2012 IEEE 14th International Symposium on High-Assurance Systems Engineering

A Resilient Architecture for Forensic Storage of Events in Critical Infrastructures

Muhammad Afzaal, Cesario Di Sarno, Luigi Coppolino, Salvatore D'Antonio, Luigi Romano
Department of Technology
University of Naples "Parthenope"
Naples, Italy
muhammad.afzaal,cesario.disarno,luigi.coppolino,salvatore.dantonio,luigi.romano@uniparthenope.it

***Abstract*—In Critical Infrastructures, forensic analysis of stored events is an essential task when a security breach occurs. The goal of forensic analysis is to provide evidence to be used as valid proofs in a legal proceeding. So, it is very important to ensure the integrity of the events stored in order to perform a correct forensic analysis. Today, most of the SIEMs used to protect the Critical Infrastructures sign the security events with RSA classic algorithm in order to ensure their integrity. The signed security events cannot be admissible as evidence if the secret key is compromised, or when the module responsible for signing operations is down for any reason. In this paper a new architecture that overcomes these limitations has been proposed. Experimental tests show the performance of our architecture and the high resilience in faulty situations, i.e. some nodes are under attack.**

***Keywords*-Forensic Storage; Threshold Cryptography; Critical Infrastructure Protection; Fault- and Intrusion-Tolerant Architecture;**

I. Rationale and Contribution

Critical Infrastructures (CI) [1] are often monitored through Security Information and Event Management (SIEM) systems which gather the events generated by devices and sensors and process them in order to avoid or mitigate the security breaches. Recently, the storage of the events related to the security breach has been proven necessary for forensic purposes. In fact, if an attacker has successfully performed a security breach, it is possible through the chain of events to find his identification. These events can also be used to try the attacker in court but they must be stored in raw format without any processing to be admissible in court. So, the major SIEM systems provide a forensic support for secure storage of events. We have analyzed two SIEM products, Assuria Log Manager (ALM) and OSSIM.The forensic storage of ALM is based on digital signatures of the security events with classic RSA [2] with SHA-256 hash function [3]. OSSIM SIEM developed by AlienVault offers the forensic storage only in its commercial version. The user manual of this product shows that it uses the classic RSA algorithm to sign the important events. The weakness of current forensic storages is that they are based on classic RSA algorithm to sign the events. RSA algorithm offers a good protection if the attacker tries through the cryptanalysis [4] to read the contents of the message. But as suggested in book [5], "Your opponent always uses her best strategy to defeat you, not the strategy that you want her to use". In fact, the classical RSA algorithm, based on a pair of public and private keys, is not tolerant to intrusions and faults because the secret key that is used to digitally sign the events is stored at one place and can be compromised. The easy search of an RSA key from hard disk drive [6] can nullify the admissibility of security events for forensic purposes. Also, if the module that signs the events is down due to DoS attack or some other reason, the system is not available to sign the security events at the time of possible security breach. If an adversary successfully steals the secret key used to sign the security events, he can easily forge the digital signatures and send such events to the database which may not help in identifying the reasons of security breach.

In this paper we present a new architecture and implementation of a forensic storage that overcomes these limitations. In fact, using threshold signature scheme instead of classic RSA, we have obtained benefits such as: unforgeability of data if some participants of the algorithm show abnormal behavior; an intrusion and fault tolerant system; identification of the participants of threshold crypto scheme that show abnormal behavior.

In order to obtain an architecture with high performance in terms of number of events stored, this is realized as distributed application. Each component of the distributed application is made using multi-threaded model programming. Also we have used an optimized data structure that allows the maximum level of concurrency between threads that perform write operations simultaneously. We even integrated our forensic storage into OSSIM SIEM to provide a new version of advanced SIEM system that ensures the integrity of data even in faulty situations. Finally, through experimental tests we have shown the performance of our Forensic Storage and how the integrity of data is ensured even in presence of attacks.

II. Background

We have used RSA Threshold Signature scheme instead of classical RSA signatures because of ability to tolerate intrusions and faults and more flexibility provided by the former technique. Threshold Cryptography allows a group of n parties to participate in the digital signature process to enforce authenticity and non-repudiation. The basic idea is to divide the secret key in n shares, where n is the number of shareholders or participants to the algorithm and to set a threshold value called k where participation of at least k shareholders is necessary in order to construct the secret. The strength of Threshold Crypto technique lies in the fact that the secret key should be divided among the shareholders in such a way that not less than the threshold number of shareholders can build the secret key and none of the shares reveals even partial information that may be helpful to construct or have an idea about the secret key.

Different methods for dividing the secret key into different shares have been proposed. In 1979, Adi Shamir [7] and Blackley [8] proposed independently the first secret share scheme.

In our work we use the threshold signature scheme proposed by Shoup [9] because it is the first practical scheme to implement digital signature in a threshold mode. Previous works on threshold signature schemes were research oriented and dealt with only mathematical details of the digital signature generation using RSA whereas Shoup's scheme provides the full algorithm to implement the digital signatures using RSA in a threshold mode.

III. Resilient Architecture for Forensic Storage

In this section we deal with the storage units dedicated to the archival of critical security events, which require properties like integrity, confidentiality, and unforgeability. A resilient architecture for forensic storage is a facility that allows secure storage of data by implementing a set of techniques which make data alteration difficult to achieve. The architecture of the forensic storage is shown in Figure 1.

The Security Events are the input data to the architecture. The incoming data carry information about the events related to a security breach. In order to be able to use this data for forensic purposes, it should be stored in raw format without any kind of processing. So, the Security event "m" in Figure 1 is considered to be in raw format. The first component starting from the left is the "Threshold Cryptography" block. We suppose that there are n participants to the algorithm, and they are indicated as $Node\_1$ $\ldots Node\_n$. After the secret key is divided into shares and distributed to all shareholders, the incoming message (containing information about the security breach) is sent in parallel to n nodes. Each node computes a hash function of the same message. The hash function returns a unique digest for this message, named h in Figure 1. The next step for each node is to encrypt the digest in order to produce a signature share (or partial signature) represented by $Sigshare\_1 \ldots Sigshare\_n$. For this purpose, each node has its own secret key share that is used to encrypt the digest. A component, called Combiner, is responsible for assembling all partial signatures received from the shareholders in order to generate a complete signature which is attached to

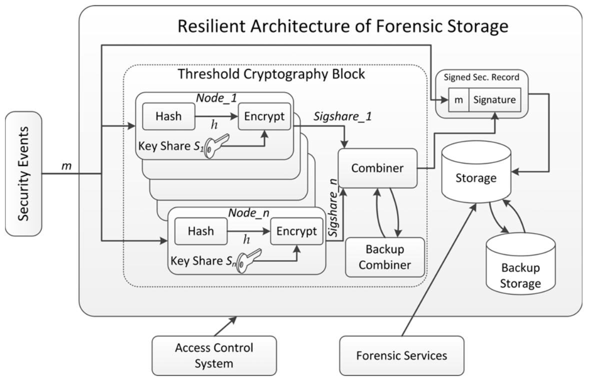


Figure 1: Resilient architecture for forensic storage.

the original message, thus forming a signed security record, i.e. a forensic record. This new record is stored for future forensic analysis. To ensure unforgeability of the records, it is common to use Write-Once-Read-Many (WORM) storage devices to preserve such records [10]. However, records can also be stored in any kind of database, if provided with mechanisms of tamper detection [11].

In Figure 1, Forensic Services provide forensic analysis services which are able to determine which data were altered, and also who the intruder is, through the information obtained by the recorded events. The Access Control System is the service that controls who can interact with a resource and how. Replication and diversity are employed to further improve the intrusion and fault tolerance in Forensic Storage. In particular, the combiner and the storage are single point of failure components within the architecture, so we chose to employ redundancy and diversity in order to avoid this weakness. Replication introduces well-known data consistency issues that can be resolved using an optimistic data consistency method [12].

IV. Design and Implementation details

The architecture described in section III has been implemented as a distributed application which is designed in a similar way to the SOA[1] architecture [13]. So, each functionality is offered as a service. When a new component has a service to offer, it must register it in a registry first. In the same way when a component wants to use a service, it performs a search within the registry as shown in Figure 2.

The registry provides a directory service and so the forensic storage can be written as a choreography of services. The registry can be a single point of failure in the architecture. To avoid this weakness, in the implementation phase, the Reliable Registry of Alberto Montresor is used [14].

The technology used to implement the distributed architecture is Java-RMI[2]. In order to improve the level

[1]Service Oriented Architecture

[2]Java - Remote Method Invocation. More details are available at http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html

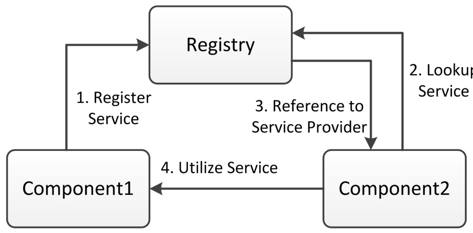


Figure 2: An example of registration and search of a service in Registry

of security when the different services communicate with each other, only communication through SSL is allowed. The chosen authentication mode is bilateral. Certificate management in Java is made possible through the use of key-store and trust-store. So all certificates have been pre-generated and recorded in the key-store and trust-store. All the components that participate in the threshold signature scheme are described below.

A. Dealer

Dealer is responsible of three things: it provides a service for generation of secret key shares, distribution of these key shares to each participant or node involved in the architecture and the distribution of verification key shares to the combiner component. In particular the service provided in the following method is used by external modules:

```
public void generateKeys(int keySize,
        int threshold, int groupPlayers)
        throws RemoteException;
```

The `threshold` and `groupPlayers`, as evident from names, are the parameters of the Shoup [9] scheme, while `keySize` represents the size in bits of each key share that will be generated. Before the method ends, a search is made in the Registry by the Dealer to find the Combiner component and the nodes responsible for partial signatures generation. The Registry informs the Dealer which nodes are available and also provides the way to contact them (the same applies to the Combiner). So, the Dealer sends each secret key share to each node. Also, it sends all the verification key shares and the full verification key to the combiner component. When each node has received the share of secret key, it sends an acknowledgment to the Dealer. If the Dealer receives acknowledgments from all nodes and the Combiner, it deletes all generated key shares. Otherwise a new log is generated with an exception. The Dealer component works only during the initialization phase. So, it is considered reliable during the generation and distribution of secret key shares.

B. Nodes

Each node is responsible of two things: when a new message arrives it calculates the hash value of the new message and then encrypts this hash value with the secret key share. The services provided by each node are accessible through two methods

```
public int setKey(BigInteger
        secretKeyShare)
        throws RemoteException;
public void setMessage(String message,
        int id) throws RemoteException;
```

The `setKey` method is used in the initialization phase. In fact when the Dealer has generated the secret key shares it distributes each secret key share to each node invoking this method. In the initialization phase, each node searches in the Registry for the service provided by Combiner component. The `setMessage` method is used from the outside to send a new message to the node. When a new message arrives, a new thread is created. The thread uses the hash function SHA-256 to calculate the digest value. Then, the thread uses the secret key share to generate the signature share. Finally, the thread sends the signature share and the id of the message to the combiner component. The id value is required by the combiner component in order to associate received signature shares to the correct message.

C. Combiner

We can divide the behavior of the combiner in two phases: initialization phase and normal phase. In the initialization phase it:

- creates the link with a database where the signed events will be stored. To protect the connection between JDBC driver and the server, we used JSSE[3] which allows the use of protocols such as SSL [15] and bilateral authentication to ensure secure communication
- receives all verification key shares to check the integrity of signature shares of each shareholder. It also receives the verification key that can be used to verify the validity of complete signature after that the combining process has been completed
- creates a hash table to store temporarily all messages and their signature shares received from multiple nodes
- creates two kinds of threads: producer and consumer which are used during normal phase.

In the normal phase the producer thread has two tasks: it stores the new incoming events in the hash table against their id and stores the signature shares sent by nodes corresponding to each event. The consumer thread works asynchronously. In fact, once it has been created, it periodically reads the whole hash table to check if it is possible to generate new complete signature. Since the time to verify all signature shares is higher than the time to verify complete signature, we always use the optimistic approach. In fact,

[3]Java Secure Socket Extension. It is included in JDK since its version 1.4.1

when the combiner receives at least k (threshold) signature shares, it generates the complete signature. Then, it verifies the integrity of this complete signature with the verification key [9]. If the complete signature is valid, it is stored in database together with the original event. If the complete signature is not valid, the combiner explores in depth each signature share and checks their integrity with corresponding verification key share. If the signature share is valid a flag is set to the true value, otherwise the flag is set to false. In this way, the next time when a new signature share arrives from the same event source, the combiner takes the first k signature shares excluding the signature share whose flag is set to false and generates a complete signature. If the complete signature is valid, the combiner stores in the database the event with its valid signature along with the faulty node otherwise the process is repeated. When a faulty node is identified, the system administrator will be informed about this in order to take necessary security measures to bring it in working condition.

The services provided by the Combiner are accessible through three methods:

```
public void setParameters(BigInteger
        verificationKeyShares[],
        BigInteger verificationKeyComplete)
        throws RemoteException;
public void setMessage(String message,
            int id) throws RemoteException;
public void setSigShare(BigInteger
            sigShare, int id, int pos)
            throws RemoteException;
```

The `setParameters` method is used in the initialization phase to store all verification key shares and full verification key. The method `setMessage` starts the producer thread that stores the message in the hash table using `id` value as key. The method `setSigShare` starts the producer thread that adds in the hash table the new signature share `sigShare` from node `pos` to the message with id value equal to `id`.

1) Optimization of the Data Structure: The methods shown in the code above `setMessage` and `setSigShare` work in parallel writing on the same hash table. Moreover also the consumer thread writes a flag in the hash table if a valid signature share or a full signature is found. This problem is known as the producer-consumer problem. There are many solutions to this problem and one of them is to perform one write operation at a time. We have found a new way to overcome this limitation partially. In particular, better organization of data in the data structure allows some write operations to be performed in parallel. The data structure that we have chosen is shown in Figure 3.

Using this data structure when a new signature share arrives, the producer thread adds it to the field `arraySignatureShare` of a message with id equal to

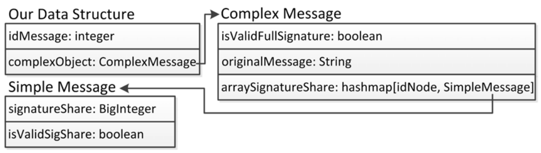

Our Data Structure
idMessage: integer
complexObject: ComplexMessage
Complex Message
isValidFullSignature: boolean
originalMessage: String
arraySignatureShare: hashmap[idNode, SimpleMessage]
Simple Message
signatureShare: BigInteger
isValidSigShare: boolean

Figure 3: Data structure used to handle all signature shares in the combiner

given `idMessage`. Given that the key of the hash table `arraySignatureShare` is the id number of the node that has generated the signature share, the next time a signature share arrives to the combiner, the producer thread does not change old records anymore. This is because, each node generates only one signature share for a given message. So, when the consumer thread checks that a signature share is valid, it can write the flag `isValidSigShare` directly without generating race condition with the producer thread. This optimization allows improving the performance of the global system.

D. Controller

The controller component does not provide any service, instead it searches the services in the registry and triggers the relevant components. It is the entry point of the whole architecture in Figure 1. In fact, it can be contacted from outside of this architecture using an https [15] connection. Each event that arrives via https on an established port will be sent to the nodes by the Controller.

In the initialization phase the controller:

- invokes the method `generateKeys` of Dealer to generate all keys
- accepts the acknowledgment of the Dealer component that confirms the correct generation and distribution of all key shares
- searches in the Registry which are the nodes that participate in the architecture and which component offers a service to combine all partial signatures.

In the normal phase the Controller relays the events coming from event sources to all nodes and to the combiner element. The Controller can be a single point of failure. In fact, an attacker may take the control of this component and modify the events before they are sent to all nodes. In order to avoid this weakness we use the principles of redundancy and diversification together with voting technique [16] [17]. In fact, we have more than one controllers to receive the same event. Then, only one event is provided as output after a decision is made through the voting mechanism.

V. Integration with OSSIM and Performance Test

We have used multi-server configuration available in OSSIM in order to integrate the Forensic Storage architecture in this SIEM. In fact, in OSSIM it is possible to configure more servers in order to process different types of events. So, it is possible through a policy to specify which events must be sent to which servers. In our case, we have created a new policy in order to redirect the security events to the forensic storage. Instead of sending all collected events, only the events with a risk value greater than a certain threshold are sent to the forensic storage. In order to support forensic activity, the chain of events related to the generated event will also be stored. It is necessary to store only the necessary events because a huge amount of events per second (EPS) are sent to the SIEM. In the SANS study [18] a benchmark is reported measuring the number of events produced from SIEM solution deployed in a mid-sized organization. In this study the authors show that the average EPS generated in normal conditions are 149.79. When two subnets are attacked the EPS value is 8,118.80. So if we consider all the events generated in a large span of time, e.g. 6 months, we have in normal conditions (without any attack) 2.4 billion events. If we use 300 bytes as the average message size, 6 months of data will require about 670GB of space (if there are no security attacks).

OSSIM provides a risk calculation in order to classify the alert level of different events. So, calculated risk is one of the metrics that provides useful information in the evaluation of a single event, but also provides useful information for the overall security state of the infrastructure. The risk value (0-10) of an event is calculated from the following parameters: RISK of the event = (Asset value * Event Priority * Event Reliability) / 25 [19]

- Asset value (0-5). The assets are: Host, Host Groups, Networks and Network Groups
- Event priority (0-5). The Priority is the importance of the event itself
- Event reliability (0-10). Reliability determines the probability of a security event being effectively related to an attack or not.

The estimated risk value is not assigned statically and can be modified. In fact, each event has an initial given risk value and it can be correlated to other events. The correlation directive is a hierarchical set of correlation rules in XML format. A single rule represents the entry point to a correlation directive. Each incoming event is matched against the rules of an active correlation directive. If the correlation directive reaches its final rule an additional event is created which is then re-injected into the OSSIM server process and/or forensic storage according to the policy established. If the final rule is not reached (either because no matching events arrive or because the correlation directive reaches its timeout) no event is generated from this correlation directive. Correlation rules can modify the event priority and the event reliability in the correlation process. The resulting event generated by a correlation directive will then have the adjusted reliability and priority, which will also lead

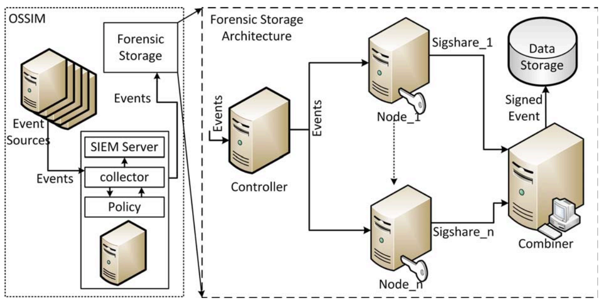


Figure 4: Deployment of OSSIM with the resilient architecture of forensic storage

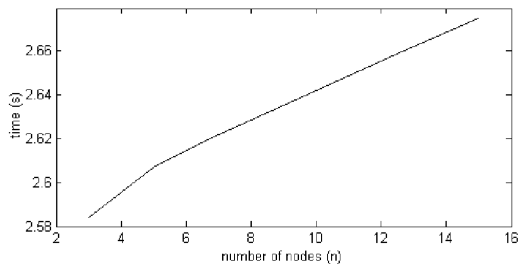


Figure 5: Time to generate 1000 valid complete signatures with k=2 and n variable

to an adjusted risk for this event. In Figure 4 we show the simplified deployment of OSSIM architecture (without redundant components) along with the forensic storage.

We have performed various experimental tests on Forensic Storage in order to measure the performance in terms of time of generation of valid complete signatures. All tests presented here have been done on machines with processor I3-2330M and 2 GB of RAM. Since it is possible in our architecture to change two values, number of shareholders (n) and threshold value (k), we show two graphs of performance: in Figure 5 we show the case in which k is fixed and n is changed; in Figure 6 we show the case in which n is fixed and k is variable. The time reported is the time for generation of valid complete signatures of 1000 events where each event has a size of 1Kbyte. In reality the single event generally has a size of 300 bytes, but in our tests we consider always an event size of 1Kbyte because when an event is to be stored forensically, the chain of events related to this event must also be stored. Also the size of secret key of each shareholder is 1024 bits.

The purpose of these tests is to evaluate the performance increase/decrease for the forensic storage as the values of k and n are changed. We can see that when k is fixed, the decrease of performance, as n increases, is very little (the performance decrease is about 0.5% when a new node is added). In fact, there is not a significant increase in the generation time of a valid complete signature because with the fixed values of threshold, the combiner always needs k

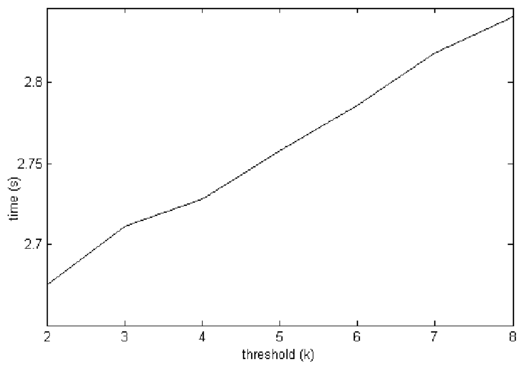
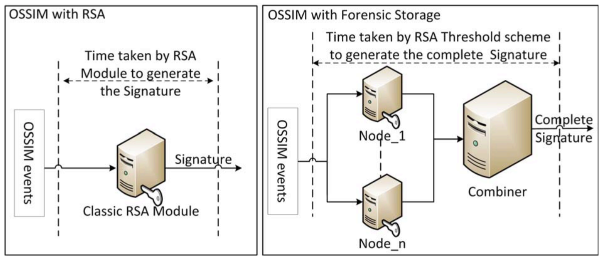


Figure 6: Time to generate 1000 valid complete signatures with n=9 and k variable

number of signature shares to generate a complete signature independently from the total number of nodes. So, the little overhead shown in the Figure 5 is due to the increased number of signature shares that the combiner has to handle. Instead, when we fix n and increase the k value, there is a more significant decrease of performance because the combiner takes more time when combining the signature shares. (the performance decrease is about 1% when the threshold value is increased by one). These experimental results can help us in the choice of the parameters k and n in a real system i.e. if average expected EPS to be signed is known, the maximum required time to generate 1000 signatures can be found and related to these experimental results. Also, the choice of n has an impact on the cost of the whole infrastructure and on the reliability while the choice of k has an impact on the granted level of protection of the secret key.

In the next experimental test we want to compare the performance of our architecture with classic RSA algorithm. This is because the most widely used SIEM systems actually include a forensic storage based on classic RSA algorithm. In particular, we focus on OSSIM which offers a forensic storage only in commercial version. In its technical documentation, it is reported that it has a component called Logger that creates a new signature using RSA algorithm. So, we have written another module that uses Java API to sign the events with RSA algorithm. Then, we have integrated this module in OSSIM as described above. In Figure 7 we show two deployments and the points considered for performance analysis. In Figure 8 we show the comparison of RSA classic algorithm with our Forensic Storage. In particular, classic RSA reaches an average signature rate of 510 signatures per second, while Forensic Storage reaches an average signature rate of 489 signatures per second. So the Forensic Storage has a penalty in terms of signature rate of about 5%. In the next section we show an attack model applied to our Forensic Storage. The Forensic Storage shows a decrease in performance as compared to classic RSA algorithm but



OSSIM with RSA
Time taken by RSA Module to generate the Signature
OSSIM events
Signature
Classic RSA Module
OSSIM with Forensic Storage
Time taken by RSA Threshold scheme to generate the complete Signature
OSSIM events
Node_1
Node_n
Combiner
Complete Signature

Figure 7: Deployment of OSSIM with two different modules integrated

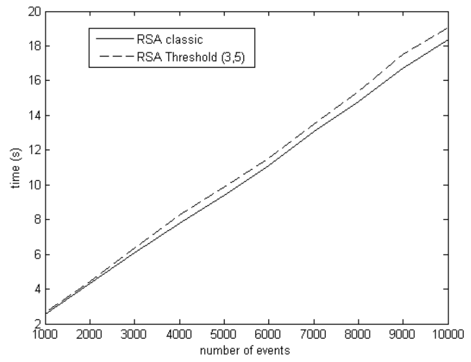


Figure 8: Time to generate a valid complete signature

we show through an experimental test that out architecture works correctly even in the presence of cyber-attacks.

VI. Attack Model and Validation

In this section, we show the case study of a specific critical infrastructure. In fact, as a partner of MASSIF project [20], we are working on a new generation SIEM with advanced features to provide an IT support in a dam critical infrastructure [21]. So, we use this scenario in order to describe the setup of our system and attack model. In particular, in dam infrastructure we have many machines that are employed to gather data coming from wireless sensor network (WSN) or other devices in order to monitor the whole infrastructure. These machines are monitored though agents that send information to the OSSIM server. Also, for test purposes, we have a Forensic Storage integrated in OSSIM with a number of nodes n equal to 5 and a threshold value k equal to 3. In this configuration we can tolerate up to $k-1 = 2$ compromised nodes. The simplified architecture is show in Figure 9 where an attack model is also shown. In particular we suppose that an attacker has corrupted a node (Node_1 in our case) of Forensic Storage using a malicious software hidden in an update of the software pre-installed on this machine. So through the malicious software he has gained remote access to the node and he can modify the signature shares before they are sent to the combiner. Also, another attacker knows the IP address of a node (Node_2 in

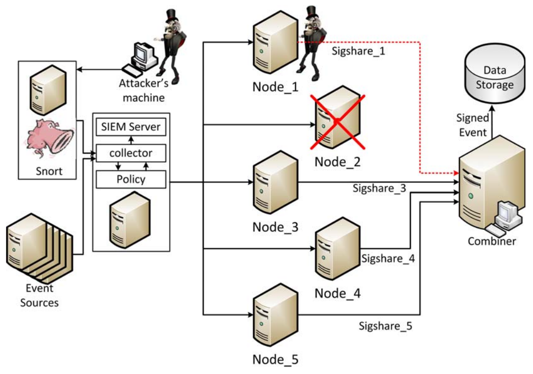
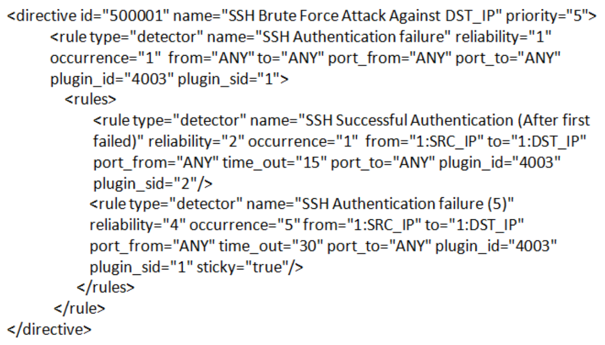


Figure 9: Simplified deployment of OSSIM for our case of study



```
<directive id="500001" name="SSH Brute Force Attack Against DST_IP" priority="5">
      <rule type="detector" name="SSH Authentication failure" reliability="1"
      occurrence="1" from="ANY" to="ANY" port_from="ANY" port_to="ANY"
      plugin_id="4003" plugin_sid="1">
        <rules>
            <rule type="detector" name="SSH Successful Authentication (After first
            failed)" reliability="2" occurrence="1" from="1:SRC_IP" to="1:DST_IP"
            port_from="ANY" time_out="15" port_to="ANY" plugin_id="4003"
            plugin_sid="2"/>
           <rule type="detector" name="SSH Authentication failure (5)"
           reliability="4" occurrence="5" from="1:SRC_IP" to="1:DST_IP"
           port_from="ANY" time_out="30" port_to="ANY" plugin_id="4003"
           plugin_sid="1" sticky="true"/>
         </rules>
      </rule>
</directive>
```

Figure 10: OSSIM directive to recognize a Brute Force Attack

our case) but has not access to the node. So, he performs a DoS attack in order to disrupt the regular service of that node.

Finally, there is another attacker who performs a brute force attack in order to discover the password of one of the machines that control an area of dam infrastructure. Some useful information necessary to understand the experiment is given below:

- a machine (under attack) with the IP address 10.20.189.1 is running Snort IDS. This machine is pre-configured to send the events to the OSSIM SIEM
- a machine with the IP address 10.20.189.2 is used by attacker to perform a SSH brute force attack.

In order to detect an SSH brute force attack, it is necessary to add the following rule to the Snort configuration file: /etc/snort/rules/ssh.rules

```
  alert tcp any any -> $HOME_NET 22
( msg: "Potential SSH Brute Force
Attack"; flags: S; detection_filter:
track by_src, count 5, seconds 20;
classtype:attempted-dos; sid:XXXXX;
rev:4;)
```

This configuration creates an alert that is triggered if the packet is a TCP packet coming from any IP address from any port to our network on port 22 and an IP address has emitted the same packet more than 5 times in 20 seconds. If these conditions are true, an alert is triggered with the message: “Potential SSH Brute Force Attack”. To start the experiment we run Snort on the attacked machine. Then suppose that the attacker tries to connect the previous machine with the SSH, providing a wrong password.

```
cdisarno@attacker:/# ssh root@10.20.189.1
root@10.20.189.1 password: ******
Permission denied, please try again.
```

After a few attempts, new Snort logs are generated and sent to collector of the SIEM in Figure 9. In order to recognize this type of attack an OSSIM directive must be written as shown in Figure 10. The directive has the following meaning: when arrives the first event (authentication failed) the first rule is activated so the first level of correlation is matched. At the second correlation level we will have two possibilities: to get almost immediately an authentication successful event or to get more authentication failed events. If the later option is matched a new event is raised with priority=5 and reliability equal to the sum of reliabilities. Then, the risk value is calculated as described above. If the risk value is greater or equal to the threshold that was previously fixed (according to the established policy), the events are sent to the Forensic Storage. The controller receives the events and sends them to all nodes and combiner. Each node calculates the signature share and sends it to combiner. In our scenario only Node_2 does not send any signature share because we have supposed that this node is off-line. Also the Node_1 sends a wrong signature share. When the combiner receives at least a threshold number of signature shares, it combines them in order to produce a complete signature. We suppose that the first group of k=3 signature shares chosen by the combiner was sent by Node_1, Nod_3, Node_4. So, the complete signature produced is verified with the verification key. This verification process fails because signature share sent by Node_1 is faulty. Then, the combiner explores in depth in order to verify each signature share. So, the node that has sent the wrong signature share is discovered and then the combiner generates a new event with high risk and sends it to the administrator. Finally when new signature shares are received, the combiner chooses a set of k signature shares. So, if the verification process is successful, a new forensic record is created. This record contains the events related to the security breach, the information which nodes show an incorrect behaviour and the signature of the events. An example of forensic record is shown in Table I.

Instead, if classic RSA module was provided in place of our Forensic Storage, a wrong behaviour would be experienced in the attack scenario that has been presented. In fact, if the attack is of the same nature as shown in Node_1 in Figure 9 RSA module provides wrong signature. Instead,

Table I: A forensic record that shows the security event, corrupted nodes and digital signatures of the security event

Message	Corrupted Node	Digital Signature
SSH Brute Force Attack Against 10.20.189.1 …	1 - 2	16063178504335868849 38568479884204520885 87846809…

if the attack is of the nature as shown in Node_2 in Figure 9 the RSA module does not provide any signature at all. In the scenario just presented we have shown how the Forensic Storage keeps working correctly even in the presence of cyber attacks where the RSA module shows an incorrect behaviour under the same conditions.

VII. Conclusion

In this work we have analyzed the limitations of the current SIEM solutions with respect to the forensic storage features. We have designed and developed a novel architecture for the forensic storage of events that overcomes the limitations of classic RSA technique used in existing SIEM systems. Then, we integrated our forensic storage into OSSIM SIEM and we have measured the performance of our solution and compared it to classic RSA algorithm. Also, through an experimental test we have shown that our solution ensures the integrity of data even in the presence of attacks. For future work, we plan to: reduce the penalty performance when compared to the classic RSA algorithm; provide an experimental test in order to measure the penalty performance when forensic storage works in faulty situations.

Acknowledgment

The research leading to these results has received funding from the European Commission within the context of the Seventh Framework Programme (FP7/2007-2013) under Grant Agreement No. 257644 (MAnagement of Security information and events in Service Infrastructures, MASSIF Project).

References

[1] J. Moteff, C. Copeland, J. Fischer, *Critical Infrastructure: What Makes an Infrastructure Critical* Library of Congress Washington DC Congressional Research Service, 2003

[2] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems". Commun. ACM, ACM, 1978, 21, 120-126

[3] Assuria Log Manager, Assuria Ltd. http://www.assuria.com/products-new/assuria-log-manager/features.html Last Accessed 7th June 2012

[4] R. Popovych, "Cryptoanalysis of RSA system of enciphering with public key". Modern Problems of Radio Engineering, Telecommunications and Computer Science, 2004. Proceedings of the International Conference, 2004, 301-302

[5] J. Hoffstein, J. Pipher, and J. H. Silverman, "An Introduction to Mathematical Cryptography". Springer Verlag, (2008)

[6] A. Shamir, and N. Someren, Playing "Hide and Seek" with Stored Keys. Proceedings of the Third International Conference on Financial Cryptography, Springer-Verlag, 1999, 118-124

[7] A. Shamir, "How to share a secret", Commun. ACM, ACM, 1979, 22, 612-613

[8] G. R. Blakley, "Safeguarding cryptographic keys". Managing Requirements Knowledge, International Workshop on, IEEE Computer Society, 1979, 0, 313

[9] V. Shoup, "Practical Threshold Signatures". EUROCRYPT'00, 2000, 207-220

[10] Q. Zhu, and W. W. Hsu, "Fossilized index: the linchpin of trustworthy non-alterable electronic records". Proceedings of the 2005 ACM SIGMOD international conference on Management of data, ACM, 2005, 395-406

[11] K. Pavlou, and R. T. Snodgrass, "Forensic analysis of database tampering". Proceedings of the 2006 ACM SIGMOD international conference on Management of data, ACM, 2006, 109-120

[12] J. Zhou, Y. Wang, and S. Li, "Data Dependence-based Optimistic Data Consistency Maintenance Method". Computer and Information Technology, 2006. CIT '06. The Sixth IEEE International Conference on, 2006, 120

[13] H. Li, and Z. Wu, "Research on Distributed Architecture Based on SOA". Communication Software and Networks, International Conference on, IEEE Computer Society, 2009, 0, 670-674

[14] A. Montresor, "A Reliable Registry for the Jgroup Distributed Object Model". University of Bologna, 1999

[15] E. Rescorla, "HTTP Over TLS". IETF, 2000

[16] A. Saidane, V. Nicomette, and Y. Deswarte, "The Design of a Generic Intrusion-Tolerant Architecture for Web Servers" Dependable and Secure Computing, IEEE Transactions on , vol.6, no.1, pp.45-58, Jan.-March 2009 doi: 10.1109/TDSC.2008.1

[17] Li Wang, Zheng Li, Shangping Ren, and K. Kwiaty, "Optimal voting strategy against rational attackers," Risk and Security of Internet and Systems (CRiSIS), 2011 6th International Conference on , vol., no., pp.1-8, 26-28 Sept. 2011 doi: 10.1109/CRiSIS.2011.6061841

[18] J. Michael Butler, "Benchmarking Security Information Event Management" http://www.sans.org/reading_room/analysts_program/eventMgt_Feb09.pdf

[19] AlienVault Risk Metrics http://www.alienvault.com/wiki/doku.php?id=user_manual:dashboards:risk:risk_metrics

[20] Scenario Requirements http://www.massif-project.eu/sites/default/files/deliverables/D2.1.1_Scenario_Requirements_v1.0_public.pdf

[21] Regan, P.J.: Dams as systems - a holistic approach to dam safety. In: 30th Annual USSD Conference Sacramento, California (2010)