP 요청과 DB 쿼리 요청이 순차적으로 발생하였는지 뒤바뀌었는지 여부를 함께 입력하여 프로그램이 시간 차이 값만으로 예측한 결과와 실제 결과가 일치하였는지 보고하도록 한다.

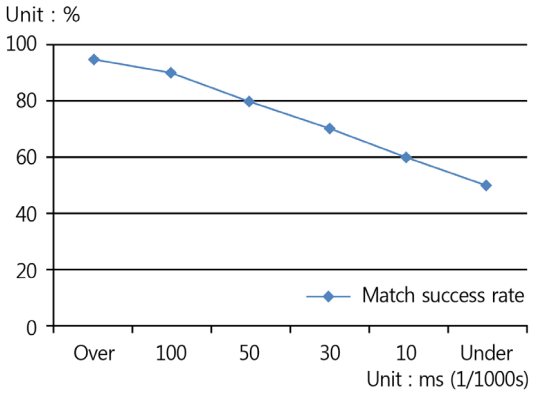
시간 차이 값 연산 수행 과정 그리고 시간 차이 값과 평균 값 비교과정은 입력 받은 모든 시간 값을 절대 값 연산을 통하여 계산한 후 미리 입력되어 있는 평균 시간 차이 값과 비교하는 과정이다.

다음으로 최소 시간 차이 값을 선택하도록 하는 루틴을 삽입하여 프로그램이 최종 결정을 위한 최솟값을 보유한 메모리의 인덱스를 유지하도록 한다.

결정 및 보고 과정에서는 최종적으로 선택된 최소 차이 값의 인덱스를 통하여 HTTP 요청과 DB 쿼리 요청간의 매칭을 최종 결정하고 시간 값과 함께 미리 입력 받은 뒤바뀜 여부 값과 비교하여 프로그램이 정상적으로 식별했는지 기록하여 성공과 실패 횟수를 출력한다. 실패한 경우에는 계산된 값들을 출력하여 실패의 원인을 분석에 활용할 수 있도록 중간 계산 값들을 출력한다.

실험 결과 시간 차이 값을 이용한 사용자 식별의 성공률은 <Figure 14>와 같다. 두 페이지 간의 평균 시간 차이 값이 0.1초 넘게 차이가 날 경우에는 손쉽게 시간 차이 값을 통한 식별이 가능하였다. 시간 차이 값의 차이가 더 줄어들 경우, 즉 두 페이지의 시간 차이 값이 비슷하게 될수록 매칭의 성공률은 50%에 수렴하게 되는 결과를 얻었다.

실험 결과를 통해 0.1초를 초과하는 경우 신뢰할 만한 수준의 식별 성공률을 얻을 수 있었다. 이는 두 페이지의 평균 시간 차이 값이 0.1초를 넘어가는 경우 시간 차이 값이 식별요소로써 매우 유용하다는 것을 보여준다. 기간 차이가 줄어들수록 성공률이 점점 하락하게 되는 이유는 시간 차이 값에서 발생하는 오차와 관련이 있다. 본 논문에서는 식별하기 위한 고유 값으로 평균



〈Figure 14〉 The Identification Success Rate According to Time Difference Value

값을 사용하였다. 고정된 값이 아니기 때문에 오차가 발생하고 두 페이지의 평균값이 비슷해질 경우에는 오차범위가 줄어들게 되고 두 페이지 간의 평균값이 크게 차이나는 경우에는 미리 측정된 평균값과 오차가 다소 크더라도 식별이 가능하게 된다. 반면에 본 실험에서 반영 되지는 않았지만 네트워크의 상태가 혼잡할 경우 평균값과의 오차가 크게 발생할 수 있게 되며 이는 식별률의 하락을 초래하게 된다.

성공률이 50%로 수렴하는 이유는 실험환경과 관련이 있다. 본 논문에서는 2명의 사용자의 겹침이 발생하였을 경우 시간 차이 값을 활용한 식별방안이다. 따라서 50%에 수렴하였다는 것은 두 페이지의 시간차이 값의 차이가 오차범위까지 줄어들어 둘 중에 하나를 선정할 수밖에 없게 된다는 것을 의미한다. 또한 실험에서 사용한 페이지의 평균값을 계산하는 과정에서 오차범위가 주로 0.01초 이내라는 점을 감안하였을 때 0.01초 미만에서 성공률이 50%라는 결과는 매우 당연하며 실험결과의 신뢰성을 보여 준다.

5. 결 론

본 논문에서는 3-tier 환경에서 웹 어플리케이션 서버를 통해 DB에 접근하는 사용자를 명확하게 식별하기 위한 식별요소를 도출하였다. 기존의 방식들은 웹 어플리케이션 서버에 추가적인 에이전트를 설치하여 사용자를 식별하는 방법을 적용하지만 에이전트를 설치하지 않고도 네트워크 패킷 분석만으로 사용자를 식별하는 방법을 제안하였다. 이를 위하여 시간 차이 값이라는 식별요소를 도출하여 사용자 식별에 적용하였으며 두 사용자의 요청의 겹침에 대하여 식별이 가능하였고 식별 요소로써 가치가 있음을 확인할 수 있었다.

시간 차이 값이 식별 요소로써 사용될 수 있는

가능성을 확인하였지만 스니핑 방식의 기술적 한계인 패킷 유실문제 그리고 시간 차이 값이 대량의 트래픽이 발생하는 환경에서 식별율이 저하될 수밖에 없는 명확한 한계점을 가지고 있다. 따라서 향후 연구에서는 두 가지 측면의 보완이 필요하다 첫째, 식별 실패 가능성을 예측하는 방법을 연구하여 실패가 발생할 경우 처리 방안을 강구하는 것이다. 둘째, 두 사용자에 대한 식별로 범위가 제한되어 있지만 더 많은 사용자에 대한 식별에 대해서도 실험을 통하여 가능성을 확인해야 한다. 더 나아가 시간 차이 값이 단일 식별 요소로 사용되기 위해서 현재까지는 보완이 필요하지만 또 다른 식별 요소를 도출하여 함께 사용할 경우 정확도를 향상을 기대할 수 있을 것이다.

References

[1] Baek, J. and Park, D., “A Study on Traceback by WAS Bypass Access Query Information of DataBase”, *Journal of Korea society of Computer and Information*, Vol. 14, No. 12, 2009, pp. 181-190.

[2] Daniel Joseph Barry, Understanding Packet Loss in Network Monitoring and Analysis Appliances, Retrived from https://esj.com/Articles/2012/12/13/Understanding-Packet-Loss.aspx?Page=2.

[3] Eckerson, W. W., “Three tier client/server architectures : achieving scalability, performance, and efficiency in client/server applications”, *Open Information Systems*, Vol. 3, 1995, pp. 46-50.

[4] Kim, H., “WhiteList SQL based database access control on web application server”, Korea University, 2014.

[5] Korea Data Agency, Data Industry White Paper. 2016.

[6] Korea Financial Telecommunications and Clearins Institute, An analysis of DB security tecknology and status, 2007.

[7] Mark Lillywhite, Agent vs Aentless : WHAT TO USE AND WHEN, Retrived from https://www.snowsoftware.com/int/blog/2017/03/23/agents-vs-agentless-what-use-and-when.

[8] Song, U., and Park, K., “Homomorphic Encryption for database outsourcing”, *Journal of the Korea Institute of Inform*, Vol. 19, No. 3, 2009, pp. 80-89.

[9] Stephen, J. B., How to choose the right server performance monitoring tools, Retrieved from, http://searchitoperations.techtarget.com/tip/How-to-choose-the-right-server-performance-monitoring-tools.

[10] UpGuard, Agent vs Agentless and Why We Chose Agentless. Retrived from https://www.upguard.com/blog/agent-vs-agentless-and-why-we-chose-agentless.

저자소개

한 정 상

2015년 홍익대학교 컴퓨터공학과를 졸업하고 2018년 중앙대학교 대학원 산업융합보안학과에서 석사학위를 취득하였다, 현재는 DB 보안업체인 피앤피시큐어에서 서버모듈 개발자로 근무 중이며 Unix 기반의 개발에 관심을 갖고 있다.

신 동 천

1991년 KAIST 전산학과에서 전산학박사 학위를 취득하였다. 1993년부터 중앙대학교에서 근무하였으며 현재는 산업보안학과에서 교수로 재직 중에 있다. 관심분야는 소프트웨어 보안, 데이터베이스 보안, 데이터 기반 보안 분석 등이다.