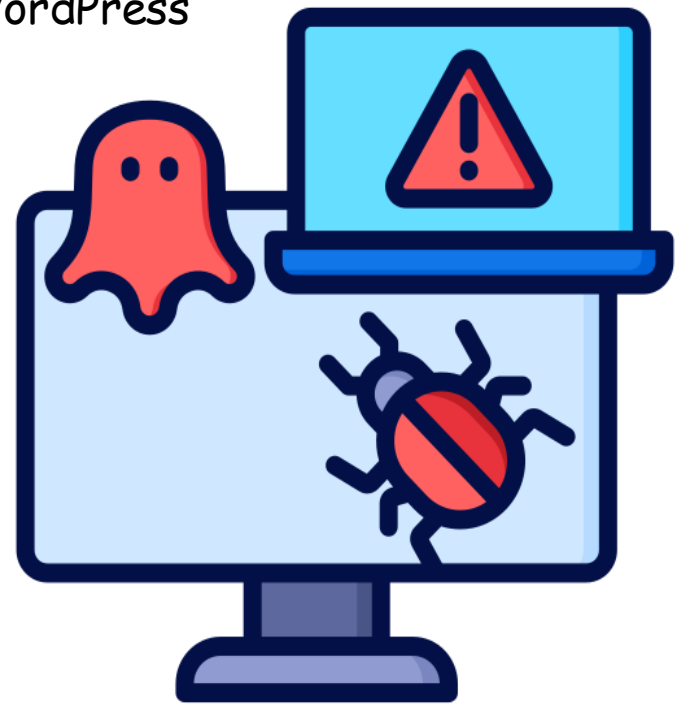
Intro to Vulnerability Research

https://github.com/elbee-cyber

```
diff --git a/net/vmw_vsock/af_vsock.c b/net/vmw_vsock/af_vsock.c
index 4bd825fa77e411..8ec128f450dfc7 100644
--- a/net/vmw_vsock/af_vsock.c
+++ b/net/vmw_vsock/af_vsock.c
@@ -330,7 +330,10 @@ EXPORT_SYMBOL_GPL(vsock_find_connected_socket);

 void vsock_remove_sock(struct vsock_sock *vsk)
 {
-       vsock_remove_bound(vsk);
+       /* Transport reassignment must not remove the binding. */
+       if (sock_flag(sk_vsock(vsk), SOCK_DEAD))
+               vsock_remove_bound(vsk);
+
        vsock_remove_connected(vsk);
 }
 EXPORT_SYMBOL_GPL(vsock_remove_sock);
@@ -815,12 +818,13 @@ static void __vsock_release(struct sock *sk, int lev
         */
        lock_sock_nested(sk, level);

+       sock_orphan(sk);
+
        if (vsk->transport)
                vsk->transport->release(vsk);
        else if (sock_type_connectible(sk->sk_type))
                vsock_remove_sock(vsk);

-       sock_orphan(sk);
        sk->sk_shutdown = SHUTDOWN_MASK;

        skb_queue_purge(&sk->sk_receive_queue);
```

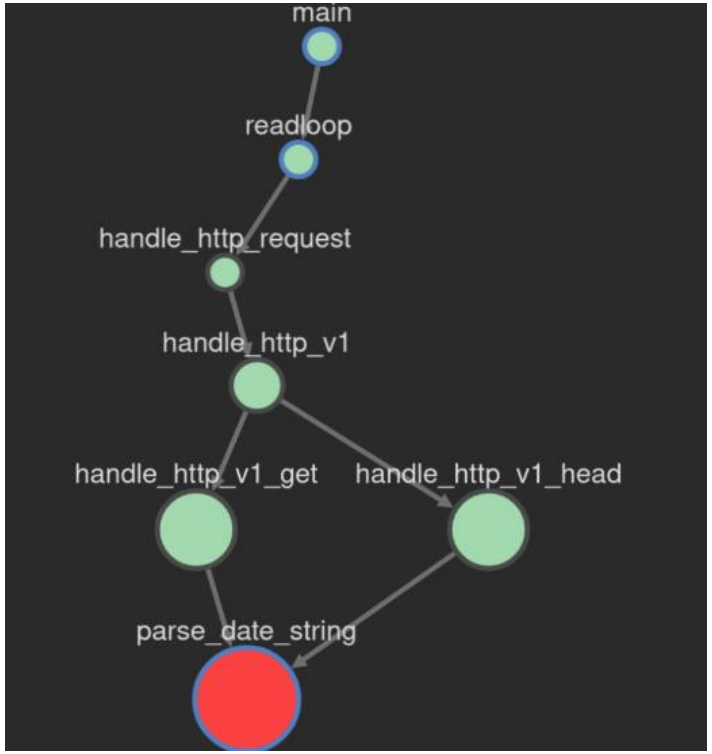
What is vulnerability research? 🎯

- The act of finding zero-days, not necessarily exploiting them (exploit development)
- One of the most important aspects is choosing a target
  - A target should be specific **software, hardware, or protocols** not tied to a specific environment.
  - Examples: Router firmware, the Linux kernel, an NFC stack, FOSS, WordPress
- Factors that can help you choose a good target
  - Does it satisfy the above?
  - How complex is it?
  - How familiar are you with the technologies you might encounter?
  - What types of bug classes could exist?
  - How easy is it to acquire?
  - How easy is it to debug?
  - Who is the vendor?
- https://nostarch.com/zero-day



Always practice responsible disclosure!! (90 days)

Attack Surfaces and Data-flows



Data-flows: How you get from a source to a sink.

How do we reach the interesting code?

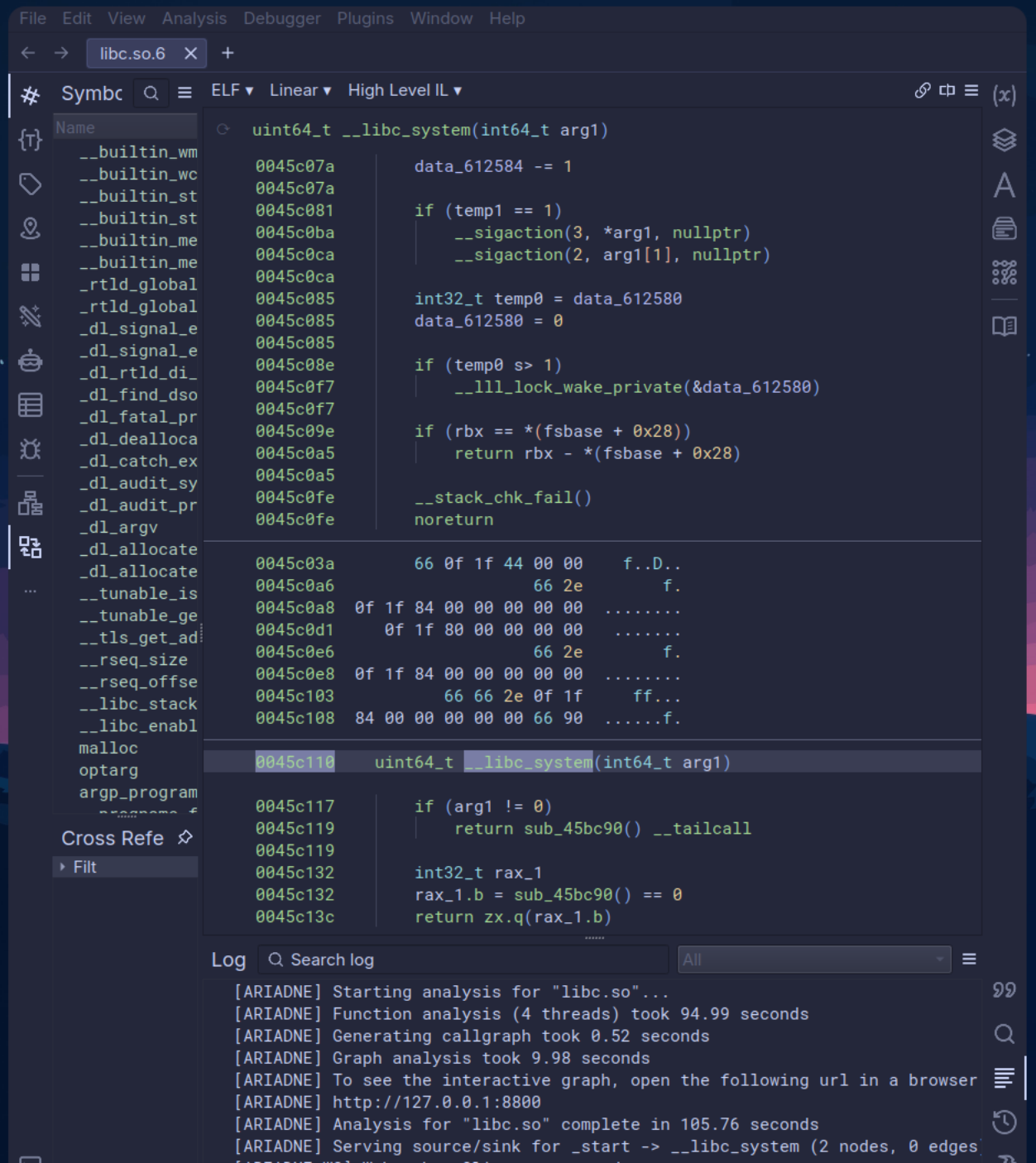
Harder on complex targets

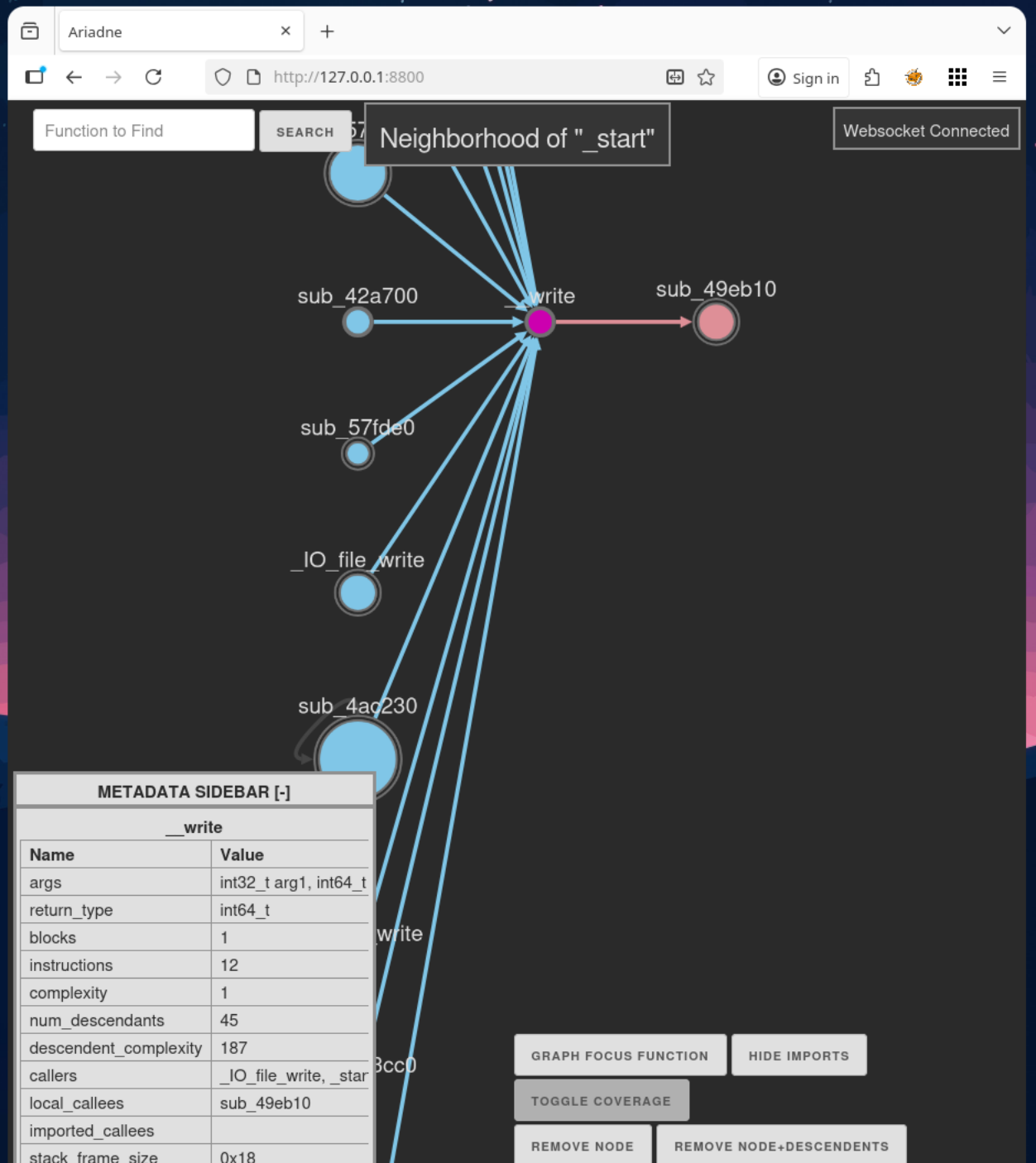
- Reaching a SQL query statement on WP is easy, but...
- How can I send a custom BT packet that hits a sink?
  - Complex state machines
  - Driver restrictions on the protocol (likely requires external hardware)

Mapping the data-flows in an attack surface will help!

For the BT example, if my target is SDP:

- Create a harness that speaks the target specs
- Confirm you can send/recv information
- This is a whole process on its own!


File Edit View Analysis Debugger Plugins Window Help
libc.so.6
Symbc
ELF ▾ Linear ▾ High Level IL ▾
uint64_t __libc_system(int64_t arg1)
0045c07a data_612584 -= 1
0045c07a
0045c081 if (temp1 == 1)
0045c0ba __sigaction(3, *arg1, nullptr)
0045c0ca __sigaction(2, arg1[1], nullptr)
0045c0ca
0045c085 int32_t temp0 = data_612580
0045c085 data_612580 = 0
0045c085
0045c08e if (temp0 s> 1)
0045c0f7 __lll_lock_wake_private(&data_612580)
0045c0f7
0045c09e if (rbx == *(fsbase + 0x28))
0045c0a5 return rbx - *(fsbase + 0x28)
0045c0a5
0045c0fe __stack_chk_fail()
0045c0fe noreturn
0045c03a 66 0f 1f 44 00 00 f..D..
0045c0a6 66 2e f.
0045c0a8 0f 1f 84 00 00 00 00 00 ....
0045c0d1 0f 1f 80 00 00 00 00 ....
0045c0e6 66 2e f.
0045c0e8 0f 1f 84 00 00 00 00 00 ....
0045c103 66 66 2e 0f 1f ff...
0045c108 84 00 00 00 00 00 66 90 ......f.
0045c110 uint64_t __libc_system(int64_t arg1)
0045c117 if (arg1 != 0)
0045c119 return sub_45bc90() __tailcall
0045c119
0045c132 int32_t rax_1
0045c132 rax_1.b = sub_45bc90() == 0
0045c13c return zx.q(rax_1.b)
Cross Refe
Filt
Log
Search log
All
[ARIADNE] Starting analysis for "libc.so"...
[ARIADNE] Function analysis (4 threads) took 94.99 seconds
[ARIADNE] Generating callgraph took 0.52 seconds
[ARIADNE] Graph analysis took 9.98 seconds
[ARIADNE] To see the interactive graph, open the following url in a browser
[ARIADNE] http://127.0.0.1:8800
[ARIADNE] Analysis for "libc.so" complete in 105.76 seconds
[ARIADNE] Serving source/sink for _start -> __libc_system (2 nodes, 0 edges

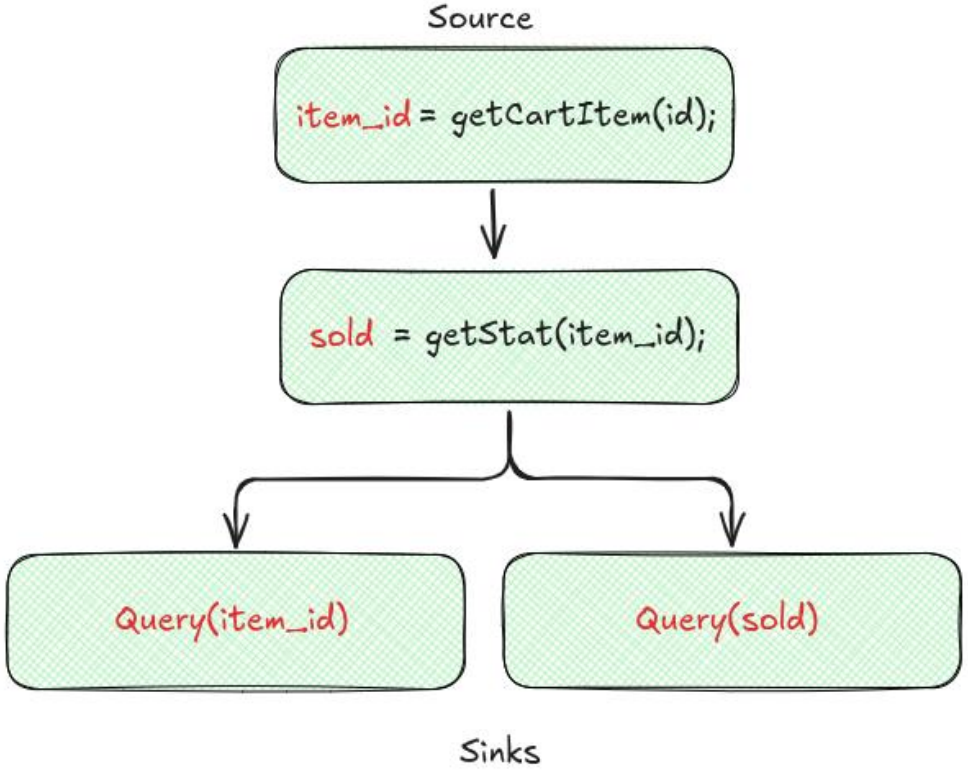

Ariadne
http://127.0.0.1:8800
Sign in
Function to Find
SEARCH
Neighborhood of "_start"
Websocket Connected
sub_42a700
write
sub_49eb10
sub_57fde0
_IO_file_write
sub_4ac230
write
METADATA SIDEBAR [-]
__write
Name
Value
args
int32_t arg1, int64_t
return_type
int64_t
blocks
1
instructions
12
complexity
1
num_descendants
45
descendent_complexity
187
callers
_IO_file_write, _star
local_callees
sub_49eb10
imported_callees
stack_frame_size
0x18
GRAPH FOCUS FUNCTION
HIDE IMPORTS
TOGGLE COVERAGE
REMOVE NODE
REMOVE NODE+DESCENDENTS

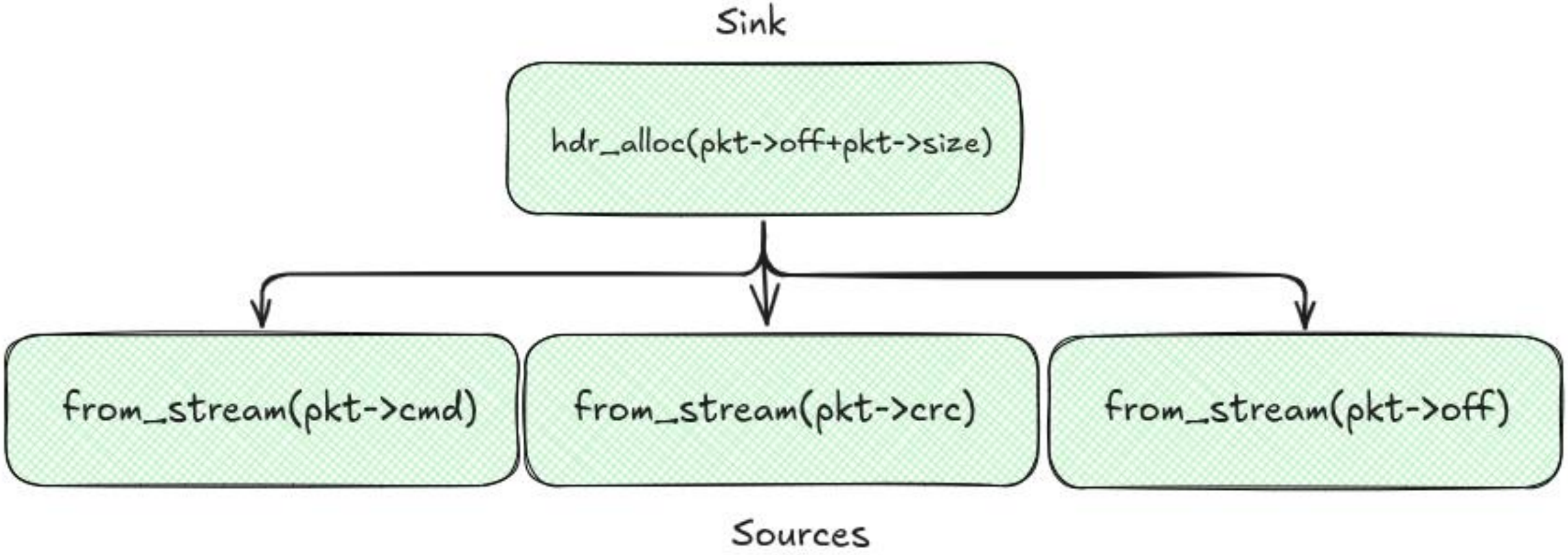
Analysis with source

Codeql and semgrep

Taint Analysis 🔗

- Source to sink and sink to source (both being lines of code).
- Finds flows from sources to sinks
  - "Tainted" variables are also included
- The goal
  - Attacker-controlled source categories
  - Interesting sink categories
- Granularity of source





Taint Analysis Workflow

- Source to sink will follow uninteresting vars (path explosion), do sink to source instead!
- Note attacker-controlled sources and interesting sinks.
- Filter out uninteresting patterns/results as you go.
- Use this analysis to find data-flows and vulnerable patterns.

Patch Diffing and Variant Analysis 

- Searching for weak code patterns based on patch history
- Patch diffing lets you learn about a patch
  - Which contributor is responsible
  - A possible source and sink pattern
    - Source: Non-zero variable with later attacker-controlled arithmetic
    - Sink: allocs, memcpys, etc
  - What structures it interacts with and at what layer
  - Maybe highlights where they don't follow specs
- Variant analysis is creating heuristics and searching for a pattern
- Why: Code reuse = bug reuse, human habits

CVE-2022-20410

```
uint16_t min_len = 0;
min_len += 8;
// "p_attrs[i].name.str_len" can take any value form 0 to UINT16_MAX
BE_STREAM_TO_UINT16(attr_entry->name.str_len, p);
// possible integer overflow, "min_len" may less than expected
min_len += attr_entry->name.str_len;
// bypass this length check
if (pkt_len < min_len) goto browse_length_error;
attr_entry->name.p_str = (uint8_t*) osi_malloc(attr_entry->name.str_len * sizeof(uint8_t));
// oob read due to "attr_entry->name.str_len" could be larger than real size of input data
BE_STREAM_TO_ARRAY(p, attr_entry->name.p_str, attr_entry->name.str_len);
```

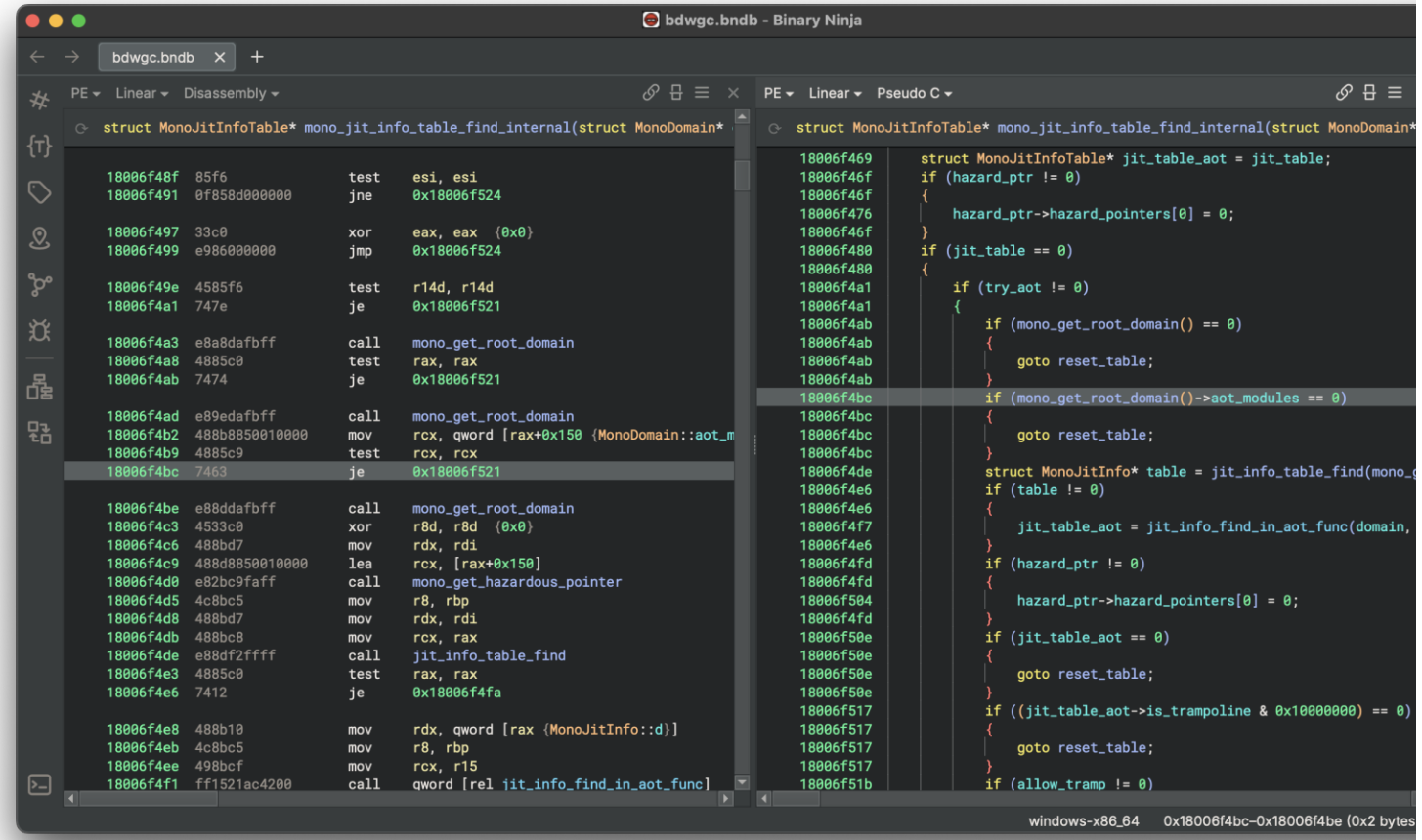
"Bypass length check due to integer overflow, leading to heap buffer overflow read"

```
// CodeQL code
override predicate isSink(DataFlow::Node sink) {
    exists(AssignAddExpr aae |
        aae.getRValue() = sink.asExpr() and
        aae.getLValue().getType().getSize() <=
        aae.getRValue().getExplicitlyConverted().getType().getSize()
    ) or
    exists(AddExpr ae |
    ) or
    exists(MulExpr me |
    ) or
    // ...
}
```

Discovered with this CodeQL query

https://www.blackhat.com/eu-22/briefings/schedule/#deep-into-android-bluetooth-bug-hunting-new-attack-surfaces-and-weak-code-patterns-28496

Binary Analysis



Binary Analysis

- No source!
- Look at binary without running it AND look at a program while it runs.
- Sometimes stuff just doesn't work:
  - A module that expects other firmware components
  - An obscure or different instruction set architecture
  - Modules that rely on drivers or hardware
- In these scenarios, emulation can be a part of your debug environment!
  - Router httpd binary -> Qemu
  - Windows driver ioctl -> Qiling
  - Decryption function from firmware blob -> Unicorn
- Other techniques for specific scenarios
  - Symbolic execution: Constraint-based execution with symbolic inputs to explore paths.

Fuzzing

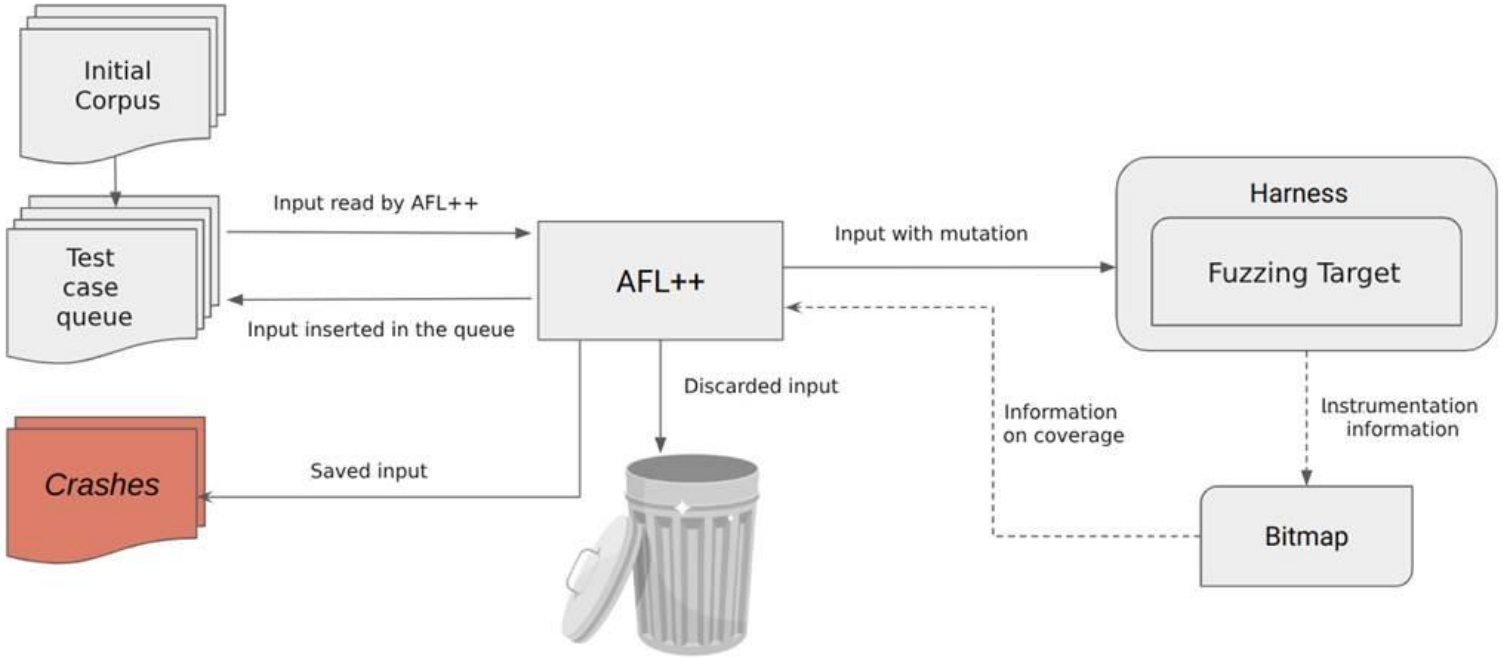
Throwing hundreds, thousands, millions, of mutated inputs at a target, with an emphasis on the feedback method that "guides" input generation.

Fuzzer go brrr

Coverage-guided: "Future mutations are guided by code coverage from previous inputs."

- Basically generation heuristic based on new lines hit.
- Combining this with a grammar can be very powerful!

- Very effective because it reaches deep code and is based on "new behavior".
- AFL++
  - Instrumented, coverage-guided, mutation fuzzer
  - Has lots of forks and built-in modes!
  - LibAFL, build your own!
- Can be combined with sanitizers
  - Trigger because of a thing
  - Abort and print debug info about that thing
- https://aflplus.plus/libafl-book
- Design your own fuzzers with custom techniques!
- Can easily be a whole class on its own!



https://www.sidechannel.blog/en/afl-and-an-introduction-to-feedback-based-fuzzing

Questions?