|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | |  | | KITRI 모의해킹 28기 | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | 작성: 보안팀 | | |  | |
|  | buffer overflow attack(rtl) | | | | | | |  |
|  | | Return To Libc attack | | | |  | | |
|  | |  | | | |  | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |

|  |
| --- |
| 1. 개요(공격개념/원리/취약점/공격 시 예상피해 등) |
|  |
| 현재 오버플로우 공격은 각종 메모리 보호 기법, ASLR기법 등등 각종 보호기법들로 인하여 공격이 거의 불가능하다. |
| 따라서 해당하는 보호기법들을 우회하기 위하여 등장한 것이 RTL 공격이다. |
| 이 공격을 활용하면, 리눅스 메모리 보호 기법 중의 하나인 NX bit를 우회하여 공격을 수행하는 것이 가능하다. |
| NX bit란 stack 상에 올라오는 여러 함수나, 코드들의 실행 권한을 제거함으로써, stack의 RET주소에 예상치 못한 |
| 쉘 코드 등이 덮어씌워져 오버플로우 공격이 발생하는 것을 방지하기 위한 메모리 보호 기법이다. |
| RTL 공격은 return to libc의 약자로 메모리에 이미 적재되어 있는 공유 라이브러리를 이용한 공격으로, 우리가 실행한 |
| 컴파일한 코드에 원하는 함수나 쉘 코드 등이 없어도, 공유 라이브러리에서 원하는 함수의 사용이 가능하다. |
| 공격자 입장에서는 root권한으로 우선 접속한 후에 /bin 하위의 여러 실행파일을 자신이 만든 실행파일로 교체하여 해 |
| 당 파일을 임의로 실행시킬 수 있고, 정상적인 실행파일들을 자기맘대로 실행시키는 것이 가능해진다. |
| 왜냐하면, /bin/하위의 파일들은 메모리에 이미 적재되어 있는 공유 라이브러리 이기 때문이다. |
| 최종적으로 정리를 해보자면, 웬만한 버퍼오버플로우 공격은 현존하는 서버에서 거의 다 막혀있다.  그러나, RTL공격 기법을 활용하면, 버퍼오버플로우 공격이 현존하는 서버에서도 가능하다.  매우 심각한 취약점이라는 의미이다. |
|  |

|  |
| --- |
|  |
| 2. 공격 결과(해당 취약점은 centos 7, 8 에서도 현존) |
| 1) 우선적으로 코드분석, 메모리 분석, 스택구조 파악, 어셈블리어로 디컴파일, 파일 및 함수 위치 찾기 등을 하기 위하여  gdb-peda 를 설치해준다.   gdb-peda는 gdb의 확장버전으로 gdb에 비하여 좀 더 자세하게 파일을 디버깅할 수 있는 기능을 제공하고, 파일 및 함수의 위치를 쉽게 찾을 수 있는 것이 장점이다. 아래는 gdb-peda 설치 방법이다. |
| sudo yum install git  // git 다운로드  git clone https://github.com/longld/peda.git ~/peda  // github에서 peda.git 가져와서 현재디렉터리/peda에 복사  echo "source ~/peda/peda.py" >> ~./gdbinit  // 기존 gdb실행경로인 gdbinit에 위에서 /peda에 넣어둔 peda.py 덮어씌움.  echo "Done! debug your program with gdb and enjoy"  // gdb-peda 설치가 완료되었습니다!! |
| 2) 인제 rtl 공격을 위해서 간단한 코드를 작성해보도록 하자. 아래는 예시 코드이다.  프로그래머의 입장에서는 프로그래밍시에 아래와 같은 식의 코드를 절대 작성해서는 안된다. |
| #include <stdio.h>  int main(){  char buf[100];  read(0,buf,200); //하단에 read함수의 원형을 설명해보겠다.  printf("%s\n",buf);  } |
| **<read 함수 원형 설명>**  #include <unistd.h>  ssize\_t read(int fd, void \*buf, size\_t nbytes);  int fd --> 읽을 파일의 파일 지시자를 의미한다.  void \*buf --> 읽어들인 데이터를 저장할 버퍼(배열)  size\_t nbytes --> 읽어들일 데이터의 최대 길이(buf의 길이보다 길어선 안된다. --> 오버플로우 발생 가능)  반환값 --> 읽어들인 데이터의 길이를 반환해준다. 무조건 nbytes가 리턴되는 것은 아니고, 중간에 파일의 끝을 만나게 되면, 거기까지만 읽어서 반환해준다.  따라서 위의 코드처럼 작성하게 되면, buf의 사이즈는 100을 선언해주었는데, 읽어들일 데이터의 최대 길이는 200까지 받아들임으로써 버퍼오버플로우 공격이 발생할 수 있다. |
| 3) 위에 작성한 rtl.c파일을 gcc로 컴파일 하기 전에 오류를 잡아주기 위하여 하단의 라이브러리를 설치해준다. |
| yum install glibc-devel.i686 libgcc.i686 libstdc++-devel.i686 |
| 4) 그러고 나서 컴파일을 실시해준다.  gcc -m32 -mpreferred-stack-boundary=2 -fno-stack-protector -fno-pic --no-pie -o rtl rtl.c |
| 위와 같이 컴파일 해주는 이유는 각종 메모리 보호 기법 및 ASLR기법을 해제하는 것이 가능하다. 다만, nx비트 보호기법 등은 위의 방법으로 해제하지 못하여 rtl공격을 시도하는 것이다. |
| 5) 위의 명령어로 컴파일을 해주었다면, gdb-peda도구를 사용하여 main함수를 어셈블리어로 변환한다. 하단에 그 결과를 첨부하였다. (명령어 disas main) |
|  |
| main 함수 disassemble 한 결과 함수가 실행되면서 메모리상에 어떠한 구조로 배치되고, 어느정도의 메모리 크기를 가지는지 등의 정보를 확인이 가능하다. 위를 살펴보면, +14라인에 ebp-0x64로 인해서 버퍼를 100사이즈 만큼 할당한다는 것을 알 수 있다. 통상 함수가 실행될때, 버퍼사이즈 후에 4bytes 만큼 sfp를 할당해 준 후에 ret 영역이 할당된다. |
| 6) 오버플로우를 발생시키기 위해서는 104bytes를 쓰레기 값으로 채워준 후에 ret영역에 접근하여 ret영역을 다른 값으로 덮어씌워서 발생시킨다. 하단에 자세한 설명이 있다. |
| ret영역에 system 함수를 호출하여, /bin/sh파일을 강제로 실행하는 오버플로우 공격을 하고자 한다.  해당 공격을 하기 위하여 우선, system 함수의 주소와 /bin/sh파일의 주소를 알아내야 한다. 해당 함수와 파일의 주소를 알기 위하여 gdb-peda를 이용한다. 해당 도구를 이용하여 main함수에 breakpoint를 걸어준 후에 system함수의 주소부터 찾고, /bin/sh 파일의 주소를 찾아내야 한다. 하단은 일련의 과정을 캡쳐한 것이다. |
|  |
| 위의 결과에서 볼 수 있듯이, system함수의 주소는 0xf7e35fa0으로 확인하였다.  /bin/sh파일의 주소는 0xf7f77115로 확인하였다. |
| 7) 최종 공격 페이로드 |
| (python -c 'print "A"\*104 + "\xa0\x5f\xe3\xf7" + "AAAA" + "\x15\x71\xf7\xf7"'; cat) | ./rtl |
| 쓰레기값인 A를 버퍼에 채워준 후에, ret값을 system함수와, /bin/sh파일의 주소로 채워준다.  하단에 결과를 첨부하였다. |
|  |
|  |
| 위를 보면 알 수 있듯이, 전혀 다른 함수인 /bin/sh가 실행된 것을 확인할 수 있다. |
|  |
|  |

|  |
| --- |
|  |
| 4. 보안진단 및 대책 |
| 1) ASLR기법을 적용해주도록 한다. ASLR기법은 프로그램이 실행될 시에 배치받는 메모리 주소를 항상 랜덤하게 배치하여, RET주소를 알아내기 힘들게 만드는 기법이다. 오버플로우 공격을 효과적으로 막는다. |
| 2) root로의 접근을 확실히 차단한다. rtl 공격은 여러가지 방지 대책을 사용한다고 해도, 라이브러리 자체를 탈취하는 공격이기 때문에, root계정을 확실히 보호하도록 한다. |
|  |
| 3) ASCII-Armor란 방법을 사용한다. 이는 공격자가 라이브러리를 호출하는 buffer overflow공격을 해도 null 바이트가 삽입된 주소로는 접근하는 것이 불가능하다. |
|  |
|  |

|  |
| --- |
|  |
| 5. 보안대책 적용시 공격결과 (최종 매뉴얼에 포함) |
| 해당 공격은 보안대책을 적용하여도, 프로그래머가 실수를 범할시에 충분히 다시 일어날 수 있는 공격이다. |
|  |
|  |
|  |
|  |
|  |