HARDWARE-ASSISTED CREDENTIAL VALIDATION

BACKGROUND

The explosion in web-based services has led to an increased need for security, especially in financial transactions. An interaction between a vendor and a financial institution across a network offers opportunities for malicious interference from hackers, such as 'spoofing' or outright identity theft, as examples.

When a user purchases a product from a vendor, the user sends sensitive financial information to the vendor. The vendor then validates the financial information with the financial institution and accepts the user's order, as an example. During this transaction, the user's financial information may be transmitted through several network links. Hackers may intercept this information, or a hacker may assume an involved entity's identity and either misappropriate the information or attempt to enter some of the other involved entity's sites. These are just examples of some problems that may occur during a transaction with which most users would be familiar, but demonstrate the problems inherent in such a transaction.

Typically, however, there are many transactions or transfers of information that may occur across the Internet or similar networks that do not involve consumers' information directly. Financial institutions may transfer information back and forth, producers and their suppliers may transfer order information, purchase order specifics, etc. All of these transactions need to be secure, or these entities become vulnerable to attack.

In addition to the growing number of transactions involving confidential information, there is a movement towards interoperability. Currently, there are several

different kinds of devices that use the Internet to communicate. True interoperability would allow these different platforms to access services, objects and servers in a platform-independent manner. For example, the Simple Object Access Protocol (SOAP) is a protocol that acts as the glue between heterogeneous software components. It offers a mechanism for bridging competing technologies in a standard way. The main goal of SOAP is to facilitate interoperability. However, the increase in interoperability may lead to even easier spoofing and misappropriation of partners' identities in network transactions.

In response to these types of problems, many entities such as vendors and banks have instituted security procedures. For example, the HyperText Transfer Protocol (HTTP) has authentication measures such as the secure socket layer (SSL) which can be used by most web browsers to employ a key to encrypt and decrypt information transmitted over the Internet (or any Network) between partners in a secure transaction. Other examples include the use of symmetric keys, asymmetric keys, session keys, tokens or other types of security credentials.

An initiating partner sends its security credentials to a receiving partner. The receiving partner then checks any incoming messages with the security credentials to ensure that each message it receives from the sending partner has credentials that match. Credentials may include a certificate, a token or a signature. Currently, these credentials are implemented and verified in software. This is not very efficient and may still be subjected to manipulation. For example, keys stored in a file system are typically managed by software applications. During the processing of the software application, the keys may

be exposed. Similarly, if the keys are stored in a database, they may be exposed after they are stored.

BRIEF DESCRIPTION OF THE DRAWINGS

Embodiments of the invention may be best understood by reading the disclosure with reference to the drawings, wherein:

Figure 1 shows a network over which a transaction may occur.

Figure 2 shows an embodiment of a system with credential validation.

Figure 3 shows an embodiment of a credential validation module.

Figure 4 shows a flowchart of an embodiment of a method to validate security credentials.

Figure 5 shows a flowchart of an embodiment of a method to validate digital signatures using the Simple Object Access Protocol.

DETAILED DESCRIPTION OF THE EMBODIMENTS

Figure 1 shows an environment in which services are provided and transactions executed between two partners in an electronic commerce environment. In this example, Partner A 10 has confidential information related to transactions occurring on its web site, such as an on-line ordering system. Partner B 12 is a supplier of Partner A from which Partner A wishes to purchase parts in order to manufacture goods to fill the orders received from its site. Partner A will transmit a purchase order to Partner B.

The purchase order may have sensitive information in it such as the financial institutions involved, legal documents, credit information, competitive pricing, account numbers and routing information that allows Partner B to confirm the purchase order. Competitive information, such as the number of units and pricing of a specific part could

also be transmitted that would allow competitors to gain unfair advantage of either Partner A or Partner B.

Currently, the data transmitted from Partner A to Partner B would more than likely be transmitted across the Internet. However, while the Internet will be used here as an example, it is not intended that application or scope of the invention be limited in any way. The network could be any distributed network in which data transmitted from one endpoint to another may make intervening hops. Similarly, the transmitted data could take many forms other than packets in an Internet Protocol (IP) network. For that reason, the discrete pieces of data transmitted will be referred to as datagrams.

As shown in Figure 1, Partner A's transmission makes 5 intervening hops between endpoint 10 and endpoint 12. For example, a hop could include a hop to an intermediate server at a financial organization, a desktop in a credit services bureau, or some third-party supplier to Partner A. Any one of these hops could be a point of attack for a hacker to assume the other partner's identity. For example, an attacker could assume Partner B's identity and garner sensitive financial data on Partner A that could be manipulated. Alternatively, an attacker could assume Partner A's identity and garner information about Partner B, or even steal the parts being order by causing Partner B, who assumes that Partner A is really Partner A, to ship parts to the attacker instead of the actual Partner A.

Current implementations of security protocols institute software processes at either end to confirm that the other partner is really the other partner. Software is inherently vulnerable to being 'fooled' or spoofed, as well as requiring often unacceptable system overhead to process the security credentials of the other party. If an attacker knows how a particular software package used for security validates credentials, that hacker could figure

out ways to steal or recreate security credentials. Figure 2 shows an embodiment of a system in which security credentials are validated by the hardware, rather than by a software process.

The system 20 of Figure 2 includes a security credential validation module 30. In this particular system embodiment, the system includes a credential validation module 22, a memory 24, a parser 26, and a port 28. This is just one example of a system configuration and the additional components are optional. Indeed, in many systems the credential generator 22 would reside separate from the credential validation module 30.

Using this system embodiment, however, it is possible to see how a networked device can employ security measures to mitigate the likelihood of attacks. For outgoing data transmissions, the credential generator 22 generates security credentials. As used here, security credentials include public-private encryption key pairs, tokens, digital signatures or any other type of credential that can be used to verify the identify of the transmitting entity. The memory 24 may store credentials generated to allow the system 20 to include the credentials in outgoing data transmissions. These data transmissions would be sent out through port 28.

Port 28 also allows the system 20 to receive datagrams. The security credentials in these datagrams would then be verified and validated by the credential validation module 30. For example, a transmission may include a public key from a partner. The security validation module would then operate on the public key to ensure that the public key transmitted with the data matches the public key previously received from that partner. This allows the receiving party to determine that it is dealing with the right partner, not an impostor.

As part of receiving the datagram through port 28, a parser may extract the security credentials from the datagram payload data. As used here, payload data refers to the data contained inside the datagram that does not include information in the datagram necessary for transmission and management of the datagram, such as the header. The parser may not be required, however, as the credentials may be received in such a format that they do not require extraction, or the credential validation module may have the capability of extracting the credentials without need for a parser.

This is shown in more detail in an embodiment of the invention shown in Figure 3. In this embodiment either the parser 26 provides an arithmetic logic unit (ALU) 36 with specific information about the security credentials, or the ALU receives it directly, as mentioned above. In this embodiment, the security credentials have at least three parts. The first part is the actual credential. The second is the method that was used to generate the credential. This may include an arithmetic algorithm executed to obtain the credential. The third part is the value of the credential.

In this particular embodiment, the ALU 36 uses the digest method provided to recalculate the credential value by operating on the credential. The comparator 38 then compares the recalculated result with the original value and determines if the credential is valid. The use of an ALU and a comparator are merely examples of hardware components that could perform this process and is not intended to limit the scope of the possible embodiments of the invention in any way.

In this manner, the security credentials are validated in the hardware of the system, leaving them a little less vulnerable than software validation, and speeding the process of validation by moving it into hardware.

An embodiment of a process of performing such a validation is shown in Figure 4. An incoming datagram with associated security credentials is optionally parsed at 40. At 42, the actual security credentials are received. As mentioned previously, the validation module may not require the credentials to be parsed. At 44, the digest of the security credentials is recalculated. A digest or a 'hash' is a representation of the security credentials resulting from series of operations performed on it, where the series of operations are the digesting method or algorithm mentioned above.

At 46, the recalculated representation or digest is compared to the provided representation or digest. If the two values compare, the security credentials are valid and the data can be trusted as being from where it appears to originate. Also, if the security credentials are valid, they may be stored at 48. Having a stored credential to be checked against incoming credentials allows the process to shorten to just a comparison of the previous received credential and a recently received credential to determine if the data is still trustworthy. However, as noted above, storing the credential is optional. Storing the credentials in hardware, rather than in a file system or database, may increase security.

A specific application of this type of credential validation may be discussed in terms of the Simple Object Access Protocol (SOAP). The payload of a SOAP message includes several elements that represent security credentials. An embodiment of a process to validate security credentials in hardware for a SOAP payload is shown in Figure 5.

At 50, the SOAP payload is parsed, producing three elements of the SOAP signature: SignatureMethod, SignedInfo, and SignatureValue. SignedInfo 54 is the information that is actually being signed by the digital signature. This is typically canonicalized, meaning transformed into a well-defined, standard format, with a method

that is shown in the element CanonicalizationMethod, shown below in the example of a SOAP payload. The SignatureMethod element 52 is the method that is used to convert the canonicalized SignedInfo into the SignatureValue 54.

The process then uses the SignatureMethod and the SignedInfo to recalculate the digest at 58 and compares the resulting signature value from that recalculation to the provided SignatureValue 56, at 60. If this passes, the process moves forward. If the calculated SignatureValue does not correctly compare to the SignatureValue that was provided, the process fails and the digital signature is presumed to be invalid. If the SignatureValue is correct, the process recalculates the digest of the references contained in a Reference element at 62. Each Reference element includes the digest method and resulting digest value calculated over the identified data object. A data object is signed by computing its digest value and a signature over that value. The signature is later checked via reference and signature validation.

At 64, the recalculated digest is compared to a provided digest value 66 as a second check on the signature validation. If that match is correct, the signature may be optionally stored at 68 for future comparison in transactions with that partner. If the match is not correct, the signature is assumed to be invalid and the process either returns to validating another signature, as shown, or progresses to handle the invalid signature. Handling of invalid signatures is outside the scope of this disclosure. An example of a SOAP payload with these elements is shown below.

```
<Signature Id="MyFirstSignature"
    xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod
          Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <SignatureMethod
          Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
      <Reference
          URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
      <Transforms>
         <Transform
           Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
          </Transforms>
        <DigestMethod
           Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
      </Reference>
      </SignedInfo>
      <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
      <KeyInfo>
        <KeyValue>
          <DSAKeyValue>
             <P>...</P><Q>...</Q><G>...</G><Y>...</Y>
          </DSAKeyValue>
        </KeyValue>
      </KeyInfo>
</Signature>
```

As mentioned above, the required SignedInfo element is the information that is actually signed. Note that the algorithms used in calculating the SignatureValue are also included in the signed information while the SignatureValue element is outside SignedInfo.

The CanonicalizationMethod is the algorithm that is used to canonicalize the SignedInfo element before it is digested as part of the signature operation. Note that this example is not in canonical form. This is an optional process and not required for implementation of embodiments of the invention.

The SignatureMethod is the algorithm that is used to convert the canonicalized SignedInfo into the SignatureValue. It is a combination of a digest algorithm and a key dependent algorithm and possibly other algorithms. The algorithm names are signed to protect against attacks based on substituting a weaker algorithm. To promote application interoperability one may specify a set of signature algorithms that must be implemented,

though their use is at the discretion of the signature creator. One may specify additional algorithms as 'recommended' or 'optional' for implementation; the design also permits arbitrary user specified algorithms.

Each Reference element includes the digest method and resulting digest value calculated over the identified data object. It also may include transformations that produced the input to the digest operation. A data object is signed by computing its digest value and a signature over that value. The signature is later checked via reference and signature validation.

KeyInfo indicates the credential to be used to validate the signature. Possible forms for credentials include digital certificates, tokens, key names, and key agreement algorithms and information, as examples. KeyInfo is optional for two reasons. First, the signer may not wish to reveal key information to all document processing parties. Second, the information may be known within the application's context and need not be represented explicitly. Since KeyInfo is outside of SignedInfo, if the signer wishes to bind the keying information to the signature, a Reference can easily identify and include the KeyInfo as part of the signature.

It must be noted that the specifics of the above message are only intended as an example, and that the use of a SOAP payload is also intended as an example to promote better understanding of embodiments of the invention. No limitation on the scope of the claims is intended, nor should any be implied.

Thus, although there has been described to this point a particular embodiment for a method and apparatus for hardware-assisted credential validation, it is not intended that such

specific references be considered as limitations upon the scope of this invention except in-so-far as set forth in the following claims.

WHAT IS CLAIMED IS:

1. A device, comprising:

a credential validation module to recalculate security credentials received in a datagram and to determine if those security credentials are valid.

2. The device of claim 1, wherein the device further comprises a parser to extract the security credentials from the payload data of the received datagram.

3. The device of claim 2, wherein the parser and the credential validation module reside together.

4. The device of claim 1, wherein the device further comprises a memory to store the security credentials.

5. The device of claim 1, wherein the credential validation module further comprises an arithmetic logic unit and a comparator.

6. The device of claim 1, wherein the security credentials are one of the group comprised of: a token, a digital signature, a cryptographic key, and a digital certificate.

7. The device of claim 1, wherein the security credentials further comprise a digital signature in compliance with the simple object access protocol.

8. A method of validating security credentials, the method comprising:

receiving a datagram including security credentials;

recalculating a representation of the security credentials; and

comparing the representation of the security credentials to a provided representation to determine if the security credentials are valid.

9. The method of claim 8, wherein the method further comprises parsing payload data in the datagram to obtain the security credentials.

10. The method of claim 8, wherein the method further comprises storing the security credentials.

11. The method of claim 8, wherein the security credentials further comprise a digesting method, a credential to be verified and a credential value.

12. The method of claim 11, wherein the credential to be verified further comprises a digital signature in compliance with simple object access protocol.

13. The method of claim 11, wherein the credential value further comprises a digital signature value in compliance with simple object access protocol.

14. A system, comprising:

a credential generator to provide outgoing security credentials;

a port to allow transmission of datagrams to other devices, wherein the transmission may include the outgoing security credentials; and

a credential validation agent to validate incoming credentials received through the port.

15. The system of claim 14, wherein the outgoing security credentials further comprise one of the group comprised of: public and private key pairs, tokens, digital certificates, and digital signatures.

16. The system of claim 14, wherein the system further comprises a memory operable to store the incoming credentials.

17. The system of claim 14, wherein the system further includes a parser to extract credentials received in datagrams through the port.