CSci 8271
Security and Privacy in Computing
Day 16: Token-level fuzzing

Stephen McCamant
University of Minnesota

Input fuzzing in user space

- Testing code like libraries or parsers by randomly generating inputs
- Commonly, start with real seed inputs, then randomly modify (mutate) them
- Coverage (AFL) or other feedback guides the search in interesting directions

Trade-offs of byte mutation and grammars

- Mutations are often just byte-level changes
  - Replace, flip bit, insert, delete, splice
- Alternative: have the fuzzing tool know the input grammar
  - Can be used for pure generation, or grammar-aware mutation
- Common disadvantages of grammar-based fuzzing:
  - Building the grammar is extra work, and it can be wrong
  - The grammar may miss interesting (e.g., illegal) inputs

Token-level fuzzing approach, JavaScript

- Middle ground: fuzz input as a sequence of tokens
  - Legal tokens are curated similar to a grammar
  - Token sequence mutated like a byte sequence
- Security application: JavaScript JITs
  - Security sensitive because JS comes from untrusted sites
  - Relatively complex language and DOM

Comparison and bug-finding results

- Applied to JS engines of top-4 browsers
  - Generally favorable comparison with vanilla AFL, grammar-based tools
  - CodeAlchemist with a large seed set gets better block coverage
- Also including a 60-day solo run, 27 unique new bugs found
  - Over $10,000 in bug bounties

Future generalization and improvement

- New applications would need token definitions, still easier than a full grammar
- Could combine with smarter seed selection
- Apply AFL scalability changes for many edges and queue entries
- Ensemble approach with other JS fuzzers