CERIAS

The Center for Education and Research in Information Assurance and Security

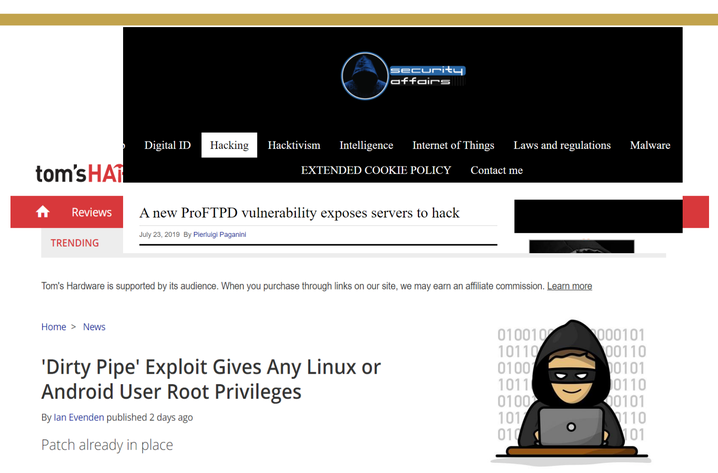
GRILLER – A FRAMEWORK FOR UNDER CONSTRAINED FUZZING

Siddharth Muralee[1], Jayashree Srinivasan[1], Akul Pillai[1], Antonio Bianchi[1], Aravind Machiry[1]
Giovanni Vigna[2], Christopher Kruegel[2]
Purdue University[1], University of California, Santa Barbara[2]

WHY ? <<

Fuzzing has been proven effective in automatically finding bugs, however:

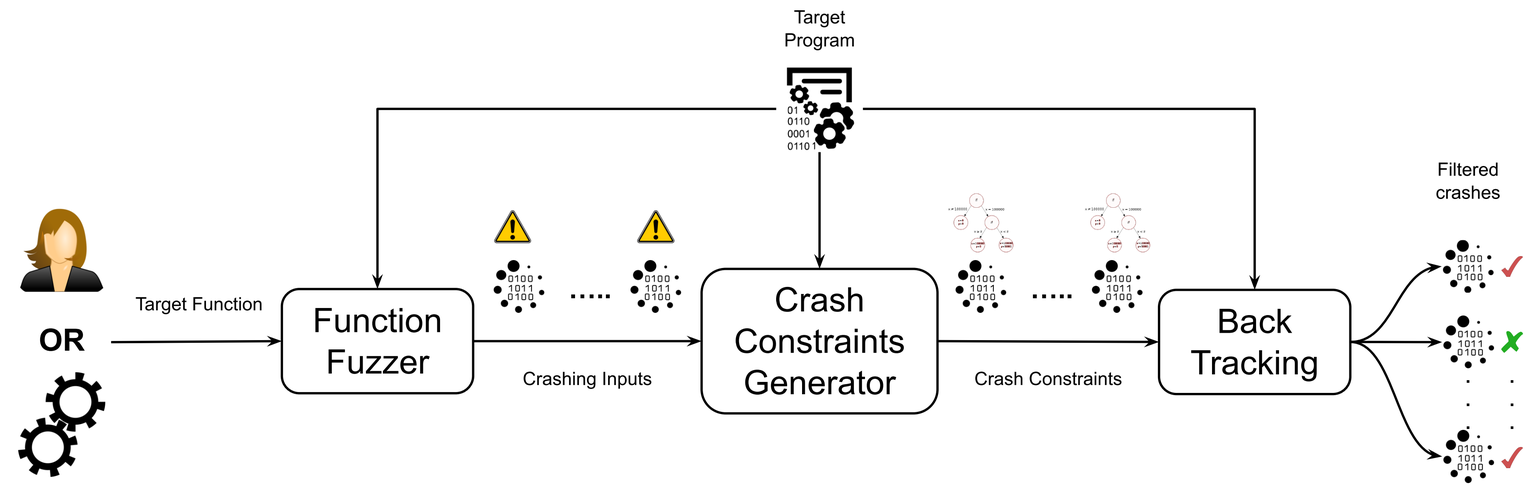
- Critical real-world software is complex and take input from several interfaces such as sockets, pipes
- Not all functions inside an application are interesting to fuzz



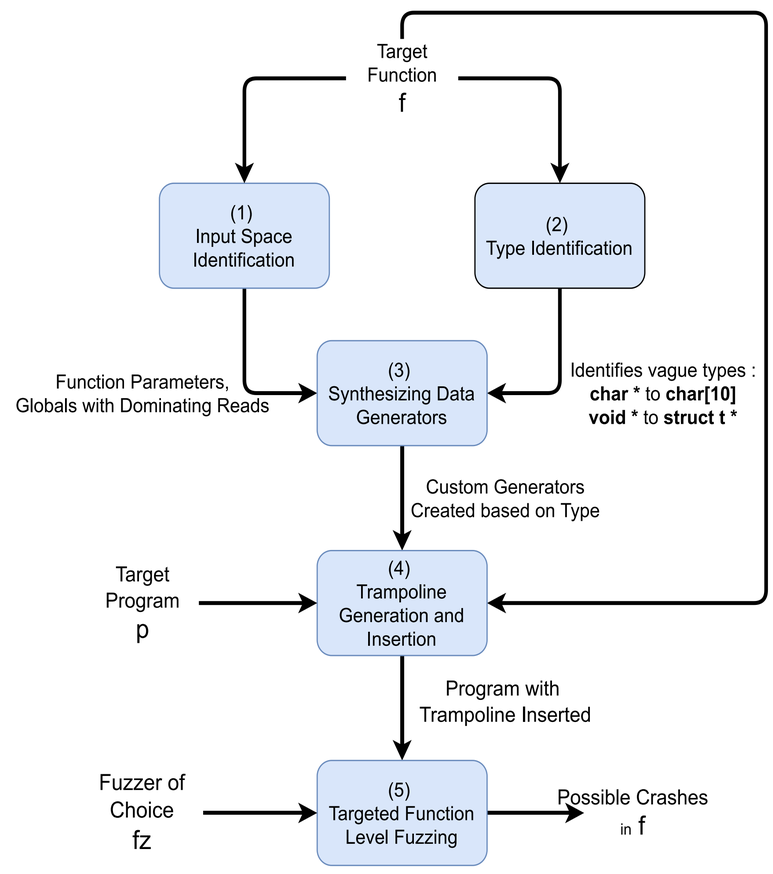
>> WHAT ?

- Automatically generate type-aware trampolines to fuzz “deep” functions
- Crash Triage and extract root cause constraints via Symbolic Execution
- Filter out false positives by back-tracing along the call graph
- Integrable with any existing fuzzer

HOW IT WORKS <<



>> FUZZING FUNCTIONS



GRILLER APPROACH <<

Leaf Function

```
struct pool {
...
bool avail;
char *buf;
unsigned size;
...
};

int add_elem(void *data,
             const char *buf,
             unsigned size) {
 struct pool *p = (struct pool *)data;
 if ( C1 p->avail) {
   p->buf = malloc(p->size*sizeof(char))
   if (p->buf) {
    p->avail = 0;
    memcpy( C2 p->buf,
     C3 C4 buf, size);
   }
   return 0;
 }
return -1;
}
```

Crashes

- C1 - Null pointer Dereference of *p*
- C2 - Buffer Overflow in *p->buf*
- C3 - Null pointer dereference of the argument *buf*
- C4 - Buffer Overread of the argument *buf*

Parent Function

```
struct pool GP[POOL_SIZE];
enum rtype { ..,NEW, UPDATE,..};

int process_req(enum rtype r,
                uint32_t idx,
                char *buf, uint32_t sz) {
if (idx < POOL_SIZE) {
  GP[idx].size = size;
  switch(r) {
    case UPDATE:
      C1 C2 C3 C4
      add_elem(&GP[idx], buf, sz);   CS1
      break;
    ...
    case NEW:
      C1 C2 C3 C4
      add_elem(&GP[idx], "HDR", 3);   CS2
      break;
  }
}
...
}
```

Call Sites in Parent Function

- C1 - Valid Arguments provided at all callsites
- C2 - Possible only from CS2
- C3 - *buf* can be null only from CS1
- C4 - *sz* can be greater than size of *buf* only from CS1

Top Level Function

```
int handle_req(enum rtype r,
               uint32_t gidx,
               int sockfd) {
 char rbuf[MAX_BUF_SIZE];
 int sz = MAX_BUF_SIZE;

 if (r == NEW) {
  sz = read(sockfd, &rbuf, sizeof(rbuf));
  if (sz <= 0)
   return -1;
    C2 C3 C4
  process_req(r, gidx, &rbuf, sz);
 }

}
```

Backtracking to top level function

- C2 - *sz* can be less than 3, since it’s coming from read
- C3 - *buf* is a local variable, hence can’t be null
- C4 - *sz* is same as the size of the buffer

COMPARISON WITH OTHER FUZZERS <<

Tool/Technique	Supports Arbitrary Functions?	Handle Complex Types?	Automated Type-based Arguments Generation?	Automatic Crash Triaging?	Extensible
LIBFUZZER	Yes	Yes	No	No	No
OSS-Fuzz	Yes	Yes	No	No	No
FTG	Yes	No	-	No	No
FUZZGEN	No	No	-	No	Yes
INTELLIGEN	Yes	No	No	No	Yes
FUDGE	No	No	-	No	Yes
GRILLER	Yes	Yes	Yes	Yes	Yes

>> CURRENT WORK

Evaluating the efficiency of function-level fuzzing across a wide variety of systems software including, Operating systems, file parsers, network applications, and shared libraries.

Currently testing on applications in the MAGMA dataset such as *libpng*, *libxml* and, *libsndfile*.

Improving the scalability by using a microservice architecture allowing fuzzing via multiple interfaces.