Table 9: List of top-venue papers related to binary diffing in the last decade. All of them evaluate the binary code or bytecode produced by default compiler optimization settings. The fifth column summarizes the code representations that are used to measure similarity. "BB" and "CFG" are short for basic block and control flow graph, respectively. ● in the last column denotes this paper compares the proposed method with the industry-standard binary diffing tool: BinDiff; ◐ indicates this paper relies on BinDiff's output; and ○ means this paper mentions BinDiff as one of related tools.

#	Authors	Conference	Topic	Representation	B.D.
1	Yu et al.	AAAI'20	A new model of neural networks that considers the order of CFG's nodes.	BB & CFG	
2	Ding et al.	S&P'19	Learning-based binary code clone search against compiler-level transformation.	Function	○
3	Zuo et al.	NDSS'19	Cross-architecture basic-block similarity comparison using neural network.	Basic block	
4	Gao et al.	FSE'18	Learning-based vulnerability search via function semantic emulation.	Function	
5	Liu et al.	ASE'18	Binary code similarity detection with deep neural network.	Function	●
6	Gao et al.	ASE'18	A semantic learning model that considers CFG and data flow graph (DFG).	CFG & DFG	
7	David et al.	ASPLOS'18	Scalable vulnerable procedure search in stripped firmware images.	BB slices	●
8	Caliskan-Islam et al.	NDSS'18	De-anonymize the authors of binary code by detecting their unique coding style.	Function	
9	Xu et al.	CCS'17	Cross-platform binary diffing via neural network-based graph embedding.	BB & CFG	○
10	Wang et al.	ASE'17	Identify similar functions in binary code using in-memory fuzzing.	Function	●
11	Kargén et al.	ASE'17	Scalable instruction trace alignment of binary code.	Trace	○
12	Ming et al.	Security'17	Binary diffing via equivalence checking of API call sliced segments.	API call slices	●
13	Xu et al.	ICSE'17	A patch analysis framework to learn security patch/vulnerability patterns.	BB & CFG	◐
14	David et al.	PLDI'17	Binary procedure similarity detection via re-optimizing basic block strands.	BB slices	●
15	Xu et al.	S&P'17	Known cryptographic function detection via bit-precise symbolic loop mapping.	Loop	●
16	Chandramohan et al.	FSE'16	Scalable binary function matching via selective inlining.	BB tracelets	●
17	Su et al.	FSE'16	Code relatives detection via efficiently matching instruction dependency graph.	Instruction graph	
18	Kalra et al.	FSE'16	Detect application-affecting changes across two library binary versions.	Function	●
19	Feng et al.	CCS'16	Scalable bug search in firmware images with CFG's numeric feature vector.	BB & CFG	○
20	Ding et al.	KDD'16	Large-scale assembly function clone search using MapReduce subgraph search.	Function	○
21	David et al.	PLDI'16	Compare similar binary procedures with small comparable basic block strands.	BB slices	●
22	Eschweiler et al.	NDSS'16	Cross-architecture search for similar binary functions using bipartite matching.	BB & CFG	●
23	Graziano et al.	Security'15	Compare malware code submitted to sandboxes to study malware developments.	BB & CFG	◐
24	Pewny et al.	S&P'15	Cross-architecture bug search in binary code via basic block sampling.	BB & CFG	○
25	Dalla Preda et al.	POPL'15	A formal semantic model for mixed syntactic/semantic binary similarity analysis.	Basic block	◐
26	Luo et al.	FSE'14	Software plagiarism detection by matching equivalent basic block subsequence.	Basic block	●
27	Egele et al.	Security'14	Search similar functions among binary code with dynamic similarity testing.	Function	●
28	David et al.	PLDI'14	Binary function search by comparing fixed length of basic block tracelets.	BB tracelets	○
29	Sharma et al.	OOPSLA'13	Verify the equivalence of two loops in binary code for loop optimizations.	Loop	
30	Schuster et al.	CCS'13	Compare control flow traces of various inputs to identify backdoor regions.	Trace	○
31	Jang et al.	Security'13	Recover the evolutionary relationship among a set of binary programs.	BB& Behavior	○
32	Hu et al.	ATC'13	Scalable malware clustering with instruction n-grams after generic unpacking.	Instruction n-grams	
33	Calvet et al.	CCS'12	Cryptographic function detection in binary code with I/O parameter sampling.	Loop	
34	Zhang et al.	ISSTA'12	Algorithm plagiarism detection with dynamic value-based approaches.	Runtime invariants	
35	Jang et al.	CCS'11	Large-scale malware similarity detection using feature hashing.	N-gram & Behavior	○
36	Chaki et al.	KDD'11	Binary code provenance-similarity detection with supervised learning.	Function	○
37	Jhi et al.	ICSE'11	Software plagiarism detection by matching critical runtime invariant values.	Runtime invariants	
38	Rosenblum et al.	ISSTA'11	Recover the provenance of source language and compiler from binary code.	Function	○
39	Comparetti et al.	S&P'10	Identify similar malicious functionality that is not active at run time.	BB & CFG	○
40	Fredrikson et al.	S&P'10	Synthesize behavior-based malware specifications to match new malware.	Syscall graph	
41	Wang et al.	CCS'09	Software plagiarism detection via system call dependence graph comparison.	Syscall graph	
42	Hu et al.	CCS'09	Find similar malware variants via fast function-call graph comparison.	Call graph	○
43	Bayer et al.	NDSS'09	Scalable malware clustering with behavioral profiles.	Behavioral profile	○
44	Sæbjørnsen et al.	ISSTA'09	Binary clone detection by matching normalized instruction sequences.	Instruction n-grams	○
45	Brumley et al.	S&P'08	Automatic 1-day exploit generation via patch difference analysis.	BB & CFG	◐