(/)

Everything you need for cybersecurity
BUILD A POWERFUL CYBERSECURITY PRACTICE
WATCH DEMO

(https://a.slashdotmedia.com/www/delivery/ck.php?oaparams=2__bannerid=30311__zoneid=27507__cb=f999cc7130ff07092b3c__oadest=https%3A%2F%2Finfo.connectwise.com%2FInd%2Fmsp%2Fbof%2Fsem%2Ffortify-demo%3Fmcid%3DREV1305%26utm_source%3Dsourceforge%26utm_medium%3Drs%26utm_campaign%3DREV1305%26utm_content%3Ddemo%26loc%3Dna)

Security (/tag/security)

Configuring One-Time Password Authentication with OTPW

by Todd A. Jacobs (/users/todd-jacobs) on March 27, 2013

Password authentication contains a lot of assumptions about security and trust. Encrypted SSH tunnels and public key verification are two common ways to ensure that your password is not compromised in transit. But, what if it's the computer you're currently typing on that can't be trusted?

This isn't just a tinfoil-hat scenario for paranoid penguinistas. There are many everyday situations and common locations where you probably should *not* use your system password, even over a secure tunnel. Examples include:

- A public computer in a hotel, library or Internet café.
- A coworker's virus-infested computer.
- A shared workstation while pair-programming.
- Any place someone could watch you type in your password.

What do all these examples have in common? Essentially, that you're trying to connect to a trusted destination from an untrusted source. This is a complete reversal of what most authentication systems were designed to address.

Take public key authentication. SSH public key authentication certainly bypasses the password prompt on the remote host, but it still requires you to trust the local machine with your private key password. In addition, once the key is decrypted with your password, the local system has full access to the sensitive key material inside.

Uh-oh—luckily, there's already a solution for this frequently overlooked problem: one-time passwords.

The combination of SSH and one-time passwords is powerful:

- The SSH protocol provides encryption of the login sequence across the network.
- A good SSH client allows you to inspect the remote host's public key fingerprint before entering your credentials. This prevents a rogue host from collecting your one-time passwords.
- The one-time password system ensures that a password can't be reused. So, even if the password is captured in transit, it's worthless to an attacker once you've logged in with it.

A number of one-time password solutions are available for UNIX-like systems. The two most well-known are S/KEY and OPIE (One-Time Passwords in Everything).

With the recent removal of OPIE from the Debian and Ubuntu repositories, the OTPW one-time password system created by Markus Kuhn provides a viable alternative. Although not a drop-in replacement for OPIE, OTPW offers comparable functionality while providing some interesting features not found in either S/KEY or OPIE.

OPIE Removal from Debian and Ubuntu Repositories

Debian began removing OPIE-related packages in early 2011, following some discussions about the security of the binaries, licensing issues and lack of upstream activity.

If you're interested in the details, the following Debian bug reports are relevant:

- http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=511582 (http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=511582)
- http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=622220 (http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=622220)
- http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=622221 (http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=622221)
- http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=622246 (http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=622246)

While the OPIE packages remain in the current Debian stable release at the time of this writing (code-named "Squeeze"), and some unofficial platform ports can be found in the debports repository, OPIE is not available in testing or unstable, and it appears unlikely to be included in the next stable release.

In particular, OTPW provides:

- Two-factor authentication, consisting of a "prefix password" and a set of autogenerated, disposable suffixes. Even if the list of suffixes falls into the wrong hands, brute force is necessary without the prefix password.
- Protection against certain race conditions and man-in-the-middle attacks through the use of password locks and triplet challenges.
- Shared-filesystem support. Because OTPW checks passwords against a list of hashed values stored in a user's home directory, one password list will work for all systems mounting the same $HOME directory.

Next, I cover installing and using OTPW, with a special focus on integration with OpenSSH.

Package Installation

To make use of OTPW, you need two binaries: otpw-bin and libpam-otpw. With Debian and Ubuntu, installation is as easy as:

```
sudo apt-get install otpw-bin libpam-otpw
```

If your distribution does not provide OTPW, you can download the source directly from the author's home page. The source tarball does not use GNU autoconf, so you will need to compile and install the binaries manually in accordance with the author's instructions.

Configure PAM

The next step in preparing the system for OTPW is configuration of libpam-otpw. A full treatment of PAM is outside the scope of this article, but I cover the most common use cases here.

Changing your PAM configuration can lock you out of your workstation or server, so it's a good idea to keep your existing terminal open until you're sure that things are working correctly. If you have console access, keep a bootable distribution or rescue disk handy. See the Testing One-Time Password Authentication with SSH sidebar for more information about testing PAM over SSH.

Testing One-Time Password Authentication with SSH

If you are configuring a remote system for OTPW, you should test your PAM stack without closing your current SSH connection. Remember, if you make a mistake with your PAM configuration, you may be unable to authenticate—even with console access—so keep a bootable distribution, such as Knoppix, SystemRescueCD or Finnix handy just in case. Meanwhile, existing logins remain unaffected because they already are authenticated.

In order to test the PAM stack properly, you can't re-use your existing SSH connection. Most recent distributions support SSH multiplexing and persistent connections out of the box, so explicitly disable these options for testing.

In addition, SSH prefers public key authentication by default. So, in order to test OTPW authentication, public key authentication needs to be temporarily disabled too.

The following invocation enables accurate testing of the SSH PAM stack, without making any system changes:

```
read -p 'Hostname: ' REMOTE_HOST &&
SSH_AGENT_PID= SSH_AUTH_SOCK= \
  ssh \
  -o PreferredAuthentications=keyboard-interactive \
  -o ControlPersist=no \
  -o ControlPath=none \
  "$REMOTE_HOST"
```

Once you have confidence that OTPW is working correctly, you also should verify that your other authentication mechanisms (namely SSH public keys and normal system passwords) continue to work as expected.

The easiest way to enable OTPW is to put it immediately above `pam_unix` in your common-auth configuration file:

```
# /etc/pam.d/common-auth
auth       sufficient pam_otpw.so
session    optional   pam_otpw.so

auth       sufficient pam_unix.so nullok_secure
auth       required   pam_deny.so
```

The order of the PAM libraries is very important. When placing OTPW first, users with an ~/.otpw file are prompted for a one-time password first, allowing fallback to standard system passwords if the OTPW login fails. Users without a ~/.otpw file simply will see the standard password prompt.

If you prefer to reverse the order, prompting for a system password before falling back to one-time passwords, just ensure that `pam_deny` comes last:

```
# /etc/pam.d/common-auth
auth       sufficient pam_unix.so nullok_secure

auth       sufficient pam_otpw.so
session    optional   pam_otpw.so

auth       required   pam_deny.so
```

If you're tempted to remove standard system passwords altogether, especially from console logins, please don't. On some systems, most notably Ubuntu systems with ecryptfs-encrypted home directories, recovering from OTPW mishaps is extremely difficult without standard system passwords.

Modifying common-auth is usually the right thing to do on a headless server or console-only system. However, workstations or servers that provide the X Window System present special problems for one-time password systems.

Some tools or applications won't work properly with OTPW because they can't display the challenge to the user. The typical symptom is usually a password dialog that never completes or seems to ignore user input. In times past, gksu and GNOME Display Manager (GDM) had this issue with OPIE. In such cases, the solution is to move OTPW out of common-auth and include it only in specific services.

For example, you can add OTPW authentication to SSH connections while using just the standard password prompt for console or GUI logins. You can do this in three easy steps:

1. Delete any lines from common-auth that reference pam_otpw.so:

```
# /etc/pam.d/common-auth on Debian Squeeze
auth       sufficient pam_unix.so nullok_secure
auth       required   pam_deny.so
```

2. Create a new OTPW include file for PAM:

```
# /etc/pam.d/otpw
auth           sufficient      pam_otpw.so
session        optional        pam_otpw.so
```

3. Include OTPW immediately *before* common-auth in /etc/pam.d/sshd:

```
# Other stuff ...

# Enable OTPW authentication.
@include otpw

# Standard Un*x authentication.
@include common-auth

# More stuff ...
```

SSH Configuration

In addition to configuring the PAM libraries, OTPW needs the following three settings in the SSH dæmon's configuration file:

```
# /etc/ssh/sshd_config
UsePrivilegeSeparation yes
UsePAM yes
ChallengeResponseAuthentication yes
```

These are usually there, but possibly commented out or set to "no", so modify them accordingly. Next, reload the SSH dæmon after modifying its configuration file:

```
# Generic Linux
sudo /etc/init.d/ssh reload

# Debian 6.0.4+
sudo service ssh reload

# Ubuntu 11.04+
sudo reload ssh
```

Generating OTPW Passwords

Once the OTPW PAM module has been configured properly, only users with an ~/.otpw file will be challenged with a one-time password dialog during login. This file contains some metadata about its contents, as well as a list of one-way hashes that will match only a valid response to a challenge.

To create this file, or to re-populate it with new passwords, use the otpw-gen utility. By default, it will create 280 password suffixes, formatted to fit on a single side of US letter-sized (8.5" x 11") paper. Because only the one-way hashes are stored in ~/.otpw, not the passwords themselves, you must capture or print the standard output of this command when the passwords are generated. You will not be able to retrieve the password list after the fact; you'll need to generate new passwords instead.

Here is what it looks like when you run the command for the first time, piping the output to your default printer:

```
$ otpw-gen | lpr
Generating random seed ...

If your paper password list is stolen, the thief
should not gain access to your account with this
information alone. Therefore, you need to memorize
and enter below a prefix password. You will have to
enter that each time directly before entering the
one-time password (on the same line).

When you log in, a 3-digit password number will be
displayed. It identifies the one-time password on
your list that you have to append to the prefix
password. If another login to your account is in
progress at the same time, several password numbers
may be shown and all corresponding passwords have to
be appended after the prefix password. Best generate
a new password list when you have used up half of
the old one.

Enter new prefix password:
Reenter prefix password:

Creating '~/.otpw'.
Generating new one-time passwords ...
```

When generating a new password list, the prompts that appear on standard error are slightly different:

```
Overwrite existing password list '~/.otpw' (Y/n)?

Enter new prefix password:
Reenter prefix password:

Creating '~/.otpw'.
Generating new one-time passwords ...
```

The first prompt ensures that you don't accidentally over-write your existing password list; the second prompt asks you for a new password. There's nothing stopping you from reusing the same prefix password on each invocation—the random seed makes duplicate hashes unlikely—but best practice is to use a new prefix each time you regenerate the password list.

If you want to generate a password list on a remote host but print to a local printer, you can do this over your SSH connection as long as you trust your localhost:

```
read -p 'Hostname: ' &
```

Note the use of `stty` to ensure that your prefix password isn't echoed to the screen. As long as your prefix password remains secure, you are no worse off using an untrusted printer than you are if your password list falls into the wrong hands. This is often a valuable security trade-off for frequent travelers.

Finally, to disable OTPW challenges for a given user, just delete the .otpw file in that user's home directory.

Using OTPW to Log In

Once you have your password list in hand, you're ready to use one-time password authentication for your SSH connection. Assuming that you don't have any identities loaded into your SSH agent, your dialog should look similar to this:

```
$ ssh localhost
Password 015:
```

The prompt with the digits is the OTPW challenge. To respond, find the matching challenge ID on the password sheet you printed earlier. Next, enter your prefix password followed by the string that follows the challenge ID.

Using "foo" as a prefix password, the following suffix list was generated. Your list and suffixes will be different, even if you use the same prefix password.

```
OTPW list generated 2012-05-06 13:40 on localhost

000 SWvv JGk5 004 =qfF q2Mv 008 sb5P h94r 012 o5aH +/GD 016 8eLV VxuA
001 xPZR :ceV 005 B=bq =mHN 009 WBSR smty 013 QMZ% +bm8 017 vjFL K4VU
002 Sj%n 9xD3 006 RrNx sJXC 010 Xr6J F+Wv 014 j=LO CMmx 018 Km8c 8Q3K
003 s7g8 NE%v 007 sd=E MTqW 011 fNKT vo84 015 fWI% MB9e 019 z8ui %eQ3

            !!! REMEMBER: Enter the PREFIX PASSWORD first !!!
```

To respond to this challenge successfully, type:

```
foo fWI% MB9e
```

at the prompt. The spaces are optional; OTPW ignores them if present.

If you answered the challenge correctly, login will proceed. Otherwise, you will be prompted with the standard system login. At this point, you can enter your standard system password, or press Return to give OTPW another try. After the system-defined number of password attempts (usually three), the login will fail and return you to the command prompt:

```
$ ssh localhost
Password 013:
Password:
Password 013:
Password:
Password 013:
Password:
Permission denied (publickey,password,keyboard-interactive).
```

To prevent simultaneous logins, or when SSH is interrupted during OTPW authentication, OTPW may lock a password. When a password is locked, your next login attempt will present a triplet challenge that requires one prefix and three suffixes to respond:

```
$ ssh localhost
Password 004/011/005:
```

Given the same password list as before, enter your triplet response as a single line, with or without spaces. The following shows how the response is composed (note that the first line below is just an informational aid; you would type only the second line below, without the pipe characters):

```
prefix | suffix 004 | suffix 011 | suffix 005
foo    | =qfF q2Mv  | fNKT vo84  | B=bq =mHN
```

Once you have successfully responded to a triplet challenge, login will proceed and the ~/.otpw.lock symlink should be removed, and your next challenge will again be a single challenge ID number.

In some cases, the lock is not removed properly. If you continue to be prompted for a triplet, you can remove the lockfile manually:

```
rm ~/.otpw.lock
```

Users with encrypted home directories that are not already mounted before login will need to take a few additional steps. See the OTPW and Encrypted Home Directories sidebar for an example.

OTPW and Encrypted Home Directories

The ecryptfs filesystem presents special problems for SSH and OTPW. By default, distributions like Ubuntu unwrap the special passphrase required to mount an encrypted home directory with the user's system password.

This is handled by the pam_ecryptfs.so module, which is included through /etc/pam.d/common-auth and others. If you authenticate using anything other than your system password, the module prompts you for a system login password in order to mount the encrypted home directory.

In practice, this means that your system password is exposed on untrusted terminals when mounting your remote home directory. This is obviously not ideal.

The best way to avoid this is to leave a console session running at all times. For example, log in at the console using your system password, and then lock the screen. As long as your console session remains active, your home directory remains mounted. As a result, you can use OTPW authentication without further changes to the system, and you won't reveal your system password during login or mounting.

However, if you still want to be able to use OTPW for SSH logins when a console session isn't running—and understand the security implications of doing so—here's how it's done.

First, you need to create a wrapper script for calling otpw-gen:

```
#!/bin/bash
set -e
otpw-gen "$@"
mv ~/.otpw /usr/local/lib/otpw/$LOGNAME/
ln -s /usr/local/lib/otpw/$LOGNAME/.otpw ~/
```

The wrapper should be placed in your path and made executable.

Next, place otpw4ecryptfs.sh (listed below) in ~/bin or /usr/local/sbin:

```
#!/bin/bash

# Purpose:
#     Enable OTPW for all users on systems with
#     ecryptfs-mounted home directories.

set -e

# Expose the underlying directories that may be
# hidden by ecryptfs.
sudo mkdir -p /mnt/real_home
sudo mount -o bind /home /mnt/real_home

# Collect all non-system users.
users=$(
    awk -F: '$1 != "nobody" \
             && $3 >= 1000  \
             && $3 < 65534  \
             {print $1}' /etc/passwd
)

# Enable OTPW for each non-system user.
for user in $users; do
    sudo mkdir -p /usr/local/lib/otpw/$user
    sudo touch /usr/local/lib/otpw/$user/.otpw
    sudo chown -R $user: /usr/local/lib/otpw/$user
    sudo chmod 755 /mnt/real_home/$user
    ln -sf /usr/local/lib/otpw/$user/.otpw \
           /mnt/real_home/$user/
    ln -sf /usr/local/lib/otpw/$user/.otpw \
           /home/$user/
done < /etc/passwd

sudo umount /mnt/real_home
```

When you run the script, it creates OTPW files that are readable by pam_otpw.so even when the user's home directory is unmounted.

Please note that this script gives read and execute permissions to all users' home directories so that pam_otpw.so can read the OTPW password files. This is not inherently a risk, but users who rely on more restrictive directory permissions may want to tighten up the permissions of files and folders in their home directories immediately afterward.

Finally, all users should run `otpw-gen-wrapper.sh` to populate and maintain their OTPW password list. Always use the wrapper instead of calling otpw-gen directly, or password generation will break the symlinks required for proper operation.

Check for Remaining Passwords

If your password list is exhausted, you will no longer be able to use OTPW to log in until a new list is generated. Likewise, if your password list doesn't contain at least three unused responses, you will not be able to use OTPW to log in when ~/.otpw.lock exists, because there are not enough challenge IDs to issue a triplet.

In addition, some of the security of OTPW comes from the randomness of the remaining challenges. The use of triplets in particular can exhaust your unused passwords rapidly, so it's a good idea to regenerate the password list whenever you fall below a minimum amount.

The OTPW author recommends regenerating the password list when less than half the original passwords remain unused, but doesn't define a minimum bound for number of passwords required for adequate randomness of challenges. A small number of unused passwords makes you more vulnerable to brute-force attacks, since there are fewer challenges to present.

The pam_otpw.so PAM module is supposed to inform the user when unused passwords fall below half of those generated. However, the PAM session functionality doesn't seem to work on Debian or Ubuntu. In addition, even if it worked, the module doesn't establish a floor to ensure sufficient randomness of challenges.

The otwp-stats.sh script shown in Listing 1 provides this missing functionality. It also allows you to define a sensible minimum for unused passwords by adjusting the MIN_PASSWORDS variable at the top of the script.

Listing 1. otwp-stats.sh

```
#!/bin/bash

# 30 unused passwords seems like a reasonable, if
# arbitrary, floor to ensure randomness and a small
# cushion against triplet exhaustion. Feel free to
# adjust this number to suit your needs.
MIN_PASSWORDS=30
OTPW_LIST="$HOME/.otpw"

# Stop processing if OTPW isn't set up for this
# user.
[ -f "$OTPW_LIST" ] || exit

# The top two lines of an OTPW file are meta-data.
TOTAL_PASSWORDS=$((`wc -l < "$OTPW_LIST"` - 2))
# Lines with dashes represent used passwords.
USED_PASSWORDS=$(egrep '^-' "$OTPW_LIST" | wc -l)
# The number of passwords remaining is a calculated
# value.
PASSWORDS_LEFT=$((TOTAL_PASSWORDS - USED_PASSWORDS))

cat << EOF
OTPW Password Statistics
------------------------
    Passwords used: ${USED_PASSWORDS:=0}
    Passwords left: $PASSWORDS_LEFT

EOF

if [ $PASSWORDS_LEFT -le $((TOTAL_PASSWORDS / 2)) ]
then
    echo "It's time to generate new OTPW passwords."
elif [ $PASSWORDS_LEFT -le $MIN_PASSWORDS ]; then
    echo "Remaining passwords at critical levels."
    echo "It's time to generate new OTPW passwords."
fi
```

Add otwp-stats.sh to your ~/.profile (or other shell startup script) to provide feedback at login:

```
# Only run script when logging in via SSH.
[ -n "$SSH_CONNECTION" ] && ~/bin/otpw-stats.sh
```

Conclusion

OTPW provides a one-time password implementation that compares favorably against OPIE and S/KEY. It is easy to integrate with SSH on most Linux systems, and remains possible to use on Ubuntu systems with encrypted home directories.

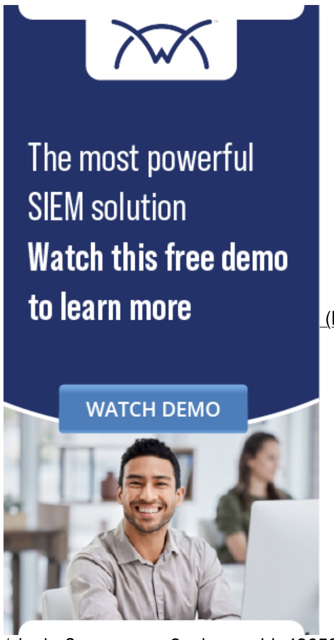
Resources

OTPW Source: http://www.cl.cam.ac.uk/~mgk25/otpw.html (http://www.cl.cam.ac.uk/~mgk25/otpw.html)

Password image (http://www.shutterstock.com/pic.mhtml?id=105375764) via Shutterstock.com

Todd A. Jacobs is the CEO of Flow Capital Group, which acquires and manages companies specializing in IT automation, DevOps & agile transformations, security and compliance, fractional CIO/CTO services, and board advisory services for cyber risk and other hot technology issues. Todd waited his whole life for Matt Smith's character on *Dr. Who* to make bow ties cool again. He lives near Baltimore, MD with his wife and son, to whom he hopes to pass on his love of Linux and technology—but perhaps not his fashion sense.

No comments yet. Be the first! (https://www.linuxjournal.com/content/configuring-one-time-password-authentication-otpw#disqus_thread)



(https://a.slashdotmedia.com/www/delivery/ck.php?oaparams=2__bannerid=43653__zoneid=40961__cb=e9dcb25ca416054028fe__oadest=https%3A%2F%2Finfo.connectwise.com%2F3p%2Feco%2Fsec%2Fgen%2Fperch-demo%3Fmcid%3DREV2105%26utm_source%3Dsourceforge%26utm_medium%3Drs%26utm_campaign%3DREV2105%26utm_content%3Ddemo%26loc%3Dna)

Recent Articles

(/content/simplifying-docker-installation-linux)

Simplifying Docker Installation on Linux (/content/simplifying-docker-installation-linux)

George Whittaker (/users/george-whittaker)

(/content/data-safety-and-efficiency-rdiff-backup)

Data Safety and Efficiency with Rdiff-backup (/content/data-safety-and-efficiency-rdiff-backup)

Patrik Dufresne (/users/patrik-dufresne)

(/content/mastering-user-and-permission-management-fortifying-your-linux-bastion)

Mastering User and Permission Management: Fortifying Your Linux Bastion (/content/mastering-user-and-permission-management-fortifying-your-linux-bastion)

George Whittaker (/users/george-whittaker)

(/content/best-tools-create-bootable-usb-iso-linux)

The 6 Best Tools to Create a Bootable USB From an ISO in Linux (/content/best-tools-create-bootable-usb-iso-linux)

George Whittaker (/users/george-whittaker)

(/content/core-knowledge-modern-linux-kernel-developer-should-have)

Core Knowledge That Modern Linux Kernel Developer Should Have (/content/core-knowledge-modern-linux-kernel-developer-should-have)

Roman Storozhenko (/users/roman-storozhenko)

(/content/linux-networking-simplified-guide-ip-addresses-and-routing)

Linux Networking: A Simplified Guide to IP Addresses and Routing (/content/linux-networking-simplified-guide-ip-addresses-and-routing)

George Whittaker (/users/george-whittaker)

Connect With Us (https://youtube.com/linuxjournalonline) (https://www.facebook.com/linuxjournal/) (https://twitter.com/linuxjournal)

Linux Journal, representing 25+ years of publication, is the original magazine of the global Open Source community.
© 2023 Slashdot Media, LLC. All rights reserved.

PRIVACY POLICY (https://slashdotmedia.com/privacy-statement/) | TERMS OF SERVICE (https://slashdotmedia.com/terms-of-use/) | ADVERTISE (/sponsors) | OPT OUT (http://slashdotmedia.com/opt-out-choices)

MASTHEAD (/CONTENT/MASTHEAD)
RSS FEEDS (/RSS_FEEDS)
AUTHORS (/AUTHOR)
ABOUT US (/ABOUTUS)

CONTACT US (/FORM/CONTACT)