[1페이지]
안녕하십니까? 시스템 해킹 입문기를 발표 할 S.C.P 김원태입니다.

[2페이지]
목차는 다음과 같습니다. 우선 "시스템 해킹이 무엇인가?"에서 시스템 해킹에 대해 간단히 알아 볼 예정입니다. 두 번째 "메모리 구조"에서 스택과 스택 프레임 구조를 이해하기 위해 메모리의 구조에 대해 설명하겠습니다. 세 번째는 시스템 해킹을 시작하기 위해 필수인 스택과 스택 프레임에 관해 설명하겠습니다. 앞으로 시스템 해킹에 대해 어떻게 공부할 것인가를 끝으로 발표를 마치겠습니다.

[3페이지]
소제목 1

[4페이지]
우선 해킹이 무엇인지 알아야합니다. 해킹이란 프로그램 원 제작자가 걸어 놓은 프로그램 코드 락 알고리즘을 뚫어서 프로그램 소스를 알아내거나 프로그램 소스를 변경해서 자기 입맛에 맞게 바꾸는 모든 행위를 포함합니다. 이러한 해킹의 분야로 동아리에서 유명한 웹 해킹과 시스템 해킹을 볼 수 있습니다. 그 중 전 시스템 해킹을 선택하였고, 시스템 해킹은 운영 체제나 소프트웨어, 하드웨어에 내재된 보안 취약점을 해킹하는 것이라고 하며 흔히 리트에서 따온 pwn을 사용하여 pwnable, 즉 포너블이라고 부릅니다. 시스템 해킹은 크게 운영체제, API, 리버싱 분야를 공부해야 하며 어셈블리어를 잘 사용하는 것이 포인트입니다. 이렇게 시스템 해킹은 무엇인지에 대해 알아보았고 다음 단계인 "메모리 구조" 로 넘어가겠습니다.

[5페이지]
소제목 2

[6페이지]
다음에 나올 스택과 스택 프레임구조에 대해 이해하기 위해 우선적으로 메모리의 구조에 대한 이해가 필요합니다.

프로그램이 실행되기 위해서는 먼저 프로그램이 메모리에 로드 되어야 하고, 프로그램에서 사용되는 변수들을 저장할 메모리도 필요합니다.

대표적인 메모리 공간은 다음과 같습니다. 1. 코드 영역입니다. 텍스트 영역이라고도 불리우며 cpu는 코드 영역에 저장된 명령어를 한 개씩 가져가서 처리하게 됩니다.
두 번째는 데이터 영역입니다. 데이터 영역은 프로그램의 전역 변수와 정적 변수가 저장되는 영역으로 데이터의 영역은 프로그램의 시작과 함께 할당되며 종료 시 소멸하게 됩니다.
세 번째는 힙 영역입니다. 힙 영역은 사용자가 직접 관리할 수 있는 '그리고 해야만 하는' 영역입니다. 그 이유는 사용자에 의해 메모리 공간이 동적으로 할당되고 해제되며 메모리의 낮은 주소에서 높은 주소의 방향으로 할당하기 때문입니다.
마지막은 스택 영역입니다. 오늘 발표 주제와도 부합하며 가장 중요하다고 생각하는 부분입니다. 호출과 관계되는 지역 변수와 매개 변수가 저장되는 영역으로 스택 영역은 함수의 호출과 함께 할당되고, 함수의 호출이 완료되면 소멸하게됩니다. 이렇게 스택 영역에 저장되는 함수의 호출 정보를 스택 프레임이라고 합니다.
(스택 프레임은 뒤에서 더욱 자세하게 다루겠습니다.) 스택 영역은 push와 pop 명령어를 사용하여 데이터를 인출하게 됩니다.

이 처럼 메모리의 구조에 대해서 알아보았고 스택 영역으로 넘어가 발표를 이어가겠습니다.

[7페이지]
소제목 3

[8페이지]
스택은 사전적 의미로 "(보통 깔끔하게 정돈된) 무더기" 라는 뜻을 가지고 있습니다.

우리가 알아야 할 스택의 가장 큰 특징은 후입선출로 Last-in First-out,LIFO라는 특징이 있습니다. 말 그대로 마지막에 들어온 데이터가 가장 먼저 나간다는 의미로 그림과 같이 프링글스를 먹게 되면 가장 마지막에 들어온 과자가 가장 먼저 나오게 되는 구조이며, 옆에 책을 정리한다고 하였을 때, 맨 위에 있는 책은 가장 마지막에 정리된 책이고 가장 아래에 있는 책을 꺼내기 위해서는 가장 위의 책부터 차근차근 치워나가야하는 것과 같은 맥락입니다.

[9페이지]
스택의 구조를 이해했으니 스택 프레임인 함수 프롤로그와 함수 에필로그를 이해해야합니다.

우선 함수 프롤로그는 함수가 시작할 때 사용되고 push ebp를 진행한 후 mov ebp,esp를 진행합니다.

함수 에필로그는 프롤로그와 반대로 함수를 종료할 때 진행이 되고 mov esp,ebp를 사용해 esp의 값을 ebp에 넣어줌으로써 esp를 정리해주어 초기값으로 복원해준 후 pop ebp를 통해 리턴되기 이전에 저장해둔 ebp값으로 복원합니다.
마지막으로 ret를 통해 함수를 종료하게 됩니다. 말로는 쉽게 설명했지만 이해하는데 상당한 시간이 소요된 부분이었습니다.

[10페이지]
구조에 대해 그림으로 설명하기 전에 필요한 어셈블리어를 마지막으로 소개하겠습니다.

첫 번째로 esp가 있습니다. Extended Stack Pointer의 약자로 스택의 최상단 주소값을 갖고 있습니다. Esp는 스택 포인터 역할을 합니다. 스택의 크기를 조정할 때 사용되는 레지스터입니다.
다음은 ebp 입니다. Extended Base Pointer로 스택 프레임의 형태로 저장된 함수의 지역변수, 전달 인자를 참조하거나 값을 바꾸거나 할 때 사용하는 레지스터입니다. 예시로 함수의 시작의 esp값을 ebp에 저장하고 이를 함수 내에서 유지해주면 esp 값이 아무리 변하더라도 ebp를 기준으로 안전하게 해당 함수의 변수, 파라미터, 복귀 주소에 접근이 가능해집니다.
다음은 SFP입니다. 말 그대로 이전 함수의 ebp값을 저장하고 있는 값입니다. 그렇다고 "SFP=EBP" 는 옳지 않습니다. 이전 함수의 ebp를 sfp에 저장함으로써 안전하게 돌아올 수 있는 것이지 sfp가 ebp랑 같다고 볼 수는 없습니다.

EIP에 대해 설명하겠습니다. Extended instruction pointer의 약자로 다음 실행할 명령어의 주소값을 저장하고 있습니다. Eip가 있기에 완료된 함수 이후에 계속 이어질 수 있습니다. Ret는 return address로 호출이 종료되면 갖고 있는 주소값으로 jmp시킵니다.

[11페이지]
가장 먼저 메인함수가 쌓이게 되고 그에 따른 ret값이 쌓이게 됩니다. 그 다음 함수 프롤로그를 사용하여 push ebp를 사용해줍니다. MOV EBP, ESP 를 사용하여 EBP값에 ESP 값을 넣어줍니다.

EBP는 세 번째 그림과 같이 해당 베이스 포인터를 지키고 있고 위로 함수들이 실행될 때마다 ESP는 쌓였다 지워졌다를 반복하게 됩니다. 그 후 함수를 마무리 하게 되면 함수 에필로그를 진행 할 것입니다.

[12페이지]
함수의 진행이 끝났다는 가정 하에 함수 에필로그가 진행됩니다.
MOV ESP,EBP와 POP EBP과정은 차후 LEA는 명령어로 함축되는 것을 알아두면 좋습니다.
따라서 ESP의 값을 EBP에 넣어준 후 저장합니다.
그 후로 pop ebp를 진행하는 데 여기서 pop ebp는 리턴되기 전에 저장해 놓은 ebp값으로 복원하는 것입니다.
따라서 그림과 같이 ret값으로 ebp가 돌아오고 ret를 사용하여 main으로 돌아오게 됩니다.

이 부분이 단편적으로는 쉬워보이지만 여러 함수들이 쌓이고 삭제되다 보면 굉장히 헷갈립니다. 또한 스크립트로 작성하다보니 설명하는 것에 큰 어려움이 있습니다. 해당 내용에 대한 추가 설명이 필요 시 질문주시면 저의 지식 선에서 최대한으로 설명해드리겠습니다.

[13페이지]
소제목 4

[14페이지]
우선 이 번 발표를 준비하면서 스택프레임에서 스택이 쌓이고 삭제되는 부분에서 많은 삽질을 하게되어 분량이 많지 않아 죄송하다는 말씀드리고 싶습니다.
그래도 이제는 구조에 대해서 90%정도 이해한 상태이며 100%를 이해하게 되면 다음과 같이 Vmware Ubuntu를 활용하여 GDB 분석을 할 것입니다. 제가 이해했던 내용과 실제 GDB를 통한 분석을 했을 때 맞게 떨어지는 지를 확인해보겠습니다.
다음으로 FTZ, CTF 문제를 접해보려 합니다. 계속 공부하면서 느낀 것이지만 많은 이론 공부도 필요하지만 제가 실습해보지 않으면 도루묵이라는 생각이 들었습니다. 그래서 선배님들의 도움과 저의 시간 투자로 문제들을 풀어보고자 합니다.
마지막으로 시스템 해킹과 리버싱을 필연적으로 둘다 공부를 해야한다는 것을 느꼈습니다. 따라서 어렵지만 슬슬 리버싱 공부에도 시간을 투자해 볼 계획입니다. 한 번에 이 모든 일들을 수행 할 수는 없을 것입니다. 하지만 한 단계, 한 단계 해결해가며 성장해가는 저의 모습을 보고
성장 속도를 높여볼 것입니다. 감사합니다.

[15페이지]
마지막으로 QnA를 진행하고 발표 마무리하겠습니다.

~~~~~~~~Q1:~~~~
~~~~~~~~Answer:~~~~

넵 이상 발표 마치겠습니다. 지금까지 들어주셔서 감사합니다!!