Changes in the Guard

Detecting Authority-Exposing Patches with Differential Guarded Flow Paths

Nathan Dautenhahn Dartmouth College and Serenitix
Sergey Bratus Dartmouth College

SURE ’26 · ACM CCS workshop · The Hague

Paper · slides · demo: serenitix.io/papers/sure26



PROBLEM

Software keeps changing, and some changes remove a check



Fixes, features, refactors and dependency bumps all arrive as patches, and coding agents now write many of them.

Reviewers check what each patch is meant to do. A refactor is not expected to change what an attacker can reach.

Sometimes it does.

Illustrative commit messages

Removed guards have had severe consequences

Heartbleed

CVE-2014-0160 · OpenSSL 1.0.1–1.0.1f

MISSING

THE GUARD

No check that the heartbeat length fits the record that arrived

CONSEQUENCE

Up to 64 KB of process memory returned per request

regreSSHion

CVE-2024-6387 · OpenSSH 8.5p1–9.7p1

REMOVED

THE GUARD

A logging refactor deleted the #ifdef that kept unsafe logging out of a signal handler

CONSEQUENCE

Potential remote code execution as root (glibc Linux)

goto fail

CVE-2014-1266 · Apple Secure Transport

BYPASSED

THE GUARD

An extra goto skipped the signature check on the server key exchange

CONSEQUENCE

A privileged network attacker may capture or modify TLS data

Dirty Pipe

CVE-2022-0847 · Linux 5.8 and later

UNSET, THEN TRUSTED

THE GUARD

Pipe-buffer flags never initialized; a later change made them matter

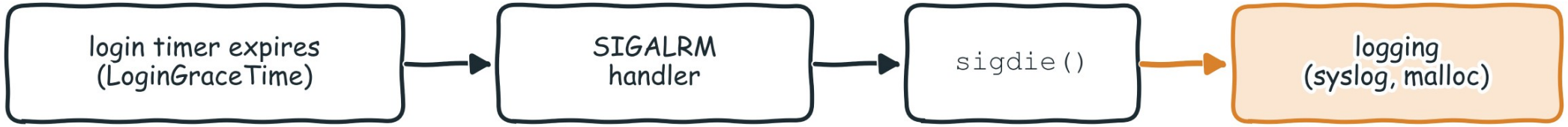
CONSEQUENCE

Unprivileged overwrite of read-only files: local privilege escalation

Historical incidents · sources in speaker notes · not AuthDiff results

OpenSSH: a logging refactor deleted a guard

What runs when a client never finishes logging in



not safe inside a signal handler

before: log.c, since the 2006 fix

```
sigdie(const char *fmt,...)
{
#ifdef DO_LOG_SAFE_IN_SIGHAND
    … do_log(SYSLOG_LEVEL_FATAL, …);
#endif
    _exit(1);
}
```

after commit 752250c (logging refactor, 2020)

```
sshsigdie(const char *file, …)
{
#ifdef DO_LOG_SAFE_IN_SIGHAND
    sshlogv(…, SYSLOG_LEVEL_FATAL, …);
#endif                                  deleted
    _exit(1);
}
```

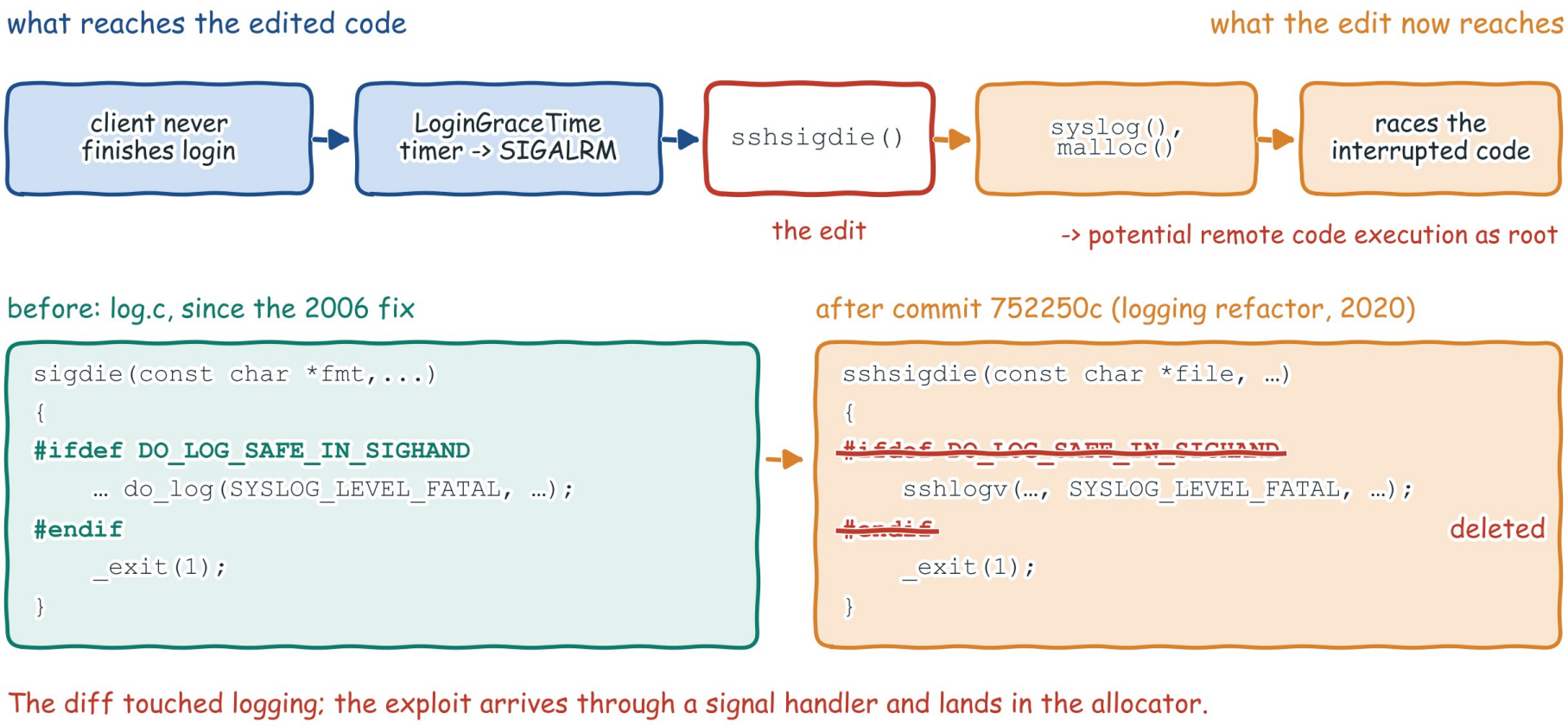
Affected builds log inside the handler again. On glibc Linux: potential remote code execution as root.

Historical incident · CVE-2024-6387 · Portable OpenSSH 8.5p1–9.7p1 · regression of the 2006 fix (CVE-2006-5051) · not an AuthDiff result

After a patch: what can an input reach that it couldn’t before?

And which check used to stop it?

OpenSSH: edited in logging, exploited in a signal handler



Non-local in both directions; deciding such effects over all executions is undecidable in general.

Historical incident · CVE-2024-6387 · OpenSSH 8.5p1–9.7p1 · code verbatim from commit 752250c (abridged) · OpenSSH 9.8 notes, Qualys advisory · not an AuthDiff result

Hypothesis: a bug is an operation reached in a state it is not authorized for

$$\exists \pi : \mathrm{Reach}(\pi, op) \land \neg \mathrm{Authorized}(op, \sigma\pi, \pi)$$

op the operation $\sigma\pi$ the values and state at the time of use π the path that produced them

The values at the time of use decide whether the operation is authorized. The guards along the path condition those values: they are the sanitization that refines the operation into what was intended.

OPERATION + SPEC

What is permitted?

VALUES + GUARDS ON THE PATH

What states can reach it?

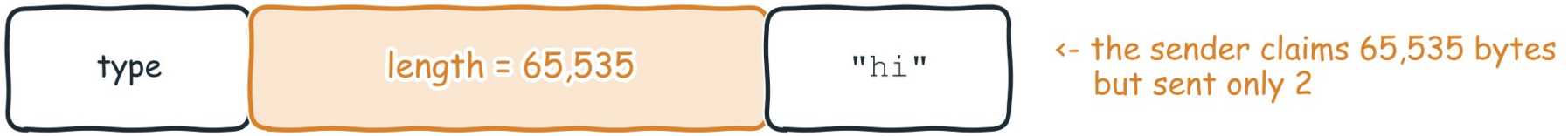
CHECKER

Can a reachable state violate it?

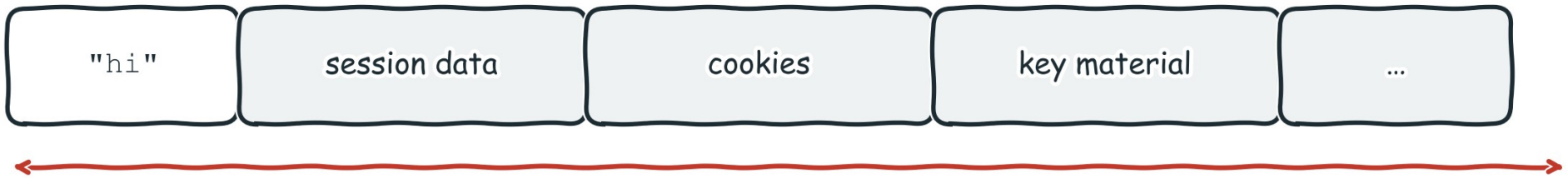
A local question at the operation, answered with facts gathered along the path. Checking it over all executions is undecidable in general; we make it a structural query.

Our running example: the heartbeat length check

Heartbeat request (from the network)



Server memory, next to the request



memcpy(bp, pl, payload) copies 65,535 bytes back to the sender

No check that the length fits the record that actually arrived.

One way to find what reaches an operation: value flow

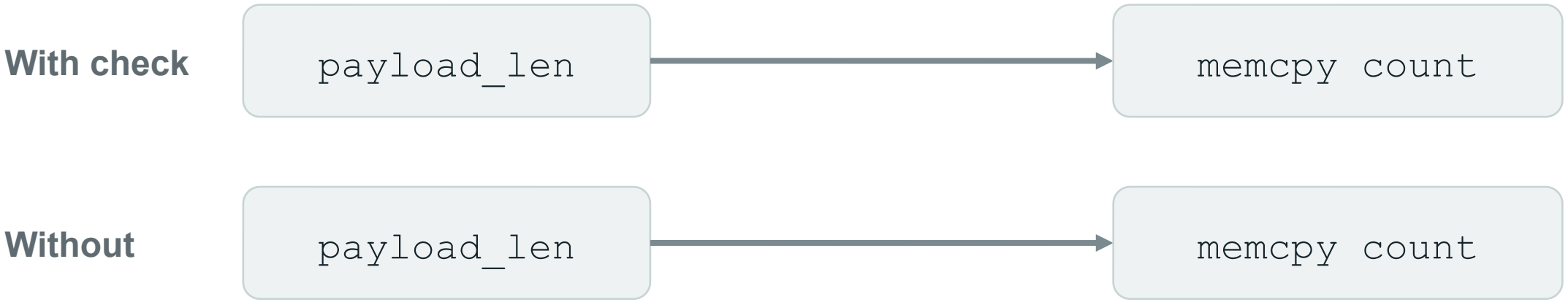
Flow



`payload_len` the length field read from the packet **`memcpy count`** how many bytes are copied into the reply

Value-flow (data-flow, taint) analysis follows each input to the operations that use it, and answers: **does data from a source reach a sensitive use?** Here: yes, the packet's length reaches the copy.

…but value flow drops the check: same answer before and after

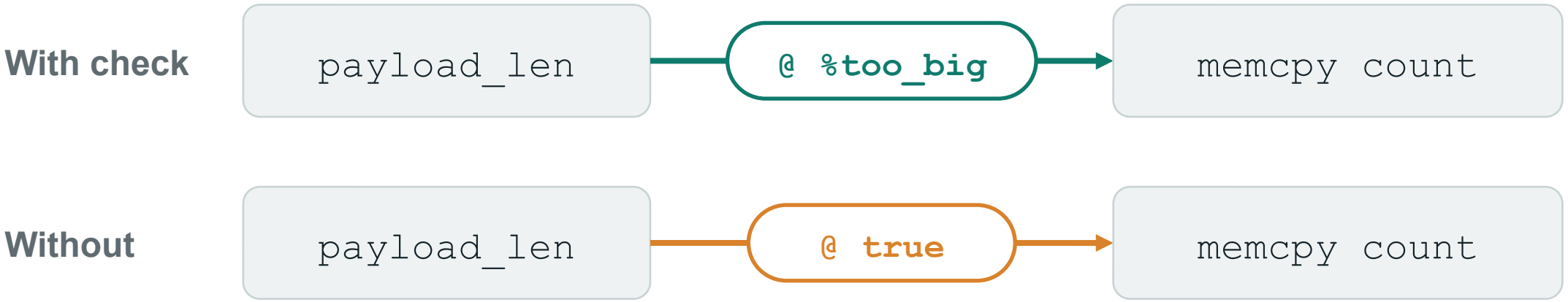


The check is not in the flow. In taint tools it sits beside the query, as a sanitizer configured for each kind of bug, so the result cannot say which check a patch removed.

On our worked example, Infer (with a taint model) and the Clang Static Analyzer reported nothing on either version.

Paper §1 (the check sits beside the query) and §8 Q2 (direct runs) · CodeQL supports custom path queries; we have not shown the comparison is inexpressible there

Add the guard to the representation: Guarded Flow Paths



Now one graph holds both halves of the insight: the use and the guard on every path into it. That is what we need to judge the operation's safety, and the two versions now differ exactly where the check was removed.

%too_big: result of comparing the length with the bytes received · @ g: flow happens only when g holds · @ true: always

Recorded by AuthDiff on the controlled heartbeat model (heartbleed_toy_safe.ll / heartbleed_toy.ll; the model's copy is call(@write))

Guards are values, so a removed check is a label change

SYNTAX (paper Fig. 2, trimmed)

```
v  ::= root(id) | derived(id, op)
op ::= load | cast:* | call(f) | icmp | …
g  ::= true | v            -- guard
e  ::= v --[g]--> op       -- use
     | op --> v            -- def

def x <- op(y1..yn) @ g
x is computed by op from y1..yn,
only when g holds
```

EXAMPLE: the fixed heartbeat check, normalized (recorded)

```
def %payload_len <- cast:zext(%payload_len_raw) @ true
def %req_len_adj <- sub(const:2, %req_len) @ true
def %too_big <- icmp(%payload_len, %req_len_adj) @ true
def %sent <- call(@write, const:1, %payload_len,
                  %req_data) @ %too_big
```

1 the check is an ordinary value: `%too_big` is the result of a compare

2 the copy is defined `@ %too_big`: it happens only when the check holds

3 delete the check and that label becomes `@ true`

guard(π) = g1 ∧ … ∧ gn a path's labels, AND-ed **unguarded(s,t)** some path has guard = true **no fixpoint needed**

Syntax: paper Fig. 2 and §4 · example: duc-proto normalize tests/heartbleed_toy_safe.ll (selected lines; the model's copy is call(@write))

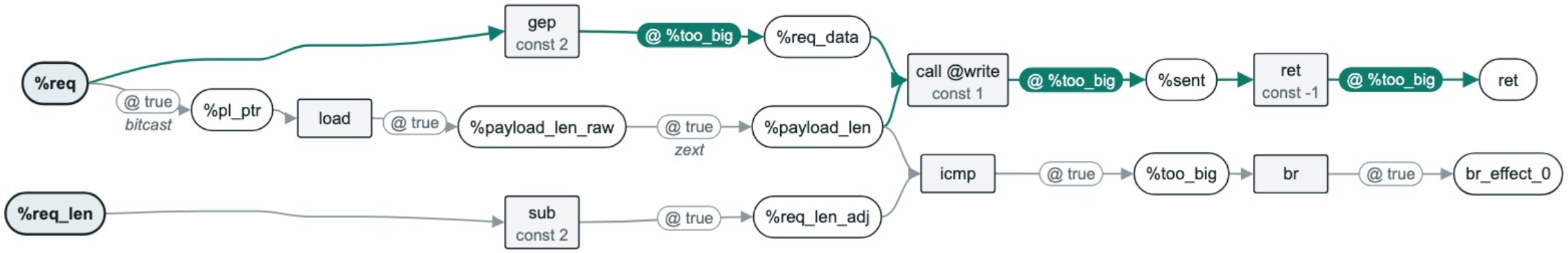
From source to guarded flow: what the tool records for the heartbeat check

SOURCE (C)

```
uint16_t payload_len = my_ntohs(req);
if (req_len < 2 ||
    (size_t)payload_len > req_len - 2)
    return;
memcpy(resp, req + 2, payload_len);
```

LIFTED BY AuthDiff (duc-proto normalize, selected lines)

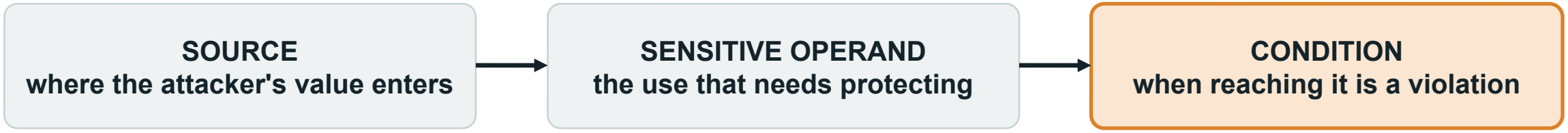
```
def %payload_len <- cast:zext(%payload_len_raw)          @ true
def %req_len_adj <- sub(const:2, %req_len)               @ true
def %too_big     <- icmp(%payload_len, %req_len_adj)     @ true
def %sent        <- call(@write, const:1, %payload_len,
                         %req_data)              @ %too_big
```



The check becomes a compare whose result, %too_big, labels the copy's use of the length (teal). Delete the check and that label becomes @ true.

heartbleed_toy_safe.c (fixed heartbeat model) · recorded normalize output · graph drawn from that output by the DUC viewer · the model performs the copy with write

With guards in the graph, a bug class is a short spec: source, sensitive operand, condition



missing_bounds_check.yaml (excerpt)

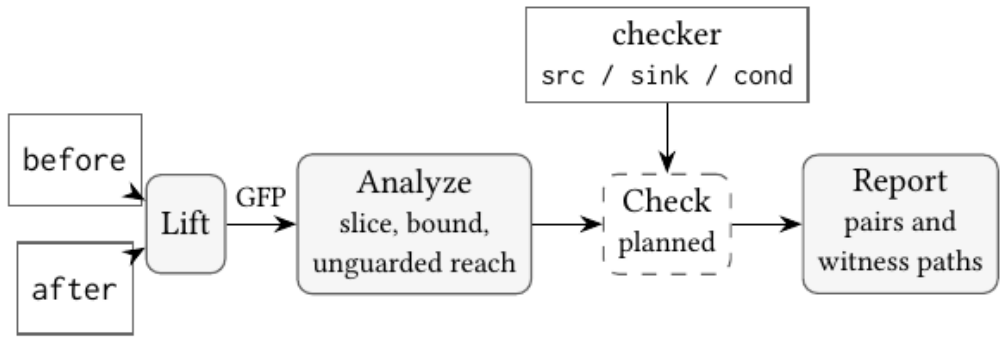
```
sources:
  - kind: load        # packet length
sinks:
  - kind: call
    callee_regex: "^(@memcpy|@memmove|...)$"
    index: 2          # the byte count
```

```
sanitizers:
  - kind: node
    op_prefix: "icmp"    # a comparison
violation:
  path: unconditioned
  sanitizer_absent:
    icmp_for_source: true
```

Condition here: the byte count is reached on an unguarded path with no comparison of the length. The graph does the hard part; the same query serves every class.

Written specification · duc-v2/experiments/specs/missing_bounds_check.yaml (abbreviated; memcpy regex shortened) · automatic evaluation of specs is planned

Subtract versions: each newly unguarded pair comes with a witness path

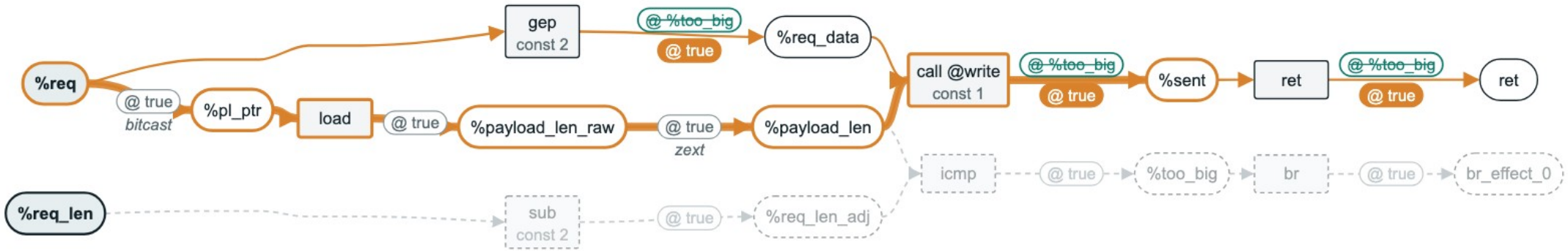


AuthDiff pipeline (paper Fig. 4); the Check stage is planned

U = (source, endpoint) pairs with an unguarded path, per version

Δ = U_after \ U_before

Witness path: the concrete chain of edges, every one @ true, showing how the input now reaches the use. The vanished guard sits on the same path in the before graph.



NEW capability paths (introduced in after) — newly unguarded source → endpoint pairs

```
+ %req -> %req_data
    %req --[true]--> %req_data
+ %req -> %sent
    %req --[true]--> %req_data --[true]--> %sent
    %req --[true]--> %pl_ptr --[true]--> %payload_len_raw --[true]--> %payload_len --[true]--> %sent
+ %req -> ret
    %req --[true]--> %req_data --[true]--> %sent --[true]--> ret
    %req --[true]--> %pl_ptr --[true]--> %payload_len_raw --[true]--> %payload_len --[true]--> %sent --[true]--> ret
```

before: 9 def lines, **3** guarded
after: 6 def lines, **0** guarded
3 def guards changed to @ true
3 new pairs · **5** witness paths
highlighted: witness 2.2

IT WORKS

Live demo: the tool answers our question on two patches

`$ duc-proto diff before.ll after.ll` every input that newly reaches something unguarded

PATCH	WHAT CHANGED	GUARD BEFORE	NEW UNGUARDED REACH (AFTER)
Document fetch	− if (!acl_check(user_id, doc_id)) return NULL;	`@ %is_ok`	`%doc_id -> %doc`
Heartbeat (model)	− if (… payload_len > req_len - 2) return;	`@ %too_big`	`%req -> %sent`

What to watch for, per patch: (1) the deleted line, (2) the guard it removed in the before graph, (3) what the input now reaches, with its witness path.

How to read a report: `+ a -> b` means input a now reaches b on a path where every edge is `[true]`. The guard it used to need is the `@ …` label in the before version.

Recorded DUC v2 fixtures OW-03 and DC-03 · lift + diff only; no checker evaluated · each run < 0.1 s, offline · fallback output on backup slides

RESULTS

Results: 17 bug classes, one specification shape

Memory safety

null deref · use after free · double free · out-of-bounds · uninitialized read

Integers & pointers

integer overflow · signedness · untrusted pointer

Injection & input handling

SQL · command · format string · path traversal · SSRF · deserialization

Authorization & secrets

object access (IDOR) · auth bypass · hardcoded credentials

Use after free (CWE-416)

```
sources:
  pointer params, allocations
sinks:
  call(@free, ptr)
  load(ptr) | store(…, ptr)
violation:
  both reached unconditioned,
  no store(null, ptr) between
```

SQL injection (CWE-89)

```
sources:
  pointer params
sinks:
  sqlite3_exec | PQexec | …
sanitizers:
  db_escape | sqlite3_bind | …
violation:
  unconditioned, no sanitizer
```

Insecure object access (CWE-639)

```
sources:
  params named *id*
sinks:
  db_fetch_doc | db_get | …
sanitizers:
  acl_check | is_owner | …
violation:
  unconditioned, no sanitizer
```

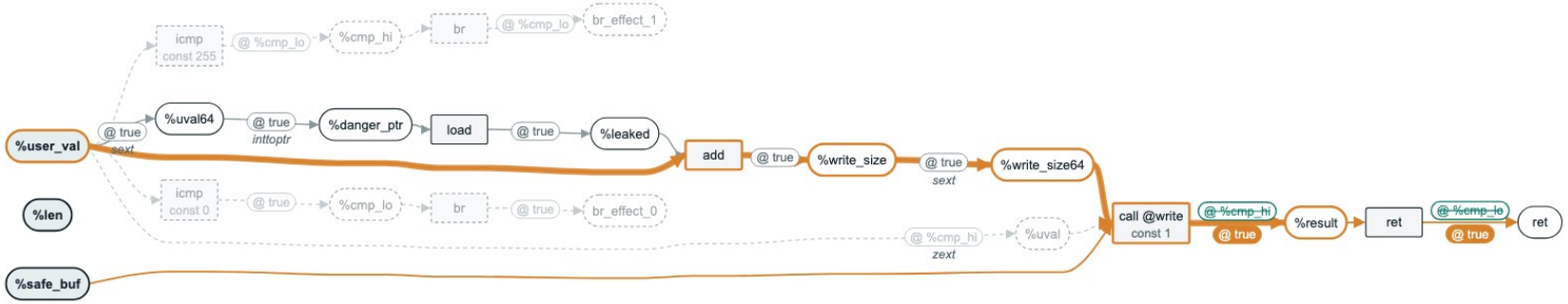
Each names a source, the sensitive operand, and the condition; the guarded-flow query and the comparison are the same for all 17.

Classes from paper Tables 1–2 · specs paraphrased from duc-v2/experiments/specs/{use_after_free,sql_injection,broken_access_control}.yaml · automatic evaluation planned

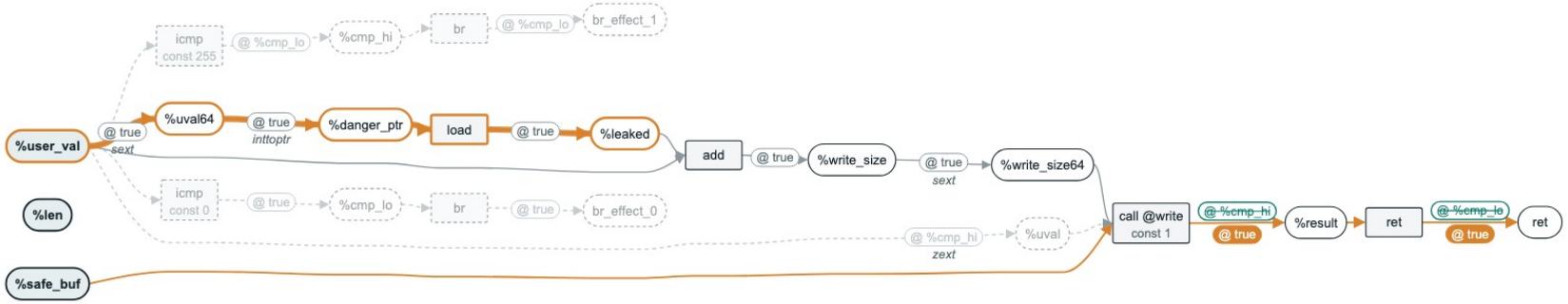
RESULTS

One removed guard, two kinds of new reach: one diff finds both

1 Write length `user_val → write_size → call @write` was @ %cmp_hi, now @ true



2 Address `user_val → uval64 → danger_ptr → load → leaked` new: the integer is now a pointer



Usually each bug class needs its own checker pass. One comparison reported both. 9 new unguarded pairs, 13 witness paths.

DUC viewer over recorded duc-proto diff, DC-02 (paper §6.1) · thick amber: the witness path for each kind · struck teal → amber pill: guard changed @ %cmp_hi → @ true · graph paths, not nine bugs

RESULTS

How the 17 classes compare with five existing tools

ID	Bug class (CWE)	GFP	Infer	CodeQL	CSA	Semgrep	Coverity
DC-01	Null dereference (476)	R	●	●	●	–	●
DC-02	Untrusted pointer (822/843)	R	○	●	–	–	○
DC-03	Out-of-bounds read/write (125/787)	R	–	●	–	–	●
DC-04	Use after free (416)	R	●	●	●	–	●
DC-05	Integer overflow (190/122)	R	○	●	–	–	●
DC-06	Format string (134)	R	–	●	–	–	●
DC-07	Double free (415)	R	●	●	●	–	●
DC-08	Signedness (195/196)	R	○	○	–	–	●
DC-09	Path traversal (22)	R	–	●	–	–	●
DC-10	Uninitialized read (457)	R	○	●	●	–	●
OW-01	SQL injection (89)	A	–	●	–	○	●
OW-02	Command injection (78)	A	–	●	–	○	●
OW-03	Insecure object access (639)	A	–	○	–	–	–
OW-04	Hardcoded credentials (798)	A	–	●	–	●	●
OW-05	SSRF (918)	A	–	●	–	○	○
OW-06	Insecure deserialization (502)	A	–	○	–	–	○
OW-07	Authentication bypass (306)	A	–	○	–	○	–

What we did

Wrote a checker specification for each of 17 classes, all over the same guarded-flow query.

What it adds

Other tools match a bug shape in one version, class by class. GFP reports, for any class, the pair that lost its guard across a patch, with a witness path.

Limits

Minimal IR models of each class plus one real OpenSSL function; specs written, automatic evaluation planned.

GFP: **R** recorded graph result **A** authors' assessment

● native ○ partial/configured – not supported. CSA, Semgrep: direct runs; Infer, CodeQL, Coverity: documented checkers

Paper Tables 1–2, transcribed (CWE-class and OWASP-class coverage) · CSA: Clang Static Analyzer

Where it stands

Works today

- Lifts LLVM IR into Guarded Flow Paths
- Reports newly unguarded paths, with witnesses
- One function at a time
- Small examples: 10 CWE classes run, plus a real OpenSSL function

Not yet

- Is the guard the right guard?
- Flows across function calls (OpenSSH needs this)
- Evaluating written checker specs automatically
- Real patch histories

Checks are authority. Review the guards a patch changes.

1. **Guarded Flow Paths**
   value flow with the guard on every edge; a removed check is a label change
2. **Specify and check**
   a property is a source, a sensitive operand, and a condition over the path
3. **Checkers over unguarded flow paths**
   17 bug classes, one query
4. **Differential authority analysis**
   AuthDiff reports what a patch newly exposes, each with a witness path

What can the input reach now, and which check used to stop it?

Thank you · serenitix.io/papers/sure26 · Nathan Dautenhahn · Sergey Bratus

Backup

OpenSSH: edited in logging, exploited in a signal handler



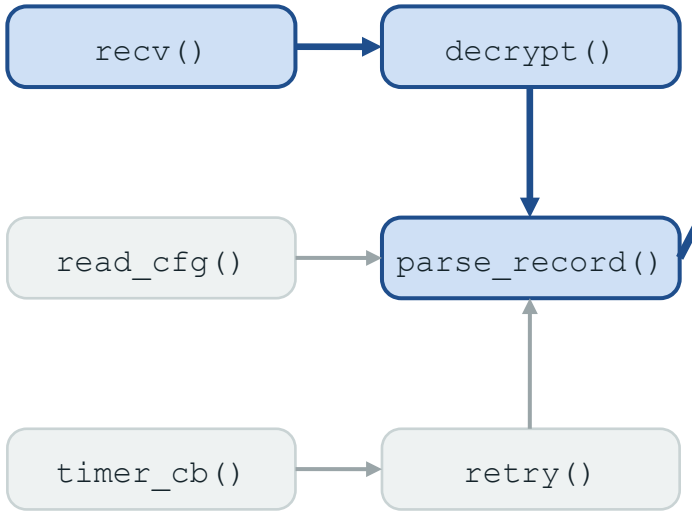
log.c · logging refactor 752250c removed the guard (struck)

The diff touched logging. The exploit lives in a signal handler that reaches it. Deciding such effects over all executions is undecidable in general.

Historical incident · CVE-2024-6387 · OpenSSH 8.5p1–9.7p1 · sources: commit 752250c, OpenSSH 9.8 notes, Qualys advisory · not an AuthDiff result (compile-time guard, interprocedural)

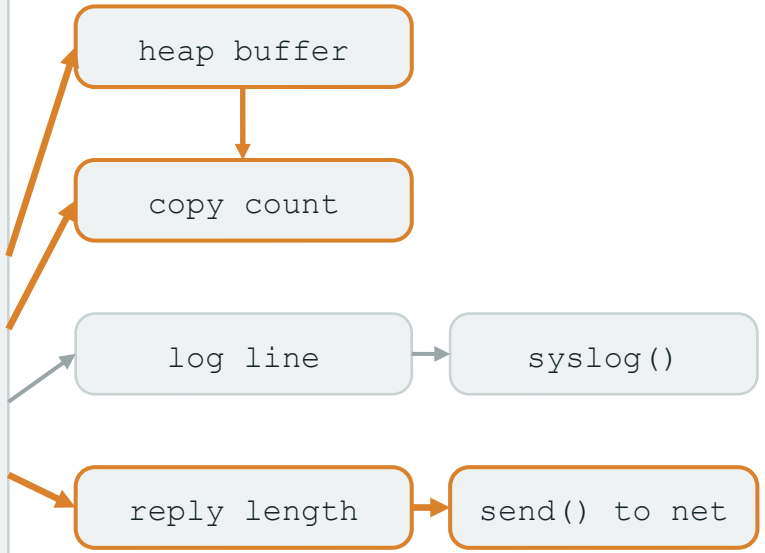
Guard changes are non-local, and undecidable in general

BACKWARD: what can reach the edited line?




```
void handle(struct pkt *p) {
  uint16_t n = read_len(p);
- if (n > p->rec_len) return;
  buf = alloc(n + PAD);
  copy(buf, p->data, n);
  log_len(n);
  reply(buf, n);
}
```

FORWARD: which uses relied on the check?

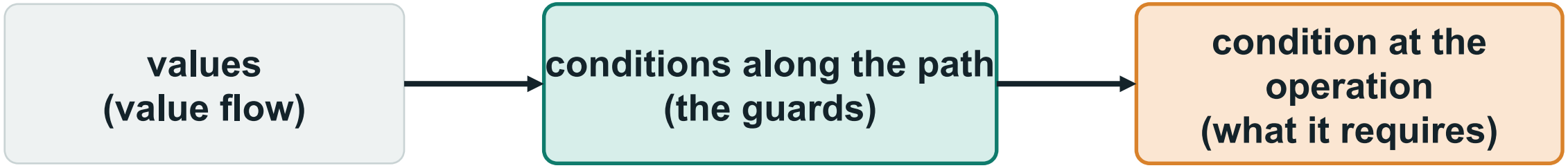


the patch: one deleted line

Blue: where the length comes from, across functions. Amber: what it reaches, across functions. The deleted check used to cut both. Deciding this over all executions is undecidable in general.

Illustration in the style of Heartbleed (functions and names are illustrative, not OpenSSL source)

Hypothesis: value flow, conditioned along the path and at the operation, decides safety



If we can collect the conditions on the values that reach an operation, along the path and at the operation, we can decide whether the use is authorized.

It also runs on real code: OpenSSL's heartbeat function

OpenSSL 1.0.1f · tls1_process_heartbeat · lifted by AuthDiff



Recorded: packet bytes -> length -> copy count, with no guard recovered on the path.

The historical fix (1.0.1g) adds the check that belongs on that arrow:

```
if (1 + 2 + payload + 16 > s->s3->rrec.length)
    return 0;   /* silently discard */
```

Real OpenSSL 1.0.1f ssl/t1_lib.c lifted by AuthDiff (vulnerable version only; no fixed-version diff) · 1.0.1g fix is historical · n2s reads 2 packet bytes as a length

Apple: the check stayed, but stopped running

SSLVerifySignedServerKeyExchange · sslKeyExchange.c (Security-55471)

```
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;

err = sslRawVerify(ctx, …);
if(err) {
    …
    goto fail;
}

fail:
    …
    return err;
```

jump

<- unconditional: always taken

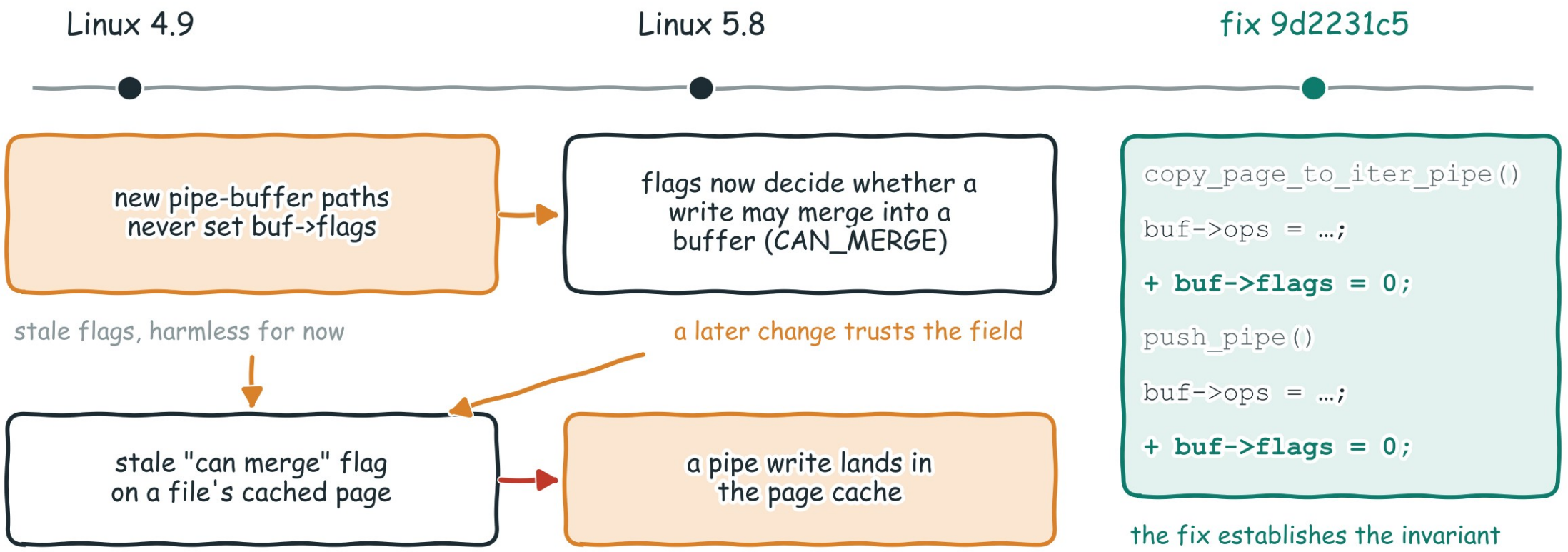
the signature check is still in the file, but never runs

err is still 0 from the last hash call: the function reports success

Apple: an attacker with a privileged network position may capture or modify data in SSL/TLS sessions.

Historical incident · CVE-2014-1266 · Apple Secure Transport (Security-55471 source; fixed in iOS 7.0.6) · not an AuthDiff result

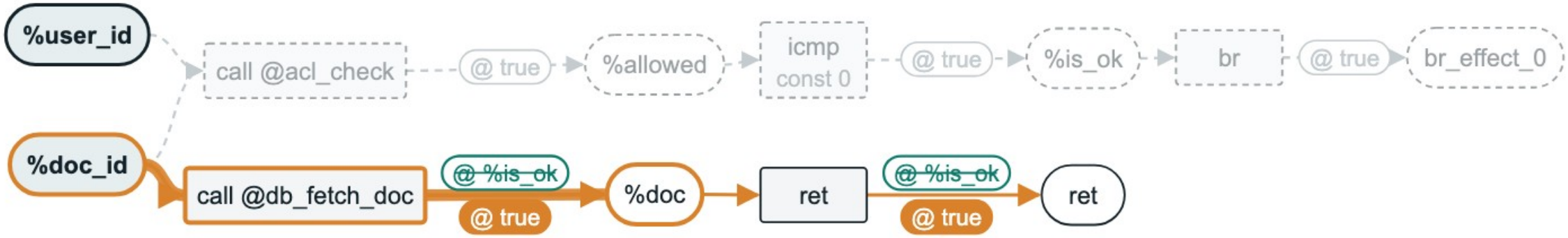
Dirty Pipe: a flag never set, then trusted



Unprivileged overwrite of read-only file data: local privilege escalation.

Historical incident · CVE-2022-0847 · Linux 5.8 and later (flags uninitialized since 4.9) · not an AuthDiff result

Backup · differential, document fetch (viewer)



NEW capability paths (introduced in after) — newly unguarded source → endpoint pairs

```
+ %doc_id -> %doc
  %doc_id --[true]--> %doc
+ %doc_id -> ret
  %doc_id --[true]--> %doc --[true]--> ret
```

before: 5 def lines, **2 guarded**
after: 2 def lines, **0 guarded**
2 def guards changed to **@ true**
2 new pairs · **2** witness paths
highlighted: witness 1.1

DUC viewer over recorded duc-proto diff output (OW-03) · presentation/viz/index.html

Backup · the checker spec, as the viewer draws it

Checker specification · `missing-bounds-check`

`experiments/specs/missing_bounds_check.yaml` · v0.1 · severity critical

written specification · evaluation planned

1 · SOURCE

Where the value comes from

param · any parameter

call · callee `^(@recv|@read|@recvfrom|@recvmsg)$`

load

→

2 · SINK (OPERAND)

What it must not reach

call · callee `^(@llvm\.memcpy\..*|@memcpy|@memmove|@bcopy)$`
operand: **argument 2**

call · callee `^(@write|@send|@sendto|@fwrite)$`
operand: **argument 2**

→

3 · REQUIRED GUARD

What must stand between

node · op `icmp*`

VIOLATION WHEN

`path: unconditioned`

`sanitizer_absent: icmp_for_source: true`

Detects an unconditioned flow from a network-derived (or untrusted) length value to the size argument of a bulk-copy operation (memcpy, memmove, write, send) without a bounds check interposed on every path.

The prototype does not evaluate this file yet. Today the check is read by hand against `duc-proto normalize` / `diff` output.

YAML (experiments/specs/missing_bounds_check.yaml) — comment lines and `_note` fields omitted

```yaml
name: missing-bounds-check
sources:
  - kind: param
    all_params: true
  - kind: call
    callee_regex: "^(@recv|@read|@recvfrom|@recvmsg)$"
  - kind: load
sinks:
  - kind: call
    callee_regex: "^(@llvm\\.memcpy\\..*|@memcpy|@memmove|@bcopy)$"
    index: 2
  - kind: call
    callee_regex: "^(@write|@send|@sendto|@fwrite)$"
    index: 2
sanitizers:
  - kind: node
    op_prefix: "icmp"
violation:
  path: unconditioned
  sanitizer_absent:
    icmp_for_source: true
severity: critical
```

Backup · demo 1: document fetch

```
$ duc-proto normalize broken_access_control_safe.ll | grep 'db_fetch_doc'
  def %doc                  <- call(@db_fetch_doc, %doc_id)    @ %is_ok
$ duc-proto diff broken_access_control_safe.ll broken_access_control.ll
@get_document:
  NEW capability paths (introduced in after):
    + %doc_id -> %doc
        %doc_id --[true]--> %doc
    + %doc_id -> ret
        %doc_id --[true]--> %doc --[true]--> ret
```

Recorded OW-03 run, DUC v2 duc-proto, 2026-11-18 · demo/recorded/case1.txt

Backup · demo 2: heartbeat model

```
$ duc-proto normalize heartbleed_toy_safe.ll | grep 'call\(@write'
  def %sent                  <- call(@write, const:1, %payload_len, %req_data)  @ %too_big
$ duc-proto diff heartbleed_toy_safe.ll heartbleed_toy.ll
@heartbeat_reply:
  NEW capability paths (introduced in after):
    + %req -> %req_data
        %req --[true]--> %req_data
    + %req -> %sent
        %req --[true]--> %req_data --[true]--> %sent
        %req --[true]--> %pl_ptr --[true]--> %payload_len_raw --[true]--> %payload_len --…
    + %req -> ret
        %req --[true]--> %req_data --[true]--> %sent --[true]--> ret
        %req --[true]--> %pl_ptr --[true]--> %payload_len_raw --[true]--> %payload_len --…
```

Recorded DC-03 run, DUC v2 duc-proto, 2026-11-18 · demo/recorded/case2.txt

Backup · demo 3: ping endpoint

```
$ duc-proto normalize cmd_injection_safe.ll | grep 'build_ping_cmd|@system'
  def %cmd                  <- call(@build_ping_cmd, %host)    @ %is_bad
  def %ret                  <- call(@system, %cmd)             @ %is_bad
$ duc-proto diff cmd_injection_safe.ll cmd_injection.ll
@run_ping:
  NEW capability paths (introduced in after):
    + %host -> %cmd
        %host --[true]--> %cmd
    + %host -> %ret
        %host --[true]--> %cmd --[true]--> %ret
    + %host -> ret
        %host --[true]--> %cmd --[true]--> %ret --[true]--> ret
```

Recorded OW-02 run, DUC v2 duc-proto, 2026-11-18 · demo/recorded/case3.txt

Backup · demo 4: integer dispatch

```
$ duc-proto normalize before.ll | grep 'call\(@write'
  def %result                <- call(@write, const:1, %safe_buf, %uval)  @ %cmp_hi
$ duc-proto diff before.ll after.ll
@safe_dispatch:
  NEW capability paths (introduced in after):
    + %safe_buf -> %result
        %safe_buf --[true]--> %result
    + %safe_buf -> ret
        %safe_buf --[true]--> %result --[true]--> ret
    + %user_val -> %danger_ptr
        %user_val --[true]--> %uval64 --[true]--> %danger_ptr
    + %user_val -> %leaked
    … (full output: demo/recorded/case4.txt)
```

Recorded DC-02 run, DUC v2 duc-proto, 2026-11-18 · demo/recorded/case4.txt