MiMi: A Java Implementation of the MicroMint Scheme*

Vesna Hassler, Robert Bihlmeyer,
Michael Fischer, Manfred Hauswirth

Technical University of Vienna, Distributed Systems Department

Abstract

In this paper we describe an experimental implementation of the MicroMint micropayment scheme in Java. We apply this scheme to purchasing Web pages. A prerequisite was to accomplish this without having to change the code of either the Web server or the Web client. We discuss the implementation issues and security considerations. Our implementation requires the local protocol handler feature offered by Sun Microsystems' HotJava 1.0 browser.

Keywords: network security, Web security, Java security, electronic commerce, micropayment schemes

1 Introduction

The main motivation for introducing so-called *micropayment schemes* into electronic commerce protocols is that not all Internet commerce applications require transactions of large amounts of money. Accordingly, the security risks related to a single purchase are not so high. It is therefore rather expensive to deploy security mechanisms suitable for high security risks. For example, a typical charge for purchasing a Web page is one cent. Consequently, the only attack worth trying would be a large-scale forgery. Therefore micropayment schemes should be aimed at preventing large-scale attacks that would involve hundreds of thousands of purchases rather than at preventing a few losses in the range of one cent.

In a micropayment scheme, typical participants are a customer, a broker and a vendor. The customer buys *digital coins* from the broker and gives them to the vendor as payment for some service. The vendor returns coins to the broker in return for payment by other means (*redemption*).

In this paper we describe an implementation of a micropayment scheme (MicroMint [7]) called *MiMi*, applied for purchasing Web pages. In this setting the vendor is an information server that charges customers for accessing its Web pages. The server is implemented as a

*This work was supported in part by a grant from Hewlett Packard Laboratories.

standalone Java application, but could also be implemented as an extension of a Web server (e.g. using the Java Servlet API [12]).

2 MicroMint

MicroMint [7] is a micropayment scheme intended for facilitating small purchases over the Internet. It offers low security, but is very fast because it makes no use of public-key cryptography. Its main advantages over other micropayment schemes [9] are as follows:

- it is off-line from the broker's point of view,
- it does not use either digital signatures or any other public-key scheme, and
- small-scale forgery attempts do not pay off.

At the beginning of each month the broker issues new coins. Unused coins are returned to the broker at the end of each month. Each coin is represented by k integer values (we use 32-bit integers) such that their hash values (i.e. MD5 digests [6]) all have identical low-order n bits. This is called a *k-way collision*. Additionally, the c high-order bits of the hash value are specified by the broker, and are different for each month. For a detailed discussion of the MicroMint scheme see [7].

3 Java Security

The necessity for a sound security concept for the Java programming language results from the fact that most Java code is intended to be automatically downloaded across the network to run on a user's machine [13]. The main problem here, from the security point of view, is how to protect the user's host and data from being damaged by running malicious Java code. Due to the Java Virtual Machine [5] concept, Java code runs on all of the most popular platforms without recompilation. In other words, Java is an implementation of Web-based executable content.

The purpose of the Java security reference model [2] is the enforcement of Java language semantics [10] and a Java-enabled application's security policy. The Java Virtual Machine (JVM) enforces the Java language security features, like access modifiers for variables and methods. It calls the Class Loader in order to ensure that class names are mapped to class code in a proper way. JVM also provides the Bytecode Verifier to validate non-system classes. And finally, the Security Manager performs run-time checks on "dangerous" methods, like file read/write operations. Each Java-enabled browser uses its own version of the Security Manager. The Security Manager policy of most browsers is usually very restrictive. For example, applets cannot access local files at all. The new release of the HotJava browser (1.0 preBeta2) enables applets to gain different access permissions based on their digital signature [11].

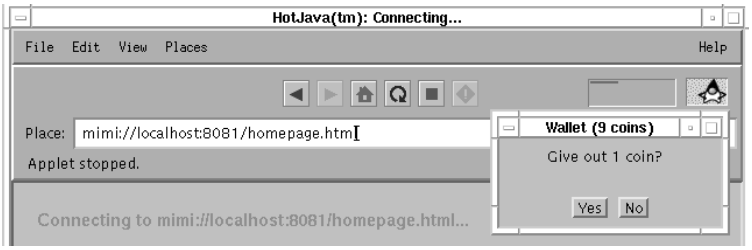

Figure 1: MiMi - Loading a Web page

The initial design of MiMi was intended to work with any Java-enabled browser. If a customer wished to purchase a Web page, an applet provided by the vendor would be downloaded by the customer's browser. This applet would take care of the communication between the customer and the vendor, i.e. its originating host, which is allowed by most browsers. However, since the vendor's applet cannot be trusted (most browsers' Security Managers do not allow non-local applets to access local resources at all), it could not read the coin(s) required for purchasing the requested page. Thus it would be necessary to have an additional, local applet that would communicate with the vendor's applet and operate on local files in which the coins and the security relevant information are stored. Unfortunately, inter-applet communication is not possible for applets with different security contexts (i.e. different Security Managers), so we had to abandon that solution.

HotJava 1.0 preBeta2 allows an applet to get access permissions for local files based on its certificate and digital signature. It is an extension of the Access Control Lists of Sun's Appletviewer [10]. This feature would allow a trusted digitally signed applet originating from the vendor to access the customer's wallet. A problem with this solution is that it might be necessary to repeatedly download the vendor's applet for each requested Web page. The vendor's applet should therefore stay resident in the browser and reactivate itself if the customer requested a new page from the same vendor.

In the current solution we don't use applets, but a locally installed protocol handler for HotJava. The client program defines a protocol called *MiMi*. The MiMi protocol handler is installed locally, so that it can get all permissions necessary to access local files without causing security problems.

4 An Overview of MiMi

MiMi comprises three Java applications (MMOrder, MMBroker and MMVendor), a protocol handler [15], as well as the HotJava 1.0 preBeta1 browser. The overall structure is depicted in Figure 2. The MiMi protocol handler enables the communication between the HotJava browser and the information server, i.e. MMVendor. Disadvantages of this approach are that the protocol handler has to be installed locally, and that this feature is

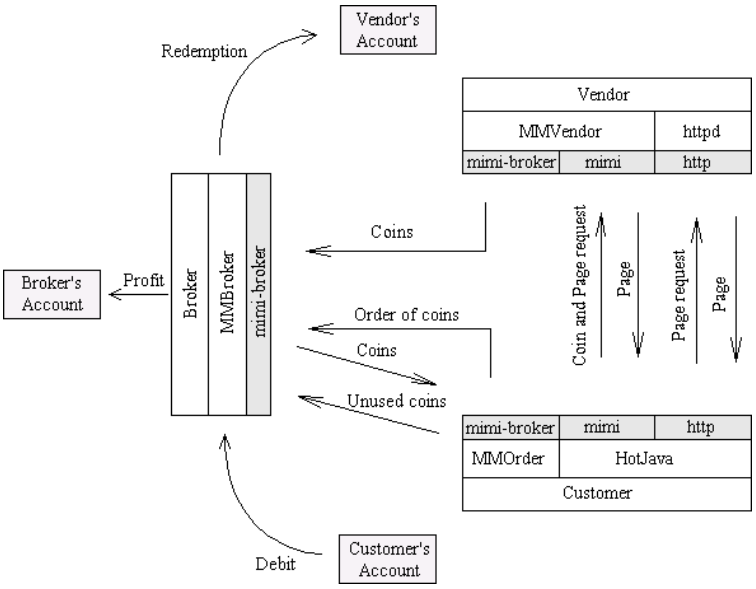

Figure 2: MiMi - An Overview

currently not supported by browsers other than HotJava.

In our example setting MMVendor requires one digital coin for purchasing any of its Web pages. The customer can buy coins from MMBroker using the MMOrder application. The coins are stored in the customer's directory, in a file called *Wallet*. MMBroker mints coins and stores them in its own wallet file.

If the user wishes to access a MMVendor's page, s/he starts HotJava and types in a MiMi URL, like *mimi://host:port/dir/page.html* (see Fig.1). When purchasing a Web page, the user is asked to pay one coin to MMVendor. MMVendor checks whether the coin is really a k-way collision, and whether it has already received it that month. If everything is correct, MMVendor accepts the coin and sends the requested page to the user. The user can view the page in his/her HotJava browser or, otherwise, the corresponding error message. At the end of each day MMVendor returns all collected coins to MMBroker. MMBroker checks each returned coin to verify whether it has been previously redeemed. For each valid coin MMBroker pays MMVendor a certain amount of money, e.g. one cent.

4.1 Some design issues

OMT model. In Fig.3 the OMT model [8] with the main vendor and customer classes is shown. For simplicity, we omitted some attributes and operations that are of little or no importance for this explanation.

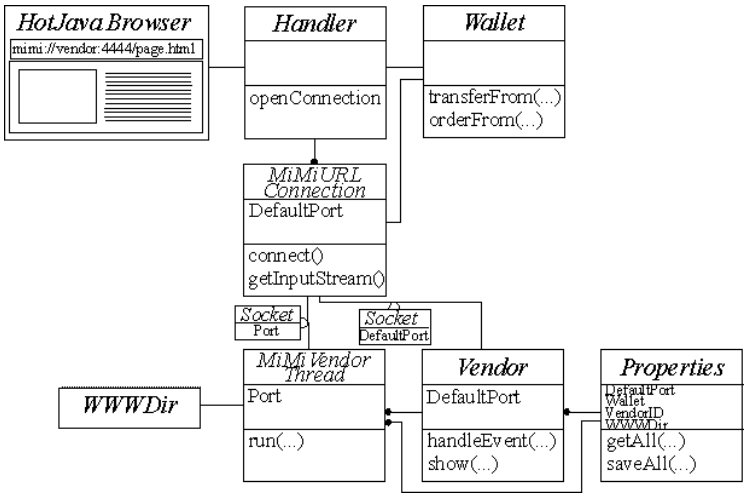

Figure 3: MiMi - The object model

How much to pay for a page? In the current MiMi implementation, one coin is required for one Web page. However, parts of a Web page (e.g. pictures) may be given as hyperlinks or as local links pointing to local files. If the reference is given as an hyperlink for the HTTP protocol, it is assumed to be public domain, so no additional coin is requested. If the reference is given as an hyperlink for the MiMi protocol, an additional coin is requested, i.e. a new window asking for a coin appears. If the customer does not want to pay for the ”extra” pages, s/he can simply refuse further payments and download only the content the originally requested reference is pointing to.

4.2 MiMi security considerations

Customer-Broker. When purchasing coins from the broker, the customer must be sure that s/he is contacting the genuine one whose coins will be accepted as expected. In other words, the broker has to be authenticated. If the customer is authenticated, the broker can automatically withdraw the appropriate amount of real money from the customer’s account, either locally at the broker or at the customer’s bank. If the customer is not authenticated, s/he can anonymously order some coins from the broker and get them after having transferred the corresponding amount of real money to the broker’s account. The coins must be transferred from the broker to the customer’s wallet in a confidential way in order to prevent eavesdropping. The current version (February 1997) of MMOrder does not include authenticity and confidentiality, but we plan to implement these security services based on the SSL protocol [4]. Another possible solution is to use secure mail.

Broker-Vendor. Digital coins that the vendor has collected from the customers are redeemed by the broker that issued them. In order to prevent the man-in-the-middle attack it is recommendable to authenticate the broker, or at least use a long-term symmetric encryption key that would provide both weak authentication and confidentiality. Otherwise, using Web spoofing techniques [3] an attacker could masquerade as the broker, collect the coins from the vendor and redeem them at the genuine broker. In order to prevent eavesdropping, this exchange should be confidential. If the vendor is authenticated, the broker can automatically transfer the real money to its account. Otherwise, the broker could issue a digitally signed check and send it to the vendor in a confidential way. Here it is also be possible to use secure mail.

Customer-Vendor. The security problems that can arise by the customer-vendor communication are stealing of coins and stealing of Web pages. One of the design goals of MicroMint is to completely avoid public-key cryptography. However, it is recommendable to use a long-term symmetric encryption key between the customer and the vendor because it provides both weak authentication and confidentiality. This method would protect against stealing of both coins and Web pages. For each exchange of the long-term key the vendor should be authenticated using some strong authentication protocol. There are also other techniques to prevent stealing of coins proposed by the authors of MicroMint, like user-specific or vendor-specific coins [7]. If the [non-specific] coins are sent in cleartext, an attacker could use Web spoofing techniques [3] to collect coins and send in return fake Web pages. However, this attacker would have to provide Web pages that in the long run look similar to the genuine pages, to a large number of customers. Otherwise, this attack would not pay. If the vendor's Web pages are sent in cleartext, an attacker could collect them, become a vendor him/herself and sell the stolen pages. However, this would be revealed pretty soon, by the genuine vendor or by an honest customer. Moreover, if the contents of the Web pages change on a daily basis (like newspapers), this type of attack does not pay at all.

5 Conclusions

In this paper we presented a simple solution for applying the MicroMint scheme to purchasing Web pages. At the moment this solution works with Sun's HotJava browser only, because we use one of its advanced features (locally installed protocol handler). We hope that in the near future this feature will be offered by other browsers as well, and that it will be possible to dynamically load the protocol handler.

The new release of HotJava 1.0 (preBeta2) enables applet authentication, so that an applet can access the local environment if it is digitally signed and if its originator has a proper certificate. Having this feature in Java-enabled Web browsers would make it possible for the protocol handler to work with applets loaded over the network, even without an integrated protocol handler support [14].

If the protocol handler could also be loaded dynamically, it would have to undergo strict security checks. This is most probably the reason why the new HotJava release (1.0 pre-Beta2) still does not allow dynamically loaded protocol handlers, although it was expected; this feature would namely require a security concept, similar to applets.

References

[1] Berners-Lee, T., R. Fielding, H. Frystyk, *Hyptertext Transfer Protocol – HTTP/1.0*, Request For Comments, RFC 1945, May 1996, URL:<ftp//ds.internic.net/rfc/rfc1945.txt>

[2] Erdos, M., B. Hartman, M. Mueller, *Security Reference Model for the Java Developer's Kit 1.0.2*, Nov 1996, URL:<http//www.javasoft.com/security/SRM.html>

[3] Felten, E.W., D. Balfanz, D. Dean, D.S. Wallach, *Web Spoofing: An Internet Con Game*, Technical Report 540-96, Department of Computer Science, Princeton University, 1996 URL:<http//www.cs.princeton.edu/sip/pub/spoofingDocumentWithLongUntypeableName.html>

[4] Freier, A.O., P. Karlton, P.C. Kocher, *The SSL Protocol. Version 3.0*, Internet Draft, March 1996, URL:<ftp//ietf.cnri.reston.va.us/internet-drafts/draft-freier-ssl-version3-01.txt>

[5] Lindholm, T., F. Yellin, *The Java Virtual Machine Specification*, Addison-Wesley: Tha Java Series, 1996

[6] Rivest, R.L., *The MD5 message-digest algorithm*, Internet Requests for Comments, RFC 1321, April 1992

[7] Rivest, R.L., A. Shamir, *PayWord and MicroMint: Two simple micropayment schemes*, also presented at the RSA '96 conference, URL:<http//theory.lcs.mit.edu/ rivest/RivestShamir-mpay.ps>,

[8] Rumbaugh, J., M. Blaha, W. Premberlani, F. Eddy, W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, 1991

[9] Hallam-Baker, P.M., *Electronic Payment Schemes*, URL:<http://www.w3.org/pub/WWW/Payments/roadmap.html>

[10] JavaSoft, *Frequently Asked Questions: Applet Security*, URL:<http://java.sun.com/sfaq/index.html>

[11] JavaSoft, *Java Security: New!*, URL:<http://www.javasoft.com/index.html>

[12] JavaSoft, *The Java Servlet Development Kit*, URL:<http://jeeves.javasoft.com/products/java-server/sdk>

[13] McGraw, G., E.W. Felten, *Java Security. Hostile Applets, Holes and Antidotes*, John Wiley & Sons, Inc., 1997

[14] Niemeyer, P., J. Peck, *Exploring Java*, O'Reilly & Associates, Inc., 1996

[15] Sun Microsystems, Inc., *HotJava 1.0 preBeta2: User's Guide: Installing a Local Protocol Handler*, 1997