PENNSTATE
1855

MetaHunt: Towards Taming Malware Mutation via Studying the Evolution of Metamorphic Virus

Li Wang, Dongpeng Xu, Jiang Ming, Yu Fu, Dinghao Wu

11-15-2019

Presented by Shixiong Jing

Software Systems Security

Innovating with Integrity. Computing with Trustworthiness.

Background

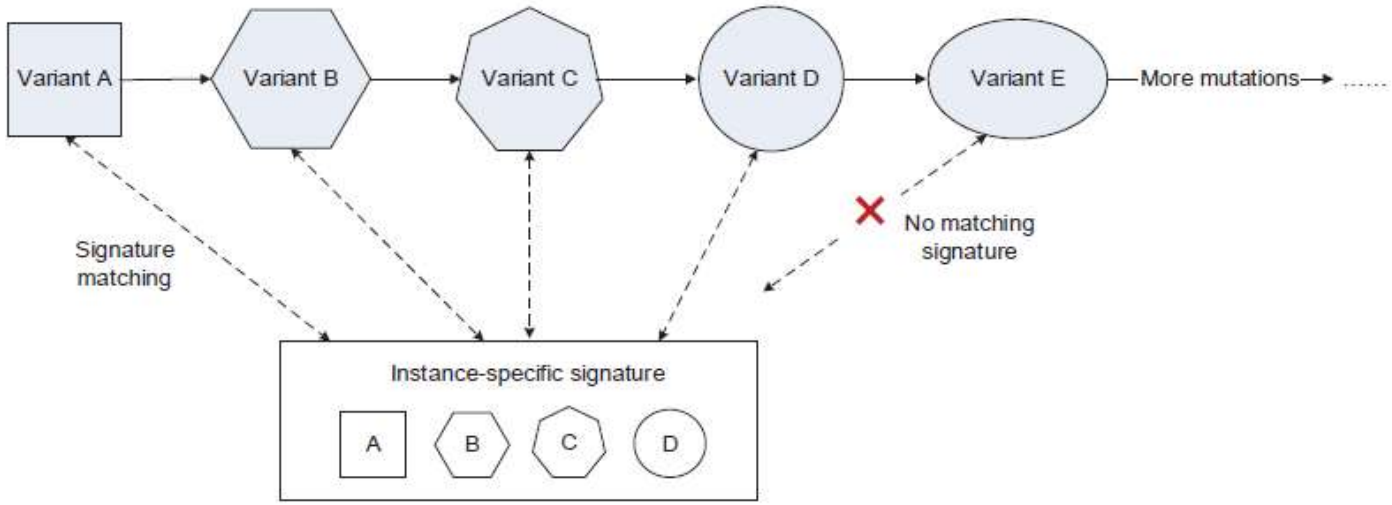
- Malware Situation
  - Increasing damages
    - 1 in 13 web requests lead to malware today. (Symantec)
    - Ransomware damage costs exceed $5 billion in 2017, 15 times the cost in 2015. (CSO online)
    - The Equifax breach costs the company over $4 billion, 2019. (Time Magazine)
  - Malware evolvement
    - Malware is adapting the anti-malware tools.
      - Polymorphic malware, metamorphic malware, and other obfuscation technologies, increasing the difficulty of malware detection
    - More and more malware are equipped with metamorphic code.
      - E.g., Virlock ransomware

Background

- A visible sign of Virlock infection

Background

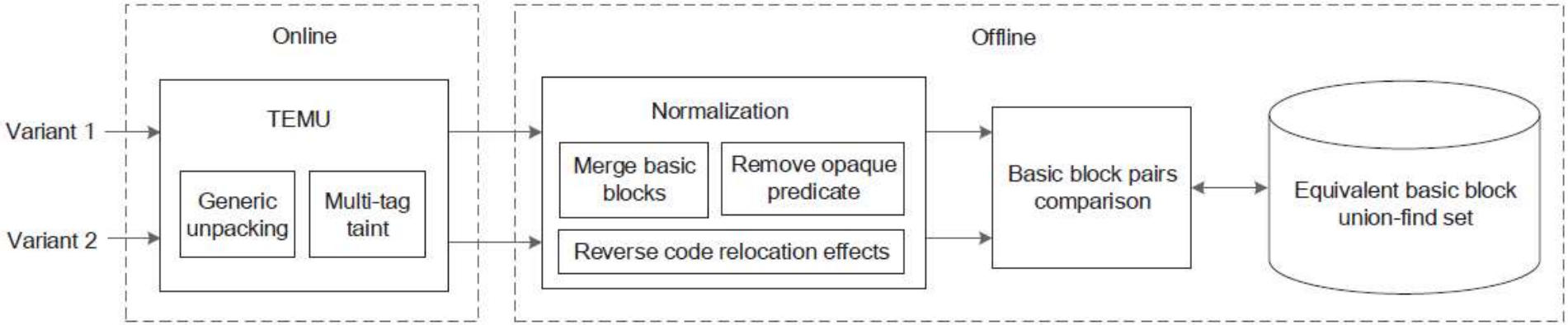
- Metamorphic malware
  - Metamorphic malware is a kind of malware which can mutate itself during propagation, so that each instance of the malware shows little resemblance to another.
  - So, it is difficult for the anti-malware tools to detect metamorphic malware.



Motivation

- We would like to :
  - Analyze metamorphic malware and understand its behavior.
  - Reveal the insight of metamorphic malware
    - Mutation mechanism
    - Mutation rules
    - How is it implemented?
    - And so on…
  - Help developing mutation insensitive anti-malware tools.

Architecture

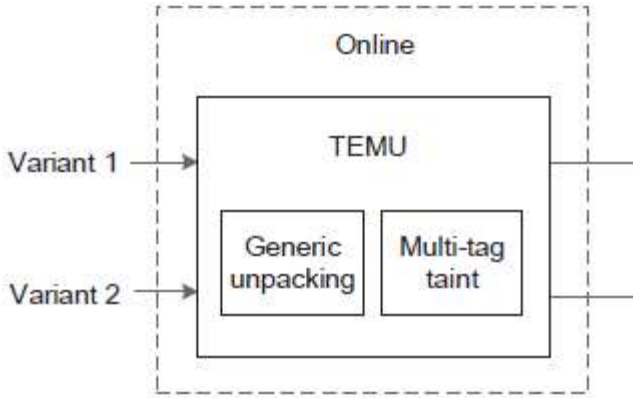


Architecture

- Two phases in MetaHunt
  - Online phase (dynamic analysis)
    - Runtime instruction trace logging
    - Input-related code with taint analysis
  - Offline phase (static analysis)
    - Basic block normalization
    - Basic block semantics comparison
    - Semantics caching

Design

- Online Phase
  - Trace logging
    - Record malware variant instructions executed during runtime.
    - Only malware behavior related basic blocks are logged, and malware packer and libraries are not recorded.
    - Input related code with taint analysis.



Design

- Offline phase
  - Normalization
  - Basic block comparison (semantics)
    - Abstract the semantics of a basic block by symbolic executing the basic block instructions.
    - Employ a theorem prover to calculate the equivalence of the formulas.
    - Decide similarity of two basic blocks.
  - Semantics memorization
    - Use union-find set to record the calculated basic block.
    - Each subset in the union-find set represents a cluster of equivalent basic blocks.

Implementation

- Implementation
  - BitBlaze
    - A binary analysis platform
    - Vine: a static analysis platform (offline stage)
  - Temu
    - An emulator, as a dynamic analysis platform
    - Used for execution trace logging (online stage)
    - Malware sandbox
  - Theorem Prover
    - A constraint solver calculating whether two formulas are equal or not.
  - Optimization
    - DiffMap, improved symbolic execution calculation

Evaluation

- Metamorphic malware samples
  - More than 1400 metamorphic variants
  - 9 mutation engines
  - A detailed study on MetaPHOR, a well-known metamorphic malware.

Evaluation

Table 1: Metamorphic engine statistics and various code mutation methods adopted.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Engine	Type	# Mutations	Reg. renaming	Dead code	Instr. reorder	Instr. substitut.	Opaque constant	call/return obfus.	Indirect jump	Opaque pred.	CFG flattening	Funct. inlining	Decryption	Conv. time (hrs)	# UF subsets	Max. subset size
Lexotan32	attached	100	✓	✓	✓	✓				✓			✓	1.5	90	8
MetaPHOR	attached	100	✓	✓	✓	✓				✓			✓	2.2	132	12
W32.Evol	attached	100	✓	✓	✓	✓				✓				1.0	52	6
NGVCK	decoupled	200	✓	✓	✓	✓	✓	✓	✓	✓				4.7	346	16
G2	decoupled	200	✓	✓	✓	✓								1.4	115	8
VCL32	decoupled	200	✓	✓	✓	✓								1.8	130	10
MPCGEN	decoupled	200	✓	✓	✓	✓								2.2	96	8
MalDiv	decoupled	150	✓	✓	✓	✓	✓	✓	✓	✓				6.8	522	34
Obfuscator-LLVM	decoupled	150	✓	✓	✓	✓	✓			✓	✓	✓		4.6	304	18

Evaluation (optimizaiton)

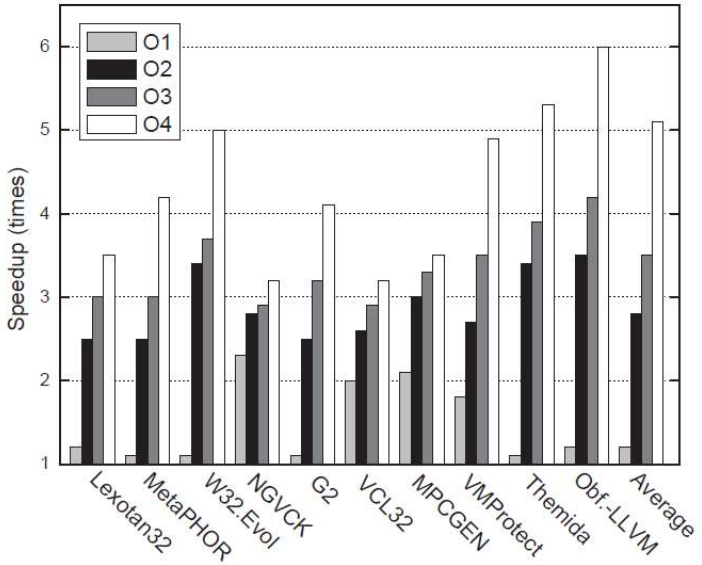


Figure 11: The impact of basic blocks fast matching when applied cumulatively: O1 (preprocessing), O2 (O1 + union-find set and DiffMap), O3 (O2 + concretizing symbolic formulas), O4 (O3 + QueryMap).

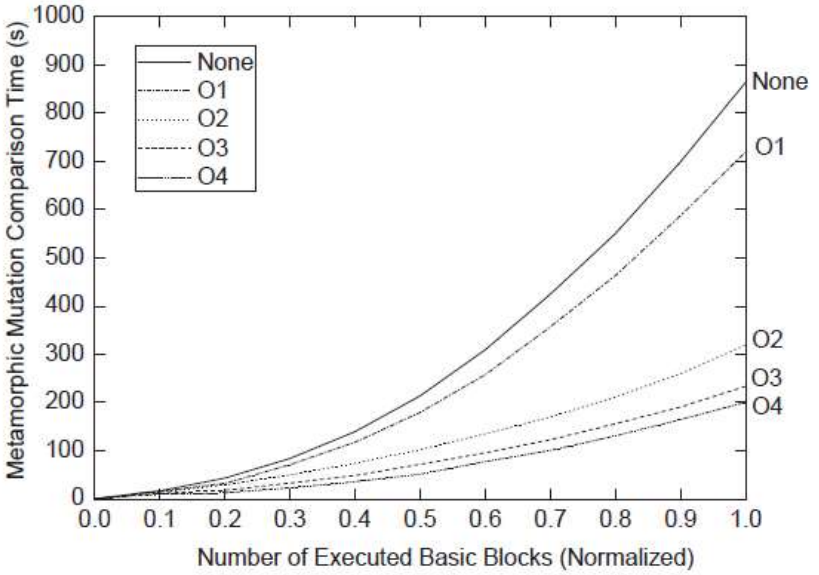


Figure 12: The effect of our optimizations over time on NGVCK family

Evaluation (with optimizaiton)

- We found:
  - The mutation capability of metamorphic malware is not unlimited.
    - Converge time (1.0 - 6.8 hrs)
    - Number of the subsets (52 - 522)
    - The maximum number of basic blocks (6 - 34)
  - Our optimization strategies greatly improve basic block matching process.

MetaPHOR Study

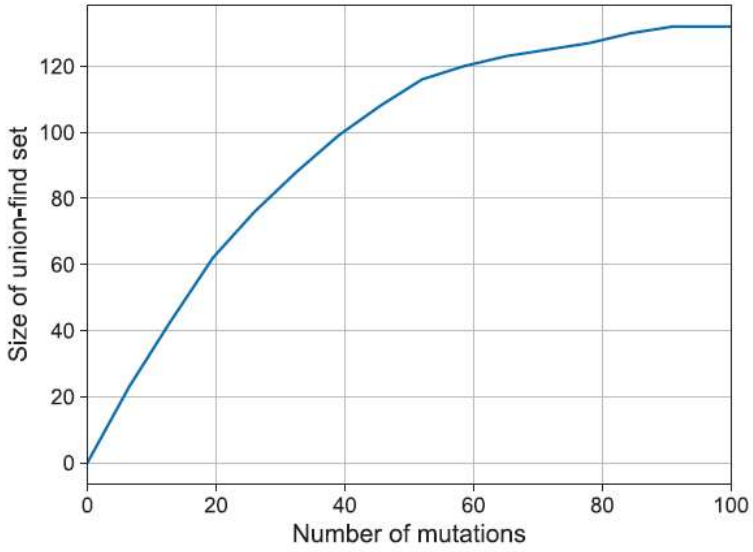


Figure 6: Size of union-find set of the variants generated by MetaPHOR.

Before	After
mov eax, 1 add eax, ecx	lea eax, [ecx+1]
push 3 pop eax	mov eax, 3
mov eax, ebx add eax, 8	lea eax, [ebx+8]
mov [eax], 3 push [eax]	push 3

mov [eax], ebx
add [eax], ecx
mov ebx, [eax]
	add ebx, ecx

mov [eax], 2
add [eax], ecx
mov ebx, [eax]
	add ecx, 2

or eax, 0	nop

Figure 9: Code compressing examples in MetaPHOR

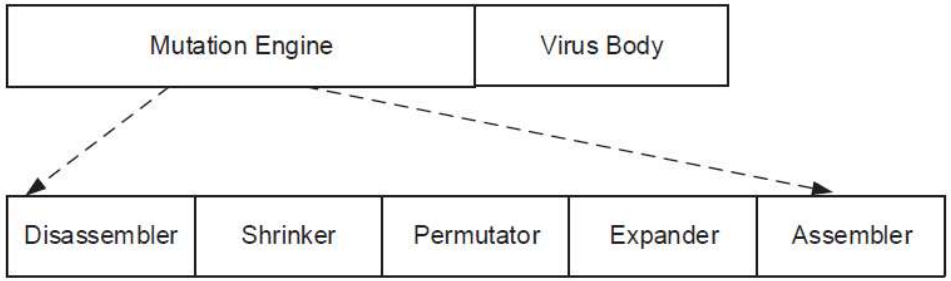


Figure 7: Structure of the MetaPHOR virus

MetaPHOR Study

- Results:
  - Mutation engine
    - Five parts: disassembler, shrinker, permutator, expender, and assembler (see Figure 7)
    - About 9000 lines of assembly code.
  - Mutation capability
    - The number of different MetaPHOR variants will be stable after 90 mutations in 6-7 hours.

Conclusion

- The mutation capability of metamorphic malware is not unlimited.
- The number of variants will eventually reach to a converge point.
- The anti-malware tools may refer to our work to improve the detection of emerging malware.
- The mutation engine is not perfect:
  - A bug in NGVCK mutation engine

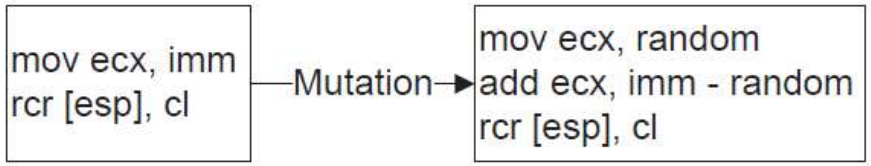


Figure 13: Example: buggy metamorphic engine implementation (add instruction may modify the value of carry flag).

Q&A