Machine Translated by Google

환경변수 및 Set-UID 프로그램 연구실

(CSI 424/524 - 2023년 가을)

저작권 © 2006 - 2016 Wenliang Du.
이 저작물은 Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License에 따라 라이센스가 부여됩니다. 자료를 리믹스, 변형 또는 제작하는 경우, 본 저작권 표시는 그대로 유지되거나 해당 저작물이 다시 출판되는 매체에 적합한 방식으로 복제되어야 합니다.

1. 개요

이 실습의 학습 목표는 학생들이 환경 변수가 프로그램 및 시스템 동작에 어떻게 영향을 미치는지 이해하는 것입니다. 환경 변수는 실행 중인 프로세스가 컴퓨터에서 작동하는 방식에 영향을 줄 수 있는 동적 명명된 값 집합입니다. 환경 변수는 1979년 Unix에 도입된 이후 대부분의 운영 체제에서 사용됩니다. 환경 변수가 프로그램 동작에 영향을 주지만 이를 달성하는 방법은 많은 프로그래머가 잘 이해하지 못합니다. 결과적으로, 프로그램이 환경 변수를 사용하지만 프로그래머가 그 변수가 사용되는지 모른다면 프로그램에 취약점이 있을 수 있습니다.

이 실습에서 학생들은 환경 변수가 작동하는 방식, 상위 프로세스에서 하위 프로세스로 전파되는 방식, 시스템/프로그램 동작에 미치는 영향을 이해합니다. 우리는 특히 환경 변수가 일반적으로 권한이 있는 프로그램인 Set-UID 프로그램의 동작에 어떤 영향을 미치는지에 관심이 있습니다.

이 실습에서는 다음 주제를 다룹니다.

- 환경 변수
- Set-UID 프로그램 • 외부 프로그램을 안전하게 호출 • 기능 유출 • 동적 로더/링커

Set-UID 메커니즘, 환경 변수 및 관련 보안 문제에 대한 자세한 내용은 다음에서 확인할 수 있습니다 .

- SEED 도서 1장과 2장, 컴퓨터 및 인터넷 보안: 실습 접근 방식, 2차 에디션, Wenliang Du.

연구실 환경. 이 실습은 SEED Ubuntu 20.04 VM에서 테스트되었습니다. SEED 웹사이트에서 사전 구축된 이미지를 다운로드하고, 자신의 컴퓨터에서 SEED VM을 실행할 수 있습니다. 그러나 대부분의 SEED 실습은 클라우드에서 수행할 수 있으며 지침에 따라 클라우드에서 SEED VM을 생성할 수 있습니다.

2가지 실험실 작업

2.1 작업 1: 환경 변수 조작

이 작업에서는 환경 변수를 설정하고 설정 해제하는 데 사용할 수 있는 명령을 연구합니다. 우리는 시드 계정에서 Bash를 사용하고 있습니다. 사용자가 사용하는 기본 셸은 /etc/passwd 파일(각 항목의 마지막 필드)에 설정됩니다. chsh 명령을 사용하여 이를 다른 쉘 프로그램으로 변경할 수 있습니다(이 실습에서는 수행하지 마십시오). 다음 작업을 수행하십시오.

Machine Translated by Google

- 환경 변수를 인쇄하려면 printenv 또는 env 명령을 사용하십시오. PWD와 같은 특정 환경 변수에 관심이 있는 경우 "printenv PWD" 또는 "env | grep PWD"를 사용할 수 있습니다.

- 환경 변수를 설정하거나 설정 해제하려면 내보내기 및 설정 해제를 사용합니다. 이 두 명령은 별도의 프로그램이 아니라는 점에 유의해야 합니다. 이는 Bash의 내부 명령 중 두 가지입니다(Bash 외부에서는 찾을 수 없습니다).

2.2 작업 2: 부모 프로세스에서 자식 프로세스로 환경 변수 전달

이 작업에서는 하위 프로세스가 상위 프로세스로부터 환경 변수를 가져오는 방법을 연구합니다. Unix에서 fork()는 호출 프로세스를 복제하여 새로운 프로세스를 생성합니다. 자식이라고 하는 새 프로세스는 부모라고 하는 호출 프로세스의 정확한 복사본입니다. 그러나 몇 가지 사항은 자식에게 상속되지 않습니다(다음 명령을 입력하여 fork() 매뉴얼을 참조하십시오: man fork). 이 작업에서는 상위 프로세스의 환경 변수가 하위 프로세스에 상속되는지 여부를 알고 싶습니다.

1단계. 다음 프로그램을 컴파일하고 실행하고 관찰 내용을 설명하십시오. 이 프로그램은 Labsetup 폴더에서 찾을 수 있습니다. a.out이라는 바이너리를 생성하는 "gcc myprintenv.c"를 사용하여 컴파일할 수 있습니다. 이를 실행하고 "a.out > file"을 사용하여 출력을 파일에 저장해 보겠습니다.

목록 1: myprintenv.c

```
#include <unistd.h> #include
<stdio.h> #include <stdlib.h>


외부 char **환경; 무효 printenv() {


    int i = 0; while
    (environ[i] != NULL) {
        printf("%s\n", environ[i]); 나++;

    }
}

무효 메인() {

    pid_t childPid; 스위치
    (childPid = 포크()) {
        사례 0: /* 자식 프로세스 */ printenv(); 출구(0); 기본값: /
            * 상위 프로세스 */ //
            printenv();
        종료(0);


    }
}
```

2단계. 이제 하위 프로세스 케이스(라인)에서 printenv() 문을 주석 처리하고 상위 프로세스 케이스(라인)에서 printenv() 문의 주석 처리를 제거합니다. 코드를 다시 컴파일하고 실행하고,

Machine Translated by Google

당신의 관찰을 설명하십시오. 출력을 다른 파일에 저장합니다.

3단계. diff 명령을 사용하여 두 파일의 차이점을 비교합니다. 결론을 도출해 보십시오.

2.3 작업 3: 환경 변수 및 execve()

이번 작업에서는 execve()를 통해 새로운 프로그램이 실행될 때 환경 변수가 어떤 영향을 받는지 연구합니다.
execve() 함수는 시스템 호출을 호출하여 새 명령을 로드하고 실행합니다. 이 기능은 절대 반환되지 않습니다. 새로운 프로세스가 생성되지 않습니다. 대신 호출 프로세스의 텍스트, 데이터, bss 및 스택이 로드된 프로그램의 텍스트, 데이터, 스택으로 덮어쓰여집니다. 기본적으로 execve()는 호출 프로세스 내에서 새 프로그램을 실행합니다. 우리는 환경 변수에 어떤 일이 일어나는지에 관심이 있습니다. 새 프로그램에 자동으로 상속됩니까?

1단계. 다음 프로그램을 컴파일하고 실행하고 관찰 내용을 설명하십시오. 이 프로그램은 단순히 현재 프로세스의 환경 변수를 인쇄하는 /usr/bin/env라는 프로그램을 실행합니다.

목록 2: myenv.c

```
#include <unistd.h>

외부 char **환경; 정수 메인() {


    char *argv[2];

    argv[0] = "/usr/bin/env"; 인수[1] = NULL;
    execve("/usr/bin/env",
    argv, NULL);

    0을 반환합니다;
}
```

2단계. 행의 execve() 호출을 다음과 같이 변경합니다. 당신의 관찰을 설명하십시오.

```
execve("/usr/bin/env", argv, 환경);
```

3단계. 새 프로그램이 환경 변수를 얻는 방법에 관해 결론을 내리십시오.

2.4 작업 4: 환경 변수 및 system()

본 작업에서는 system() 함수를 통해 새로운 프로그램이 실행될 때 환경 변수가 어떻게 영향을 받는지 연구합니다. 이 함수는 명령을 실행하는 데 사용되지만 명령을 직접 실행하는 execve()와 달리 system()은 실제로 "/bin/sh -c 명령"을 실행합니다. 즉, /bin/sh를 실행하고 쉘에 요청합니다. 명령을 실행합니다.

system() 함수의 구현을 살펴보면 execl()을 사용하여 /bin/sh를 실행하는 것을 볼 수 있습니다. execl()은 execve()를 호출하여 환경 변수 배열을 전달합니다. 따라서 system()을 사용하면 호출 프로세스의 환경 변수가 새 프로그램 /bin/sh에 전달됩니다.

이를 확인하려면 다음 프로그램을 컴파일하고 실행하십시오.

Machine Translated by Google


```
#include <stdio.h> #include
<stdlib.h>

정수 메인() {

    system("/usr/bin/env"); 0을 반환합니다;

}
```

2.5 작업 5: 환경 변수 및 Set-UID 프로그램

Set-UID는 Unix 운영 체제의 중요한 보안 메커니즘입니다. Set-UID 프로그램이 실행되면 소유자의 권한을 가정합니다. 예를 들어, 프로그램의 소유자가 루트인 경우 누구든지 이 프로그램을 실행하면 프로그램은 실행 중에 루트의 권한을 얻습니다. Set-UID를 사용하면 많은 흥미로운 작업을 수행할 수 있지만 사용자의 권한이 상승하므로 매우 위험합니다. Set-UID 프로그램의 동작은 사용자가 아닌 프로그램 논리에 의해 결정되지만 사용자는 실제로 환경 변수를 통해 동작에 영향을 미칠 수 있습니다. Set-UID 프로그램이 어떻게 영향을 받는지 이해하기 위해 먼저 환경 변수가 사용자 프로세스에서 Set-UID 프로그램 프로세스에 의해 상속되는지 여부를 살펴보겠습니다.

1단계. 현재 프로세스의 모든 환경 변수를 인쇄할 수 있는 다음 프로그램을 작성합니다.

```
#include <stdio.h> #include
<stdlib.h>

외부 char **환경; 정수 메인() {


    int i = 0; while
    (environ[i] != NULL) {
        printf("%s\n",environ[i]); 나++;

    }
}
```

2단계. 위 프로그램을 컴파일하고 소유권을 루트로 변경한 후 Set-UID 프로그램으로 만듭니다.

```
// 프로그램 이름이 foo라고 가정합니다. $ sudo chown root foo $
sudo chmod 4755 foo
```

3단계. 셸에서(루트 계정이 아닌 일반 사용자 계정에 있어야 함) 내보내기 명령을 사용하여 다음 환경 변수를 설정합니다(이미 존재할 수도 있음).

- 길
- LD 라이브러리 경로
- 모든 이름(이것은 사용자가 정의한 환경 변수이므로 원하는 이름을 선택하세요).

이러한 환경 변수는 사용자의 셸 프로세스에서 설정됩니다. 이제 셸에서 2단계의 Set-UID 프로그램을 실행하세요. 쉘에 프로그램 이름을 입력하면 쉘은 하위 프로세스를 분기합니다.

Machine Translated by Google

그리고 자식 프로세스를 사용하여 프로그램을 실행합니다. 쉘 프로세스(상위)에서 설정한 환경 변수가 모두 Set-UID 하위 프로세스에 들어가는지 확인해주세요. 당신의 관찰을 설명하십시오. 놀라운 일이 있다면 설명해 보세요.

2.6 작업 6: PATH 환경 변수 및 Set-UID 프로그램

호출된 쉘 프로그램으로 인해 Set-UID 프로그램 내에서 system()을 호출하는 것은 매우 위험합니다.
이는 쉘 프로그램의 실제 동작이 PATH와 같은 환경 변수의 영향을 받을 수 있기 때문입니다. 이러한 환경 변수는 악의적일 수 있는 사용자가 제공한 것입니다. 악의적인 사용자는 이러한 변수를 변경하여 Set-UID 프로그램의 동작을 제어할 수 있습니다. Bash에서는 다음과 같은 방법으로 PATH 환경 변수를 변경할 수 있습니다(이 예에서는 PATH 환경 변수의 시작 부분에 /home/seed 디렉터리를 추가합니다).

```
$ 내보내기 경로=/home/seed:$PATH
```

아래의 Set-UID 프로그램은 /bin/ls 명령을 실행하도록 되어 있습니다. 그러나 프로그래머는 ls 명령에 대해 절대 경로가 아닌 상대 경로만 사용합니다.

```
정수 메인() {

    system("ls"); 0을 반환합
    니다 .
}
```

위 프로그램을 컴파일하고 소유자를 루트로 변경한 후 Set-UID 프로그램으로 만드십시오. 이 Set-UID 프로그램이 /bin/ls 대신 자신의 악성 코드를 실행하도록 할 수 있습니까? 가능하다면, 당신의 악성코드는 루트권한으로 실행되고 있습니까? 관찰한 내용을 기술하고 설명하십시오.

참고: system(cmd) 함수는 /bin/sh 프로그램을 먼저 실행한 다음 이 쉘 프로그램에 cmd 명령을 실행하도록 요청합니다. Ubuntu 20.04(및 이전 여러 버전)에서 /bin/sh는 실제로 /bin/dash를 가리키는 심볼릭 링크입니다. 이 쉘 프로그램에는 Set-UID 프로세스에서 자신이 실행되는 것을 방지하는 대책이 있습니다. 기본적으로 dash가 Set-UID 프로세스에서 실행되는 것을 감지하면 즉시 유효 사용자 ID를 프로세스의 실제 사용자 ID로 변경하여 본질적으로 권한을 삭제합니다.

우리의 피해자 프로그램은 Set-UID 프로그램이므로 /bin/dash의 대응책으로 우리의 공격을 막을 수 있습니다. 이러한 대응책 없이 공격이 어떻게 작동하는지 확인하기 위해 이러한 대응책이 없는 다른 셸에 /bin/sh를 연결합니다. Ubuntu 20.04 VM에 zsh라는 쉘 프로그램을 설치했습니다. /bin/sh를 /bin/zsh에 연결하려면 다음 명령을 사용합니다.

```
$ sudo ln -sf /bin/zsh /bin/sh
```

2.7 작업 7: LD PRELOAD 환경 변수 및 Set-UID 프로그램

이 작업에서는 Set-UID 프로그램이 일부 환경 변수를 처리하는 방법을 연구합니다. LD PRELOAD, LD LIBRARY PATH 및 기타 LD *를 포함한 여러 환경 변수는 동적 로더/링커의 동작에 영향을 미칩니다. 동적 로더/링커는 런타임 시 실행 파일에 필요한 공유 라이브러리를 로드하고(영구 저장소에서 RAM으로) 연결하는 운영 체제(OS)의 일부입니다.

Linux에서는 ld.so 또는 ld-linux.so가 동적 로더/링커입니다(각각 다른 유형의 바이너리에 해당).
해당 동작에 영향을 미치는 환경 변수 중에서 LD LIBRARY PATH 및 LD PRELOAD는 다음과 같습니다.

Machine Translated by Google

이 연구실에서 우리가 관심을 두고 있는 두 가지입니다. Linux에서 LD LIBRARY PATH는 표준 디렉토리 세트보다 먼저 라이브러리를 검색해야 하는 콜론으로 구분된 디렉토리 세트입니다. LD PRELOAD는 다른 모든 라이브러리보다 먼저 로드할 추가 사용자 지정 공유 라이브러리 목록을 지정합니다. 이번 과제에서는 LD PRELOAD에 대해서만 공부하겠습니다.

1단계 . 먼저 일반 프로그램을 실행할 때 이러한 환경 변수가 동적 로더/링커의 동작에 어떤 영향을 미치는지 살펴보겠습니다. 다음 단계를 따르십시오.

1. 동적 링크 라이브러리를 만들어 보겠습니다. 다음 프로그램을 만들고 이름을 mylib.c로 지정합니다. 그것은 기본적으로 libc의 sleep() 함수를 재정의합니다.

```
#include <stdio.h> void sleep
(int s) {

    /* 이것이 특권 프로그램에 의해 호출되는 경우,
        여기서 손해를 입을 수 있습니다! */ printf("나는 자지
    않습니다!\n");
}
```

2. 다음 명령을 사용하여 위 프로그램을 컴파일할 수 있습니다(-lc 인수에서 두 번째 문자는 ℓ임):

```
$ gcc -fPIC -g -c mylib.c $ gcc -shared -o
libmylib.so.1.0.1 mylib.o -lc
```

3. 이제 LD PRELOAD 환경 변수를 설정합니다.

```
$ 내보내기 LD_PRELOAD=./libmylib.so.1.0.1
```

4. 마지막으로 다음 프로그램 myprog를 위의 동적 링크와 동일한 디렉터리에 컴파일합니다. 라이브러리 libmylib.so.1.0.1:

```
/* myprog.c */ #include
<unistd.h> int main() {


    수면(1); 0을 반
    환합니다 .
}
```

2단계. 위의 작업을 완료한 후 다음 조건에서 myprog를 실행하고 어떤 일이 일어나는지 관찰하십시오.

- myprog를 일반 프로그램으로 만들고 일반 사용자로 실행합니다.
- myprog를 Set-UID 루트 프로그램으로 만들고 일반 사용자로 실행합니다.
- myprog를 Set-UID 루트 프로그램으로 만들고 LD PRELOAD 환경 변수를 다시 내보냅니다. 루트 계정으로 실행하세요.

Machine Translated by Google

- myprog를 Set-UID user1 프로그램으로 만들고(즉, 소유자는 다른 사용자 계정인 user1임) 다른 사용자 계정(루트가 아닌 사용자)에서 LD PRELOAD 환경 변수를 다시 내보내고 실행합니다.

3단계. 동일한 프로그램을 실행하더라도 위에 설명된 시나리오에서 다양한 동작을 관찰할 수 있어야 합니다. 차이의 원인이 무엇인지 파악해야 합니다. 여기서 환경 변수가 중요한 역할을 합니다. 주요 원인을 파악하기 위한 실험을 설계하고, 2단계의 동작이 다른 이유를 설명하세요. (힌트: 하위 프로세스는 LD * 환경 변수를 상속받지 못할 수 있습니다).

2.8 작업 8: system() 과 execve()를 사용하여 외부 프로그램 호출

system()과 execve()는 모두 새 프로그램을 실행하는 데 사용될 수 있지만 system()은 Set-UID 프로그램과 같은 권한 있는 프로그램에서 사용되는 경우 매우 위험합니다. 우리는 PATH 환경 변수가 system()의 동작에 어떤 영향을 미치는지 살펴보았습니다. 변수가 쉘 작동 방식에 영향을 미치기 때문입니다. execve()는 쉘을 호출하지 않기 때문에 문제가 없습니다. 쉘을 호출하는 것은 또 다른 위험한 결과를 낳는데, 이번에는 환경 변수와는 아무런 관련이 없습니다. 다음 시나리오를 살펴보겠습니다.

Bob은 감사 기관에서 근무하며 사기 혐의가 있는 회사를 조사해야 합니다. 조사 목적을 위해 Bob은 회사의 Unix 시스템에 있는 모든 파일을 읽을 수 있어야 합니다. 반면에 시스템의 무결성을 보호하기 위해 Bob은 어떤 파일도 수정할 수 없어야 합니다. 이 목표를 달성하기 위해 시스템의 수퍼유저인 Vince는 특별한 set-root-uid 프로그램(아래 참조)을 작성한 다음 Bob에게 실행 권한을 부여했습니다. 이 프로그램에서는 Bob이 명령줄에 파일 이름을 입력해야 하며 그런 다음 /bin/cat을 실행하여 지정된 파일을 표시합니다. 프로그램은 루트로 실행되므로 Bob이 지정하는 모든 파일을 표시할 수 있습니다. 그러나 프로그램에는 쓰기 작업이 없기 때문에 Vince는 Bob이 이 특수 프로그램을 사용하여 파일을 수정할 수 없다고 확신합니다.

목록 3: catall.c

```
int main(int argc, char *argv[]) {

    char *v[3]; char *명
    령;

    if(argc < 2) {
        printf("파일 이름을 입력하세요.\n"); 1을 반환합니다 .

    }

    v[0] = "/bin/cat"; v[1] = 인수[1]; v[2] = NULL; 명령 = malloc(strlen(v[0]) + strlen(v[1])
    + 2); sprintf(명령어, "%s %s", v[0], v[1]);


    // 다음 중 하나만 사용하세요. 시스템(명령); // execve(v[0], v,
    NULL);


    0을 반환합니다 ;
}
```

1단계: 위 프로그램을 컴파일하여 루트 소유의 Set-UID 프로그램으로 만듭니다. 프로그램은 system()을 사용하여 명령을 호출합니다. 당신이 Bob이라면 시스템의 무결성을 손상시킬 수 있습니까? 을 위한

Machine Translated by Google

예를 들어, 쓸 수 없는 파일을 제거할 수 있습니까?

2단계: system(command) 문을 주석 처리하고 execve() 문의 주석 처리를 제거합니다. 프로그램은 execve()를 사용하여 명령을 호출합니다. 프로그램을 컴파일하고 이를 루트 소유의 Set-UID로 만듭니다. 1단계의 공격이 여전히 작동하나요? 관찰한 내용을 기술하고 설명해주세요.

2.9 작업 9: 기능 누출

최소 권한 원칙을 따르기 위해 Set-UID 프로그램은 루트 권한이 더 이상 필요하지 않은 경우 루트 권한을 영구적으로 포기하는 경우가 많습니다. 더욱이 때로는 프로그램이 사용자에게 제어권을 넘겨야 하는 경우도 있습니다. 이 경우 루트 권한을 취소해야 합니다. setuid() 시스템 호출을 사용하여 권한을 취소할 수 있습니다. 매뉴얼에 따르면 "setuid()는 호출 프로세스의 유효 사용자 ID를 설정합니다. 호출자의 유효 UID가 루트인 경우 실제 UID와 저장된 set-user-ID도 설정됩니다." 따라서 유효 UID 0을 가진 Set-UID 프로그램이 setuid(n)을 호출하면 프로세스는 모든 UID가 n으로 설정되는 일반 프로세스가 됩니다.

권한을 취소할 때 흔히 저지르는 실수 중 하나는 기능 누출입니다. 프로세스가 여전히 특권을 갖고 있을 때 일부 특권 기능을 얻었을 수도 있습니다. 권한이 다운그레이드될 때 프로그램이 해당 기능을 정리하지 않으면 권한이 없는 프로세스에서 계속 액세스할 수 있습니다.
즉, 프로세스의 유효 사용자 ID가 비특권이 되더라도 프로세스는 특권 기능을 보유하고 있기 때문에 여전히 특권을 갖습니다.

다음 프로그램을 컴파일하고 소유자를 루트로 변경한 후 Set-UID 프로그램으로 만듭니다. 일반 사용자로 프로그램을 실행하십시오. 이 프로그램의 기능 유출 취약점을 악용할 수 있습니까? 목표는 일반 사용자로 /etc/zzz 파일에 쓰는 것입니다.

목록 4: cap Leak.c

```
무효 메인() {

    int fd; 문자
    *v[2];

    /* /etc/zzz가 중요한 시스템 파일이고 * 권한 0644를 가진 루트가 소유하고 있다고 가정합니다.

      * 이 프로그램을 실행하기 전에 먼저 /etc/zzz 파일을 * 만들어야 합니다. */ fd = open("/etc/zzz",
      O_RDWR | O_APPEND); if (fd == -1) { printf("/etc/zzz
    를 열 수 없습니다\n"); 출구(0);



    }

    // 파일 설명자 값을 인쇄합니다. printf("fd is %d\n", fd);


    // 유효 uid를 실제 uid와 동일하게 만들어 // 권한을 영구적으로 비활성화합니다. setuid(getuid());



    // 실행 /bin/sh v[0] = "/bin/sh";
    v[1] = 0; execve(v[0], v, 0);
```

Machine Translated by Google


```
}
```