POSTER: How to Securely Record Logs based on ARM TrustZone

Seungho Lee
Graduate School of Information Security, Korea University
Seoul, Republic of Korea
happy365.lee@gmail.com

Wonsuk Choi
Graduate School of Information Security, Korea University
Seoul, Republic of Korea
wonsuk85.choi@gmail.com

Hyo Jin Jo
School of Software, Hallym University
Gangwon-do, Republic of Korea
hyojinjo86@gmail.com

Dong Hoon Lee*
Graduate School of Information Security, Korea University
Seoul, Republic of Korea
donghlee@korea.ac.kr

ABSTRACT

A number of logs are generated from IT devices. Since logs have important information regarding a system, they are used for finding the trace of an intrusion or obtaining important information through a big data analysis. Hence, the logs have become a major attack surface for attackers. To protect logs, IT devices require secure logging methods as a mandatory service. Secure logging can provide detection of malicious manipulation of logs and verification of their origin. In this paper, we propose a secure logging method satisfying forward and backward secrecy based on ARM TrustZone for embedded systems, which enables to efficiently generate secure logs through inter-process communication without modification of the existing system (Syslog). Also, we show that the proposed method does not require extra overhead compared with the existing logging method.

CCS CONCEPTS

• **Security and privacy** → *Trusted computing*.

KEYWORDS

Secure log; Forward secrecy; ARM TrustZone

ACM Reference Format:
Seungho Lee, Wonsuk Choi, Hyo Jin Jo, and Dong Hoon Lee. 2019. POSTER: How to Securely Record Logs based on ARM TrustZone. In *ACM Asia Conference on Computer and Communications Security (AsiaCCS '19), July 9–12, 2019, Auckland, New Zealand*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3321705.3331001

1 INTRODUCTION

If abnormal behaviors occur in a system, an administrator first checks the logs generated from the system. In particular, chronologically recorded logs become crucial evidence in a digital forensic process for criminal investigation. Nowadays, service providers collect logs generated by various devices connected in the IoT environment and reproduce them as valuable information through big data analysis, and logging as a service (LaaS) technology is also rapidly developing. Thus, logs are one of the most important resources for backward and toward event analysis in the system.

Even though this is true, attackers are still able to maliciously manipulate the logs. For example, an attacker who breaks into a system can try to delete a log to clear his footprint or modify logs to achieve their attack goals. Therefore, logs must detect malicious operations, and unauthorized users should not be able to view the logs. In addition, recorded logs must provide assurance that they have been recorded reliably by the right device. The logs that contain these features are called audit trail, audit log, or secure log. In this paper, we representatively call it a secure log.

The best way to keep logs secure is to store them in an external secure storage with read-only permission. However, it is not an efficient way because it requires additional costs and considerations in embedded systems having limited hardware resources. Embedded systems generally adopt ARM Architecture, which is a low-power machine based on a reduced instruction set computer (RISC). Furthermore, ARM provides a trusted execution environment (TEE) called TrustZone, a hardware-based isolated execution environment, to provide security services. We propose an efficient method for providing a secure logging method based on ARM TrustZone without modifying the existing logging service Syslog. In particular, we add a session key which is regenerated at a time interval of a periodic time order to set an accessible boundary to the key. Thus, a user can only decrypt logs within a predefined time boundary related to a key.

In summary, our contributions are:

- Designing and implementing a secure logging service based on ARM TrustZone without modifying the existing Syslog.
- Proposing a new method that a key can only decrypt logs within the configured time boundary.
- Evaluating the proposed method on a real target board based on ARM TrustZone.

2 RELATED WORKS

Bellare and Yee define the forward integrity security property for the first time in [1]. Schneier and Kelsey propose a method providing forward secrecy using a one-way hash key chain for detecting log manipulation and preventing unauthorized access in [5]. Holt proposes various applications using symmetric keys with public

*Corresponding author: Dong Hoon Lee (e-mail: donghlee@korea.ac.kr).

AsiaCCS '19, July 9–12, 2019, Auckland, New Zealand

ACM ISBN 978-1-4503-6752-3/19/07.
https://doi.org/10.1145/3321705.3331001

key infrastructure in [2]. Karande et al. extend [6] based on Intel-SGX which is a secure guard extension (SGX) provided by Intel, instead of TPM [3]. Based on previous works, Lee et al. propose a vehicular recording system called T-Box based on ARM TrustZone [4]. Even though all previous works satisfy forward secrecy which means the past keys should not be compromised by future keys, when a current key is exposed, they could not prevent generating the future keys. Thus, we provide backward secrecy by applying an additional session key chain operated with predefined time.

3 SYSTEM DESIGN

3.1 Notation

Table 1: Notation in this paper

Notation	Description
$L(i,j)$	$(i * blocksize + j)$'th generated log by an App.
$LE(i,j)$	$(i * blocksize + j)$'th log-entry generated by T-logger
$sKey_v$	v's private key
$pKey_v$	v's public key
$InitSecretKey$	Initial secret key to generate $InitKey$
$InitKey$	Initial key to generate $Skey$
$SKey(s)$	s'th session key which is regenerated at a predefined time interval
$BKey(i)$	Initial key of i'th block which is derived from previous $BKey(i-1)$
$LKey(i,j)$	Encryption key for $(i * blocksize + j)$'th log
$MKey(i,j)$	MAC key for $(i * blocksize + j)$'th log
$H(data)$	Hash operation with data
$C(i,j)$	$(i * blocksize + j)$'th encrypted log
$MAC(D,K)$	Generating message authentication code of D by using K
$ENC(D,K)$	Encrypting D with K
$SIGN(D,K)$	Signing D with K

3.2 Overview

Embedded systems including ARM TrustZone are divided into a rich execution environment (REE) in which Linux runs, and a trusted execution environment (TEE) in which a secure operating system runs. An application running in REE is called a client application (CA), and an application running in TEE is called a trusted application (TA).

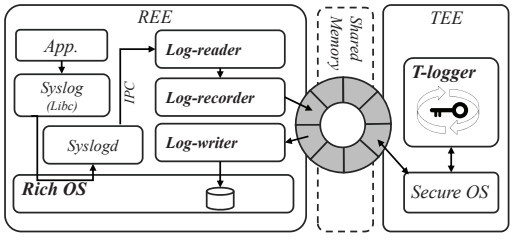

Figure 1: System architecture of the secure logging service

In our solution, there are three CAs called Log-reader, Log-recorder, and Log-writer for supporting TEE, and there is one TA called T-logger (shown in Figure 1). When an application generates logs through a system call in Libc, the Log-reader receives the logs from the Syslog daemon via an inter-process communication (IPC) channel and forwards them to the Log-recorder. Log-recorder, Log-writer, and T-logger share logs through a shared memory consisting of circular ring buffers for efficient memory management. When the Log-reader writes the logs to the buffer,the Log-recorder transfers one log among all of them to the T-logger running in TEE. The T-logger encrypts the log then generates a MAC regarding the log or attaches a digital signature for a block including all log-entries. The logs processed by the T-logger are written to an internal storage or transmitted to a remote server by the Log-writer.

3.3 Structure of log-entry and an initial Block

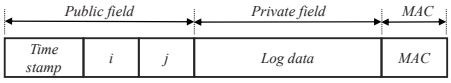

Figure 2: Structure of a secure log-entry

Figure 2 is shown the structure of a log-entry. A log-entry consists of a public field, a private field, and a message authentication code. The public field has information which can be accessed by everyone, the private field is an encrypted field which is not disclosed, and the last field is MAC for ensuring integrity for a log-entry. The public field contains a timestamp and a sequential ordering number obtained from a monotonic counter in TEE, and the private field is log data generated from an application in REE.

$$InitKey \leftarrow H(InitSecretKey)$$
$$InitBlock = ENC(InitKey, pKey_{app})|Metadata$$
$$Signature = SIGN(H(InitBlock), sKey_{dev})$$
$$Blk(0) = InitBlock|Signature$$

An initial block consists of $InitKey$, metadata, and a signature. The $InitKey$ is for generating keys in a hash chain, metadata is for checking log types: such as a timestamp and unique information about the device or the purpose of logs, the last one is a signature of the initial block signed by a device's private key.

3.4 Hash chain and Key management

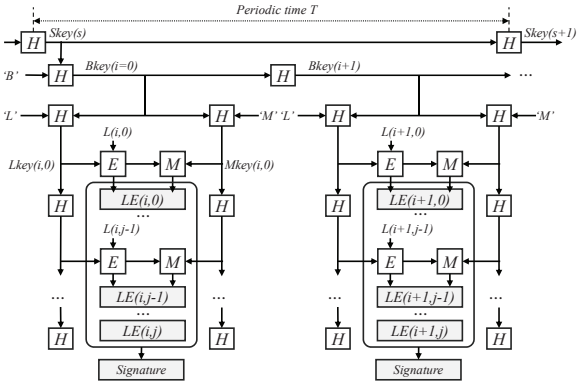

Figure 3: Key managing for generating secure log

We construct a block with a number of log-entries for reducing computational overhead and generate a digital signature on each block for providing a non-repudiation security property. When

a user wants to verify logs, he can use a device's public key or can request a specific party to restore the $InitSecretKey$ through Shamir's secret sharing scheme (SSS). The $InitSecretKey$ which derives $InitKey$ is shared with participants. The purpose of applying SSS is to make evidence for digital forensics.

In this paper, we periodically regenerate a $SKey$ from the latest key through a hash chain at a time interval in configured time. It makes a user can decrypt only the logs of the corresponding interval time by using a $BKey$ which is derived from a $SKey$. The origin scheme applying a time interval is from in [1]. In previous research, $BKeys$ were only able to protect previous logs, which were recorded before the time when a user received a specific $BKey$, but our method has an advantage which is able to protect logs generated after the periodic time T from the corresponding time. In Figure 3, $SKeys$ are updated through a hash chain during every periodic time T, and logs delivered by applications are sequentially encrypted and stored. If a log-entry is full in a current block, the block is signed with a private key of the device. The digital signature of the previous block may be used in various ways with a block-chain network. The following Algorithm 1 presents the part of operations in Figure 3 as pseudo codes.

3.5 Verification

When a verifier requests log verification from a subject having a $sKey_{app}$, who can decrypt the $InitKey$ in an initial block, the subject first checks a timestamp in the $publicField$ of the requested log-entry. What if the subject is a trusted server, when the request is valid, the trusted server derives the corresponding $BKey(0)$ from the decrypted $InitKey$ with time T through a hash chain. After $BKey(0)$ is securely transferred to the verifier, the verifier can decrypt and verify logs. In this part, the single $BKey$ can restore only the logs recorded during the time T. The $BKey(0)$ is only able to derive further $BKey(0+\delta)$ positioned in the same period related to $SKey(s)$. The δ is one of the index of blocks recorded during time T.

4 EVALUATION

Table 2: Results of the performance measurement (unit: us)

Type	Syslog	T-logger1	T-logger2
Syslog	23.7	32.4	32.0
Log-reader/recorder/writer	-	42.0/25.0/12.1	43.8/29.9/12.4
T-logger	-	347.1	3,106.8
Total	**23.7**	**458.6**	**3,224.9**

Table 2 shows the results of measuring the average overhead at each step while generating 128 logs at the Hisilicon Kirin 620 SoC. The total is the time it takes to record one log. The type of Syslog is for calling a system call using the existing Syslog. T-logger1 measures the time spent for only MAC operations without signing operations, and T-logger2 has a block size of 16 log-entries, which means it includes 8 signing operations. Since applications call a syslog system call asynchronously through a circular buffer, there is no significantly distinguishable overhead. An additional time cost is only from T-logger and the time of world-switching, but it is addressed by background processes. If the buffer size is enough for stacking unprocessed logs, T-logger is able to handle all logs delivered from applications.

Algorithm 1 Generating secure logs

Input: $L(i,j)$ /*A log delivered from applications*/
Output: $LE(i,j), Signature$ /*Encrypted log with MAC or Sign.*/

if *Timer is expired on configured time T* **then**
  $SKey(s) = H(SKey(s-1))$
  $i \leftarrow reset$
end if
if j is the first log-entry in a block **then**
  if *i is the first block* **then**
    $BKey(0) = H(SKey(s)|'B')$
  else
    $BKey(i) = H(BKey(i-1))$
  end if
  $LKey(i,0) = H(BKey(i)|'L')$
  $MKey(i,0) = H(BKey(i)|'M')$
end if
if j *equals the block size* **then**
  $Signature = SIGN(H(Blk_{(i)}), sKey_{dev})$
  $i = i+1$
  $j \leftarrow$ *set the first log-entry*
else
  $LKey(i,j) = H(LKey(i,j-1)$
  $MKey(i,j) = H(MKey(i,j-1)$
  $C(i,j) = ENC(L(i,j), LKey(i,j)$
  $publicField = timestamp|i|j$
  $code = MAC(H(publicField|C(i,j)), MKey(i,j))$
  $LE(i,j) = publicField|C(i,j)|code$
  $j = j+1$
end if

5 DISCUSSION AND CONCLUSION

Since the proposed method was designed to regenerate a session key every predefined time interval, both decryption and verification are only available within a given time boundary. Moreover, the proposed method does not require extra overhead and modification of Syslog. As one limitation, we did not consider malicious applications producing malicious logs. We expect this limitation can be resolved with a strong access control scheme such as security-enhanced Linux (SELinux), but we are researching a practical access control method for logs by using attribute-based encryption, which is even secure against a rooted system. We hope to discuss these topics.

REFERENCES

[1] Mihir Bellare and Bennet Yee. 1997. *Forward integrity for secure audit logs.* Technical Report. Technical report, Computer Science and Engineering Department, University of California at San Diego.
[2] Jason E Holt. 2006. Logcrypt: forward security and public verification for secure audit logs. In *ACM International Conference Proceeding Series*, Vol. 167. 203–211.
[3] Vishal Karande, Erick Bauman, Zhiqiang Lin, and Latifur Khan. 2017. Sgx-log: Securing system logs with sgx. In *ACM ASIACCS 2017*. ACM, 19–30.
[4] Seungho Lee, Wonsuk Choi, Hyo Jin Jo, and Dong Hoon Lee. 2019. T-Box: A forensics-enabled trusted automotive data recording method. *IEEE Access* (2019).
[5] Bruce Schneier and John Kelsey. 1999. Secure audit logs to support computer forensics. *TISSEC* (1999), 159–176.
[6] Arunesh Sinha, Limin Jia, Paul England, and Jacob R Lorch. 2014. Continuous tamper-proof logging using TPM 2.0. In *International Conference on Trust and Trustworthy Computing*. Springer, 19–36.