In-Process Fuzzing With Frida

WiSec 2021 Tutorials

Dennis Heinze
ERNW GmbH

ERNW
providing security.

Fuzzing with Frida

https://github.com/seemoo-lab/toothpicker

https://github.com/ttdennis/fpicker

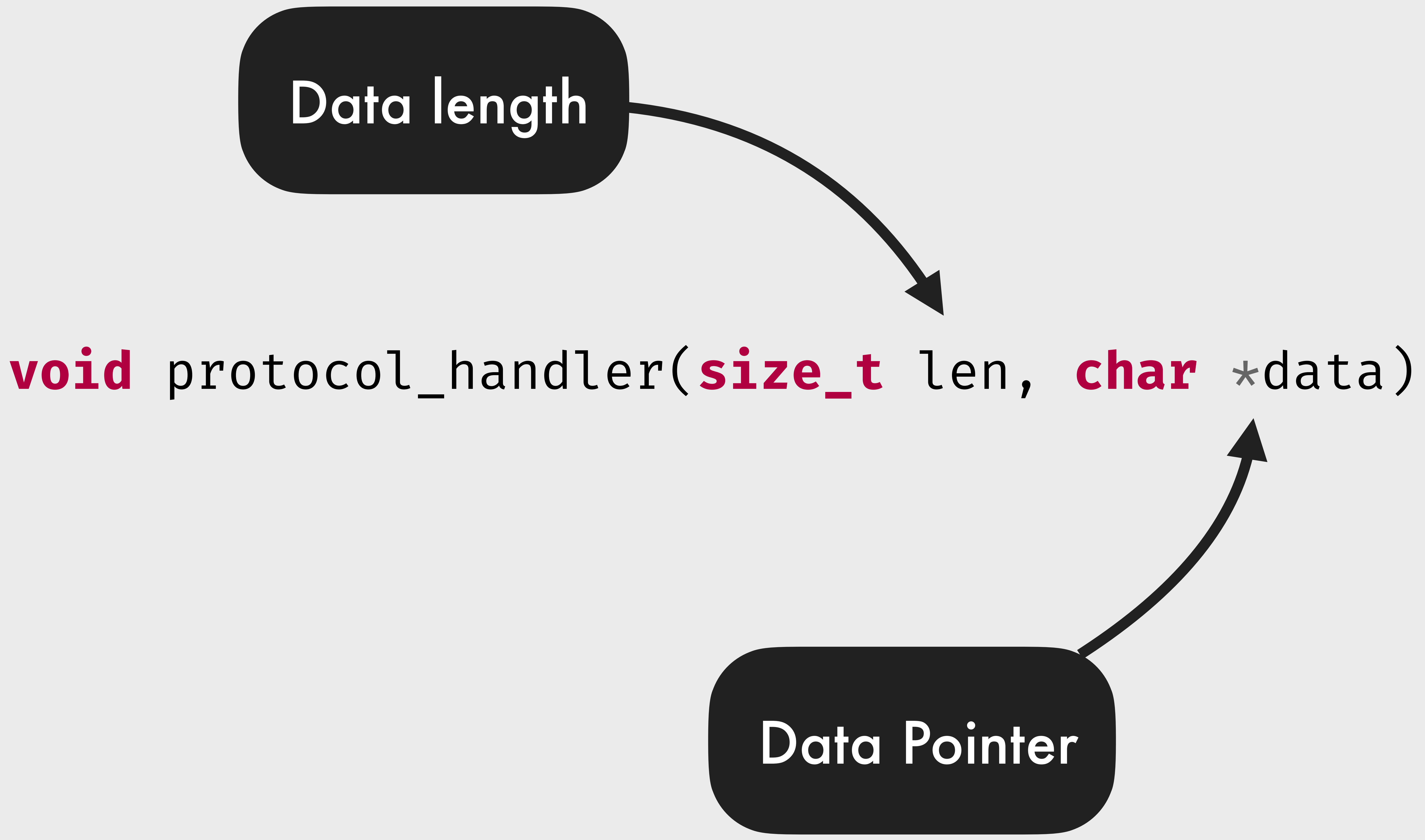
FRIDA

General installation:	https://frida.re/docs/installation/
Frida on iOS:	https://frida.re/docs/ios/
Frida on Android:	https://frida.re/docs/android/

```c
void protocol_handler(size_t len, char *data)
```

Data length

```
void protocol_handler(size_t len, char *data)
```



```
void protocol_handler(size_t len, char *data)
```

```c
void protocol_handler(size_t len, char *data)
```

```javascript
Interceptor.attach(Module.getExportByName(null, 'protocol_handler'), {
  onEnter(args) {
    console.log('data length: ' + args[0]);
    console.log(Memory.readByteArray(args[1], parseInt(args[0])));
  }
});
```

```c
void protocol_handler(size_t len, char *data)
```

```javascript
Interceptor.attach(Module.getExportByName(null, 'protocol_handler'), {
  onEnter(args) {
    console.log('data length: ' + args[0]);
    console.log(Memory.readByteArray(args[1], parseInt(args[0])));
  }
});
```

```c
void protocol_handler(size_t len, char *data)
```

```javascript
Interceptor.attach(Module.getExportByName(null, 'protocol_handler'), {
  onEnter(args) {
    console.log('data length: ' + args[0]);
    console.log(Memory.readByteArray(args[1], parseInt(args[0])));
  }
});
```

```c
void protocol_handler(size_t len, char *data)
```

```javascript
Interceptor.attach(Module.getExportByName(null, 'protocol_handler'), {
  onEnter(args) {
    console.log('data length: ' + args[0]);
    console.log(Memory.readByteArray(args[1], parseInt(args[0])));
  }
});
```

```
zsh
❯ ./protocol_handler 8081
```

```
zsh
❯ nc localhost 8081
```

```
zsh
❯
```

```
❯ ./protocol_handler 8081
```

```
❯ nc localhost 8081
```

```
❯
```

```c
void protocol_handler(size_t len, char *data)
```

```javascript
Interceptor.attach(Module.getExportByName(null, 'protocol_handler'), {
  onEnter(args) {
    Memory.writeByteArray(args[1], [0x41, 0x41, 0x41, 0x41]);
  }
});
```

```c
void protocol_handler(size_t len, char *data)
```

```javascript
Interceptor.attach(Module.getExportByName(null, 'protocol_handler'), {
  onEnter(args) {
    Memory.writeByteArray(args[1], [0x41, 0x41, 0x41, 0x41]);
  }
});
```

See ToothFlipper by jiska: https://github.com/seemoo-lab/toothpicker/tree/master/inplace-fuzzer

```
void protocol_handler(size_t len, char *data)
```

```
Interceptor.attach(Module.getExportByName(null, 'protocol_handler'), {
  onEnter(args) {
    Memory.writeByteArray(args[1], [0x41, 0x41, 0x41, 0x41]);
  }
});
```

See ToothFlipper by jiska: https://github.com/seemoo-lab/toothpicker/tree/master/inplace-fuzzer

```
zsh
❯ ./protocol_handler 8081
```

```
zsh
❯ nc localhost 8081
```

```
zsh
❯
```

```
zsh
❯ ./protocol_handler 8081
```

```
zsh
❯ nc localhost 8081
```

```
zsh
❯
```

```c
void protocol_handler(size_t len, char *data)
```

```javascript
const addr = Module.getExportByName(null, "protocol_handler");
const protocol_handler = new NativeFunction(addr, 'void', ['int', 'pointer']);

const str = "sample buffer\n";
const buf = Memory.allocUtf8String(str);
const len = str.length;

protocol_handler(len, buf);
```

```c
void protocol_handler(size_t len, char *data)
```

```javascript
const addr = Module.getExportByName(null, "protocol_handler");
const protocol_handler = new NativeFunction(addr, 'void', ['int', 'pointer']);

const str = "sample buffer\n";
const buf = Memory.allocUtf8String(str);
const len = str.length;

protocol_handler(len, buf);
```

```c
void protocol_handler(size_t len, char *data)
```

```javascript
const addr = Module.getExportByName(null, "protocol_handler");
const protocol_handler = new NativeFunction(addr, 'void', ['int', 'pointer']);

const str = "sample buffer\n";
const buf = Memory.allocUtf8String(str);
const len = str.length;

protocol_handler(len, buf);
```

```
zsh
❯ ./protocol_handler 8081
```

```
zsh
❯
```

```
zsh
❯ ./protocol_handler 8081
```

```
zsh
❯
```

Fuzzing

```c
void protocol_handler(size_t len, char *data)
```

```javascript
const addr = Module.getExportByName(null, "protocol_handler");
const protocol_handler = new NativeFunction(addr, 'void', ['int', 'pointer']);

const chars = [..."abcdefghijklmnopqrstuvwxyz0123456789 ,.-+#*'=ß?"];

while (true) {
    const len = Math.floor(Math.random() * 80);
    const rand = [...Array(len)].map(i=>chars[Math.random()*chars.length|0]).join('');

    const buf = Memory.allocUtf8String(rand + '\n');
    protocol_handler(len, buf);
}
```

```c
void protocol_handler(size_t len, char *data)
```

```javascript
const addr = Module.getExportByName(null, "protocol_handler");
const protocol_handler = new NativeFunction(addr, 'void', ['int', 'pointer']);

const chars = [..."abcdefghijklmnopqrstuvwxyz0123456789 ,.-+#*'=ß?"];

while (true) {
    const len = Math.floor(Math.random() * 80);
    const rand = [...Array(len)].map(i=>chars[Math.random()*chars.length|0]).join('');

    const buf = Memory.allocUtf8String(rand + '\n');
    protocol_handler(len, buf);
}
```

```c
void protocol_handler(size_t len, char *data)
```

```javascript
const addr = Module.getExportByName(null, "protocol_handler");
const protocol_handler = new NativeFunction(addr, 'void', ['int', 'pointer']);

const chars = [..."abcdefghijklmnopqrstuvwxyz0123456789 ,•-+#*'=ß?"];

while (true) {
    const len = Math.floor(Math.random() * 80);
    const rand = [...Array(len)].map(i=>chars[Math.random()*chars.length|0]).join('');

    const buf = Memory.allocUtf8String(rand + '\n');
    protocol_handler(len, buf);
}
```

```c
void protocol_handler(size_t len, char *data)
```

```javascript
const addr = Module.getExportByName(null, "protocol_handler");
const protocol_handler = new NativeFunction(addr, 'void', ['int', 'pointer']);

const chars = [..."abcdefghijklmnopqrstuvwxyz0123456789 ,.-+#*'=ß?"];

while (true) {
    const len = Math.floor(Math.random() * 80);
    const rand = [...Array(len)].map(i=>chars[Math.random()*chars.length|0]).join('');

    const buf = Memory.allocUtf8String(rand + '\n');
    protocol_handler(len, buf);
}
```

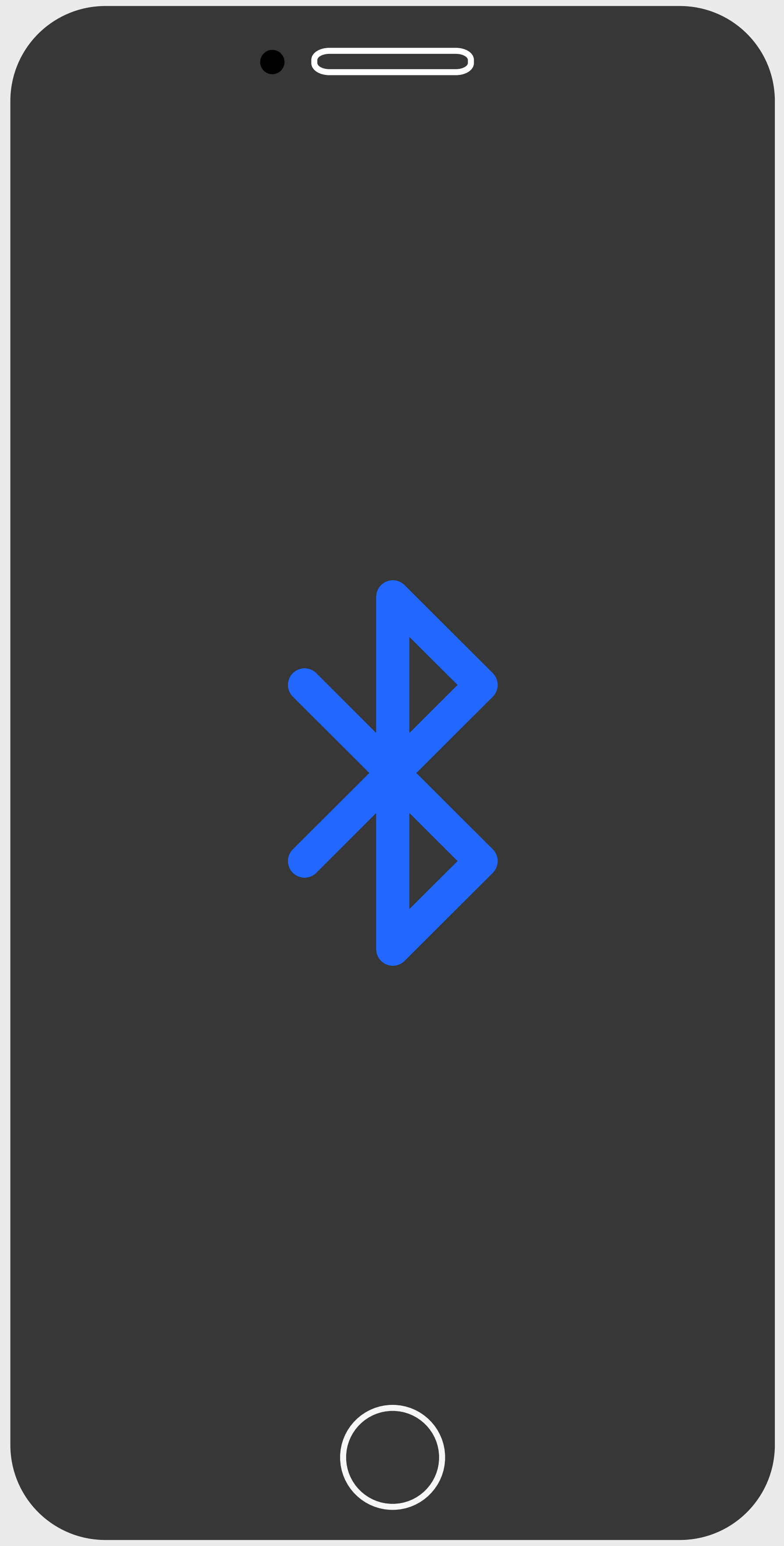
```
❯ ./protocol_handler 8081
```

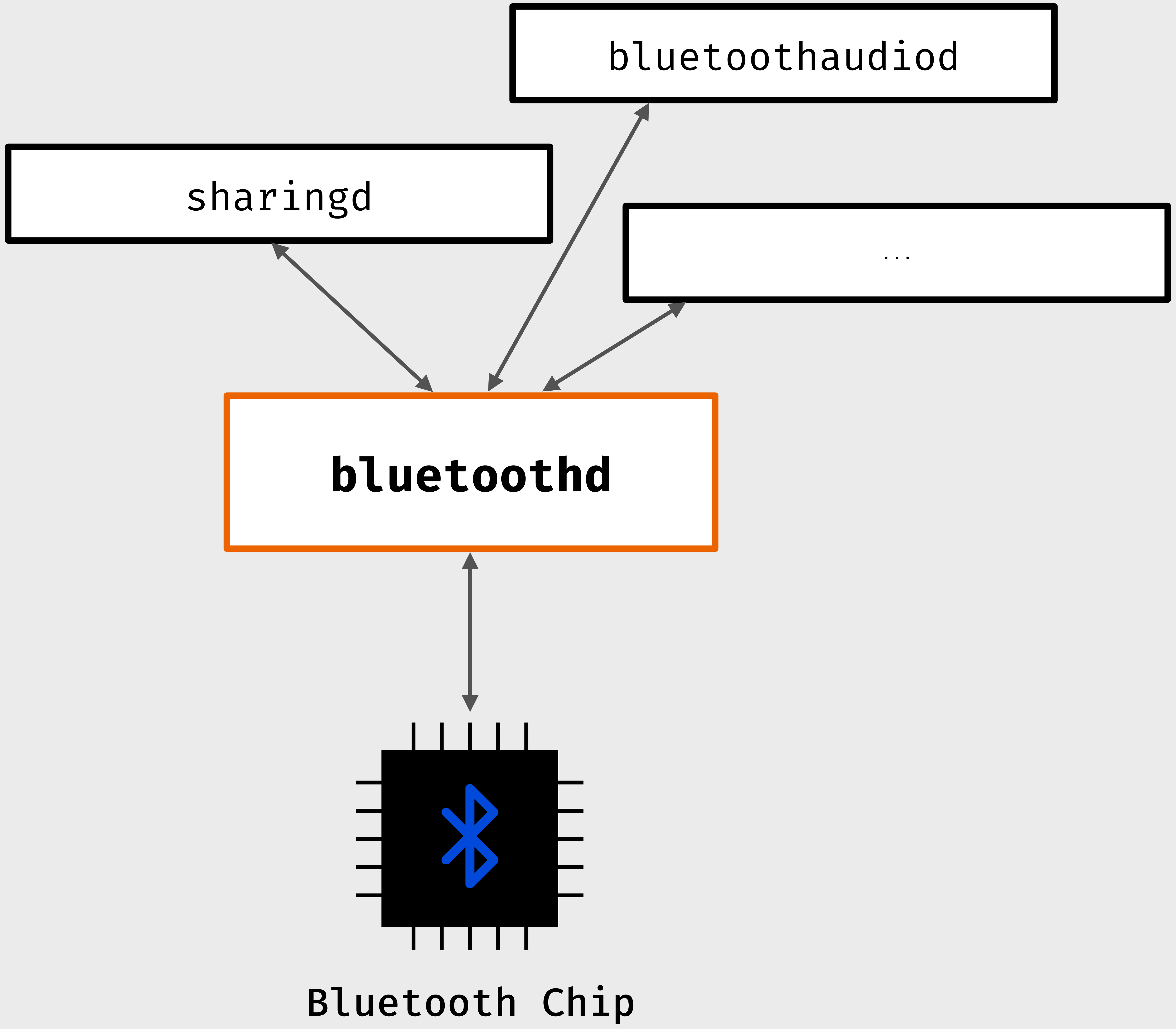
```
❯
```

```
zsh
❯ ./protocol_handler 8081
```

```
zsh
❯
```

Fuzzing iOS Bluetooth




bluetoothaudiod
sharingd
...
bluetoothd
Bluetooth Chip

```c
void acl_reception_handler(short handle, size_t len, char *data)
void acl_reception_handler(short handle, size_t len, char *data)
```

Connection handle value of the Bluetooth connection

```c
void acl_reception_handler(short handle, size_t len, char *data)
```

Connection handle value of the Bluetooth connection

Data and length of received ACL data

```c
void acl_reception_handler(short handle, size_t len, char *data)
```

Connection handle value of the Bluetooth connection

We need this!

Data and length of received ACL data

```c
bt_connection_t *create_connection(char *bd_addr, int state)
bt_connection_t *create_connection(char *bd_addr, int state)
```

Create a Bluetooth connection structure

```c
bt_connection_t *create_connection(char *bd_addr, int state)
```

Create a Bluetooth connection structure

Set the handle value of the connection:

```c
*(short*)connection = 0x11;
```

```c
bt_connection_t *create_connection(char *bd_addr, int state)
```

Create a Bluetooth connection structure

Set the handle value of the connection:

```c
*(short*)connection = 0x11;
```

Now we can call the reception handler with our fuzzing data

```c
acl_reception_handler(0x11, len, data);
```

```c
bt_connection_t *create_connection(char *bd_addr, int state)
```

```javascript
const fn_addr = base.add(symbols.create_acl_connection);
const create_connection = new NativeFunction(fn_addr.sign(), "pointer",
    ["pointer", "char"]);

const bd_addr = Memory.alloc(6);
bd_addr.writeByteArray([0xf4, 0xaf, 0xe7, 0x15, 0x51, 0xbc]);

const handle = create_connection(bd_addr, 1);
if (handle == 0) {
    console.error("Handle with this BD addr probably already exists.");
}
Memory.writeShort(handle, 0x11);
```

```c
bt_connection_t *create_connection(char *bd_addr, int state)
```

```javascript
const fn_addr = base.add(symbols.create_acl_connection);
const create_connection = new NativeFunction(fn_addr.sign(), "pointer",
    ["pointer", "char"]);

const bd_addr = Memory.alloc(6);
bd_addr.writeByteArray([0xf4, 0xaf, 0xe7, 0x15, 0x51, 0xbc]);

const handle = create_connection(bd_addr, 1);
if (handle == 0) {
    console.error("Handle with this BD addr probably already exists.");
}
Memory.writeShort(handle, 0x11);
```

```c
bt_connection_t *create_connection(char *bd_addr, int state)
```

```javascript
const fn_addr = base.add(symbols.create_acl_connection);
const create_connection = new NativeFunction(fn_addr.sign(), "pointer",
    ["pointer", "char"]);

const bd_addr = Memory.alloc(6);
bd_addr.writeByteArray([0xf4, 0xaf, 0xe7, 0x15, 0x51, 0xbc]);

const handle = create_connection(bd_addr, 1);
if (handle == 0) {
    console.error("Handle with this BD addr probably already exists.");
}
Memory.writeShort(handle, 0x11);
```

```c
bt_connection_t *create_connection(char *bd_addr, int state)
```

```javascript
const fn_addr = base.add(symbols.create_acl_connection);
const create_connection = new NativeFunction(fn_addr.sign(), "pointer",
    ["pointer", "char"]);

const bd_addr = Memory.alloc(6);
bd_addr.writeByteArray([0xf4, 0xaf, 0xe7, 0x15, 0x51, 0xbc]);

const handle = create_connection(bd_addr, 1);
if (handle == 0) {
    console.error("Handle with this BD addr probably already exists.");
}
Memory.writeShort(handle, 0x11);
```

```javascript
Interceptor.replace(base.add(symbols.bt_forceDisconnect),
    new NativeCallback(function(a, b) {
        return 1;
    }, "int", ["pointer", "pointer"])
);

const startSecurityPolicyEnforcement_addr =
    base.add(symbols.startSecurityPolicyEnforcement);
const register_timeout_addr = base.add(symbols.registerTimeout);
const orig_register_timeout = new NativeFunction(register_timeout_addr.sign(),
    "int64", ["pointer", "pointer", "pointer", "pointer"]);

Interceptor.replace(register_timeout_addr, new NativeCallback(function(fn, b, c, d) {
    if(parseInt(fn,16) == parseInt(startSecurityPolicyEnforcement_addr,16)) {
        return 0;
    }
    return orig_register_timeout(fn, b, c, d);
}, "int64", ["pointer", "pointer", "pointer", "pointer"]));
```

```
Interceptor.replace(base.add(symbols.bt_forceDisconnect),
    new NativeCallback(function(a, b) {
        return 1;
    }, "int", ["pointer", "pointer"])
);

const startSecurityPolicyEnforcement_addr =
    base.add(symbols.startSecurityPolicyEnforcement);
const register_timeout_addr = base.add(symbols.registerTimeout);
const orig_register_timeout = new NativeFunction(register_timeout_addr.sign(),
    "int64", ["pointer", "pointer", "pointer", "pointer"]);

Interceptor.replace(register_timeout_addr, new NativeCallback(function(fn, b, c, d) {
    if(parseInt(fn,16) == parseInt(startSecurityPolicyEnforcement_addr,16)) {
        return 0;
    }
    return orig_register_timeout(fn, b, c, d);
}, "int64", ["pointer", "pointer", "pointer", "pointer"]));
```

```
Interceptor.replace(base.add(symbols.bt_forceDisconnect),
    new NativeCallback(function(a, b) {
        return 1;
    }, "int", ["pointer", "pointer"])
);

const startSecurityPolicyEnforcement_addr =
    base.add(symbols.startSecurityPolicyEnforcement);
const register_timeout_addr = base.add(symbols.registerTimeout);
const orig_register_timeout = new NativeFunction(register_timeout_addr.sign(),
    "int64", ["pointer", "pointer", "pointer", "pointer"]);

Interceptor.replace(register_timeout_addr, new NativeCallback(function(fn, b, c, d) {
    if(parseInt(fn,16) == parseInt(startSecurityPolicyEnforcement_addr,16)) {
        return 0;
    }
    return orig_register_timeout(fn, b, c, d);
}, "int64", ["pointer", "pointer", "pointer", "pointer"]));
```

```javascript
Interceptor.replace(base.add(symbols.bt_forceDisconnect),
    new NativeCallback(function(a, b) {
        return 1;
    }, "int", ["pointer", "pointer"])
);

const startSecurityPolicyEnforcement_addr =
    base.add(symbols.startSecurityPolicyEnforcement);
const register_timeout_addr = base.add(symbols.registerTimeout);
const orig_register_timeout = new NativeFunction(register_timeout_addr.sign(),
    "int64", ["pointer", "pointer", "pointer", "pointer"]);

Interceptor.replace(register_timeout_addr, new NativeCallback(function(fn, b, c, d) {
    if(parseInt(fn,16) == parseInt(startSecurityPolicyEnforcement_addr,16)) {
        return 0;
    }
    return orig_register_timeout(fn, b, c, d);
}, "int64", ["pointer", "pointer", "pointer", "pointer"]));
```

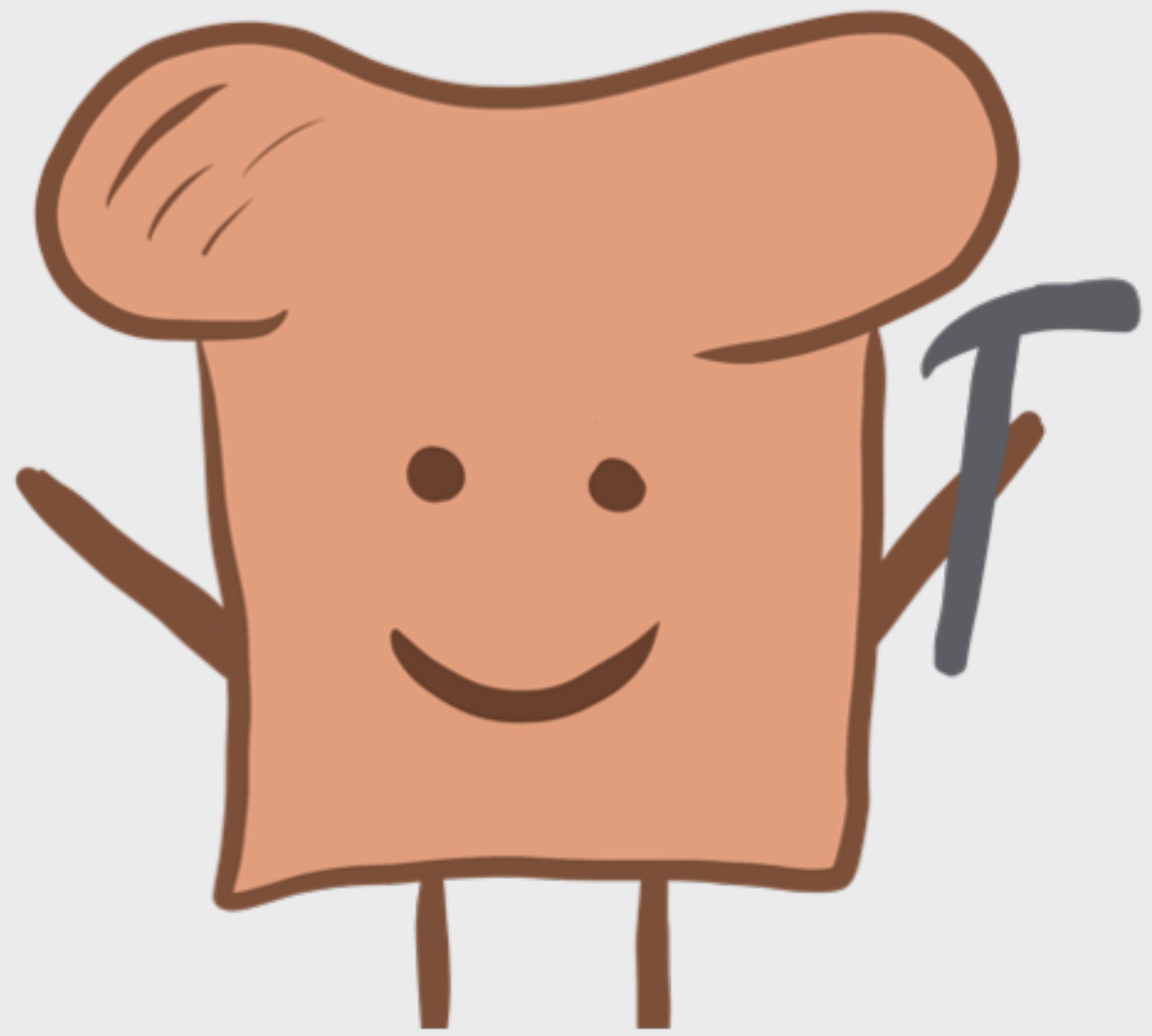
```
Interceptor.replace(base.add(symbols.bt_forceDisconnect),
    new NativeCallback(function(a, b) {
        return 1;
    }, "int", ["pointer", "pointer"])
);

const startSecurityPolicyEnforcement_addr =
    base.add(symbols.startSecurityPolicyEnforcement);
const register_timeout_addr = base.add(symbols.registerTimeout);
const orig_register_timeout = new NativeFunction(register_timeout_addr.sign(),
    "int64", ["pointer", "pointer", "pointer", "pointer"]);

Interceptor.replace(register_timeout_addr, new NativeCallback(function(fn, b, c, d) {
    if(parseInt(fn,16) == parseInt(startSecurityPolicyEnforcement_addr,16)) {
        return 0;
    }
    return orig_register_timeout(fn, b, c, d);
}, "int64", ["pointer", "pointer", "pointer", "pointer"]));
```

More details on iOS Bluetooth (fuzzing):

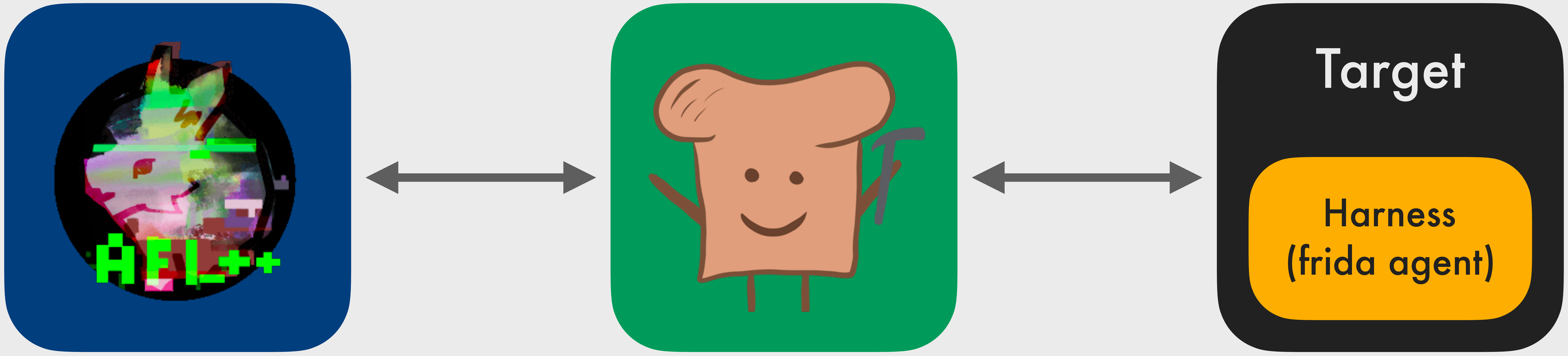
https://github.com/seemoo-lab/toothpicker

https://www.usenix.org/system/files/woot20-paper-heinze.pdf

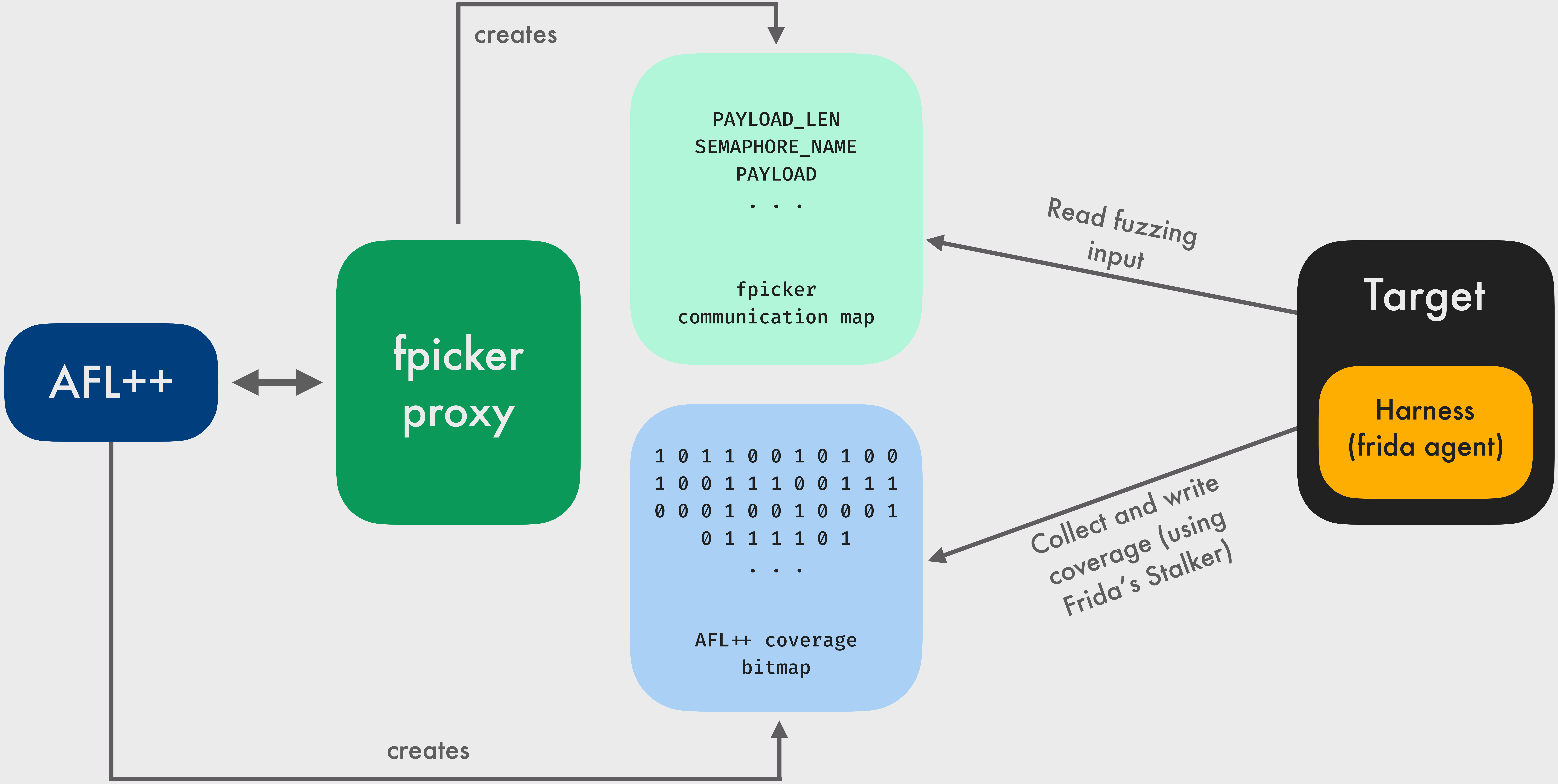
fpicker



fpicker



```
afl-fuzz -i ./in -o ./out -- ./fpicker -m afl -u shm -e attach -p target -f harness.js
```


creates
PAYLOAD_LEN
SEMAPHORE_NAME
PAYLOAD
. . .
fpicker communication map
Read fuzzing input
Target
Harness (frida agent)
AFL++
fpicker proxy
1 0 1 1 0 0 1 0 1 0 0
1 0 0 1 1 1 0 0 1 1 1
0 0 0 1 0 0 1 0 0 0 1
0 1 1 1 1 0 1
. . .
AFL++ coverage bitmap
Collect and write coverage (using Frida's Stalker)
creates

More details on fpicker and AFL++ integration:

https://github.com/ttdennis/fpicker

https://insinuator.net/2021/03/fpicker-fuzzing-with-frida/

```
const Fuzzer = require("../../harness/fuzzer.js");

class TestFuzzer extends Fuzzer.Fuzzer {
    constructor() {
        const fn_addr = Module.getExportByName(null, "protocol_handler");
        const protocol_handler = new NativeFunction(proc_fn_addr,
            "void", ["int", "pointer"]);

        super("protocol_handler", protocol_handler, fn_addr);
    }

    prepare() {}

    fuzz(payload, len) {
        this.target_function(parseInt(len), payload);
    }
}
const f = new TestFuzzer();
exports.fuzzer = f;
```

```
const Fuzzer = require("../../harness/fuzzer.js");

class TestFuzzer extends Fuzzer.Fuzzer {
    constructor() {
        const fn_addr = Module.getExportByName(null, "protocol_handler");
        const protocol_handler = new NativeFunction(proc_fn_addr,
            "void", ["int", "pointer"]);

        super("protocol_handler", protocol_handler, fn_addr);
    }

    prepare() {}

    fuzz(payload, len) {
        this.target_function(parseInt(len), payload);
    }
}
const f = new TestFuzzer();
exports.fuzzer = f;
```

```
const Fuzzer = require("../../harness/fuzzer.js");

class TestFuzzer extends Fuzzer.Fuzzer {
    constructor() {
        const fn_addr = Module.getExportByName(null, "protocol_handler");
        const protocol_handler = new NativeFunction(proc_fn_addr,
            "void", ["int", "pointer"]);

        super("protocol_handler", protocol_handler, fn_addr);
    }

    prepare() {}

    fuzz(payload, len) {
        this.target_function(parseInt(len), payload);
    }
}
const f = new TestFuzzer();
exports.fuzzer = f;
```

```
const Fuzzer = require("../../harness/fuzzer.js");

class TestFuzzer extends Fuzzer.Fuzzer {
    constructor() {
        const fn_addr = Module.getExportByName(null, "protocol_handler");
        const protocol_handler = new NativeFunction(proc_fn_addr,
            "void", ["int", "pointer"]);

        super("protocol_handler", protocol_handler, fn_addr);
    }

    prepare() {}

    fuzz(payload, len) {
        this.target_function(parseInt(len), payload);
    }
}
const f = new TestFuzzer();
exports.fuzzer = f;
```

```
const Fuzzer = require("../../harness/fuzzer.js");

class TestFuzzer extends Fuzzer.Fuzzer {
    constructor() {
        const fn_addr = Module.getExportByName(null, "protocol_handler");
        const protocol_handler = new NativeFunction(proc_fn_addr,
            "void", ["int", "pointer"]);

        super("protocol_handler", protocol_handler, fn_addr);
    }

    prepare() {}

    fuzz(payload, len) {
        this.target_function(parseInt(len), payload);
    }
}
const f = new TestFuzzer();
exports.fuzzer = f;
```

```
const Fuzzer = require("../../harness/fuzzer.js");

class TestFuzzer extends Fuzzer.Fuzzer {
    constructor() {
        const fn_addr = Module.getExportByName(null, "protocol_handler");
        const protocol_handler = new NativeFunction(proc_fn_addr,
            "void", ["int", "pointer"]);

        super("protocol_handler", protocol_handler, fn_addr);
    }

    prepare() {}

    fuzz(payload, len) {
        this.target_function(parseInt(len), payload);
    }
}
const f = new TestFuzzer();
exports.fuzzer = f;
```

```
const fuzzer = require("../../harness/fuzzer.js");
const bluetoothd = require("./bluetoothd.js");

const base = Module.getBaseAddress("bluetoothd");

class ACLFuzzer extends fuzzer.Fuzzer {
    constructor() {

        const acl_rec_addr = base.add(bluetoothd.symbols.ACL_reception_handler);
        const acl_reception_handler = new NativeFunction(
            acl_rec_addr, "void", ["int64", "int64", "pointer"]);

        super("bluetoothd", acl_reception_handler, acl_rec_addr);
        this.payload_buffer = Memory.alloc(0x100);
    }
```

```javascript
const fuzzer = require("../../harness/fuzzer.js");
const bluetoothd = require("./bluetoothd.js");

const base = Module.getBaseAddress("bluetoothd");

class ACLFuzzer extends fuzzer.Fuzzer {
    constructor() {

        const acl_rec_addr = base.add(bluetoothd.symbols.ACL_reception_handler);
        const acl_reception_handler = new NativeFunction(
            acl_rec_addr, "void", ["int64", "int64", "pointer"]);

        super("bluetoothd", acl_reception_handler, acl_rec_addr);
        this.payload_buffer = Memory.alloc(0x100);
    }
```

```
const fuzzer = require("../../harness/fuzzer.js");
const bluetoothd = require("./bluetoothd.js");

const base = Module.getBaseAddress("bluetoothd");

class ACLFuzzer extends fuzzer.Fuzzer {
    constructor() {

        const acl_rec_addr = base.add(bluetoothd.symbols.ACL_reception_handler);
        const acl_reception_handler = new NativeFunction(
            acl_rec_addr, "void", ["int64", "int64", "pointer"]);

        super("bluetoothd", acl_reception_handler, acl_rec_addr);
        this.payload_buffer = Memory.alloc(0x100);
    }
```

```
  prepare() {
    this.handle = bluetoothd.setupFakeACLConnection();

    // send a MP ping to increase l2cap mtu
    Memory.writeByteArray(this.payload_buffer,
      [0x03, 0x00, 0x30, 0x00, 0xF0, 0x00, 0x00]);
    this.target_function(0x11 + (0x20<<8), 7, ptr(this.payload_buffer));
  }

  fuzz(payload, len) {
    const handle = 0x11 + (0x20 << 8);
    this.target_function(handle, parseInt(len), payload);
  }

}

const f = new ACLFuzzer();
exports.fuzzer = f;
```

```
    prepare() {
        this.handle = bluetoothd.setupFakeACLConnection();

        // send a MP ping to increase l2cap mtu
        Memory.writeByteArray(this.payload_buffer,
            [0x03, 0x00, 0x30, 0x00, 0xF0, 0x00, 0x00]);
        this.target_function(0x11 + (0x20<<8), 7, ptr(this.payload_buffer));
    }

    fuzz(payload, len) {
        const handle = 0x11 + (0x20 << 8);
        this.target_function(handle, parseInt(len), payload);
    }

}

const f = new ACLFuzzer();
exports.fuzzer = f;
```

```
    prepare() {
        this.handle = bluetoothd.setupFakeACLConnection();

        // send a MP ping to increase l2cap mtu
        Memory.writeByteArray(this.payload_buffer,
            [0x03, 0x00, 0x30, 0x00, 0xF0, 0x00, 0x00]);
        this.target_function(0x11 + (0x20<<8), 7, ptr(this.payload_buffer));
    }

    fuzz(payload, len) {
        const handle = 0x11 + (0x20 << 8);
        this.target_function(handle, parseInt(len), payload);
    }

}

const f = new ACLFuzzer();
exports.fuzzer = f;
```

```
    prepare() {
        this.handle = bluetoothd.setupFakeACLConnection();

        // send a MP ping to increase l2cap mtu
        Memory.writeByteArray(this.payload_buffer,
            [0x03, 0x00, 0x30, 0x00, 0xF0, 0x00, 0x00]);
        this.target_function(0x11 + (0x20<<8), 7, ptr(this.payload_buffer));
    }

    fuzz(payload, len) {
        const handle = 0x11 + (0x20 << 8);
        this.target_function(handle, parseInt(len), payload);
    }

}

const f = new ACLFuzzer();
exports.fuzzer = f;
```

Compile harness script (frida agent):

```
frida-compile acl-fuzzer.js -o harness.js
```

Run AFL++ with fpicker against bluetoothd on iOS:

```
afl-fuzz -i ./in -o ./out -- ./fpicker -m afl -e attach -p bluetoothd -f harness.js
```

ssh

```
iphone:/usr/bin/afl $ ./afl-fuzz -m none -i ./acl_classic/corpus -o ./acl_classi>
```

zsh

```
❯ idevicesyslog | grep bluetoothd
```

```
iphone:/usr/bin/afl $ ./afl-fuzz -m none -i ./acl_classic/corpus -o ./acl_classi>
```

```
❯ idevicesyslog | grep bluetoothd
```

https://github.com/ttdennis/fpicker

E-Mail: dennis@bluetooth.lol
Twitter: @ttdennis

ERNW
providing security.