Pluggable Authentication Module for Windows NT

Naomaru Itoi
Peter Honeyman

Center for Information Technology Integration
University of Michigan
Ann Arbor

Abstract

To meet the challenge of integrating new methods and technologies into the Internet security framework, it is useful to hide low-level authentication mechanisms from application programmers, system administrators, and users, replacing them with abstractions at a higher level. The Pluggable Authentication Method approach popular in Linux, Solaris, and CDE offers one such abstraction.

To implement PAM in NT, we replaced the standard Graphical Identification and Authentication module with one that processes PAM tables. This provides security administrators with a flexible tool to plan and implement authentication policy across a wide range of computing platforms.

GINA is woven into the NT logon procedure, making it a difficult module to test and debug. Our PAM-based GINA solves this problem by allowing authentication mechanisms to be replaced and tested without forcing a reboot.

1 Introduction

Security technologies are constantly evolving to meet the demands of Internet services. For example, network authentication protocols such as Kerberos [SNS88, KNT91], and Netware 4.0 [REF] undergo periodic revision to meet new challenges. Similarly, the basis of secure authentication evolves, replacing password-based methods with ones that depend on smartcards or biometrics.

To meet the challenge of integrating new methods and technologies into the Internet security framework, it is useful to hide low-level authentication mechanisms (or AMs) from application programmers, system administrators, and users, replacing them with abstractions at a higher level. This allows the underlying mechanisms to be replaced as needed without changing APIs, documentation, or the "user experience."

The Pluggable Authentication Module (PAM) framework provides an attractive abstraction for user identification and authentication. PAM defines a generic API for authentication mechanisms, hiding the underlying mechanisms. This provides for easy replacement of authentication components and offers an attractive solution to the "single sign-on" problem for users [SS95

PAM is implemented in Linux, Solaris, and the Common Desktop Environment (CDE), and is a *de facto* standard authentication framework. OSF and IETF are also conducting PAM standardization efforts.

In this paper, we describe an implementation of PAM in Windows NT. NT has a replaceable component for user authentication, called Graphical Identification and Authentication (GINA) [MS96]. We implement a subset of PAM, which we call NI_PAM, and integrate it into NT via the GINA. As it turns out, NI_PAM also greatly helps development and configuration of new GINA methods.

The remainder of this paper is organized as follows. In Section 2, we describe PAM. In Section 3, we explain the Windows NT authentication mechanism, especially GINA, and state problems with GINA. Section 4 details our design of NI_PAM to improve GINA. We discuss results of the project and conclude in Section 5.

2 Pluggable Authentication Module

PAM is a framework that provides a generic way of authenticating users. PAM makes authentication components replaceable, defines a generic API for authentication mechanisms, and provides single sign-on for users. We discuss the role of each of these features.

2.1 Replaceable authentication modules

A system administrator may be required to replace an authentication mechanism and its components when new mechanism are developed, or because of a change in security policy. PAM provides a dynamically configurable ("pluggable") authentication mechanism by separating applications such as login, ftp, telnet, *etc*. from low-level authentication modules such as Kerberos, Netware, *etc*. PAM is controlled by a configuration table that defines the behavior of application programs. The configuration table is maintained by system administrators. Table 2.1 shows a simple example of the PAM configuration table.

Service	module_type	flag	module_path	options
login	auth	required	pam_unix_auth.so	
login	auth	required	pam_kerberos.so	use_first_pass
login	auth	optional	pam_netware.so	use_mapped_pass

Table 2.1: Sample PAM configuration table

The example, a configuration for a login application, uses three authentication mechanisms: UNIX local, Kerberos, and Netware. The `required` flag on a row means that a user must be authenticated by the listed mechanism to be accepted by the login program. The `optional` flag indicates that the system should attempt to authenticate the user to the listed mechanism, but the user can be accepted by login even if this authentication fails.

In the example, the login program requires a new user to be authenticated successfully with UNIX local authentication and Kerberos authentication. If she is authenticated by both, she is allowed to login. In addition, the login program attempts to authenticate the user to Netware, but this step is optional, so the user can login even if Netware authentication fails.

Changing the behavior of an application program is easily accomplished by modifying a PAM configuration table. For example, one can require login to use S/KEY authentication mechanism by adding a line to the configuration table:

```
login     auth          required   pam_skey.so         use_first_pass
```

Removing the requirement for Kerberos authentication is accomplished by removing the line for `pam_kerberos.so`. Observe that this modification can be made without (necessarily) writing, compiling, or installing any code.

2.2 Generic API for application programs

Without PAM, applications must engage mechanisms specific to the various authentication methods, e.g., by calling `krb5_get_in_tkt()`. With PAM, applications do not call authentication mechanisms directly. Instead, they call PAM API functions such as `pam_authenticate()`. Figure 2.2 shows examples of the PAM API.

`pam_start()`	Initiate an authentication transaction
`pam_end()`	Terminate the authentication transaction
`pam_authenticate()`	Verify the identity of the current user
`pam_chtok()`	Change password

Table 2.2: PAM API examples

As a high-level abstraction, the PAM API hides details of low-level authentication mechanisms, which insulates applications written with PAM API from authentication mechanisms. Modification of the PAM configuration table does not affect application programs. For example, in Section 2.1, the login program is not changed when an authentication mechanism is added or removed.

2.3 Single Sign-On

A computing infrastructure often supports multiple authentication mechanisms. In such an environment a user is often required to remember the various ways of authenticating, multiple user names, and proper passwords. Users prefer to remember and use only one password, and to obtain access to all resources by typing the password only once. This feature is called single sign-on [HAR95].

PAM provides single sign-on to users by sharing passwords among authentication mechanisms. In Figure 2.1, the `use_first_pass` option specified for Kerberos authentication directs PAM to use the password that the user provided. In this case, the user has the same UNIX and Kerberos password. The `use_mapped_pass` in the Netware entry specifies that the password given by the user is not the Netware password, but is used to derive the Netware password, which might be necessary if Netware has special rules on password formats not enforced by other authentication methods. This lets PAM give passwords to all authentication mechanisms, while asking a user to type a password only once.

2.4 Example

To show the advantages of using PAM, we look at an example of how PAM simplifies login with multiple authentication mechanisms. Figure 2.3 depicts a computer system without PAM. The login program requires UNIX Local and Kerberos authentication. A method for authenticating in both UNIX and Kerberos must be hard coded into the login program. If a system administrator wants to enable access to Netware file service, she must add Netware authentication to the login by hard coding an authentication method for Netware into the login program. Worse yet, if a user wants to use `ftp` that accesses Netware, someone must then code this method into the `ftp` application, getting little advantage from the earlier modifications to the login application.

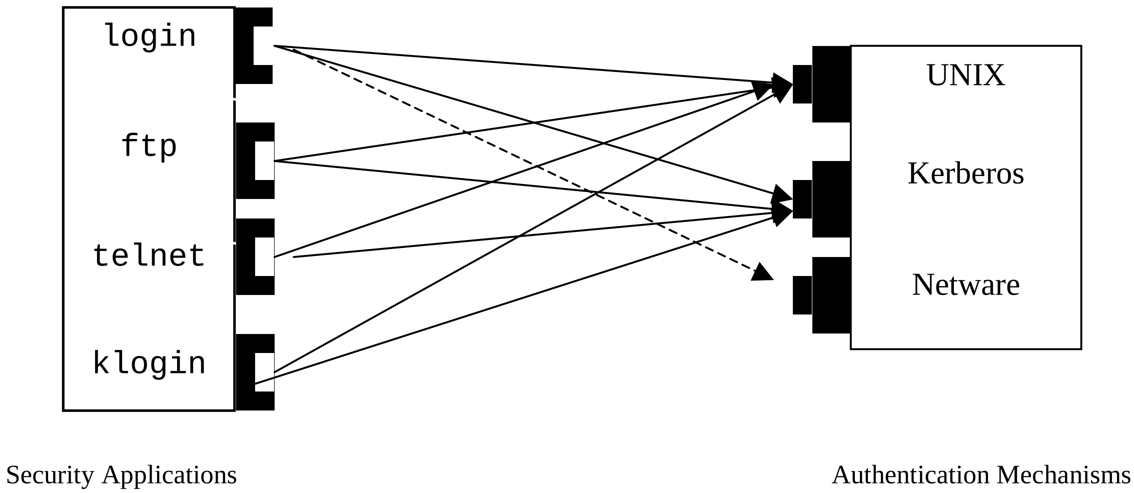

Figure 2.3: UNIX authentication without PAM

Figure 2.4 shows how PAM solves the problem.

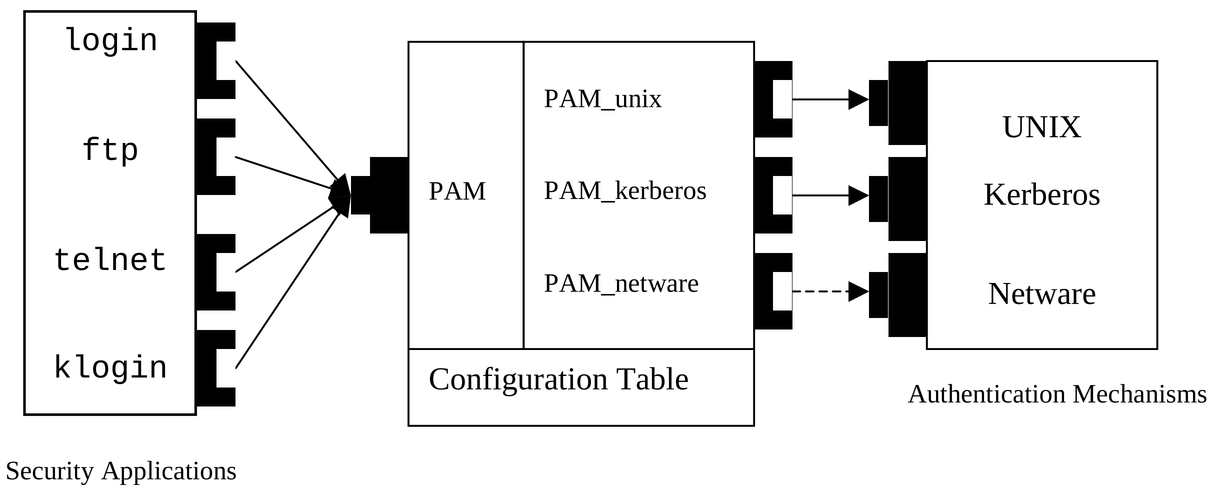

Figure 2.4: UNIX authentication with PAM

Here, applications are written with a single, generic PAM API call. Now the system administrator can add Netware authentication to the login program without writing any code in login or anywhere else. Instead, he adds one line in the PAM configuration table:

```
login    auth         required   pam_netware.so    use_first_pass
```

As shown in the example, PAM simplifies task of system administrators, application programmers, and users. As PAM is a part of Linux, Solaris, and CDE, it is becoming increasingly popular.

3 GINA: the NT authentication module

The component of NT responsible for interactive logon is called `Winlogon.exe`. To allow software developers to modify the authentication system of NT, *e.g.*, to support new authentication mechanisms, a part of `Winlogon` is designed to be replaceable. That part is the Graphical Identification and Authentication DLL, or GINA [MS96].

3.1 GINA

GINA, a dynamic link library loaded by `Winlogon`, is responsible for authenticating users. Authentication action is required when a user tries to logon, lock the screen, or logoff. At these times, `Winlogon` calls certain functions of GINA. In Windows NT, a user types SAS (Ctrl-Alt-Del keys combination) to indicate she wants to logon, logoff, or lock the screen. Table 3.1 shows some examples of GINA APIs called by `Winlogon`.

`WlxInitialize()`	Initialize GINA
`WlxLoggedOutSAS()`	Called when SAS is received and no user is logged on. This indicates that a logon attempt is made. GINA can return indicating the user is authenticated, or rejected
`WlxWkstaLockedSAS`	Called when SAS is received and the workstation is locked. GINA can return indicating the workstation is to remain locked, or the workstation is to be unlocked
`WlxLogoff()`	Called to notify GINA of a logoff operation on the computer

Table 3.1: GINA API examples

To explain how GINA works, Figure 3.2 shows interaction among a user, Winlogon.exe, and GINA.dll.

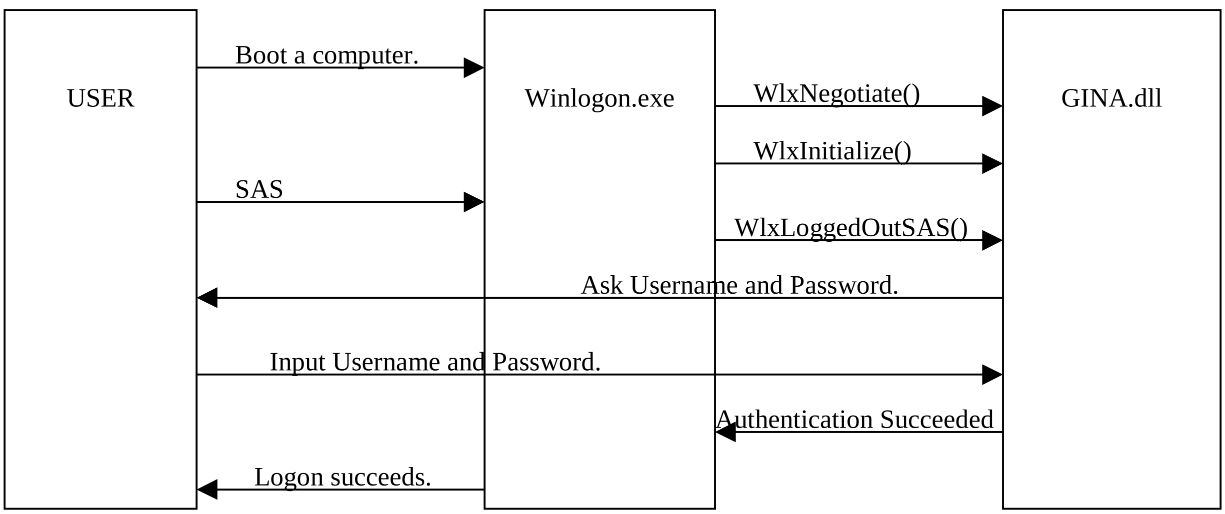

Figure 3.2: Interaction among a user, Winlogon, and GINA.

3.2 Problems with GINA

GINA is replaceable, which allows software developers to develop their own authentication mechanism. Many GINA's have been written for various authentication mechanisms, *e.g.* at the University of Michigan, we see frequent use of a Kerberos4-GINA, a Netware-GINA, *etc*. However, the structure of GINA poses some problems, which we now describe.

GINA is not PAM

While GINA is replaceable from Winlogon, GINA does not have the essential feature of PAM: authentication mechanisms are not replaceable in GINA. Figure 3.3 shows the standard Microsoft GINA, which authenticates a user to the NT local security system.

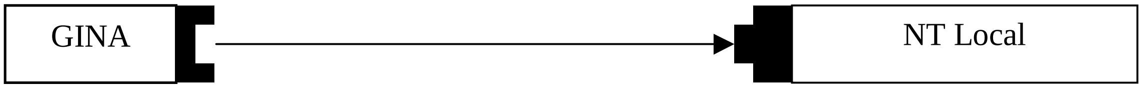

Figure 3.3: Microsoft GINA

Figure 3.4 shows Kerberos4-GINA, which authenticates a user to Kerberos4 as well as to Windows NT local.

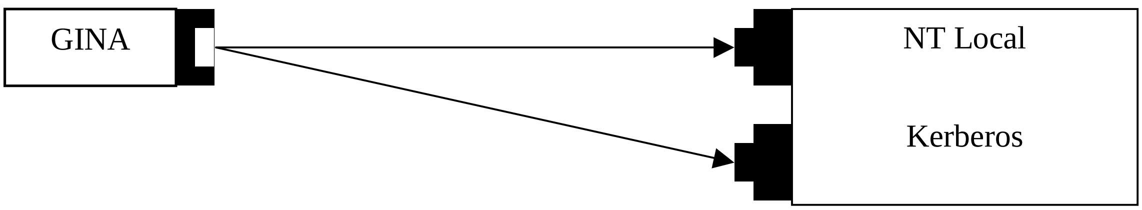

Figure 3.4: Kerberos4-GINA

We see that GINA has the same problems discussed in the context of UNIX login in Section 2. The login application in Figure 2.3 requires hand coding for each authentication method. GINA shares this architectural inconvenience

By implementing PAM in GINA, we can solve the problem. Figure 3.5 shows our view of GINA with PAM. For clarity, we refer to our implementation of PAM in NT “NI_PAM”.

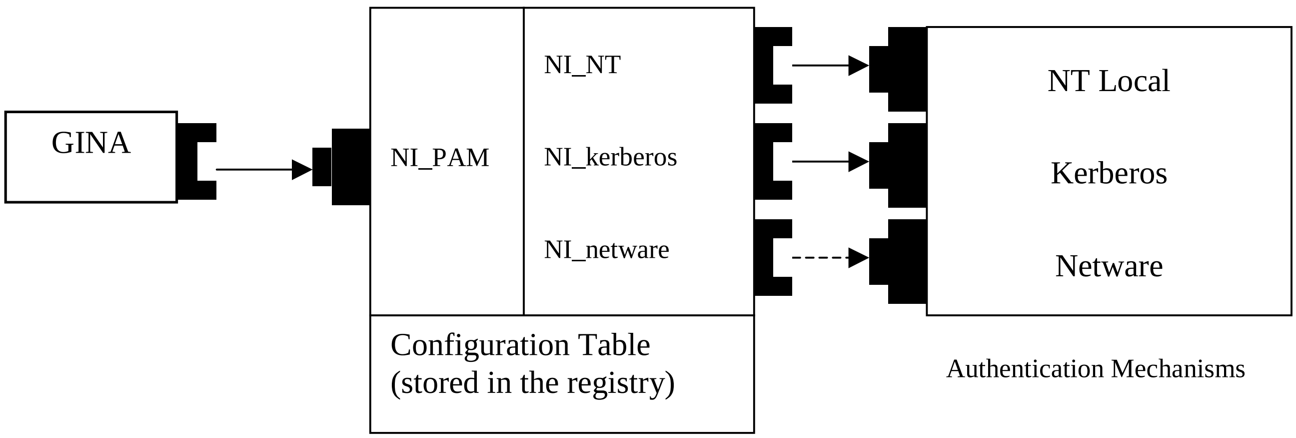

Figure 3.5: GINA with NI_PAM

GINA is hard to develop

Because UNIX applications run as user processes, applications such as login and telnet are relatively straightforward to develop and debug. GINA, on the other hand, is deeply integrated in NT logon, which makes it much harder to develop: a GINA developer must reboot the workstation for each new test of GINA. Furthermore, if the GINA does not work properly, the workstation hangs and cannot be restarted by usual methods. In addition, a GINA developer must use a debugger in an unusual way because GINA runs before logon is accomplished. In our experience, these two problems make GINA development very time consuming. In contrast, NI_PAM eases GINA development by letting us test new authentication mechanisms without installing a new GINA, which obviates rebooting.

Lack of cross-platform security administration method

Implementing an authentication policy in NT is much different than in UNIX. The contrast between replacing GINA in Windows and adjusting PAM configuration tables in UNIX is particularly striking. This difference is a burden on system administrators charged with cross-platform security administration, who are forced to be familiar with both approaches. By installing NI_PAM in Windows NT, system administrators can use a common configuration method across UNIX and NT platforms. We are finding this to be a very popular feature.

4 Design of NI_PAM

As we saw in the previous section, implementing PAM in Windows NT aids the development and configuration of NT authentication. In undertaking the development of NI_PAM, we adhered to the following principles.

- NI_PAM employs configuration tables identical to PAM

  We believe the table-based approach to configuration is sufficiently powerful and easy to use for NT authentication. Furthermore, identical configuration tables simplifies cross-platform security administration, as described in Section 3.2.

- The configuration table is stored in the Windows NT registry.

  In NT, the *registry* contains configuration information for both operating system and application software, *e.g.*, version numbers for software, search paths, names of a DLL called by the operating system, *etc.*

- Application programs other than GINA can call NI_PAM.

- NI_PAM assumes only one password for all authentication mechanisms.

We do not employ a password-mapping scheme in PAM. We discuss this further in Section 5.3.

4.1 NI_PAM components

Our implementation consists of three components: NI_PAM, NI_GINA, and the modules that implement authentication mechanisms, which we now describe.

NI_PAM

An application program such as GINA calls NI_PAM.dll. This is the dynamic link library that implements PAM in NT. On being called with a username, realm, and password, NI_PAM reads the configuration table in the registry and calls authentication methods with the combined arguments. NI_PAM receives a return value from each authentication module. Depending on the requirements of the PAM configuration table, NI_PAM returns to the application with success or failure.

For example, consider the (simplified) configuration table shown in Table 4.1.

Service	Flag	module_path
login	Required	ni_krb4.dll
login	Optional	ni_nw.dll

Table 4.1: A simple example of the NI_PAM configuration table.

When activated by Winlogon, GINA calls ni_authenticate with the username, realm, and password as arguments. By reading the configuration table, NI_PAM knows it should call the Kerberos-4 specific module, ni_krb4.dll, and the Netware specific module, ni_nw.dll. So NI_PAM calls ni_sm_authenticate(username, realm, password) of ni_krb4.dll and ni_nw.dll. The flag for ni_nw.dll is “optional”, so NI_PAM returns NI_SUCCESS if ni_krb4.dll succeeds in authenticating the user.

NI_GINA

NI_GINA, the GINA used with NI_PAM, is called by Winlogon and calls NI_PAM functions. For example, when a user tries to login, Winlogon calls `WlxLoggedOutSAS()` in NI_GINA. NI_GINA requests username, realm, and password from the user, and sends the information to NI_PAM, calling ni_authenticate with username, realm, and password as arguments. If NI_PAM returns NI_SUCCESS, NI_GINA does some post-authentication work, and returns SUCCESS to Winlogon.

Authentication Mechanism Specific Module

An Authentication Mechanism Specific Module (AMSM) implements mechanism specific authentication. For example, the AMSM for Kerberos-5, ni_krb5.dll, calls `krb5_get_in_tkt()` to get initial credentials from a Kerberos-5 KDC; the AMSM for Netware-4.0, ni_nw.dll), calls `NWDSLogin()`; *etc.*

Figure 4.2 shows the structure of the whole system that constitutes NI_PAM.

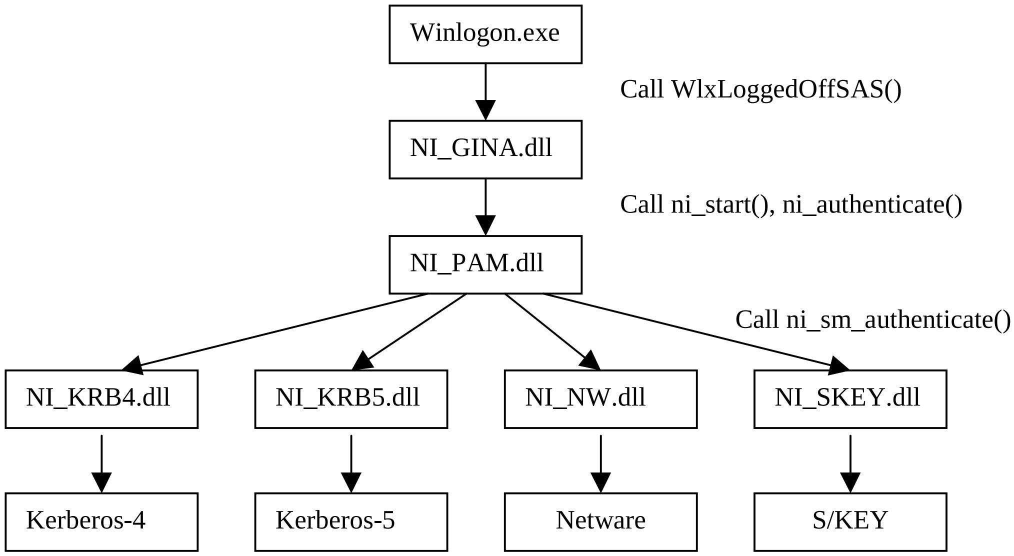

Figure 4.2: NI_PAM structure.

4.2 NI_PAM APIs

DLLs in NI_PAM structure export the following functions.

NI_GINA.dll Exports all functions defined in GINA [MS96].

NI_PAM.dll Defines the following data structure and exports the following functions.

```
// Data Structure
typedef struct _ni_struct {
      WCHAR username[StrLen];          // user name
      WCHAR domain[StrLen];            // domain name
      WCHAR password[StrLen];          // password
      WCHAR oldPassword[StrLen];       // old password, for change pw.
      WCHAR newPassword[StrLen];       // new password, for change pw.
      INT   value;                     // used for smartcard.
} NISTRUCT;

// prototypes
BOOL WINAPI ni_start(HANDLE hWlx, PVOID pWinlogonFunctions);
      // Start NI_PAM transaction, read configuration tables.

BOOL WINAPI ni_obtainpass(NISTRUCT *nistruct);
      // Attempt to obtain username and password from smartcard AMSM.

BOOL WINAPI ni_authenticate(NISTRUCT *);
      // Attempt authentication to AM's, following the configuration tables.

BOOL WINAPI ni_logout();
      // Logout from AM's.

BOOL WINAPI ni_end();
      // END NI_PAM transaction.
```

AMSM **Exports the following functions.**

```
BOOL WINAPI ni_sm_authenticate(NISTRUCT *niStruct);
      // Authenticate a user to an authentication mechanism.
```

```
BOOL WINAPI ni_sm_logout();
```
// Logout from an authentication mechanism; destroy credentials.

```
BOOL WINAPI ni_sm_obtainpass(NISTRUCT *nistruct);
```
// Returns username and password if this is smartcard AMSM. Else, it returns NULL.

4.3 NI_PAM interaction

To show how the whole system works, we follow the function calls that occur in user authentication, given the structure in Figure 4.2 and the following configuration table.

Service	module_type	Flag	module_path
login	auth	Required	ni_krb4.dll
login	auth	Required	ni_krb5.dll
login	auth	Optional	ni_nw.dll

- User issues SAS. Winlogon receives SAS. Winlogon.exe calls WlxLoggedOffSAS() in NI_GINA.
- NI_GINA.dll receives WlxLoggedOffSAS() and calls ni_start() in NI_PAM.
- NI_PAM.dll receives ni_start(), initializes global valuables, and reads the configuration table, and returns.
- NI_GINA demands (username, realm, password) from the user.
- NI_GINA.dll calls ni_authenticate() of NI_PAM.
- NI_PAM.dll receives ni_authenticate(), calls ni_sm_authenticate() of NI_KRB4.dll, NI_KRB5.dll, and NI_NW.dll.
- NI_KRB4.dll, NI_KRB5.dll, NI_NW.dll attempt to authenticate the user. They return NI_SUCCESS if they succeed otherwise NI_FAILURE.
- If NI_KRB4.dll and NI_KRB5.dll succeed, NI_PAM.dll returns NI_SUCCESS, otherwise NI_FAILURE
- If NI_PAM.dll returns NI_SUCCESS, NI_GINA.dll attempts local NT authentication. If the attempt succeeds, NI_GINA returns SUCCESS to Winlogon, else authentication fails, and NI_GINA returns FAILURE to Winlogon.
- Winlogon receives the result from NI_GINA. If success, user can logon. If not, she is rejected.

5 Results and Future Work

In this section we describe the status of the project, discuss the results, state future directions, and conclude.

5.1 NI_PAM status

NI_PAM is still experimental software. Here is the status as of this writing.

- NI_PAM.dll is implemented and tested. We still run it in Debug mode, not yet in Release mode.
- NI_KRB4.dll and NI_NW.dll are implemented and tested. They support authentication and changing password.
- NI_KRB5.dll is implemented and tested. It supports authentication.
- NI_SC.dll (smartcard AMSM) is implemented and tested. Currently, this module stores passwords in the clear.

- NI_GINA.dll is implemented and tested, but needs more work before being deployed, including password changing, screen lock, static account, *etc.*

5.2 Results

Without NI_PAM, limitations in NT debugging facilities make a GINA very difficult to develop. Separation of NI_GINA from the other parts involved in authentication (NI_PAM and AMSMs) helps the development by allowing us to develop NI_PAM and AMSM's apart from NI_GINA. Indeed, we were able to implement NI_GINA by adding fewer than 20 lines of PAM-specific code to an existing GINA. Thereafter, we were able to use NT's powerful debugging tools to develop NI_PAM and a handful of AMSMs.

Adding and modifying AMSMs in the future is straightforward. Dynamic configuration of the NT authentication system is now as easy as PAM because NI_PAM uses the same configuration table as PAM. NI_PAM also provides single sign-on to a user, and with a smartcard module, the user does not have to type her password even once. Although we have not implemented any applications other than NI_GINA that uses NI_PAM, the NI_PAM API is generic and rich enough to be used by security-sensitive applications other than NI_GINA.

5.3 Future Directions

Static Account

With tens of thousands of mobile users in our computing environment, it is not feasible to store account information for every potential user in every NT machine that she may use. An alternative would be to allow a user to authenticate to a static account (*e.g.* "guest"). A system keeps users' account and profiles information in server machines, and protects the database with some authentication mechanism (e.g. Kerberos-4). After authenticating, the system could retrieve user account and profiles from a secure server and use that information to personalize the local machine.

NI_PAM does not support static accounts and profiles, but they seem to be essential to supporting NT in large-scale computing environments, so we plan to address this problem.

Password consistency

Consistency in the various password databases is a hard problem. In the current NI_PAM, the password is stored in each authentication mechanism; *e.g.,* Kerberos has its own password database, as does Netware, *etc.* In this scheme, it is hard to keep consistency in passwords in AMs. When a user attempts to change her password, some AMs may fail while others succeed.

For example, suppose a user wants to change password in both Kerberos and Netware. NI_PAM calls ni_chpass(old_password, new_password). Now suppose that after successfully changing her Kerberos password, the system suffers some sort of network failure and that Netware fails to change the user's password entry. The system has inconsistent password databases – a new password for Kerberos, but an old password for Netware – and NI_PAM can no longer provide single sign-on.

Use of a single password for all AM's has other problems.

- Security strength of the whole system is reduced to the strength of the weakest AM. A password compromised in the weakest AM yields the password to all AMs.
- Requirements for passwords are different among AMs. For example, some AM may require a password length more than 8 letters, while another AM may require less than 8 letters.

NI_PAM needs a way to use different passwords for each AM, while still preserving single sign-on. A common approach to this problem is to use an encrypted "key ring." In essence, a key ring acts as a secure password database, protected by an AM. The password entered by the user is the one that unseals the key ring. After authenticating to the AM guarding the database, the system can obtain passwords for all other

AMs from the key ring. We are considering this approach, with an eye toward storing the key ring on a secure directory server.

Smartcard

Another way to store secrets securely is to store them on a secure token, such as a smartcard [H+97]. A smartcard can improve password-based authentication; while passwords are short, easy-to-guess, and stored in users' heads, keys in smartcards are long, randomly generated, and stored in a physically protected smartcard.

In addition to computer authentication, smartcards have many other potential applications, especially in the realm of electronic commerce. We will eventually see all these functions integrated into smartcards. NT authentication with smartcards is one step toward these goals.

GINA details

Our focus is on authentication, yet GINA has other important roles to play. NI_GINA needs a better GUI, AFS support, and support for screen lock before we can deploy it.

5.4 Conclusion

We implemented a dynamically configurable authentication, a generic API for GINA and application programs, and a single sign-on in NT. Our approach was modeled after the success of UNIX PAM at solving these problems. Although we found some aspects of the NT authentication system hard to work with, NI_GINA made life a lot easier. We have implemented NI_GINA, which integrates PAM functionality into GINA, and are planning for campus deployment.

Acknowledgement

We thank Andy Adamson for advice and ideas. We are grateful to Allan Bjorklund for letting us use his GINA as a starting point.

References

[SS95] V. Samar and R. Schemers, "Unified Login with Pluggable Authentication Modules (PAM)," Request For Comments: 86.0, Open Software Foundation (October 1995).

[HAR95] Peter Honeyman, William A. Adamson and Jim Rees, "Joining Security Realms: A Single Login for Netware and Kerberos," *Proc. of Fifth USENIX UNIX Security Symp.*, Salt Lake City (June 1995).

[H+97] Peter Honeyman, Andy Adamson, Kevin Coffman, Janani Janakiraman, Rob Jerdonek, and Jim Rees, "Secure Videoconferencing," *Proc. of Seventh USENIX Security Symp.*, pp. 123-130 (January, 1998).

[KNT91] John T. Kohl, B. Clifford Neuman, and Theodore Y. T'so, "The Evolution of the Kerberos Authentication System," *Distributed Open Systems*, pp. 78-94. IEEE Computer Society Press (1994).

[MS96] Microsoft, "Winlogon User Interface," *Microsoft Win32 Software Development Kit for Microsoft Windows* (1996)

[SNS88] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller, "Kerberos: An Authentication Service for Open Network Systems," *Proc. of the Winter 1988 USENIX Conf.* (February 1988).

Contact

We plan to make NI_PAM freely available. Please contact Naomaru Itoi at itoi@eecs.umich.edu. The project homepage is http://www-personal.engin.umich.edu/~itoi/