Secure Remote Access to an Internal Web Server

Christian Gilmore David Kormann Aviel D. Rubin

AT&T Labs – Research, Florham Park, NJ, USA
{cgilmore,davek,rubin}@research.att.com

Abstract

We address the problem of secure remote access to a site's internal web server from outside the firewall. The goal is to give authorized users access to sensitive information, while protecting the information from others. We implemented our solution using a one-time password scheme for client authentication and SSL for confidentiality. Our main design considerations were security, performance, ease of use, availability, and scale. We were further constrained by the desire to leave our firewall and local infrastructure unchanged.

1 Introduction

Most large organizations today have to resort to using firewalls to protect themselves from would-be hostile attackers on the Internet. While firewalls are an inconvenience, when well administered, they do a reasonable job of isolating an organization from the rest of the world. The security policy usually amounts to total trust of all insiders and total mistrust of outsiders, where the firewall defines the boundary [2].

As a result of classifying users as insiders or outsiders, companies can have different internal and external web views. People inside the firewall have access to the internal site, which contains sensitive data such as company strategy, business plans, etc. In addition, they can update personal information such as the payroll data and benefit allocations. The latter is often protected by simple passwords on top of SSL connections [5].

The external view of a web site is usually commercial and geared towards customers. Companies often use their external web site to display product information, details for investors, and other marketing material. More sophisticated organizations use their external web site for customer transactions, billing, and customer service.

It is common practice for the same fully qualified domain name to refer to the internal site when a user access the web from the inside, and to the external site when an external user makes a request. For example, here at AT&T, `www.research.att.com` is an alias to `akalice.research.att.com`, a web server behind the firewall, and `www.research.att.com` is an alias for `akpublic.research.att.com` for users outside the firewall. Employees can even have two versions of their home page, both referenced by `http://www.research.att.com/info/username`. The web page on `akalice` may contain proprietary information, whereas the copy on `akpublic` should not.

A problem arises when insiders travel outside of the firewall boundary. Most sites allow users `telnet` and `ftp` access to their machines from the outside so that they can read e-mail and edit files. Usually, the users are authenticated through some strong one-time password mechanism in hardware or software [8]. While users have a legitimate right to access the internal web server, it is not accessible to them. The best they can do is telnet to an internal machine and run a text-based browser, such as lynx [`http://www.slcc.edu/lynx/`]. While text-based browsers can be quite useful, they have severe drawbacks. There is no support for multimedia, executable content, helper applications and other recent features of browsers. Even worse, since telnet connections are usually unencrypted, the web content travels to the remote site in the clear.[1] Finally, public Internet kiosks (e.g. at the airport) may contain access to HTML browsers but no access to telnet or other Internet services.

Abadi *et. al.* present a solution that uses a web tunnel on the firewall for access from outside the firewall [1]. Their solution requires changes to the firewall and the use of client-side Javascript in the browser. Our goal is to allow access without any modification to the internal infrastucture, and in particular with-

[1] It is likely that SSH [19], SSL telnet, and IPSEC which solve this problem, will gain in popularity.

out changing the firewall or the end web server. We achieve this without requiring Javascript.

We layer a strong client authentication mechanism on top of SSL to allow legitimate users access to an internal web site from outside of the firewall. The idea is to make the session as transparent as possible for the user without compromising the security of the information. We do this without any changes to our firewall or internal infrastructure. The system allows users to access the internal web from outside the firewall. This is useful for employees who are on the road, or who receive their home access from a third party Internet Service Provider (ISP).

2 Client authentication

Strong client authentication schemes exist for applications such as `telnet` and `ftp`. Most notable are challenge/response hardware tokens and one-time password schemes based on hash chaining [8, 9] or pseudorandom functions [17].

Client authentication for the web exists in the form of client certificates issued by organizations such as Verisign and Thawte. However, the identity verification mechanisms of these organizations are often inadequate [18] and thus, these mechanisms are not in widespread use. We implemented our scheme using hash chaining [11] (see Section 5.1).

3 Our environment

Following standard practice, our firewall policy allows hosts behind the firewall to establish TCP connections to hosts outside the firewall on any port, while inbound connections are tightly restricted. Thus any mechanism that gives access to users on the outside must involve either opening an inbound port on the firewall or initiating a connection from the inside. Our intent is to layer our solution on top of an existing infrastructure, so we opt for the latter.

Another feature of our firewall is that it tears down inactive connections every 15 minutes. We initiate our service from behind the firewall, so we have to make sure that there is always a way for external clients to contact the internal web server when despite the fact that idle connections are torn down by the firewall.

We assume that users require occasional access from untrusted sites such as Internet cafes or terminal rooms at conferences. Our only requirement is that the web client be SSL enabled. We build user authentication into our protocol. We refer to the web client as a Dumb Web Terminal (DWT). We strive to treat the DWT as "untrusted" to the extent possible; nonetheless, the administrator of the DWT has complete control of all data coming in and out of the machine. So, in our model, we must assume that any internal web content viewed on a DWT could be secretly recorded and copied by the site administrator. What we do not allow, however, is for the administrator to be able to access *other* sensitive content by virtue of observing something.

It is possible that Virtual Private Network (VPN) technology could be used to allow access to the internal web, but we have not adopted any of these products at our site. Furthermore, we believe that many institutions would prefer to use a lightweight, free solution such as ours, rather than to invest in VPN products just for internal web access.

4 Architecture

Our architecture is diagramed in Figure 1. The main component of our system is the proxy. Of necessity, there is one subcomponent inside the firewall and one on the outside. A user on a DWT connects to the proxy through the Internet with an authentication request, using a special URL that contains his/her username. Next, the proxy contacts the authentication server to verify that the request is from a valid user. Once authentication completes, requests from the user are forwarded to the internal web server, which responds as usual. No modification is required to the DWT, the firewall or the internal web server.

Figure 2 shows the proxy in more detail. The internal machine, which we call `pushweb`, maintains a control connection to the external machine called `absent`. This is necessary because our firewall does not allow connections from `absent` to `pushweb`. Users at a DWT request a connection to `absent` by typing in a URL with `absent.research.att.com` as the hostname. When `absent` receives a connection from the browser, it records some information about the connection and sends a request along the control connection. Pushweb then opens a data connection to `absent`. Absent uses the data connection to forward requests to `pushweb`, which forwards them to the web server. The web server processes the request and returns an HTTP reply to `pushweb`. The reply is forwarded to `absent`, which sends it back to the DWT, where it is displayed for the user.

Web pages received from the web server may contain links to other pages behind the firewall. If the links are not changed, then future requests will not

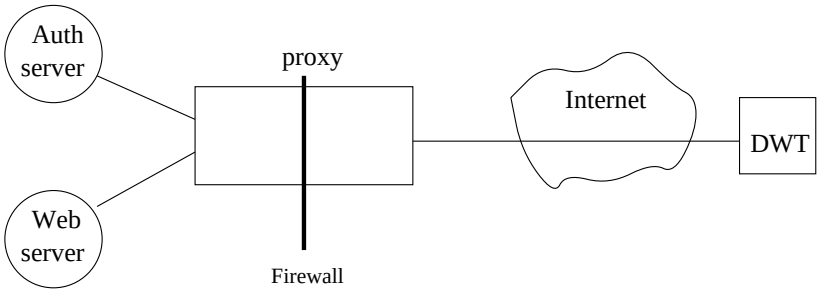

Figure 1: **Architecture:** The main components of the system. The proxy has one subcomponent behind the firewall and one on the outside.

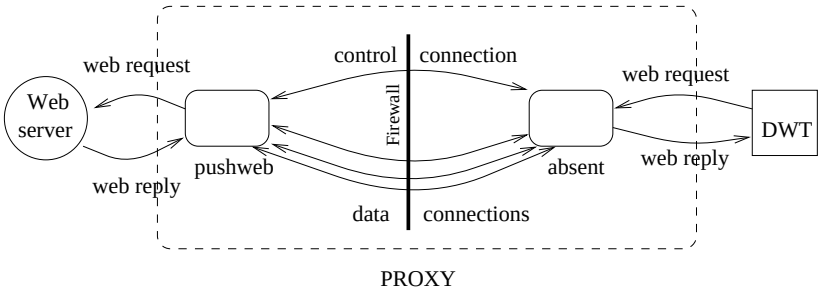

Figure 2: **The proxy:** The proxy consists of an internal and an external component. We use a machine that we call `pushweb`, on the inside, and a machine called `absent` on the outside. Pusheb opens a control connection to `absent`. When a web requests comes from a DWT, `absent` uses the control connection to instruct `pushweb` to open a data connection, which is used for the actual communication.

be able to access internal pages. For example, a link may be of the form:

```
<a href="http://myhost.research.att.com/
proprietary.html">Business plan</a>.
```

When the user clicks on "Business plan", the DWT attempts to connect directly to the machine `myhost`, and, of course, the firewall does not allow this. To solve this, `pushweb` does some processing on web pages before sending them to `absent`. First, all relative URLs are translated to absolute URLs with host names and directories. URLs that are outside of the trusted domain are not changed further. However, URLs that are behind the firewall are prepended with some security information (see Section 5.4). Then, new URLs are constructed that point to `absent` and include the original URL. For example, the URL

http://www.research.att.com/projects/

is rewritten as

https://absent.research.att.com/geturl=user/
2b5db86c1f6e/http://www.research.att.com/projects/

In general, rewritten URLs contain the following information:

- `https://absent.research.att.com/` Every rewritten URL starts with this string. It points the DWT to `absent` on port 443.
- `cmd=user` "cmd" can be login, geturl, logout, or OTP_resp and indicates the action to be taken by `pushweb` with the request. "user" contains the name of the user. This is his/her login account ID.
- `hex data` This is explained in Section 5.4.
- `original-url` This is the original URL that was contained in the page, converted to an absolute URL if necessary.

Recall that one of our goals is to make the browsing experience the same as when users are behind the firewall. Rewriting URLs achieves this. Pages appear the same to users, but when links are clicked on, the pages are requested through `absent`. The only differences users might notice are the appearances of codified URLs in the message window of the browser when the mouse passes over links, and the URL that is displayed in the location window.

In our system, `absent` forwards all requests to `pushweb`. Pushweb in turn, removes the substring

`https://absent.research.att.com/` and processes the remainder of the request based on the values of `cmd` and `user`.

5 Authentication and security

Security is paramount when we consider exporting private and confidential information outside of the firewall. We must assure that only valid users can access the internal web, while active attackers on the Internet cannot. In this section, we describe our techniques for authenticating clients and maintaining the privacy of the information from illegitimate outsiders.

5.1 Hash chaining

The original idea for hash chaining is due to Lamport [11]. There are two phases to authentication using hash chaining. In the *initialization phase*, a user picks a strong password, pw and a number, n, and using a well-known cryptographically strong one-way hash function, f, computes $y = f^n(pw)$. This amounts to n applications of f to the password. The value y is stored on an authentication server.

In the *authentication phase*, the user sends $y' = f^i(pw)$, where i is initially $n-1$, to the authentication server. The server checks to see if $y = f(y')$. If so, authentication is successful, otherwise it fails. If successful, the authentication server replaces y with y', the user decrements i by 1, and the process continues.

The security of the system lies in the fact that an eavesdropper on the network cannot compute any one-time passwords from previously used passwords. The use of a cryptographically strong hash function for f ensures that.

There is an Internet RFC [7] that describes a standard one-time password scheme that uses hash chains for authentication. The S/KEY and OPIE one-time password systems are freely available, widely used implementations of this standard.

5.2 User authentication

We use OPIE for authentication. To use Absent, users must have an entry in the OPIE keys database. To do this, they run the `absent_init` program which is a subset of the functionality of `opie_init`. The users specify the number of one-time passwords, n and a secret passphrase, pw. MD5 [16] is used as the one-way hash function, f. The program runs `setuid` to the database administrator and writes $n, f^n(pw)$ into the OPIE database, along with some other information about the users. At this point, users are initialized to use Absent.

An initialized user can use any dumb web terminal (DWT), authenticate, and access the internal web. The first thing a user does is issue a login request by typing in a special URL.

https://absent.research.att.com/login=user

The mechanics of the communication between `absent` and `pushweb` are discussed in Section 6. For the purposes of this section, we assume a data connection between `absent` and `pushweb` exists. `absent` forwards the request to `pushweb`, which immediately negotiates an SSL connection with the DWT (see next section), while `absent` blindly forwards data packets between them.[2] At this point, `absent` acts as a wire.

Once the SSL connection is established, `pushweb` looks up the user in the OPIE database. If the user is registered, `pushweb` submits a request to the authentication server which generates an OPIE challenge. Pushweb then constructs an HTML page with a form for the user to enter the OPIE response and sends the page over the SSL connection to the DWT. An OPIE challenge is of the form

otp-md5 386 bu5414 ext

where otp-md5 indicates that MD5 is the hash function, 386 is the number of times to iterate the function, and bu5414 is the seed for the generator. Figure 3 shows the OPIE challenge page. The user enters the response into the form and clicks the submit button. A list of one-time passwords can be printed on paper in advance or computed using a calculator. To compute a response, the user enters the seed, the number of times to iterate and the secret passphrase into an OPIE calculator. One of the reasons we chose OPIE is that calculators are becoming widespread. We use a public domain application on our Palm Pilot [`http://www.linet.it/pilot/a/pilototp.zip`] to authenticate.

After the user submits the one-time password, it passes over the SSL connection to `pushweb`, which sends the response to the authentication server. If the authentication succeeds, then an entry is created in a user table, and the page requested in the challenge form is returned to the user. One of the fields

[2] This is similar to SSL tunneling. We simply forward packets and do not use Netscape's tunneling protocol (Internet draft by Luotonen, 1997) because that relates to proxying SSL from a browser using the CONNECT command, and we are assuming that the user may not have access to the proxy settings in the browser.

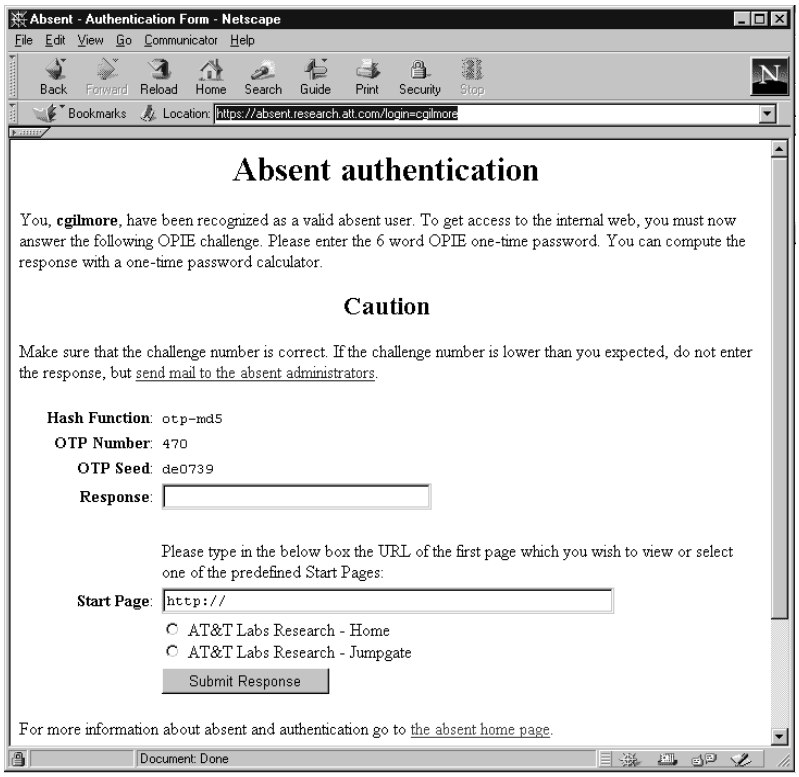

Figure 3: **Opie challenge page:** The user receives this page when logging in for the Absent service.

in the user table entry is an expiration time. In our system an authentication is valid for 20 minutes. After that, the user is presented with another challenge page.

It is important that the one-time password challenge and response occur over an established SSL connection. One-time password systems such as OPIE and S/KEY have been shown to be vulnerable to active attacks [17]. The confidentiality and replay prevention properties of SSL ensure that a play-in-the-middle attack (e.g. where the response from the user is blocked and then later used by the intruder) is not possible.

5.3 Connection confidentiality

Packets between the DWT and the firewall are vulnerable to sniffing attacks. Therefore, we must establish a private channel between pushweb and the DWT. While this channel passes through the `absent` proxy, the proxy is not privy to the data passed along it; it simply forwards packets along the channel. To accomplish this, we use HTTP over SSL, as implemented in the Apache-SSL web server. We further restrict the set of ciphers supported on the server to those providing "U.S. domestic-quality" encryption, as shown in Table 1.

It is important that users check the security information about their SSL connection to ensure that they are communicating with `pushweb`. Otherwise, an imposter could substitute some other valid server certificate and elicit OPIE passwords from the user, or feed him/her bogus content. Most browsers are configured to warn the user if the name in a certificate does not match the site requested. Thus, the certificate for `pushweb` actually contains the name `absent.research.att.com`.

At present we do not support internal SSL servers. If an internal server uses SSL, then we have to layer our secure connection over that SSL connection. This is presents some problems (see Section 8).

5.4 Other security features

5.4.1 Authenticated URLs

One of the important security features of our system is that no adversary should be able to access the internal web server. To that end, we bind every URL to an authenticated user. When a user authenticates using the one-time password scheme, an entry is created in a user table on the `pushweb` server. We generate a

```
SSL_RSA_WITH_RC4_128_MD5
SSL_RSA_WITH_RC4_128_SHA
SSL_RSA_WITH_IDEA_CBC_SHA
SSL_RSA_WITH_DES_CBC_SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA
SSL_DH_DSS_WITH_DES_CBC_SHA
SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA
SSL_DH_RSA_WITH_DES_CBC_SHA
SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA
SSL_DHE_DSS_WITH_DES_CBC_SHA
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
SSL_DHE_RSA_WITH_DES_CBC_SHA
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
```

Table 1: The SSL ciphersuites used by pushweb

random key for the user and add it to the table. Recall that when a user retrieves a page, all of the internal URLs are rewritten. The format of the new URL is

> https://absent.research.att.com/geturl=user/
> 2b5db86c1f6e/original-url

The field immediately preceding the original-url contains some security information. The first two characters, in this case 2b, correspond to the hexidecimal representation of the length in bytes of the original-url (see Section 5.4.2). The remaining data, 5db86c1f6e, represent the output of a Message Authentication Code (MAC) function, truncated to the 40 most significant bits. To compute the MAC, the server uses the key in the user table and the original URL. Truncating a MAC makes it more difficult for an attacker to exhaustively search for the key because there are many possible keys that could produce the shortened output [13, 15]. In choosing 40 bits, we trade off the amount of work it requires for an attacker to exhaustively generate a valid MAC versus the length of the URLs. Most systems limit URL length to 256 bytes, so we wanted to add as little as possible to this. To generate a valid URL with a valid MAC, an attacker would have to test on the order of 2^{40} URLs with random MACs. The birthday attack [13] that would require 2^{20} trials does not work here because the sample space for URLs is limited to valid URLs on the server. Since every trial requires a request from the server and keys expire every 20 minutes, it is infeasible for an attacker to generate a valid URL. In addition, we log unsuccessful requests, and an exhaustive search is immediately obvious on the server.

When the `pushweb` server receives a URL, it first checks that the user is valid (i.e. registered with Absent). It then checks that the length of the URL is correct. Finally, it retrieves the key from the user table, computes the MAC, and compares the most significant 40 bits to the MAC. If all of these tests are correct then the page is retrieved. If the key has expired, the server sends a new one-time password authentication form. This results in a new MACing key for the user after successful authentication. Thus, old URLs with MACs from expired keys, are useless.

5.4.2 CGI scripts

When a user submits information to the server in a form using a GET method, a URL containing the names and values of the input boxes is passed from the DWT to the server. This URL cannot be MACed in advance by `pushweb` because there is no way to know what values the user will enter. For example, say that a CGI script is referenced by the following URL (where `absent` represents `absent.research.att.com`)

> https://**absent**/geturl=alice/32a5d386cf6e/http://
> www.research.att.com/~alice/cgi-bin/reg.cgi

The URL sent to the server when the user submits could be

> https://**absent**/geturl=alice/32a5d386cf6e/http://
> www.research.att.com/~alice/cgi-bin/reg.cgi&name=bob

The MAC is correct for the first 50 bytes of the original URL, but it does not include "&name=bob". Therefore, we include the length, in hex, of the *original* URL that references the CGI script. In the above example the length is 50, which is hex 32.

The most significant 40 bits of the actual MAC are $0xa5d386cf6e$.

The effect is that authenticated users can execute CGI scripts as long as their keys are valid. A CGI script that is MACed by an expired key cannot be invoked. A potential danger is that an attacker who can surmise that a particular request is a CGI form can replay the message to cause the script to execute again with the same input data as before. Fortunately, SSL protects against replay attacks, so our system is not vulnerable to this.

5.4.3 Generating random keys

The security of any system that uses cryptographic operations lies in the unpredictability of the keys. As described above, our system requires that the server generate a secret key corresponding to each user to compute the MACs of URLs. We employ an expensive function to generate as random a seed as possible, followed by a speedy operation to generate user keys from the initial randomness.

Our system employs the `randlib` package from cryptolib [10] as a "true" random number generator. This package collects as much information as possible from the host environment and mixes it using cryptographic functions. The software uses information about network connections, the process table, memory, disk, etc. The process is very tedious and slow. Therefore, we call the `truerand` function only upon server startup to generate a master key. Then, we use DES [14] as a pseudorandom function with the master secret as the key, to generate all other MAC keys.

It is important to note that the recent results showing 56 bit DES to be vulnerable to exhaustive search [6] do not in any way impact the appropriateness of DES as a pseudo random number generator (PRNG). When DES is used as a PRNG, there are no plaintext/ciphertext pairs for an attacker to use, and thus, a DES cracking machine cannot be used to predict the output of the generator. This is true because the MAC keys that we generate (the output of DES) and the initial random seed (obtained from `randlib`) are never available to the attacker.

After generating the master secret, a counter is initialized. Then, every time a new key is needed, we compute $new\text{-}key = DES(master\text{-}key, counter)$ and then increment the counter. This has the advantage that computing new keys is fast, while cracking these keys without knowledge of the master key is difficult.

To illustrate this, we describe how the attacker might crack the master-key from which the user keys are derived. First, the attacker must collect URLs containing MACs from one user. These are limited in number by the lifetime of the key (20 minutes) and the number of links in the requests made by the user. In addition, the URLs are generally unavailable to outsides in our system because SSL is used. Next, the attacker must compute the user's key. This amounts to breaking HMAC, a task considered to be infeasible. Even if the attacker finds a key that computes all of the MACs correctly, there is no guarantee that it is the right key because the truncation of the MACs to 40 bits results in many possible keys (see Section 5.4.1). Finally, the attacker knows that user keys are produced by applying DES to a counter, so a reasonable guess can be made about the input value. Thus, after doing all of this work, the attacker can produce a pair $\{P, C\}$ where P is a range of possible plaintext (some counter value) and C is a possible ciphertext (a user's MAC key). To obtain more pairs, the attacker must break HMAC for another user. The best known techniques for breaking DES involve 2^{43} plaintext/ciphertext pairs [13]. As stated above, brute force in not an option because even the ciphertexts are not known to the attacker. We conclude that it is not feasible for an attacker to crack the master key. For additional security, we note that the master key does not represent any state in the system; it is only used to seed a PRNG. So, it can be changed at any time by calling `truerand` again.

5.4.4 Other issues

For maximum security, it is important that users clear the memory and disk cache and then kill their browser. In addition, we include the HTTP directive `Cache-Control: no-cache` in every page. This has the effect of forcing the browser not to cache pages. The method is not fool-proof, as users could still save the page they are viewing onto the remote machine, but there is a limit to how much we can protect the information from users who are determined to expose it.

6 Implementation

Wherever possible, we used existing software to implement Absent. This both simplifies security analysis (if we can assume the component parts to be secure) and reduces our coding effort (particularly in the case of the internal proxy and one-time password systems). For this reason, only the Absent daemon itself consists entirely of original code; most of the other components make use of exiting code.

The Absent system consists of two daemons that implement the functions of the external server

(absentd) and internal proxy (pushweb), and the protocol with which these daemons communicate over a control channel. The control channel is opened by pushweb at startup. The protocol on this channel is fairly simple, consisting of five messages:

- $HELO(timestamp, mac)$: Sent by pushweb when opening the control channel. absentd checks that the timestamp is within a reasonable amount of time from the current time (to prevent replay attacks) and ensures that the supplied MAC is in fact HMAC-MD5(secret, timestamp), where *secret* is a MACing secret constructed at installation time and shared by the two daemons. If the MAC matches and the source of the connection is the configured address of pushweb, the connection is assumed by absentd to be valid. If not, the connection attempt is rejected.

- *COPEN(id, timestamp, client_sockaddr, MAC)*: When a client connects to absentd, this message is sent to pushweb to indicate that a new data connection for the client should be opened. This connection will be used for proxied data between the client and pushweb. The *client_sockaddr* is a Berkeley-style socket address, indicating the site from which the client is connecting. The *id* argument is used by absentd to identify the client; it has no meaning to pushweb, and is essentially an opaque value which should simply be returned. The *timestamp* and MAC fields are checked by pushweb as in the $HELO$ message, with the exception that the MAC covers all the arguments to the control message.

- $COPEN\_R(id, timestamp, MAC)$: The $COPEN\_R$ message is not, strictly speaking, control channel protocol. It is sent along a new data data channel by pushweb after the connection is opened. On receipt of the message, absentd does the usual checks on *timestamp* and MAC and, if the *id* value refers to a waiting connection, begins acting as a proxy for the client.

- $PING(timestamp, MAC)$, $PONG(timestamp, MAC)$: Our firewall times out inactive connections after a period of time. To prevent this, Absent implements a simple keepalive protocol. These messages implement that protocol. Periodically, either side may send a $PING$ message. The sending side expects to receive a $PONG$ message within a reasonable period of time. If none is received, the control connection is assumed to be dead; if absentd notices this, it stops accepting new client connections until a new control connection is established (existing connections continue to be serviced). If pushweb notices this, it attempts to reestablish the control connection.

Note that all control messages have associated MACs over their arguments. Both daemons are fairly draconian about dealing with incorrect MAC values. If an incorrect MAC is received, the control connection is immediately closed. This suggests a fairly simple denial-of-service attack based on the injection of bogus packets. No attempt has been made at this point to repair this problem, and the authors welcome suggestions.

absentd is a simple (roughly 900 lines), standalone C program. Aside from implementation of the protocol described above, it is an unremarkable blind proxy.

The internal daemon, pushweb, is more complex. The daemon consists of three parts:

- A modified Apache web server
- The URL-rewriting handler
- The one-time password interface

The choice of Apache as the base upon which to build pushweb was driven primarily by the fact that Apache itself provides most of the features pushweb requires:

- An SSL implementation (with Apache/SSL)
- A proxy
- A web server
- Available source.

6.1 Modifications to Apache

The modifications to the Apache source are minimal; they consist mainly of replacing Apache's connection-accepting code with code that on receipt of a $COPEN$ message on the control channel, creates a new connection to the absent daemon. Once the socket address of this new connection is set to the *client_sockaddr* value supplied with the $COPEN$ message, the connection appears to apache to be a normal client connection from the client address. This spoofing allows Apache's access control and logging functions to behave normally, and permits us to use the remainder of the Apache code unmodified. The changes to Apache source entail fewer than 100 lines of C.

6.2 URL-rewriting handler

We've writen an apache handler to deal with all tasks that are preformed once the connection is established. These include handling requests, authenticating users, fetching web documents, rewriting URLs in HTML documents, and returning the document to the client (through `absent`). Apache allows third parties to write handlers to fit into certain phases of each transaction. We've written our handler in perl to utilize its powerful regular expression functionality and to avoid recompiling every time the code changes. The changes require only a minor addition to the Apache startup configuration files. This handler, which involves fewer than 500 lines of code, is invoked during the URL translation phase.

The first step is to determine whether the command value is `login, OTP_response, logout`, or `geturl`. For a `login` request, the code ensures that the user is registered and sends the OTP challenge to the client. When an `OTP_response` command is received, we check the validity of the response. If valid, the user is logged in, and the requested page (included as part of the OTP_response) is returned to the client with all of the URLs on the page that reference our internal domain rewritten. Upon a `logout` request, the MAC of the logout URL is checked and, if valid, the user is logged out. The log out consists of removing their entry in the table on `pushweb`, in particular, deleting their MAC key. Otherwise, we assume that it is a forgery and return an error.

When a `geturl` command is received, our handler checks the MAC of the requested URL and, if valid, submits the request to the internal web server. If the `Content-type` of the response from the server is `"text/html"`, the document is then parsed to identify all links on the page. Every link containing a relative URL is converted to one containing an absolute URL by adding the "http://" protocol, the complete server name (such as music.research.att.com), and the proper path information (e.g. replacing ../foo.html with /dir/foo.html) to the URL if they are missing. A second pass is then made through the document to prepend the Absent information to each URL. Finally, a logout button is added to the page, and the `Content-length` header is adjusted to match the new larger document length. If the `Content-type` is not `"text/html"`, the response from the server remains unedited. Finally, the HTTP response is returned to the client.

6.3 Performance

The most significant performance bottleneck in Absent is the parsing engine. Therefore, we built a special-purpose parser that leaves most HTML untouched, and is concerned only with tags that can contain URLs. In addition to the overhead of parsing, a MAC is computed for each link, resulting in 2 computations of MD5. As URL length is limited to 256 bytes, the hash function only iterates once for each call. Thus, a page containing 750 links requires 1500 iterations of MD5. However, we find that this is only significant (i.e. perceptible) for pages with a tremendous number of links. Most pages are processed very quickly with an unnoticeable overhead.

As an extreme example, we examined a page with 3398 links. The size of the original HTML file is 116,997 bytes. This page takes 18 seconds to parse and convert all the URLs, including changing relative URLs to absolue, parsing, and computing MACs on our `pushweb`, a Sparc Ultra 2. The same operation took almost a minute using an off the shelf parser. The resulting page is 363,497 bytes long, a 311% increase in size. Thus, performance is strongly tied to the number of links on a page. It is our experience that most pages contain few enough links that the added latency is not noticeable.

7 Security assessment

The security of our system rests on the security of the underlying mechanisms and their composition. We use off-the-shelf software components such as OPIE, SSLEAY, and HMAC. These packages have been heavily scrutinized by experts in the security field, so we have some confidence in them. We constantly monitor bug reports on relevant newsgroups and mailing lists, and plan to upgrade immediately any component that is discovered to have a security problem when a patch is released.

7.1 Compromise of absent

We constantly and carefully monitor `absent`. However, this machine is outside of the firewall, and it is reasonable to assume that it will come under attack. Assuming that a sophisticated attacker manages to become root on `absent` without our noticing, we examine the possible consequences. The attacker's goals are the following:

Denial of service The attacker prevents valid users from being able to use the system.

Passively eavesdrop on a user's session The attacker attempts to see the contents of the user's interaction with the internal web server without diverting from the SSL protocol.

Serve bogus information to a user In this attack, the attacker masquerades as the internal web server and serves up fictitious information to the user.

Obtain valid one-time passwords In this scenario, the attacker's goal is to fool the user into exposing a one-time password with a lower number than any previously used.

Access the internal web The attacker attempts to use its control over `absent` to bypass the authentication mechanism and access protected content on the internal web server.

Obtain root on pushweb The attacker attempts to use its control over `absent` to attack `pushweb`.

Compromise of internal network The attacker attempts to use its control over `absent` to compromise the internal network including control over machines, and files.

The first attack, denial of service, is not preventable as an attacker who controls `absent` can easily close all sockets on the machine and refuse to communicate with anyone. We monitor the machine for this condition and can detect such denial of service attacks. The second attack is more difficult for the attacker. Absent serves only as an SSL proxy. It blindly forwards SSL data between `pushweb` and the client. There are no encryption/decryption keys stored on `absent`. Therefore, there is no way that the attacker can eavesdrop on a session without breaking SSL or performing a more active attack.

To serve bogus information to the user, the attacker must establish an SSL connection with the browser. To do this, it must serve a valid certificate. Such certificates are not too difficult to obtain. If such an attack is successful, the server could fool the user into revealing secret one-time passwords. The only way to prevent this attack is to require users to check the security information in the certificate when they use the system, and to verify that the name of the server in the certificate is "absent".[3]

[3]Even though the SSL connection is between the DWT and `pushweb`, the client certificate we serve has the name `absent` in it. This is because the DWT connects to an address that starts https://absent.research.att.com/ so we use the name absent in the certificate so that the browser won't complain that the name of the server and the name in the certificate don't match. The private key corresponding to the public key in the certificate is kept only on `pushweb`.

Without compromising `pushweb`, the internal server, SSL, or one-time passwords, there is no way to use access to `absent` to get to the internal web server. This is because the only messages coming from `absent` to `pushweb` are control messages instructing `pushweb` to open data connections. These connections are used to forward SSL traffic. An attacker on `absent` can exhaust resources on `pushweb`, but that is the extent of the damage possible. Similarly, barring buffer overflow attacks and other such vulnerabilities related to bugs in the software, there is no way to use root access on `absent` to break into `pushweb` or the internal network more easily than from an arbitrary host on the Internet.

7.2 Compromise of pushweb

Because `pushweb` runs behind the firewall, a root compromise could be devastating. Besides compromising all access control on web content, an attacker could launch attacks on the internal file system and on user accounts. To compromise this machine, an attacker needs to exploit vulnerabilities in the `pushweb` code or existing weaknesses in the firewall. The latter is a problem independent of our service, and we assume that others are protecting the perimeter. We were very careful about memory allocation to avoid buffer overflow problems (the leading cause of software security flaws [4].)

We take the following special precautions on `pushweb`.

- The `pushweb` server runs as user *nobody*, which has permissions only to read and write files needed for the Absent service.
- No other services are available from `pushweb`.
- There are no regular user accounts on `pushweb`, just administrative accounts to manage the Absent service.
- All important actions and especially error conditions are logged, and the logs are monitored closely. We hope eventually to log on a WORM (write once-read many) disk.
- All machines except the internal web server are configured to refuse connections from `pushweb`.

These precautions make it more difficult for an attack on `pushweb` to lead to further compromise of the internal network. The logs are crucial to penetration

detection and recovery. To date, there have been no successful compromises of `pushweb` or `absent` (to our knowledge).

8 Limitations

One fundamental problem in our system is the inability to access secure servers behind the firewall. The reason is that the SSL protocol establishes a secure connection between the server and the end client. If an external user contacts an internal server through `absent`, there are two possibilities. The first is that the user's remote machine and the secure server establish an SSL connection directly. The second option is that the client and `absent` establish an SSL connection, and `absent` establishes a secure connection to the server and acts as a forwarding agent. Each of these approaches is problematic.

In the first scenario, there is no way to rewrite the URLs to point back to `absent` so any attempt to access a link on a page from a secure server is blocked by the firewall. Thus, it must be possible to see the contents of pages served to the client (the second scenario). However, as the administrators of `absent`, we do not want the responsibility of being able to observe traffic that is supposed to be confidential. For example, if a user accesses his/her private payroll data, and a dispute later arises about a web transaction, it would be possible to blame us, since we had access to the data. For this and other, similar reasons, we do not support access to secure servers behind the firewall.

Another limitation of Absent is that some functionality is lost when URLs are dynamically generated on the client side by a scripting language such as Javascript. There is no way to parse the HTML and find these URLs. If the URLs reference something behind the firewall, the subsequent request will fail. There is nothing that can be done about this without analyzing the scripting code, and this is known to be very hard to do. It's actually impossible for general-purpose code [3].

As mentioned in Section 3, we do not have VPN technology available to us. It is clear that an integrated virtual private network is a better solution than the one we have provided. First, it is more transparent to the users, and second, it is more secure. This is because our system is composed of several different components, SSL, Apache, OPIE, and our own code. While the security of one-time passwords is believed to be well understood, and SSL has been analyzed carefully, little or no analysis of the composition of these systems has been done. In fact protocol composition is a very hard problem and has led to security problems in the past [12]. Given our goal of providing internal web access from sites such as terminal rooms at conferences and Internet cafes, it seems that a VPN solution is not feasible.

9 Conclusions

We present Absent, a system for providing secure access to an internal web server from outside of the firewall. We make use of the secure socket layer (SSL) protocol to achieve confidentiality and one-time passwords for user authentication. Absent is designed to minimize change to our local infrastructure and to make use of off-the-shelf security components. The key design considerations were security, performance, ease of use, availability and scale. Our system is currently in production use by researchers at AT&T. The code is freely available at `http://www.research.att.com/projects/absent`.

Acknowledgements

We thank Sam Alexander, Peter Honeyman, Larry Jackel, Michael Kocheisen, Urs Muller, Michael Reiter, and Stuart Stubblebine, for helpful comments.

References

[1] Martin Abadi, Andrew Birrell, Raymie Stata, and Edward Wobber. Secure web tunneling. *Proceedings of the Seventh International World Wide Web Conference. Computer Networks and ISDN Systems*, 30:531–539, April 1998.

[2] Bill Cheswick and Steve Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley Publishing Company, 1994.

[3] M. E. Davis and E. J. Weyuker. *Computability, Complexity, and Languages*. Academic Press, Inc, 1983.

[4] Simson Garfinkel and Gene Spafford. *Practical Unix & Internet Security*. O'Reilly & Associates, Inc., 1996.

[5] Simson Garfinkel and Gene Spafford. *Web Security & Commerce*. O'Reilly & Associates, Inc., 1997. Appendix C.

[6] John Gilmore. EFF builds DES cracker that proves that data encryption standard is insecure. *EFF press release*, July 1998.

[7] N. Haller and C. Metz. A one-time password system. *Internet Request For Comments (RFC) 1938*, May 1996.

[8] Neil Haller. The s/key(tm) one-time password system. *Symposium on Network and Distributed System Security*, pages 151–157, February 1994. ftp://thumper.bellcore.com/pub/nmh/skey/.

[9] Naval Research Labs. Opie software distribution, 1996. ftp://ftp.nrl.navy.mil/pub/security/opie/.

[10] John B. Lacy. CryptoLib: Cryptography in software. *USENIX Security Conference IV*, pages 1–18, 1993.

[11] Leslie Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–771, November 1981.

[12] David Martin, S. Rajagopalan, and Aviel D. Rubin. Blocking java applets at the firewall. *Proc. Internet Society Symposium on Network and Distributed System Security*, pages 16–26, 1997.

[13] A.J. Menezes, P. V. Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[14] National Bureau of Standards. Data encryption standard. *Federal Information Processing Standards Publication*, 1(46), 1977.

[15] Bart Preneel and Paul C. van Oorschot. On the security of iterated message authentication codes. *IEEE Transactions on Information theory*, 1998.

[16] R. Rivest. The md5 message digest algorithm. *RFC 1321*, April 1992.

[17] Aviel D. Rubin. Independent one-time passwords. *USENIX Journal of Computing Systems*, 9(1), 1996.

[18] Aviel D. Rubin, Daniel Geer, and Marcus J. Ranum. *Web Security Sourcebook*. John Wiley & Sons, Inc., 1997.

[19] Tato Ylonen. SSH - secure login connections over the internet. *USENIX Security Conference VI*, pages 37–42, 1996.