CS342 Computer Security — **Handout # 8**
Prof. Lyn Turbak — **February 11, 2016**
Wellesley College — ***Revised February 20, 2016***

Password Cracking Exercises

These are notes from the *John the Ripper* password cracking exercise we did on our Ubuntu VMs in lecture on Thu. Feb. 11, 2016. The exercises described here have been modified from those done in lecture to clarify aspects of the password cracking process.

Introduction

In the `wendy` account of your Ubuntu VM, the *John the Ripper* password cracking software has been installed in the directory `~wendy/john-1.7.9-jumbo-7`. It has two subdirectories:

1. The subdirectory `~wendy/john-1.7.9-jumbo-7/run` contains executables, password lists, and configuration files needed to run the cracker.

2. The subdirectory `~wendy/john-1.7.9-jumbo-7/doc` contains detailed documentation on installing and running the cracker, the various cracking mode, understanding the transformation rules,

The following exercises will assume that you're logged in as `wendy` and connected to `~wendy/john-1.7.9-jumbo-7/run`:

```
wendy@cs342-ubuntu-1:~$ cd ~/john-1.7.9-jumbo-7/run
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$
```

Your Ubuntu VM is preconfigured with three user accounts that have the following passwords:

account name	**password**
`wendy`	`Tr0ub4dor&3`
`guest`	`guest`
`gdome`	`IAmDome`

You might also have an account with your own username. You will be changing the passwords of the `guest` and `gdome` accounts several times in these exercises. **I recommend you do not change the passwords of `wendy` or your personal account, since losing access to these by forgetting a password can be catastrophic!**

By default, the password cracker runs in three consecutive stages:

1. In **single crack mode**, it will try an extensive set of transformation rules on the account names and "real" names of each user to generate candidate passwords.

2. In **wordlist mode**, it will try a much smaller set of transformation rules on each word in the wordlist (in this case, `password.lst`) to generate candidate passwords.

3. In **incremental mode**, it will will use a " brute force" algorithm to generate all strings up to a certain length (by default this is 8) to generate candidate passwords.

We will explore these modes individually before we try them together.

Exercise 1: Password Cracking in Wordlist Mode

Step 1: Configure `john.conf`

We will first experiment with wordlist mode. The transformation rules for wordlist mode are specified in the `run` directory in in `john.conf`, which in our original Ubuntu VMs is a symbolic link to one of two several possible files:

```
 wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ ls -al john.conf*
lrwxrwxrwx 1 wendy wendy    14 Feb 19 13:40 john.conf ->
   john.conf.single-rules-in-wordlist
-rw-rw-r-- 1 wendy wendy 41327 Feb 16 12:18 john.conf.orig
-rw-rw-r-- 1 wendy wendy 46296 Feb 10 01:15 john.conf.single-rules-in-wordlist
```

If `john.conf` is already a link to `john.conf.orig`, you're ready for this exercise. But if not, you'll need to change it as follows:

```
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ rm john.conf
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ ln -s john.conf.orig john.conf
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ ls -al john.conf*
lrwxrwxrwx 1 wendy wendy    14 Feb 19 16:32 john.conf -> john.conf.orig
-rw-rw-r-- 1 wendy wendy 41327 Feb 16 12:18 john.conf.orig
-rw-rw-r-- 1 wendy wendy 46296 Feb 10 01:15 john.conf.single-rules-in-wordlist
```

Step 2: Save `john.pot` and `john.rec`

John stores cracked passwords in the file `john.pot` and the state of the current cracking process in `john.rec`. If you've already done some password cracking, you don't want the state in these files to influence the results of the experiments in this handout, but you don't want to lose this state either. If you have a `john.pot` and/or `john.rec` file, rename them as shown below, and later restore them (as described in the last section of this handout). If you don't have a `john.pot` or `john.rec` file, go to the next step.

```
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ ls -al john.pot
-rw------- 1 wendy wendy 436 Feb 18 07:30 john.pot
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ mv john.pot john.pot.saved
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ ls -al john.pot.*
-rw------- 1 wendy wendy 436 Feb 18 07:30 john.pot.saved
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ ls -al john.rec
-rw------- 1 wendy wendy 84 Feb 18 19:14 john.rec
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ mv john.rec john.rec.saved
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ ls -al john.rec*
-rw------- 1 wendy wendy 84 Feb 18 19:14 john.rec.saved
```

Step 3: Review the word lists

There are several wordlists for password cracking in the `run` directory:

- `tiny.lst` is a very small list of 14 words used for exercises like this one. It contains the following words. Like all the wordlists, it contains one word per line.

  ```
  password
  dog
  123
  abc123
  computer
  ```

```
zephyr
8675301
scoobydoo
cat in the hat
Chris
Elizabeth
HARLEY
Tr0ub4dor\&3
"#a@B.c?!"
```

- `password.lst` is a list of about 3500 common passwords.
- `big.lst` is a list with over 5 million entries.

Step 4: Change the passwords of guest and gdome

To begin this exercise, use `sudo passwd` to change the password of `guest` and `gdome` to be very simple transformations of a word in `tiny.list` (e.g., capitalizing the first character or all characters, reversal, adding a 1 a the end of the word). Do not change `wendy`'s password, which is already one of the entries in `tiny.list`.

For concreteness, suppose we change `guest`'s password to `retupmoc` and `gdome`'s password to `Zephyring`.

Step 5: Create the unshadowed passwd file

To run the password cracker, it's necessary to create a single file that has all the relevant information of both `/etc/passwed` and `/etc/shadow`. This is accomplished with the `unshadow` command:

```
sudo ./unshadow /etc/passwd /etc/shadow > unshadowed1
```

> If you get an AVX error when executing the `unshadow` command, then, as `wendy`, open a terminal window and execute the following two commands:
>
> ```
> cd ~/john-1.7.9-jumbo-7/src
> sudo make clean linux-x86-any
> ```
>
> When this completes, you should be able to execute the `unshadow` command without an error.

Although it's not necessary to examine the `unshadowed1` file, below are some relevant lines. Note that each account line now includes a salted and hashed password.

```
root:$6$6BdRbSEX$V6ip72A9tR5vhrKwyAU/8GQvwtU0/gEpt78diQLM9HGKJIrE35Z8fers9.D99DVb
    3E0xGZGH2KQTa2Mv/ViNA/:0:0:root:/root:/bin/bash
wendy:$6$FoFIUVCi$/FUkEldzfdJXMefmv/s76m4wRpeZPnHjsVdJ9p0.QgKWuZmVcjt5J53lZ8Sifj9
    Q3Pm6n6ukR9p8A143mnE2Q0:1000:1000:Wendy,,,:/home/wendy:/bin/bash
gdome:$6$qLRfg4g2$41mUIpCI.UmSDf5Gyr3Hu8yAgvNj5iUZ1aDfHwgpspjWFzC59Wc4xwvwmpBVXEm
    4JIq3whqI0ddpiJXI056dB0:1001:1002:Georgia Dome,,,:/home/gdome:/bin/bash
guest:$6$uIJ28SVB$LY.0b2IbYZgbhrskD1hXqglt6.a6piUWFS1bTPN.XKnT6SJg/B8fux091K61Sf/
    07oJqDkUFMfLv80l0smRrN/:1100:1100:guest:/home/guest:/bin/sh
```

Step 6: Run the Password Cracker in Wordlist Mode

Now we run the password cracker in wordlist mode using `tiny.lst` as our wordlist on the unshadowed file:

```
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ ./john --wordlist=tiny.lst --rules
    unshadowed1
Warning: detected hash type "sha512crypt", but the string is also recognized as
    "crypt"
Use the "--format=crypt" option to force loading these as that type instead
Loaded 4 password hashes with 4 different salts (sha512crypt [32/32])
Tr0ub4dor&3      (wendy)
retupmoc         (guest)

guesses: 2  time: 0:00:00:02 78.34% (ETA: Fri Feb 19 19:29:06 2016)  c/s: 248
    trying: 3harley

Zephyring        (gdome)
guesses: 3  time: 0:00:00:03 DONE (Fri Feb 19 19:29:07 2016)  c/s: 245  trying:
    Harleying
Use the "--show" option to display all of the cracked passwords reliably
```

As the cracker finds a password, it prints it out. It finds `wendy`'s and `guest`'s passwords almost immediately, and only takes a short while to find `gdome`'s. The lines that begin with `guesses` were printed when I hit the space bar. They show the current candidate password being tried, as well as how many cracks per second (c/s) the cracker is trying.

In this case it found 3 of 4 passwords (the 4th is for user `root`, which it did not find).

Exercise 2: Understanding Wordlist Transformation Rules

How did the password cracker transform the words in `tiny.list`? It used a set of *Wordlist mode rules* that are specified in `john.conf.orig` in a section labeled `[List.Rules:Wordlist]` (figure 1).

Each of the rules is specified by a line written in an obscure language for describing transformations on strings. The full details of this language are described in the `doc/RULES` file. It is not necessary to understand all the details, but here are a few examples to give you an understanding of what rules look like:

- The rule `:` means to make no changes to the word
- The rule `-c >3 !?X l Q` means

  `-c`: if the hash being used is case-sensitive (and the one Ubuntu uses is)

  `>3`: and the word is has more than 3 characters

  `!?X`: and the word contains only alphanumeric characters

  `l`: then lowercase the word

  `Q`: but only add it to the resulting word list if if isn't already there.

- The rule `-c <* >2 !?A c $1` means

  `-c`: if the hash being used is case-sensitive

  `(<*`: and the word is smaller than the maximum system-specified length

  `>2`: and the word has more than 2 characters

```
# Wordlist mode rules
[List.Rules:Wordlist]
# Try words as they are
:
# Lowercase every pure alphanumeric word
-c >3 !?X l Q
# Capitalize every pure alphanumeric word
-c (?a >2 !?X c Q
# Lowercase and pluralize pure alphabetic words
<* >2 !?A l p
# Lowercase pure alphabetic words and append ’1’
<* >2 !?A l $1
# Capitalize pure alphabetic words and append ’1’
-c <* >2 !?A c $1
# Duplicate reasonably short pure alphabetic words (fred -> fredfred)
<7 >1 !?A l d
# Lowercase and reverse pure alphabetic words
>3 !?A l M r Q
# Prefix pure alphabetic words with ’1’
>2 !?A l ^1
# Uppercase pure alphanumeric words
-c >2 !?X u Q M c Q u
# Lowercase pure alphabetic words and append a digit or simple punctuation
<* >2 !?A l $[2!37954860.?]
# Words containing punctuation, which is then squeezed out, lowercase
/?p @?p >3 l
 Words with vowels removed, lowercase
/?v @?v >3 l
# Words containing whitespace, which is then squeezed out, lowercase
/?w @?w >3 l
# Capitalize and duplicate short pure alphabetic words (fred -> FredFred)
-c <7 >1 !?A c d
# Capitalize and reverse pure alphabetic words (fred -> derF)
-c <+ >2 !?A c r
# Reverse and capitalize pure alphabetic words (fred -> Derf)
-c >2 !?A l M r Q c
# Lowercase and reflect pure alphabetic words (fred -> fredderf)
<7 >1 !?A l d M ’l f Q
# Uppercase the last letter of pure alphabetic words (fred -> freD)
-c <+ >2 !?A l M r Q c r
# Prefix pure alphabetic words with ’2’ or ’4’
>2 !?A l ^[24]
# Capitalize pure alphabetic words and append a digit or simple punctuation
-c <* >2 !?A c $[2!3957468.?0]
# Prefix pure alphabetic words with digits
>2 !?A l ^[379568]
# Capitalize and pluralize pure alphabetic words of reasonable length
-c <* >2 !?A c p
# Lowercase/capitalize pure alphabetic words of reasonable length and convert:
# crack -> cracked, crack -> cracking
-[:c] <* >2 !?A \p1[lc] M [PI] Q
# Try the second half of split passwords
-s x**
-s-c x** M l Q
```

Figure 1: The default Wordlist mode rules in `john.conf`.

  `!?A`: and the word contains only alphabetic characters

  `c`: then capitalize the word

  `$1`: and append to the end of it the digit `1`.

- The rule `-c <* >2 !?A c $[2!3957468.?0]` is a shorthand for twelve separate rules that result from choosing one of the twelve characters delimited by the square brackets. For example:

  ```
  -c <* >2 !?A c $2
  -c <* >2 !?A c $!
  ...
  -c <* >2 !?A c $?
  -c <* >2 !?A c $0
  ```

Thankfully, each of the wordlist rules in figure 1 is preceded by a comment summarizing what it does. But not all rules (especially the ones for single crack mode, see below) are so well-commented, so it's nice to have another way to understand what the rules are doing.

Fortunately, we can gain insight into this process by executing the following command, which runs all the transformation rules on the words in `tiny.lst` and writes all the transformed words to `tiny-transformed.txt`.

```
./john --wordlist=tiny.lst --rules --stdout > tiny-transformed.txt
```

The resulting file has 416 words. You should create and study this file on your Ubunutu VM, and match up the output to the sequence of rules in figure 1. For example, the first 14 words in `tiny-transformed.txt` are the result of the do-nothing rule `:`, so these are just the 14 words unchanged from `tiny.lst`. The next three words

```
chris
elizabeth
harley
```

result from the lowercasing rule, which only adds results to the wordlist if they're not already there. The next seven words

```
Password
Dog
Abc123
Computer
Zephyr
Scoobydoo
Harley
```

result from capitalizing alphanumeric words that begin with a letter. Note that because `cat in the hat` includes spaces, it is not alphanumeric, and so is not transformed. Also note that "capitalizing" `HARLEY` yields `Harley`.

You should examine the rest of `tiny-transformed.txt` to see how it matches up with the rules in figure 1.

Keep in mind that the rules in figure 1 are **all** the rules that are applied to words in the word list. There aren't many of them! So there are **lots** of transformations on wordlist words that will never be attempted by the password cracker with this default rules, such as reversing a word and adding a digit, or using *leetspeak* transformations like `3` for `e`, `4` for `a`, etc. Of course, nothing prevents a determine hacker from extending the default wordlist rules with many more.

Exercise 3: Password Cracking in Single Crack Mode

Now we're ready to try password cracking in single-crack mode, which uses transformations on combinations of the account name and other user information from the /etc/passwd file.

Step 1: Add two users to /etc/passwd

To illustrate the complex behavior of single crack mode, we'll add the following two lines to /etc/passwd:

```
cello:x:1200:1200:Alex Smith:/home/cello:/bin/bash
violin:x:1201:1201:Mary Ruth Jones:/home/violin:/bin/bash
```

Use sudo emacs -nw to create an Emacs editor with root privileges for making these changes to /etc/passwd. This editor window will be within the terminal and will not have GUI features. If you prefer a GUI version of Emacs, execute the following two lines instead:

```
sudo su -
emacs &
```

Step 2: Change passwords

Now let's use sudo passwd to change the passwords of users guest, gdome, cello, and violin to be related to their account name or real name. Here are some ideas, but you should experiment with others:

guest password: tseug

gdome password: !domegdome!

cello password: acello97

violin password: jonesmary2019

Step 3: Make a new unshadowed file

We'll need to make a new unshadowed password file for running the password cracker:

```
sudo ./unshadow /etc/passwd /etc/shadow > unshadowed2
```

Step 4: Remove john.pot

We don't need the john.pot file that recorded the passwords from our previous experiment, so delete it from the run directory:

```
rm john.pot
```

Step 5: Run the Password Cracker in Single Crack Mode

Now we run the password cracker in single crack mode on unshadowed2.

```
./john --single unshadowed2
```

After executing the command, keep hitting the space bar to see candidates that the cracker is trying, For example, figure 2 shows some candidates from a sample run that is able to crack all four passwords from above.

```
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ ./john --single unshadowed2
Warning: detected hash type "sha512crypt", but the string is also recognized as "crypt"
Use the "--format=crypt" option to force loading these as that type instead
Loaded 6 password hashes with 6 different salts (sha512crypt [32/32])
guesses: 0  time: 0:00:00:00 0.73% (ETA: Sat Feb 20 12:46:18 2016)  c/s: 254  trying: Wwend
guesses: 0  time: 0:00:00:01 1.20% (ETA: Sat Feb 20 12:47:41 2016)  c/s: 252  trying: asmit
tseug            (guest)
guesses: 1  time: 0:00:00:02 1.66% (ETA: Sat Feb 20 12:48:18 2016)  c/s: 255  trying: JonesruthJonesruth
guesses: 1  time: 0:00:00:03 2.31% (ETA: Sat Feb 20 12:48:27 2016)  c/s: 259  trying: YRAMSENOJ
guesses: 1  time: 0:00:00:04 2.86% (ETA: Sat Feb 20 12:48:37 2016)  c/s: 258  trying: elloA
guesses: 1  time: 0:00:00:06 5.45% (ETA: Sat Feb 20 12:48:08 2016)  c/s: 261  trying: iolinjones
guesses: 1  time: 0:00:00:07 6.56% (ETA: Sat Feb 20 12:48:04 2016)  c/s: 262  trying: gdomegeorgia2
guesses: 1  time: 0:00:00:08 7.02% (ETA: Sat Feb 20 12:48:11 2016)  c/s: 263  trying: jmary7
guesses: 1  time: 0:00:00:10 7.85% (ETA: Sat Feb 20 12:48:25 2016)  c/s: 264  trying: smithalexd
guesses: 1  time: 0:00:00:15 9.98% (ETA: Sat Feb 20 12:48:48 2016)  c/s: 266  trying: jviolin#
guesses: 1  time: 0:00:00:18 11.18% (ETA: Sat Feb 20 12:48:59 2016)  c/s: 267  trying: celloalex;
guesses: 1  time: 0:00:00:24 13.67% (ETA: Sat Feb 20 12:49:13 2016)  c/s: 268  trying: Celloa
guesses: 1  time: 0:00:00:25 14.04% (ETA: Sat Feb 20 12:49:16 2016)  c/s: 267  trying: Gdomedomee
guesses: 1  time: 0:00:00:27 14.97% (ETA: Sat Feb 20 12:49:18 2016)  c/s: 268  trying: Jruthn
guesses: 1  time: 0:00:00:28 15.34% (ETA: Sat Feb 20 12:49:20 2016)  c/s: 268  trying: Jonest
guesses: 1  time: 0:00:00:31 16.63% (ETA: Sat Feb 20 12:49:24 2016)  c/s: 268  trying: Violin)
guesses: 1  time: 0:00:00:36 18.66% (ETA: Sat Feb 20 12:49:30 2016)  c/s: 268  trying: Jonesmary}
guesses: 1  time: 0:00:00:38 19.77% (ETA: Sat Feb 20 12:49:30 2016)  c/s: 268  trying: emodg7
guesses: 1  time: 0:00:00:40 20.60% (ETA: Sat Feb 20 12:49:32 2016)  c/s: 268  trying: CSMITH6
guesses: 1  time: 0:00:00:42 21.44% (ETA: Sat Feb 20 12:49:33 2016)  c/s: 268  trying: Georgiadome!!
guesses: 1  time: 0:00:00:45 23.56% (ETA: Sat Feb 20 12:49:29 2016)  c/s: 268  trying: 4l3xc3llo
guesses: 1  time: 0:00:00:47 28.55% (ETA: Sat Feb 20 12:49:02 2016)  c/s: 268  trying: asmithcello
guesses: 1  time: 0:00:00:53 31.05% (ETA: Sat Feb 20 12:49:08 2016)  c/s: 268  trying: 4cello
guesses: 1  time: 0:00:00:54 31.51% (ETA: Sat Feb 20 12:49:09 2016)  c/s: 268  trying: 9vjones
guesses: 1  time: 0:00:00:58 33.27% (ETA: Sat Feb 20 12:49:12 2016)  c/s: 269  trying: Qjonesruth
guesses: 1  time: 0:00:01:00 34.10% (ETA: Sat Feb 20 12:49:13 2016)  c/s: 269  trying: Zsmithalex
guesses: 1  time: 0:00:01:02 34.84% (ETA: Sat Feb 20 12:49:15 2016)  c/s: 269  trying: IgGeorgia
guesses: 1  time: 0:00:01:04 35.67% (ETA: Sat Feb 20 12:49:17 2016)  c/s: 269  trying: RRuth
guesses: 1  time: 0:00:01:06 36.41% (ETA: Sat Feb 20 12:49:19 2016)  c/s: 269  trying: YSAlex
guesses: 1  time: 0:00:01:09 37.61% (ETA: Sat Feb 20 12:49:21 2016)  c/s: 269  trying: .georgiagdome
guesses: 1  time: 0:00:01:12 38.90% (ETA: Sat Feb 20 12:49:23 2016)  c/s: 269  trying: ‘alexsmith
guesses: 1  time: 0:00:01:13 39.37% (ETA: Sat Feb 20 12:49:23 2016)  c/s: 269  trying: themary
guesses: 1  time: 0:00:01:15 40.66% (ETA: Sat Feb 20 12:49:22 2016)  c/s: 269  trying: drSmith
guesses: 1  time: 0:00:01:17 41.95% (ETA: Sat Feb 20 12:49:21 2016)  c/s: 269  trying: 4jruths
guesses: 1  time: 0:00:01:20 43.25% (ETA: Sat Feb 20 12:49:22 2016)  c/s: 269  trying: C_alex
guesses: 1  time: 0:00:01:23 44.82% (ETA: Sat Feb 20 12:49:23 2016)  c/s: 269  trying: onesmaryj
guesses: 1  time: 0:00:01:26 46.95% (ETA: Sat Feb 20 12:49:21 2016)  c/s: 269  trying: JonesviolIn
guesses: 1  time: 0:00:01:29 49.26% (ETA: Sat Feb 20 12:49:18 2016)  c/s: 269  trying: Wendywndy
guesses: 1  time: 0:00:01:32 52.49% (ETA: Sat Feb 20 12:49:13 2016)  c/s: 269  trying: a.lexsmith
guesses: 1  time: 0:00:01:35 53.78% (ETA: Sat Feb 20 12:49:14 2016)  c/s: 269  trying: jones90
acello97         (cello)
guesses: 2  time: 0:00:01:38 55.26% (ETA: Sat Feb 20 12:49:15 2016)  c/s: 269  trying: jonesmary05
guesses: 2  time: 0:00:01:41 56.93% (ETA: Sat Feb 20 12:49:15 2016)  c/s: 269  trying: Violinmary94
guesses: 2  time: 0:00:01:44 58.50% (ETA: Sat Feb 20 12:49:15 2016)  c/s: 269  trying: dgdome71
guesses: 2  time: 0:00:01:46 59.70% (ETA: Sat Feb 20 12:49:15 2016)  c/s: 269  trying: ruthviolin84
guesses: 2  time: 0:00:01:49 61.36% (ETA: Sat Feb 20 12:49:15 2016)  c/s: 269  trying: Jviolin72
guesses: 2  time: 0:00:01:55 64.69% (ETA: Sat Feb 20 12:49:15 2016)  c/s: 269  trying: jmaryH
guesses: 2  time: 0:00:01:59 66.91% (ETA: Sat Feb 20 12:49:15 2016)  c/s: 269  trying: ViolinjonesG
!domegdome!      (gdome)
guesses: 3  time: 0:00:02:05 70.51% (ETA: Sat Feb 20 12:49:15 2016)  c/s: 269  trying: %wendywendy%
guesses: 3  time: 0:00:02:08 72.82% (ETA: Sat Feb 20 12:49:13 2016)  c/s: 269  trying: ruthjones69
guesses: 3  time: 0:00:02:11 75.23% (ETA: Sat Feb 20 12:49:12 2016)  c/s: 269  trying: ruth55
guesses: 3  time: 0:00:02:14 77.54% (ETA: Sat Feb 20 12:49:10 2016)  c/s: 269  trying: Jones40
guesses: 3  time: 0:00:02:16 79.11% (ETA: Sat Feb 20 12:49:09 2016)  c/s: 269  trying: Ruthjones57
guesses: 3  time: 0:00:02:19 81.42% (ETA: Sat Feb 20 12:49:08 2016)  c/s: 269  trying: Wendywendy777
guesses: 3  time: 0:00:02:22 83.82% (ETA: Sat Feb 20 12:49:07 2016)  c/s: 269  trying: maryjones00000
guesses: 3  time: 0:00:02:25 86.13% (ETA: Sat Feb 20 12:49:06 2016)  c/s: 269  trying: maryjones333333
guesses: 3  time: 0:00:02:27 88.81% (ETA: Sat Feb 20 12:49:03 2016)  c/s: 269  trying: ruthmary1970
guesses: 3  time: 0:00:02:30 91.12% (ETA: Sat Feb 20 12:49:02 2016)  c/s: 269  trying: wendy1993
jonesmary2019    (violin)
guesses: 4  time: 0:00:02:33 94.26% (ETA: Sat Feb 20 12:49:00 2016)  c/s: 269  trying: wendywendy1961
guesses: 4  time: 0:00:02:34 DONE (Sat Feb 20 12:48:52 2016)  c/s: 269  trying: root1900
```

Figure 2: Transcript of a single crack mode run that found four passwords. Lines beginning `guesses:` were printed out each time the space bar was pressed. Blank lines have been edited out.

Exercise 4: Understanding Single Crack Transformation Rules

We'd like to know exactly the sequence of words that are tried in single crack mode. Sadly, the `--stdout` option does not work in `--single` mode, so *John* does not provide a way to see these like it does with the transformation of words in a word list.

However, based on candidates seen in transcripts like figure 2, we can reverse engineer much of what single crack mode does. It appears that this mode does the following for each user:

- splits the user information on spaces and lowercases the results. E.g. `Alex Smith` becomes the two words `alex` and `smith`, and `Mary Ruth Jones` becomes the three words `mary`, `ruth` and `jones`.

- adds to the single mode wordlist the following:

  - *base words* consisting of the account name and each of the lowercased split words from the user info. For account `cello`:

    ```
    cello
    alex
    smith
    ```

  - all concatenations of the first letter of one base word and a whole other based word. In this case:

    ```
    calex
    csmith
    acello
    asmith
    scello
    salex
    ```

  - all concatenations of distinct pairs of base words. In this case:

    ```
    celloalex
    cellosmith
    alexcello
    alexsmith
    smithcello
    smithalex
    ```

The above process is repeated for each user, and all the words are collected into a single word list for single crack mode. Then that word list is processed by the single crack mode transformation rules in the part of `john.conf` that begins with `[List.Rules:Single]`. This section has way more rules than the rule section for regular wordlists. Unfortunately, they are mostly uncommented, and can be challenging to understand.

For example the rule `-[:c] l /[aelos] s\0\p[4310$] (?\p1[za] \p1[:c]` is a compressed way to express ten separate leetspeak rules, the first of which is "Lowercase the word. If the word contains an `a`, replace all occurrences of `a` by `4`." There are similar rules for transforming `e` to `3`, `l` to `1`, `o` to `0`, and `s` to `$`. Additionally, each such rule has a variant in which the first character of the resulting word is capitalized if it's alphabetic. Note that this rule replaces only one specific character by its leetspeak equivalent. Other rules are necessary to replace multiple characters in the same word.

To get a better idea of what these rules do, we can create a new version of `john.config.orig` named `john.config..single-rules-only` in which we replace the section labeled `[List.Rules:Wordlist]` by the rules in `[List.Rules:Single]`. Then we can run *John* in wordlist mode with the `--stdout` option on a handcrafted wordlist based on /etc/passwd to see all the transformations. The steps to do this are detailed below.

Step 1: Download `john.config..single-rules-only`

The file `john.config..single-rules-only` mentioned above is in the `cs342 download` folder on `cs.wellesley.edu`. In the `run` directory, download it as follows:

```
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ scp -r
    gdome@cs.wellesley.edu:/home/cs342/download/john.conf.single-rules-only .
```

Step 2: Install `john.config..single-rules-only` as `john.conf`

For this experiment, we need to change `john.conf` to `john.config.single-rules-only`. Do this as follows:

```
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ rm john.conf
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ ln -s john.conf.single-rules-only
    john.conf
```

Step 3: Create `cello.lst`

For our single crack experiment, we'll use a word list derived for user Alex Smith with account `cello`. In the `run` directory, make a file `cello.lst` with the following 15 lines:

```
cello
alex
smith
calex
csmith
acello
asmith
scello
salex
celloalex
cellosmith
alexcello
alexsmith
smithcello
smithalex
```

Step 4: Run the cracker

Now we run the password cracker in wordlist mode on `cello.lst` with the `--stdout` option:

```
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ ./john --wordlist=cello.lst
    --rules --stdout > cello-transformed.txt
words: 12835  time: 0:00:00:00 DONE (Sat Feb 20 18:19:17 2016)  w/s: 183357
    current: smithalex1900
```

Step 5: Study `cello-transformed.txt`

Remarkably, the `cello-transformed.txt` file contains 12,835 words — and that's just for one user! You should scroll through it to get a sense for the kinds of transformations applied in single crack mode.

It's worth noting that there are some single crack rules that are **not** applied in wordlist mode. Rules beginning with `-p` are rules that work on "word pairs". These rules are only active in single crack mode and cannot be simulated in wordlist mode:

```
# Some word pair rules...
# johnsmith -> JohnSmith, johnSmith
-p-c (?a 2 (?a c 1 [cl]
# JohnSmith -> john smith, john_smith, john-smith
-p 1 <- $[ _\-] + l
# JohnSmith -> John smith, John_smith, John-smith
-p-c 1 <- (?a c $[ _\-] 2 l
# JohnSmith -> john Smith, john_Smith, john-Smith
-p-c 1 <- l $[ _\-] 2 (?a c
# johnsmith -> John Smith, John_Smith, John-Smith
-p-c 1 <- (?a c $[ _\-] 2 (?a c
# Applying different simple rules to each of the two words
-p-[c:] 1 \p1[ur] 2 l
-p-c 2 (?a c 1 [ur]
-p-[c:] 1 l 2 \p1[ur]
-p-c 1 (?a c 2 [ur]
```

If you search through `cello-transformed.txt`, you won't find transformations like `AlexSmith`, `Alex_Smith`, `alex-smith`, even though these transformations *would* be generated within single crack mode.

Exercise 5: Password Cracking in Incremental Mode

To test incremental mode, we can used the invocation

```
./john --incremental unshadowed2
```

Figure 3 shows a transcript of a run in incremental mode.

In this mode, the cracker will generate all possible combinations of characters up to a given length (by default, the length is 8). However, as is evident from the transcript, it generates the candidates in a nonobvious order. You can read more about incremental mode and how to control it in `doc/MODES` and `doc/CONFIG`.

All Together Now

To run all three mode of the cracker (single crack mode, followed by wordlist mode, followed by incremental mode), just invoke `./john` on the unshadowed file. This is controlled by settings in the `john.conf` file, so before doing this, you'll first want to set `john.conf` back to `john.conf.orig`, and then possibly edit it.

```
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ rm john.conf
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ ln -s john.conf.orig john.conf
```

For example, the wordlist used by `./john` is specified in a line near the top of `john.conf.orig`. To use `password.lst`, for example, this line should read:

```
Wordlist = $JOHN/password.lst
```

As before, you can see the current candidate being tested by pressing the space bar.

To stop the cracker in such a way that you can restart it, type Ctrl-c Ctr-c. This saves the state of the cracker in the file `john.rec`. To restart the cracker from the saved point, use

```
./john --restore
```

```
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ ./john --incremental
   unshadowed2
Warning: detected hash type "sha512crypt", but the string is also
   recognized as "crypt"
Use the "--format=crypt" option to force loading these as that type
   instead
Loaded 6 password hashes with 6 different salts (sha512crypt [32/32])
Remaining 2 password hashes with 2 different salts
guesses: 0  time: 0:00:00:01 0.00%  c/s: 246  trying: 0064805
guesses: 0  time: 0:00:00:02 0.00%  c/s: 253  trying: steetine
guesses: 0  time: 0:00:00:03 0.00%  c/s: 255  trying: 49382118
guesses: 0  time: 0:00:00:04 0.00%  c/s: 256  trying: morris
guesses: 0  time: 0:00:00:05 0.00%  c/s: 257  trying: sanada
guesses: 0  time: 0:00:00:06 0.00%  c/s: 258  trying: pig
guesses: 0  time: 0:00:00:07 0.00%  c/s: 259  trying: salar
guesses: 0  time: 0:00:00:08 0.00%  c/s: 259  trying: allier
guesses: 0  time: 0:00:00:09 0.00%  c/s: 259  trying: artsis
guesses: 0  time: 0:00:00:10 0.00%  c/s: 258  trying: berom
guesses: 0  time: 0:00:00:11 0.00%  c/s: 258  trying: 0100379
guesses: 0  time: 0:00:00:12 0.00%  c/s: 258  trying: 0139141
guesses: 0  time: 0:00:00:13 0.00%  c/s: 258  trying: shilert
guesses: 0  time: 0:00:00:14 0.00%  c/s: 259  trying: sheenes
guesses: 0  time: 0:00:00:15 0.00%  c/s: 259  trying: spiel
guesses: 0  time: 0:00:00:16 0.00%  c/s: 259  trying: bugbr
guesses: 0  time: 0:00:00:17 0.00%  c/s: 260  trying: santine
guesses: 0  time: 0:00:00:18 0.00%  c/s: 260  trying: 0239986
guesses: 0  time: 0:00:00:19 0.00%  c/s: 261  trying: salanta
guesses: 0  time: 0:00:00:20 0.00%  c/s: 260  trying: suness
guesses: 0  time: 0:00:00:21 0.00%  c/s: 260  trying: sussey
guesses: 0  time: 0:00:00:22 0.00%  c/s: 260  trying: mereed
guesses: 0  time: 0:00:00:23 0.00%  c/s: 260  trying: menisa
guesses: 0  time: 0:00:00:24 0.00%  c/s: 260  trying: asd102
guesses: 0  time: 0:00:00:25 0.00%  c/s: 261  trying: assep1
guesses: 0  time: 0:00:00:26 0.00%  c/s: 261  trying: 0224953
guesses: 0  time: 0:00:00:27 0.00%  c/s: 261  trying: rji
guesses: 0  time: 0:00:00:28 0.00%  c/s: 261  trying: stephie
guesses: 0  time: 0:00:00:29 0.00%  c/s: 261  trying: 0078740
guesses: 0  time: 0:00:00:30 0.00%  c/s: 261  trying: staffic
guesses: 0  time: 0:00:00:31 0.00%  c/s: 261  trying: marting
guesses: 0  time: 0:00:00:32 0.00%  c/s: 261  trying: macking
guesses: 0  time: 0:00:00:33 0.00%  c/s: 261  trying: mictus1
guesses: 0  time: 0:00:00:34 0.00%  c/s: 261  trying: minden1
guesses: 0  time: 0:00:00:35 0.00%  c/s: 261  trying: morthen
guesses: 0  time: 0:00:00:36 0.00%  c/s: 261  trying: moottlE
guesses: 0  time: 0:00:00:37 0.00%  c/s: 261  trying: bonet
guesses: 0  time: 0:00:00:38 0.00%  c/s: 261  trying: stonny
guesses: 0  time: 0:00:00:39 0.00%  c/s: 261  trying: alivo1
guesses: 0  time: 0:00:00:40 0.00%  c/s: 262  trying: shish2
```

Figure 3: Transcript of a incremental mode. Lines beginning `guesses:` were printed out each time the space bar was pressed. Blank lines have been edited out.

Restoring Your Settings

After performing the experiments on this handout, it's important to restore the state of the password cracker to its original state. You can do that via the following steps.

Step 1: Restore `john.conf` to `john.conf.orig`

```
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ rm john.conf
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ ln -s john.conf.orig john.conf
```

Step 2: Restore `john.pot` to `john.rec`

If you saved the files `john.pot` to `john.rec` earlier, you should restore them now:

```
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ cp \texttt{john.pot.saved}
    \texttt{john.pot}
wendy@cs342-ubuntu-1:~/john-1.7.9-jumbo-7/run$ cp \texttt{john.rec.saved}
    \texttt{john.rec}
```

I suggest using `cp` rather than `mv` in the last step so that you still have the `.saved` files should you want to use them again.