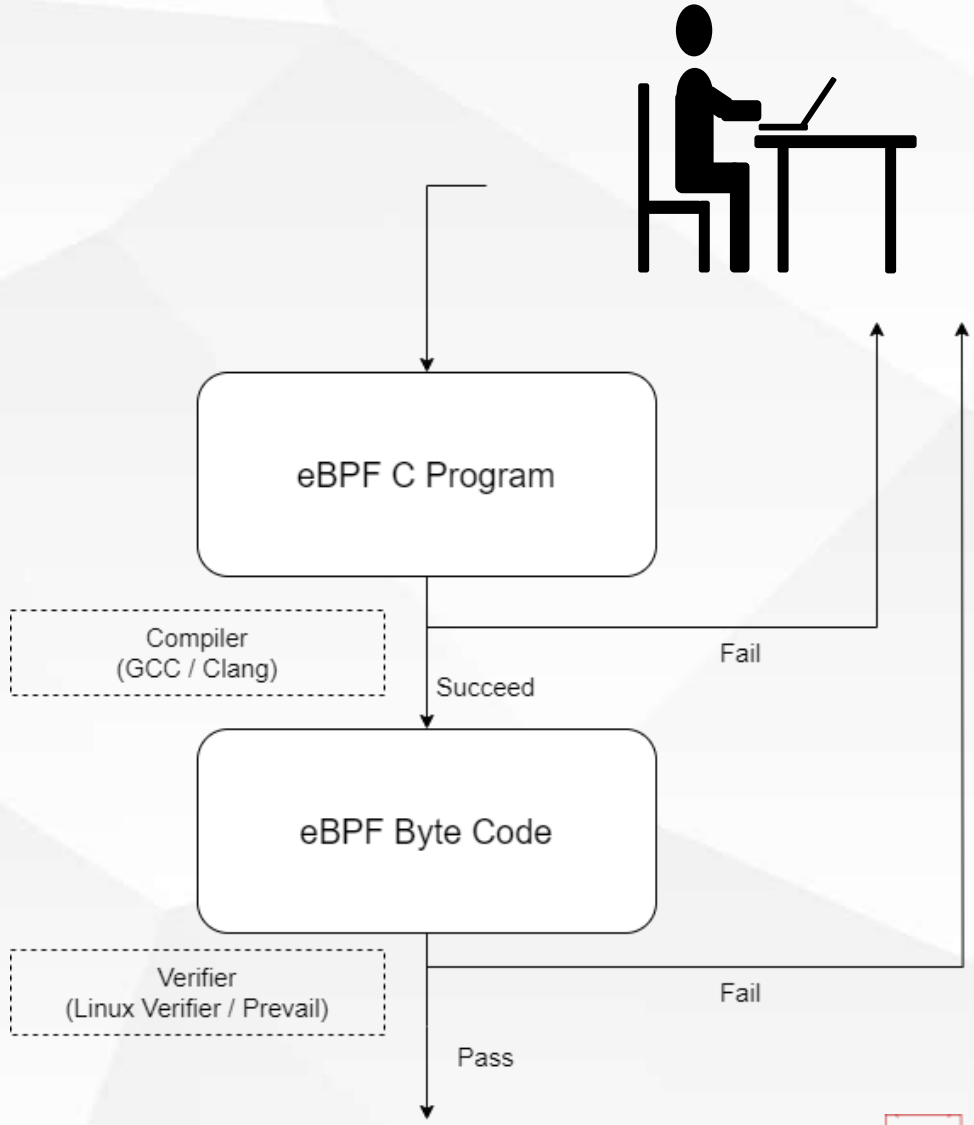
上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

SJTU

VEP: A Two-stage Verification Toolchain for Full eBPF Programmability

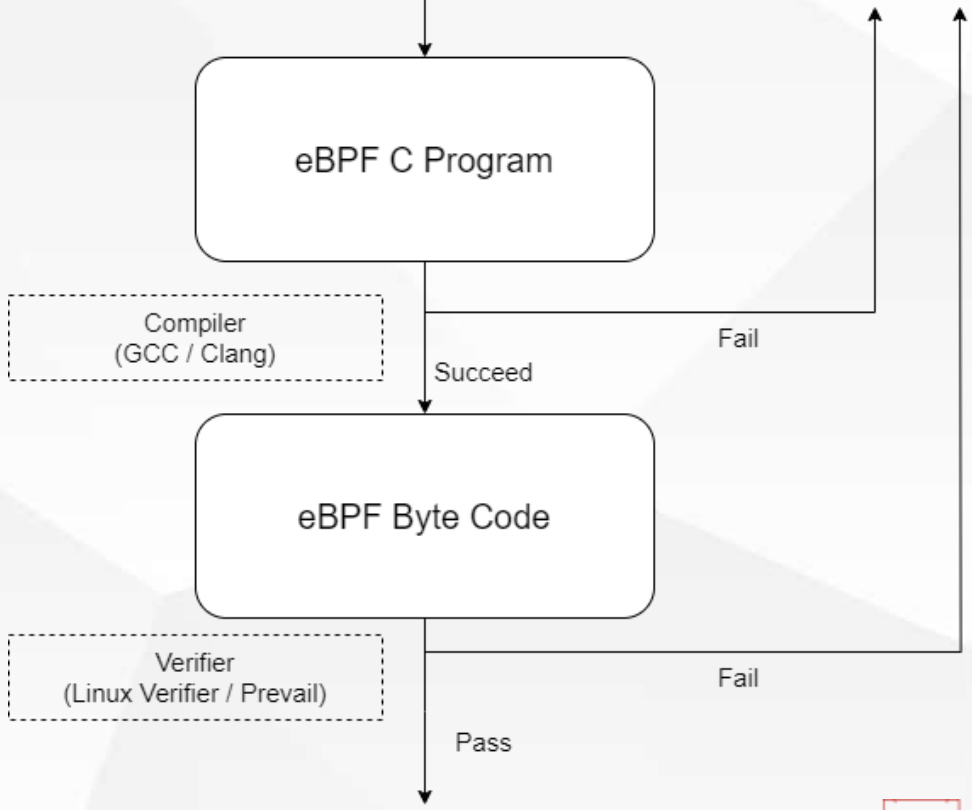
Xiwei Wu, Yueyang Feng, Tianyi Huang,
Xiaoyang Lu, Shengkai Lin, Lihan Xie,
Shizhen Zhao and Qinxiang Cao

Background



Verifier : reject unsafe programs

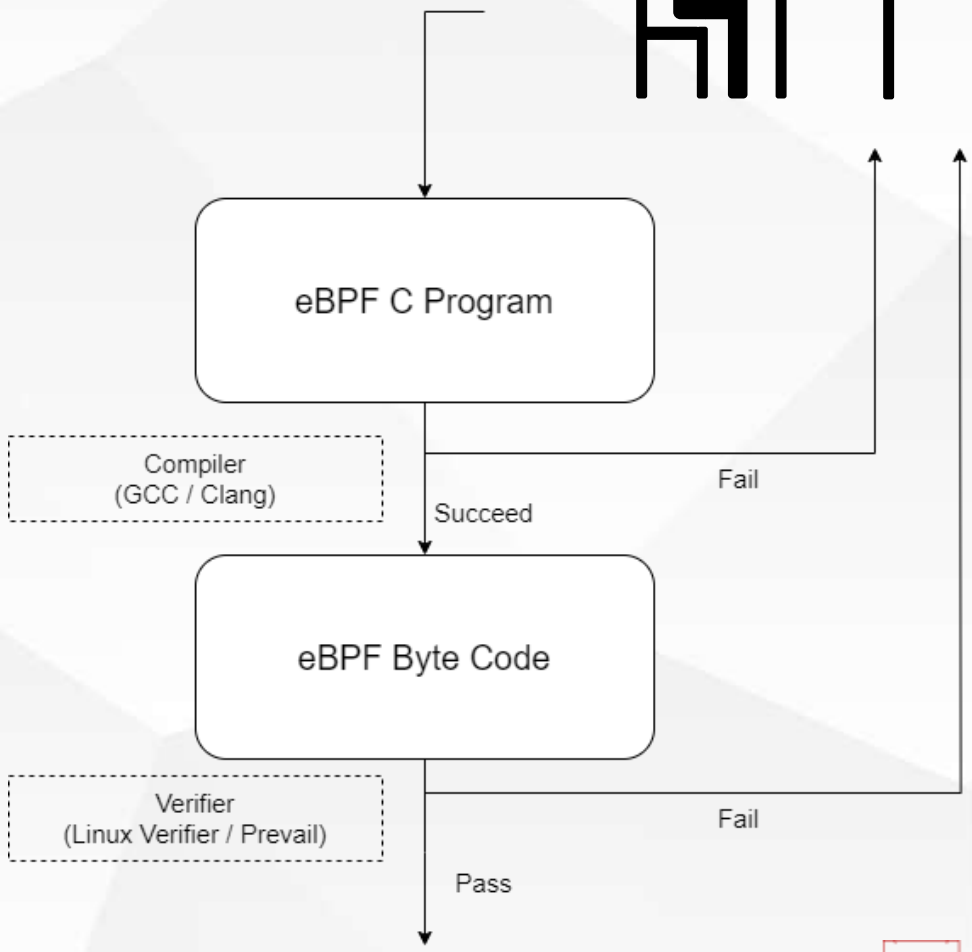
Problem 1: Too much false positive



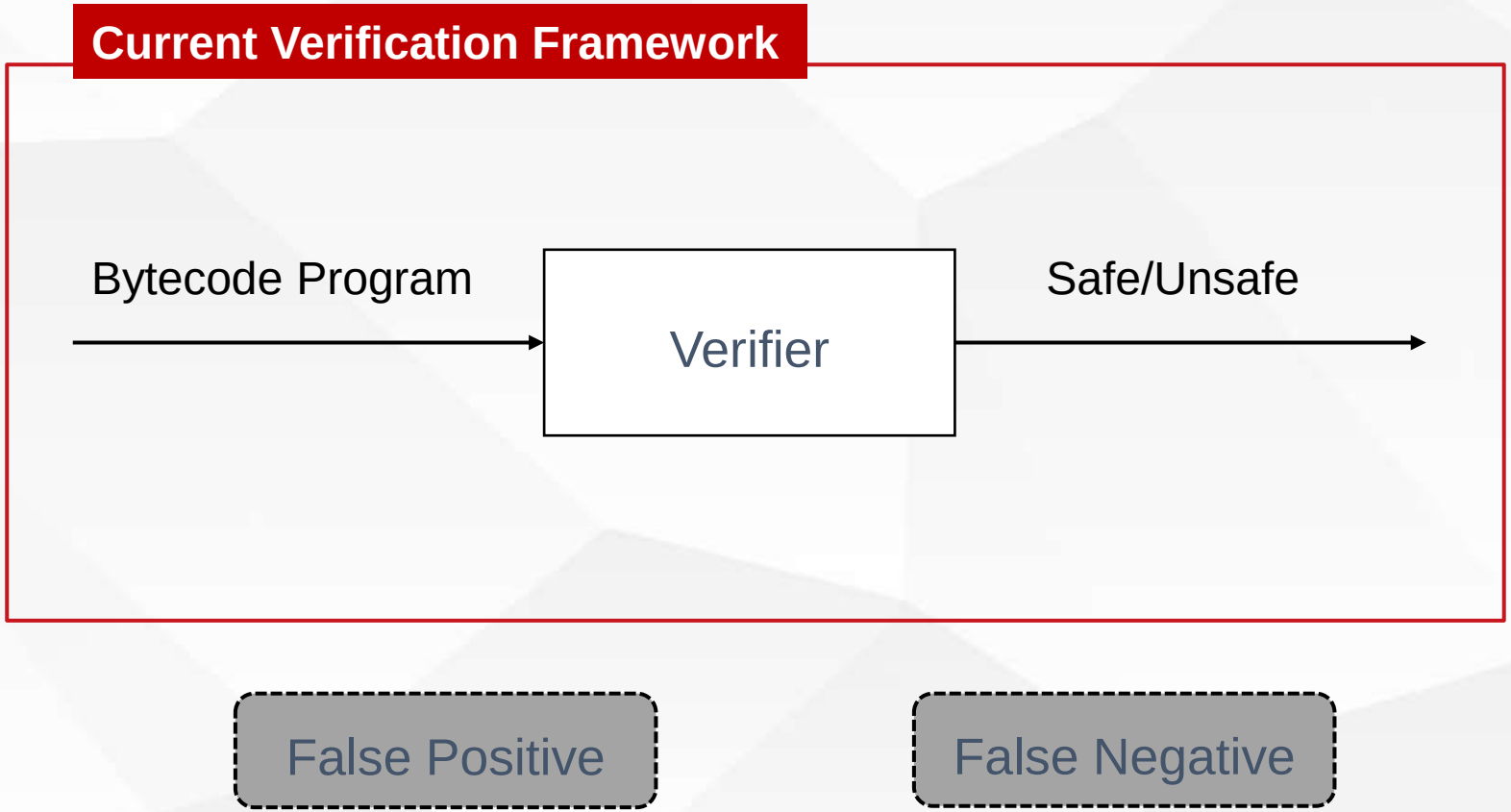
Verifier : reject “unsafe” programs

Problem 1: Too much false positive

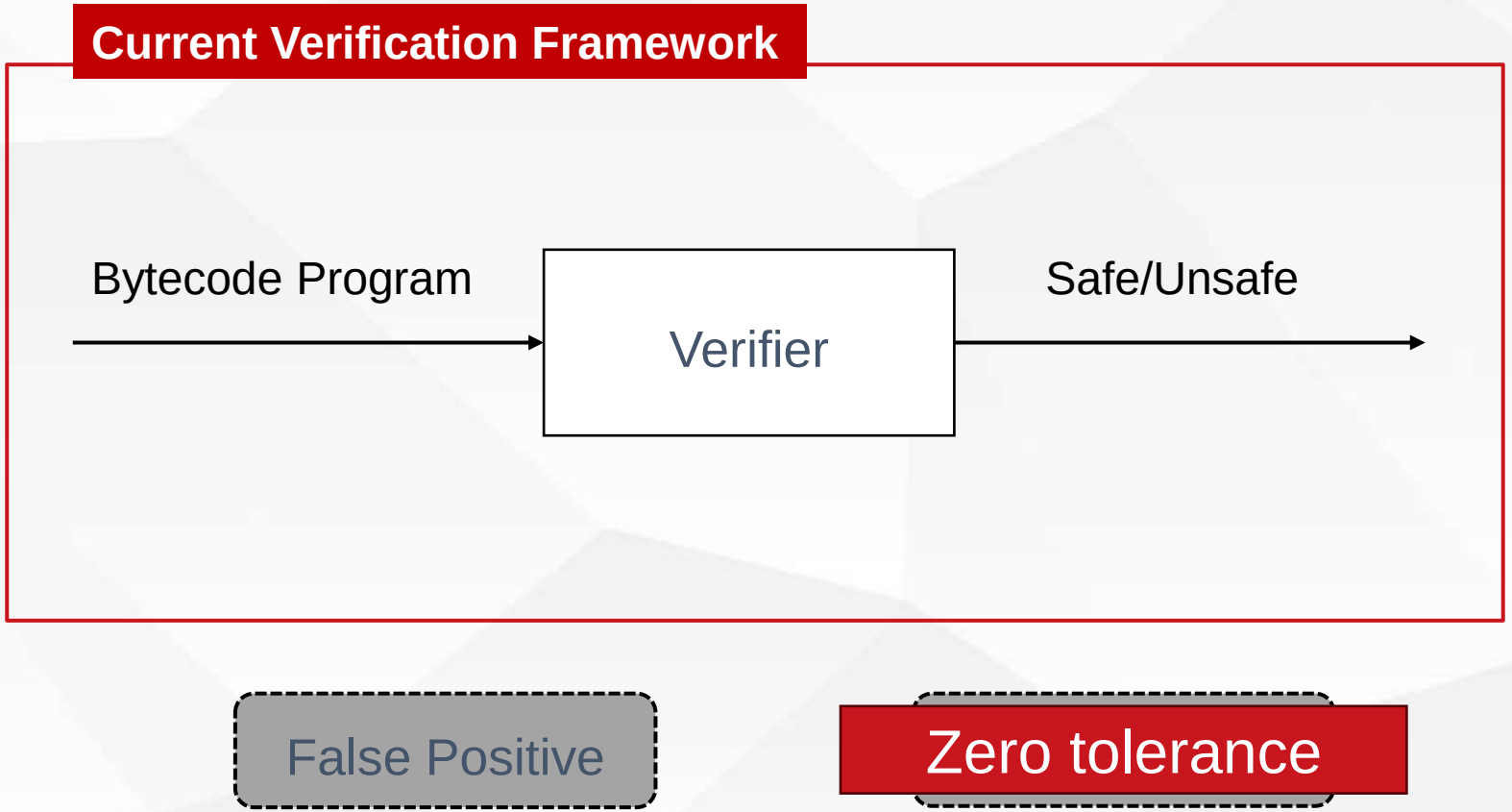
	Linux Verifier	Prevail
Methods	Control Flow Graph Analysis	Abstract Interpretation
Verification Power	--	-
Programmability	---	-
Resource consumption	Low	High



Kernel verifier choice



Kernel verifier choice



Problem 2 : unreadable feedback

```
256: (7b) *(u64 *)(r7 + 48) = r1
257: (7b) *(u64 *)(r7 + 40) = r1
258: (7b) *(u64 *)(r7 + 32) = r1
259: (7b) *(u64 *)(r7 + 24) = r1
260: (7b) *(u64 *)(r7 + 16) = r1
261: (7b) *(u64 *)(r7 + 8) = r1
262: (7b) *(u64 *)(r7 + 0) = r1
263: (7b) *(u64 *)(r7 + 40) = r0
264: (b7) r1 = 2
265: (7b) *(u64 *)(r7 + 32) = r1
266: (bf) r1 = r6
267: (18) r2 = 0xffff880428988b00
269: (18) r3 = 0xffffffff
271: (bf) r4 = r7
272: (b7) r5 = 128
273: (85) call 25
R4 type=map_value expected=fp
-- END PROG LOAD LOG --
```

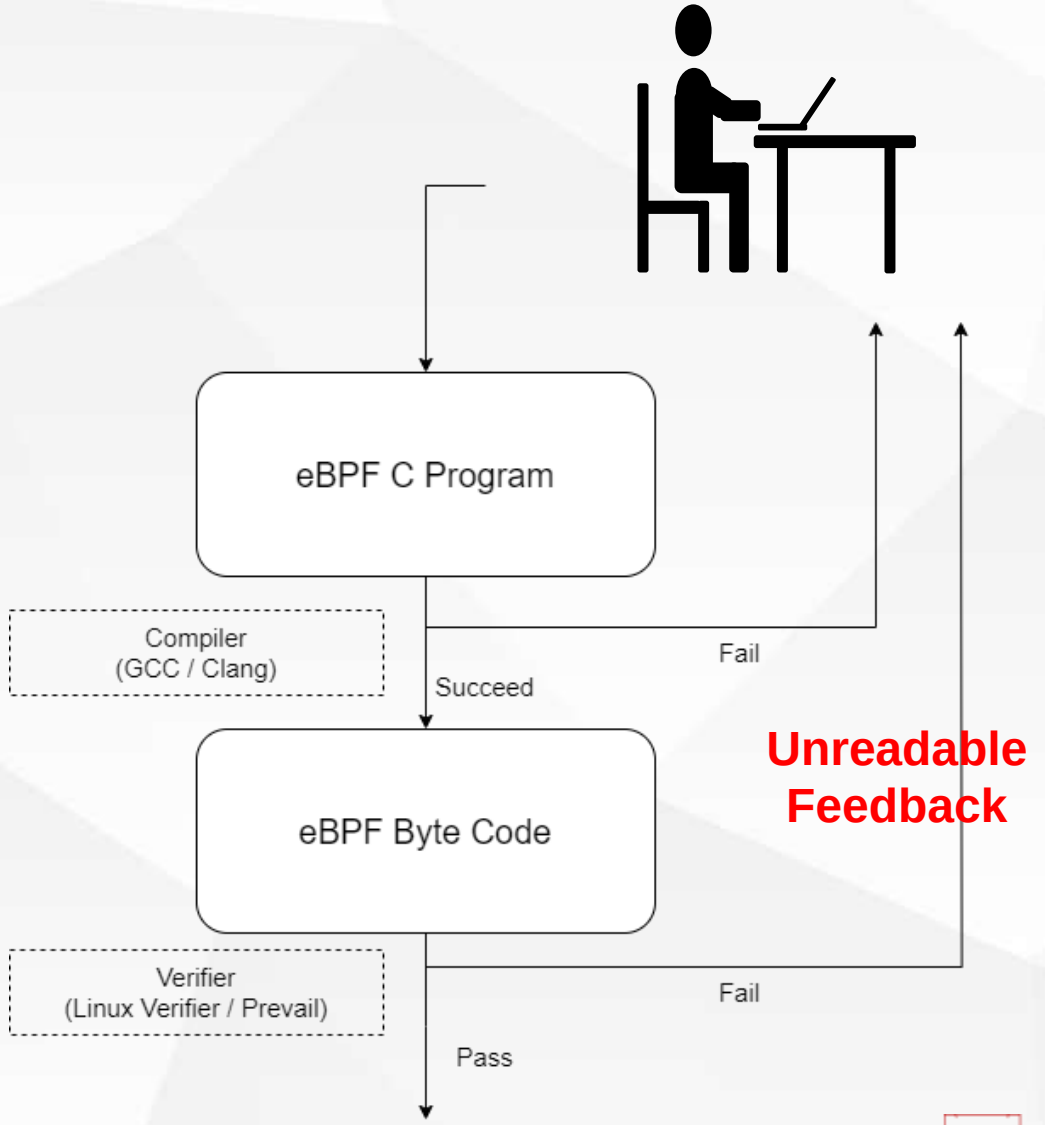


Figure from https://zhuanlan.zhihu.com/p/590851484

Problem 2 : unreadable feedback

Qualified C Programming | Home | About QCP | To VEP | yashen | Logout | Check

A | 16px | Create | Save | sll_auto.c (ReadOnly) | Example Files | User Files | User Assertion

```
1  #include "verification_stdlib.h"
2  #include "verification_list.h"
3  #include "sll_shape_def.h"
4
5  struct list* malloc_list(int data)
6  /*@ With data0
7      Require data == data0 && emp
8      Ensure __return != 0 && __return -> data == data0 && __return -> next == 0
9  */;
10
11 void free_list(struct list * x)
12 /*@ With d n
13     Require x -> data == d && x -> next == n
14     Ensure emp
15 */;
16
17 struct list * sll_copy(struct list * x)
18 /*@ Require listrep(x)
19     Ensure  listrep(__return) * listrep(x)
20  */
21 {
22     struct list *y, *p, *t;
23     y = malloc_list(0);
24     t = y;
25     p = x -> next;
26
```

Log

```
File: Unknown
Function: sll_copy
```

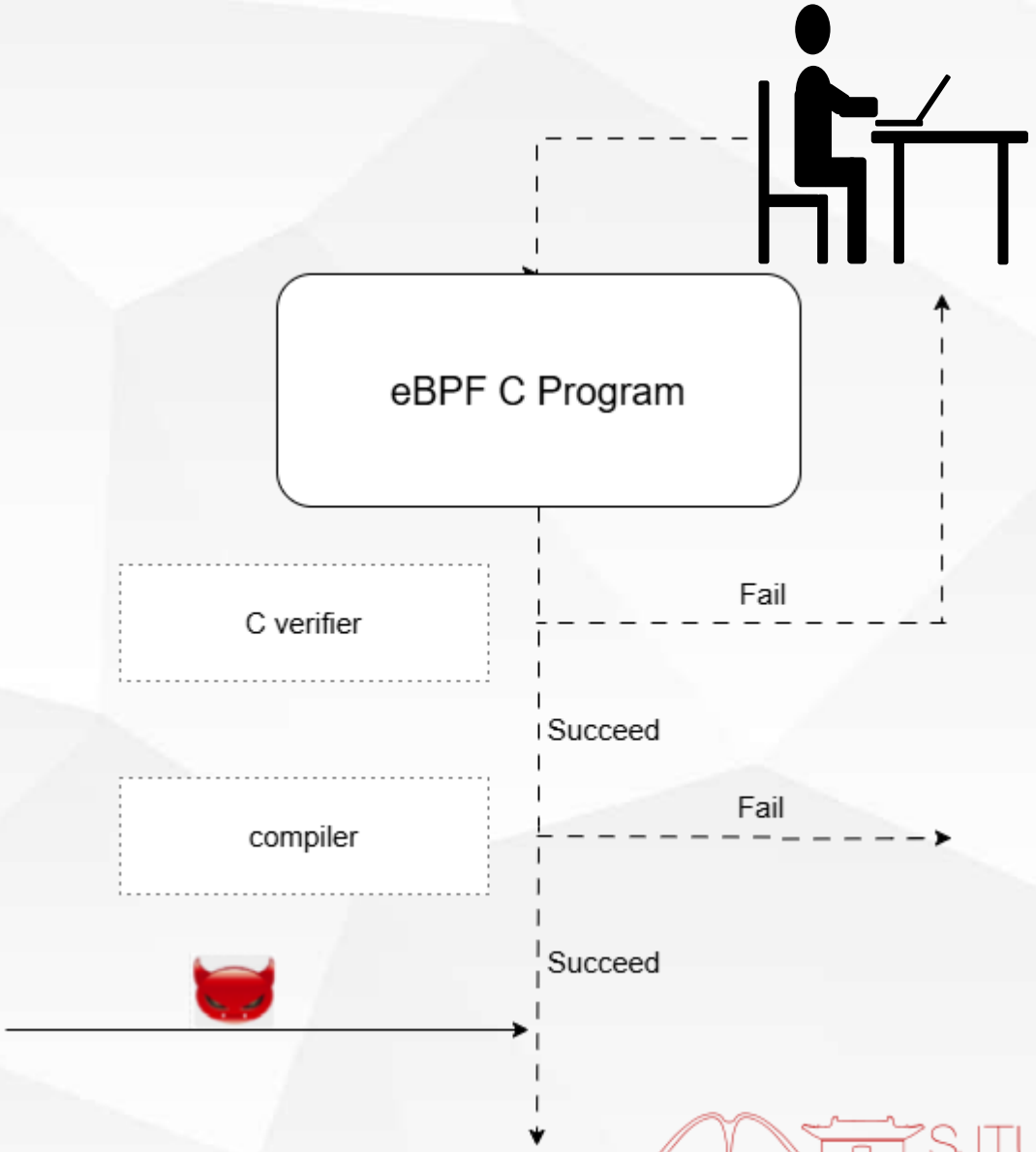
▶ Normal

Output

```
Error: fatal error: Cannot derive the precondition of Memory Read.

Line: 25

Witness Summary:

Function: sll_copy
  2 auto-solved witnesses
  0 witnesses need manual solving

  partial_solve:
    Total: 1
    Auto-solved: 1
    Manual: 0

  safety_checker:
    Total: 1
    Auto-solved: 1
    Manual: 0
```

C verifier vs. Bytecode verifier

	C Verifier	Bytecode verifier
Input	C programs	Bytecode programs
Feedback	+	-
Resource consumption	High	Low
TCB	Large	Small



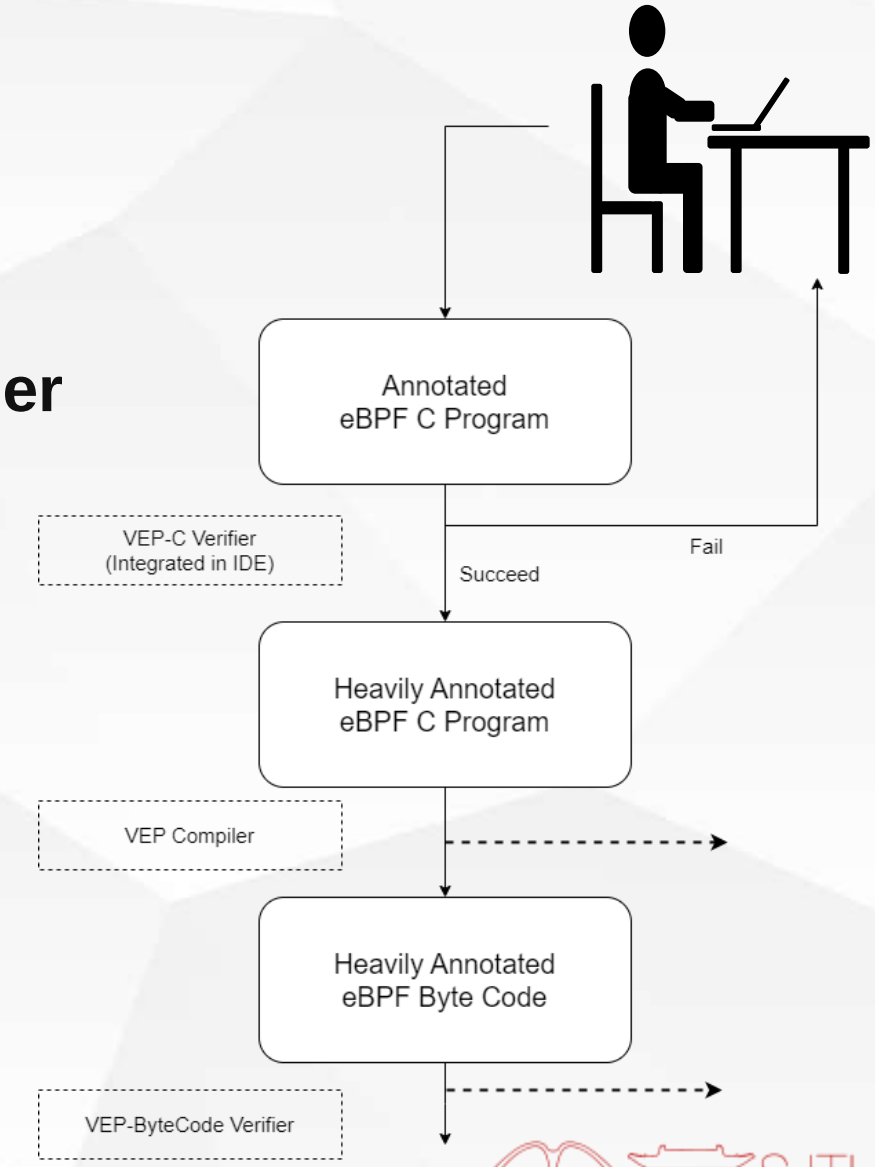
Summary of existing verifier

- Too much false positive
- Feedback is not friendly enough for programmers

Overview of VEP

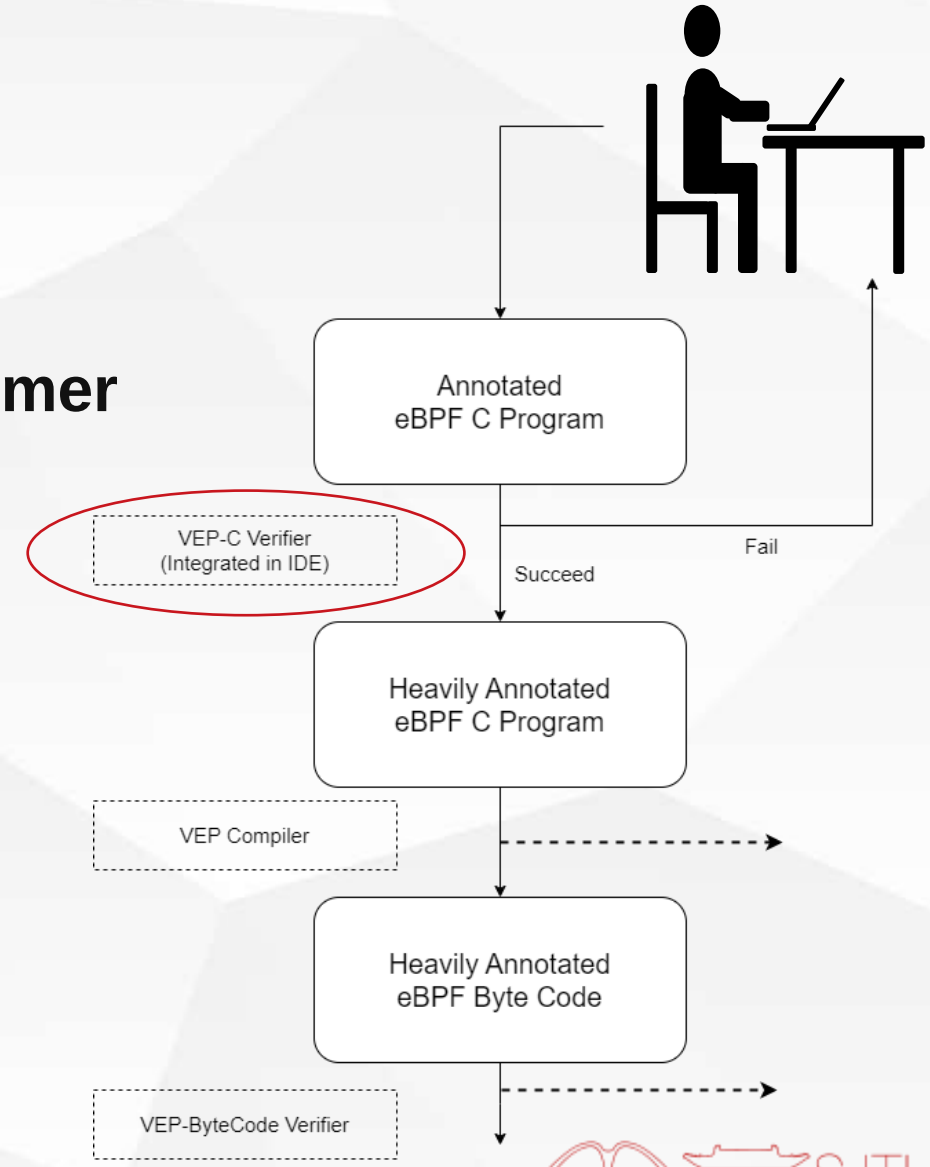
Design

- **Verification power is not enough**
  - Require user provide additional annotations
- **Feedback is not friendly enough for programmer**
  - Two stage verification



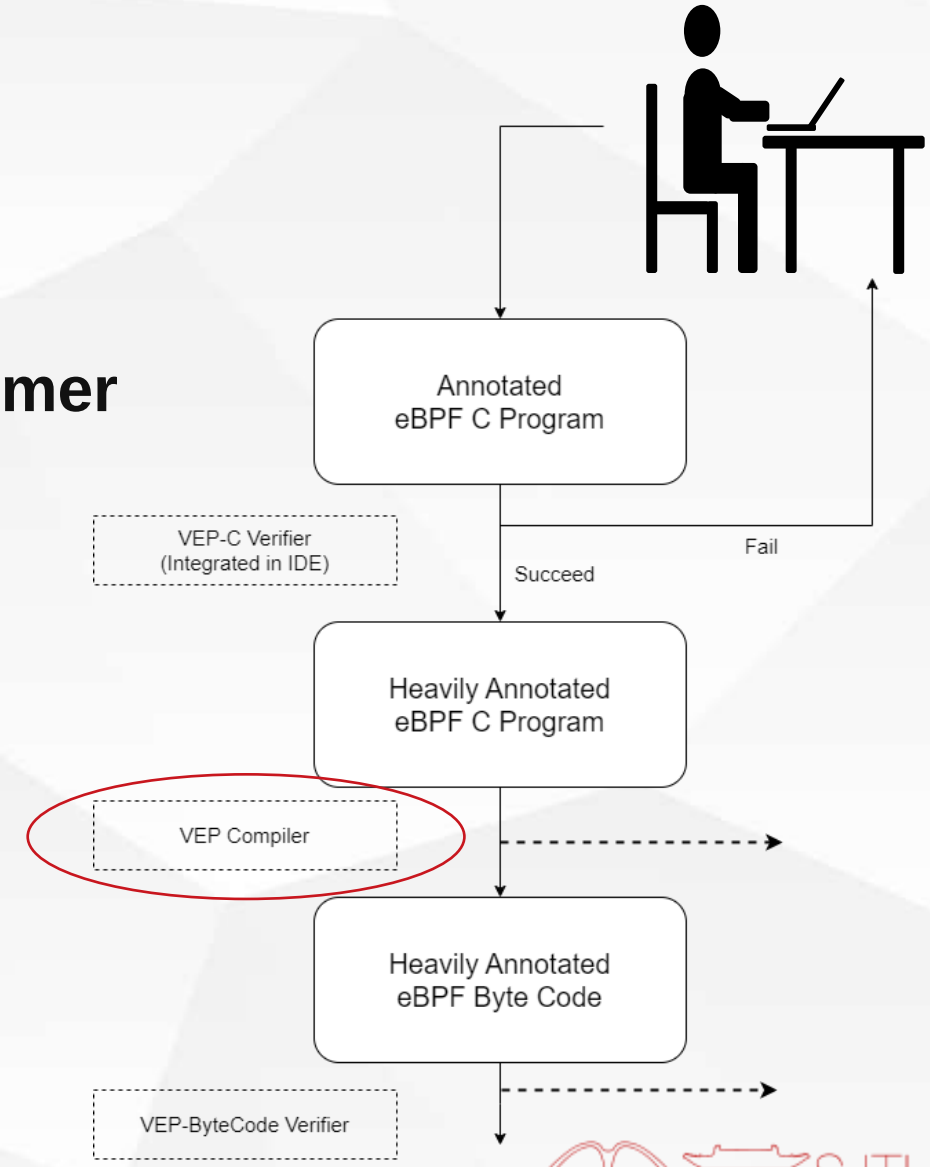
Design

- **Verification power is not enough**
  - Require user provide additional annotations
- **Feedback is not friendly enough for programmer**
  - Two stage verification



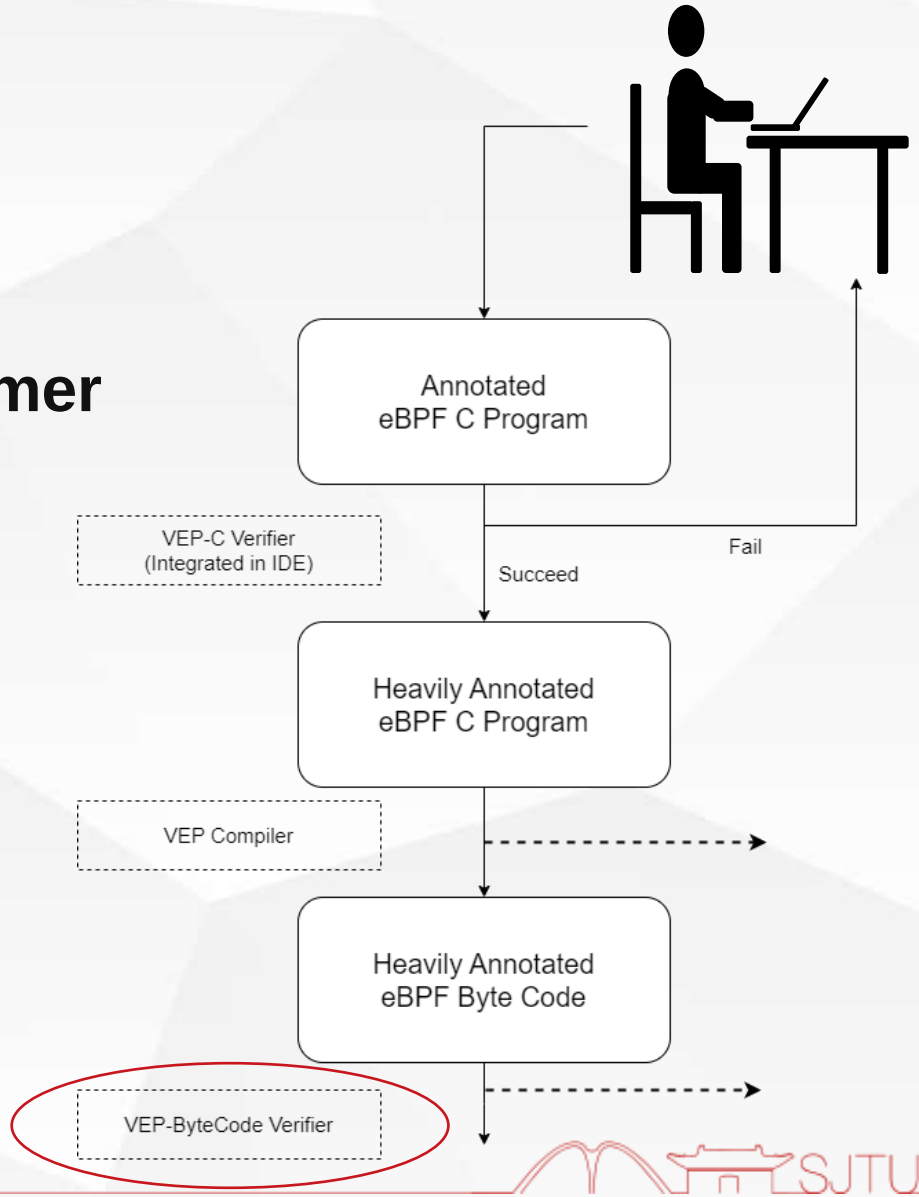
Design

- **Verification power is not enough**
  - Require user provide additional annotations
- **Feedback is not friendly enough for programmer**
  - Two stage verification



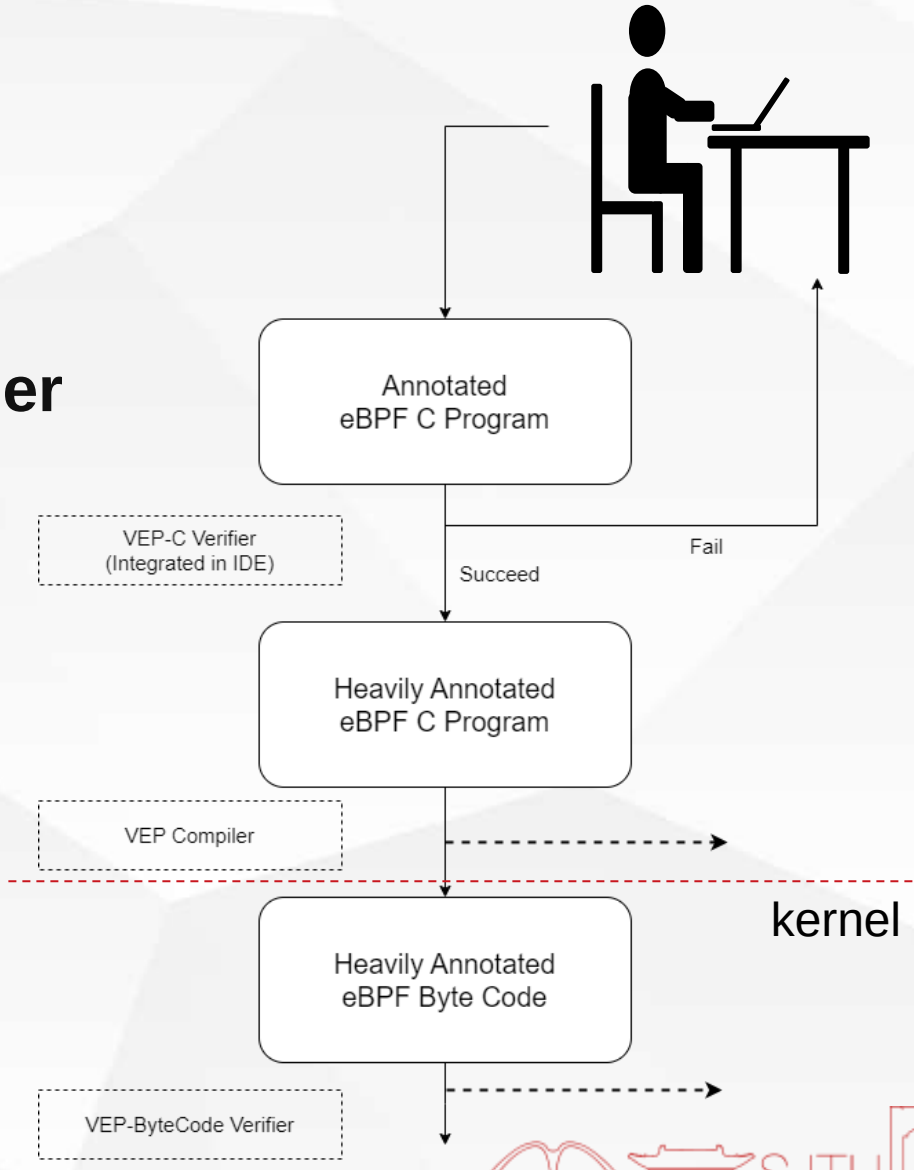
Design

- **Verification power is not enough**
  - Require user provide additional annotations
- **Feedback is not friendly enough for programmer**
  - Two stage verification

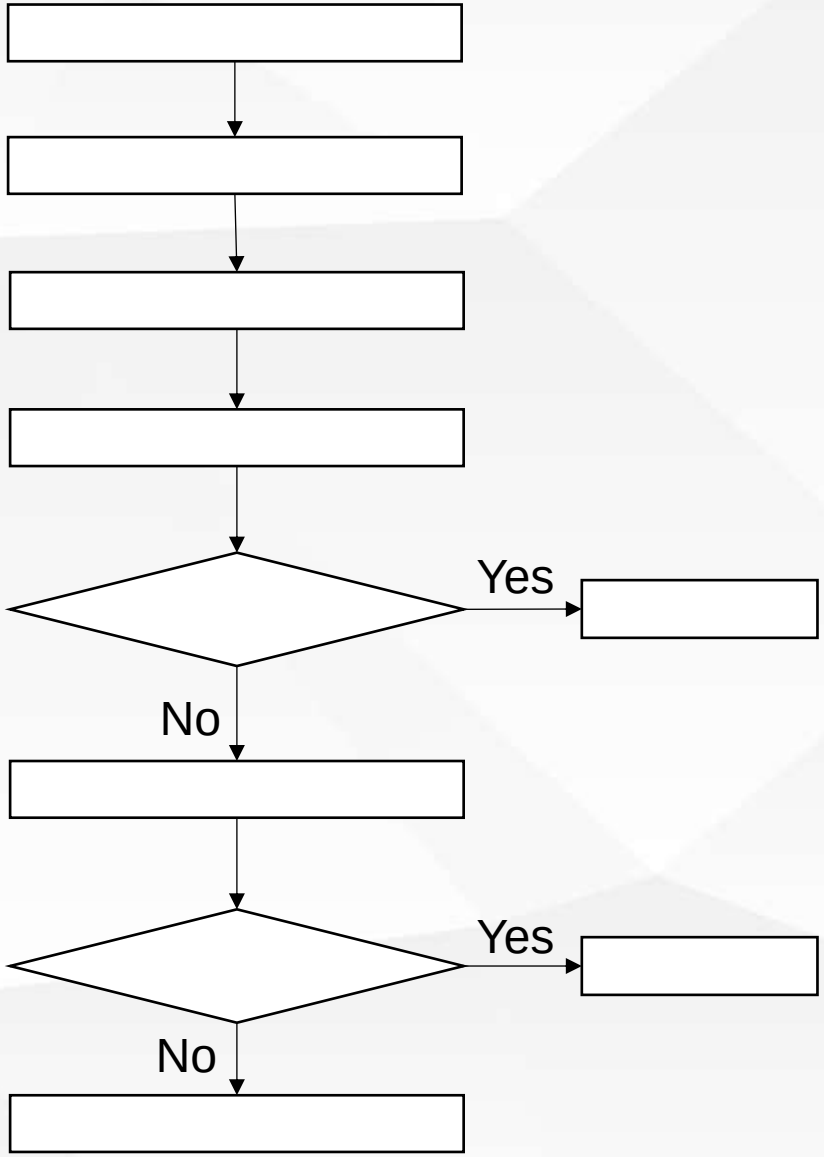


Design

- **Verification power is not enough**
  - Require user provide additional annotations
- **Feedback is not friendly enough for programmer**
  - Two stage verification

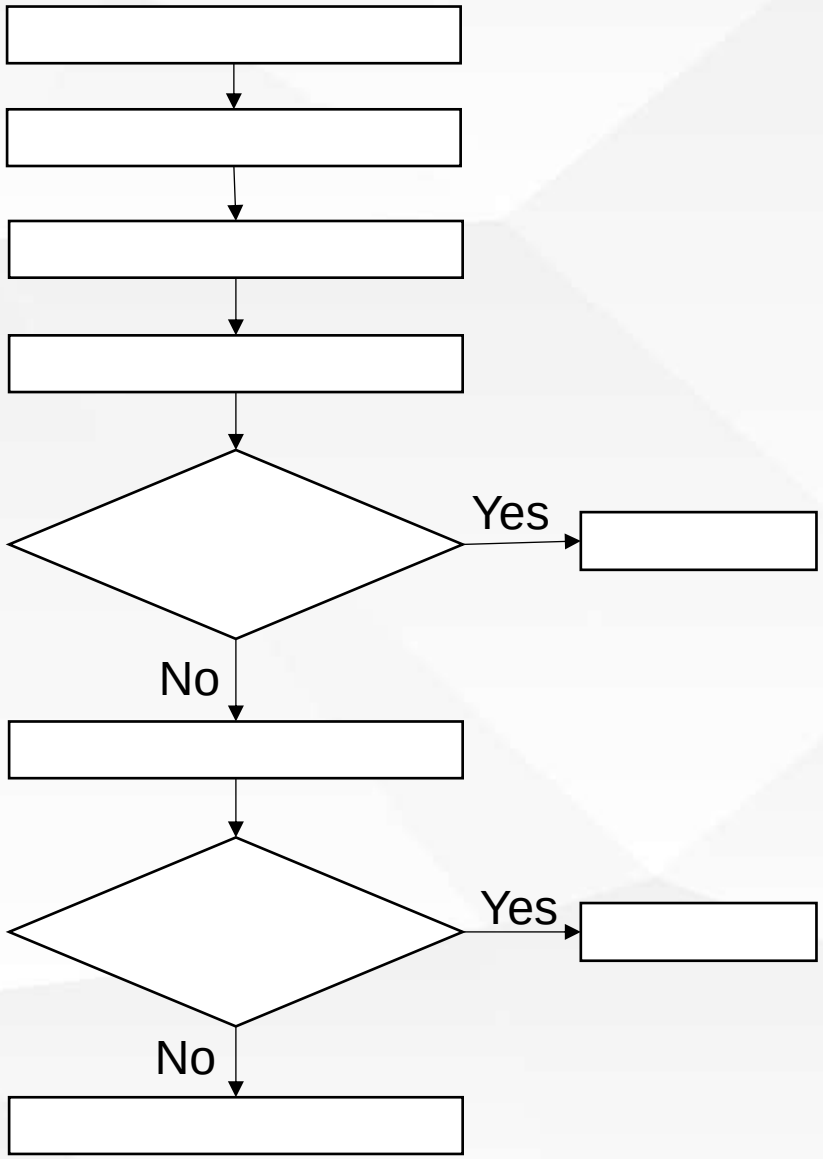


Example



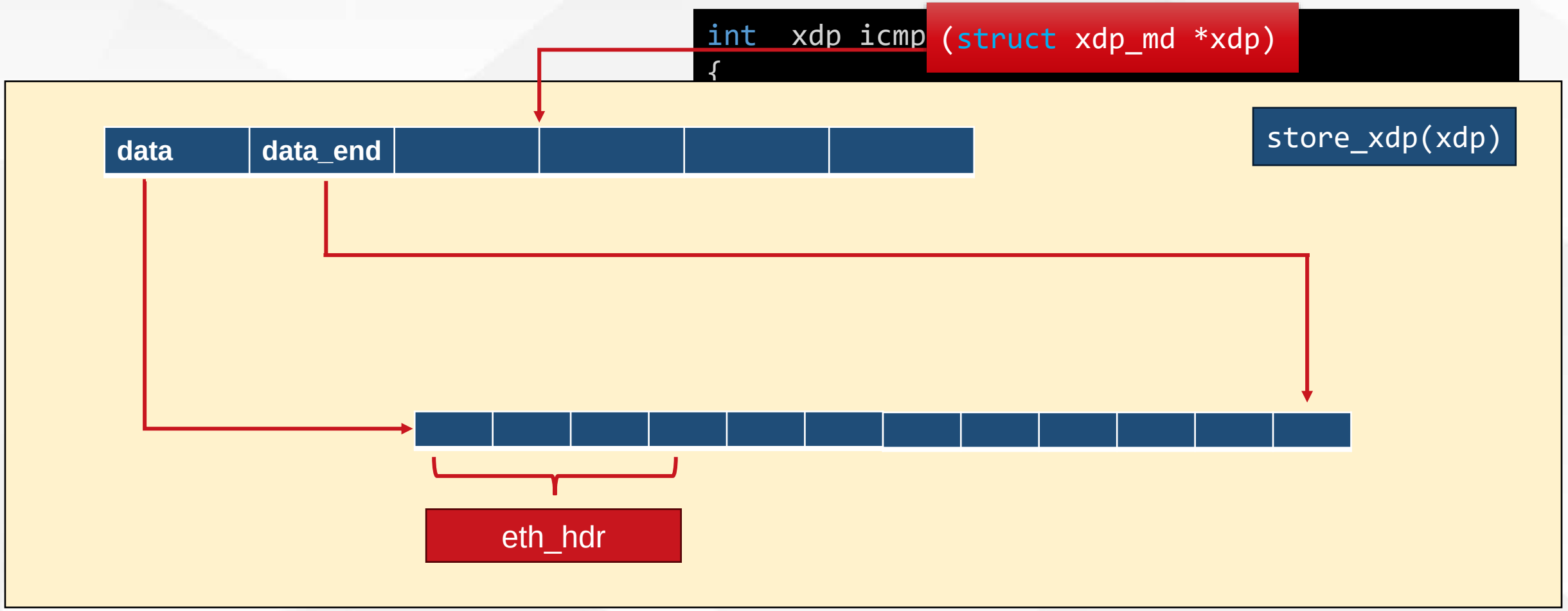

```
int _xdp_icmp(struct xdp_md *xdp)
{
 void *data_end = (void *)xdp->data_end;
 void *data = (void *)xdp->data;
 struct eth_hdr *eth = data;
 if ((void*) (eth + 1) > data_end)
  return 1;

 unsigned int h_proto = eth->h_proto;

 if ((int)h_proto == htons(2048))
  return handle_ipv4(xdp);
 else
  return 2;
}
```

Example

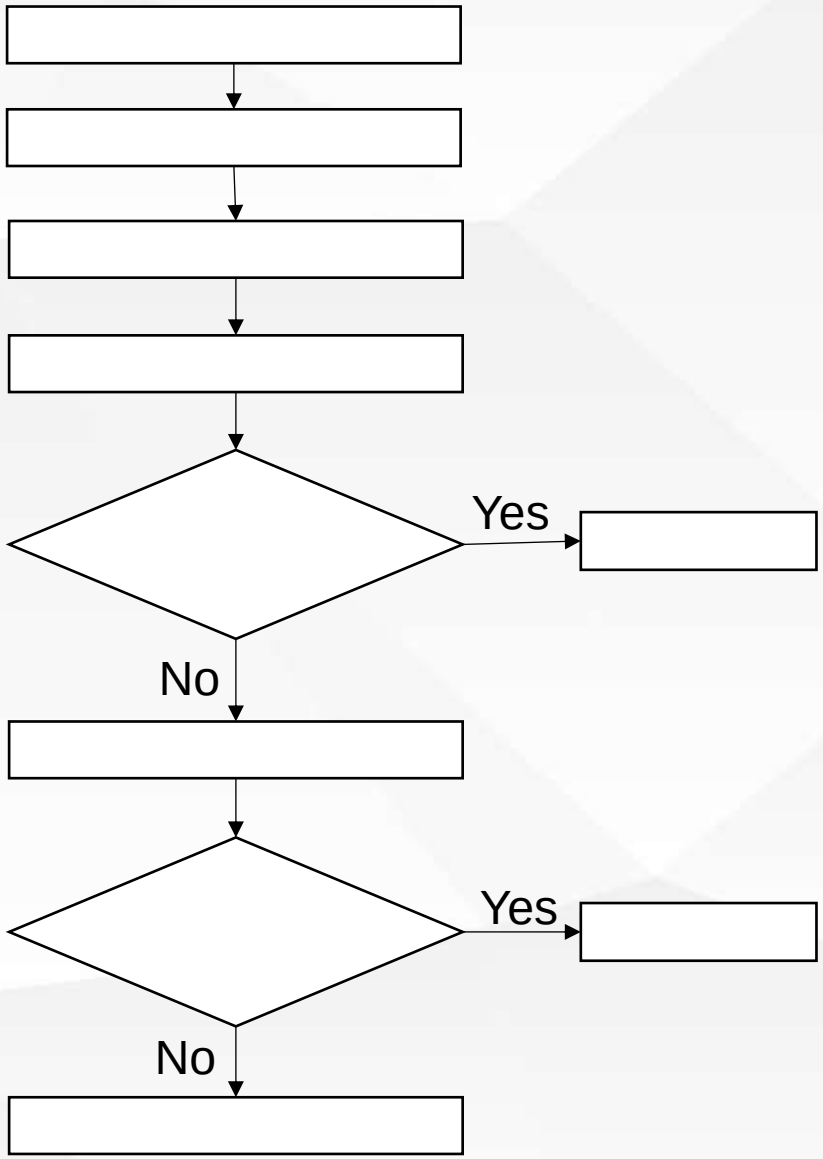



```
int _xdp_icmp (struct xdp_md *xdp)
{
 void *data_end = (void *)xdp->data_end;
 void *data = (void *)xdp->data;
 struct eth_hdr *eth = data;
 if ((void*) (eth + 1) > data_end)
  return 1;

 unsigned int h_proto = eth->h_proto;

 if ((int)h_proto == htons(2048))
  return handle_ipv4(xdp);
 else
  return 2;
}
```

Example


int xdp_icmp (struct xdp_md *xdp)
{
data
data_end
store_xdp(xdp)
eth_hdr

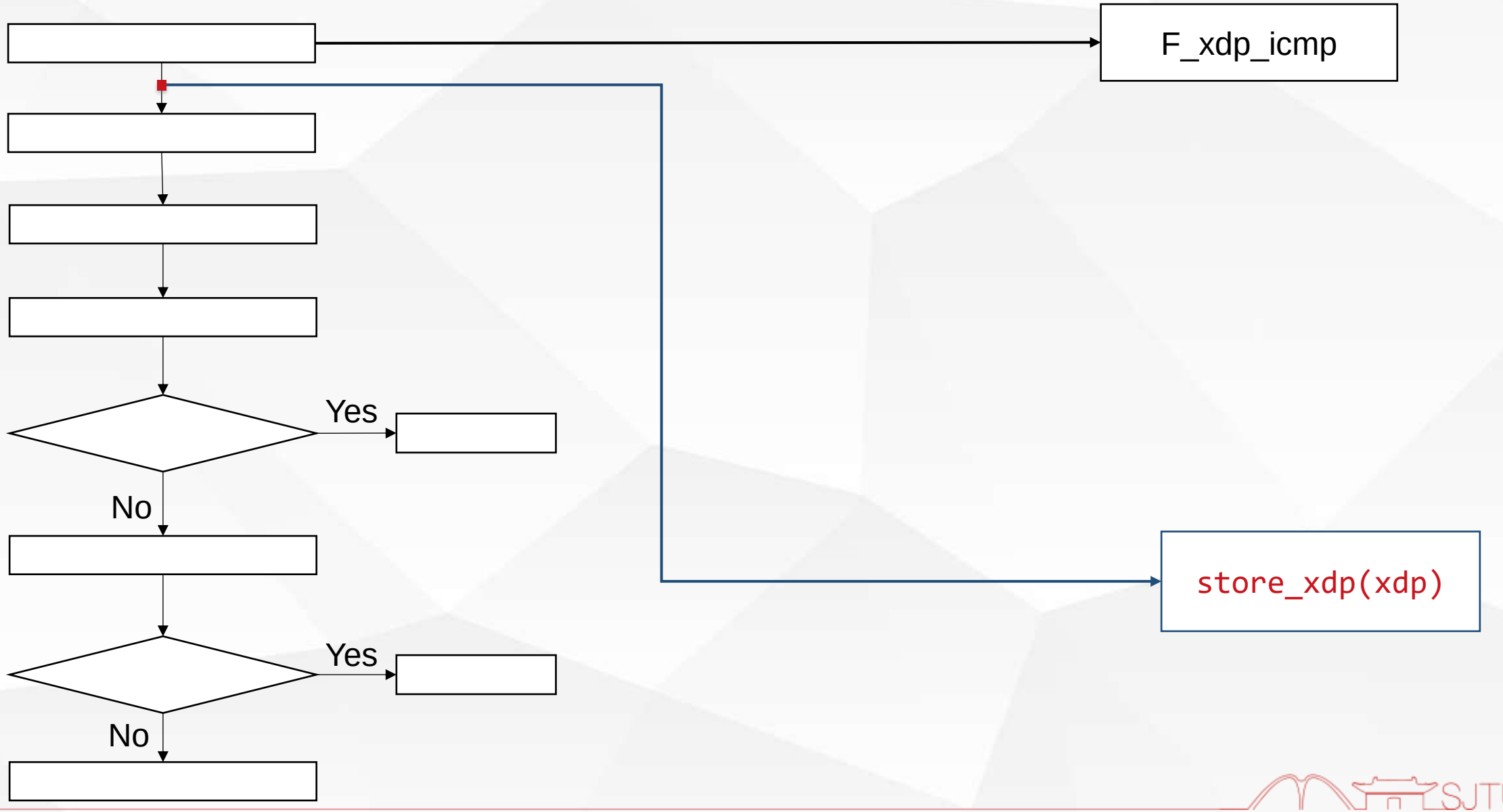
Example



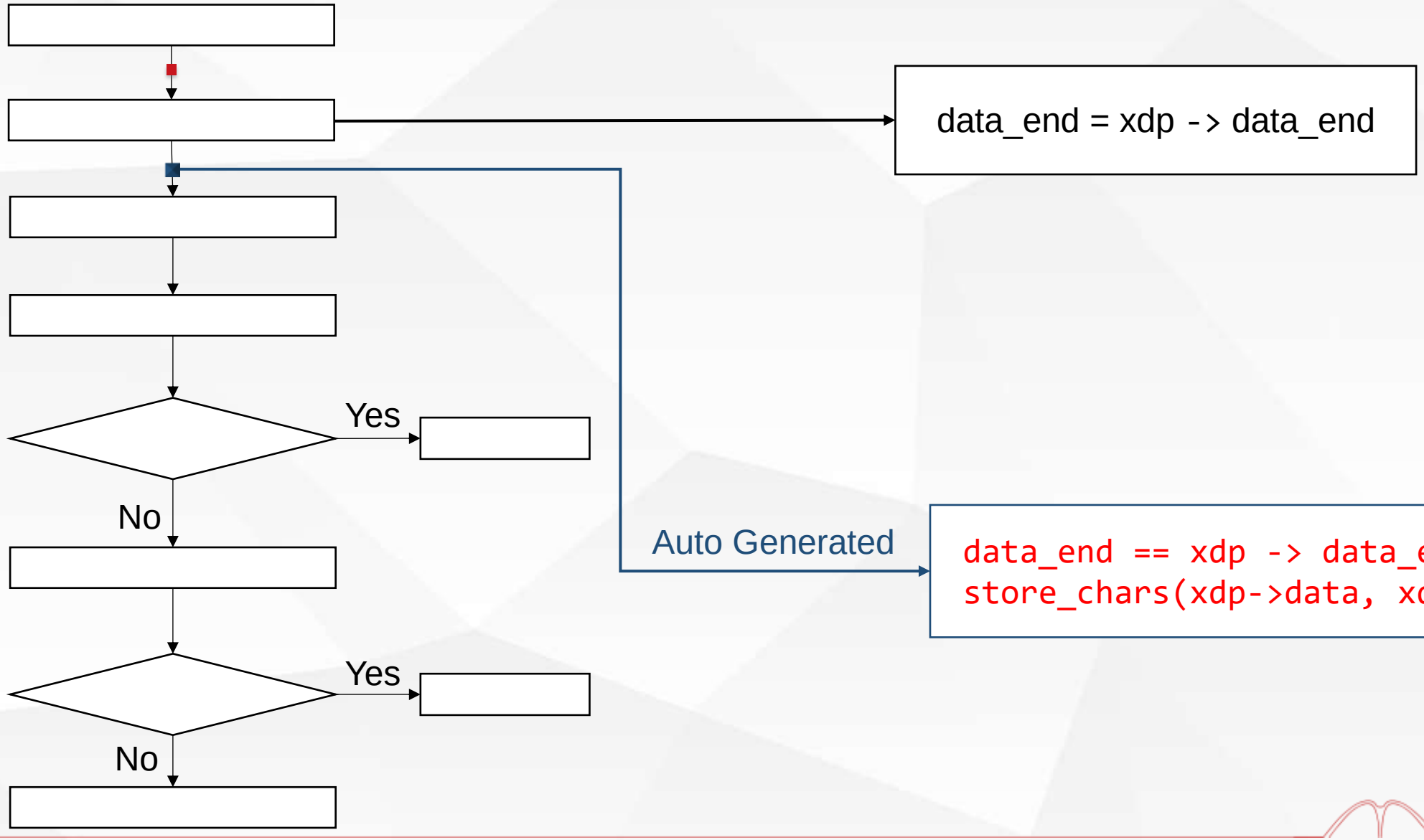

```
int _xdp_icmp(struct xdp_md *xdp)
/*@
 Require store_xdp(xdp)
 Ensure store_xdp(xdp)
*/
{
 void *data_end = (void *)xdp->data_end;
 void *data = (void *)xdp->data;
 struct eth_hdr *eth = data;
 if ((void*) (eth + 1) > data_end)
  return 1;

 unsigned int h_proto = eth->h_proto;

 if ((int)h_proto == htons(2048))
  return handle_ipv4(xdp);
 else
  return 2;
}
```

VEP-C

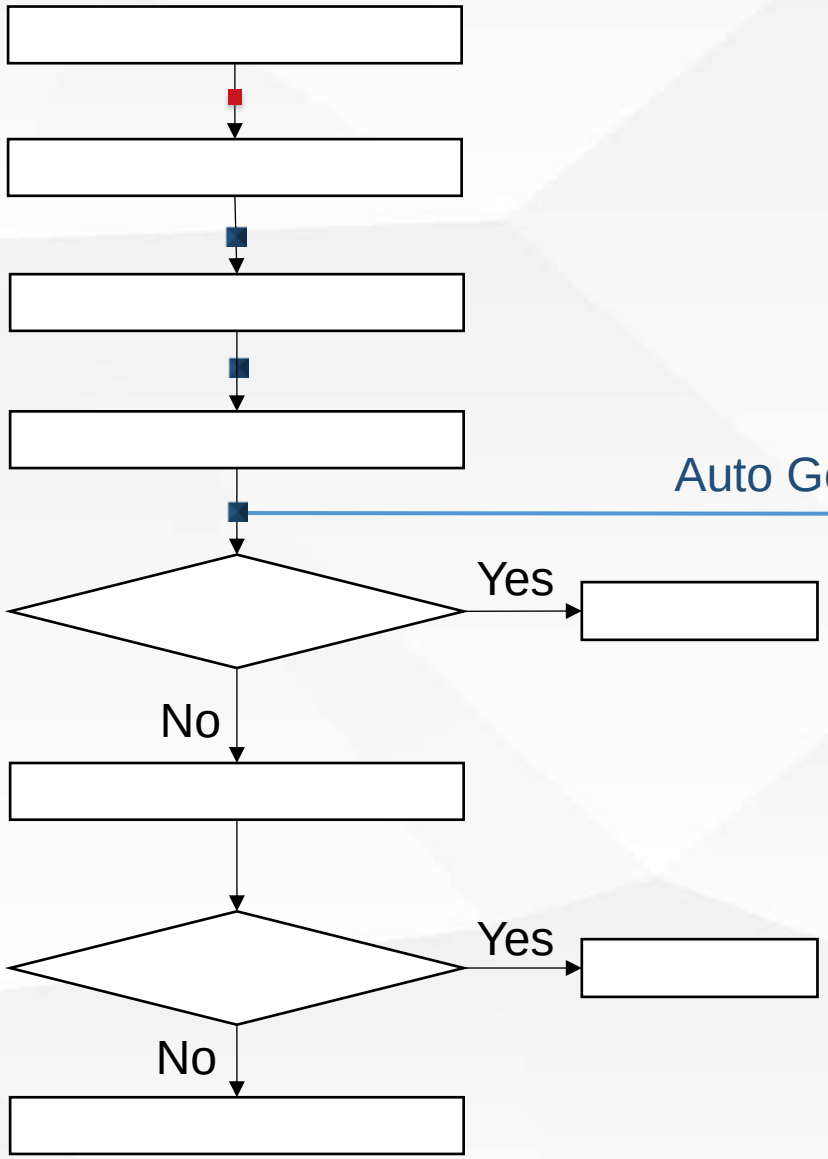
Example -- VEP-C



Example -- VEP-C



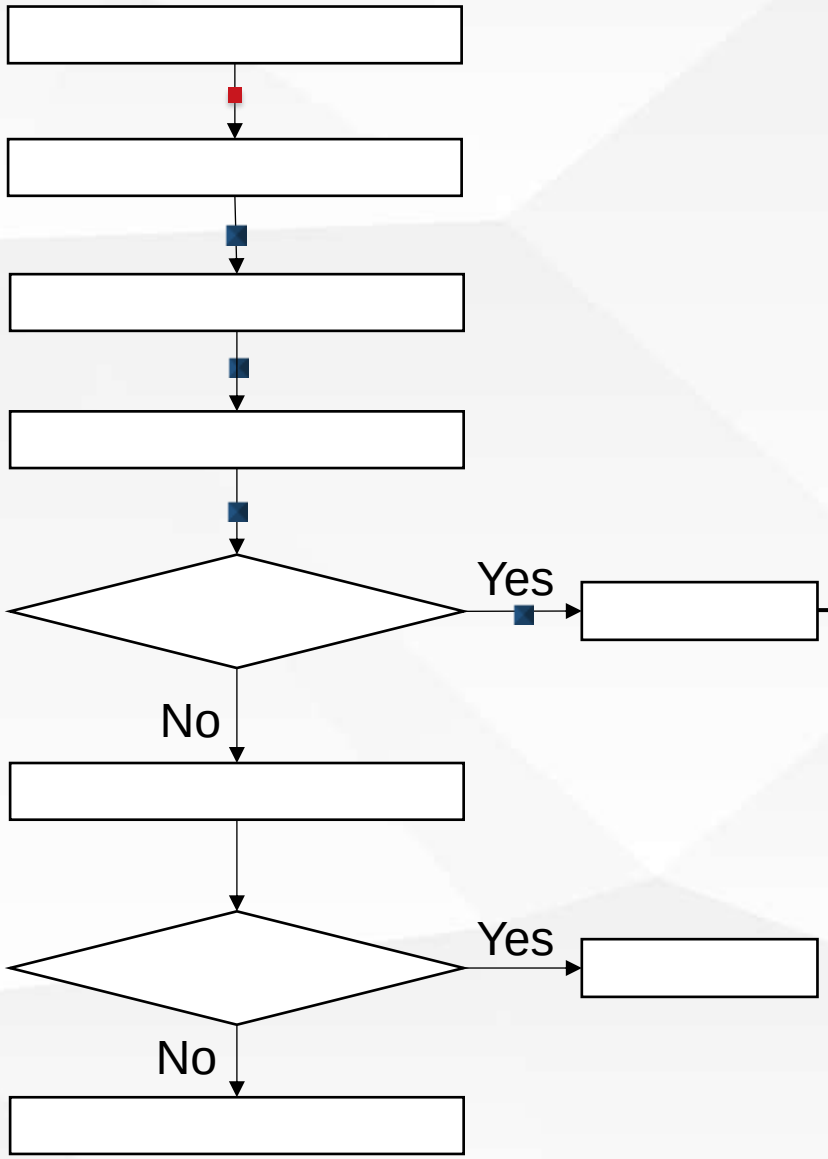
Example -- VEP-C



Auto Generated

```
data_end == xdp -> data_end &&
data == xdp -> data &&
eth == data &&
store_chars(xdp->data, xdp->data_end)
```

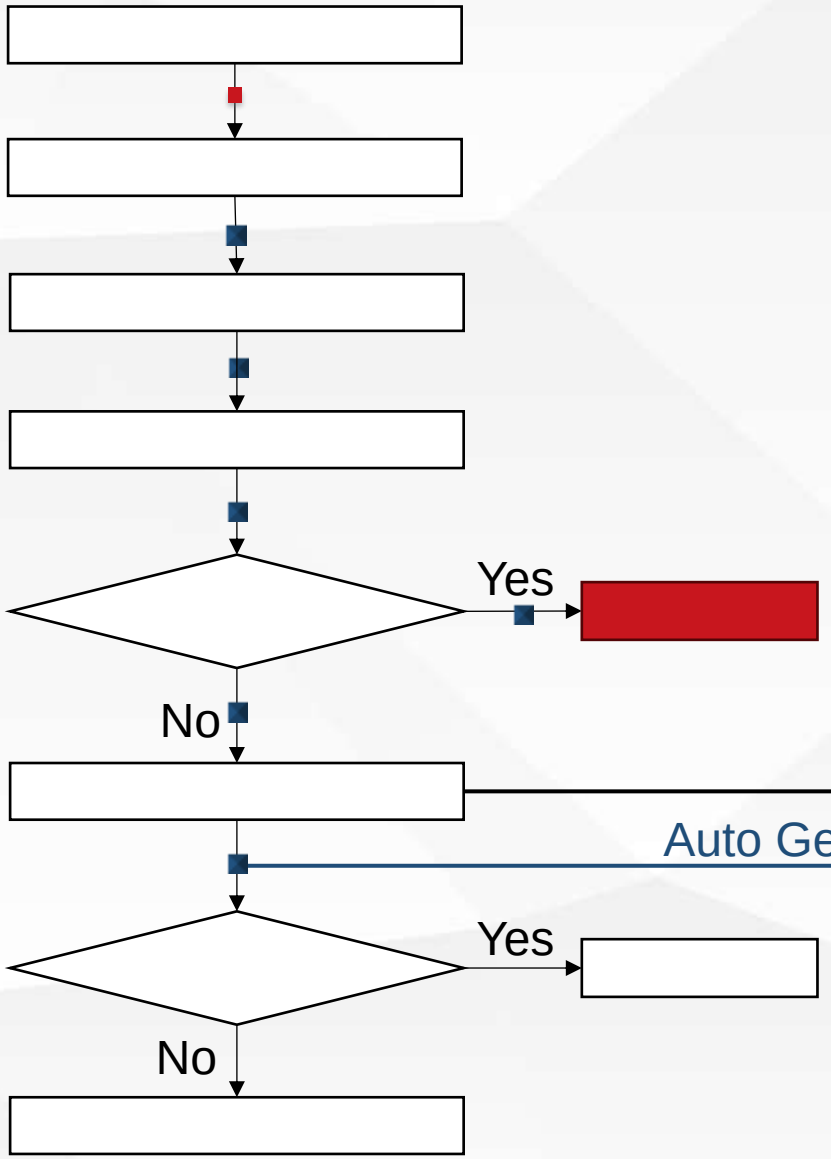
Example -- VEP-C



Auto Check

```
return 1 :

data_end == xdp -> data_end &&
data == xdp -> data &&
eth == data &&
eth + 1 > data_end &&
store_chars(xdp->data, xdp->data_end)
|--
  store_xdp(xdp)
```

Example -- VEP-C

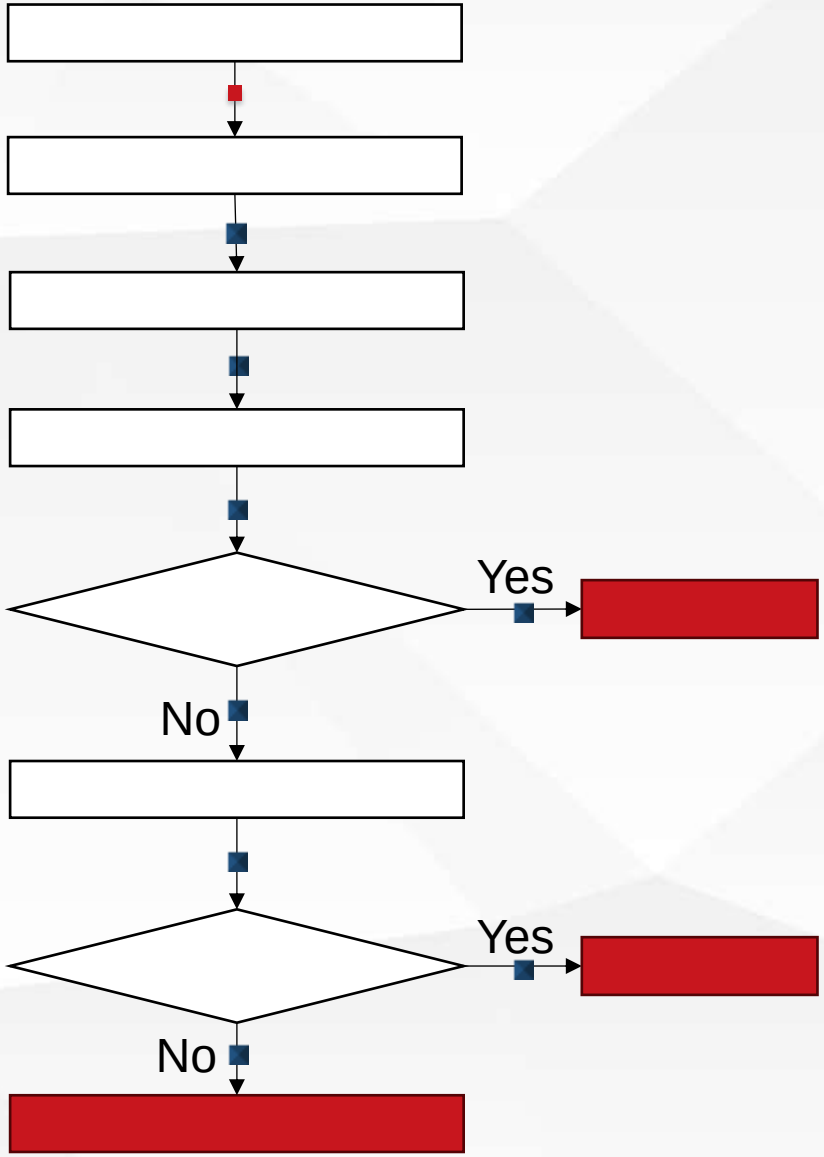


h_proto = eth -> h_proto

Auto Generated

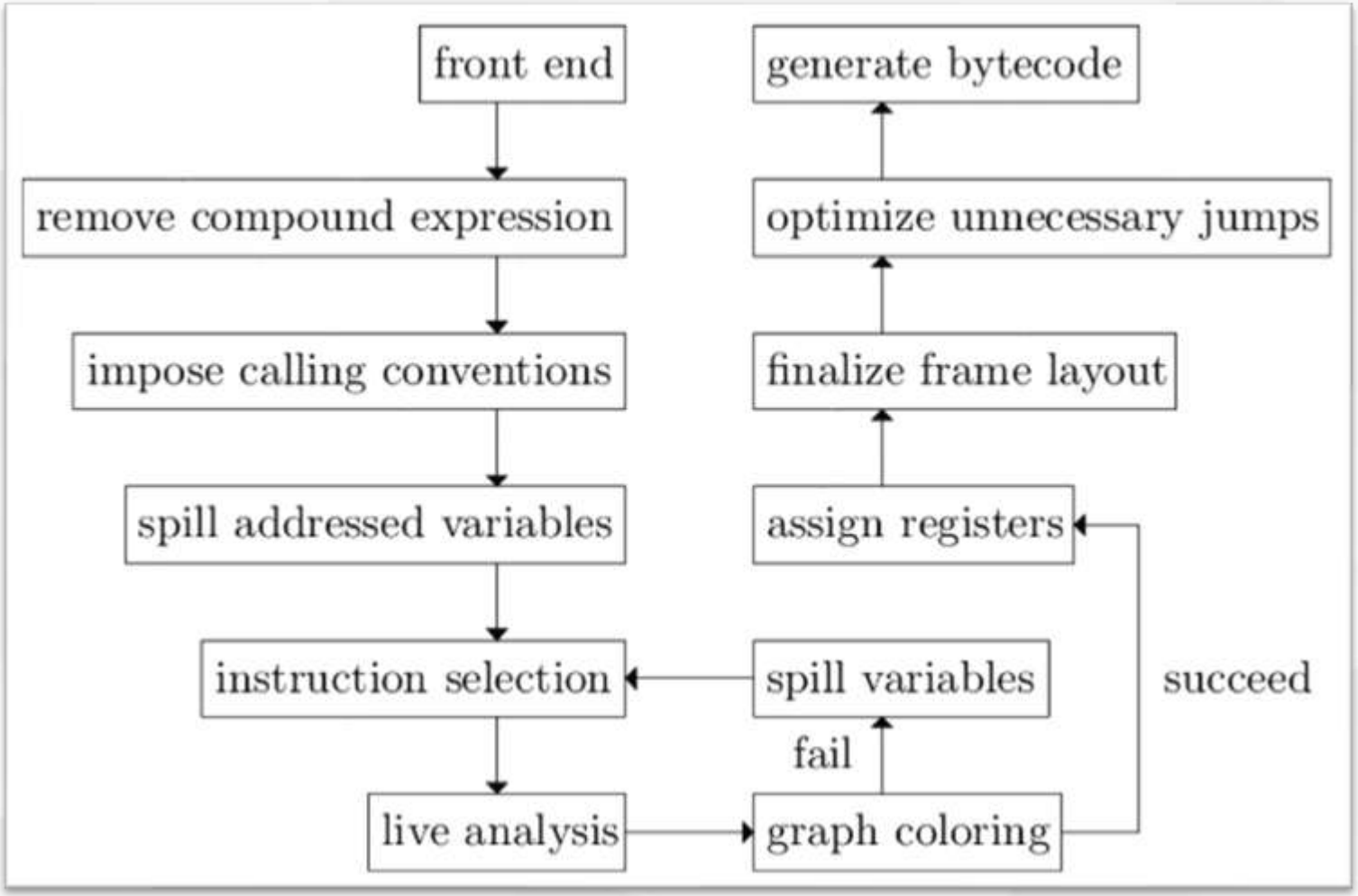
```
data_end == xdp -> data_end &&
data == xdp -> data &&
eth == data &&
eth + 1 <= data_end &&
h_proto == eth -> h_proto &&
store_uchars(eth->h_dest, eth->h_dest + 6) **
store_uchars(eth->h_source, eth->h_source + 6) **
store_chars(eth_end, xdp->data_end)
```

Example– heavily annotated C



VEP-compiler

VEP-compiler

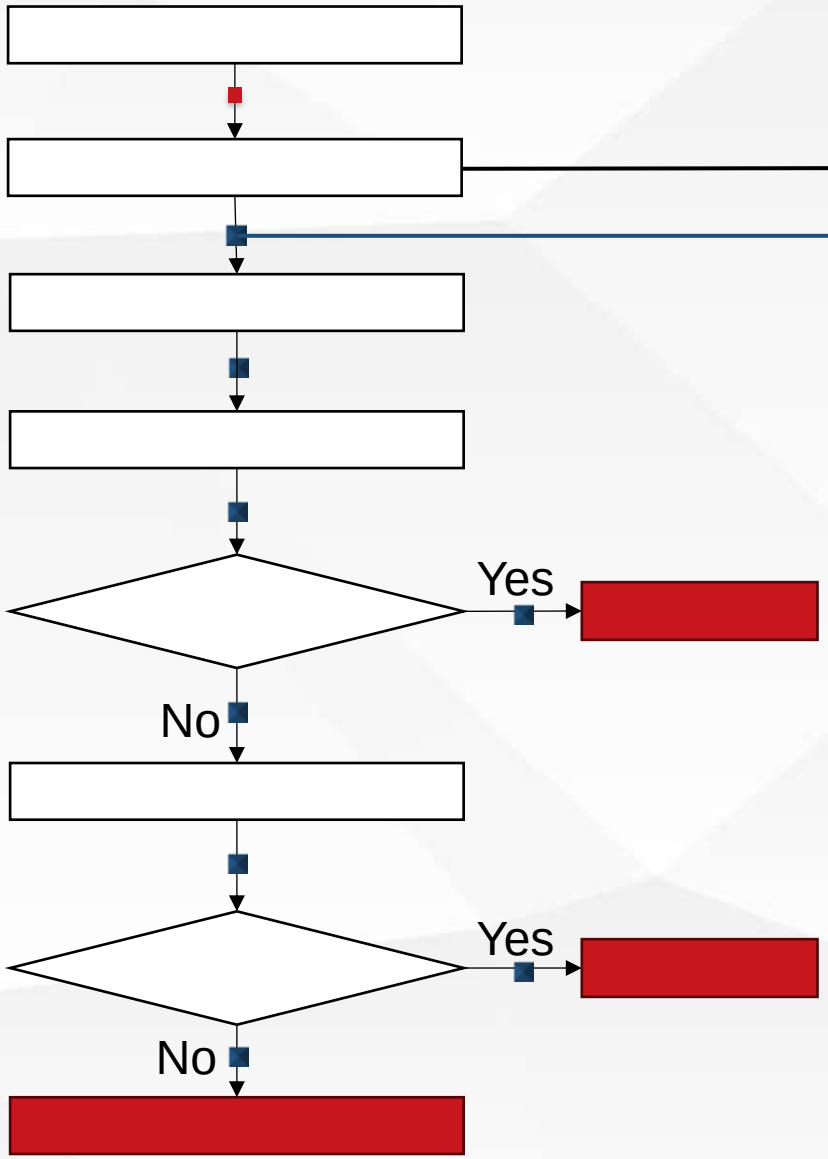


VEP-compiler

```
void strncpy (char *p1, char *p2, __u32 n)
/*@ With l1 l2
    Require chars(p1,n,l1) * chars(p2,n,l2)
    Ensure ∃ l3,
           chars(p1,n,l3) * chars(p2,n,l2) */;
```

```
strncpy:
/*@ With l1 l2 _R1 _R2 _R3 _R6 _R7 _R8 _R9
    Require
      _R1 == R1 &&
      _R2 == R2 &&
      _R3 == R3 &&
      _R6 == R6 && _R7 == R7 &&
      _R8 == R8 && _R9 == R9 &&
      chars(R1, R3, l1) * chars(R2, R3, l2)
    Ensure
      _R6 == R6 && _R7 == R7 &&
      _R8 == R8 && _R9 == R9 &&
      chars(_R1,_R3,l2) * chars(_R2,_R3,l2) */
```

Example– heavily annotated bytecode



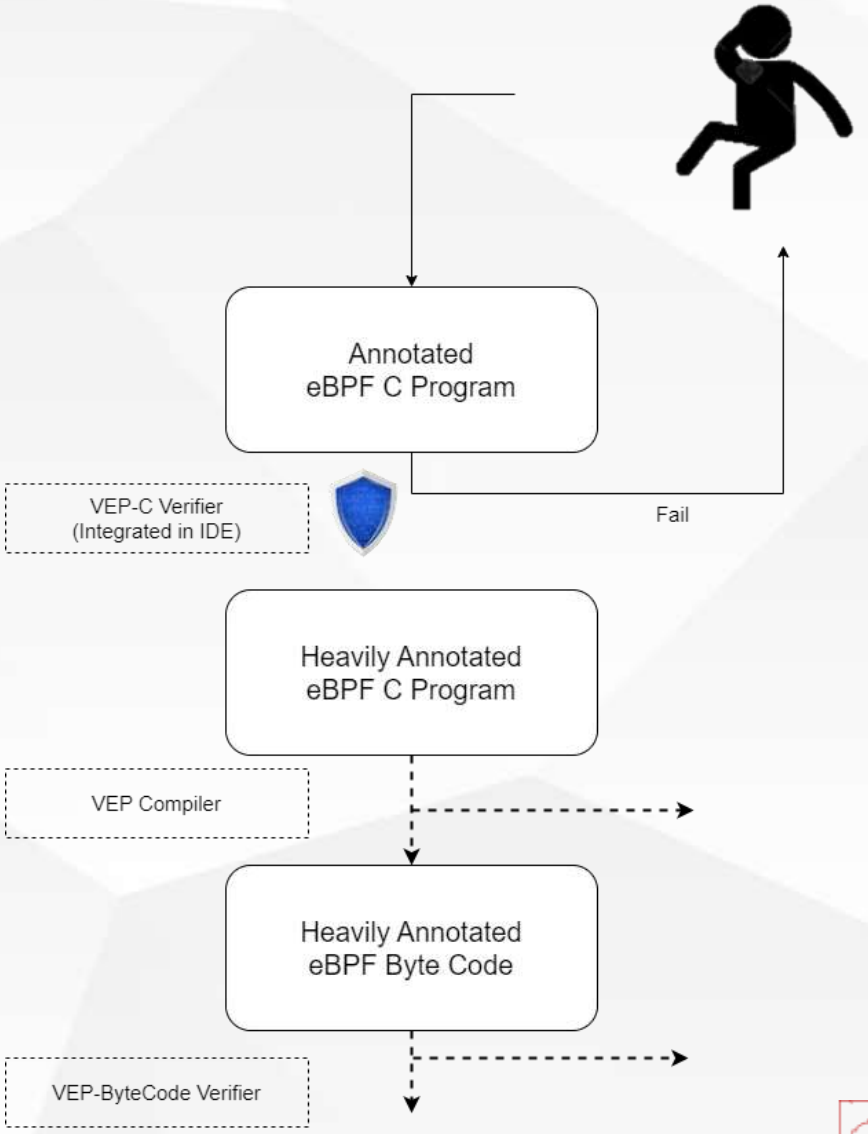

```
/* data_end = xdp -> data_end */
r0 = *(u32 *)(r1 + 4)
w4 = w0
r4 <<= 32
r4 >>= 32
```

```
r0 == *(r1+4) &&
r4 == low32(r0) && … &&
store_chars(*r1, *(r1+4)) * …
```

Discussion

Discussion

How to find bugs in development ?

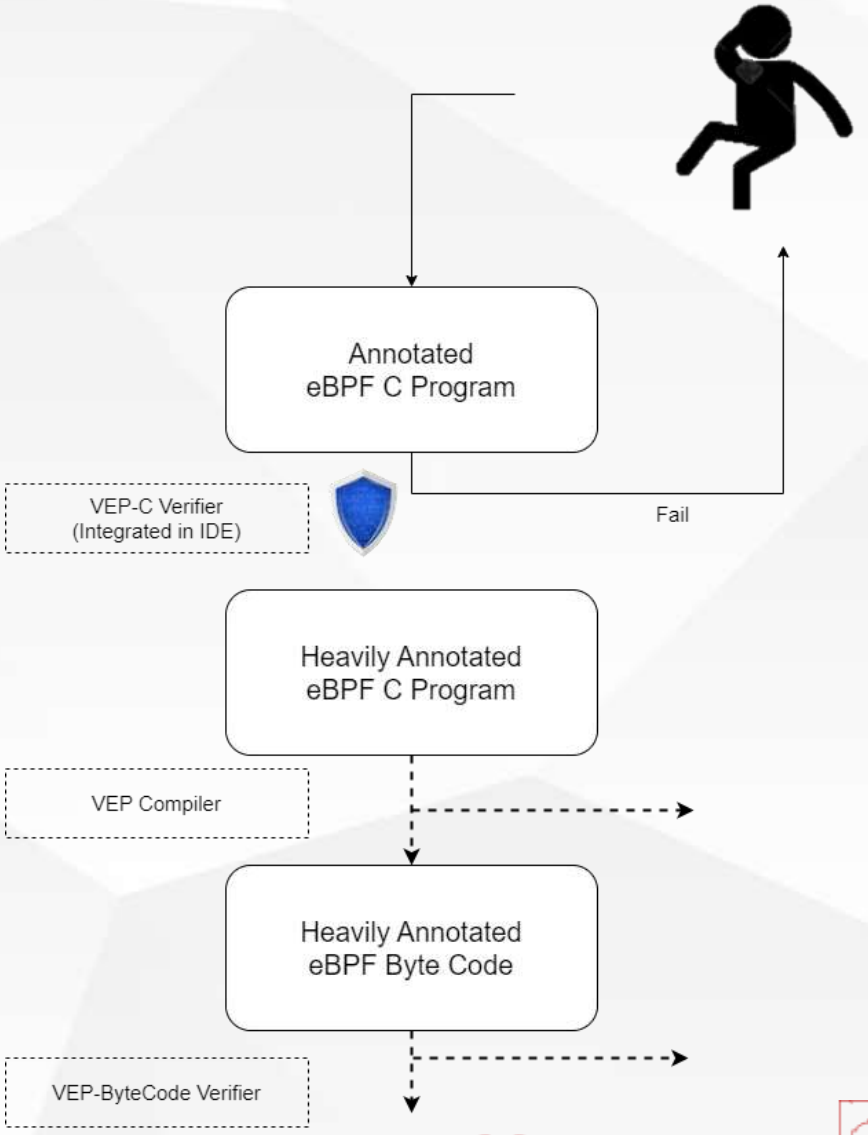


Discussion

How to find bugs in development ?

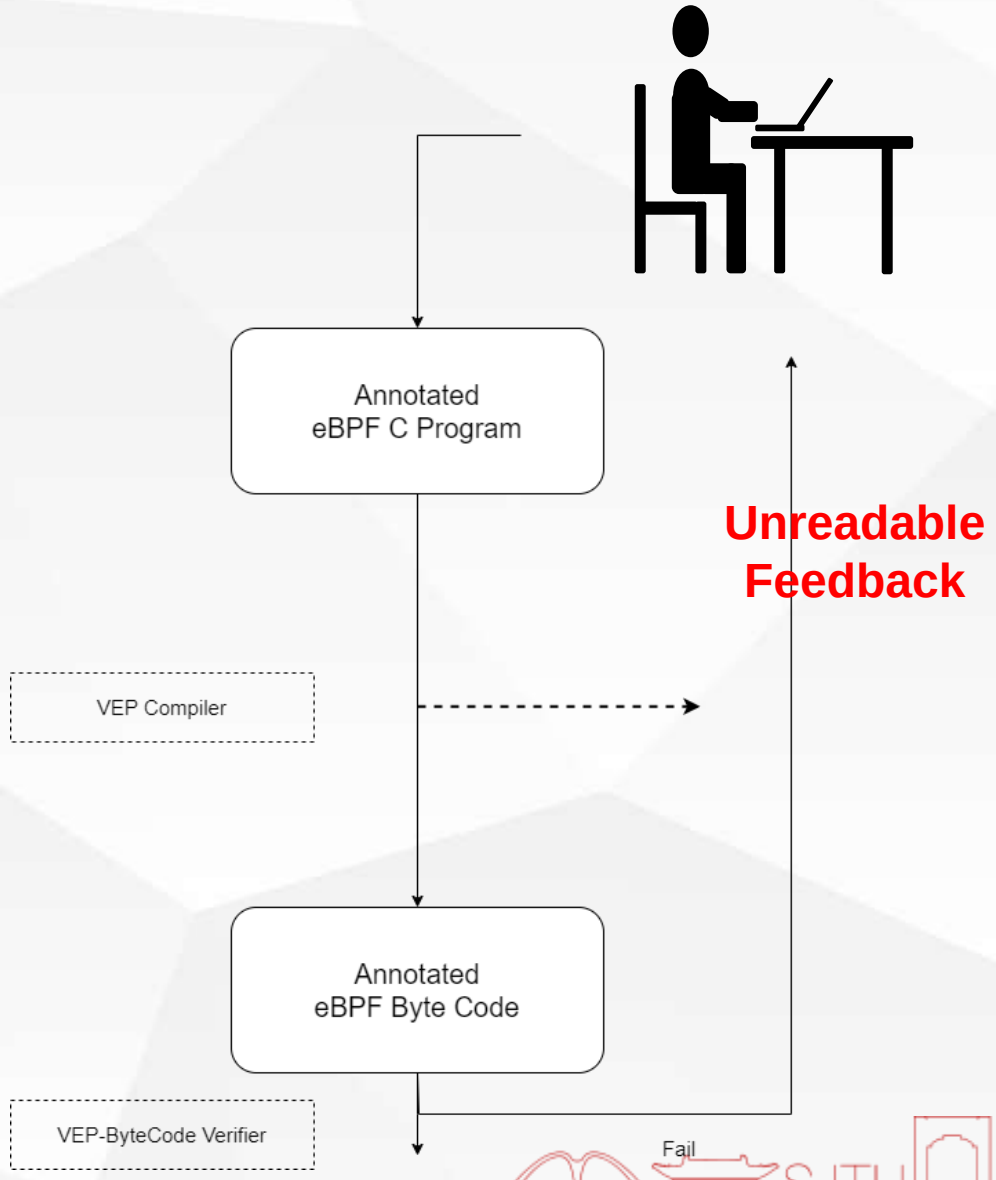
```
17  struct list * sll_copy(struct list * x)
18  /*@ Require listrep(x)
19      Ensure  listrep(__return) * listrep(x)
20   */
21  {
22      struct list *y, *p, *t;
23      y = malloc_list(0);
24      t = y;
25      p = x -> next;
```

```
Error: fatal error: Cannot derive the precondition of Memory Read.

Line: 25
```



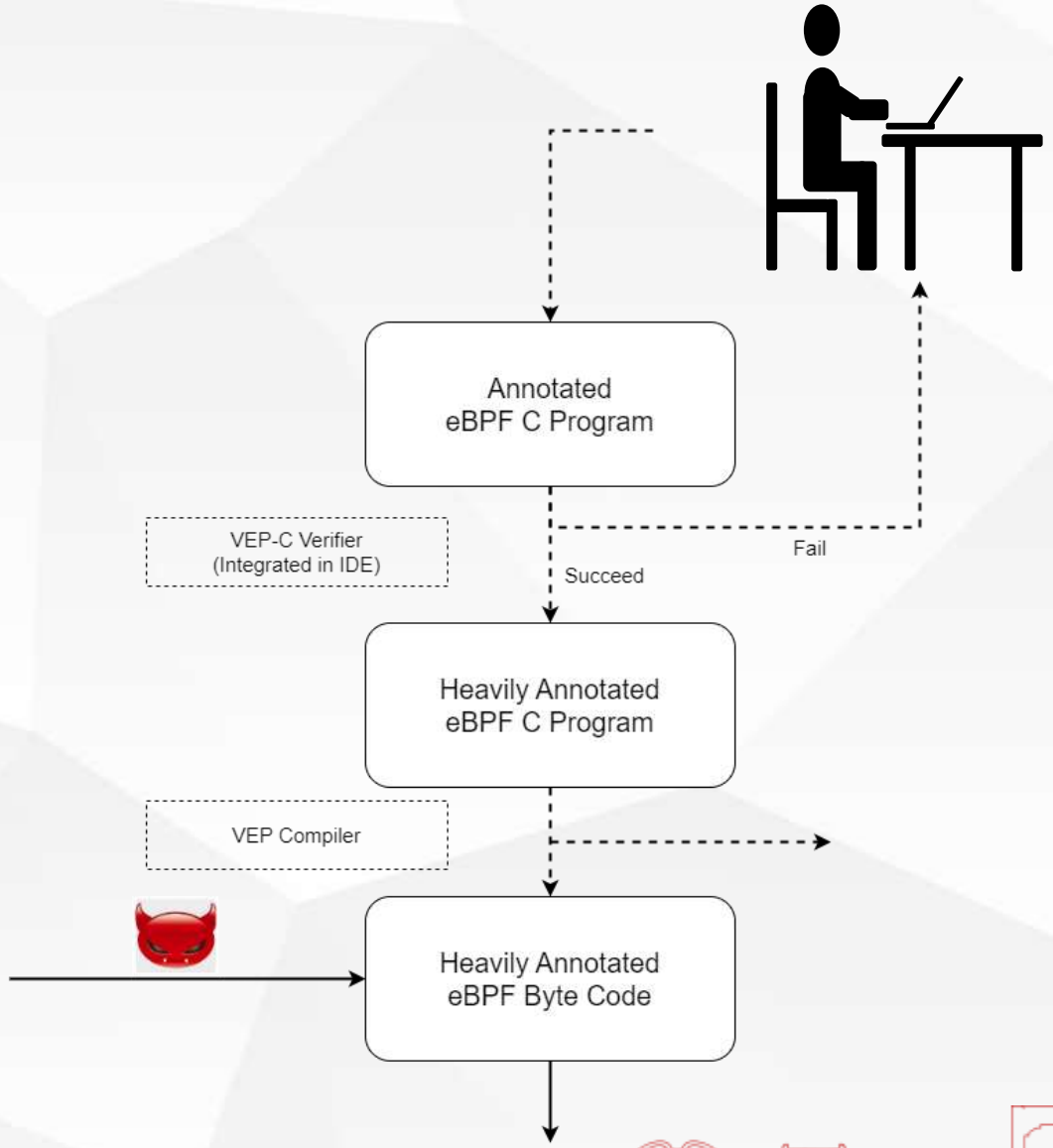
Discussion

- How to find bugs in development ?
- Why not use only bytecode verifier ?



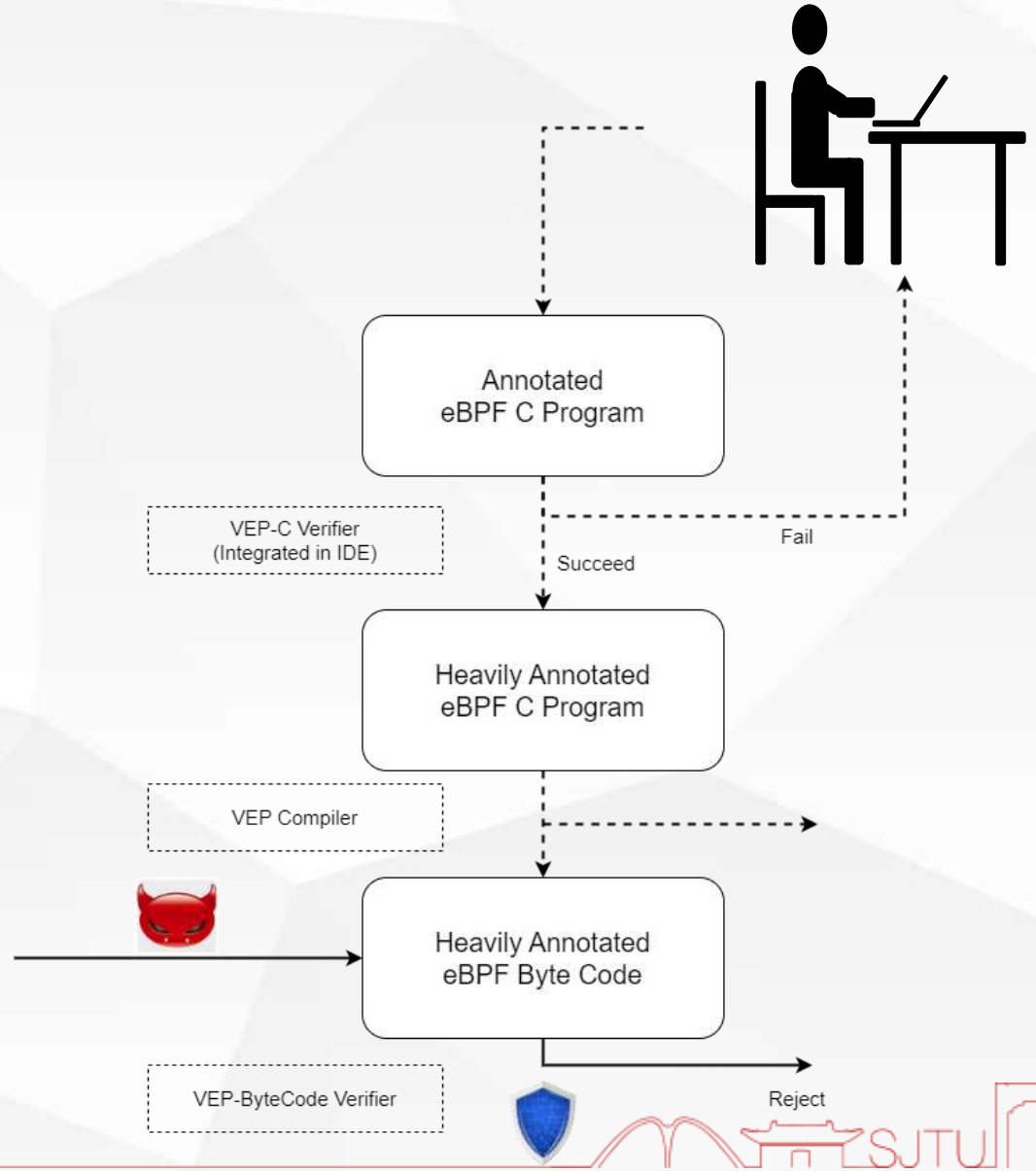
Discussion

- How to find bugs in development ?
- Why not use only bytecode verifier ?
- Why not use only C verifier ?



Discussion

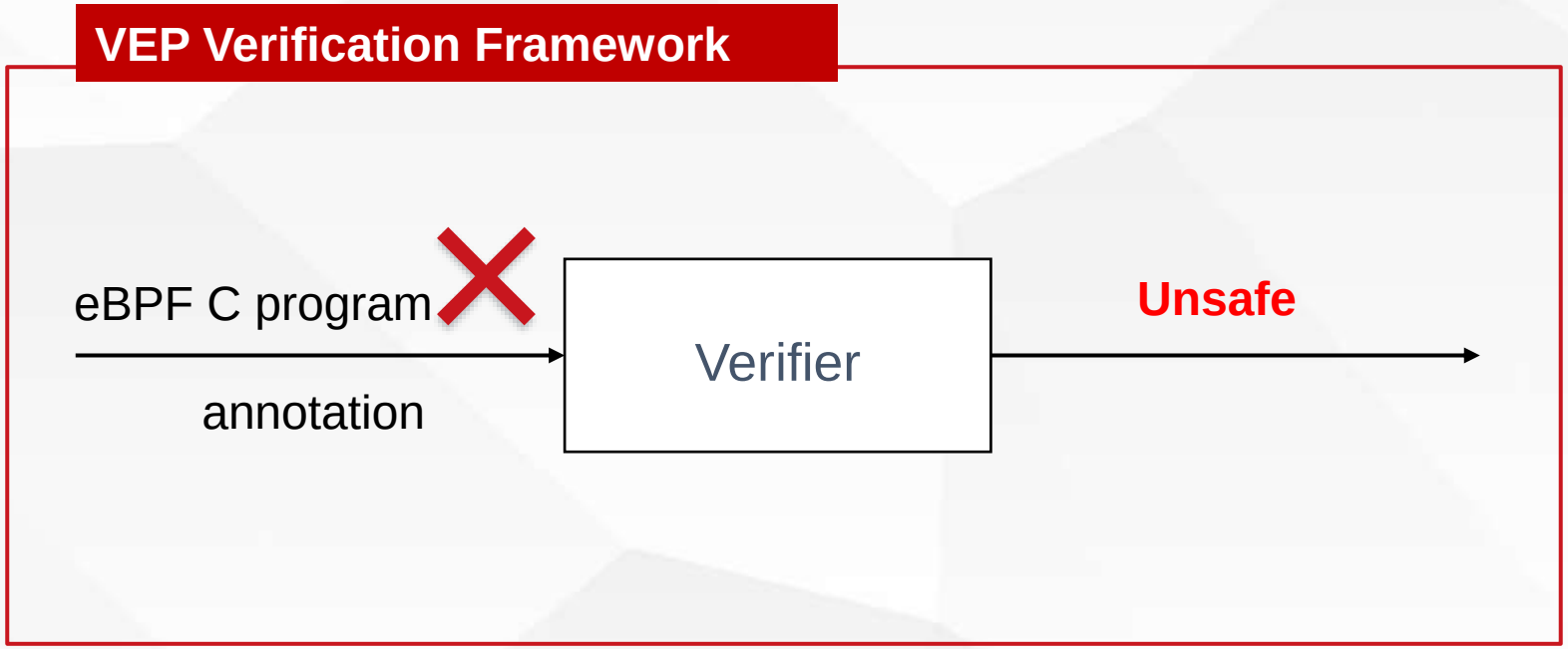
- How to find bugs in development ?
- Why not use only bytecode verifier ?
- Why not use only C verifier ?
- How does VEP defend against attacks ?



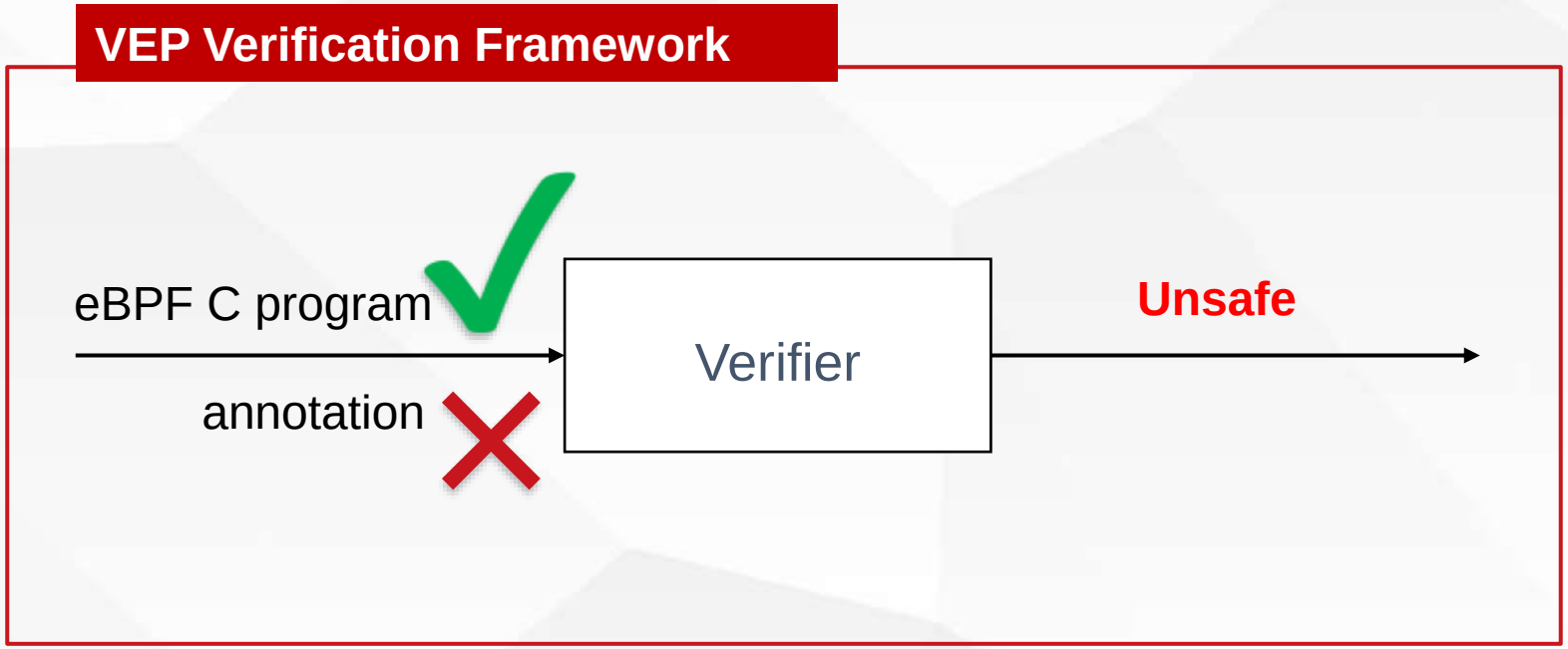
Discussion

- How to find bugs in development ?
- Why not use only bytecode verifier ?
- Why not use only C verifier ?
- How does VEP defend against attacks ?
- False positive & False negative ?

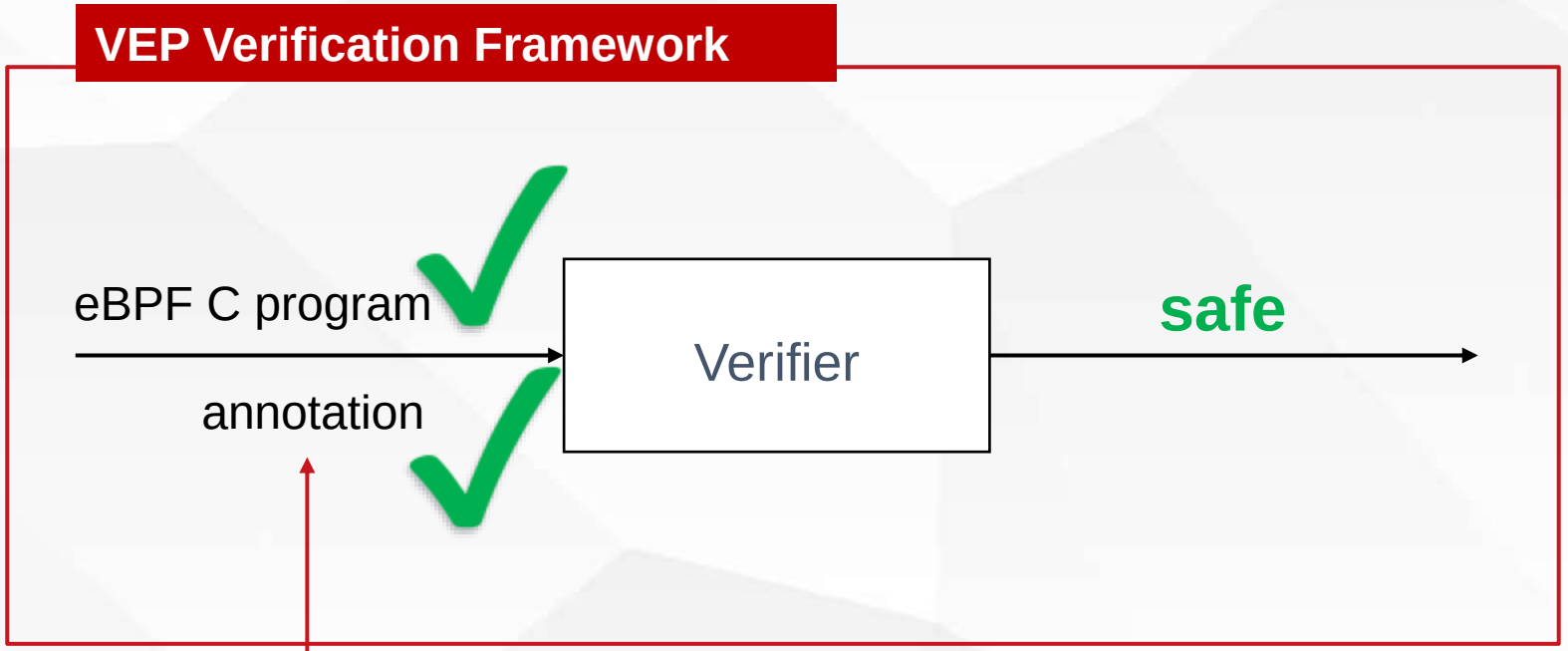
Kernel verifier choice



Kernel verifier choice



Kernel verifier choice



Hoare logic and separation logic research promise the existence[1]

[1]Hoare C A R. An axiomatic basis for computer programming[J]. Communications of the ACM, 1969, 12(10): 576-580.

Evaluation

Evaluation

Programs		Code Lines	Linux verifier		PREVAIL	
			Time(ms)	Memory(KB)	Time(ms)	Memory(KB)
Linux Samples	sockex1_kern	29	1.03	4194	2.05	4978
	syscall_tp_kern(enter)	38	1.03	4194	3.17	4931
	cpustat_kern(frequency)	93	1.13	4190	11.30	6377
	cpustat_kern(idle)	116	1.11	4196	23.52	7918
	xdp_adjust_tail_kern	47	0.64	4196	6.69	5539
	syscall_tp_kern(exit)	35	0.99	4190	3.02	4927
	lathist_kern(on)	77	1.09	4144	10.18	6025
	trace_event_kern	65	fail	-	48.37	6730
	tcp_iw_kern	68	0.65	4152	9.82	5664
	tcp_rwnd_kern	50	0.77	4155	6.69	5404
PREVAIL Samples	twomaps	34	fail	-	2.43	5056
	twotypes	33	fail	-	3.80	5278
	map_in_map	36	fail	-	5.17	5054
	stackok	13	1.02	4114	74.67	5279
	loop	21	fail	-	fail	-
	packet_start_ok	14	1.08	4200	1.19	4942
	twostackvars	47	0.53	4140	20.9	5267
	packet_access	28	0.58	4153	2.74	5216
	bpf2bpf	13	0.51	4154	0.16	4157
	dependent_read	13	0.55	4154	fail	-

Evaluation

Programs		Assertion Lines	Proof Lines	VEP-C		VEP-compiler	VEP-eBPF	
				Time(ms)	Memory(KB)	Time(ms)	Time(ms)	Memory(KB)
Linux Samples	sockex1_kern	3	1140	4.06	5882	0.35	2.53	2284
	syscall_tp_kern(enter)	7	2253	5.33	6391	0.37	2.57	2396
	cpustat_kern(frequency)	11	8357	50.33	15887	2.62	7.89	4292
	cpustat_kern(idle)	11	19390	158.12	32569	6.09	21.32	7167
	xdp_adjust_tail_kern	10	739	4.27	5852	0.32	2.41	2236
	syscall_tp_kern(exit)	7	2253	5.28	6396	0.35	2.47	2379
	lathist_kern(on)	9	4180	23.47	18502	2.47	21.69	8034
	trace_event_kern	6	7136	67.33	18841	1.15	5.79	3353
	tcp_iw_kern	6	161	12.57	12150	1.75	9.11	4182
	tcp_rwnd_kern	6	19231	63.88	19154	1.84	8.44	4342
PREVAIL Samples	twomaps	2	2310	6.07	7119	0.46	2.77	2521
	twotypes	4	9449	15.49	8288	0.73	4.40	2665
	map_in_map	3	261	2.97	5995	0.39	2.31	2348
	stackok	4	1848	4.32	5223	0.19	1.96	2028
	loop	8	4042	10.02	7682	0.55	2.84	2453
	packet_start_ok	3	1245	2.64	5174	0.21	2.03	2126
	twostackvars	12	18446	40.23	12422	1.43	3.96	3047
	packet_access	3	3669	8.85	7075	0.50	3.42	2659
	bpf2bpf	7	13	0.70	4436	0.11	1.76	1932
	dependent_read	3	648	2.56	4920	0.19	2.14	2106

Evaluation

Programs		Code Lines	Linux verifier		PREVAIL	
			Time(ms)	Memory(KB)	Time(ms)	Memory(KB)
Stringlib	strcpy	34	fail	-	fail	-
	strncpy	34	1.65	4212	fail	-
	strcat	44	fail	-	fail	-
	strncat	43	fail	-	fail	-
	strlen	19	fail	-	fail	-
	strncmp	31	2.69	4316	fail	-
	strcmp	32	fail	-	fail	-
	memset	28	1.14	5168	39.89	7267
	strchr	28	fail	-	fail	-
	memchr	28	fail	-	fail	-
Unsafe Program	badhelpercall	6	reject	-	reject	-
	badmapptr	24	reject	-	reject	-
	badrelo	20	reject	-	reject	-
	ctxoffset	21	reject	-	reject	-
	nullmapref	23	reject	-	reject	-
	badhelpercall2	22	reject	-	reject	-
	packet_overflow	14	reject	-	reject	-
	wronghelper	20	reject	-	reject	-
	mapunderflow	23	reject	-	reject	-
	packet_reallocate	22	reject	-	reject	-
Key_connection		63	fail	-	fail	-

Evaluation

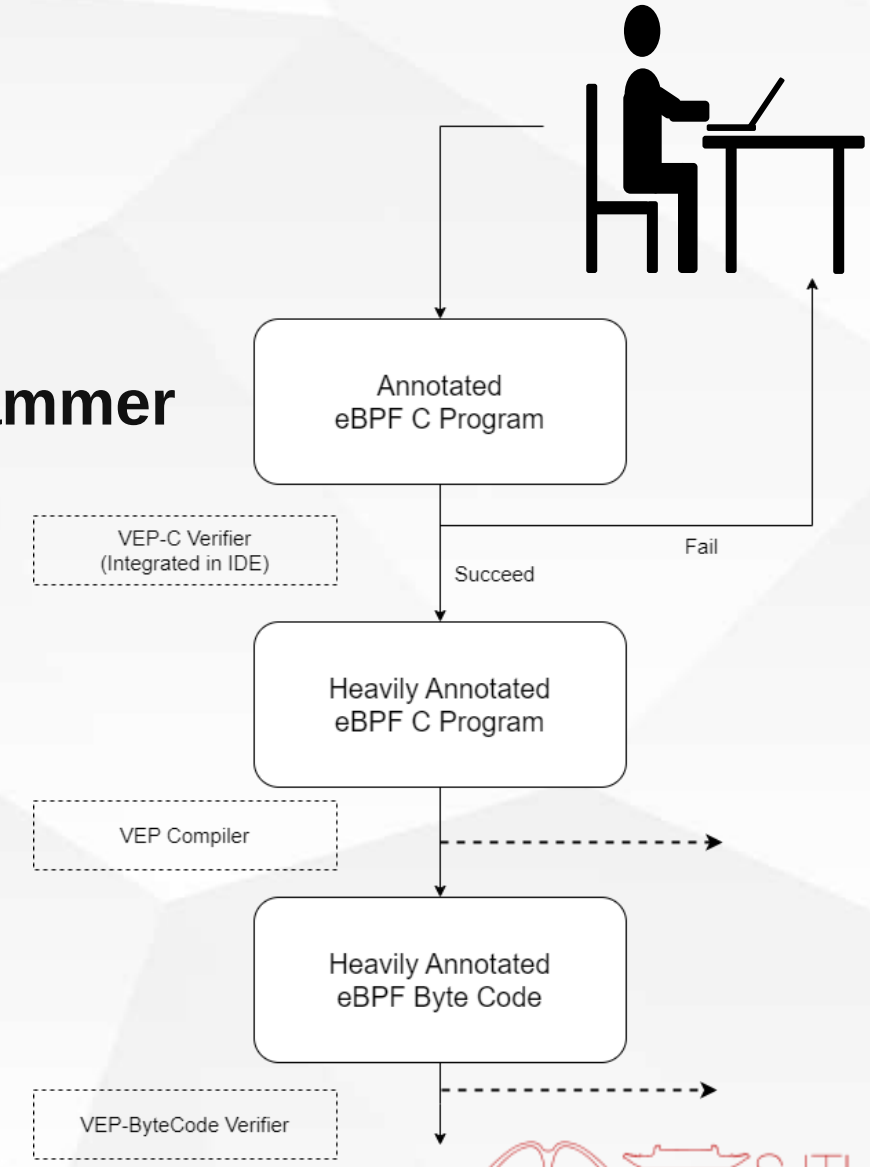
Programs		Assertion Lines	Proof Lines	VEP-C		VEP-compiler	VEP-eBPF	
				Time(ms)	Memory(KB)	Time(ms)	Time(ms)	Memory(KB)
Stringlib	strcpy	9	6051	12.09	8172	0.66	2.67	2510
	strncpy	11	4242	8.69	7888	0.59	2.61	2557
	strcat	17	12820	31.17	11778	1.39	3.41	2907
	strncat	17	14254	36.60	12612	1.16	3.54	2970
	strlen	9	2035	5.02	5946	0.29	2.23	2236
	strncmp	11	3976	8.45	7994	0.54	2.58	2470
	strcmp	9	5934	14.27	8528	0.76	2.25	2574
	memset	11	1925	3.78	6271	0.35	2.18	2281
	strchr	9	2329	6.94	6962	0.40	1.66	2463
	memchr	9	2551	7.64	6683	0.42	3.19	2517
Unsafe Program	badhelpercall	4	reject	2.82	1618	-	-	-
	badmapptr	3	reject	3.05	1701	-	-	-
	badrelo	3	reject	2.60	1575	-	-	-
	ctxoffset	3	reject	2.70	1436	-	-	-
	nullmapref	3	reject	4.38	2275	-	-	-
	badhelpercall2	4	reject	2.46	1402	-	-	-
	packet_overflow	3	reject	4.31	2115	-	-	-
	wronghelper	3	reject	2.73	1652	-	-	-
	mapunderflow	3	reject	2.78	1781	-	-	-
	packet_reallocate	3	reject	5.36	2955	-	-	-
Key_connection		17	5819	16.24	8534	0.56	2.48	2440

Evaluation

Programs	Total Assertion	Total Proofs	VEP-C				compiler	VEP-eBPF			Total Code
Source	Lines	Lines	PR	MaxT (ms)	AvgT (ms)	MaxM (KB)	AvgT (ms)	MaxT (ms)	AvgT (ms)	MaxM (KB)	Lines
Linux samples	76	64840	10/10	158.12	39.46	32569	1.73	21.69	8.42	8034	618
PREVAIL samples	49	41931	10/10	40.23	9.38	12422	0.47	4.40	2.76	3047	252
StringLib	112	56117	10/10	36.60	13.47	12612	0.66	3.54	2.63	2970	321
Unsafe Programs	32	-	10/10	5.36	3.32	2955	-	-	-	-	195
Key_Connection	17	5819	1/1	16.24	16.24	8534	0.56	2.48	2.48	2440	63

Conclusion

- P1: Verification power is not enough
- S: Require user provide additional annotations
- P2: Feedback is not friendly enough for programmer
- S: Two stage verification




Thanks
饮 水 思 源 爱 国 荣 校

Backup-strcmp

```
static inline int strcmp (char *p1, char *p2)
{
    //#pragma unroll
    unsigned int i;
    for (i = 0;  ; i++) {
        char c1 = p1[i];
        char c2 = p2[i];
        if (c2 == (char)0) break;
        if (c1 != c2) return 0;
        if (c1 == (char)0)
            break;
    }
    return 1;
}
```

Backup-strcmp

```
static inline int strcmp (char *p1, char *p2)
/*@ With n0 m0 l0 n1 m1 l1
    Require 0 <= n0 && l0[n0] == 0 && n0 < m0 &&
            0 <= n1 && l1[n1] == 0 && n1 < m1 &&
            chars(p1, m0, l0) * chars(p2, m1, l1)
    Ensure chars(p2, m1, l1) * chars(p1, m0, l0)
*/
{
    unsigned int i;
    /*@ Inv 0 <= i && i <= n0 && i <= n1 */
    for (i = 0;  ; i++) {
        char c1 = p1[i];
        char c2 = p2[i];
        if (c2 == (char)0) break;
        if (c1 != c2) return 0;
        if (c1 == (char)0)
            break;
    }
    return 1;
}
```