User-Centric Account Management and Heterogeneous Password Changing

Doug Hughes – Auburn University

ABSTRACT

There are a plethora of password changing programs available for users and systems administrators to use. Most of the existing password changers, however, fall short in some way. Either they are still text based, requiring users to use unfamiliar tools in unfamiliar environments, or they are part of an all encompassing framework – designed to completely supplant the entire existing account creation, maintenance and distribution process. Our program is designed, first, to be usable without training or human support. Second, it is designed to use existing distribution databases such as NIS and LDAP. We integrate many of the features of prior password changers such as the Cracklib library, character classes, and rules for password selection. We try to provide easy extensibility both in terms of database support and by providing other user-focussed programs that use the same authentication framework.

Introduction

Auburn University College of Engineering has two primary platforms for supporting a broad clientele. These are Windows boxes which use SMB [1] for file service, and UNIX boxes which use NFS for file service. In order to use any network services, a user must be authenticated. For the UNIX side, this means NIS, NIS+, or LDAP.

To provide authentication to NT boxes we use Samba [2]. To do this semi-securely [3] requires a separate encrypted password database. In their respective, encrypted formats, the UNIX password and the Microsoft password are incompatible, irreversible one way hashes; there is no method to generate one from the other. This means neither system's builtin password changing mechanisms can be used. This irreversibility problem can be applied by extension to other one way password formats such as BSD44, Apple formats (e.g., CAP [4]) and several Linux formats.

Early in 1999 we decided that we should replace the existing – functional but archaic – text-based password changer with a Web-based mechanism. The old system required users to login to a UNIX machine to change their passwords. Many of our new users had no experience with applications like telnet, ssh, or even UNIX. Text based solutions requiring login to an unfamiliar environment were difficult to support. A Web design allowed greater platform availability and improved usability over our previous system. We also had an opportunity to revise the entire password changing system and integrate other services such as email forwarding.

We wrote our code primarily in PHP [5]. Some functions required a PHP loadable module written in C. One helper program was written in Perl.

Prior Art

A lot of work has been put into account management systems over the years. Many USENIX, LISA, and even *;login:* articles have been written concerning these works. Nearly all of them have a systems administrator focus, which is natural when one considers that systems administrators are always trying to make their jobs easier. However, the works tend to focus on systems administrator specific tasks such as creating, adding, deleting, and ongoing maintenance of accounts. When usability was addressed, it was usually most evident from a systems administrator's point of view: make less work; provide more scalability.

Auburn University College of Engineering (hereafter referred to as AUCOE) already had account maintenance tools in place with no obvious need for wholesale replacement. We required an intuitive user interface – to allow users to be able to do simple tasks on their accounts without any sort of training – and minimal support.

The main differences between the AUCOE framework and prior implementations are:

1. It integrates prior work and tools such as Cracklib [6], character classes, etcetera. (some, but not all, of the others also do this.)
2. It provides easy extensibility to other formats such as LDAP and custom databases.
3. It is much simpler in scope than the others. Some of them are enormous and handle everything from account creation to interfacing with a human resources database. The AUCOE changer provides a means for users to easily change their passwords.
4. Our main focus is user usability vs. administrator usability.
5. The incremental approach builds on top of existing frameworks such as NIS, NIS+, Samba, and flat files, rather than replacing them.
6. Decoupling password choosing from distribution and propagation allows us to not have to worry as much about locking and conflict resolution.

The author was not able to gather meaningful data – because of age and lack of access to the original papers – on these password and account management systems: Maryland [7], ACMAINT [8], Apollo [9]. References are provided at the end for those having hardcopy proceedings.

Many of the features of the following command-line, pro-active password checkers have been integrated into our password changer: npasswd [10], passwd+ [11], ANLpasswd [12], Epasswd [13]. The password vetting routines from these programs tend to be tightly coupled with an existing command-line interface and propagation mechanism. Rather than extracting the proactive changing logic and rules from any of these programs, it was easier for us to use Cracklib and construct the various password rules and classes in native PHP.

Information about current availability of and updates to other user account management systems was difficult to collect. Some of the systems listed may have had updates more recent than the author was aware. Some of them were designed to manage access control where certain users are only authorized to use certain machines, or the home directory may differ based upon the machine. All of them had, as a primary focus, the goals of modernizing and automating the maintenance of user accounts at a site; password changing, password synchronization, and usability, if mentioned at all, were typically secondary. The most currently relevant systems are compared in Table 1 based upon the following criteria:

- Design philosophy
- Custom database vs. existing database (e.g., NIS, LDAP)
- Distribution and synchronization methods for accounts and passwords
- Extensibility – perceived degree of difficulty of adding a new output format, heterogeneous machines, configuration intricacies, etc.
- User interface focus
- Language(s) written using and/or configured with
- Release status (may be out of date)
- Other characteristics that may be of interest

Genesis

The framework used by the AUCOE password changing system came into being ad hoc. The project that originated the framework provided a means by which users could forward their email. The previous forwarding program was text-based and required users to login to a UNIX machine to run it. The number of users wishing to forward their email was growing. The time spent supporting these non-UNIX users was growing proportionately.

We desired to avoid CGI and its inherent security difficulties and call-out overhead. After a short period of investigation, comparing the tradeoffs of mod_perl [19] versus learning PHP, a new language, the author chose PHP. PHP combined C and Perl syntax without the special variable cruft. PHP had support for persistent file handles. Finally, PHP appeared to be easier for non-experts to learn and use quickly.

Name	Design Philosophy	Database Type	Distrib./ Sync.	Extensibility
Shuse [14]	sync accts across mult mach's with central database	custom, central	NFS, NIS, FTP, sockets	uses expect
Agus [15]	acct. over multi archs using CCSO [13]	custom (fixed?)	Kerberos, VMS, Unix, etc.	fixed DB but godo for new accounts
NAMS [16]	client/server daemon with modules replaces NIS	custom but open (ASCII key & data)	TCP/IP socket (client/server)	modular design
Accountworks [17]	ease hiring process and account creation	Sybase (other..)	central client/server but no passwd sync	probably difficult
Ganymede [18]	provide central DB push to NIS, LDAP, etc.	Central, RAM, OO	NIS, NIS+, LDAP, etc	properly changing schemas not for timid
Auburn COE	Provide usability atop existing dist. mechanisms	use existing (NIS, etc.)	use existing (NIS, etc)	Use PHP, module, or other (e.g., TCP/IP)

Table 1: Survey of similar systems.

In keeping with the tenets of user-interface design, the password change form was designed to be simple but usable. User feedback was incorporated to make the form what it is today (Figure 1). The results were positive both in terms of user satisfaction and reduced support by administrative staff. We were encouraged to try again: with passwords.

Design Goals

Our system for changing passwords was designed to meet certain goals:

- It must have universal accessibility.
- It must be secure.
- It must be easy for novices to use.
- It must not rely upon extensive online help.
- It should *have access* to online help for users having trouble.
- It must not inhibit expert users.
- It must be scalable.
- It must provide clear and meaningful, jargon-free responses.
- It should build upon existing distribution and synchronization mechanisms: NIS, NIS+, Samba, LDAP.
- It should be easy to extend as other distribution and synchronization mechanisms become available.
- It should borrow techniques from prior password changing implementations.

We now provide finer detail on these goals. Why are they here? How are they achieved?

It Must Have Universal Accessibility

In other words, use the Web. It is time consuming to develop clients for various architectures. The flexibility and speed of dedicated, platform-dependent password clients was not indicated.

It Must Be Secure

Because users are providing information that gives access to their accounts, and because secure, local access cannot be assumed (though it could be enforced), it is imperative that *no* information be transmitted over the network in the clear. There are multiple paths of transmission involved. See the **Security** section for more details.

The Web server must verify the username and old password for authenticity. Our implementation provides multiple ways to do this.

1. Make the Web server a NIS slave.
   PHP can directly access DBM files, so verifying passwords by giving the Web user account read-access to the encrypted password database is possible. (Before anybody gets apoplectic, please continue reading.) If your Web server is single-purpose, contains no extra CGI, no other user scripts, and no unknown functionality, this is relatively safe.
2. Use the provided pwcheck daemon.
   The pwcheck daemon uses a secure algorithm to verify the username and password. If neither is correct, a non-specific negative result is returned: users are informed that either their username or password is incorrect.

Name	User Interface	Construct/ Config	Release Status	Other Features
Shuse [14]	text based	Tcl/Expect with a little bit of C	Only Sheridan College	
Agus [15]	unknown	90% Perl, 10% C	N/A at publication	account clusters, fine-grained machine access control
NAMS [16]	text based (customized npasswd)	C daemons, ASCII DB config	prev. ftp.cs.jmu.edu	account clusters
Accountworks [17]	Web based	DB driven + Perl, sybperl, Notes, sh, etc.	not available	designed for non-techies; huge scope
Ganymede [18]	Java Applets, count on training	140K+ java lines, web-based config	www.arlut.utexas. edu/gash2	DB limits, good access control
Auburn COE	Web forms	PHP, some C	www.eng. auburn. edu/~doug	easy to add new features, e.g., forward mail

Table 1b: Survey of similar systems, cont.

3. Use another authentication mechanism.
   While not provided, it would be fairly easy to interface with LDAP, NIS+, or another authentication database to verify a user's identify. One of the difficulties of this approach is choosing between the lesser of two evils. Do you wish to give your Web user root access to the entire authentication database or do you wish to provide a setuid root helper program that becomes the user prior to checking the encrypted password? This is the primary reason why the *pwcheck* mechanism is provided. It is discussed in more detail in the **Security** section.

It Must:

Be Easy For Novices To Use

Rely Upon Extensive Online Help, and

Provide Access To Online Help For Users Having Trouble

These three goals are complementary, and might even be merged into one. The distinctions among ease of use, having online help, and relying upon online help, though subtle, are important. The interface must be intuitive. One of the well-known facts of user-interface design is that users rarely, if ever, read any online help before attempting to do something. They usually dive in oblivious to the circling sharks. The interface **must** be intuitive.

If a user cannot sit down and use a password changing program without training or referring to a manual, the program should be re-evaluated. We have gone through several iterations of user feedback and modification. There is, however, a difference between using the program without requiring a manual and referring to supplemental help should the password choice be insufficient. We avoid the requirement to read online help before-hand by providing abbreviated rules for choosing a password right at the top of the Web form. The rules (Figure 2) are specific enough to be easily understood, but small enough to fit entirely on the form and in the browser window.

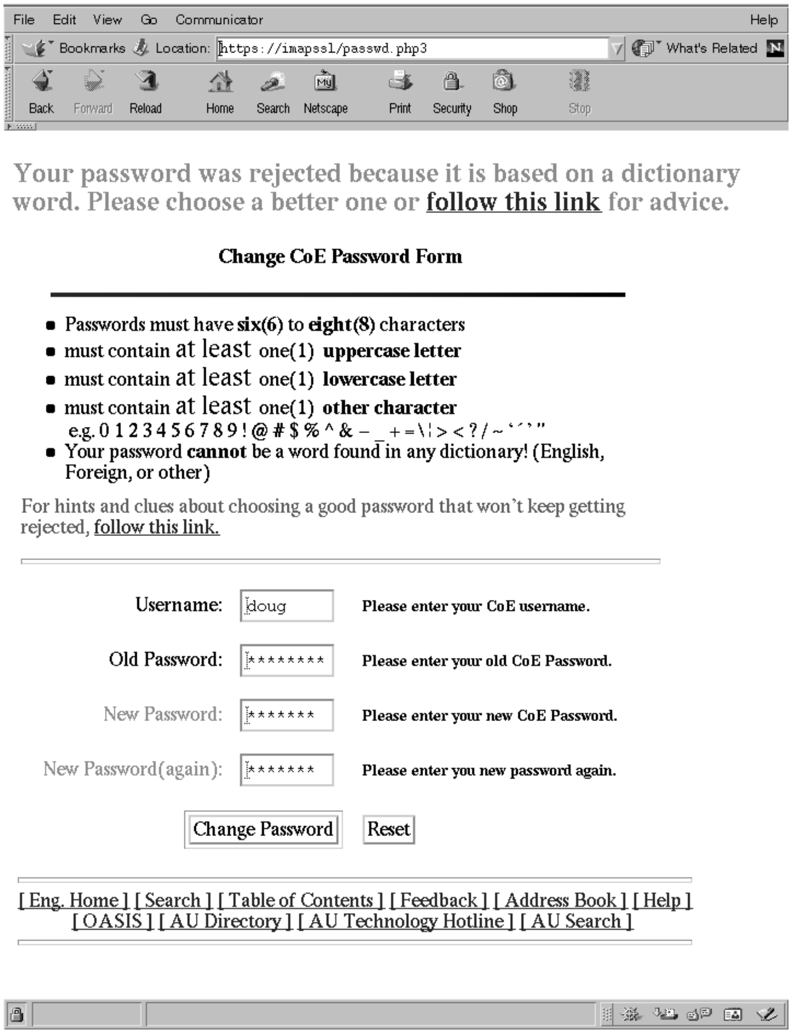

Figure 1: Sample user interface.

After each rejection of a password based upon the given rules, the user is given the opportunity to follow a link which gives suggestions for and examples of choosing a password. Failure messages are communicated in bold red letters in a larger font and written at the top of the page as the form is redrawn, giving clear visual feedback about problems.

It Must Not Inhibit Experts

By providing a short set of rules, but not overwhelming the user with extensive directions, the interface is usable by experts.

It Must Be Scalable

Most of the scalability (and locking) issues are pushed back to the distribution mechanism and thus avoided. The interface must still, however, be able to handle the case where a number of users connect simultaneously. The worst case scenario can probably be attributed to university environments when new accounts are generated at the beginning of each term. Even in this scenario, however, it is extremely unlikely for more than a few users to change their passwords at a given time. Many of our users opt to keep the random FIPS-181 [20] style passwords that they are issued. Even if there was a Freshman computer lab where the instructor dictated that all students must change their passwords, changes would still not be simultaneous. People read, type, and think about their passwords at different rates. The worst case would probably not be more than 5 to 10 simultaneous password changes, which is easily achieved even with a modest PC as the Web server. In practice, even in its 18 months of existence, it has been unusual to have more than one person trying to change a password in any given five minute interval.

Locking issues, similarly, are not a great concern. Most of the methods to change passwords in the existing back-end databases are already serialized. NIS, NIS+, and LDAP already have their own locking mechanisms. Even when we update our Samba password file with the encrypted MD5 hashes, we use a simple, Perl, single-threaded daemon to handle network requests from the Web server and update the flat Samba password file. (See the **Extensibility** section).

It Must Provide Clear and Meaningful, Jargon-free Feedback

User feedback issues have been partially previously addressed. At the risk of appearing to climb onto a soapbox, this is an area that is ignored far too often by systems administrators developing user interfaces for users. Users do not care – and should not be burdened – with messages about errors in some anonymous line of computer code. We systems administrators, as a community, are particularly guilty of these transgressions. The AUCOE program gives messages in plain English such as 'New

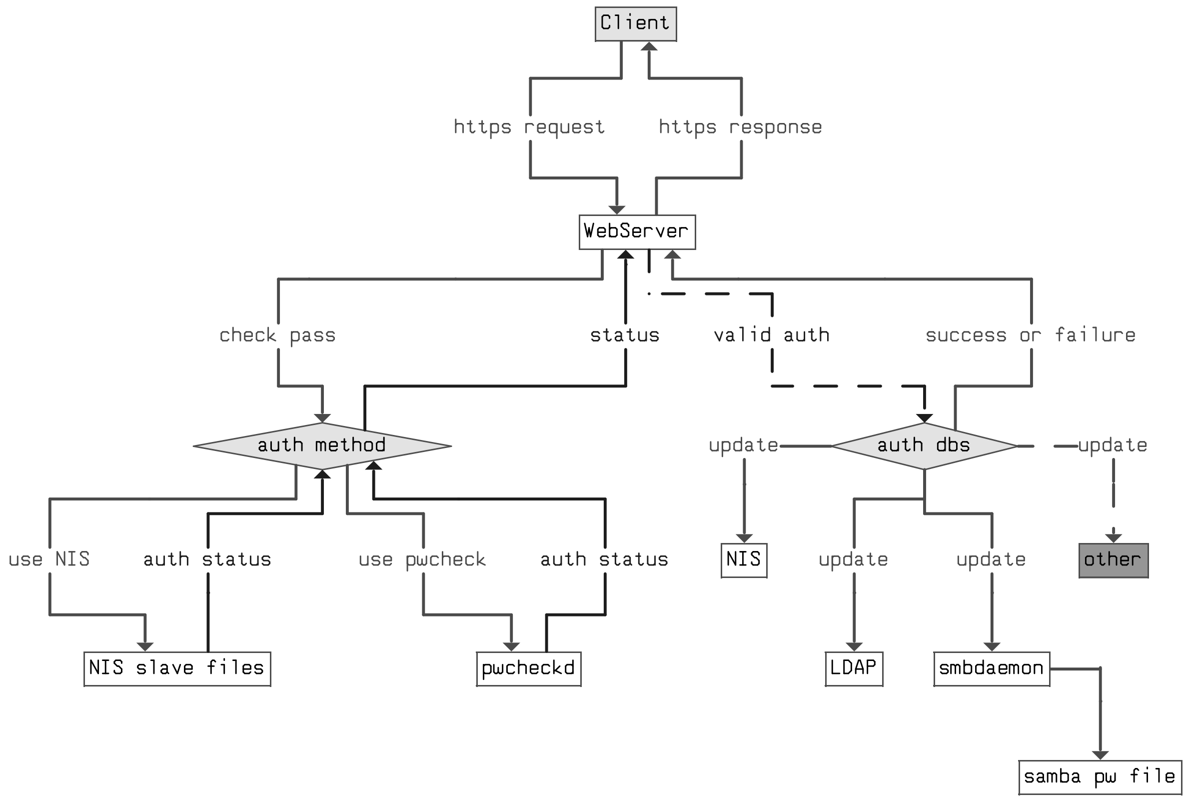

Figure 2: Password verification and update flowchart.

passwords do not match. Please retype them.' , or 'Passwords require at least six characters', or 'The new password you entered is too similar to your full name or username.' Upon successful completion, the program explicitly tells the user: 'Your password has been changed successfully and will be ready for you to use anywhere in engineering in the next 10 minutes.', giving them explicit expectations and scope.

It Should Build Upon Existing Distribution And Synchronization Mechanisms

One of the drawbacks – or features, depending upon your point of view – with some of the other systems is that they require wholesale replacement of existing mechanisms. This means modifying every machine, or replacing the entire account generation mechanism, or tailoring lots of configuration files. Sites starting with nothing, sites having special access requirements, or sites mired in complicated legacy scripts may have no qualms about replacing an existing system. We provide an incremental approach to sites that, like us, already have established account generation and maintenance mechanisms. It is not always possible to build incrementally, but in our case it was desirable.

It Should Be Easy To Extend as Other Distribution and Synchronization Mechanisms Become Available.

PHP, like Perl, is a "kitchen sink" language. It contains builtin interfaces to many databases and a

```
// Who do you want administrative mail (errors, etc) send to?
$Admin_Staff = "admin";

// General Vars - length of passwords
$MAXlength = 8;
$MINlength = 6;

// Error logging (daemon|error) (consult syslog.h)
$SYSLEVEL = 27;
// comment this out if you don't have NIS.. If you do this, you
// better have LDAP (or skills to modify passwd.php3)
// Note - this is for submitting the password change.
$have_NIS = 1;
$NIS_domain = "eng.auburn.edu";

// This is for checking the current password
// For machines with direct access to shadow password file,
// set $direct_access = 1; and set password file like the following
// line:
// $pw_dbm = "/var/yp/domainname/passwd.adjunct.byname";
$direct_access = 0;

// Where do you keep your user .forward files?
$forward_path = "/forwards";
// If you use the forward function, choose the location of your php_file
// helper auxiliary program
$php_forward_helper = "/etc/local/php_file";

// For getting page count database info. Basically, nobody really
// cares about this but us (Auburn). It's used in paper.php3.
$getpages = "/etc/local/getpages";

// Change these for LDAP
// ldap_uid should be an account with create and modify privileges on
// everything in your Samba and NIS ldap trees.

$have_ldap = 1;
$ldap_server = "ldap.your.domain";
$ldap_uid = "root";
$ldap_pw = "rootpw";
$ldap_smbdn = "ou=people,dc=eng,dc=auburn,dc=edu";      // Samba tree
$ldap_nisdn = "ou=people,dc=eng,dc=auburn,dc=edu";      // NIS/UNIX tree

// For socket based manual updates to an smbpasswd file, this is
// the host. Uncomment it here and change it as appropriate
// $SMBHOST = "smbhostname";
```

Figure 3: PHP configuration file.

rich set of string processing and system interface functions. It also has a well-documented and easy to use C API for extension. The Cracklib module is written as a loadable extension, as are methods for generating Samba hashes, and calling NIS for password update. TCP/IP sockets and cryptographic functions are also builtin providing yet another means for extension: client/server communication.

Architecture and Implementation

The code, where possible, was written in PHP for Apache [21]. A dynamically loadable PHP module was constructed as an enhancement. We also created a Perl helper daemon on the Samba PDC to accept network connections from the secure Web server and update the Samba password file.

Figure 2 shows a high-level flowchart of a password change transaction. The 'auth method' box encapsulates all of the functionality for checking to see that the username and old password are valid, as well as running the rule checks and Cracklib on the new password. Upon successful validation, the respective authentication databases are updated and status is returned to the user. The dashed "valid auth" line is conditional upon success of the authentication method. The dashed "update" line to the *other* box represents generic extensibility.

Configuration is currently managed through a PHP include file: *globals.php3*. Figure 3 shows a sample, commented *globals.php3*. This file lets you define and configure your authentication services so that when a user changes a password, the appropriate databases are updated.

The provided PHP loadable module is written in C. There are four main functions in this module. The *checkpass* function is used to verify the username and password as accepted from the user's browser. It calls *pwcheckd* on the server as detailed in the **Security** section. The *crack* function calls the Cracklib library with the user's desired new password. The *passwd* function calls the RPC *yppasswd* function to change the user's password in the NIS password map; the NIS master is set at compile time in the Makefile. Lastly, the *smbpasswd* function takes the user's plain text password, creates the NT hash pair, and returns the ASCII representation to PHP for transmission to the *smbdaemon* program. Transmission is accomplished with standard sockets.

Smbdaemon is a simple Perl program running on the Samba PDC machine. It is a 130 line Perl program at this writing. It listens on a well known port for updates, and then writes them to the private Samba *smbpasswd* file.

Security

There are five major areas where security needs to be addresses:

- On the Web server machine.
- Between the Web browser and the Web server.
- Between the Web server and the *pwcheckd* daemon (if used).
- Between the Web server and the authentication database (during updates).
- In ancillary programs.

On The Web Server Machine

Changing passwords has broad security implications. We strongly recommend that you dedicate a machine to these 'user services'. Do not install this software on your general Web server. Do not give users interactive accounts on the machine itself. Users given direct access to the machine may have access to files that they should not.

Browser to Server

As discussed in the goals section, the transactions between the client browser and the Web server must be encrypted using HTTPS, preferably using 128 bit clients. You should force your Web server to only accept high grade security connections.

If you decide to setup the Web server as a NIS client, you will authenticate users by directly comparing their encrypted passwords in the shadow password file with their passwords passed from the browser. In this case, you do not need to worry about passing the user's password over the network (because the files are local – except of course during periodic NIS synchronization updates). You do, however, need to be doubly certain that you do not give any users shell access to the machine. Additionally, you should setup NIS to use shadow passwords.

Pwcheckd

Pwcheckd has been designed to allow the Web server to verify the password of a user without passing the clear text or the encrypted version of the password over the network. The authentication transactions are summarized in the following list.

1. The Web server calls a function to verify the username and password which in turn calls *pwclient* in the loadable module.
2. *Pwclient* connects to *pwcheckd* on the designated port of a known server and asks for the UNIX password salt [22] for a given username.
3. *Pwcheckd* checks *tcp_wrappers* for valid client access.
4. *Pwcheckd* sends the salt back to *pwclient*.
5. *Pwclient* uses the salt and UNIX password algorithm to encrypt the user's supplied password.
6. *Pwclient* makes an MD5 hash of the user's encrypted password and current time and sends both hash and time to *pwcheckd*.
7. *Pwcheckd* receives the hash and time[1] from the user, and compares *pwclient*'s time with the

[1] You should use a time synchronization program like NTP [23] between the Web server and the machine running *pwcheckd* to keep the clocks synchronized.

server's time. If the time is not within plus or minus 30 seconds, the attempt is logged as a possible replay attack and the user is not authenticated.

8. *Pwcheckd* makes an MD5 hash of the user's actual password and time supplied by *pwclient* and compares with the hash from *pwclient*. Success or failure is returned to *pwclient*.

Server to Database Updates

Server to database update security is going to be dependent upon the authentication databases used. For LDAP, the current implementation uses the LDAP root password to open the database and update the encrypted portions of the user's LDAP entry. This password is currently stored in the *globals.php3* file. Since *globals.php3* contains only PHP variable assignments, as long as you do not have your Web server setup to allow people to fetch PHP3 files, it will be safe. To bring home an earlier point: do not give users shell access on this machine.

For NIS, PHP calls the RPC *yppasswd* function (in the PHP loadable module) which exposes the user's old password in clear text while the new password is transmitted in encrypted format. Because the update is fast, the risk is small and mitigated in other ways. (You do have your Web server and NIS server on the same secured, switched network, right?)

You may be wondering why the *pwcheckd* mechanism appears to be so much more stringent and secure than the database updates. The underlying authentication databases have a considerable influence on the update mechanisms available. The author hopes that these mechanisms can be improved in the future. For now, keep your Web server and your master databases on the same, secure network – preferably locked in a room or closet with no user accessible jacks or VLANs. Even if you do not follow this advice, the exposure of an encrypted password to prying eyes is typically[2] less of a concern than exposure of the plain-text original.

Ancillary Programs

For the contributed Samba password update program, the NT hashed password pair is sent via a socket to a server program running on the Samba PDC which replaces any existing Samba entry with the new one. This is a generic mechanism that could easily be extended or replaced. As configured, *smbdaemon* only accepts connections from the Web server. No plain-text password is transmitted, but the NT hashes are plain-text equivalent in that, if they are stolen, they will give access to the server as that user (a well-known Microsoft problem).

There is another ancillary program, *php_file*, not directly related to the password changing functionality. If you do not wish to use the mail forwarding functionality, remove *formail.php3* and the *php_file* program. Since Web servers are typically run either as the user nobody, or as a special account such as www or www-data, the helper program must be setuid to be able to edit the user's .forward file. While the *suexec* functionality of Apache would appear to be suited for this task, it has a number of shortcomings that make it less ideal.

- *Suexec* is not installed by default and requires special re-compilation of the Web server in many distributions. This, in turn, requires additional configuration, care and knowledge. Our set of tools is intended to be easier to run out of the box.
- The program to be executed by *suexec* must be resident in the WWW space and owned by the effective user doing the executing. Instituting this would require that all users have a copy of *php_file* in their Web directory space. This could be automated, of course, but would necessitate yet another process during account creation and deletion, as well as a small waste of space. This residency requirement could potentially involve thousands of new directories in the WWW space as well; since we recommend a dedicated Web server, a new directory would need to be created for every user and the *suexec*-able *php_file* must be copied into that directory.
- Altering the *suexec* code is potentially hazardous. There are many warnings about doing so. Though constructing new setuid code has its own perils, the author opted for the *perceived* simplicity of this approach. The alternative was altering a complex program covered with virtual no trespassing signs and barbed wire.

Consequently, *php_file* is setuid root. The username and password are given to *php_file* to verify. In its current implementation it lacks the authentication flexibility of the password changer. Instead, it uses the system to fetch the user's actual password given the supplied username. It then encrypts the supplied password and compares it with the actual password. If they do not match, the user is informed. If the actual password and supplied, encrypted password match, *php_file* becomes the user (setuid). During the process of writing a new *.forward* file or removing an old one, it makes sure to avoid symbolic link replacement attacks[3] by calling atomic functions. The user is expected to own the directory where the *.forward* file is located and the *.forward* file itself. If any of these conditions is not satisfied, the user is informed of an error, and a *syslog*(3) is sent about a possible attack.

Extensibility

By choosing PHP and its broad base of support databases and functions, much of the extensibility of

[2]See following paragraph.

[3]When a cracker exploits a race condition among system services in combination with symbolic links to cause unintended effects.

this project is builtin. Using network sockets for client-server interprocess communication is trivial. Accessing a NIS encrypted password database is as simple as granting permissions to the DBM file and opening it. Likewise, LDAP and encryption functions are present. Even so, we needed to construct a loadable module to provide a few supplementary functions.

Luckily, PHP makes it particularly easy to extend itself via dynamically loadable shared object libraries. The AUCOE supplied module contains some RPC clients, a Samba hash generator, and a wrapper for Cracklib.

The Perl Samba password changing daemon is provided as a generic model for quick and dirty extensibility. It receives connections from the network using a simple *socket*(2) *accept*(3) loop, verifies the connection is from the Web server, reads the record from the network connection, and writes it to the private Samba *smbpasswd* file. It also does some memory caching optimizations. (see the code).

Individuals wishing to add further functions in PHP should take a look at the *ldappw.php3* file and see how all of the LDAP functionality is encapsulated into a single object. Multiple LDAP server connections can be instantiated independently. This is the model that future extensions should use, and that the rest of the code will, eventually, be re-written to use.

Limitations and Future Directions

The code has not undergone significant outside testing. It should have an *AutoConf*(1) configuration for choosing options and a better installation procedure than copying files. More of the code should be converted to a class/object interface like the LDAP framework. The password verification mechanisms could be made more generic by allowing more choices like *PAM*(3) and *nsswitch.conf*(4).

We constructed a GUI front-end and middleware between the login and the various independent functions. This allows us to offer an easy way for users to login once at the beginning and automatically get access to the various functions (password changing, forwarding mail, access printer accounting information, generating one time passwords, etc.) However, it is currently not as comely as it might be and is fairly site specific.

We have also thought about integrating the rule-based configuration language of something like npasswd or passwd+, but given the extreme thoroughness of Cracklib, it may add marginal benefit.

Acknowledgments

This project could not have been completed without the help of various people and the support of the College of Engineering at Auburn University. Special thanks to Jerry Carter, for providing the Samba hash generation code and for some of the LDAP integration help, and to director Stephen Henderson who always supports us in our endeavors. Thanks to the PHP core and documentation team for putting together an outstanding programming environment, the Apache team for Web server integration, and the folks at Debian for making it easy to keep all of the packages and their dependencies up to date. Also thanks to my wife for understanding my work ethic and my two year old son who provides a wonderful source of stress relief (No, no.. **you** eat it..)

Availability

The code is currently available via anonymous ftp from ftp.eng.auburn.edu in pub/doug/AUCOEpw.tar.gz or available from the author's tools page at http://www.eng.auburn.edu/~doug/second.html

It is currently in production beta state – it works, but needs lots of configuration via globals.php3. The code is known to work on Solaris2.6 and above and on Linux, specifically Debian.

Author Information

Doug Hughes received a BE in Computer Engineering from Penn State University in 1991. His first exposure to UNIX was on a Harris HCX-7 system connected to the Internet, UUCP, and the BITNET.

After graduation he worked at GE Aerospace post-RCA merger, and through the Martin Marietta merger and various smaller buy-outs. He managed to escape in 1994 (just prior to the Lockheed merger). In the mean time he gathered experience in large scale software development, systems administration, network administration, and database administration.

He worked as the Senior Network Engineer for the College of Engineering at Auburn University from 1994 until 2000, when he accepted a position with Global Crossing. At the time of publication submission he was still working for Auburn. He can be contacted electronically at doug@eng.auburn.edu (which will probably remain active indefinitely).

References

[1] Microsoft Corporation, "Microsoft Networks SMB File Sharing Protocol (Document Version 6.0p)," Redmond, Washington, January 1, 1996.

[2] Allison, Jeremy, "The Samba File and Print Server," *;login:*, November 1997 NT Special: 12-18.

[3] Leighton, Luke Kenneth Casson, "Samba and Windows NT Security Interoperability," *Proceedings of the 3rd Large Installation Systems Administration of Windows NT Conference (LISA-NT)*, Seattle, WA, July 30 – August 2, 2000, Lake Forest, CA, USENIX, 2000.

[4] Hornsby, David, Columbia Appletalk Package. http://www.cs.mu.oz.au/appletalk/cap.html, University of Melbourne, Australia.

[5] Originally by Rasmus Lerdorf, *Portable Home Page*, http://www.php4.org/ , 1994.

[6] Muffet, Alec, *Cracklib: A ProActive Password Sanity Library*, http://www.users.dircon.co.uk/~crypto/ , 1997.

[7] Cottrell, Pete, "Password File Management at the University of Maryland," *Proceedings of the Large Installation Systems Administrators Conference, Philadelphia, PA, April 9-10, 1987*, Lake Forest, CA, USENIX, 1987. 32-33.

[8] Curry, David A., Samuel D. Kimery, Kent C. De La Croix, Jeffrey R. Schwab, "ACMAINT: An Account Creation and Maintenance System for Distributed UNIX Systems," *Proceedings of the 4th Systems Administration Conference (LISA '90)*, Colorado Springs, CO, October 18-19, 1990, Lake Forest, CA, USENIX, 1990, 1-10.

[9] Pato, Joseph N., Elizabeth Martin, Betsy Davis, "A User Account Registration System for a Large (Heterogeneous) UNIX Network," *Proceedings of the USENIX Conference*, Dallas, TX, Winter 1988, Lake Forest, CA, USENIX, 1988, 155-161.

[10] Hoover, Clyde, *npasswd*. Last updated July 13, 1999, http://www.utexas.edu/cc/unix/software/npasswd .

[11] Bishop, Matt, "Anatomy of a Proactive Password Changer," *Proceedings of the 2nd Usenix Security Symposium*, Baltimore MD, September 14-17, 1992, Lake Forest, CA, USENIX 1992, 171-184.

[12] *ANLPassword*, Source code, Last updated Feb 13, 1995, ftp://info.mcs.anl.gov/pub/systems/anlpasswd.tar.Z .

[13] Davis, Eric Allen, *Epasswd: Solving the Heterogeneous Password Program Problem*, http://www.nas.nasa.gov/Groups/Security/epasswd/ .

[14] Spencer, Henry, "Shuse At Two: Multi-Host Account Administration," *Proceedings of the 11th Systems Administration Conference*, (LISA '97), San Diego, CA, October 26-31, 1997, Lake Forest, CA, USENIX 1997, 65-69.

[15] Riddle, Paul, Paul Danckaert, Matt Metaferia, "AGUS: An Automatic Multi-Platform Account Generation System," *Proceedings of the 9th Systems Administration Conference (LISA '95)*, Monterey, CA, September 17-22, 1995, Lake Forest, CA, Usenix 1995, 171-180.

[16] Harris, J. Archer and Gregory Gingerich. "The design and implementation of a network account management system." *Proceedings of the 10th Systems Administration Conference (LISA '96)*, Chicago, IL, September 29 – October 4, 1996, Lake Forest, CA, USENIX, 1996. 181-189.

[17] Arnold, Bob, "Accountworks: Users Create Accounts on SQL, Notes, NT, and UNIX," *Proceedings of the 12th Systems Administration Conference (LISA '98)*, Boston, MA, December 6-11, 1998, Lake Forest, CA, USENIX, 1998, 49-61.

[18] Abbey, Jonathan, Michael Mulvaney, "Ganymede: An Extensible and Customizable Directory Management Framework," *Proceedings of the 12th Systems Administration Conference (LISA '98)*, Boston, MA, December 6-11, 1998, Lake Forest, CA, USENIX, 1998, 197-218.

[19] Mod_Perl, http://perl.apache.org/ .

[20] "Automated Password Generator (APG)," Federal Information Processing Standards Publication 181, October 5, 1993, National Institute of Standards, http://www.itl.nist.gov/fipspubs/fip181.htm .

[21] Apache Web Server. The Apache Software Foundation, http://www.apache.org/ .

[22] Spafford, Gene, Simson Garfinkel, *Practical UNIX & Internet Security, 2nd Edition*, Sebastopol, CA: O'Reilly, 1996.

[23] Mills, David L., "Network Time Protocol (Version 3) Specification, Implementation," *RFC 1305*, March 1992.