Coverage Guided Fuzzy Testing

mit LLVM libFuzzer

Adrian Imboden

adi@thingdust.com
adrian.imboden@komaxgroup.com

`https://github.com/adrianimboden/cppusergroup-coverage-guided-fuzzing`

April 8, 2019

Inhalt

Inhalt

Ziel heute

- Sehen, wie einfach man Fuzzer heutzutage einsetzen kann
- Möglichst viel selber Hand anlegen

Was ist Fuzzing?

Eine Technik, meist automatisiert oder halbautomatisiert, welche:

Was ist Fuzzing?

Eine Technik, meist automatisiert oder halbautomatisiert, welche:

- ungültige
- unerwartete
- oder zufällige

Eingabe in ein Programm gibt und dann prüft, ob die Software:

Was ist Fuzzing?

Eine Technik, meist automatisiert oder halbautomatisiert, welche:

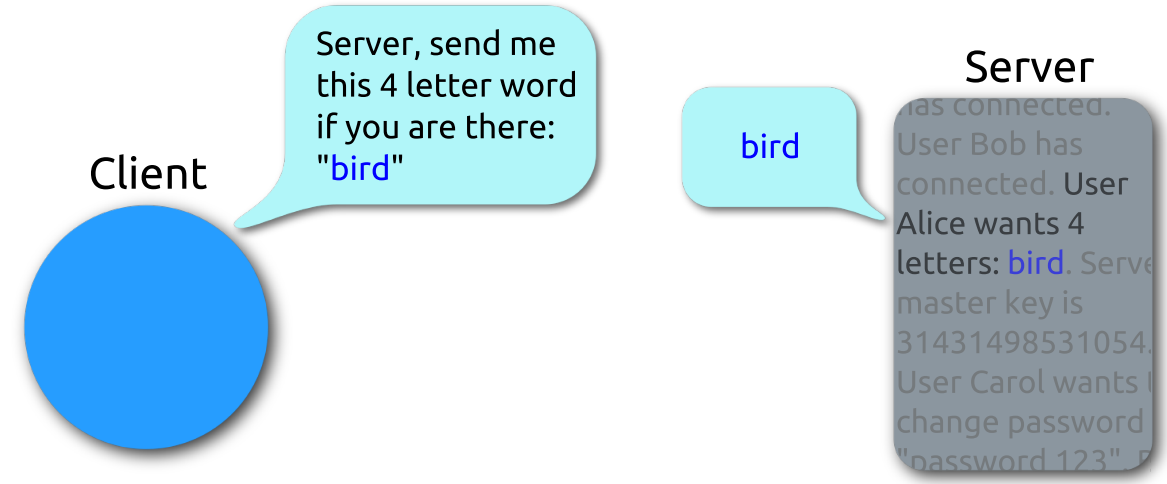
- ungültige
- unerwartete
- oder zufällige

Eingabe in ein Programm gibt und dann prüft, ob die Software:

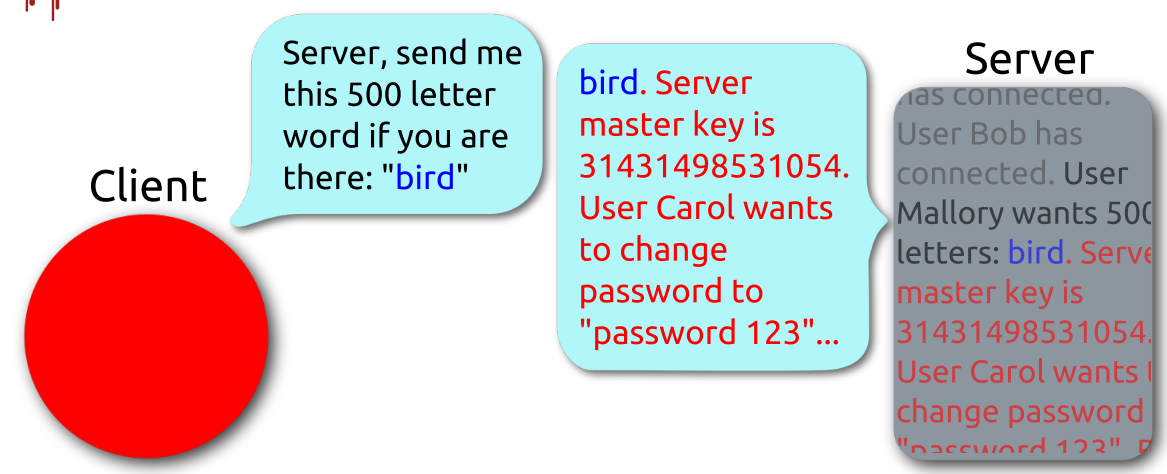
- Crasht
- Assertions auslöst
- Races beinhaltet
- Leaks beinhaltet
- ...

Demo Heartbleed

Heartbeat – Normal usage



Heartbeat – Malicious usage



Inhalt

Mein erster eigener Fuzzer

my_fuzzer.cpp:

```
1  #include <string>
2
3  extern "C" int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
4    if (std::string{data, data + size} == "mypassword") {
5      abort();
6    }
7    return 0; // Non-zero return values are reserved for future use.
8  }
```

Mein erster eigener Fuzzer

my_fuzzer.cpp:

```cpp
#include <string>

extern "C" int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
  if (std::string{data, data + size} == "mypassword") {
    abort();
  }
  return 0; // Non-zero return values are reserved for future use.
}
```

Build:

```
clang++-8 -g -fsanitize=fuzzer,address my_fuzzer.cpp
```

Mein erster eigener Fuzzer

my_fuzzer.cpp:

```cpp
#include <string>

extern "C" int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
  if (std::string{data, data + size} == "mypassword") {
    abort();
  }
  return 0; // Non-zero return values are reserved for future use.
}
```

Build:

```
clang++-8 -g -fsanitize=fuzzer,address my_fuzzer.cpp
```

Run:

```
./a.out
```

Mein erster eigener Fuzzer

my_fuzzer.cpp:

```cpp
#include <string>

extern "C" int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
  if (std::string{data, data + size} == "mypassword") {
    abort();
  }
  return 0; // Non-zero return values are reserved for future use.
}
```

Build:

```
clang++-8 -g -fsanitize=fuzzer,address my_fuzzer.cpp
```

Run:

```
./a.out
```

→ Hands On

Beispielsanwendung

Suche nach Base64 Implementierung auf Google:

Beispielsanwendung

Suche nach Base64 Implementierung auf Google:

> *I have made some modifications to this old post. This one works way faster than before. Its other advantage is smooth handling of corrupt data as well.*

Beispielsanwendung

Suche nach Base64 Implementierung auf Google:

> *I have made some modifications to this old post. This one works way faster than before. Its other advantage is smooth handling of corrupt data as well.*

Perfekt für unsere Zwecke: Schnell und gutes Error Handling.
Dann lassen wir den Code doch mal durch den Fuzzer laufen:

`https://raw.githubusercontent.com/adrianimboden/cppusergroup-coverage-guided-fuzzing/master/base64_example.cpp`

→ Hands On

Inhalt

Schlusswort

Für das beste Fuzzing Ergebnis:

- Möglichst kleine Programmteile auf einmal
- Möglichst schnelles Programm
- In Kombination mit Sanitizers (ASan, MSan, UBSan, evtl. TSan)

Weitere Themen für Interessierte

- Corpus
- Eigene/Andere Mutatoren

→ `https://llvm.org/docs/LibFuzzer.html`

The End

Noch fragen?