키 지연 노출에 기반을 둔 로그 전송을 고려한 로그 저장 기법★

강석규* · 박창섭**

요 약

IT 시스템에서 로그는 과거의 중요 이벤트를 보여주는 지표가 된다. 따라서 시스템에 문제가 발생했을 시에 그 원인을 찾고 문제를 해결하는데 사용하기 때문에 저장된 로그의 무결성을 보장하는 것이 중요하다. 기존의 로그 시스템에서는 키 정보가 노출되더라도 저장된 로그의 변조를 탐지하기 위한 다양한 기법들이 제안되었다. 현재 로그의 무결성을 보장하기 위한 연구는 로그의 전송과 저장하는 부분을 분리하여 진행되고 있다. 본 논문에서는 로그의 전송과 저장 시 무결성을 보장하는 로그 시스템을 소개한다. 또한, 제안 로그시스템이 만족하는 보안요구사항과 기존의 연구된 기법들보다 전송 및 저장 시 계산적으로 효율적임을 증명한다.

Log Storage Scheme Considering Log Transmission Based on Time-Delayed Key Disclosure

Seok-Gyu Kang* · Chang-Seop Park**

ABSTRACT

In IT system, logs are an indicator of the previous key events. Therefore, when a security problem occurs in the system, logs are used to find evidence and solution to the problem. So, it is important to ensure the integrity of the stored logs. Existing schemes have been proposed to detect tampering of the stored logs after the key has been exposed. Existing schemes are designed separately in terms of log transmission and storage. We propose a new log system for integrating log transmission with storage. In addition, we prove the security requirements of the proposed scheme and computational efficiency with existing schemes.

Key words : Audit Log, Forward Secrecy, Log Transmission, Log Storage.

접수일(2015년 8월 31일), 수정일(1차: 2015년 9월 15일), 게재확정일(2015년 9월 21일)

★ 본 연구는 미래창조과학부 및 한국인터넷진흥원의 "2015년 정보보호 석사과정 지원사업"의 연구결과로 수행되었음.

★ 본 연구는 2014년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임. (NRF-2014R1A1A2055074)

* 단국대학교/컴퓨터학과

** 단국대학교/컴퓨터학과(교신저자)

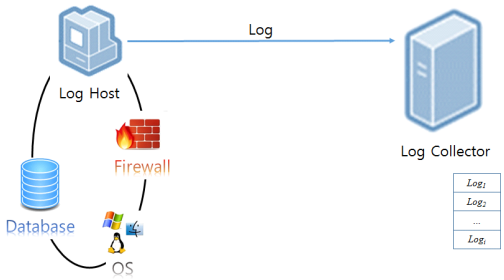
1. 서 론

로그는 IT 시스템에서 발생하는 이벤트를 저장해 놓은 것이다. 로그는 문제 발생 시 그 원인을 찾고, 악의적인 행동을 조사하고, 사용자들의 행동을 분석하는데 사용하기 때문에 로그가 변조되지 않았음을 증명하는 것이 중요하다. 또한, 로그는 한 시스템의 과거와 현재의 상태를 보여주는 지표가 된다. 이러한 로그가 담긴 로그 파일은 사용자의 행동들이 포함된 대부분의 시스템 이벤트들을 저장해놓기 때문에 악의적인 공격자에게 중요한 공격 대상이 된다. 시스템을 탈취한 공격자는 자신의 흔적을 남기지 않기 위해 로그 파일을 공격하거나 로그 서비스를 중단시키려 한다[1]. 따라서 공격자로부터 로그 변조를 막기 위한 장치가 반드시 필요하다. 로그 시스템은 기본적으로 로그를 발생시키는 로그 호스트(Log Host)와 하나 이상의 로그 호스트로부터 로그를 전송받아 수집하는 로그 컬렉터(Log Collector)로 구성된다. 로그 호스트는 발생된 로그를 로그 컬렉터에 전송하고, 로그 컬렉터는 전송받은 로그를 추가적인 검증 값을 같이 저장하면서 전송받은 로그의 무결성을 보장하는 것이 일반적이다. 또한, 추가적인 제 3의 저장소에 로그를 저장해 무결성을 보장하는 경우도 있다. 그 예로는 WORM (Write Only Read Many) 스토리지와 원격 로그 서버를 들 수 있다.

로그의 무결성을 보장하기 위한 다양한 연구 [1-5]가 진행 중에 있다. 하지만 기존의 기법들은 전송받은 로그의 전송과정에서 무결성 검증 값은 버리고 새로운 로그 무결성 검증 값을 생성해 저장해 놓는 기법들이 대부분이다. 본 논문에서는 로그 호스트에서 전송 시에 사용한 로그의 무결성 검증 값을 로그 컬렉터에서 로그의 무결성 검증 값으로 사용하는 기법을 소개한다.

2. 관련연구

2.1 로그 시스템



(그림 1) 로그 시스템

로그 시스템은 로그를 발생시키고 발생된 로그를 로그 컬렉터로 전송하는 로그 호스트와 하나 이상의 로그 호스트로부터 로그를 전송받아 저장하는 로그 컬렉터로 이루어진다. 로그 호스트에서 발생한 로그는 로그 컬렉터로 전송되게 되고, 전송된 로그는 로그 컬렉터에 안전하게 저장된다. 로그 호스트의 예로는 OS, Database, 방화벽이나 이식형 의료기기 등이 있다. 이러한 로그 호스트에서 발생한 로그들은 보안 사고나 비정상적인 행위 탐지 및 포렌식 분석 등에 활용되기 때문에 안전하게 전송되고 저장되어야 한다. 만약 전송 중인 로그나 저장된 로그가 적절하게 보호되지 않으면, 여러 환경과 마찬가지로 로그에 대한 공격자의 삽입, 삭제, 변조 등의 공격에 노출된다. 로그 호스트에서는 전송 중인 로그에 대한 공격에 대비하고, 로그 컬렉터에서는 저장된 로그에 대한 공격에 대비해야 한다.

2.2 위협 모델 및 보안 요구 사항

로그 시스템에서 공격자들의 대표적인 공격 방법으로는 세 가지가 있다. 첫 번째는 로그가 저장된 파일 전체를 삭제하는 공격 방법인 파일 삭제 공격(Total Deletion Attack)이다. 이러한 공격 방법은 공격자가 비밀 정보를 알지 못하는 상황에서 할 수 있는 가장 간단한 방법이다.

두 번째는 저장된 로그들의 일부를 삭제하는 잘라내기 공격(Truncation Attack)[1]이 있다. 이러한 공격 방법 역시 공격자가 비밀 정보를 알지 못하는 상황에서 할 수 있는 방법이다.

세 번째는 검출 지연 공격(Delayed Detection Attack)[1]이다. 이 공격 방법은 공격자가 저장된 로그들

의 어느 한 시점의 키를 알아낸 경우 얻은 키를 사용하여 저장된 로그를 변조 공격하는 방법이다. 이러한 공격은 공격 당시에는 공격 유무를 알 수 없고, 로그 검증 시 검출된다.

현재 저장된 로그의 무결성을 보장하기 위해 WORM 스토리지나 원격 로그 서버를 운영한다. 하지만 WORM 스토리지는 하드웨어 공정 단계에서 악성코드가 심어질 가능성이 있기 때문에 저장된 로그의 훼손 문제나 물리적 손상과 도난의 위험성 또한 완전히 배제할 수 없다. 원격 로그 서버를 운영하는 경우는 서버와 실시간 온라인 상태를 유지할 수 없을 경우에 대한 해결책이 필요하다. 따라서 암호학적 기법을 사용하여 저장된 로그의 무결성을 보장하여야 한다. 다음은 암호학적 기법에 적용되어야 하는 보안요구사항들이다.

로그의 무결성을 보장하는 가장 간단한 방법은 각 로그별 무결성 인증 값을 생성해 저장하는 방식이다. 생성된 인증 값으로 각 로그들의 위변조 확인은 가능하지만 파일 전체가 삭제되거나 일부 로그들이 삭제되는 경우는 탐지가 어렵다. 이를 대비하기 위해 Tsudik 등[1]은 각 로그들의 무결성 인증 값들을 하나의 인증 값으로 대체 가능하도록 하는 “통합 검증 태그” 생성 방식을 제안하였다.

로그는 사후 자료나 사전 지표로 사용될 수 있기 때문에 저장된 로그에 대한 무결성 검증이 필요하다. 로그의 무결성 인증 값 생성 시 동일한 키를 사용한다면 로그들의 변조가 쉽게 이루어질 수 있다. 따라서 로그의 무결성 인증 값은 매번 상이한 키를 사용하여 생성해야 한다. 또한 검증 시에는 인증 값 생성 시 사용된 키를 도출할 방식이 존재해야 한다. 이를 통하여 “전방 안전성”을 만족할 수 있다.

또한 공격자의 침입 시점이나 특정 이벤트가 발생한 시점이 중요한 증거가 될 수 있다. 따라서 로그들의 순서에 대한 위·변조를 막고 이를 탐지할 수 있는 방법이 적용되어야 한다. 이와 같이 로그들의 순서에 대한 무결성을 보장하는 것을 “흐름 안전성”이라 한다.

로그는 다양한 분야에서 사용하기 때문에 인증 값 생성과 검증 시에 같은 키에 의존하게 된다면 키를 알고 있는 내부 담당자 또한 저장된 로그들의 위·변조가 가능하게 된다. 따라서 인증 값 생성과 검증 시에 상이한 키를 사용하여야 한다. 이를 만족하는 로그 시스템은 “공개 검증이 가능한 로그시스템”이라 한다.

2.3 대칭키 기반의 로그 저장 기법

대칭키 기반 암호 알고리즘을 사용하여 저장된 로그를 보호하는 기법들은 동일한 키를 사용하여 무결성 인증 값을 생성하고 검증하게 되므로 키를 안전하게 보장해야 한다. 또한 매번 다르게 사용한 키는 검증 시에도 도출할 방법이 있어야 하므로 일방향 해쉬함수 $H(.)$를 사용하여 도출하는 경우가 대부분이다. 모든 키를 도출할 수 있는 Seed 값을 생성해 안전하게 저장하고 이후에 사용되는 키는 해쉬함수를 사용해 도출할 수 있다. 이렇게 도출된 키와 로그 데이터를 MAC 생성 함수를 이용해 무결성 인증 값을 생성한다. 대칭키 기반의 기법들은 공개키 기반의 함수들 보다 비교적 빠른 해쉬 함수와 $HMAC$ 생성 함수를 사용하기 때문에 로그 컬렉터에서의 오버헤드가 적다. 하지만 무결성 인증 값 생성과 검증에 사용되는 키가 동일하기 때문에 공개 검증에 어려움이 있다.

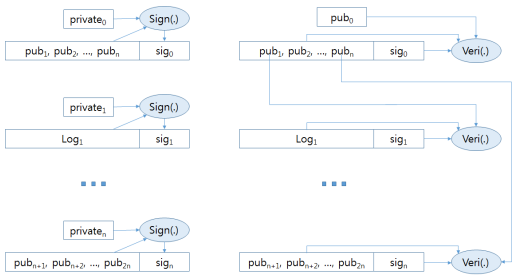
대칭키 기반의 로그 저장 기법 중 Schneier-Kelsey 기법[2]은 로그 컬렉터에서 메시지 인증 코드를 사용해 로그의 무결성을 보장한다. 전송된 로그의 메시지 인증 코드 생성에 사용되는 키는 A_{+1} $H(A_i)$를 통해 도출되고, Seed 값인 A_0는 안전하게 저장한다. 각각의 로그 항목은 로그 데이터를 의미하는 M_i와 이전의 모든 로그를 담은 해쉬 값 Y_i, 그리고 전방 안전성을 만족하는 메시지 인증 코드 값인 Z_i로 구성된다. Y_i는 $Y_i = H(M_i \| Y_{i-1})$로 계산되고, $Y_0 = H(M_i)$로 정의된다. Z_i는 $Z_i = HMAC(Y_i, A_i)$로 계산되어 3개의 항목들이 저장된다. Z_i를 생성할 때 사용된 A_i는 즉시 A_{i+1}로 덮어 쓰도록 하고, Z_f 생성 시 A_f는 삭제하도록 한다.

Tsudic-Ma 기법[1]도 Schneier-Kelsey 기법과 마찬가지로 키를 해쉬 체인을 통하여 도출하게 되고, 메시지 인증 코드를 이용하여 무결성 인증 값을 생성한다. 하지만, 이전 기법과의 두 가지 큰 차이를 보인다. 첫 번째는 통합 검증 태그를 사용한다는 점이다. A_i를 사용하여 통합 검증 태그 $MAC_{,i}$를 만들고, 이 통합 검증 태그를 통해 저장된 로그들의 무결성을 하

나의 인증 값으로 보장한다. 또 다른 차이점은 두 개의 통합 검증 태그를 저장한다는 것이다. 두 번째 통합 검증 태그는 첫 번째 통합 검증 태그와는 다른 Seed 값 를 사용하여 키 를 도출하고, 도출된 키로 새로운 통합 검증 태그 $MAC_{,i}$를 생성한다. 이렇게 생성된 두 번째 통합 검증 태그는 안전한 제 3의 저장소에 저장함으로서 내부 공격자로의 공격에 대비한다.

2.4 공개키 기반의 로그 저장 기법

Holt[3]는 전자 서명을 이용한 로그 저장 기법을 제안하였다. 로그 호스트에서 전송받은 로그와 그 로그를 각각의 서로 다른 개인키로 서명하여 저장하는 기법으로 저장과 검증은 (그림 2)와 같다.



(그림 2) Holt의 로그 저장 및 검증 기법

로그 컬렉터에서는 로그를 수신하기 전 하나의 공개키/개인키 쌍 $ub_0, priv_0$를 생성한다. 그리고 로그들의 서명을 만들기 위한 n개의 개인키/공개키 쌍 $meta = (pub_1, priv_1), ..., (pub_n, priv_n)$을 생성한다. 생성된 공개키들은 $priv_0$로 서명하여 $sig_0 = Sign(priv_0, meta)$를 보관한다. 이 후 수신한 로그들을 각각의 개인키로 서명해 $sig_i = Sign(priv_i, Log_i)$ $(0 < i < n)$를 저장하고, 서명에 사용된 개인키는 즉시 삭제한다. $priv_{n-1}$을 사용할 경우 새로운 $meta$를 만들어 $priv_n$으로 서명한 후 저장하고 $priv_{n+1}$로 다음 로그를 서명한다.

이렇게 저장된 로그들의 무결성을 검증하기 위해서는 먼저 pub_0으로 $meta$의 서명 sig_0를 $Veri(pub_0, sig_0, meta)$로 확인한다. 그 후 각각의 공개키로 해당 로그와 서명이 유효한 지 $Veri(pub_i, sig_i, Log_i)$로 확인한다. 이 검증을 모두 만족시킨다면 저장된 로그의 무결성을 보장한다고 할 수 있다. 이 기법에서 모든 로그에 대한 공개키 쌍이 필요하다. 때문에 공개키 생성을 위한 비용과 모든 로그에 서명을 위한 비용으로 연산적인 부담이 크다. 저장 시에도 모든 로그에 서명을 갖기 때문에 공간적 효율성 측면에서도 부담이 된다. 또한, 로그 전송 시 사용된 로그 무결성 검증 값을 버리고 새로운 저장용 무결성 검증 값을 생성한다.

2.5 Syslog 전송 메커니즘

Syslog-Sign[4]은 송신자 인증, 메시지 무결성, 재사용 공격 저항, 메시지 분실 탐지가 가능한 Syslog 전송 [5] 메커니즘이다. 이러한 서비스는 서명 블록으로 제공되며, 서명 블록에 들어가는 내용은 <표 1>과 같다.

로그 호스트에서 로그 전송 시 무결성을 보장하기 위해 Count, Hash block, Signature의 필드가 사용된다. Syslog-Sign에서는 매 로그 전송 시 서명 작업을 피하기 위해 Hash Block에 여러 로그의 해쉬 값을, Count에는 로그들의 수를 담는다. 그리고 서명블록의 서명 값을 Signature에 담아 로그 컬렉터로 전송한다. 이때 로그 수 제한은 패킷이 2048 octet을 넘지 않는 최대값으로 한다. 로그 컬렉터에서는 전송받은 서명블록의 Count, Hash Block, Signature 필드들의 값을 검증하여 로그의 무결성, 분실 여부 등을 확인할 수 있다.

<표 1> Signature Block Format

Field	Size of octet
Version	4
Reboot Session ID	1-10
Signature Group	1
Signature Priority	1-3
Global Block Counter	1-10
First Message Number	1-10
Count	1-2
Hash Block	variable
Signature	variable

Syslog-Sign에서 TLS와 같은 전송 보안을 사용하

지 않는다면, Syslog와 서명 블록은 모두 평문의 형태로 전송되기 때문에 네트워크 관리자 혹은 공격자가 전송되는 패킷의 내용을 확인해 볼 수 있다. 하지만 전송 중 로그의 무결성은 서명 블록의 서명을 통해 보장되고, 서명에 사용된 개인키가 공개키와 대응되는 것을 확인함으로써 송신자 인증도 가능하게 된다. 이러한 서비스를 제공해주는 서명을 자주 사용하게 됨으로써 로그 호스트에서의 많은 계산량이 요구된다. 이와 같은 문제점을 해결하기 위해 본 논문에서는 서명대신 해쉬 함수를 이용한 메시지 인증 코드와 키 지연 노출 기법[6]을 사용하여 전송 시 로그의 무결성을 보장하고 송신자 인증을 만족시키며 로그 호스트의 계산 부담을 줄인다.

3. 제안 기법

본 논문에서 제안하는 로그 시스템은 기존에 연구된 기법들과는 다른 관점에서 접근한다. 첫째, 로그의 전송과 저장을 하나의 시스템으로 보고 로그 시스템을 설계한다. 기존의 연구들에서는 로그 전송과 저장을 분리하여 연구되었다. 로그 전송에서는 전송 시 보안 요구 사항만 고려한 연구가 진행되었고, 로그 저장에서는 로그 호스트로부터의 전송은 안전하다고 가정하고 연구가 진행되었다. 하지만, 이러한 접근은 구현상 새로운 무결성 인증 값을 생성하면서 추가적인 오버헤드가 발생하게 된다. 제안 기법에서는 추가적인 오버헤드를 줄이기 위해 로그의 전송에 사용된 무결성 인증 값을 저장 시에도 사용할 수 있도록 설계한다.

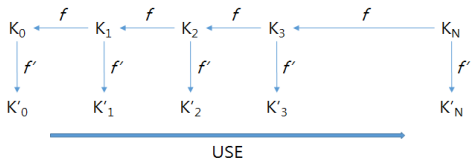
둘째, 대칭키 알고리즘과 공개키 알고리즘을 모두 사용하여 설계한다. 기존의 로그 저장 기법을 소개한 논문에서는 하나의 알고리즘만 사용하여 기법들을 설계하였다. 대칭키 알고리즘과 공개키 알고리즘은 각각의 장단점이 분명하고 두 알고리즘이 만족시키는 보안 요구 사항의 한계가 있다. 따라서 제안 기법에서는 두 알고리즘을 모두 사용하여 더 많은 장점과 보안 요구 사항을 취합하여 로그 시스템을 취합한다. 본 논문에서 제안 하는 자세한 로그 전송 및 저장 과정은 다음과 같다.

<표 2> 표기법

표기법	설명
	로그파일의 마지막 인덱스
	i번째 패킷
Log_i	i번째 로그
H_ID	로그 호스트의 ID
$priv\_X$	X의 개인키
pub_X	X의 공개키
$Sign(.)$	전자 서명 함수
$Veri(.)$	서명 검증 함수
$H(.)$	약한 충돌 해쉬 함수
$HMAC(.)$	메시지 인증 코드 생성 함수
MAC_i	$HMAC(K'_i, Log_i)$
$MAC_{1,i}$	$H(MAC_{i-1}, MAC_i)$
S_{AC}	$Sign(priv\_Host, MAC_{1,f})$
S_{Key}	$Sign(priv\_Host, K_N)$

3.1 로그 전송 과정

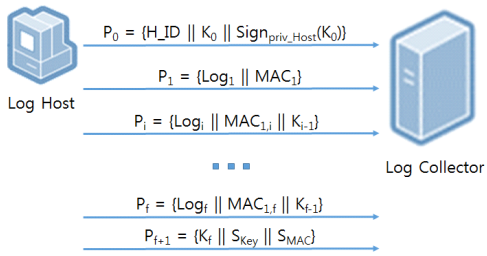
이번 절에서는 제안 기법에서 안전한 로그 전송을 위한 로그 호스트와 로그 컬렉터에서의 작업을 설명한다. 로그 전송에 사용되는 키는 로그 호스트에서 다음과 같은 과정을 통해 생성한다.



(그림 3) 키 도출

- 임의의 을 생성한다.
- K_N $f(K_{+1})$를 통해 N개의 키 재료를 생성한다.
- $K'_i = f'(K_i)$를 키로 사용한다.

위에 사용된 f와 f'은 $f(x) = HMAC(x, 0)$, $f'(x) = HMAC(x, 1)$과 같이 사용한다. 이와 같이 도출된 키를 사용하여 로그 호스트는 로그 컬렉터에게 다음과 같은 과정을 통해 로그를 전송한다.



(그림 4) 로그 전송

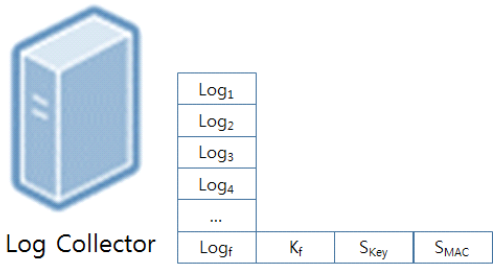
- $H\_ID \| K_0 \| Sign(priv\_Host, K_0)\}$ 를 전송하여 로그 컬렉터에 K_0를 등록한다.
- $P_1 = \{Log_1 \| MAC_1\}$을 전송한다.
- $P_i = \{Log_i \| MAC_{1,i} \| K_{i-1}\}$을 전송한다.
- $P_f = \{Log_f \| MAC_{1,f} \| K_{f-1}\}$을 전송한다.
- $P_{f+1} = \{K_f \| S_{ey} \| S_{MAC}\}$을 전송한다.
- N개의 키를 모두 사용한 경우 키 생성 단계로 돌아가 반복한다.

이와 같이 전송된 로그는 로그 컬렉터에서 무결성 검증을 하고 저장하게 된다. 전송된 로그의 무결성 검증은 다음과 같은 과정을 거친다.

- $Veri(pub\_Host, K_0, Sign(priv\_Host, K_0))$ 를 확인한다.
- P_i를 전송받아 버퍼에 저장한다.
- P_{i+1}을 전송받아 $MAC_{1,i}$와 $H(MAC_{1,i-1}, HMAC(K'_i, Log_i))$가 일치하는지 확인한다.
- P_{f+1}를 전송받아 $MAC_{1,f}$와 $H(MAC_{1,f-1}, HMAC(K'_f, Log_f))$가 일치하는지 확인한다.
- $Veri(pub_Host, K_f, S_{Key})$를 확인한다.
- $Veri(pub_Host, MAC_{1,f}, S_{MAC})$을 확인한다.

3.2 로그 저장 및 검증 과정

로그 컬렉터는 로그 호스트로부터 전송받은 로그를 추가적인 무결성 인증 값을 생성하지 않고 전송받은 패킷을 그대로 저장한다. 또한, 저장된 로그의 무결성을 보장하기 위해 , S_{ey}, S_{MAC}이 같이 저장된다. 저장된 로그의 무결성 검증은 다음과 같은 과정을 거친다.



(그림 5) 로그 저장 구조

- $Veri(pub_Host, K_f, S_{Key})$를 확인한다.
- $K_i = f(K_{i+1})$를 통해 키 재료를 생성한다.
- $K'_i = f'(K_i)$를 통해 키를 도출한다.
- $MAC_{1,f}$를 계산한다.
- $Veri(pub_Host, MAC_{1,f}, S_{MAC})$을 확인한다.

위의 과정 중 만족하지 못하는 단계가 있다면, 그 로그파일 내 로그들의 무결성은 보장할 수 없다.

4. 분 석

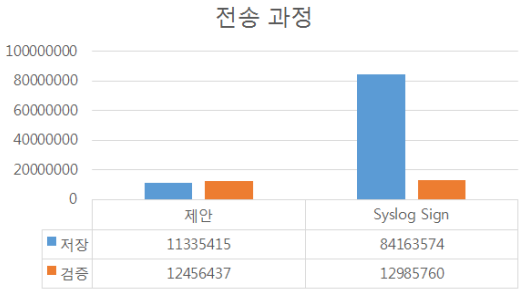
로그는 다양한 분야에서 사용되고, 상당한 양의 로그가 발생하는 경우가 많기 때문에 계산적 효율성이 중요한 요소가 된다. 따라서 이장에서는 제안 기법에 대한 전송, 저장, 검증 시 계산적 효율성을 분석한다. <표 3>은 각 구조 구현에 사용된 환경이다.

<표 3> 구현 환경

구분	환경
CPU	Pentium 2 30GHz
RAM	4GB
OS	Ubuntu
kernel	3.11.0-19-generic
Library	OpenSSL-1.0.1g

4.1 로그 전송 시 계산적 효율성 분석

로그 전송 과정에서는 RFC 로그 전송 표준인 Syslog-Sign과 제안 구조를 비교하였다. 전송에 사용된 로그는 표준에 기준이 되는 2048 octets의 길이에 맞게 설정하였고, 이 로그를 1000개를 전송하는 것으로 가정하였다. 제안 구조 역시 같은 로그를 사용하여 로그 호스트와 로그 컬렉터의 처리시간을 각각 계산하고 이를 비교하였다.

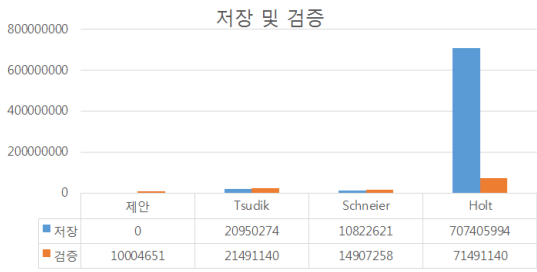


(그림 6) 전송 처리 시간 비교

Syslog-Sign에서는 주기적으로 서명 블록을 생성하고, 생성된 서명 블록마다 한번의 서명 연산이 들어가기 때문에 HMAC을 대부분 사용한 제안 기법에 비해 많은 오버헤드를 갖는 것을 알 수 있다.

4.2 로그 저장 및 검증 시 계산적 효율성 분석

기존의 저장 구조의 경우에는 로그 호스트와 로그 컬렉터 간의 전송 과정이 안전하다고 가정하고 연구가 진행되어 전송 과정에 대한 구체적인 방법이 제시하지 않는다. 하지만 로그의 전송과 저장에서 무결성을 보장하는 것은 공통된 보안 요구 사항이므로 전송 과정을 배제할 경우 추가적인 오버헤드가 발생한다. 앞에서 언급한 바와 같이 제안 기법에서는 전송된 로그와 무결성 검증을 위한 통합 검증 태그를 그대로 저장하게 된다. 기존의 저장 구조는 전송된 로그를 이용하여 추가적인 작업을 통해 저장할 로그의 무결성을 보장하도록 하였다. 저장에 사용되는 로그는 전송에 사용된 로그를 그대로 사용하여 제안 기법과 기존 기법들의 저장 및 검증 처리 시간을 비교한다.



(그림 7) 저장 및 검증 처리 시간 비교

전송되는 로그를 그대로 저장하는 제안 기법에는 저장 시의 소모비용이 들지 않지만 기존 기법들은 새로운 무결성 인증 값을 생성해 저장하기 때문에 오버헤드가 발생한다. 각 기법마다 오버헤드의 차이는 구조의 차이도 있지만 사용되는 암호화 기법의 차이가 오버헤드에 가장 큰 영향을 미친다. 공개키 알고리즘을 사용한 기법의 경우 대칭키 알고리즘을 사용한 기법보다 상당한 오버헤드를 갖는다. 서명은 HMAC 함수보다 보통 0배 느린 속도를 보이기 때문에 구조별 차이보다 더 큰 영향을 미친다. 제안 기법에서는 대부분 HMAC 함수를 사용하고 서명을 최소한으로 사용해 검증 시 오버헤드를 줄인다.

4.3 보안 요구 사항 분석

<표 4> 구조별 보안 요구 사항

	Schneier	Tsudik	Holt	제안
전방 안전성	O	O	O	O
공개 검증	X	X	O	O
통합 검증 태그	X	O	X	O
개별 검증	O	X	O	X
흐름 안전성	O	O	O	O
제3의 저장소	X	O	X	X

기존에 연구된 저장 기법들은 대칭키 기반 알고리즘으로 MAC을 생성하는 방식과 공개키 기반 알고리즘으로 서명을 생성하는 방식으로 나누어진다. 개인키 기반 알고리즘은 저장된 로그의 검증을 위해서는 키 도출을 위한 Seed 값을 로그 검증자에게 전송해야 한다. Seed 값을 전송받은 로그 검증자는 저장된 모든 로그의 위변조가 가능하기 때문에 로그 검증자를 완전히 신뢰할 수 없다면 Schneier와 Tsudik의 기법은

적용하기 어렵다. 반면에, Holt와 제안 기법의 경우, 공개키 기반 알고리즘으로 생성되는 서명은 생성과 검증에 사용되는 키가 서로 상이하기 때문에 로그 검증자에게 검증용 공개키를 전송해 주더라도 저장된 로그는 위변조로부터 안전하다.

로그 파일에서 발생 가능한 공격들 중 로그 일부를 삭제하는 잘라내기 공격을 탐지하기 위해서는 로그 파일 전체의 무결성을 보장하는 통합 검증 태그가 필요하다. 통합 검증 태그를 사용하는 Tsudik과 제안 기법은 잘라내기 공격을 탐지 가능하다. 하지만 통합 검증 태그만 사용하기 때문에 로그 각각에 대한 무결성 인증 값은 저장하지 않기 때문에 개별 검증은 어렵다. 반대로 Schneier와 Holt의 기법에서는 통합 검증 태그를 사용하지 않고 로그에 대한 무결성 인증 값만 사용하기 때문에 로그 파일 전체에 대한 무결성을 보장하기는 힘들지만 로그의 개별 검증은 가능하다.

로그 시스템은 키가 노출되거나 내부 공격자가 키를 획득한 후 로그를 위변조할 가능성이 존재하기 때문에 키가 노출되어도 이전에 저장된 로그의 무결성은 보장되어야 한다. 대칭키 기반의 알고리즘을 사용할 경우 해쉬 체인을 사용하여 키를 매번 바꿔주고 이전의 키를 삭제함으로써 저장된 로그의 무결성을 보장하여 전방 안전성을 만족한다. 또한, 로그 파일 내의 로그를 재배치하더라도 탐지가 가능하다. Holt의 기법에서는 모든 로그의 서명에 사용되는 키를 상이하게 사용하고 서명에 사용된 개인키는 삭제되기 때문에 저장된 로그의 위변조는 탐지가 가능하다. 로그의 재배치는 Meta 값을 확인한 후 각 로그에 맞는 공개키로 확인하기 때문에 탐지가 가능하다. 제안 기법에서는 키를 지연 노출시키기 때문에 이전에 저장된 로그의 무결성을 보장하고 로그 파일의 경우 통합 검증 태그를 서명해 저장하기 때문에 재배치의 경우도 탐지가 가능하다.

5. 결 론

본 논문에서는 이전에 연구된 기법들과는 다르게 로그의 전송과 저장을 모두 고려한 로그 시스템을 제안하였다. 이러한 로그 시스템은 로그 호슬와 로그 컬렉터의 계산 부담을 줄여준다. 또한, 로그의 무결성 인증 값 생성에 대칭키와 공개키 알고리즘을 모두 사용하여 계산 부담은 줄이고 더 많은 보안 요구 사항을 만족하도록 설계하였다.

로그는 다양한 분야에서 시스템 분석과 증거물로 사용되기 때문에 로그가 위변조되지 않았다는 무결성 보장이 필요하다. 그리고 각 분야에서 필요한 보안 요구 사항이 다르고 시스템에서 사용 가능한 연산량이 제한된 경우가 많기 때문에 다양한 시스템에 적용 가능한 로그 저장 기법이 필요하다. 또한, 실제 시스템에서의 위협 모델을 파악 및 분석하고 그에 맞는 보안 요구 사항과 로그 저장 기법이 필요하므로, 이에 대한 연구가 필요할 것으로 예상된다.

참고문헌

[1] D. Ma and G. Tsudik, "A New Approach to Secure Logging, ACM Transactions on Storage", vol 5, no. 1, Article 2, 2009.

[2] B. Schneier and J. Kelsey, "Cryptographic Support for Secure Logs on Untrusted Machines", The Seventh Usenix Security Symposium Proceedings, pp 53-62, 1998.

[3] J. E. Holt, "Logcrypt:Forward Security and Public Verification for Secure Audit Logs", Conferences in research and practice in information technology , Vol 54, 2006.

[4] J. Kelsey, J. Callas, and A. Clemm, "Signed Syslog Messages", RFC 5848, May 2010.

[5] R. Gerhards, "The Syslog Protocol", RFC 5424, March 2009.

[6] A. Perrig, R. Canetti, J. D. Tygar and D. Song, "The TESLA Broadcast Authentication Protocol", In CryptoBytes, Vol.5, No.2, Summer/Fall 2002, pp. 2-13.

[7] D. Ma and G. Tsudik, "Forward-Secure Sequential Aggregate Authentication", In Proc. of the IEEE Symposium on Security and

Provacy, pp. 86-91, Berkeley, May 2007.

[8] B. Schneier and J. Kelsey, "Secure Audit Logs to Support Computer Forensics", ACM Transactions on Information and System Security, vol.2, no.2, pp. 159-176, 1999.

[9] A. A. Yavuz and P. Ning, "BAF: An Efficient Publicly Verifiable Secure Audit Logging Scheme for Distributed System", In Proc. of the Annual Computer Security Applications Conference, pp. 219-218, Honolulu, Dec. 2009.

[10] A. A. Yavuz, P. Ning, and M. Reiter, "BAF and FI-BAF: Efficient and Publicly Verifiable Cryptographic Schemes for Secure Logging in Resource-Constrained Systems", ACM Transactions on Information and System Security, vol. 15, Issue 2, pp. 9:1-9:28, July 2012.

[저 자 소 개]

강 석 규 (Seok-Gyu Kang)

2014년 2월 단국대학교
컴퓨터학과 학사
2015년 3월~현재 단국대학교
컴퓨터학과 석사과정

email : ksg890528@naver.com

박 창 섭 (Chang-Seop Park)

1983년 2월 연세대학교
경제학과 학사
1987년 2월 Lehigh University
컴퓨터과학과 석사
1990년 2월 Lehigh University
컴퓨터과학과 박사
1990년 3월~현재 단국대학교
컴퓨터학과 교수

email : csp0@dankook.ac.kr