14

막간: 메모리 API

이 막간에서 우리는 U의 메모리 할당 인터페이스에 대해 논의합니다.아니야
시스템. 제공된 인터페이스는 매우 간단하므로 챕터
짧고 요점[1]. 우리가 해결하는 주요 문제는 다음과 같습니다.

> 씨럭스: 시간오우티영형ㅏ찾아ㅏND중나이중에모리
> 유에서아니야/C 프로그램, 할당 및 관리 방법 이해
> 메모리는 강력하고 안정적인 소프트웨어를 구축하는 데 중요합니다. 일반적으로 사용되는 인터페이스는 무엇입니까? 어떤 실수를 피해야 합니까?

14.1 메모리의 종류

C 프로그램을 실행할 때 할당되는 메모리에는 두 가지 유형이 있습니다. 첫 번째는 호출**스택**메모리, 할당 및 할당 해제가 관리됩니다.*암묵적으로*프로그래머를 위한 컴파일러에 의해; 이러한 이유로 때때로 호출됩니다.**자동적 인**메모리.

C에서 스택에 메모리를 선언하는 것은 쉽습니다. 예를 들어 함수에 공간이 필요하다고 가정해 보겠습니다.함수()정수에 대해엑스.이러한 메모리 조각을 선언하려면 다음과 같이 하면 됩니다.

```
무효 함수() {
    정수 x; // 스택에 정수를 선언합니다. . .

}
```

컴파일러는 나머지 작업을 수행하여 를 호출할 때 스택에 공간을 확보합니다.함수().함수에서 돌아올 때 컴파일러는 메모리를 할당 해제합니다. 따라서 호출 호출을 넘어서는 일부 정보를 유지하려면 해당 정보를 스택에 남겨두지 않는 것이 좋습니다.

[1]사실, 우리는 모든 챕터가 그렇게 되기를 바랍니다! 그러나 이것은 더 짧고 더 뾰족하다고 생각합니다.

우리를 두 번째 유형의 기억, 즉**더미**모든 할당 및 할당 해제가 수행되는 메모리*명시적으로*프로그래머가 처리합니다. 무거운 책임, 의심의 여지가 없습니다! 그리고 확실히 많은 버그의 원인입니다. 그러나 주의를 기울이고 주의를 기울이면 그러한 인터페이스를 문제 없이 올바르게 사용할 수 있습니다. 다음은 힙에 정수를 할당하는 방법의 예입니다.

```
무효 함수() {
    int *x = (int *) malloc(sizeof(int)); . . .

}
```

이 작은 코드 조각에 대한 몇 가지 참고 사항입니다. 먼저 스택과 힙 할당이 모두 이 라인에서 발생한다는 것을 알 수 있습니다. 먼저 컴파일러는 해당 포인터(정수 *x);이후에 프로그램이 호출할 때malloc(),힙에 정수 공간을 요청합니다. 루틴은 그러한 정수의 주소를 반환합니다(성공 시 또는없는실패 시), 프로그램에서 사용할 수 있도록 스택에 저장됩니다.

힙 메모리의 명시적 특성과 더 다양한 사용으로 인해 힙 메모리는 사용자와 시스템 모두에게 더 많은 문제를 제시합니다. 따라서 나머지 논의의 초점입니다.

14.2malloc()부르다

그만큼**malloc()**호출은 매우 간단합니다. 힙에 공간을 요청하는 크기를 전달하면 성공하고 다음에 대한 포인터를 다시 제공합니다.
새로 할당된 공간 또는 실패하고 반환없는[2].

매뉴얼 페이지는 malloc을 사용하기 위해 무엇을 해야 하는지 보여줍니다; 유형 남자 말록명령줄에서 다음을 볼 수 있습니다.

```
# <stdlib.h> 포함
. . .
무효 *malloc(size_t 크기);
```

이 정보에서 헤더 파일을 포함하기만 하면 됩니다.stdlib.hmalloc을 사용합니다. 사실, 모든 C 프로그램이 기본적으로 연결되는 C 라이브러리에는malloc()그 안에; 헤더를 추가하면 컴파일러가 호출 여부를 확인할 수 있습니다.malloc()올바르게(예: 올바른 유형의 올바른 수의 인수 전달).

단일 매개변수malloc()유형이 걸립니다사이즈 t필요한 바이트 수를 간단히 설명합니다. 그러나 대부분의 프로그래머는 여기에 숫자(예: 10)를 직접 입력하지 않습니다. 실제로 그렇게 하는 것은 좋지 않은 형태로 간주될 것입니다. 대신 다양한 루틴과 매크로가

[2]참고없는C에서 는 전혀 특별한 것이 아니라 값 0에 대한 매크로일 뿐입니다.

티IP: 여암탉나N디OUBT, 티RY나티영형유타
사용 중인 일부 루틴이나 연산자가 어떻게 작동하는지 확실하지 않은 경우 단순히 시도하고 예상대로 작동하는지 확인하는 것 외에는 다른 방법이 없습니다. 매뉴얼 페이지나 기타 문서를 읽는 동안
유용합니다. 실제로 어떻게 작동하는지가 중요합니다. 코드를 작성하고 테스트하십시오! 그것은 의심할 여지 없이 코드가 원하는 대로 동작하도록 하는 가장 좋은 방법입니다. 사실, 그것이 우리가 말한 내용을 다시 확인하기 위해 한 것입니다.크기()사실이었다!

활용. 예를 들어 배정도 부동 소수점 값을 위한 공간을 할당하려면 다음과 같이 하면 됩니다.

```
더블 *d = (더블 *) malloc(sizeof(double));
```

와, 많네요.더블-잉! 이 호출malloc()사용 크기()운영자가 적절한 양의 공간을 요청합니다. C에서 이것은 일반적으로 다음과 같이 생각됩니다.*컴파일 타임*연산자는 실제 크기를 알고 있음을 의미합니다.*컴파일 시간*따라서 숫자(이 경우 double의 경우 8)가 인수로 대체됩니다.malloc().이러한 이유로, 크기()함수 호출이 아닌 연산자로 올바르게 간주됩니다(함수 호출은 런타임에 발생함).

유형뿐만 아니라 변수의 이름을 전달할 수도 있습니다. 크기(),그러나 경우에 따라 원하는 결과를 얻지 못할 수 있으므로 주의하십시오. 예를 들어 다음 코드 스니펫을 살펴보겠습니다.

```
int *x = malloc(10 * sizeof(int)); printf("%d\n",
sizeof(x));
```

첫 번째 줄에서 우리는 10개의 정수 배열을 위한 공간을 선언했는데, 이는 훌륭하고 멋집니다. 그러나 우리가 사용할 때크기()다음 줄에서는 4(32비트 시스템) 또는 8(64비트 시스템)과 같은 작은 값을 반환합니다. 그 이유는 이 경우,크기()우리는 단순히 얼마나 큰지 묻고 있다고 생각합니다*바늘*정수에 대한 것은 우리가 동적으로 할당한 메모리의 양이 아닙니다. 그러나 때때로크기()예상대로 작동합니다:

```
정수 x[10];
printf("%d\n", sizeof(x));
```

이 경우 컴파일러가 40바이트가 할당되었음을 알 수 있는 정적 정보가 충분합니다.

주의해야 할 또 다른 장소는 문자열입니다. 문자열을 위한 공간을 선언할 때 다음 관용구를 사용하십시오.malloc(strlen(s) + 1),함수를 사용하여 문자열의 길이를 가져옵니다.strlen(),문자열 끝 문자를 위한 공간을 만들기 위해 1을 추가합니다. 사용 크기() 여기에 문제가 발생할 수 있습니다.

당신은 또한 그것을 알 수 있습니다malloc()유형에 대한 포인터를 반환무효의. 그렇게 하는 것은 C에서 주소를 다시 전달하고 프로그래머가 그 주소로 무엇을 할지 결정하게 하는 방법일 뿐입니다. 프로그래머는**깁스**; 위의 예에서 프로그래머는 반환 유형을malloc()에 대한 포인터로더블.캐스팅은 컴파일러와 코드를 읽을 수 있는 다른 프로그래머에게 "예, 제가 하는 일을 알고 있습니다."라고 말하는 것 외에는 아무것도 수행하지 않습니다. 의 결과를 캐스팅하여malloc(),프로그래머는 단지 약간의 안심을 주고 있습니다. 캐스트는 정확성을 위해 필요하지 않습니다.

14.3무료()부르다

결과적으로 메모리 할당은 방정식의 쉬운 부분입니다. 언제, 어떻게, 메모리를 해제할지 여부를 아는 것은 어려운 부분입니다. 더 이상 사용하지 않는 힙 메모리를 해제하려면 프로그래머는 다음을 호출하기만 하면 됩니다.**무료()**:

```
int *x = malloc(10 * sizeof(int)); . . .

무료(x);
```

루틴은 하나의 인수를 취합니다.malloc(). 따라서 할당된 영역의 크기는 사용자가 전달하지 않으며 메모리 할당 라이브러리 자체에서 추적해야 합니다.

14.4 일반적인 오류

의 사용에서 발생하는 많은 일반적인 오류가 있습니다.malloc() 그리고무료().다음은 학부 운영 체제 과정을 가르칠 때 반복해서 본 몇 가지입니다. 이 모든 예제는 컴파일러의 엿보기로 컴파일 및 실행됩니다. C 프로그램을 컴파일하는 것은 올바른 C 프로그램을 빌드하는 데 필요하지만 배우게 될 것이므로 충분하지 않습니다(종종 어려운 방법으로).

올바른 메모리 관리는 실제로 많은 최신 언어에서 다음을 지원하는 문제였습니다. **자동 메모리 관리**. 그러한 언어에서는 다음과 유사한 것을 호출하는 동안malloc() 메모리 할당(보통**새로운**또는 새 개체를 할당하는 것과 유사한 것), 여유 공간을 위해 무언가를 호출할 필요가 없습니다. 오히려,**가비지 컬렉터**실행하고 더 이상 참조하지 않는 메모리를 파악하고 해제합니다.

메모리 할당을 잊어버린 경우

많은 루틴은 호출하기 전에 메모리가 할당될 것으로 예상합니다. 예를 들어, 루틴 strcpy(dst, src)소스 포인터에서 대상 포인터로 문자열을 복사합니다. 그러나 주의하지 않으면 다음과 같이 할 수 있습니다.

```
char *src = "안녕하세요"; 문
자 *dst;                    // 앗! 할당되지 않은
strcpy(dst, src); // 세그폴트 및 다이
```

티IP: 나티씨생략영형아르 자형나티아르 자형안6=나티나에스씨정확한
프로그램이 컴파일(!)되었거나 한 번 또는 여러 번 올바르게 실행되었다고 해서 프로그램이 올바른 것은 아닙니다. 많은 사건들이 당신이 그것이 효과가 있다고 믿는 지점에 이르게 하기 위해 공모했을지 모르지만,
상황이 바뀌고 멈춥니다. 일반적인 학생 반응은 "하지만 이전에는 효과가 있었습니다!"라고 말하는(또는 소리를 지르는 것)입니다. 그런 다음 컴파일러, 운영 체제, 하드웨어 또는 심지어 (감히 말해서) 교수를 비난합니다. 그러나 문제는 일반적으로 코드에서 문제가 있다고 생각하는 위치에 있습니다. 다른 구성 요소를 비난하기 전에 작업하고 디버그하십시오.

이 코드를 실행하면**세그멘테이션 오류**삼에 대한 멋진 용어입니다.**당신은 메모리에 잘못된 일을 했어 당신은 프로그래머이고 나는 화가 난다.**

이 경우 적절한 코드는 다음과 같을 수 있습니다.

```
char *src = "안녕하세요";
문자 *dst = (문자 *) malloc(strlen(src) + 1); strcpy(dst, src); // 올
바르게 작동
```

또는 다음을 사용할 수 있습니다.strdup()당신의 삶을 훨씬 더 쉽게 만듭니다. 읽기 strdup자세한 내용은 매뉴얼 페이지를 참조하십시오.

메모리를 충분히 할당하지 않음

관련 오류는 충분한 메모리를 할당하지 않는 것입니다.**버퍼 오버 플로우**. 위의 예에서 일반적인 오류는*거의*목적지 버퍼를 위한 충분한 공간.

```
char *src = "안녕하세요";
char *dst = (char *) malloc(strlen(src)); // 너무 작은! strcpy(dst, src); // 올바르
게 작동
```

이상하게도 malloc이 구현되는 방식과 기타 여러 세부 사항에 따라 이 프로그램은 종종 올바르게 실행되는 것처럼 보입니다. 어떤 경우에는 문자열 복사가 실행될 때 할당된 공간의 끝을 너무 지나서 한 바이트를 쓰지만 어떤 경우에는 더 이상 사용되지 않는 변수를 덮어쓰므로 무해합니다. 어떤 경우에는 이러한 오버플로가 엄청나게 해로울 수 있으며 실제로 시스템의 많은 보안 취약점의 원인이 됩니다[W06]. 다른 경우에, malloc 라이브러리는 여하튼 약간의 추가 공간을 할당했고 따라서 프로그램은 실제로 다른 변수의 값에 낙서하지 않고 아주 잘 작동합니다. 다른 경우에도 프로그램은 실제로 오류가 발생하고 충돌합니다. 따라서 우리는 또 다른 귀중한 교훈을 얻습니다. 한 번 올바르게 실행되더라도 그것이 정확하다는 의미는 아닙니다.

삼생소하게 들릴지 모르지만 이러한 불법적인 메모리 액세스를 분할 오류라고 하는 이유를 곧 알게 될 것입니다. 그것이 계속 읽을 동기가 아니라면 무엇입니까?

할당된 메모리 초기화를 잊어버린 경우

이 오류로 전화를 겁니다.malloc()그러나 새로 할당된 데이터 유형에 일부 값을 채우는 것을 잊으십시오. 이러지 마! 잊어버리면 프로그램은 결국**초기화되지 않은 읽기**, 여기서 알 수 없는 값의 일부 데이터를 힙에서 읽습니다. 그 안에 무엇이 있을지 누가 알겠습니까? 운이 좋다면 프로그램이 여전히 작동하도록 하는 일부 값(예: 0). 운이 좋지 않다면 무작위적이고 해로운 것입니다.

메모리 확보를 잊음

또 다른 일반적인 오류는 다음과 같습니다.**메모리 누수**, 메모리를 해제하는 것을 잊었을 때 발생합니다. 장기간 실행되는 애플리케이션 또는 시스템(예: OS 자체)에서는 메모리가 천천히 누출되면 결국 메모리가 부족해지고 이 시점에서 다시 시작해야 하기 때문에 이는 큰 문제입니다. 따라서 일반적으로 메모리 덩어리가 끝나면 해제해야 합니다. 가비지 수집 언어를 사용하는 것은 여기에서 도움이 되지 않습니다. 여전히 일부 메모리 청크에 대한 참조가 있는 경우 가비지 수집기가 이를 해제하지 않으므로 메모리 누수는 더 현대적인 언어에서도 여전히 문제로 남아 있습니다.

어떤 경우에는 전화하지 않는 것처럼 보일 수 있습니다.무료()합리적이다. 예를 들어, 귀하의 프로그램은 수명이 짧고 곧 종료됩니다. 이 경우 프로세스가 죽으면 OS는 할당된 모든 페이지를 정리하므로 메모리 누수가 발생하지 않습니다. 이것은 확실히 "작동"하지만(7페이지의 옆 부분 참조) 개발하는 것은 아마도 나쁜 습관일 수 있으므로 그러한 전략을 선택하는 데 주의하십시오. 장기적으로 프로그래머로서의 목표 중 하나는 좋은 습관을 개발하는 것입니다. 그 습관 중 하나는 메모리 관리 방법을 이해하고 (C와 같은 언어로) 할당한 블록을 해제하는 것입니다. 그렇게 하지 않고 벗어날 수 있다 하더라도 명시적으로 할당한 모든 바이트를 해제하는 습관을 갖는 것이 좋습니다.

완료하기 전에 메모리 확보

때때로 프로그램은 사용을 마치기 전에 메모리를 해제합니다. 그러한 실수는**매달린 포인터**, 그리고 당신이 짐작할 수 있듯이 그것은 또한 나쁜 것입니다. 이후에 사용하면 프로그램이 충돌하거나 유효한 메모리를 덮어쓸 수 있습니다(예:무료(),하지만 전화를 걸었다malloc()다시 다른 것을 할당하고 잘못 해제된 메모리를 재활용함).

반복적으로 메모리 해제

프로그램은 때때로 메모리를 두 번 이상 해제하기도 합니다. 이것은 로 알려져 있습니다**더블 무료**. 그렇게 하는 결과는 정의되지 않습니다. 상상할 수 있듯이 메모리 할당 라이브러리는 혼란스러워 온갖 이상한 일을 할 수 있습니다. 충돌은 일반적인 결과입니다.

卜옆: 여HYN영형중에모리나에스엘EAKED영형NCE와이우리의피로세스이자형XITS

수명이 짧은 프로그램을 작성할 때 다음을 사용하여 일부 공간을 할당할 수 있습니다. malloc().프로그램이 실행되고 완료될 예정입니다. 호출할 필요가 있습니까?무료() 나가기 직전에 여러 번? 하지 않는 것이 잘못된 것처럼 보이지만 실제 의미에서 메모리는 "손실"되지 않습니다. 그 이유는 간단합니다. 시스템에는 실제로 두 가지 수준의 메모리 관리가 있습니다.

메모리 관리의 첫 번째 수준은 OS에 의해 수행되며, 프로세스가 실행될 때 메모리를 할당하고 프로세스가 종료될 때(또는 그렇지 않으면 죽을 때) 다시 가져옵니다. 두 번째 관리 수준은 *이내에*예를 들어 호출할 때 힙 내에서 각 프로세스 malloc()그리고무료().전화를 받지 못해도무료() (따라서 힙에서 메모리 누수), 운영 체제는 회수합니다.*모두*의 기억

프로그램 실행이 완료될 때 프로세스(코드, 스택 및 관련되는 경우 힙 페이지 포함). 주소 공간의 힙 상태에 관계없이 OS는 프로세스가 종료되면 해당 페이지를 모두 다시 가져오므로 해제하지 않았음에도 불구하고 메모리가 손실되지 않도록 합니다.

따라서 수명이 짧은 프로그램의 경우 메모리 누수가 작동 문제를 일으키지 않는 경우가 많습니다(형식이 좋지 않은 것으로 간주될 수 있음). 오래 실행되는 서버(예: 절대 종료되지 않는 웹 서버 또는 데이터베이스 관리 시스템)를 작성할 때 누출된 메모리는 훨씬 더 큰 문제이며 애플리케이션에서 메모리가 부족할 때 결국 충돌로 이어질 것입니다. 물론 메모리 누수는 특정 프로그램 내에서 훨씬 더 큰 문제입니다. 운영 체제 자체입니다. 우리에게 다시 한 번 보여줍니다: 커널 코드를 작성하는 사람들은 가장 힘든 일을 하고 있습니다...

부름무료()틀리게

우리가 논의하는 마지막 문제 중 하나는무료()틀리게. 결국, 무료()에서 받은 포인터 중 하나만 전달하기를 기대합니다.malloc()더 일찍. 다른 값을 전달하면 나쁜 일이 발생할 수 있습니다. 따라서 그러한**유효하지 않은 자유**위험하며 물론 피해야 합니다.

요약

보시다시피, 메모리를 남용하는 방법은 많이 있습니다. 빈번한 메모리 오류로 인해 코드에서 이러한 문제를 찾는 데 도움이 되는 도구의 전체 생태계가 개발되었습니다. 둘 다 확인하세요**정화하다**[HJ92] 및 **발그린**[SN05]; 둘 다 기억 관련 문제의 원인을 찾는 데 탁월합니다. 이러한 강력한 도구를 사용하는 데 익숙해지면 도구 없이 어떻게 버텼는지 궁금할 것입니다.

14.5 기본 OS 지원

논의할 때 시스템 호출에 대해 이야기하지 않았다는 것을 눈치채셨을 것입니다. malloc()그리고무료().그 이유는 간단합니다. 시스템 호출이 아니라 라이브러리 호출입니다. 따라서 malloc 라이브러리는 가상 주소 공간 내의 공간을 관리하지만 자체는 OS를 호출하여 더 많은 메모리를 요청하거나 일부를 시스템에 다시 해제하는 일부 시스템 호출 위에 구축됩니다.

그러한 시스템 호출 중 하나는브르크,프로그램의 위치를 변경하는 데 사용됩니다.**부서지다**: 힙의 끝 위치. 하나의 인수(새 중단의 주소)를 취하므로 새 중단이 현재 중단보다 크거나 작은지 여부에 따라 힙 크기를 늘리거나 줄입니다. 추가 통화sbrk증분을 전달하지만 그렇지 않으면 유사한 목적을 수행합니다.

다음 중 하나를 직접 호출해서는 안 됩니다.브르크또는sbrk.메모리 할당 라이브러리에서 사용합니다. 그것들을 사용하려고 하면 무언가가 (끔찍하게) 잘못될 것입니다. ~에 충실하다malloc()그리고 무료()대신에.

마지막으로 다음을 통해 운영 체제에서 메모리를 얻을 수도 있습니다. mmap()전화. 올바른 인수를 전달하여mmap()만들 수 있습니다 **익명의**프로그램 내의 메모리 영역 — 특정 파일과 관련되지 않은 영역**스왑 공간**, 가상 메모리에서 나중에 자세히 논의할 것입니다. 이 메모리는 힙처럼 취급되어 관리될 수도 있습니다. 의 매뉴얼 페이지를 읽으십시오.mmap()자세한 사항은.

14.6 기타 호출

메모리 할당 라이브러리가 지원하는 몇 가지 다른 호출이 있습니다. 예를 들어,호출()메모리를 할당하고 반환하기 전에 0으로 만듭니다. 이것은 메모리가 0이라고 가정하고 스스로 초기화하는 것을 잊어버리는 오류를 방지합니다(위의 "초기화되지 않은 읽기" 단락 참조). 루틴진짜 할당()또한 무언가(예: 배열)를 위한 공간을 할당한 다음 무언가를 추가해야 할 때 유용할 수 있습니다.진짜 할당()새로운 더 큰 메모리 영역을 만들고 이전 영역을 복사한 다음 새 영역에 대한 포인터를 반환합니다.

14.7 요약

메모리 할당을 다루는 몇 가지 API를 소개했습니다. 항상 그렇듯이 기본 사항만 다루었습니다. 자세한 내용은 다른 곳에서 볼 수 있습니다. 자세한 내용은 C 책 [KR88]과 Stevens [SR05](7장)를 읽으십시오. 이러한 많은 문제를 자동으로 감지하고 수정하는 방법에 대한 최신 논문은 Novark et al. [N+07]; 이 문서에는 또한 일반적인 문제에 대한 멋진 요약과 문제를 찾고 수정하는 방법에 대한 몇 가지 깔끔한 아이디어가 포함되어 있습니다.

참고문헌

[HJ92] R. Hastings, B. Joyce의 "Purify: 메모리 누수 및 액세스 오류의 빠른 감지". 유세닉스 Winter '92. *이제 상용 제품이 된 멋진 Purify 도구 뒤에 있는 종이.*

[KR88] Brian Kernighan, Dennis Ritchie의 "C 프로그래밍 언어". 1988년 프렌티스 홀. *C 개발자가 만든 C 책. 한 번 읽고 프로그래밍을 한 다음 다시 읽은 다음 책상 근처나 프로그래밍하는 모든 위치에 두십시오.*

[N+07] G. Novark, ED Berger, BG Zorn의 "Exterminator: 높은 확률로 메모리 오류 자동 수정". PLDI 2007, 샌디에이고, 캘리포니아. *메모리 오류를 자동으로 찾고 수정하는 방법에 대한 멋진 문서와 C 및 C++ 프로그램의 많은 일반적인 오류에 대한 훌륭한 개요입니다. 이 문서의 확장 버전은 CACM(2008년 12월 51권, 12호)에서 사용할 수 있습니다.*

[SN05] J. Seward, N. Nethercote의 "Valgrind를 사용하여 비트 정밀도로 정의되지 않은 값 오류 감지" 유세닉스 '05. *valgrind를 사용하여 특정 유형의 오류를 찾는 방법.*

[SR05] “미국의 고급 프로그래밍아니아환경”, W. Richard Stevens, Stephen A. Rago. 애디슨-웨슬리, 2005. *우리는 전에도 말했지만, 다시 말하겠습니다. 이 책을 여러 번 읽고 의심스러울 때마다 참고 자료로 사용하십시오. 저자들은 이 책의 내용을 읽을 때마다 C 프로그래밍을 다년간 했음에도 불구하고 어떻게 새로운 것을 배우는지에 항상 놀란다.*

[W06] T. Werthman의 "버퍼 오버플로 공격 및 대책에 대한 조사". 이용 가능: www.nds.rub.de/lehre/seminar/SS06/Werthmann BufferOverflow.pdf. *버퍼 오버플로와 이로 인해 발생하는 보안 문제에 대한 좋은 조사입니다. 많은 유명한 익스플로잇을 참조합니다.*

숙제(코드)

이 숙제에서 당신은 메모리 할당에 대해 어느 정도 익숙해질 것입니다. 먼저 버그가 있는 프로그램을 작성합니다(재미!). 그런 다음 삽입한 버그를 찾는 데 도움이 되는 몇 가지 도구를 사용합니다. 그러면 이러한 도구가 얼마나 멋진지 깨닫고 미래에 이를 사용하여 자신을 더 행복하고 생산적으로 만들 수 있습니다. 도구는 디버거입니다(예: gdb)그리고 메모리 버그 감지기라고 하는발그린 [SN05].

질문

1. 먼저 다음과 같은 간단한 프로그램을 작성하십시오.null.c정수에 대한 포인터를 생성하고 다음으로 설정합니다.없는,그런 다음 역참조하려고 합니다. 이것을 실행 파일로 컴파일하십시오.없는.이 프로그램을 실행하면 어떻게 됩니까?

2. 다음으로 기호 정보가 포함된 이 프로그램을 컴파일합니다(-지 fl아). 그렇게 하면 더 많은 정보를 실행 파일에 넣어 디버거가 변수 이름 등에 대한 더 유용한 정보에 액세스할 수 있도록 합시다. 다음을 입력하여 디버거에서 프로그램을 실행하십시오.gdb null그리고 나서, 한 번gdb실행 중, 입력 중 운영.gdb는 무엇을 보여줍니까?

3. 마지막으로발그린이 프로그램의 도구입니다. 우리는 사용할 것입니다멤체크 의 일부인 도구발그린무슨 일이 일어나는지 분석하기 위해. 다음을 입력하여 실행합니다.valgrind --leak-check=예 null입니다.이것을 실행하면 어떻게 될까요? 도구의 출력을 해석할 수 있습니까?

4. 다음을 사용하여 메모리를 할당하는 간단한 프로그램을 작성하십시오. malloc()그러나 종료하기 전에 해제하는 것을 잊습니다. 이 프로그램이 실행되면 어떻게 됩니까? 사용할 수 있습니까?gdb그것에 어떤 문제를 찾기 위해? 어때요 발그라인드(다시 --누출 확인 = 예 fl아)?

5. 정수 배열을 생성하는 프로그램을 작성하십시오.데이터크기 100 사용말록;그런 다음 설정데이터[100]제로. 이 프로그램을 실행하면 어떻게 됩니까? 다음을 사용하여 이 프로그램을 실행하면 어떻게 됩니까?발그린?프로그램이 맞습니까?

6. 정수 배열(위와 같이)을 할당하고 해제한 다음 배열 요소 중 하나의 값을 인쇄하려고 시도하는 프로그램을 만듭니다. 프로그램이 실행되나요? 사용하면 어떻게 되나요? 발그린그 위에?

7. 이제 재미있는 값을 free에 전달합니다(예: 위에서 할당한 배열의 중간에 있는 포인터). 무슨 일이야? 이러한 유형의 문제를 찾는 도구가 필요합니까?

8. 메모리 할당에 대한 다른 인터페이스를 시도해 보십시오. 예를 들어, 다음을 사용하는 간단한 벡터와 같은 데이터 구조 및 관련 루틴을 만듭니다.진짜 할당() 벡터를 관리합니다. 벡터 요소를 저장하려면 배열을 사용하십시오. 사용자가 벡터에 항목을 추가할 때진짜 할당()더 많은 공간을 할당합니다. 그러한 벡터는 얼마나 잘 수행됩니까? 연결 목록과 비교하면 어떻습니까? 사용 발그린버그를 찾는 데 도움이 됩니다.

9. 더 많은 시간을 할애하고 사용에 대해 읽으십시오.gdb그리고발그린.도구를 아는 것이 중요합니다. 시간을 보내고 미국에서 전문 디버거가 되는 방법을 배우십시오.아니야및 C 환경.