제 4회 KISA 해킹방어대회 예선전 풀이

작성자: 동명대학교 정보보호동아리 THINK-1팀 태인규 (graylynx at gmail.com)
최영남 (mysunset at nownuri.net)

정보통신부와 한국정보보호진흥원에서는 해킹방어 기법 및 관련 기술 동향 파악을 통해 침해사고 대응능력을 높일 수 있는 계기를 마련하고, 정보교류 활성화와 기술의 향상을 도모하기 위해 다음과 같이「제4회 해킹방어대회」를 개최합니다.

□ 대회 개요

o 인터넷 침해사고 예방 능력, 탐지 및 대응 능력 평가

□ 참가대상 및 방법

o 참가대상 : 정보보호에 관심있는 자로 2인 이하의 팀을 구성하여 신청

o 참가신청기간 : 4월 16일(월) ~ 4월 30일(월)

o 참가신청방법 : 참가신청서를 작성하여 이메일로 신청 (hdcon2007@knsp.org)

※ 접수 후 2일 이내로(토, 일요일 제외) 회신이 없는 경우 재신청요망

□ 대회 내용

o 예선 대회

- 5. 3(목) ~ 4(금), 온라인상으로 진행

※ 상세 시간은 추후 공지

- 주어진 시간동안 온라인 접속하여 해킹기법 문제풀이 진행
- 평가팀은 각 참가팀이 해결한 문제 수를 평가하여 상위팀을 본선진출팀으로 선정
- 동일 점수의 경우 선착순에 우선함

o 본선 대회

- 5. 17(목), KISA 5층 상황전파실

※ 상세 시간은 추후 공지

- 참가팀에게 동일한 취약점이 존재하는 시스템을 각 1대씩 배포하여 평가 실시
- 침해사고 예방 능력 평가
- 침해사고 탐지 및 대응 능력 평가
- 문제별 점수를 차등 부여하여 최종 점수로 우승자 선정

□ 시상 내용

o 대상 (정보통신부장관상) : 상금 200만원 및 상장 (1팀)

o 금상 (한국정보보호진흥원장상) : 상금 100만원 및 상장 (2팀)

o 은상 (한국정보보호진흥원장상) : 상금 50만원 및 상장(2팀)

※ 대학동아리 상위 2팀에게 부상으로 해외컨퍼런스 참가경비 지원

처음 서버에 접속 했을 때 화면입니다. 참가 신청서를 제출할 때 적어놓은 아이디로 Enroll 하면 서버에 등록할 수 있습니다. 아침 10시가 조금 넘어서 서버가 열렸는데, 접속 과부화로 거의 기어가는 듯 하더군요. 참가자들의 뜨거운 열기를 느낄 수 있었습니다. 저희는 거의 10시 30분이 되어서야 스테이지1 문제를 볼 수 있었습니다.

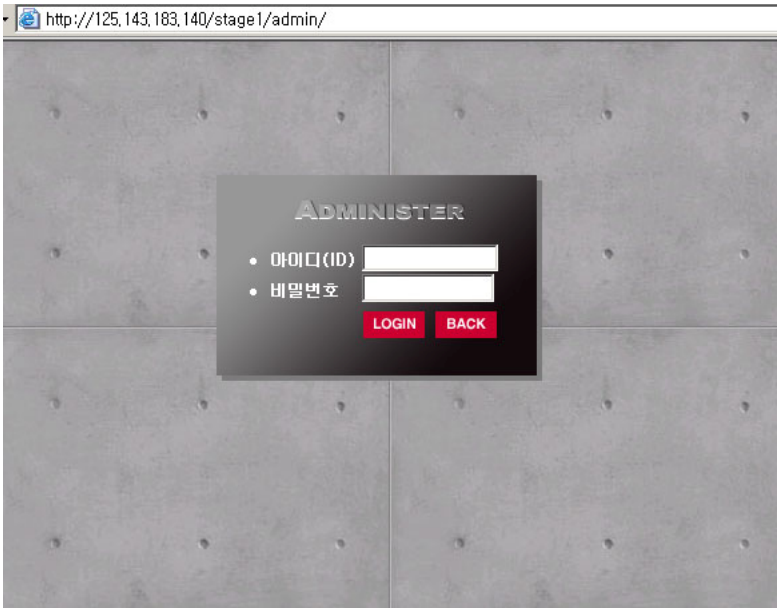
스테이지1

현재 단계는 0 입니다.
1번문제 : 서버관리자 마동탁는 새로운 페이지를 구성하기 위해 작업중이다.
새로운 페이지를 모두 구성하기도 전에 보안 취약점이 발생하였다.
이를 이용하여 다음 단계의 패스워드를 획득하라.
ID : stage1
PASSWORD : W21c0m22HHHHDDDD

등록 후에 알려주는 주소로 아이디와 패스워드를 치고 들어가면 다음과 같은 화면이 나옵니다.

문제에서 '서버관리자' 와 '새로운 페이지를 구성하기 전' 이라는 부분에 초점을 맞췄습니다. 시작한지 얼마 되지 않아, /admin/ 이라는 디렉토리를 찾아낼 수 있었습니다.

http://125.143.183.140/stage1/admin/ 주소로 들어가면 아래와 같은 화면이 뜹니다.

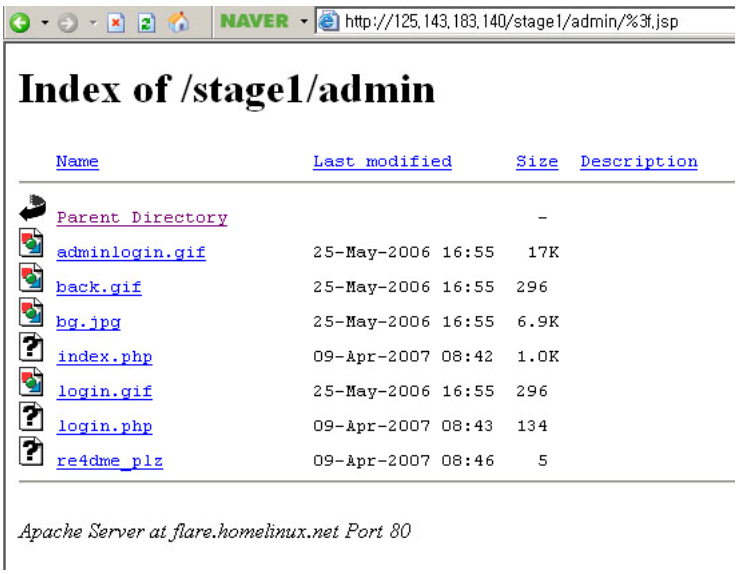


작년 문제로 미뤄봤을 때 SQL Injection 과 Cookie Spoofing 문제일 가능성이 높아서 그쪽으로 방향을 잡고 공격하기 시작했습니다. 그러나 아무리 공격해도 열릴 기미가 안보이더군요. 삽질을 시작한지 2시간을 넘기고 '스테이지1이 이렇게 어려울 리가 없는데,,' 하며 난감해 하던 중 힌트가 떴습니다.

STAGE1 힌트

WAS, Directory Indexing,

WAS 가 jsp 와 관련이 있다는 건 알고 있었기에, jsp 취약점 위주로 google 에서 검색한 결과를 토대로 공격하기 시작했고, 결국 %3f.jsp 취약점을 통해 아래와 같이 디렉토리 내의 파일들을 볼 수 있었습니다.



위의 파일들 중 re4me_plz 라는 파일을 열어보시면 스테이지2의 비밀번호가 들어있습니다.

%3f.jsp 에 대한 자세한 내용은 헐랭이의 이글루를 (http://swbae.egloos.com/1422851) 참조하시기 바랍니다.

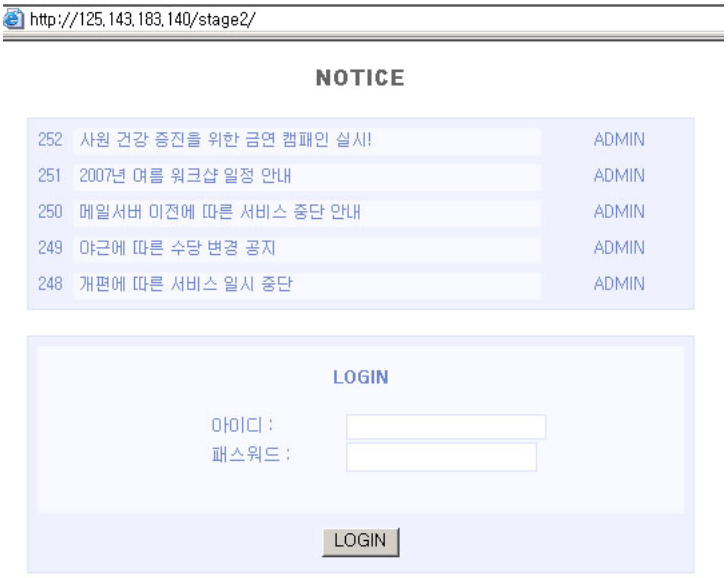
스테이지2

현재 단계는 1 입니다.

2번문제 : 웹 어플리케이션 개발자 고길동은 과중한 업무로 인하여 중대한 보안 취약점이 존재하

는 웹 어플리케이션을 개발하였다.
이 취약점을 발견하고 이를 이용하여 다음 단계의 패스워드를 획득하라.
ID : stage2, PASSWORD : 획득한 패스워드
게시판 접속 계정은(guest / guest)입니다.

스테이지2의 주소로 아이디와 패스워드를 치고 들어가면 다음과 같은 화면이 나옵니다.



문제에서 알려준 대로 아이디와 패스워드에 guest/guest 를 치고 들어가면 다음과 같은 게시판이 나타납니다.

저희가 처음 들어갔을 때 이미 몇몇 분이 먼저 들어오셨더군요. 그때 게시물 번호가 20번대 였는데, 점심시간이 지나자 게시물의 수가 급격히 늘어나더라는,,

아무리 생각해봐도 출제자의 의도를 알 수 없었습니다. 취약점이 있을 거라고 생각되는 파일이라곤 view.php 와 write.php 인데, 우리 팀은 view.php 파일에 더 비중을 두었습니다. 패스워드를 알아내는 것이 목적이기 때문에 아무래도 글 쓰는 것 보단 글을 읽는 것과 관련이 있을 거라 생각했기 때문입니다.

view.php?category=notice&num=248 에서 num 변수에 범위에 벗어난 값을 입력하면 '아직 공개되지 않은 게시물 입니다' 라고 출력이 되어서, 혹시나 Brute-force 문제가 아닐까 하여 스크립트를 작성하여 돌렸다가 서버가 잠시 먹통이 되기도 했습니다. (죄송합니다T_T)

view.php 는 아무리 봐도 아닌 거 같고, 공지게시판에 글이나 써보자 라는 생각으로 이리저리 삽질한 결과, write.php 를 실행할 때, Referer 값과 HTTP 헤더 변수 값을 동시에 notice 로 조작해서 보내니 스테이지3의 암호가 나왔습니다. 이 때 약간 어리둥절했는데, '지금은 실전 PT가 아니라 어디까지나 모의로 만들어진 문제들이니까,' 라고 생각했습니다. 덕분에 다음 스테이지를 푸는데 약간 도움이 되었다는,, ^^;

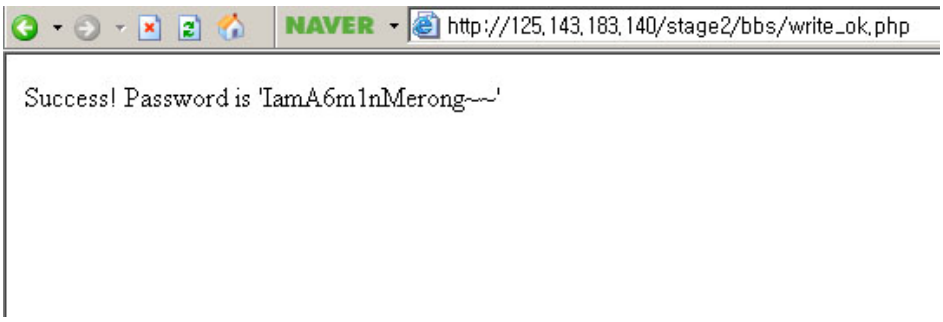
```
POST http://125.143.183.140/stage2/bbs/write_ok.php HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave
/vnd.ms-powerpoint, application/msword, */*
Referer: http://125.143.183.140/stage2/bbs/write.php?category=notice
Accept-Language: ko
Content-Type: multipart/form-data; boundary=---------------------------7d79c251204b8
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50
Host: 125.143.183.140
Content-Length: 548
Pragma: no-cache
Cookie: PHPSESSID=3c9ed9da2b2fa88e5283c7b0397911fb

-----------------------------7d79c251204b8
Content-Disposition: form-data; name="category"

notice
-----------------------------7d79c251204b8
Content-Disposition: form-data; name="name"

guest
-----------------------------7d79c251204b8
Content-Disposition: form-data; name="title"
```

Paros 에서 HTTP 헤더를 조작해서 보낼 때 위 그림과 같이 보내면, 스테이지3의 패스워드 출력.



스테이지3

현재 단계는 2 입니다.
3번문제 : 웹 보안관리자 마루치는 전반기 보안점검에서 게시판에 심각한 취약점이 존재한다는 것을 전해 들었다. 그러나 마루치는 이 취약점을 도저히 찾을 수가 없었다. 여러분이 이 취약점을 찾고 이용하여 다음단계의 패스워드를 획득하라.
ID : stage3, PASSWORD : 획득한 패스워드

스테이지3의 주소로 아이디와 패스워드를 치고 들어가면 다음과 같은 화면이 나옵니다.

처음에는 Directory/File guessing 이라고 생각하고 게시판과 관련된 모든 단어들을 유추해서 넣기 시작했습니다. 그래서 write.php write_ok.php common.php 등의 파일들을 발견했는데, 문제를 푸는 데에는 직접적으로 관련이 없어 보였습니다. 계속 삽질을 하면할수록 이게 삘이 안 오더군요. 삽질 다하고 지쳐갈 때 쯤, 스테이지3의 힌트가 떴습니다.

STAGE3 힌트

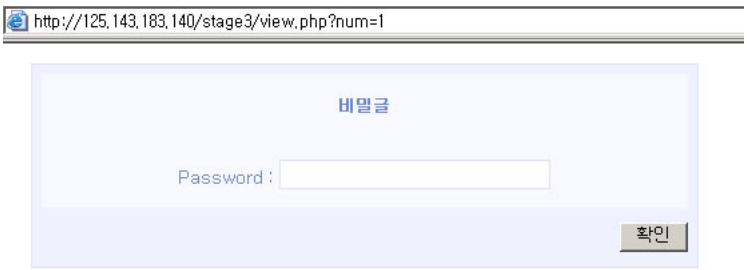
SQL Injection

이 상황에서 우리가 조작할 수 있는 데이터들이라 함은 HTTP 헤더밖에 없다는 생각에 모든 HTTP 헤더 변수에 SQL Injection 을 시도했습니다. 이리저리 대입해본 결과 User-Agent 헤더에 SQL Injection 취약점이 존재했고, 이를 통해 보이지 않던 게시판을 볼 수 있었습니다.

```
GET http://125.143.183.140/stage3/ HTTP/1.0
Accept: */*
Accept-Language: ko
Pragma: no-cache
User-Agent: THINK' or 1=1'
Host: 125.143.183.140
Proxy-Connection: Keep-Alive
Cookie: PHPSESSID=3c9ed9da2b2fa88e5283c7b0397911fb
Authorization: Basic c3RhZ2UzOklhbUE2bTFuTWVyb25nfn4=
```

위와 같이 User-Agent 헤더를 조작하면, 아래와 같은 게시판을 찾을 수 있습니다.

느낌상 왠지 1번 게시물에 패스워드가 적혀 있을 것 같았습니다. 눌러보니,,



뭔가 또 입력하라고 나오네요. 별다른 힌트가 없는 걸로 보아 이것 역시 SQL Injection 이라 생각하고 공격했습니다. 수 차례의 삽질 끝에,

" or ""="

를 입력해서 스테이지4의 패스워드를 볼 수 있었습니다.

스테이지4

현재 단계는 3 입니다
4번문제 : 쉬어가는 페이지이다. 게임을 즐기며 취약점을 찾아라.
ID : stage4, PASSWORD : 획득한 패스워드

스테이지4의 주소로 아이디와 패스워드를 치고 들어가면 다음과 같은 화면이 나옵니다.

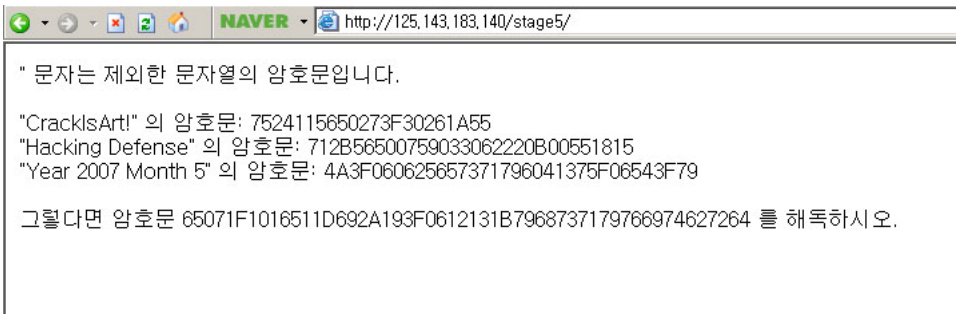
Flash 게임입니다. 우리는 컴퓨터를 먹어서 1000만점 이상 기록해야지만, 다음 스테이지의 패스워드를 얻을 수 있습니다. 마침 문제들에 질려가던 차에 플레이를 해봤는데, 2만점까지 하다가 눈에서 착시현상이 일어나길래 그만뒀습니다. 실제 플레이를 해서 1000만점이 되는 건 대회 끝나기 전에는 불가능하겠더군요.

Flash 파일을 디컴파일 해보면 외부 라이브러리를 가져와서 특정 값들로 암호화를 한 뒤 score 변수에 담아서 위의 주소로 호출하게 되어있습니다. 그러므로 암호화 라이브러리를 링크해서 의도적으로 score 값을 조작한 다음에 보내줘야 합니다. 아무래도 나중에 많은 분들이 막혀서 난이도를 내린 게 아닐까 생각되네요. 이 문제는 대한 풀이는 따로 저희 팀원 최영남군이 쓴 문서를 참조하시기 바랍니다. (첨부파일: 문제4번풀이.hwp)

스테이지5

현재 단계는 4 입니다
5번문제 : 공각기동대 공안9과에서는 사건의 결정적인 단서가 될 파일 하나 입수했다.
그러나 해당 내용이 암호화 되어 있어 수사의 진전이 없는 상태이다.
다만 관련자 심문을 통해 몇가지 규칙을 알아냈다.
암호화문을 해독하여 다음 단계의 패스워드를 획득하라.
ID: stage5 , PASSWORD: 획득한 패스워드

스테이지5의 주소로 아이디와 패스워드를 치고 들어가면 다음과 같은 화면이 나옵니다.



후,, 문서를 쓰다가 그때의 악몽이 다시금 떠오르네요. 지금 생각해봐도 이 문제는 추가힌트 없이는 시간 내에 풀기 어렵다고 생각합니다. 저희가 저 문제를 처음 접했을 때가 첫날 오후 6시 정도였던 걸로 기억하는데요, 잠 2시간 자고 그 다음날 오후3시까지 쉬지 않고 풀었습니다. 처음에 ROT 와 XOR 이 사용되었다는 힌트 정도라도 주었으면 그렇게 피곤하게 풀진 않았을 텐데요. T_T

처음으로 돌아가서,, 일단 주어진 문자열들의 규칙성을 찾는데 집중했습니다만, 규칙을 찾을 수 없었습니다. 이리도 해보고 저리도 해보고 했지만, 경우의 수가 너무나 많았기에 사막에서 모래알 찾는 심정이더군요. 아무리 해도 안보이길래 그냥 스테이지4의 게임으로 시간이나 때우면서 힌트를 기다렸습니다. 그러던 중 밤 9시 좀 안되서 첫 힌트가 떴습니다.

STAGE5 힌트

암호화 순서
1. 변형된 ROT
2. 변형된 배타 연산 암호화

추가 힌트
1. 문자열의 배열을 변형된 ROT로 암호화
2. 1에서 암호화된 문자열 XOR(규칙에 의해 선별된 문자)

추가 힌트
1. ROT Table에는 영어 대소문자, 숫자, 스페이스만 사용
2. ROT Table 배열은 영어 알파벳 순이 아님

암호화 알고리즘의 순서는 이렇습니다.

1. Plain-Text 의 각각의 문자가 변형된 ROT-n 에 의해 다른 문자로 치환
2. 규칙에 의해 선별된 문자와 XOR

우리는 세 문장의 Plain-Text 와 Encrypted-Text 를 가지고 있고 여기서 규칙성을 발견하여 주어진 암호문을 해독해야 합니다. ROT 로 치환된다는 건 어떤 값으로 치환되건 그 문자의 빈도수는 같습니다. 하지만 여기에 특정 값들로 XOR 하기 때문에 그 빈도수가 헝클어지게 됩니다.

우리는 우선 CrackIsArt! 문장과 Year 2007 Month 5 문장에서 같은 세 번째 위치의 소문자 a 에 주목했습니다. ROT 를 통해 어떤 값으로 치환되었다고 해도 그 두 값이 같을 텐데, 결과로 나온 암호문에는 두 값이 달랐습니다. 이로써 각 문장마다 XOR 테이블이 다르다는 것을 알아냈습니다.

또한 CrackIsArt! 문장과 Hacking Defense 문장의 ack 문자열에 주목했습니다. 문자 c 와 k 에 해당하는 암호화된 값이 같으므로 (0x56, 0x50 <- 저희는 암호화된 값이 헥사 코드라고 가정했습니다) 첫 문장의 XOR 테이블의 4, 5번째 값이 세 번째 문장의 XOR 테이블의 3, 4 번째 값과 같다는 것을 알아냈습니다.

사실 이 때 조금만 더 주의 깊게 생각했다면 XOR 테이블의 규칙을 발견할 수 있었을 텐데, 불행히도 저희는 '헐,, 이것만 가지고 어쩌라고?' 하며 출제자만 탓하고 있었지요. 흐흐,, 새벽 내내 삽질하다가 결국 둘다 쓰러져서 잤습니다. 그리고 둘째 날 오후 12시 정도에 추가힌트로 XOR 테이블이 공개되었습니다.

추가 힌트
1. ROT 테이블 배열(키보드 배열)

```
('2', '3', '4', '5', '6', '7', '8', '9', '0',
 'Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P',
 'A', 'S', 'D', 'F', 'G', 'H', 'J', 'K', 'L', 'Z',
 'X', 'C', 'V', 'B', 'N', 'M', 'q', 'w', 'e', 'r',
 't', 'y', 'u', 'i', 'o', 'p', 'a', 's', 'd', 'f',
 'g', 'h', 'j', 'k', 'l', 'z', 'x', 'c', 'v', 'b',
 'n', 'm', ' ' );
```

2. 위의 테이블에 없는 문자열은 ROT 안함

헐,, QWERTY 순서일 줄이야! 생각지도 못했던 ROT 테이블 규칙에 혀를 내두르면서 새벽 동안 작성했던 자동화 코드를 샤샥 수정해서 바로 돌려보았습니다.

결과는?
51 번째 rotation 에서 다음과 같은 ROT 테이블을 발견할 수 있었습니다.

```
51--
Plain Text:        [ CrackIsArt! ]
Text(hex)           [ 43 72 61 63 6B 49 73 41 72 74 21 ]

After ROT(str): [ ugc73LvCgh! ]
After ROT(hex): [ 75 67 63 37 33 4C 76 43 67 68 21 ]
After ROT(bin): [ 01110101 01100111 01100011 00110111 00110011 01001100 01110110
 01000011 01100111 01101000 00100001 ]

XOR table(str): [ CrackIsArt ]
XOR table(hex): [ 00 43 72 61 63 6B 49 73 41 72 74 ]
XOR table(bin): [ ]

Encrypted(hex): [ 75 24 11 56 50 27 3F 30 26 1A 55 ]
Encrypted(bin): [ 01110101 00100100 00010001 01010110 01010000 00100111 00111111
 00110000 00100110 00011010 01010101 ]
--
```

오호라! Plain-Text 의 두 번째 문자부터 마지막 문자를 제외한 값이 바로 ROT 테이블이었습니다!
위에서 사용된 코드는 다음과 같습니다.

```
#include <stdio.h>
#include <string.h>

char rottable[] = "234567890QWERTYUIOPASDFGHJKLZXCVBNMqwertyuiopasdfghjklzxcvbnm ";

void printbin(char *buffer)
{
        int i;

        for(i = 0; i < strlen(buffer); i++)
        {
```

```
if((buffer[i] & 128) == 128)
        printf("1");
else
        printf("0");

if((buffer[i] & 64) == 64)
        printf("1");
else
        printf("0");

if((buffer[i] & 32) == 32)
        printf("1");
else
        printf("0");

if((buffer[i] & 16) == 16)
        printf("1");
else
        printf("0");

if((buffer[i] & 8) == 8)
        printf("1");
else
        printf("0");

if((buffer[i] & 4) == 4)
        printf("1");
else
        printf("0");

if((buffer[i] & 2) == 2)
        printf("1");
else
        printf("0");

if((buffer[i] & 1) == 1)
        printf("1 ");
else
        printf("0 ");
```

```
	}
}

void analyze(char *str, char *str2)
{
	int i, j, offset;
	int cnt = 0, LEN;
	char *pstr, *str1;

	str1 = (char *)malloc(strlen(str) + 1);
	LEN = strlen(str);

	for(cnt = 0; cnt < 64; cnt++)
	{
		strncpy(str1, str, strlen(str));

		printf("%d--₩n",cnt);
		printf("Plain Text:₩t[ %s ]₩nText(hex)₩t[ ", str1);
		for(i = 0; i < LEN; i++)
			printf("%02X ", str1[i]);
		printf("]₩n");

		for(i = 0; i < LEN; i++)
		{
			for(j = 0; j < strlen(rottable); j++)
			{
				if(str1[i] == '!' || str1[i] == '1')
				{
					break;
				}
				if(str1[i] == rottable[j])
				{
					offset = j - cnt;
					if(offset < 0)
						offset = strlen(rottable) + offset;
					str1[i] = rottable[offset];
					break;
				}
			}
```

```
                }

                printf("₩nAfter ROT(str):₩t[ %s ]₩nAfter ROT(hex):₩t[ ", str1);
                for(i = 0; i < LEN; i++)
                        printf("%02X ", str1[i]);

                printf("]₩nAfter ROT(bin):₩t[ ");
                printbin(str1);

                for(i = 0; i < LEN; i++) {
                        str1[i] ^= str2[i];
                }
                printf("]₩n₩nXOR table(str):₩t[ ");
                for(i = 0; i < LEN; i++) {
                        printf("%c", str1[i]);
                }
                printf(" ]₩nXOR table(hex):₩t[ ");
                for(i = 0; i < LEN; i++) {
                        printf("%02X ", str1[i]);
                }
                printf("]₩nXOR table(bin):₩t[ ");
                printbin(str1);

                printf("]₩n₩nEncrypted(hex):₩t[ ");
                for(i = 0; i < LEN; i++) {
                        printf("%02X ", str2[i]);
                }
                printf("]₩nEncrypted(bin):₩t[ ");
                printbin(str2);

                printf("]₩n--₩n");
        }
}

int main()
{
        char p_str1[] = "CrackIsArt!";
        char e_str1[] = "₩x75₩x24₩x11₩x56₩x50₩x27₩x3f₩x30₩x26₩x1a₩x55";
```

```
	char p_str2[] = "Hacking Defense";
	char e_str2[] =
	"₩x71₩x2b₩x56₩x50₩x07₩x59₩x03₩x30₩x62₩x22₩x0b₩x00₩x55₩x18₩x15";

	char p_str3[] = "Year 2007 Month 5";
	char e_str3[] =
	"₩x4a₩x3f₩x06₩x06₩x25₩x65₩x73₩x71₩x79₩x60₩x41₩x37₩x5f₩x06₩x54₩x3f₩x79";

	analyze(p_str1, e_str1);
	printf("₩n");
	//analyze(p_str2, e_str2);
	printf("₩n");
	//analyze(p_str3, e_str3);
	printf("₩n");

	return 0;
}
```

이를 통해 ROT 테이블과 로테이션 값, 그리고 XOR 테이블의 규칙을 모두 알아낼 수 있었습니다. 여기서 복호화 프로그램을 작성하는 것은 쉬운 일이죠. 결국 다음과 같은 코드를 작성하여 다음 스테이지의 패스워드를 알아낼 수 있었습니다.

```
[graylynx@redhat72 kisa]$ cat level5.c
#include <stdio.h>
#include <string.h>

const char rotnum = 11;
const char rottable[] =
"234567890QWERTYUIOPASDFGHJKLZXCVBNMqwertyuiopasdfghjklzxcvbnm ";

char rot(char src)
{
	int i, j, offset;

	for(i = 0; i < strlen(rottable); i++) {
		if(src == rottable[i]) {
			offset = i - rotnum;
			if(offset < 0)
				offset += strlen(rottable);
```

```
                        return rottable[offset];
                }
        }
        return src;
}

char* decode(char *enctext)
{
        int i;
        char *buffer;

        buffer = (char *)malloc(strlen(enctext) + 1);

        buffer[0] = rot(enctext[0]);
        for(i = 1; i < strlen(enctext); i++)
                buffer[i] = rot(enctext[i] ^ buffer[i - 1]);

        buffer[strlen(buffer) + 1] = 0x00;
        return buffer;
}

int main(int argc, char *argv[])
{
        if(argc == 2)
                printf("Plain-Text: [ %s ]₩n", decode(argv[1]));

        return 0;
}
[graylynx@redhat72 kisa]$ ./level5 `perl -e 'print "₩x65₩x07₩x1f₩x10₩x16₩x51₩x1d₩x69₩x2a₩x19₩x3f₩x06₩x12₩x13₩x1b₩x79₩x68₩x73₩x71₩x79₩x76₩x69₩x74₩x62₩x72₩x64"'`
Plain-Text: [ KISA H6_CONTEST-2007050304 ]
[graylynx@redhat72 kisa]$
```

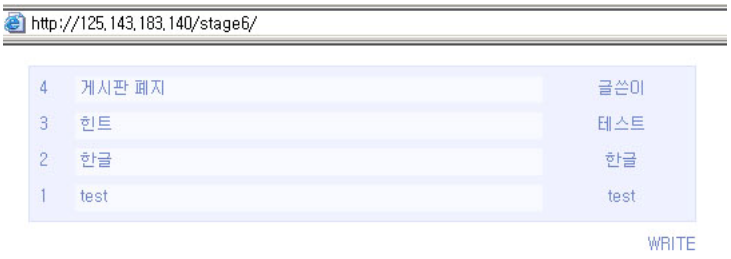
음,, 이 문제는 제일 오래 풀었던 문제이고, 나름대로 그 규칙성에 놀라면서 꽤 재미있게 풀었던 문제지만 약간의 아쉬움을 감출 수 가 없습니다. 삽질의 여지가 너무 많았던 게 이 문제의 아쉬운 점입니다. 사실 XOR 테이블을 유추하는 코드에서도 처음에는 %c 루프가 아닌 %s 로 바로 출력하다 보니 첫 글자가 0x00 인 XOR 테이블을 놓쳐서 엄청난 삽질을 하게 되는 계기가 되었습니다. T_T 또한 ROT 테이블을 모르면 XOR 규칙을 알아내기 힘들고, XOR 규칙을 모르면 ROT 테이

블을 알아내기 힘든 문제의 특성 때문에 두 가지의 미지수에 대한 가정과 가능성이 너무 많았습니다. 위에 설명에서는 한번에 바로 알아낸 것 같지만 저런 결과를 뽑기까지에는 이 글에는 생략된 수많은 추측과 가정, 경우의 수에 대한 삽질이 있었습니다.

스테이지6

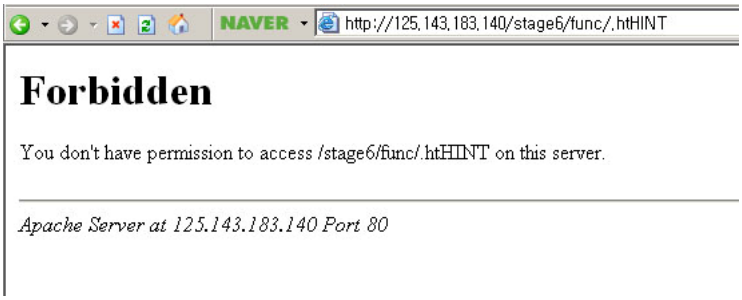
현재 단계는 5 입니다.
6번문제 : 이전 보안 문제를 적절하게 패치하고 의기양양해 있던 웹보안관리자 마루치는 회사 웹서버가 침해 사고를 당했다는 천둥벼락같은 소식을 들었다. 여러분이 이 취약점을 찾아 다음단계의 패스워드를 획득하라.
ID: stage6, PASSWORD: 획득한 패스워드

스테이지6의 주소로 아이디와 패스워드를 치고 들어가면 다음과 같은 화면이 나옵니다.

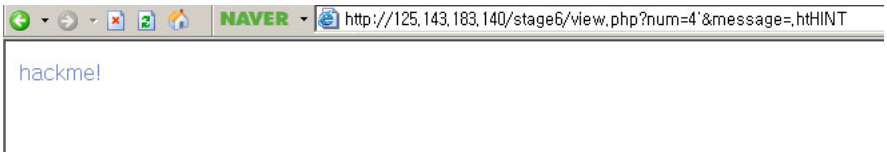


3번 힌트 게시물이 의심스럽습니다.

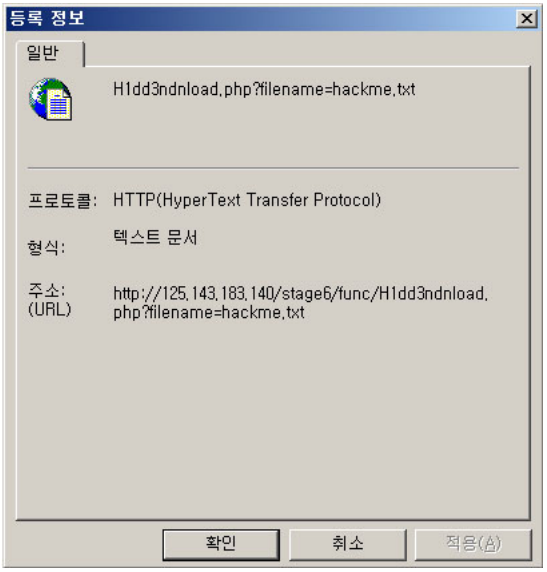
HINT 링크를 눌러보면,



위와 같이 접근이 불가합니다.
그런데 message=error 이 살짝 의심스럽네요. 역시 수 차례의 삽질 끝에, num=4 까지를 ' 로 묶고, message=.htHINT 로 조작하고 view.php 를 호출하니 아래와 같이 화면이 나왔습니다.

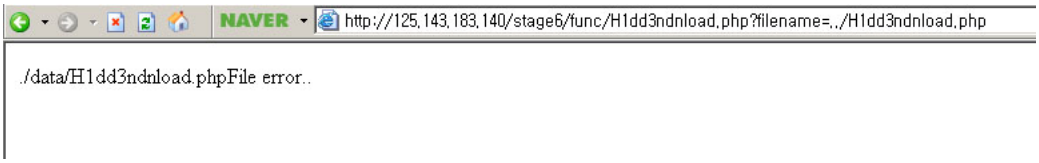


hackme! 링크를 보면,

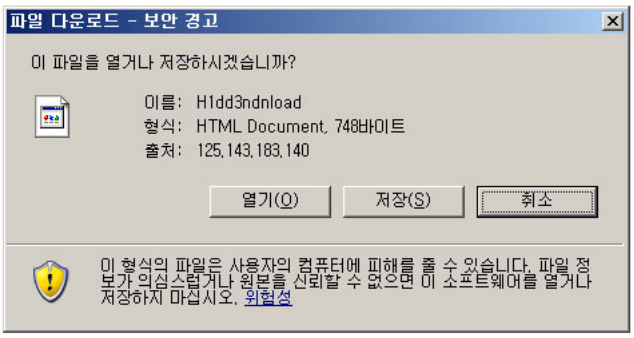


/func/H1dd3ndnload.php?filename=hackme.txt 에 링크되어 있습니다.

한눈에 다운로드 취약점이라는 생각이 들어서, 이리저리 삽질한 결과 hackme.txt 는 /func/data 에 존재한다는 것을 알아냈습니다. 별달리 받을 파일이 없으므로, H1dd3ndnload.php 를 분석하기 위해 다운로드를 시도했습니다.



아무래도 ../ 이 필터링 되는 것 같습니다. ..../// 를 입력해 봤습니다.



다운로드 받은 뒤, 파일을 열어보니,,

```php
?php
  /*
      download.php 와 같은 폴더에 존재하는 dnP455W0rd.php 파일에
      다음 레벨 Password가 존재합니다.
  */
      @extract($HTTP_GET_VARS);
      @extract($HTTP_POST_VARS);

  $filename = trim($filename);
  if ( $filename == "" ) { die("Invalid filename!"); }

  // 잘못된 수정으로 인해 download 취약점이 여전히 존재하죠?
  $filename = str_replace("../", "", $filename);
```

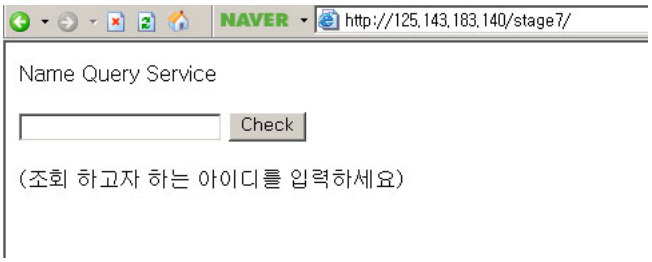
dnP455W0rd.php 파일을 친절하게 알려주네요. 또한 예상했던 대로, 밑에는 ../ 를 필터링 하는 부분이 존재했습니다. 해당하는 파일을 받아서 열면 다음 스테이지의 패스워드가 보입니다.

```php
<?php
    /*
    PASSWORD: M1551ON_P4552D!!
    */
```

스테이지7

현재 단계는 6 입니다.
7번문제 : 해당 페이지에 존재하는 보안 취약점을 찾아 이를 이용하여 다음단계의 패스워드를 획득하라.
ID: stage7, PASSWORD: 획득한 패스워드

스테이지7의 주소로 아이디와 패스워드를 치고 들어가면 다음과 같은 화면이 나옵니다.



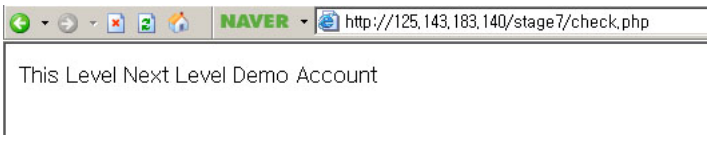
저희가 스테이지7을 접했을 때가 상대적으로 다른 팀에 비해 늦었는지, 이미 힌트가 공개되어 있습니다. 헐--;

STAGE7 힌트

XPath Injection

XPath Injection 은 XML 에서 데이터를 조회할 때 SQL Injection 과 마찬가지로 도중에 ' 나 " 등의 문자를 통해 특정코드를 삽입 시킬 수 있는 취약점 입니다. 자세한 내용은 http://www.packetstormsecurity.org/papers/bypass/Blind_XPath_Injection_20040518.pdf 를 참조해 주세요.

XML 에 대한 지식이 전무했기 때문에 저희도 막 Google 에서 자료를 찾고 문서를 찾고 하느라 정신이 없었습니다. Example code 들을 하나씩 대입해보다가 ' or 1=1 or '1'='1 에서 Injection 이 가능하다는 것을 알아냈습니다.



위 코드로 Injection 을 시도하면 위와 같은 문자열들이 나옵니다. 음 무슨 말이지는 잘 모르겠고 소스 보기를 했을 때 문자들이 다 붙어있는 것으로 보아 각각의 데이터에서 따로 뽑아온 값들인 듯 했습니다. 또한 역시 삽질을 통해 stage7 을 입력하면 This Level 이라는 메시지가,, stage8 을 입력하면 Next Level 이라는 메시지가,, demo 를 입력하면 Demo Account 라는 메시지가 출력된

다는 것만 알아냈습니다.

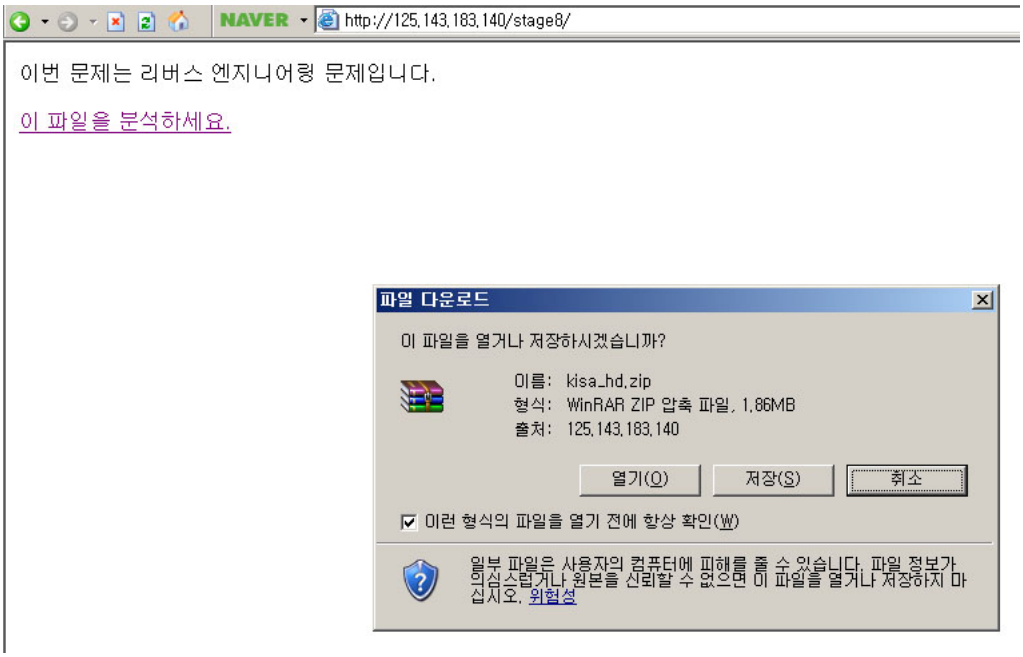
하지만 분명 저 값들이 다는 아닌 듯 했습니다. 무엇인가 다른 곳에 저장되어있는 값을 출력하려면 함수가 중간에 들어가야 하는데, XML 서적에서 substring() 함수가 비슷한 기능을 하는 것을 보고 다음과 같이 공격하여 다음 레벨의 패스워드를 볼 수 있었습니다.

' or substring(//child::node()) or 1=1 or '1'='1

스테이지8

현재 단계는 7 입니다.
8번문제 : 평소 업무시간에 메신저를 즐겼던 하모씨는 최근 사내에 필수 소프트웨어가 설치된 이후로 메신저를 사용할 수 없었다. 해당 프로그램을 크랙하여 그 프로그램의 패스워드를 획득하라.
ID: stage8, PASSWORD: 획득한 패스워드

스테이지7의 주소로 아이디와 패스워드를 치고 들어가면 다음과 같은 화면이 나옵니다.



사실 저희는 대회기간 동안 이 문제를 풀지 못했습니다. 윈도우 리버싱에 대한 지식이 너무 부족해서 결국 언팩킹도 하지 못하고 그저 빨리 끝내기만 바라면서 문서정리하고 있었습니다. ^^; 대회 끝나고 언팩킹에 대해 공부하고 나서야 겨우 풀 수 있었습니다.

언팩킹을 글로 설명하기에는 제 능력이 부족하므로, 실제 언팩킹하는 장면을 녹화하였습니다.
(첨부파일: kisa_hd_unpacking.swf)

언팩킹을 하고 난 다음 ResourceHacker 나 WinHex 등의 프로그램으로 열어보면 BMP 헤더의 이미지파일 두개가 있습니다. 에디터를 이용해서 이를 각각 파일로 뽑아냅니다.
(첨부파일: kisa_hd_img1.bmp, kisa_hd_img2.bmp)

이 두 파일의 내용 중 서로 다른 부분이 있는데, 그 부분을 뽑아내면 다음과 같습니다.
(첨부파일 kisa_hd_diff.txt)

각각의 부분에서 한쪽 파일은 0 이고 다른 한쪽 파일은 1 입니다. 여기서 값이 다른 영역의 마지막 주소값 AAB2D – 처음 주소값 AAA36 을 하면 십진수 247 이 나옵니다. 즉, 값이 다른 영역의 크기는 248 이고, 이것이 비트라고 가정한다면 248 / 8 = 31 이 나오게 됩니다.

서로 값이 다른 부분은 1, 이 부분을 제외한 부분은 0 이라 가정하고 일렬로 나열한 뒤, 8개씩 쪼개어 보면 다음과 같은 값이 나옵니다.

11111000 11000011 11011100 11001111 11011000 10001101 11011001 10001010 11101011
11000100 11001110 10001010 11101101 11000011 11000100 11001111 11001111 11011000
11000011 11000100 11001101 11110101 11100011 11111001 11110101 11101000 11101011
11111001 11101111 11010100 10001011

첫 비트가 모두 1인 것으로 보아 아스키 값에 특정 연산이 되어있음을 생각해볼 수 있습니다. 31 글자의 아스키 값이라고 생각하고 다음과 같은 약간의 Brute-force 를 돌려서 답을 얻어낼 수 있었습니다.

```
[graylynx@redhat9 kisa_hd]$ cat kisa_hd.c
#include <stdio.h>

int main()
{
        int i, j;
        unsigned char password[] =
"₩xf8₩xc3₩xdc₩xcf₩xd8₩x8d₩xd9₩x8a₩xeb₩xc4₩xce₩x8a₩xed₩xc3₩xc4₩xcf₩xcf₩xd8₩xc3₩xc
4₩xcd₩xf5₩xe3₩xf9₩xf5₩xe8₩xeb₩xf9₩xef₩xd4₩x8b";
```

```
        for(i = 0; i < 256; i++) {
                printf("%02X: ", i);
                for(j = 0; j < 31; j++) {
                        printf("%c", password[j] ^ i);
                }
                printf("₩n");
        }

        return 0;
}
[graylynx@redhat9 kisa_hd]$ gcc -o kisa_hd kisa_hd.c
[graylynx@redhat9 kisa_hd]$ ./kisa_hd > a
[graylynx@redhat9 kisa_hd]$ strings a
..
..
A9: Qjufq$p#Bmg#Djmffqjmd₩JP₩ABPF}"
AA: River's And Gineering_IS_BASE~!
AB: Shwds&r!@oe!Fhoddshof^HR^C@RD
..
..
```

빙고~ 답은 River's And Gineering_IS_BASE~! 입니다.

이상으로 모든 풀이를 마치겠습니다.
부족한 글 끝까지 읽어주셔서 감사합니다.