Assignment 0x01

Frank Kaiser – 1742945, Jan Martin – 1796943

November 19, 2017

Contents

1 Task 1: Account Recovery

1.1 Most important recommendations for the account recovery process

General

- Don't store passwords in plain text, but checksums.
- Balance ease of use with challenge.
- Regularly revalidate information like current email address, phone numbers etc.

Email Authentication

- Focus on recovering the account - not all features
- Don't send their old password, but a new, randomly generated password. Or a token.
- Notify real account holder over all channels available.

Knowledge-Based

- Don't allow easily guessed information.
- Plan for cultural differences.
- Changes should require at least as much authority as recovery.

Social

- Don't spam or otherwise alienate the users social contacts required for information.

Multi-Channel Authentication

- Is only useful if independent, so sending a SMS to a phone that is used to recover the account is not helpful.

1.2 Example: dotasource.de

As test object I used the web site `www.dotasource.de`.

This is what the Lost Password page looks like:

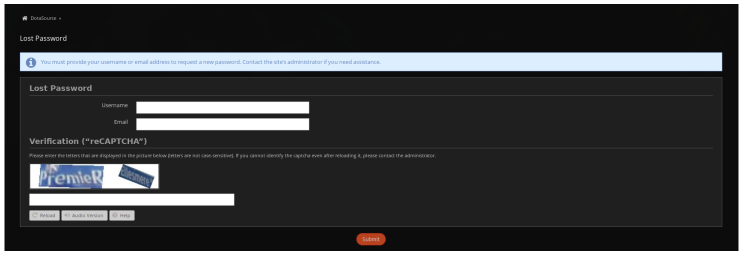


Figure 1: Lost Password

When entered a non-existent username or E-mail address the page responds that the user does not exist or no account with that e-mail is registered.

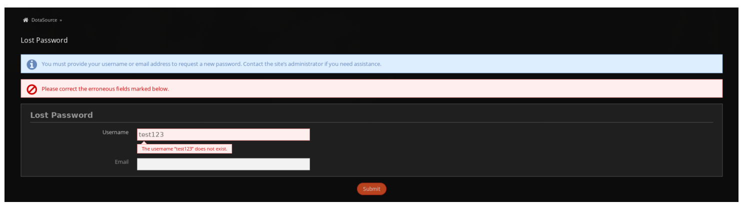


Figure 2: User does not exist

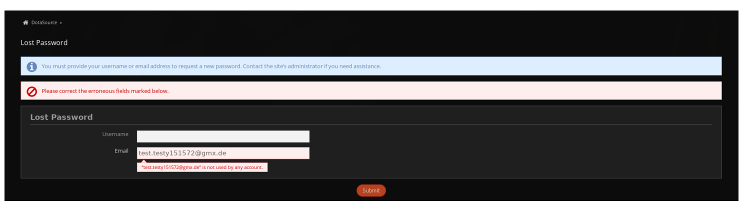


Figure 3: E-mail not used by an account

When entering correct information an email is sent to the account's address, containing a link for recovery.

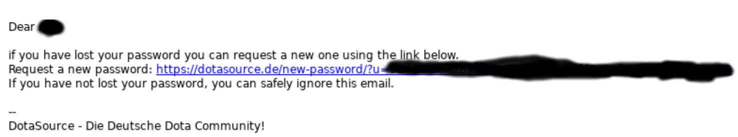


Figure 4: Reset Link

When clicked on the link another email with a new, random generated password in plain text is sent. Sessions that are still logged in don't lock you out.

But for a new login the password is in immediate effect. To change the password again, you have to login using the given password and visit the account management site.

What is definitely missing is that it is never asked if the given email address is still valid. Also it should be changed that the system responds that the user/email does not exist. Therefore a message like "If the user exists, an email was sent to his address" would be more suitable. Though given this is a public forum, it still would be possible to create an account and look at all registered user names.

2 Task 2: Public-Key Authentication in SSH

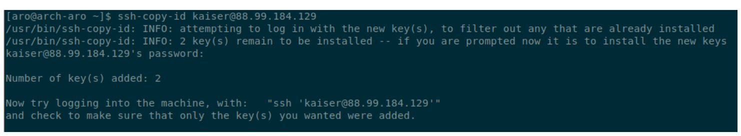
2.1 GNU/Linux

To generate the ssh-key pair I used the command `ssh-keygen` which generates by default a 2048 bit long rsa key.

```
[aro@arch-aro ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/aro/.ssh/id_rsa): /home/aro/.ssh/id_rsa_psi
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/aro/.ssh/id_rsa_psi.
Your public key has been saved in /home/aro/.ssh/id_rsa_psi.pub.
The key fingerprint is:
SHA256:E/pB4YMB3MhgfOa0LrcTliMzLw23/T/lMFPCtS9sgDo aro@arch-aro
The key's randomart image is:
+---[RSA 2048]----+
| .o+.+. .        |
| .. B .+ .  .    |
|   = .. =o . .   |
|    o  o.o+ o    |
|   . ...S  = .   |
|  * O E. o+ = .  |
|   % * ..  B .   |
|  . * .    . .   |
|   . . .....     |
+----[SHA256]-----+
```

Figure 5: ssh-keygen

To copy the key on the server `ssh-copy-id user@88.99.184.129` was used. Since I already had a public key for another server and the exercise was to create one, both keys got uploaded to the server as seen in the picture 6.



```
[aro@arch-aro ~]$ ssh-copy-id kaiser@88.99.184.129
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 2 key(s) remain to be installed -- if you are prompted now it is to install the new keys
kaiser@88.99.184.129's password:

Number of key(s) added: 2

Now try logging into the machine, with:   "ssh 'kaiser@88.99.184.129'"
and check to make sure that only the key(s) you wanted were added.
```

Figure 6: save public key on server

The following log-in worked without the user password for the server. Only the optional password for the private ssh key was required. The keys on the server are stored in `~/.ssh/authorized_keys`. See figure 7

```
[aro@arch-aro ~]$ ssh kaiser@88.99.184.129
Linux psi-introsp 4.9.0-4-amd64 #1 SMP Debian 4.9.51-1 (2017-09-28) x86_64

Last login: Fri Nov 17 20:32:43 2017 from 188.194.245.11
kaiser@psi-introsp:~$ cd .ssh/
kaiser@psi-introsp:~/.ssh$ ls
authorized_keys  known_hosts
kaiser@psi-introsp:~/.ssh$ cat authorized_keys
```

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQCoUxV87YG14gRvtXeFI5YmBgILxzXVY5BRJ46fvmPGrKeNsr1a0Z5MAk5VQJuSmAXTe0jIImyBy0iVOT
OtWkrsPerw56t+MKKURMgErNI0G+g1d8f0CWtW4say5GUDX/DEe6d22/TBtD4Log4TmlJ8T+qOvmElvBfxuTqmvKv5Culzl/Lhr2UF1MOc1utq0sINvNTH
e+77QoFhn0CSaye4+BCGUWTf59waFqBs5i8O6wsgRe5lkMsrO51msGusfZ4Mlf6EaSoz/W7Ss6F2rtkDh7OJYI1yh6b+a0uMVrtoDnexDq+tPvlEDTpkAa
+BQ5TtECtHKvbuUnZEqsGlxLlHN+3TLTMvvgWNQ8PaSk5RVze9HjMVF3eVyp8V2lYNqD/Tv73kUSZvZc327TFN56lkC69JsvHFYkTckTZmc9l+SLN/9uaW
UDFN5xLqWTata29GQVkKAkZdXOWmkUBSqDM/lH8NorWyW7u3L4XcNzUS7FhcwktfJlCtLI4mCwmN+Xr/8OPffDyZYpn/wnu0XdRrogU5og6y1fJ37bxnXB
6sJ4ZiUUIDF25MpE1Y8Ppplcur7cSQOdcXgWfYxwVKBwo4deGEeu64dTxoazpc9WWg1a/sINa6bW9MHj0dPj7b9s45VEGhlB5Eo+W09NMFXAlEb661aOMT
3bYjCyI+VuDpeQ== frank-phillip.kessler@stud.uni-bamberg.de
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQD9znThVNgDjKOf4EcHJcRooxiGha2DvdJjSpFZPWBZuJNz7ae7mKeGmTzGyae0l5jlyhp9ka03tjriP2
Niw3PILVkKXO9Bd+K+G4M2pXuayBzgdqfgkjidM1K9WQLB4t1PZuS0iMWTJHgmW8Y19XzqbHzkcmtzXzMYO52IDBD/+ELcKHL4tyZy58sAJicSq3X0rMHL
93N4tFx87s7tq/9H/kHo7yqbL7099Rz3icqJzlICuOFsOJ5pyIozTtV+iukzTHuWIZ/gT0PB/emtkiZDXUjxCKbhQcpnd9/IWSuPmw4+TQbJgsLDnk3420
srYmC3CRKcJD7QGOKZ+tuXYBB9 aro@arch-aro

```
kaiser@psi-introsp:~/.ssh$
```

Figure 7: location of keys on server

The ssh-keys on your local machine are by default stored in `~/.ssh/id_rsa.pub` for the public part and the private one in `~/.ssh/id_rsa`. (Provided you did create a rsa key) See figure 8.

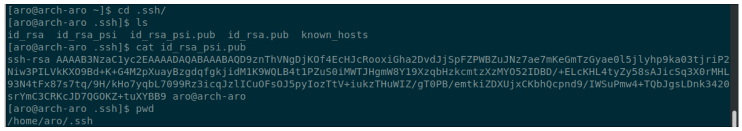


Figure 8: location of keys on local machine

2.2 Windows

On a windows machine the same tools from openSSH were not available. That's why the procedure was a little bit different. Here PuTTY was used.

To generate the key the tool `putty-gen` was used. Then we logged into the server via ssh and created the file `authorized_keys` in `~/.ssh/` and pasted the key into it using nano. Lastly, we checked that only we have write access to the file by `ls -l`

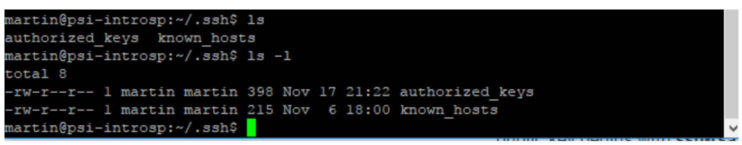


Figure 9: permission check

In figure 10 you can see the successful login by using the ssh-key pair.

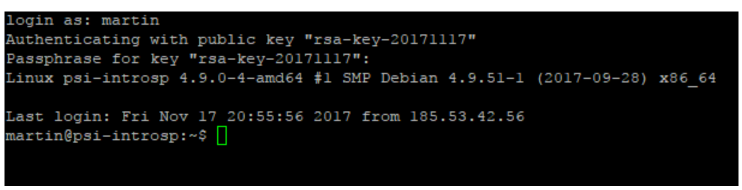


Figure 10: windows ssh-key login

3 Task 3: Buffers in C: The Vigenre Cipher

The source code for the exercise can be found in the following file: Vigenere Cipher

To compile: gcc -Wall vigenere_cipher.c -o "output-name" Optional flag: -DDEBUG

Below is the source code readable:

```c
#include <ctype.h>
#include <stdio.h>
#include <string.h>

int getRotation(char c) {
  /*According to ascii 'A' transfers to 65
  and Z to 90. By subtracting 65 of the char we get
  the rotation. */
  return c - 65;
}

int main(int argc, char *argv[]) {
  char key[256];
  char word[256];

  printf("Type in the key\n");
  fgets(key, sizeof(key), stdin);

  printf("What do you want to encrypt?\n");
  fgets(word, sizeof(word), stdin);

  // Number of char that were not uppercase
  int cntSkipped = 0;
  // length of the key - 1 to know when to start from 0
  int keyLength = strlen(key) - 1; // remove \n
  int keyPosition = 0;             // index of the key word

  for (int i = 0; i < strlen(word); i++) {
              /* Set keyPosition to 0 when end of key is reached
              i is substracted by the number of skipped chars
              so the % operator works as intended */
    if ((i > 0) & ((i - cntSkipped) % keyLength == 0)) {
      keyPosition = 0;
    }
    // ignore lowercases and whitespaces
    if (!isupper(word[i])) {
      cntSkipped++;
      continue;
    }
```

```
    int rotation = getRotation(key[keyPosition]);
    keyPosition++;

#ifdef DEBUG
    printf("%i ", rotation);
#endif

    word[i] = ((word[i] - 'A' + rotation) % 26) + 'A';
  }
  printf("%s", word);

  return 0;
}
```

4 Task 4: Memory Management on the Stack

To get the Success! message you have to do a buffer overflow by giving more than 16 characters as input string. After that you can put input chars on the stack. You can set two variables at once, because memory was allocated for variables 'myvalue' and 'mystring'. The stack is first in, last out, i.e. the first element put into it, will be the last one read. Because of this you have to put in the chars in reverse order. The resulting command looks like this:

```
echo -e "0000000000000000\xef\xbe\xad\xde" | ./memory
```

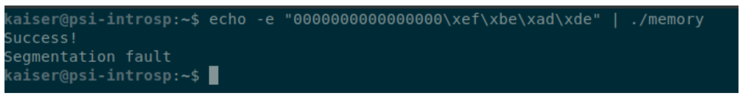


Figure 11: Successful input