발표 스크립트 - 0529

FSB_심화 문제풀이로 발표 시작하겠습니다.

목차입니다. 이번 발표는 Dreamhack에서 FSB문제 풀이 두 개를 가져와 풀어보았습니다.

basic_exploitation_002

먼저 basic_exploitation_002 입니다.

드림핵에서 basic_exploitation_002 문제를 선택하면 이렇게 문제 정보가 먼저 뜨게 됩니다.

문제 정보들을 읽고 문제 풀이 시작하겠습니다.

p.4 에서도 알 수 있지만 kali로 환경을 다시 한번 확인해 주었습니다.

Partial RELRO 로 GOT Overwrite 가 가능하고, NX enabled 로 NX 보호 기법이 활성화 되어있음과 동시에 쉘 코드 실행 불가능한 것을 볼 수 있습니다.

다음은 basic_exploitation_002 의 코드입니다.

이 코드 31줄에서 Read 로 0x80 만큼 문자열을 입력 받기 때문에 Buffer over Flow 는 발생하지 않습니다.

바로 밑 32줄에서 printf 를 buf 로 받기 때문에 Format String Bug 가 발생하게 됩니다.

이제 코드를 자세히 설명하겠습니다.

get_shell 함수는 system 함수로 shell 을 출력하고, flag 를 획득할 수 있는 함수입니다.

main 함수입니다.

buf 배열 0x80 (128byte) 를 할당하고 printf 함수로 buf 값을 출력합니다. exit 함수의 got 주소를 get_shell 주소로 변조를 해야 바이너리가 종료되지 않고 shell 이 실행될 수 있습니다.

바로 실습으로 넘어가도록 하겠습니다.

바이너리 실행

./basic_exploitatin_002 를 실행시킨 후 aaaaaaaaaaaaaa 를 입력해봤더니 그대로 사용자의 입력을 받아 출력되는 것을 알 수 있습니다.

이번엔 aaaa 와 포맷스트링 문자인 %x 를 입력해주었습니다.

32bit, 4byte 체제이기 때문에 aaaa 를 입력해 주었고 aaaa 를 그대로 입력 한 후 바로 뒤의 첫번째 offset 에 a 의 아스키코드인 61616161 이 저장되는 것을 볼 수 있습니다.

여기서 get_shell 을 실행시키기 위해서는 exit 를 이용해야합니다. 즉, exit 가 실행될 때 exit 대신 get_shell 이 실행되게 만들어줘야 하기 때문에 exit_got 에 get_shell 함수 주소로 Overwrite 해야합니다.

이렇기에 필요한 것은 exit_got 주소와 get_shell 주소입니다.

GDB 정적분석 - exit 함수

exit 함수를 알기 위해 gdb 로 disass exit 를 해준 후 exit_got 주소인 0x804a024 를 알아냈습니다.

GDB 정적 분석 - 함수 주소

get_shell 주소를 알아내기 위해 gdb 에서 info func 을 하여 함수들을 보고 찾아내 get_shell 주소인 0x0848609 를 알아냈습니다.

Exploit

exit_got 에 get_shell 주소를 %n 으로 한 번에 넘겨주기에는 0x80486609 = 134,514,185 로 너무 값이 크기 때문에 각 2byte 씩 잘라서 exit_got + 2 에는 0x804 를, exit_got 에는 0x8609 를 넘겨주었습니다.

Payload

주소를 적느라 8byte가 사용되었으므로 **%2052c(0x804)** 가 아닌 **%2044(0x804 - 0x8)** 를 사용합니다.

마찬가지로 0x8609 - 0x804(2052) = 32261byte 인 **%33261c** 를 사용합니다.

2byte 를 넘기므로 %hn 을 사용했고 %1$hn은 |exit_got + 2| 만. 즉, 처음 2byte 를 인자로 받겠다는 뜻입니다.

이렇듯 exit_got 에는 get_shell 함수 주소가 들어가게 되고, 마지막에 exit 가 실행되면 exit 대신에 get_shell 함수가 실행되게 됩니다.

Payload 코드 작성

payload 코드를 작성 하기 전 다시 드림핵 문제 페이지에서 접속 정보를 확인해 줍니다.

Vs code 를 wsl 로 연결 해 준 후 드림핵 접속 정보에 있는 호스트와 포트 번호를 입력해 준 후 위의 Payload 를 불러와서 연결시켜 주는 코드를 하나 만들어 줍니다.

Flag

다시 wsl 로 돌아와서 755 로 권한 부여를 해 준 후 Python 을 실행시켜 주면 플래그 값을 볼 수 있습니다.

그 플래그 값을 드림핵에 입력해서 제출하게 되면 문제가 성공적으로 풀리게 됩니다.

---

basic_exploitation_003

그 다음 basic_exploitation_003 입니다.

앞 순서와 동일하지만 하나씩 간단히 설명 하며 넘어가도록 하겠습니다.

마찬가지로 드림핵에서 basic_exploitation_003 문제를 선택하면 이렇게 문제 정보가 먼저 뜨게 됩니다.

문제 정보들을 읽고 문제 풀이 시작하겠습니다.

p.22 에서도 알 수 있지만 kali로 환경을 다시 한번 확인해 주었습니다.

Partial RELRO 로 GOT Overwrite 가 가능하고, NX enabled 로 NX 보호 기법이 활성화 되어있음과 동시에 쉘 코드 실행 불가능한 것을 볼 수 있습니다.

다음은 basic_exploitation_003 의 코드입니다.

이 코드 23줄에서 sprintf 를 통해 전달할 때 format string 을 이용하므로 format string bug 가 발생 가능하게 됩니다.

이제 코드를 자세히 설명하겠습니다.

get_shell 함수는 system 함수로 shell 을 출력하고, flag 를 획득할 수 있는 함수입니다.

main 함수입니다.

heap_buf 를 0x80 만큼 동적 할당 하고 steak_buf 를 0x90 만큼 할당합니다.

sprintf 함수를 이용하여 printf 함수의 got 주소나 return 주소를 get_shell 로 덮기 가능한 구조입니다.

바이너리 실행

./basic_exploitatin_003 를 실행시킨 후 aaaaaaaaaaaaaa 를 입력해봤더니 그대로 사용자의 입력을 받아 출력되는 것을 알 수 있습니다.

이번엔 aaaa 와 포맷스트링 문자인 %x 를 입력해주었습니다.

32bit, 4byte 체제이기 때문에 aaaa 를 입력해 주었고 aaaa 를 그대로 입력 한 후 바로 뒤의 첫번째 offset 에 a 의 아스키코드인 61616161 이 저장되는 것을 볼 수 있습니다.

GDB 정적분석 - 함수 주소

exit 함수를 알기 위해 gdb 로 disass printf 를 해준 후 printf_got 주소인 0x804a010 를 알아냈습니다.

get_shell 주소를 알아내기 위해 gdb 에서 info func 을 하여 함수들을 보고 찾아내 get_shell 주소인 0x0848669 를 알아냈습니다.

disass main 실행 결과로 보아 stack_buf 의 구조가 ebp-0x98 인 것을 알 수 있습니다.

이를 통해 stack_buf 가 ebp 로부터 0x98(152byte) 떨어져 있다는 것을 알 수 있습니다.

Exploit

스택 구조를 보면 stack_buf + SFP(4byte) + RET(4byte) 이므로 152 + 4byte 를 하여 156byte 만큼 채워 RET 부분에 get_shell 함수의 주소를 넣어줍니다.

Payload 코드 작성

payload 코드를 작성 하기 전 다시 드림핵 문제 페이지에서 접속 정보를 확인해 줍니다.

Vs code 를 wsl 로 연결 해 준 후 드림핵 접속 정보에 있는 호스트와 포트 번호를 입력해 준 후 위의 Payload 를 불러와서 연결시켜 주는 코드를 하나 만들어 줍니다.

Flag

다시 wsl 로 돌아와서 755 로 권한 부여를 해 준 후 Python 을 실행시켜 주면 플래그 값을 볼 수 있습니다.

그 플래그 값을 드림핵에 입력해서 제출하게 되면 문제가 성공적으로 풀리게 됩니다.

이상으로 발표 마치겠습니다.