[REPORT - 실습2]

-Brute Force Attack-

과 목 명 : 컴퓨터보안 - 02
교 수 : 김영부 교수님
학번 : 2020112119
이름 : 강동희
제출일 : 2023.03.28

목 차

1 서 론

1.1 문제 정의

1. **Brute Force Attack (무작위 대입 공격)**
   4 ~ 8자리의 패스워드를 무작위로 10개 생성하여 Brute Force Attack(모든 경우의 문자를 생성하여 비밀번호를 찾을 때까지 비교)을 구현하여 분석.

1.2 문제 분석

1. **Brute Force Attack[1] (무작위 대입 공격)**
   Password Cracking 방법 중 하나이며, 특정 암호를 풀기 위해 가능한 모든 값을 대입하는 것을 의미한다. 이론적으로 충분한 시간이 존재한다면 암호화된 정보를 해독할 수 있지만, 취약점 측면에서 Brute Force Attack보다 빠른 공격이 존재한다. Brute Force Attack에도 다양한 유형이 존재한다. 여러 계정의 비밀번호를 대상으로 흔히 사용된 패스워드를 시도할 수 있으며, Dictionary/다른 데이터 침해에서 획득한 일반적인 비밀번호를 시도할 수 있기도 한다. 실습에서는 숫자, 알파벳, 특수문자를 조합하여 모든 경우의 수를 대입하여 매칭하는 방법을 이용하여 비밀번호를 알아낸다.

2. **Brute Force Attack의 이론적 요소**
   조합 가능한 모든 문자열을 대입하므로, 실습에서 진행하고자 하는 경우에 대한 시간 복잡도는 다음 <표1>과 같다.

-	숫자 (10)	알파벳 (52)	특수문자 (32)	숫자 알파벳 (62)	숫자 특수문자 (42)	알파벳 특수문자 (84)	숫자 알파벳 특수문자 (94)
4자리	$O(10^4)$	$O(52^4)$	$O(32^4)$	$O(62^4)$	$O(42^4)$	$O(84^4)$	$O(94^4)$
5자리	$O(10^5)$	$O(52^5)$	$O(32^5)$	$O(62^5)$	$O(42^5)$	$O(84^5)$	$O(94^5)$
6자리	$O(10^6)$	$O(52^6)$	$O(32^6)$	$O(62^6)$	$O(42^6)$	$O(84^6)$	$O(94^6)$
7자리	$O(10^7)$	$O(52^7)$	$O(32^7)$	$O(62^7)$	$O(42^7)$	$O(84^7)$	$O(94^7)$
8자리	$O(10^8)$	$O(52^8)$	$O(32^8)$	$O(62^8)$	$O(42^8)$	$O(84^8)$	$O(94^8)$

<표 1. 비밀번호 조합과 자릿수에 따른 시간 복잡도>

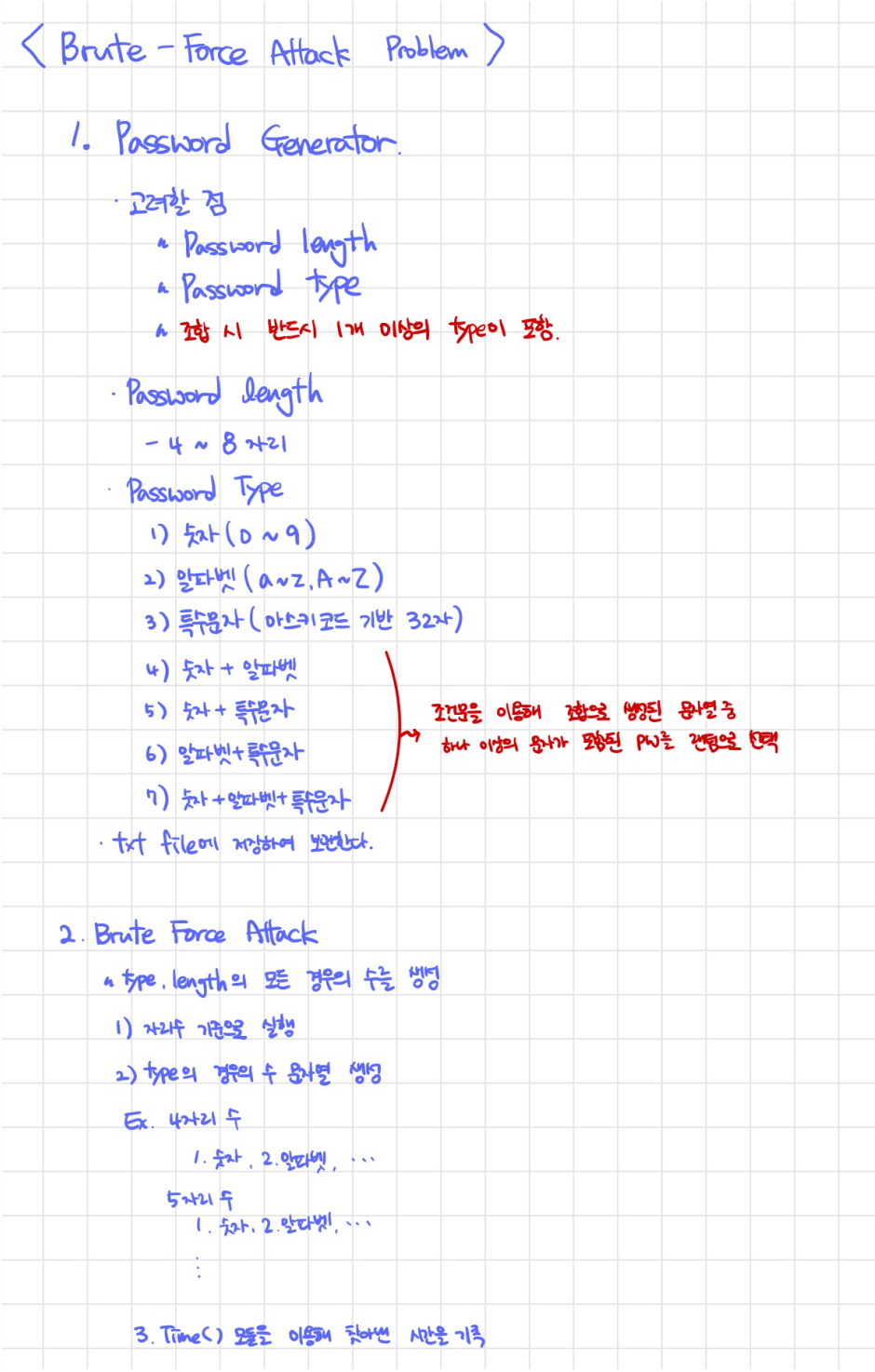
암호 생성에 사용되는 문자 조합의 개수나 자릿수에 따라 시간이 기하급수적으로 증가하는 것을 알 수 있다. 이를 일반화하면 비밀번호를 생성하는데 사용되는 문자를 k개라고 할 때, 이를 조합하여 N 자릿수 비밀번호를 생성한다고 하면 모든 문자열을 비교하게 되었을 때 $O(k^N)$의 시간이 소요된다.

[1] https://ko.wikipedia.org/wiki/무차별_대입_공격

실습을 진행하면서 랜덤으로 생성한 비밀번호에 대해 매칭된 시간을 기록하여 시간 복잡도와 유사한 형태인지 분석을 진행하고자 한다.

3. 문제 해결 설계

실습을 진행하기 위해 알고리즘 설계를 다음 <그림1>과 같이 진행하였고, 이를 기반으로 직접 구현을 하였다.



<그림 1. Brute Force Attack 구현 고려 사항>

2 본 론

2.1 실습 : Brute Force Attack

다음 소스코드는 핵심 구현 내용을 정리하기 위해 일부 발췌하였다.
전체 소스코드 및 주석(함수 설명)은 **4. APPENDIX**에 첨부하였다.

1. gen_pw.py
   목적: 패스워드 유형과 길이에 따라 랜덤으로 생성

```
11. def pw_type(type, length):
12.     passwords = []
13.     if type == 0: #숫자 조합
14.         for _ in range(10): #10 개씩 생성하기 위함
15.             password = ''.join(random.sample(list1, length)) #랜덤 조합을 통해 패스워드
    길이의 문자열 생성
16.             passwords.append(password) #생성된 문자열을 리스트에 저장
```

<코드 1. gen_pw.py의 패스워드 랜덤 생성 구현 부분>

pw_type() 함수에서 패스워드의 유형(총 7가지)을 패스워드 길이(4~8자리)마다 10개씩 생성하도록 구현하였다. <코드1>은 유형 1~3인 단일 리스트의 조합으로 생성하는 경우의 코드이다.

```
28. if type == 3: #숫자+알파벳 조합
29.         count = 0
30.         while(count <= 9): #10 개씩 생성하기 위함
31.             password = "".join(random.choice(list1 + list2) for _ in range(length))
#조합된 유형(숫자+알파벳)으로 랜덤 생성
32.             if all(any(word in password for word in lst) for lst in (list1, list2)):
#그 중 모든 리스트에서 적어도 문자 1 개 이상 포함된 문자열을 선택하여
33.                 passwords.append(password) #리스트에 저장
34.                 count += 1
```

<코드 2. gen_pw.py의 패스워드 랜덤 생성 중 여러 리스트 조합 부분>

조합을 통해 랜덤으로 패스워드를 생성 시 조합에 사용된 리스트의 원소가 포함되지 않은 경우가 발생할 수 있다. 따라서 랜덤으로 생성된 문자열 중 조합에 사용된 리스트의 원소가 반드시 1개 이상 포함된 경우만 password 리스트에 포함되도록 구현하였다. 위 코드에서는 숫자, 알파벳 조합으로 32번 라인에서 list1(숫자)과 list2(알파벳)의 원소가 1개 이상 포함된 경우를 조건문을 통해 검증한 뒤 리스트에 저장하도록 구현하였다.

2. Brute_force.py

목적: 모든 조합을 생성하여 Password Cracking 진행

```
# 모든 경우의 수를 생성하여 pw_list 와 매칭
def brute_force(answer_list):
    count = 0
    timer = []
    #패스워드 매칭까지 걸린 시간 측정
    start = time.process_time()
    for pw_len in range(4, 8):
        #
        for password in itertools.product(pw_list, repeat=pw_len):
                # brute force 로 맞출 문자열 생성
                password = "".join(password)
                # 생성된 패스워드가 매칭될 경우
                if password in answer_list:
                    #매칭된 시간 측정
                    end = time.process_time()
                    #찾은 비밀번호 출력
                    print(f"비밀번호 {password} 찾았습니다.")
                    #걸린 시간 및 남은 패스워드 개수 출력
                    print(f'{end-start} 진행도 : {count+1}/{len(answer_list)}')
                    timer.append(end-start)
                    print("Loading .. ")
                    count += 1
                #모든 리스트와 매칭될 경우 종료
                if count == len(answer_list):
                    print(timer)
                    exit()
```

<코드 3. Brute_force.py에서 모든 조합을 생성 및 매칭하는 부분>

<코드 3>은 파이썬의 itertools.product 모듈을 이용해 pw_list인 숫자,알파벳,특수문자를 가지고 길이 및 유형에 맞게 모든 경우의 문자열을 생성한다. 코드 사이에 time() 모듈을 통해 매칭된 시간을 측정하며, 매칭된 경우 매칭된 패스워드와 시간, 남은 패스워드의 개수를 출력하도록 구현하였다.

3. main.py

목적: test_pw가 있는 txt파일을 가져와 brute force attack 실행

```
#동일한 폴더 안에 있는 함수 가져오기
import gen_pw, Brute_force
# 패스워드 10 개씩 랜덤으로 생성하는 함수 실행
def run_generation():
    gen_pw.run()
#brute force 진행하는 함수
def run_brute_force(pw_length):
    print('Loading .. ')
    Brute_force.run(pw_length)
#문자열의 길이에 해당하는 txt 를 가져와 brute force 를 진행한다.
run_brute_force(4)
```

<코드 4. main.py에서 작성한 함수를 실행하는 코드>

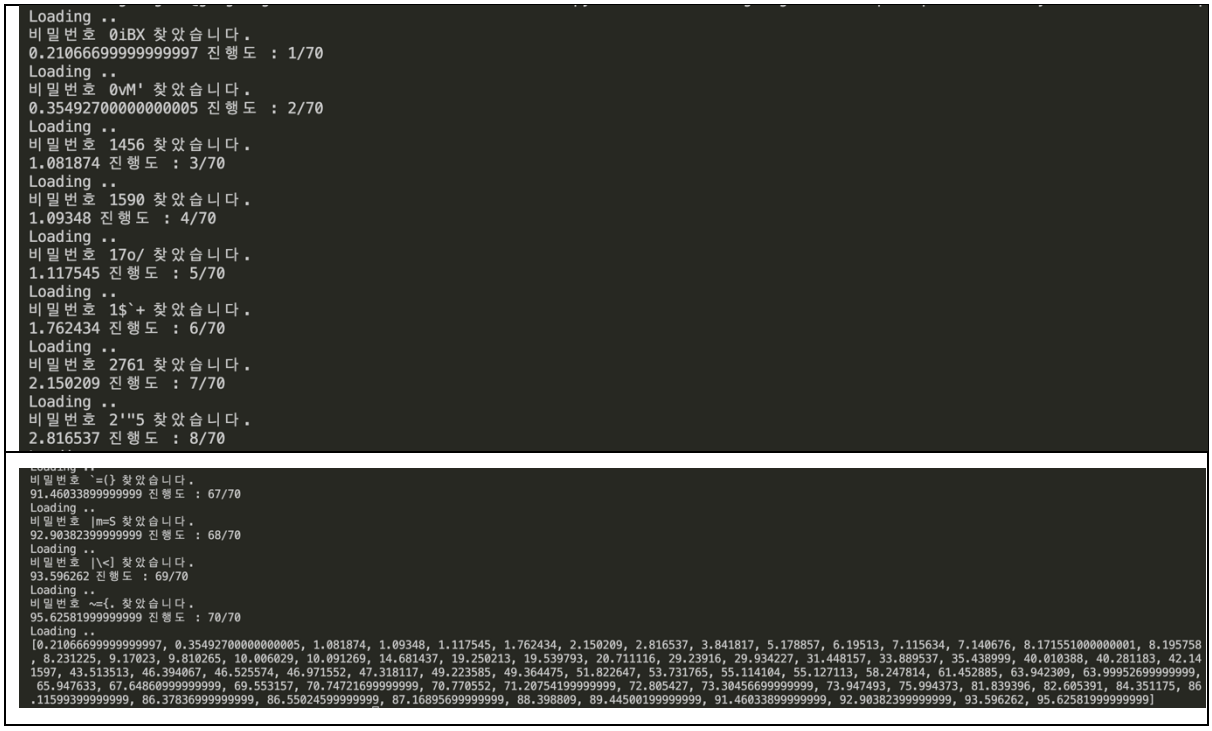
1,2에서 작성한 함수의 결과를 확인할 수 있도록 main 파일을 구성하였다.

3 본 론

3.1 실습 결과

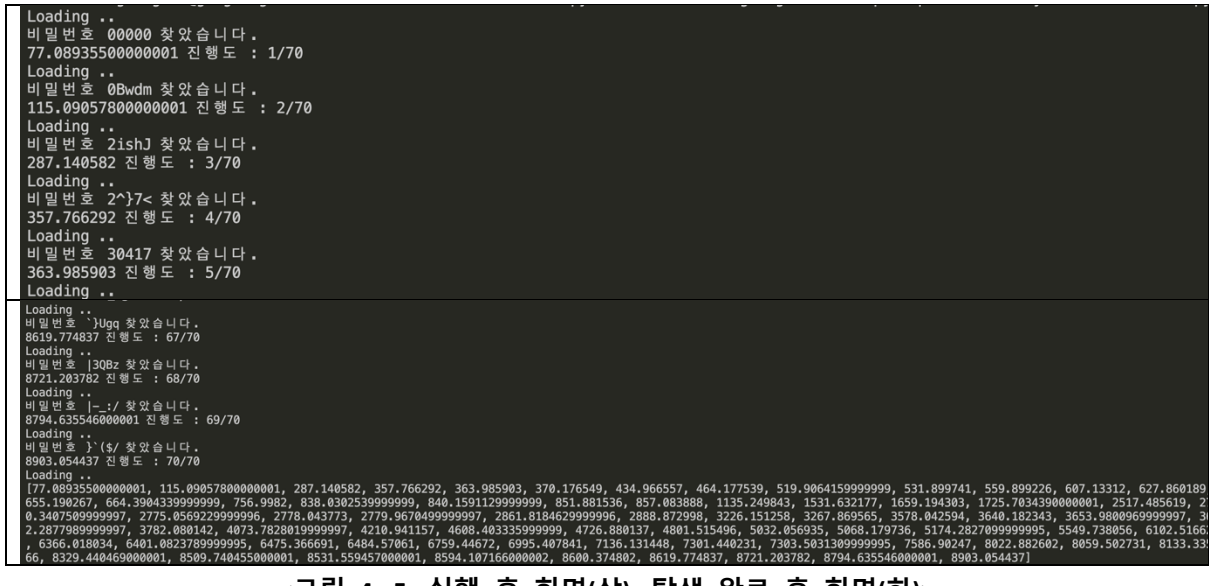
1. 실습 결과 화면

1) 4자리



<그림 2, 3. 실행 후 화면(상), 탐색 완료 후 화면(하)>

2) 5자리



<그림 4, 5. 실행 후 화면(상), 탐색 완료 후 화면(하)>

3) 6,7,8자리

기한 내에 모든 결과를 확인하지 못했다.

2. 랜덤 패스워드 생성

먼저, 문자열 유형과 길이에 따른 패스워드 생성에 대한 결과를 확인했다. txt(텍스트 파일)에 길이별로 자동으로 저장하여, 엑셀 파일을 통해 리스트를 분할하여 저장하였다. 2.1.1 gen_pw.py에서 언급했듯이 리스트의 조합을 통해 랜덤으로 생성할 경우 모든 리스트의 원소가 포함되지 않은 경우가 발생할 수 있었지만, 조건문을 추가하여 적어도 1개 이상의 원소를 포함할 경우에만 저장하도록 구현하여 다음과 같은 패스워드를 생성할 수 있었다.

1) 패스워드 길이가 4자리인 경우

4자리	type1	type2	type3	type4	type5	type6	type7
	숫자	알파벳(대,소)	특수문자	숫자+알파벳	숫자+특수기호	알파벳+특수기호	숫자+알파벳+특수기호
1	8540	Shgk	=!`?	8a68	"3.5	Q,Y'	₩8BA
2	7061	eJHR	|₩<]	JS7U	3(?|	["yb	Sg1{
3	7231	kARz	%(`_	0iBX	2'"5	Vt"%	K17,
4	1590	yOYE	(=>[	JG7b	*?/5	>A~`	.0Az
5	2761	MoVP	~={.	MBx6	8%₩	+zF-	9₩MH
6	1456	YOeN	^>:)	Dct1	₩#-1	DA.J	jn1#
7	5730	uTxP	[]!@	9YBx	1$`+	O=^B	17o/
8	8796	sJll	]?!|	GUa1	@581	|m=S	9:]n
9	6830	tcfx	`={}	Fo0t	,1]<	)rem	0vM'
10	9342	Kxhp	(;`~	x0FR	$4$3	'ZO/	i{V8

<그림 6. 4자리 패스워드 무작위 생성 결과(PW4.txt)>

2) 패스워드 길이가 5자리인 경우

5자리	type1	type2	type3	type4	type5	type6	type7
	숫자	알파벳(대,소)	특수문자	숫자+알파벳	숫자+특수기호	알파벳+특수기호	숫자+알파벳+특수기호
1	76032	snvOd	(_"|)	t96HF	%{?{9	`'be;	w?'0p
2	60139	APfCZ	|-_:/	Nl2SA	.<~?6	U~us%	5L"xC
3	43659	tzclZ	!?-._	Q06HF	5&![?	C!Q₩%	`-9Z(
4	00000	MGufn	[>('<	Bh8x7	4'.,!	+MftJ	PX8V!
5	36075	hqkiG	%>(:/	BMg4a	2^}7<	$%/Tq	|3QBz
6	89250	xtQkj	@".=$	2ishJ	3':3>	;^+Z!	-40l'
7	52318	BulEn	]~@/-	b0r6h	7])!?	Ha-{^	s9l"E
8	30417	pTSse	.:]{_	4Vhap	%7~2&	gS+Xf	[77]h
9	69431	sqkwg	_>/%!	0Bwdm	7[!:(	F#~SP	R9*Y*
10	84307	Lltbm	}`($/	fok6O	`8:,3	`}Ugq	ss9;0

<그림 7. 5자리 패스워드 무작위 생성 결과(PW5.txt)>

3) 패스워드 길이 6자리인 경우

6자리	type1	type2	type3	type4	type5	type6	type7
	숫자	알파벳(대,소)	특수문자	숫자+알파벳	숫자+특수기호	알파벳+특수기호	숫자+알파벳+특수기호
1	520619	OXTLfM	@,^"]}	2ndehq	}20_59	ekY)|q	w1G₩%;
2	072185	fJyEDb	-;)&^.	jtA8Y3	[`1=_9	E|T]G<	3X@md`
3	086327	KOFkBA	&{'^"!	b24Dnh	}[:1)^	HoiE-R	E{w+H0
4	652381	FMclTA	@₩;'"|	v2b00K	%#8-]3	h)('LH	sn*8?
5	610324	rbqVic	%-(/):	hVG97Z	?$~9:/	#XUKE=	yA%cO3
6	348052	etkTiR	_<!.@]	AX23d5	[!68[^	L*umMg	P_nLx5
7	548230	rbUVPj	;$>("!	tcm0uj	6)}&!^	!esaRQ	I^8q+&
8	376401	CUtDkJ	$}#@~(	zmFJ7A	5.&)&%	K!laoq	B/Fb7
9	291730	OGoafN	@₩*),<	QJ1q35	?1@-%*	{az`ho	[5*(R#
10	296157	NYWjhG	=^_-}<	e9CJbQ	~.91*]	xw]Gg!	5US$U.

<그림 8. 6자리 패스워드 무작위 생성 결과(PW6.txt)>

4) 패스워드 길이 7자리인 경우

7자리	type1	type2	type3	type4	type5	type6	type7
	숫자	알파벳(대,소)	특수문자	숫자+알파벳	숫자+특수기호	알파벳+특수기호	숫자+알파벳+특수기호
1	3809617	FqghCJI	,(~)_/=	kTH045f	`=>(`0#	N=Q₩_S	{M)3Vaw
2	6784539	ivenxlz	&,+-)_|	q8txR99	^>]07%_	}J.}+}d	lY[5O^i
3	5837492	KJNsvYl	₩!}).{,	jjlGE4n	{;8|]+}	}:btwzd	4O/UBz`
4	3019547	tsJAyhl	,₩!_?#=	nwv6UXc	!!(2>&,	ZC^V^Zb	1vL^n0=
5	5942061	UcjyQBD	-~)%_(!	gk7Ggv1	75*;`:/	>QcX.^v	ld8-p0W
6	6423918	wjsRreZ	{-`_*%~	Apdgu1n	@]-"^4"	:aW#?Pg	7₩kVdrH
7	2837650	OrQeUWn	[*"#-:,	CvE1kir	35+>^&*	OQq+k₩r	?|-8z*1
8	9741658	dOzKiAa	,(:%=-$	sTytH8T	,5:"%3_	=yOJpWx	#2cy1t#
9	2631498	EPywqeN	*,;("&_	a1LrEiH	>0"#=$!	%s#{.*~	j*!31DC
10	5960317	Hvmykde	$[;(_<:	c0PYs27	**+!+0-	j%e{w.l	b9m$z`0

<그림 9. 7자리 패스워드 무작위 생성 결과(PW7.txt)>

5) 패스워드 길이 8자리인 경우

8자리	type1	type2	type3	type4	type5	type6	type7
	숫자	알파벳(대,소)	특수문자	숫자+알파벳	숫자+특수기호	알파벳+특수기호	숫자+알파벳+특수기호
1	04968175	bahlgcqK	*'<-#>+;	z7dwbV1N	,4-`,#.}	:Es/GBX)	zZ1_&M:t
2	57430269	aPJMseCF	]&#~)*<$	e5G5Kb1i	(4%~8-+.	zFRq[?Mx	b!₩W:NG5
3	09756123	xqBRTsFY	=/)#<]}	oUbzEY4C	/.<.;01`	bZyfGA~G	=fPgcu0s
4	73859604	fZplrdNh	>!}^,@]#	eqrgbl8p	^@`9%(63	[:*[|?{L	7:c-2XJ)
5	06453217	pXewRYUg	[,*₩!(|#	Uf1S1hlq	2+(="7(2	W$LOMWx,	Z8iCAl}R
6	13509864	gwWKxVZe	#.,*)-"`	nGqHeQR8	$&(4@~&@	shp=oPFl	l7w-Vu7n
7	01245738	fkVZrNSG	{,@<$|/`	8yjKcncw	8`,;47,$	~Gy&|Yr₩	EWv,₩6yk
8	98064275	woSMWRDs	#'=_<:]$	bL4xb83K	90_[2_2"	t<%Vnvwb	!>0_.@Zw
9	84103675	YBfPlLQS	₩^-?,@*:	VWuK3ah5	:-@<~[4|	O>`(kQDF	/Y9lVyXD
10	87593640	WRABkraP	;'/.=)$|	V6Yn3WQ9	#}:0]?$&	hz_PWfGF	L;(m8f4L

<그림 10. 8자리 패스워드 무작위 생성 결과(PW8.txt)>

3. Brute Force Attack 결과

1) 패스워드 길이 4자리인 경우

4자리 패스워드 탐색 소요 시간										
0	0.210667	0.354927	1.081874	1.09348	1.117545	1.762434	2.150209	2.816537	3.841817	5.178857
10	6.19513	7.115634	7.140676	8.171551	8.195758	8.231225	9.17023	9.810265	10.006029	10.091269
20	14.681437	19.250213	19.539793	20.711116	29.23916	29.934227	31.448157	33.889537	35.438999	40.010388
30	40.281183	42.141597	43.513513	46.394067	46.525574	46.971552	47.318117	49.223585	49.364475	51.822647
40	53.731765	55.114104	55.127113	58.247814	61.452885	63.942309	63.999527	65.947633	67.64861	69.553157
50	70.747217	70.770552	71.207542	72.805427	73.304567	73.947493	75.994373	81.839396	82.605391	84.351175
60	86.115994	86.37837	86.550246	87.168957	88.398809	89.445002	91.460339	92.903824	93.596262	95.62582

<그림 11. 4자리 패스워드 탐색 소요 시간(초)>

7가지 유형의 총 70개의 패스워드를 탐색하는 데 약 1분 35초 정도 소요되었다.

2) 패스워드 길이 5자리인 경우

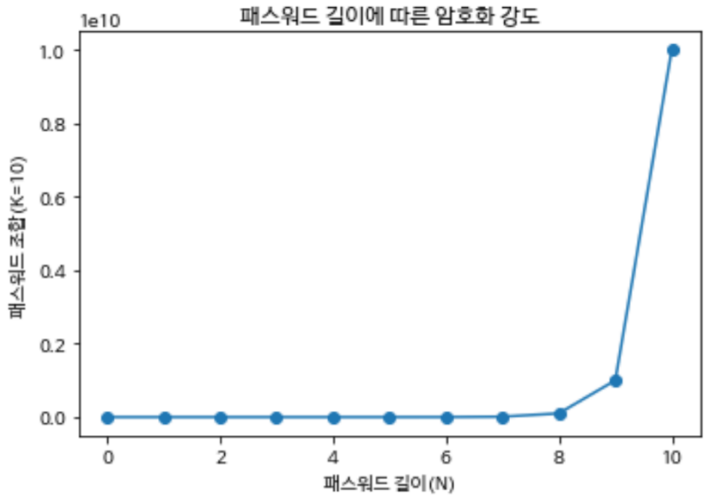
5자리 패스워드 탐색 소요 시간										
0	77	115	287	357	363	370	434	464	519	531
10	559	607	627	655	664	756	838	840	851	857
20	1135	1531	1659	1725	2517	2770	2775	2778	2779	2816
30	2888	3226	3267	3578	3640	3653	3702	3782	4073	4210
40	4608	4726	4801	5032	5068	5174	5549	6102	6366	6401
50	6475	6484	6759	6995	7136	7301	7303	7586	8022	8059
60	8133	8329	8509	8531	8594	8600	8619	8721	8794	8903

<그림 12. 5자리 패스워드 탐색 소요 시간(초)>

7가지 유형의 총 70개 패스워드를 탐색하는 데 약 148분(2시간 28분) 정도 소요되었다.

3.2 고찰(Discussion)

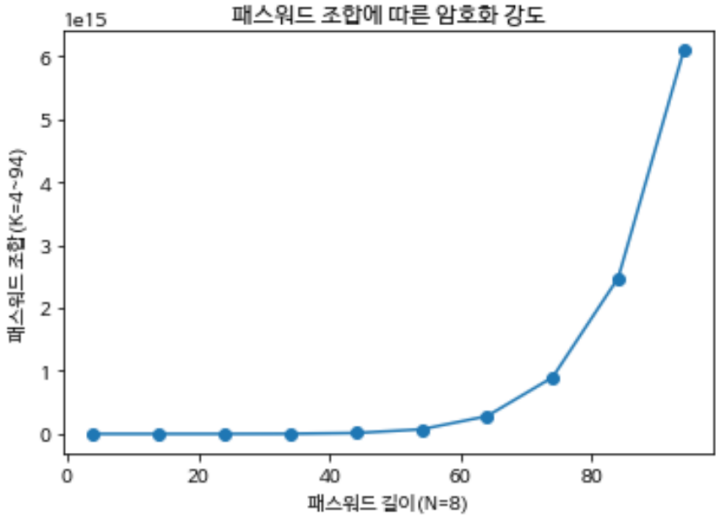
1) 패스워드 길이에 따른 암호화 강도 분석



<그래프 1. 패스워드 길이에 따른 Time Complexity>

패스워드의 길이 N, 패스워드 조합 개수 k라 하면, 이론적 시간 복잡도는 $O(k^N)$ 이다. 패스워드의 길이에 따른 시간 복잡도를 확인하기 위해 패스워드 조합의 개수(k)는 10으로 고정하고, 패스워드의 길이(N)을 0부터 10자리로 변수로 두어 그래프를 그려 확인해보았다. 우상향 그래프로 패스워드의 자리수가 8자리부터 급진적으로 증가하는 형상을 확인할 수 있어, 시간 복잡도 측면에서 패스워드의 암호화 강도를 높이고자 한다면 8자리 이상으로 설정하는 것이 바람직하다고 볼 수 있다.

2) 패스워드 조합 유형에 따른 암호화 강도 분석



<그래프 2. 패스워드 조합 유형에 따른 Time Complexity>

패스워드 조합 유형에 따른 시간 복잡도를 확인하기 위해 패스워드의 길이(N)는 8자리로 고정하고, 패스워드의 조합을 4부터 실습에서 사용한 조합의 개수인 94개까지 변수로 두어 그래프를 그려보았다. 조합에 따른 강도를 확인해보면 약 60개부터 시간 복잡도가 급격하게 증가하는 모습을 확인할 수 있고, 암호화의 강도를 향상시키고자 한다면, 60개의 조합을 통해, 실습에서 사용한 조합으로는 숫자+알파벳, 알파벳+특수문자, 숫자+알파벳+특수문자의 조합으로 패스워드를 구성하는 것이 바람직하다고 분석할 수 있다.

3.3 느낀 점

지난 실습1의 고찰에서 간략하게 패스워드의 강도를 Brute Force로 찾은 경우의 그래프를 통해 시간 복잡도를 확인하였다. 이번 실습을 통해 실제로 패스워드를 다양한 조건속에서 생성하여 Brute Force Attack 코드를 구현해보았다. 가장 아쉬웠던 점은 근래 파이썬만 사용하다보니 연산 속도 측면에서 보다 빠른 속도로 결과를 확인할 수 있는 C++, JAVA를 사용하지 못해 모든 조합, 길이에 따른 결과를 확인하지 못해 아쉬웠다. 파이썬의 경우에도 함수를 조금 더 간결하게 리팩토링을 진행하였다면 확인할 수 있었을 것 같은 아쉬움도 있다. 하지만 지난 실습에서 여러가지 케이스별로 확인을 추가적으로 이번 실습에서 진행하여 알아보고자 했던 패스워드 길이 별 암호화 강도, 패스워드 조합에 따른 암호화 강도를 확인하여 패스워드의 길이는 8자리 이상, 패스워드 조합은 알파벳+숫자, 알파벳+특수문자, 숫자+알파벳+특수문자의 조합(60개 이상의 조합)으로 생성할 경우 암호화 강도가 높아진다는 것을 확인할 수 있었다.

4 APPENDIX

4.1 소스코드

〈소스코드1. gen_pw.py〉

```
import string
import random
#string 모듈을 통해 숫자, 알파벳(대&소), 특수문자를 리스트로 생성
list1 = string.digits
list2 = string.ascii_letters
list3 = string.punctuation
#패스워드 길이마다 유형별로 조합하여 랜덤으로 패스워드를 생성하는 함수
def pw_type(type, length):
    passwords = []
    if type == 0: #숫자 조합
        for _ in range(10): #10 개씩 생성하기 위함
            password = ''.join(random.sample(list1, length)) #랜덤 조합을 통해 패스워드
길이의 문자열 생성
            passwords.append(password) #생성된 문자열을 리스트에 저장
    if type == 1: #알파벳 조합
        for _ in range(10): #10 개씩 생성하기 위함
            password = ''.join(random.sample(list2, length)) #랜덤 조합을 통해 패스워드
길이의 문자열 생성
            passwords.append(password) #생성된 문자열을 리스트에 저장
    if type == 2: #특수문자 조합
        for _ in range(10): #10 개씩 생성하기 위함
            password = ''.join(random.sample(list3, length)) #랜덤 조합을 통해 패스워드
길이의 문자열 생성
            passwords.append(password) #생성된 문자열을 리스트에 저장
    if type == 3: #숫자+알파벳 조합
        count = 0
        while(count <= 9): #10 개씩 생성하기 위함
            password = "".join(random.choice(list1 + list2) for _ in range(length))
#조합된 유형(숫자+알파벳)으로 랜덤 생성
            if all(any(word in password for word in lst) for lst in (list1, list2)): #그
중 모든 리스트에서 적어도 문자 1 개 이상 포함된 문자열을 선택하여
                passwords.append(password) #리스트에 저장
                count += 1
    if type == 4: #숫자+특수문자 조합
        count = 0
        while(count <= 9): #10 개씩 생성하기 위함
            password = "".join(random.choice(list1 + list3) for _ in range(length)) #유형
3 과 동일
            if all(any(word in password for word in lst) for lst in (list1, list3)):
                passwords.append(password)
                count += 1

    if type == 5: #알파벳+특수문자 조합
        count = 0
```

```
        while(count <= 9): #10 개씩 생성하기 위함
            password = "".join(random.choice(list2 + list3) for _ in range(length)) #유형
3 과 동일
            if all(any(word in password for word in lst) for lst in (list2, list3)):
                passwords.append(password)
                count += 1
    if type == 6: #숫자+알파벳+특수문자 조합
        count = 0
        while(count <= 9): #10 개씩 생성하기 위함
            password = "".join(random.choice(list1 + list2 + list3) for _ in
range(length)) #유형 3 과 동일
            if all(any(word in password for word in lst) for lst in (list1, list2,
list3)):
                passwords.append(password)
                count += 1
    return passwords
def run():
    for length in range(4, 9): #4 자리~8 자리의 패스워드를 생성한다.
        file = f'/Users/kangdonghee/Desktop/Computer Security/실습/week2/PW{length}.txt'
#미리 생성한 자릿수별로 구분된 파일을 불러온다.
        with open(file, "w+") as pw_file: #파일을 열어 write 한다.
            for type in range(0, 7): #모든 유형별로 생성
                pw_file.write('\n'.join(pw_type(type, length))) #생성된 패스워드를 파일에
작성한다.
                pw_file.write('\n')
        pw_file.close() #파일 닫기
        passwords = [] #길이 패스워드에 대해 초기화
    return print('Generation Done.') #생성 종료 문구 출력
```

〈소스코드2. Brute_force.py〉

```
import time
import itertools
import string
# 가능한 문자들의 리스트 (숫자, 알파벳, 특수문자)
pw_list = string.digits+string.ascii_letters+string.punctuation
# 모든 경우의 수를 생성하여 pw_list 와 매칭
def brute_force(answer_list):
    count = 0
    timer = []
    #패스워드 매칭까지 걸린 시간 측정
    start = time.process_time()
    for pw_len in range(4, 8):
        #
        for password in itertools.product(pw_list, repeat=pw_len):
                # brute force 로 맞출 문자열 생성
                password = "".join(password)
                # 생성된 패스워드가 매칭될 경우
                if password in answer_list:
                    #매칭된 시간 측정
                    end = time.process_time()
                    #찾은 비밀번호 출력
                    print(f"비밀번호 {password} 찾았습니다.")
                    #걸린 시간 및 남은 패스워드 개수 출력
                    print(f'{end-start} 진행도 : {count+1}/{len(answer_list)}')
                    timer.append(end-start)
                    print("Loading .. ")
                    count += 1
                #모든 리스트와 매칭될 경우 종료
                if count == len(answer_list):
                    print(timer)
                    exit()

def import_test_pw(length):
    #정답 리스트를 가져온다
    test_pw_file = f'/Users/kangdonghee/Desktop/Computer
Security/실습/week2/PW{length}.txt'
    with open(test_pw_file) as f:
        test_pw = f.readlines()
    #라인 하나씩 가져와 리스트에 저장
    test_pw = [line.rstrip('\n') for line in test_pw]
    return test_pw
#패스워드 탐색 시작 함수
def run(pw_length):
    test_pw = import_test_pw(pw_length)
    brute_force(test_pw)
    return print('Done.')
```

〈소스코드3. main.py〉

```
#동일한 폴더 안에 있는 함수 가져오기
import gen_pw, Brute_force
# 패스워드 10 개씩 랜덤으로 생성하는 함수 실행
def run_generation():
    gen_pw.run()
#brute force 진행하는 함수
def run_brute_force(pw_length):
    print('Loading .. ')
    Brute_force.run(pw_length)
#문자열의 길이에 해당하는 txt 를 가져와 brute force 를 진행한다.
run_brute_force(4)
#run_brute_force(5)
#run_brute_force(6)
#run_brute_force(7)
#run_brute_force(8)
```

〈소스코드 4. 고찰(패스워드 길이에 따른 암호화 강도 분석)〉

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 10, 11)
# [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
y = 10**x
# [1.e+00 1.e+01 1.e+02 1.e+03 1.e+04 1.e+05 1.e+06 1.e+07 1.e+08 1.e+09 1.e+10]

plt.plot(x, y)
plt.xlabel("패스워드 길이(N)")
plt.ylabel("패스워드 조합(K=10)")
plt.title("패스워드 길이에 따른 암호화 강도")
plt.scatter(x,y)
```

〈소스코드 5. 고찰(패스워드 조합 개수에 따른 암호화 강도 분석)〉

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(4, 94, 10) #패스워드의 길이
y = x**8
plt.plot(x, y)
plt.xlabel("패스워드 길이(N=8)")
plt.ylabel("패스워드 조합(K=4~94)")
plt.title("패스워드 조합에 따른 암호화 강도")
plt.scatter(x,y)
```