GUIFuzz++:

Unleashing Grey-box Fuzzing on Desktop Graphical User Interfacing Applications

Dillon Otto
University of Utah

Tanner Rowlett
University of Utah

Stefan Nagy
University of Utah

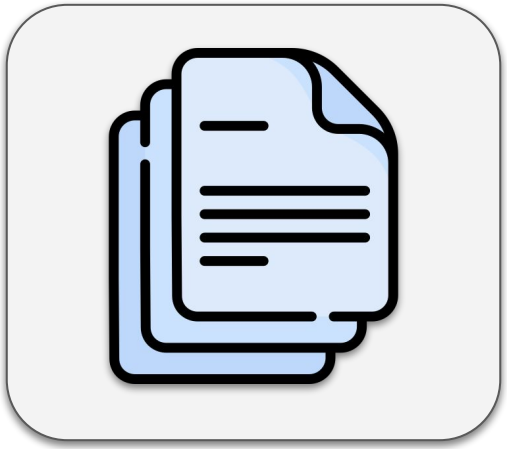
FUTURES^3 LAB
FUTURE TECHNOLGY FOR USABLE, RELIABLE, & EFFICIENT SECURITY OF SOFTWARE & SYSTEMS

Motivation: Fuzzing Software Interfaces

- **Interfaces:** vectors by which fuzzer-generated inputs **influence program logic**

Motivation: Fuzzing Software Interfaces

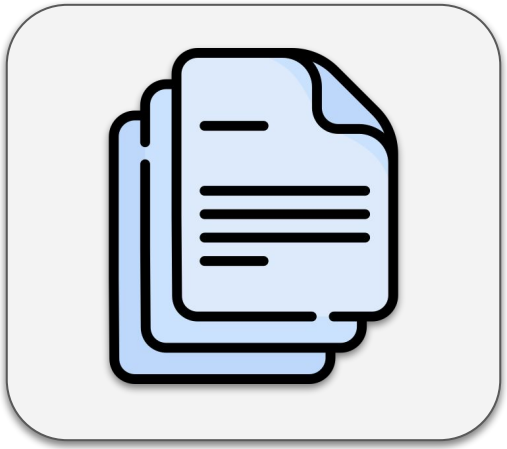
- **Interfaces:** vectors by which fuzzer-generated inputs **influence program logic**



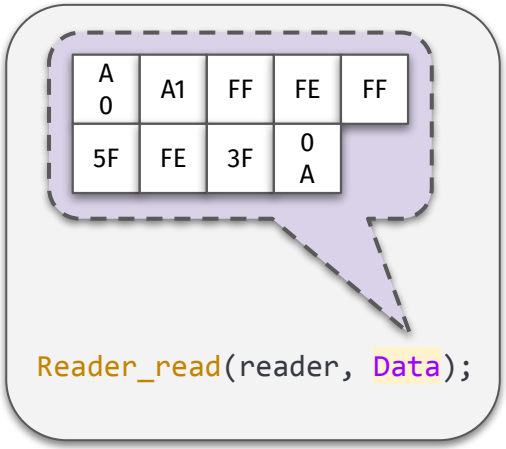
File-based Interfaces
(e.g., AFL++)

Motivation: Fuzzing Software Interfaces

- **Interfaces:** vectors by which fuzzer-generated inputs **influence program logic**



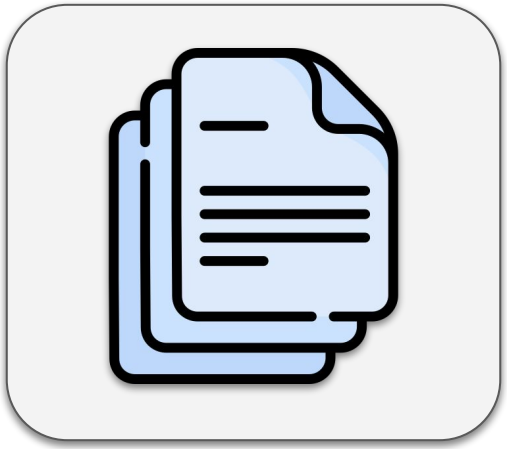
File-based Interfaces
(e.g., AFL++)



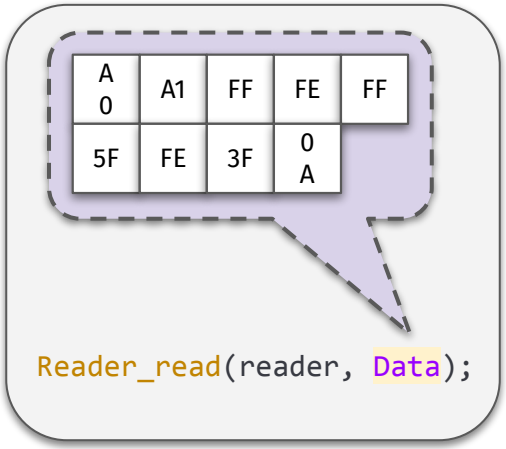
Memory-based Interfaces
(e.g., libFuzzer, Nyx)

Motivation: Fuzzing Software Interfaces

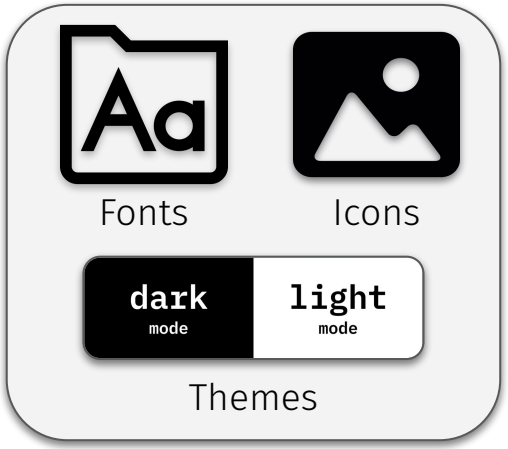
- **Interfaces:** vectors by which fuzzer-generated inputs **influence program logic**



File-based Interfaces
(e.g., AFL++)



Memory-based Interfaces
(e.g., libFuzzer, Nyx)



Environment-level Interfaces
(e.g., EnvFuzz)

Motivation: Fuzzing Software Interfaces

- **Interfaces:** vectors by which fuzzer-generated inputs **influence program logic**

To reproduce this bug, I make a short PoC script that repeating several actions using WinAPI.

- Click "Show side panel"
- Click “Guest”
- Click “Chromium”
- Click “Hide side panel”
- Click the other window.
- Click the current window. (fuzzing target)

issues.chromium.org/issues/40060211

The current work assessed U.S. medical device recalls during 2012-15, with the goal of understanding the impact and nature of user interface (UI) software errors in medical devices. Based on information from the Food and Drug Administration's public and internal recall databases, 423 (~140/year) medical device recalls were identified as resulting from UI software errors, which accounted for nearly one-half of recalls caused by software errors during the same period. A total of 499 UI software errors were identified as the root causes of medical device recalls, and a

pubmed.ncbi.nlm.nih.gov/31162965/

The FAA received reports earlier this year of three incidents of display electronic unit (DEU) software errors on Model 737 NG airplanes flying into runway PABR in Barrow, Alaska. All six display units (DUs) blanked with a selected instrument approach to a runway with a 270-degree true heading, and all six DUs stayed blank until a different runway was selected. The Integrated Standby Flight Display (ISFD) and Heads-Up-Display (HUD) remained operational during this failure of the primary flight displays. The

govinfo.gov/content/pkg/FR-2019-12-27/pdf/2019-27966.pdf

- **Problem:** applications’ **graphical user interfaces** often **expose critical bugs**

Challenges: Why GUIs are Difficult Targets

- Existing fuzzers **broadly fail on GUIs** due to **three fundamental hurdles:**

Challenges: Why GUIs are Difficult Targets

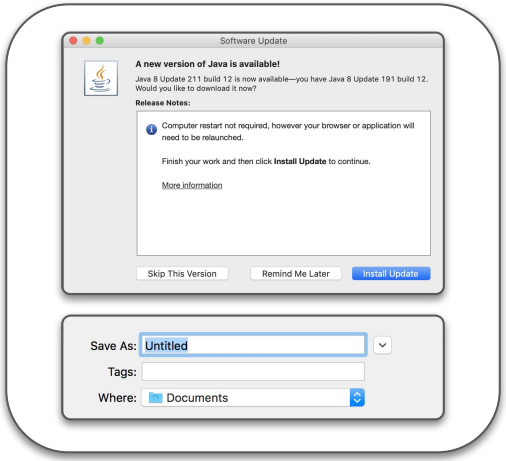
- Existing fuzzers **broadly fail on GUIs** due to **three fundamental hurdles:**

1. Facilitating GUI Interaction
(exercising GUI behaviors)

Challenges: Why GUIs are Difficult Targets

- Existing fuzzers **broadly fail on GUIs** due to **three fundamental hurdles:**

1. Facilitating GUI Interaction
(exercising GUI behaviors)

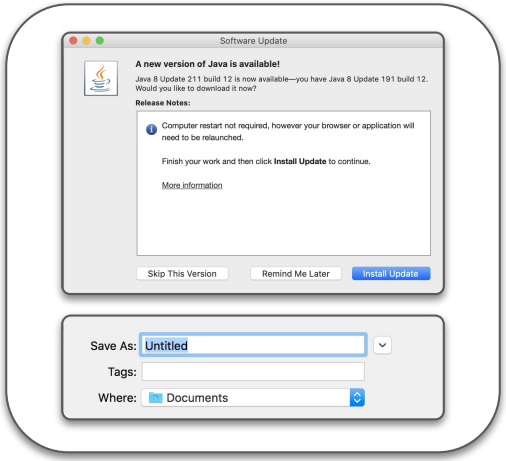


2. Thwarting GUI Obstacles
(first-/third-party pop-ups)

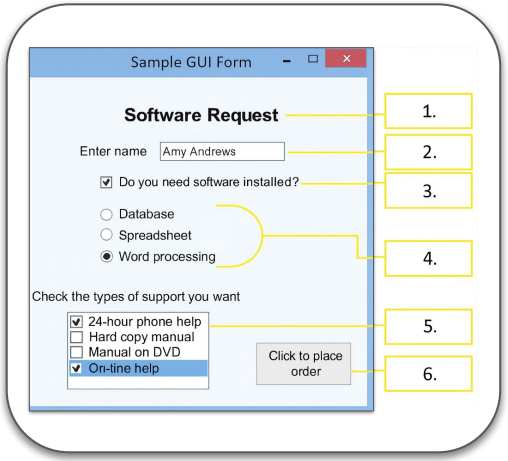
Challenges: Why GUIs are Difficult Targets

- Existing fuzzers **broadly fail on GUIs** due to **three fundamental hurdles:**

1. Facilitating GUI Interaction
(exercising GUI behaviors)



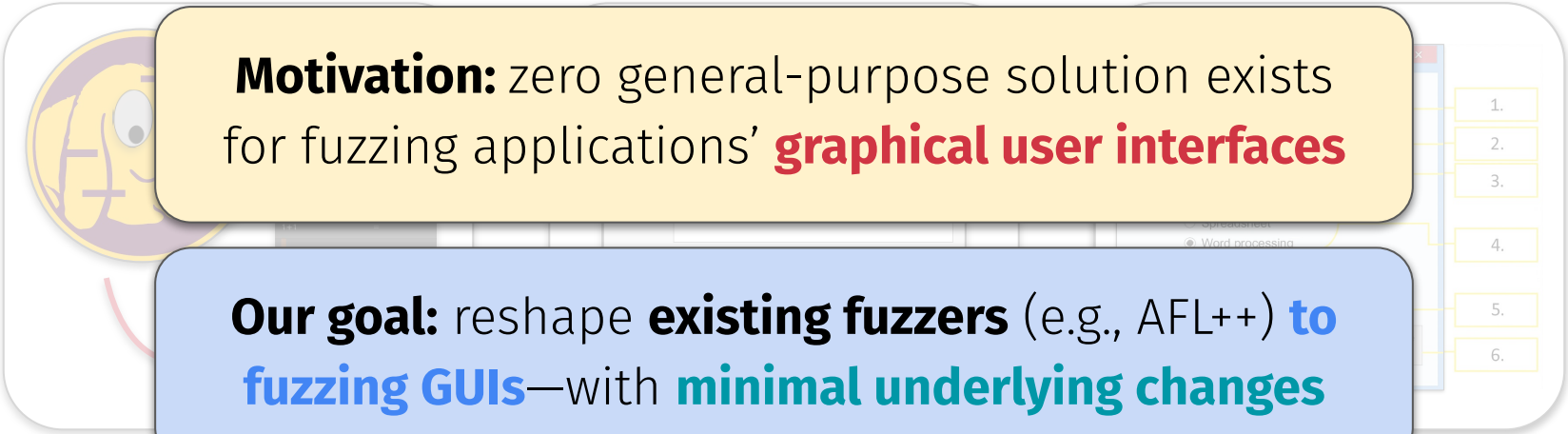
2. Thwarting GUI Obstacles
(first-/third-party pop-ups)



3. Attaining Precise Interaction
(ensuring steady progress)

Challenges: Why GUIs are Difficult Targets

- Existing fuzzers **broadly fail on GUIs** due to **three fundamental hurdles:**



Motivation: zero general-purpose solution exists for fuzzing applications' **graphical user interfaces**

Our goal: reshape **existing fuzzers** (e.g., AFL++) **to fuzzing GUIs**—with **minimal underlying changes**

1. Facilitating GUI Interaction
(exercising GUI behaviors)

2. Thwarting GUI Obstacles
(first-/third-party pop-ups)

3. Attaining Precise Interaction
(ensuring steady progress)

Solution: Making GUI-agnostic Fuzzers GUI-aware

- **Idea:** translate fuzzers' random bytes into **distinct interactions**
  - **The GUI Interaction Interpreter**
- **Three-byte** interaction structure:
  - **Operand 1:** instruction opcode
    - Close, keypress, click, and dragging
    - Higher opcodes **modulo'd to range**

Op Structure	Description of GUI Interaction
`00 FF FF`	**Close** currently-active window, ignoring the last two operands.
`01 CC FF`	**Input** the key press corresponding to the extended `ASCII` encoding of primary operand `CC`, ignoring the second operand. ***Ex:*** `01 7F FF` → input extended `ASCII` key press "`DEL`".
`02 XX YY`	**Click** the location (`X%`, `Y%`) relative to the current window's dimensions, offset from its bottom-left coordinate (`0,0`). ***Ex:*** `02 A0 1B` → click relative position (`62.5%`, `10.5%`).
`03 XX YY`	**Drag** the cursor from its *current* position to the *new* position (`X%`, `Y%`) relative to the current window's dimensions, offset from its bottom-left coordinate (`0,0`). ***Ex:*** `03 00 80` → drag to relative position (`0%`, `50%`).
`NN AA BB`	**All higher opcodes** (i.e., `04`–`FF`): normalize the opcode via `(NN % 4)`, reinterpreting the transformed opcode accordingly. ***Ex:*** `B2 2C 9F` → reinterpret as *click* operation `02 2C 9F`.

Solution: Making GUI-agnostic Fuzzers GUI-aware

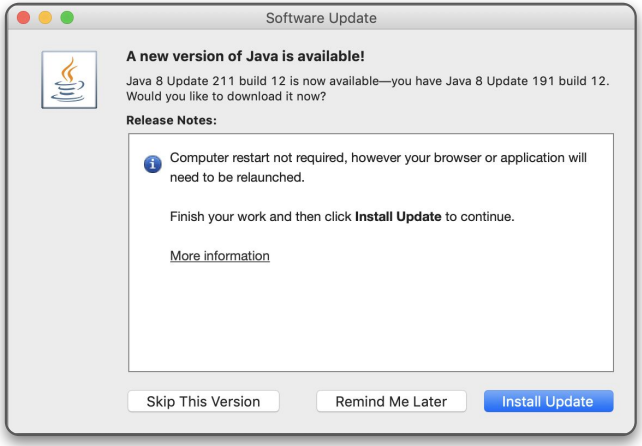
- **Idea:** translate fuzzers' random bytes into **distinct interactions**
  - **The GUI Interaction Interpreter**
- **Three-byte** interaction structure:
  - **Operand 1:** instruction opcode
    - Close, keypress, click, and dragging
    - Higher opcodes **modulo'd to range**
  - **Operands 2–3:** instruction semantics
    - **Clicks/drags:** relative coordinates
    - **Keypresses:** input ASCII character

Op Structure	Description of GUI Interaction
`00 FF FF`	**Close** currently-active window, ignoring the last two operands.
`01 CC FF`	**Input** the key press corresponding to the extended `ASCII` encoding of primary operand `CC`, ignoring the second operand. ***Ex:*** `01 7F FF` → input extended `ASCII` key press "`DEL`".
`02 XX YY`	**Click** the location (`X%`, `Y%`) relative to the current window's dimensions, offset from its bottom-left coordinate (`0,0`). ***Ex:*** `02 A0 1B` → click relative position (`62.5%`, `10.5%`).
`03 XX YY`	**Drag** the cursor from its *current* position to the *new* position (`X%`, `Y%`) relative to the current window's dimensions, offset from its bottom-left coordinate (`0,0`). ***Ex:*** `03 00 80` → drag to relative position (`0%`, `50%`).
`NN AA BB`	**All higher opcodes** (i.e., `04–FF`): normalize the opcode via `(NN % 4)`, reinterpreting the transformed opcode accordingly. ***Ex:*** `B2 2C 9F` → reinterpret as *click* operation `02 2C 9F`.

- **Outcome:** repurpose existing fuzzers to GUIs **with no changes to their core**

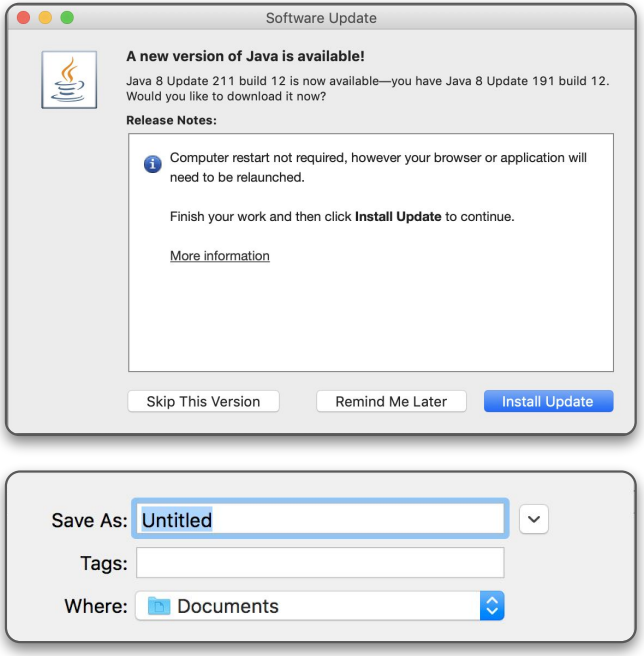
Solution: Tackling Window Interference

- **Idea:** lightweight **window tracking** to catch/close unwanted windows
  - **Third-party windows** (e.g., Java popup)
    - Hook window system and close any not matching **target's PID**



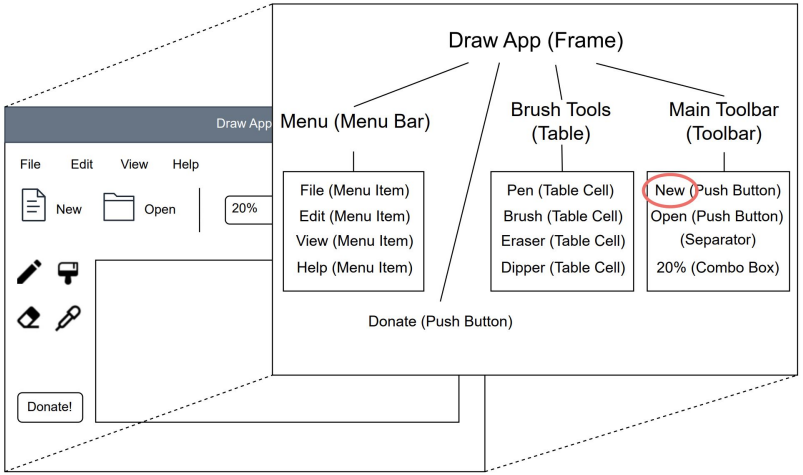
Solution: Tackling Window Interference

- **Idea:** lightweight **window tracking** to catch/close unwanted windows
  - **Third-party windows** (e.g., Java popup)
    - Hook window system and close any not matching **target's PID**
  - **First-party windows** (e.g., file explorer)
    - Terminate windows containing **forbidden titles** (e.g., "SAVE")
- **Outcome:** efficient mitigation of interference **with little changes**
  - **AFL++:** less than 10 new lines of code



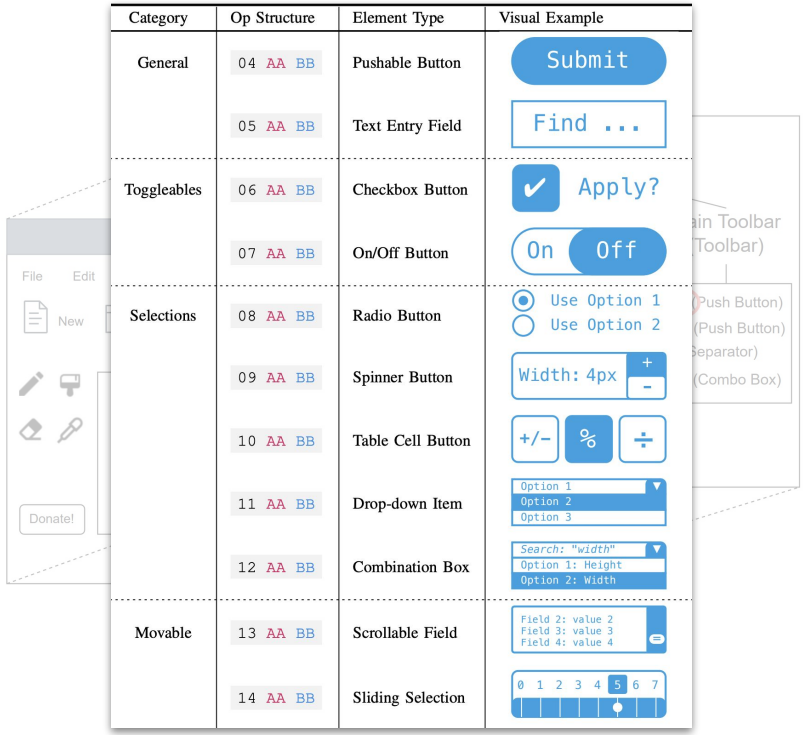
Solution: Maximizing GUI Fuzzing Precision

- **Idea:** leverage accessibility interfaces to **pinpoint concrete GUI elements**
  - Based on **AT-SPI** screen-reader framework
- Extend existing **3-byte GUI operations**
  - Capture AT-SPI's **dynamic GUI tree**



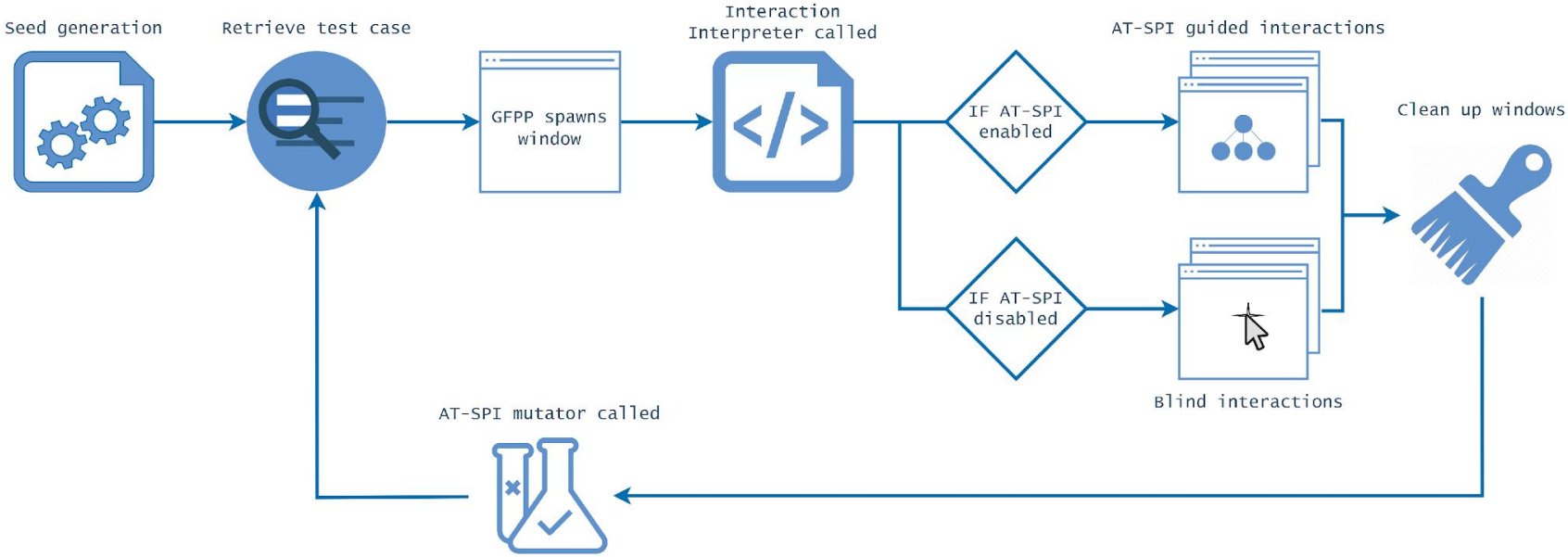
Solution: Maximizing GUI Fuzzing Precision

- **Idea:** leverage accessibility interfaces to **pinpoint concrete GUI elements**
  - Based on **AT-SPI** screen-reader framework
- Extend existing **3-byte GUI operations**
  - Capture AT-SPI's **dynamic GUI tree**
  - **Group GUI elements** into lists by type
  - Reshape **operands 2–3** as list indices
- **Outcome:** precise GUI interactions with **zero modification** to fuzzer core



Implementation: GUIFuzz++

- **World's first general-purpose GUI fuzzer for desktop GUI applications**
  - Built atop AFL++, PyAutoGUI (GUI interaction), x11-utils (window management), and AT-SPI



Evaluation: Overview

- Evaluated on **12 real-world GUI apps**
  - Three GUI frameworks: **Qt**, **GTK**, and **Xorg**
- Performed five 24-hour campaigns:
  - Each seeded with **30 random interactions**
- Performed **ablation study** of supported modes since no suitable competitors
  - **Config 1:** grey-box with AT-SPI
  - **Config 2:** grey-box without AT-SPI
  - **Config 3:** black-box with AT-SPI
  - **Config 4:** black-box without AT-SPI

Program	Description	Base GUI
Dia [27]	Graphic Design	GTK
Glaxnimate [3]	Animation	Qt
KCalc [29]	Calculator	Qt
KolourPaint [30]	Image Editor	Qt
LabPlot [31]	Data Plotting	Qt
LibreCAD [36]	3-D Modeling	Qt
MATE-calc [38]	Calculator	GTK
PlotJuggler [8]	Data Plotting	Qt
QCAD [41]	3-D Modeling	Qt
Skrooge [32]	Finance	Qt
Umbrello [33]	UML Editor	Qt
XCalc [52]	Calculator	Xorg

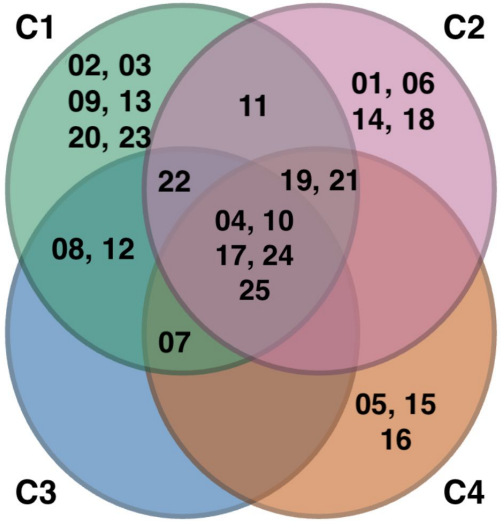
Evaluation: Speed, Code Coverage, and GUI Bugs

- **Takeaways:** on more complex apps, **code coverage *with* AT-SPI** performs best

Program	C1: Grey-box w/ AT-SPI			C2: Grey-box w/o AT-SPI			C3: Black-box w/ AT-SPI			C4: Black-box w/o AT-SPI		
	Speed	CodeCov	Bugs	Speed	CodeCov	Bugs	Speed	CodeCov	Bugs	Speed	CodeCov	Bugs
Dia	2491.2	**12990.4**	0	1631.6	12074.8	**1**	**3154.0**	12101.4	0	3047.0	11364.8	0
Glaxnimate	2562.6	**34312.2**	**2**	2850.2	30537.2	0	6625.6	30890.0	0	**7374.0**	29405.3	0
KCalc	2546.8	**8013.8**	1	2914.5	7616.4	1	**7212.6**	6625.0	1	6364.6	5473.8	**2**
KolourPaint	1859.8	**5845.2**	**3**	2019.4	4880.0	1	5579.4	5615.2	2	**7595.8**	4988.8	1
LabPlot	2190.0	25170.6	**3**	2588.8	25114.0	2	4647.8	26402.6	2	**6037.0**	**27732.2**	1
LibreCAD	2001.6	**39613.6**	1	2453.3	38703.3	1	6760.2	39076.8	0	**8453.4**	36529.4	0
MATE-calc	1989.0	1196.6	0	2421.0	1349.0	0	5542.8	1353.4	0	**6490.0**	**1359.6**	**2**
PlotJuggler	2411.2	**17979.8**	1	1968.2	16342.2	1	5389.2	17083.6	1	**5697.6**	16201.0	1
QCAD	2647.0	61598.0	0	3020.2	61568.4	**1**	**7075.6**	**61734.2**	0	6928.2	61626.4	0
Skrooge	1754.8	33794.2	0	1575.0	**34052.6**	0	6040.4	33550.4	0	**7822.2**	32642.4	0
Umbrello	2279.0	**23431.0**	**5**	3243.4	17900.8	3	6854.0	20335.8	1	**8110.0**	16693.6	2
XCalc	6344.8	**441.8**	2	5560.0	441.4	2	**28078.8**	440.6	2	27887.2	439.8	2
GEOMEAN:	2431.4	**12079.5**	**1.9**	2536.9	11409.2	1.3	6572	11635.3	1.4	**7555.6**	10961	1.5

Evaluation: New GUI Bugs

- Uncovered **23 new GUI errors**
  - Thus far, **14 are confirmed or fixed**
  - Crashes span **memory corruptions**



ID	Program	Bug Type	Brief Desc.	New	Status
01	Dia	Bad Free	Color area (transient)	✔	
02	Glaxnimate	Segfault	Improper closing	✔	
03	Glaxnimate	Segfault	Invalid cut/pastes	✔	
04	KCalc	Invalid Ptr	Inserting open parent	✔	
05	KCalc	Segfault	Left bit shift overflow		
06	KolourPaint	Heap UAF	Specific tools with undo	✔	
07	KolourPaint	Segfault	Buggy bug report menu	✔	
08	KolourPaint	Segfault	Shortcut settings dropdowns	✔	
09	KolourPaint	Segfault	Print preview zooming	✔	
10	LabPlot	Invalid Ptr	Invalid column insert	✔	
11	LabPlot	Heap UAF	Pinning spreadsheets	✔	
12	LabPlot	Heap UAF	Pinning matrices	✔	
13	LibreCAD	Heap UAF	Invalid plugin usage	✔	
14	LibreCAD	Heap UAF	Consecutive points	✔	
15	MATE-calc	Bad Free	Invalid square roots		
16	MATE-calc	Bad Free	Empty inverse trig functions	✔	
17	PlotJuggler	Segfault	Quickly close button docker	✔	
18	QCAD	Segfault	Tool use in multiple sheets	✔	
19	Umbrello	Segfault	Birds eye after discard	✔	
20	Umbrello	Heap UAF	Multiple sequence diagrams	✔	
21	Umbrello	Heap UAF	Undo after discard	✔	
22	Umbrello	Segfault	Print Preview after discard	✔	
23	Umbrello	Segfault	Cut on empty diagram	✔	
24	XCalc	FPE	Invalid modulus	✔	
25	XCalc	FPE	Invalid modulus	✔	

Conclusion: Why GUIFuzz++?

- Existing fuzzers limited to **only data-level interfaces** (e.g., files, memory, environment)
- No options today overcome **GUIs' obstacles**
  - Interaction, interference, and maximizing precision
  - Thus, GUIs remain **universally unfuzzed in practice**

Conclusion: Why GUIFuzz++?

- Existing fuzzers limited to **only data-level interfaces** (e.g., files, memory, environment)
- No options today overcome **GUIs' obstacles**
  - Interaction, interference, and maximizing precision
  - Thus, GUIs remain **universally unfuzzed in practice**
- Our solution: **GUIFuzz++**
  - A suite of **lightweight mechanisms** that introspect application windows and precisely dispatch events
  - **Outcome: systematic GUI fuzzing** via repurposing conventional GUI-agnostic fuzzers such as **AFL++**

Key Results:

23 total bugs,

14 confirmed,

AFL++ integration **coming soon!**

Thank you!

github.com/FuturesLab/GUIFuzzPlusPlus

Contact:
tanner.rowlett@utah.edu
@trowlett0
@trowlett0.bsky.social

FUTURES^3
LAB FUTURE TECHNOLGY FOR USABLE, RELIABLE, & EFFICIENT SECURITY OF SOFTWARE & SYSTEMS