


# Homework 3

## ▼ An empirical study of the reliability of UNIX utilities

 An empirical study of the reliability of UNIX utilities.pdf [https://drive.google.com/file/d/1LAYdjt7gUWPMjsU5MnYYc\\_hTlybjbKOU/view?usp=drivesdk](https://drive.google.com/file/d/1LAYdjt7gUWPMjsU5MnYYc_hTlybjbKOU/view?usp=drivesdk)

- What experience motivate the authors to conduct this more systematic experiment? Please explain what was the source of random inputs in that experience.
  - One of the authors was connected to workstation from home using a dial up line. There was rain and storm. This was causing phone line to transmit spurious characters. The author was barely able to write a command before the noise scrambled it. This noise caused the system crash which was not expected. Some minimal error message and system exit was expected, however, due to this, the system crashed. This led them to believe there might be serious bugs in the system.
  - The source of random noise was the rain that was messing with the dial up phone line.
- What is the most common cause for crash?
  - Random or unexpected input is the most common reason to crash.

## ▼ EXE: Automatically Generating Inputs of Death


 EXE Automatically Generating Inputs of Death.pdf [https://drive.google.com/file/d/1puriyqCt8Ot6rW1VwN3yQGNC\\_RrCczvFC/view?usp=drivesdk](https://drive.google.com/file/d/1puriyqCt8Ot6rW1VwN3yQGNC_RrCczvFC/view?usp=drivesdk)

- How does EXE (or what does exe-cc insert to) find inputs that can crash the program?
  - The EXE compiler check around all assignments, expressions and branches to determine if the operand is symbolic or concrete. For concrete no, inputs

are generated. For symbolic, constraints are generated as to what the input can be. This also takes into account any asserts the programmer may have added. Using this process, the compiler passes all the symbolic operands and their constraints to the EXE runtime system and then the system tries all the inputs, initially without constraints and then with constraints to get paths that may exit or have error or hang.

- How does EXE map C code to symbolic constraints?
  - For any symbolic input, it is passed to EXE runtime. This adds constraints for the current path of the branch and later executes all the path of the constraints by forking the process. At the time of execution, EXE represents symbolic data as array of 8-bit vectors. First it checks for what memory locations in checked code hold symbolic values. Then it translates expression to bit vector based on constraints.

## ▼ A First Step Towards Automated Detection of Buffer Overrun Vulnerabilities

 [A-First-Step-Towards-Automated-Detection-of-Buffer-Overrun-Vulnerabilities-Paper-David-Wagner.pdf](https://drive.google.com/file/d/1QtDakzwXEi0J3s3QKRugBFCAoinEMuRE/view?usp=drive_sdk)

- In the "Smashing the Stack for Fun and Profit" paper, the author suggested using grep to find use of dangerous libc APIs like strcat as potential location for buffer overflow. Compare to that simple static analysis (i.e., grep strcat), how does the approach proposed in this paper improves the precision and reduces false positives?
  - Grep can result in large number of false positives that need human intervention to classify them as vulnerable or not. The approach proposed by the authors to treat strings like abstract data types, create direction insensitive graphs and use integer range constraints for strings will reduce such false positives and also increase scalability.
- Static analysis trades precision for scalability, give an example of the imprecise modeling (i.e., heuristics) discussed in this paper.

- The paper says to solve integer range constraints, to solve for counting to infinity problem, we could introduce a widening operator that raises variables involved in cycles to  $[-\infty, \infty]$