Compiling from source

Compiling an application from source essentially means that you are manually creating a package using the source code.

In this example, I'm compiling the John the Ripper password cracker from source, on an Ubuntu machine.

To being, ensure that the required packages are installed:

```
apt install make gcc gzip
```

download and unzip (if zipped) source code file

```
root@test:~# wget https://www.openwall.com/john/k/john-1.9.0.tar.gz
--2024-05-23 15:03:10--  https://www.openwall.com/john/k/john-1.9.0.tar.gz
Resolving www.openwall.com (www.openwall.com)... 193.110.157.242
Connecting to www.openwall.com (www.openwall.com)|193.110.157.242|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 13110145 (13M) [application/octet-stream]
Saving to: ‘john-1.9.0.tar.gz’

john-1.9.0.tar.gz
100%[=============================================================================>]  12.50M
8.23MB/s    in 1.5s

2024-05-23 15:03:11 (8.23 MB/s) - ‘john-1.9.0.tar.gz’ saved [13110145/13110145]

root@test:~# ls -l
total 12808
-rw-r--r-- 1 root root 13110145 Apr 12  2019 john-1.9.0.tar.gz

root@test:~# tar -xvzf john-1.9.0.tar.gz
```

cd into the source code directory

```
root@test:~# ls -l
total 12812
```

```
drwxr-xr-x 5 root root     4096 May 23 15:03 john-1.9.0
-rw-r--r-- 1 root root 13110145 Apr 12  2019 john-1.9.0.tar.gz

root@test:~# cd john-1.9.0
root@test:~/john-1.9.0# ls -l
total 12
drwxr-xr-x 2 root root 4096 May 23 15:03 doc
drwxr-xr-x 2 root root 4096 May 23 15:03 run
drwxr-xr-x 2 root root 4096 May 23 15:03 src

root@test:~/john-1.9.0# cd src/
```

Within the src directory, you'll see the actual source code files - there can sometimes be many thousands of these files depending on the application.

run the make command to view the available options for compilation:

```
make
```

This will typically show the available options for hardware (CPU) compatibility, as shown below:

```
root@test:~/john-1.9.0/src# make
To build John the Ripper, type:
        make clean SYSTEM
where SYSTEM can be one of the following:
linux-x86-64-avx512      Linux, x86-64 with AVX-512 (some 2017+ Intel CPUs)
linux-x86-64-avx2        Linux, x86-64 with AVX2 (some 2013+ Intel CPUs)
linux-x86-64-xop         Linux, x86-64 with AVX and XOP (some AMD CPUs)
linux-x86-64-avx         Linux, x86-64 with AVX (some 2011+ Intel CPUs)
linux-x86-64             Linux, x86-64 with SSE2 (most common)
linux-x86-avx512         Linux, x86 32-bit with AVX-512 (some 2017+ Intel CPUs)
linux-x86-avx2           Linux, x86 32-bit with AVX2 (some 2013+ Intel CPUs)
linux-x86-xop            Linux, x86 32-bit with AVX and XOP (some AMD CPUs)
linux-x86-avx            Linux, x86 32-bit with AVX (2011+ Intel CPUs)
linux-x86-sse2           Linux, x86 32-bit with SSE2 (most common, if 32-bit)
linux-x86-mmx            Linux, x86 32-bit with MMX (for old computers)
linux-x86-any            Linux, x86 32-bit (for truly ancient computers)
linux-mic                Linux, Intel MIC (first generation Xeon Phi)
linux-arm64le            Linux, ARM 64-bit little-endian w/ASIMD (best)
linux-arm32le-neon       Linux, ARM 32-bit little-endian w/NEON (best 32-bit)
linux-arm32le            Linux, ARM 32-bit little-endian
```

```
linux-alpha              Linux, Alpha
linux-sparc64            Linux, SPARC 64-bit
```

To view the current CPU architecture of your system run:

```
name -a
```

In this example, the system is running x86_64:

```
root@test:~# uname -a
Linux test 5.15.0-106-generic #116-Ubuntu SMP Wed Apr 17 09:17:56 UTC 2024 x86_64 x86_64
x86_64 GNU/Linux
```

Next, we want to compile the code using the correct CPU architecture:

```
make clean linux-x86-64
```

Now that the code has been compiled we can access the binary for the application. In this example, the binary is located within the run directory:

```
root@test:~/john-1.9.0/run# pwd
/root/john-1.9.0/run
root@test:~/john-1.9.0/run# ls -l
total 20084
-rw------- 1 root root 4086722 May 29  2013 alnum.chr
-rw------- 1 root root 1950539 May 29  2013 alpha.chr
-rw------- 1 root root 5720262 May 29  2013 ascii.chr
-rw------- 1 root root  465097 May 29  2013 digits.chr
-rwxr-xr-x 1 root root  323680 May 23 15:06 john
-rw------- 1 root root   35972 Mar 21  2019 john.conf
-rw------- 1 root root 1184244 May 29  2013 lm_ascii.chr
-rw------- 1 root root 1161863 May 29  2013 lower.chr
-rw------- 1 root root 2464980 May 29  2013 lowernum.chr
-rw------- 1 root root 1209621 May 29  2013 lowerspace.chr
-rwx------ 1 root root    1432 May 29  2013 mailer
-rwx------ 1 root root     842 May 29  2013 makechr
-rw------- 1 root root   26325 May 29  2013 password.lst
-rwx------ 1 root root    4782 May 29  2013 relbench
lrwxrwxrwx 1 root root       4 May 23 15:06 unafs -> john
lrwxrwxrwx 1 root root       4 May 23 15:06 unique -> john
lrwxrwxrwx 1 root root       4 May 23 15:06 unshadow -> john
```

```
-rw------- 1 root root  668568 May 29  2013 upper.chr
-rw------- 1 root root 1220961 May 29  2013 uppernum.chr
```

Time to test.

```
root@test:~/john-1.9.0/run# ./john --test
Benchmarking: descrypt, traditional crypt(3) [DES 128/128 SSE2]... DONE
Many salts:     5636K c/s real, 5647K c/s virtual
Only one salt:  5386K c/s real, 5386K c/s virtual
```

Revision #2
Created 2023-09-24 12:02:47 UTC by Daniel
Updated 2024-05-23 16:08:35 UTC by Daniel