Egg Hunt in Tesla Infotainment: A First Look at Reverse Engineering of Qt Binaries

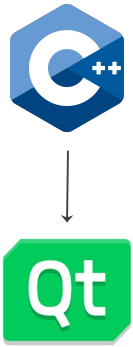
Haohuang Wen and Zhiqiang Lin

Department of Computer Science and Engineering
The Ohio State University

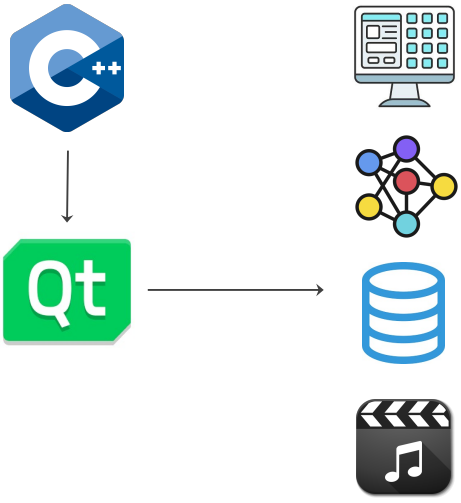
Aug 10^{th}, 2023

In Proceedings of the 32nd USENIX Security Symposium (USENIX Security'23), Anaheim, CA, August 2023

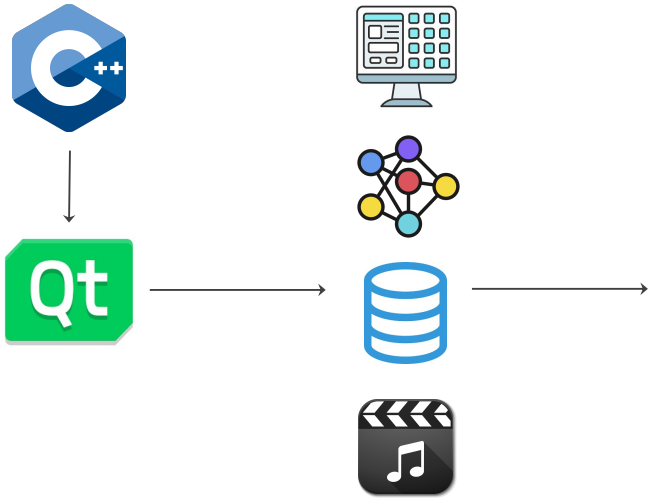
Background



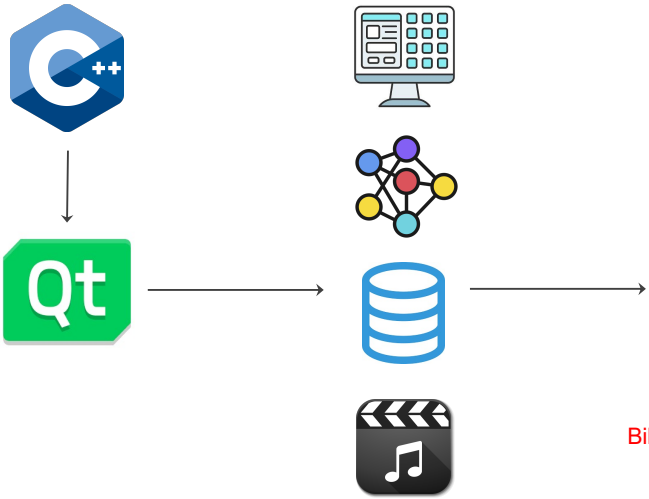
Background



Background



Background



Billions of applications across 70+ industries!

How Popular is Qt?

Name	Category	# Repository	%
Qt	Framework	45,635	35.70%
ROS	Robotics	16,796	13.14%
Boost	Framework	6,205	4.85%
MFC	Framework	4,409	3.45%
Cocos2d	Game Engine	3,587	2.81%
OpenFrameworks	Framework	3,264	2.55%
JUCE	Framework	2,204	1.72%
PCL	Robotics	1,719	1.34%
imgui	GUI	1,557	1.22%
wxWidgets	GUI	1,076	0.84%
Cinder	Framework	1,042	0.82%
Allegro	Game Engine	958	0.75%
Godot	Game Engine	682	0.53%
GamePlay	Game Engine	561	0.44%
dlib	Framework	547	0.43%

Table: Top 15 most popular C++ frameworks among all open-sourced repositories from GitHub.

Motivation

Enabling Security Analysis of Qt Programs

- Reverse engineering (RE) is one of the keys to vet Qt binaries

Motivation

Enabling Security Analysis of Qt Programs

- Reverse engineering (RE) is one of the keys to vet Qt binaries
- Existing C++ binary analysis tools can be applied [ida, ghi, SWS+16]

Motivation

Enabling Security Analysis of Qt Programs

- Reverse engineering (RE) is one of the keys to vet Qt binaries
- Existing C++ binary analysis tools can be applied [ida, ghi, SWS+16]

Binary RE Challenges

- **Control Flow Graph (CFG) Recovery**. Indirect control flow transfers such as callbacks and indirect calls [PCvdV+17, VDVGC+16]

Motivation

Enabling Security Analysis of Qt Programs

- Reverse engineering (RE) is one of the keys to vet Qt binaries
- Existing C++ binary analysis tools can be applied [ida, ghi, SWS+16]

Binary RE Challenges

- **Control Flow Graph (CFG) Recovery**. Indirect control flow transfers such as callbacks and indirect calls [PCvdV+17, VDVGC+16]
- **Symbol Recovery** (e.g., names/types of functions/variables). Code stripping during binary compilation [TTN+19, SCD+18]

Key Insights

Unique Insights from Qt's Mechanisms

Key Insights

Unique Insights from Qt's Mechanisms

1. Qt's Signal and Slot

Key Insights

Unique Insights from Qt's Mechanisms

1. Qt's Signal and Slot
   - Originally designed for efficient function callback implementation among GUIs

Key Insights

Unique Insights from Qt's Mechanisms

1. Qt's Signal and Slot
   - Originally designed for efficient function callback implementation among GUIs
   - *We instead leverage it to identify Qt-specific function callbacks*

Key Insights

Unique Insights from Qt's Mechanisms

1. Qt's Signal and Slot
   - Originally designed for efficient function callback implementation among GUIs
   - *We instead leverage it to identify Qt-specific function callbacks*
2. Qt's Dynamic Introspection

Key Insights

Unique Insights from Qt's Mechanisms

1. Qt's Signal and Slot
   - Originally designed for efficient function callback implementation among GUIs
   - *We instead leverage it to identify Qt-specific function callbacks*
2. Qt's Dynamic Introspection
   - Originally designed for run-time class member query and update

Key Insights

Unique Insights from Qt's Mechanisms

1. Qt's Signal and Slot
   - Originally designed for efficient function callback implementation among GUIs
   - *We instead leverage it to identify Qt-specific function callbacks*
2. Qt's Dynamic Introspection
   - Originally designed for run-time class member query and update
   - *We repurpose it to recover rich semantic symbols from the binary program*

Qt's Signal and Slot Mechanism

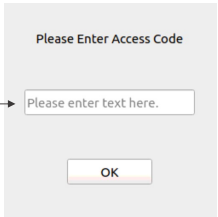
```
1  MainWindow::MainWindow() {
2     ...
3     // Create lineEdit instance
4     v0 = operator.new(0x30)
5     QLineEdit(v0)
6     *(this + 0x30) = v0
7     ...
8
9
10
11
12
13
14
15 }
```

Qt's Signal and Slot Mechanism

```
1  MainWindow::MainWindow() {
2     ...
3     // Create lineEdit instance
4     v0 = operator.new(0x30)
5     QLineEdit(v0)
6     *(this + 0x30) = v0
7     ...
8     // Register callbacks
9     connect(*(this+0x30),"2textChanged(QString)"
10           , this, "1updateText(QString)", 0)
11
12    connect(*(this+0x30),"2editingFinished()"
13           , this, "1handleInput()", 0)
14    ...
15 }
```

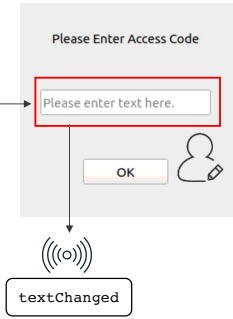
Qt's Signal and Slot Mechanism

```
1  MainWindow::MainWindow() {
2     ...
3     // Create lineEdit instance
4     v0 = operator.new(0x30)
5     QLineEdit(v0)
6     *(this + 0x30) = v0
7     ...
8     // Register callbacks
9     connect(*(this+0x30),"2textChanged(QString)"
10            , this, "1updateText(QString)", 0)
11
12    connect(*(this+0x30),"2editingFinished()"
13            , this, "1handleInput()", 0)
14    ...
15 }
```



Qt's Signal and Slot Mechanism

```
1  MainWindow::MainWindow() {
2     ...
3     // Create lineEdit instance
4     v0 = operator.new(0x30)
5     QLineEdit(v0)
6     *(this + 0x30) = v0
7     ...
8     // Register callbacks
9     connect(*(this+0x30),"2textChanged(QString)"
10            , this, "1updateText(QString)", 0)
11
12    connect(*(this+0x30),"2editingFinished()"
13            , this, "1handleInput()", 0)
14    ...
15 }
```

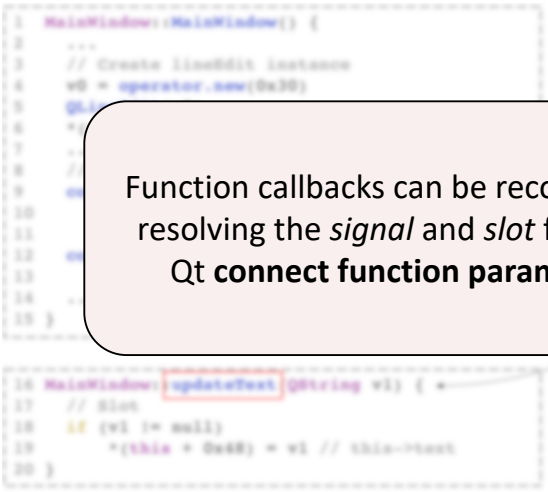


Qt's Signal and Slot Mechanism

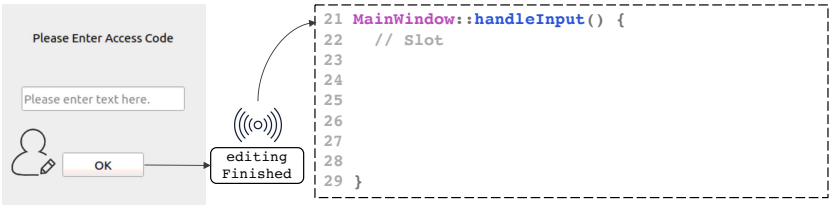
```
1  MainWindow::MainWindow() {
2     ...
3     // Create lineEdit instance
4     v0 = operator.new(0x30)
5     QLineEdit(v0)
6     *(this + 0x30) = v0
7     ...
8     // Register callbacks
9     connect(*(this+0x30),"2textChanged(QString)"
10            , this, "1updateText(QString)", 0)
11
12    connect(*(this+0x30),"2editingFinished()"
13            , this, "1handleInput()", 0)
14    ...
15 }
```


```
16 MainWindow::updateText(QString v1) {
17    // Slot
18    if (v1 != null)
19        *(this + 0x48) = v1 // this->text
20 }
```

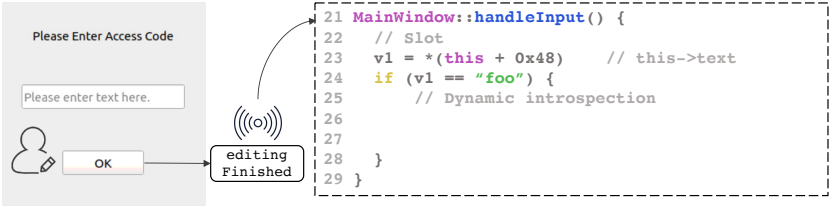
Qt's Signal and Slot Mechanism



Qt's Dynamic Introspection



Qt's Dynamic Introspection



window.text = "foo"

Qt's Dynamic Introspection

editing Finished

```
21 MainWindow::handleInput() {
22   // Slot
23   v1 = *(this + 0x48)     // this->text
24   if (v1 == "foo") {
25       // Dynamic introspection
26       this->setProperty("text", "bar")
27       qDebug() << v1  // Will print out "bar"
28   }
29 }
```

window.text = "foo"

⇩

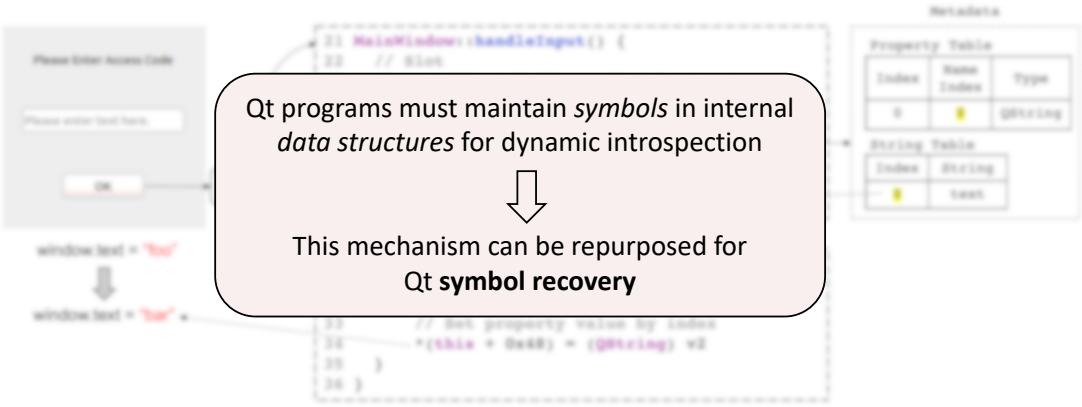
window.text = "bar"

Qt's Dynamic Introspection

Qt's Dynamic Introspection

Qt's Dynamic Introspection

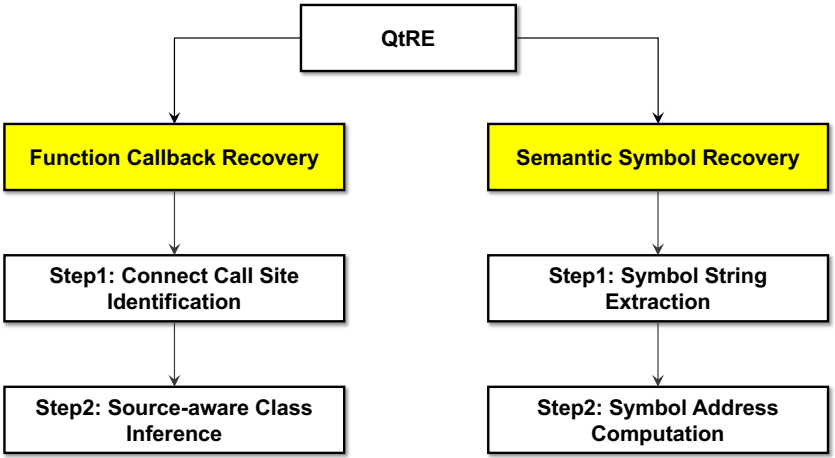
Qt's Dynamic Introspection



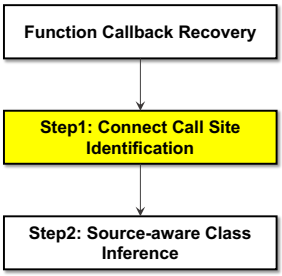
Qt programs must maintain *symbols* in internal *data structures* for dynamic introspection

This mechanism can be repurposed for Qt **symbol recovery**

Function Callback Recovery



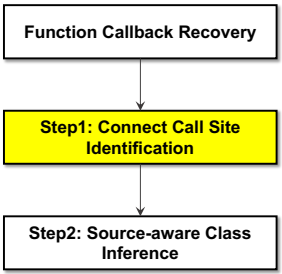
Function Callback Recovery



Type	Name	Param.0	Param.1	Param.2	Param.3	Param.4
		Signal Class	Signal Sig.	Slot Class	Slot Sig.	Type
1	connect	QObject*	fptr*	QObject*	fptr*	int
2	connect	QObject*	char*	QObject*	char*	int

Table: Connect functions and argument types.

Function Callback Recovery

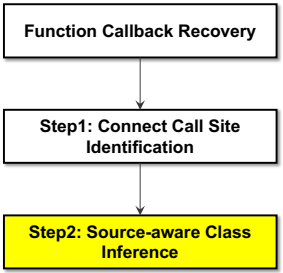


Type	Name	Param.0	Param.1	Param.2	Param.3	Param.4
		Signal Class	Signal Sig.	Slot Class	Slot Sig.	Type
1	connect	QObject*	fptr*	QObject*	fptr*	int
2	connect	QObject*	char*	QObject*	char*	int

Table: Connect functions and argument types.

Both signal/slot classes and function signatures need to be resolved due to polymorphism in C++
e.g., A.foo() vs. B.foo()

Function Callback Recovery



$$\textsc{ThisPointer}\ \frac{p = this}{this \mapsto class}$$

$$\textsc{FuncParam}\ \frac{p \mapsto v \quad v \in Parameter(f)}{Type(v)}$$

$$\textsc{FuncRetVal}\ \frac{p \mapsto v \quad v = f(...)}{ReturnType(f)}$$

$$\textsc{GlobalVar}\ \frac{p \mapsto v \quad v \in GlobalVariables}{Type(v)}$$

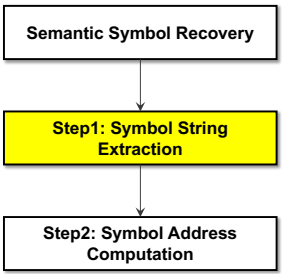
$$\textsc{HeapVar}\ \frac{p \mapsto HeapAlloc(v,size)}{Constructor(v)}$$

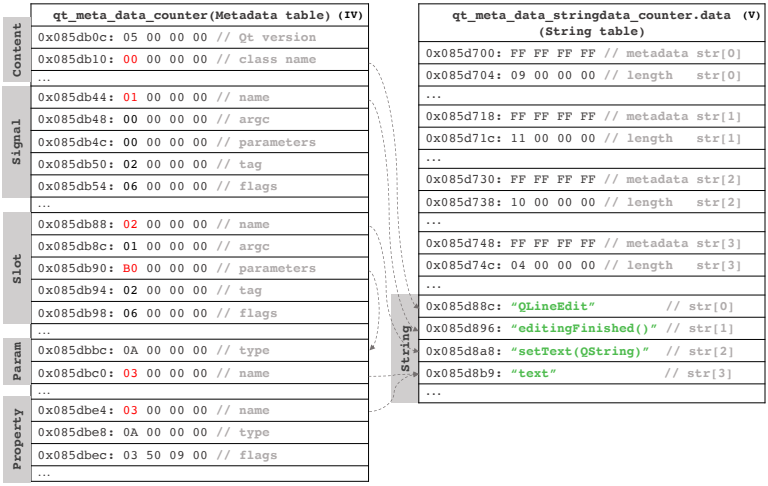
$$\textsc{StackVar}\ \frac{p \mapsto v \quad v \in StackVariables}{Type(v)}$$

$$\textsc{SigMatching}\ \frac{\exists! f,\ Signature(f) = signature}{Class(f)}$$

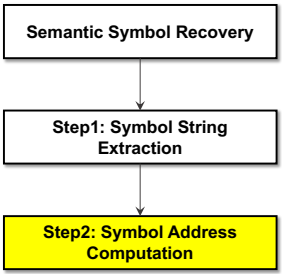
Figure: Class inference rules

Semantic Symbol Recovery





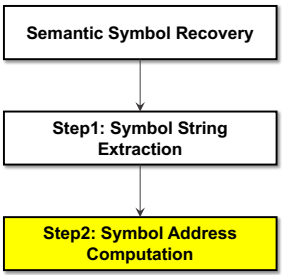
Semantic Symbol Recovery



Solution

- Leverage the code transition and computation logic in `qt_metacall`
- `qt_metacall` is a class-specific member function generated by Qt MOC compiler

Semantic Symbol Recovery



Solution

- Leverage the code transition and computation logic in `qt_metacall`
- `qt_metacall` is a class-specific member function generated by Qt MOC compiler
- Unit-level symbolic execution

Evaluation Setup

Experiment Environment

- We implemented QTRE atop GHIDRA [ghi]

Evaluation Setup

Experiment Environment

- We implemented QtRE atop Ghidra [ghi]
- Two suites of Qt binary of evaluation
  - 80 binaries from KDE Plasma desktop image (open-source)
  - 43 binaries from the infotainment software of a Tesla Model S (closed-source)

Evaluation Setup

Experiment Environment

- We implemented QtRE atop Ghidra [ghi]
- Two suites of Qt binary of evaluation
  - 80 binaries from KDE Plasma desktop image (open-source)
  - 43 binaries from the infotainment software of a Tesla Model S (closed-source)
- Each binary has at least one callback/symbol instance available

Evaluation Results

Quantifying FP and FN with KDE Programs (RQ1)

- 16 KDE binaries selected for FP / FN validation

Binary Name	Source Repo.	# Ins.(K)	Time(s)	Callback Recovery				Symbol Recovery							
				# Total	# Recover	%	# Valid	# Prop.	# Signal	# Slot	# Param.	# Total	# Recover	%	# Valid
libKF5KHtml	khtml	104,234	965	192	74	38.5%	74	9	14	96	68	187	185	98.9%	185
libKF5GlobalAccel	kglobalaccel	63,560	244	4	1	25%	1	8	1	3	10	22	22	100%	22
libKF5KIOCore	kio	119,425	96	146	137	93.8%	137	6	42	9	46	103	103	100%	103
libKF5ItemModels	kitemmodels	14,906	26	51	42	82.4%	42	13	19	29	61	122	122	100%	122
libKF5IconThemes	kiconthemes	185,804	43	21	8	38.1%	8	3	4	13	9	29	29	100%	29
libKF5CompactDisc	libkcompactdisc	112,781	4	3	1	33.3%	1	0	9	17	17	43	43	100%	43
						.....									
Total	N/A	2,093,399	1,568	796	598	75.1%	598	96	246	325	497	1164	1161	99.7%	1161

Table: Partial results of validated function callbacks and symbols from KDE ground truth.

Evaluation Results

Quantifying FP and FN with KDE Programs (RQ1)

- 16 KDE binaries selected for FP / FN validation
- 796 functions callbacks and 1164 symbols recovered from these binaries

Binary Name	Source Repo.	# Ins.(K)	Time(s)	Callback Recovery				Symbol Recovery							
				# Total	# Recover	%	# Valid	# Prop.	# Signal	# Slot	# Param.	# Total	# Recover	%	# Valid
libKF5KHtml	khtml	104,234	965	192	74	38.5%	74	9	14	96	68	187	185	98.9%	185
libKF5GlobalAccel	kglobalaccel	63,560	244	4	1	25%	1	8	1	3	10	22	22	100%	22
libKF5KIOCore	kio	119,425	96	146	137	93.8%	137	6	42	9	46	103	103	100%	103
libKF5ItemModels	kitemmodels	14,906	26	51	42	82.4%	42	13	19	29	61	122	122	100%	122
libKF5IconThemes	kiconthemes	185,804	43	21	8	38.1%	8	3	4	13	9	29	29	100%	29
libKF5CompactDisc	libkcompactdisc	112,781	4	3	1	33.3%	1	0	9	17	17	43	43	100%	43
						.....									
Total	N/A	2,093,399	1,568	796	598	75.1%	598	96	246	325	497	1164	1161	99.7%	1161

Table: Partial results of validated function callbacks and symbols from KDE ground truth.

Evaluation Results

Quantifying FP and FN with KDE Programs (RQ1)

- 16 KDE binaries selected for FP / FN validation
- 796 functions callbacks and 1164 symbols recovered from these binaries
- No FP reported. FN exists due to memory aliasing that failed the class inference and symbol address computation.

Binary Name	Source Repo.	# Ins.(K)	Time(s)	Callback Recovery				Symbol Recovery							
				# Total	# Recover	%	# Valid	# Prop.	# Signal	# Slot	# Param.	# Total	# Recover	%	# Valid
libKF5KHtml	khtml	104,234	965	192	74	38.5%	74	9	14	96	68	187	185	98.9%	185
libKF5GlobalAccel	kglobalaccel	63,560	244	4	1	25%	1	8	1	3	10	22	22	100%	22
libKF5KIOCore	kio	119,425	96	146	137	93.8%	137	6	42	9	46	103	103	100%	103
libKF5ItemModels	kitemmodels	14,906	26	51	42	82.4%	42	13	19	29	61	122	122	100%	122
libKF5IconThemes	kiconthemes	185,804	43	21	8	38.1%	8	3	4	13	9	29	29	100%	29
libKF5CompactDisc	libkcompactdisc	112,781	4	3	1	33.3%	1	0	9	17	17	43	43	100%	43
						.....									
Total	N/A	2,093,399	1,568	796	598	75.1%	598	96	246	325	497	1164	1161	99.7%	1161

Table: Partial results of validated function callbacks and symbols from KDE ground truth.

Evaluation Results

Real-world Qt Binaries (RQ2)

- 80/43 binaries extracted from KDE/Tesla

Result	KDE		Tesla	
	#	%	#	%
# Total Binary	80	100%	43	100%
Callback Recovery				
├─# Total callback	3,972	100%	8,845	100%
└─# Recovered callback	3,323	83.7%	7,544	85.3%
├─# Type-1 connect	2,992	75.3%	0	0
└─# Type-2 connect	331	8.3%	7,544	85.3%
Source of Class Objects				
├─ # This pointer	280	4.2%	7,433	49.3%
├─ # Function parameters	113	1.7%	295	2.0%
├─ # Function return value	88	1.3%	1,371	9.1%
├─ # Global variable	83	1.2%	3,509	23.3%
├─ # Stack variable	5,984	90.0%	389	2.6%
├─ # Heap variable	88	1.3%	652	4.3%
└─ # Signature matching	15	0.2%	1,439	9.5%
Symbol Recovery				
└─# Total recovered symbols	4,362	100%	20,611	100%
├─# Property	817	18.7%	951	4.6%
├─# Signal	1,182	27.1%	9,266	45.0%
├─# Slot	841	19.3%	3,326	16.1%
└─# Function parameter	1,522	34.9%	7,068	34.3%

Table: Callback and symbol recovery results.

Evaluation Results

Real-world Qt Binaries (RQ2)

- 80/43 binaries extracted from KDE/Tesla
- 10,867 callbacks and 24,973 symbols recovered from these 123 binaries

Result	KDE		Tesla	
	#	%	#	%
# Total Binary	80	100%	43	100%
Callback Recovery				
├─# Total callback	3,972	100%	8,845	100%
└─# Recovered callback	3,323	83.7%	7,544	85.3%
├─# Type-1 connect	2,992	75.3%	0	0
└─# Type-2 connect	331	8.3%	7,544	85.3%
Source of Class Objects				
├─ # This pointer	280	4.2%	7,433	49.3%
├─ # Function parameters	113	1.7%	295	2.0%
├─ # Function return value	88	1.3%	1,371	9.1%
├─ # Global variable	83	1.2%	3,509	23.3%
├─ # Stack variable	5,984	90.0%	389	2.6%
├─ # Heap variable	88	1.3%	652	4.3%
└─ # Signature matching	15	0.2%	1,439	9.5%
Symbol Recovery				
└─# Total recovered symbols	4,362	100%	20,611	100%
├─# Property	817	18.7%	951	4.6%
├─# Signal	1,182	27.1%	9,266	45.0%
├─# Slot	841	19.3%	3,326	16.1%
└─# Function parameter	1,522	34.9%	7,068	34.3%

Table: Callback and symbol recovery results.

Evaluation Results

Real-world Qt Binaries (RQ2)

- 80/43 binaries extracted from KDE/Tesla
- 10,867 callbacks and 24,973 symbols recovered from these 123 binaries
- Different sources contribute to the class inference algorithm

Result	KDE		Tesla	
	#	%	#	%
# Total Binary	80	100%	43	100%
Callback Recovery				
├# Total callback	3,972	100%	8,845	100%
└# Recovered callback	3,323	83.7%	7,544	85.3%
├# Type-1 connect	2,992	75.3%	0	0
└# Type-2 connect	331	8.3%	7,544	85.3%
Source of Class Objects				
├ # This pointer	280	4.2%	7,433	49.3%
├ # Function parameters	113	1.7%	295	2.0%
├ # Function return value	88	1.3%	1,371	9.1%
├ # Global variable	83	1.2%	3,509	23.3%
├ # Stack variable	5,984	90.0%	389	2.6%
├ # Heap variable	88	1.3%	652	4.3%
└ # Signature matching	15	0.2%	1,439	9.5%
Symbol Recovery				
└# Total recovered symbols	4,362	100%	20,611	100%
├# Property	817	18.7%	951	4.6%
├# Signal	1,182	27.1%	9,266	45.0%
├# Slot	841	19.3%	3,326	16.1%
└# Function parameter	1,522	34.9%	7,068	34.3%

Table: Callback and symbol recovery results.

Evaluation Results

Efficiency (RQ3)

- On average, it takes QtRE 1.7 minutes to analyze a binary

Comparison with Other RE Tools (RQ4)

Evaluation Results

Efficiency (RQ3)

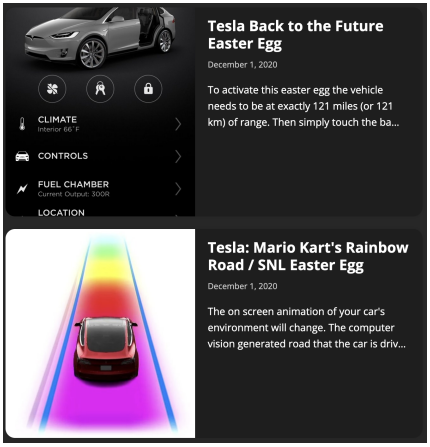
- On average, it takes QtRE 1.7 minutes to analyze a binary

Comparison with Other RE Tools (RQ4)

- Compare QtRE with three SOTA RE tools IDA Pro, Ghidra, and Angr
- They cannot identify any callbacks/symbols recovered by QtRE

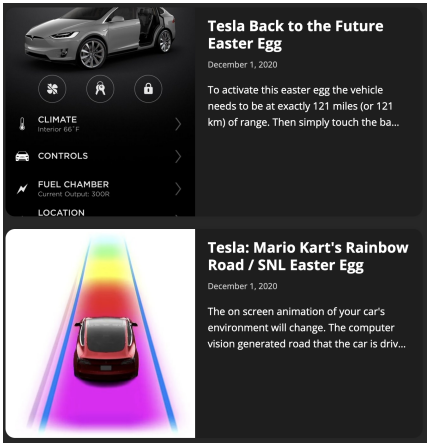
Result	KDE	Tesla	Total
# Total call graph edges			
├─# Recovered by Angr [SWS+16]	791,907	1,395,093	2,187,000
│ └─# Callback recovered	0	0	
├─# Recovered by Ghidra [ghi]	432,843	987,263	1,420,106
│ └─# Callback recovered	0	0	
└─# Recovered by Ghidra [ghi] w/ QtRE	436,166	994,807	1,430,973
└─# Callback from QtRE	3,323	7,544	10,867
# Total symbols			
├─# Recovered by Angr [SWS+16]	97,109	1,171,990	1,269,099
├─# Recovered by Ghidra [ghi]	97,109	1,171,990	1,269,099
└─# Recovered by Ghidra [ghi] w/ QtRE	101,471	1,192,601	1,294,072
└─# Symbol from QtRE	4,362	20,611	24,973

Egg Hunt in Tesla's Infotainment System



Easter eggs in Tesla vehicles [eas]

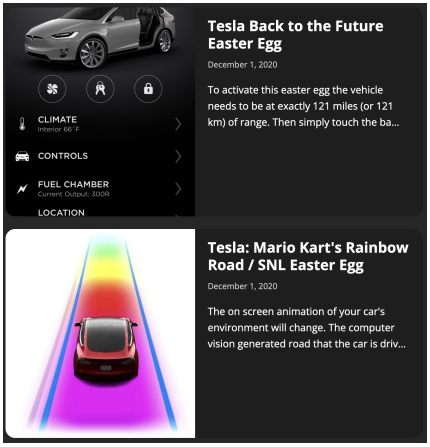
Egg Hunt in Tesla's Infotainment System



Easter eggs in Tesla vehicles [eas]

- Do they raise security concerns?
- How to systematically identify them?

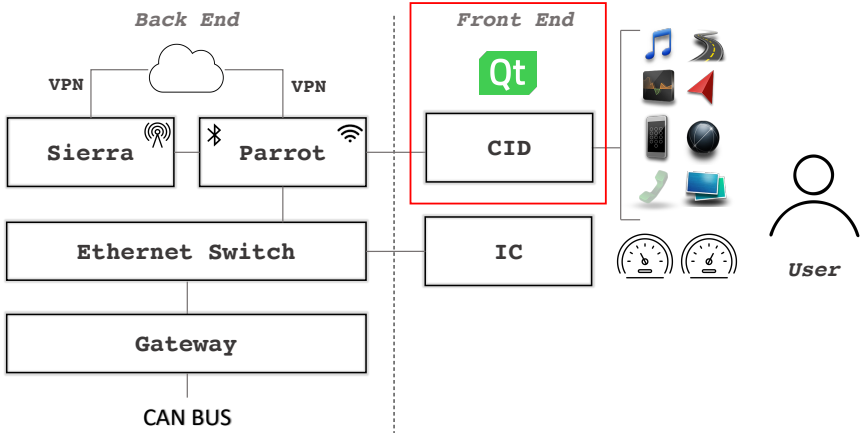
Egg Hunt in Tesla's Infotainment System



Easter eggs in Tesla vehicles [eas]

- Do they raise security concerns?
- How to systematically identify them?
  - Coverage-based fuzzing (emulation required)
  - Input validation analysis on Qt binaries

Egg Hunt in Tesla's Infotainment System



Experiment Setup and Findings

Experiment Setup

- The user input variables are selected based on the symbols from QTRE

Class Name	Var./Func. Name	Symbolic Address
QLineEdit	text()	N/A
QLineEdit	text	*(*(*(λ+4)+300)+8)
QAbstractSpinBox	text	*(*(λ+4)+452)
QDoubleSpinBox	text	*(*(λ+4)+452)
QSpinBox	text	*(*(λ+4)+452)
QDateTimeEdit	text	*(*(λ+4)+452)
TextField	text	*(*(λ+796))
PasswordTextField	text	*(*(λ+796))
WebEntryField	text	*(*(λ+796))
NavigationSearchBox	text	*(*(λ+796))
CompleterTextField	text	*(*(λ+796))
ExtEntryField	text	*(*(λ+796))

Table: Taint analysis sources (λ: `this` pointer).

Experiment Setup and Findings

Experiment Setup

- The user input variables are selected based on the symbols from QTRE
- Identify Easter eggs from the constant values compared against user inputs

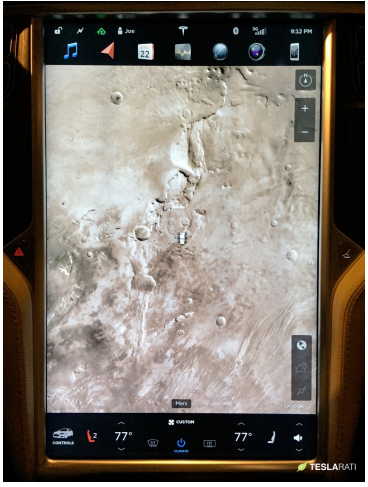
Class Name	Var./Func. Name	Symbolic Address
QLineEdit	text()	N/A
QLineEdit	text	*(*(*(λ+4)+300)+8)
QAbstractSpinBox	text	*(*(λ+4)+452)
QDoubleSpinBox	text	*(*(λ+4)+452)
QSpinBox	text	*(*(λ+4)+452)
QDateTimeEdit	text	*(*(λ+4)+452)
TextField	text	*(*(λ+796))
PasswordTextField	text	*(*(λ+796))
WebEntryField	text	*(*(λ+796))
NavigationSearchBox	text	*(*(λ+796))
CompleterTextField	text	*(*(λ+796))
ExtEntryField	text	*(*(λ+796))

Table: Taint analysis sources (λ: `this` pointer).

Experiment Setup and Findings

Category	Content	Description
Easter Egg	"007"	Submarine Easter egg
	"modelxmas"	Show holiday lights
	"42"	Change car name
	"mars"	Turn map into Mars surface
	"transport"	Transport mode
	"performance"	Performance mode
	"showroom"	Showroom mode
Access Token	SecurityToken1	Enable diagnostic mode
	SecurityToken2	Enable diagnostic mode
	crc(token)==0x18e5a977	Enable developer mode
	crc(token)==0x73bbee22	Enable developer mode
Master Pwd	"3500"	Exit valet mode

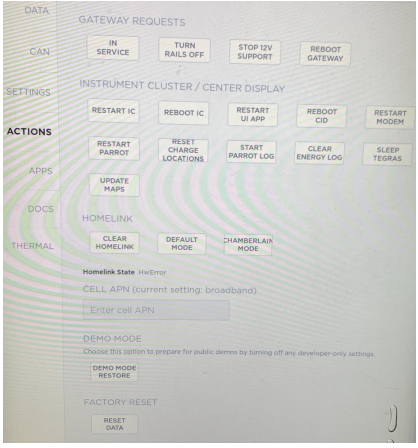
Table: Hidden commands from Tesla firmware.



Experiment Setup and Findings

Category	Content	Description
Easter Egg	"007"	Submarine Easter egg
	"modelxmas"	Show holiday lights
	"42"	Change car name
	"mars"	Turn map into Mars surface
	"transport"	Transport mode
	"performance"	Performance mode
	"showroom"	Showroom mode
Access Token	SecurityToken1	Enable diagnostic mode
	SecurityToken2	Enable diagnostic mode
	crc(token)==0x18e5a977	Enable developer mode
	crc(token)==0x73bbee22	Enable developer mode
Master Pwd	"3500"	Exit valet mode

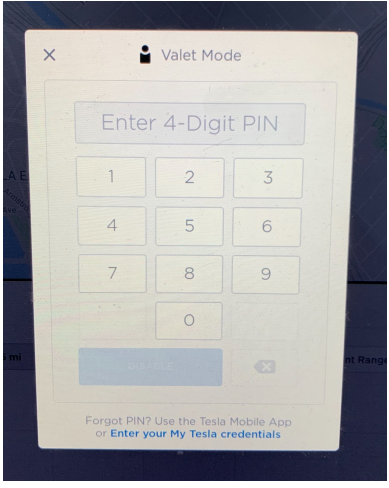
Table: Hidden commands from Tesla firmware.



Experiment Setup and Findings

Category	Content	Description
Easter Egg	"007"	Submarine Easter egg
	"modelxmas"	Show holiday lights
	"42"	Change car name
	"mars"	Turn map into Mars surface
	"transport"	Transport mode
	"performance"	Performance mode
	"showroom"	Showroom mode
Access Token	SecurityToken1	Enable diagnostic mode
	SecurityToken2	Enable diagnostic mode
	crc(token)==0x18e5a977	Enable developer mode
	crc(token)==0x73bbee22	Enable developer mode
Master Pwd	"3500"	Exit valet mode

Table: Hidden commands from Tesla firmware.



Experiment Setup and Findings

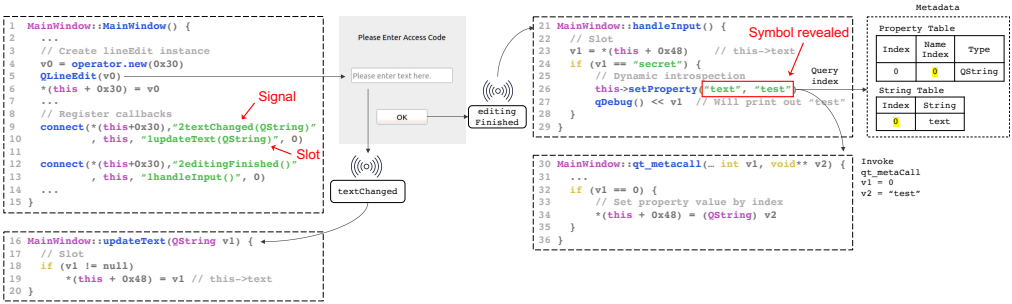
Category	Content	Description
Easter Egg	"007"	Submarine Easter egg
	"modelxmas"	Show holiday lights
	"42"	Change car name
	"mars"	Turn map into Mars surface
	"transport"	Transport mode
	"performance"	Performance mode
	"showroom"	Showroom mode
Access Token	SecurityToken1	Enable diagnostic mode
	SecurityToken2	Enable diagnostic mode
	crc(token)==0x18e5a977	Enable developer mode
	crc(token)==0x73bbee22	Enable developer mode
Master Pwd	"3500"	Exit valet mode

Table: Hidden commands from Tesla firmware.

Disclosure

The Tesla security team acknowledged our findings in 2022/4 and have eliminated the feasible paths for exploiting these hidden commands in the latest firmware.

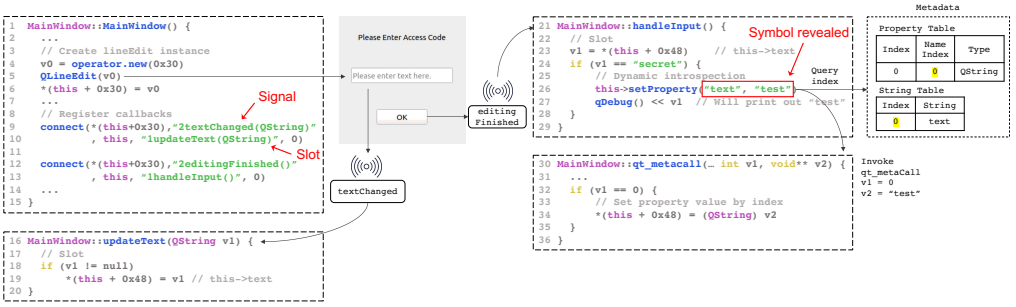
Takeaway



Takeaway & Future Work

- QtRE: A static analysis tool that leverages Qt's unique insights for function callback and symbol recovery

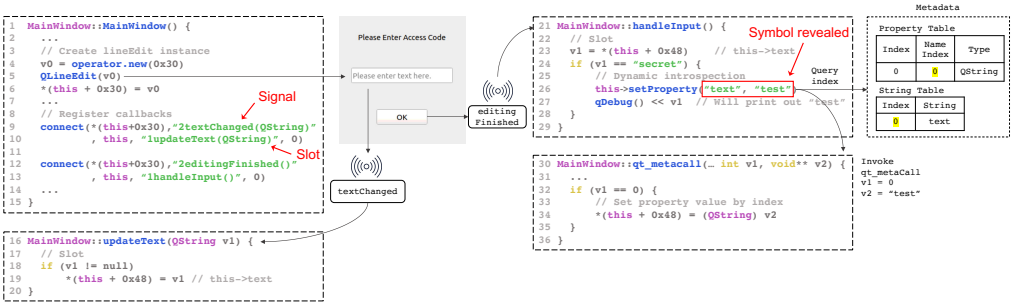
Takeaway



Takeaway & Future Work

- QtRE: A static analysis tool that leverages Qt's unique insights for function callback and symbol recovery
- Use QtRE to analyze security-critical Qt applications (automobiles, embedded systems, medical devices)

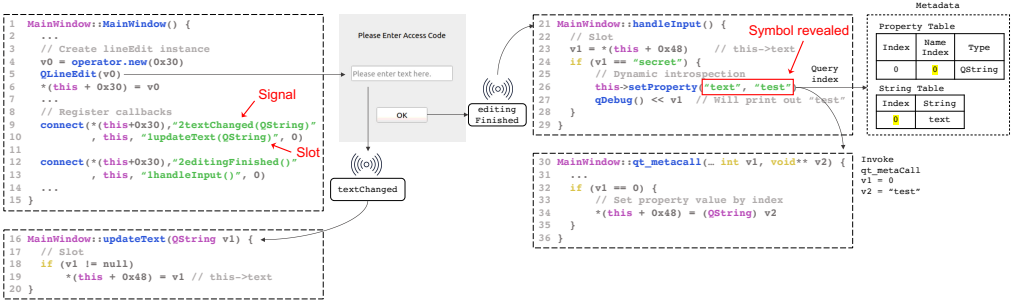
Takeaway



Takeaway & Future Work

- QtRE: A static analysis tool that leverages Qt's unique insights for function callback and symbol recovery
- Use QtRE to analyze security-critical Qt applications (automobiles, embedded systems, medical devices)
- Integrate QtRE with state-of-the-art fuzzers for GUI-fuzzing (e.g., Easter egg hunting)

Takeaway



Takeaway & Future Work

- QtRE: A static analysis tool that leverages Qt's unique insights for function callback and symbol recovery
- Use QtRE to analyze security-critical Qt applications (automobiles, embedded systems, medical devices)
- Integrate QtRE with state-of-the-art fuzzers for GUI-fuzzing (e.g., Easter egg hunting)

The source code is available at https://github.com/OSUSecLab/QtRE.

Thank You

Egg Hunt in Tesla Infotainment: A First Look at Reverse Engineering of Qt Binaries

Haohuang Wen and Zhiqiang Lin

Department of Computer Science and Engineering
The Ohio State University

Aug 10^{th}, 2023

References I

A complete list of all tesla easter eggs and hidden features, https://www.notateslaapp.com/tesla-easter-eggs/.

Ghidra, https://ghidra-sre.org/.

The IDA Pro disassembler and debugger, http://www.hex-rays.com/idapro/.

Andre Pawlowski, Moritz Contag, Victor van der Veen, Chris Ouwehand, Thorsten Holz, Herbert Bos, Elias Athanasopoulos, and Cristiano Giuffrida, *Marx: Uncovering class hierarchies in c++ programs.*, NDSS, 2017.

Edward J Schwartz, Cory F Cohen, Michael Duggan, Jeffrey Gennari, Jeffrey S Havrilla, and Charles Hines, *Using logic programming to recover c++ classes and methods from compiled executables*, Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 426–441.

Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Andrew Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, et al., *Sok:(state of) the art of war: Offensive techniques in binary analysis*, 2016 IEEE Symposium on Security and Privacy (SP), IEEE, 2016, pp. 138–157.

Hieu Tran, Ngoc Tran, Son Nguyen, Hoan Nguyen, and Tien N Nguyen, *Recovering variable names for minified code with usage contexts*, 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, 2019, pp. 1165–1175.

Victor Van Der Veen, Enes Göktas, Moritz Contag, Andre Pawoloski, Xi Chen, Sanjay Rawat, Herbert Bos, Thorsten Holz, Elias Athanasopoulos, and Cristiano Giuffrida, *A tough call: Mitigating advanced code-reuse attacks at the binary level*, 2016 IEEE Symposium on Security and Privacy (SP), IEEE, 2016, pp. 934–953.