ASSESSING THE EFFECTS OF SOURCE LANGUAGE ON BINARY SIMILARITY TOOLS

LANDEN DOTY
UNIVERSITY OF KANSAS

AGENDA

1. **Background & Motivation**
   What is binary diffing and why source language?
2. **Study 1 – *BSim***
   Is there an effect at all?
3. **Study 2 – *BinDiff***
   Can we isolate and understand the effects?
4. **Wrap Up**
   Final Thoughts and Q&A

BACKGROUND

WHAT IS BINARY DIFFING?

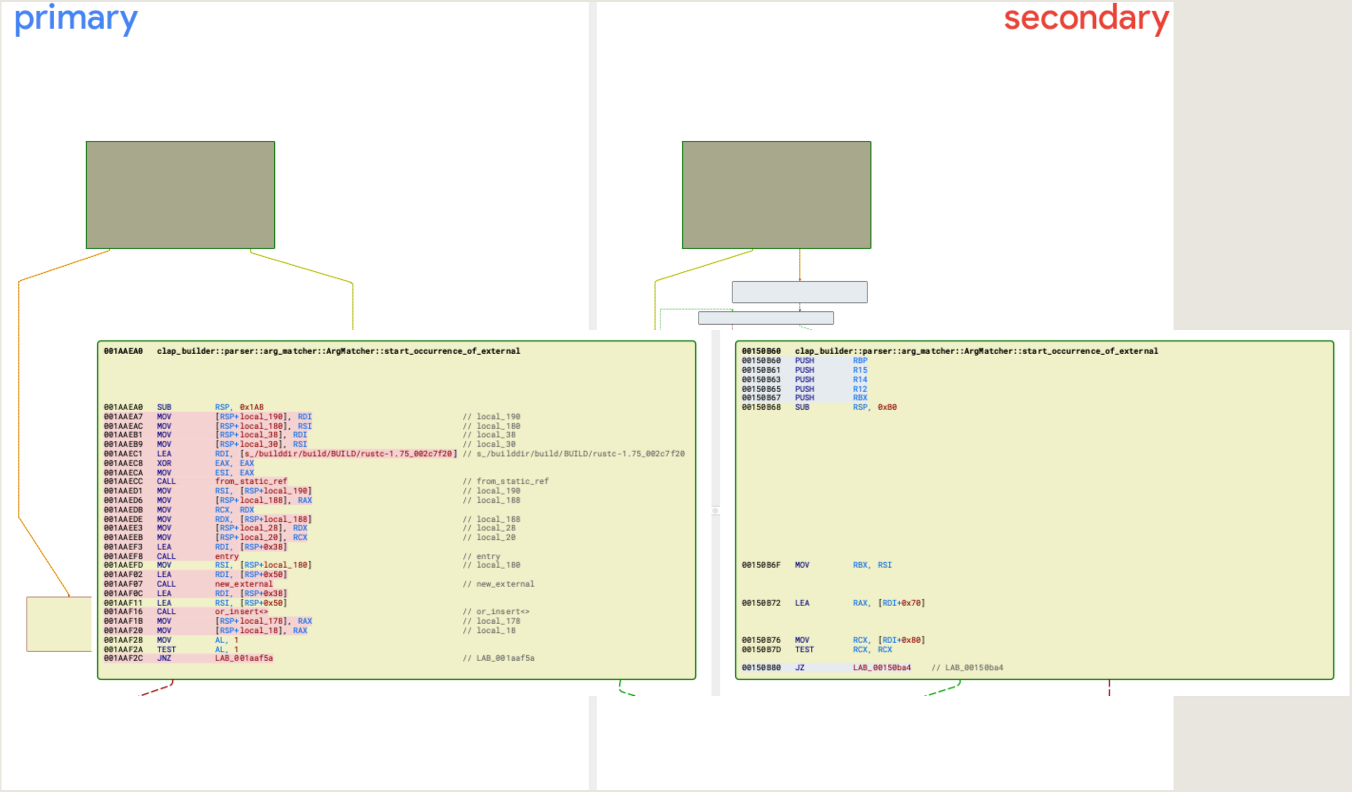
The process of comparing binary-level code segments for the purpose of identifying similarities and/or modifications

Processes include…

- Graph isomorphism
- Approximate nearest neighbor
- Machine learning

Features include…

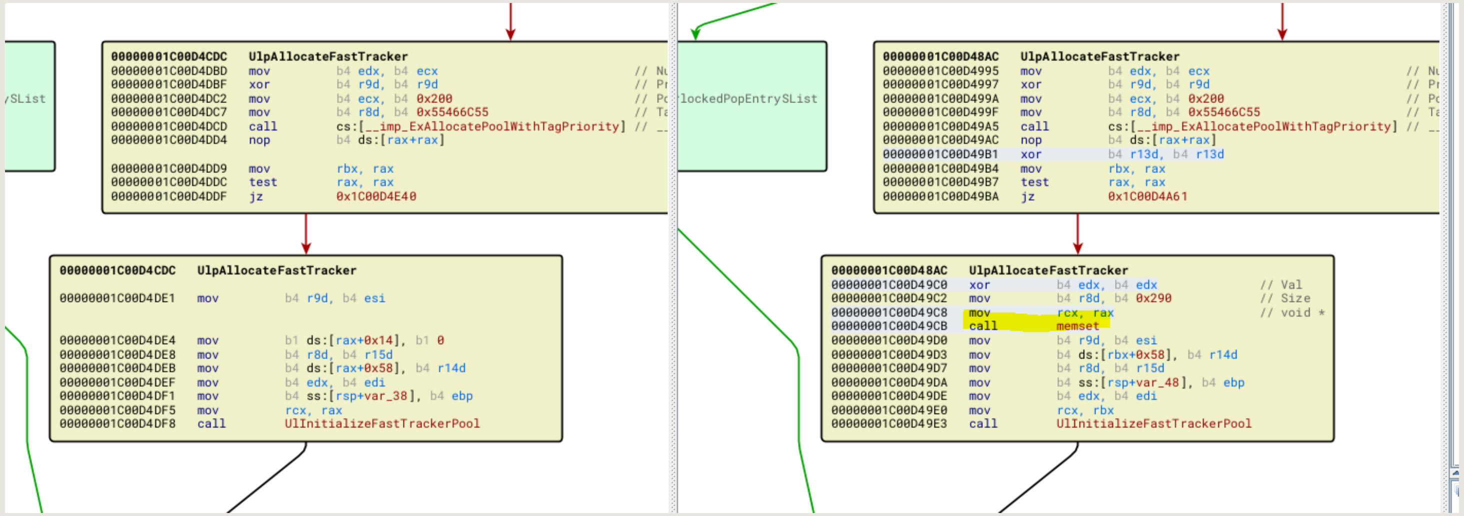
- Instruction mnemonics
- Control and data flow information
- Execution traces



BinDiff by Zynamics

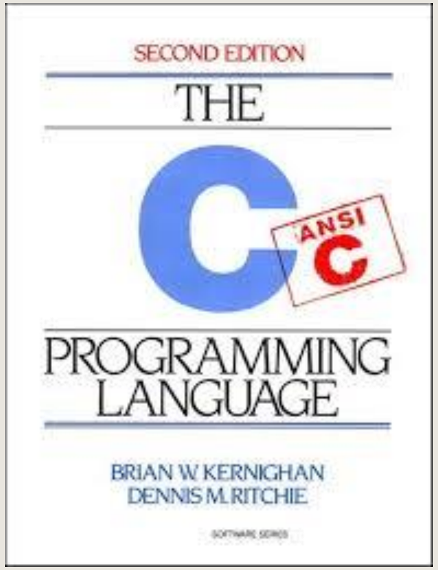
APPLICATIONS OF BINARY DIFFING

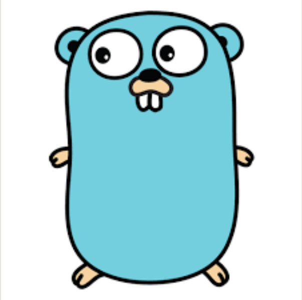
- **Vulnerability Research**
  - *"Patch diffing"*
  - *Static library detection*
- **Malware Analysis**
  - *"Family" clustering/attribution*
  - *Static detection*
- **Other**
  - *Intellectual property protection*
  - *General reverse engineering*

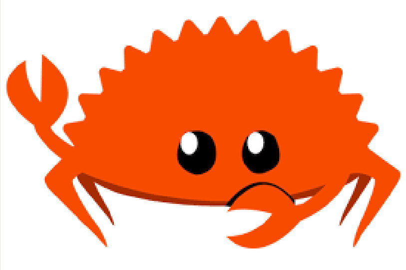


Patch diffing CVE-2022–21907 ([*Chris Hernandez*](#))

WHY LOOK AT SOURCE LANGUAGE??



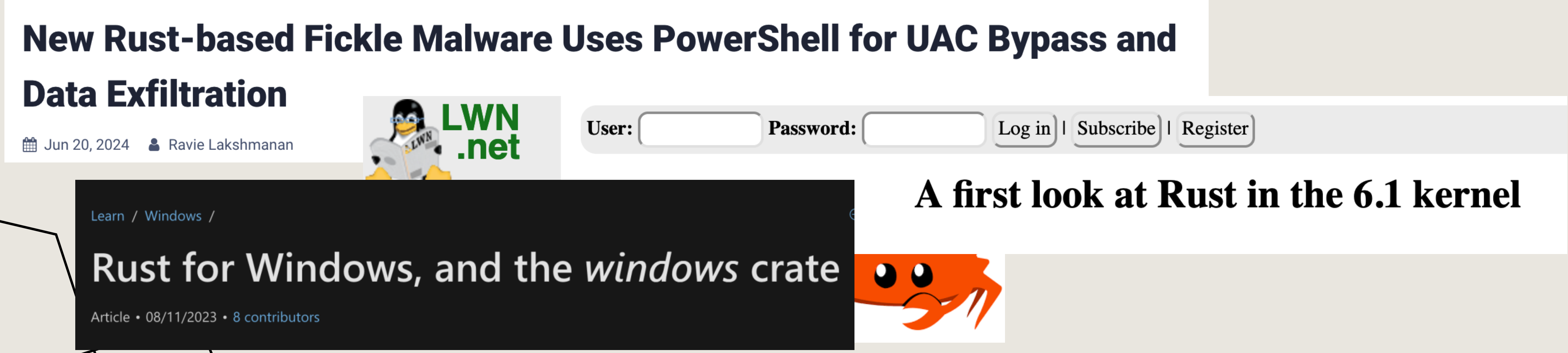




- *Hundreds* of binary similarity/diffing papers evaluated on C-compiled binaries
- The largest binary similarity benchmark is ***entirely C***
- **But there are so many more languages that compile to native code!**

LET'S JUST PICK ONE – WHY RUST?

- Love it or hate it, *Rust is here to stay*
- Finding its way into widely used system code like Windows and even the Linux Kernel
- Malware authors love it too!
  - Over 2900% uptick in cybercriminal discussions concerning Rust (*Reliaquest*)
  - *FickleStealer, RustyStealer, Embargo, Akira, etc.*



AGENDA

1. **Background & Motivation**
   What is binary diffing and why source language?

2. **Study 1 – *BSim***
   Is there an effect at all?

3. **Study 2 – *BinDiff***
   Can we isolate and understand the effects?

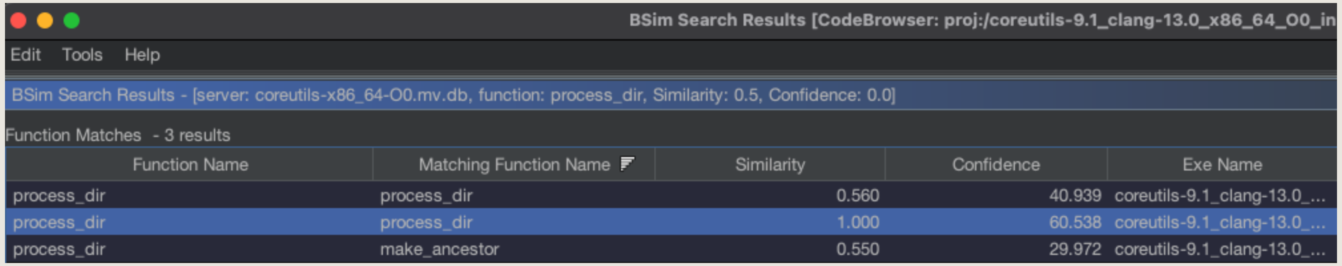
4. **Wrap Up**
   Final Thoughts and Q&A

STUDY 1 - *BSIM*

UNDERSTANDING BSIM

Quick Bio:

- **B**ehavioral **Sim**ilarity
- Developed by the NSA as an extension to Ghidra
    - Find Ghidra on Github: https://github.com/NationalSecurityAgency/ghidra/
- Built for *performant* binary similarity queries using **Locality Sensitive Hashing**
- Stores known binaries in a *database* that can be queried when analyzing new samples

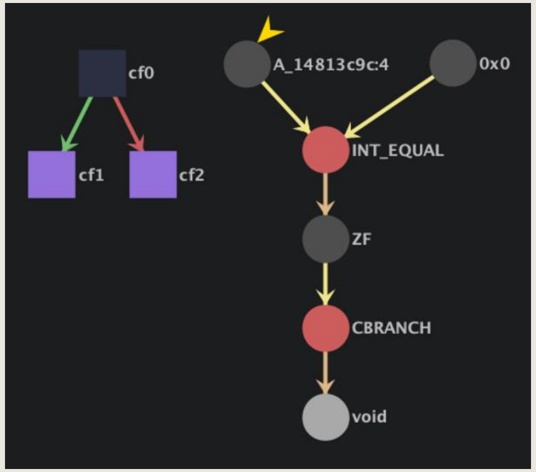


UNDERSTANDING BSIM

BSim Features

- Derived from *P-Code* lifted from the target binary
- Encode both *control flow* and *data flow information*
    - Basic block in/out-degrees
    - Edge type (True/False, Direct/Indirect calls)
    - Variable sizes
    - Opcodes
    - Variable properties like input argument, return value, constant, global, etc.

```
bl0:
 mov eax, [0x14813c9c]
 test eax, eax
 jnz bl2;
bl1:
 call func1
 mov [0x14813c9c], eax
bl2:
 ret
```

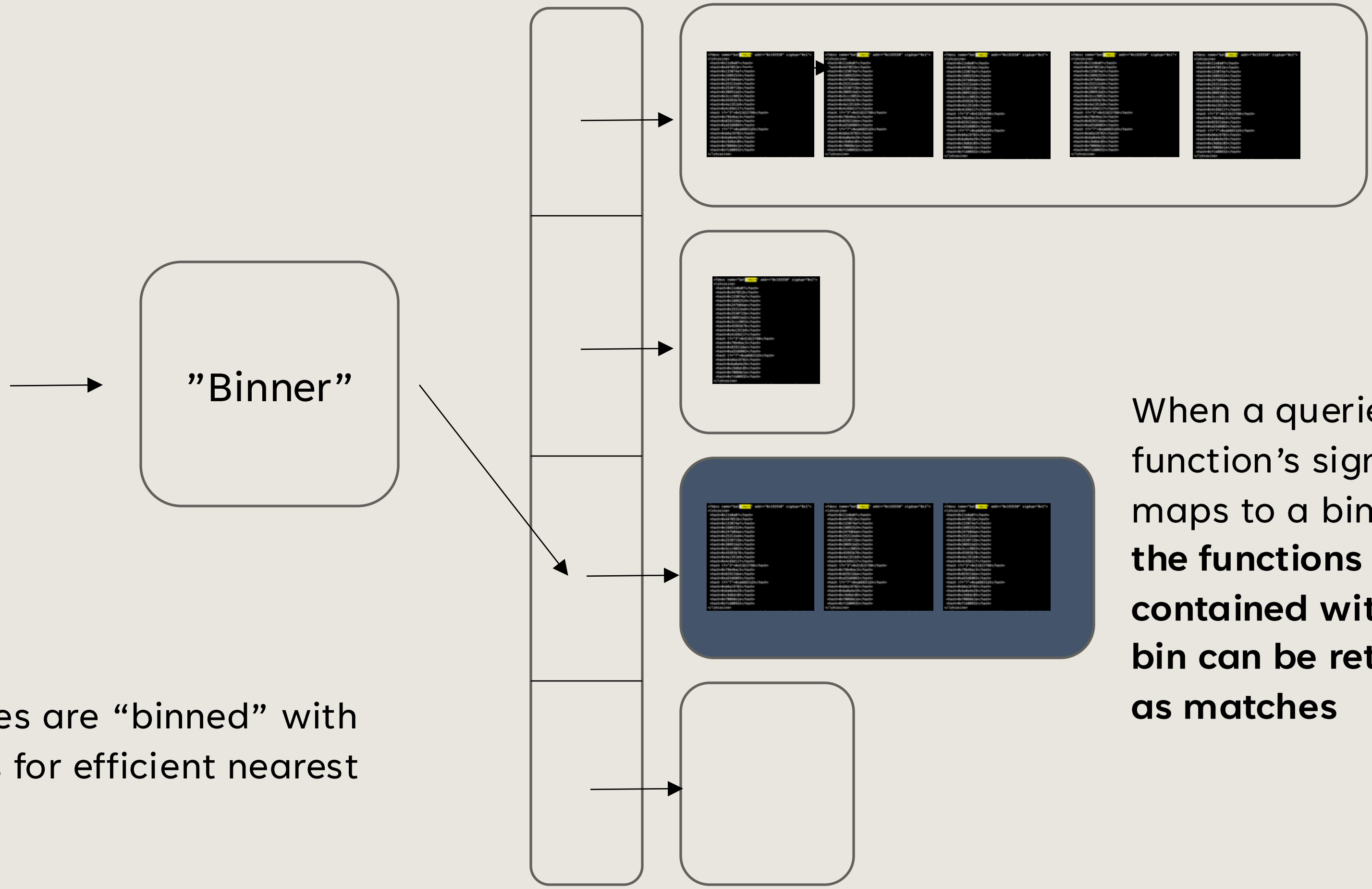


Example – *Combined* Feature Type

UNDERSTANDING BSIM

A collection of bins and their contained function signatures make up a BSim database

```
<fdesc name="bat::main" addr="0x193550" sigdup="0x1">
<lshcosine>
 <hash>0x11e0a0f</hash>
 <hash>0x44f051b</hash>
 <hash>0x133074af</hash>
 <hash>0x16092524</hash>
 <hash>0x24fb0dae</hash>
 <hash>0x25312ed4</hash>
 <hash>0x2538f15b</hash>
 <hash>0x30891bd2</hash>
 <hash>0x3ccc9053</hash>
 <hash>0x45993b76</hash>
 <hash>0x4a1351b9</hash>
 <hash>0x4c69d11f</hash>
 <hash tf="3">0x51622f88</hash>
 <hash>0x79b46ac3</hash>
 <hash>0x82921bbe</hash>
 <hash>0xa55d6083</hash>
 <hash tf="7">0xab6831d3</hash>
 <hash>0xb6a19782</hash>
 <hash>0xba0a4e29</hash>
 <hash>0xc9d6dc05</hash>
 <hash>0xf0068e1e</hash>
 <hash>0xfcb00932</hash>
</lshcosine>
```



When a queried function’s signature maps to a bin, **only the functions contained within that bin can be returned as matches**

Function signatures are “binned” with similar signatures for efficient nearest neighbor queries

UNDERSTANDING BSIM

Comparison Metrics

- ***Function coefficients*** are weighted according to a precomputed table of *term* and *inverse document frequencies*
- ***Similarity*** is essentially a weighted *cosine similarity* of function signatures using the coefficients
- ***Confidence*** uses precomputed probabilities distributed with BSim to scale similarity based on the shared features in the two signatures

$$f.coeff = \texttt{tf\_weights[f.tf]} \times \texttt{idf\_weights[f.idf]} \tag{2}$$

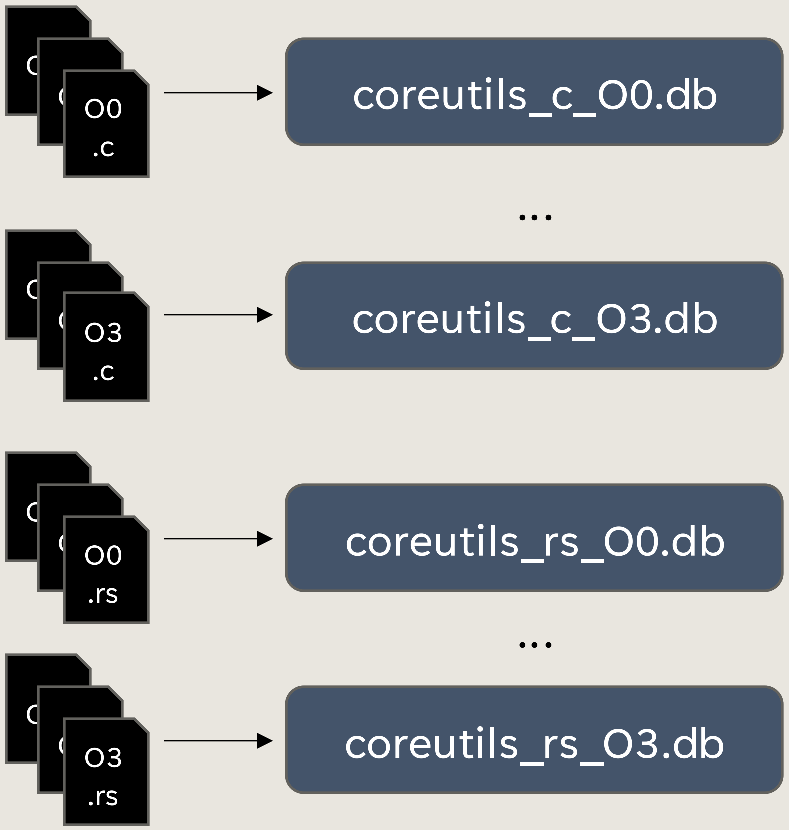
$$\texttt{Similarity} = \frac{\sum_{f \in V_F^{(1\cdot 2)}} f.coeff}{\texttt{len}(\texttt{V}_\texttt{F}^{(1)}) \times \texttt{len}(\texttt{V}_\texttt{F}^{(2)})} \tag{3}$$

$$\sum_{f \in V_F^{(1\cdot 2)}} f.coeff - \texttt{numflip} * \frac{\texttt{norm\_probflip0} + \texttt{norm\_probflip1}}{\texttt{max}(\texttt{len}(\texttt{V}_\texttt{F}^{(1)}), \texttt{len}(\texttt{V}_\texttt{F}^{(2)}))}$$
$$- \texttt{diff} * \frac{\texttt{norm\_probdiff0} + \texttt{norm\_probdiff1}}{\texttt{max}(\texttt{len}(\texttt{V}_\texttt{F}^{(1)}), \texttt{len}(\texttt{V}_\texttt{F}^{(2)}))} + \texttt{addend} \tag{4}$$

EXPERIMENTAL SETUP

Table 1: Experiment Databases

Language	Compiler	Package	Opt. Level	Functions
C	Clang	GNU Coreutils	O0	17,471
C	Clang	GNU Coreutils	O1	11,981
C	Clang	GNU Coreutils	O2	12,380
C	Clang	GNU Coreutils	O3	11,659
Rust	Rustc	Uutils Coreutils	O0	563,638
Rust	Rustc	Uutils Coreutils	O1	147,420
Rust	Rustc	Uutils Coreutils	O2	128,553
Rust	Rustc	Uutils Coreutils	O3	113,144



DATA COLLECTION

1. Analyze each function in each binary in Ghidra

O0
.c

```java
public class QueryAllFunctions extends GhidraScript {

        private static final int MATCHES_PER_FUNC = -1;
        private static final double SIMILARITY_BOUND = 0.0;
        private static final double CONFIDENCE_BOUND = 0.0;
```

coreutils_c_O0.db

2. Record *all* matches from BSim via a *GhidraScript*

We query only on the database containing binaries at the same optimization level and compiler!


```
uuid,queryfn,resultfn,similarity,confidence
"09c0927e-24e7-41d4-a2de-f1220c59589f","blake2b_stream","blake2b_stream","1.0","72.7018277366852"
"7daacbb5-d7db-4e20-aaad-233b8f6ae5ab","base32_encode","base32_encode","0.5738623658164462","107.69871409314666"
"c742d146-2024-4b04-98b3-e51f3c96e312","base32_encode_alloc","base64_encode_alloc","0.7749401219531178","52.694200771660334"
"c742d146-2024-4b04-98b3-e51f3c96e312","base32_encode_alloc","base32_encode_alloc","0.7749401219531178","52.694200771660334"
"72e9ac88-0d52-4adf-9bf8-61c51977fa09","isbase32","isbase64","0.9999999999999998","16.153736089120216"
"72e9ac88-0d52-4adf-9bf8-61c51977fa09","isbase32","isbase32","0.9999999999999998","16.153736089120216"
"b7cea07c-133d-4a7f-8f4e-a218ee18acab","base32_decode_ctx_init","base64_decode_ctx_init","1.0000000000000002","10.102697049120216"
"b7cea07c-133d-4a7f-8f4e-a218ee18acab","base32_decode_ctx_init","base32_decode_ctx_init","1.0000000000000002","10.102697049120216"
"3779dac1-535e-4a21-a081-829a44687683","base32_decode_ctx","base32_decode_ctx","0.31222333542072445","41.31937972078967"
```


3. Record function names, similarity, and confidence of each match

RESULTS

RQ1. Does source language degrade binary similarity results and, if so, to what extent?

Accuracy – Rate at which BSim returned a matching symbol

Table 2: Overall Accuracy - Baseline (C)

Query Level, DB Level	Accuracy (%)
O0,O0	100.0
O0,O1	69.20
O0,O2	67.79
O0,O3	66.60
O1,O1	100.00
O1,O2	98.83
O1,O3	97.80
O2, O2	100.00
O2,O3	99.26
O3,O3	100.00

Table 3: Overall Accuracy - Experimental (Rust)

Query Level, DB Level	Accuracy (%)
O0,O0	93.61
O0,O1	45.93
O0,O2	33.72
O0,O3	32.43
O1,O1	99.17
O1,O2	92.35
O1,O3	91.09
O2, O2	99.24
O2,O3	97.90
O3,O3	98.96

We observe a *consistently lower* accuracy rate in our Rust dataset

RESULTS

RQ2. How does source language affect clustering-based binary similarity tools?

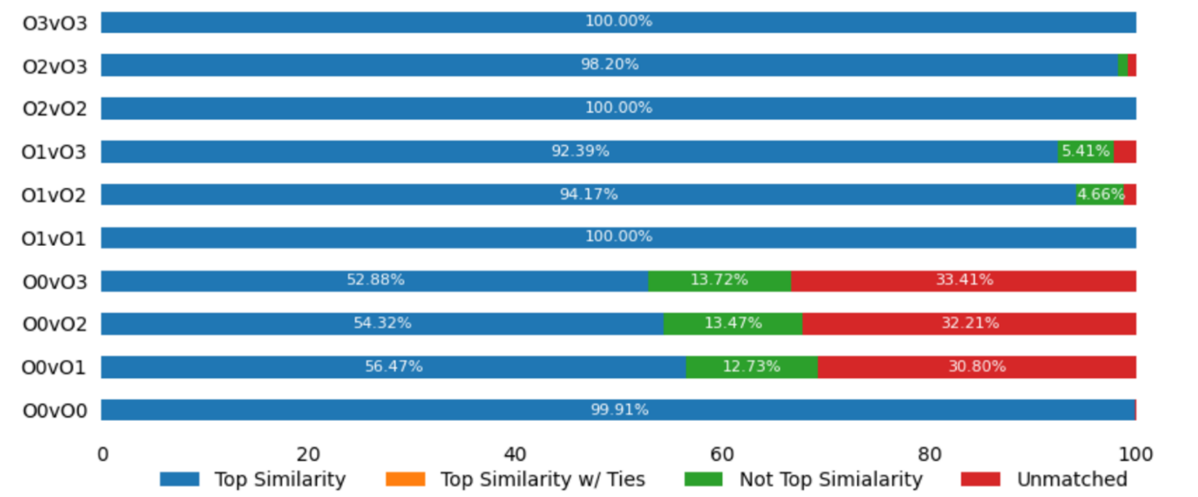


Figure 6: Top Match Accuracy - Baseline (C)

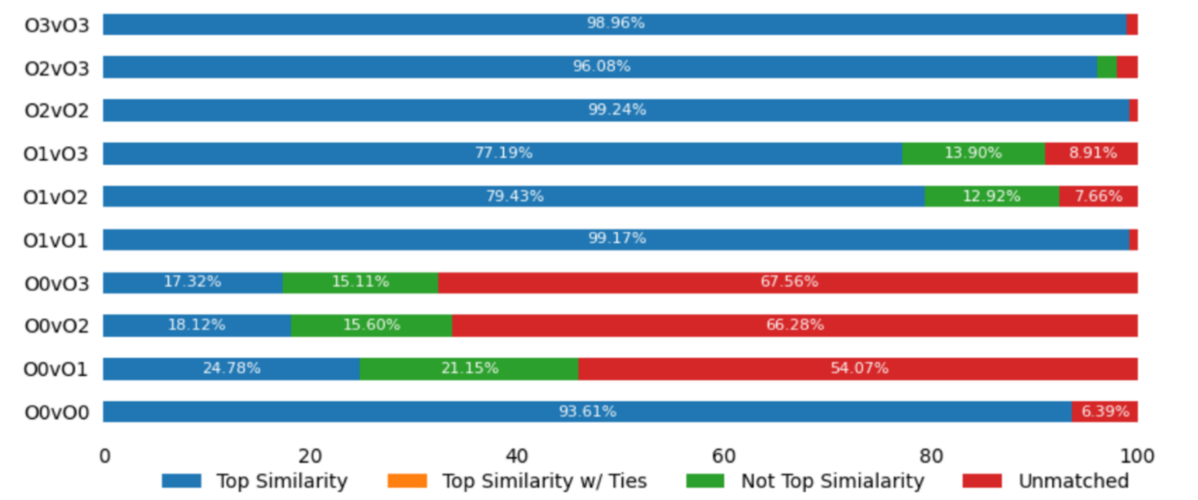


Figure 7: Top Match Accuracy - Experimental (Rust)

We observe a higher rate of false positives *before* the correct function is matched

RESULTS

RQ2. How does source language affect clustering-based binary similarity tools?

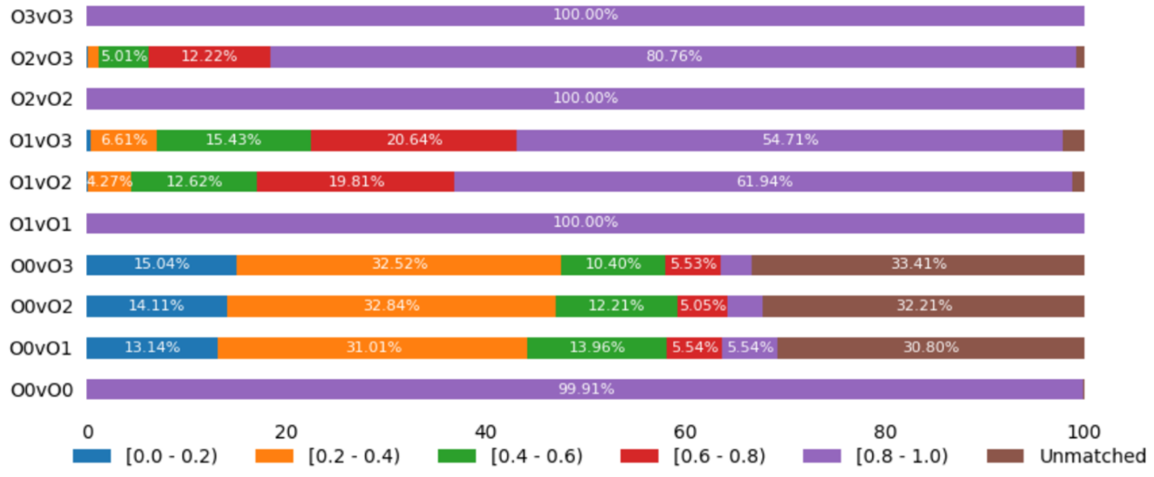


Figure 8: Match Accuracy by Similarity - Baseline (C)

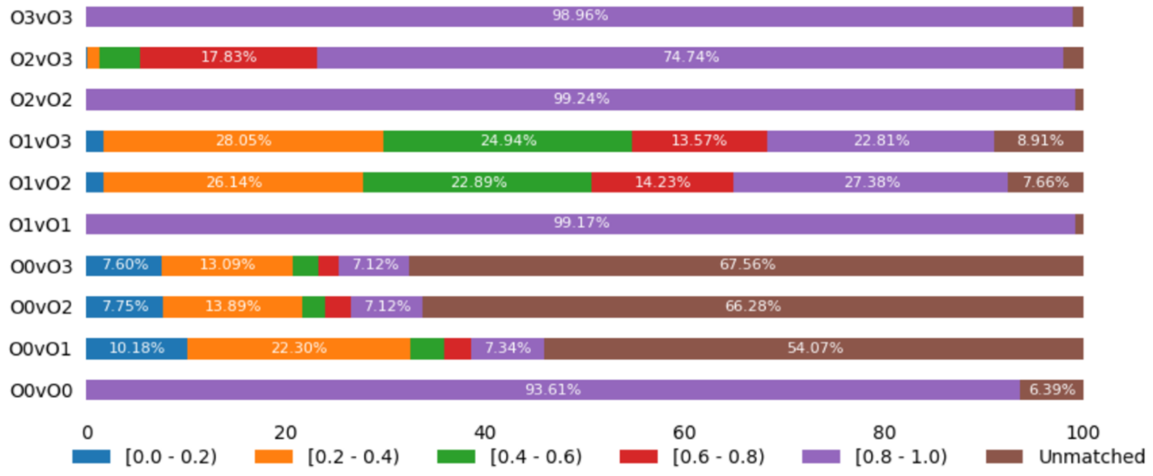


Figure 9: Match Accuracy by Similarity - Experimental (Rust)

We observe a larger distribution of *low similarity* but *correct* matches

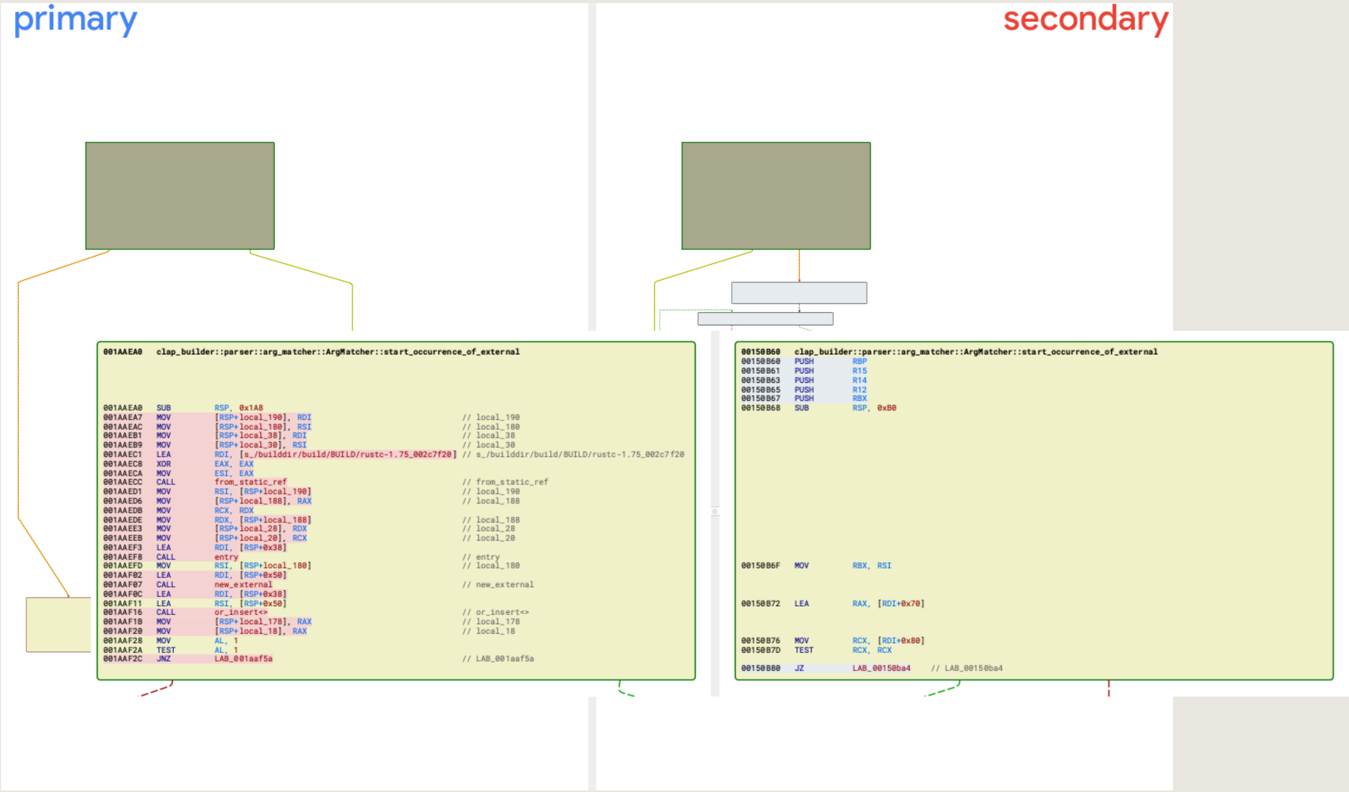
AGENDA

1. **Background & Motivation**
   What is binary diffing and why source language?
2. **Study 1 – *BSim***
   Is there an effect at all?
3. **Study 2 – *BinDiff***
   Can we isolate and understand the effects?
4. **Wrap Up**
   Final Thoughts and Q&A

STUDY 2 - *BINDIFF*

WHY BINDIFF?

- *BinDiff* performs similarity matching using only *structural features*
    - Control flow and call graphs
    - Basic block edges
    - Instruction mnemonics
- **We can use BinDiff to isolate and measure the degree of structural change in Rust binaries compared to C!**



BinDiff by Zynamics

BINDIFF DEBRIEF

Comparison Metrics

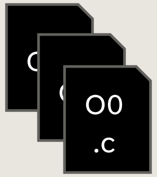
- ***Similarity*** is a weighted sum of the ratio of matched edges, basic blocks, and instructions and is multiplied by *confidence*
- ***Confidence*** is the average algorithm attribute confidence/quality

Similarity	Confidence	Address	Primary Name	Type	Address	Secondary Name
0.10	0.99	0012C8E0	uu_chroot::uu_chroot::uumai...	Normal	00119680	uu_chroot::uu_chroot::uumai...
0.01	0.02	0012CD80	uu_chroot::uu_chroot::uumai...	Normal	00115480	alloc::raw_vec::RawVec<u8,_...
0.13	0.34	00134B40	uu_chroot::uu_chroot::uu_app	Normal	0011DA10	uu_chroot::uu_chroot::uu_app
0.15	0.39	001363C0	uu_chroot::uu_chroot::set_g...	Normal	00120EA0	core::ptr::drop_in_place<st...
0.05	**0.10**	**00135530**	**uu_chroot::uu_chroot::set_c...**	**Normal**	**0011F3F0**	**uu_chroot::uu_chroot::set_c...**

The first match was by name, which has a high confidence/quality. However, the observed similarity is very low as the function's CFG is greatly changed between versions.

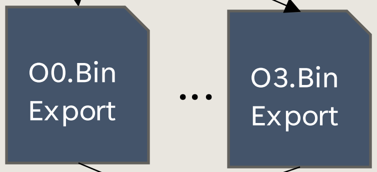
DATA COLLECTION

1. Analyze each function in Ghidra




```
/**
 * Exports Ghidra disassembly data into BinExport v2 format.
 *
 * @author Andras Gemes
 * @author Christian Blichmann
 */
public class BinExportHeadless extends GhidraScript {
```

2. Generate *.BinExport* for each binary



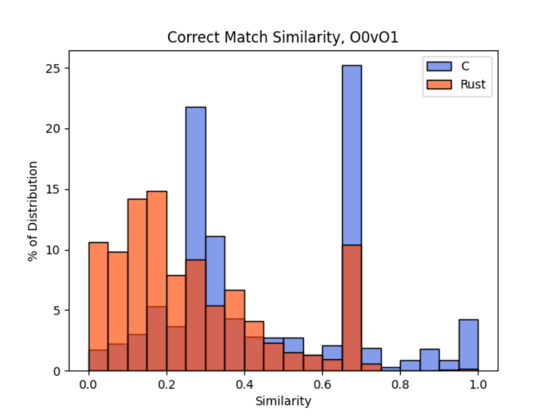
3. Perform comparison via BinDiff

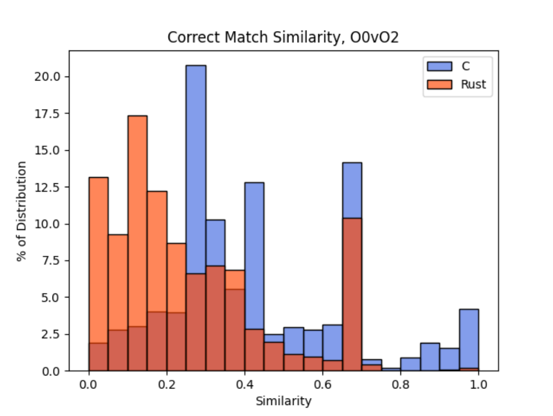
	primary fn	secondary fn	primary bin	secondary bin	similarity	confidence
0	uu_du::uu_du::uumain<core::iter::adapters::clo...	uu_du::uu_du::uumain<core::iter::adapters::clo...	uutils-coreutils_rustc_x86_64_O0_du.BinExport	uutils-coreutils_rustc_x86_64_O2_du.BinExport	0.007702	0.109097
1	uu_du::uu_du::uumain::uumain<core::iter::adapt...	hashbrown::map::HashMap<uu_du::FileInfo,_(),_s...	uutils-coreutils_rustc_x86_64_O0_du.BinExport	uutils-coreutils_rustc_x86_64_O2_du.BinExport	0.141888	0.368546
2	uu_du::uu_du::Stat::new	uu_du::uu_du::Stat::new	uutils-coreutils_rustc_x86_64_O0_du.BinExport	uutils-coreutils_rustc_x86_64_O2_du.BinExport	0.257144	0.880797
3	uu_du::uu_du::read_block_size	uu_du::uu_du::read_block_size	uutils-coreutils_rustc_x86_64_O0_du.BinExport	uutils-coreutils_rustc_x86_64_O2_du.BinExport	0.179667	0.377541
4	uu_du::uu_du::du	uu_du::uu_du::du	uutils-coreutils_rustc_x86_64_O0_du.BinExport	uutils-coreutils_rustc_x86_64_O2_du.BinExport	0.071401	0.305655
...	...	...	...	...	...	...

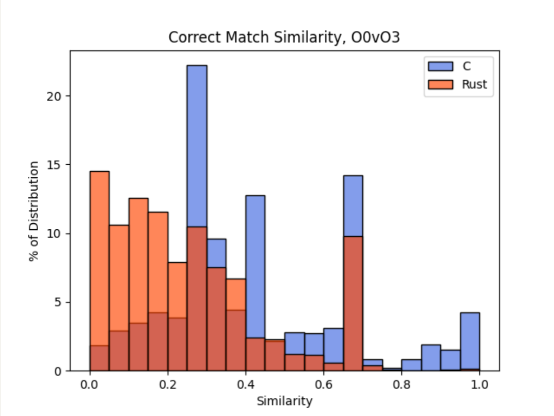
4. Parse *.BinDiff* result file and record results

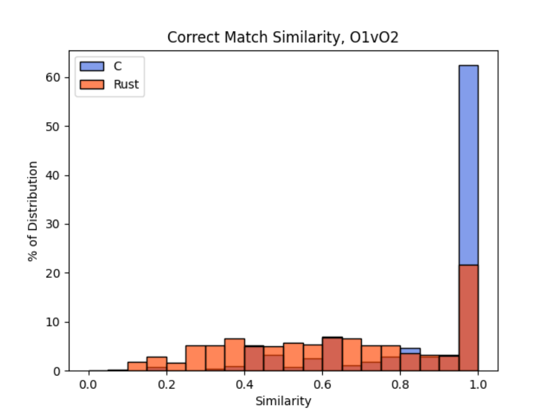
RESULTS

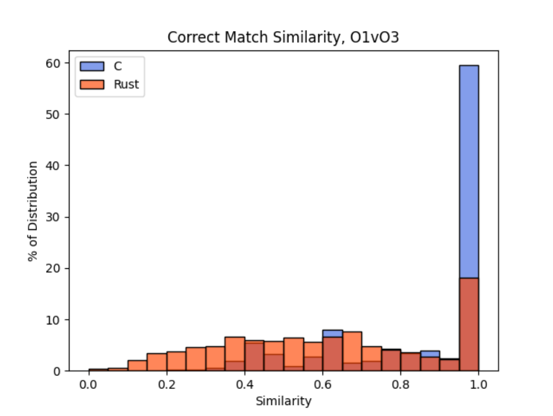
RQ1. How significantly do Rust binaries change structurally, when compiled at varied optimizations, compared to C binaries?

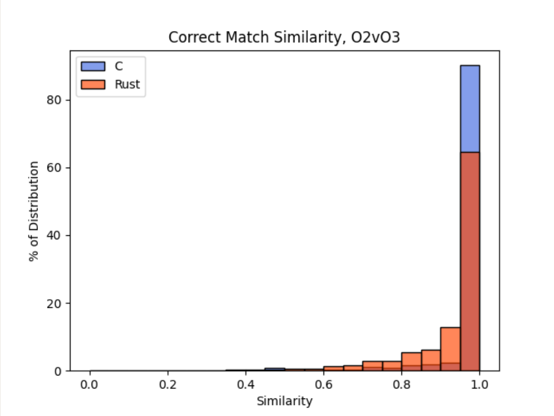






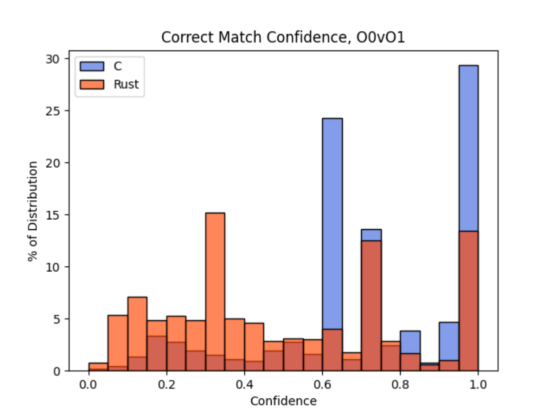


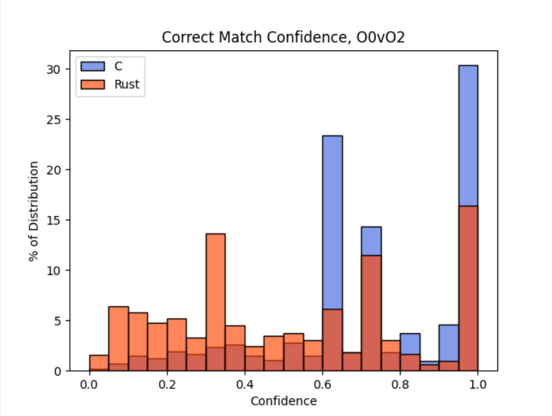


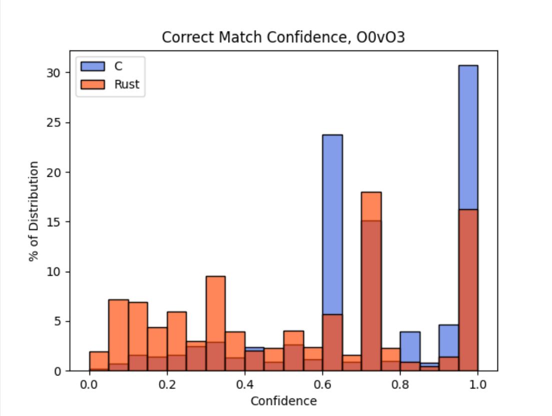


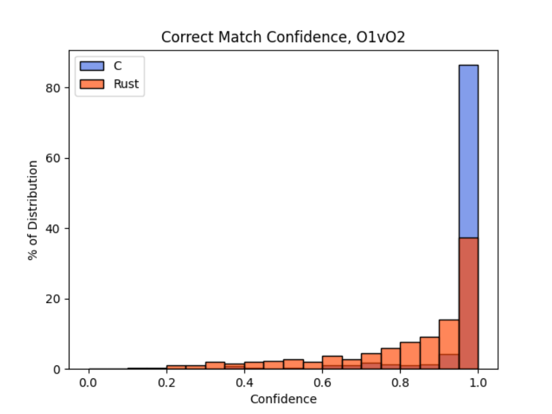
RESULTS

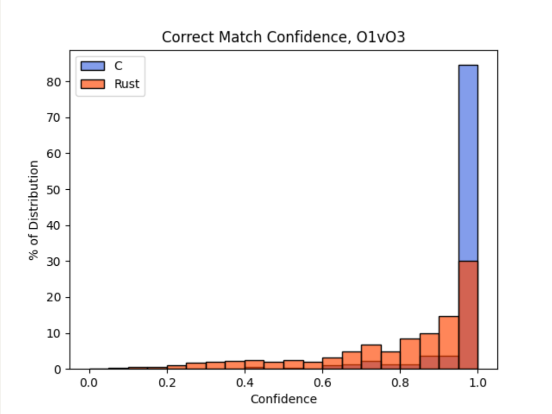
RQ2. Do the structural changes of Rust binaries degrade the quality of matches?

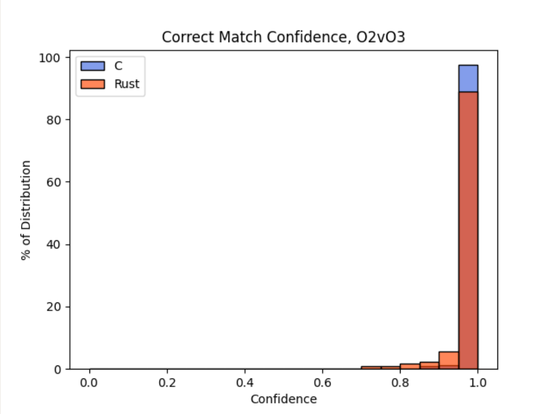












AGENDA

1. **Background & Motivation**
   What is binary diffing and why source language?
2. **Study 1 – *BSim***
   Is there an effect at all?
3. **Study 2 – *BinDiff***
   Can we isolate and understand the effects?
4. **Wrap Up**
   Final Thoughts and Q&A

THANK YOU!

Landen Doty
landoty@ku.edu