Large-Scale Malware Indexing Using Function-Call Graphs

ACM CCS 2009
Xin Hu, Tzi-cker Chiueh, Kang G. Shin

김영철

2016. 7. 8.

INTRODUCTION

- SMIT (Symantec Malware Indexing Tree)
  - Graph Edit Distance & B-tree Index 응용
  - graph-similarity search
  - 크기가 큰 function-call graph에도 적용 가능
  - 10만~100만 크기의 데이터베이스에 효과적인 query를 지원하는 것이 목표

RELATED WORK

- Byte-level signature, run-time behavior 이용
  - n-gram of byte codes
  - sequence of system-call event
- Indexing Technique + Graph 이용
  - GraphGrep, Tree+, TALE
- Approximate graph-matching + index graph
  - graph-edit distance, approximate edit distance, VPT …
  - 이 논문의 접근방법

FUNCTION-CALL GRAPH EXTRACTION

- 프로그램의 함수를 세 가지로 분류
  - Local functions – 악성코드 작성자에 의해 생성
  - Statically-linked library functions – Libc, MFC …
  - Dynamically-imported functions – dll 파일

FUNCTION-CALL GRAPH EXTRACTION

- Function-call graph 추출 과정
  - packing 되어있는지 확인 후, recursively unpack
  - IDA pro 를 이용하여 disassemble
  - 함수들을 symbolic name 으로 labeling
    - Dynamically-imported functions – IAT 파싱
    - Statically-linked library functions – FLIRT 사용
    - Local functions – symbolic name 이 없을 경우, ‘sub_xxxx’ 형태로 labeling

FUNCTION-CALL GRAPH EXTRACTION

- Local function 매칭을 위한 단계
  - sequence of call instructions
  - mnemonic or opcode sequence of instructions
  - SMIT 가 함수에서 위의 두 가지 정보를 추출한다.

GRAPH-SIMILARITY METRIC

- Graph Edit Distance
  - Vertex-edit operations
    relabel, insert, remove
  - Edge-edit operations
    insert, remove
  - SMIT 에서 각각의 edit operations 들에 대해 unit cost 를 할당

GRAPH-SIMILARITY METRIC

- Approximating Graph-Edit Distance Using Graph Matching
  - extended vertex $V^*_g = V_g \cup G_{g\_dummy}$
  - extended graph $g^* = (V^*_g, E_g, \mathcal{L}_g, L_g \cup \{G_{g\_dummy}\})$

  - algorithm
    1. C_E 는 맵핑된 edge의 수
    2. EdgeCost = $(|E_g| - C_E) \times c(RE) + (|E_h| - C_E) \times c(IE) = |E_g| + |E_h| - 2 \times C_E$
    3. NodeCost += 그래프 g의 노드가 h의 더미와 매칭될 때, +c(RV)
       그래프 h의 노드가 g의 더미와 매칭될 때, +c(IV)
    4. NodeCost += 일치하는 두 노드에 대해서 label 이 다를 때, +(R)
    5. Edit distance, $ed_{\varnothing(g,h)}$ = NodeCost + EdgeCost

GRAPH-SIMILARITY METRIC

- Exploiting Instruction-Level Information
  - mnemonic sequence, CRC value, symbolic name 이용
  - neighborhood-driven algorithm 이용

    $C = \{ v : v \in V_g \cap V_h \}$, v는 atomic function이라고 지칭

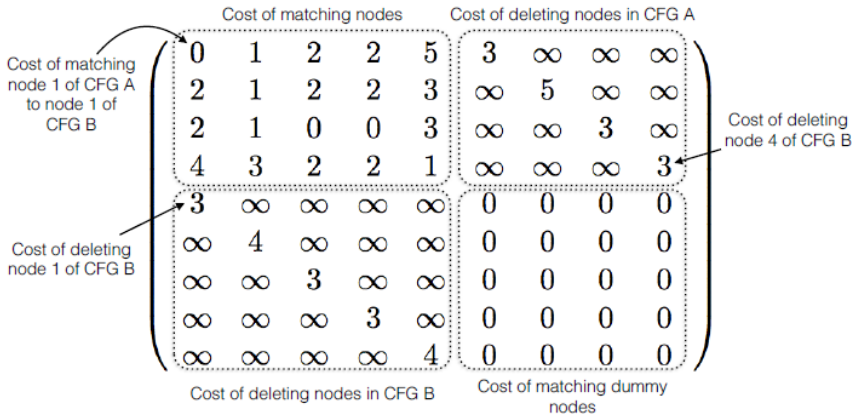
    $f1 \in V_g^r = V_g - C$, $f2 \in V_h^r = V_h - C$

    f1 = f2 라고 판정되면 집합 C에 추가

    동일한 함수가 나오지 않을 때까지 계속해서 과정 반복
  - 위의 알고리즘 결과로 남은 함수들에 대해 Hungarian algorithm 적용

GRAPH-SIMILARITY METRIC

- Bipartite Graph Matching
  - global knowledge 없이 cost matrix 를 만들기 때문에 정교함 하락
    - ➔ optimized Hungarian algorithm 개발
  - optimized Hungarian algorithm
    1. X = V_g^r+더미 Y = V_h^r+더미를 가지고 complete bipartite graph 생성
    2. 각각의 edge 는 x 와 y 의 추정된 맵핑 cost 로 할당



GRAPH-SIMILARITY METRIC

- Bipartite Graph Matching
  - 실제 두 노드의 매칭 비용은 relabeling cost 로 얻을 수 있음
  - 더 나은 방법으로 true edit cost 를 찾기 위한 알고리즘 개발

    ➔ Relabeling Cost 와 Niehgborhood Cost 계산

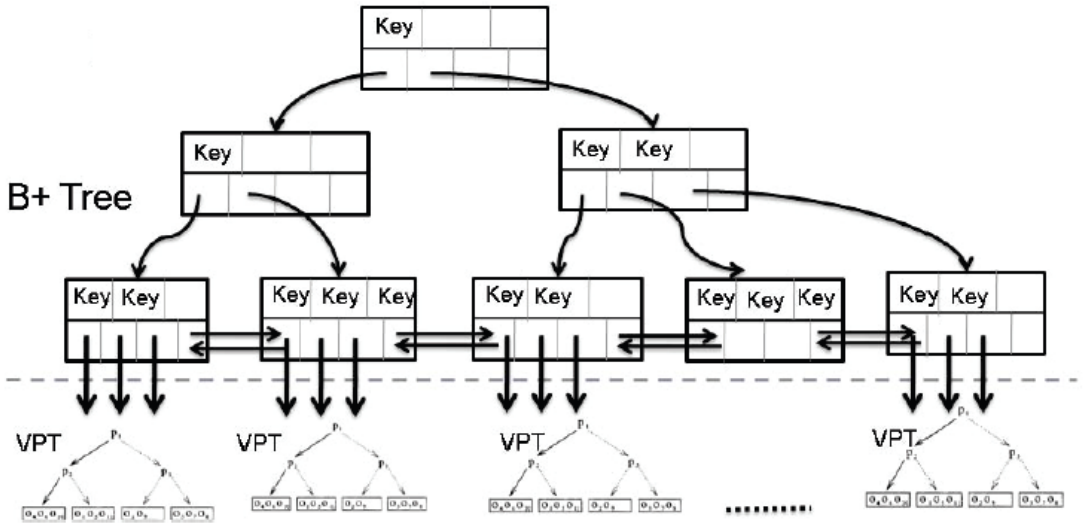
    1. Relabeling Cost : node i 와 node j 가 같지 않으면, $C_{i,j}$ = c(R)
    2. Outgoing Neighborhood Cost : $|N^g_{Out}(i)| + |N^h_{Out}(j)| - 2 \times |N^g_{Out}(i) \cap N^h_{Out}(j)|$
    3. Incoming Neighborhood Cost : Outgoing Neighborhood Cost 와 유사

GRAPH-SIMILARITY METRIC

- Neighbor-Biased Hungarian Algorithm
  - 이미 매칭된 노드들의 이웃들을 고려한 방법
  - 먼저 매칭된 함수들의 set을 찾고, 각각의 함수들의 unmatched neighbor의 매칭 비용을 감소시킴
  - update 된 cost를 가지고 나머지 그래프들에 대해 Hungarian algorithm 적용

MULTI-RESOLUTION INDEXTING

- nearest-neighbor search 지원
  - ➔ optimistic Vantage Point Tree 사용
- minimize the number of computations
  - ➔ two-level indexing scheme 사용
    level 1 – B^+-tree index
    level 2 – VPT tree index



MULTI-RESOLUTION INDEXTING

- B^+-tree Index Based on Malware Features
  - feature vector v = (N_i, N_f, N_x, N_m)
  - backward sibling pointer, forward sibling pointer
  - malware query가 주어지면 feature vector추출, key로 사용