Chunky

a suite of wordlist mangling tools

Chunky is a suite of tools which can be linked together in a powerful piping structure to create wordlists exactly like you want them. The results can be piped into a text file to create a dictionary, or directly into chunky's brute force testing modules.

Do to Chunky's piping structure (it pipes all output into the next node), I am unable to put any type of usage statements into the programs, so I have them all here. If you use the programs wrong, it will most likely terminate itself.

We will use two files for all examples. The contents of the files are as follows:

wordlist1.txt	wordlist2.txt
a	1
Bad	42
c	3

Chunky is (currently) composed of three different mangling programs in addition to the cracking modules:

1) feednode

- feednode takes one parameter: a wordlist. One word per line. It reads this wordlist line-by-line and outputs each word with 'cout'. By default, 'cout' outputs to the screen, but it is intended that you 'pipe' the output to another mangling program. Ex.

```
C:\chunky\feednode wordlist1.txt

a

Bad

c

C:\chunky\feednode wordlist1.txt > wordlist1_copy.txt
```

Example 1 printed the contents of wordlist1.txt to the screen. Example 2 output the words into another file, effectively copying wordlist1.txt.

feednode is intended to be used as a starting "node" for all of chunky's configurations. Each program is referred to as a "node", and feednode is the first one which is called, feeding out a wordlist to cout where it can be printed to the screen, piped to a file, or piped to another node.

2) appendicts

- appendicts takes one or two parameters: a wordlist and an optional parameter '-a'. It assumes that input will be piped to it on the command line from another node. It then takes each word of the wordlist supplied, and appends it to each word piped in from the previous node. If the optional '-a' flag is supplied, it tells appendicts to output the *original* words that were piped in, as well. Ex.

```
C:\chunky\feednode wordlist1.txt | appendicts wordlist2.txt

a1
```

```
a42

a3

Bad1

Bad42

Bad3

c1

c42

c3

C:\chunky\feednode wordlist1.txt | appendicts wordlist2.txt -a

a

a1

a42

a3

Bad

Bad1

Bad42

Bad3

c

c1

c42

c3

C:\chunky\feednode wordlist1.txt | appendicts wordlist1.txt

aa

aBad

ac

Bada

BadBad

Badc

ca

cBad

cc
```

Example 1 appended each word from wordlist2.txt to each word of wordlist1.txt and output the results to the screen. Example 2 did the same thing, but also forwarded each word from wordlist1.txt in its original format. Example 3 demonstrates that any combination of wordlists can be used, even the same wordlist.

You can make as sophisticated of rules as you want:

```
C:\chunky\feednode wordlist1.txt | appendicts wordlist1.txt | appendicts wordlist2.txt > results.txt
```

appends each word of wordlist1.txt to each word of wordlist1.txt, then appends each word of wordlist2.txt to each of *those* words and outputs the results to 'results.txt'.

3) toggle_case

- toggle_case takes one or two parameters: a wordlist and an optional parameter '-a'. It assumes that input will be piped to it on the command line from another node. It then takes each word piped in from the previous node and switches the case of the first character. **THE FIRST CHARACTER ONLY!** No other characters are touched. If the optional '-a' flag is supplied, it tells toggle_case to output the *original* words that were piped in, as well. Ex.

```
C:\chunky\feednode wordlist1.txt | toggle_case

A

bad

C

C:\chunky\feednode wordlist1.txt | toggle_case -a

a

A

Bad

bad

c

C

C:\chunky\feednode wordlist1.txt | toggle_case -a > results.txt
```

Example 1 toggled the case of the first character of each word of wordlist1.txt. Example2 did the same, but also forwarded each word from wordlist1.txt in its original format. Example 3 did the same as example 2 did, but demonstrates how the output can be piped into a text file.

the cracking modules:

Sometimes actually generating a text file containing every candidate for complicated mangling configurations may not be the best idea. Chunky allows the flexibility to perform various mangling rules on the command line and pipe the output directly into a cracker module as such:

```
C:\chunky\feednode ctEnglish.txt | appendicts ctEnglish.txt -a | appendicts ctEnglish.txt -a |
appendicts 0-999.txt -a | chunky_md5 A25DF8D4B2D152DD924ACBA5FC7C2682
```

there are cracking modules for md4, md5, kerberos5 preauth, mscache, and ntlm - single and multihash for each algorithm. These crackers are scalar 32-bit, but the hashing routines are highly optimized. It's difficult to benchmark their performance, especially as it likely depends on the mangling rule configuration which formulates the words fed into the cracking modules. Following is a listing of each cracking module and their command line syntax:

chunky_md4, chunky_md5, chunky_ntlm

```
... | chunky_... <32-byte hash>
```

ex. `feednode words.txt | chunky_md5 A25DF8D4B2D152DD924ACBA5FC7C2682`

chunky_mscache

```
... | chunky_mscache <32-byte hash> <username>
```

ex. `feednode words.txt | chunky_mscache C30301545B3C18BAECF4339147335DFA`

chunky_kerb5

```
... | chunky_kerb5 <104-byte encrypted timestamp> <year packet was recovered>
```

ex. `feednode words.txt | chunky_kerb5 02E837D06B2AC76891F388D9CC36C67A2A9785BF5036C45D3843490BF9C228E8C18653E10CE58D7F8EF119D2EF4F92B1803B1451 2008`

chunky_md4_multihash, chunky_md5_multihash, chunky_ntlm_multihash

```
... | chunky_... <hash file>
```

ex. `feednode words.txt | chunky_md5_multihash md5hashes2crk.txt`

chunky_mscache_multihash

```
... | chunky_mscache_multihash <hash file> <username file>
```

ex. `feednode words.txt | chunky_mscache_multihash mscachehashes.txt usernames.txt`

chunky_kerb5_multihash

```
... | chunky_kerb5_multihash <hash file> <year packets were recovered>
```

ex. `feednode words.txt | chunky_kerb5_multihash enctimestamps.txt 2008`

Examples:

As you can see, chunky provides great flexibility and scalability within the rules of its nodes. Here is an example of how a sophisticated wordlist could be created using chunky:

```
C:\chunky\feednode English.txt | toggle_case -a > ctEnglish.txt

C:\chunky\feednode ctEnglish.txt | appendicts ctEnglish.txt -a | appendicts ctEnglish.txt -a |
appendicts 0-999.txt -a > ctEnglishPow3_0-99.txt
```

would create a wordlist which would recover the following passwords:

bigfatliar85 (wordwordwordnumber)

password123 (wordnumber)

BoysLikeGirls (WordWordWord)

Blink182 (Wordnumber)

as well as many others...

many, *many* others. In this scenario, I would pipe the output directly into a cracking module.

author: Nathanael Warren

created: 08/14/09

last updated: 09/04/09