Accurate Compiler and Optimization Independent Function Identification

Derrick McKee[1,2], Nathan Burow[2], Mathias Payer[3]

1. Purdue University 2. MIT Lincoln Laboratory 3. EPFL

PURDUE UNIVERSITY Department of Computer Science

EPFL

hexhive

Motivation

- Reverse engineering binaries is required for many purposes
  - Malware analysis and family identification
  - Library version and patch application
  - Copyright violation detection
- 10^5 new daily malware samples demands an automated solution

Why is reverse engineering binaries difficult?

- No debug symbols or type information
- Highly dependent on compilation environment
  - `strlen` assembly can change by up to 70%
- Similar binary code implies function similarity, but dissimilar code does not imply differences in function semantics

Existing Solutions

- **Static**
  - **BinDiff** - Control-flow Graph Isomorphism
  - **Asm2Vec** - NLP embedding
  - **IDA** - Proprietary function signatures

- **Dynamic**
  - **BLEX** - Measured code feature vector
  - **IMF-SIM** - Measured code feature vector

All existing solutions measure code properties, which are fragile and highly variable.

What is IOVec Function Identification?

- Semantic binary function identifier
- Requires no source code
- Sets of program state changes is the unique function fingerprint
- Highly resistant to changes in compilation environment, purposeful obfuscation, and architecture changes

IOVFI uses program state changes to identify functions in stripped binaries.

Input/Output Vectors (IOVecs)

- Stores an initial program state, and an expected program state after function execution

- A function “accepts” an IOVec if it executes to completion starting with the initial state, and the resulting program state matches the expected program state

- The set of accepted IOVecs is the function signature

IOVecs store program state transformations largely preserved by all compilers.

IOVec Function Identification

```c
int my_func(int a, int b, int* c) {
    *c = a / b;
    return 0;
}
```

IOVec

Arg 0: $\mathbb{Z}$
Arg 1: $\mathbb{Z} - \{0\}$
Arg 2: Any valid address
Memory: Any value

Input Program State

Return: 0
Syscalls: None
Memory: Value at c contains a / b

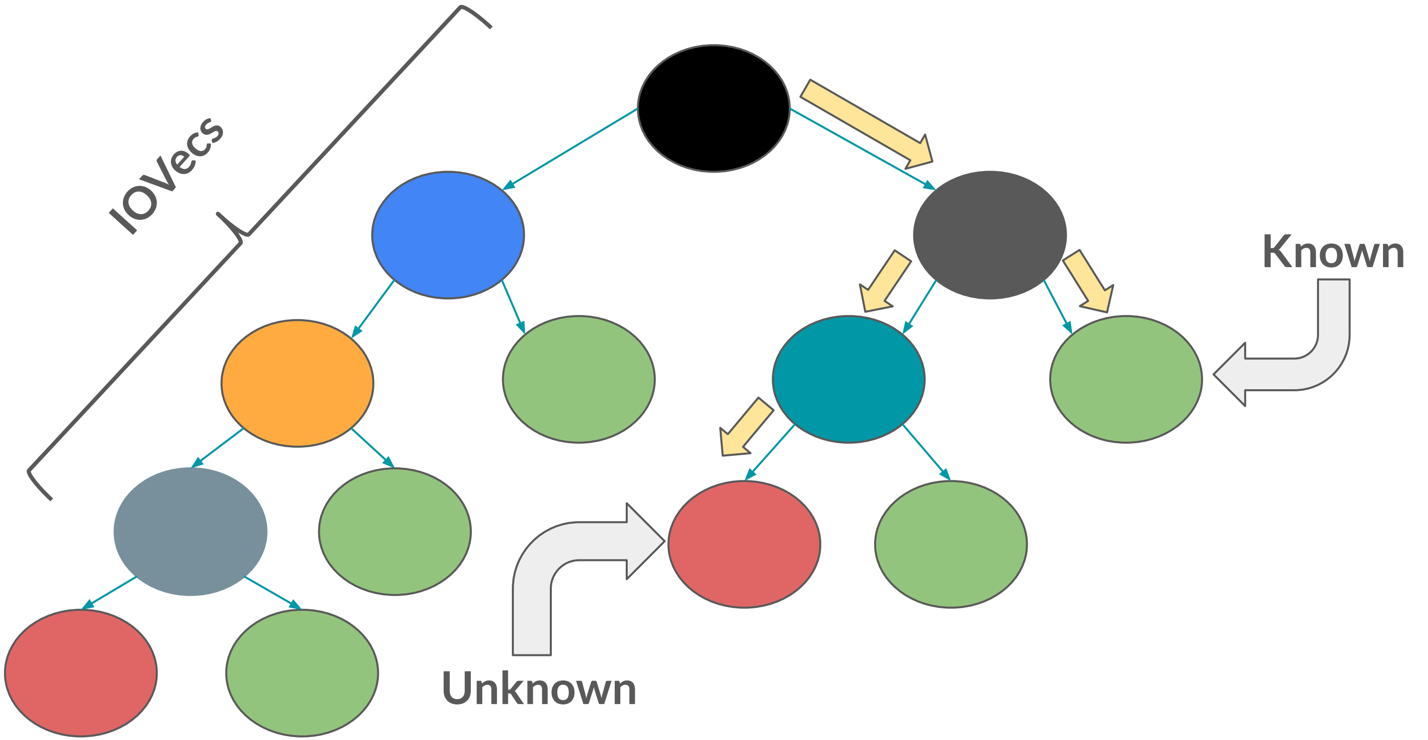
Expected Program State

IOVFI Training Phase

- IOVFI utilizes a guided mutational fuzzer to discover IOVec sets for each function in a binary
- Each function is given every generated IOVec
- A binary tree is generated with functions on leaves, and IOVecs as internal nodes
- Function identification involves traversing the binary tree

IOVFI classifies functions by creating a searchable binary tree of IOVecs.

IOVFI Binary Tree Example



IOVFI Experimental Setup

- We compile `coreutils` using Clang and GCC at -O{0,1,2,3}
- We generate a binary tree from `wc`, `realpath`, and `uniq`
- We identify functions in `du`, `dir`, `ls`, `ptx`, `sort`, `true`, `logname`, `whoami`, `uname`, and `dirname`
- We report F-Score, the harmonic mean of precision and recall

Comparison with BinDiff 6

Evaluation Compilation Environment \ Binary Tree Compilation Environment		O0 Clang: IOVFI F-Score	O0 Clang: Improvement over BinDiff	O0 GCC: IOVFI F-Score	O0 GCC: Improvement over BinDiff
O0	Clang	**.856**	**24%**	.836	53%
	GCC	.823	48%	**.838**	**22%**
O1	Clang	.735	87%	.734	99%
	GCC	.695	67%	.690	68%
O2	Clang	.696	122%	.686	140%
	GCC	.674	100%	.659	133%
O3	Clang	.692	132%	.689	140%
	GCC	.755	139%	.748	201%

Widening Accuracy Gap

Comparison with Asm2Vec

Evaluation Compilation Environment \ Binary Tree Compilation Environment		O0 Clang (Asm2Vec F-Score)	O0 Clang (IOVFI F-Score)	O0 GCC (Asm2Vec F-Score)	O0 GCC (IOVFI F-Score)
O0	Clang	**.952**	**.856**	.224	.836
	GCC	.296	.823	**.951**	**.838**
O3	Clang	.0656	.692	.0370	.689
	GCC	.0519	.755	.0108	.748

Large Binary Accuracy

	O1		O3	
	Clang	GCC	Clang	GCC
libz	.717	.850	.765	.772
libpng	.633	.695	.629	.639
libxml2	.699	.802	.700	.733

Cross Architecture Accuracy

	O0		O3	
	Clang	**GCC**	**Clang**	**GCC**
wc	.835	.805	.795	.860
realpath	.820	.803	.737	.842
uniq	.880	.866	.796	.877

Conclusion

- IOVFI semantically identifies functions in binaries
- Uses program state transformations as function fingerprints
- Resilient to broad changes in compilation environments and architecture, a first-in-class feature
- Source available at https://github.com/HexHive/IOVFI

Email: **derrick@geth.systems**
Twitter: **@unbound_brewer**

PURDUE UNIVERSITY | Department of Computer Science

EPFL

hexhive