Security On the Move: Indirect Authentication Using Kerberos

Armando Fox and Steven Gribble
University of California at Berkeley
{fox,gribble}@cs.berkeley.edu

Abstract

Even as mobile computing and network computing are gaining momentum, Internet security is sharing the spotlight. Security and authentication on open networks is already a difficult problem, even without the additional risks posed by wireless media and the additional software constraints imposed by mobile computing devices with capabilities more modest than those of full-blown laptops. We describe an implemented indirect protocol called Charon, which provides authentication and secure communication to clients by leveraging the strong protocol and deployed infrastructure of Kerberos IV. Charon consists of a portable proxy module that runs as untrusted, unprivileged code, and an extremely lightweight client module that runs quite efficiently even on our Sony MagicLink PDA. This partitioning of functionality makes Charon attractive for ISP's and network computing as well as existing mobile devices. Charon's security is at least as strong as that of Kerberos--the user's password never leaves the mobile device, and Charon cannot obtain Kerberos services for the user without the user's explicit cooperation on each request. In effect Charon allows the mobile device to function as a ***smart card****. We describe our implementation of the protocol and a sample secure* ***rlogin*** *application, and also describe how Charon can be used to implement cross-roaming agreements between mobile computing domains.*

1 Introduction

1.1 Security, Open Networks, and Mobile Computing

The explosive growth of the Internet and the World Wide Web has fueled the image of personal digital assistants (PDA's), laptop computers, and mobile devices of all kinds as valuable "information retrieval" devices, rather than stand-alone "islands" of computation. Surveys have shown that mobile users want *access* to the Internet and their desktop, not a new "killer app" for mobile devices. Wireless infrastructure is rapidly being deployed to enable "untethered" mobile computing [2].

Unfortunately, security and authentication in unprotected networks such as the Internet is a difficult problem [3, 1, 4], and the wireless medium is physically easier to compromise than wired media. Much attention has been focused recently on securing this link [5, 6, 7]. However, link-level encryption and authentication solves only part of the problem. Users still need to authenticate themselves to *services* and servers higher than this level. Users of mobile computers will want access to resources in a visited mobile environment, on the strength of credentials validated by their home environment. Users of network computing devices [8], and the service providers to which those devices connect, will both want authentication for connection to the service and privacy when retrieving content, especially sensitive content such as email. Even users of pagers and similar communication devices can benefit from a mutually-authenticated, secure communication channel, if the mechanism is lightweight enough that it is not cumbersome to implement on such devices.

1.2 Charon: Indirect Authentication and Secure Communication

In this paper we present Charon, an implemented protocol based on Kerberos [9] for *indirect authentication* and secure communications with PDA-class mobile devices. By indirect, we mean that most of the computational resources needed to conduct the authentication protocol and establish a secure channel are located at a *proxy*, a process running on a desktop workstation in the wired infrastructure. This approach simplifies the client software considerably.

Charon provides three important benefits to its clients:

- A means of authenticating themselves to a service attachment point using a Kerberos-based protocol;
- A secure communication channel to that attachment point, at least as strong as those provided by Kerberos;
- access to two-way-authenticated Kerberized services (in an existing Kerberos infrastructure).

All three benefits are realized without requiring the full Kerberos library to be implemented on the client: instead the functionality is *partitioned* between the client and the proxy.

The idea of proxied service has been applied in many scenarios [10, 11, 12, 13]. The application architecture of the Daedalus wireless networking project [14], called GloMop, is built around the idea of proxied services; Charon was originally implemented in this context, although its applicability is much farther-reaching and its implementation essentially independent of the GloMop architecture.

The paper is organized as follows. In the remainder of this section we motivate our approach by showing why PDA's, network computers, smart personal communicators (such as the GeoWorks-based Nokia 9000 [15]), and similar devices represent an interesting design point, despite placing significant constraints on software development. We

describe why Kerberos is a desirable mechanism but difficult to port to such clients, and how our approach circumvents these constraints. Section 2 describes the Charon protocol in detail, pointing out how it differs from the unmodified Kerberos protocol, and describes the models of trust and secure access Charon provides to the client. Section 3 describes our implementation of Charon (both client and proxy components) and a sample *krlogin* application on a Sony MagicLink PDA, whose characteristics are representative of the clients we are targeting. Section 4 compares Charon with Kerberos in terms of resistance to a variety of attacks. Section 5 comments on the role of Charon within larger mobile computing and proxy-based computing projects, and how it instantiates some ideas for enhanced security presented in the original work on Kerberos.

1.3 PDA's and Network Computers

Mobile computing devices span a spectrum across which computational power and local resources are traded off for portability, high availability, and long battery life. At one end of this spectrum are laptops, which are really portable desktop computers—they have CPU's of comparable ability and significant local memory and persistent storage. Laptops pay for these resources in physical size, weight (typically 3-6 pounds), and poor battery life (typically 2-5 hours). At the other end of the spectrum are devices such as the InfoPad [16], which function as mobile terminals and have virtually no local computational power or storage capability. Such devices are lighter and offer much higher battery life, but they are useless without extensive network and computing support infrastructure.

Somewhere between these extremes are PDA's and their future cousins, network computers [8]. Both have moderate computational and memory resources and offer significant simplicity and cost advantages compared to laptops; PDA's typically store applications and data in persistent memory and can operate stand-alone, whereas NC's rely on a network infrastructure for access to both content and applications in portable languages such as Java [17]. NC's in particular are expected to comprise an increasingly large target audience of users (and service providers) who will certainly be concerned about authenticated and secure access to the network infrastructure.

1.4 Why Kerberos?

Kerberos [9] is a time-tested, widely-deployed system that provides authentication and the establishment of secure channels in open networks. Kerberos clients authenticate themselves to servers by presenting *tickets* for each service. Tickets are distributed by a central trusted server within each administrative domain, and are constructed so that only clients possessing the appropriate key(s) are able to decrypt and use them. Kerberos includes specific features to prevent forgery of client or server identity, detect replay attacks [3], establish secure channels between endpoints through safe distribution of temporary session keys, and minimize the likelihood that the user's Kerberos password will be compromised (it never leaves the user's workstation, and all traces of it are destroyed once the user has authenticated herself). The strengths and weaknesses of Kerberos are analyzed in detail in [1].

The infrastructure necessary to support Kerberos has been deployed within the administrative domains of many sites. Presumably the administrators of these domains have taken reasonable precautions to secure the "trusted" components of Kerberos. By leveraging this infrastructure, we avoid introducing yet another trusted component in the system and assuming the additional burden of proving it secure.

1.5 Porting Kerberos is Out of the Question

Traditional Kerberos clients access Kerberos client-side services by linking against a Kerberos library, *libkrb*, which in turn makes use of an encryption library, *libdes*. Two factors complicate a direct port of Kerberos to PDA's or NC's: limited hardware and software resources and heterogeneous development environments. We describe each in turn.

Today's PDA's lack the memory, computing power, and persistent storage necessary to support the Kerberos library. As a first order approximation, observe that this library is 663KB in size under Solaris 2.4, and the runtime image of *kinit*, a trivial Kerberos client that performs initial user authentication, is 525KB. In contrast, the Sony MagicLink PIC-1000 PDA, whose capabilities are representative of today's devices, has a total of 2MB of SRAM which functions as both the persistent store and a shared heap for all applications, the operating system, and the GUI components.

Indeed, it is meaningless to speak of "porting" most Unix software to today's PDA's: even with more memory and a faster CPU, the development environments for most PDA's are so different from Unix (and from each other) that in many cases we may only speak of *rewriting* the software to achieve the same functionality. Unix idioms do not apply; Unix-like raw I/O facilities are usually not provided; and the standard set of Unix libraries implementing kernel abstractions such as memory allocation services and standard I/O are absent.

Kerberos is particularly problematic, and has a reputation among system administrators of being notoriously difficult to compile even on Unix systems [18]. Besides its size and complexity (9,000 lines of C source), it relies heavily on Unix idioms and libraries, use of Unix I/O (including Berkeley sockets) and memory services, and knowledge of hardware semantics such as the byte ordering and word-size of the host architecture. None of this functionality is easy to implement or mimic on current PDA class devices, or on most non-UNIX systems for that matter.

The Charon indirect authentication protocol addresses these limitations by *displacing* complexity from the client to

the proxy: in Charon, only a DES encryption routine—relatively portable and requiring no outside library support—runs on the client, and the Kerberos library runs on the proxy.

2 Charon Design

2.1 Charon Security Goals and Assumptions

The security goals of Charon are as follows:

1. Charon may be trusted with temporary session keys for particular services it contacts on the client's behalf, but not with the user's Kerberos password or with sufficient information to impersonate the user in case Charon is compromised. I.e. each time an additional service is requested, the client's cooperation is required to gain access.
2. If Charon is undermined, the worst the attackers can do is obtain valid tickets for a currently-accessed Kerberized service. These tickets can be given short lifetimes to minimize the damage attackers can do.
3. Eavesdropping on the PDA-to-proxy connection should be no more useful than eavesdropping on any (unprotected) network connection in a Kerberos infrastructure.

Charon assumes the following about its environment and usage model:

1. The user wishing to authenticate herself has a Kerberos password in her home *realm* (Kerberos administrative domain).
2. Some workstation that the client can easily contact is running a proxy that understands the Charon protocol. (In section 3.1 we describe our portable implementation of the proxy side of Charon for Unix workstations.) The proxy is registered as a Kerberized service in the same realm as the user desiring authentication.
3. The user trusts the integrity of the proxy software, as she would trust the integrity of an *rlogin* daemon (e.g.) not to capture and compromise her password.

2.2 Client-to-Proxy Authentication: Overview

This overview of the Charon protocol is designed to allow readers with detailed knowledge of Kerberos to understand the essential features of the protocol without reading the detailed description in section 2.3. Readers desiring a review of the Kerberos protocol should refer to Appendix A.

In Charon, all interaction with the KDC and the TGS is displaced from the client to the proxy: the client communicates exclusively with the proxy, which also implies that the client need not be TCP/IP-capable. The handshake consists of two phases. During the first phase of the handshake (illustrated in figure 1b), the client uses Charon as a smart router to obtain a TGT; from the point of view of the KDC and TGS, Charon appears to be a client. During the second phase of the handshake (figure 1c), the client and Charon cooperate to obtain a service ticket from the TGS, by having the client request a ticket for the Charon Kerberized service. After this two-phase handshake has succeeded, the session key included with the service ticket can be used to secure the channel between the client and Charon. The client can also request access to other Kerberized services (figure 1d).

The other important difference between Charon and Kerberos is that the proxy does not possess all the information necessary to negotiate for services on behalf of the client. Specifically, the client retains the session key between itself and the TGS. This key is required in order to construct the timestamped authenticators (see [9] or Appendix A) that must be presented with each request for service; therefore Charon cannot obtain service on behalf of the client without the client's explicit cooperation.

2.3 Client-to-Proxy Authentication: Detailed Description

In describing the messages exchanged in the Charon protocol, we use the following terminology:

Kerberized service: A service (e.g. *rlogin*) that requires its clients to authenticate themselves using Kerberos, and is capable of authenticating itself to its clients using Kerberos.

Kerberos server: A workstation that hosts one or more Kerberized services. This requires that the credentials for each hosted service be securely installed on the workstation by the Kerberos administrator.

C: the client's principal (i.e. the user). The user knows a password from which the key K_c can be derived.

K_c: the key derived from the client's Kerberos password.

S: the principal corresponding to some Kerberized service, for example, Kerberized *rlogin*.

K_s: The service's secret key, stored locally (and securely) on the server.

KDC: Key Distribution Center, a secure database that knows the secret keys of every Kerberos principal in its *realm* (administrative domain).

TGS: The Kerberos ticket-granting server.

TGT: The initial ticket returned by the TGS which, when decrypted by the client using K_c, can be used to prove the client's identity in future transactions (in conjunction with a valid authenticator) without requiring the principal to supply K_c again.

K_{tgs}: the key of the ticket-granting server (TGS).

A_p: a Kerberos *authenticator* (containing a name, address, and timestamp) for principal P.

$K_{c,tgs}$: the one-time[1] session key (generated by TGS) between client C and the TGS.

$K_{c,s}$: the session key between client C and server S.

$T_{c,s}$: a ticket authorizing client C to access service S.

{x}K: message x encrypted under key K.

Ch: the Charon agent (part of the proxy process) running at the proxy workstation.

P: the proxy service as a Kerberos principal.

K_p: the proxy's secret key, which allows the proxy to authenticate itself as a Kerberos service.

$K_{c,p}$: the session key (generated by the KDC) for use between the client and the proxy.

A summary of the messages exchanged during the first phase of the Charon protocol is as follows. These 4 messages replace messages 1-2 in the unmodified Kerberos protocol.

1. C→Ch: the client indicates its desire to establish secure authenticated channel with Charon.
2. Ch→KDC: Charon requests a TGT on the client's behalf. This message is identical to message 1 of the unmodified Kerberos protocol.
3. KDC→Ch: $\{K_{c,tgs}, \{T_{c,tgs}\}K_{tgs}\}K_c$

   The TGT packet is returned to Charon; this message is identical to message 2 of the unmodified protocol.
4. Ch→C: $\{K_{c,tgs}, \{T_{c,tgs}\}K_{tgs}\}K_c$

   The TGT packet is forwarded verbatim to the client. The client prompts the user for his/her Kerberos password, converts it into a DES key, and uses DES to decrypt the TGT packet. In unmodified Kerberos, the user would instead type the password on the workstation keyboard.

At this point in our protocol, in addition to the client's key K_c, the client possesses the contents of the TGT packet: the TGT itself $\{T_{c,tgs}\}K_{tgs}$, and a session key $K_{c,tgs}$ to use when talking with the TGS. The proxy possesses only the contents of message 3 above (the encrypted TGT packet); in particular it cannot decrypt message 3 (since it does not know K_c), and therefore it cannot possess $K_{c,tgs}$. The client will then request a ticket for the proxy service in the second phase of the handshake:

5. C→Ch: P, $\{T_{c,tgs}\}K_{tgs}$, $\{A_c\}K_{c,tgs}$

   Request ticket for proxy service. The client must construct an authenticator A_c and encrypt it using $K_{c,tgs}$, obtained in step 4. (Compare this with message 3 in the unmodified Kerberos protocol: in that case, the workstation can construct the authenticator directly, since it knows $K_{c,tgs}$.) Since a timestamp is included in the authenticator, the client's clock must be at least loosely synchronized with that of the TGS (to within 5 minutes for Kerberos IV). We do not address in this paper exactly how synchronization should be achieved, but note that a poorly synchronized client-side clock can only result in denied service; it does not weaken security of the system.
6. Ch→TGS: P, $\{T_{c,tgs}\}K_{tgs}$, $\{A_c\}K_{c,tgs}$

   Charon relays the information to the TGS as a well-formed Kerberos request. Note that the forwarded message contains the (encrypted) TGT originally returned in message 3 and the authenticator constructed by the client; both are necessary for the TGS to honor the request. This is an important property: since only the client can construct the authenticator, Charon cannot construct a valid request for a service without the client's cooperation.
7. TGS→Ch: $\{K_{c,p}, \{T_{c,p}\}K_p\}K_{c,tgs}$

   The TGS returns the proxy ticket and a session key for use between the client and the proxy, all encrypted with $K_{c,tgs}$, so that only the client can decrypt this message. Note that there are *two* copies of $K_{c,p}$ in this message: one copy is contained in the encrypted ticket $T_{c,p}$, and only the entity that can decrypt the ticket can get that copy. In this case the entity is Charon, which alone possesses the key K_p.
8. Ch→C: $\{K_{c,p}, \{T_{c,p}\}K_p\}K_{c,tgs}$

   The ticket and key are forwarded verbatim to the client, which decrypts them using $K_{c,tgs}$ and extracts (the first copy of) $K_{c,p}$.
9. C→Ch: $\{T_{c,p}\}K_p$, $\{A_c\}K_{c,p}$

   The client forwards the proxy ticket and a constructed authenticator to Charon. The authenticator proves that the client knows $K_{c,p}$. Since Charon has access to the Kerberos credentials K_p of the proxy, it can decrypt $\{T_{c,p}\}K_p$ and extract $K_{c,p}$ from $T_{c,p}$.

At this point, the following are true:

- Both the client and the proxy are in possession of the session key $K_{c,p}$, which can be used to encrypt further communication between the client and the proxy.
- None of the keys $K_{c,tgs}$, K_c, K_p, and $K_{c,p}$ are ever transmitted in the clear; nor are either of the tickets $T_{c,tgs}$ and $T_{c,p}$.
- Charon believes (see discussion in section 4.5) that the user is who he/she claims to be.

1: Bellovin and Merrick have observed [1] that a Kerberos session key is really a *multi-session key,* since it is used for all transactions with that service for the lifetime of the ticket. They describe a simple change to the protocol that would allow the use of true one-time session keys.

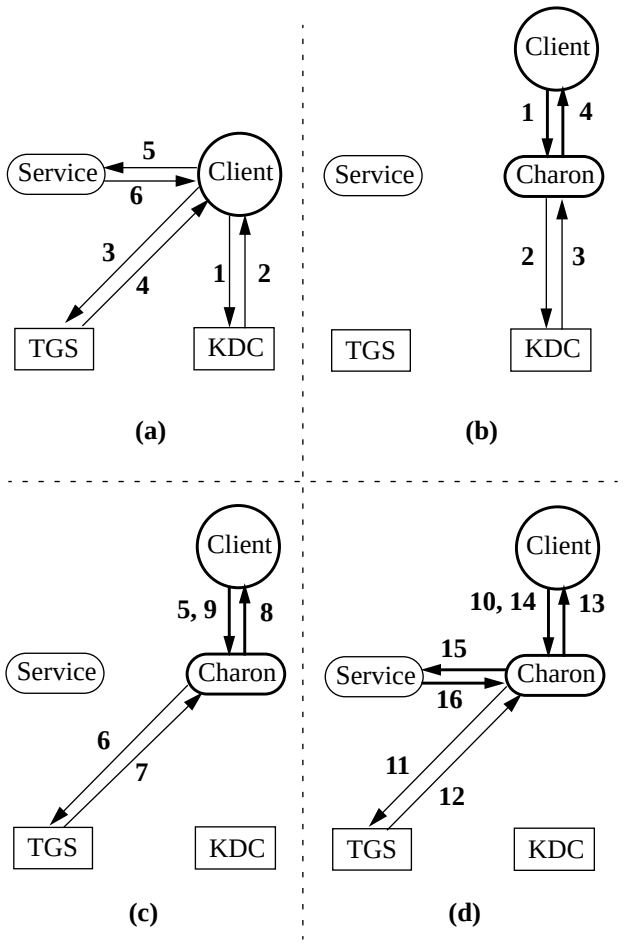

Figure 1: (a) The Kerberos protocol. (b) Obtaining a TGT via Charon. (c) Obtaining a session key for use with Charon. (d) Obtaining a service key via Charon.

- Charon does not know $K_{c,tgs}$.

To access additional Kerberized services, messages similar to 5-9 can be exchanged again to set up a session key between the client and the service (messages 10 through 16 in figure 1d). This session key can then be securely relayed to Charon, which can interact with the service on the client's behalf. This approach requires more work on the client, but it ensures that the only keys Charon ever possesses are session keys; furthermore, it does not have the session key $K_{c,tgs}$ necessary to construct the authenticators necessary for requesting a new service ticket from the TGS or decrypting the returned service ticket, and it also does not have the Kerberos password of the user necessary to obtain the session key $K_{c,tgs}$.

An alternate approach that places more trust in Charon is for the client to reveal to Charon $K_{c,tgs}$ over the established secure channel, thus allowing Charon to negotiate for Kerberized services directly (since it can now construct the authenticators that must accompany the TGT on each request). In this case, Charon still doesn't have the user's Kerberos password, but because it has $K_{c,tgs}$, it can do more damage should it be compromised. Specifically, an attacker who controls $K_{c,tgs}$ can impersonate the client's principal for the lifetime of the TGT, which is specified at the time the TGT is requested but in practice may be several hours.

2.4 End-to-End vs. Proxied Kerberized Services

Once the client has authenticated itself to the proxy, there are two models for client access to Kerberized services, particularly those such as *telnet* that provide an encrypted channel:

- Proxied service: The proxy knows the session key for the service. It decrypts data from the server, optionally performs some transformations on it, and optionally re-encrypts it before passing it to the client.
- End-to-end service: The proxy does *not* know the session key for the service; it acts purely as a router (and possibly network gateway) for the client.

We motivate the proxied service model by pointing to the success of our proxy architecture at addressing client variation with respect to hardware, software and connectivity [19]. For example, we have built (separately) an MH-compatible MIME email client for the Sony MagicLink. This client retrieves MIME email via a proxy which performs *distillation* [20] on included MIME images, optimizing the images for display on the PDA's screen. Clearly, such transformations are impossible unless the proxy has access to decrypted data.

The proxy can, of course, re-encrypt the data (using the client-to-proxy session key $K_{c,p}$ obtained as described in section 2.3) before forwarding it to the client. Re-encryption may be unnecessary if the network can perform separate link-level encryption, as is the case with GSM [6].

Granting the proxy session keys to Kerberized services implies a certain amount of trust in the proxy. We maintain that the required amount of trust is not unreasonable: it is the same trust placed in existing Kerberized services such as *krlogind*.

3 Charon Implementation

3.1 Charon Protocol Module

For rapid prototyping and portability, we have been using Perl 5 [21] as our prototyping language. Accessing the Kerberos IV client-side library *libkrb.a* from Perl required some glue code, which was relatively painless to create due to the extensive facilities Perl 5 provides for exactly this linking process. [We expect to make this software freely available by final revision time.]

Kerberos also makes use of a DES [22] encryption library *libdes.a*. Clients normally do not access the encryption library directly: they compose and decompose encrypted text via calls to the Kerberos library, which in turn calls the

encryption library. Kerberos was specifically designed this way so that the encryption module would be easily replaceable, although in practice all currently deployed implementations of Kerberos IV use DES. As it turns out, the *krlogin* client, which we used as an example application, allows for an encrypted connection by explicitly calling DES for encryption.

Our proxy consists of a set of modules that provide various proxied services to a connected client. The Charon protocol module takes control when a client first connects to the proxy. After authenticating the client, Charon remains available to provide access to Kerberized services and to provide encrypted streams as an option for any service. The Charon module consists of about 500 lines of Perl; a "minimal" proxy, including Charon and all proxy support code but no access to other services, consists of about 1500 lines of Perl.

3.2 Proxy's Kerberos Service Key

Every Kerberized service is required to have a key that it can use to authenticate itself to clients. As explained in section 2, Charon bootstraps a secure channel to the client by treating the proxy as a Kerberized service.

For ease of implementation, we chose to make the proxy a service of type *rcmd*, which is the same type used for existing *rlogin* and *telnet* services on our development machine. The motivation for this decision was that the *rcmd* service key was already stored in *krb.srvtab* on our machine and entered into the KDC; had we chosen to make the proxy a different type of Kerberized service, an appropriate entry would have to have been added to both our machine and the KDC. This process requires intervention by a system administrator, so in the interest of expediency we chose to avoid it.

However, we stress that in general this is a *bad* approach for several reasons:

- The proxy is a fundamentally different Kerberos service from either *rlogin* or *telnet*, and possessing a ticket to use the proxy should not necessarily entitle the client to establish an *rlogin* session—yet in our initial implementation this is exactly the case.
- To read the *rcmd* credentials, the proxy must run as *root*. As we discuss in section 4.6, in most scenarios this constitutes a needless security risk.

We return to administrative issues in section 4.6.

3.3 Client-side implementation on Sony MagicLink

The Charon protocol was designed in such a way that the client-side component must only perform a small set functions and have minimal knowledge of Kerberos protocols and ticket formats. Specifically, the client-side components must be able to do the following:

- Construct authenticators, which have a relatively simple format and include a timestamp.
- Extract session keys from tickets. Session keys happen to be the first 8 bytes of a ticket, so no complex parsing of the ticket structure is necessary by clients.
- Communicate with the proxy over some sort of reliable stream. In our implementation, we used a dial-in connection and a standard LAPM-capable [23] modem, obviating the need for a TCP/IP stack on the client.
- Perform DES encryption and decryption.

The DES library is the most complex component of the client-side Charon implementation. This is a remarkable statement, given the relative complexity of the full Kerberos protocol and implementation. Our design pushes this complexity off of the client while maintaining the security and integrity of Kerberos. The design also completely shields client-side applications from the details of the authentication mechanism.

We chose to implement the client-side components of Charon on a Sony MagicLink PDA [24], which features a relatively rich OS (MagicCap) and a C compiler. Nuances of the MagicCap OS forced us to rewrite much of the *libdes* library. Our implementation of the MagicCap DES library contains about 1200 lines of source code, resulting in a compiled image size of about 30KB. The rest of the Charon client (complete with a modest user interface) required another 750 lines of source (15KB compiled), for a total footprint of approximately 45KB on the PDA—about 9% of the Unix implementation of *kinit*. As we discussed in section 1.5, this economy of size is essential when dealing with PDA class devices.

3.4 Adding Access to Kerberized Services

In traditional Kerberos, a Kerberos client authenticates itself to a server via two Kerberos library functions: `get_ad_tkt` to obtain a service ticket for the service (using an existing TGT), and `krb_sendauth` to present the ticket to the remote server and effect (possibly mutual) authentication. The Charon module provides direct access from Perl to `krb_sendauth`, but it cannot call `get_ad_tkt` directly because that routine tries to use the session key $K_{c,tgs}$ for two operations:

- to construct the authenticator that must accompany any request to the TGS;
- to decrypt the response returned by the TGS.

Recall that as described in section 2.3, $K_{c,tgs}$ is known only to the client, not to Charon. Therefore Charon must ask the client to construct the authenticator to be passed to the TGS, and when the TGS sends a reply, Charon must pass the reply packet to the client for decryption.

To address this difference, Charon provides a function `charon_get_ad_tkt`, which obtains a ticket using the Charon protocol messages 5-9 rather than the unmodified

Kerberos messages 3-4. `charon_get_ad_tkt` is therefore a functional replacement for the standard Kerberos `get_ad_tkt`, and can be called by any proxy module that wants to provide client access to a Kerberized service.

3.5 A Test Application: *krlogin*

As a test application, we modified an existing MagicCap *rlogin* client to take advantage of our Charon protocol. The resulting *krlogin* client is fully Kerberized and can support encrypted traffic. Only trivial modifications were necessary, since the authentication and encryption operations provided by the client-side Charon implementation are completely transparent to the rlogin client.

On the proxy side, access to a Kerberized service is provided via a proxy module that implements the specific service. For example, the *krlogin* module understands how to open a logical stream from a client, and negotiate for a (possibly encrypted) *rlogin* session to a specified host. The *krlogin* module invokes messages 10-16 of the Charon protocol (implemented in the Charon module) to perform mutual Kerberos authentication between the client and the *krlogind* on the target machine.

3.6 "Porting" to Other Devices

A wide variety of devices could potentially benefit from the Charon protocol, including a large number of extremely heterogeneous clients and laptop computers of differing operating systems, computational prowess, and communication abilities. Attempting to design software that can be easily ported across this large class of target devices is difficult at best, and seems intractable for software of any complexity.

Because of these difficulties and the impoverished nature of the targeted client devices, both the Charon protocol and the client-side components required to implement it were specifically designed to be as simple as possible. Rewriting these components on new devices is intended to be a relatively simple task. For example, in our implementation, the file that describes the format and contents of Charon protocol messages between the client and the proxy is actually used as input to a recursive-descent parser, which generates code to parse and verify the integrity of the different messages. [Note to reviewers: implementations of Charon clients in Java [17]and Geos [25] are currently underway. Our experience will allow us to evaluate the above claim.]

3.7 Performance of DES on PDA

Our recent experience porting a GIF renderer [19] and audio stream decoder [26] to the MagicLink device strongly suggested that the limiting factor in Charon performance would be the speed of DES encryption and decryption. We have found that the Charon handshake takes about 2 seconds (using the MagicLink's 2400 baud modem) when talking to a "dummy" test server that generates a canned set of responses for the client, and about 5 seconds when talking to the real Charon module and interacting with Kerberos. This is not surprising since the communication between Charon and the Kerberos KDC and TGS seems to take about 2-3 seconds. These results suggest that the "untrusted Charon" model, in which the client never relinquishes $K_{c,tgs}$ to the proxy but instead constructs an authenticator for every new service request, will yield acceptable performance even on PDA-class devices.

[Note to reviewers: Since the *krlogin* client is still under development, we do not yet know how well DES will perform for interactive *rlogin* traffic. We expect to have these measurements by the final revision deadline.]

4 Charon Security and End-to-End Attacks

This section presents an examination of potential end-to-end attacks in a Charon implementation and attacks on the protocol itself. There are three logical endpoints in our system: the client, the proxy (including Charon), and the Kerberos infrastructure agents (in particular, the TGS and the KDC). Each of these could potentially be compromised by an attacker. We omit discussion of attacks on the Kerberos protocol itself, as these are treated in detail elsewhere [1,27]

4.1 Attacks by Malicious Clients

The greatest strength of the Kerberos architecture is its near immunity to attack by malicious clients; this strength is inherited by the Charon protocol. This strength is important since it is relatively easy to subvert a client in a distributed open system, but the Kerberos architecture prevents such subverted clients from gaining access to services. The only information given freely to a client by both the Kerberos and Charon protocols is a TGT encrypted with both the client's key K_c and the TGS's key K_{tgs}; if the TGT is not properly decrypted by the client, or if an invalid authenticator is returned to Charon in step 5 of our protocol, then the TGS will detect this after step 6 and refuse to grant service tickets. Although a TGT that has been decrypted by a client can be captured by sniffing network traffic, such a TGT is useless without the session key $K_{c,tgs}$ that had accompanied it, as $K_{c,tgs}$ is required to construct an authenticator. Replay attacks that include captured authenticators are also futile because of the timestamp embedded in the authenticator[2] and because the attacker does not possess $K_{c,tgs}$.

4.2 Attacks on the Charon Host Workstation

If the workstation on which the proxy is running is compromised, then it is possible for the attacker to obtain the Kerberos credentials stored in the *krb.srvtab* file. This implies that such the attacker can then obtain all session keys, secret keys, and tickets that Charon ever possesses.

2: There is a well-known Kerberos attack that involves compromising a network time service (see, for instance, [1]). We do not address this attack, but merely assert that our protocol is no worse that Kerberos in this regard.

Fortunately, this includes neither the client's key K_c, nor the session key $K_{c,tgs}$ for constructing authenticators. A compromised Charon proxy workstation implies that services already requested by a client can be commandeered, but new services cannot be initiated by the attacker by impersonating the user.

4.3 Attacks on the File System

We must include as a logical endpoint the file system on which the Charon and proxy executables are stored. Networked filesystem attacks resulting in unauthorized write access to a filesystem [28] or on-the-wire binary patching [29] can result in compromised binaries being executed on the workstation, thus compromising the workstation. We guard against this by running Charon and the proxy from a local file system, trusting the compiler that generated the executables.

A file system attack can be used, for example, to compromise the *kinit* program used to obtain an initial TGT. Users type their Kerberos password into *kinit*; if any part of *kinit* (including the DES encryption library) is replaced with a subverted version via a file system attack, attackers can easily discover a user's Kerberos password and gain access to the system. Charon does not suffer from this problem—the critical *kinit* functionality resides on the client, which is presumed to be a PDA-class device and by nature cannot be subverted by such an attack.

4.4 Timing Out Kerberos Tickets

A further defense against attackers is to request Kerberos tickets with short lifetimes. In the Charon protocol, the client principal indicates to Charon the desired lifetime of each requested ticket. If a ticket expires, a new ticket must be requested.

Note that this approach even applies to "session-oriented" services such as *krlogin*, since the expiration date of a Kerberos ticket indicates the latest date at which it can be presented. Once an *krlogin* session is started, the Kerberos ticket that started it is not used again; therefore the ticket can be immediately destroyed or timed out, to prevent an attacker from using it to open a new *krlogin* session as the user. The attacker could try to take over the existing *krlogin* session by attacking the TCP connection, but without the original *krlogin* ticket, the attacker cannot discover the session key being used to encrypt session traffic.

4.5 Strength of Charon's Protocol

In [27], a logic for examining the integrity of an authentication protocol is described. This logic was used to formally verify the correctness of the Kerberos protocol, and to help extract the list of assumptions that must be made for the protocol to remain valid.

The same proof of the Kerberos protocol from [27] can be used *unmodified* to prove the correctness of the Charon handshake protocol. This is a strong statement - it implies that the same set of assumptions must be made when using Charon that must be made for Kerberos, and that our modifications to the Kerberos protocol do not change its ideal, abstract logical properties. This is intuitive in retrospect; Charon acts as a data tunnel during the first half of the authentication handshake, and as both a data tunnel and a service during the second half of the handshake. When acting as a data tunnel, it can be considered as part of the network infrastructure (from a logical standpoint). Since Kerberos already assumes that the network is untrustworthy, using Charon as a data tunnel (essentially a router) does not affect the strength of the protocol.

If the client decides to relinquish service session keys to the proxy, obviously the security properties and assumptions of the Charon protocol diverge from those of Kerberos. The client must assume that the proxy is trustworthy and uncompromised. Although strictly speaking, this same assumption must be made when clients converse with Kerberized services such as *krlogin*, relinquishing session keys to a proxy means that the proxy could potentially perform actions as the client, but that the client never becomes aware of.

The specific assumptions that must be made about the Kerberos (and therefore Charon) protocol are:

- all principals must trust that the KDC and TGS are not compromised, which implies that the secret keys shared with them are safe
- principals must trust that the KDC and TGS generate good session keys, and that those session keys are never known to anyone besides the KDC, the TGS, and the principals participating in the session
- principals must believe that timestamps embedded in tickets and authenticators are "fresh", i.e. that duplicated or old tickets and authenticators used in a replay attack can be detected and rejected

The fundamental implication of these assumptions is that parties that want to communicate with each other securely must trust each other, and that all parties involved (principals, the KDC, and the TGS) are uncompromised, including the clocks that those parties access.

4.6 Who Does Charon Run As?

There already exist several programs that start up in privileged mode (i.e. setuid to *root*), accept a connection from an arbitrary source, authenticate a user, and then continue to run as that user by changing the effective UID. *Rlogind* is an example of such a program.

Charon (and the proxy) need not use this model. Instead, the proxy can run as an unprivileged fictitious user, say *charon*, who has no login shell or home directory and exists solely for the purpose of running Charon. This is possible because a valid Kerberos tickets is honored regardless of the UID of the presenter. If Charon has obtained a Kerberos *rlo-*

gin ticket for user *joe*, then Charon can present that ticket to start an *rlogin* session as *joe* even if Charon is running as user *charon*. We remind the reader that this configuration does not pose a serious security risk: Charon must also present a valid authenticator, which requires the client's cooperation to construct.

This model works well when all services the user wants to access are Kerberized. If, however, the user wants to access certain non-Kerberized services (for example, her home directory mounted on a non-Kerberized traditional NFS server, which authenticates by UID), our model will fail. One obvious solution is to allow the proxy to run with the user's UID, but in order to set the effective UID the proxy must start out running as *root*—just as current programs like *rlogind* do. Since programs of nontrivial complexity represent inherent security weaknesses which have been successfully exploited [30], we prefer to avoid the traditional method and are still investigating alternate ways to provide legitimate users access to non-Kerberized services.

We point out that this is only an issue when the requested service performs authentication based on UID; it is not an issue for services such as Web access, where the authentication occurs at the application level.

5 Other Applications and Work in Progress

5.1 Authentication for Network Service Providers

Charon can be used to provide relatively lightweight but strong authentication for users connecting to a network service provider. The connection might be an IP connection for accessing conventional Internet services such as the Web, or it might be for two-way paging. Charon can provide users of pagers and similar devices with authenticated, secure communications in both directions. Because the client side of the Charon implementation is lightweight by design, it is reasonable to consider implementing it on such devices.

A good target for such an experiment is the UC Berkeley Office of Telecommunications Services (TCS), which provides dial-up IP access via 14.4 or 28.8 modem to about 20,000 users in the UC Berkeley community. Since low-speed connections make Web surfing painful, we plan to deploy distillation-based Web proxies [20] as a value-added service to end users. Charon can be used to authenticate the users entitled to the service, and if the users so desire, to provide a secure connection to other Internet resources such as their email.

5.2 Charon, Daedalus, and GloMop

The Daedalus project [14] is investigating wireless overlay networks and cross-roaming agreements between administrative domains. Proxied services are provided to wireless clients in the Daedalus system via a proxy architecture called GloMop, which is described extensively elsewhere [31,19]. The Charon module of the GloMop proxy will be used within the Daedalus architecture in two ways:

- to authenticate users of the GloMop proxy in their home domain (Kerberos realm);
- to authenticate users visiting another domain, by verifying that they can authenticate themselves to some home domain with which the visited domain has a pre-established trust agreement.

The second function can be realized by exploiting support in Kerberos for inter-realm authentication. It requires only that a shared secret be established between any pair of KDC's whose Kerberos realms have established cross-roaming agreements. Charon is an attractive option for Daedalus because it is a strong protocol, because it uses a widely deployed authentication infrastructure, and because it requires only minimal changes to each domain's Kerberos database to establish a cross-authentication channel.

5.3 PDA's as Smart Cards

One flaw in the existing Kerberos implementation is that users must type their Kerberos passwords into the *kinit* program to obtain an initial TGT. If these keystrokes are captured (for instance by snooping unprotected traffic to the X server, tampering with keyboard device drivers, or simply peering over somebody's shoulder) then an attacker can impersonate the attacked principal.

With Charon, an attacker would have to compromise software on principals' PDA's in order to capture their Kerberos passwords. PDA's are very personal devices that are rarely left unattended, and they are seldom connected to a network in such a manner than an over-the-network attack is possible. It is therefore much harder to compromise a PDA so as to capture keystrokes from it, and then recompromise it to cause it to transmit the captured keystrokes to an attacker. In a sense, the PDA has become a *smart card* in this capacity. Steiner et al. mention this idea in [9].

5.4 Conclusions

We have presented an implemented protocol, Charon, for authentication and secure communication between a client (PDA-class device or network computer) and a proxy (or service attachment point). Charon boasts the following properties:

- Charon is built on top Kerberos, a widely-deployed, secure, and field-tested authentication system.
- Only DES encryption and decryption need be ported to clients, which typically suffer from limited computing resources and completely dissimilar development environments. We were able to keep our implementation down to 45 KB on the Sony MagicLink PDA, as compared to 525KB for Unix *kinit*.
- The client communicates exclusively with a proxy

that provides indirect authentication and can serve as a network gateway, eliminating the need for the client to implement full TCP/IP.

- Neither the user's Kerberos password nor the key required to construct Kerberos authenticators (to request access to services) ever leave the client, so Charon cannot impersonate the user without her cooperation.
- No sensitive data is transmitted in the clear. In addition to access to encrypted Kerberized services, Charon can be used to encrypt the client-to-proxy link for existing unencrypted services.
- Charon is at least as immune as Kerberos to protocol-based attacks, and more immune to certain end-to-end attacks: in effect PDA's can be used as smart cards for authenticating the user.

5.5 Acknowledgments

Charon was done as a semester project for a course in wireless mobile communications and a course in distributed systems, with the initial goal of providing an attractive authentication and cross-roaming mechanism for Daedalus. Eric Brewer pointed out the wider applicability of indirect authentication for network computers, intelligent personal communicators, and ISP's.

Ian Goldberg and David Wagner, cypherpunks extraordinaire, provided many useful insights on both the practical aspects and the security aspects of the Charon design and implementation.

Venkata Padmanabhan suggested the supremely snappy catch phrase "Security on the Move" for the paper title.

6 Bibliography

[1] S. M. Bellovin and M. Merritt. Limitations of the Kerberos authentication system. In *Proceedings USENIX Winter Conference 1991*, Dallas, Texas, USA, 1991.

[2] D. Cox. Wireless personal communications: What is it? *IEEE Personal Communications*, pages 20–35, Apr 1995.

[3] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *CACM*, 21(12):993–999, December 1978.

[4] H. Jerome and J. Saltzer. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, Sep 1975.

[5] A. Aziz and W. Diffie. Privacy and authentication for wireless local area networks. *IEEE Personal Communications*, pages 25–31, First Quarter 1994.

[6] D. Brown. Techniques for privacy and authentication in personal communication systems. *IEEE Personal Communications*, August 1995.

[7] Y. F. et al. Security issues in a cdpd wireless network. *IEEE Personal Communications*, 2(4):16–27, Aug 1995.

[8] T. R. Halfhill. Inside the web pc. *Byte Magazine*, pages 44–56, March 1996.

[9] J. G. Steiner, C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings USENIX Winter Conference 1988*, pages 191–202, Dallas, Texas, USA, February 1988.

[10] C. Brooks, M. S. Mazer, S. Meeks, and J. Miller. Application-specific proxy servers as http stream transducers. In *Proceedings of the Fourth International World Wide Web Conference*, Dec 1995.

[11] M. A. et al. Caching proxies: Limitations and potentials. In *Proceedings of the Fourth International World Wide Web Conference*, Boston, MA, USA, Dec 1995.

[12] Netscape Communications Corporation. The Netscape Proxy Server (home page). "http://www.pinncomp.com/netscape/proxy_server.html".

[13] B. Zenel. A proxy based filtering mechanism for the mobile environment. Thesis Proposal, Mar 1996.

[14] Randy H. Katz et al. The daedalus project (home page). http://daedalus.cs.berkeley.edu.

[15] Geoworks and Nokia. Nokia 9000 communicator (home page). "http://www.geoworks.com/htmpages/9000.htm".

[16] B. Barringer et al. Infopad: A system design for portable multimedia access. In *Calgary Wireless 94 Conference*, July 1994. Available at http://infopad.eecs.berkeley.edu/infopad-ftp/papers/1994/infopad_system_design.wireless94.

[17] Sun Labs. The java language: A white paper. Available at http://java.sun.com.

[18] E. Anderson. Personal communication, May 1996.

[19] Currently under blind review.

[20] A. Fox and E. A. Brewer. Reducing www latency and bandwidth requirements via real-time distillation. In *Proceedings of the Fifth International World Wide Web Conference*, Paris, France, May 1996. World Wide Web Consortium.

[21] L. Wall and R. L. Schwartz. *Programming perl*. O'Reilly & Associates, Inc., 1991.

[22] National Bureau of Standards. Data encryption standard. Technical Report Publication 46, Federal Information Processing Standards, 1977.

[23] ITU-T Recommendation v.42. International Telecommunication Union, March 93.

[24] Sony Corporation. Sony MagicLink PDA (home page). "http://www.sel.sony.com/SEL/Magic/".

[25] Geos operating system for PDA's (home page). "http://www.geoworks.com/htmpages/sso.htm".

[26] S. Gribble. Real-time streams to PDA's. "http://www.cs.berkeley.edu/~gribble/cs294-3_multimedia/project/project.html".

[27] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, Feb 1990.

[28] W. R. Cheswick and S. M. Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley Professional Computing Series. Addison-Wesley, 1994.

[29] I. Goldberg, D. Wagner, E. A. Brewer, and P. Gauthier. Basic flaws in internet security and commerce. "http://www.cs.berkeley.edu/ gauthier/endpoint-security.html".

[30] S. M. Bellovin. Personal communication, May 1996.

[31] Eric A. Brewer et al. The GloMop project (home page). http://www.cs.berkeley.edu/~fox/glomop.

Appendix A: Review of Kerberos Authentication Protocol

In this section we review the unmodified Kerberos protocol, so that we can then describe Charon by indicating how it differs from Kerberos. The Kerberos authentication protocol is an example of the Needham and Schroeder [3] key distribution protocol. The protocol is depicted in figure 1a—we briefly review it using the following terminology. Recall that in Kerberos, each entity capable of authenticating itself is referred to as a *principal*, and that to each principal corresponds a secret *key* known only to itself and the Kerberos Key Distribution Center (KDC).

In the first stage of the protocol, the client attempts to obtain a ticket-granting ticket (TGT); this step requires the client's principal to supply his Kerberos password. Once obtained, the TGT allows the client to request tickets for additional services from the TGS, without requiring the client's principal to retype the Kerberos password each time. These additional tickets are presented to the appropriate service as a means of proving the authenticity of the client's request for service. The exact messages exchanged are summarized as follows:

Messages 1-2: the client obtains a TGT. These are the messages exchanged by the *kinit* program.

1. C→KDC:

   The client requests a TGT from the TGS.
2. KDC→C: $\{K_{c,tgs}, \{T_{c,tgs}\}K_{tgs}\}K_c$

   The KDC returns the TGT (encrypted with the TGS's key) and a session key for use with the TGS, all encrypted with the client's key.

Messages 3-4: the client obtains a ticket for a service

3. C→TGS: S, $\{T_{c,tgs}\}K_{tgs}$, $\{A_c\}K_{c,tgs}$

   After decrypting message 2 using K_c, the client requests a ticket for service S, presenting the TGT (to authenticate itself) and a constructed authenticator (to guard against replay attacks) as credentials.
4. TGS→C: $\{K_{c,s}, \{T_{c,s}\}K_s\}K_{c,tgs}$

   The TGS returns a ticket for the service encrypted with the service's key, and a session key for use with that service, all encrypted with $K_{c,tgs}$.

Messages 5-6: the client requests service

5. C→S: $\{T_{c,s}\}K_s$, $\{A_c\}K_{c,s}$

   After decrypting message 4 using $K_{c,tgs}$, the client presents the ticket $T_{c,s}$ and authenticator A_c to the service.
6. S→C: service begins.