Fuzzing Samba: a tutorial

Douglas Bagnall *<douglas.bagnall@catalyst.net.nz>*

Prerequisites

If you want to follow along you need

- a Samba source tree
- honggfuzz
- samba-fuzz-seeds

https://lists.samba.org/archive/samba-technical/2021-May/136567.html

Examples use a new Ubuntu 20.04 cloud instance

Clone the repos

```
# Samba
git clone git://git.samba.org/samba.git

# Honggfuzz
git clone https://github.com/google/honggfuzz.git

# Samba Fuzz Seeds
git clone https://gitlab.com/samba-team/samba-fuzz-seeds.git
```

Honggfuzz build dependencies

```
sudo apt install libbfd-dev libunwind-dev clang-11
```

Any clang version since 5 is OK

Honggfuzz build

```
cd honggfuzz
make
```

you can `make install`, but I don't

Samba build dependencies

apt install this and that for Samba:

```
cd samba
sudo bootstrap/generated-dists/ubuntu2004/bootstrap.sh
```

(look in `bootstrap/generated-dists/` for other distros).

Samba build

not what you normally do.

(so `rm -r bin` in an pre-used Samba directory)

Samba build

```
./buildtools/bin/waf -C configure \
    --enable-libfuzzer \
    --address-sanitizer \
    CC=~/honggfuzz/hfuzz_cc/hfuzz-clang \
    LINK_CC=~/honggfuzz/hfuzz_cc/hfuzz-clang \
    --disable-warnings-as-errors \
    --abi-check-disable  \
    --without-gettext

make -j
```

Samba build details

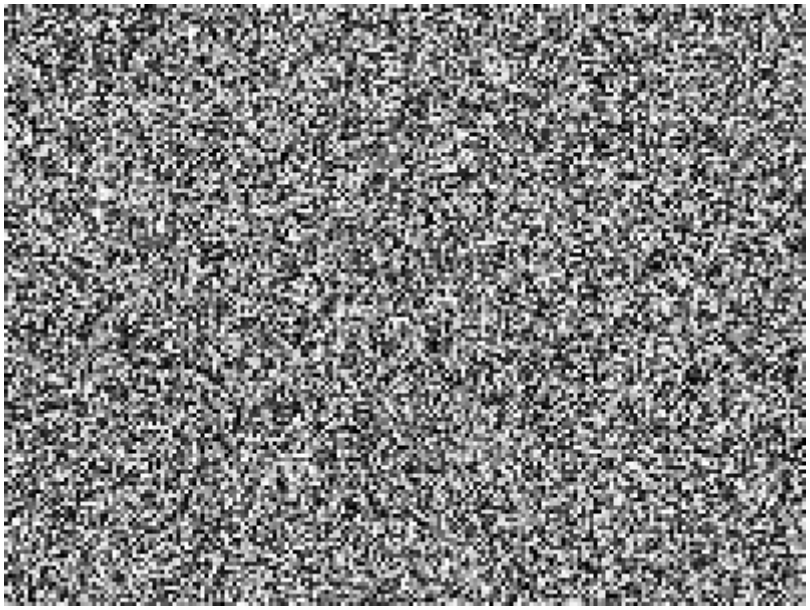
- `--enable-libfuzzer` turn on fuzzing (using *libfuzzer API*)
- `--address-sanitizer` reliably find memory errors
- `--disable-warnings-as-errors` clang has different warnings
- `CC=~/honggfuzz/hfuzz_cc/hfuzz-clang` instrumenting compiler
- `LINK_CC=~/honggfuzz/hfuzz_cc/hfuzz-clang` compiler as linker
- `--without-gettext`, `--abi-check-disable` avoid problems we don’t need

Samba fuzz seeds

forget about this for now.

What is fuzzing?

- testing a program with random input



We can’t just do this

```
bin/smbd < /os/urandom
```

- multiple interfaces
- *some* packet structure
- infinite uninteresting inputs

blackbox fuzzing *is* a thing that we won’t talk about

Coverage based fuzzing

instrument the binary
to see which branches are taken

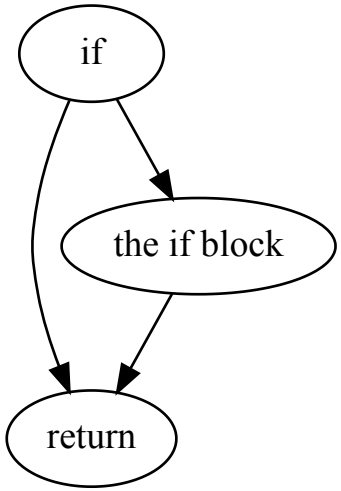
when an input takes new branches, make similar inputs

Darwinian evolution,
with the binary providing the fitness landscape

Control flow graph

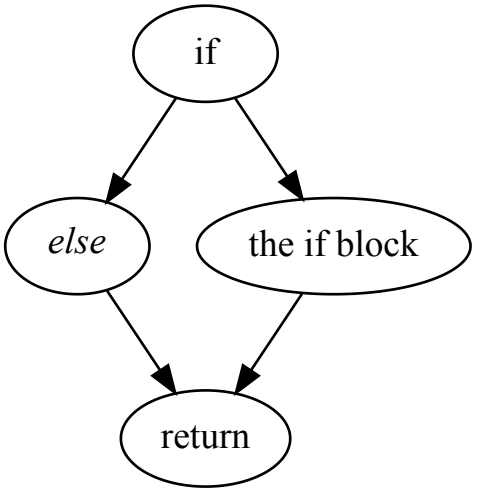
```
flags = CVAL(inbuf, 1);
if (flags & 1) {
        dgram->header.flags.more = true;
        count++;
}
return count;
```

We want to follow all the edges



Control flow graph

```
flags = CVAL(inbuf, 1);
if (flags & 1) {
        dgram->header.flags.more = true;
        count++;
}
return count;
```



(implicit else apparently makes it more tractable)

fuzz targets

Don’t attack the monolith,
attack interesting chunks that

- are of manageable size
- bring data across trust boundaries
- work with byte arrays (datablobs)

libfuzzer fuzz target API

We write binaries with these entry points

```
int LLVMFuzzerTestOneInput(uint8_t *input, size_t len)
{
        return 0;
}


int LLVMFuzzerInitialize(int *argc, char ***argv)
{
        return 0;
}
```

`LLVMFuzzerTestOneInput()` returns zero or crashes

A contrived example

```
int LLVMFuzzerTestOneInput(uint8_t *input, size_t len)
{
        if (len != 5) {
                return 0;
        }
        if (strncmp((char*)input, "SAMBA", 5) == 0) {
                abort();
        }
        return 0;
}
```

```
int LLVMFuzzerTestOneInput(uint8_t *input, size_t len)
{
        if (len != 5) {
                return 0;
        }
        if (strncmp((char*)input, "SAMBA", 5) == 0) {
                abort();
        }
        return 0;
}
```

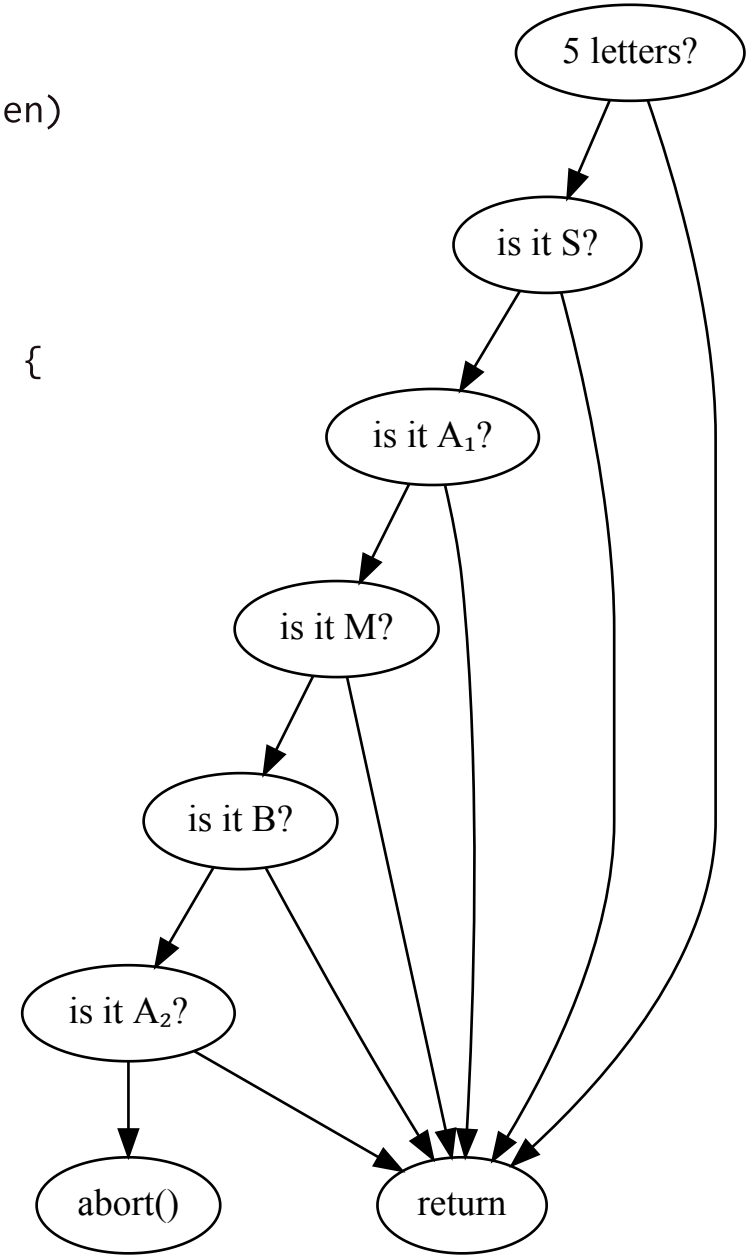
5 chars → 40 bits → ~ 1,000,000,000,000 combinations for a blackbox fuzzer *(if it knows to use 5 bytes)*

and a process start-up for each one

*256 * 5 + ε* for a coverage based fuzzer

```
int LLVMFuzzerTestOneInput(uint8_t *input, size_t len)
{
        if (len != 5) {
                return 0;
        }
        if (strncmp((char*)input, "SAMBA", 5) == 0) {
                abort();
        }
        return 0;
}
```

with libfuzzer, no process startup



finding fuzzable functions

You can do worse than

```
git grep -P 'parse' :/*.h
git grep -P 'pull' :/*.h
git grep -P 'get' :/*.h
```

but knowing the code really helps.

A nice target function

- takes a datablob (`uint8_t *data`, `size_t len`)
- maybe has a target struct (`**struct packet p`) or tallocs one
- doesn't ask for *too* much else
- does an interesting amount of work
- works closely with the outside world
- is actually used

Not quite suitable example

```
int ctdb_req_message_pull(uint8_t *buf, size_t buflen,
                          struct ctdb_req_header *h,
                          TALLOC_CTX *mem_ctx,
                          struct ctdb_req_message *c);
```

only called from tests
but reaches into real world code?

Probably too small example

```
char *ldb_base64_encode(TALLOC_CTX *mem_ctx,
                        const char *buf,
                        int len);
```

doesn’t do very much.

Our target

```
bool parse_lpq_entry(enum printing_types printing_type,
                     char *line,
                     print_queue_struct *buf,
                     print_status_struct *status,
                     bool first);
```

only called from `generic_queue_get()`
which gets output of another system process

- we won't find a security bug
- we might find/fix a printing bug

parse_lpq_entry()

```
bool parse_lpq_entry(enum printing_types printing_type,char *line,
             print_queue_struct *buf,
             print_status_struct *status,bool first)
{
    bool ret;

    switch (printing_type) {
        case PRINT_SYSV:
            ret = parse_lpq_sysv(line,buf,first);
            break;
        case PRINT_AIX:
            ret = parse_lpq_aix(line,buf,first);
            break;
        case PRINT_HPUX:
            ret = parse_lpq_hpux(line,buf,first);
            break;

                /* 13 or 14 of these ... */

        case PRINT_LPROS2:
            ret = parse_lpq_os2(line,buf,first);
            break;
#ifdef DEVELOPER
        case PRINT_VLP:
        case PRINT_TEST:
```

```
                /* 13 or 14 of these ... */

        case PRINT_LPROS2:
            ret = parse_lpq_os2(line,buf,first);
            break;
#ifdef DEVELOPER       /* in this case, we aren't DEVELOPER */
        case PRINT_VLP:
        case PRINT_TEST:
            ret = parse_lpq_vlp(line,buf,first);
            break;
#endif /* DEVELOPER */
        default:
            ret = parse_lpq_bsd(line,buf,first);
            break;
    }

    /* We don't want the newline in the status message. */
    {
        char *p = strchr_m(line,'\n');
        if (p) {
            *p = 0;
        }
    }

    /* in the LPRNG case, we skip lines starting by a space.*/
    if (!ret && (printing_type==PRINT_LPRNG) ) {
        if (line[0]==' ') {
```

```
            return ret;
        }
    }
```

```
/* in the LPRNG case, we skip lines starting by a space.*/
if (!ret && (printing_type==PRINT_LPRNG) ) {
    if (line[0]==' ') {
        return ret;
    }
}

if (status && !ret) {
    /* a few simple checks to see if the line might be a
        printer status line:
        handle them so that most severe condition is shown */
    int i;
    if (!strlower_m(line)) {
        return false;
    }

    switch (status->status) {
        case LPSTAT_OK:
            for (i=0; stat0_strings[i]; i++) {
                if (strstr_m(line,stat0_strings[i])) {
                    fstrcpy(status->message,line);
                    status->status=LPSTAT_OK;
                    return ret;
                }
            }
            FALL_THROUGH;
        case LPSTAT_STOPPED:
```

```
for (i=0; stat1_strings[i]; i++) {
    if (strstr_m(line,stat1_strings[i])) {
        fstrcpy(status->message,line);
        status->status=LPSTAT_STOPPED;
        return ret;
```

```
            case LPSTAT_STOPPED:
                for (i=0; stat1_strings[i]; i++) {
                    if (strstr_m(line,stat1_strings[i])) {
                        fstrcpy(status->message,line);
                        status->status=LPSTAT_STOPPED;
                        return ret;
                    }
                }
                FALL_THROUGH;
            case LPSTAT_ERROR:
                for (i=0; stat2_strings[i]; i++) {
                    if (strstr_m(line,stat2_strings[i])) {
                        fstrcpy(status->message,line);
                        status->status=LPSTAT_ERROR;
                        return ret;
                    }
                }
                break;
        }
    }

    return ret;
}
```

start with the template

```
int LLVMFuzzerInitialize(int *argc, char ***argv)
{
        return 0;
}

int LLVMFuzzerTestOneInput(uint8_t *input, size_t len)
{
        return 0;
}
```

Add the target function

```
int LLVMFuzzerTestOneInput(uint8_t *input, size_t len)
{
        enum printing_types printing_type;
        char *line;
        print_queue_struct pq_buf = {0};
        print_status_struct status = {0};
        bool first;

        /* TODO convert input here */

        parse_lpq_entry(printing_type,
                        line,
                        &pq_buf, /* out */
                        &status, /* out */
                        first);
        return 0;
 }
```

We can’t alter the input

```
enum printing_types printing_type;
print_queue_struct pq_buf = {0};
print_status_struct status = {0};
bool first;

/* we need to NUL-terminate but can't always do this */
if (len < 1) {
        return 0;
}
input[len - 1] = '\0';

parse_lpq_entry(printing_type,
                (char *)input,
                &pq_buf, /* out */
                &status, /* out */
                first);
```

Copy to a char buffer

```
#define MAX_LENGTH (1024 * 1024)
char line[MAX_LENGTH + 1];

 int LLVMFuzzerTestOneInput(uint8_t *input, size_t len)
 {
        enum printing_types printing_type;
        print_queue_struct pq_buf = {0};
        print_status_struct status = {0};
        bool first;

        if (len > MAX_LENGTH) {
                return 0;
        }

        memcpy(line, input, len);
        line[len] = '\0';

        parse_lpq_entry(printing_type,
                        line,
                        &pq_buf, /* out */
                        &status, /* out */
                        first);
```

get printing_type

```
#define MAX_LENGTH (1024 * 1024)
char line[MAX_LENGTH + 1];

int LLVMFuzzerTestOneInput(uint8_t *input, size_t len)
{
        enum printing_types printing_type;
        print_queue_struct pq_buf = {0};
        print_status_struct status = {0};
        bool first;
        unsigned x;
        if (len < 1 || len > MAX_LENGTH) {
                return 0;
        }

        x = input[0];
        /* There are ~14 types, default goes to bsd */
        printing_type = x & 15;
        input++;
        len--;

        memcpy(line, input, len);
        line[len] = '\0';

        parse_lpq_entry(printing_type,
                        line,
```

```
&pq_buf, /* out */
&status, /* out */
first);
```

get `first` flag

```
#define MAX_LENGTH (1024 * 1024)
char line[MAX_LENGTH + 1];

int LLVMFuzzerTestOneInput(uint8_t *input, size_t len)
{
        enum printing_types printing_type;
        print_queue_struct pq_buf = {0};
        print_status_struct status = {0};
        bool first;
        unsigned x;
        if (len < 1 || len > MAX_LENGTH) {
                return 0;
        }

        x = input[0];
        /* There are ~14 types, default goes to bsd */
        printing_type = x & 15;
        input++;
        len--;

        first = x & 16 ? true : false;

        memcpy(line, input, len);
        line[len] = '\0';
```

```
parse_lpq_entry(printing_type,
                line,
                &pq_buf, /* out */
                &status, /* out */
                first);
```

```
int LLVMFuzzerInitialize(int *argc, char ***argv)
{
        return 0;
}

#define MAX_LENGTH (1024 * 1024)
char line[MAX_LENGTH + 1];

int LLVMFuzzerTestOneInput(uint8_t *input, size_t len)
{
        enum printing_types printing_type;
        print_queue_struct pq_buf = {0};
        print_status_struct status = {0};
        bool first;
        unsigned x;
        if (len < 1 || len > MAX_LENGTH) {
                return 0;
        }

        x = input[0];
        /* There are ~14 types, default goes to bsd */
        printing_type = x & 15;
        input++;
        len--;
        first = x & 16 ? true : false;
        memcpy(line, input, len);
        line[len] = '\0';

        parse_lpq_entry(printing_type,
                        line,
                        &pq_buf, /* out */
                        &status, /* out */
                        first);
```

Add headers by trial and error

```
#include "../../source3/include/includes.h"
#include "printing.h"
#include "lib/util/string_wrappers.h"
#include "fuzzing/fuzzing.h"   /* always this */

int LLVMFuzzerInitialize(int *argc, char ***argv)
{
        return 0;
}

#define MAX_LENGTH (1024 * 1024)
char line[MAX_LENGTH + 1];
```

Add wscript_build stanza

- lib/fuzzing/wscript_build

```
bld.SAMBA_BINARY('fuzz_tiniparser',
                 source='fuzz_tiniparser.c',
                 deps='fuzzing tiniparser talloc afl-fuzz-main',
                 fuzzer=True)


bld.SAMBA_BINARY('fuzz_parse_lpq_entry',
                 source='fuzz_parse_lpq_entry.c',
                 deps='fuzzing afl-fuzz-main smbd_base',
                 fuzzer=True)

bld.SAMBA_BINARY('fuzz_oLschema2ldif',
                 source='fuzz_oLschema2ldif.c',
                 deps='fuzzing oLschema2ldif-lib afl-fuzz-main',
                 fuzzer=True)
```

wscript_build fuzzing boilerplate

- lib/fuzzing/wscript_build

```
bld.SAMBA_BINARY('fuzz_parse_lpq_entry',
                 source='fuzz_parse_lpq_entry.c',
                 deps='fuzzing afl-fuzz-main smbd_base',
                 fuzzer=True)
```

- deps='fuzzing afl-fuzz-main ...'
- fuzzer=True — build in fuzzing mode

minimal deps are best — more so than usual

build

```
$ make
PYTHONHASHSEED=1 WAF_MAKE=1  ./buildtools/bin/waf build
Waf: Entering directory `/home/douglasb/src/samba-fuzz/bin/default'
        Selected embedded Heimdal build
[2661/3865] Compiling lib/fuzzing/fuzz_parse_lpq_entry.c
[3822/3865] Linking bin/default/lib/fuzzing/fuzz_parse_lpq_entry
Waf: Leaving directory `/home/douglasb/src/samba-fuzz/bin/default'
'build' finished successfully (4.332s)
```

run it and see what happens

```
$ bin/fuzz_parse_lpq_entry
=================================================================
==1222626==ERROR: AddressSanitizer: odr-violation (0x7fa17eef5ca0):
  [1] size=88 'ndr_table_secrets' source3/librpc/gen_ndr/ndr_secrets.c:1004:34
  [2] size=88 'ndr_table_secrets' source3/librpc/gen_ndr/ndr_secrets.c:1004:34
These globals were registered at these points:
  [1]:
    #0 0x560acce3184d in __asan_register_globals (/home/douglasb/src/samba-fuzz/bin/defau
    #1 0x7fa17eec425b in asan.module_ctor (/home/douglasb/src/samba-fuzz/bin/shared/priva

  [2]:
    #0 0x560acce3184d in __asan_register_globals (/home/douglasb/src/samba-fuzz/bin/defau
    #1 0x7fa16ca5679b in asan.module_ctor (/home/douglasb/src/samba-fuzz/bin/shared/priva

==1222626==HINT: if you don't care about these errors you may set ASAN_OPTIONS=detect_odr
SUMMARY: AddressSanitizer: odr-violation: global 'ndr_table_secrets' at source3/librpc/ge
==1222626==ABORTING
```

This isn't normal, relates to shared libraries

https://bugzilla.samba.org/show_bug.cgi?id=14200

(smbd_base contains too much?)

Ignore that for now

```
$ ASAN_OPTIONS=detect_odr_violation=0 bin/fuzz_parse_lpq_entry
Accepting input from '[STDIN]'
Usage for fuzzing: honggfuzz -P [flags] -- bin/fuzz_parse_lpq_entry
^C


$ ASAN_OPTIONS=detect_odr_violation=0 time bin/fuzz_parse_lpq_entry /etc/hosts
Accepting input from '/etc/hosts'
Usage for fuzzing: honggfuzz -P [flags] -- bin/fuzz_parse_lpq_entry
```

(It takes over 2 seconds to process /etc/hosts)

- `bin/fuzz_parse_lpq_entry` is a libfuzzer *fuzz target*
- and a standalone executable

Persistent fuzzing with Honggfuzz

honggfuzz has too many options and makes too many files

- Wrap it in scripts
- Put files in `./fuzz/`

```
$ cd ~/samba
$ mkdir fuzz
$ cp ~/samba-fuzz-seeds/honggfuzz/{run, rotate} fuzz/
```

Link honggfuzz binary into samba/fuzz

```
$ cd fuzz
$ ln -s ~/honggfuzz/honggfuzz .
$ cd -
```

optionally link or copy in public seeds

```
$ cd fuzz
$ ln -s ~/samba-fuzz-seeds/seeds seeds
```

(be careful what you publish)

fuzz/run

```
#!/bin/bash

# Copy this into $SAMBA_ROOT/fuzz

set -e

N=$(basename $1)

if [ -f $N ]; then
    echo "USAGE: $0 <fuzz-target> [args]"
    exit
fi

MAXSIZE=10000
HERE=$(dirname $0)
ROOT=$HERE/..
```

fuzz/run bin/fuzz_parse_lpq_entry

```
$HERE/honggfuzz   \
    -V            \  # verify crashes (is it flappy?)
    -W $DEST      \  # crashing inputs in fuzz/results/fuzz_parse_lp
    -T            \  # timeout is failure (SIGVTALRM)
    -F $MAXSIZE   \  # maximum input size is 10000 bytes
    -S            \  # use sanitizers
    -t 20         \  # timeout (seconds)
    -U            \  # save smaller crash files, if found
    -R $REPORT    \  # report to fuzz/results/HONGGFUZZ-fuzz_parse_l
    -P            \  # persistent mode
    -i $SEED_DIR/ \  # find/save seeds in fuzz/seeds/fuzz_parse_lpq_
    $@            \  # any further arguments to fuzz/run go to hongg
    -- $ROOT/bin/$N  # bin/fuzz_parse_lpq_entry
```

```
-------------------------[  0 days 00 hrs 00 mins 03 secs ]----------------------
  Iterations : 15,963 [15.96k]
  Mode [3/3] : Feedback Driven Mode
      Target : fuzz/../bin/fuzz_parse_lpq_entry
     Threads : 2, CPUs: 4, CPU%: 194% [48%/CPU]
       Speed : 7,810/sec [avg: 5,321]
     Crashes : 0 [unique: 0, blocklist: 0, verified: 0]
    Timeouts : 0 [20 sec]
 Corpus Size : 339, max: 10,000 bytes, init: 0 files
  Cov Update : 0 days 00 hrs 00 mins 00 secs ago
    Coverage : edge: 465/337,545 [0%] pc: 3 cmp: 11,643
--------------------------------- [ LOGS ] ------------------/ honggfuzz 2.4 /-
Sz:90 Tm:254us (i/b/h/e/p/c) New:0/0/0/0/0/9, Cur:0/0/0/104/0/137
Sz:55 Tm:269us (i/b/h/e/p/c) New:0/0/0/0/0/5, Cur:0/0/0/112/0/159
Sz:59 Tm:415us (i/b/h/e/p/c) New:0/0/0/0/0/10, Cur:0/0/0/110/0/244
Sz:3 Tm:1,025us (i/b/h/e/p/c) New:0/0/0/0/0/9, Cur:0/0/0/108/0/104
Sz:267 Tm:328us (i/b/h/e/p/c) New:0/0/0/0/0/1, Cur:0/0/0/66/0/25
Sz:64 Tm:1,231us (i/b/h/e/p/c) New:0/0/0/0/0/5, Cur:0/0/0/163/0/1034
Sz:94 Tm:625us (i/b/h/e/p/c) New:0/0/0/0/0/3, Cur:0/0/0/112/0/196
Sz:107 Tm:260us (i/b/h/e/p/c) New:0/0/0/0/0/3, Cur:0/0/0/71/0/23
Sz:65 Tm:136us (i/b/h/e/p/c) New:0/0/0/0/0/10, Cur:0/0/0/107/0/86
Sz:877 Tm:162us (i/b/h/e/p/c) New:0/0/0/0/0/7, Cur:0/0/0/21/0/72
Sz:47 Tm:911us (i/b/h/e/p/c) New:0/0/0/0/0/1, Cur:0/0/0/129/0/391
```

```
------------------------[  0 days 00 hrs 01 mins 01 secs ]----------------------
  Iterations : 363,200 [363.20k]
  Mode [3/3] : Feedback Driven Mode
      Target : fuzz/../bin/fuzz_parse_lpq_entry
     Threads : 2, CPUs: 4, CPU%: 200% [50%/CPU]
       Speed : 3,889/sec [avg: 5,954]
     Crashes : 1 [unique: 1, blocklist: 0, verified: 0]
    Timeouts : 0 [20 sec]
 Corpus Size : 684, max: 10,000 bytes, init: 0 files
  Cov Update : 0 days 00 hrs 00 mins 00 secs ago
    Coverage : edge: 546/337,545 [0%] pc: 3 cmp: 14,294
--------------------------------- [ LOGS ] ------------------/ honggfuzz 2.4 /-
Sz:226 Tm:265us (i/b/h/e/p/c) New:0/0/0/0/0/3, Cur:0/0/0/116/0/338
Crash: saved as 'fuzz/results/fuzz_parse_lpq_entry/SIGABRT.PC.5555555dc1dc.STACK
.c81c3371.CODE.-6.ADDR.0.INSTR.lea____-0x848(%rbp),%rdx.fuzz'
Sz:274 Tm:374us (i/b/h/e/p/c) New:0/0/0/1/0/2, Cur:0/0/0/79/0/139
Sz:669 Tm:81us (i/b/h/e/p/c) New:0/0/0/0/0/1, Cur:0/0/0/40/0/38
[2021-05-05T12:25:59+0000][W][51700] arch_checkWait():238 Persistent mode: pid=5
1702 exited with status: SIGNALED, signal: 6 (Aborted)
Launching verifier for HASH: c81c3371 (iteration: 1 out of 5)
Persistent mode: Launched new persistent pid=51708
Sz:208 Tm:116us (i/b/h/e/p/c) New:0/0/0/0/0/1, Cur:0/0/0/79/0/156
Sz:92 Tm:463us (i/b/h/e/p/c) New:0/0/0/0/0/3, Cur:0/0/0/87/0/128
```

fuzz/results/HONGGFUZZ-fuzz_parse_lpq_entry.txt

```
===================================================================
TIME: 2021-05-05.12:26:02
===================================================================
FUZZER ARGS:
 mutationsPerRun : 5
 externalCmd     : NULL
 fuzzStdin       : FALSE
 timeout         : 20 (sec)
 ignoreAddr      : (nil)
 ASLimit         : 0 (MiB)
 RSSLimit        : 0 (MiB)
 DATALimit       : 0 (MiB)
 wordlistFile    : NULL
 dynFileMethod   :
 fuzzTarget      : fuzz/../bin/fuzz_parse_lpq_entry
CRASH:
```

Does it show up outside of ASAN/Honggfuzz?

- Sometimes the instrumented libc is wrong/different
- trusted old friends gdb and valgrind work better without sanitizers
- solution: make an “AFL” build[1]

1. no actual use of AFL is implied

```
$ cd
$ git clone samba samba-afl
$ cd samba-afl
$ ./buildtools/bin/waf -C configure \
                --enable-afl-fuzzer  \
                 --disable-warnings-as-errors \
                 --abi-check-disable   \
                 --without-gettext --enable-debug  \
      && make -j
```

- `--enable-afl-fuzzer`, not `--enable-libfuzzer`
- no sanitisers
- no special compilers
- `--enable-debug` is no longer implicit

--enable-afl-fuzzer

Without `--enable-afl-fuzzer`, wscript_build's `afl-fuzz-main` is a no-op.

```
bld.SAMBA_BINARY('fuzz_parse_lpq_entry',
                 source='fuzz_parse_lpq_entry.c',
                 deps='fuzzing afl-fuzz-main smbd_base',
                 fuzzer=True)
```

With it, it links in a main() function.

afl-fuzz-main.c, conceptually

```
int main(int argc, char *argv[]) {
        size_t size = 0;
        int i;

        LLVMFuzzerInitialize(&argc, &argv);

        for (i = 1; i < argc; i++) {
                uint8_t *buf = (uint8_t *)file_load(argv[i],
                                                    &size,
                                                    0,
                                                    NULL);
                ret = LLVMFuzzerTestOneInput(buf, size);
                TALLOC_FREE(buf);
        }
}
```

this is why all Samba fuzzers need the optional `LLVMFuzzerInitialize()`

Valgrind agrees there is a problem

```
$ valgrind bin/fuzz_parse_lpq_entry fuzz/results/fuzz_parse_lpq_entry/SIGABRT.PC.5555555c
==148585== Memcheck, a memory error detector
==148585== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
[...]
==148585==
==148585== Invalid read of size 1
==148585==    at 0x4D6DCBA: ____strtol_l_internal (strtol_l.c:292)
==148585==    by 0x4D69743: atoi (atoi.c:27)
==148585==    by 0x497C8B4: LPRng_time (lpq_parse.c:228)
==148585==    by 0x497CC04: parse_lpq_lprng (lpq_parse.c:300)
==148585==    by 0x497EF33: parse_lpq_entry (lpq_parse.c:1079)
==148585==    by 0x109771: LLVMFuzzerTestOneInput (fuzz_parse_lpq_entry.c:53)
==148585==    by 0x1092D9: main (afl-fuzz-main.c:46)
[...]
```

```
$ hd fuzz/results/fuzz_parse_lpq_entry/SIGABRT.PC.5555555dc1dc.STACK.c8fc8e0d.CODE.-6.ADD
00000000  66 73 6d 62 32 5f 63 72  65 64 69 74 73 38 6f 6e  |fsmb2_credits8on|
00000010  6c 69 6e 65 36 32 32 32  32 b0 2d 6e 6f 74 20 09  |line62222.-not .|
00000020  66 65 6e 63 65 20 80 80  80 80 70 72 69 6f 72 69  |fence ....priori|
00000030  64 6f 77 6e 20 32 75 6e  6e 36 32 32 32 32 b0 2d  |down 2unn62222.-|
00000040  6e 6f 74 20 09 66 63 65  20 80 80 80 80 70 72 69  |not .fce ....pri|
00000050  6f 72 69 64 6f 77 6e 20  32 75 6e 6e 69 74 c2 75  |oridown 2unnit.u|
00000060  65 2e 65 6e 64 73 64 32  32 70 32 69 64 62 73 61  |e.endsd22p2idbsa|
00000070  62 6c 65 64 2d 36 41 75  67 3a 2d 2d 20 32 6f 44  |bled-6Aug:-- 2oD|
00000080  65 63 73 61 62 6c 65 64  4d 61 72 75 67 3a 2d 2d  |ecsabledMarug:--|
00000090  20 32 65 02 00 00 00 2e  6f 44 65 63 66 66 6c 66  | 2e.....oDecfflf|
000000a0  72 65 65 d4 81 5e 64 32  01 00 00 00 00 00 00 00  |ree..^d2........|
000000b0  32 32 32 32 32 72 32 32  32 6e 69 74 c2 75 64 73  |22222r222nit.uds|
000000c0  64 62 5f 70 61 73 73 77  6f 72 64 5f 61 75 64 69  |db_password_audi|
000000d0  74 64 73 73 64 69 74 64  73                       |tdssditds|
000000d9
```

a small amount of time in gdb

```
        t = localtime(&jobtime);
        if (!t) {
                return (time_t)-1;
        }

        if ( atoi(time_string) < 24 ){
+               if (strlen(time_string) < 7) {
+                       return (time_t)-1;
+               }
                t->tm_hour = atoi(time_string);
                t->tm_min = atoi(time_string+3);
                t->tm_sec = atoi(time_string+6);
        } else {
+               if (strlen(time_string) < 18) {
+                       return (time_t)-1;
+               }
```

next steps

1. open printer bug, submit patch 🥇
2. run the fuzzer more
3. attempt to link to less than `smbd_base`
4. run it over a weekend
5. triage any crashes
6. minimise seeds
7. push minimised seeds to samba-fuzz-seeds
8. submit fuzzer upstream

Minimising seeds

```
$ fuzz/run bin/fuzz_parse_lpq_entry -M
```

finds a minimal set of seeds that cover all edges

if you have crashes, this will include crashes

Google's OSS-Fuzz

OSS-Fuzz continuously fuzzes participating open source projects

Including all upstream Samba fuzz targets

90 day disclosure

using seeds from samba-fuzz-seeds

example: https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=25504

Disclosing crashing seeds is disclosing crashes

Pushing fuzzers that quickly find crashes

- is a *bit* like disclosing crashes
- sets you up for a 90 day disclosure window

but pushing “safe” fuzzers seems a bit boring

Miscellaneous tips and tricks

Templating fuzzers (`fuzz_ndr_X`)

- Build and pre-processor trickery
- makes 159 fuzzers from 1 file
- fuzzing NDR pull, push, and print

splitting the blob

```
if (len < 6) {
        return 0;
}
if (len > MAX_LENGTH) {
        return 0;
}

pattern_len = PULL_LE_U16(input, 0);
insert_len = PULL_LE_U16(input, 2);
target_len = PULL_LE_U16(input, 4);
input += 6;
len -= 6;
if (pattern_len + insert_len > len) {
        return 0;
}
```

Input size

- edit `fuzz/run` (`MAXSIZE=...`)
- starting small is often good
- some effects are very size dependent

  (e.g. tdb mmap() sensitive to page size)

Undefined behaviour sanitizer

- Add `--undefined-sanitizer`
- does not crash, creates `HF.sanitizer.log.$PID` files
- can't map these pids to input files
- but the problem is usually clear

```
../../source3/printing/lpq_parse.c:406:28: runtime error: signed int
SUMMARY: UndefinedBehaviorSanitizer: undefined-behavior ../../source
```

False positives?

Address sanitizer sometimes sees what nothing else can

is it hallucinating?

does e.g. its `sscanf()` work differently?

False positives — timeouts

In some places, we realloc a byte at a time in a loop:

```
things = talloc_realloc(mem_ctx, things, struct thing, num_things + 1);
num_things++;
```

realloc is *very* slow under address sanitizer (~100x)

Problems

tevent

How to crank the handle to take a tevent fuzzer though its lifecycle?

So much context

- Some functions seem to need the entire server as context

lack of tests

Fuzz targets are generalised unit tests (sans assertions)

fuzzers fall naturally out of unit-tests

i.e. unit-testable *is* fuzzable

The case in point

```
static int generic_queue_get(const char *printer_name,
                             enum printing_types printing_type,
                             char *lpq_command,
                             print_queue_struct **q,
                             print_status_struct *status)
{
        char **qlines;
        int fd;
        int numlines, i, qcount;
        print_queue_struct *queue = NULL;

        /* never do substitution when running the 'lpq command' since we can't
           get it right when using the background update daemon.  Make the calle
           do it before passing off the command string to us here. */

        print_run_command(-1, printer_name, False, lpq_command, &fd, NULL);

        if (fd == -1) {
                DEBUG(5,("generic_queue_get: Can't read print queue status for p
```

Nobody wants to pay for security

- Team members or their employers foot the bill

Fuzzing is just as easy for the bad guys

They benefit from open source too!

?