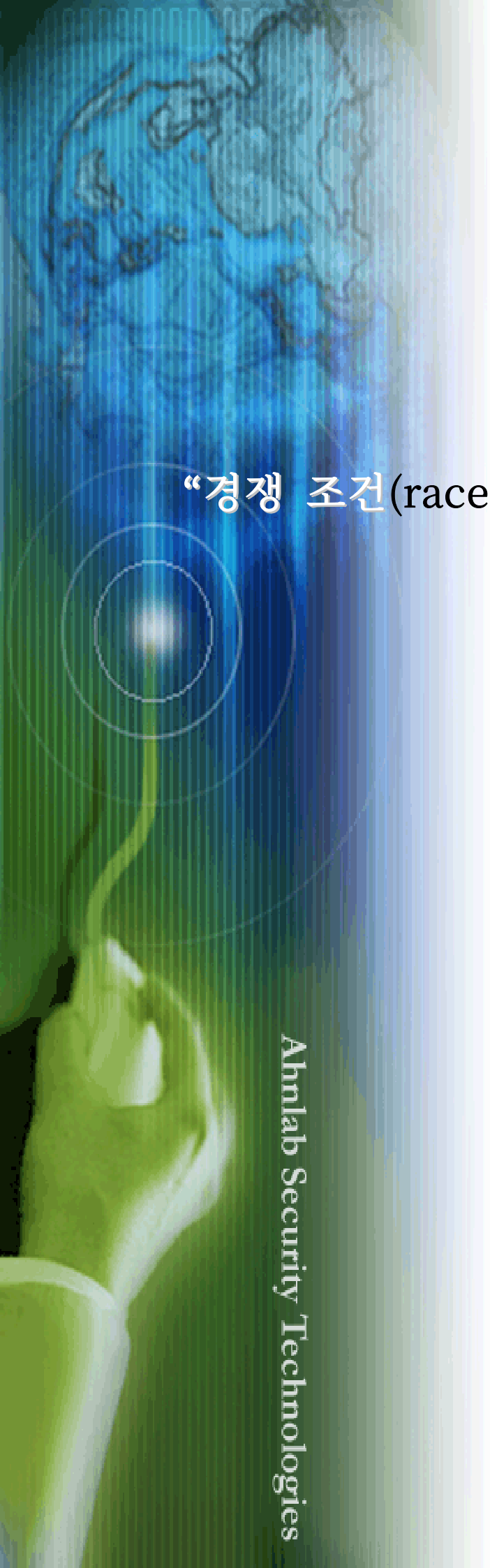

Technical Report

“경쟁 조건(race condition)”을 이용한 해킹

www.ahnlab.com

안병선(asec@ahnlab.com)
안철수연구소 시큐리티대응센터(ASEC)

목 차

1. 머리말 3

2. 경쟁 조건이란? 3

3. 심볼릭 링크(symbolic link) 4

4. 왜 "경쟁 조건"인가? 5

5. 실제 사례 5

5.1. SunOS sendmail(/bin/mail) 5

5.2. Solaris 2.x ps(/usr/bin/ps) 6

5.3. 그 외 6

6. 대책 7

7. 결론 7

1. 머리말

유닉스는 여러 개의 프로세스(process)를 지원하는 운영체제이며. 여러 사람으로 하여금 정보를 공유할 수 있도록 만들어져 있다. 그러나 이 훌륭한 운영체제에서도 프로그래밍 상의 실수로, 사용자 편의를 위해 작성되었던 프로그램이 거꾸로 시스템을 공격하는 수단이 되는 경우가 발생한다. 일반 사용자가 프로그램의 버그를 이용하여 갑자기 관리자(super user)의 권한을 획득하고 비정상적인 명령을 수행하는 사태가 발생하는 것이다.

해커들이 로컬 호스트(local host)에서 관리자의 권한을 획득하는 방법으로 유사이래 수많은 기법들이 알려졌으나, 여기서는 전통적인 해킹 기법 중 하나인 "경쟁 조건(race condition)"을 이용한 해킹에 대해 기술하고자 한다.

2. 경쟁 조건이란?

경쟁 조건이란 두 프로세스간에 리소스(resource)를 사용하기 위해서 다투는 과정으로, 본래 해킹 기법을 칭하는 것이 아니었다. 아래 익히 알려진 유명한 코드를 통해 경쟁 조건의 원래 의미를 알아보자.

```
/*----- race condition EXAMPLE ------*/
void main(void)
{
      int childpid;
      static int a;

      if((childpid = fork()) > 0) { /* 부모 프로세스 */
      for(; a < 100; a++) {
            If (a % 2 == 0) { printf("O"); }
            }
      exit(0);
      }
      else { /* 자식 프로세스 */
            for(; a < 100; a++) {
                  if (a % 2 == 1) { printf("X"); }
                  }
            exit(0);
      }
}
/*-------------------------------*/
```

Copyright ⓒ2002 by Ahnlab, Inc. All rights reserved.

fork()라는 함수는 동일한 작업을 하는 프로세스를 하나 더 띄우는 함수이다. fork()를 호출할 때, 그 결과값은 새로 만들어진 프로세스의 번호가 되는데, 이로써 프로세스를 구분한다. 그러면 위의 주석에 보이듯이 원래의 프로세스는 O라는 문자를 50번 출력하고, 자식 프로세스는 X를 50번 출력한다. 이때, 유닉스는 시분할 정책을 사용하므로, 프로세스마다 시간을 각각 할당하게 되며, 이때 어느 쪽에 시간을 더 배당하느냐가 문제가 된다.

상식적으로 OXOXOX… 이렇게 출력되리라고 예측할 수 있지만, 실제 실행해보면 OOOXOXOOOXXXXXXOOXOXX… 와 같이 비주기적으로 출력이 되는 것을 알 수 있다. 이것이 바로 경쟁조건에 의한 결과이다. (단, 결과는 테스트 환경에 따라 가변적이다)

3. 심볼릭 링크(symbolic link)

유닉스에는 심볼릭 링크라는 멋진 기능이 존재한다. 일반적으로 어떤 파일을 접근하고자 할 때, 그 전체 경로명를 입력하는 것은 상당히 귀찮은 일이다. 물론 PATH라는 좋은 환경변수가 존재하기는 하나, 조금 불충분하다고 생각할 수 있다. 심볼릭 링크는 해당 경로명으로 바로 연결해 주는 기능으로써, 예를 드는 것이 이해가 빠를 것이다.

준상이 /home/project/AI/SIM 이라는 디렉토리에서 연구를 수행한다고 하고, 홈 디렉토리는 /user/junsang이라고 하자. 준상은 연구를 할 때 항상 cd /home/project/AI/SIM과 같은 명령어를 입력하여 디렉토리를 변경해야 한다. 이를 매일같이 한다면 무척 귀찮을 것이다. 이런 경우 심볼릭 링크를 사용하면 간단히 해결된다. 준상의 홈 디렉토리에서 'ln -s /usr/project/AI/SIM .link' 라고 입력하면, ".link"라는 링크 파일이 만들어 진다. 이후에는 cd .link라고만 입력하면 .link는 /home/project/AI/SIM으로 연결되어 있으므로, 자동으로 연구 수행을 위한 디렉토리로 이동하게 되는 것이다. 상당히 편리한 기능이다.

심볼릭 링크는 위와 같이 편리하게 이용할 수 있으나, 위험성을 내포하고 있다. 유닉스에 존재하는 일반적인 버그에 수시로 나타나는 것이 바로 이 심볼릭 링크인데, 그 이유를 살펴보도록 하자.

Alpha란 프로그램이 hello라는 파일을 열어서, 그곳에 무언가를 써준다고 가정한다. 이때 만약 당신이 권한(permission)이 된다면, hello라는 파일을 지우고, 심볼릭 링크로 'ln -s ~xxx/.rhosts hello' 라고 해놓으면 어떻게 되겠는가? Alpha 프로그램이 실행되면 hello라는 파일을 여는데, hello는 ~xxx/.rhosts라는 파일에 링크되어 있으므로, Alpha는 ~xxx/.rhosts를 열어서 쓰게 된다. (물론 개발자가 신중히 프로그램을 작성하면 이런 일을 없앨 수 있다.)

Copyright ⓒ2002 by Ahnlab, Inc. All rights reserved.

전형적인 예제가 elm에 존재하는 autoreply라는 버그였는데, 이는 /tmp에 관리자 소유의 SETUID인 arep.???? 라는 파일을 생성한다. 이때, 생성되는 파일을 없애고 심볼릭 링크를 /.rhosts로 연결시킨다면, 그리고 /.rhosts라는 파일에 "+ +"을 쓴다면 모든 시스템에서 관리자의 권한으로 아무런 제약 없이 로그인 할 수 있게 되는 것이다.

4. 왜 "경쟁 조건"인가?

위와 같다면 왜 경쟁 조건이 등장하게 되었을까? 그냥 링크시켜서 하면 되는 것인데, 왜 프로세스 간의 리소스 경쟁이 나타나는가? 이것은 방어자와 공격자의 싸움에서 나타난 결과라고 할 수 있다.

프로그래머는 드디어 위와 같은 autoreply의 문제점을 인식하고, 새로운 방법을 모색하게 되었다. 그것은 바로 lstat()를 이용하여 먼저 수정하고자 하는 파일의 상태를 검사한 후에, open()시켜서 처리하도록 한 것이다. 이로써 아무런 문제가 없는 듯이 보였다. 그러나, 여기에서 경쟁 조건이 등장하게 된다.

lstat()이 호출된 후 open()이 호출되기까지에는 분명히 시차가 존재한다. 그 시차를 적절히 이용한 것이 바로 경쟁 조건인데, 먼저 경쟁 프로그램을 백그라운드로 돌린다. 경쟁 프로그램은 공격하고자 하는 프로그램이 수정하는 파일을 연속해서 지우고, 링크를 만들고 하는 작업을 반복한다. 이 상태에서 공격하고자 하는 프로그램이 실행된다면 어떻게 되겠는가?

lstat()이 검사한 후, open()이 호출되는 아주 작은 시간의 갭 사이에 심볼릭 링크가 되어 버린다면 lstat()은 무용지물이 되고 만다. 그러나, 그것이 이루어 질지 안 이루어 질지는 미지수이다. 처음에 언급한 것처럼 운영체제가 프로세스에 시간을 어떻게 할당할지는 알 수 없기 때문이다.

이제는 경쟁 조건이라 불리는 해킹 기법이 왜 등장하게 되었는지 알게 되었으리라고 생각한다. 대표적인 실례를 몇 가지 살펴보도록 하자.

5. 실제 사례

5.1. SunOS sendmail(/bin/mail)

SunOS의 sendmail은 외부로부터 온 전자 우편을 받아서 그것을 /var/spool/mail/$USER 위치에 넣

Copyright ⓒ2002 by Ahnlab, Inc. All rights reserved.

는다. $USER라는 것은 해당 사용자를 의미이다. 문제는 그 사용자의 권한으로 넣어주기 위해서 (chown) 관리자 권한으로 프로그램이 돌아가야 한다는 것이다.

/var/spool/mail의 디렉토리의 권한이 777로 설정되었다고 가정한다. 이때, /etc/passwd에는 존재하고, /var/spool/mail에는 존재하지 않는 사용자가 있다고 생각해보자. ftp, sundiag등 전자 우편이 배달될 일이 없는 아이디가 분명히 있을 것이다. 이런 경우 경쟁 조건으로 시스템에 취약점이 생긴다.

ftp란 아이디를 가정해 보자. 먼저 경쟁 프로그램을 만든다. 그것은 /var/spool/mail/ftp를 지웠다가 그 파일을 'ln -s /.rhosts /var/spool/mail/ftp'로 링크하는 것을 반복하는 것이라고 생각한다. 이제 경쟁 프로그램을 실행한 후에, ftp 아이디로 아주 적절한 내용를 넣어서 전자우편을 보내는 것이다!

sendmail은 lstat()를 통해 심볼릭 링크를 판별한다. 그러므로, 성공 확률은 반반이며 전자우편의 내용이 /.rhosts에 들어가거나, 파일이 없었던 경우라면 ftp라는 파일이 생성되는 것으로 마무리 될 것이다.

5.2. Solaris 2.x ps(/usr/bin/ps)

여기부터는 간략히 그 메커니즘만을 설명하기로 한다. Solaris 2.x에 있는 ps는 현재 실행 중인 프로세스들을 나열해 주는데, 커널(kernel)로부터 현재 실행 중인 프로세스들의 정보를 얻어서 임시로 /tmp 디렉토리에 데이터를 보관한다. 이때 ps_data라는 파일을 ps를 실행시킨 사용자의 UID로 생성하고 나서 관리자 소유의 SETUID 파일로 변경한다. ps_data를 생성하고 나서 권한을 변경하기 전에 다른 프로세스가 ps_data를 지우고 /bin/sh로 심볼릭 링크시켰다고 생각해 보자. 아무 것도 모르는 ps 프로세스는 ps_data를 관리자 소유의 SETUID 파일로 바꿀 것이고 결국은 심볼릭 링크된 /bin/sh가 관리자 소유의 SETUID 파일로 바뀌게 되는 것이다. 그러면 어떤 사용자든지 /bin/sh을 수행시킨 사용자는 관리자의 권한을 가지는 쉘(shell)을 얻게 된다.

5.3. 그 외

SunOS의 /usr/ucb/lpr은 프린트할 때 사용되는 응용 프로그램이다. 프린터와 컴퓨터의 속도 차이가 크기 때문에 프린트할 파일을 스풀 영역에 임시로 보관하게 되는데 이때 임시 파일은 1000을 주기로 이름이 반복된다. 임시 파일을 추측할 수 있다면 이 임시 파일을 미리 다른 시스템 파일로 심볼릭 링크 시켜둠으로써 시스템 파일을 수정할 수 있게 된다.

Copyright ⓒ2002 by Ahnlab, Inc. All rights reserved.

6. 대책

경쟁 조건을 이용한 공격은 특정한 조건이 충족되어야 성공할 수 있다는 특징을 가지고 있다. 따라서 이 조건을 원천적으로 봉쇄한다면 경쟁 조건을 이용한 공격은 완벽하게 막을 수 있다. 여러 가지 대책 중에 간단히 두 가지만 소개한다.

- 첫째, 임시 파일을 만들지 않는다.

근래에 시스템이 발전하면서, 어느 정도는 메모리 내에서 처리가 가능하다. 임시 파일을 만드는 것은 경쟁 조건의 원인이 된다. 만일 굳이 임시 파일을 생성시킬 경우에는 추측이 불가능하도록 해야 한다.

- 둘째, create와 open의 구분을 확실히 한다.

경쟁 조건을 이용한 공격이 가능한 이유 중에 하나가, 두 시스템 콜이 수행되는 사이에 아무런 변화가 없을 것이라 가정하고 프로그램을 작성하기 때문이다. 이를 방지하기 위해서는 프로그래밍 시에 많은 주의를 기울여야 한다. C언어로 프로그램을 작성할 경우 open("file", O_CREATE | O_EXCL) 와 같이 파일이 존재하지 않은 경우에만 생성하도록 해야 한다. 그리고 링크 정보를 검사하고 나서 파일을 연 후에도 파일이 제대로 열렸는지 다시 한 번 확인해야 한다.

7. 결론

지금까지 경쟁 조건에 대해 다루었다. 유닉스 시스템에서도 이렇듯 시스템 프로그램에 버그가 존재할 수 있으며, 여기에 적극 대처하는 것이 필요하다.

Copyright ⓒ2002 by Ahnlab, Inc. All rights reserved.