

팀명 : GAZUAAA

팀원 :

14012848 최재현 (조장)

15012977 고창훈

15012978 서성관

15012981 이소영

16011783 박근진



세종대학교
SEJONG UNIVERSITY

리버스 엔지니어링 Team Mission

Mission3_RE_PasswordManager

1. 구현 룰을 따르면서, 리버싱을 어렵게 하기 위해 작성한 코드를 구현동기와 함께 설명하시오. (Min 3 items)

1-A. 코드 분석

사용자가 입력하는 평문을 암호문으로 바꾸는 코드가 동작하는 개괄적인 단계는 다음과 같다.

Step 1) 입력되는 패스워드 데이터에 대해 마스킹 처리한다.

Step 2) 문자열이 숫자일때 인덱스+3 하고 , 문자일때 인덱스+10 한다

Step 3) 두개의 문자열을 입력받은 뒤 문자열의 아스키코드를 정수로 받아 두개의 배열의 덧셈과 곱셈을 조합하여 하나의 배열을 생성하고 문자형으로 변환한다.

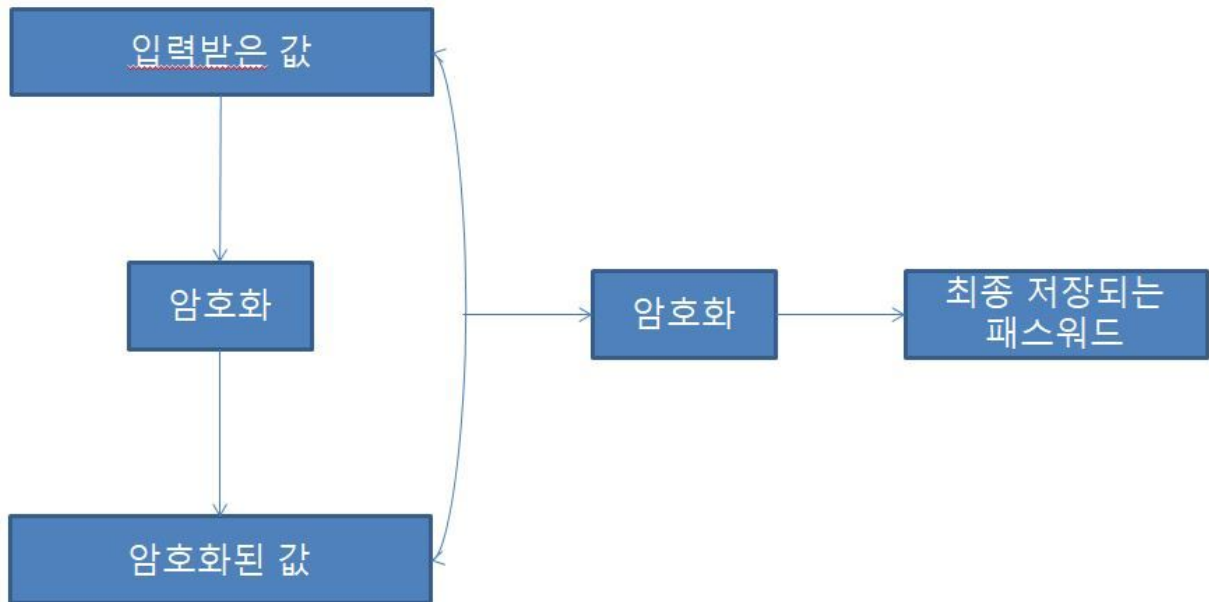
Step 4) 위에서 암호화 된 문자열들을 비트연산, 분기 및 루프를 이용해 섞어준다.

Step 5) 사용자가 입력한 패스워드의 유효성을 판단할 때는 위와 동일한 과정을 거쳐 패스워드를 암호화해, 나온 출력값을 password.txt에 저장된 암호문과 비교하여 유효성을 검사한다.

위의 과정에서 설명했듯이, 일반 평문을 사용자가 입력할 경우 숫자, 문자에 관한 기본적인 Encryption 과정으로서 index shift 과정을 거쳐 Basic 한 암호문을 생성한다.(1차적 암호화) 그 후 기존에 있던 사용자의 입력값과 Encryption 과정을 거친 암호문(key 역할) 을 덧셈과 곱셈으로 조합하여 하나의 배열로 만든다. (2차적 암호화)

마지막으로 비트연산과 분기및 루프를 이용하여 나열되어 있는 배열의 순서를 다시 섞어준다.(3차적 암호화)

이러한 구조를 통해 Decompile 과정을 거치고 알고리즘의 구조를 완전히 파악한다 하더라도 역함수를 찾기 어렵게 만들었고, 역함수를 찾는다 하더라도 key value로서 사용된 암호문을 다시 복호화 시키는 것이 어렵게 만들었다.



[그림] Step 3 과정 도식화

```

password.txt - 텍스트
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
958595989985786890099068006875575099570090776078999889550856688995577805566678588908599980779999888687079959059998678908907898689098986
90005987706590695856566660879905909659856070909708887060876560590075700659607598085578857886586089068555678979896869709099957885688780
399969797576787068995987080009880869895598576068665787590708989988557889059659656900908658876788860678895986777988077577065897979989867
095705568970685789560996508778099058970007078088705839663775197916863070256627554605378647961875390017762965477749562067266646964569398
520994885189040754879157749072995196545593800470915703576270529671985379626064877480628672650365939994875300927894895378639674685389749
791555270935881988470045502880466610002707409939703895199516752077276926873806155048663596489815563550357846663787290838093067306525062
775360930093907270038994902216422924202226414042254236134744193219413521263249311933204328411921304436233012194246221611172148324832294
236123834151449331631494129443824152330113034352148142822101325134012481420343844154345223732391315433532303230442714374116413843384135
441542192329224932203317212542182228213523153420432931293320341613463146211811271328131614253338412044471448443641262337432821192447243
023372417413542303440343924404446142843102319231842302129442533402449232032184219243843293339222923473418343841202315434812173446424943
392125311642281319341612173229131933201116214941401140133541163446424922352236242023494315243663897188726654989359940094658397919974067
290728072759166810661970495840872568295840654669200946953986408025853988487619671887208929761787488947901996255835882509389619962089296
919584775298840082808160810083897467047904779366710064659177838972875300839552669190946063697479836882579186829053790450026952066295645
564908477037552957157510704795485526554565106019793777255615771597490026504795290749761968286818691895295615871006370937894600179648081
1795388845681970277720593587498647692550490936601976298728092859258935951979178028891700267540752805279828673086306739604007465528683956
280826854907405947681000260736002869307919782508307935853957390928992968357688974688206710601767295619972785277026083069350037884775385
8400930861095276629864097405518502996179617091708368716882707285918573687265919074908108635072976160617864957298892590277122722102126421
644263249433614103127113742452138133023101227212931281438112622384248422921182235313834464326441532264229332622283310314541102329332633
392429331623452427344541252147312734301437113614481440421914174436411643491316411024271128324523172226414534274448442911183336141922393
327223644291248341742174130114032254217344931281448133643152230241843172226241844482117234924473416131541272147342532492428342844494447
114521273328423943163316433822184346243721302149141543163428212024271148214831292148332732181427421521351220441914483430211931181317423
5214631404127221822194419224841361427334733184340
  
```

[그림] default password 의 Password.txt

1-B. 알고리즘의 구현 동기

패스워드를 암호화 해 저장하면, 암호화 알고리즘의 역함수가 존재 한다면, 역공학자가 결국에는 패스워드를 평문으로 계산할 수 있다고 생각하였고, 이를 막기 위해 패스워드를 암호화 키로 이용하고자 하였습니다. 하지만 패스워드를 키로 이용해 코드 내에 공개된 평문을 암호화하였을때, 평문과 암호문이 공개된 상태에서 역함수를 이용하여 역공학을 진행한다면 키를 유추할 수 있을 것이고, 이는 처음에 저희가 해결하고자했던 문제점을 해결하지 못하였습니다. 따라서 저희는 평문과 키 값이 모두 사용자 입력한 패스워드에 종속되게끔 알고리즘을 작성하였고, 패스워드의 길이 등 패스워드에 대한 정보가 없는 역공학자의 입장에서 암호화 알고리즘을 역을 계산했다고 하더라도, 패스워드를 쉽게 계산할 수 없도록 하였습니다.

void case2(char *ch)

```
void case2(char *ch) {
    int **arr = (int**)malloc((sizeof(int*) * 1000));
    for (int i = 0; i < strlen(ch); i++) {
        arr[i] = (int*)malloc((sizeof(int) * 1000));
    }
    int cnt = 0, m, n;
    int chcnt = 0;
    int idx;
    char tmpc, chb;
    n = sqrt(strlen(ch)) + 1;
    m = sqrt(strlen(ch)) + 2;
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < i + 1; j++) {
            if (j < n) {
                arr[j][i - j] = ++cnt;
            }
        }
    }

    for (int i = 1; i < n; i++) {
        for (int j = 0; j < n - i; j++) {
            if (j < m)
                arr[i + j][m - 1 - j] = ++cnt;
        }
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (arr[i][j] - 1 < strlen(ch)) {
                if (chcnt == 0) {
                    idx = arr[i][j] - 1;
                    chb = ch[arr[i][j] - 1];
                    chcnt++;
                }
                else {
                    tmpc = ch[arr[i][j] - 1];
                    ch[arr[i][j] - 1] = chb;
                    chb = tmpc;
                }
            }
        }
    }
    ch[idx] = chb;
}
```

case2함수는 문자열을 섞기 위해 만들었고, 2차원 배열을 이용하여 대각선으로 배열에 정수를 담고 1행 부터 순서대로 배열에 담겨있는 값이 문자열의 길이보다 작다면 그 값대로 문자열을 섞는다. 다음에 사용되는 case2함수들의 기본 틀로 사용 했다.

void case3(char *ch)

```
void case3(char *ch) {
    int cnt = 0;
    int cnt1, cnt2;
    int **a = (int**)malloc(sizeof(int*) * 1000);
    for (int i = 0; i < strlen(ch); i++) {
        a[i] = (int*)malloc(sizeof(int) * 1000);
    }

    int n, m;
    int i = 0, j = -1, aa = -1;
    int m1, n1, idx;
    int chcnt = 0;
    char tmpc, chb;

    m = sqrt(strlen(ch)) + 1;
    n = sqrt(strlen(ch));
    m1 = m;
    n1 = n;
    cnt1 = 0, cnt2 = 0;

    while (1) {
        aa *= -1;
        while (cnt1 < m) {
            j += aa;
            a[i][j] = ++cnt1;
            cnt1++;
        }
        cnt1 = 0;

        cnt--;
        while (cnt2 < n) {
            a[i][j] = ++cnt2;
            i += aa;
            cnt2++;
        }

        i -= aa;
        cnt2 = 0;
        n--;
        m--;
        if (cnt == n1 * m1)
            break;
    }

    for (int i = 0; i < n1; i++) {
        for (int j = 0; j < m1; j++) {
            if (a[i][j] - 1 < strlen(ch)) {
                if (chcnt == 0) {
                    idx = a[i][j] - 1;
                    chb = ch[a[i][j] - 1];
                    chcnt++;
                }
                else {
                    tmpc = ch[a[i][j] - 1];
                    ch[a[i][j] - 1] = chb;
                    chb = tmpc;
                }
            }
        }
        ch[idx] = chb;
    }
}
```

case3 함수도 문자열을 섞기 위해 만들었다. 2차원 배열을 이용하여 달팽이 모양으로 배열에 정수를 담고 1행 부터 순서대로 배열에 담겨있는 값이 문자열의 길이보다 작다면 그 값대로 문자열을 섞는다. 다음에 사용되는 case3함수들의 기본 틀로 사용 했다.

void composition(char *ch)

```
void composition(char *ch) {  
    while (i > -1) {  
        if (ch[i] >= 'a' && ch[i] <= 'v') {  
            if (ch[len - i - 1] >= '0' && ch[len - i - 1] <= '9') {  
                tmp = ch[i];  
                ch[i] = ch[len - i - 1];  
                ch[len - i - 1] = tmp;  
            }  
        }  
        i--;  
    }  
  
    for (i = 0; i < len; i++) {  
        for (j = len - 1; j >= 0; j--) {  
            tmp = ch[i];  
            ch[i] = ch[j];  
            ch[j] = tmp;  
        }  
    }  
  
    for (i = 0; i < len; i++) {  
        for (j = len - 1; j >= 0; j--) {  
            if (ch[i] >= 'a' && ch[i] <= 'h' && ch[j] >= '0' && ch[j] <= '9') {  
                tmp = ch[i];  
                ch[i] = ch[j];  
                ch[j] = tmp;  
            }  
        }  
    }  
}
```

패스워드가 유출되는 것을 방지하고 혼란을 가중시키기 위해 for문 과 while문 등의 분기와 루프를 다량으로 사용하여 composition 함수를 구현하였다 . 암호문이 문자열 중 숫자로만 이루어져있지만 리버싱 하는 과정에서 복잡하고 오랜 시간이 걸리게 하기 위해 위에 첨부한 코드 사진 중 빨간박스에 포함된 부분과 같이 if문에 문자나 특수문자 등의 의미없는 값도 적용해두었다.

void change(char *ch, int a, char b, int n) , void change2(char *ch, int a, int i)

```
void change(char *ch, int a, char b, int n) {  
    ch[a] ^= b;  
    ch[a] &= 0x0f;  
    ch[a] |= 0x30;  
    ch[a] = (((ch[a] - 48) % 10 + 48 + n % 10) - 48) % 10 + 48;  
}
```

```
void change2(char *ch, int a, int i) {  
    if (i == 0) {  
        if (ch[a] & (0x01)) (ch[a] = (ch[a] >> 1) | 0x08);  
        else (ch[a] = (ch[a] >>= 1) & 0x07);  
    }  
    if (i == 1) {  
        if (ch[a] & (0x01)) (ch[a] = (ch[a] >> 1) | 0x08);  
        else (ch[a] = (ch[a] >>= 1));  
    }  
    if (i == 2) {  
        if (ch[a] & (0x01)) (ch[a] = (ch[a] >> 1) | 0x01);  
        else (ch[a] = (ch[a] >>= 1) & 0x07);  
    }  
    if (i == 3) {  
        if (ch[a] & (0x01)) (ch[a] = (ch[a] >> 1) | 0x01);  
        else (ch[a] = (ch[a] >>= 1));  
    }  
    if (i == 4) {  
        if (ch[a] & (0x01)) (ch[a] = (ch[a] >> 1) | 0x04);  
        else (ch[a] = (ch[a] >>= 1) & 0x07);  
    }  
    if (i == 5) {  
        if (ch[a] & (0x01)) (ch[a] = (ch[a] >> 1) & 0x07);  
        else (ch[a] = (ch[a] >>= 1) & 0x0b);  
    }  
    if (i == 6) {  
        if (ch[a] & (0x01)) (ch[a] = (ch[a] >> 1) | 0x04);  
        else (ch[a] = (ch[a] >>= 1));  
    }  
    if (i == 7) {  
        if (ch[a] & (0x01)) (ch[a] = (ch[a] >> 1) | 0x02);  
        else (ch[a] = (ch[a] >>= 1) & 0x07);  
    }  
    if (i == 8) {  
        if (ch[a] & (0x01)) (ch[a] = (ch[a] >> 1) | 0x02);  
        else (ch[a] = (ch[a] >>= 1));  
    }  
    if (i == 9) {  
        if (ch[a] & (0x01)) (ch[a] = (ch[a] >> 1) & 0x0d);  
        else (ch[a] = (ch[a] >>= 1) & 0x0e);  
    }  
    ch[a] &= 0x0f;  
    ch[a] |= 0x30;  
    ch[a] = (((ch[a] - 48) % 10 + 48 + i % 10) - 48) % 10 + 48;  
}
```

change함수와 change2함수는 전체 암호화를 마친 암호문의 결과가 숫자로 나오기 때문에 그 암호문을 찾기 어렵도록 숫자로된 문자열을 추가하고 섞을 때 사용 할 숫자로 구성된 문자열을 만들기 위해 만들었다. 다른 문자열을 섞는 함수와 같이 사용하여 XOR연산, 다양한 경우의 shift연산을 해서 평문을 찾기 힘들게 한다.

void intcase2(char *ch, int nike)

```
void intcase2(char *ch, int nike) {
    int **arr = (int**)malloc((sizeof(int*) * 1000));
    for (int i = 0; i < strlen(ch); i++) {
        arr[i] = (int*)malloc((sizeof(int) * 1000));
    }
    int cnt = 0, m, n;
    int chcnt = 0;
    int idx;
    char tmpc, chb;
    nike += 3;
    n = sqrt(strlen(ch)) + 1;
    m = sqrt(strlen(ch)) + 2;
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < i + 1; j++) {
            nike++;
            if (j < n) {
                arr[j][i - j] = ++cnt;
            }
        }
    }
    for (int i = 1; i < n; i++) {
        nike++;
        for (int j = 0; j < n - i; j++) {
            if (j < m)
                arr[i + j][m - 1 - j] = ++cnt;
        }
    }
    nike++;
}
```

```

nike++;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        if (arr[i][j] - 1 < strlen(ch)) {
            nike += 2;
            if (chcnt == 0) {
                change2(ch, arr[i][j] - 1, (nike) % 10);
                idx = arr[i][j] - 1;
                chb = ch[arr[i][j] - 1];
                chcnt++;
            }
            else {
                //-----
                nike++;
                change2(ch, arr[i][j] - 1, (nike) % 10);
                change(ch, arr[i][j] - 1, chb, nike);
                //-----
                tmpc = ch[arr[i][j] - 1];
                ch[arr[i][j] - 1] = chb;
                chb = tmpc;
            }
            nike++;
        }
    }
}
ch[idx] = chb;
}
```

intcase2 함수는 case2의 방식과 change2, change 를 활용하여 문자열을 XOR하고, shift하고 섞어서 그 결과를 숫자문자로 뽑는다. 문자열을 암호화하고 후에 다른 암호문과 섞어서 길이를 늘이기 위해 만들었다.

void intcase3(char *ch, int nike)

```
void intcase3(char *ch, int nike) {
    int cnt = 0, cnt1, cnt2, n, m, i = 0, j = -1, aa = -1, m1, n1, idx;
    int chcnt = 0;
    char tmpc, chb;
    int **a = (int**)malloc(sizeof(int*) * 1000);
    for (int i = 0; i < strlen(ch); i++) {
        a[i] = (int*)malloc(sizeof(int) * 1000);
    }
    n = sqrt(strlen(ch)) + 1;
    m = sqrt(strlen(ch)) + 2;
    m1 = m;
    n1 = n;
    cnt1 = 0, cnt2 = 0;
    while (1) {
        aa = -1;
        while (cnt1 < m) {
            j += aa;
            a[i][j] = ++cnt;
            nike++;
            cnt1++;
        }
        cnt1 = 0;

        cnt--;
        while (cnt2 < n) {
            nike += 3;
            a[i][j] = ++cnt;
            i += aa;
            cnt2++;
        }
        i -= aa;
        cnt2 = 0;
        n--;
        m--;
        if (cnt == n1 * m1)
            break;
    }

    for (int i = 0; i < n1; i++) {
        nike++;
        for (int j = 0; j < m1; j++) {
            if (a[i][j] - 1 < strlen(ch)) {
                chchange2(ch, a[i][j] - 1, (nike) % 10);

                if (chcnt == 0) {
                    idx = a[i][j] - 1;
                    chb = ch[a[i][j] - 1];
                    chcnt++;
                }
                else {
                    //-----
                    chchange2(ch, a[i][j] - 1, (nike) % 10);
                    chchange(ch, a[i][j] - 1, chb, nike);
                    //-----
                    tmpc = ch[a[i][j] - 1];
                    ch[a[i][j] - 1] = chb;
                    chb = tmpc;
                }
            }
        }
        ch[idx] = chb;
    }
}
```

intcase3 함수는 case3의 방식과 change2, change를 활용하여 문자열을 XOR하고, shift하고 섞어서 그 결과를 숫자문자로 뽑는다. 문자열을 암호화하고 후에 다른 암호문과 섞어서 길이를 늘이기 위해 만들었다.

char* itoa(int val, char * buf, int radix)

```
char* itoa(int val, char * buf, int radix) {
    char* p = buf;

    while (val) {
        if (radix <= 10)
            *p++ = (val % radix) + '0';
        else {
            int t = val % radix;
            if (t <= 9)
                *p++ = t + '0';
            else
                *p++ = t - 10 + 'a';
        }

        val /= radix;
    }

    *p = '\0';
    return buf;
}
```

enc에 저장된 정수의 값들을 문자형으로 바꿔서 새로운 변수에 이어 붙이게 하였다.

char *int2char(int *p, int cnt)

```
char *int2char(int *p, int cnt) {
    char *b;
    int i = 0;
    b = (char *)malloc(sizeof(char) * 10000);
    char c[10000] = { '\0' };
    for (i = 0; i < cnt; i++) {
        itoa(p[i], &b[i], 10);
        strcat(c, &b[i]);
    }

    return &c[0];
}
```

아스키 코드 값의 범위가 넘어가는 정수 들이 저장된 int형 배열 전체를 0~9의 숫자가 나열된 char형 배열로 변환한다.

char *func(char *pass, char *cpass)

```
char *func(char *pass, char *cpass) {
    int *enc;
    int add;
    int i, j, k = 0;

    enc = (int*)malloc(sizeof(int)*(int)(strlen(pass) + strlen(cpass) - 1) + 1);
    for (j = 0; j < (int)strlen(pass) + (int)strlen(cpass) - 1; j++) {
        enc[j] = 0;
    }
    for (i = 1; i < strlen(cpass); i++) {
        add = 0;
        for (j = 0; j < i; j++) {
            add += pass[i - 1 - j] * cpass[j];
        }
        enc[i - 1] = add;
    }
    for (i = (int)strlen(cpass); i <= (int)strlen(pass) - 1; i++) {
        add = 0;
        for (j = 0; j < strlen(cpass); j++) {
            add += pass[i - 1 - j] * cpass[j];
        }
        enc[i - 1] = add;
    }
    for (i = (int)strlen(cpass); i > 0; i--) {
        k++;
        add = 0;
        for (j = 0; j < i; j++) {
            add += pass[strlen(pass) - 1 - j] * cpass[j + k - 1];
        }
        enc[strlen(pass) - 2 + k] = add;
    }

    return int2char(enc, strlen(pass) + strlen(cpass) - 1);
}
```

두개의 문자열을 입력 받아 합성하여 하나의 char형 배열로 출력하도록 하였다. 입력하는 두 문자열은 모두 사용자가 입력하는 비밀번호에 종속되는 값이다. 입력 받은 문자열의 아스키 코드를 정수로 받아 두 개의 배열을 덧셈과 곱셈을 하도록 조합하였다. 또한 출력되는 결과는 아스키 코드 값을 벗어나는 수(10^4 의 자리수 이상)로 정수형을 문자형으로 변환하여 저장하였다. 따라서 역공학의 관점에서는 각 숫자들의 자리수를 알지 못하고, 키의 길이 또한 알지 못하므로, 암호문을 유출라도 나열된 숫자들에서 정확한 암호문을 찾을 수 없도록 하였다.

역공학의 관점에서 초기의 특정 암호가 아닌 다른 여러 가지 암호문으로 해석할 여지가 생기지만, 각각의 암호가 func함수의 역함수를 통과하면 몇몇개의 문자열 쌍이 생성되고, 이 문자열 쌍이 그 전의 암호알고리즘을 역으로 통과했을 때, 평문이 같아야한다는 점, 또한 그 평문이 문자,숫자,특수문자의 아스키 코드 값의 범위에 있어야한다는 점을 고려할때 충돌되는 문자열은 극히 적을 것이라 생각된다.

char* encrypt_isdigit(char *str)

```
char* encrypt_isdigit(char *str) {
    int x = 0;

    for (x = 0; x < strlen(str); x++) {

        if (isdigit(str[x]))
        {
            str[x] = (str[x] - '0' + 3) % 10 + '0';
        }
    }

    return str;
}
```

사용자가 입력한 password 에 digit value(숫자 값) 가 있을 경우 3칸씩 뒤로 shift 시키는 함수이다.

char* encrypt_rot10(char *str)

```
char* encrypt_rot10(char *str) {

    int i = 0;
    int c = 0;

    for (i = 0; i < strlen(str); i++) {
        if (str[c] >= 'a' && str[c] <= 'm') str[c] += 10;
        else if (str[c] >= 'A' && str[c] <= 'M') str[c] += 10;
        else if (str[c] >= 'n' && str[c] <= 'z') str[c] -= 10;
        else if (str[c] >= 'N' && str[c] <= 'Z') str[c] -= 10;
        c++;
    }

    return str;
}
```

encrypt_rot10은 Caesar cipher 와 비슷하게 사용자가 입력한 password value 중 알파벳 값이 들어올 경우 알파벳 값 만을 10칸 씩 뒤로 shift 시키는 함수이다. 이는 비록 쉽게 간파되는 방식이긴 하나 다른 암호화 방식 알고리즘과 혼용하여 사용하는 것으로 단점을 보완하였다.

void data_masking(char *arr)

```
void data_masking(char *arr) {  
  
    char rarr[100];  
    int i = 0;  
  
    while ((arr[i] = getch()) != 'Wr') {  
        if (arr[i] == 8) {  
            if (i > 0) {  
                printf("Wb");  
                printf(" ", stdout);  
                printf("Wb");  
  
                arr[i - 1] = 'W0';  
                i--;  
            }  
        }  
        else {  
            printf("*");  
            rarr[i] = arr[i];  
            i++;  
        }  
    }  
    printf("Wn");  
    rarr[i] = 'W0';  
    strcpy(arr, rarr);  
}
```

입력하는 실제 패스워드 데이터를 노출시키지 않기 위해 데이터 필드 마스킹 효과를 이용하여 data_masking 함수를 구현하였다.
이 함수는 실제 패스워드 데이터에 대한 수정이나 변경, 변화가 없으며 실제 패스워드 데이터를 특수문자인 *로 변조함으로써 실제 패스워드 데이터의 의미를 알 수 없도록 하고 또한 데이터의 유출을 방지할 수 있다.

char* encrypt_basic(char *str)

```
char* encrypt_basic(char *str) {
    int x = 0;
    char *tem;
    char *add;
    tem = (char*)malloc(sizeof(char) * 100000);
    add = (char*)malloc(sizeof(char) * 100000);
    strcpy(tem, str);
    strcpy(add, str);
    encrypt_isdigit(str);
    encrypt_rot10(str);

    for (x = 0; x < strlen(str); x++) {
        str[x] = ((str[x] - 33) + 10) % 95 + 33;
    }

    for (int j = 0; j < 2; j++) {
        for (int i = 0; i < 3; i++) {
            if (strlen(add) % 2 == 0) {
                charcase2(add, strlen(add));
                charcase3(add, strlen(add));
            }
            else {
                charcase3(add, strlen(add));
                charcase2(add, strlen(add));
            }
        }
        strcat(tem, add);
    }

    for (int j = 0; j < 2; j++) {
        for (int i = 0; i < 3; i++) {
            if (strlen(add) % 2 == 0) {
                charcase3(add, strlen(add));
                charcase2(add, strlen(add));
            }
            else {
                charcase2(add, strlen(add));
                charcase3(add, strlen(add));
            }
        }
        strcat(str, add);
    }

    composition(str);
    case2(str);
    str = func(str, tem);
    strcpy(add, str);
    for (int i = 0; i < 3; i++) {
        if (strlen(str) % 2 == 0) {
            intcase2(add, strlen(str) % 10);
            intcase3(add, strlen(str) % 10);
        }
        else {
            intcase3(add, strlen(str) % 10);
            intcase2(add, strlen(str) % 10);
        }
        strcat(str, add);
    }

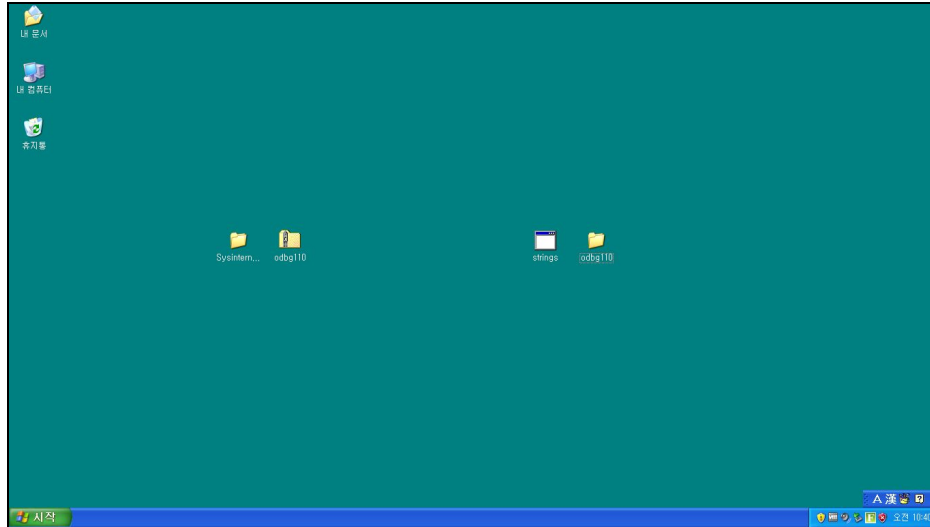
    case3(str);
    composition(str);

    return str;
}
```

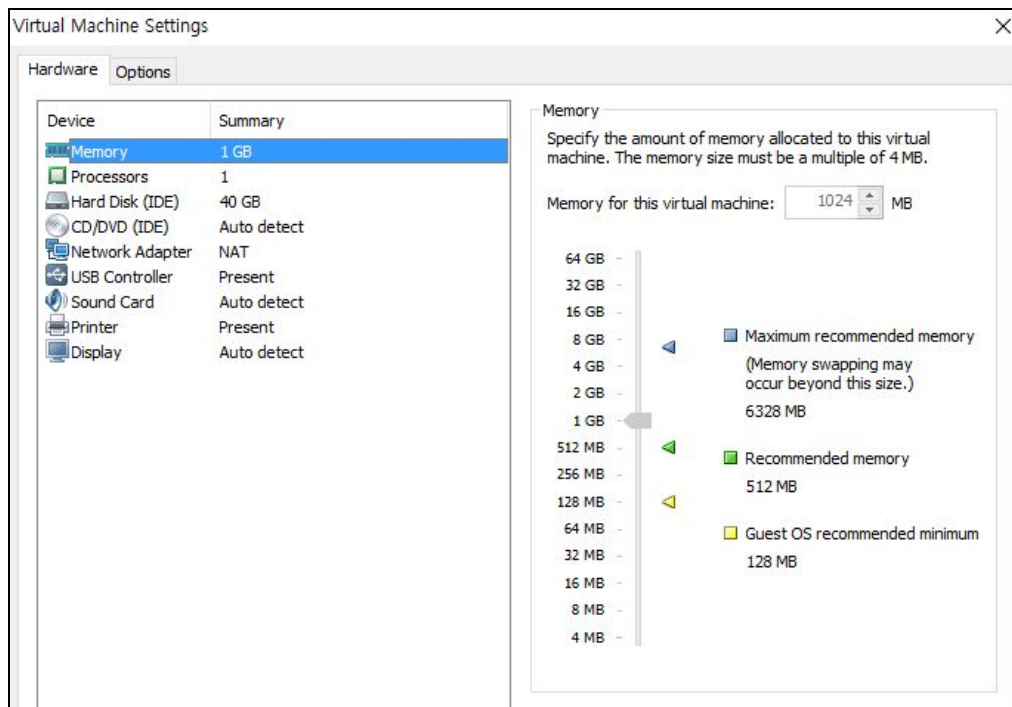
위의 암호화 알고리즘 들을 합쳐 하나의 함수로 만들었다. encrypt_basic 함수에 평문을 입력하면 암호화된 문이 출력되도록 하여, 추후 유지 보수 및 프로그램의 변경에 있어 용이하도록 하였으며, 기존의 encrypt_isdigit, encrypt_rot10 으로 암호화 한 결과물을 문자열을 섞는 composition 및 case 함수 들로 다시 뒤섞을 뿐만 아니라 convolution(“func()”) 과정 전에 한번 더 shift 시켜 다중 암호화를 가능케 하는 역할이다.

2. 안티리버싱 여부를 검증하기 위한 테스트 과정을 설명하시오.

2-A. 테스트 환경



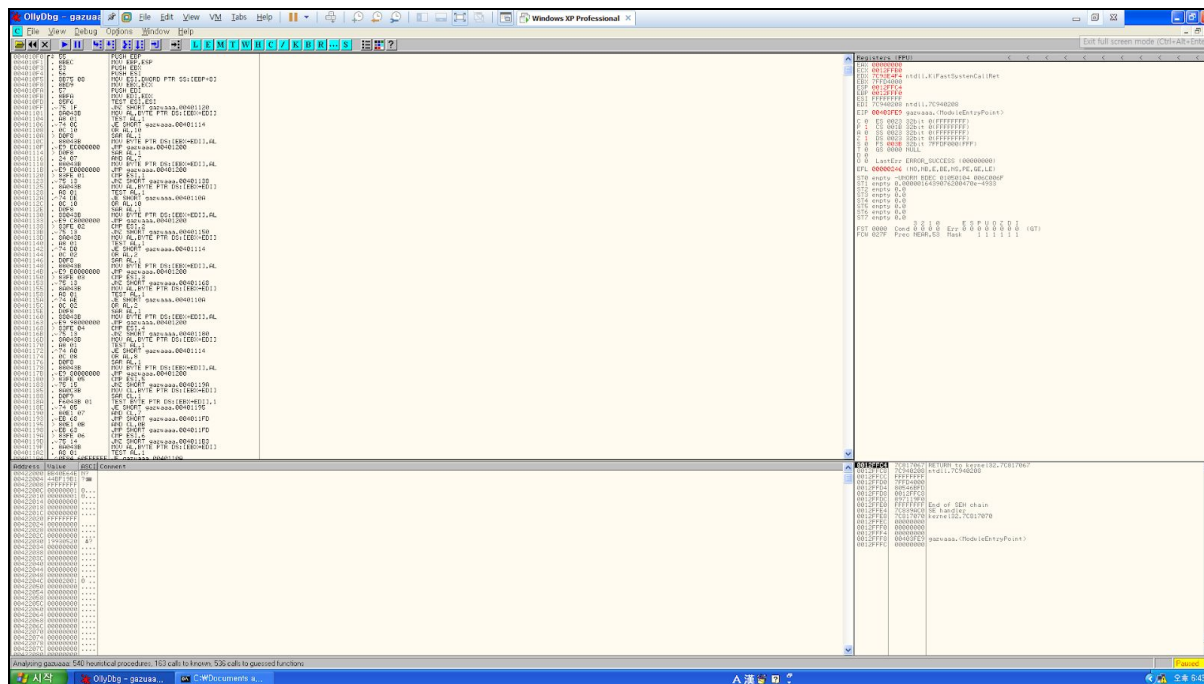
운영체제 : window xp 32bit professional version ISO



HW 사양

2-B. 디버깅을 통한 확인

XP 운영체제 환경에서 OllyDbg Tool (OLLYDBG 32bit - 1.0.10.0 ver) 을 사용해 .exe 실행파일을 열어본 결과 아래 그림과 같이 default password를 제외하고는 string value로 검출되는 비밀번호가 없음을 확인 할 수 있다.



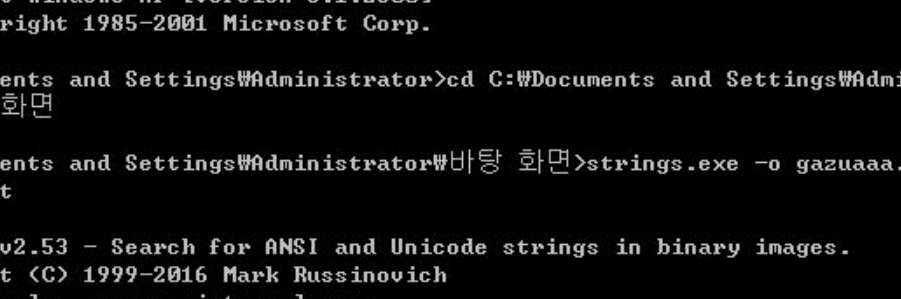
Address	Disassembly	Text string
00404E00	PUSH <i>gzuwaaa</i> .0041B238	ASCII "f[all]loc"
00404F0E	PUSH <i>gzuwaaa</i> .0041B24C	ASCII "f[isFree"
00404F18	PUSH <i>gzuwaaa</i> .0041B24C	ASCII "f[isFree"
00404F48	PUSH <i>gzuwaaa</i> .0041B268	ASCII "f[isSetValue"
00404F53	PUSH <i>gzuwaaa</i> .0041B268	ASCII "f[isSetValue"
00404F67	PUSH <i>gzuwaaa</i> .0041B27C	ASCII "InitializeCriticalSectionEx"
00404F7A	PUSH <i>gzuwaaa</i> .0041B27C	ASCII "InitializeCriticalSectionEx"
00406028	MOV DWORD PTR DS:[ESI+34], <i>gzuwaaa</i> .0041B8	ASCII "(null)"
0040715D	MOV DWORD PTR DS:[ESI+34], <i>gzuwaaa</i> .0041B8	UNICODE "(null)"
00407177	MOV DWORD PTR DS:[ESI+34], <i>gzuwaaa</i> .0041B8	ASCII "(null)"
00408B69	PUSH <i>gzuwaaa</i> .0041C460	UNICODE "minkernel\crts\win\inc\corecrt_internal_stdio.h"
00408B8E	PUSH <i>gzuwaaa</i> .0041C46C	UNICODE " _crt_ctors:float_point_value:as_double"
00408C0A	PUSH <i>gzuwaaa</i> .0041C46C	UNICODE " _crt_ctors:float_point_value:as_double"
00408D0E	PUSH <i>gzuwaaa</i> .0041C460	UNICODE "minkernel\crts\win\inc\corecrt_internal_stdio.h"
00408E63	PUSH <i>gzuwaaa</i> .0041C540	UNICODE " _crt_ctors:float_point_value:as_float"
00408E6B	PUSH <i>gzuwaaa</i> .0041C59C	UNICODE " _is_double"
0040C0AB	ASCII "Fe",0	
0040CB5F	PUSH <i>gzuwaaa</i> .0041C704	UNICODE "mscoree.dll"
0040CB87	PUSH <i>gzuwaaa</i> .0041C71C	ASCII "CrtExitProcess"
0040CC83	CMF EAX,10000	UNICODE "::~:i:"
0040CCF7	CMF EAX,10000	UNICODE "::~:i:"
0040CD06	CMF EAX,10000	UNICODE "::~:i:"
0040E236	PUSH <i>gzuwaaa</i> .0041CC50	ASCII "cos"
0040E261	PUSH <i>gzuwaaa</i> .0041CC54	ASCII "UIT-g"
0040E28A	PUSH <i>gzuwaaa</i> .0041CC54	ASCII "UIT-gLEUNICODE"
0040E29F	PUSH <i>gzuwaaa</i> .0041CC64	ASCII "UNICODE"
0040E4FA	PUSH <i>gzuwaaa</i> .0041D140	ASCII "CompareStringEx"
0040E55A	PUSH <i>gzuwaaa</i> .0041D140	ASCII "CompareStringEx"
0040E63C	PUSH <i>gzuwaaa</i> .0041D128	ASCII "AreFileApisANSI"
0040E646	PUSH <i>gzuwaaa</i> .0041D128	ASCII "AreFileApisANSI"
0040E70A	PUSH <i>gzuwaaa</i> .0041D13C	ASCII "f[all]loc"
0040E75E	PUSH <i>gzuwaaa</i> .0041B24C	ASCII "f[isFree"
0040E7B4	PUSH <i>gzuwaaa</i> .0041B254	ASCII "f[isSetValue"
0040E80A	PUSH <i>gzuwaaa</i> .0041B268	ASCII "f[isSetValue"
0040E863	PUSH <i>gzuwaaa</i> .0041B27C	ASCII "InitializeCriticalSectionEx"
0040E8B6	PUSH <i>gzuwaaa</i> .0041D19C	ASCII "CHeapStringEx"
0040E8C3	PUSH <i>gzuwaaa</i> .0041D19C	ASCII "CHeapStringEx"
0040E943	PUSH <i>gzuwaaa</i> .0041D1B4	ASCII "LocalNameToLCID"
0040E94D	PUSH <i>gzuwaaa</i> .0041D1B4	ASCII "LocalNameToLCID"
0040E9D0	PUSH <i>gzuwaaa</i> .0041D19C	ASCII "GetCurrentPackageId"
0040E9E4	PUSH <i>gzuwaaa</i> .0041D178	ASCII "GetCurrentPackageId"
0041024F	PUSH <i>gzuwaaa</i> .0041D208	ASCII "e+000"
00410E5F	MOV EAX, <i>gzuwaaa</i> .0041D1C8	ASCII "1"
00410E5E	MOV DWORD PTR SS:[EBP-24], <i>gzuwaaa</i> .0041D1	ASCII "NaN(SNaN)"
00410E70	MOV EAX, <i>gzuwaaa</i> .0041D1CC	ASCII "inf"
00410E74	MOV EAX, <i>gzuwaaa</i> .0041D1D4	ASCII "nan"
00410E84	MOV DWORD PTR SS:[EBP-13], <i>gzuwaaa</i> .0041D1	ASCII "nan(snan)"
00410E9E	MOV ESI, <i>gzuwaaa</i> .0041D1D0	ASCII "NaN"
00410E9D	MOV DWORD PTR SS:[EBP-14], <i>gzuwaaa</i> .0041D1	ASCII "NaN(ind)"
00410F01	MOV DWORD PTR SS:[EBP-C], <i>gzuwaaa</i> .0041D1	ASCII "nan(ind)"
00413200	CMF EDI,10000	UNICODE "::~:i:"
00413223	PUSH <i>gzuwaaa</i> .0041D028	UNICODE "CONTIN"
004134B6	CMF EAX,10000	UNICODE "::~:i:"
00415E4F	PUSH <i>gzuwaaa</i> .0041F0BC	ASCII "1#IND"
00415E57	PUSH <i>gzuwaaa</i> .0041F0BC	ASCII "1#SNaN"
00415E6D	PUSH <i>gzuwaaa</i> .0041F0AC	ASCII "1#ONAN"
00415E64	PUSH <i>gzuwaaa</i> .0041F0D4	ASCII "1#INF"
00415E8B	PUSH <i>gzuwaaa</i> .0041F0F4	ASCII "CONTIN"
00415E90	MOV DWORD PTR SS:[EBP-20], <i>gzuwaaa</i> .00420A	ASCII "log10"
00415E90	MOV DWORD PTR SS:[EBP-20], <i>gzuwaaa</i> .00420A	ASCII "log10"
00415E98	MOV DWORD PTR SS:[EBP-20], <i>gzuwaaa</i> .00420A	ASCII "log"
00415E98	MOV DWORD PTR SS:[EBP-20], <i>gzuwaaa</i> .00420A	ASCII "log"
00415EAB	MOV DWORD PTR SS:[EBP-20], <i>gzuwaaa</i> .00420A	ASCII "exp"
00415F01	MOV DWORD PTR SS:[EBP-20], <i>gzuwaaa</i> .00420A	ASCII "pow"
00415F0D	MOV DWORD PTR SS:[EBP-20], <i>gzuwaaa</i> .00420A	ASCII "exp"
00415F52	MOV DWORD PTR SS:[EBP-20], <i>gzuwaaa</i> .00420A	ASCII "asin"
00415F52	MOV DWORD PTR SS:[EBP-20], <i>gzuwaaa</i> .00420A	ASCII "acos"
00415F6A	MOV DWORD PTR SS:[EBP-20], <i>gzuwaaa</i> .00420A	ASCII "sqrt"
00415F76	MOV DWORD PTR SS:[EBP-20], <i>gzuwaaa</i> .00420A	ASCII "pow"
00415F76	MOV DWORD PTR SS:[EBP-20], <i>gzuwaaa</i> .00420A	ASCII "pow"

Search for ▶ All referenced text strings 로 따로 검출한 Strings value 들 에서도 검출 사항이 없었으며, 당초 password를 저장 시 문자열 shift 옹 encrypt 와 string value 등을 섞는 알고리즘, convolution(func ()) 등을 전부 적용시켜 저장하고, 또 이를통해 비교 및 대조를 한다

(재귀함수 등은 코드 작업에 사용하지 않았으므로 메모리 주소상의 끝나지 않는 루프는 없다.)

2-C. Strings 를 통한 확인

sysinternals 사의 strings 툴을 이용하여 확인한 결과 밑의 결과물과 같이 default password 인 “sejong security 2017!”를 제외한 다른 패스워드 정보가 검출되지 않았다.



```
C:\명령 프롬프트
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>cd C:\Documents and Settings\Administrator\바탕 화면

C:\Documents and Settings\Administrator\바탕 화면>strings.exe -o gazuaaa.exe > output.txt

Strings v2.53 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2016 Mark Russinovich
Sysinternals - www.sysinternals.com

C:\Documents and Settings\Administrator\바탕 화면>
```

