OWASP

OWASP Seoul Chapter 7월 세미나
25/07/22 19:00

| Team EmulParty |

EmulParty Advance

: beyond x86 for beginners

경량 ISA 기반 오픈소스 보안 실습 아키텍처

이재원(PM) · 조용진(개발 총괄)

문석주 멘토 · 장형범 PL
서희영 · 강다윤 · 최민준 · 박정은 · 한상기

GIF/영상은
하단 링크를 참고해주세요

화이트햇 스쿨
WhiteHat School

발표 자료 링크 : https://docs.google.com/presentation/d/1rDqCGIxBH0cDRgvbrjqvA1aKgIxgi5tfms-ejChVf1E/edit?usp=sharing

EmulParty

EMP

WhiteHatSchool Gen3 Team Project

에뮬레이터Emulator × 아모르파티AmorFati

에뮤르파티

화이트햇 스쿨 3기 팀 프로젝트
「미니 에뮬레이터 만들기」

문석주 Mentor

장형범 PL

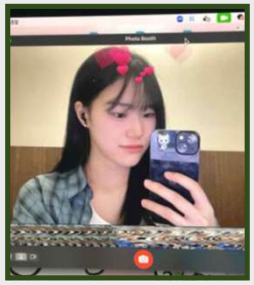
발 표

이재원 PM

발 표

조용진 Dev.

서희영 Dev.

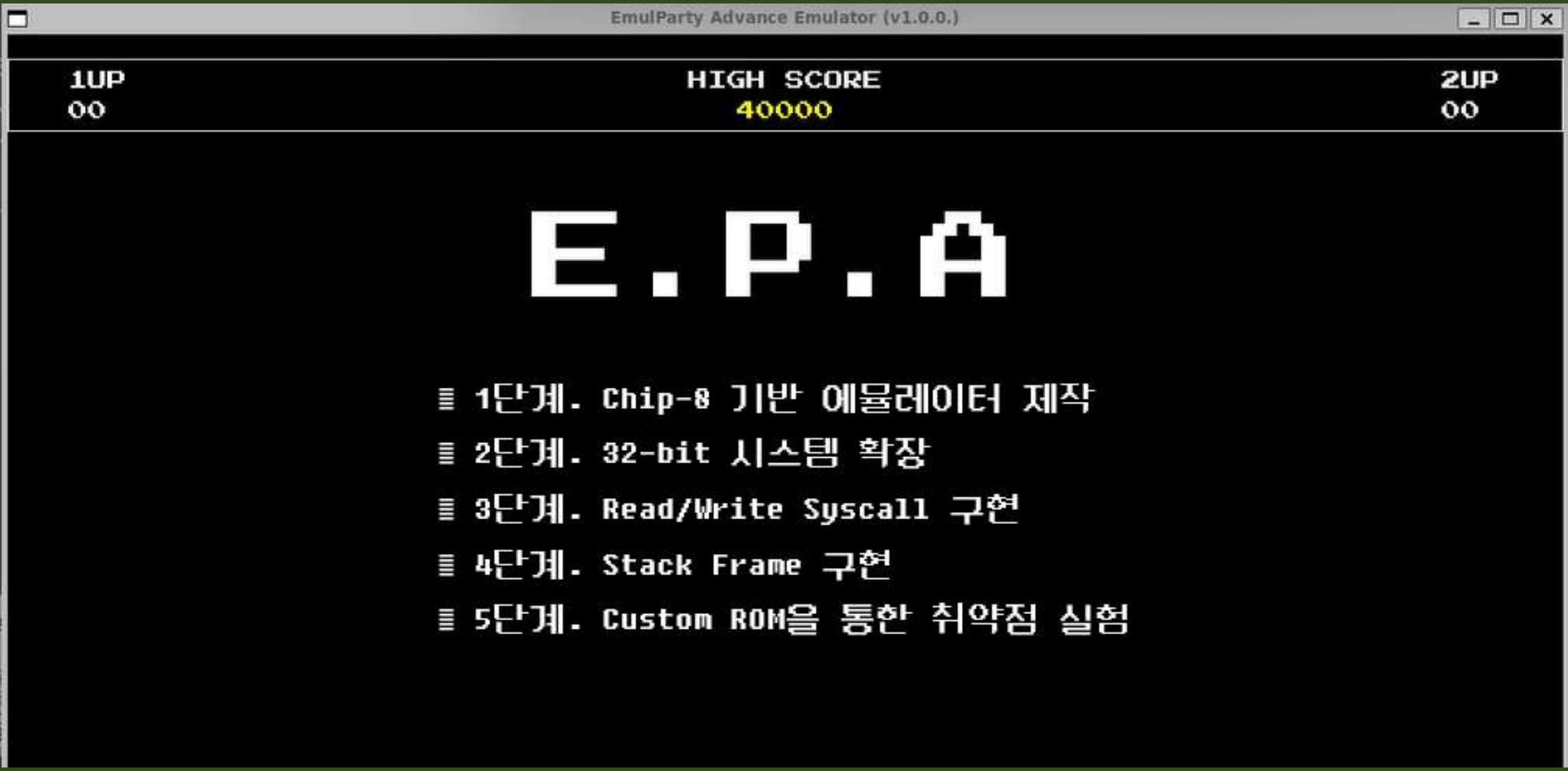


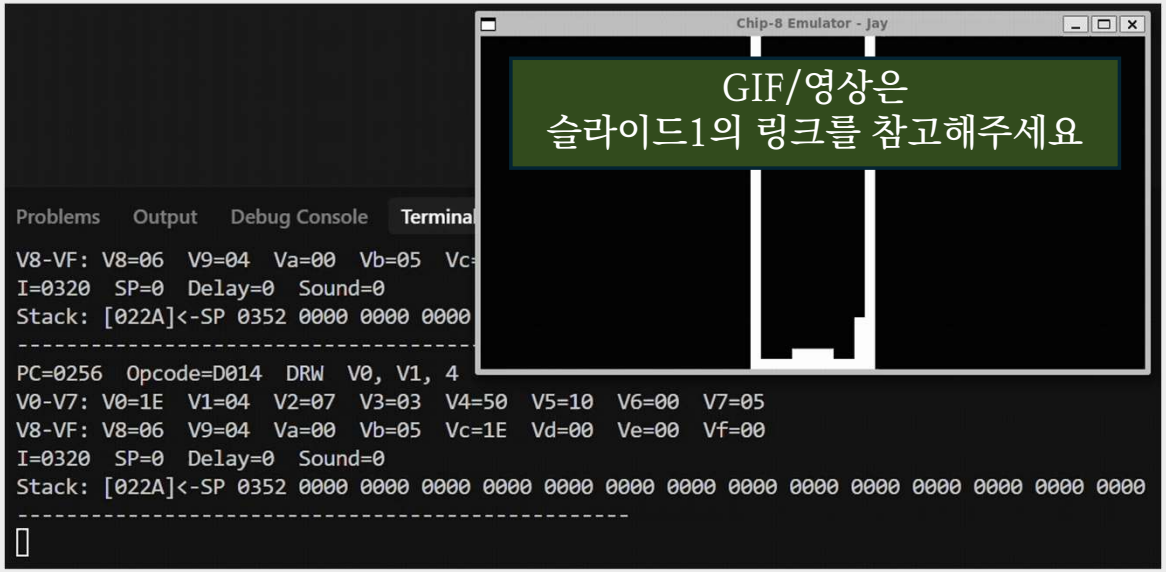
강다윤 Dev.

최민준 Dev.

박정은 Dev.

한상기 Dev.


EmulParty Advance Emulator (v1.0.0.)
1UP
00
HIGH SCORE
40000
2UP
00
E.P.A
1단계. Chip-8 기반 에뮬레이터 제작
2단계. 32-bit 시스템 확장
3단계. Read/Write Syscall 구현
4단계. Stack Frame 구현
5단계. Custom ROM을 통한 취약점 실험



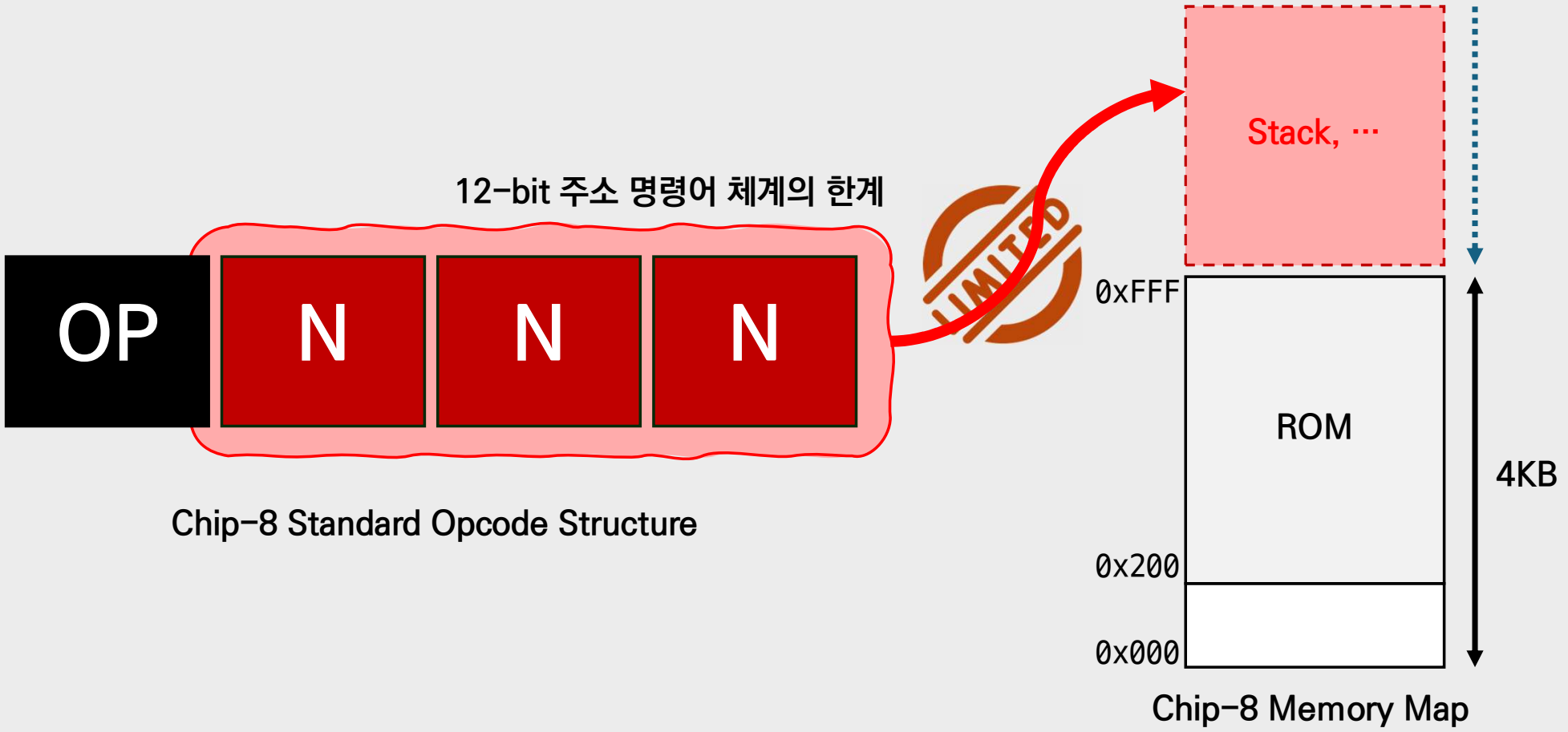
▲ Chip-8 Emulator 디버거 실행 장면 (tetris.ch8 ROM)

```
PC=0200  Opcode=A2B4  LD   I, 02B4
V0-V7: V0=00  V1=00  V2=00  V3=00  V4=00  V5=00  V6=00  V7=00
V8-VF: V8=00  V9=00  Va=00  Vb=00  Vc=00  Vd=00  Ve=00  Vf=00
I=0000  SP=0  Delay=0  Sound=0
Stack: [0000]<-SP 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
```

A2B4	LD I, addr	ANNN - 인덱스 레지스터 I에 0xNNN을 로드

인덱스 레지스터 I에 0x2B4를 로딩한다.

디버거 & ROM 실행
→ 동작 분석 가능

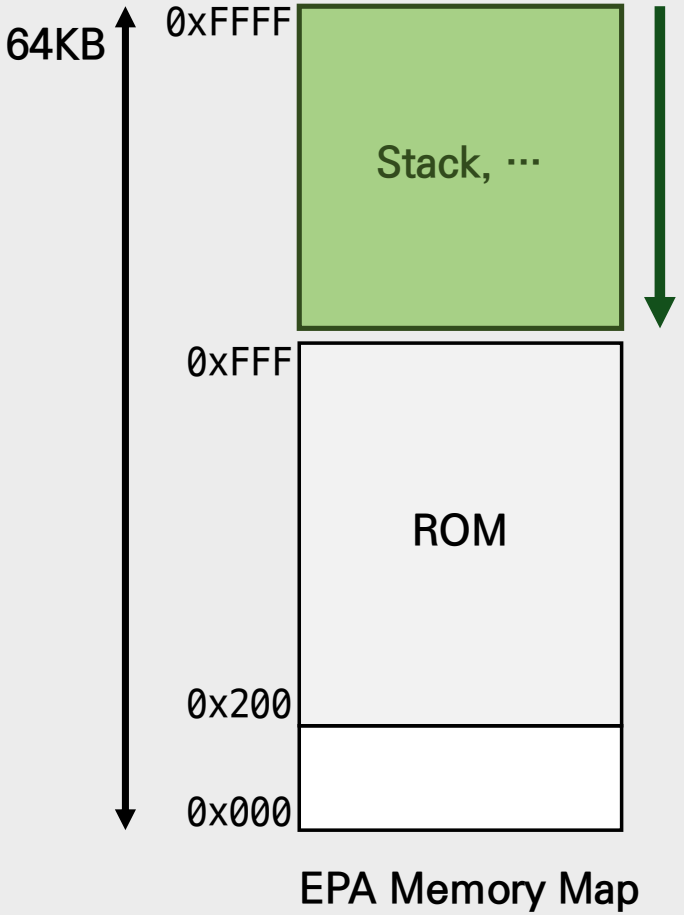


Chip-8 Standard Opcode Structure

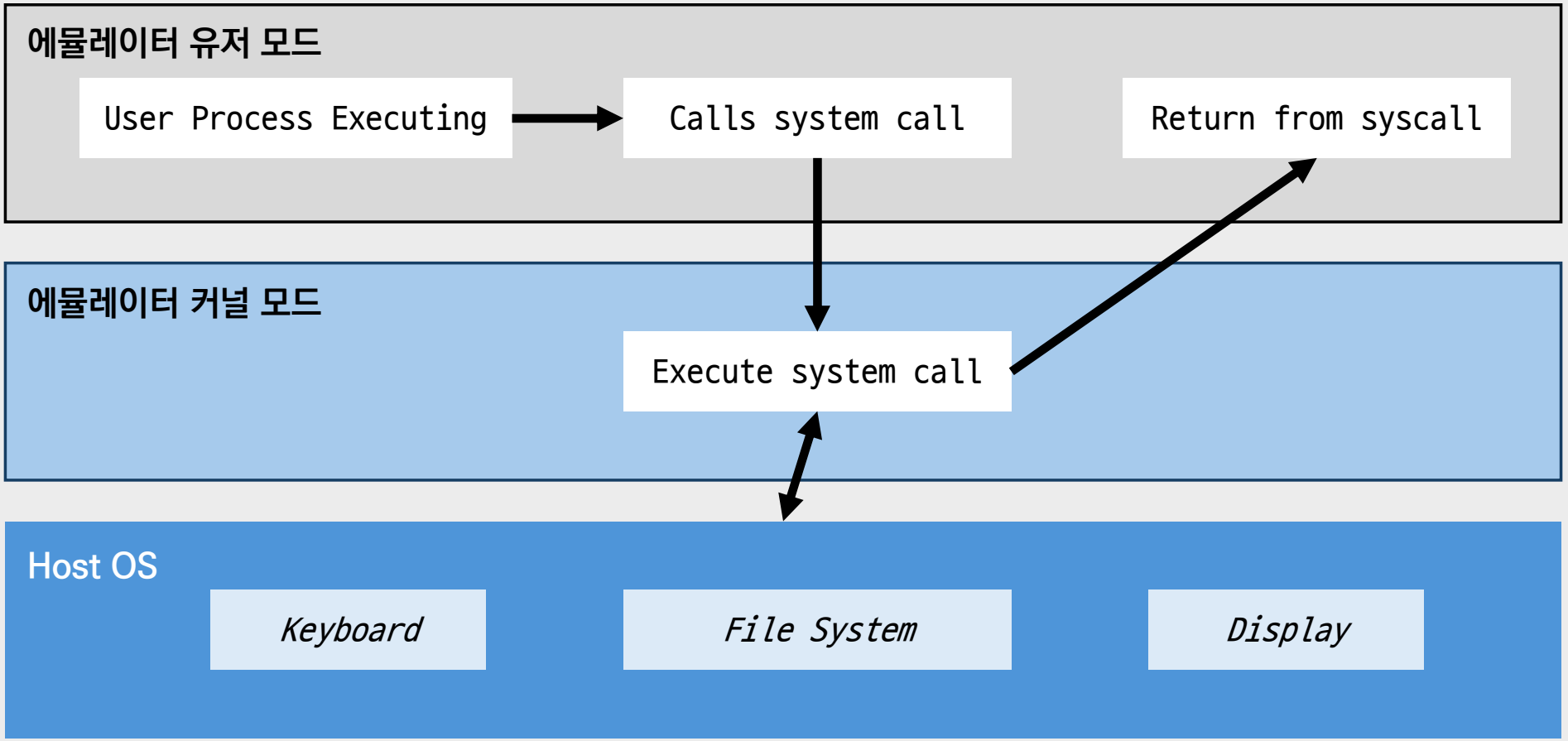
Chip-8 Memory Map

명령어 한계 극복 · 직관적인 고정 길이 명령어 체계 유지

EPA Extended Opcode Structure

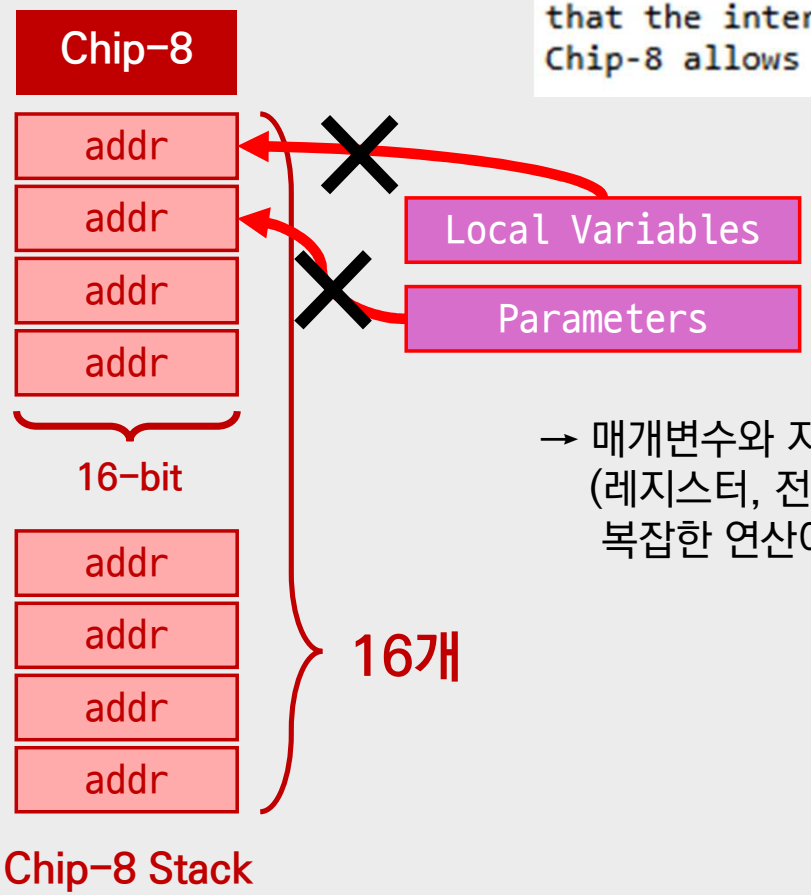


EPA Memory Map

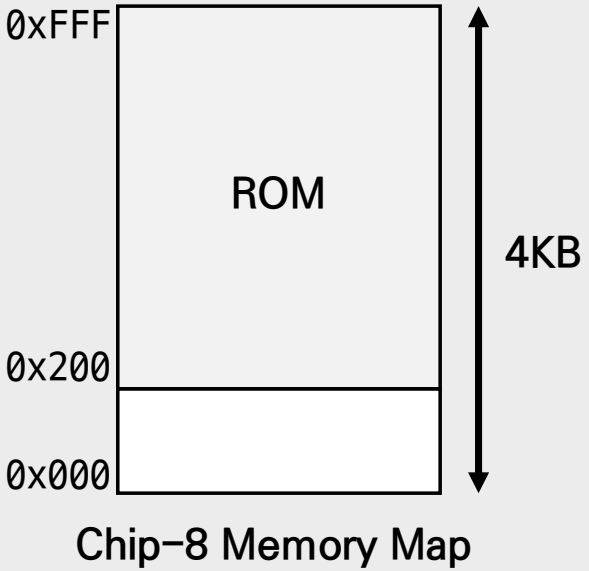

에뮬레이터 유저 모드
User Process Executing
Calls system call
Return from syscall
에뮬레이터 커널 모드
Execute system call
Host OS
Keyboard
File System
Display

GIF/영상은
슬라이드1의 링크를 참고해주세요

The stack is an array of 16 16-bit values, used to store the address that the interpreter shoud return to when finished with a subroutine. Chip-8 allows for up to 16 levels of nested subroutines.



→ 매개변수와 지역변수 저장 불가
(레지스터, 전역 메모리 과부하로
복잡한 연산이나 재귀 호출 어려움)



Stack Frame의 본질

EBP를 기준으로 “각 함수 호출마다 독립된 작업 공간“ 확보

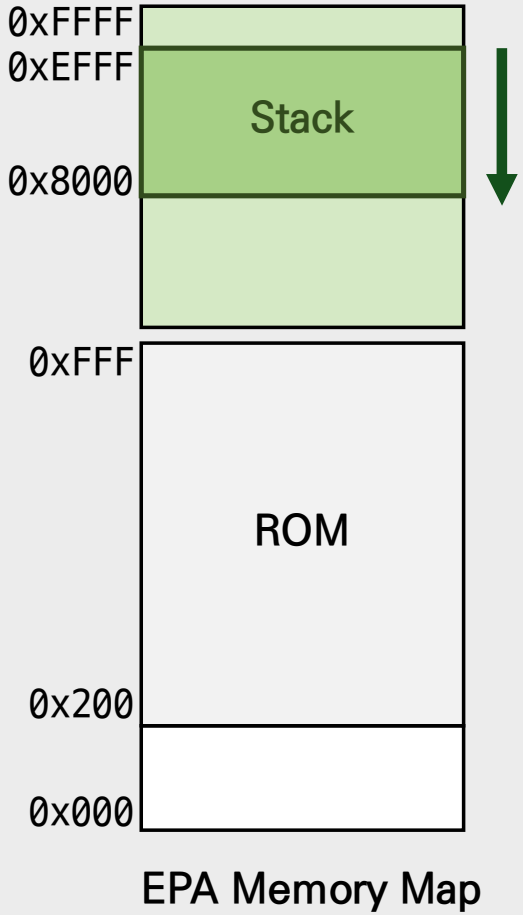
x86-64의 호출 규약을 참고(caller cleanup)

프레임 프롤로그

```
push ebp
mov  ebp, esp
sub  esp, <locals>
```

프레임 에필로그

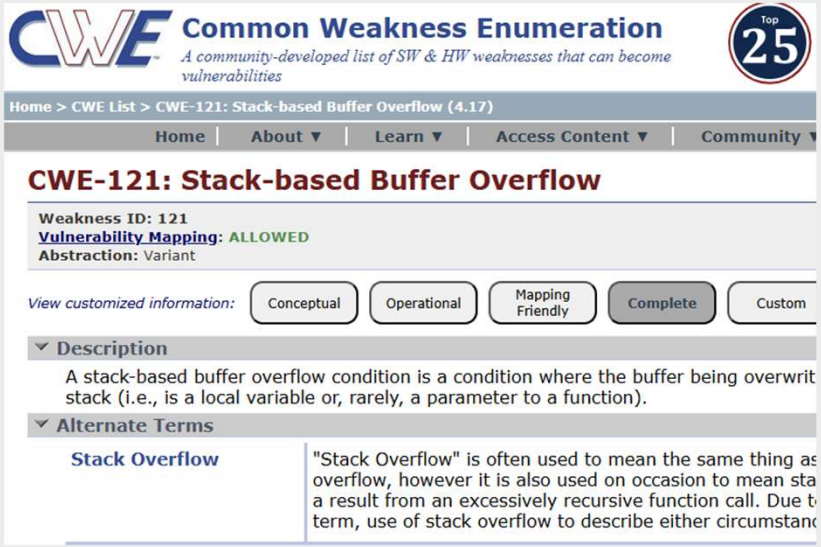
```
mov  esp, ebp
pop  ebp
ret  <n>
```



EPA Memory Map

▶ 중첩된 호출이 일어나도 함수마다 독립된 스택 구역 사용

▶ 복귀 시 ebp 체인을 따라가며 콜 스택을 추적할 수 있음



✅ **주어진 환경에서 실험 가능한가?**

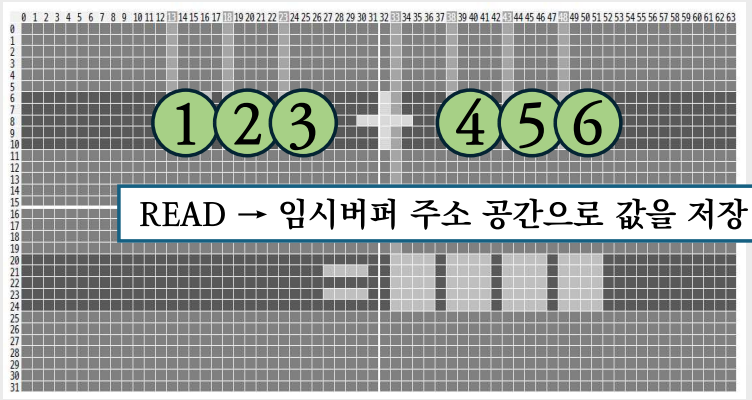
Read Syscall | Stack Frame | Write Syscall

✅ **지금까지 구현한 디버거 사양으로 원리를 보여줄 수 있는가?**

레지스터 | 스택 프레임

> **스택**에 할당된 버퍼(지역변수나 함수 인수)가 고정 크기를 초과하는 **입력**으로
> 다른 메모리를 덮어쓰면서, 그 뒤에 저장된 중요한 데이터를 변조할 수 있는 상태.

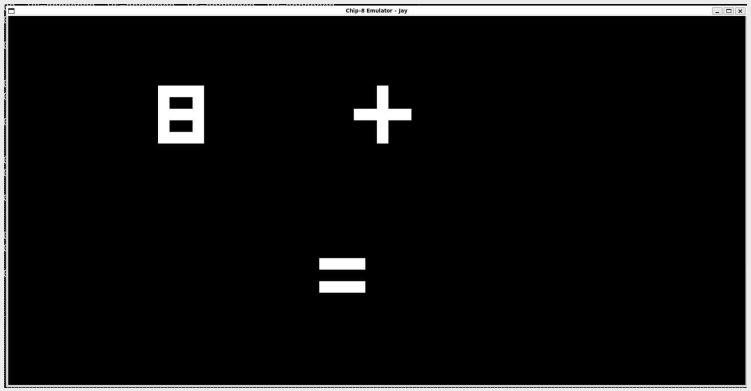
목표 : READ & 스택 프레임 동작을 활용해서 덧셈 계산기 만들기 (*실험 고려)



Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0600	01	02	03													
0610																
0620	04	05	06													
0630																

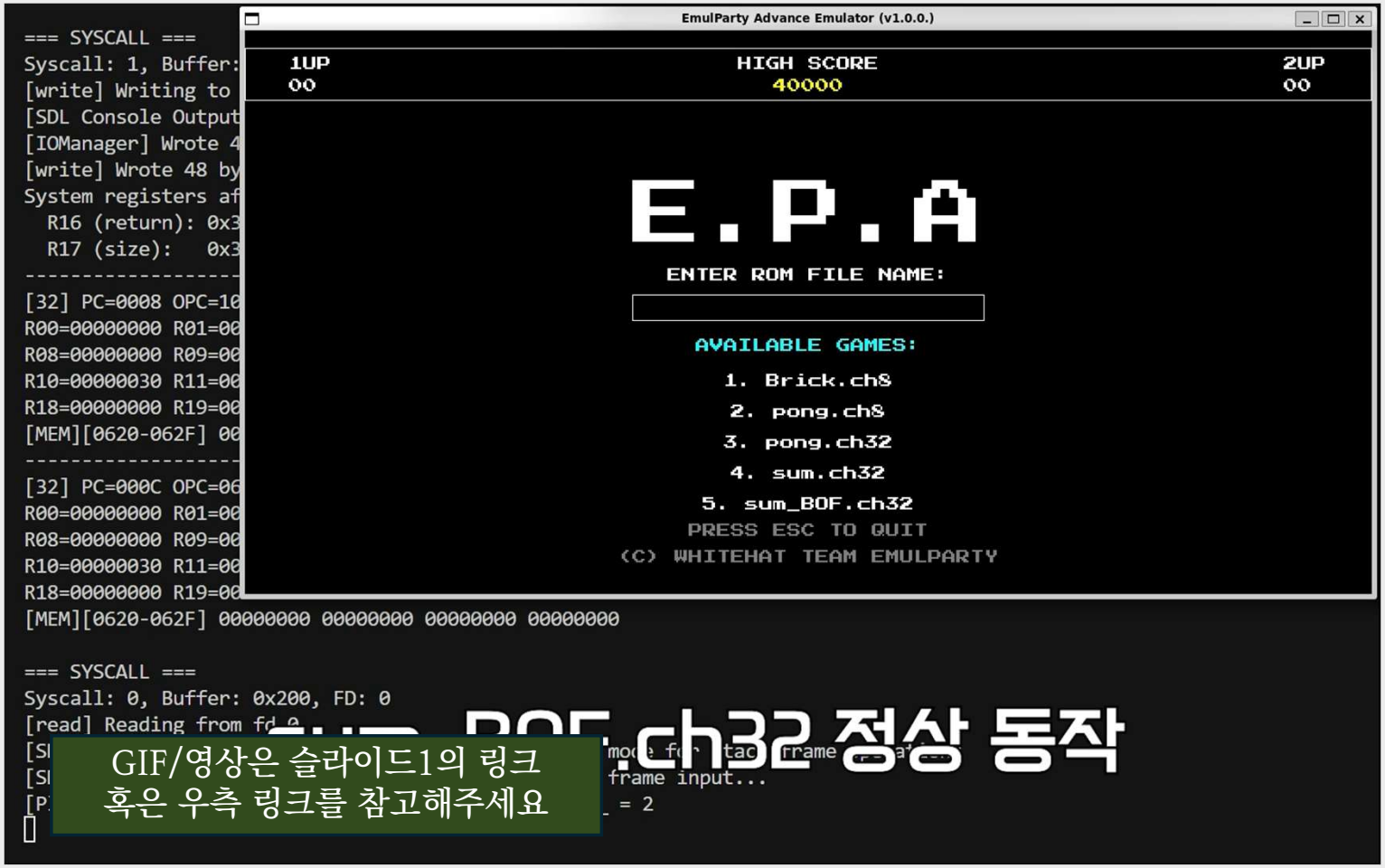
E(Enter) 키로 sum 동작을 스택 프레임에서 수행

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
EFE0																
EFF0	a	r	g	2	a	r	g	1	이	전	B	P	리	턴	주	소
	04	05	06	-	01	02	03	-	00	00	F0	00	00	00	?	?



[READ] 키패드 입력 → 임시 버퍼 주소 공간에 값 저장 → 스택 프레임으로 입력 값 복사 → CALL Func(덧셈 함수) → 스택 프레임 내의 인자 접근 및 계산 → 결과 리턴

(* x86 함수 호출 규약상 함수 인자는 호출 전에 스택에 PUSH, CALL 시에 스택 프레임 생성)



디버거 작동 시 하단 Stack Frame 영역 메모리에 주목

[HOW TO PLAY sum_BOF.ch32]

- ▼ 피연산자를 나란히 입력
- ▼ E('F'에 맵핑, Enter)키로 계산

https://youtu.be/tdlNDdrqbrM

디버거 작동 시 하단 Stack Frame 영역 메모리에 주목

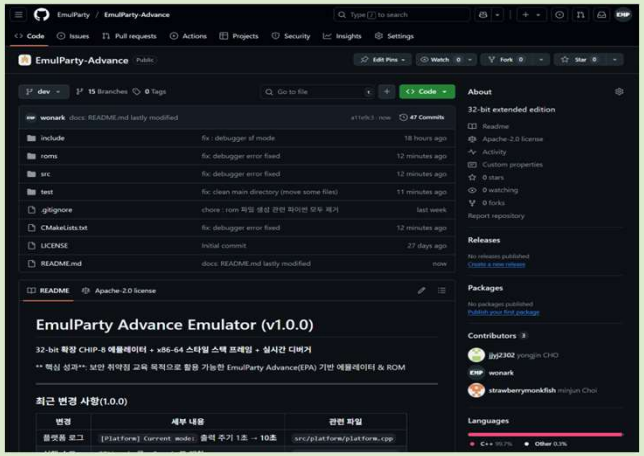
[HOW TO PLAY sum_BOF.ch32]

- ▼ 피연산자를 나란히 입력
- ▼ **뒤에 악의적인 값을 입력**
- ▼ E('F'에 맵핑, Enter)키로 계산

https://youtu.be/zTfB_F_8Ik4

open source

GitHub



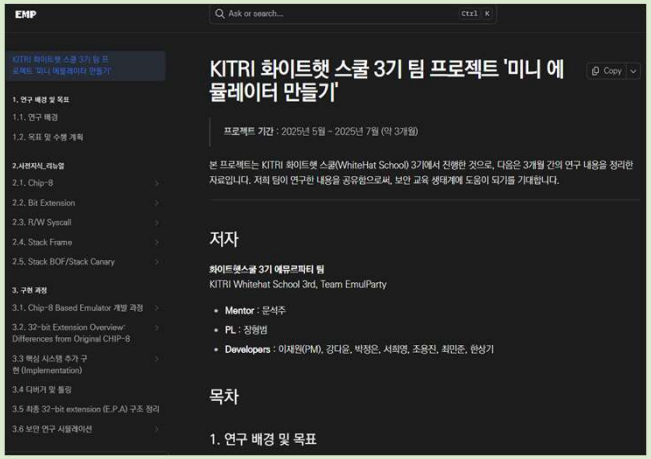
GitHub 메인 페이지

#README.md #LICENSE

`EmulParty-Advance`

소스코드 공개

GitBook



GitBook 메인 페이지

#연구배경및목표
#사전지식 #구현과정
#결론

프로젝트 내역 공개

Docs

ROM 가이드 문서

#ROM설계법 #도표설명포함
#Custom명령어활용법

입문자 학습용 문서 공개

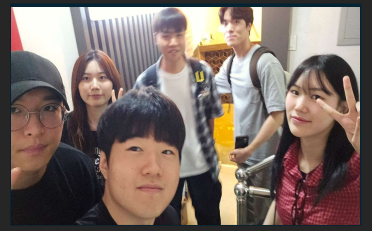
5. 결론과 느낀 점

1 Chip-8 기반 에뮬레이터 만들기

2 취약점 실험 고려 VM/ISA 확장 + 에뮬레이터 확장

3 보안 취약점 실험 설계 · 시연

open source™



OWASP

| Team EmulParty |

EmulParty Advance

: beyond x86 for beginners

경량 ISA 기반 오픈소스 보안 실습 아키텍처

Team EmulParty

문석주 멘토 · 장형범 PL

이재원 · 조용진 · 서희영 · 강다윤 · 최민준 · 박정은 · 한상기

QnA

질의응답

화이트햇 스쿨
WhiteHat School