DR. CHECKER

A *Soundy* Analysis for Linux Kernel Drivers

*Aravind Machiry, **Chad Spensky**, Jake Corina, Nick Stephens, Christopher Kruegel, and Giovanni Vigna*

University of California, Santa Barbara

USENIX Security 2017

First, a story…

First, a story…

First, a story…

$ mkdir driver_checker

First, a story…

$ mkdir dr_checker

First, a story…

DR. CHECKER: A Soundy Analysis for Linux Kernel Drivers

Aravind Machiry, Chad Spensky, Jake Corina, Nick Stephens,
Christopher Kruegel, and Giovanni Vigna
{*machiry, cspensky, jcorina, stephens, chris, vigna*}*@cs.ucsb.edu*
University of California, Santa Barbara

Why Drivers?

Why Drivers?

```
$ ls linux

    /arch        /block       /certs       /kernel      /crypto      /include      /init        /virt        /ipc       /samples

   /drivers    /firmware    /scripts       /fs          /net         /tools        /mm          /usr         /lib       /sound    /security

$
```

Why Drivers?

```
$ ls linux

    /arch        /block       /certs       /kernel      /crypto      /include      /init        /virt        /ipc        /samples

   /drivers    /firmware    /scripts       /fs          /net         /tools        /mm          /usr         /lib        /sound     /security

$
```

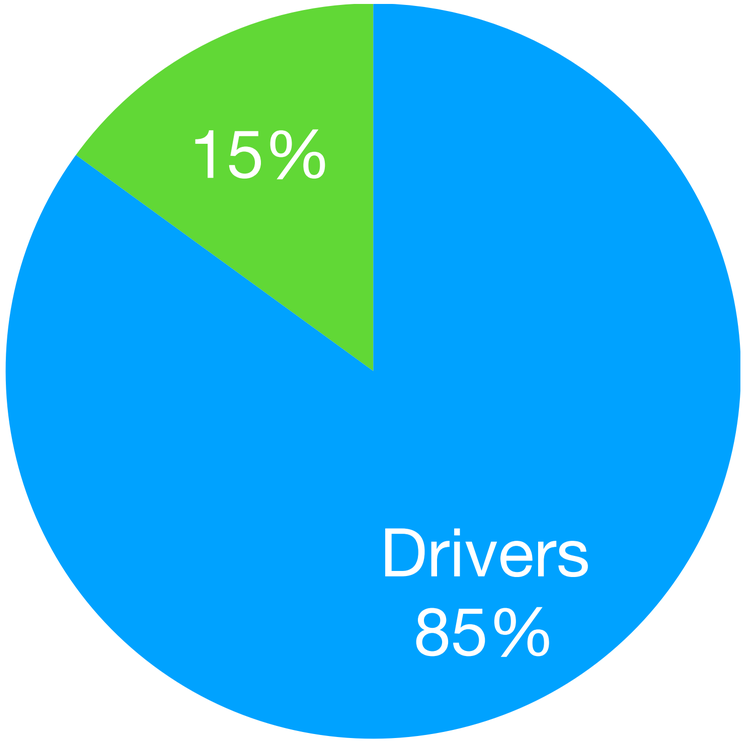
Why Drivers?

```
$ ls linux

  /arch      /block     /certs     /kernel    /crypto    /include    /init      /virt      /ipc      /samples

 /drivers  /firmware  /scripts     /fs        /net       /tools      /mm        /usr       /lib      /sound   /security

$ find bugs
```

Why Drivers?

CVE - Common Vulnerability and Exposure

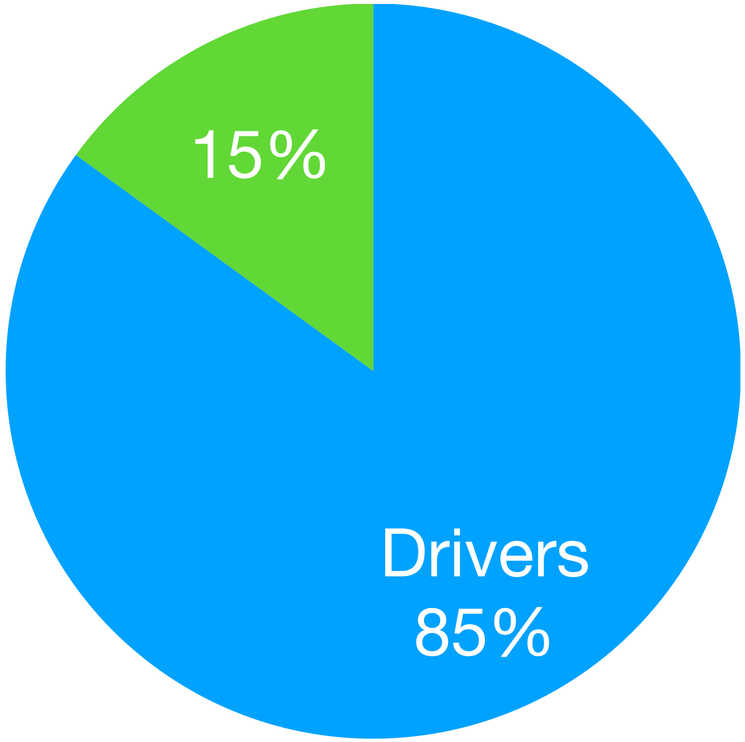
Why Drivers?



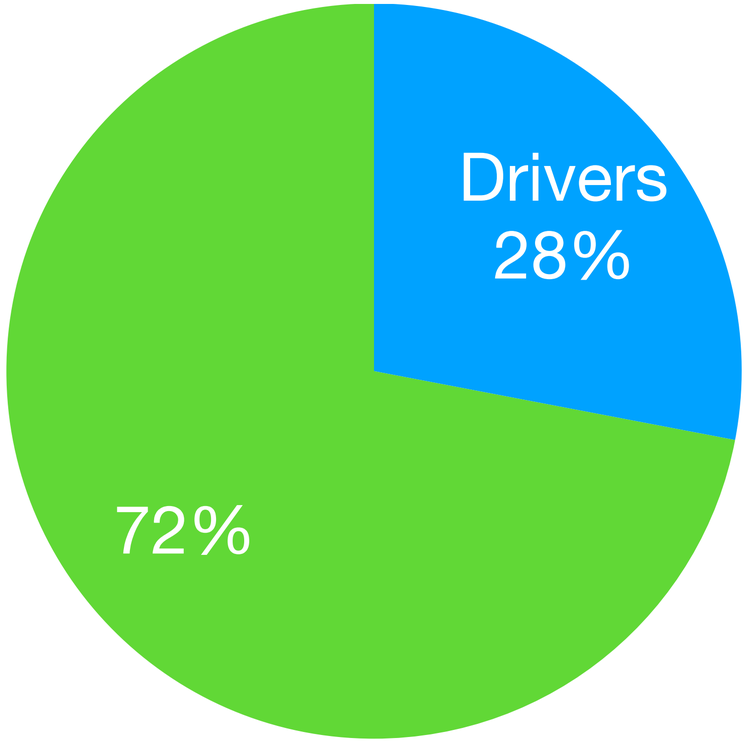
Bugs in Windows XP (2003)

CVE - Common Vulnerability and Exposure

Why Drivers?



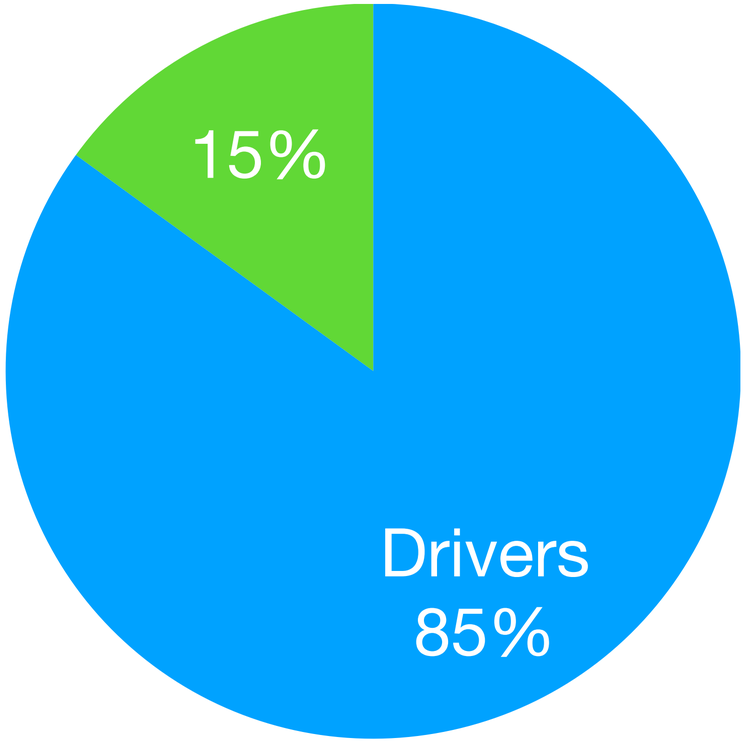
Bugs in Windows XP (2003)



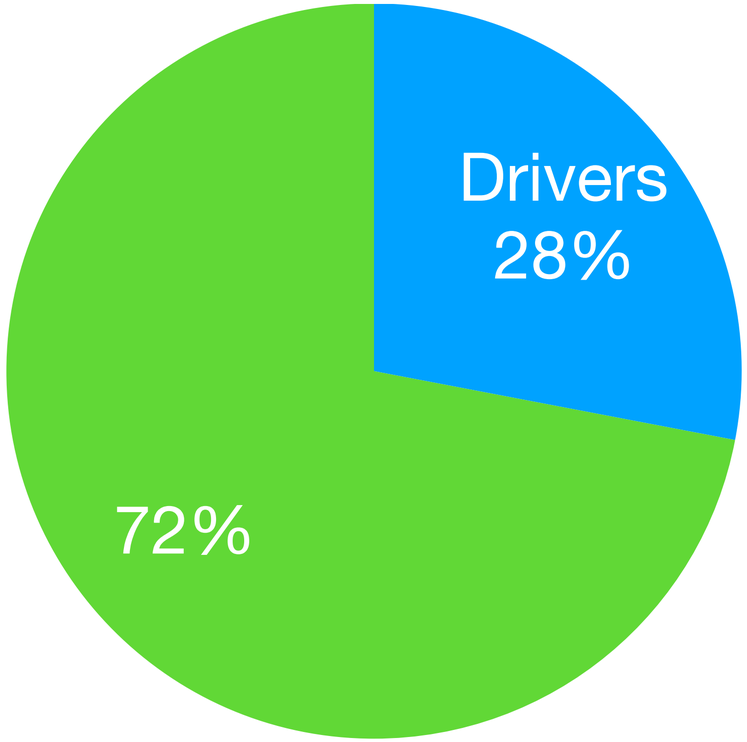
Linux Kernel CVEs (2016-2017)

CVE - Common Vulnerability and Exposure

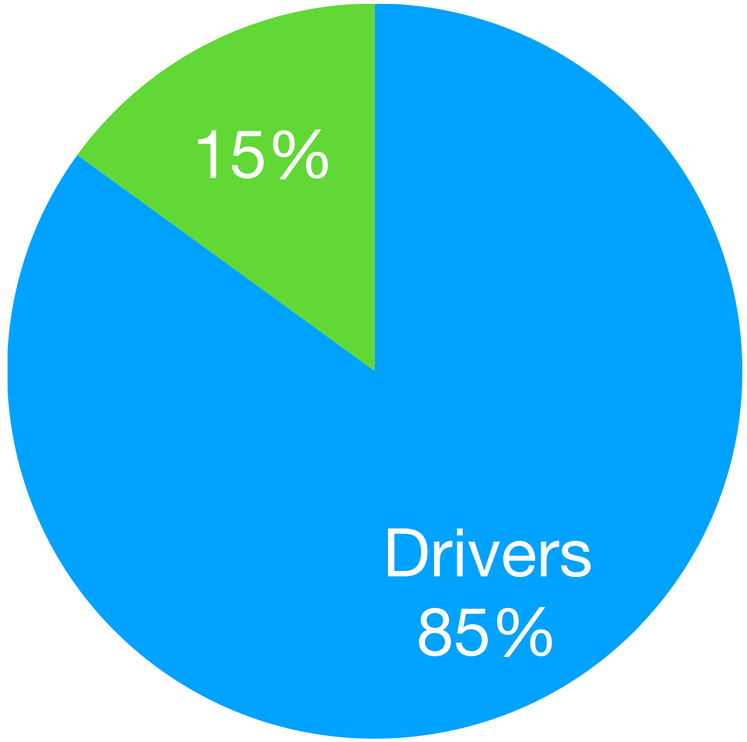
Why Drivers?



Bugs in Windows XP (2003)



Linux Kernel CVEs (2016-2017)

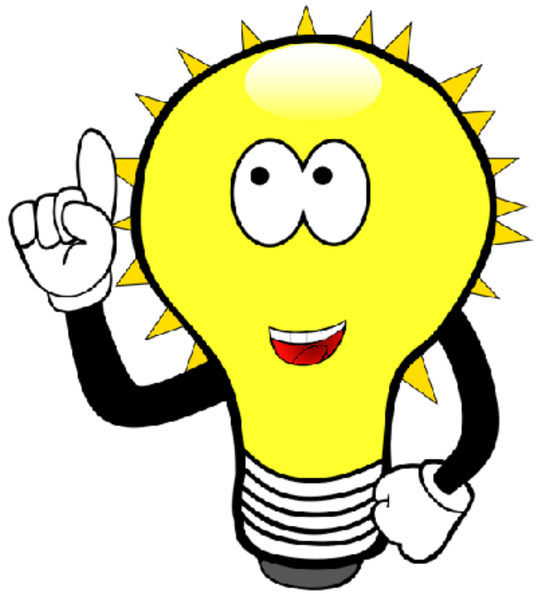


Reported bugs in Android (2016)

CVE - Common Vulnerability and Exposure

Motivation

Only analyze the drivers!



Program Analysis for Bug Finding

Program Analysis for Bug Finding

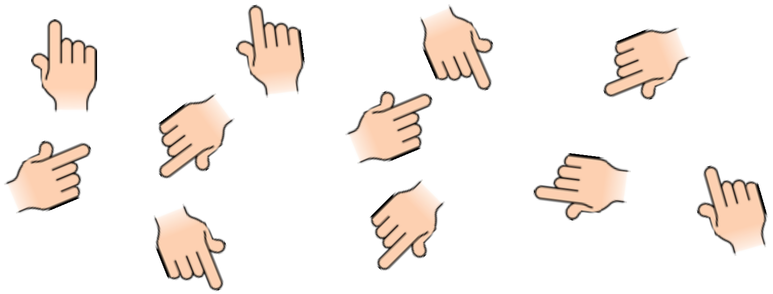
- **Points-to Analysis:** Determines all storage locations that a pointer can point to
  - ***Example bug:*** *Kernel code pointer to user-controlled memory*

Program Analysis for Bug Finding

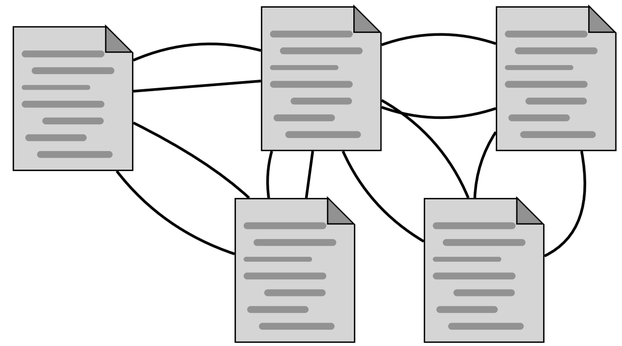
- **Points-to Analysis:** Determines all storage locations that a pointer can point to
  - ***Example bug:*** *Kernel code pointer to user-controlled memory*
- **Taint Analysis:** Determines all of the locations that are affected by user-supplied (tainted) data
  - ***Example bug:*** *User provided data used as length in copy_from_user()*

Program Analysis on Kernel Code

- Pointers… Everywhere!
  - State explosion



- Inter-procedural calls to core functions
  - State explosion



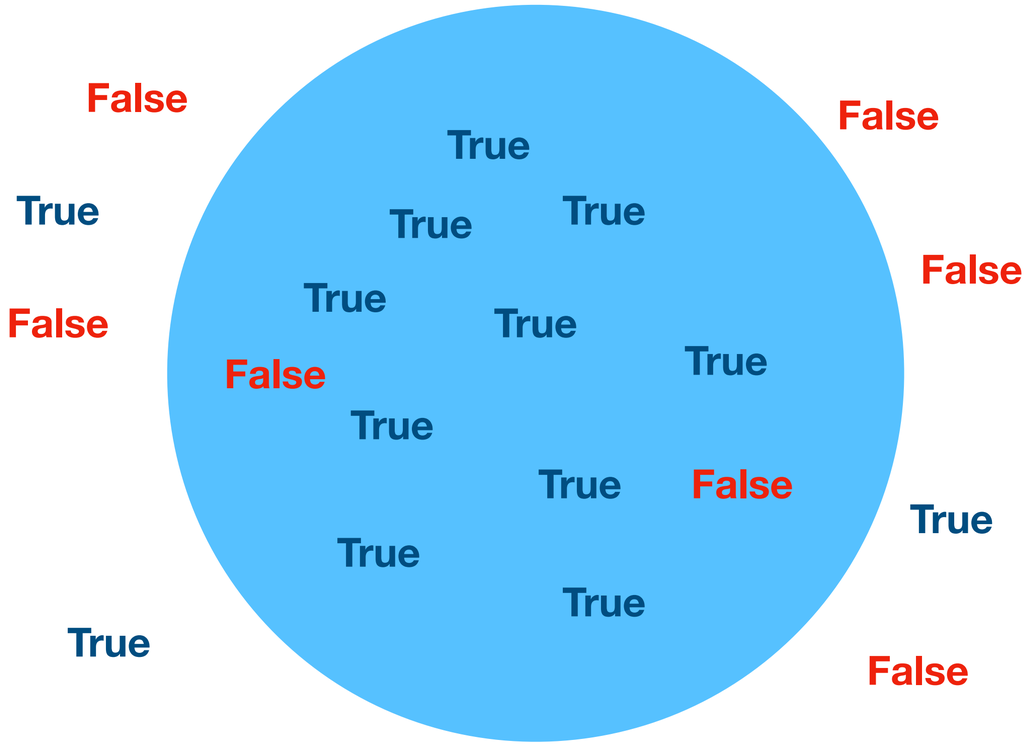
Precision vs. Soundness

Precise

Sound

Precision vs. Soundness

Precise

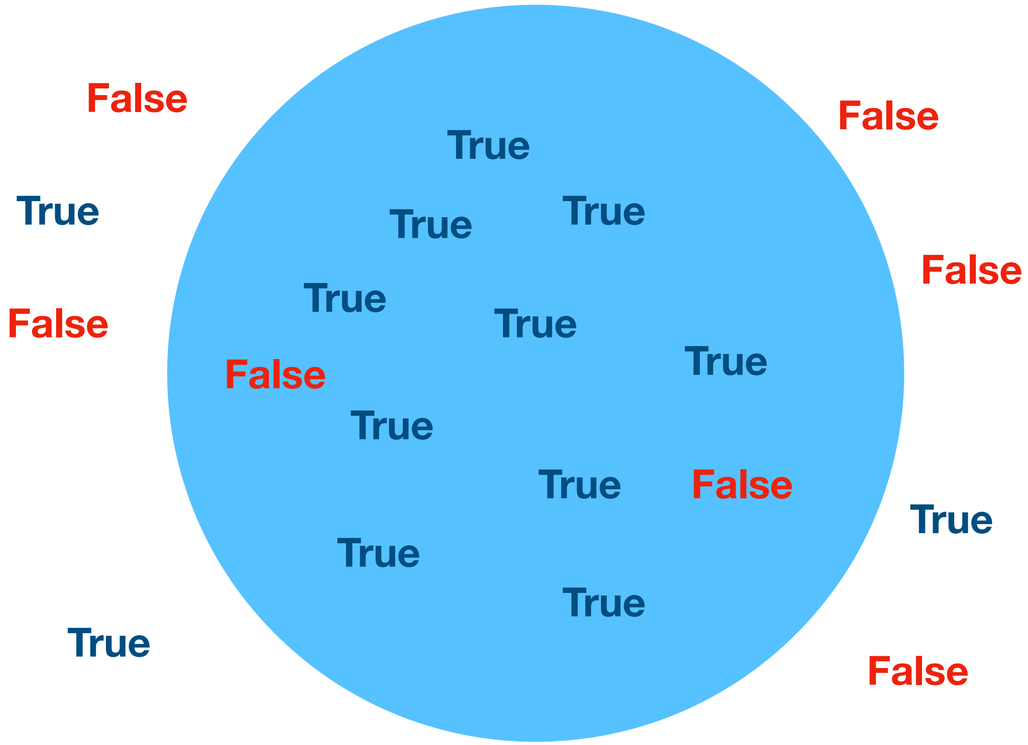


Most of the things reported are true

Sound

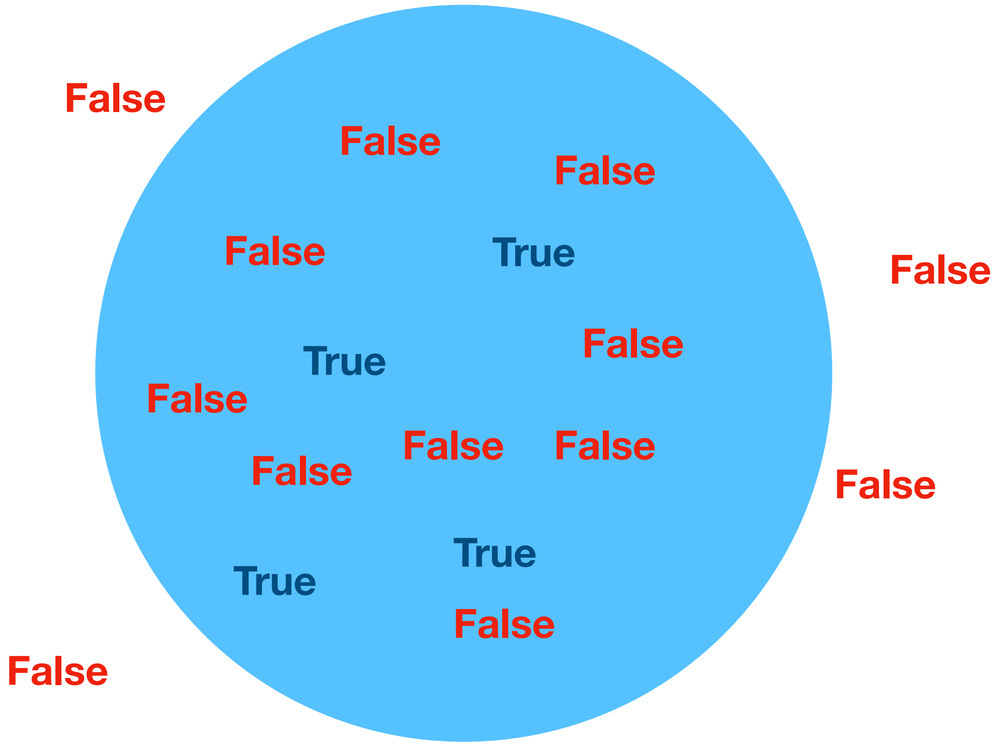
Precision vs. Soundness

Precise



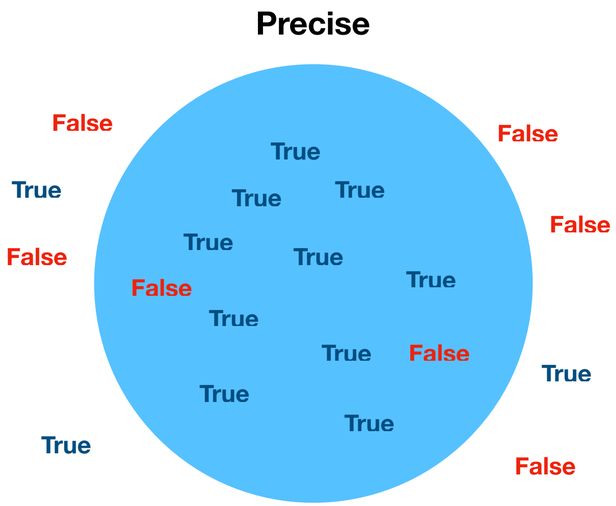
Most of the things reported are true

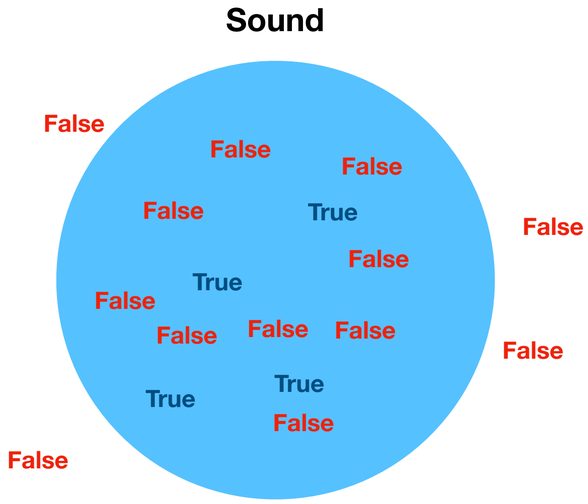
Sound



Everything that is true is reported

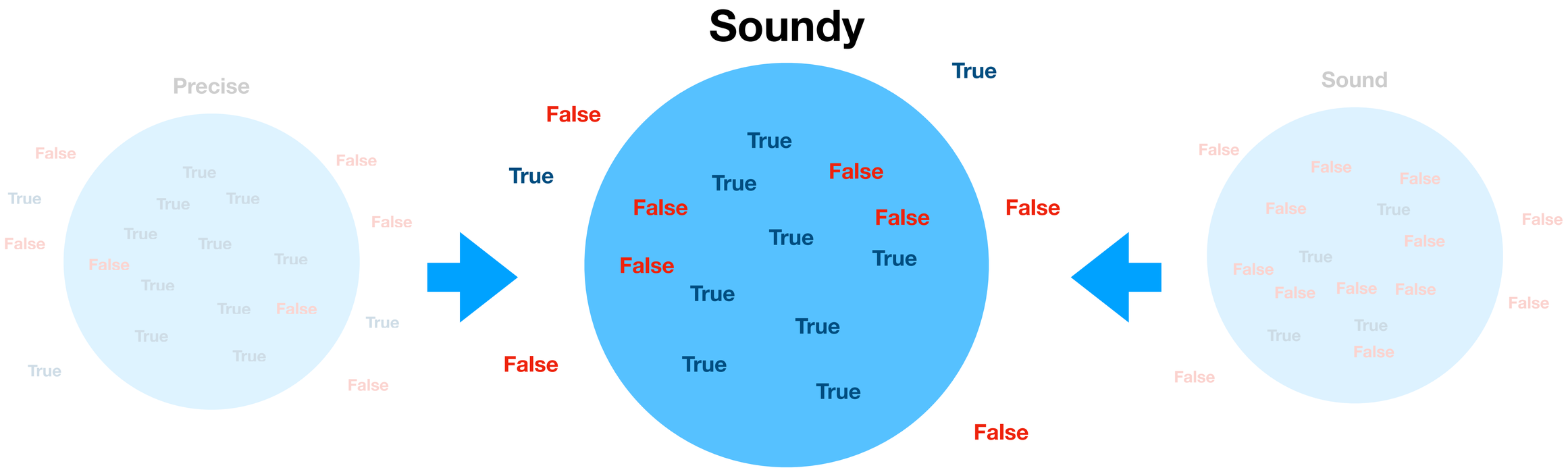
Soundiness





Violate soundness to achieve higher precision and practical computational constraints

Soundiness



Violate soundness to achieve higher precision and practical computational constraints

Dr. Checker: Assumptions

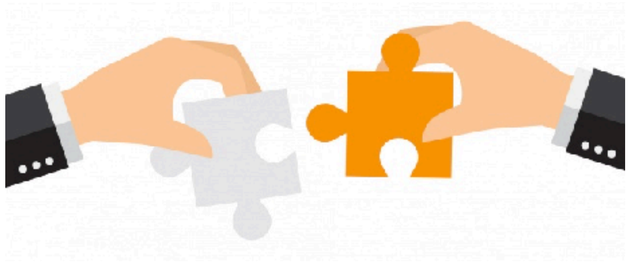
(1) All non-driver code is implemented perfectly

(2) Only evaluate loops until a reaching definition

(3) All calls are traversed exactly once, even in loops

Dr. Checker: Design

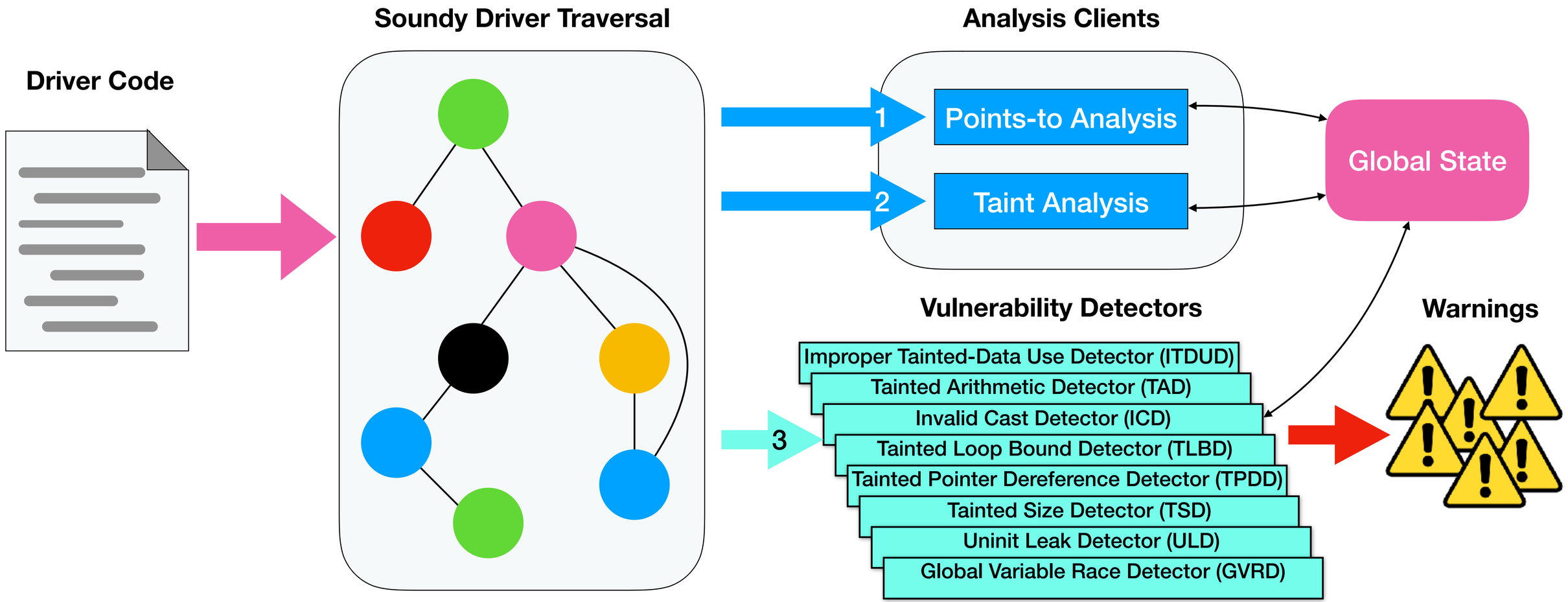
- Modular framework to enable flexible development



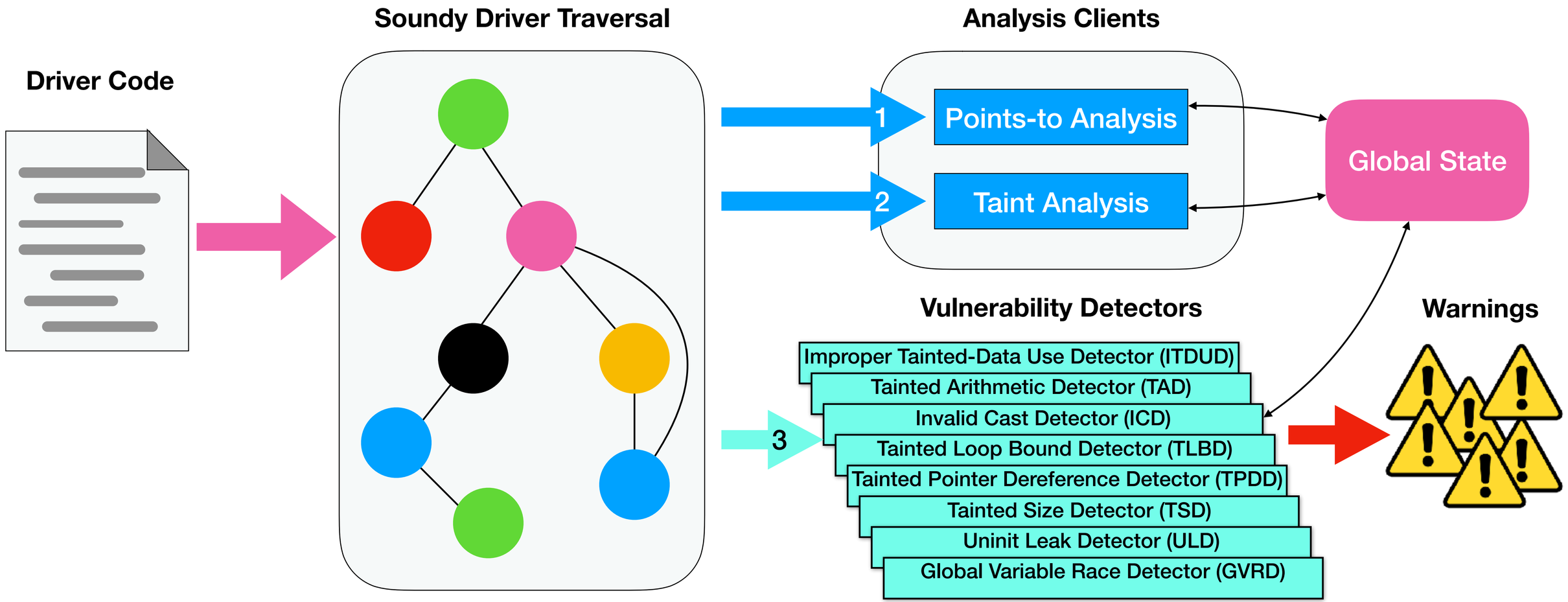
- Simultaneously employ numerous vulnerability detectors

- Open source: github.com/ucsb-seclab/dr_checker

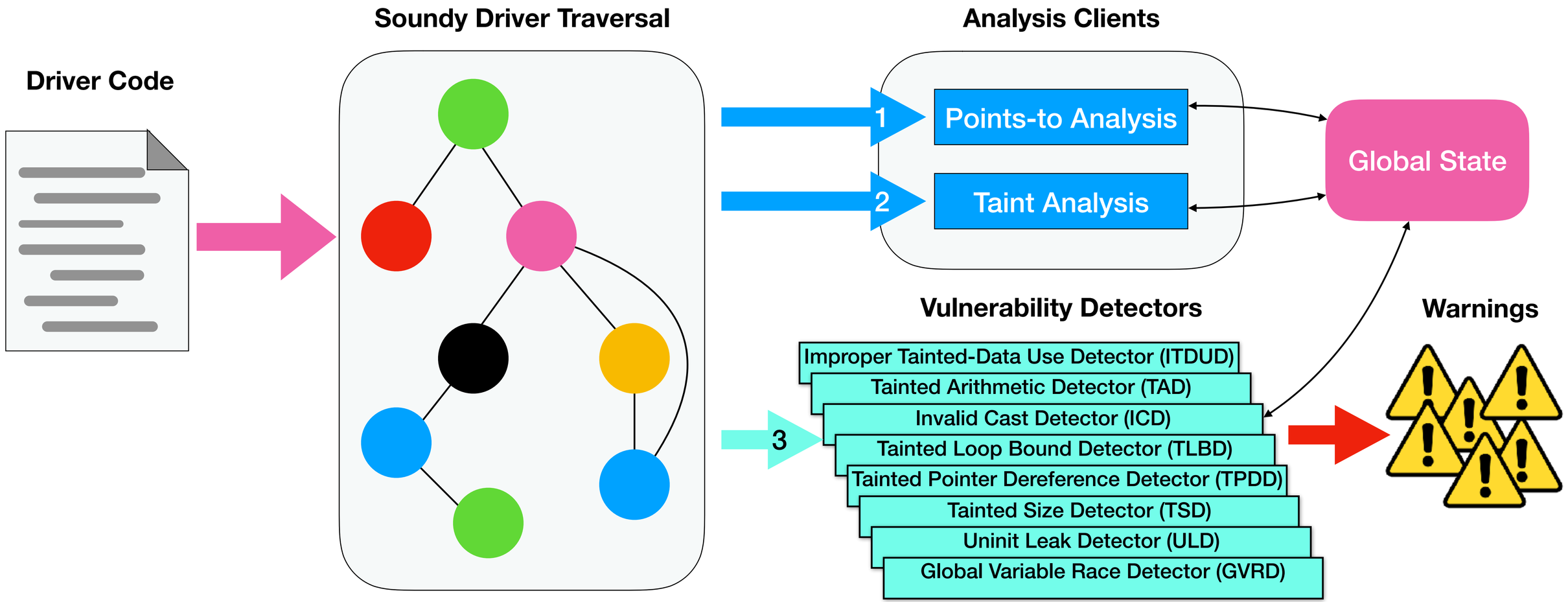
Dr. Checker: Design



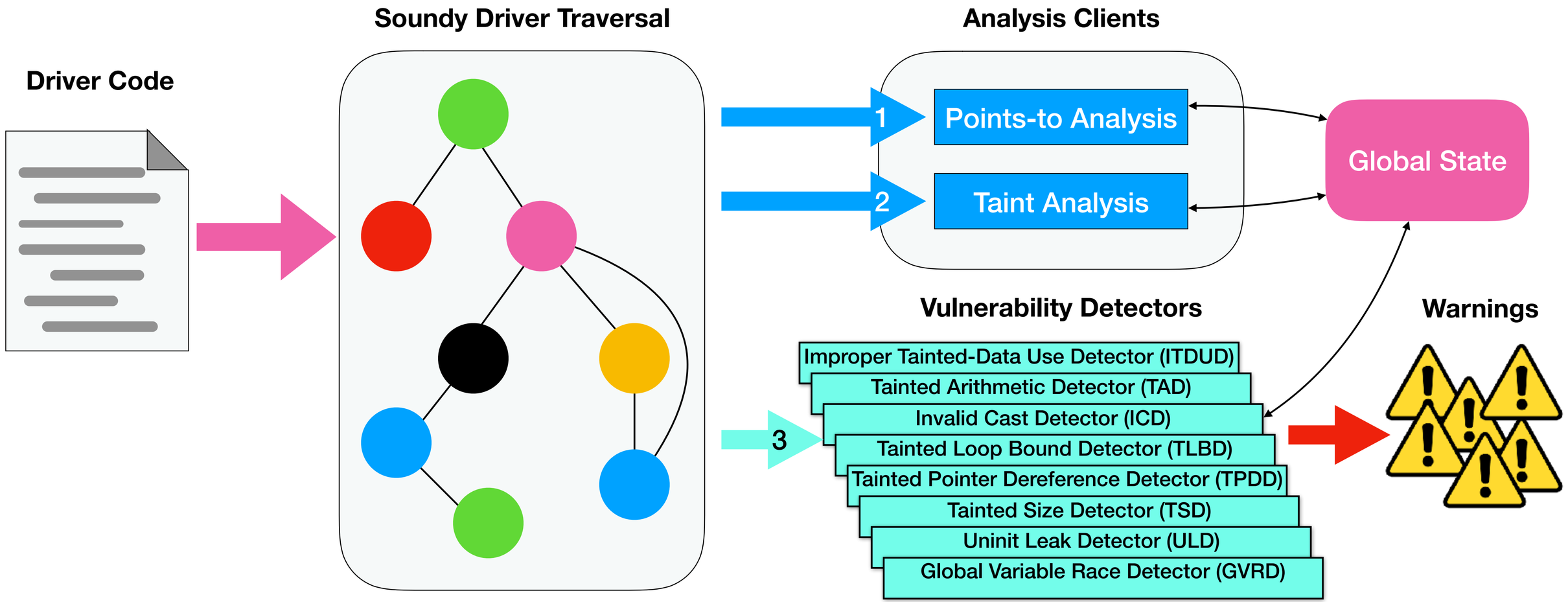
Dr. Checker: Design



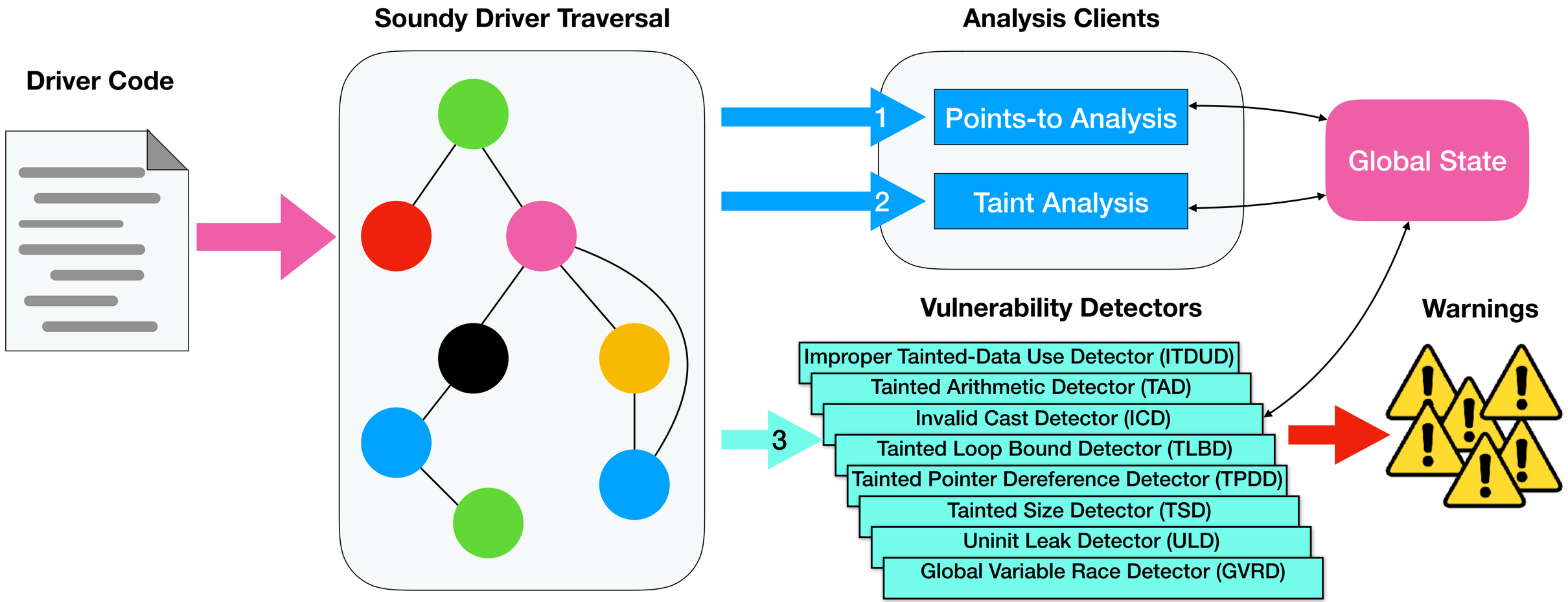
Dr. Checker: Design



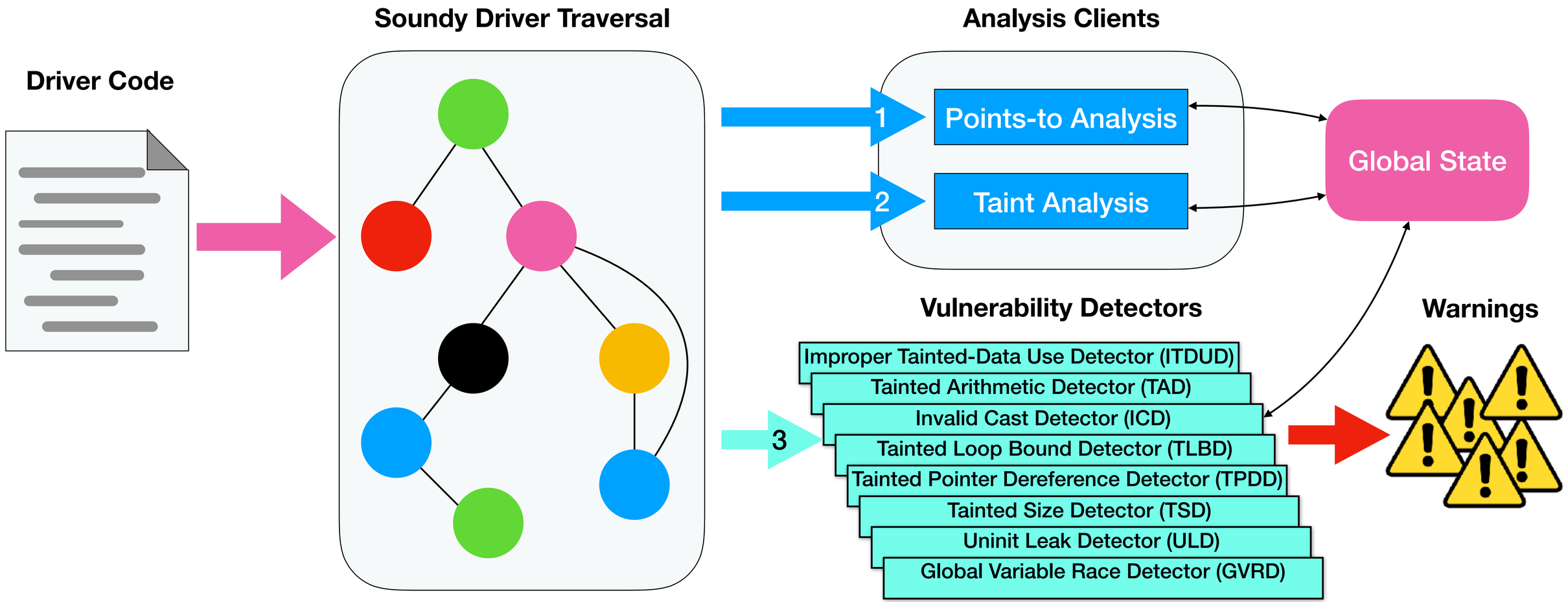
Dr. Checker: Design



Dr. Checker: Design



Dr. Checker: Design



Soundy Driver Traversal

Soundy Driver Traversal

- **Context-sensitive:** Analysis for each function call is done in the context of the calling function

Soundy Driver Traversal

- **Context-sensitive:** Analysis for each function call is done in the context of the calling function
- **Field-sensitive:** The ability to differentiate between different fields in a memory structure

Soundy Driver Traversal

- **Context-sensitive:** Analysis for each function call is done in the context of the calling function
- **Field-sensitive:** The ability to differentiate between different fields in a memory structure
- **Flow-sensitive:** The ability to track data usage (e.g., taint) throughout a program, according to its control flow

Soundy Driver Traversal

```
struct kernel_obj ko;

void internal_function(int *ptr) {
    *ptr += 1;
}

void entry_point(void *user_ptr, int len) {
  curr_data->item = &ko;

  copy_from_user(&ko, user_ptr, len);

  for (int i = 0; i < ko.count; i++) {
    internal_function(&(ko.data[i]));
  }

  dangerous_function(curr_data->buf);
  dangerous_function(curr_data->item);
  kernel_function(curr_data->item);

}
```

Soundy Driver Traversal

Taint Analysis

user_ptr

len

```
struct kernel_obj ko;

void internal_function(int *ptr) {
   *ptr += 1;
}

void entry_point(void *user_ptr, int len) {
  curr_data->item = &ko;

  copy_from_user(&ko, user_ptr, len);

  for (int i = 0; i < ko.count; i++) {
    internal_function(&(ko.data[i]));
  }

  dangerous_function(curr_data->buf);
  dangerous_function(curr_data->item);
  kernel_function(curr_data->item);

}
```

Soundy Driver Traversal

Taint Analysis

user_ptr

len

```
struct kernel_obj ko;

void internal_function(int *ptr) {
   *ptr += 1;
}

void entry_point(void *user_ptr, int len) {
  curr_data->item = &ko;

  copy_from_user(&ko, user_ptr, len);

  for (int i = 0; i < ko.count; i++) {
    internal_function(&(ko.data[i]));
  }

  dangerous_function(curr_data->buf);
  dangerous_function(curr_data->item);
  kernel_function(curr_data->item);

}
```

Field-sensitive

Soundy Driver Traversal

Taint Analysis

user_ptr

len

ko

curr_data->item

Taint Source

```
struct kernel_obj ko;

void internal_function(int *ptr) {
   *ptr += 1;
}

void entry_point(void *user_ptr, int len) {
  curr_data->item = &ko;

  copy_from_user(&ko, user_ptr, len);

  for (int i = 0; i < ko.count; i++) {
    internal_function(&(ko.data[i]));
  }

  dangerous_function(curr_data->buf);
  dangerous_function(curr_data->item);
  kernel_function(curr_data->item);

}
```

Field-sensitive

Warning: Improper Tainted-Data Use

Soundy Driver Traversal

Taint Analysis

user_ptr

len

ko

curr_data->item

Taint Source

```
struct kernel_obj ko;

void internal_function(int *ptr) {
   *ptr += 1;
}

void entry_point(void *user_ptr, int len) {
  curr_data->item = &ko;

  copy_from_user(&ko, user_ptr, len);

  for (int i = 0; i < ko.count; i++) {
    internal_function(&(ko.data[i]));
  }

  dangerous_function(curr_data->buf);
  dangerous_function(curr_data->item);
  kernel_function(curr_data->item);

}
```

Field-sensitive

Warning: Improper Tainted-Data Use

Warning: Tainted Loop Bound

Soundy Driver Traversal

Taint Analysis

user_ptr

len

ko

curr_data->item

Taint Source

```
struct kernel_obj ko;

void internal_function(int *ptr) {
    *ptr += 1;
}

void entry_point(void *user_ptr, int len) {
  curr_data->item = &ko;

  copy_from_user(&ko, user_ptr, len);

  for (int i = 0; i < ko.count; i++) {
    internal_function(&(ko.data[i]));
  }

  dangerous_function(curr_data->buf);
  dangerous_function(curr_data->item);
  kernel_function(curr_data->item);

}
```

Field-sensitive

Warning: Improper Tainted-Data Use

Warning: Tainted Loop Bound

Soundy Driver Traversal

Taint Analysis

user_ptr

len

ko

curr_data->item

Taint Source

```
struct kernel_obj ko;

void internal_function(int *ptr) {
   *ptr += 1;
}

void entry_point(void *user_ptr, int len) {
  curr_data->item = &ko;

  copy_from_user(&ko, user_ptr, len);

  for (int i = 0; i < ko.count; i++) {
    internal_function(&(ko.data[i]));
  }

  dangerous_function(curr_data->buf);
  dangerous_function(curr_data->item);
  kernel_function(curr_data->item);

}
```

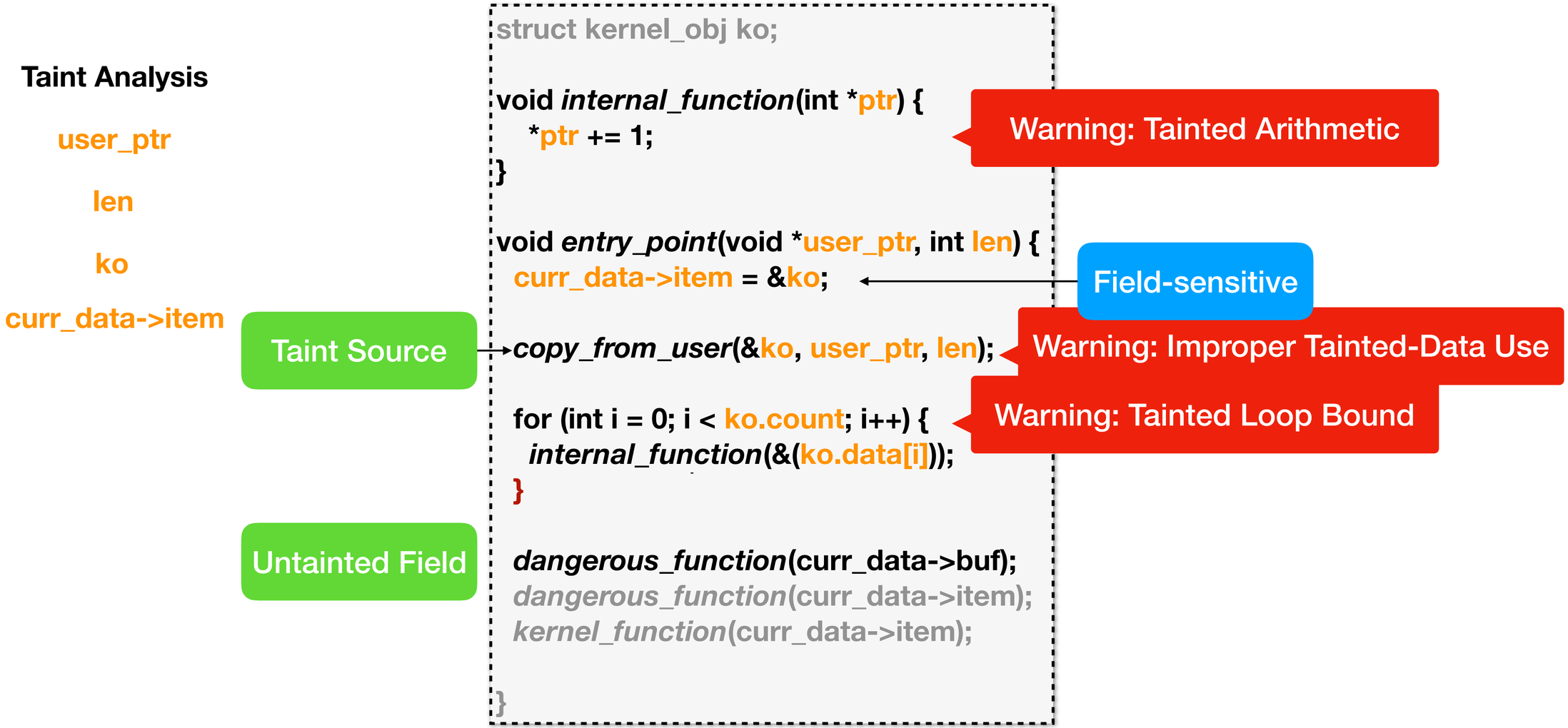
Warning: Tainted Arithmetic

Field-sensitive

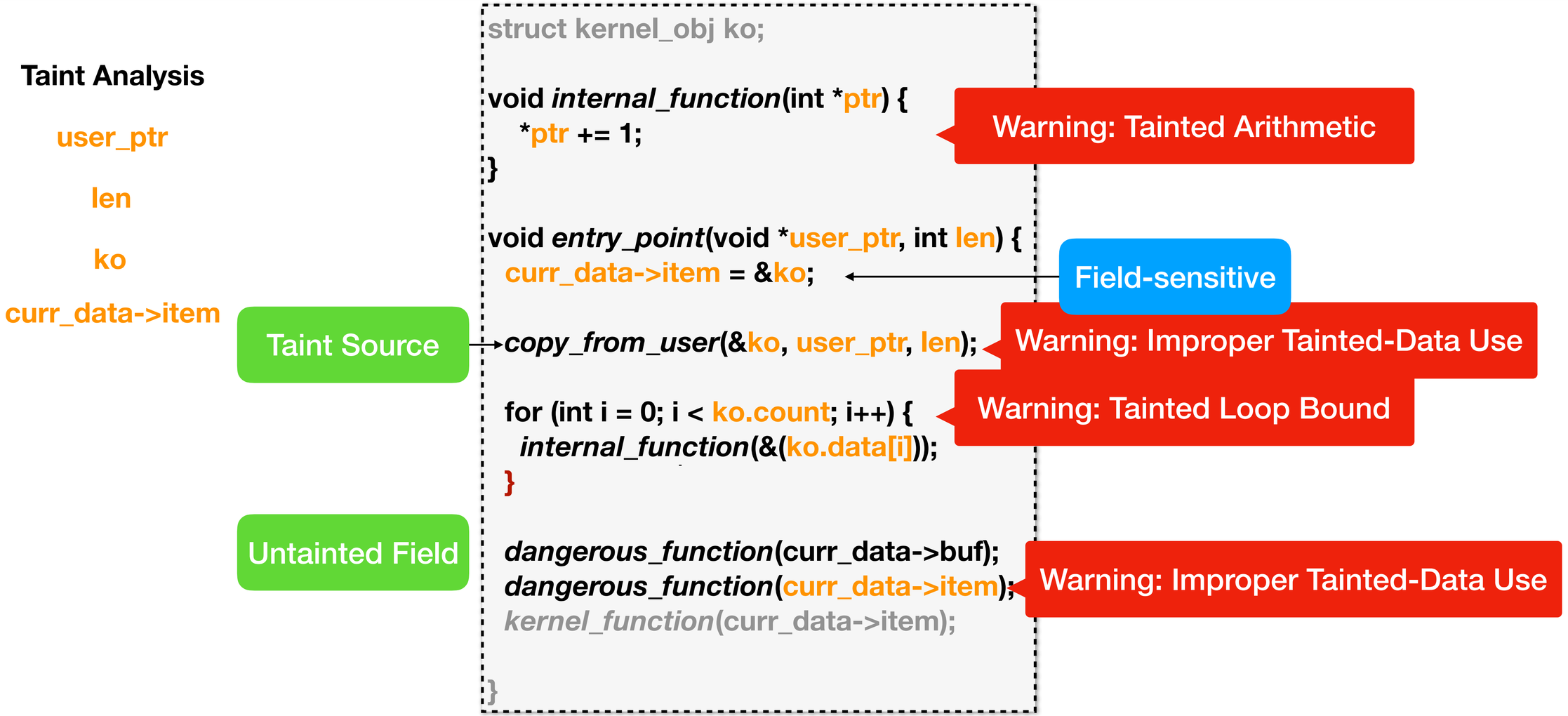
Warning: Improper Tainted-Data Use

Warning: Tainted Loop Bound

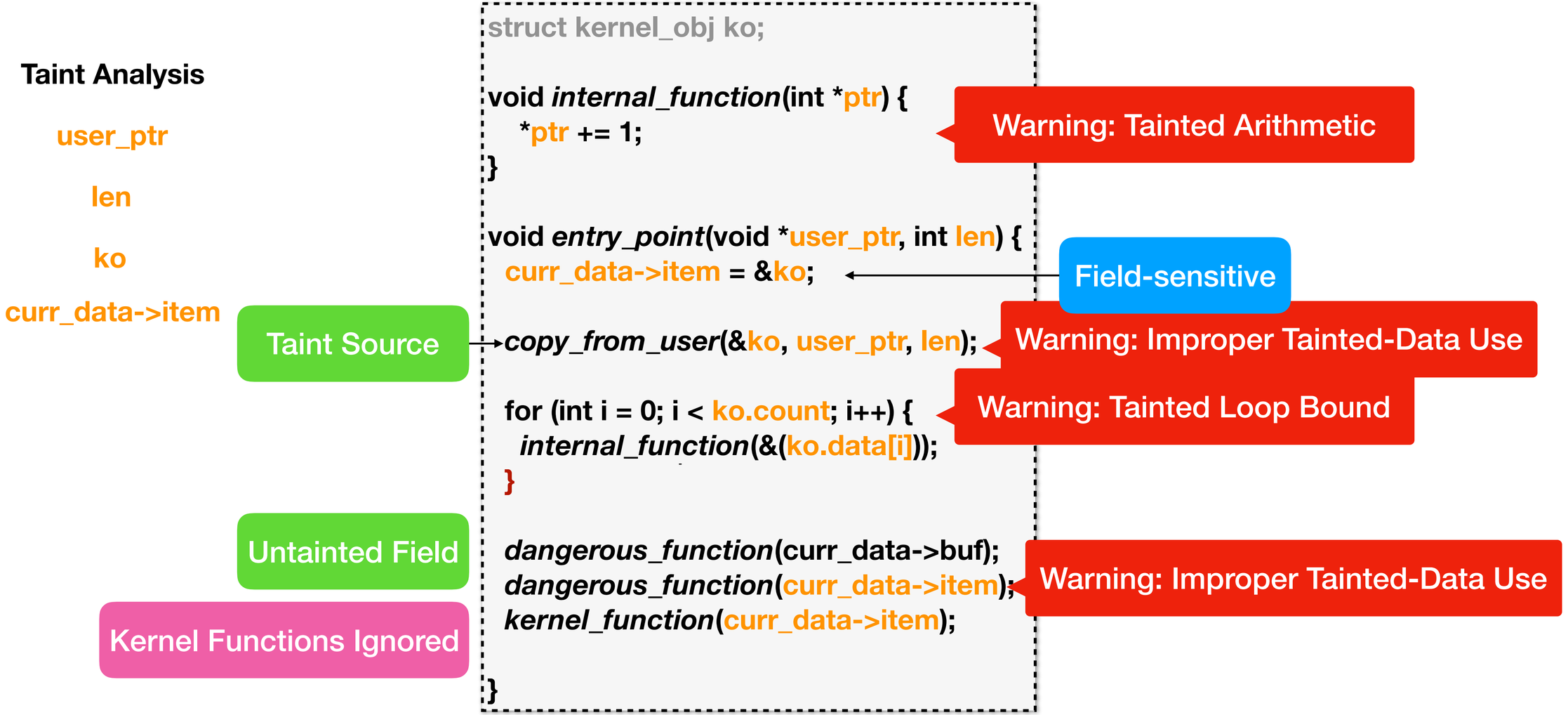
Soundy Driver Traversal



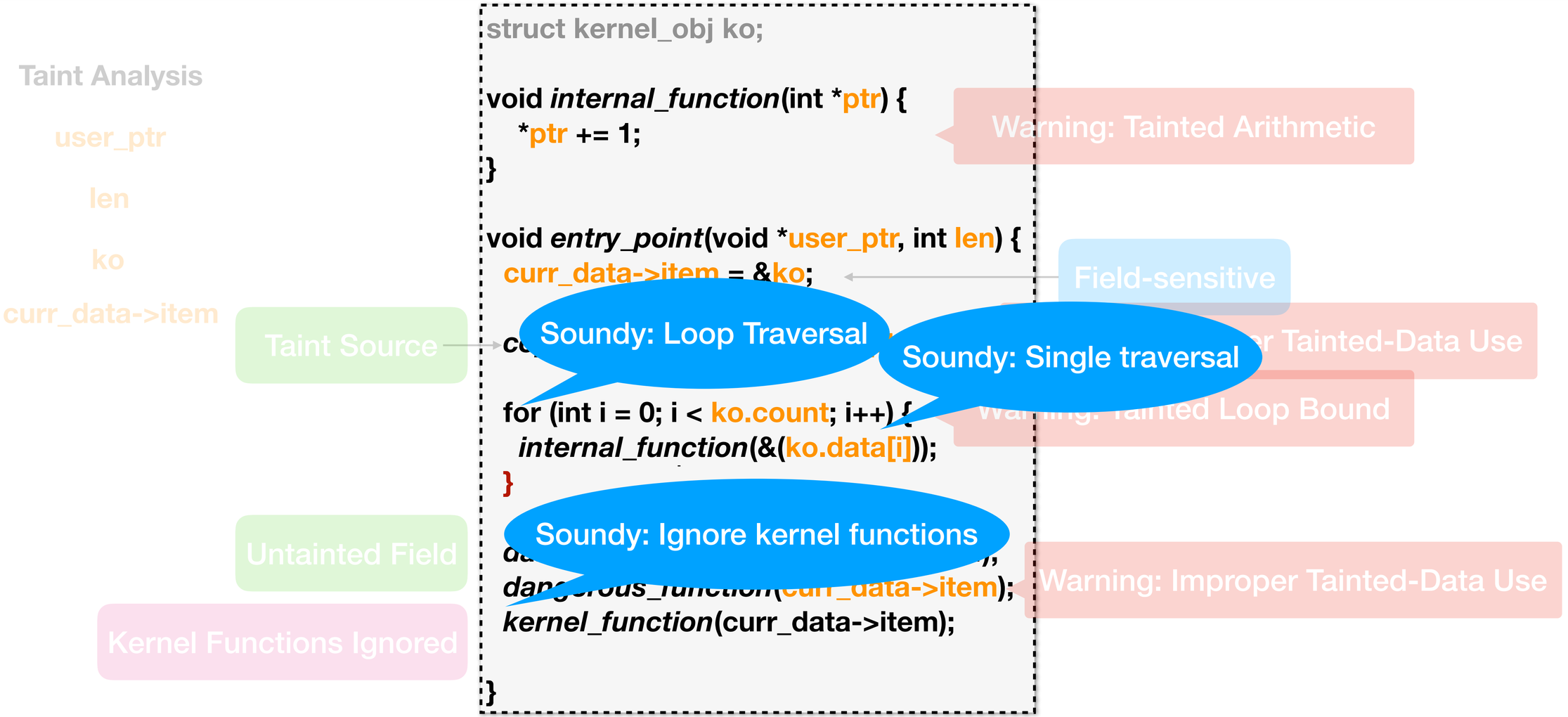
Soundy Driver Traversal



Soundy Driver Traversal

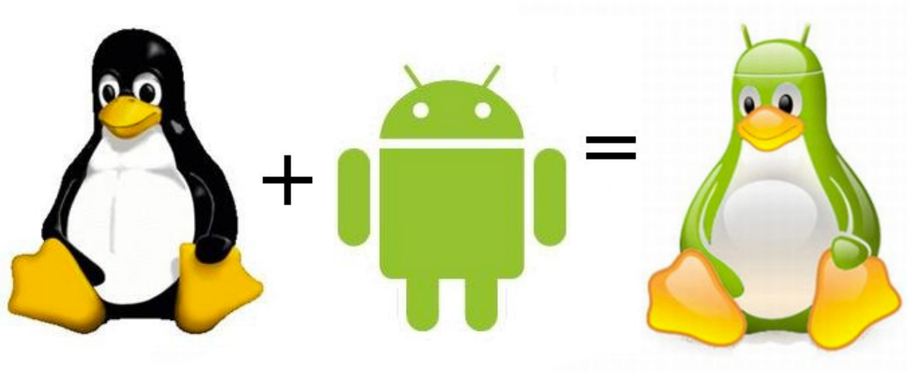


Soundy Driver Traversal



Identifying Vendor Drivers

- *diff* with mainline sources



- Extract code-names from vendor configuration files

Driver Entry Points

- **File Operations**
- **Attribute Operations**
- **Socket Operations**
- **Wrapper Functions**

Entry Type	Argument(s)	Taint Type
Read *(File)*	char **buf*, size_t *len*	Direct
Write *(File)*	char **buf*, size_t *len*	Direct
Ioctl *(File)*	long *args*	Direct
DevStore *(Attribute)*	const char **buf*	Indirect
NetDevIoctl *(Socket)*	struct **ifreq*	Indirect
V4Ioctl	struct v412_format **f*	Indirect

Evaluation: Mobile Kernels

Amazon Echo (5.5.0.3)

Amazon Fire HD8 (6th Generation, 5.3.2.1)

HTC One Hima (3.10.61-g5f0fe7e)

Sony Xperia XA (33.2.A.3.123)

Huawei Venus P9 Lite (2016-03-29)

Samsung Galaxy S7 Edge (SM-G935F NN)

HTC Desire A56 (a56uhl-3.4.0)

LG K8 ACG (AS375)

ASUS Zenfone 2 Laser (ZE550KL / MR5-21.40.1220.1794)

3.1 Million lines of driver code

Other Tools

- **Flawfinder** — pattern-based bug detector
- **RATS (Rough Auditing Tool for Security)** — pattern-based bug detector
- **Sparse** — compiler-based bug detector
- **cppcheck** — all-in-one static analysis bug detector

Other Tools: Analysis

Feature	cppcheck	flawfinder	RATS	Sparse	Dr. Checker
Extensible	✔				✔
Inter-prodecural					✔
Handles pointers					✔
Kernel specific				✔	✔
No manual annotations	✔	✔	✔		✔
Requires compilable sources	✔			✔	✔
Sound					
Tracable Warnings				✔	✔

Other Tools: Warnings

Kernel	cppcheck	flawfinder	RATS	Sparse
Qualcomm	18	4,365	693	5,202
Samsung	22	8,173	2,244	1,726
Hauwei	34	18,132	2,301	11,320
Mediatek	168	14,230	3,730	13,771
	242	**44,900**	**8,968**	**31,929**

Dr. Checker

Warnings per Kernel *(Count / Confirmed / Bug)*

Detector	Huawei	Qualcomm	Mediatek	Samsung	Total
TaintedSizeDetector	62 / 62/ 5	33 / 33 / 2	155 / 155 / 6	20 / 20 / 1	270 / 268 / 14
TaintedPointerDereferenceChecker	522 / 155 / 12	264 / 264 / 3	465 / 459 / 6	479 / 423 / 4	1,760 / 1,301 / 25
TaintedLoopBoundDetector	75 / 56 / 4	52 / 52 / 0	73 / 73 / 1	78 / 78 / 0	278 / 259 / 5
GlobalVariableRaceDetector	324 / 184 / 38	188 / 108 / 8	548 / 420 / 5	100 / 62 / 12	1,160 / 774 / 63
ImproperTaintedDataUseDetector	81 / 74 / 5	92 / 91 / 3	243 / 241 / 9	135 / 134 / 4	551 / 540 / 21
IntegerOverflowDetector	250 / 177 / 6	196 / 196 / 2	247 / 247 / 6	99 / 87 / 2	792 / 707 / 16
KernelUninitMemoryLeakDetector	9 / 7 / 5	1 / 1 / 0	8 / 5 / 5	6 / 2 / 1	24 / 15 / 11
InvalidCastDetector	96 / 13 / 2	75 / 74 / 1	9 / 9 / 0	56 / 13 / 0	236 / 109 / 3
	1,449 / 728 / 78	**901 / 819 / 19**	**1,748 / 1,607 / 44**	**973 / 819 / 24**	**5,071 / 3,973 / 158**

Precision: 78%

Dr. Checker

Warnings per Kernel *(Count / Confirmed / Bug)*

Detector	Huawei	Qualcomm	Mediatek	Samsung	Total
TaintedSizeDetector	62 / 62/ 5	33 / 33 / 2	155 / 155 / 6	20 / 20 / 1	270 / 268 / 14
TaintedPointerDereferenceChecker	522 / 155 / 12	264 / 264 / 3	465 / 459 / 6	479 / 423 / 4	1,760 / 1,301 / 25
TaintedLoopBoundDetector	75 / 56 / 4	52 / 52 / 0	73 / 73 / 1	78 / 78 / 0	278 / 259 / 5
GlobalVariableRaceDetector	324 / 184 / 38	188 / 108 / 8	548 / 420 / 5	100 / 62 / 12	1,160 / 774 / 63
ImproperTaintedDataUseDetector	81 / 74 / 5	92 / 91 / 3	243 / 241 / 9	135 / 134 / 4	551 / 540 / 21
IntegerOverflowDetector	250 / 177 / 6	196 / 196 / 2	247 / 247 / 6	99 / 87 / 2	792 / 707 / 16
KernelUninitMemoryLeakDetector	9 / 7 / 5	1 / 1 / 0	8 / 5 / 5	6 / 2 / 1	24 / 15 / 11
InvalidCastDetector	96 / 13 / 2	75 / 74 / 1	9 / 9 / 0	56 / 13 / 0	236 / 109 / 3
	1,449 / 728 / 78	**901 / 819 / 19**	**1,748 / 1,607 / 44**		**5,071 / 3,973 / 158**

Precision: 78%

Dr. Checker

Warnings per Kernel *(Count / Confirmed / Bug)*

Detector	Huawei	Qualcomm	Mediatek	Samsung	Total
TaintedSizeDetector	62 / 62/ 5	33 / 33 / 2	155 / 155 / 6	20 / 20 / 1	270 / 268 / 14
TaintedPointerDereferenceChecker	522 / 155 / 12	264 / 264 / 3	465 / 459 / 6	479 / 423 / 4	1,760 / 1,301 / 25
TaintedLoopBoundDetector	75 / 56 / 4	52 / 52 / 0	73 / 73 / 1	78 / 78 / 0	278 / 259 / 5
GlobalVariableRaceDetector	324 / 184 / 38	188 / 108 / 8	548 / 420 / 5	100 / 62 / 12	1,160 / 774 / 63
ImproperTaintedDataUseDetector	81 / 74 / 5	92 / 91 / 3	243 / 241 / 9	135 / 134 / 4	551 / 540 / 21
IntegerOverflowDetector	250 / 177 / 6	196 / 196 / 2	247 / 247 / 6	99 / 87 / 2	792 / 707 / 16
KernelUninitMemoryLeakDetector	9 / 7 / 5	1 / 1 / 0	8 / 5 / 5	6 / 2 / 1	24 / 15 / 11
InvalidCastDetector	96 / 13 / 2	75 / 74 / 1	9 / 9 / 0	56 / 13 / 0	236 / 109 / 3
	1,449 / 728 / 78	**901 / 819 / 19**	**1,748 / 1,607 / 44**		**5,071 / 3,973 / 158**

Precision: 78%

Dr. Checker

Warnings per Kernel *(Count / Confirmed / Bug)*

Detector	Huawei	Qualcomm	Mediatek	Samsung	Total
TaintedSizeDetector	62 / 62/ 5	33 / 33 / 2	155 / 155 / 6	20 / 20 / 1	270 / 268 / 14
TaintedPointerDereferenceChecker	522 / 155 / 12	264 / 264 / 3	465 / 459 / 6	479 / 423 / 4	1,760 / 1,301 / 25
TaintedLoopBoundDetector	75 / 56 / 4	52 / 52 / 0	73 / 73 / 1	78 / 78 / 0	278 / 259 / 5
GlobalVariableRaceDetector	324 / 184 / 38	188 / 108 / 8	548 / 420 / 5	100 / 62 / 12	1,160 / 774 / 63
ImproperTaintedDataUseDetector	81 / 74 / 5	92 / 91 / 3	243 / 241 / 9	135 / 134 / 4	551 / 540 / 21
IntegerOverflowDetector	250 / 177 / 6	196 / 196 / 2	247 / 247 / 6	99 / 87 / 2	792 / 707 / 16
KernelUninitMemoryLeakDetector	9 / 7 / 5	1 / 1 / 0	8 / 5 / 5	6 / 2 / 1	**24 / 15 / 11**
InvalidCastDetector	96 / 13 / 2	75 / 74 / 1	9 / 9 / 0	56 / 13 / 0	236 / 109 / 3
	1,449 / 728 / 78	**901 / 819 / 19**	**1,748 / 1,607 / 44**	**973 / 819 / 24**	**5,071 / 3,973 / 158**

Precision: 78%

Dr. Checker

Warnings per Kernel *(Count / Confirmed / Bug)*

Detector	Huawei	Qualcomm	Mediatek	Samsung	Total
TaintedSizeDetector	62 / 62/ 5	33 / 33 / 2	155 / 155 / 6	20 / 20 / 1	270 / 268 / 14
TaintedPointerDereferenceChecl	**522 / 155 / 12**	/ 264 / 3	465 / 459 /	**479 / 423 / 4**	/ 1,301 / 25
TaintedLoopBoundDetector	75 / 56 / 4	52 / 52 / 0	73 / 73 / 1	78 / 78 / 0	278 / 259 / 5
GlobalVariableRaceDetector	324 / 184 / 38	188 / 108 / 8	548 / 420 / 5	100 / 62 / 12	1,160 / 774 / 63
ImproperTaintedDataUseDetector	81 / 74 / 5	92 / 91 / 3	243 / 241 / 9	135 / 134 / 4	551 / 540 / 21
IntegerOverflowDetector	250 / 177 / 6	196 / 196 / 2	247 / 247 / 6	99 / 87 / 2	792 / 707 / 16
KernelUninitMemoryLeakDetector	9 / 7 / 5	1 / 1 / 0	8 / 5 / 5	6 / 2 / 1	24 / 15 / 11
InvalidCastDetector	96 / 13 / 2	75 / 74 / 1	9 / 9 / 0	56 / 13 / 0	236 / 109 / 3
	1,449 / 728 / 78	**901 / 819 / 19**	**1,748 / 1,607 / 44**	**973 / 819 / 24**	**5,071 / 3,973 / 158**

Precision: 78%

Zero-day Bug

```
static char call status ;
...
static ssize_t accdet_store_call_state( struct device driver *ddri , const char *buf , size t count) {

   int ret = sscanf(buf, "%s", &call status);

   if (ret != 1) {
      ACCDETDEBUG("accdet: Invalid values\n");
      return -EINVAL;
    }

   …
}
```

A buffer overflow bug detected in Mediatek's Accdet driver

Zero-day Bug

buf can contain more than one char !

```
static char call status ;
...
static ssize_t accdet_store_call_state( struct device driver *ddri , const char *buf , size t count) {

   int ret = sscanf(buf, "%s", &call status);

   if (ret != 1) {
      ACCDETDEBUG("accdet: Invalid values\n");
      return -EINVAL;
    }

   …
}
```

A buffer overflow bug detected in Mediatek's Accdet driver

Zero-day Bug

```
static char call status ;
...
static ssize_t accdet_store_call_state( struct device driver *ddri , const char *buf , size t count) {

   int ret = sscanf(buf, "%s", &call status);

   if (ret != 1) {
      ACCDETDEBUG("accdet: Invalid values\n");
      return -EINVAL;
    }

   …
}
```

buf can contain more than one char !

Warning: Improper Tainted-Data Use

A buffer overflow bug detected in Mediatek's Accdet driver

Zero-day Bug

```
static char call status ;
...
static ssize_t accdet_store_call_state( struct device driver *ddri , const char *buf , size t count) {

   int ret = sscanf(buf, "%s", &call status);

   if (ret != 1) {
      ACCDETDEBUG("accdet: Invalid values\n");
      return -EINVAL;
    }

   …
}
```

buf can contain more than one char !

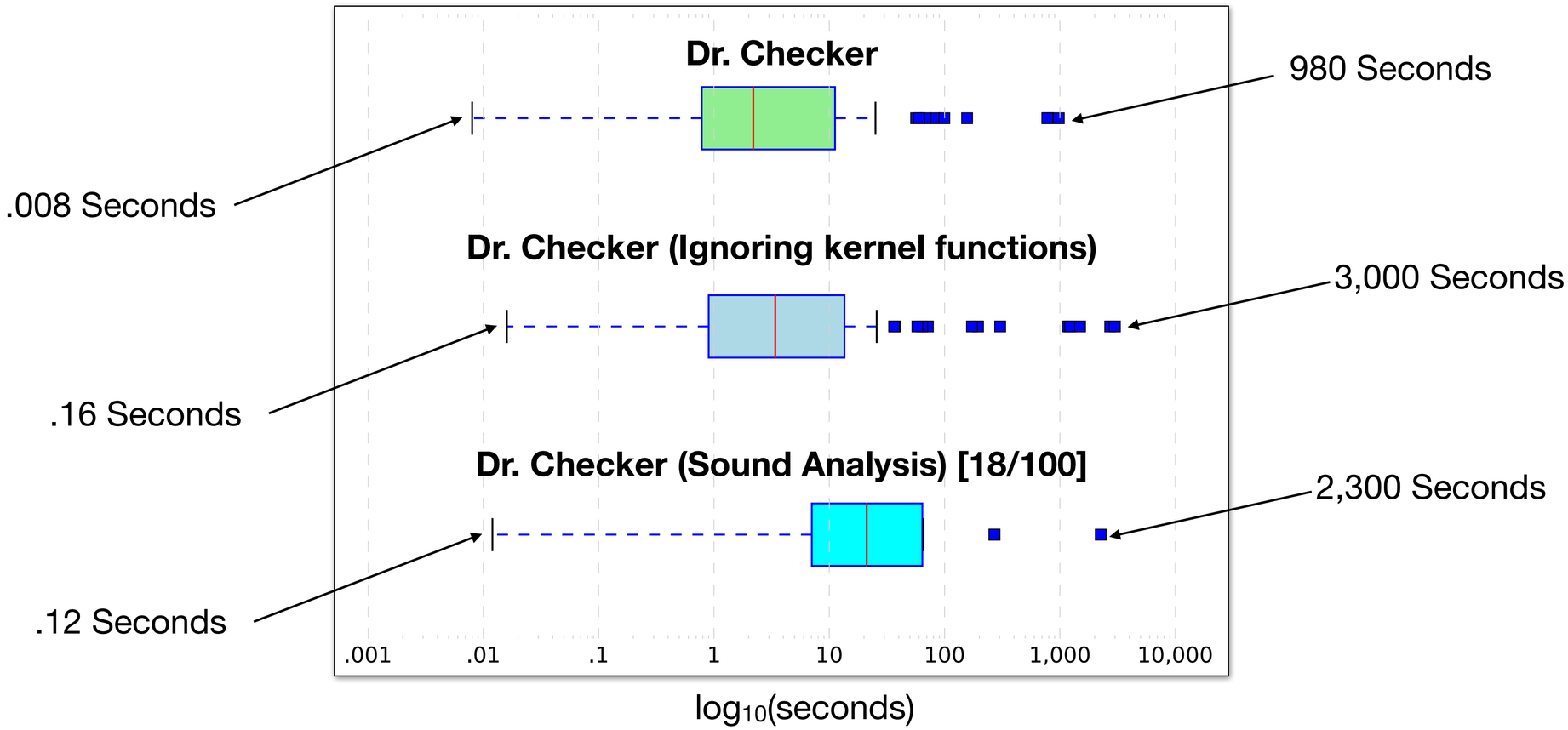
Warning: Improper Tainted-Data Use

ret is checked, but it's too late

A buffer overflow bug detected in Mediatek's Accdet driver

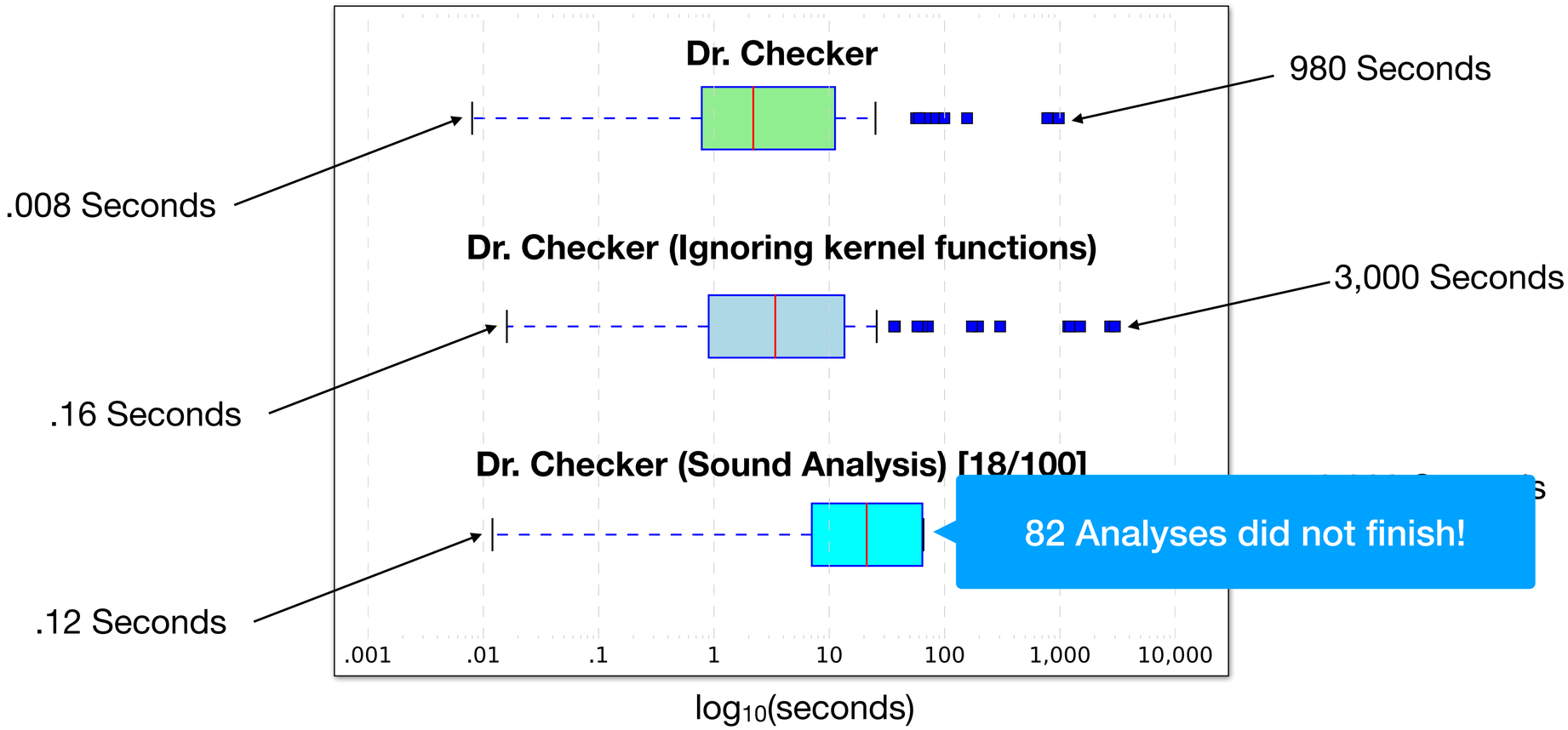
Results: Soundy vs. Sound

Time to analyze 100 randomly selected entry points



Results: Soundy vs. Sound

Time to analyze 100 randomly selected entry points



Conclusion

- **Modular bug-finding tool for Linux kernel drivers**
- **Soundy program analysis techniques to maintain practicality**
- **Scalable tool capable of employing multiple vulnerability detectors**
- **158 previously undiscovered zero-day bugs**
- **Open-source project to encourage more development/collaboration**

Help Make Drivers Great Again

github.com/ucsb-seclab/dr_checker

Aravind Machiry (machiry@cs.ucsb.edu)

Chad Spensky (cspensky@cs.ucsb.edu)