Webcard: a Java Card Web Server

Jim Rees and Peter Honeyman

Center for Information Technology Integration
University of Michigan
Ann Arbor

`http://smarty.citi.umich.edu/`

Abstract

Webcard is a Java application that implements a TCP/IP stack and HTTP server and runs on a Schlumberger Cyberflex Access smartcard. In this report, we describe the architecture and implementation of Webcard and the constraints and assumptions that influenced its design. Complete sources for the application and its supporting environment are available.

1 Introduction

Smartcards have numerous properties that make them useful in a security infrastructure:

- smartcards are tamper-resistant;
- smartcards export a restricted API that limits access to content and functionality;
- this API includes generic cryptographic functionality.

In combination with these influences, the inherent mobility and convenient form factor of smartcards suggests deployment in application domains that offer secure, personalized services; consequently, smartcard specifications are dictated by an international standard. Consequently, they are deployed worldwide in a variety of infrastructures and applications.

Consider, for example, a health card application, in which personal medical information is stored securely on a smartcard. Health care consumers maintain strict physical security of their personal data by storing the health card in their wallets. However, when it comes time to use the information, the health card must be accessed by a proprietary application provided by the card manufacturer or system integrator. So while the health care consumer maintains the security of her personal information, she does not really control it, and is not even able to examine it.

We suggest an alternative. Providing access to confidential information through web protocols can preserve the security and availability of critical, confidential information when it is needed, but has two additional benefits:

- the information can be retrieved without special applications, i.e., by any web browser; and
- the information can be accessed remotely

The former benefit provides true control over personal data, while the latter dovetails with telemedicine applications, so that the health care consumer need not be physically present in the clinical setting.

The predominance of Internet protocols governing network communication cries for integration of smartcards with Internet technologies. The success of the Internet is also due to worldwide acceptance of strict standards on packet formats and concomitant semantics. The first step toward smartcard integration with Internet technologies is the development of a compliant Internet communication stack on a smartcard. In this paper, we describe a prototype implementation of IP, TCP, and HTTP on a commercial smartcard, Schlumberger's Cyberflex Access Java Card.

The rest of this paper is organized as follows. First, we describe the environment in which this research was performed. Next, we discuss naming and addressing issues that arise when integrating smartcards with the Internet. The next section describes our implementation in detail. We conclude with a section that discusses the results and describes our plans for further development.

2 Development environment

The Program for Smartcard Technology at the University of Michigan's Center for Information Technology Integration (CITI) is a research partnership with Schlumberger's Austin Product Center. The Program is actively engaged in research projects that enhance and extend the capabilities of smartcards. Among CITI's goals in the Program, two stand out:

- innovative computer security applications of smartcards, and
- new models of interaction with smartcards.

To these ends, we developed Webcard, a web server that is entirely contained in a commercial, off-the-shelf smartcard.

Webcard accomplishes both of CITI's objectives in the categories of research stated above. Webcard takes advantage of the inherent security properties of smartcards, such as tamper resistance and a programming interface appropriate for security applications. In contrast to the arcane, operating system dependent applications characteristic of the smartcard industry, Webcard offers a radically new mode of interacting with smartcards, one that is enabled by any Internet-capable web browser.

3 Location dependence

Internet services are bound to Internet addresses, which are themselves tightly woven to the Internet routing infrastructure. The very mobility and security of smartcards complicates the challenge of making rendezvous between arbitrary clients and smartcard-based servers.

Preserving security suggests an end-to-end approach so that intermediate systems along the network path merely forward datagrams. This limits the security considerations to the client and server protocols and applications.

To achieve end-to-end communication, each smartcard must be independently addressable, i.e., each smartcard must have its own Internet address. One option, the one we have chosen, is to assign a fixed IP address to each smartcard. The choice of address dictates the path of IP packets directed to the card's address. While this severely limits mobility, we find it useful in our prototype implementation as a quick way to test out ideas unrelated to network routing.

Another option is to assign an address dynamically to each smartcard as it enters the Internet infrastructure, e.g., with RARP [6] or DHCP [2]. Dynamic DNS server updates [5] can provide fixed domain names for smartcards, solving the service rendezvous problem. However, this depends on the availability of secure, dynamic DNS servers.

4 Technical details

Webcard is a web server running on a Schlumberger Cyberflex Access Java Card [15]. The card is programmed by the manufacturer to implement a Java virtual machine (JVM), recognizing a sizable subset of the Java programming language. Specifically, Cyberflex implements the Java Card 2.0 specification [12]. Java Card is intended to support multiple applications on a single card, as described in ISO 7816-4 [8] and EMV 96 [14]. Webcard is written as a single Java Card application (variously called an applet or cardlet).

The Cyberflex Access card has 16 KB of EEPROM and about 1.2 Kbytes of RAM. These limited resources make it very difficult to implement a full, standards-compliant version of TCP/IP [3, 4]. While that is our ultimate goal, we must also accommodate the size limitations imposed by current smartcards; we find it useful and interesting to see how much we can accomplish in as little space as possible.

As a first step toward implementing a standards compliant TCP/IP stack, we elected to implement a minimal, functional server. Our main "robustness" criterion is to produce a server that responds to valid inputs and does not crash when presented with invalid inputs. We depend on the TCP peer to assure reliable operation.

HTTP [1], TCP, and IP specify many requirements, many of which are rarely or never used in practice. For our prototype implementation, we elected to elide those specifications that are not required in normal operation. To determine which parts of the protocol are actually used, we captured tcpdump traces of HTTP transactions from several different clients against an existing server. The assumptions described below are based on the observed traces.

4.1 One connection at a time

The Webcard server is simplified by making the assumption that only one connection is active at any time. This allows the server to preserve state for a single connection until a new request comes in. This also eliminates the need to time out defunct connections and to respond to most state change requests. However, most web browsers run requests in parallel, so the server must not return pages with inline content such as images.

It should not be difficult to relax this restriction. The only connection state kept by the Webcard is the file name; TCP state, which is remembered but never used; and TCP port, to enforce the one connection restriction. Connections can be discarded in LRU order as new connection requests arrive, eliminating the need for a timer, which is unavailable on the Cyberflex Access platform.

4.2 HTTP

The server speaks a subset of the HTTP 1.0 protocol, which is simpler and easier to implement than HTTP 1.1 or later. Earlier versions of HTTP, such as HTTP 0.9, are unable to communicate with Webcard, but these clients are now very rare. Modern web clients implement HTTP 1.1 or later, which are required to be backward compatible with HTTP 1.0.

Each request is handled as an individual TCP connection. The HTTP status line, "HTTP/1.0 200 OK," and the HTTP headers are stored in the files being served, so the server itself does not generate any headers or send any data other than what is in the file.

An HTTP 1.0 GET request consists of the string "GET," followed by one space character, followed by a server-relative URL. (Webcard does not support any other methods, such as HEAD, POST, or PUT.) For now, URLs are assumed to be three characters, with the last two characters being the file name. (ISO 7816-4 file names are two bytes.)

When the server receives a request, it selects the requested file. It does not store any other state that reflects the identity of the requested file. This implies that only a single HTTP connection can be active at any time, as described above.

4.3 TCP

The server has no configuration information. The network connection is point-to-point, so all incoming packets are assumed to be addressed to the server. The TCP stack simply swaps the source and destination addresses when it constructs a reply packet. No subnet or routing information is required.

Webcard discards any packets not addressed to the HTTP port (TCP port 80). Any TCP options are ignored.

The TCP state machine only has three states: LISTEN, ESTABLISHED, and FIN-WAIT-1. It is incapable of initiating a connection, and does not have the corresponding SYN-SENT state. It also does not have a CLOSED state. Although Webcard keeps track of the TCP state, it makes no use of this information.

The TCP stack never retransmits. This eliminates the need for timers, which are unavailable anyway, and for keeping track of (most) TCP state. We assume the TCP peer retransmits when necessary. In practice, packets are rarely dropped.

The state machine responds to four types of packets. A SYN elicits a SYN ACK reply and transitions to ESTABLISHED, without waiting for the peer to ACK the SYN. We assume that the SYN ACK will not be dropped and will eventually arrive. This assumption is benign: if SYN ACK does get dropped, the peer will retransmit the SYN, allowing connection establishment to proceed.

HTTP 1.0 allows only one line of text to be sent to the server; following our restrictions to HTTP 1.0 described above, any packet with data is assumed to be a complete HTTP GET request. Webcard URLs are exactly three bytes. We assume that the seven bytes in a GET URL request arrive in a single, unfragmented TCP segment. The server extracts the URL from this request and selects the given file in the ISO 7816-4 file system. If the file does not exist, the server selects a file named "nf", which contains a "404 Not Found" error message. The data packet elicits an ACK of the client's sequence number.

A FIN elicits an ACK and transitions the TCP state machine to LISTEN. HTTP clients always wait for the server to close the connection, so there is no CLOSE-WAIT or LAST-ACK state. If the client does try to close the connection prematurely, it will wait in vain for FIN from the Webcard and will be stuck in FIN-WAIT-2 indefinitely. Most TCP clients eventually recover from this.

An ACK with no data attached elicits data from the currently selected file. There is no windowing – data is sent when the ACK for the previous segment arrives. Webcard sequence numbers always start at zero, so the client's ACK number gives the offset into the file.

Webcard does not check the client's checksum and ignores the offered window, which is benign as the card never sends more than one unacked segment of 248 bytes.

The push flag, urgent flag and pointer, and RST packets are all ignored. Outgoing packets always offer a small fixed window. The actual size of this window is unimportant – we assume the client will never want to send more than 17 bytes.

4.4 IP

Incoming packets are assumed to contain no IP options. It would not be difficult to process options, but in practice IP options are never used. The IP header checksum must be computed with 16 bit arithmetic because the card does not implement 32 bit arithmetic operations. The checksum routine is simplified by observing that an IP header is never long enough to overflow a 16 bit sum.

The MRU (incoming MTU) is limited by the ISO interface to slightly less than 256 bytes. Webcard does not implement IP reassembly, because the only important incoming information is the URL, which fits in the first 17 bytes.

4.5 Cardlet details

Cyberflex extends Java Card in a number of ways. Cyberflex cardlets contain a main method in addition to the Java Card methods. This allows them to support standalone programs. Webcard does not depend on this feature.

A cardlet must have at least three methods, "install," "select," and "process." The install method is invoked once at the time the card is initialized. It creates and initializes the objects needed by the applet. The select method is invoked at the time the cardlet is selected, usually via the "select" application protocol data unit (or APDU). A cardlet can be set as the default for the card, in which case that cardlet is implicitly selected whenever the card is used.

The process method does all the work. When an APDU is sent to the card, that APDU is passed to the process method of the currently selected cardlet. IP packets are sent to the Webcard encapsulated in an APDU that gets passed to the process method.

On reset, the Cyberflex Access default loader waits for an incoming APDU and passes it to the ip7816 cardlet. If the APDU is an IP packet (INS=0xFE), the cardlet processes the APDU; otherwise the cardlet passes the APDU back to the default loader.

The Webcard cardlet extracts the data length, destination port, and several other fields from the IP and TCP headers, then enters the TCP state machine. It then constructs a reply packet if needed, optionally attaches outgoing data to it, computes TCP and IP checksums, and sends the reply packet as outgoing 7816 data.

At several points in this process the cardlet calls apdu.waitExtension() to send a 7816 no-op to the card terminal. This prevents the terminal from timing out while the card is processing.

The Webcard cardlet is about 1200 bytes of Java byte code, leaving about 14 Kbytes of space for web content.

4.6 Card management

Content is loaded onto the Webcard using SCFS [11], CITI's extension to the UNIX operating system, which mounts any ISO 7816-4 smartcard file system into the UNIX file system name space.

Cardlets can be written in any Java development environment; we tend to use standard UNIX editors and Sun Microsystem's JDK [13] for compiling into byte code. A Cyberflex-specific tool called MakeSolo converts the class file into a cardlet ready for downloading with another tool from the Cyberflex development kit.

4.7 Host interface

The Cyberflex Access card includes an ISO 7816-3 [9] interface. We use this framing protocol instead of implementing a more conventional serial protocol such as SLIP or PPP. A daemon running on OpenBSD encapsulates (or *tunnels*) IP packets in 7816 APDUs, with no additional headers or processing.

The maximum size of an APDU is 256 bytes. The tunnel daemon does not implement IP fragmentation, and truncates any packet too big to fit in an APDU. The source code for the OpenBSD tunnel daemon can be found on the Webcard home page.

Each incoming packet results in at most one reply packet. Cyberflex Access supports 7816-3 T=0 protocol, so the reply packet is retrieved by the daemon with a "get response" APDU.

Routing packets to the Webcard requires external advertisement of the existence of the tunnel. At CITI, we assign the Webcard an otherwise unused IP address from the local subnet's address space and install a static route on our upstream router. On the host to which the card reader is attached, we configure with the following commands:

```
# configure the tunnel
ifconfig tun0 141.211.169.2 smarty
# route through the tunnel
route add smarty 141.211.169.2
# start the tunnel daemon
ip7816d 141.211.169.2
```

4.8 Physical characteristics

The physical dimensions of Webcard correspond to ISO 7810 ID-1: 85.6 x 54 x .76 mm. [7]. Of this, roughly 10 x 12 mm is chip carrier. The chip itself is less than 25 square mm. in size.

5 Discussion

Webcard performance is less than spectacular: approximately 130 bytes per second. We believe this can be accounted for in the main by code path through the JVM. We plan to address performance issues when we are satisfied with functionality.

We are participating in an IETF-governed standardization effort to provide for interoperability among Internet smartcard developers. An RFC describing IP encapsulation in ISO 7816-3 has been drafted and submitted to the IETF for consideration and development [10]. Our Webcard implementation complies with the first draft of the RFC.

We intend to extend the functionality of Webcard in many directions, but are mostly concerned with providing better HTTP, TCP, and IP compliance. Our first priority is to address "hosts requirements" such as ICMP functionality, which proves useful in remotely diagnosing problems with IP.

With a more functional TCP/IP stack in hand, we plan to investigate the potential of remote method invocations from host applications. We are also interested in investigating IPv6 and mobile IP for the flexibility they offer to the highly mobile computers embedded in smartcards.

6 Availability

A Webcard demonstration, which includes the Java source code and an image of the card, is at `http://smarty.citi.umich.edu/`. Complete source code for the cardlet, tunnel daemon, and I/O libraries can be found on CITI's smartcard home page, `www.citi.umich.edu/projects/smartcard`.

7 Acknowledgments

We thank Scott Guthery, Tim Jurgensen, and Bertrand du Castel for valuable advice and suggestions. This work was partially supported by Schlumberger, Inc.

References

[1] T. Berners-Lee, R. Fielding, and H. Frystyk. RFC 1945: Hypertext transfer protocol - HTTP/1.0, May 1996.

[2] R. Droms. RFC 2131: Dynamic host configuration protocol, March 1997.

[3] J. Postel (Ed.). RFC 791: Internet protocol - DARPA Internet program protocol specification, September 1981.

[4] J. Postel (Ed.). RFC 793: Transmission control protocol - DARPA Internet program protocol specification, September 1981.

[5] P. Vixie (Ed.), S. Thomson, Y. Rekhter, and J. Bound. RFC 2136: Dynamic updates in the domain name system (DNS UPDATE), April 1997.

[6] R. Finlayson, T. Mann, J. J. Mogul, and M. Theimer. RFC 903: A reverse address resolution protocol, June 1984.

[7] International Organization for Standardization. ISO/IEC 7810: Identification cards – Physical characteristics, 1995.

[8] International Organization for Standardization. ISO/IEC 7816-4: Integrated circuit(s) cards with contacts. Part 4: Interindustry commands for interchange, 1995.

[9] International Organization for Standardization. ISO/IEC 7816-3: Integrated circuit(s) cards with contacts. Part 3: Electronic signals and transmission protocols, 1997.

[10] S. Guthery, Y. Baudoin, J. Posegga, and J. Rees. IP and ARP over ISO 7816-3, February 2000.

[11] Naomaru Itoi, Peter Honeyman, and Jim Rees. SCFS: A unix filesystem for smartcards. In *Proc. USENIX Workshop on Smartcard Technology*, Chicago, May 1999.

[12] Sun Microsystems. Java Card 2.0 programming concepts, October 1997.

[13] Sun Microsystems. Java Card applet developer's guide, July 1998.

[14] Europay International S.A., MasterCard International Inc., and Visa International Service Assoc. EMV '96 - Integrated circuit card specification for payment systems, May 1998.

[15] Schlumberger. Cyberflex access programmer's guide, 1998.