GENSEC - Designing a security subsystem

Andrew Bartlett <abartlet@samba.org>

25th April 2005

Abstract

This paper attempts to introduce the GENSEC security subsystem and client credentials interfaces to interested bystanders and developers on the Samba4 project. GENSEC attempts to abstract details about user authentication into a single place, providing a simple API to the rest of Samba, while remaining compatible with Microsoft's implementations of the same security protocols. Likewise the client credentials API provides Samba's client components with a single, simple way to handle usernames and passwords.

Background

Samba4 is the ambitious new development branch of the long-standing free/open source Samba project, a CIFS implementation which since 1992 has built compatibility with numerous clients and servers, but particularly those of Microsoft's Windows suite. In this new version, the Samba Team has taken on an unparalleled challenge of matching current Windows versions exactly, in terms of network protocols and features. While providing many challenges, this goal provides an opportunity to implement things *right*. This document is about this new development branch, and unless stated otherwise references to Samba mean the current Samba4 development project.

The series of subsystems presented in this paper are the culmination of four years of thought and development, since the first 'Authentication rewrite' work on the then Samba HEAD development branch back in 2001.

Because Samba takes the challenge to match Microsoft's latest releases exactly, the issues surrounding Active Directory and modern security technologies quickly came to the fore. It is no longer be possible to just pretend to be NT4 and hope that the clients did not expect any particularly difficult behaviour. With this incarnation of Samba these challenges are being tackled, not just worked around.

Finally, while the word 'security' does mean many different things, this paper addresses the issues as they stem from authentication and the related problems of data privacy and integrity over a network.

Security functions

Before we take a more in depth look at the technologies, it is perhaps wise to first describe the question that security subsystems should be able to answer:

who are you?.

Who you are

The most common operation performed by a security subsystem is proving a user's identity. In a common example, a user asserts their identity with their login user-name and password (the step of authentication), and then asserts (by authorization) that 'only I should be able to get at my mailbox'. This is an easy problem to solve in the abstract: you simply prove who you are, and proceed to be authorized for other operations. Performing this operation in a manner that is secure to attack, and network efficient, is a more difficult task however.

Who they are

One of the less-considered, but equally important matters that a security subsystem should take care of is peer authentication. It is readily accepted that the user must prove their identity to a server, but with a few exceptions (mostly involving 'secure' websites), users do not expect a server to prove its identity to them.

Proof of server identity is a very important issue, as soon as any of the information given by that server needs to be trusted. Often called 'mutual authentication', a solution to this problem ensures that the server is 'trusted' not to provide malicious data, such as an invalid address book, and that it will behave properly with information, such as credit card numbers, given to it. Imagine if your bank website was being spoofed, and no matter where you asked the bank to transferred your money, it was always diverted into the attacker's swiss bank account? Further imagine if the attacker continued to spoof your Bank's website, to give the indication that the money had never left your account. These are the kind of problems we must deal with in mutual authentication.

Data integrity

Once the identities of a server and client has been established, there must be some way to ensure that only that server or client can continue to communicate with the other. Otherwise, it may be possible for an impostor to take over a connection and impersonate one party to the other. This connection hijacking could allow unauthorized access to privileged documents, or server administration functionality.

The practice of data integrity, also known as 'signing', typically requires a cryptographic calculation designed such that both sides can prove these facts: This signature could only have be performed by the other party, and could only have been constructed over the data received. If the output of that calculation does not match the value supplied by the other party, the message must be considered compromised.

Data encryption

Whenever data flows over a communications network, it becomes open to attacks simply by observation. Should a user's password be sent in clear-text, an attacker could use it to later log in as that user. Likewise, if a confidential document were retrieved, an attacker in a position to watch the network traffic could also read that document.

The practice of data secrecy, also known as 'sealing', 'privacy' and 'confidentiality', involves the encryption of specific data portions, or the entire communications channel, such that only the other party can decrypt it.

Transparency and Single Sign on

One of the biggest challenges in building a security subsystem is correct implementation of 'Single Sign On' and single source of password solutions. Users expect that if they have 'logged on' to the network, that further network access will not require them to re-enter their passwords, and any security subsystem should be designed to accommodate this. Though simple in concept, the challenge is to design the security subsystem in a way that does not allow the user to unwittingly compromise their own security.

Likewise, the choice of data integrity and encryption functions should be transparent to the user, and as transparent as possible to the applications using the security subsystem.

Authorization problems

The problem of authentication and the problem of authorization are often lumped together, and this is certainly the case in Active Directory. We will first need to consider them separately.

Once a user is authenticated, that is, they are who they claim to be, or more simply they know the password to the account, a server must then determine what resources that user may access. What files may they read/write? What hosts may they login to? This process is authorization, and even extends to such ideas as impersonation; a user may be authorized to impersonate another user.

The tempting thing to do when designing real-world systems is to embody a great deal of authorization information in the authentication process; information such as group membership may be returned as a product of the login process, and even evaluated to determine if an authentication attempt should succeed. Unfortunately, it links the two processes very tightly: preventing a single authentication identity from having multiple authorization identities. That said, the two are linked (as they are in AD) for reasons of network efficiency.

The problem

This paper details the design and development of a security subsystem for Samba. GENSEC, as it has become known, provides Samba client and server modules with a consistent interface to functionality including authentication, data integrity, data privacy and user credentials (used for authorisation).

The idea of a generic security API is not new - such APIs and protocols have been available for many years, and in many guises [1]. The reach of the Samba project is such that it will clearly need to implement, by some mechanism or other, a wide variety of these protocols, including SASL, GSS-API, SPNEGO[7] as well as the proprietary NTLMSSP[6, 2]. This is not unique to Samba, and in the wider open source world we see individual applications introduce similar abstraction layers, or adopt the Open Source Cyrus-SASL library to provide one.

History

The need for a centralised security subsystem became clear in the development of Samba 3.0. Samba 3.0 contains three distinct, and incomplete implementations of NTLMSSP[1], at least two implementations of SPNEGO[2], a very simple SASL client and an SCHANNEL implementation. While it did work, the lack of clear boundaries around many parts of this code made extracting and consolidating this infrastructure a nightmare. Using it and extending it was no walk in the park either. A lack of clear interfaces also meant that libsmbclient and smbclient were largely unable to use Kerberos session credentials, even if they were available. With this new development effort, the opportunity was grasped early, and implemented before too much code was written, allowing clean boundaries to be drawn.

This centralisation requirement also ensured that we would always have the same set of security mechanisms available, wherever they were appropriate: not limited by their original source modules.

The Microsoft pattern with SSPI

On the Microsoft side of the fence, it is well known that Microsoft uses a subsystem called SSPI (Security Support Provider Interface)[4] to handle almost all their network authentication and encryption interactions. This module, modeled after GSSAPI but without API compatibility, provides all Windows applications and the OS itself, with a single interface to these 'security functions'. This model was chosen not only for quite sensible software engineering reasons, but also to provide a single point of audit (and key weakening) for encryption export controls.

Sadly, Microsoft did not always use SSPI, and clearly has some private hooks to certain parts of the back-end functionality. As such, certain behaviours appear in the network protocols that cannot be strictly emulated via the public API, nor via GSSAPI, were we to place our modules behind that framework. These behaviours include, in particular, the use of the 'user session key' directly in arbitrary encryption and digest functions, rather than the use of SSPI functions for these purposes.

[1] As well as the CIFS client and server, NTLMSSP was separately implemented in the RPC server and in the ADS LDAP client.

[2] The SPNEGO implementations were in the CIFS client, server and separately in `ntlm_auth`.

Supporting Negotiation

One of the prime requirements of the GENSEC system is to support the SPNEGO protocol, a security negotiation protocol used extensively by Microsoft to select a real protocol used to handle authentication on a particular connection. As such, GENSEC has been designed with recursion in mind; this GENSEC module should be able to choose another to perform the final task, while allowing the negotiation details to be handled inside the SPNEGO module itself.

Single Sign On

Following on from the special handling of SPNEGO is a particular problem for the implementation of Single Sign On solutions. On a Unix-like platform, the only Single Sign On technology currently available is Kerberos, and this is selected (typically) via SPNEGO. This means that the calling application has no idea if Kerberos is available, nor if the user has sufficient credentials to use it. (Only NTLMSSP may be available, or the KDC may be un-contactable, which would require a password prompt).

In short, Samba's client applications must no longer unconditionally prompt for a password, and must instead provide some appropriate callback.

Protocol Scope

The biggest challenge (and the failure of the previous efforts in this area) is the shear scope of the protocols involved. Previous efforts did not attempt to address all the host protocols at once, nor did they address or even allow for the full scope of security protocols.

Host Protocols

At this stage, the host protocols which require security support in Samba are:

- CIFS
- DCE-RPC
- LDAP
- HTTP

Security Protocols

Likewise, any solution we construct must also correctly handle the number of security protocols we implement:

NTLMSSP

NTLMSSP is a generic encapsulation of the NTLM challenge-response authentication protocol, which derives from the early days of the CIFS protocol. NTLMSSP adds negotiation of options including NTLM2 and the strength of the subsequent signing or encryptions systems. Described on-line in both an Open Group publication[6] and on Eric Glass's Davenport NLTM page[2], NTLMSSP is now a pretty well-understood wrapping of NTLM, which we already know well.

NTLMSSP is perhaps also the most-abused authentication protocols, particularly as it lead the way in 'single sign on' HTTP (initially a very Microsoft-only world, and with due disregard for established standards), but also finds itself in almost every networks product produced by Microsoft. Unlike Kerberos, there is no need in NTLM for the client or server to know each other by some other means, which allows both attacks and connectivity. If the user's login credentials (shared automatically by the system) are not sufficient, the user is typically prompted for a password.

Kerberos

Kerberos[5], originally from MIT's project Athena, is a cryptographically secure trusted-third-party security system. Kerberos version 5 (krb5) is the current standard in RFC1510[3]. Suffering the key disadvantage of often requiring that DNS function correctly, and that the client and server must belong to the same (or trusting) realms, Kerberos is chosen by default between windows clients in Active Directory, and Microsoft has extended Kerberos to behave in ways useful to it's role on the network.

New clarifications to the Kerberos 5 standard have recently been approved by the IETF.

GSSAPI

GSSAPI is a wrapping layer around security protocols, designed to make them easier to use. It typically wraps Kerberos version 5, but in theory can wrap other systems. It has the 'honour' of being the inspiration behind SSPI, which has had far more success in the network landscape than GSSAPI itself.

SPNEGO

This is a GSS security negotiation protocol, which means it fits into GSSAPI in a network sense. It is intended to also fit into a the normal GSSAPI libraries. Also known as SNEGO, work is progressing to make the P again really mean 'protected'. SPNEGO in the current network reality will negotiate between Kerberos and NTLMSSP, with the client typically choosing Kerberos if it can successfully obtain a ticket for that particular server, otherwise transparently and without warning falling back to NTLMSSP.

SCHANNEL

SCHANNEL is the security mechanism used between Microsoft client workstations and servers for domain membership, and uses the machine trust account. It is setup by first making calls on the NETLOGON DCE/RPC server, which sets up a shared secret to allow the client to connect to other RPC services. Interestingly the encryption algorithms are actually very similar to those used and documented for Microsoft's `arcfour-hmac-md5` Kerberos encryption type, which made implementing this otherwise unknown standard easier.

Others

This list is expected to grow, particularly as LDAPv3 has DIGEST-MD5 as a 'mandatory to implement' security mechanism. It may be possible to link to Cyrus-SASL, to optionally obtain a potentially arbitrary number of additional mechanisms.

Building the solution

Building our own

The Samba project is perhaps unique in the extent to which it is expected to be very, very portable: between hardware platforms, operating systems and choice of software libraries. This presents a particular challenge, and when we are faced with implementing Microsoft-compatible security protocols, it is doubly so.

The Samba Team policy of self-reliance avoids having a large number of external libraries that we must mandate be installed by administrators, and ensures that we can fix (within our own area of control) any issues that come up in protocol conformance. Likewise, there are aspects of control that we require, that violate all the 'good design' rules that an externally library may suggest we comply with, as we have found with Kerberos and GSSAPI in particular.

Additionally, there are some protocols (such as NTLMSSP and SCHANNEL) that are not fully implemented elsewhere, and for which the Samba implementation is the most advanced, and any generic security solution would need mechanisms to import that portion of Samba.

NTLMSSP

The NTLMSSP library was brought forward from Samba 3.0, and has been updated from that point. This code, built in the early days of the Samba 3.0 project to support SPNEGO includes a micro-implementation of NDR generation and parsing, suitable for the small packets used in the NTLMSSP exchange. The code has proved surprisingly stable, and has now been extended to handle NTLM2 signing.

It was the comparative success of this code in Samba 3.0 (used in the CIFS client, CIFS server and the DCERPC client) that strongly influenced the design of GENSEC.

SPNEGO

The SPNEGO code used by GENSEC was derived from the code contributed by Anthony Ligouri to Samba 3.0's `ntlm_auth` utility. This needed substantial extension, but unlike the code used in the rest of Samba 3.0, this was quite practical as it already possessed distinct parse and logic layers.

This code now selects between the registered GENSEC mechanisms, in choosing a suitable security protocol (essentially NTLM or Krb5 for now) for use on the connection.

SCHANNEL

Samba now includes a new SCHANNEL implementation, only distantly derived from that in Samba 3.0. Now better separated from the rest of the DCE-RPC code than it was in Samba 3.0, this is handled almost entirely as a normal GENSEC module.

Using the Heimdal library

In an apparent contradiction with the above, we are also investigating a strong tie with the Heimdal implementation of Kerberos and GSSAPI. This contradiction comes about because we do not wish to re-implement the entire Kerberos and GSSAPI libraries, but require features that to this point are only implemented in our custom release of that library.

The idea is that we will statically link with this library, rather than require the entire system convert to our choice.

Client Credentials Interface

Samba provides a credentials management system, for the consistent handling of login credentials, be they user-names, passwords or implicit Kerberos credentials.

The particular feature of this interface is the callbacks, and the 'level of specification'. By allowing the credentials code to guess the user-name, for example, we gain usability benefits. However we allow this to be overridden, and we know that now the user-name has actually been specified. Likewise, a password may be specified or if one is supplied, a callback run.

Separate from GENSEC

For now at least, there are parts of Samba which deal with passwords but do not deal with GENSEC - these include in particular the 'basic' session setup code, where extended security is not negotiated. These do not need to carry the full weight of GENSEC, which they otherwise cannot use, so the specification of passwords has been abstracted into a smaller module. GENSEC completely depends on this smaller module, referencing those methods directly.

A single context pointer

Before the introduction of the new credentials code, it was common to have a set of arguments in the form 'user-name, 'domain' and 'password'. Because this API was used in the very high-level functions, as well as the actual low-level implementation, it was not possible to callback for a password when it was actually needed, nor specify a realm instead of a domain.

The solution was to replace all these arguments with a single context pointer, on which the low-level code may now inquire for the information it actually requires, and which may be expanded without changing all the layers in-between. Likewise, the command-line parsing code now has a single place to fill in the information it knows, as well as how well it knows it (guessed from an environment variable, specified on the command line etc).

Interfaces

`cli_credentials_init()` Create a new, uninitialised credentials context.

`cli_credentials_get_*()` (Various interfaces) Return a value off the context, potentially calling the supplied callback to get the information.

`cli_credentials_set_*()` (Various interfaces) Set a particular value onto the context, where it will be attached with `talloc()`. As well as specifying the value, the caller must specify 'how well' they know the value, as either `CRED_GUESS` (say user-name from `LOGNAME` environment variable) or `CRED_SPECIFIED` (user explicity specified).

`cli_credentials_guess()` Guess the username, password and domain from the available environment variables.

`cli_credentials_set_conf()` Specify basic details from the smb.conf, without reference to environment variables. The user-name and password (at least) must still be supplied separately.

`cli_credentials_set_machine_account()` Specify that the local machine account credentials should be used, with the values retrieved from the secrets database. This abstracts the database operations away from the callers.

`cli_credentials_set_anonymous()` It is common to require anonymous connections, and this call marks a context as anonymous (a call to `cli_credentials_set_conf()` is still required).

`cli_credentials_get_anonymous()` Because some protocols behave differently on anonymous connections, this returns `True` if the credentials are anonymous.

Server NTLM authentication

In a similar manner to the separation of client credentials from GENSEC, the issue of NTLM authentication has been abstracted away from the server-side GENSEC modules. Instead, the Samba 3.0-style authentication subsystem continues to handle authentication, as referenced from GENSEC, 'basic' session setups as well as remote NETLOGON operations.

GENSEC Interfaces

Internal Library

GENSEC provides a limited set of interfaces to the rest of Samba, in a deliberate policy to keep it as a separate subsystem within the Samba infrastructure. GENSEC loads it's modules in the same way as other Samba components do: each module provides an initialisation function, which registers the available operations with GENSEC.

Typically, subsystems within Samba only have to deal with the following primary APIs:

`gensec_client_start()` Start GENSEC in client mode

`gensec_server_start()` Start GENSEC in server mode

`gensec_start_mech_by_*()` (A number of similar APIs) Start a particular GENSEC mechanism. This is perhaps one of the more powerful aspects of GENSEC, as it ensures that GENSEC, not the caller, figures out the mapping between wire protocol IDs and actual security mechanisms. This allows for new security protocols to 'just work'.

`gensec_update()` Pass length-bounded packets of data between the peers, until both are satisfied with the negotiation.

`gensec_set_credentials`() This API links an existing credentials context to this GENSEC context.

Other APIs are provided to obtain information such as the name of the client. Because GENSEC is all `talloc()` based, all memory allocated by GENSEC is controlled within the talloc() system, and a call to `talloc_free(gensec_security)` will deallocate and cleanup the entire GENSEC system.[3]

If data encryption or integrity protection is required, then the following APIs provide this:

`gensec_sign_packet()` Produce a signature for the supplied packet

`gensec_check_packet()` Validate an incoming signature

`gensec_seal_packet()` Encrypt the supplied packet

`gensec_unseal_packet()` Validate and decrypt the incoming packet

`gensec_wrap()` 'Wrap' a packet, in a way determined by the mechanism, and which may sign, or encrypt the data if this was negotiated.

`gensec_unwrap()` 'Unwrap' a packet, in a way determined by the mechanism, and which may validate any signature, or decrypt the data if required.

The latter two APIs were introduced to cope with backing GENSEC onto the standard GSSAPI, which uses this format.

[3]For more information on talloc(), please see talloc_guide.txt in the root of the Samba4 source tree

C, Python etc Libraries

While a externally accessible C library is not currently available for GENSEC, it is hoped that by keeping a tight set of interfaces, it will not prove difficult to construct a stable library interface in the future. Likewise, the possibility to add Python bindings is very interesting.

`ntlm_auth`

Samba3 provides a binary utility known as `ntlm_auth` to allow external programs to use Samba as a partial outsourcing of their security subsystems. This allows programs such as Squid and mod_ntlm_winbind to use `ntlm_auth` without knowledge of the NTLMSSP or SPNEGO protocols, and without needing to talk to domain controllers directly. Samba4 continues this practice, and by calling `ntlm_auth` with the correct `--helper-protocol` argument, external programs may access GENSEC over a clean, replaceable stdio interface.

Module design

GENSEC uses modules to provide the actual implementation of each of the security protocols, and these register with the GENSEC subsystem at initialisation time. The GENSEC internal interface is very similar to that exposed to external users, except that each module may or may not provide certain capabilities: some modules, such as the current `gensec_krb5` module, may not be able to perform data encryption or integrity protection.

Each module declares a structure of function pointers, and those pointers left NULL are considered unimplemented. When designing a new module, it is best to look at current examples of other modules in the source tree, as these give a guide to the style, and expectations of new GENSEC modules. As an example, the NTLMSSP module initialisation is show in Figure 1.

The first few lines declare information about the module - the name for debugging, the protocol names for SASL (as used in LDAP, for example), the assigned number for use on DCE-RPC, and the OID that is used for GSSAPI and SPNEGO. Modules may be enabled or disabled by default, so as to allow experimental modules to be built, but inactive.

The Session Key function breaks all the abstractions that should be present in such a security system, but are required for operation on CIFS, due to the way that 'session keys' are used in SMB signing.

Future requirements

GENSEC and the other security subsystems with which it collaborates are not the end of the line in this area: there is still a lot of work to do, particularly as we try and assist other projects in the use of this infrastructure, and as we cope with newer requirements on the Samba code generally.

```
static const struct gensec_security_ops gensec_ntlmssp_security_ops = {
        .name            = "ntlmssp",
        .sasl_name       = "NTLM",
        .auth_type       = DCERPC_AUTH_TYPE_NTLMSSP,
        .oid             = GENSEC_OID_NTLMSSP,
        .enabled         = True,
        .client_start    = gensec_ntlmssp_client_start,
        .server_start    = gensec_ntlmssp_server_start,
        .update          = gensec_ntlmssp_update,
        .sig_size        = gensec_ntlmssp_sig_size,
        .sign_packet     = gensec_ntlmssp_sign_packet,
        .check_packet    = gensec_ntlmssp_check_packet,
        .seal_packet     = gensec_ntlmssp_seal_packet,
        .unseal_packet   = gensec_ntlmssp_unseal_packet,
        .wrap            = gensec_ntlmssp_wrap,
        .unwrap          = gensec_ntlmssp_unwrap,
        .session_key     = gensec_ntlmssp_session_key,
        .session_info    = gensec_ntlmssp_session_info,
        .have_feature    = gensec_ntlmssp_have_feature
};
```

Figure 1: NTLMSSP GENSEC module initialisation

Asynchronous request support

The particular area that looms as a change for GENSEC is that of asynchronous request support. This involves saving state regarding the progress of the GENSEC transaction into the context, such that the state machine can return to the main processing loop, and be recalled later. This is currently an issue in the NTLMSSP server, where we contact a remote domain controller, and we must process other packets while we wait for a reply. Likewise, we would hope not to block in the GENSEC client, while we wait for packets to return from a KDC we may need to talk to.

Moving beyond Samba

As GENSEC becomes more useful, we also should seriously consider how it is best used outside Samba. For example, many projects are now using Samba's ntlm_auth to handle NTLMSSP authentication, but at some point soon they may wish to handle SPNEGO, and more particularly the signing and sealing of the subsequent data streams. It is likely that this may require GENSEC to be rewritten into an external library, (as it is rather Samba-specific at this point), but this is yet to be determined.

References

[1] Jeremy Allison. Security soup. Jan 2004. URL http://www.linux.org.au/conf/2004/eventrecord/LCA2004-cd/papers/securitysoup

[2] Eric Glass. *The NTLM Authentication Protocol*, 2003. URL `http://davenport.sourceforge.net/ntlm.html`.

[3] J. Kohl and C. Neuman. *The Kerberos Network Authentication Service (V5)*, September 1993. URL `ftp://ftp.isi.edu/in-notes/rfc1510.txt`. RFC 1510.

[4] Microsoft Corporation. *The Security Support Provider Interface*, 1999. URL `http://www.microsoft.com/windows2000/techinfo/howitworks/security/sspi2000.a`

[5] MIT. Kerberos: The network authentication protocol. URL `http://web.mit.edu/kerberos/www/`. 2004.

[6] Open Group. *The Open Group ActiveX Core Technology Reference*, chapter 11 - NTLM. URL `http://www.opengroup.org/comsource/techref2/NCH1222X.HTM`.

[7] Sanj Surati and Michael Muckin. Dec 2002. URL `http://msdn.microsoft.com/library/en-us/dnsecure/html/http-sso-2.asp`.