pydictor爆破字典生成指南

作者: LANDGREY ● 创建时间 2017年11月21日 00:17 ● 更新时间 2018年2月27日 01:22

0x00：简介

pydictor是一个使用python语言开发，遵循GPLv3协议的开源命令行工具，主要用来帮助安全研究人员生成称心如意的暴力破解字典。

以功能强大、简洁实用、适用场景多、自定义程度强为开发目标。

开源地址：pydictor

0x01：特点与功能

今天主要是讲pydictor如何结合渗透测试过程常见的场景使用，特点与功能REAME有详细讲解，下面只梳理一下大概脉络，方便下文的理解。

特点:

1. 完全使用python的原生库写成，不需要额外安装其它任何的python模块;
2. 同时支持python 2.7+ 和python 3.4+版本，可在Windows、Linux和Mac平台上运行;
3. 可自定义化程度高，留出很多可配置规则的文件;
4. 爆破必备,新老皆宜.

功能:

1. 基于三大字符集(d:数字 L:小写字母 c:大写字母)的基础字典;
2. 基于自定义字符集(包括特殊字符)的字典;
3. 排列组合字典(几个字符或字符串的所有排列可能);
4. 用配置文件或者符合pydictor字典语法的字符串直接生成字典;
5. 析取网页中可能有意义的原始单词字典;
6. 基于关键词生成针对性密码字典;
7. 基于性别生成中国公民身份证后4/6/8位字典;
8. 生成一段时间内的生日字典(自定义位数);
9. 用pydictor的handler功能润色下自己的字典;
10. 基于个人信息和规则生成社会工程学字典(呃，蹭下知名度，本质还是基于关键词，重在密码规则模式)
11. 一系列和字典的整个生命周期有关的内置工具;
    包括字典合并、合并后去重、字典去重、单词频率统计、安全擦除字典;
12. 一系列和生成优化字典有关的选项;
    包括自定长度范围、字典加前缀、加后缀、编码或加密字典、用1337模式、控制字典所用规则的程度、根据数字、字符和特殊

0x02：使用场景

早期开发是为了让功能匹配使用场合，后期开发是让具体场景拥有对应的功能。

01：字典合并

字典都不是凭空捏造或生成的，一般都会参考前辈们公布的字典。所以，先收集百八十个字典，放到一个目录下，把字典合并起来吧。

1. 合并目录/网站路径爆破字典
2. 合并子域名字典
3. 合并用户名字典
4. 合并弱密码字典
5. 其它各式各样的字典

```
python pydictor.py -tool combiner /my/dict/dirpath -o comb.txt
```

02：词频统计

但是有时候我们通常不需要那么大的字典，选合并后字典的出现频率最高的前1000条保存吧。筛选出

```
最常用的网站路径/子域名/用户名/弱密码/...
```

修改lib/data/data.py中counter_split变量指定的分隔符(默认"\n")，也可以统计其它字符分隔的字典词频.

```
python pydictor.py -tool counter vs comb.txt 1000
```

03：去除重复项

面对合并后的超大字典，还是不舍得只要频率高的词，路径字典有时候还是多多益善。去重下，照单全收

```
python pydictor.py -tool uniqifer comb.txt --output uniq.txt
```

或者直接合并加去重

```
python pydictor.py -tool uniqbiner /my/dict/dirpath --output uniq.txt
```

04：枚举数字字典

准备好字典了，拿最基础的试试手

1. 爆破4位或6位数字手机短信验证码
2. 爆破用户名ID值

生成4位纯数字字典

```
python pydictor.py -base d --len 4 4
```

05：简单用户名字典

不能确定是否存在某用户时，试试1位到3位的拼音字典，加上123456这样的几个弱口令，说不定就有意外收获：

```
python pydictor.py -base L --len 1 3 -o dict.txt
```

06：后台管理员密码字典(明文传输)

经常遇到的测试场景了，就是一个登录页，把收集到的信息都用上，生成后台爆破字典，比如

```
域名:test.land.com.cn
编辑名:张美丽、Adaor、midato
公司名:上海美丽大米有限责任公司(如有雷同纯属巧合)
座机:568456
地址:xxx园区A座312室
```

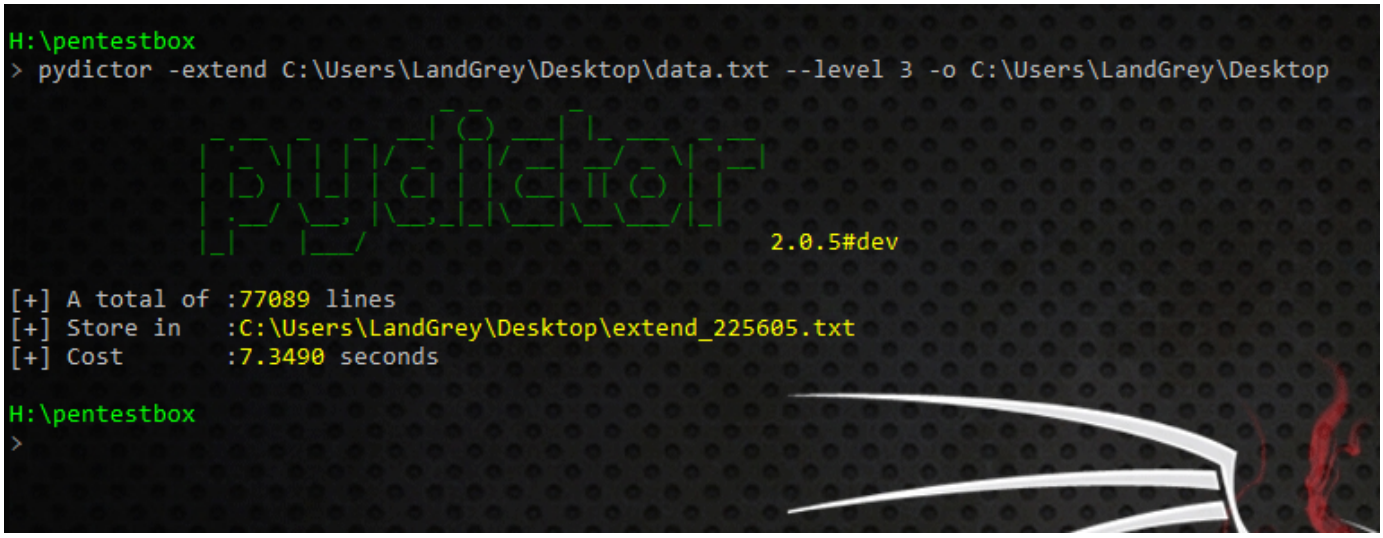
把自己常用的弱口令字典复制到wordlist/Web 目录下，最终生成的字典会包含它们；

然后把下列信息写入/data.txt

```
test
land
zhangmeili
meili
zml
Adaor
midato
meilidami
mldm
shmldm
568456
A312
```

生成字典：

```
python pydictor.py -extend /data.txt --level 3 --len 4 16
```



弱口令字典 + 部分信息 + 生成规则 + level 3，最终生成了七万多条密码，一部分密码如下：

```
28815   !@#meili
28816   qwemeili
28817   QWEmeili
28818   Meili
28819   Meili.
28820   Meili!
28821   Meili#
28822   Meili@
28823   Meili?
28824   Meili\
28825   Meili/
28826   MeiliA
28827   Meili_
28828   Meilia
28829   Meili000
28830   Meili007
28831   Meili321
28832   Meili520
28833   Meili521
28834   Meili678
28835   Meili789
28836   Meili999
28837   Meili@qq
28838   MeiliABC
28839   MeiliXYZ
28840   Meilixyz
28841   Meili_qwer
28842   Meili11111
28843   Meili12345
28844   Meili54321
28845   Meiliadmin
28846   Meili
28847   Meiliabc123
28848   Meili123abc
28849   Meili123!@#
28850   Meili654321
28851   Meili1980
28852   Meili1981
28853   Meili1982
28854   Meili1983
```

07：后台管理员密码字典(前台普通加密)

有时候网站的密码可能不是直接明文传输过去的，程序员会用js简单加密下再传输过去，比如base64编码、md5加密，这时候可以用--encode参数生成加密字典

```
python pydictor.py -extend /data.txt --level 3 --len 4 16 --encode b64
python pydictor.py -extend /data.txt --level 3 --len 4 16 --encode md5
```

08：后台管理员密码字典(前台js自定义加密)

高级点的程序员，还喜欢前端自定义个js加密方法，把用户名和密码加密后传输过去，比如

这时候，普通爆破工具基本都无能为力了，但是却依旧可以通过pydictor来生成字典；

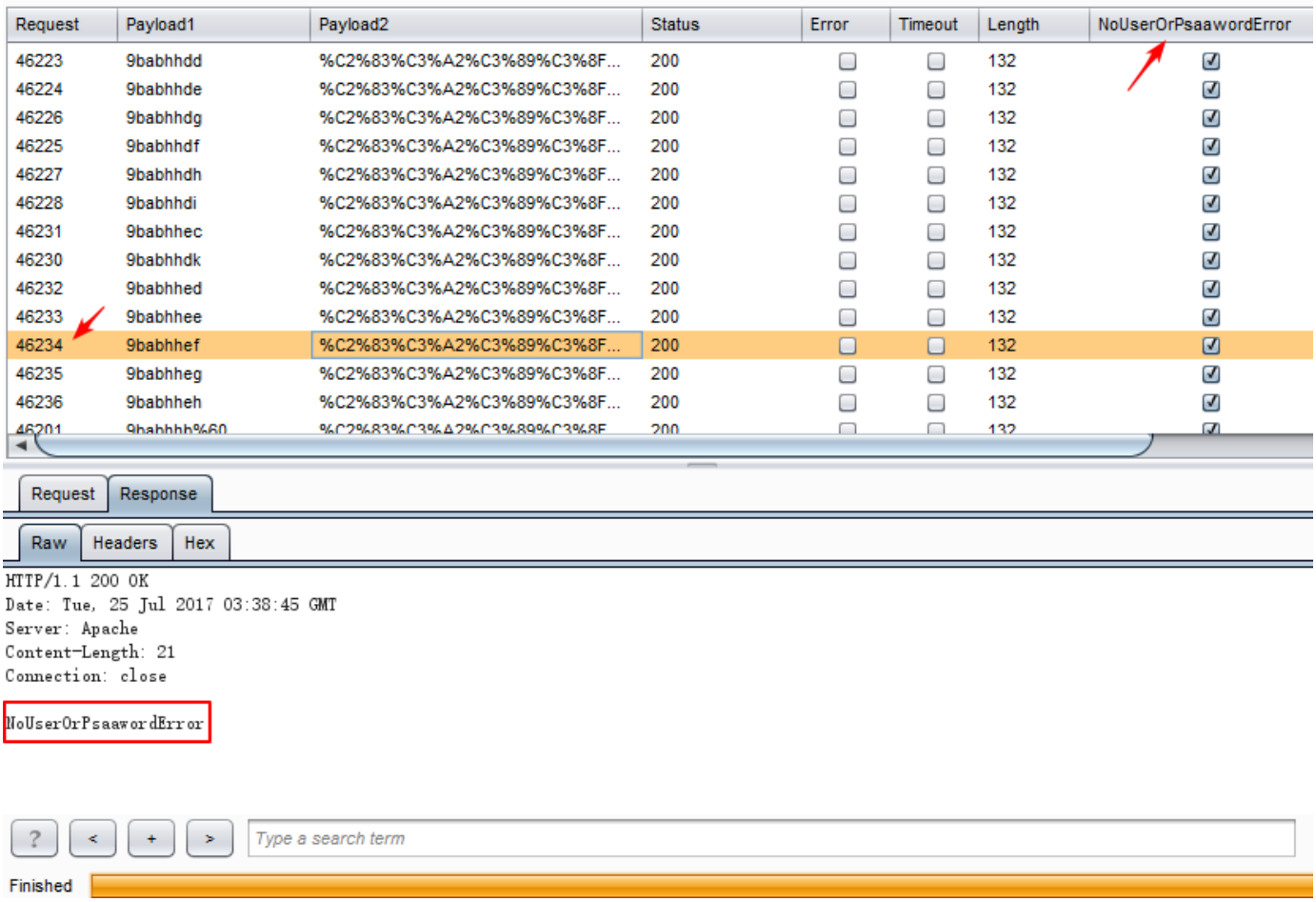
修改/lib/fun/encode.py 文件的 test_encode()函数，用python语法仿照上图的加密方式再实现一遍加密：

```
def test_encode(item):
    c = chr(ord(item[0]) + len(item))
    for i in range(1, len(item)):
        c += chr(ord(item[i]) + ord(item[i - 1]))
    return quote(c)
```

然后运行命令，生成按照前端js加密方法加密后的密码字典，可以直接用burpsuite加载

```
python3 pydictor.py -extend /data.txt --level 3 --len 4 16 --encode test
```

最后通过这种方式生成符合前端加密方法的用户名字典，先探测出存在的用户名，再结合几个弱密码，爆破出来100多个弱口令。



需要注意的是，一般生成加密字典前要生成一个没加密的字典，因为每一项在文件中的顺序是一致的，所以爆破出来密码后，可以通过行数对照去没加密的字典中查找明文。

09：复杂格式的字典

例如，你通过shoulder hack和一些信息，猜到别人的密码大概是

```
Cxhai【三位或四位数字】_abc123@【qq,163,wy,mail中的一个】，然后md5加密的值
```

这种复杂格式的字典，pydictor也可以轻松的生成

```
python pydictor.py --conf "Cxhai[0-9]{3,4}<none>_abc123@[qq,163,wy,mail]{1,1}<none>" --encode md5
```

没加密前的字典：

```
16847   Cxhai3211_abc123@wy
16848   Cxhai3211_abc123@mail
16849   Cxhai3212_abc123@qq
16850   Cxhai3212_abc123@163
16851   Cxhai3212_abc123@wy
16852   Cxhai3212_abc123@mail
16853   Cxhai3213_abc123@qq
16854   Cxhai3213_abc123@163
16855   Cxhai3213_abc123@wy
16856   Cxhai3213_abc123@mail
16857   Cxhai3214_abc123@qq
16858   Cxhai3214_abc123@163
16859   Cxhai3214_abc123@wy
16860   Cxhai3214_abc123@mail
16861   Cxhai3215_abc123@qq
16862   Cxhai3215_abc123@163
16863   Cxhai3215_abc123@wy
16864   Cxhai3215_abc123@mail
16865   Cxhai3216_abc123@qq
16866   Cxhai3216_abc123@163
16867   Cxhai3216_abc123@wy
16868   Cxhai3216_abc123@mail
16869   Cxhai3217_abc123@qq
16870   Cxhai3217_abc123@163
16871   Cxhai3217_abc123@wy
16872   Cxhai3217_abc123@mail
16873   Cxhai3218_abc123@qq
16874   Cxhai3218_abc123@163
16875   Cxhai3218_abc123@wy
16876   Cxhai3218_abc123@mail
16877   Cxhai3219_abc123@qq
16878   Cxhai3219_abc123@163
16879   Cxhai3219_abc123@wy
16880   Cxhai3219_abc123@mail
16881   Cxhai3220_abc123@qq
16882   Cxhai3220_abc123@163
16883   Cxhai3220_abc123@wy
16884   Cxhai3220_abc123@mail
16885   Cxhai3221_abc123@qq
16886   Cxhai3221_abc123@163
```

最终加密后的字典：

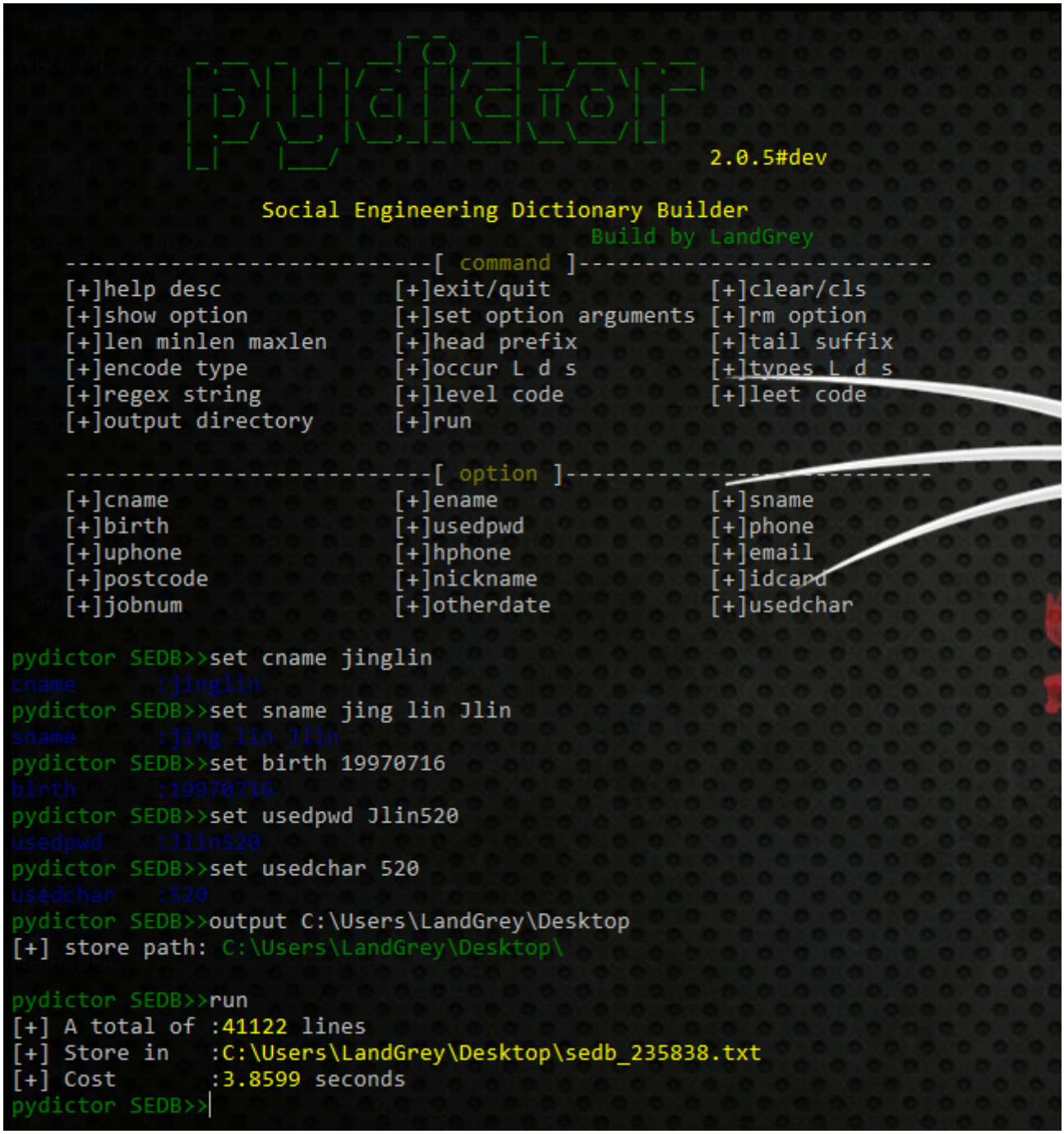
```
24235 b9177775af7561176f5bb000a98f0757
24236 db72d3a928bebdf686ce6d95d5829999
24237 75b46701ec763f80f8ddd01c32beee08
24238 fbfdfb4c29120640edbe53e283edfecb
24239 52a844bbb456ede007919ef9c87f9bbc
24240 c5101498b6fc95cbae221613ff4493ec
24241 02d7c3695b40422c22d1a7211f421f8d
24242 1dcabe457a8df2b437619561f1a39b3a
24243 03e1329df63fddf9b21b8457bf97d367
24244 092098749ca06e68d7403b2f2d561720
24245 7c4d8f2ee54530d0d9dce47119d35d00
24246 f20affdbbff5d9ae77a4d22da87568f2
24247 eff180ae4fe03bdb127bc6281f780c52
24248 4e5604d0735fab6c36eb98faa390e8af
24249 ac505e57963c70bbb6bfd96275fea8b6
24250 84fab8de60acc003d23dc13dada92617
24251 2d1a6b62b6401c231fc2292425158f6d
24252 d0f638fa5a8c3c63e7051273a93f1d2d
24253 6fa5b8dc2130d2523354dd8878ce7165
24254 770f20f67ff28ad96b80316c3981fd96
24255 fbe4df492fee0f668814383929 63fe84
24256 08172cba1f0f9707b6812a7b7c3e0667
24257 679b6b1c9b6c598d6ebe506437863600
24258 b6473c02b9a5f5e60c6d8ba9eb1acc5e
24259 4d5a396dee471dedb7a5431d4d81d5fd
24260 e7762df81e201551502987d9833b4f0c
24261 f458709bd8a8833f0c22c44036730010
24262 6c26a4d6aa94d3dac60aa7c794a733ae
24263 eb8301c7f37605c420cc552454747ca2
24264 2c7d8cca85a09520bbec4dec98f073f1
24265 2813362014118a819a97ab70144d9546
24266 a2976fe13c32ccb21a206eabb2f118cf
24267 e10494afa0a8e13c35f34e24c6ec1b5b
24268 26b641af8b6eeb04cb2c120ec0c48a0c
24269 d38cf9fdbb82e764b2b5fc53e4aa91de
24270 985f5e6d50ad44df934bf06095650ac8
24271 908763047dcc102361eab6a1422a9aba
24272 f58d0bcb3ad87e87f253472835c04362
24273 e58bc55f9351fedc72c70c060cc227f4
24274 aeb0a4c319f829761167f5894b24fab1
```

10：社会工程学字典

通过配置文件定义的规则和一部分内置代码逻辑，你可以输入一些关于个人的信息，生成关于某个人可能用的密码，比如，我只知道一个的如下信息

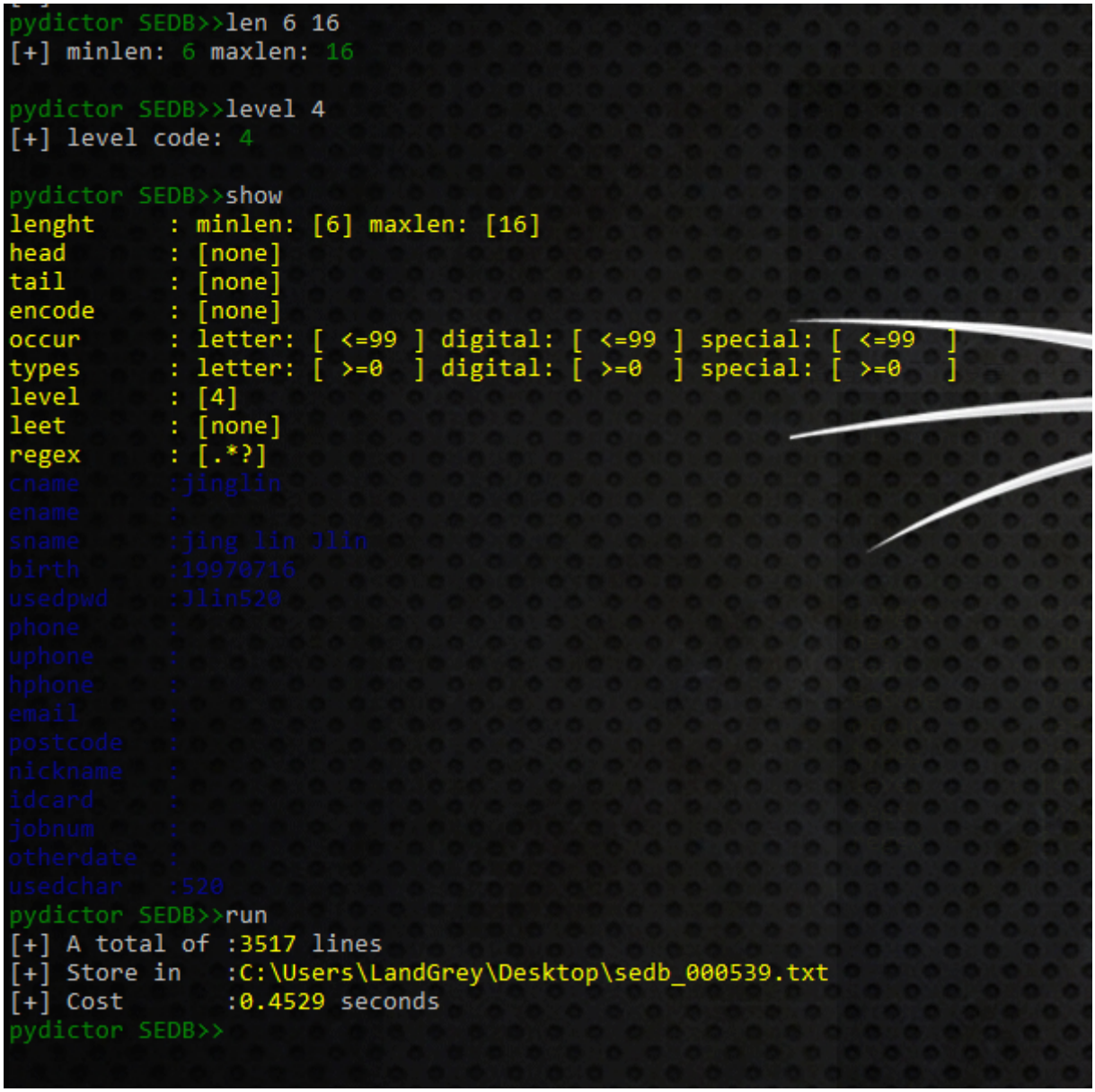
```
姓名： 景林
生日：1997年7月16日
以前用过密码：Jlin520
```

然后一波操作，生成了四万多条密码



嫌密码太多了？没事，只要长度6-16的，级别设置大点，密码会少很多；

查看下当前配置，重新生成字典，只有三千多条了


pydictor SEDB>>len 6 16
[+] minlen: 6 maxlen: 16
pydictor SEDB>>level 4
[+] level code: 4
pydictor SEDB>>show
lenght : minlen: [6] maxlen: [16]
head : [none]
tail : [none]
encode : [none]
occur : letter: [<=99] digital: [<=99] special: [<=99]
types : letter: [>=0] digital: [>=0] special: [>=0]
level : [4]
leet : [none]
regex : [.*?]
cname :jinglin
ename :
sname :jing lin Jlin
birth :19970716
usedpwd :Jlin520
phone :
uphone :
hphone :
email :
postcode :
nickname :
idcard :
jobnum :
otherdate :
usedchar :520
pydictor SEDB>>run
[+] A total of :3517 lines
[+] Store in :C:\Users\LandGrey\Desktop\sedb_000539.txt
[+] Cost :0.4529 seconds
pydictor SEDB>>

```
jinglin
Jinglin
Jlin520
jlin520
19970716
970716
1997716
jinglinjinglin
jinglinJinglin
jinglin19970716
jinglin07161997
jinglin970716
jinglin071697
jinglin1997
jinglin0716
jinglin1997716
jinglin716
jinglin97716
jinglin71697
Jinglinjinglin
JinglinJinglin
Jinglin19970716
Jinglin07161997
Jinglin970716
Jinglin071697
Jinglin1997
Jinglin0716
Jinglin1997716
Jinglin716
Jinglin97716
Jinglin71697
19970716jinglin
19970716Jinglin
1997071619970716
1997071607161997
19970716970716
19970716071697
199707161997
199707160716
199707161997716
```

11：处理自己的字典

退一万步来讲，上面的字典都帮不了你，但是pydictor的handler功能还是可以帮你根据具体的使用场景来处理自己的字典，让自己原本的字典适用各种场合。

比如:

对方密码策略要求是6到16位；必须有数字和字母，不允许有特殊字符；前端js对密码base64编码后传输到后端。

可以用下面的命令处理自己原先的字典raw.txt，生成符合本次爆破场景的字典：

```
python pydictor.py -tool handler /wordlist/raw.txt --len 6 16 --occur ">0" ">0" "<=0" --er
```

0x03：结语

pydictor的常见使用场景都简单介绍过了，另外还有一些特殊字典，比如身份证后几位、生日日期字典；内置的专门用来破解SSH弱口令的键盘模式字典 等等，就不一一介绍了，相信自己看看就能理解。

结合目标的爆破场景，合理使用pydictor，人人都是爆破小能手。

<