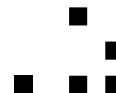


## 目次

1	Randori: fend off multiple attackers	2
1.1	Fully based on PAM ( <i>Pwn All Malware</i> ) . . . . .	2
1.2	PAM module . . . . .	2
1.3	OpenSSH . . . . .	4
1.4	Randori for Apache . . . . .	5



## Randori: fend off multiple attackers

Fully based on PAM (Pwn All Malware)

Randori (乱取り) is a form of practice in which a designated aikidoka defends against multiple attackers in quick succession. [<https://en.wikipedia.org/wiki/Randori>]

Basically it is my <http://github.com/avuko/aiki> PoC on steroids.

Shoutout to **0xBF** (ONSec-Lab) for giving us [https://github.com/ONsec-Lab/scripts/tree/master/pam\\_steal](https://github.com/ONsec-Lab/scripts/tree/master/pam_steal). All of the below is based on that simple, great idea.

Also thanks to @micheloosterhof for being approachable when I had questions and comments about cowrie (<https://github.com/micheloosterhof/cowrie>).

### PAM module

This PAM module will log to `/var/log/randori.log` all services, remote hosts, usernames and passwords (make sure `/var/log/randori.log` is read/writable).

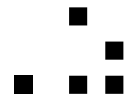
Satisfy prerequisites:

```
sudo apt-get install build-essential dpatch fakeroot\
devscripts equivs lintian quilt libpam0g-dev
```

`pam_randori.c`

```
/*
 * pam_randori - get remote service/clientip/username/password from
 * brute-force attacks
 *
 * Usage: add "auth required pam_randori.so"
 * into /etc/pam.d/common-auth
 * just above "auth requisite pam_deny.so"
 *
 * Reload services using PAM to start getting output.
 * Perhaps needless to add, but you might want to
 * only log in with keys :)
 */

#include <stdio.h>
#include <string.h>
#include <time.h>
#include <security/pam_modules.h>
#define LOGFILE "/var/log/randori.log"
```



```
PAM_EXTERN int pam_sm_authenticate(pam_handle_t * pamh, int flags
                                   ,int argc, const char **argv)
{
    int retval;

    const void *servicename;
    const char *username;
    const void *password;
    const void *rhostname;
    FILE *log;

    time_t now;
    struct tm ts;
    char timestamp[80];

    time(&now);
    // Format time, "yyyy-mm-ddThh:mm:ss+zzzz"
    ts = *localtime(&now);
    strftime(timestamp, sizeof(timestamp), "%Y-%m-%dT%H:%M:%S%z", &ts);

    /* get the name of the calling PAM_SERVICE. */
    retval=pam_get_item(pamh, PAM_SERVICE, &servicename);

    /* get the RHOST ip address. */
    retval=pam_get_item(pamh, PAM_RHOST, &rhostname);

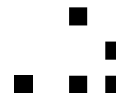
    retval = pam_get_user(pamh, &username, NULL);

    retval = pam_get_item(pamh, PAM_AUTHTOK, &password);

    /* As opposed to the original pam_steal, I DO care about
     * non-existing user passwords.
     * Perhaps we should drop attempts without a password later
     */
    //if (password != NULL) {
    log = fopen (LOGFILE, "a");
    fprintf(log, "%s\u2002%s\u2002%s\u2002%s\u2002%s\u2002\n", (char *) timestamp,
              (char *) servicename, (char *) rhostname,
              (char *) username, (char *) password);
    fclose( log);

    return PAM_IGNORE;

    //}
```



```
}

PAM_EXTERN int pam_sm_setcred(pam_handle_t *pamh, int flags,
                              int argc, const char **argv)
{
    return PAM_IGNORE;
}
```

Run `make.sh`

```
#!/bin/sh

set -e

rm -f pam_randori.so
gcc -g -O2 -MT pam_randori_la-pam_randori.lo -MD -MP -MF\
pam_randori_la-pam_randori.Tpo -c pam_randori.c -fPIC -DPIC -o\
pam_randori_la-pam_randori.o
gcc -shared pam_randori_la-pam_randori.o -lpam_misc -lpam -Wl,\
-soname -Wl,pam_randori.so -o pam_randori.so
rm -f pam_randori_la-pam_randori.Tpo pam_randori_la-pam_randori.o

cp pam_randori.so /lib/x86_64-linux-gnu/security/
```

Add `pam_randori.so` to `/etc/pam.d/common-auth`.

TIL: Never try to test things like this manually when it is 2 AM. Unless you want a `deny == success` pam configuration (oops).

Anyway, the below will not log valid credentials, but will log any other non-valid attempts. Also, I wrote `testlogins.sh` to verify this, because...

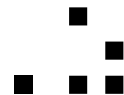
Your `/etc/pam.d/common-auth` should now look something like this:

```
auth    [success=2 default=ignore] pam_unix.so nullok_secure
auth    required                  pam_randori.so
auth    requisite                 pam_deny.so
auth    required                  pam_permit.so
```

## OpenSSH

You need to build OpenSSH from source. Yes, this is necessary. OpenSSH, instead of keeping the original password, throws out a (rather haphazardly chosen?) string when incorrect credentials are entered:

```
grep -n INCORRECT auth-pam.c
822:    /* t char junk[] = "\b\n\r\177INCORRECT"; */
```



In order not to mess with the original code too much (and because I'm already way out of my comfort zone writing/editing C), I made a simple change only:

```
/* XXX avuko: 2017-19-06T17:00:00 Tweak to return the password entered
   for a non existing account */
/* t char junk[] = "\b\n\r\177INCORRECT"; */
/* const char junk[] = "ABCDEFGHJKLMNOPQRS"; */
char *ret = NULL;
size_t i, l = wire_password != NULL ? strlen(wire_password) : 0;

if (l >= INT_MAX)
    fatal("%s: password length too long: %zu", __func__, l);

ret = malloc(l + 1);
for (i = 0; i < l; i++)
    ret[i] = wire_password[i];
    /* ret[i] = junk[i % (sizeof(junk) - 1)]; */
ret[i] = '\0';
// ret = strcat(wire_password, '\0');
return ret;
```

Yes, that is very, very likely a fully unnecessary loop. But then there is **CVE-2016-6210-2**, so lets leave well enough alone.

Randori for Apache

get apache2 and the apache pam module

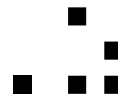
```
apt install apache2
sudo apt-get install libapache2-mod-authnz-external pwauth
sudo apt-get install libapache2-mod-authz-unixgroup
sudo a2enmod authnz_external authz_unixgroup
```

Edit `/etc/apache2/mods-enabled/authnz_pam.conf`

```
<Location />
  AuthType Basic
  AuthName "admin"
  AuthBasicProvider PAM
  AuthPAMService apache
  Require pam-account apache
</Location>
```

Edit/create `/etc/pam.d/apache`

```
@include common-auth
```



And that was all there was to it. Next, we need a lot of finishing touches with **<VirtualHost>** magic or setting up multiple different **pam** modules for **Location**.