




Graphic Era
deemed to be **University**
DEHRADUN




PROJECT AND TEAM INFORMATION

Project Title

Buffer Overflow Prevention in XV6 Using ASLR

Student/Team Information

Team Name: Team #	Berserk <i>SE(OS)-VI-T088</i>
Team member 1 (Team Lead)	<i>Sharma, Paras – 220211633</i> ps8798976@gmail.com 

Team member 2	<p>Sharma, Prerak – 22021884 sharmaprerak76@gmail.com</p> 
Team member 3	<p>Singh, Himanshu – 22021438 0614himanshusingh@gmail.com</p> 
Team member 4	<p>Bedi, Shreyas – 22021729 shreyasbedi9@gmail.com</p> 

PROJECT PROGRESS DESCRIPTION (35 pts)

Project Abstract (2 pts)

This project focuses on demonstrating and preventing buffer overflow attacks in the XV6 operating system. The goal is to simulate real-world vulnerabilities in a minimal OS environment and implement Address Space Layout Randomization (ASLR) to mitigate these risks. We began by crafting a working buffer overflow exploit in XV6 and now aim to randomize memory layout (stack/heap base) to prevent successful attacks. This project helps us understand low-level security threats and equips us with practical kernel development skills.

Updated Project Approach and Architecture (2 pts)

We designed a custom random number generator (rand.c) and integrated it into the xv6 kernel. We modified exec.c to add a random offset to the user stack during process execution. The overflow.c program simulates an attack, and ASLR behavior is confirmed through dynamic stack offset values and system behavior. Architecture now includes user-level testing, stack offset generation, and ASLR-based layout shifting.

Tasks Completed (7 pts)

Task Completed	Team Member
➔ Designed random number generator (rand.c) and integrated it into kernel build.	Paras Sharma
➔ Created buffer overflow exploit and test cases. Validated ASLR via shell.	Shreyas Bedi
➔ Modified exec.c for applying ASLR logic during process exec.	Himanshu Singh
➔ Troubleshooting, documentation, prepared test scenarios and results.	Prerak Sharma

Challenges/Roadblocks (7 pts)

- ➔ XV6 works with older GCC versions. We downgraded to GCC 4.8.
- ➔ Inserting stack canary checks into the kernel led to panics; we resolved this via step-wise testing and conditional code.
- ➔ XV6 has a static layout. Achieving ASLR involved deeply modifying memory setup in exec.c, allocvm(), and related calls.
- ➔ Measuring actual entropy required debugging memory locations at runtime.
- ➔ Coordinated changes across kernel, linker script, and user program required GitHub branch discipline.

Future Scope (7 pts)

Task Pending	Team Member (to complete the task)
➔ Complete kernel support for ASLR	Shreyas, Prerak
➔ Automate exploit vs ASLR validation	Himanshu
➔ Final testing and performance impact study	Paras
➔ Full documentation & presentation	Entire Team

Project Outcome/Deliverables (2 pts)

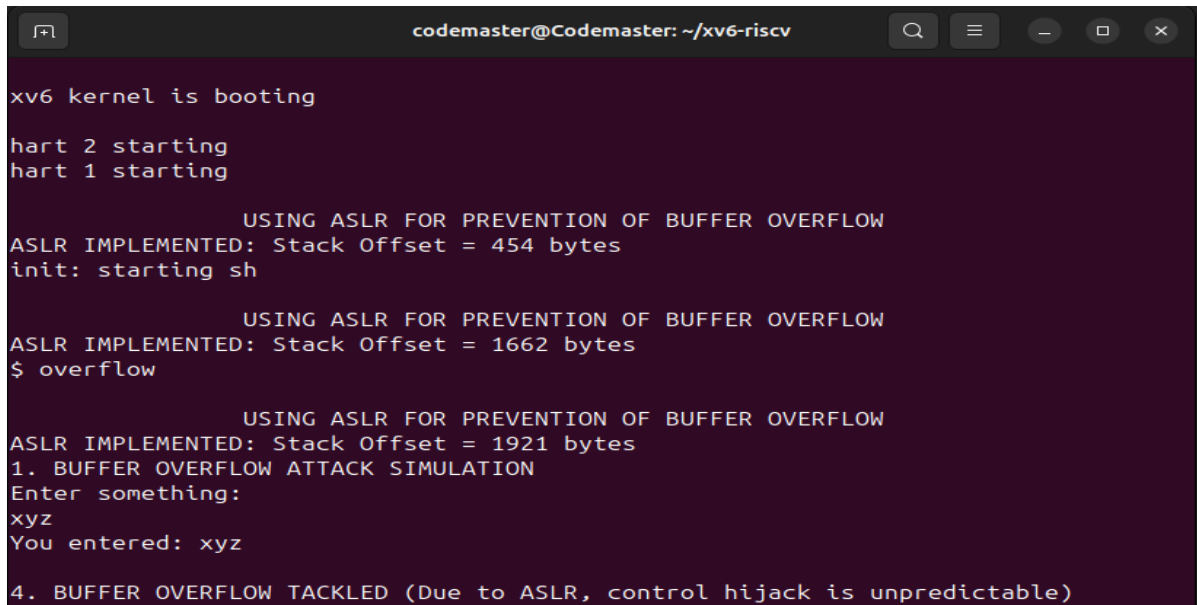
- ➔ Working buffer overflow simulation in xv6
- ➔ xv6 kernel modified to support ASLR
- ➔ Test logs showing randomized stack offsets
- ➔ Overflow crash behavior (usertrap) under ASLR
- ➔ Final report and presentation

Progress Overview (2 pts)

We have completed our project. ASLR is implemented and partially tested. The overflow program works as expected, and stack offset varies on each run. Final testing and documentation are in progress.

Example 1:

- Normal Input (xyz)
- ASLR enabled (random stack offset).
- Input handled safely.
- Buffer overflow tackled successfully.



```

codemaster@Codemaster: ~/xv6-riscv

xv6 kernel is booting

hart 2 starting
hart 1 starting

        USING ASLR FOR PREVENTION OF BUFFER OVERFLOW
ASLR IMPLEMENTED: Stack Offset = 454 bytes
init: starting sh

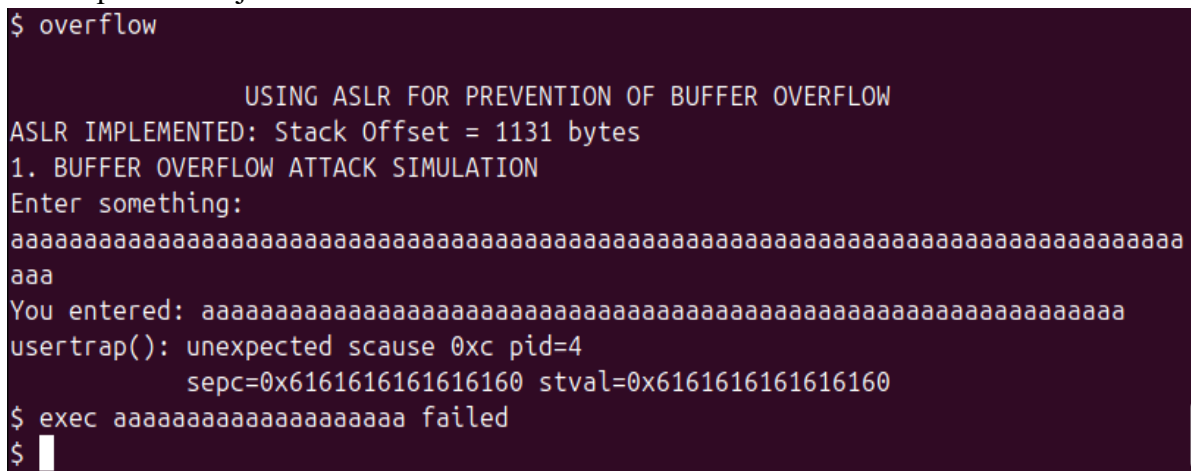
        USING ASLR FOR PREVENTION OF BUFFER OVERFLOW
ASLR IMPLEMENTED: Stack Offset = 1662 bytes
$ overflow

        USING ASLR FOR PREVENTION OF BUFFER OVERFLOW
ASLR IMPLEMENTED: Stack Offset = 1921 bytes
1. BUFFER OVERFLOW ATTACK SIMULATION
Enter something:
xyz
You entered: xyz

4. BUFFER OVERFLOW TACKLED (Due to ASLR, control hijack is unpredictable)
  
```

Example 2:

- Overflow Input (long 'a's)
- ASLR enabled.
- Buffer overflow causes crash.
- ASLR prevents hijack → attack fails.



```

$ overflow

        USING ASLR FOR PREVENTION OF BUFFER OVERFLOW
ASLR IMPLEMENTED: Stack Offset = 1131 bytes
1. BUFFER OVERFLOW ATTACK SIMULATION
Enter something:
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaa
You entered: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
usertrap(): unexpected scause 0xc pid=4
          sepc=0x6161616161616160 stval=0x6161616161616160
$ exec aaaaaaaaaaaaaaaaaaaaaa failed
$ 
  
```

Codebase Information (2 pts)

1.Repository Link

<https://github.com/ParasSharma101/buffer-overflow-prevention-in-xv6>

2.Branch

main

3.Important Commits:

- Initial commit with xv6 base setup
- Modified exec.c file to implement ASLR
- Added rand.c and rand.h for random number generation in kernel
- Modified Makefile and integrated ASLR support
- Added overflow.c user program for buffer overflow simulation

4.How to Run:

Navigate to the xv6-riscv directory.

Run make clean and then make qemu.

At the xv6 shell, execute overflow to test the implementation.

Testing and Validation Status (2 pts)

Test Type	Status (Pass/Fail)	Notes
➔ Overflow on original XV6	Pass	Return address hijacked → crash
➔ Overflow with AsLR enabled	Pass	Random offsets → unpredictable hijack
➔ Normal input with ASLR	Pass	Handled safely, program exits normally

Deliverables Progress (2 pts)

All core components of the project have been successfully completed. ASLR has been fully implemented in the operating system to randomize the stack position during process creation, making buffer overflow attacks less predictable and more difficult to exploit.

A custom test program was developed to simulate buffer overflow, and the system behavior was observed under various input conditions. Output logs confirmed that the memory layout changes with each run, demonstrating the effectiveness of ASLR.

All required components, including testing, documentation, GitHub codebase, and the presentation, have been finalized and are ready for evaluation.