第三届 eBPF开发者大会

www.ebpftravel.com

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

上海交通大学 曹钦翔

（更新于2025年4月18日下午）

中国·西安

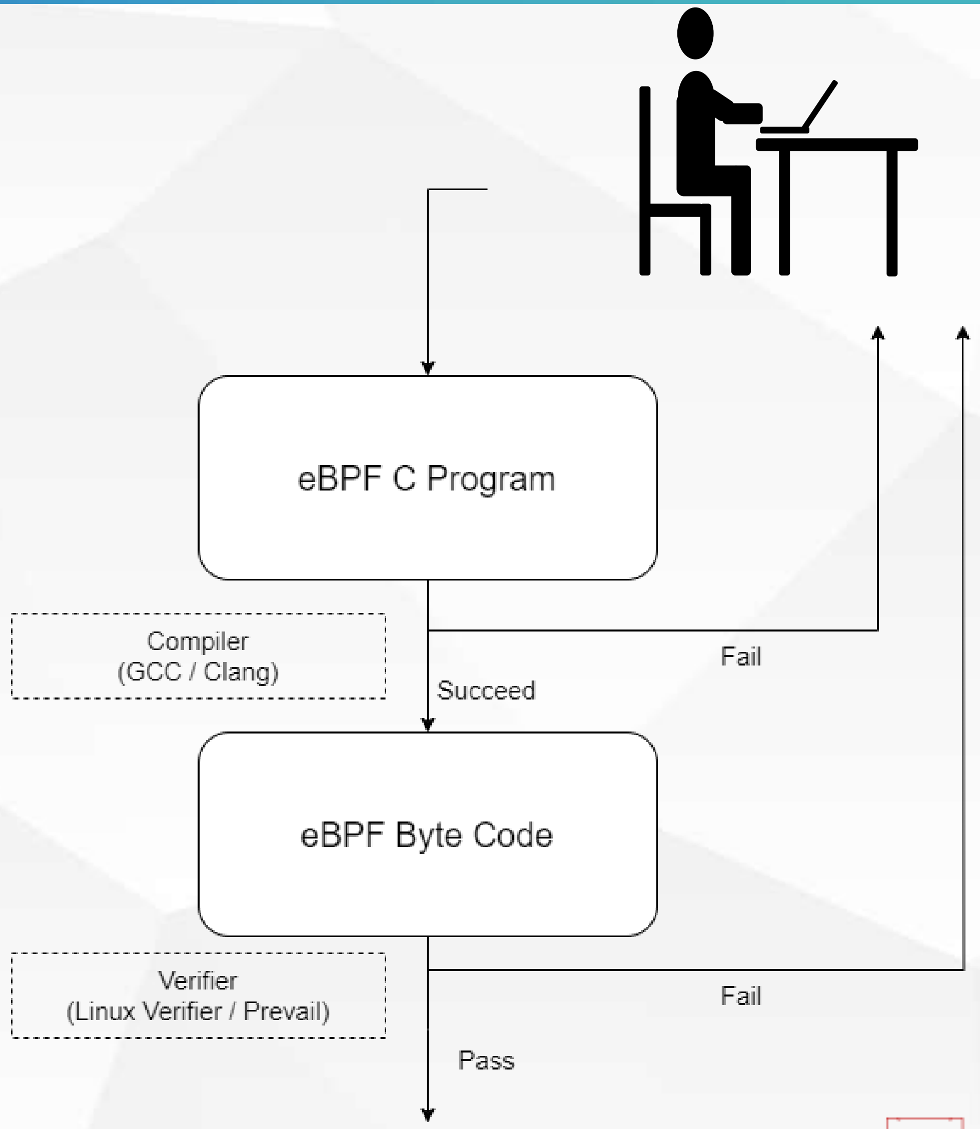
目录

① 背景介绍

背景介绍：

eBPF verifier带来的安全保障与开发挑战

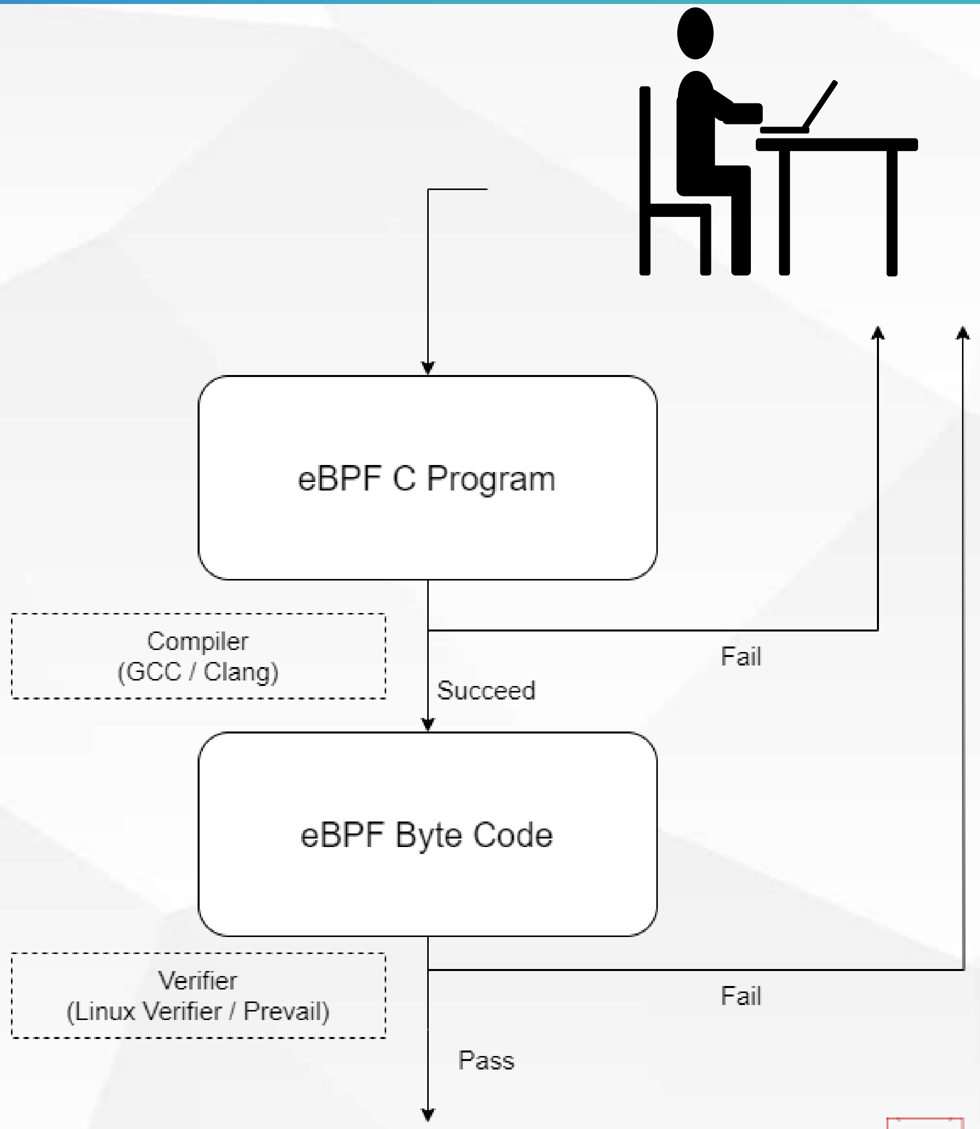
通过verifier排除不安全程序



背景介绍：

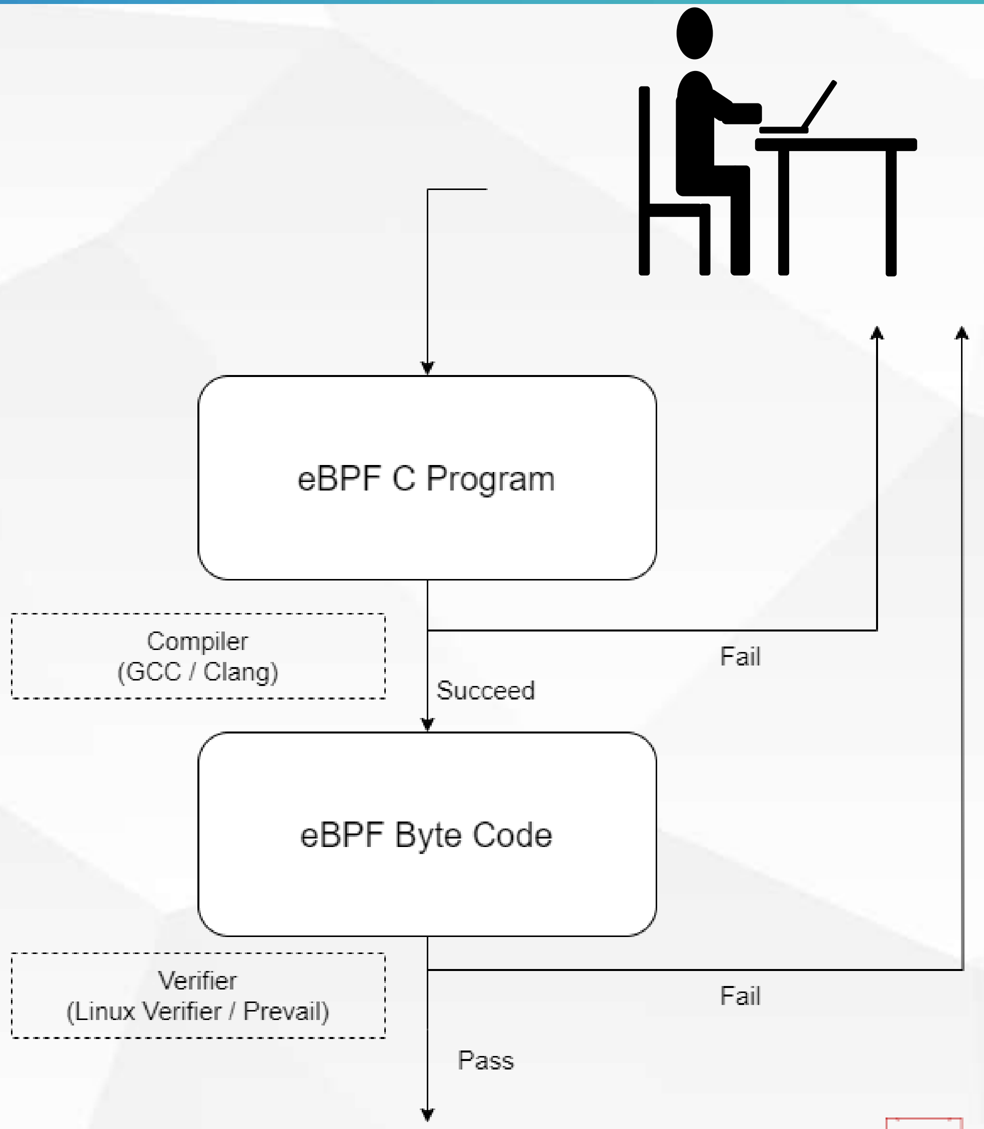
eBPF verifier带来的安全保障与开发挑战

通过verifier排除
verifier认为不安全的程序



背景介绍：
eBPF verifier带来的安全保障与开发挑战

	Linux Verifier	Prevail
程序分析技术	简单控制流图扫描	抽象解释
安全性验证能力	弱	中
对编程的限制	限制极大	限制较大
验证的资源消耗	小	大



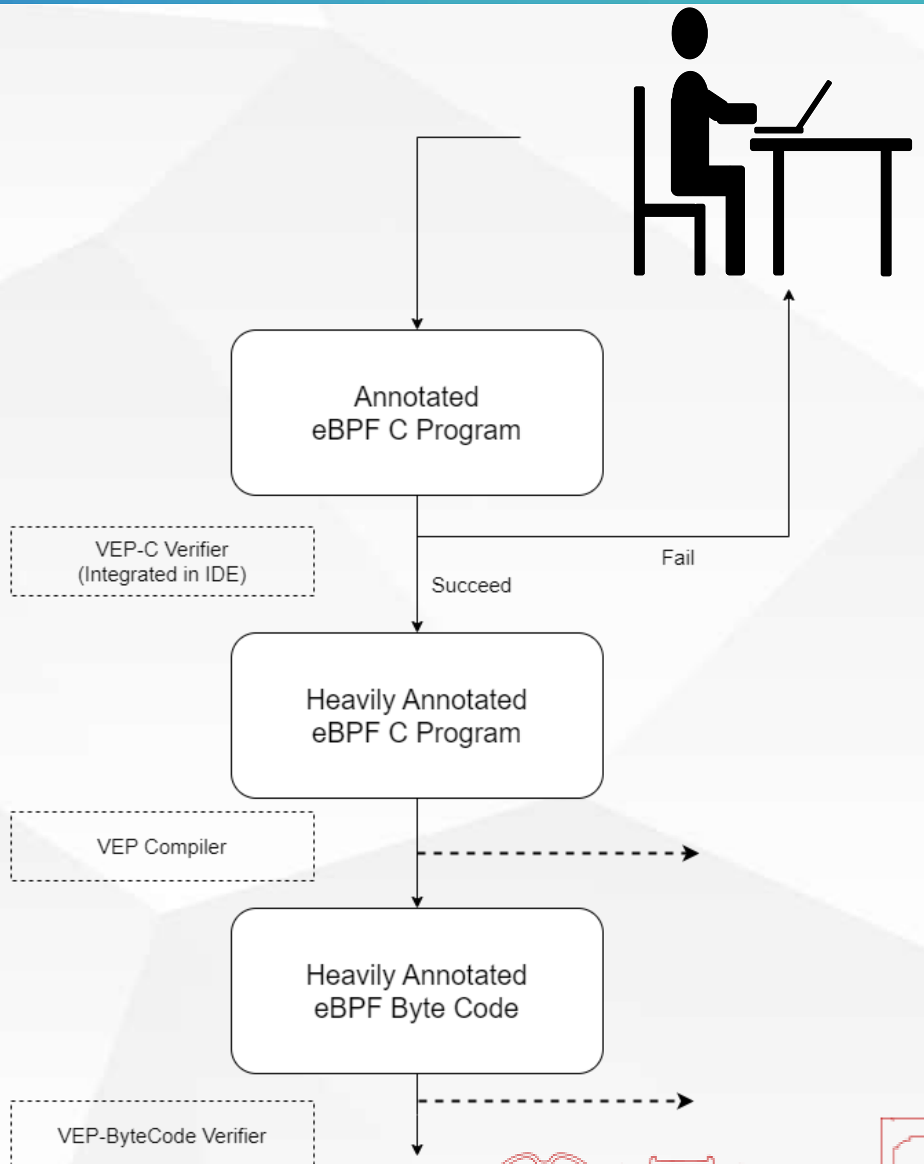
eBPF verifier带来的安全保障与开发挑战

1. 为了保证操作系统安全，verifier必须保证零漏报，但是允许误报
2. verifier的误报越多，对eBPF编程开发的限制就越大
3. 在verifier仅仅以程序为输入的前提下，要减少误报，就要引入复杂的验证算法，其所消耗的内存与验证时间也会大幅增加
4. 即使使用程序验证领域前沿成果，也无法在上述设定下根本解决verifier对eBPF编程开发限制较大的问题

② 技术框架

VEP技术框架

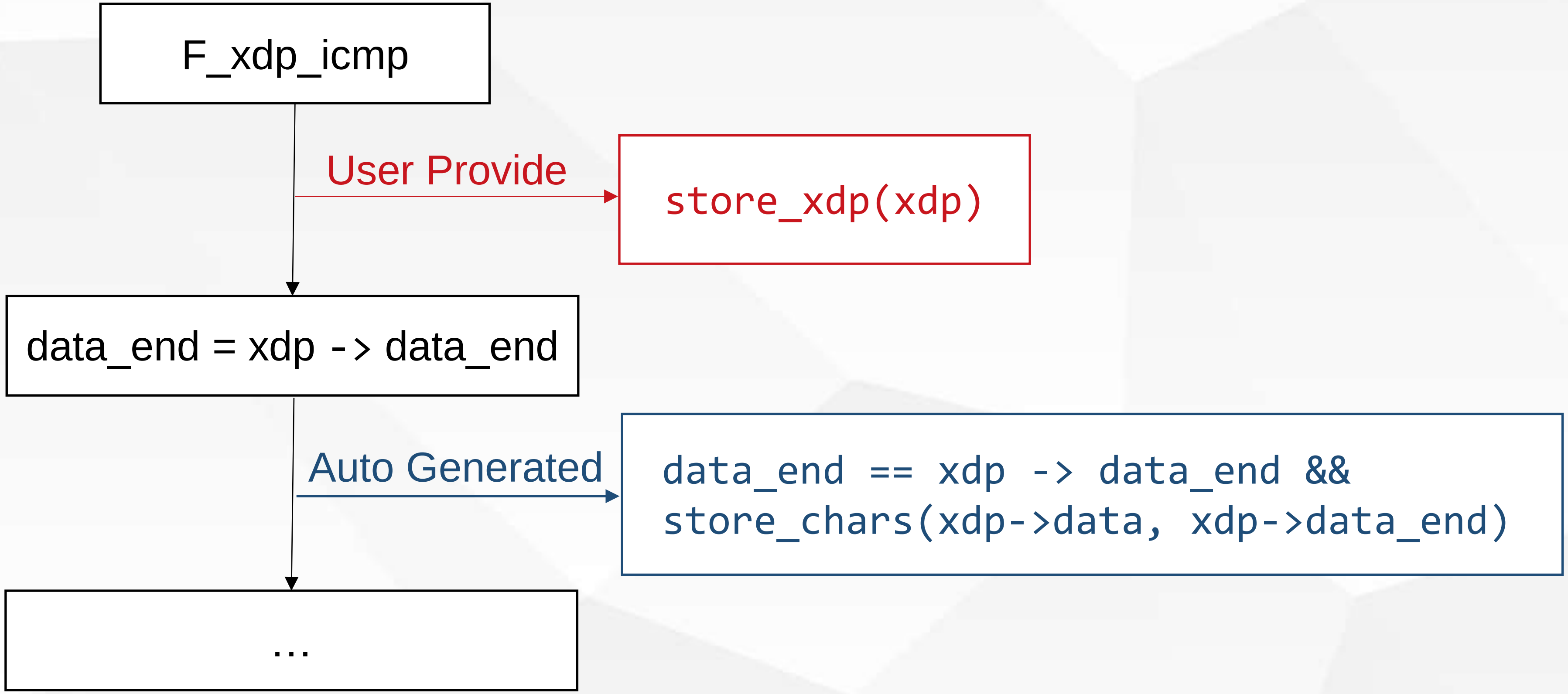
1. 验证能力不足以精准判断程序的安全性
   → 要求开发人员提供额外的断言标注
2. C程序验证（信息更丰富）or 字节码验证（可信基更小）
   → 采用两阶段验证方案



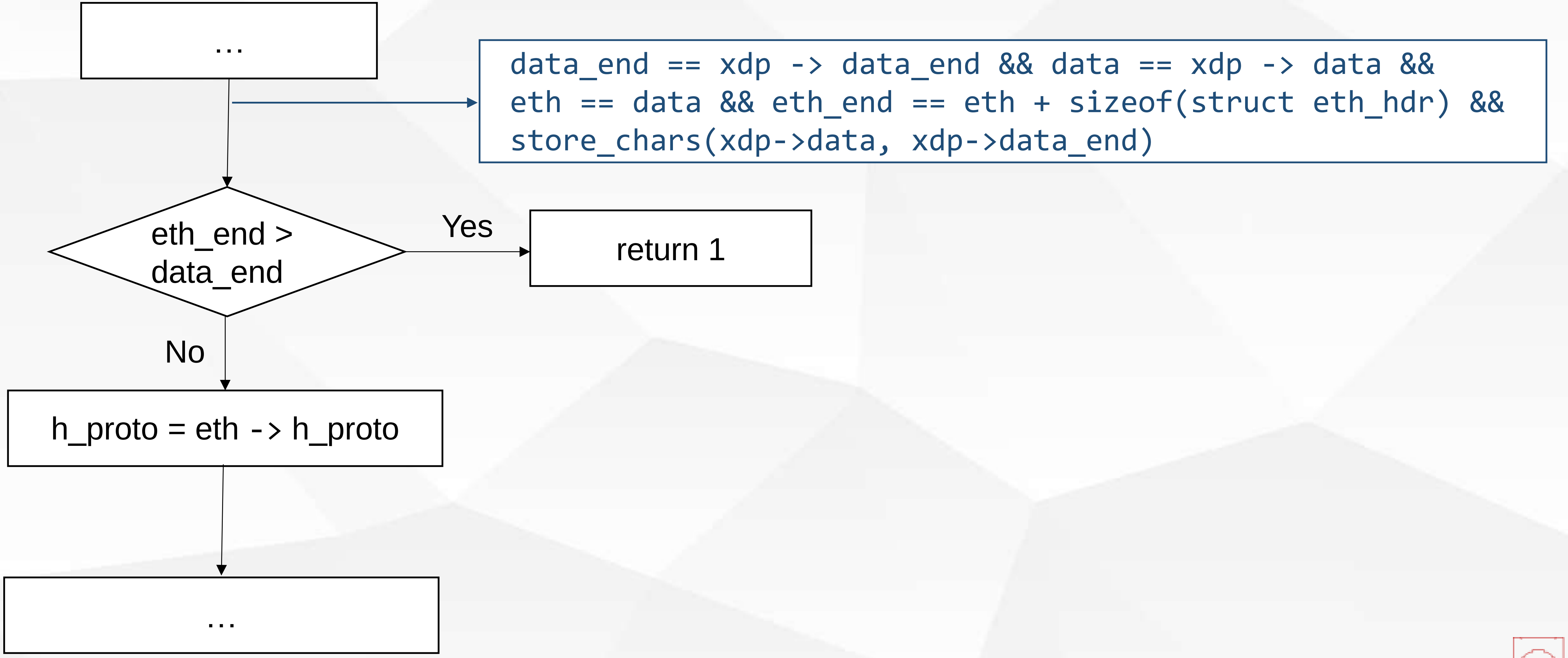
例子

```
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 (eth_hdr + 1 > data_end)
    return 1;
  unsigned int h_proto = eth -> h_proto;
  if (h_proto == htons(2048))
    return handle_ipv4(xdp);
  else
    return 2;
}
```

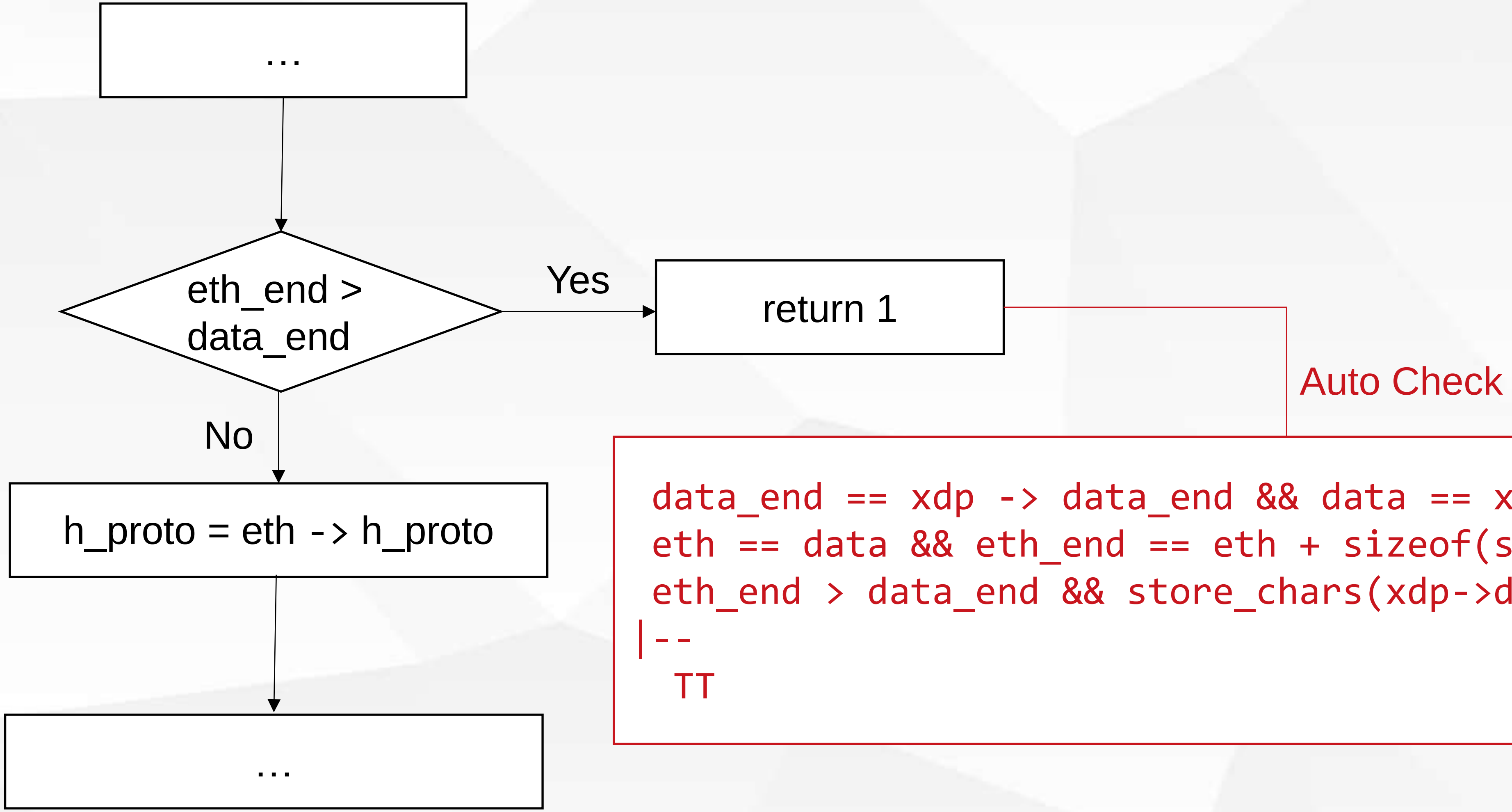
VEP技术框架 – VEP-C



VEP技术框架 – VEP-C

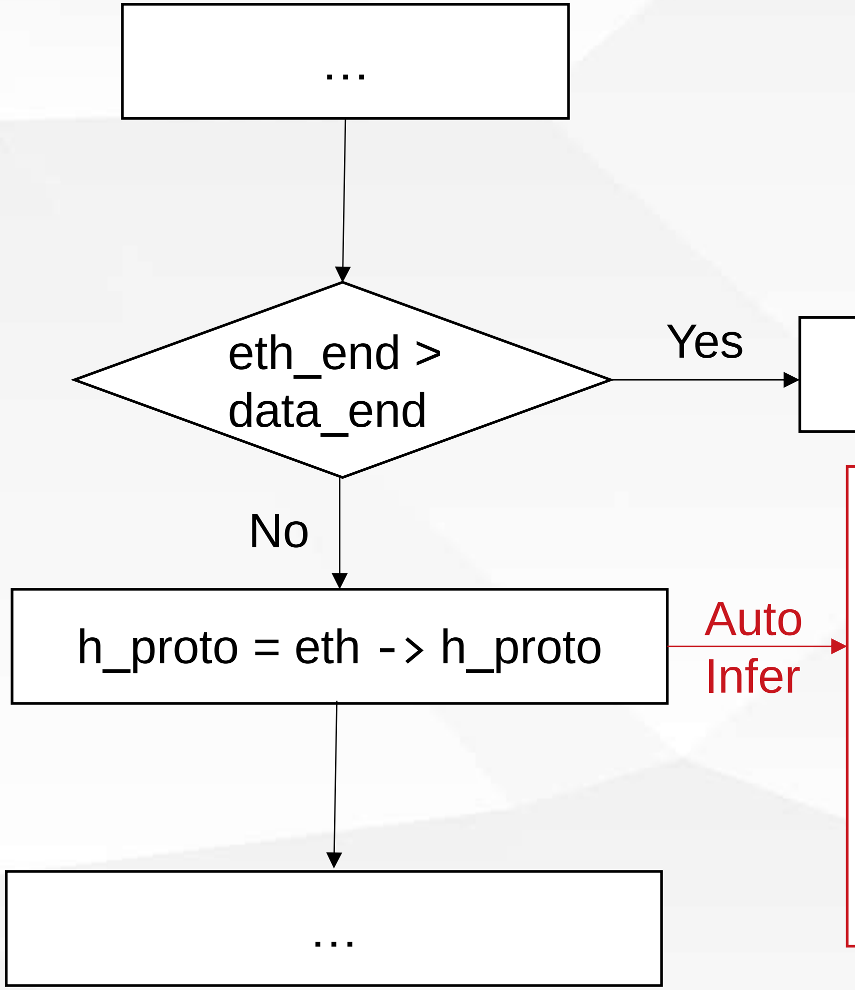


VEP技术框架 – VEP-C



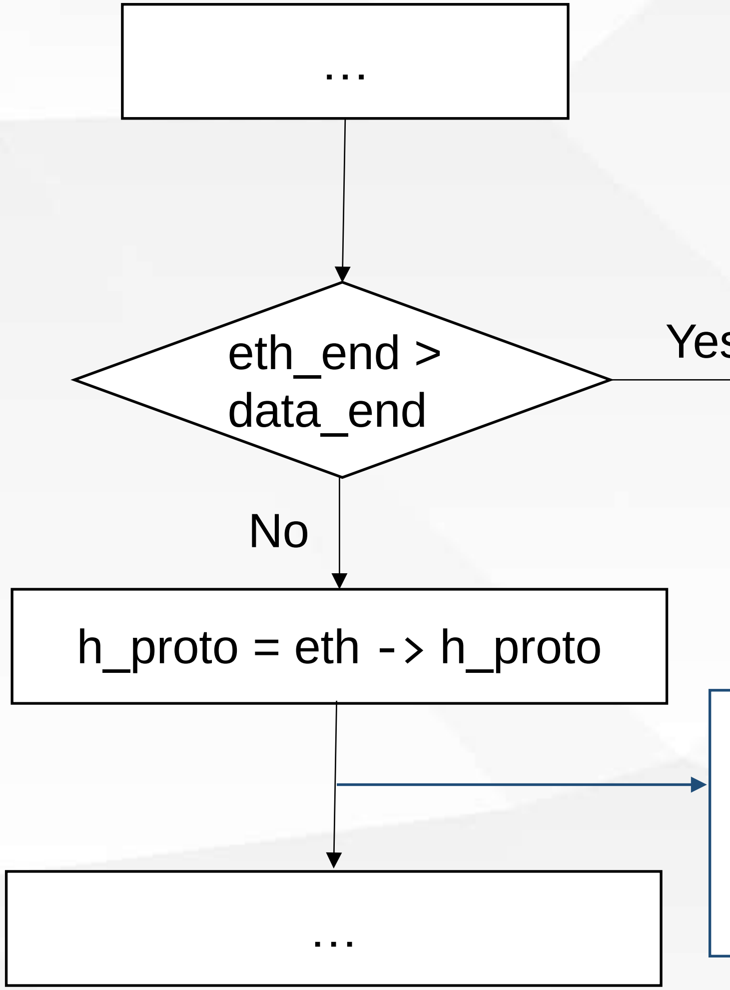

```
data_end == xdp -> data_end && data == xdp -> data &&
eth == data && eth_end == eth + sizeof(struct eth_hdr) &&
eth_end > data_end && store_chars(xdp->data, xdp->data_end)
|--
 TT
```

VEP技术框架 – VEP-C



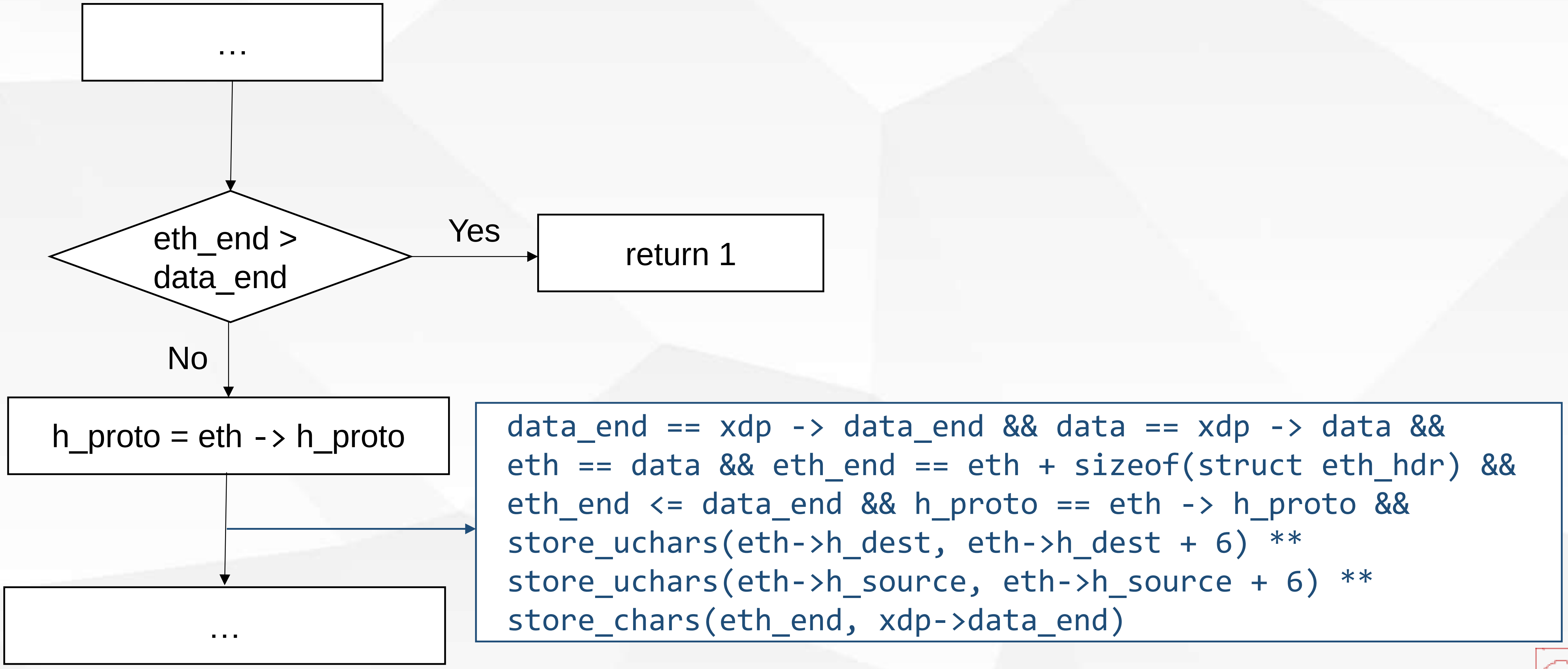

```
data_end == xdp -> data_end && data == xdp -> data &&
eth == data && eth_end <= data_end &&
eth_end == eth + sizeof(struct eth_hdr) &&
store_chars(xdp->data, xdp->data_end)
|--
 exists v R, store_uint(&(eth->h_proto), v) ** R
```

VEP技术框架 – VEP-C

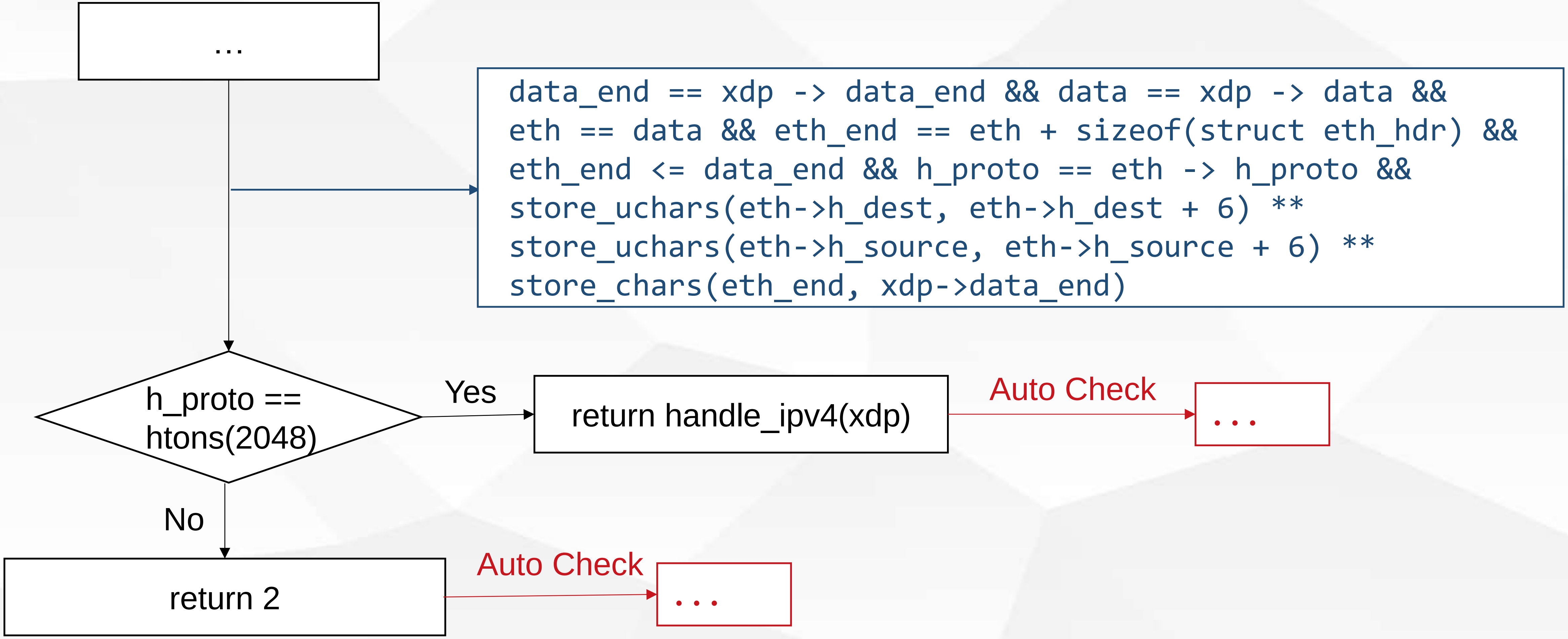



```
data_end == xdp -> data_end && data == xdp -> data &&
eth == data && eth_end == eth + sizeof(struct eth_hdr) &&
eth_end <= data_end && store_eth(eth) **
store_chars(eth_end, xdp->data_end)
```

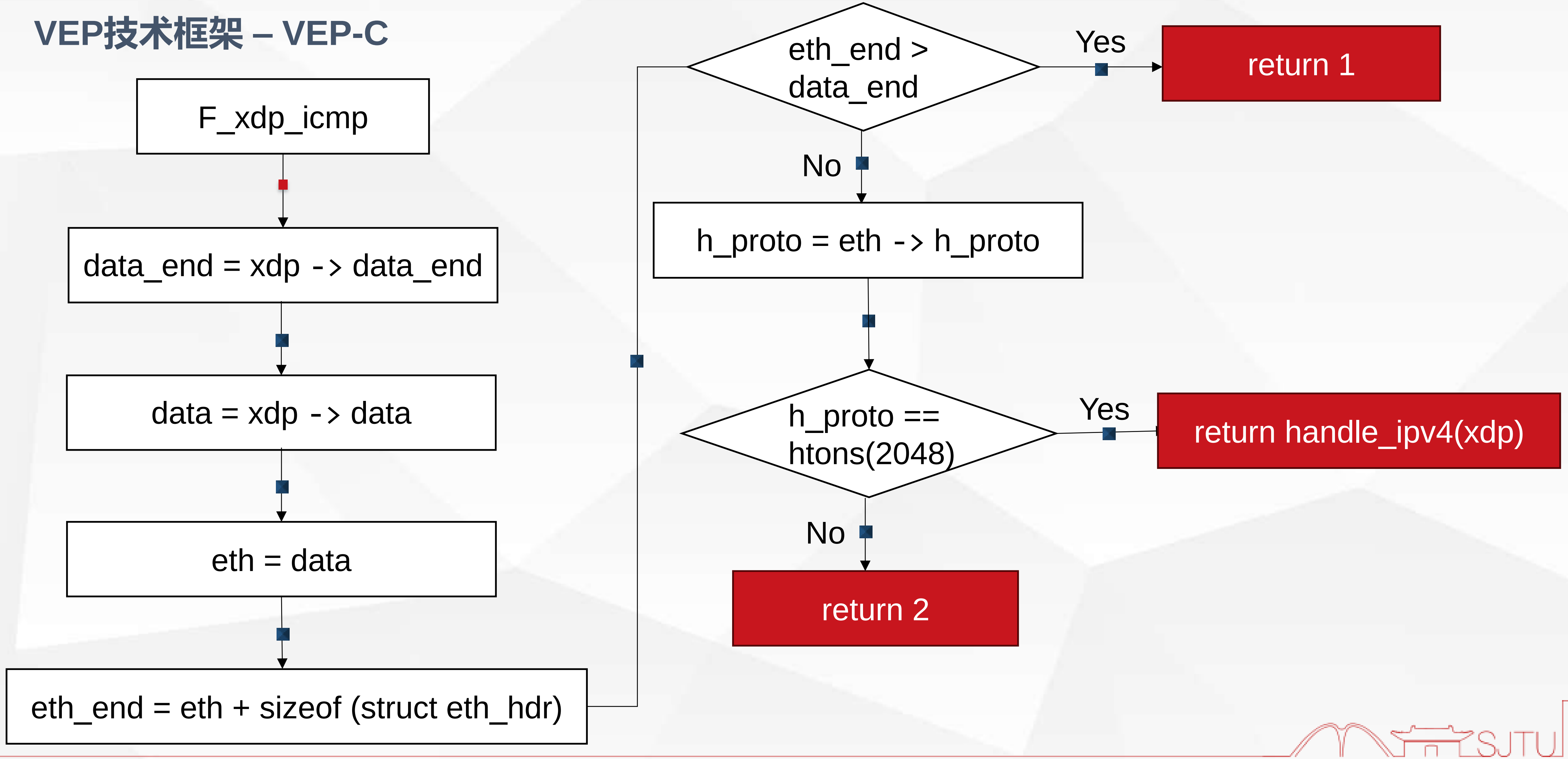
VEP技术框架 – VEP-C



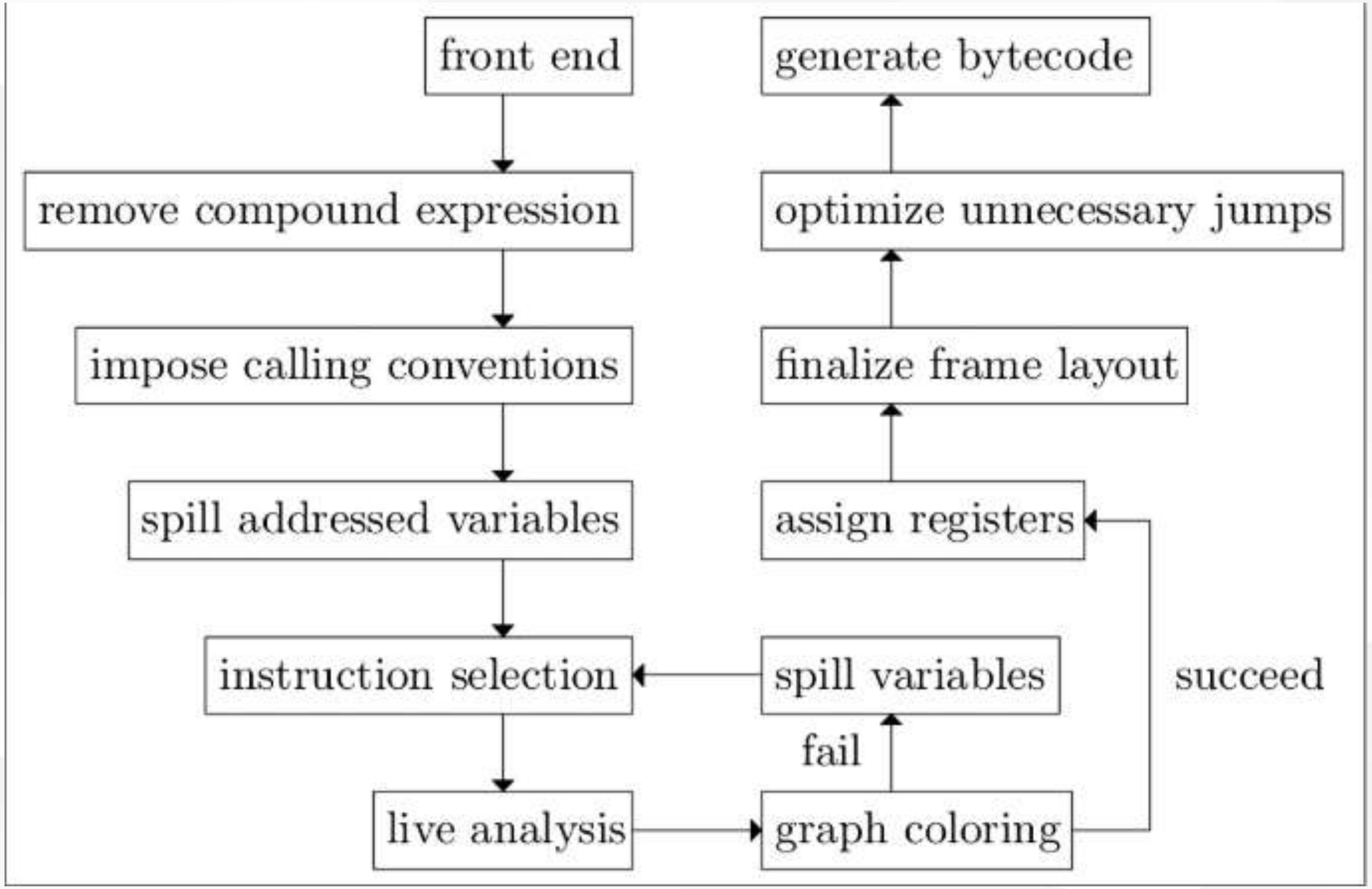
VEP技术框架 – VEP-C



VEP技术框架 – VEP-C



VEP技术框架 – VEP-compiler

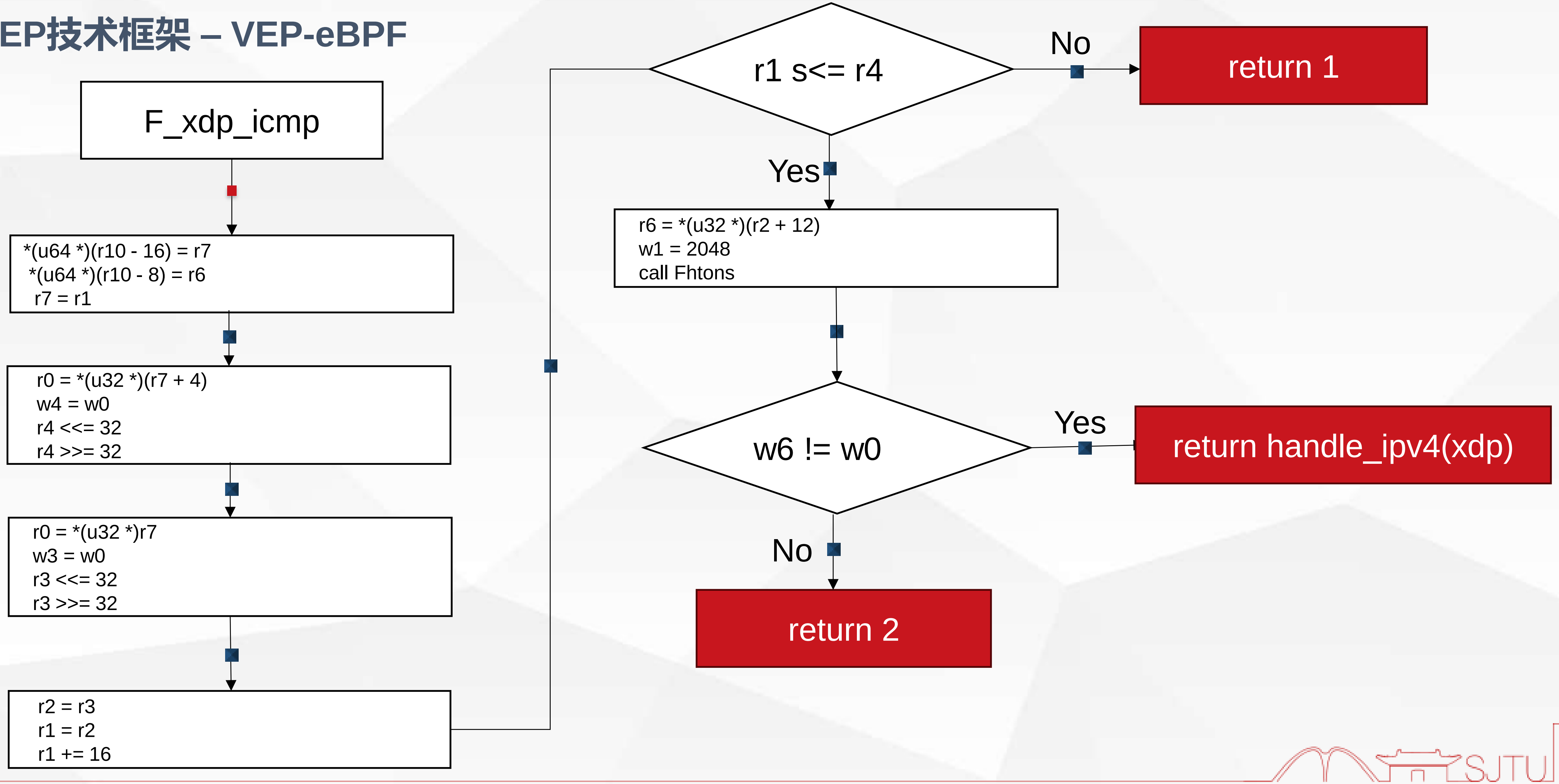


VEP技术框架 – VEP-compiler

```
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) */;
```

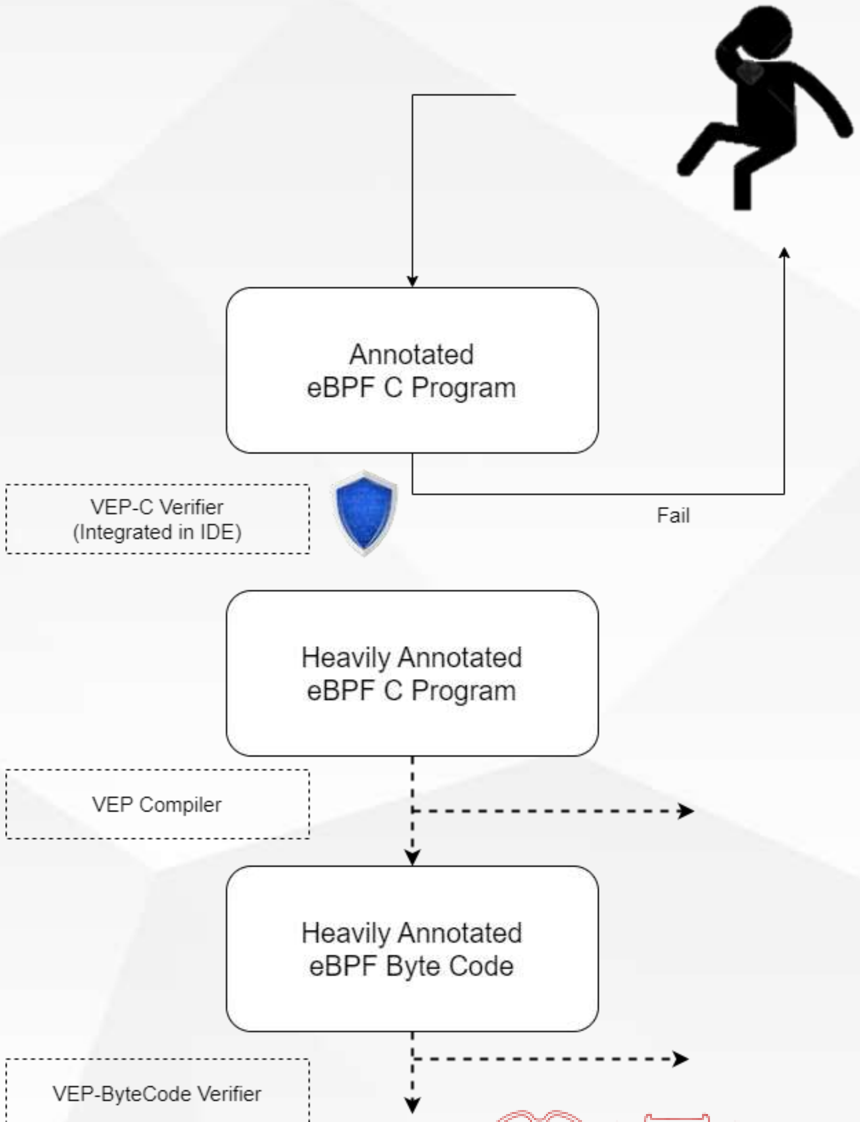
```
strncpy:
/*@ With l1 l2 _R1 _R2 _R3 _R6 _R7 _R8 _R9
    Require
      _R1 == R1 &&
      _R2 == R2 &&
      _R3 == R3 &&
      _R6 == R6 && _R7 == R7 &&
      _R8 == R8 && _R9 == R9 &&
      chars(R1, R3, l1) * chars(R2, R3, l2)
    Ensure
      _R6 == R6 && _R7 == R7 &&
      _R8 == R8 && _R9 ==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

VEP技术框架 – VEP-eBPF



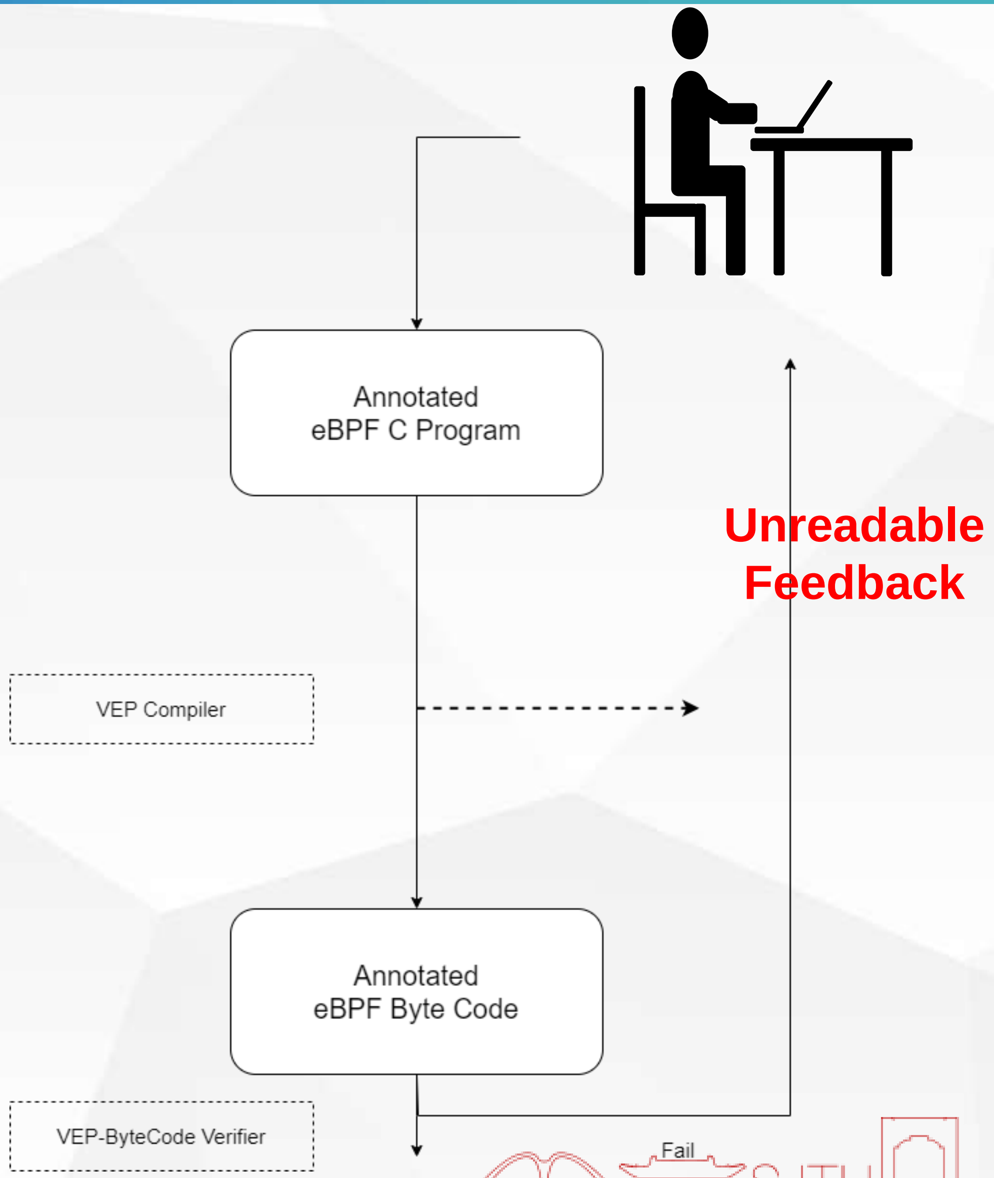
VEP技术框架 – 讨论

1. 如果开发中的eBPF程序有错误有安全问题，如何能够发现这些问题?



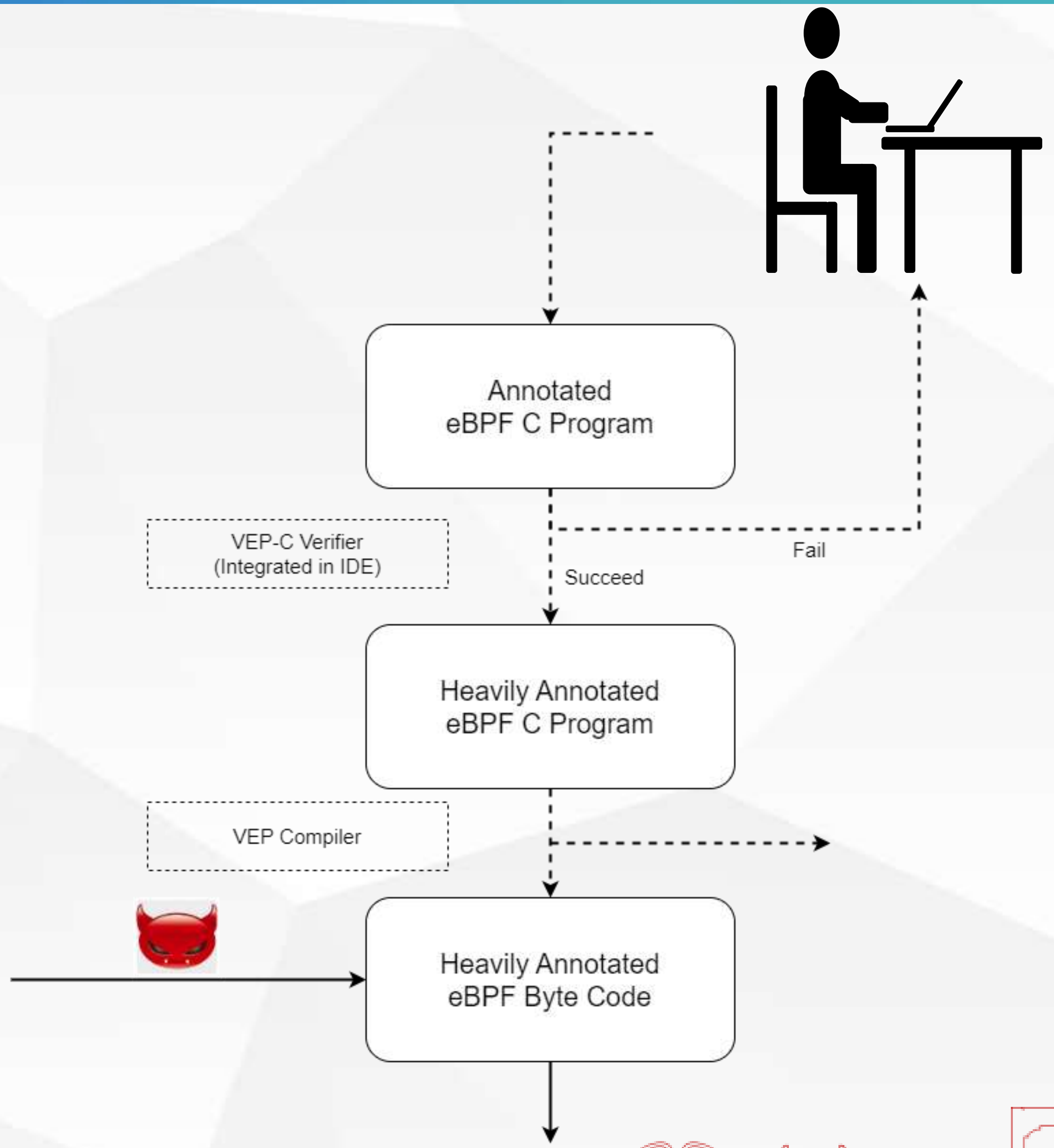
VEP技术框架 – 讨论

1. 如果开发中的eBPF程序有错误有安全问题，如何能够发现这些问题?
2. 为什么不只使用bytecode-verifier?



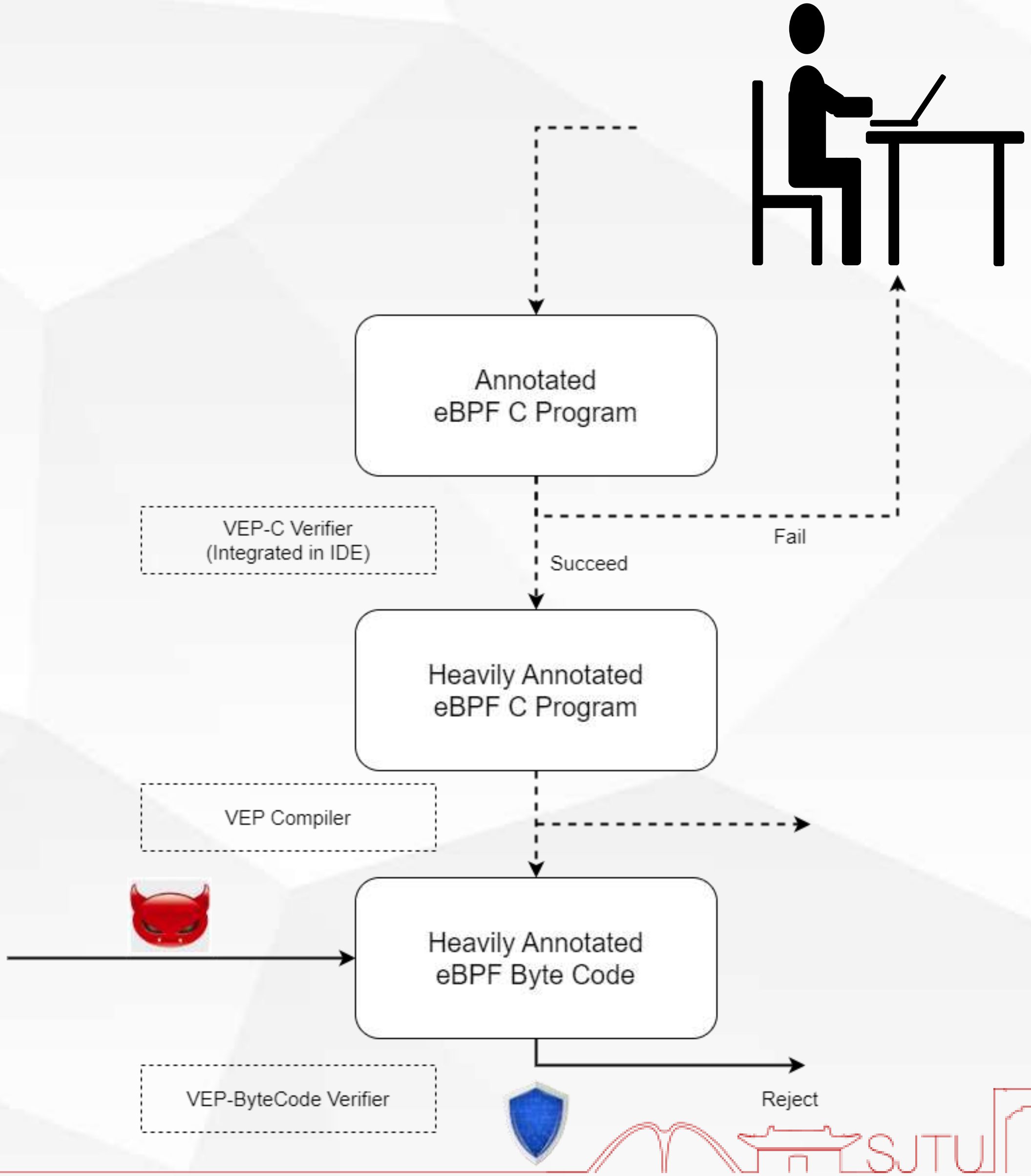
VEP技术框架 – 讨论

1. 如果开发中的eBPF程序有错误有安全问题，如何能够发现这些问题?
2. 为什么不只使用bytecode-verifier?
3. 为什么不只使用C-verifier?



VEP技术框架 – 讨论

1. 如果开发中的eBPF程序有错误有安全问题，如何能够发现这些问题?
2. 为什么不只使用bytecode-verifier?
3. 为什么不只使用C-verifier?
4. 如何应对恶意攻击?



VEP技术框架 – 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	-

VEP技术框架 – 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

VEP技术框架 – 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	-

VEP技术框架 – Evaluation

Programs		Assertion Lines	Proof Lines	VEP-C Time(ms)	VEP-C Memory(KB)	VEP-compiler Time(ms)	VEP-eBPF Time(ms)	VEP-eBPF 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

③ 效果演示

谢谢！

中国·西安

① 背景介绍—Example

```
int badhelpercall()
{    char buffer[1];
     return bpf_get_current_comm(buffer, 20); }
```

```
int badhelpercall()
{    char buffer[1];
     char buffer2[20];
     return bpf_get_current_comm(buffer, 20); }
```

False Negative ?

① 背景介绍—Example

```
void memset (char *p1, char v, __u32 n)
{
    for (__u32 i = 0; i < n; i++)
        p1[i] = v;
    return ;
}
```

False Positive ?

① 背景介绍—Annotation

```
void memset (char *p1, char v, __u32 n)
/*@ With l1
    Require chars(p1, n, l1)
    Ensure exists l2, chars(p1, n, l2)
*/
{
    for (__u32 i = 0; i < n; i++)
        p1[i] = v;
    return ;
}
```

① 背景介绍—挑战

For user

- Programmability : do not change the codes to fit verifier
- Efficiency
- Automatic

For verifier

- Trade-offs among efficiency, resource consumption, and potential false negatives and false positives

For verification

- The scale of TCB

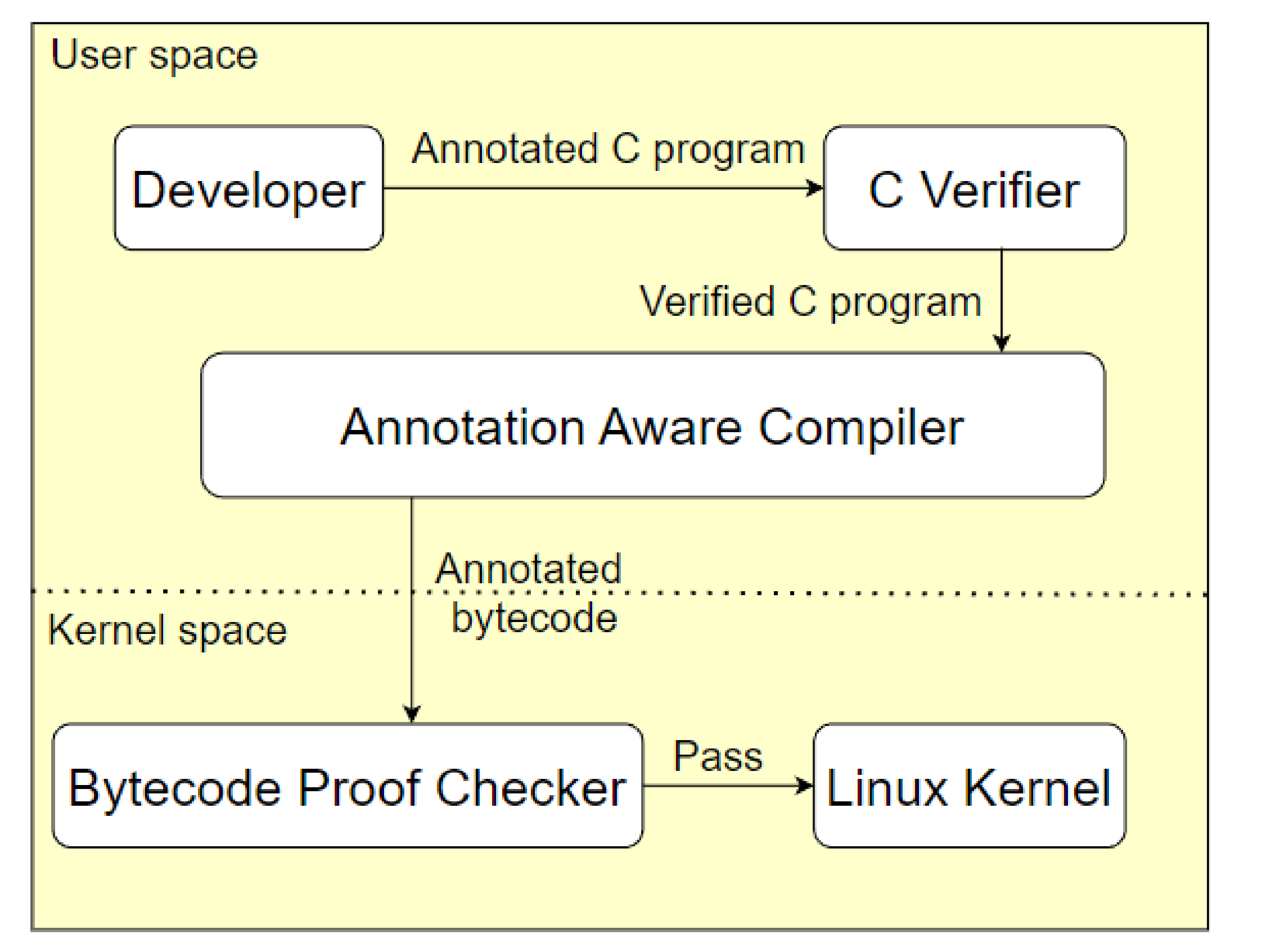
① 背景介绍—Our Choice

Small and efficient to be a part of the kernel.

Final TCB is as minimal as possible.

Highly automated and easy to use.

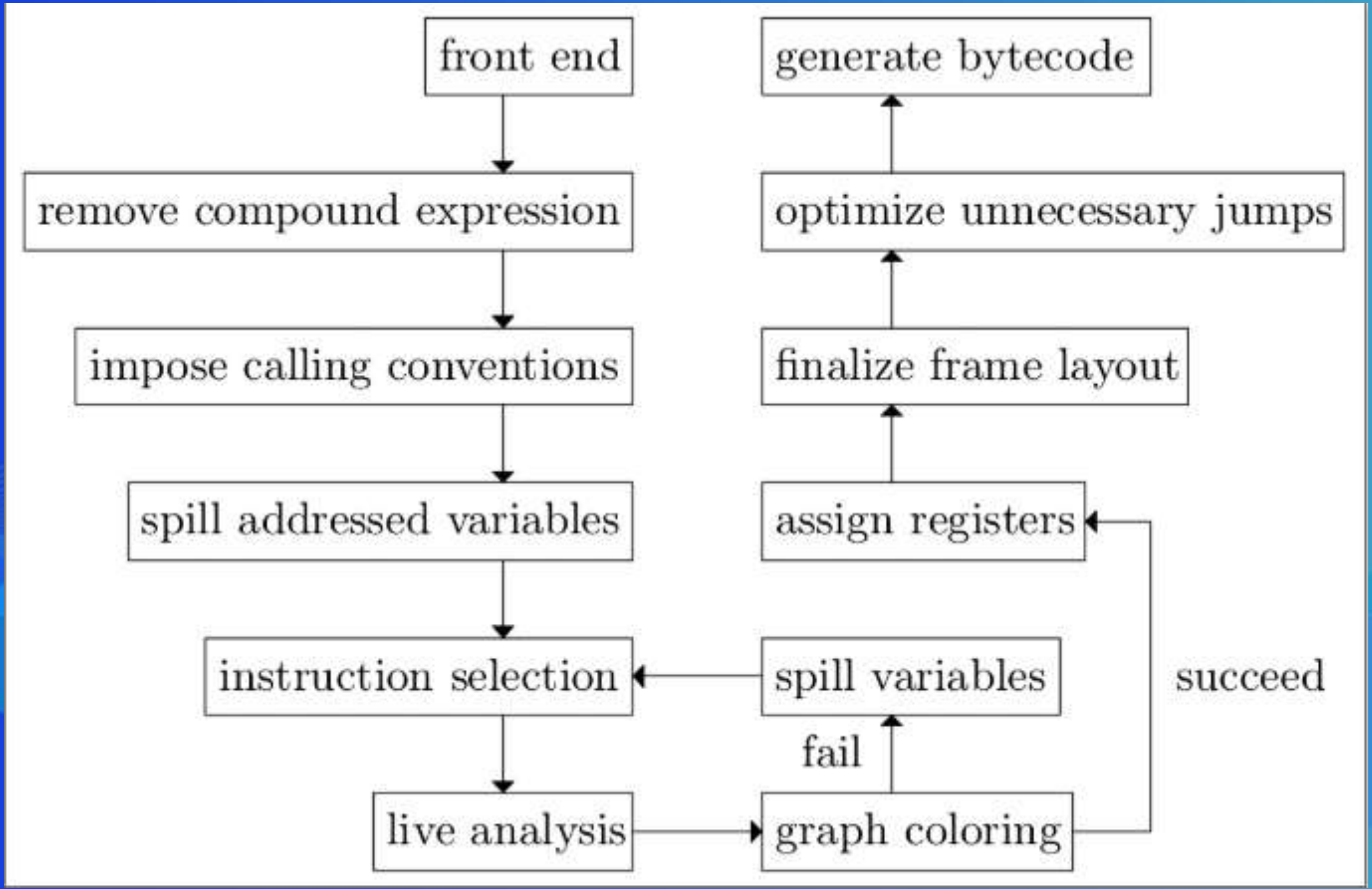
② VEP介绍


User space
Developer
Annotated C program
C Verifier
Verified C program
Annotation Aware Compiler
Annotated
bytecode
Kernel space
Bytecode Proof Checker
Pass
Linux Kernel

② VEP介绍—VEP-C

② VEP介绍—VEP-compiler

② VEP介绍—VEP-compiler



② VEP介绍—VEP-compiler

Calling Conventions

```
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) */
```

② VEP介绍—VEP-compiler

Register Allocation

```
int x, y, *p, *q;
x = 0; y = 1;
p = &x; q = 0;
//@ y == x + 1 && p != q
return y;
```

↓

```
*(R10 - 4) = 0
 R0 = 1
 R1 = R10
 R1 -= 4
 R1 = 0 // p and q are not used
 /*@ R1 == *(R10-4) + 1 &&
     ∃ _p _q, _p != _q */
 ret
```

② VEP介绍—VEP-eBPF

② VEP介绍—Discussion

Why not use only the C verifier ?

Why not use only the bytecode verifier ?

How much programmer effort was required for annotations ?

③ 总结和展望

③ 总结与展望—Evaluation

Programs Source	Total Code Lines	Linux verifier PR	Linux verifier MaxT (ms)	Linux verifier AvgT (ms)	Linux verifier MaxM (KB)	PREVAIL PR	PREVAIL MaxT (ms)	PREVAIL AvgT (ms)	PREVAIL MaxM (KB)
Linux samples	618	9/10	1.13	0.94	4196	10/10	48.37	12.48	7918
PREVAIL samples	252	6/10	1.08	0.71	4200	8/10	74.67	13.88	5279
StringLib	321	3/10	2.69	1.83	5168	1/10	39.89	39.89	7267
Unsafe Programs	195	10/10	0.051	0.038	4340	10/10	34.88	6.05	5312
Key_Connection	63	0/1	0.025	0.025	4304	0/1	2.312	2.312	5004

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

③ 总结与展望—未来工作

Towards Functional Correctness

Towards Less Annotations

Towards Compilation Optimization

③ 总结与展望—未来工作

Backup

```
void memset (char *p1, char v, __u32 n)
/*@ With l1
    Require chars(p1, n, l1)
    Ensure exists l2, chars(p1, n, l2)
*/
{
    /*@ Inv: exists l2,
         0 <= i <= n && chars(p1,n,l2) */
    for (__u32 i = 0; i < n; i++)
        p1[i] = v;
    return ;
}
```

Backup

```
void memset (char *p1, char v, __u32 n) {
    __u32 i = 0;
    /*@ i == 0 && chars(p1,n,l1) */
    /*@ Inv: exists l2,
         0 <= i <= n && chars(p1,n,l2) */
    for (; i < n;) {
        p1[i] = v;
        i++;
    }
    return ;
}
```

Backup

```
void memset (char *p1, char v, __u32 n) {
    __u32 i = 0;
    /*@ Inv: exists l2,
         0 <= i <= n && chars(p1,n,l2) */
    for (; i < n;) {
    /*@ exists l2, 0 <= i < n && chars(p1,n,l2) */
        p1[i] = v;
        i++;
    }
    return ;
}
```

Backup

```
void memset (char *p1, char v, __u32 n) {
    __u32 i = 0;
    /*@ Inv: exists l2,
         0 <= i <= n && chars(p1,n,l2) */
    for (; i < n;) {
        p1[i] = v;
    /*@ exists l2 l3, 0 <= i < n &&
        l3[0:i] == l2[0:i] && l3[i] == v &&
        l3[i+1:n] == l2[i+1:n] &&
        chars(p1,n,l3) */
        i++;
    }
    return ;
}
```

Backup

```
void memset (char *p1, char v, __u32 n) {
     __u32 i = 0;
     /*@ Inv: exists l2,
          0 <= i <= n && chars(p1,n,l2) */
     for (; i < n;) {
          p1[i] = v;
          i++;
     /*@ exists l2 l3, 0 <= i-1 < n &&
          l3[0: i-1] == l2[0: i-1] && l3[i-1] == v &&
          l3[i:n] == l2[i:n] &&
          chars(p1,n,l3) */
     }
     return ;
}
```

Backup

```
void memset (char *p1, char v, __u32 n) {
     __u32 i = 0;
     /*@ Inv: exists l2,
           0 <= i <= n && chars(p1,n,l2) */
     for (; i < n;) {
          p1[i] = v;
          i++;
     }
     /*@ exists l2, i == n && chars(p1,n,l2) */
     return ;
}
```

VEP技术框架 – VEP-eBPF

```
F_xdp_icmp:
   *(u64 *)(r10 - 16) = r7
   *(u64 *)(r10 - 8) = r6
   r7 = r1
   r0 = *(u32 *)(r7 + 4)
   w4 = w0
   r4 <<= 32
   r4 >>= 32
   r0 = *(u32 *)r7
   w3 = w0
   r3 <<= 32
   r3 >>= 32
   r2 = r3
   r1 = r2
   r1 += 16
```

```
if r1 s<= r4 goto L1065
  w0 = 1
 goto L1072
L1065:
  r6 = *(u32 *)(r2 + 12)
  w1 = 2048
  call Fhtons
 if w6 != w0 goto L1068
  r1 = r7
  call Fhandle_ipv4
 goto L1072
L1068:
  w0 = 2
L1072:
  r6 = *(u64 *)(r10 - 8)
  r7 = *(u64 *)(r10 - 16)
  exit
```