Smartcard Firewalls Revisited

Henrich C. Pöhls and Joachim Posegga

Universität Hamburg, FB Informatik,
Sicherheit in Verteilten Systemen (SVS),
Vogt-Kölln-Str. 30, D-22527 Hamburg
svs-office@informatik.uni-hamburg.de

Abstract. Smartcards are being used as secure endpoints in computer transactions. Recently, the connectivity of smartcards has increased and future smartcards will be able to communicate over the TCP/IP protocol. In this work, we explore options for using a smartcard as an active node in a communication network rather than as an endpoint.

We envision in particular a proxy firewall running on a smartcard and combining the best of both worlds: the smartcard as a secure environment, and the proxy firewall for securing the network. Facilitating the various security options smartcards offer, we show how to design a secure network firewall on a smartcard. We illustrate the usefulness of such a device in several scenarios.

Life was simple before World War II.
After that, we had systems.

Rear Admiral Grace Murray Hopper

1 Introduction

Smartcards of the latest generation are becoming "network citizens" [13], they are able to participate natively in TCP/IP based networks and possess their own implementation of a TCP/IP stack [12, 6, 5]. We will refer to these as *networked smartcards* throughout the paper.

The core idea of this paper is to design a network firewall within a networked smartcard and route TCP/IP traffic between a single host system and the Internet through this card. Note that this differs from what is known as "applet firewalls" in Javacard[1]: this is a software feature of the Java Card platform to isolate Java objects within the card. Our approach instead suggests that the whole card works as a network firewall for a single network host – a personalized firewall on a smartcard.

We will illustrate the concept of a firewall on a smartcard and provide design ideas on how the concept can be implemented on networked smartcards that will emerge in the near future. An implementation of a firewall on a networked smartcard itself is not documented at this stage, as the cards themselves are still prototypes and hardly available outside the labs of card manufacturers.

J. Domingo-Ferrer, J. Posegga, and D. Schreckling (Eds.): CARDIS 2006, LNCS 3928, pp. 179–191, 2006.

A firewall on a smartcard would be a small device that has a smartcard with two Ethernet connectors; each one provides its own TCP/IP stack, natively implemented on the smartcard. Today's networked smartcard prototypes are offering just one TCP/IP connection, usually connected over Universal Serial Bus (USB); having two separate connections, however, is technically feasible. We use an additional layer of routing on an intermediate system, to overcome the current restriction of a single network interface.

Our usage model of a smartcard as a network firewall differs from the predominant usage model of smart cards: Instead of the card being an end-point of a communication, we envision the networked smartcard as a transparent network node that acts as a traffic filtering node. Let us consider this more closely.

Cheswick, Bellovin and Rubin define a firewall as "a collection of components placed between two networks, that collectively have the following properties:

- All traffic from inside to outside, and vice-versa, must pass through the firewall.
- Only authorized traffic, as defined by the local security policy, will be allowed to pass.
- The firewall itself is immune to penetration." [17, page 13]

Implementing such firewall functionality on a smart card allows us to take advantage of the secure environment a card provides, thus firewall functions executed in a smartcard environment are better protected than those running on conventional platforms built upon complex operating systems.

The security environment offered by smartcards therefore allows us to come closer to the third property in the above list.

Furthermore, Smartcards provide additional security features that can be facilitated by a firewall running on a smartcard:

- Secure storage for firewall rule sets,
- storage for cryptographic credentials (e.g. certificates) used for network login,
- cryptographic functions to validate credentials or to cryptographically protect network communication (e.g. VPN connections).

We also physically separate the firewall from the host system: This has, among others, the following advantages:

- It eliminates the need to handle security critical firewall functionality in a potentially untrusted host system.
- The "mobility" of the smartcard allows to easily move a user's "personalized" firewall from one host PC to another.
- The "physical" form factor of a firewall in a smart card can make the use of a firewall more comprehensible for the average user[1].

[1] It will show its presence/absence more clearly.

1.1 Paper Outline

Our paper is organized as follows:

We first consider the network architecture needed for a network firewall (Section 2) and show how this can be implemented on future smartcards even though they will only feature a single network connection. Further, we suggest a design for a simple firewall that would actually run on a smartcard (Section 3). In Section 4, we condense all the security assumptions made and discuss the level of security reached by a firewall on a smartcard. We sketch a few application scenarios to illustrate scenarios for using a firewall on a smartcard in Section 5. After reviewing related work (Section 6) we draw conclusions of our research in Section 7.

2 Network Architecture

Our goal is to move all (or at least security-relevant) firewall operations from an untrusted host system to a smartcard. To offer network firewall functionality, all incoming (and outgoing) Internet packets need to pass through the card. The firewall on the smartcard will then decide upon routing or discarding packets according to the rule set stored locally in card.

2.1 Emerging "Networked" Smartcard

A networked smartcard natively supports TCP/IP connections by implementing a card-internal TCP/IP stack and it provides a Universal Serial Bus (USB) connector for outward connections [12, 6].

This means no additional hardware or software needs to be installed on the PC: A Plug-and-Play (PnP) aware operating system (OS) will detect the card once it is inserted into the systems USB port. The OS will then automatically install the card as a network device. Depending on the implementation the OS may also route TCP/IP messages from and to the Internet via the system's regular network connection.

According to smartcard vendors, networked smartcards provide a single network connection to the host via USB (at least) through one of the following options [12]:

1. USB → encapsulated Serial → PPP → RAS
2. USB → Remote NDIS[2]
3. USB → CDC[3] Ethernet Emulation Model (EEM)[4] → Ethernet drivers

There might be other options to connect a host to the networked smartcard, but these seem to be most common [12, 6].

[2] Remote NDIS (RNDIS) is a Microsoft specification for network devices on dynamic Plug and Play I/O buses such as USB [11].

[3] Also known as Communications Device Class (short CDC), the USB standard also defines an Ethernet Control Model (short CDC Ethernet)[3], but CDC EEM is newer and seems to be favoured.

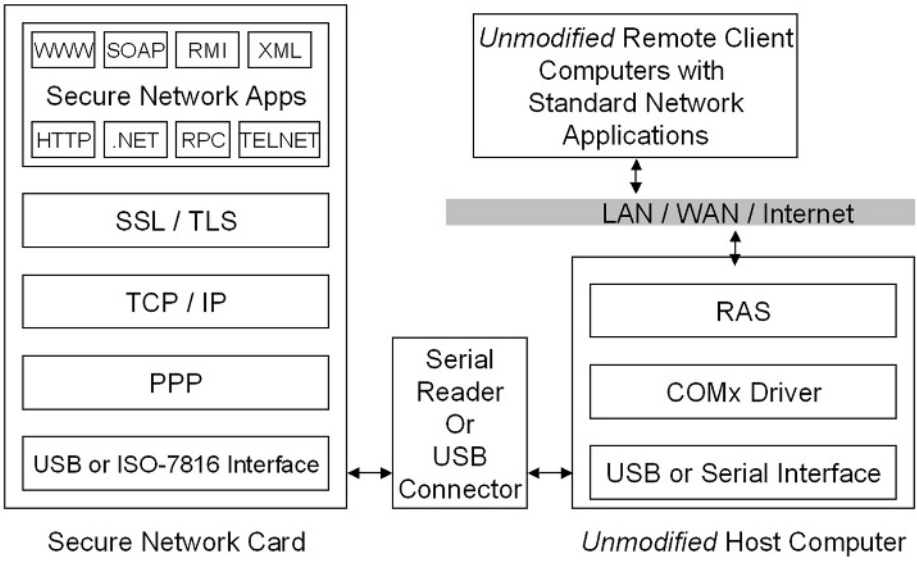

Fig. 1. Secure Network Card Architecture [12]

Figure 1 depicts the general architecture of Schlumberger/Axalto's version of such a networked card: Their smartcard offers TCP/IP functionality and can be connected via USB to a host computer, which can then be used to gain access to a network.

A clear benefit of a network connection over USB is that it requires little or no modifications to the host system as it relies on standards that are (or should be) supported by most operating systems. This provides interoperability and mobility.

Since networked smartcard are extensions of today's non-networked smartcards, an off-the-shelf networked smartcard will likely provide:

- a "Java-based operating system" [6]
- a 32-bit chip [6]
- a minimum of 24k RAM [6]
- a 64 kByte EEPROM
- a USB 2.0 connection [6]

The throughput of the smartcard's network connection is limited by the speed of the underlying USB 2.0 connection: Theoretical performance of USB 2.0 in HiSpeed mode is 480 Mbit/s, but measuring TCP/IP connections over a USB 2.0 port shows actual speeds between 7 Mbit/s and 80 Mbit/s[4]. This provides an estimate of the maximum load the firewall, connected over the USB link, can cope with.

[4] Measured with NETIO[15] using a packet size of 32 kBytes and 4 kBytes on two Windows XP computers connected by a USB ethernet adapter (D-LINK DUB-E100[2]).

2.2 Towards a Firewall on a Networked Smartcard

In order to securely act as a network firewall, all TCP/IP packets must be securely routed through the smartcard and subsequently through the firewall application running on it.

To avoid circumvention the smartcard has to support two network communications: One connection to the untrusted network ("outside") and one to the host ("inside"), often called a "dual homed" system. Rigorously, this can only be achieved if the smartcard provided two physical connections and two separate TCP/IP stacks; off-the-shelf networked smartcards will likely not offer two separate physical connections in the first instance.

To assess firewalls inside a smartcard, we can reduce the restrictions of having two physically separate connections: In order to test a prototype implementation we only need two logically separate network connections to the smartcard.

2.3 Routing Packets Through a Smartcard

To establish two at least logically separated connections to the firewall we use an intermediate system; such a system has three physical interfaces:

- Two physical interfaces, one to the untrusted network (outside), one to the computer that will be protected (inside), and
- one internal interface over the USB connection to the smartcard, which runs the firewall application.

The intermediate system needs to take care that all packets will travel through the smartcard. The networked smartcard uses its single physical connection (TCP/IP-over-USB) and the resulting network adapter on the intermediate system will be assigned two IP addresses to it. Logical separation will then be based on two distinct IP addresses (either source or destination). Figure 2 shows the concept.

This setting hides a "single homed" smartcard with a firewall from the client and the untrusted network: they only have connections to the intermediate system, and this appears "dual homed".

For security reasons we must of course assume reliable and secure routing of packets within the intermediate system. The security-relevant routing (filtering) is, however, not done on the host. The security impact of this decision is analysed in Section 4. For a prototype implementation, this is not the most relevant aspect, since the goal is only to demonstrate that a network firewall on a smartcard is possible and to assess implications thereof. Once an "ideal" smartcard hardware with two physical network interfaces becomes available, the routing can be moved back to the realm of the card with little impact on the software design.

The intermediate system would also allow us to use a non-networked smartcard as well, but we focus on networked smartcards (see Section 4).

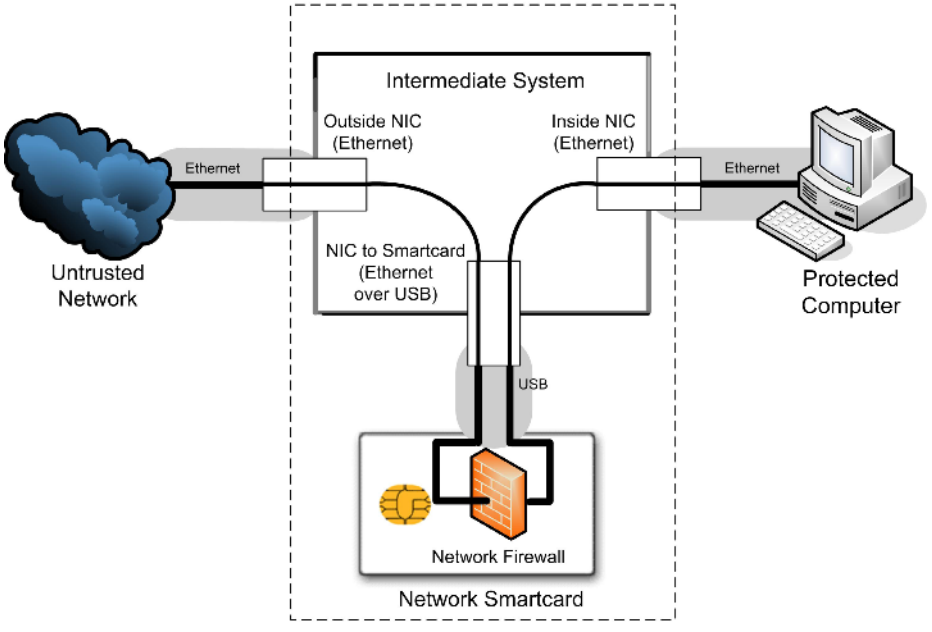

Fig. 2. Single connection over USB to smartcard is hidden by intermediate system

3 Firewall Design

In the previous section we discussed how to overcome the missing two physical network interfaces. We will now present the design of the firewall software.

In order to provide firewall functionality on a per-packet basis (or packet-filter mode) all IP packets arriving at the smartcard need to be inspected by the firewall implementation: it will decide whether to discard or accept a packet according to a local rule set. The firewall must be invoked for each and every packet that is handled by the TCP/IP network stack to enforce this, which is normally achieved by using hooks. Hooks allow for example the Linux firewall "netfilter" (also known as "iptables") to be invoked whenever a packet is received: "Netfilter is a set of hooks inside the Linux kernel that allows kernel modules to register callback functions with the network stack. A registered callback function is then called back for every packet that traverses the respective hook within the network stack"[14].

Firewall code then depends on the possibility to register callback functions. These need to be provided by the network stack's code. In the case of off-the-shelf networked smartcards we do not expect to have such hooks available, and it will not be possible (for non-technical reasons) to modify the TCP/IP stack implementation for the card holder/user. So there is no way to build firewall functionality on a per packet basis.

Without modification of the smartcard's TCP/IP stack code, a firewall must run as an application on the networked smartcard. This has the disadvantage of not being able to catch malformed or unwanted packets on the lower levels (i.e. TTL or IP-Flags). The firewall serves as a proxy (proxy-mode), so decisions can be based on both IP addresses (source and destination), both ports and potentially the contents of the network protocol data.

This functionality can be programmed using sockets, and we assume that socket functionality is available on a networked smartcard. Assuming further a Java-based OS, there will be functions like Socket, ServerSocket or DatagramSocket as in the Java package java.net [16]. Using sockets we are able to design a firewall on a TCP/UDP proxy level.

The procedure for proxying HTTP is straightforward:

1. Opening a listening proxy on the service's port that is to be controlled (e.g. port 80 for HTTP)
2. Listening for an incoming connection and waiting for a HTTP request (HTTP version 1.1)
3. On request: Analysing the request to find the server the client wishes to connect to
4. Filtering of the request according to the firewall rule set
5. If the request is allowed: Opening of a new socket connecting to the server and forwarding the HTTP request
6. Receiving the server's answer
7. Filtering the answer according to the firewall rule set
8. If the answer is allowed: Forwarding the server's answer to the client using the existing connection to the client

The firewall will close/abort the connection to the client or to the server if the request or the answer is denied.

For a fully operational HTTP proxy the above code outline needs to be extended with the following additional features (which can be omitted in a concept prototyping[5]):

1. Resolving host names (found in requests) to IP addresses (used to send the request to the server) by DNS lookups.
2. Handling of multiple concurrent connections.
3. Handling server responses (ICMP destination unreachable) elegantly for the requesting client.

3.1 Smartcard Security Features

Following the "best of both worlds"-approach we want to build upon the security functions offered by smartcards as often as possible (see also Section 4); for example:

[5] The first two features will be limited by the number of concurrent connections offered by the smartcard's TCP/IP stack. The additional features will also increase the amount of memory needed.

The rule set is stored in a file protected by the smartcard. Using command-oriented access conditions "write" or "update" commands are only executed if the administrative credential (or: PIN) is presented to the card. This enforces authorization for rule set changes.

In the case of an HTTP proxy, SSL-secured connections provide an easy starting ground for the use of the smartcard's functionality. For example the validation of the server credential (SSL certificate) can be left to the smartcard's cryptographic functions. As a next step, the secure storage of the smartcard could hold the user's key and certificate for mutual authentication. The smartcard will only use the user's key to authorise to the server after the user has been authenticated/authorized.

3.2 Towards Running Proxy Code on a Networked Smartcard

The proxy code can be implemented for HTTP connections and extended to HTTPS to make use of the aforementioned smartcard security functions. In a first simple implementation the proxy can be written as a single thread, opening a maximum of two concurrent connections. The actual code is lean and our assumed technical specifications (see Section 2.1) make it feasible to implement this design on a networked smartcard.

To provide security for the user's network connections the firewall in a smartcard needed to provide proxy functionality for all protocols used by the user's client. This ranges from simple variants like HTTP to more complex protocols as FTP. The more complex protocols will require more connections, more logic, and thus consume more memory (both for the code and during runtime). As a next step we plan to implement a simple proxy firewall (for IP address-based HTTP requests) on a networked smartcard prototype.

4 Security - Assumptions and Gains

In this section we will shortly summarize security decisions and assumptions we made throughout the previous sections, to highlight the overall security reached by a network firewall on a smartcard.

4.1 Security Assumptions

We showed that the lack of a second interface can be mitigated by an intermediate system. The following security properties of this intermediate system were assumed:

- The OS of the intermediate system reliably routes all network packets through the smartcard. No packets can travel directly from the inside to the outside interface or vice versa.
- The intermediate system provides at least tamper evidence, so that an educated user can detect that the intermediate device has been manipulated.

– The intermediate system is small, so it is easy to carry the firewall system (smartcard together with intermediate system). Small firewall appliances are already available – not based on smartcards though (more details in Section 5). This physical aspect also means that intermediate systems are not shared: Instead smartcard and intermediate system are given to the user from the same trusted authority.

Smartcards are secure and tamper-resistant computing environments. Additionally to the security properties usually credited to smartcards, we make some assumptions on the smartcard's network connection:

– An attacker is not able to attack the smartcard's OS or a running application in the card by sending maliciously formed packets to the network interface. The TCP/IP stack is robust and secure.
– The firewall is the only application running on the card.
– The smartcard's TCP/IP implementation allows the smartcard to receive arbitrary, but standard-conform, network packets and will transfer them to the firewall code.
– For a firewall in proxy-mode: The smartcard's OS handles packets not addressed to a listening port on the smartcard in a secure fashion (i.e. drop/reject the packet). This makes attacks or connections at lower IP levels impossible.
– For a firewall in packetfilter-mode: The smartcard's OS provides "hooks"[6] that allow the firewall to intercept network packets at IP level. Once an application registered with such a hook all packets are handed over to the application and further processing is delayed.

4.2 Security of Firewall on Smartcard

Under the above assumptions the firewall running on the networked smartcard can offer increased security compared to firewalls embedded in traditional computers and operating systems. The secure computing environment provided by the smartcard increases the security of the firewall. It physically separates it from the host system, and provides additional security functionality. Furthermore, a networked smartcard with an embedded TCP/IP stack means increased security of the firewall's TCP/IP handling, as it runs in a secure environment. In the ideal case, all routing of packets through the firewall would be carried out by the trusted smartcard itself using two physical interfaces. But the intermediate system needs only very limited functionality, as it barely acts as a router. This can be implemented more securely than on the general purpose computer that is behind the firewall. However, the role of the intermediate system is solely for prototyping, to show that an implementation of a network firewall on a smartcard is feasible.

As assumed, malformed network traffic is correctly handled by the underlying network stack. So, the firewall only needs to handle packets conforming to

[6] A hook allows registering callback functions.

standards. Especially the proxy firewall code will only receive TCP/UDP connections on the ports it is listening to. The proxy firewall will check whether the rule set allows or forbids such a connection from the TCP/IP information available. It will then additionally be able to inspect the data part if it is a correctly formed request. All packets that are not addressed to the service's port that the proxy firewall listens to, are automatically and securely discarded. This allows to restrict the services that are allowed, and it makes the code leaner and more clearly, thus reducing software or configuration errors.

The rule set is needed to make the firewall's decision. Storing this rule set in a file protected by the smartcard using command-oriented access conditions limits the access to this rule set. Only if the administrative credential (PIN) is presented to the smartcard "write" or "update" commands are allowed. Thus, authorization for rule set changes can be enforced. The smartcard's access control can also be used to restrict certain connections: Either restricting the service as such, by providing access control on the opening of a socket for listening. Or restricting connections to certain servers by controlling the access to less restrictive rule sets.

We can also envision the use of the smartcard firewall for securing connections: As a secure network node, the smartcard can ensure that the traffic that traverses it is additionally secured or validated. This involves the validation and use of credentials (SSL certificates, encryption Keys), which can again be stored in protected files in the smartcard. Finally, the smartcard's cryptographic functions can care for encryption, decryption, certificate verification, signing, etc.. In the simplest case, the user's connection is "proxied" via SSL to the server.

5 Possible Applications

This section sketches a few possible applications of a firewall on a smartcard. They are discussed separately, to highlight certain aspects, but could be combined into one firewall on smartcard. The focus of this section is to motivate the application of networked smartcards as a secure network node, it does not present market-ready applications. Some of the presented scenarios likely require more processing or memory power than today's smartcards offer, but advances in hardware will make them implementable in the future.

5.1 Portable Firewall Box

To overcome the problem of the lacking separate hardware connections, and to further increase the security and portability of a firewall on a smartcard, an embedded system can perform the operations of the intermediate system. Such an embedded system would come with two Ethernet ports (RJ45 connectors) for the network connections and a smartcard reader. It would care for routing network packets and would provide a "dual homed" system, hiding the smartcard from the host. This would prevent a malicious (manipulated) host from modifying the routing of packets and circumventing the firewall. The device could be powered

by batteries, an external power-supply (perhaps from a free USB-port), or it would rely upon power-over-ethernet.

There are small, portable security devices on the market: Examples are the mGuard smart from Innominate [10] and ZyXEL's ZyWALL P1 [18]. The approach of using a smartcard as a platform for the firewall enhances the security and could reduce the device's size further.

5.2 Managed Firewall for Mobile Access or Personal Use

The firewall's rule set can be stored on the smartcard and the smartcard's access control allows us to restrict access in such a way, that the rule set can be altered solely by authorized principals. In this application scenario the firewall rule set is centrally managed.

One option is central management using a policy server to deploy secure mobile access to certain servers, e.g. for managing access of a mobile workforce in a company. The network connection between the smartcard and the central policy server can be protected with SSL and mutual authentication. Every time the firewall is connected to the network, it first tries to connect to the policy server to download, verify, and install the latest firewall rule set. In such a way a company can enforce its security policy even for mobile clients.

For personal use, the customization of the firewall's rule could be provided as a service to "end"-users. This meant secure firewall configuration without the need of local configuration. The need for customised configurations could be indicated on a Web page presented by the smartcard. The smartcard would then forward the request to the service provider, who exclusively maintains the card-internal firewall's rule set.

5.3 Secure Remote Network Access

The mobile client might already use Virtual Private Networks (VPN) and certificates to authenticate to the servers.

The smartcard's firewall application could establish a secure tunnel for remotely accessing e.g. a corporate network. The user's credentials can be securely stored on the smartcard. The user would need to connect to the smartcard with a Web browser and enter the password to unlock her credentials. The smartcard's firewall application would then mutually authenticate to the server and establish session keys for an encrypted tunnel. All signing, encrypting, and certificate-checking during this process can done by the firewall in the smartcard. Thus, the networked smartcard acts as a secure VPN gateway. All this is carried out outside the user's computer, and the use of a VPN is transparent to the user, the user's operating system, and the user's applications.

6 Related Work

Related work on network connected smartcards either concentrates on the implementation of network capabilities in the smartcard, or on the implementation of servers running inside smartcard.

Honeyman and Rees showed in [9] that smartcards can indeed become part of the network: "The Webcard is a TCP/IP stack and web server written in Java that runs on a Schlumberger Cyberlfex card; the card is connected to the Internet via an ISO 7816 T=0 serial link at 55.8 Kbps. The card terminal is connected to an OpenBSD server running a simple daemon that forwards packets between the card and the Internet via a tunnel device. All ip, tcp, and http processing is handled by the card, and all web content is stored on the card." [9]

Guthery, Kehr, and Posegga [7, 8] have presented a related approach where a Web server in a GSM SIM smartcard provided services to he Internet.

Muller and Deschamps [13] showed that smartcards can be networked and act as connection endpoint as either clients or servers.

Our usage of a network-capable smartcard is different in that we go beyond the point of being a connection endpoint: we also consider cards as part of a network infrastructure. Our firewall on a smartcard provides network functionality as a network node, rather than being an endpoint.

7 Conclusion and Outlook

TCP/IP stacks will be an integral part of tomorrows smatcards, turning the cards into network nodes; the consequences of this, both in terms of applications using smartcards, as well as in terms of security implications for the usage of cards are still to be explored[7].

Whilst most approaches we have encountered so far consider networked smartcards as communication end points, we took the concept further and considered smartcards as part of a network infrastructure. Consequently, we suggest to implement security-critical applications on such cards, for instance a network firewall, which is what we explored in this paper.

Our approach combines the security of a smartcard environment and the network security offered by firewalls: The smartcard provides a high security platform for the firewall to run on, and the firewall protects a network "behind" the smartcard.

TCP/IP stacks are part of future smartcards and so the smartcard is facilitated to provide security for network connections. Furthermore, we envision that the secure storage of credentials and the cryptographic functions of a smartcard provide a strong basis for network security devices.

Our paper introduced the the design of a proxy firewall that run as an application on a network smartcard without modification of the smartcard's network stack. Lower levels of a firewall would require modifications to the networked smartcard's TCP/IP stack implementation; still a proxy allows restricting network traffic: As it is located at the highest layer in the protocol stack it even allows filtering unwanted content and access control based on user authentication. Under the assumption that the underlying network stack is not vulnerable, a highly secure implementation of a proxy firewall is possible.

[7] As an example: The concept of proximity between a card and the card holder will be gone, since traditional routing of TCP/IP packets does not care about it.

There are obvious limitations of our approach, one is bandwidth to the (USB-) smartcard, another is the lack of a second network interface in the upcoming generation of networked smartcards. Both restrictions are likely to vanish over time with advances in technology, but we believe that even the current restrictions allow for reasonable applications.

References

1. Zhiqun Chen. *Java Card Technology for Smart Cards: Architecture and Programmer's Guide.* Addison-Wesley, 2000.
2. D-Link. Usb 2.0 fast ethernet adapter dub-e100. `www.dlink.com/products/?model=DUB-E100`.
3. USB Implementers Forum. Universal serial bus class definitions for communication devices. `www.usb.org/developers/devclass_docs/usbcdc11.pdf`, January 1999.
4. USB Implementers Forum. Universal serial bus communications class subclass specification for ethernet emulation model devices rev. 1.0, February 2005.
5. Gemplus. Press release: Gemplus paves the way for future java card platform. `www.gemplus.com/press/archives/2005/rd/27-06-2005-javaone.html`, June 2005.
6. Giesecke & Devrient GmbH. Internet smart card. `www.gi-de.com/portal/page?_pageid=36,53930&_dad=portal&_schema=PORTAL`.
7. Scott Guthery, Roger Kehr, and Joachim Posegga. How to turn a GSM SIM into a web server. In Josep Domingo-Ferrer, David Chan, and Anthony Watson, editors, *Proc. IFIP Fourth Working Conference on Smart Card Research and Applications (CARDIS 2000).* Kluwer Academic Publishers, 2000.
8. Scott Guthery, Roger Kehr, Joachim Posegga, and Harald Vogt. GSM SIMs as Web servers. In *Seventh Intern. Conf. on Intelligence in Services and Networks*, Athens, Greece, Februar 2000. Short Paper.
9. Peter Honeyman and Jim Rees. Webcard: a java card web server. In Josep Domingo-Ferrer, David Chan, and Anthony Watson, editors, *Proc. IFIP Fourth Working Conference on Smart Card Research and Applications (CARDIS 2000).* Kluwer Academic Publishers, 2000.
10. Innominate. Datasheet: mguard smart. `www.innominate.com/images/stories/documents/datasheets/db_smart_en.pdf`, 2005.
11. Microsoft. Ndis - network driver interface specification. `www.microsoft.com/whdc/device/network/ndis/default.mspx`.
12. Michael Montgomery, Asad Ali, and Karen Lu. Secure network card - implementation of a standard network stack in a smart card. In *Proc. IFIP Fourth Working Conference on Smart Card Research and Applications (CARDIS 2000).* Kluwer Academic Publishers, 2000.
13. Christophe Muller and Eric Deschamps. Smart cards as first-class network citizens. 4th Gemplus Developer Conference, Singapore, November 2002.
14. netfilter. website. `www.netfilter.org`.
15. Kai Uwe Rommel. Netio - network throughput benchmark, version 1.14, 1997.
16. SUN. Package java.net description. `java.sun.com/j2se/1.4.2/docs/api/java/net/package-summary.html`.
17. Aviel D. Rubin William R. Cheswick, Steven M. Bellovin. *Firewalls and Internet Security 2nd ed.* Addison Wesley, 2003.
18. ZyXEL. Datasheet: Zywall p1. `ftp://ftp.zyxel.com/ZyWALLP1/document/ZyWALLP1_v2.0_Datasheet.pdf`, March 2005.