Exemplifying Attack Identification and Analysis in a Novel Forensically Viable Syslog Model

Steena Dominica Steven Monteiro
Dept. of Computer Science, UMC 4205
Utah State University
Logan, UT 84322
steena.m@aggiemail.usu.edu

Robert F. Erbacher
Dept. of Computer Science, UMC 4205
Utah State University
Logan, UT 84322
Robert.Erbacher@usu.edu

Abstract

This research builds on our method for validating syslog entries proposed in [5]. The goal of the proposed method is to allow syslog files to be forensically viable. The goal with this phase of the work is to implement the proposed method and evaluate the forensic validity of the method under real-world conditions. This paper discusses that implementation and the ability for the generated authentication logs and access fingerprints to both identify malicious activity and identify the source of this activity. While work has been done to develop secure log files, i.e., making them tamper resistant, there has been no prior work to ensure they are forensically valid.

1. Introduction

This research aims to provide a mechanism that will validate and authenticate syslogs for computer forensic analysis. Syslogs are often smoking guns [9] in an organization where a computer or network attack has occurred due to the immense amount of information they contain. Syslogs may also contain evidence of illegal or inappropriate activity by the user of an individual system. Traditionally, when computer evidence needs to be collected, the entire system is taken off-line and the hard drive treated as evidence. With a network attack, there could be evidence in syslog files throughout the entire organization. This makes it unfeasible to take the systems with potential evidence off-line, especially when considering the frequency at which network based attacks actually occur. Since syslog entries are traditionally duplicated on a central repository, the syslog facility provides a means by which the evidence can be collected without taking systems off-line, assuming of course the syslog files can be made to be legally admissible.

Computer forensics, a relatively new field of research, needs a method with appropriate authentication mechanisms in place by which syslogs can be used as relevant evidence in court. Syslogs, which have been designed more from an event logging perspective than an evidence-oriented one, are system treasure maps that chart and pinpoint attacks and attack attempts. Over the past few years, research on securing syslogs has yielded enhanced syslog protocols that focus primarily on tamper prevention and detection. However, many of these protocols, though effective from a security perspective, are inadequate when forensics needs comes into play.

Security research on logs has focused on securing audit logs and protecting them from intrusion and malicious manipulations. This is exhibited in syslog variants such as syslog_reliable [10] and syslog_ng [11]. These variants do not deal with the specific needs of forensic viability. This essentially entails the validation of syslog entries as they are created as well as providing resistance and detection of modifications and deletions. The research presented by Jiqiang et al. [3] presents a schema that describes a secure logging architecture from a forensic viewpoint. However, although the aim suggests securing audit logs for use in forensic analysis, the method presented in the paper does not get into the necessary details of

validating log entries and the manner in which they will actually be scanned for their authenticity or tested for their genuineness.

The goal of the research presented here is to create a forensically viable syslog facility. By building on our proposed methodology described in [5], this work discusses our implementation of the method and the ability for the method to identify attacks and analyze those attacks. No other research has focused extensively on making syslogs forensically viable.

The work in [5] documents the background of previous syslog related work, the various variants of the syslog protocol, their failings, and describes briefly the techniques that have been used over time to secure logs, either system ones or audit logs. The authors show that the proposed method satisfies all the goals of digital forensic evidence formulated by Dixon [2].

The mechanism proposed in [5] generates authentication traces, which are succinct messages that compliment the corresponding syslog entry and comprise a message and digital fingerprints of the entities involved. All the hosts on the network are assumed to be unsafe and therefore, the transmission of the authentication traces between the server and each host system is secured by using a challenge response mechanism, which is a modification of the Needham-Schroeder protocol. The original protocol was formulated with the aim of securing communication pathways between unsafe hosts.

2. Background

Every computer-based activity on a system typically leaves an electronic trace [9]. The level of understandability provided by these traces and the credibility offered by them depends on the level of security in place on the system. Electronic traces in verifiable forms can be considered as digital evidence. In order to verify system log files we must ensure that the log files are resistant to deletions and modifications; i.e., it may not be possible to prevent truncation of a log file but such modifications must be detectable. Additionally, further verification must be added to the syslog protocol to validate where the syslog entries came from. Specifically, this is done using system fingerprints, user fingerprints, and application fingerprints.

The weakness of the syslog protocol [6] lies in the fact that it uses UDP, a connectionless and unreliable protocol, stores system event information in plain text format, and transmits system event data across the network in plain text format. With regard to the three components of security—authenticity, confidentiality, and integrity—syslogs can be manipulated by a malicious insider or an outside attacker by exploiting these inherent weaknesses. Thus, all three components expected of security can be violated. The immense size of the syslog file, the lack of a solid relationship between the entities that generate a syslog entry, the ease of availability of tools like crypt-cat and netcat make compromising the authenticity, confidentiality, and integrity easy.

Confidentiality of logs: Several open source network tools, such as tcpdump, can be used to capture syslog entries that are transmitted in plain text to a central logging system. A wily attacker can analyze the contents of these packets to determine the corresponding syslog entry. This compromises the confidentiality of syslog files since the attacker now knows the kind of spurious entries that should be injected into the syslog file to camouflage their activities.

Integrity of logs: Syslog entries that are stored on a central logging repository are open to being manipulated by an attacker. In addition, the UDP protocol that is used to transmit the syslog entries is exceedingly vulnerable to capture, replay, and various man-in-the middle attacks.

Authenticity of syslogs: Currently, there is no means to forensically associate the system and the service that generated a syslog entry. That is, there is no means to authenticate the fact that a service on a system has generated an entry. Readily available tools such as netcat,

cryptcat, etc. can be used to flood a logging server and subject it to attacks such as intentional flooding and denial of service.

Matt Bishop has defined authenticity, integrity, and confidentiality to be the basic tenets of any system or entity that aims to be secure. The violation of any of these tenets results in the entity being termed as vulnerable, such as the syslogs under consideration.

Our prior work [5] proposed a new electronic trace by using a modification of the Needham Schroeder protocol [7]. The secure transmission of system fingerprints, user fingerprints, and application fingerprints is ensured by using a modification of the Needham Schroeder protocol. This protocol was developed to secure communication between two hosts by the use of session keys, random numbers, and nonces. In this method, the session keys are replaced by public keys for each system on the network. We term the public keys assigned to every authentic system, K_{System}. Similar to the original protocol, these keys are generated pseudo randomly at the authentication module and are assigned to each of the systems. The weakness of the Needham Schroeder protocol lies in the use of timestamps. In the originally suggested protocol, timestamps were used explicitly. The use of timestamps explicitly enables the manipulation of messages by changing the network clock and manipulating network latency. However, this is eliminated in our proposed version due to the use of digital fingerprints, which are hashed values of various system parameters and timestamps.

Research in [1] has assigned levels of credibility to forensic evidence similar to the Fujita scale, which determines hurricane magnitudes. The level of evidence that this proposed mechanism provides maps on to the C5 level, thereby implying that evidence is tamperproof and asserts a match between independent sources of evidence, which in this case are the authentication traces and syslogs. The evidence at this level, however, can be erroneous due to temporal loss or data loss.

The simplicity of the protocol used for its transmission (UDP) and the plaintext format, in which they are stored, make syslogs feeble sources of evidence. There is no way a patchy log on a server can be used as evidence to prove the occurrence of an attack in a court of law. On the same lines, the ease by which the syslog protocol can be tampered with reinforces the need for a backbone umbrella mechanism in place that will still hold in the event that the syslog mechanism fails. However, while positioning a forensic-friendly mechanism on the network, making it secure and resilient against attacks such as the man-in-the-middle, spurious syslog entry injection, etc.

Currently, there is no protocol in place that forensically links together every entity involved in the generation of a syslog entry. The method that we propose ties together every entity involved in the generation of a syslog entry using digital fingerprints and authentication traces. Syslogs need a mechanism where the authenticity of every entity involved in the generation of a syslog entry is vouched for. The digital fingerprints are generated using the RS algorithm. Cryptographic algorithms were not exhaustively considered for the formulation of this prototype and therefore, the RS algorithm was used here. A more intensive implementation in the future will make use of cryptographic hash functions. However, the authentication of hashes, which are indeed the digital fingerprints, is challenging and has yet to be explored in further implementation.

3. Model Overview

Figure 1 shows an overview of our proposed model in the form of a UML sequence diagram. Currently, in order for syslogs to be worthy of being considered as evidence in forensics, what is needed is an authentication mechanism that reinforces and authenticates what the system log file presents. The entities involved are the user, the application, the system, the client syslog

daemon, the authentication module, and the syslog server. The client syslog daemon and the syslog server are not shown explicitly in this overview diagram.

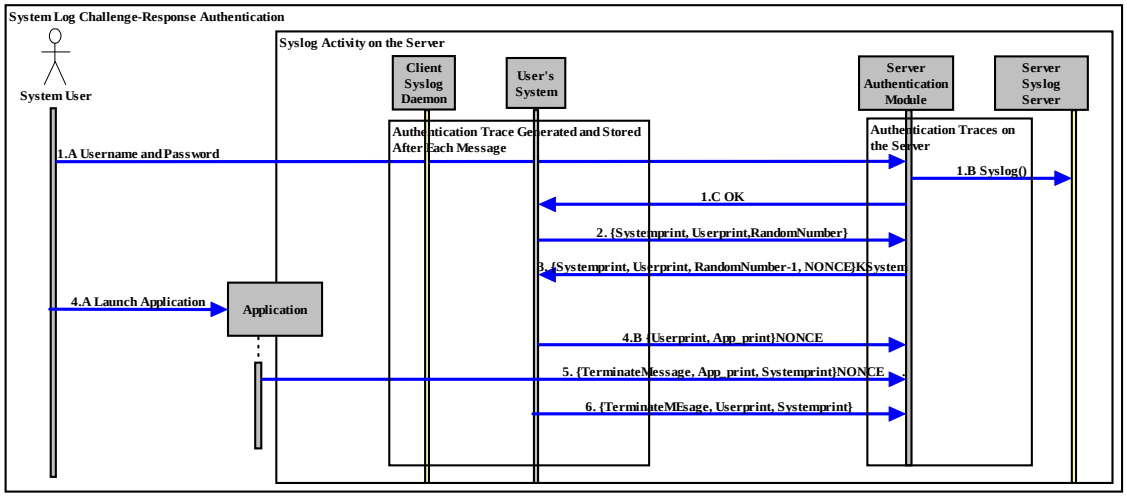

Figure 1: Sequence diagram of the proposed system. The entities are represented along the x-axis and time is represented along the y-axis.

In our proposed protocol, there are two servers, an authentication server and a logging server. The authentication server records every authentication that occurs and maintains their timestamps. Since this server needs to act as a form of backup in the event that system logs on the logging server are tampered with or additional evidence is needed to verify a claim, it will have a minimum number of processes running, limited accessibility, and constrained resource availability. Further, this server can decipher the entries in the individual prints and verify the authenticity of a fingerprint. The logging server stores actual log entries and is the main storage system for these log entries.

In addition to the background processes of syslog generation and authentication trace generation, which are umbrella processes that exist throughout a session, the proposed approach comprises three main steps, which in turn comprise one or more phases within themselves. More details of these steps, including their formal specification, can be found in [5].

3.1 User authentication

This is based on desired login authentication procedures and is geared toward ensuring that only authorized users access the system. The user is authenticated by the server.

3.2 Challenge response before the user, system, and application become active

This step encapsulates and comprises the generation of user fingerprints, application fingerprints, and system fingerprints. Furthermore, in order to cement and secure the transmission of these fingerprints and the authentication traces, which are generated by individual systems, several challenge response steps have been incorporated.

- **Phase 1:** System Connection Establishment
- **Phase 2:** System Connection Establishment Response
- **Phase 3:** Application Event Entry Generation

3.3 Messages log the termination of the application, logging off of the user, and the shutting off the system

This authentication mechanism ensures that an entity granted login privileges is logged in and is the entity sending event messages. However, with regard to computer forensics, a mechanism to verify the termination of an authorized entity is also needed. This step details a secure and logged termination of the entities involved in the generation of a syslog entry.

- **Phase 1:** Application Termination
- **Phase 2:** System Connection Termination

4. Fingerprints

In physical forensics, fingerprints are one of the key factors that reveal evidence about the perpetrator or identify key entities (people or objects) involved in a crime. Creating digital versions of fingerprints of every entity involved in the generation of a syslog entry promotes and emphasizes the need to make every entity responsible for ensuring its forensic viability.

4.1 User Fingerprints

User fingerprints tightly bind the user and the system used. The user print can be considered as simulating a real life fingerprint. When a fingerprint is considered in the real world, factors such as location and time are also taken into account before arriving at conclusions. Thus, for the cyber version of user fingerprints, similar types of information must be included; i.e., user identifying characteristics, time, and system identifying characteristics. This ties a specific user to a particular system at a specific time. More specifically, we propose using the following to create a user fingerprint:

- Username and password
- System mac address
- Login time

4.2 Application Fingerprints

Application fingerprints are similar to user fingerprints. The application fingerprint will be generated for every application that is launched on a system. Their primary role is to identify and distinguish between legal applications and illegal ones launched by specific users on a system. As with user fingerprints, the goal is to provide as much identifying information as possible. In this case, we are attempting to validate what application is being run, by whom, when, and from where. Thus, application fingerprints would use the following pieces of information:

- Launch time
- Username
- System mac address
- Application identifier

4.3 System Fingerprints

System fingerprints are often used by operating systems manufacturers to register the system that the operating system was installed on and ensure it is not transferred to a new system in violation of the operating system license. The concept of system fingerprints essentially relies on the fact that once deployed most systems rarely have their configuration change, especially

in business environments. For home users, while some sophisticated users will upgrade individual components of their system, the majority of home users will not. Many different characteristics can be used to identify a system uniquely. Some possibilities include:

- The number of processors
- Disk space
- System mac address
- CPU ID
- Installed applications
- Disk drive identifier, serial number

4.4 Authentication Traces

An authentication trace is an entry that is generated on every system on the network and records the generation of system, user, and application fingerprints along with the associated timestamps. Authentication traces on each system can be viewed only by administrators. The traces will typically be a message along with the prints and the timestamp of the event.

5. Fingerprint Generation

User fingerprints, application fingerprints, and system fingerprints are generated using the RS hashing algorithm, which is known to have low collision rate. The RS algorithm, which is a general-purpose hashing algorithm developed by Robert Sedgwick [8] is used to generate hashes, i.e., fingerprints.

Robert Sedgwick's hashing algorithm is a rotative hashing algorithm that uses rotative hashing [1]. In rotative hash functions, unlike its counterpart, the values are bit-shifted. Sometimes combinations of both right and left bit shifts are used. For increased security, bit shifts are sometimes prime numbers. The intermediate value that is yielded at each step is added to an aggregative value. The result that is yielded is the value of the final aggregation. An example:

$$hash = hash^{-1} \oplus (t << p) \otimes (t >> q)$$

The algorithm is coded as follows. However, different keys are used for the user, application, and the system fingerprints.

```
for(int keyLength=0; keyLength<fingerPrintKey.length();
    keyLength++){
  long intermediateUserChar = (long)
       fingerPrintKey.charAt(keyLength);
  fingerPrintH = (fingerPrintH << 4) + intermediateUserChar;
  fingerPrintG = fingerPrintH & 0xF0000000L;
  if (fingerPrintG != 0)
    fingerPrintH ^= fingerPrintG >>> 24;
  fingerPrintH &= ~fingerPrintG;
}
return (long)(fingerPrintH);
```

This user print yields→ `155990563`

For the user fingerprint, the key is a concatenation of the username, the time of user log in, and the user ID that was generated when he/she logged in. Keys in a hash function are required

to be unique so as to avoid collisions and enable faster look up. The keys here are concatenation of three parameters that will most certainly be unique across logins in an organization.

The system fingerprint is generated in the same way. We have found that the hard disk serial ID that is hardcoded by a manufacturer is the only unique parameter than can actually distinguish one system from another. The hard drive serial IDs, which are assigned to every partition on the hard drive, were another parameter that was considered. However, these IDs can be changed when the disk is reformatted. Another parameter that was considered was the CPU ID. A run of an application on laboratory systems revealed that all CPU IDs that belong to computers ordered in bulk are the same. The MAC address was not considered as a potential parameter due to the ease by which a person with reasonable computer knowledge can change and even spoof a MAC address. The key used in this case is the hard disk serial ID. This was identified and verified to be unique. Therefore, the hard disk serial ID and the system bootup time are together used as a key for generating the system fingerprint. The system print is generated in the following way:

```
systemFingerPrint(){
  String HDDSerialNumber= "97LET9BET";
  String systemFingerPrintKey=
         HDDSerialNumber.concat(systemBootupTime);

  for(int keyLength=0; keyLength<systemFingerPrintKey.length();
      keyLength++){
    long systemIntermediateChar=(long)
         systemFingerPrintKey.charAt(keyLength);
    systemPrintH = (systemPrintH << 4) + systemIntermediateChar;

    systemPrintG = systemPrintH & 0xF0000000L;
    if (systemPrintG != 0)
      systemPrintH ^= systemPrintG >>> 24;
    systemPrintH &= ~systemPrintG;
  }
return (long)(systemPrintH);
}
```

An example of a system fingerprint yielded by this method → `161044579`

```
applicationFingerPrint(String applicationName, long
applicationID, String appLaunchTimestamp) {

  for(int keyLength=0;
      keyLength<applicationFingerPrintKey.length();
      keyLength++){
    long appIntermediateChar = (long)
         applicationFingerPrintKey.charAt(keyLength);
    appPrintH = (appPrintH << 4) + appIntermediateChar;
    appPrintG = appPrintH & 0xF0000000L;
    if (appPrintG != 0)
      appPrintH ^= appPrintG >>> 24;
    appPrintH &= ~appPrintG;
  }
```

```
return (long)(appPrintH);
}
```

An example of an application print yielded by this method → `76274804`

The application fingerprint is necessary in order to validate applications. Here, a concatenation of the username, application ID, and the applicationTimeStamp is used as the key in the fingerprint generation.

The extent of this implementation is the generation of the authentication traces, digital fingerprints, and the simulated syslog entries. The implementation was carried out with the aim of deriving a prototype of the proposed method. The challenge response mechanism will be incorporated as part of the remaining implementation.

6. Backtracking an Attack

Syslog entries typically comprise the following parameters—hostname, facility, priority, message, and timestamp. This implementation simulated a syslog logging facility. The purpose of this was to compare an authentication trace and be able to get to the fingerprint from the syslog. After an attack occurs, parameters from the syslog can be used to obtain the corresponding entry contained in the authentication trace. An important point to be noted is that time is a crucial factor in the generation of an authentication trace and the corresponding syslog entry. The user, application, and system are the facilities considered in this implementation of syslogs. Their priorities are hardcoded here since this implementation mainly serves as an example and validation of how authentication traces and syslog entries can be used in tandem to trace back and form evidence. The research in [5] suggested that authentication traces can be used to back track to an attack. This paper shows this can be actually carried out. This is because every parameter that is considered in the generation of a fingerprint can be essentially obtained from the corresponding syslog entry in the log file. Therefore, this paper shows the way in which attacks detected in the syslog entry can be backtracked using a combination of the authentication traces, the syslog file, and the hash function (here, the RS algorithm).

An example:

The following authentication trace shows a login by user “steena” and the corresponding syslog entry.

```
steena logged in at 2008-02-06 12:49:33 with user ID
   7524389880967786033 with user finger print 155990563 the
system print is161044579
C:\Program Files\Internet Explorer\IEXPLORE.exe launched at
   2008-02-06 12:49:41 with ID   1524843500148472672 with
fingerprint  88504721
```

The corresponding syslog entries with format host name, facility, priority, message, and timestamp.

```
localhost 4  10 steena has logged in at  2008-02-06 12:49:33
localhost 6  12 C:\Program Files\Internet Explorer\IEXPLORE.exe
launched at     2008-02-06 12:49:41
```

Repeated bad logins at a particular system will yield corresponding authentication traces and syslog entries. However, the occurrence of repeated bad logins will be logged by the authentication traces and not by the system logs unless they are configured to do so.

```
Incorrect login with username:   steena occurred     at2008-02-07
04:50:02  with userID  56032638045929763with           userprint
   188996098
 Incorrect login with username:  steena occurred     at2008-02-07
04:50:28  with userID  893624388610789281 8with         userprint
   188996200
 Incorrect login with username:  steena occurred     at2008-02-07
04:50:43  with userID   2404564924573438423    with    userprint
   188996163
```

An attack by a malicious insider will cause the username, which is known to be exploited. In this simple emulation of system logs, we have explicitly logged a bad login instead of a series of repeated logins by a valid user.

6.1 Reconstruction of a User Fingerprint

The user fingerprint comprises the username, the user ID, and the time of login. These values can be obtained from the syslog entry. A hash of these parameters using the RS function will yield the corresponding fingerprint. The absence of authentication traces would only reveal the persistent login by user “steena.” A closer examination of the system logs and its corresponding authentication trace can even possibly reveal the identity of the person behind the attack. A small script to check and match users who have already logged in and their log in times can possibly reveal this. A more complex implementation aims to assign appropriate priorities and facility numbers to every entity involved in the system.

An important point to be noted while logging events to a central repository is that the local system time for each individual system should be used instead of the server time. This is because authentication traces are generated and are representative of activity by entities on those individual systems. The use of server time would lead to misinterpretation of events on those systems. This was noted during the current implementation when entries were being logged successfully but had a clear disparity with regard to timestamps in their corresponding authentication trace entries.

7. Use of Authentication Traces and Syslogs Under Certain Scenarios

Authentication traces and syslogs can be used in other circumstances other than backtracking an attack, which of course, is its primary aim. The three scenarios below attempt to exemplify some of these characteristics. For these scenarios, consider the fictitious entity SecurityVille. SecurityVille is an organization where every user has a dedicated system and a login username and password. Andy is the administrator; Fred is the forensic analyst; Steve is the malicious insider, who is also an employee; William is a wily external attacker, and Arby is another employee. Authentication traces are maintained on every system and on the server. Syslogs are maintained only on the server.

7.1 Scenario One: Syslog File Deletion

The SecurityVille network has been taken offline due to an attack by William. Fred knowing the immense repository of information that syslogs contain begins searching for the syslog file on the server. However, William knowing this too, has deleted it.

The authentication traces serve as complimentary evidence. Although the fingerprints are indecipherable at a glance, a further inspection of the authentication traces can yield an almost complete reconstruction of the syslog file, thereby showing the origin of the attack, its modus operandi, and to a limited extent the severity of the attack.

7.2 Scenario Two: Spurious Entry Injection into the Syslog File

During a fortnightly inspection of the syslog file, Andy notices that certain entries appear to be invalid, i.e., not matching the authentication traces. Clearly, someone has managed to alter the syslog file on the server. The corporate network logs, router logs, and switch logs do not reveal any suspicious activity. As it happens, the attack originated from an internal source: Steve has managed to gain access to the server and injected spurious entries into the syslog.

An inspection of his authentication trace reveals that he has managed to install a rogue application on his system. His traces reveal the name of an unknown application.

7.3 Scenario Three: Application Updates

FortyTwo, which is an accounting software used by the employees is scheduled to undergo updates every two weeks.

In the method proposed here, before an application launches, it needs to go through the challenge response mechanism. The application fingerprint is then calculated on the fly. When the application has been updated and has to restart, its print is recalculated and the restart is treated as the launch of a new application. Since application IDs are assigned on the fly and are documented, the automatic updates would not affect the generation of the application prints and their transmission. Currently, the authentication trace generation has no mechanism to determine if an update has occurred or if the user has merely chosen to close and launch the application again. However, a close examination of the traces across systems and the system logs would reveal this update if a pattern of restarting an application is seen across multiple systems. Further, since the application name is listed in the authentication trace, this pattern will be readily found. An application update occurring while the application is not running would not lead to any suspicious traces, the desired result.

8. Conclusions

The proposed model aims to provide a mechanism to authenticate and validate syslogs. Although syslogs have been researched extensively from the security perspective, they have not received sufficient attention from the forensics point of view and the need for legal admissibility. The fingerprints assigned to every entity involved in system log generation will enable the validation of these entities. More importantly, since digital evidence is treated in the same way as documentary evidence [4], a means to authenticate and verify its authenticity is needed. The proposed model aims to provide resilience against common attacks launched against syslogs—system log truncation and man-in-the-middle attacks, which are currently of the most significant problems, associated with using system logs as evidence in court. For

instance, the credibility of system log files as evidence could easily be attacked in court and invalidated.

With the proposed method, if a malicious insider carries out suspicious activity, this activity can be traced back to the offender. Their system identity can then be forensically verified by hashing the values available in the syslog file and the authentication traces, using the RS algorithm, and matching them with the prints in the authentication traces. This mechanism is limited to be able to trace back to insiders. The ability to trace back to an outside attacker is beyond the scope of our proposed method, though the internal compromised identity would be identified.

9. Future Work

This paper focuses on the examination of the implementation of components of the proposed model. The paper also examines how the model would be used and behaves in real world scenarios. This is merely the first step in implementing and validating the proposed model. Already, the effectiveness of the model is becoming apparent. Next steps will include a more complete testing platform and actual simulation of real world attacks and anomalies. These tests will validate the resilience of the proposed method against expected attacks, for instance:

1. Denial of service attacks—the aim here is to bombard the server with syslog messages and try to use the authentication traces to recover the system and the network again. Since authentication traces are more succinct, informative, and secure than the traditional syslog messages, the ability to recover a system and identify the attack will be the focus. This would test the effectiveness of the authentication traces to actually hold the framework in place when the server is attacked from outside the local network.
2. Attacks against the syslog file—truncation, spurious entry injection, and deletion. One of the main issues that syslogs have to deal with is the abrupt truncation and deletion of syslog entries in the log file. An attacker could randomly delete syslog file entries or the syslog file as a whole. Syslog entries can be recovered by referring back to the authentication traces.
3. Man-in-the middle attacks against the challenge response framework. This is an attack against our proposed protocol at a fundamental level. This test will validate the resilience of the protocol against attempts to break down the challenge response framework.
4. Rogue applications—detection and identification of rogue applications on the network. The ability for rogue applications to compromise the integrity of the syslog files remains a concern. Thus, the goal of this test is to validate the resilience of the application fingerprints and the corresponding authentication traces.

10. References

[1] Casey, E., "Error, Uncertainty, and Loss in Digital Evidence," *International Journal of Digital Evidence*, Vol. 1, No. 2, 2002, pp. 1-45.

[2] Dixon, P.D., "An Overview of Computer Forensics," *IEEE Potentials*, Vol. 24, No. 5, Dec. 2005, pp. 7-10.

[3] Jiqiang, L., Zhen, H., and Zengwei, L., "Secure Audit Logs Server to Secure Logs to Support Computer Forensics in Criminal Investigations," Proceedings of the 2002 IEEE Region 10 Conference on *Computers, Communications, Control and Power Engineering*, October 2002.

[4] Kurzban, S., "Authentication of Computer Generated Evidence in United States Federal Courts." *The Journal of Law and Technology*, 1995.

[5] Monteiro, S., and Erbacher, R.F., "An Authentication and Validation Mechanism for Analyzing Syslogs Forensically,"*ACM SIGOPS Operating Systems Review*, 2008, *To Appear*.

[6] Nawyn, K. E., “A Security Analysis of System Event Logging with Syslog.” SANS Institute, no. As part of the Information Security Reading Room. (2003).

[7] Needham, R. and Schroeder, M., “Using Encryption for Authentication in Large Networks of Computers,” *Communications of the ACM*, Vol. 21, No. 12, 1978, pp 993-999.

[8] Sedgwick, R., *ALGORITHMS*, Second Edition, Addison-Wesley Publishing, Company, Inc., 1988.

[9] Volovino, L., “Electronic Evidence and Computer Forensics,” *Communications of the Association for Information Systems*, Vol. 12, 2003, pp. 457-468.

[10] http://www.ietf.org/internet-drafts/draft-ietf-syslog-sign-21.txt

[11] http://www.balabit.com/network-security/syslog-ng/