Behavioral Attestation for Web Services (BA4WS)

Masoom Alam
Institute of Management Sciences, Pakistan
mmalam@imsciences.edu.pk

Xinwen Zhang
Samsung Information Systems America, San Jose, CA, USA
xinwen.z@samsung.com

Mohammad Nauman
Institute of Management Sciences, Pakistan
nauman@imsciences.edu.pk

Tamleek Ali
International Islamic University, Pakistan
tamleek@imsciences.edu.pk

ABSTRACT

Service Oriented Architecture with underlying technologies like web services and web service orchestration opens new vistas for integration among business processes operating in heterogeneous environments. However, such dynamic collaborations require a highly secure environment at each respective business partner site. Existing web services standards address the issue of security only on the service provider platform. The partner platforms to which sensitive information is released have till now been neglected. Remote Attestation is a relatively new field of research which enables an authorized party to verify that a trusted environment actually exists on a partner platform. To incorporate this novel concept in to the web services realm, a new mechanism called WS-Attestation has been proposed. This mechanism provides a structural paradigm upon which more fine-grained solutions can be built. In this paper, we present a novel framework, Behavioral Attestation for Web Services, in which XACML is built on top of WS-Attestation in order to enable more flexible remote attestation at the web services level. We propose a new type of XACML policy called XACML behavior policy, which defines the expected behavior of a partner platform. Existing web service standards are used to incorporate remote attestation at the web services level and a prototype is presented, which implements XACML behavior policy using low-level attestation techniques.

Categories and Subject Descriptors

D.4.6 [**OPERATING SYSTEMS**]: Security and Protection—*Access controls; Information flow controls*

General Terms

Security

SWS'08, October 31, 2008, Fairfax, Virginia, USA.

Keywords

Remote attestation, Web Services, behavioral attestation

1. INTRODUCTION

Service Oriented Architecture (SOA) is a software paradigm for achieving loose coupling and addressing interoperability concerns among business partners. It allows business agility and flexibility as needed by today's complex business processes. However, the benefits of SOA cannot be completely delivered until a secure environment exists at each respective business partner site. In this regard, several web service security standards are in practice like WS-Security [12] for addressing confidentiality and integrity of XML documents, WS-Trust [11] for trust negotiation among business partners and XACML [27] for the specification of access policies. These standards focus on security issues relating to a service provider platform only. Security issues arising after the release of a resource to a service requester platform such as its further dissemination and usage control etc., have mostly been neglected. These issues require some tamper proof security mechanisms at the service requester platform, which cannot be realized using software-based solutions alone [14, 20].

In order to provide hardware based security, the *Trusted Computing Group* (TCG) [5] has introduced a new technology, called *Trusted Computing*. In this technology, PCs, consumer electronic devices, PDA's and other mobile devices are equipped with a special hardware chip called *Trusted Platform Module* (TPM). In accordance with other security hardware extensions (cf. [13, 14]) the Trusted Platform Module (TPM) is empowered with cryptographic mechanisms to remotely certify the integrity of the application or system software running on the device. This feature – called *Remote Attestation* – provides assurance that a trusted environment actually exists on a remote platform.

We believe that the advent of trusted computing technology and particularly remote attestation will change the way in which security aspects are envisioned while integrating business processes with each other. Among such efforts [28, 25], WS-Attestation [28] is a prominent approach for the incorporation of remote attestation in to the web services architecture. However, this approach is focused on high-level communication between different entities involved in a remote attestation scenario but does not deal with the problem of specifying the trustworthiness of a service requestor platform at a fine-grained level. TCG defines trust as, "*the*

expectation that a device will behave in a particular manner for a specific purpose" [6]. In the stated goal of TCG, trust is associated with the expected behavior of a platform for a specific purpose, such as web service access or usage of an object.

In this paper, we present a novel approach – Behavioral Attestation for Web Services (BA4WS) – where the expectations from a service requestor platform regarding the usage of an object are specified in a concrete XACML policy called XACML behavior policy. The XACML behavior policy is built on top of WS-Attestation in order to enable a more fine-grained and flexible mechanism for incorporating attestation at the web services level.

Outline: Section 2 presents our motivations with the help of a use case from medical domain and provides the background information about remote attestation and WS-Attestation. We describe the XACML behavior policy in Section 3. The BA4WS framework is detailed in Section 4. Our prototype system is presented in Section 5. We conclude this paper and present our ongoing work in Section 6.

2. BACKGROUND

2.1 Motivating Example

In order to illustrate BA4WS-framework's functionality, we take an example application scenario from medical domain. In our case, doctors, nurses and administrators are given restricted access to resources (or patient's data) at the hospital's main site. Take a service requester[1], attempting to access the online patient records, the following steps are performed during authentication, authorization and attestation:

1. The SR authenticates herself to the hospital site and is assigned a role (e.g., surgeon, nurse, general practitioner etc.)

2. After authentication, the security gateway evaluates the SR's eligibility for the requested resource according to her role and static and dynamic constraints (e.g., access on working days only).

3. In our scenario, the medical data does not necessarily remain within the domain of the hospital. Thus, before releasing any sensitive data, the SP performs attestation of the SR platform to ensure that a trusted environment actually exists on the SR platform.

4. Afterwards, the security gateway attaches a security policy to the requested object. This policy dictates the future usage of the requested object e.g., the number of times a medical record can be accessed etc.

Upon successful authentication, authorization and attestation, the requested information is released to the SR.

2.2 Trusted Computing and Remote Attestation

The Trusted Computing Group has defined an open set of specifications for providing hardware based security. The key component in the trusted computing architecture is the Trusted Platform Module (TPM) chip which provides secure storage for data and cryptographic keys. Each TPM is uniquely identified by its Endorsement Key (EK) that is burned in to it by its manufacturer. According to [22], "the endorsement key is a 2,048-bit RSA public and private key pair, which is created randomly on the chip at manufacture time and cannot be changed. The private key never leaves the chip, while the public key is used for attestation and for encryption of sensitive data sent to the chip". An Attestation Identity Key (AIK) – generated by the TPM and signed by a trusted third party, uniquely identifies the particular users of a target platform.

The TPM is equipped with special purpose registers called Platform Configuration Registers (PCRs). Each PCR can store a 160-bit hash value and at each reboot, the PCRs are reset. Each software component of the platform is mapped to 160-bit hash value that represents its good state. Each new hash is concatenated with an existing hash in the PCR and the new hash is stored in the same PCR. In this way, a small number of PCRs can be used to measure all the components of a platform. Further, a Stored Measurement Log (SML) is used to keep track of the sequence of measurements in the PCRs.

Remote attestation is a mechanism in which a platform presents the PCR measurements within its TPM to a challenger. In a typical remote attestation scenario, a platform's TPM collects the requested PCR values with their indices, signs them with an AIK and returns them to the challenger along with the SML. Using SML and PCR measurement values, it is possible for a challenger to re-compute the PCR values and compare them with their expected values. Based on the comparison, the challenger can make a decision whether the target platform is in a valid state or not. For a comprehensive introduction to remote attestation and trusted computing, we refer the reader to [20].

There are four prominent approaches proposed in the literature for the realization of remote attestation. Configuration-based attestation requires that an SR present the trusted configurations of its platform to an SP. Trusted configurations means that, for example, the SR provides a proof signed by the TPM that a particular security module is installed on the SR platform. Based on these evidences, the SP evaluates the trustworthiness of the SR platform.

Another approach, Integrity Measurement Architecture (IMA) [23], uses binary attestation. In IMA, an SR presents the binary hashes of all its components loaded after booting. An SP verifies the sequence of these binary hashes and concludes as to the trustworthiness of the SR using this information. PRIMA [15] enhances the IMA approach by taking the hash of the required components only. Moreover, it also analyzes the information flow to and from the target application using seLinux [2] policies.

Reporting all system components through binary hashes or revealing all system configurations might give rise to severe efficiency problems and security threats, respectively. To overcome these limitations, property-based attestation [21, 18] proposes to map a set of related system configurations to some meaningful properties. For example, the property, "Multi Level Security (MLS) enabled" can be mapped to system configurations such as the support of MLS in SELinux policies. In this way, the configurations of a target platform are not disclosed to an SP [8].

Each of these existing low-level remote attestation tech-

[1]Hereafter, we refer to the Service Provider as SP, the Service Requester as SR and the Validation Service as VS.

niques is useful in some scenarios but becomes infeasible in others. For example, revealing all system configurations in a highly critical environment may cause a denial of service attack [24]. Moreover, none of these approaches specify an explicit benchmark with which the trustworthiness of an SR platform can be compared.

In our previous work, we have presented a novel approach – Model-based Behavioral Attestation (MBA) [8] – which combines these existing low-level techniques and uses them in a supporting manner so that the weaknesses of one can be addressed by another. BA4WS is a step forward in this direction. It uses behavioral attestation at the web services level allowing for the incorporation of low level attestation techniques in a supporting manner using existing web services security standards.

2.3 WS-Attestation

In order to leverage the concept of remote attestation at the web services level, a new mechanism – called WS-Attestation – has been proposed. The proposal extends WS-Trust [11] to define how an SP and SR interact with each other for attestation purposes. The main entities involved in WS-Attestation are a Service Provider (SP) – which needs to attest the integrity of a platform, a Service Requester (SR) – the target platform which reports its integrity and a Validation Service (VS) – the trusted third-party which can perform attestation on behalf of the SP. The main idea behind the introduction of a VS is to address privacy issues involved in revealing system configurations via PCRs and SML. After attestation, a VS advocates about certain properties of an SR like `PlatformIntegrity = true` etc.

In order to support flexible communication between SP, SR and VS, WS-Attestation proposes three different architectural models (cf. Fig 1). In the *Pushed Model* (cf. Fig 1a), an SR takes an attestation credential from a VS and presents it to the SP. In the *Pulled Model* (cf. Fig 1b), the SR sends its PCRs and SML along with the service request. The SP then requests the VS for the verification of different properties of the SR using its sent PCRs. Finally, in the *Delegated Model* (cf. Fig 1c), the SP requests the VS to perform attestation on its behalf. Depending on the underlying attestation scenario, any combination of these three models can be utilized.

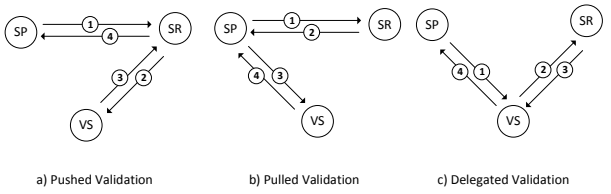

Figure 1: WS-Attestation Models

However, WS-Attestation provides only a structural paradigm on which more fine-grained attestation frameworks can be developed. More specifically, in WS-Attestation, the specification of an explicit criteria for the trustworthiness of an SR platform is left unspecified. In this paper, we present a novel approach where an XACML policy, called behavior policy, is used as a benchmark for measuring the integrity status of an SR platform. The contributions in this paper are threefold: Firstly, we define a mechanism for explicitly specifying the criteria for the trustworthiness of an SR platform. Secondly, we create an abstraction layer at the web services level through which existing low-level attestation techniques can be combined to specify the expected behavior of an SR platform. Finally, our approach enables a predictive analysis of the behavior of the SR platform. In the following sections, our approach is presented in detail.

3. XACML BEHAVIOR POLICY

XACML is an OASIS standard for the specification of complex access control policies. It is composed of three main components: 1) an XML based language for the definition of restricted and complex access control scenarios, 2) a request/response protocol which provides insulation from the underlying environment and 3) an abstraction layer in the form of a data flow model which provides distributed access control [16] across different platforms and any customization for a particular environment.

An XACML policy can be formalized as follows:

DEFINITION 1. *An XACML policy p is a 5-tuple $(S, O, R, 2^C, 2^{O_b})$, where S is a set of subjects, O is a set of objects, R is a set of rights, C is a set of permission predicates which can be evaluated to either $true$ or $false$ and O_b is a set of obligations.*

An XACML policy specifies the *conditions* C under which *subjects* S are allowed to exercise *rights* R on *objects* O under *obligations* O_b.

In our scenario (cf. Section 2.1), an SP (i.e. the hospital) attaches an XACML policy – called *usage policy* – to the medical data before dispatching it. The attached XACML policy dictates the future usage of the medical data (e.g., the number of times the medical data can be read). However, before releasing medical data to a private clinic or laboratory, the SP verifies that a trusted environment actually exists on the SR platform. In other words, the SP explicitly specifies a *criteria for the trustworthiness* of the SR platform regarding the enforcement of its usage policy attached to the medical object. Based on this, we define trust as follows:

DEFINITION 2. *Trust is the expectation that a service requester SR identified by (s, o, r), where $s \in S$, $o \in O$ and $r \in R$, will enforce the conditions C of a particular XACML usage policy p as expected by a service provider SP.*

These expectations of the SP are formalized in a concrete XACML policy called *Behavior Policy*. Contrary to a normal XACML policy which specifies the conditions under which a particular resource is accessible, an XACML behavior policy formalizes the expectations of an SP from an SR platform. It defines a benchmark through which an SP can predict that its usage policies will be enforced correctly by the SR. In other words, an XACML behavior policy specifies a set of requirements from an SR platform which are necessary for the correct handling of the protected object.

The key elements in an XACML behavior policy are `<Condition>`s and their corresponding `<Target>`s. The `<Condition>`s in a behavior policy are constructed in a manner such that the fulfillment of these `<Condition>`s ensures that the usage policy attached to the protected object will indeed be enforced. Thus, the `<Condition>`s in the behavior policy are heavily influenced by the usage policy. For example,

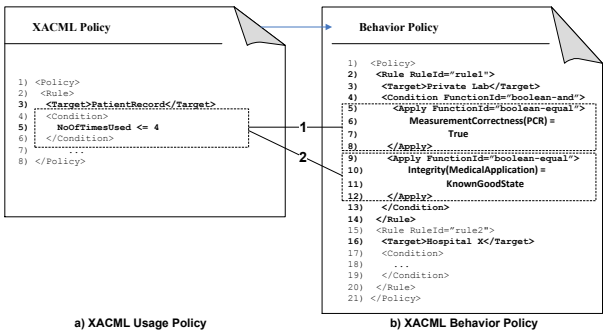

Figure 2: XACML Usage and Behavior Policies

`<Condition>` in Figure 2a (line 5) specifies that *"PatientRecord is accessible at most 4 times"* at the SR platform. The corresponding behavior policy (cf. Fig 2b) contains `<Condition>`s that specify the expectations of the SP. For instance, the SP expects that this usage `<Condition>` can only be fulfilled in a trustworthy way, if 1) the integrity of all the components loaded after booting on the SR platform is reported correctly and 2) the integrity of the medical application itself is in a known good state. Note that, in Figure 2, the policies are shown in a high-level syntax. For details, please refer to Section 4 and 5.

The `<Target>` of each `<Rule>` of an XACML behavior policy identifies a particular set of SR platforms with similar expectations. For instance, in Figure 2b, the expectation from `Private Lab` is specified in the first `<Rule>` (line 3) whereas the second rule (line 16) specifies the expectations from `Hospital X`. In this way, the SP can explicitly specify individual domains and corresponding expectations from platforms belonging to these domains. We define an XACML behavior policy as follows:

DEFINITION 3. *An XACML behavior policy b is a 4-tuple* $(S, O, R, 2^{C_b})$ *where S is a set of subjects, O is a set of objects, R is a set of rights and* C_b *is a set of behavior conditions specifying the expectations of an SP from an SR platform regarding the enforcement of an XACML usage policy p. We define the Behavior Association function BA as a partial function which maps the conditions in an XACML policy to a set of behavior conditions in the behavior policy b. Formally:*

$$BA : C \leftharpoonup 2^{C_b}$$

where $\leftharpoonup$ *represents the behavior association relation.*

Note that, for the sake of simplicity, we omit the behavior of obligations in an XACML usage policy without lack of generality.

The mappings between the `<Condition>`s in an XACML usage policy p to a set of `<Condition>`s in the behavior policy is an important step in the realization of BA4WS framework. Depending on the underlying computing environment, different mappings can be defined for a single XACML usage policy. For instance, for a relatively closed environment such as client machines within the hospital, the administrator can define a relaxed set of mappings. These mappings may require only property-based attestation. For a more open environment, such as collaborating labs outside the hospital's domain, a more restricted set of mappings might be defined. Due to space limitations, mappings are not discussed in this paper.

```
1)  <PolicySet>…
2)    <Policy RuleCombiningAlgId="deny-overrrides"> ...
3)     <Rule Effect="Permit" RuleId="ba4ws:rule1">
4)       <Target>PrivateLab</Target>
5)       <Condition FunctionId="boolean-and">
6)        <Apply FunctionId="boolean-equal">
7)          <Apply FunctionId="urn:vs1.ims.edu:measurementcorrectness">
8)            <AttributeSelector RequestContextPath="/request/PCRs/PCR" />
9)          </Apply>
10)         <AttributeValue DataType="boolean">
11)           true
12)         </AttributeValue>
13)       </Apply>                                          } Medical
14)       <Apply FunctionId="boolean-equal">                  Application
15)         <Apply FunctionId="urn:vs1.ims.edu:softwareintegrity">  Integrity
16)           <AttributeSelector
17)              RequestContextPath="/request/SML/med_app_path" />
18)         </Apply>
19)        <AttributeValue DataType="boolean">
20)          true
21)         </AttributeValue>
22)       </Apply>
23)      <Condition>
24)     </Rule>
25)     <Rule Effect="Permit" RuleId="ba4ws:rule2">
26)      <Target>...</Target>
27)      <Condition FunctionId="boolean-equal">
28)         <SubjectAttributeDesignator
29)           DataType="X.509Certificate"                   } MLS Enabled
30)           AttributeId="urn:vs1.ims.edu:mls-enabled" />
31)       <AttributeValue>true</AttributeValue>
32)      </Condition>
33)     </Rule>
34)   </Policy>
35) </PolicySet>
```

Figure 4: A Detailed XACML Behavior Policy for Fig 2b

Using the XACML behavior policy, an SP can specify, at a fine-grained level, which `<Condition>`s are expected from which `<Target>`s. Thus, the XACML behavior policy can be used as a benchmark for measuring the trustworthiness of an SR platform. Below, we discuss how an attestation architecture proposed by WS-Attestation can be incorporated within the normal data flow model of XACML using the proposed XACML behavior policy.

4. BEHAVIORAL ATTESTATION FOR WEB SERVICES

WS-Attestation proposes three models for remote attestation. In this section, we illustrate how the BA4WS framework can be applied to these three models.

According to the normal data flow model of XACML, a `Private Lab` (the SR) makes a request (cf. Fig 3 – Step 1) for a `PatientRecord` (the resource), which is intercepted by the Policy Enforcement Point (PEP) at the hospital site (the SP). The PEP authenticates the SR and transforms the request in XACML form, which is then forwarded (Step 2) to the Policy Decision Point (PDP) through the Context Handler (CH). The PDP retrieves the XACML policy corresponding to the access request of the SR (Step 3). This XACML policy is called *server-side* XACML policy.

A server-side XACML policy is composed of access constraints and an optional pointer to an XACML behavior policy. The access constraints in the server-side policy evaluate the SR's eligibility for the requested resource according to static and dynamic constraints.

The server-side access policy references an XACML behavior policy using the `<PolicySetIdReference>` element of XACML. The separation between a server-side policy and a behavior policy enables an SP to specify a set of platforms for which attestation is mandatory. This classification also helps to identify sub-domains within an SR platform from which attestation is required.

An XACML behavior policy can be evaluated according to the three models of WS-Attestation as follows:

Delegated Model:.

In the Delegated Model, the SP delegates the verification of different characteristics of an SR platform to different

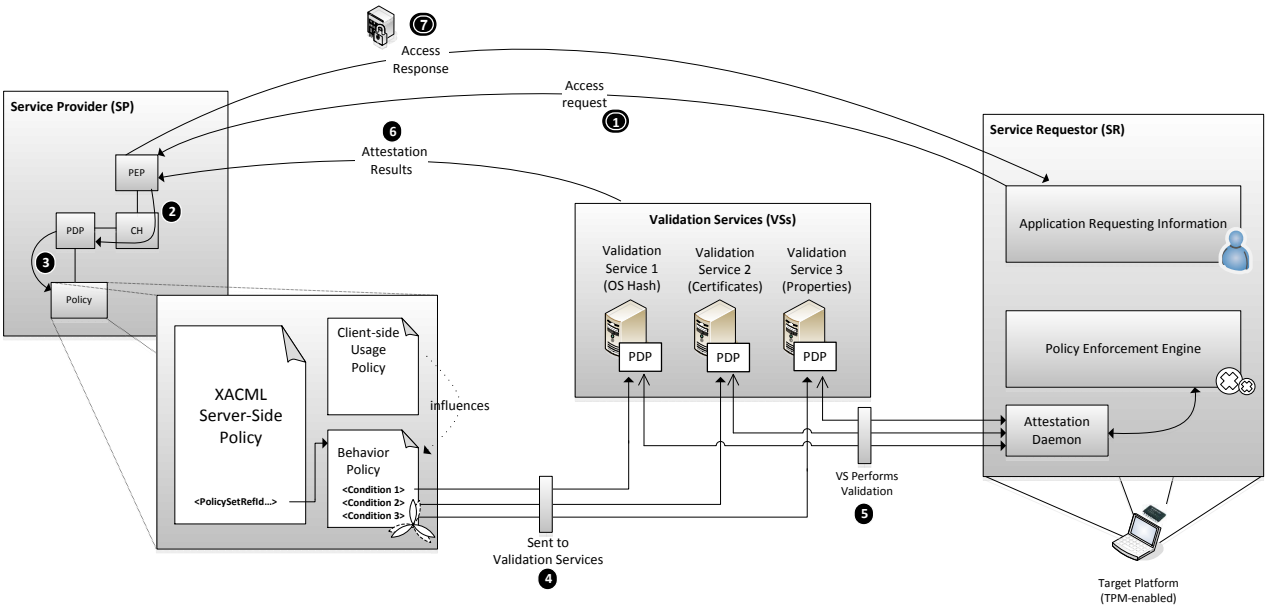

Figure 3: BA4WS Platform Architecture for the Delegated Model

VSs. This makes the Delegated Model, the most important model for incorporating attestation in heterogeneous and distributed environments. On the other hand, as a standard language, different conditions of XACML can be evaluated by different VSs. Thus, XACML is a highly suitable choice for different attestation models of WS-Attestation.

We have leveraged this flexibility of XACML to distribute different XACML conditions across different VSs of WS-Attestation. For example, in Figure 4, the first `<Condition>` is evaluated by the VS at the URN `urn:vs1.ims.edu` (lines 5–23). The `<Condition>` specifies that 1) the PCR values sent by an SR are trusted and 2) that the integrity of the medical application is in a good known state. The extended XACML functions, MeasurementCorrectness and SoftwareIntegrity, take the PCRs and the path of the medical application in the SML as input and return either true or false.

The second `<Condition>` (lines 27–32) specifies that Multi-Level Security is enabled at the SR platform. Implementation details for these conditions are presented in Section 5.

The SP (the hospital) constructs an XACML policy with the help of `<Condition>`s and `<Target>`s in the behavior policy and sends it to the VSs. The VSs then attest the SR (Step 5) and return the result of the attestation to the SP (Step 6). Upon, successful validation of the SR platform by the designated VSs, the combined result of the XACML behavior policy decides whether the resource should be released or not (Step 7).

The major advantage of using the delegated model in this scenario is that different validation services specialized for different purposes and domains can be employed. For example, an SP can contact two validation services: One specialized in known good configurations of a particular operating system and another specialized in a specific domain, such as hospitals. The attestation results of these two can be combined through XACML rule combining algorithms to provide the same flexibility as in XACML policy evaluation.

Pulled Model:.

The only difference between the pulled model and the delegated model is that, in the pulled model, the SR sends its PCRs and SML along with the web service request. Such security requirements can be explicitly specified in WS-Policy [26]. Afterwards, the SP sends the XACML behavior conditions along with the SML and PCRs to the VS. In this way, the SR does not have any direct contact with the VS. For example, in Figure 4, the first `<Condition>` (lines 5–23) in the behavior policy can be sent to the VS along with the PCRs and SML sent by the SR.

Pushed Model:.

In the pushed model, the SR is requested to produce the credentials attested by some VS. It is the responsibility of the SR platform to contact the VS and get the required credential.

We came to the conclusion that the specification of trustworthiness requirements from an SR platform using XACML allows an SP to specify its expectations in a concrete policy. On the other hand, an XACML behavior policy can combine different attestation models of WS-Attestation. This means that the specification of the behavior policy is independent of the underlying architectural models of WS-Attestation. For instance, the behavior policy given in Figure 4 combines the pulled model (first rule) and the pushed model (second rule) in a single XACML behavior policy. Moreover, a combination of low-level attestation techniques can be employed through a single XACML behavior policy.

In the following, we share our experiences regarding the implementation of the delegated model of WS-Attestation using XACML behavior policy.

5. IMPLEMENTATION

For the demonstration of our BA4WS framework, we have developed a prototype of the delegated model discussed in Section 4. In our prototype, the SP sends an XACML behavior policy to a VS. We have constructed this VS specifically

for performing attestation of a Linux Fedora Core system. The SP asks this VS to perform validation on its behalf through the behavior policy. Here, we do not discuss the mechanism of communication between the involved parties. Instead, we discuss the details of XACML function extension for incorporating attestation at the web services level and the means of integrity measurement. Since we build the whole framework on existing web services standards, the interested reader may refer to [7, 28] for a detailed discussion regarding the mechanism of communications. In the following, we first present the details of XACML function extensions. Afterwards, a brief note on the evaluation of our prototype implementation is given.

5.1 XACML Function Extension

The Sun's XACML implementation [3] provides a flexible way for extending XACML through new functions. We have utilized this flexibility for creating two functions related to remote attestation. 1) MeasurementCorrectness and 2) SoftwareIntegrity. The behavior policy given in Figure 4 utilizes these two functions for the purpose of attestation (lines 7, 15).

The *MeasurementCorrectness* function requests the SR platform to return the Stored Measurement Log (SML) along with the current value of its tenth PCR, signed by the TPM. We have used Trusted Java [4] libraries for the implementation of the daemon listening on the client for such requests. Trusted Java provides a Java software stack (jTSS) for the purpose of communicating with the TPM through the TPM driver. The SR platform was a Dell Optiplex 745 Desktop equipped with a TPM. We created[2] an Attestation Identity Key (labeled '`bacaik`') through the jTSS and registered it with a PrivacyCA provided by IAIK [1] using the Endorsement Key certificate created earlier. This `bacaik` was used for the purpose of signing the PCRs.

The VS asks the SR for attestation along with a nonce (for ensuring freshness of the values returned). Upon receiving an attestation request from the VS, our daemon running on the SR reads the SML from `/sys/kernel/security`. It also asks the TPM to provide a *quote* over the value of it's tenth PCR. The TPM attests the values of its PCRs through the quote. The TPM takes the index of the PCR to quote, a nonce for assuring the freshness of the quote and an AIK for signing the value with. It appends the current PCR value with the nonce and digitally signs the result. This resulting structure is called a TPM quote over a PCR value.

Upon receiving this quote from the SR, the MeasurementCorrectness function first verifies the digital signature on the *received* quote to verify that it has indeed been signed by the TPM of the SR. Afterwards, the *expected* value of the PCR is computed using the SML.

If the expected values of the PCR and those returned by the SR match and if the nonce is returned in the quote, the VS can conclude that: 1) the SML sent by the client is indeed correct as it is signed by a valid TPM and 2) the values of the SML are fresh as the TPM has also signed the nonce while performing the quote.

For providing assurance of the integrity of our medical application, we used the *SoftwareIntegrity* function. This function verifies the hash of the application provided in the SML. Since the correctness and freshness of the software hash is assured by the TPM signature, the only requirement for establishing the integrity of the software is that the hash be of a known good value. For this purpose, we created a small database of known good hashes of two different versions of our application. If the hash provided by the client corresponds to a known good value already present in the database, the VS can certify to the integrity of the application. The requirement that the usage history of the medical application is accessible only to the medical application can be fulfilled through mandatory access control mechanisms such as SELinux Loadable Policy Modules (LPM) [17]. For details, please refer to [10, 9]. Figure 5 shows the working of the prototype in detail.

[2]Note that in the following description of our implementation, we omit some of the minor details of operations performed by the TPM for the sake of clarity.

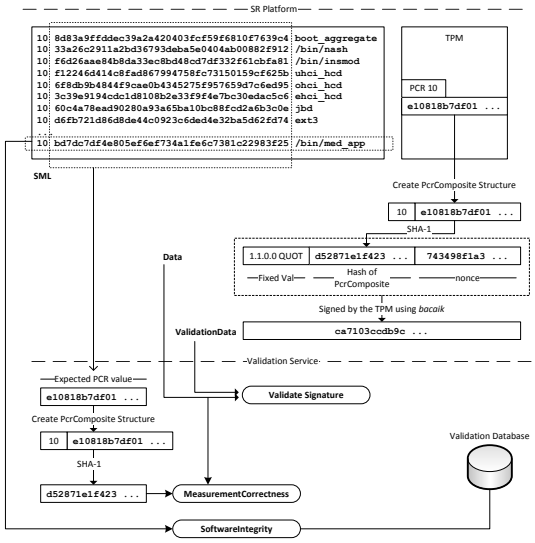

Figure 5: BA4WS Prototype Implementation

5.2 Evaluation

In BA4WS, the SP interrupts the normal process of access request and response and performs attestation in the middle. We understand that this causes a time delay but we believe that for security-concerned web services, establishing the integrity of a SR platform is essential and is worth the time delay. Moreover, the results of the VSs can be cached by the SP to reduce this problem. Here, we describe the time taken during the actual validation process in our scenario.

The time taken from the start of the validation request to the validation response was 4991 ms. This involves the web service invocation, integrity measurement and reporting. The time taken during the integrity measurement and signing on the SR platform was 3925 ms. This time was expected because the TPM performs asymmetric encryption which is computationally expensive. The time taken for the re-computation and comparison of the SML with the PCR values was 416 ms. Overall, we believe that with caching support, this approach is very feasible, especially considering the added advantages of knowing in advance that the platform to which a resource is being released is in a healthy state.

6. CONCLUSION

In this paper, we have presented a novel framework – BA4WS – which enables finer-granular attestation at the

web services level. Our framework incorporates XACML on top of WS-Attestation. We have introduced a new type of XACML policy called behavior policy. This policy allows an administrator to specify her expectations from an SR platform regarding the enforcement of her usage policies. Thus the SP can verify, before releasing a protected object, that its usage policies will indeed be enforced as expected.

Currently, the mappings between usage conditions and behavior conditions are not automatic. This feature requires a comprehensive transformation framework which is part of our ongoing work. Further, the specification of a more flexible usage control model, such as UCON [19] and the behavior of XACML obligations will also be explored in our future work.

7. REFERENCES

[1] IAIK: Institute for Applied Information Processing and Communications, Graz University of Technology. http://www. iaik. tugraz. at/.

[2] Security-Enhanced Linux (SELinux). http://www. nsa. gov/ selinux/.

[3] Sun's XACML Implementation. http://sunxacml. sourceforge. net.

[4] Trusted Computing for the Java(tm) Platform. available at,. http://trustedjava. sourceforge. net/.

[5] Trusted Computing Group (TCG). https://www.trustedcomputinggroup.org/.

[6] TCG Specification Architecture Overview v1.2, page 11-12. Technical report, Trusted Computing Group, April 2004.

[7] SAML 2.0 profile of XACML v2.0. Technical report, OASIS, February 2005.

[8] M. Alam, X. Zhang, M. Nauman, T. Ali, and J.P. Seifert. Model-based Behavioral Attestation. In *SACMAT '08: Proceedings of the thirteenth ACM symposium on Access control models and technologies.*, New York, NY, USA, 2008. ACM Press.

[9] Masoom Alam, Qi Li, Xinwen Zhang, and Jean-Pierre Seifert. Usage control platformization via trustworthy selinux. In *ASIACCS'08: Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security*, 2008.

[10] Masoom Alam, Jean-Pierre Seifert, and Xinwen Zhang. A model-driven framework for trusted computing based systems. In *EDOC '07: Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference*, page 75, Washington, DC, USA, 2007. IEEE Computer Society.

[11] S. Anderson, J. Bohren, T. Boubez, et al. Web Services Trust Language (WS-Trust). *Public draft release, Actional Corporation, BEA Systems, Computer Associates International, International Business Machines Corporation, Layer*, 7.

[12] B. Atkinson, G. Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, J. Klein, B. LaMacchia, P. Leach, J. Manferdelli, H. Maruyama, et al. Web Services Security (WS-Security). *Version*, 1.

[13] Advanced Micro Devices. *AMD Secure Virtual Machine Architecture Reference Manual*. AMD, 2005.

[14] David Grawrock. *The Intel Safer Computing Initiative Building Blocks for Trusted Computing*. Intel Press, http://www.intel.com/intelpress/sum_secc.htm, 2005.

[15] Trent Jaeger, Reiner Sailer, and Umesh Shankar. PRIMA: Policy-Reduced Integrity Measurement Architecture. In *SACMAT '06: Proceedings of the eleventh ACM symposium on Access control models and technologies*, pages 19–28, New York, NY, USA, 2006. ACM Press.

[16] Markus Lorch, Seth Proctor, Rebekah Lepro, Dennis Kafura, and Sumit Shah. First experiences using xacml for access control in distributed systems. In *XMLSEC '03: Proceedings of the 2003 ACM workshop on XML security*, pages 25–37, New York, NY, USA, 2003. ACM.

[17] F. Mayer, K. MacMillan, and D. Caplan. *SELinux by Example: Using Security Enhanced Linux*. Prentice Hall, 2006.

[18] Aarthi Nagarajan, Vijay Varadharajan, and Michael Hitchens. Trust Management for Trusted Computing Platforms in Web Services. In *STC 07: The Second ACM Workshop on Scalable Trusted Computing, under ACM CCS 07*, Virginia, USA, 2007. ACM.

[19] Jaehong Park and Ravi Sandhu. Towards Usage Control Models: Beyond Traditional Access Control. In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 57–64, New York, NY, USA, 2002. ACM Press.

[20] Siani Pearson. *Trusted Computing Platforms: TCPA Technology in Context*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2002.

[21] Ahmad-Reza Sadeghi and Christian Stüble. Property-based Attestation for Computing Platforms: Caring about Properties, not Mechanisms. In *NSPW '04: Proceedings of the 2004 Workshop on New Security Paradigms*, pages 67–77, New York, NY, USA, 2004. ACM Press.

[22] David Safford, Jeff Kravitz, and Leendert van Doorn. Take Control of TCPA. *Linux J.*, 2003(112):2, 2003.

[23] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, pages 16–16, Berkeley, CA, USA, 2004. USENIX Association.

[24] Elaine Shi, Adrian Perrig, and Leendert Van Doorn. BIND: A Fine-Grained Attestation Service for Secure Distributed Systems. In *SP '05: Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 154–168, Washington, DC, USA, 2005. IEEE Computer Society.

[25] Z. Song, S. Lee, and R. Masuoka. Trusted web service. *The Second Workshop on Advances in Trusted Computing (WATCŠ06 Fall)*, 2006.

[26] Web Services Policy 1.2. http://www. w3. org/ Submission/ WS-Policy/.

[27] XACML 2.0 Specification Set. Available at: http://www.oasis-open.org/ committees/ tc_home.php? wg_abbrev=xacml.

[28] S. Yoshihama, T. Ebringer, M. Nakamura, S. Munetoh, T. Mishina, and H. Maruyama. WS-Attestation: Enabling Trusted Computing on Web Services. *Test and Analysis of Web Services*, pages 441–469, 2007.