Detecting Callback Related Deep Vulnerabilities in Linux Device Drivers

Dr. Tuba Yavuz
Assistant Professor (tuba@ece.ufl.edu)
University of Florida
Outline

• Background & The Problem
• Approach: Staged Static Analysis
• Evaluation
• Related Work
• Conclusion and Future Directions
What is a callback?

• A software framework defines an Application Programming Interface (API) to support the development of application components.
• An extensible/event-based framework defines a set of events that should be handled by the application components.
• The framework also defines for each event the signature of a special function called the **callback**.
• Any application that wants to handle an event needs to implement a callback and register it with the framework.
• The generic part of the framework, some API function, calls the callback.
The Challenge with Callbacks

some_APP_function

do_something1()
some_API()
do_something3()

some_API

... funcptr_to_some_APP_callback()
...

some_APP_callback

... do_something2()
...

Application
Framework
Indirect call
The Challenge with Callbacks

What if sth1 & sth2 conflict?

some_APP_function
  do_something1()
some_API()
do_something3()

some_APP_callback
  ...
do_something2()
  ...

some_API
  ...
funcptr_to_some_APP_callback()
  ...

Application
Framework
Indirect call
The Challenge with Callbacks

What if sth2 & sth3 conflict?

some_APP_function

- do_something1()
- some_API()
- do_something3()

some_APP_callback

- ...
- do_something2()
- ...

some_API

- ...
- funcptr_to_some_APP_callback()
- ...

Application

Framework

Indirect call
The Challenge with Callbacks

What if the call chain is deep?

some_APP_function

- do_something1()
- some_API()
- do_something3()

some_API

- ... some_other_framework_func()
- ...

some_APP_callback

- ...
- do_something2()
- ...

Application

Framework

Indirect call
A Real Callback Related Deep Vulnerability

- `usb_audio_probe`
  - `snd_usb_create_streams()`
  - `snd_card_free()`

- `snd_card_free`
  - `kobject_put()`

- `snd_usb_free`
  - `kfree(umidi)`

- `snd_usbmidrawmidi_free`
  - `snd_usbmidrawmidi_free()`

Double-free vulnerability
CVE-2016-2384
Outline

• Background & The Problem
• **Approach: Staged Static Analysis**
• Evaluation
• Related Work
• Conclusion and Future Directions
Approach

• Static Analysis
  o No need for hardware or emulation

• Staged
  o Apply imprecise and scalable analysis to extract a model of the interaction between framework and application modules
  o Apply precise analysis on the application module(s) under the guidance of the extracted model
MOXCAFE Tool

[Diagram showing the workflow of the MOXCAFE Tool]

1. Collect Metadata: (call graphs, function pointer types, calls via function pointers)
2. Extract Implicit Calls
3. Pass I: Summary Mode
   - Inter-procedural path-sensitive analysis
4. Pass II: Summary Aware Mode
   - Inter-procedural path-sensitive analysis
5. Bug Report

[Diagram showing the connections between the steps]
Metadata Collection

• Function pointer definitions
  o Callback Types in the form (type,field)
    • e.g., (struct snd_rawmidi, private_free)
  o Callback Signatures in the form of a function signature
    • e.g., void (*) (struct kref *)

• Function pointer bindings
  o Data structure initializations
  o Assignment statements in function bodies

• Function pointer call sites
  o Which function gets passed as an actual argument to which functions?
Extracting Implicit Calls

• The goal is to construct an extended call graph of the application modules

• An extended call graph includes all nodes and edges in a basic call graph of the application module

• An extended call graph also has special edges that summarize call chains from APIs to application callbacks
A basic call graph

usb_audio_probe

snd_usb_create_streams

snd_usb_create

kfree

snd_usbmidi_free

snd_usbmidi_rawmidi_free

snd_card_free

snd_midi

sound subs.

kernel subs.

Indirect call
A call chain from an API to a callback

... kobject_put() ...

... snd_card_free ...

... snd_usbmidi_rawmidi_free ...

... snd_usbmidi_free() ...

Indirect call

sound subs.
kernel subs.
usb midi
An extended call graph (ECG)

- `usb_audio_probe`
- `snd_usb_create_streams`
- `snd_usb_create`
- `kfree`
- `snd_usbmid create`
- `snd_usbmid free`
- `snd_card_free`
- `snd_usbmid rawmidi_free`
Constructing an ECG

- Start from the **global basic call graph**, which is a combination of the basic call graphs of the application modules and the framework modules.
- Find pairs of \((f_i, \text{CB type}_k)\) and \((f_j, \text{CB sig}_m)\), where \(f_i\) calls a callback of \(\text{CB type}_k\) and \(f_j\) calls a callback that implements the signature \(\text{CB sig}_m\).
- Propagate \((f_i, \text{CB type}_k)\)s and \((f_j, \text{CB sig}_m)\)s through the global basic call graph and infer new \((f_i, \text{CB type}_k)\)s and \((f_j, \text{CB sig}_m)\)s.
- TC consists of \((f_i, \text{CB type}_k)\)s.
- SC consists of \((f_j, \text{CB sig}_m)\)s.
Propagating \((f, CB\ Type)\)

**Given**

\[
TC = \{(f_2, CBT_3),
        (f_1, CBT_2),
        \ldots\}
\]

**Updated**

\[
Type(f_2) = CBT_2
\]
Propagating \((f, \text{ CB Type})\)

Given

\[ TC = \{(f_2, \text{CBT}_3), (f_1, \text{CBT}_2), \ldots\} \]

\[ \text{Type}(f_2) = \text{CBT}_2 \]

Updated

\[ TC = \{(f_2, \text{CBT}_3), (f_1, \text{CBT}_2), (f_1, \text{CBT}_3), \ldots\} \]

\[ \text{CG} = \{(f_0, f_1), \ldots\} \]
Propagating \((f, \text{ CB Type})\)

**Given**

\[ \text{TC} = \{(f_2, \text{CBT}_3), (f_1, \text{CBT}_2), \ldots\} \]

\[ \text{Type}(f_2) = \text{CBT}_2 \]

**Updated**

\[ \text{TC} = \{(f_2, \text{CBT}_3), (f_1, \text{CBT}_2), (f_1, \text{CBT}_3), \ldots\} \]

\[ \text{TC} = \{(f_2, \text{CBT}_3), (f_1, \text{CBT}_2), (f_1, \text{CBT}_3), (f_0, \text{CBT}_3), \ldots\} \]

\[ \text{CG} = \{(f_0, f_1), \ldots\} \]

Implements the callback type (TC)
Propagating (f, CB Sig)

Given

Updated

\[ SC = \{(f_2, CBS_3), (f_1, CBS_2), \ldots\} \]

\[ SP = \{(f_1, f_2), \ldots\} \]

\[ \text{Sig}(f_2) = CBS_2 \]
Propagating \((f, \text{CB Sig})\)

**Given**

\[
\text{SC} = \{(f_2, \text{CBS}_3), (f_1, \text{CBS}_2), \ldots\} \\
\text{SP} = \{(f_1, f_2), \ldots\} \\
\text{Sig}(f_2) = \text{CBS}_2
\]

**Updated**

\[
\text{SC} = \{(f_2, \text{CBS}_3), (f_1, \text{CBS}_2), \ldots\} \\
\text{CG} = \{(f_0, f_1), (f_1, f_2) \ldots\}
\]

- \text{f}_0 \quad \text{f}_1 \quad \text{CBS}_2 \quad \text{CBS}_3

---

Implements the callback signature (SC)

Gets as an argument (SP)
Propagating \((f, CB\ Sig)\)

- **Implements the callback signature \((SC)\)**
- **Gets as an argument \((SP)\)**

### Given

- \(SC = \{(f_2, CBS_3), (f_1, CBS_2), \ldots\}\)
- \(SP = \{(f_1, f_2), \ldots\}\)
- \(Sig(f_2) = CBS_2\)

### Updated

- \(SC = \{(f_2, CBS_3), (f_1, CBS_2), (f_1, CBS_3), \ldots\}\)
- \(SC = \{(f_2, CBS_3), (f_1, CBS_2), (f_1, CBS_3), \ldots\}\)
- \(CG = \{(f_0, f_1), (f_1, f_2), \ldots\}\)
- \(CG = \{(f_0, f_1), (f_1, f_2), \ldots\}\)
A call chain: API - App CB

1. `snd_usbmidi_rawmidi_free`
2. `snd_do_card_free`
3. `release_card_device`
4. `(struct device, release)`
5. `(struct kobj_type, release)`
6. `void (*) (struct kref*)`
7. `kref_put`
8. `kobject_put`
9. `kobject_release`
10. `kobject_cleanup`
11. `device_release`
12. `(struct snd_rawmidi, private_free)`

Flow:
- `TC`
- `CG`
- `SP`
A call chain: API - App CB

1. **snd_usbmidirawmidi_free**
2. **snd_do_card_free**
3. **release_card_device**
4. **(struct device, release)**
5. **(struct kobj_type, release)**
6. **void (*) (struct kref*)**
7. **kobject_release**
8. **kobject_cleanup**
9. **device_release**
10. **(struct snd_rawmidi, private_free)**

Connections:
- TC from **snd_do_card_free** to **release_card_device**
- CG from **release_card_device** to **snd_do_card_free**
- TC from **(struct device, release)** to **device_release**
- TC from **(struct kobj_type, release)** to **kobject_cleanup**
- SC from **void (*) (struct kref*)** to **kref_put**
- SP from **kref_put** to **kobject_release**
- CG from **kobject_release** to **kref_put**
- CG from **kobject_cleanup** to **kobject_release**
A call chain: API - App CB

snd_usbmi_di_rawmidi_free

snd_do_card_free

release_card_device

(struct device, release)

(struct kobj_type, release)

(void (*)(struct kref*))

snd_card_free

device_release

kobject_release

kobject_cleanup

kobject_put

kref_put

snd_card_free

(struct snd_rawmidi, private_free)
A call chain: API - App CB

- `snd_usbmidi_rawmidi_free`
- `snd_do_card_free`
- `release_card_device`
- `(struct device, release)`
- `(struct kobj_type, release)`
- `void (*) (struct kref*)`
- `kobject_cleanup`
- `device_release`
- `kobject_release`
- `kobject_put`
- `kref_put`
- `snd_card_free`
A call chain: API - App CB

snd_usbmidi_rawmidi_free

snd_do_card_free

release_card_device

(struct snd_rawmidi, private_free)

device_release

kobject_cleanup

kobject_release

kref_put

snd_card_free

void (*) (struct kref*)

(struct device, release)

(struct kobj_type, release)
A call chain: API - App CB

- snd_usbmidi_rawmidi_free
  - snd_do_card_free
  - release_card_device
  - (struct device, release)
- (struct kobj_type, release)
- void (*)(struct kref*)
  - kref_put
  - kobject_put
  - kobject_release
  - kobject_cleanup
  - device_release
  - (struct snd_rawmidi, private_free)
A call chain: API - App CB

 snd_usbmidi_rawmidi_free

 snd_do_card_free

 release_card_device

 (struct device, release)

 (struct kobj_type, release)

 void (*)(struct kref*)

 device_release

 kobject_cleanup

 kobject_release

 kref_put

 snd_card_free

 kref_put

 kobject_release

 kobject_cleanup

 device_release

 release_card_device

 snd_do_card_free

 snd_usbmidi_rawmidi_free

 (struct snd_rawmidi, private_free)
An extended call graph (ECG)

- `usb_audio_probe`
- `snd_usb_create_streams`
- `snd_usb_create`
- `kfree`
- `snd_usb_create_free`
- `snd_usbmidmidi_free`
- `snd_card_free`
- `snd_usbmidmidi_rawmidi_free`

- `usb midi` (gray)
- `sound subs.` (blue)
- `kernel subs.` (blue)
- `Indirect call` (dashed line)
MOXCAFE Tool

Diagram:

1. Framework Database
2. Application Modules
3. Collect Metadata: (call graphs, function pointer types, calls via function pointers)
4. Extract Implicit Calls
5. Metadata
6. Metadata + Extended Call Graph
7. Path Summaries
8. Pass I: Summary Mode
   - Inter-procedural path-sensitive analysis
9. Pass II: Summary Aware Mode
   - Inter-procedural path-sensitive analysis

University of Florida
Detecting Deep Bugs

- Pass I (Summary Mode): Perform path-sensitive static analysis on the application module(s) and summarize bug relevant data-flow inside callbacks
- Pass II (Summary-aware Mode): Perform path-sensitive static analysis on the application module(s) while consulting the Extended Call Graph and integrating the data-flow summaries of callbacks at callsites of API, which may call some application callbacks
- Focused on double-free and double-locking in this paper
Path-sensitive Analysis

• Let IPS denote the generic state representation for a generic inter-procedural path-sensitive analysis
• We extend IPS with metadata that enables detection of double-free and double-locking bugs
• The idea is to record the history of bug-relevant operations
• Both summary mode and the summary-aware mode use the same extended IPS
• We use type information while tracking the metadata
Metadata for double-free

- AL: Type of objects allocated and not freed
- FR: Type of objects that have been freed
- FRA: Type of objects that have been freed without prior allocation
- Each path is extended with these three sets
- For any path that starts from a callback function, a path summary is recorded using (AL, FR, FRA)
Metadata for double-locking

• ACQ: Type of objects acquired and not released
• RL: Type of objects released
• ACQR: Type of objects acquired without a prior release
• Each path is extended with these three sets
• For any path that starts from a callback function, a path summary, is recorded using (ACQ, ACQR, RL)
Summary Mode Example

Path Summary 1:
AL = {...}
FR = FRA = {struct snd_usb_midi}

Path Summary 2:
...
Summary-Aware Mode Example

```
AL = {struct snd_usb_midi}
FR = {} FRA = {}
```

```
snd_usb_create_streams()
AL = {struct snd_usb_midi}
FR = {struct snd_usb_midi} FRA = {}
```

```
snd_card_free()
```

Callback
```
snd_usbmidrawmidi_free
```

Path Summary 1:
```
AL = {...} 
FR = FRA = {struct snd_usb_midi}
```

Path Summary 2:
```
... 
```

(\(AL^2, FR^2, FRA^2\)) (\(AL^1, FR^1, FRA^1\))

Assume callback not executed

Assume callback executed & propagate path summaries

usb_audio_probe
Implementation

- MOXCAFE has been implemented on top of the LLVM compiler framework
- Metadata collection on top of Clang AST Analyzer
- Summary and Summary-aware mode on top of Clang Static Analyzer
- Linux kernel 4.14-rc2 was used and compiled with the clang compiler using the `clang-kernel-build` project
Outline

• Background & The Problem
• Approach: Staged Static Analysis
• Evaluation
• Related Work
• Conclusion and Future Directions
Evaluation

• Applied MOXCAFE to 40 Linux device drivers from a variety of kernel subsystems
  o Video, sound, gadget, dwc3, serial, tty, block, scsi, network, usb core, input, and hid

• Extended Call Graph Generation
  o 18% false positive rate for the inferred API to callback edges
  o False positives due to infeasible data-flow

• Deep bug detection
  o Detected 4 new and 2 known callback related deep vulnerabilities
  o 3 double-free, 3 double-locking
  o 60% false positive rate
  o Comparison with an extended call graph generated based on DSA’s call graph
    • Could only detect 2 deep bugs
Timing Evaluation (in secs)

<table>
<thead>
<tr>
<th>Stages</th>
<th>Min</th>
<th>Max</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metadata Collection</td>
<td>9.65</td>
<td>749.30</td>
<td>207.53</td>
</tr>
<tr>
<td>Extract Implicit Calls</td>
<td>0.06</td>
<td>19.40</td>
<td>3.62</td>
</tr>
</tbody>
</table>

![Box plot showing comparison between MOXAFE ECG and DSA ECG for PASS I and PASS II](image-url)
Double-free in usbtv: CVE-2017-17975

API to callback chain:

- **v4l2_device_put**
- **kref_put**
- **v4l2_device_release**

Framework callback: **v4l2_device_release**

Application callback: (struct v4l2_device, release)
Double-free in f_midi CVE-2018-20961

API to callback chain:

1. usb_ep_queue
2. DWC3_gadget_ep_queue
3. __dwc3_gadget_ep_queue
4. __dwc3_gadget_kick_request
5. DWC3_gadget_giveback
6. usb_gadget_giveback_request

Framework callback

Application callback
(struct usb_request, complete)
Double-locking in f_hid CVE-2019-14763

API to callback chain:

- usb_ep_queue
- dwc3_gadget_ep_queue
  - __dwc3_gadget_ep_queue
  - __dwc3_gadget_kick_request
  - dwc3_gadget_giveback
    - usb_gadget_giveback_request

Framework callback

Application callback
(struct usb_request, complete)
Double-free in f_loopback

API to callback chain:

1. `usb_ep_disable`
2. `dwc3_gadget_ep_disable`
3. `__dwc3_gadget_ep_disable`
4. `dwc3_remove_requests`
5. `dwc3_gadget_giveback`
6. `usb_gadget_giveback_request`

mouseover:

```
usbsb_ep_disable

Framework callback
```

```
dwc3_gadget_ep_disable

Framework callback
```

```
__dwc3_gadget_ep_disable
```

```
dwc3_remove_requests
```

```
dwc3_gadget_giveback
```

```
usb_gadget_giveback_request
```

mouseover:

```
(Application callback
(struct usb_request, complete)

mouseover:

```

mouseover:

```

mouseover:

```

mouseover:

```

mouseover:

```

mouseover:

```
Secure Development with Callbacks

- More documentation needed!!!
  - Documentation for API
    - The type of callbacks involved
    - Conditions when callbacks get executed
      - e.g., while holding a lock, on error paths, etc.
  - Documentation for Applications
    - Pre and post conditions
    - Calling contexts

- Secure Interface and Data Structure Design
  - Facilitating application of secure programming practices while using callbacks
    - Not so easy!!!
  - Requires careful design of the callback signatures and the data structures that involve callbacks
How to set the pointer to null after free?

Would changing the parameter into a double pointer work?

```c
#include <sound/usb/midi.h>

// in sound/usb/midi.c
// the callback function
int snd_usbmidi_rawmidi_free(struct snd_rawmidi *rmidi)
{
    struct snd_usb_midi *umidi = rmidi->private_data;
    snd_usbmidi_free(umidi);
    ...
}

// in sound/usb/midi.c
// helper function performing the deallocation
void snd_usbmidi_free(struct snd_usb_midi *umidi)
{
    ...
    kfree(umidi);
}
```
Outline

• Background & The Problem
• Approach: Staged Static Analysis
• Evaluation
• Related Work
• Conclusion and Future Directions
Related Work

• Milanova et al. SCAM’02
  o Our results supports their empirical evidence that inexpensive pointer analysis is sufficient for call graph construction

• Lattner et al., PLDI’07, Data Structure Analysis (DSA)
  o A context-sensitive, flow insensitive, and a field sensitive points-to analysis
  o Known to scale to the Linux kernel and used in some modern program analysis tools
  o Loses field sensitivity when casting is involved leading to imprecision
  o Incompleteness is suspected due to inability to handle pointer arithmetic
    • The widely used container_of macro in the Linux kernel

• Tang et al., POPL’15
  o Summarizes data-flows in library calls that involve callbacks
  o Our approach abstracts away data flow in library calls and focuses on the summarization of the callbacks only to achieve scalability
Outline

• Background & The Problem
• Approach: Staged Static Analysis
• Evaluation
• Related Work
• Conclusion and Future Directions
Conclusion

- Imprecise points-to analysis is sufficient for constructing call graphs of C programs.
- Stages static analysis is effective in detecting callback related deep vulnerabilities and scales to the Linux kernel.
- Avoiding security problems for callback based frameworks requires documentation for the application and the framework and a careful design of the API signatures and the generic data structures.
Future Directions

• Extending the Clang static analyzer to handle multiple modules
• API modeling
• Incorporating knowledge of the programming model to points-to analysis
• Extending the application to other system software such as implementations of communication protocols
Questions?

Thank you for your attention