KFuzzTest

Fuzzing the Unfuzzable Inside the Kernel

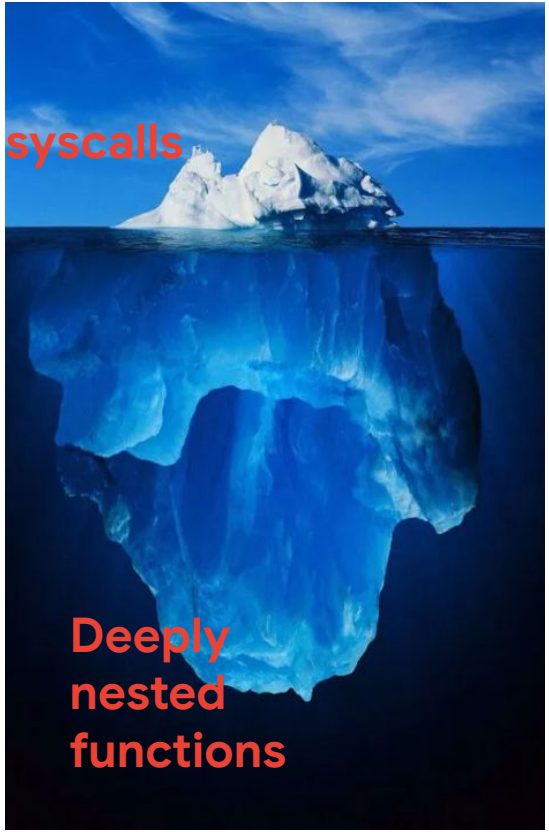
Ethan Graham *ETH Zürich*

The Testing Gap

Many critical functions are hard to fuzz

- Parsers
- Format converters
- New code being added to the kernel

Unit testing doesn't help us find new test cases



Goal

- Make it easy to fuzz individual functions
- Integrate with existing fuzzing infrastructure
- Help find and prevent bugs in the kernel

KFuzzTest in Action

KFuzzTest Example

Let's fuzz the following function:

```
/**
 * pkcs7_parse_message - Parse a PKCS#7 message
 * @data: The raw binary ASN.1 encoded message to be parsed
 * @datalen: The size of the encoded message
 */
struct pkcs7_message *pkcs7_parse_message(const void *data, size_t datalen)
```

The Simple Case

```
FUZZ_TEST_SIMPLE(test_pkcs7_parse_message)
{
    pkcs7_parse_message(data, datalen);
}
```

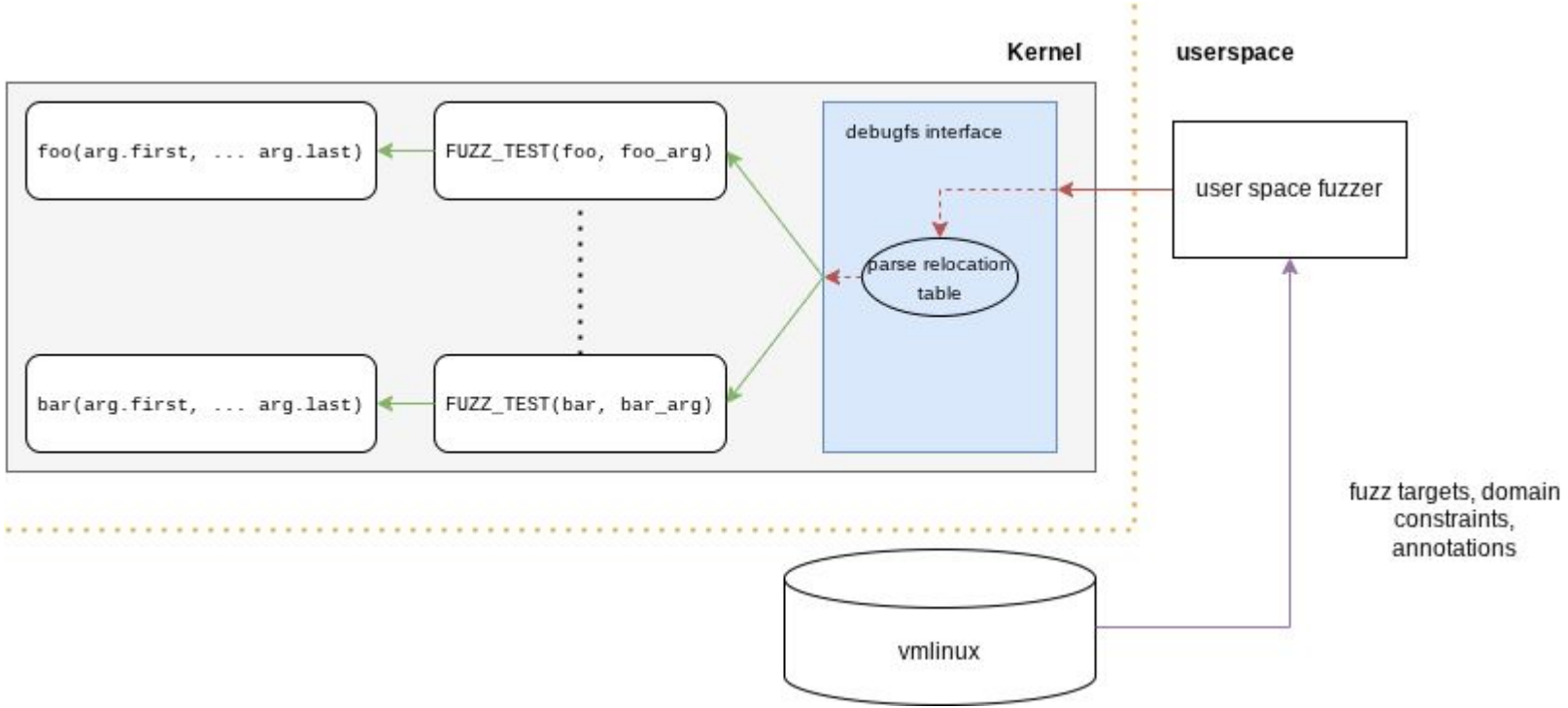
The General Case

```
struct pkcs7_parse_message_arg {
    const void *data;
    size_t datalen;
};

FUZZ_TEST(test_pkcs7_parse_message, struct pkcs7_parse_message_arg)
{
    KFUZZTEST_EXPECT_NOT_NULL(pkcs7_parse_message_arg, data);
    KFUZZTEST_ANNOTATE_ARRAY(pkcs7_parse_message_arg, data);
    KFUZZTEST_ANNOTATE_LEN(pkcs7_parse_message_arg, datalen, data);

    pkcs7_parse_message(arg->data, arg->datalen);
}
```

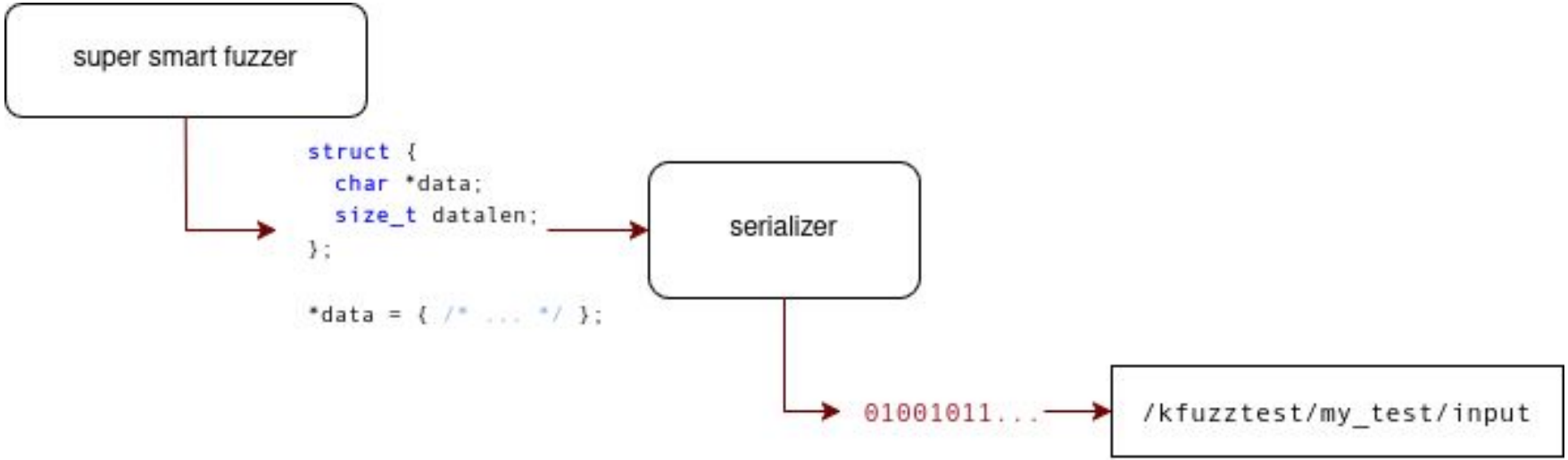
Constraints and Annotations

```
struct pkcs7_parse_message_arg {
    const void *data;
    size_t datalen;
};

FUZZ_TEST(test_pkcs7_parse_message, struct pkcs7_parse_message_arg)
{
    KFUZZTEST_EXPECT_NOT_NULL(pkcs7_parse_message_arg, data);
    KFUZZTEST_ANNOTATE_ARRAY(pkcs7_parse_message_arg, data);
    KFUZZTEST_ANNOTATE_LEN(pkcs7_parse_message_arg, datalen, data);

    pkcs7_parse_message(arg->data, arg->datalen);
}
```

High-level Design



Invoking a KFuzzTest Target



Proof of the Hunt

Case Study - PKCS#7 Parser

Simulate a realistic off-by-one bug in a real function

```
- ret = asn1_ber_decoder(&pkcs7_decoder, ctx, data, datalen);
+ ret = asn1_ber_decoder(&pkcs7_decoder, ctx, data, datalen + 1);
```

Case Study - PKCS#7 Parser

```c
struct pkcs7_message *pkcs7_parse_message(const void *data, size_t datalen)
{
    struct pkcs7_parse_context *ctx;
    struct pkcs7_message *msg = ERR_PTR(-ENOMEM);
    int ret;

    ctx = kzalloc(sizeof(struct pkcs7_parse_context), GFP_KERNEL);
    if (!ctx)
        goto out_no_ctx;
    ctx->msg = kzalloc(sizeof(struct pkcs7_message), GFP_KERNEL);
    if (!ctx->msg)
        goto out_no_msg;
    ctx->sinfo = kzalloc(sizeof(struct pkcs7_signed_info), GFP_KERNEL);
    if (!ctx->sinfo)
        goto out_no_sinfo;
    ctx->sinfo->sig = kzalloc(sizeof(struct public_key_signature),
                              GFP_KERNEL);
    if (!ctx->sinfo->sig)
        goto out_no_sig;

    ctx->data = (unsigned long)data;
    ctx->ppcerts = &ctx->certs;
    ctx->ppsinfo = &ctx->msg->signed_infos;

    /* Attempt to decode the signature */
    ret = asn1_ber_decoder(&pkcs7_decoder, ctx, data, datalen + 1);
    if (ret < 0) {
        msg = ERR_PTR(ret);
        goto out;
    }

    ret = pkcs7_check_authattrs(ctx->msg);
    if (ret < 0) {
        msg = ERR_PTR(ret);
        goto out;
    }

    msg = ctx->msg;
    ctx->msg = NULL;

out:
    while (ctx->certs) {
        struct x509_certificate *cert = ctx->certs;
        ctx->certs = cert->next;
        x509_free_certificate(cert);
    }
out_no_sig:
    pkcs7_free_signed_info(ctx->sinfo);
out_no_sinfo:
    pkcs7_free_message(ctx->msg);
out_no_msg:
    kfree(ctx);
out_no_ctx:
    return msg;
}
EXPORT_SYMBOL_GPL(pkcs7_parse_message);
```

Case Study - PKCS#7 Parser

```
struct pkcs7_message *pkcs7_parse_message(const void *data, size_t datalen)
{
    struct pkcs7_parse_context *ctx;
    struct pkcs7_message *msg = ERR_PTR(-ENOMEM);
    int ret;

    ctx = kzalloc(sizeof(struct pkcs7_parse_context), GFP_KERNEL);
    if (!ctx)
        goto out_no_ctx;
    ctx->msg = kzalloc(sizeof(struct pkcs7_message), GFP_KERNEL);
    if (!ctx->msg)
        goto out_no_msg;
    ctx->sinfo = kzalloc(sizeof(struct pkcs7_signed_info), GFP_KERNEL);
    if (!ctx->sinfo)
        goto out_no_sinfo;
    ctx->sinfo->sig = kzalloc(sizeof(struct public_key_signature),
                GFP_KERNEL);
    if (!ctx->sinfo->sig)
        goto out_no_sig;

    ctx->data = (unsigned long)data;
    ctx->ppcerts = &ctx->certs;
    ctx->ppsinfo = &ctx->msg->signed_infos;

    /* Attempt to decode the signature */
    ret = asn1_ber_decoder(&pkcs7_decoder, ctx, data, datalen + 1);
    if (ret < 0) {
        msg = ERR_PTR(ret);
        goto out;
    }

    ret = pkcs7_check_authattrs(ctx->msg);
    if (ret < 0) {
        msg = ERR_PTR(ret);
        goto out;
    }

    msg = ctx->msg;
    ctx->msg = NULL;

out:
    while (ctx->certs) {
        struct x509_certificate *cert = ctx->certs;
        ctx->certs = cert->next;
        x509_free_certificate(cert);
    }
out_no_sig:
    pkcs7_free_signed_info(ctx->sinfo);
out_no_sinfo:
    pkcs7_free_message(ctx->msg);
out_no_msg:
    kfree(ctx);
out_no_ctx:
    return msg;
}
EXPORT_SYMBOL_GPL(pkcs7_parse_message);
```

Bug is here!

Case Study - PKCS#7 Parser

```
BUG: KASAN: slab-out-of-bounds in asn1_ber_decoder+0×1533/0×1600 lib/asn1_decoder.c:260
Read of size 1 at addr ffff88800badeb51 by task syz.4.623/5080

CPU: 0 UID: 0 PID: 5080 Comm: syz.4.623 Tainted: G                 N  6.17.0-rc4+ #130
PREEMPT(voluntary)
Tainted: [N]=TEST
Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.16.3-debian-1.16.3-2 04/01/2014
Call Trace:
 <TASK>
 __dump_stack lib/dump_stack.c:94 [inline]
 dump_stack_lvl+0×71/0×a0 lib/dump_stack.c:120
 print_address_description mm/kasan/report.c:378 [inline]
 print_report+0×174/0×4f6 mm/kasan/report.c:482
 kasan_report+0×ce/0×100 mm/kasan/report.c:595
 asn1_ber_decoder+0×1533/0×1600 lib/asn1_decoder.c:260
 pkcs7_parse_message+0×27f/0×6d0 crypto/asymmetric_keys/pkcs7_parser.c:140
 kfuzztest_logic_test_pkcs7_parse_message crypto/asymmetric_keys/tests/pkcs7_kfuzz.c:21 [inline]
 kfuzztest_write_cb_test_pkcs7_parse_message+0×18f/0×1d0
crypto/asymmetric_keys/tests/pkcs7_kfuzz.c:15
RIP: 0033:0×7fbf3716ad8e
```

Points of Contention

Serialization Complexity

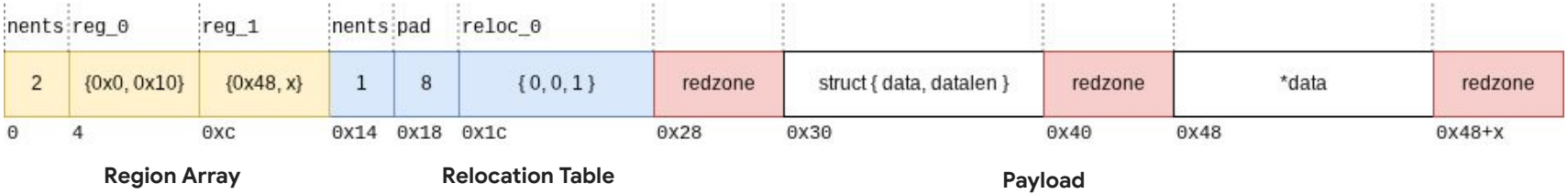
KFuzzTest relies on a custom serialization format

- Express pointer-pointee relationships
- Sandbox fuzzer inputs through validation

But lots of plumbing needed by a fuzzer...

Serialization Complexity

```
struct {
    char *data;
    size_t datalen;
}
```



KFuzzTest Today

KFuzzTest Code

- Under review
  - Currently Patch v3
- General interest
  - More tests == good
- Hoping to get more fuzz targets upstream

syzkaller

- Native support for KFuzzTest out of the box
  - Dynamic target discovery → **no need to write descriptions**
  - Continuous fuzzing of checked-in fuzz targets
- Support for a new “local” fuzzing program
  - Like syzkaller, but no orchestration
  - “syzkaller-lite”

Other fuzzers

- `kfuzztest-bridge` → turn blobs into structured inputs
  - E.g., `/dev/urandom`
- FUZZ_TEST_SIMPLE supports a simplified interface
  - Easy support for `LLVMFuzzOneInput()`

Thanks!