ODYSSEY: Policy-Driven Anonymizer for Handheld Wireless Devices Privacy

Camille Gaspard
cgg04@aub.edu.lb
Department of Electrical and Computer Engineering
American University of Beirut
Beirut, Lebanon

Abstract - In this paper we present ODYSSEY (pOlicy-Driven anonYmizer for handheld WireleSS dEvices privacY) as an efficient security architecture for assuring privacy through applying selective confidentiality and integrity on web traffic between wireless handheld devices and the Internet. The security gateway, or the anonymizer, sits between the client and the Internet and hides the identity of the user to surfed web sites by taking two responsibilities: First is maintaining privacy by acting as a proxy node to the client. Second is preserving confidentiality and integrity by applying content-based encryption and hashing on the web traffic between the gateway and the client. The system is a scalable, policy-based solution capable of evolving and adapting to suit the security requirements of a wide range of wireless devices with various capabilities and resources. The customizable policy-based architecture in the system can be configured to provide privacy through applying security services according to the content and sensitivity of the network data in the web traffic. In addition, ODYSSEY is autonomous since that it will continuously update the policy based on existing policy rules and user behavior. This gives ODYSSEY considerable performance gains over existing standard anonymizers that use bulk encryption such as SSL. ODYSSEY is designed as a platform-independent architecture and can be seamlessly used to surf the Internet from different client platforms. ODYSSEY does not impose any requirements on surfed web sites.

I. INTRODUCTION

Almost all portable devices such as laptops, palmtops, and smart phones are equipped with built-in Internet connectivity and web browsers which makes them capable of browsing web content anywhere and anytime without being bound to one location. New mobile commerce (m-commerce) applications such as banking and mobile payments are becoming possible and many existing electronic commerce (e-commerce) applications can be modified to fit into the mobile environment.

Although the adoption of m-commerce services is showing some acceleration, great concerns are raised about user privacy and security of the sensitive data flowing over the wireless links where data confidentiality and integrity may be compromised by unauthorized access. Wireless web surfers have special and unique requirements compared to standard Internet users. Usually the wireless web traffic is limited in bandwidth with high latency and frequent disconnections using devices that vary greatly in capabilities and resources. For this reason, the protocols used in securing wireless enterprise web traffic have to be designed specifically for operation in wireless environments and must address the needs and requirements of a large variety of wireless devices which are, in their majority, severely constrained in terms of processor speed, memory resources, storage capacity, and battery power. This diversity makes the implementation of a unique security standard, which encompasses the whole device range, infeasible. A least-common denominator security standard that targets devices with limited memory and slow processors would be unfair for powerful devices and would not meet their security requirements, and in the same sense, a security standard that addresses high-end devices would neither fit nor perform efficiently on limited-resource devices. What is needed is a security protocol that can be customized and configured to perform the security operations flexibly, taking into consideration the memory capabilities and the processing and battery power of the device, the wireless network latency, and the specific requirements of the enterprise application. This ensures the efficient operation of the same application on a wide range of devices and wireless networks. Moreover, this protocol must be extensible, scalable and capable of evolving to meet new challenges and to adapt to new application requirements.

In this paper we present ODYSSYE, an efficient security platform for protecting the privacy and integrity of web traffic exchanged between enterprise mobile wireless devices and the Internet. ODYSSEY is an optimized, policy-driven security architecture capable of evolving to meet the security requirements of a wide range of wireless devices with diverse capabilities and resources. ODYSSEY employs content-based encryption and hashing methodologies that secure network data based on sensitivity and relevance. This gives ODYSSEY major performance advantages over existing bulk encryption protocols such as SSL [1] and TLS [2]. ODYSSEY uses PRIDE [11] as its content-based security engine.

The rest of the paper is organized as follows: in Section II, we review some previous work related to the security of web traffic and m-commerce. In Section III, we describe the main design features of ODYSSEY and its architectural components. Results for a prototype PRIDE (ODYSSEY's security engine) implementation are shown in Section IV. Conclusions and are presented in Section V while current and future work are discussed in VI. Acknowledgements are stated in section VII.

II. PREVIOUS WORK

The default security protocol for protecting web traffic on the Internet today is SSL which is used by standard anonymizers. This protocol provides point-to-point security by

establishing a secure channel on top of TCP, where it supports server authentication using certificates, confidentiality, and message integrity. Optionally, there is support for cryptographic client authentication. Despite its success in securing Internet applications, SSL, in some of its parts, is too compute- and memory-intensive for wireless devices, thus Sun Microsystems has developed a light-weight SSL implementation called KSSL [3] which is a client-side implementation of SSL version 3.0, supporting the RSA-RC4-128-MD5 and RSA-RC4-40-MD5 cipher suites. The main concept behind KSSL is represented in reusing previous session results such as certificate parsing results and master secrets, so as to avoid repeated SSL handshakes. This helps in avoiding complex, resource-intensive public-key operations on the client device. Moreover, KSSL does not implement client authentication since it requires CPU-intensive private-key RSA operations. In spite of the advantage provided by KSSL in securing m-commerce applications, its performance is considered unacceptable when the client needs to communicate with different servers or when browsing sensitive content. This is due to the fact that in such cases a full-handshake SSL operation needs to be carried out every time the client connects to a new web server. The full handshake takes 10–13 seconds on a CDPD wireless network using a 20 MHz Palm PDA. This performance bottleneck can be avoided when the client communicates with the same server repeatedly by performing an abbreviated handshake that makes use of the previous session certificate parsing result and master key, hence bypassing resource-intensive public-key operations. However, storing the master secret and reusing it repeatedly in generating the symmetric ciphering keys raises some concerns about the security of this master key on the wireless device which can be easily snooped or stolen. This is a very important issue to consider since nearly all the security of SSL depends on keeping the master secret private. Moreover, SSL is a bulk encryption protocol that indiscriminately encrypts all the network data without regard to its type or sensitivity. To SSL, network data is of one type, and there is no categorization of this network data based on content or sensitivity. Thus, when using SSL, all the network data is encrypted by the same cryptographic key-strength, which can be unnecessary or even undesirable for some wireless applications. This fact about SSL will become more evident and pressing as the mobile commerce network expands; thus the need will arise for a configurable security model with flexible encryption schemes to meet a range of different requirements [4, 5].

III. ODYSSYE Design and Architecture

This section gives an overview of the design and architecture of ODYSSYE on the client side, the anonymizer, and on the proxy that is used to optimize the performance in trusted intranets. An abstract view of the major components of the security architecture and their interrelationships is shown in Fig. 1.

A. The Security Policy

The Security Policy controls the overall security behavior of the web traffic protected by ODYSSEY. The policy configuration is stored in an XML-formatted document in the anonymizer. This policy is used by the Server Security Engine (see Fig. 1). Each subscriber user to the anonymizer has his/ her own policy. XML is used as the definition language for the high-level Security Policy on the anonymizer. Each user can customize his/ her policy using a web based application on the anonymizer.

The Policy Decision Point is a main component in the security engine. It ensures that the policy configurations are applied on all web traffic which the Security Engine is supposed to protect.

The Policy is divided into two main parts (see Fig. 2): the first part specifies security-related attributes and parameters, while the second deals with how security mechanisms are applied on the web content. The security-related attributes control the behavior of the Security Engine for confidentiality, integrity and key management. Following is a list of these attributes: 1) encryption algorithm; 2) hashing algorithm; 3) key management algorithm; 4) session-keys life time. The second part of the policy is divided into two main sections. The first section specifies the security classes that are to be used by the policy. These classes are a set of security specifications that can be applied later to different types of web content. These security classes are characterized by the following attributes: 1) class name; 2) security level; 3) integrity enforcement; 4) force over local networks.

ODYSSEY supports three main security levels: a High Security level, equivalent to an AES [10] 256-bit key length; a Medium Security level, which is equivalent to an AES 192-bit key length; and a Low Security level, which is equivalent to an AES 128-bit key length. The integrity enforcement attribute in the policy specifies if data integrity needs to be applied on the web traffic. The “force over local” attribute, when set to true, forces the Security Engine running on a proxy server not to decrypt secured web content destined to the local intranet. This is very important when critical web data is transmitted to non-trusted local networks.

The second part of the Policy deals with web content. It specifies the security level and scope for each type of content, by either stating explicitly these security settings or by referring to a certain security class.

Two main components represent the mechanisms using which the security engine is able to successfully select what to encrypt and in what security level. These components are: 1) trigger and 2) scope.

Web content is identified using triggers which might be one of three: 1) filename; 2) tag type; 3) regular expression; 4) web site’s domain. These triggers can be combined together to be more selective in choosing the data to be encrypted. Filenames can be any valid file name, an asterisk (*) specifies all files. Tag type is any HTML tag type. Regular expressions are used to match and secure sensitive patterns that appear frequently in web traffic.

The Security Engine, when finding a match according to the trigger element, encrypts the web traffic that is in the scope of the scope element. The scope can be one of the followings: 1)whole web site; 2) current tag; 3) certain file types in the current web site; 4) current HTML page. These scope elements can be combined in order to be more granular in selecting the data to be encrypted.

All these identification mechanisms help the Policy Enforcement Point to specify the encryption boundaries and security levels of the web data stream.

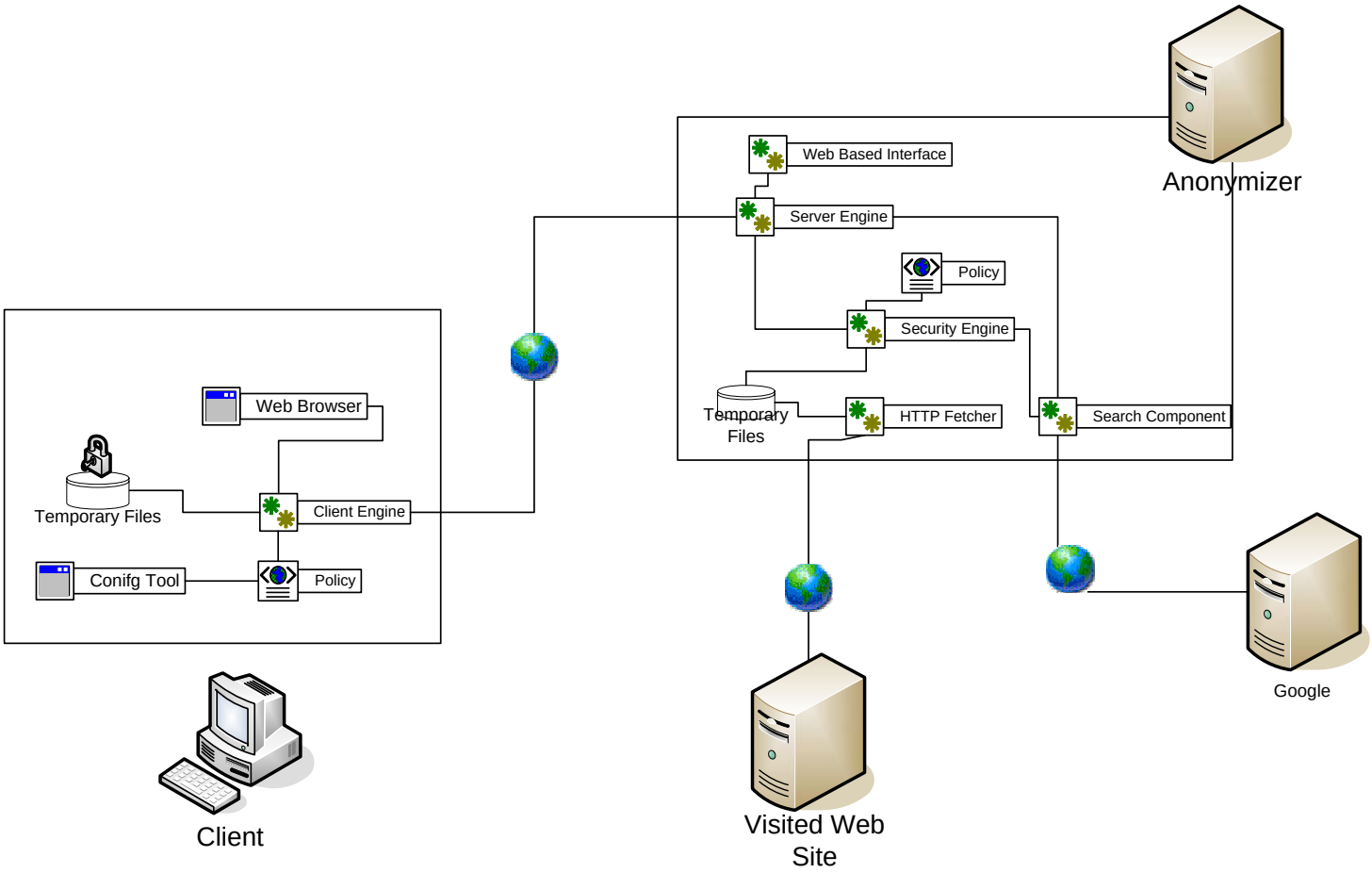

Figure 1. ODYSSEY General Architecture

B. Policy Customization

The XML policy configuration file is loaded to the security engine when the user requests a web site from the anonymizer. The policy information present in the policy configuration store was designed to be human-readable, given the nature of XML, so that it can be easily configured by system administrators using simple XML or text editors. Users can customize their policies via a web interface available at the anonymizer web server farm.

C. Policy Caching and Growing

Policy resolving and caching is an important feature that is performed on the anonymizer server and that is handled by the Policy Loader. For example, in an m-commerce web server, the administrator may need to protect all credit card transactions against tampering and eavesdropping. This is done by adding a corresponding regular expression pattern key in the server-level policy store.

Each policy resolving process might involve reading multiple keys in the policy and making a decision to selectively encrypt the web traffic. The policy keys can be cashed in the memory of the anonymizer server to alleviate the process of policy parsing in later requests by the same user. This procedure does not impose a lot of overhead on the system since, when the server's Security Engine resolves a new policy, it compacts the resolved settings and caches them for later reuse.

Caching improves the policy loading time by using a compacted binary copy of the security policy instead of parsing it every time it is requested. However, due to the possible modification of the policy configuration by administrators, a hash is used to determine whether the policy information in the server cache is up to date or not.

The policy might contain some keys that augment the policy store by adding new keys to the policy store. For example, one key might state that whenever finding "credit card" to add a key to encrypt all traffic belonging to that web site's domain.

D. Web Traffic Confidentiality and Integrity

The Security Engine is the component responsible for providing data confidentiality and integrity based on the sensitivity of the web traffic (see Fig. 1). To support a flexible encryption scheme, ODYSSEY identifies the content to be secured in the security policy. It is the responsibility of the server-side Security Engine (see Fig. 1) to perform the encryption and hashing operations of the web data and to construct the secure response based on the request coming from the client and the security policy. The Security Engine on the client-side accepts the secure response, performs the necessary integrity verification and decryption operations, constructs the original response message as sent by the surfed web site and delivers it to the

```
<policy Encryption_Algorithm="AES" Key_Management="DH" MAC_Algorithm="HMAC" Key_Lifetime="5">
  <classes>
    <add Classname="SecureClass1" Security_Level="Medium_Security" Integrity_Enforcement="No"
      ForceOverLocal="No"/>
    <add Classname="SecureClass2" Security_Level="High_Security" Integrity_Enforcement="Yes"
      ForceOverLocal="Yes"/>
  </classes>
  <keys>
    <key Class="SecureClass2">
      <trigger Domain="www.SecureWebSite.com"/><scope Filename="*"/></key>
    <key Class="SecureClass1">
      <trigger Domain="www.Amazon.com"/><scope Filename="orderdetails.aspx"/></key>
    <key Class="SecureClass1">
      <trigger Match="regularexpression"/><scope Filename="*.php,*.asp"/></key>
    <key Security_Mode="No_Encryption">
      <trigger Domain="www.aub.edu.lb"/><scope Filename="*.bmp,*.avi,*.css,*.js"/></key>
    <key>
      <trigger Match="regularexpression"/>
      <AddKey>
        <key Class="SecureClass2">
          <trigger Domain="?"/><scope Filename="*"/></key></AddKey></key>
  </keys>
</policy>
```

Figure 2. ODYSSEY's Policy

web browser. This is done based on the request mode in the security policy. Sending a secure post-back request form the client to the server follows the same procedure. In this case the client-side and the server-side Security Engines will switch roles and the security operations and their strengths will be based on the mode of the "post-back" interaction type in the security policy.

The following procedures summarize the overall request/response model of ODYSSEY on the server and client sides. The server: 1) receives incoming requests from the client; 2) coordinates with the key management server in order to initialize a new secure session; 3) parses the policy store and resolves the security settings, if no cache exists; 4) caches the resulted settings; 5) HTTP fetcher retrieve the requested document or file; 6) applies confidentiality and integrity settings on the retrieved document; 7) attaches to the response an encrypted version of the necessary policy description; 8) flags the response as secured and identifies if the post-back from the client needs to be secured as well; 9) finally, it receives the encrypted post-back from the client, decrypts and forwards it to the appropriate web site.

On the other hand, the client: 1) receives requests from the web browser and forwards them to the anonymizer server; 2) waits for the response and checks for its security flag; 3) forwards the response as it is to the web browser, if security flag is not set; 4) decrypts parts of the web traffic and checks for its integrity based on the decrypted policy, if security flag is set; 5) sends the response as plaintext back to the web browser; 6) encrypts and sends post-backs that are to be secured based on the policy.

Most of the intelligence of the client-side Security Engine can be moved to an Enterprise Proxy Security Engine that resides on the local enterprise's network and serves clients in a certain trusted intranet. This alleviates the load from the client machines especially in the case of limited devices, and boosts the performance of the whole security system. In case the secured web traffic is flagged to force security over local networks, the client-side machine will have to take responsibilities again. This is the case when highly critical data is transmitted over non trusted local network.

IV. Implementation

ODYSSYE was designed in a platform-independent manner and its design features are optimized for implementation on a wide range of servers and wireless client platforms. An implementation of ODYSSYE's security engine was developed in a previous project (PRIDE [11]). PRIDE was developed for a Pocket PC client device using the .NET Compact Framework 2.0 specifications and a server running Windows Server 2003

In this section, a brief description of the implementation along with the technologies, tools and devices used are presented. We begin by describing the server and the client environments.

PRIDE server-side components were implemented on a Windows Server 2003 machine. The PRIDE server-side security engine was developed as an HTTP filter using IIS 6 and ASP.Net 2.0. This security engine is responsible of intercepting the incoming HTTP requests to the server and securing the responses according to a policy file that is specific to each web folder. The responses are standard HTTP responses and browsing content-base secured pages in a normal browser will allow the user to see the unencrypted data.

The client machine is an HP iPAQ Pocket PC h6340. The .Net Compact Framework 2.0 was used to develop the PRIDE client-side security engine. The client is connected to the server using a USB Sync cable provided with the PDA device.

To simulate a real-life scenario where securing web traffic is a requirement, we chose to test PRIDE on an e-commerce web site. A typical example is Amazon.com. We simulated purchase operations and downloaded the HTML pages along with all their related components. The total size of the tested site was 9.3 MB. The elements were distributed as follows: 2.5 MB of HTML pages, 2.2 MB of images and 4 MB of product manuals, i.e. PDF, and video content.

A list of the web site files was fed to the client engine which was responsible of fetching and decrypting these files from the server. The tests were done in three stages. Each stage was repeated many times to make ensure consistency of the results. The first stage was conducted with the policy set on the server to send all content as clear-text without any encryption. This phase was conducted in order to measure the time the client needs to download the content. The second phase was to set the policy to encrypt everything; i.e. bulk encryption. In this case, the system is equivalent to a standard bulk encryption protocol. The third phase of the tests was to implement the PRIDE content-based encryption methodology. The policy was built based on the following analysis: 300 KB of the data was considered highly sensitive. This data can not be content-base encrypted. This consisted of sensitive images; 1.2 MB of HTML files were eligible to be selectively encrypted. The selection of what to encrypt was based on regular expressions; 7.8 MB of images and PDF files were considered to be not sensitive and thus sent unencrypted. This category consists mainly of product-related data such as images and few manuals in the format of PDF and HTML.

PRIDE showed a great performance enhancement by selectively encrypting sensitive data and sending unimportant data unencrypted. The time needed to download the contents to the client device was calculated in order to focus on the overhead introduced by the encryption and related processes. As Table 1 shows, it was possible to reduce the time that bulk encryption takes to secure a web site by 83% (from 42 sec to 7 sec). From a speed point of view, PRIDE is almost six times faster than bulk encryption.

	Encryption time	**Encryption Speed**	**Key Length**
Bulk Encryption	42 seconds	224 KB/Sec	192 bits
PRIDE Encryption	7 seconds	1.3 MB/Sec	128/192/256 bits

Table 1. PRIDE ClientTest Results

V. CONCLUSION

In this paper we presented ODYSSEY, an efficient security architecture for assuring privacy and security through applying selective confidentiality and integrity on web traffic between wireless handheld devices and the Internet. ODYSSYE is a customized security protocol capable of operation in diverse wireless environments with different resources, capabilities, and configurations. The customizable policy-based architecture in the system can be configured to provide privacy through applying security services according to the content and sensitivity of the network data in a web transaction. ODYSSEY does not impose any requirements on surfed web sites. Previous implementation [11] showed ODYSSYE's engine to be almost 6 times faster than bulk encryption protocols.

VI. CURRENT AND FUTURE WORK

The results shown in this work were extracted from PRIDE's implementation. PRIDE is the security engine used by ODYSSEY. Currently, our research group is working on implementing a full prototype implementation of ODYSSYE. Future work includes using SSL in a content-based manner in order to be able to use standard web browsers (that have built-in SSL capabilities) instead of adding the security plug-in to client's machines.

VII. Acknowledgments

This research work is part of the activities within the security research group in the ECE department at AUB. This research was supervised by Dr. Ayman Kayssi, Dr. Ali Chehab and Wassim Itani to each of whom I feel very grateful.

REFERENCES

[1] A. Freier, P. Karlton, and P. Kocher, *The SSL Protocol Version 3.0*, Internet-Draft, November 1996.

[2] T. Dierks and C. Allen,*The TLS Protocol – Version 1.0*, Internet –Draft, November, 1997.

[3] V. Gupta and S. Gupta, "Securing the Wireless Internet," *IEEE Communications*, December 2001.

[3] W. Itani and A. Kayssi, "J2ME End-to-End Security for M-Commerce", *in Proc. IEEE Wireless Communications and Networking Conference*, March 2003.

[4] W. Itani and A. Kayssi, "SPECSA: a Scalable, Policy-driven, Extensible, and Customizable Security Architecture for Wireless Enterprise Applications", *in Proc. IEEE IPCCC Workshop on Information Assurance* (WIA04), April 2004, Phoenix, Arizona.

[6] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, E. Simon, *XML-Signature Syntax and Processing*, W3C Recommendation, 2002, *http://www.w3. org/TR/xmldsig-core/.*

[7] T. Imamura, B. Dillaway, E. Simon, *X.M.L. Encryption, Syntax and Processing*, W3C Recommendation, 2002, *http://www.w3.org/TR/xmlenc-core/.*

[8] Organization for the Advancement of Structured Information Standards, SAML 1.0 Specification, OASIS, 2002, *http://www.oasisopen.org/ committees/download.php/1383/oasis-sstc-saml-1.0-pdf.zip.*

[9] Telecommunication standardization sector of ITU, *ASN.1 Encoding Rules: XMLEncoding Rules* (XER), ITU-TRecommendationX.693, *http://www.itu.int/ITU-T/studygroups/com17/ languages/X.693_0903.pdf*

[10] J. Daemen and V. Rijmen, "Rijndael, the advanced encryption standard," *Dr. Dobb's Journal*, March 2001.

[11] W. Itani, C. Gaspard, A. Kayssi, and A. Chehab, "PRIDE: Policy-Driven Web Security for Handheld Wireless Devices".