/fuzzing-dictionary

Source: `~/.claude/skills/tob-testing-handbook-skills/skills/fuzzing-dictionary/SKILL.md`

---

name: fuzzing-dictionary type: technique description: > Fuzzing dictionaries guide fuzzers with domain-specific tokens. Use when fuzzing parsers, protocols, or format-specific code.

Fuzzing Dictionary

A fuzzing dictionary provides domain-specific tokens to guide the fuzzer toward interesting inputs. Instead of purely random mutations, the fuzzer incorporates known keywords, magic numbers, protocol commands, and format-specific strings that are more likely to reach deeper code paths in parsers, protocol handlers, and file format processors.

Overview

Dictionaries are text files containing quoted strings that represent meaningful tokens for your target. They help fuzzers bypass early validation checks and explore code paths that would be difficult to reach through blind mutation alone.

Key Concepts

Concept	Description
Dictionary Entry	A quoted string (e.g., `"keyword"`) or key-value pair (e.g., `kw="value"`)
Hex Escapes	Byte sequences like `"\xF7\xF8"` for non-printable characters
Token Injection	Fuzzer inserts dictionary entries into generated inputs
Cross-Fuzzer Format	Dictionary files work with libFuzzer, AFL++, and cargo-fuzz

When to Apply

Apply this technique when:

- Fuzzing parsers (JSON, XML, config files)
- Fuzzing protocol implementations (HTTP, DNS, custom protocols)
- Fuzzing file format handlers (PNG, PDF, media codecs)
- Coverage plateaus early without reaching deeper logic
- Target code checks for specific keywords or magic values

Skip this technique when:

- Fuzzing pure algorithms without format expectations
- Target has no keyword-based parsing
- Corpus already achieves high coverage

Quick Reference

Task	Command/Pattern
Use with libFuzzer	`./fuzz -dict=./dictionary.dict ...`
Use with AFL++	`afl-fuzz -x ./dictionary.dict ...`
Use with cargo-fuzz	`cargo fuzz run fuzz_target -- -dict=./dictionary.dict`
Extract from header	`grep -o '".*"' header.h > header.dict`

Task	Command/Pattern
Generate from binary	`strings ./binary | sed 's/^/"&/; s/$/&"/' > strings.dict`

Step-by-Step

Step 1: Create Dictionary File

Create a text file with quoted strings on each line. Use comments (`#`) for documentation.

Example dictionary format:

```
# Lines starting with '#' and empty lines are ignored.

# Adds "blah" (w/o quotes) to the dictionary.
kw1="blah"
# Use \\ for backslash and \" for quotes.
kw2="\"ac\\dc\""
# Use \xAB for hex values
kw3="\xF7\xF8"
# the name of the keyword followed by '=' may be omitted:
"foo\x0Abar"
```

Step 2: Generate Dictionary Content

Choose a generation method based on what's available:

From LLM: Prompt ChatGPT or Claude with:

```
A dictionary can be used to guide the fuzzer. Write me a dictionary file for fuzzing a <PNG parser>. Each line should be a quoted string or key-value pair like kw="value". Include magic bytes, chunk types, and common header values. Use hex escapes like "\xF7\xF8" for binary values.
```

From header files:

```
grep -o '".*"' header.h > header.dict
```

From man pages (for CLI tools):

```
man curl | grep -oP '^\s*(--|-)\K\S+' | sed 's/[,.]$//' | sed 's/^/"&/; s/$/&"/' | sort -u > man.dict
```

From binary strings:

```
strings ./binary | sed 's/^/"&/; s/$/&"/' > strings.dict
```

Step 3: Pass Dictionary to Fuzzer

Use the appropriate flag for your fuzzer (see Quick Reference above).

Common Patterns

Pattern: Protocol Keywords

Use Case: Fuzzing HTTP or custom protocol handlers

Dictionary content:

```
# HTTP methods
"GET"
"POST"
"PUT"
"DELETE"
"HEAD"

# Headers
"Content-Type"
"Authorization"
"Host"

# Protocol markers
"HTTP/1.1"
"HTTP/2.0"
```

Pattern: Magic Bytes and File Format Headers

Use Case: Fuzzing image parsers, media decoders, archive handlers

Dictionary content:

```
# PNG magic bytes and chunks
png_magic="\x89PNG\r\n\x1a\n"
ihdr="IHDR"
plte="PLTE"
idat="IDAT"
iend="IEND"

# JPEG markers
jpeg_soi="\xFF\xD8"
jpeg_eoi="\xFF\xD9"
```

Pattern: Configuration File Keywords

Use Case: Fuzzing config file parsers (YAML, TOML, INI)

Dictionary content:

```
# Common config keywords
"true"
"false"
"null"
"version"
"enabled"
"disabled"

# Section headers
"[general]"
"[network]"
"[security]"
```

Advanced Usage

Tips and Tricks

Tip	Why It Helps

Combine multiple generation methods	LLM-generated keywords + strings from binary covers broad surface
Include boundary values	`"0"`, `"-1"`, `"2147483647"` trigger edge cases
Add format delimiters	`:`, `=`, `{`, `}` help fuzzer construct valid structures
Keep dictionaries focused	50-200 entries perform better than thousands
Test dictionary effectiveness	Run with and without dict, compare coverage

Auto-Generated Dictionaries (AFL++)

When using `afl-clang-lto` compiler, AFL++ automatically extracts dictionary entries from string comparisons in the binary. This happens at compile time via the AUTODICTIONARY feature.

Enable auto-dictionary:

```
export AFL_LLVM_DICT2FILE=auto.dict
afl-clang-lto++ target.cc -o target
# Dictionary saved to auto.dict
afl-fuzz -x auto.dict -i in -o out -- ./target
```

Combining Multiple Dictionaries

Some fuzzers support multiple dictionary files:

```
# AFL++ with multiple dictionaries
afl-fuzz -x keywords.dict -x formats.dict -i in -o out -- ./target
```

Anti-Patterns

Anti-Pattern	Problem	Correct Approach
Including full sentences	Fuzzer needs atomic tokens, not prose	Break into individual keywords
Duplicating entries	Wastes mutation budget	Use `sort -u` to deduplicate
Over-sized dictionaries	Slows fuzzer, dilutes useful tokens	Keep focused: 50-200 most relevant entries
Missing hex escapes	Non-printable bytes become mangled	Use `\xXX` for binary values
No comments	Hard to maintain and audit	Document sections with `#` comments

Tool-Specific Guidance

libFuzzer

```
clang++ -fsanitize=fuzzer,address harness.cc -o fuzz
./fuzz -dict=./dictionary.dict corpus/
```

Integration tips:

- Dictionary tokens are inserted/replaced during mutations
- Combine with `-max_len` to control input size
- Use `-print_final_stats=1` to see dictionary effectiveness metrics
- Dictionary entries longer than `-max_len` are ignored

AFL++

```
afl-fuzz -x ./dictionary.dict -i input/ -o output/ -- ./target @@
```

Integration tips:

- AFL++ supports multiple `-x` flags for multiple dictionaries
- Use `AFL_LLVM_DICT2FILE` with `afl-clang-lto` for auto-generated dictionaries
- Dictionary effectiveness shown in fuzzer stats UI
- Tokens are used during deterministic and havoc stages

cargo-fuzz (Rust)

```
cargo fuzz run fuzz_target -- -dict=./dictionary.dict
```

Integration tips:

- cargo-fuzz uses libFuzzer backend, so all libFuzzer dict flags work
- Place dictionary file in `fuzz/` directory alongside harness
- Reference from harness directory: `cargo fuzz run target -- -dict=../dictionary.dict`

go-fuzz (Go)

go-fuzz does not have built-in dictionary support, but you can manually seed the corpus with dictionary entries:

```
# Convert dictionary to corpus files
grep -o '".*"' dict.txt | while read line; do
    echo -n "$line" | base64 > corpus/$(echo "$line" | md5sum | cut -d' ' -f1)
done

go-fuzz -bin=./target-fuzz.zip -workdir=.
```

Troubleshooting

Issue	Cause	Solution
Dictionary file not loaded	Wrong path or format error	Check fuzzer output for dict parsing errors; verify file format
No coverage improvement	Dictionary tokens not relevant	Analyze target code for actual keywords; try different generation method
Syntax errors in dict file	Unescaped quotes or invalid escapes	Use `\\` for backslash, `\"` for quotes; validate with test run
Fuzzer ignores long entries	Entries exceed `-max_len`	Keep entries under max input length, or increase `-max_len`
Too many entries slow fuzzer	Dictionary too large	Prune to 50-200 most relevant entries

Related Skills

Tools That Use This Technique

Skill	How It Applies
libfuzzer	Native dictionary support via `-dict=` flag
aflpp	Native dictionary support via `-x` flag; auto-generation with AUTODICTIONARIES
cargo-fuzz	Uses libFuzzer backend, inherits `-dict=` support

Related Techniques

Skill	Relationship

fuzzing-corpus	Dictionaries complement corpus: corpus provides structure, dictionary provides keywords
coverage-analysis	Use coverage data to validate dictionary effectiveness
harness-writing	Harness structure determines which dictionary tokens are useful

Resources

Key External Resources

AFL++ Dictionaries Pre-built dictionaries for common formats (HTML, XML, JSON, SQL, etc.). Good starting point for format-specific fuzzing.

libFuzzer Dictionary Documentation Official libFuzzer documentation on dictionary format and usage. Explains token insertion strategy and performance implications.

Additional Examples

OSS-Fuzz Dictionaries Real-world dictionaries from Google's continuous fuzzing service. Search project directories for `*.dict` files to see production examples.

Revision #6
Created 2026-02-18 08:40:11 UTC by John
Updated 2026-07-26 20:01:28 UTC by John