Section 13

어디로 뛸까? 메모리 지도 그려보기

지난 시간에 우린 메모리에 원하는 값을 입력해 넣는 연습을 해보았고,
이 연습을 통해 return address를 원하는 값으로 바꿀 수 있게 되었습니다.
즉, 원하는 메모리 주소로 뛰어 프로그램의 실행 흐름을 변경할 수 있게 된 것입니다.

이제부터 우리는 전체 메모리 영역 중에 과연 어디로 뛰는 것이 해킹에 도움이 될지에 대해 생각해 보겠습니다.

먼저 알아야 할 것은 과연 우리가 뛸 수 있는 메모리 주소의 범위가 어떻게 되느냐는 것입니다.

가장 낮은 메모리 주소는 0이므로 시작점은 0이 될 것이고..
그렇다면 가장 높은 메모리 주소는 얼마나 될까요?

이는 여러분이 사용하는 CPU의 종류와 관련이 있습니다.
흔히 우리가 CPU를 선택할 때 32비트니 64비트니 하며 비트수를 따지게 됩니다. 이 비트수에 따라 CPU가 한번에 처리할 수 있는 데이터의 크기가 달라지고, 이 크기가 곧 우리가 접근할 수 있는 메모리의 가장 높은 주소가 됩니다.

우리의 실습 환경인 Redhat 6.2는 32비트 환경에서 작동하기 때문에 지금까지 그래왔던 것처럼 앞으로도 32비트 CPU를 기준으로 설명해 나가겠습니다.

만약 여러분이 사용하고 있는 CPU가 64비트여도 실습용 Redhat 6.2 Vmware 이미지가 32비트 환경으로 세팅되어 있으므로 32비트와 동일하게 작동합니다.

비트란 것은 0 혹은 1을 표현할 수 있는 단위를 의미하기 때문에, 32비트로 표현할 수 있는 최대 값은 32개의 1, 즉 11111111111111111111111111111111이 됩니다.

그리고 이 값을 계산기를 이용해서 16진수로 바꾸어보면, 0xFFFFFFFF이 됩니다. 즉, 32비트 CPU 환경에서 접근하거나 표현할 수 있는 가장 높은 메모리 주소가 바로 이 0xFFFFFFFF 값입니다. 이 이상의 값은 직접적으로 표현할 수도, 데이터를 저장하거나 읽어오기 위해 접근할 수도 없습니다.

다시 한번 이 16진수 0xFFFFFFFF 값을 10진수로 바꾸면 4294967295가 되며, 이는 4GB 크기와 동일합니다. 32비트 CPU에선 아무리 RAM을 많이 장착한다고 해도 최대 4GB 까지밖에 인식하지 못하는데, 그 이유가 바로 4GB 이상의 메모리 주소는 표현할 수도, 접근할 수도 없기 때문입니다.

그럼 이제부터 본격적으로 메모리 지도를 그려나가며 과연 어느 주소에 무엇이 위치하고 있는지에 대해 알아보겠습니다. 여기서 메모리 지도란, 메모리의 어느 부분에 어떤 용도의 데이터들이 위치하게 되는지 그 기본 규칙을 이해하기 위한 간단한 구조의 그림을 말합니다.

우리가 표현할 수 있는 메모리 범위가 0에서부터 0xFFFFFFFF인 것을 알았으므로 다음과 같은 빈 공간의 그림을 그린 후 내용을 하나씩 채워나가도록 합시다.

그런데 잠깐, 아마 어떤 분들은 이런 의문을 갖게 되셨을 겁니다.

13.어디로 뛸까? 메모리 지도 그려보기

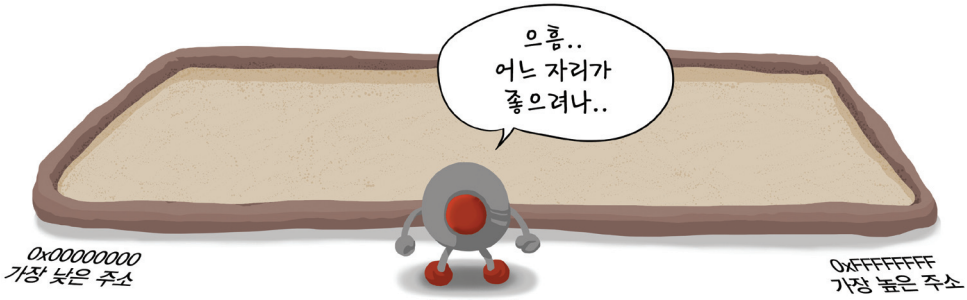
'그럼 만약 장착된 RAM이 2기가밖에 안 되는 컴퓨터에선 메모리 지도도 달라지나요?'

그렇지는 않습니다. 현대의 운영체제들은 "가상 메모리"와 "Swapping"이라고 불리는 메모리 관리 방식을 사용하고 있는데, 이를 이용하면 항상 동일한 메모리 지도 구조를 가지며, 실제 가지고 있는 RAM 크기보다 더 큰 메모리 영역을 사용할 수 있게 됩니다. 가상 메모리란, 4GB 크기의 커다란 가상메모리와 실제로 장착된 물리적 메모리를 서로 연결(Mapping)시켜놓은 개념으로서, 모든 가상메모리로의 접근은 커널의 Paging이란 과정을 거쳐 물리 메모리 주소로 자동 변환됩니다. 그래서 실제 물리 메모리 영역을 벗어나는 메모리 주소에 접근해도 결국엔 올바른 물리 메모리 주소를 잘 찾아가게 됩니다.

그리고 Swapping이란, 현 시점에서 바로 사용되지 않는 메모리 영역을 용량이 넉넉한 하드디스크에 임시 보관해 놓음으로써, 실제 소유하고 있는 물리 메모리 용량보다 훨씬 더 큰 메모리를 확보할 수 있게 되는 것을 말합니다.
(위 개념들에 대한 더욱 구체적인 내용은 "심화편"에서 설명드리도록 하겠습니다.)

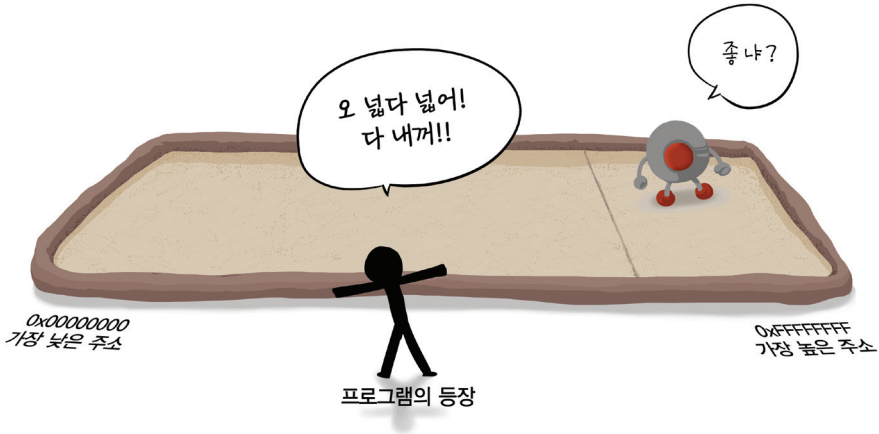
그럼 이제 0x00000000 ~ 0xFFFFFFFF 메모리 범위 안에 과연 어떠한 것들이 자리를 차지하고 있는지에 대해 알아보겠습니다.

운영체제에 있어서 가장 중요한 핵심부는 커널(Kernel)이라는 말을 이미 들어보셨거나 앞으로 공부를 해나가시면서 많이 듣게 되실 텐데요, 메모리 지도의 뒷쪽 1기가 영역에는 바로 이 커널 및 커널이 사용하는 데이터들이 자리잡게 됩니다.



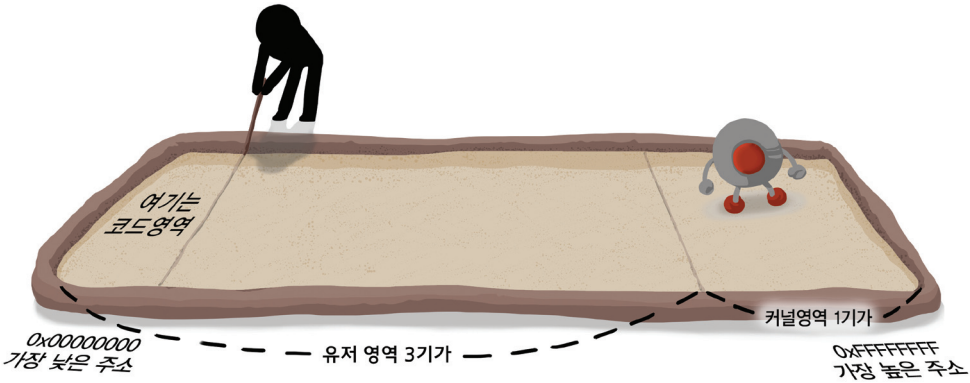
커널의 등장

그리고 남은 3기가의 영역을 우리의 프로그램이 사용하게 됩니다.



이처럼 두 부분으로 나뉜 영역을 각각 유저 영역과, 커널 영역이라고 부르게 됩니다.

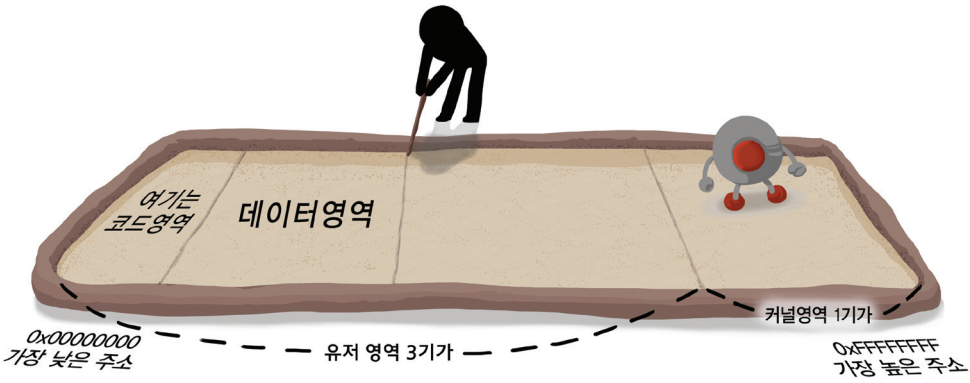
무려 3기가의 영역을 할당 받은 우리의 프로그램은 이 영역을 다시 여러 부분으로 나누어 사용합니다. 비슷한 성격의 정보들을 서로 모아서 메모리 관리 효율을 높이기 위함입니다. 컴퓨터 공학에선 이렇게 나뉘어진 영역을 세그먼트(segment)라고 부릅니다.



3기가의 유저 메모리 영역 중 가장 앞쪽엔 코드 영역이 자리잡게 됩니다.
코드 영역이란, CPU가 읽어 해석할 수 있는 기계어들이 위치하게 되는 영역을 말합니다. 우리가 작성한 메인 프로그램의 기계어 코드가 바로 이 영역에 위치하게 됩니다.

하나의 프로그램이 실행되면, 이 코드 영역에 저장된 기계어 명령 집합들이 순차적으로 읽혀 나가며 프로그램이 작동을 하게 됩니다.

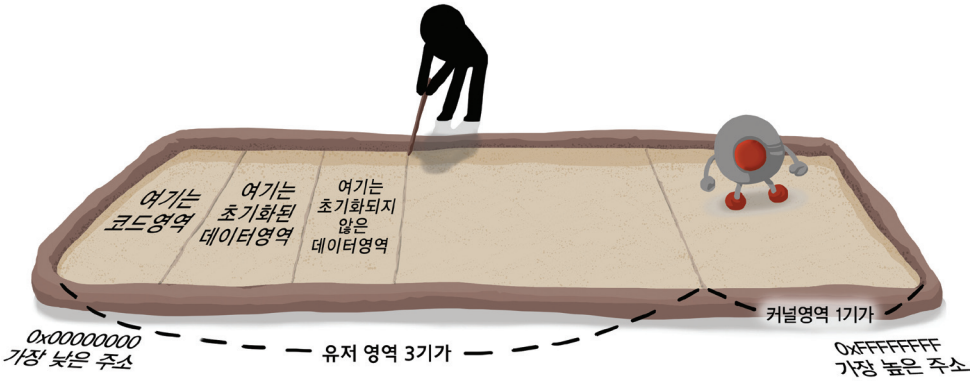
다음은 데이터 영역이라는 곳입니다.



13.어디로 뛸까? 메모리 지도 그려보기

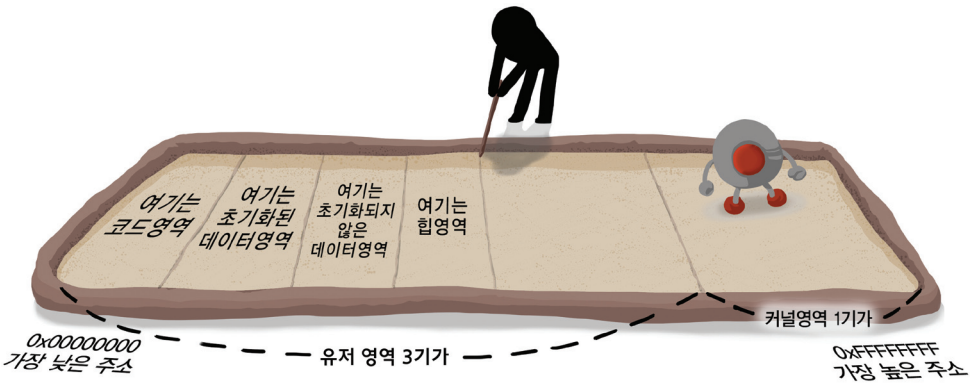
데이터 영역이란, 전역변수, 정적변수 등 각종 변수들이 실제로 위치하게 되는 메모리 영역인데, 변수를 선언할 때 초기화를 시켜주었는지 그렇지 않았는지에 따라 그림과 같이 저장되는 위치가 달라집니다.

즉, 다음과 같이 다시 두 개의 영역으로 구분하는 것이 가능합니다.



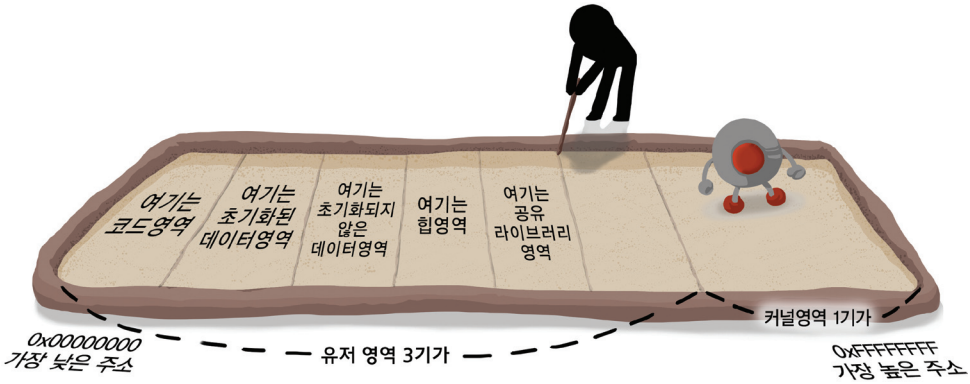
보시다시피, 초기화 된 변수들은 더 낮은 주소, 반대로 초기화되지 않은 변수들은 더 높은 주소 영역에 위치하게 됩니다.

다음은 힙(heap) 영역입니다.



힙 영역이란, malloc() 등 동적으로 메모리를 할당받는 함수를 통해 생성된 영역을 의미합니다. 함수 내에서만 유효한 지역 변수나, 프로그램 실행 내내 메모리에 상주하게 되는 전역 변수와는 달리, 동적 변수는 프로그래머가 원하는 시점에 메모리를 할당 받거나 해제할 수 있기 때문에 메모리를 더욱 효율적으로 관리할 수 있는 방식입니다.

그 오른쪽은 공유 라이브러리 영역입니다



공유 라이브러리란, 우리의 메인 프로그램이 내부적으로 사용하는 라이브러리 함수와 관련된 파일을 말합니다. 만약 메인 프로그램 내에서 printf() 라이브러리 함수를 사용했다면, 실제 이 함수를 가지고 있는 /lib/libc.so.6 파일이 이 영역에 적재되는 것입니다.

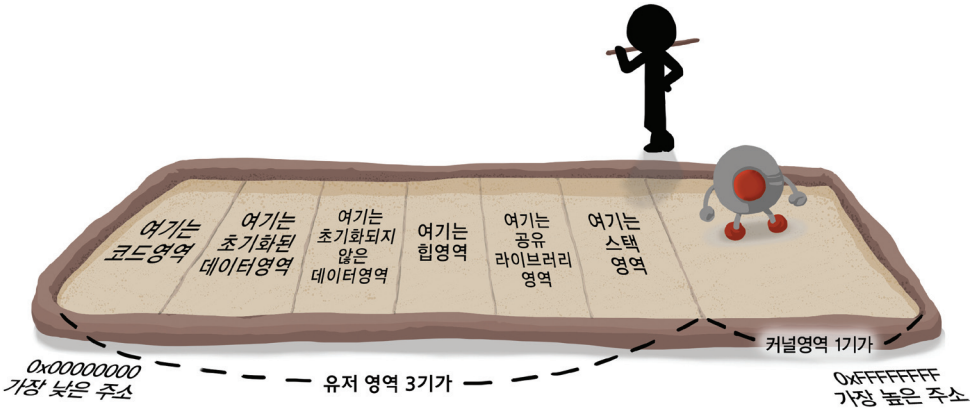
그리고 만약, 메인 프로그램 내에서 printf() 라이브러리 함수 하나만을 사용하고 있다고 할 때, libc.so.6 내의 printf() 함수 단 하나만 메모리에 적재되는 것이 아닌, printf()를 가지고 있는 libc.so.6 파일이 통째로 메모리에 올라가게 됩니다. 그래서 결국 libc.so.6 안의 모든 함수들이 이 라이브러리 영역에 위치하게 됩니다.

만약 libc.so.6 라이브러리 파일 내에 포함된 함수 목록이 궁금하다면 /usr/bin/nm 유틸리티를 사용하면 됩니다.

13.어디로 뛸까? 메모리 지도 그려보기

```
$ /usr/bin/nm /lib/libc.so.6  | more
...
008f5630 T ctime_r
009c802c d current_rtmax
009c8028 d current_rtmin
009c9bf0 b curshell
008adb10 T cuserid
00941800 T daemon
...
```

마지막으로 스택이란 영역은 함수 호출과 관련된 정보들이 위치하게 되는 영역입니다.

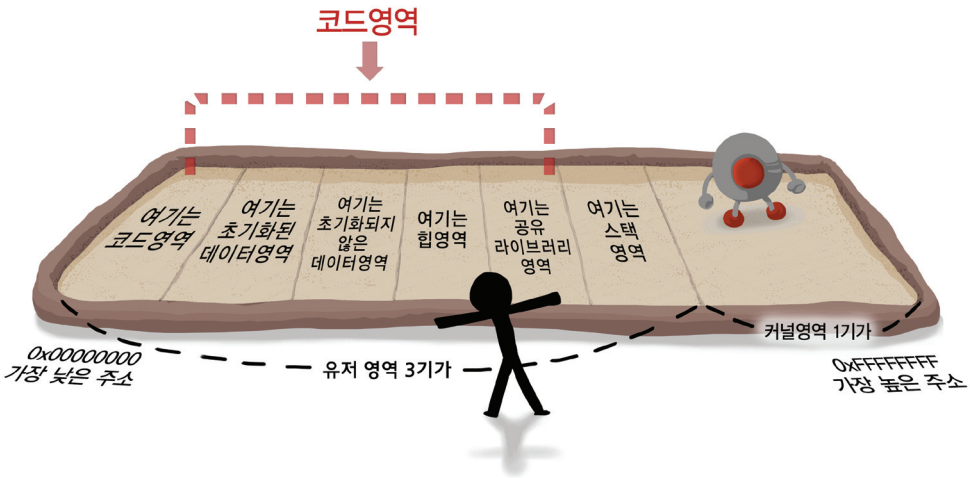


함수의 인자들, 리턴 어드레스, 그리고 함수 내에서 사용되는 지역 변수들이 바로 이곳에 저장됩니다. 그리고 그 외에도 몇 가지 값들이 스택 영역에 저장되어 있는데, 대표적인 예가 쉘의 환경변수(environment) 값들입니다.

이러한 값들은 모두 버퍼 오버플로우 공격과 밀접한 관계를 가지고 있는 것들이기 때문에, 우리가 가장 큰 관심을 갖고 들여다 봐야 할 영역이 되겠습니다.

이렇게 하여 유저 메모리 영역을 총 여섯 부분으로 나누어보았습니다.
참고로 유저 메모리 영역을 구분하는 방식은 서적이나 문서마다 조금씩 다를 수 있습니다.

예를 들어 코드 영역과 공유 라이브러리 영역을 합쳐서 그냥 하나의 코드 영역으로 보는 경우도 있고,



초기화된 데이터 영역과, 비초기화된 데이터 영역, 그리고 힙 영역을 통틀어서 하나의 큰 데이터 영역으로 보는 경우도 있으니 참고 바랍니다.