

## **Research in Industrial Projects for Students (RIPS) 2008 Symantec Project Description: A Graph-Matching Problem**

Industrial Partner: Symantec Research Labs

Industrial Advisers: Dr. Tzi-cker Chiueh & Darren Shou {tzi-cker\_chiueh, darren\_shou}@symantec.com

Faculty Mentor: Sudhir Kumar Singh (suds@ee.ucla.edu)

### **1 Background**

How can we know if a new malware sample is a variant of an existing malware? The influx of malicious code variants puts a large strain on analysis:

Symantec gathers malicious code data from over 120 million desktops that have deployed Symantec antivirus products in consumer and corporate environments. In the last six months of 2007, Symantec detected 499,811 new malicious code threats—this is a 136 percent increase over the previous period, when 212,101 new threats were detected, and a 571 percent increase over the last half of 2006. In total, there were 711,912 new threats detected in 2007, compared to 125,243 threats in 2006, an increase of 468 percent. This brings the overall number of malicious code threats identified by Symantec to 1,122,311, as of the end of 2007. This means that almost two thirds of all malicious code threats currently detected were created during 2007 (Symantec 2007).

Many of the influxes of variants are not due to original code, but mass-produced attempts to foil antivirus detection, a tactic made successful by the Storm Worm. The Storm Worm's creators used a variety of techniques, from rapidly changing variants to fast-flux hosting, to continue to spread variants of the program to victims' PCs. In addition, web-based infection kits that deliver a different variant to each victim's PC have also made analysis more difficult.

### **2 The Problem**

Many computer security and system management problems can be reduced to a graph-matching problem: Given a graph  $G$ , find the member in a graph database  $D$  that is most similar to  $G$  according to some distance metric. One example of this abstract problem is to check if a new malware sample is a variant of an existing malware. The other example is to check if a system configuration that exhibits an erroneous behavior is similar to any member in a diagnosis database that contains solutions to previously encountered problematic system configurations. Another example is to check if a program assignment is a near-duplicate copy of some pre-existing program. While one can solve this problem by sequentially comparing  $G$  against every member in  $D$ , building an index for  $D$  beforehand can significantly reduce the service time for such nearest-neighbor graph search queries. The goal of this project is to explore the design space of graph database indexing, and empirically compare them using real-world graphs.

### **3 Candidate Starting Points & Directions**

Usually, the techniques for graph database indexing work in a filter-and-verification framework involving three stages: –

(i) Identify the sub-structure of graph to be used as an index feature and then design a graph-index based on this feature. Often there is a tradeoff between pruning power (i.e. the structural information retained) and the size of the database, as well as, time complexity of building/updating the database. For example, an indexing based on subgraphs of up to a max size is more structurally informative (and consequently have more pruning power) than an indexing based on paths of up to a max size but the former will be generally more computationally expensive and less compact than the later.

(ii) Splitting the query graph into sub-structures of same type as the index feature

(iii) Matching: testing the remaining graphs for (subgraph) isomorphism against the query graph

The stage (i) is mostly offline, however the later two stages are to be processed in an online manner. As we know, the stage (iii) is computationally very expensive (Subgraph-Isomorphism is known to be NP-Complete), and a technique which performs very small number of isomorphism testing (i.e. it maintains a compact database and is very good at pruning) is desirable. Further, in stage (i), an index feature should be chosen to keep a nice balance between loss in structural information and computational complexity in building/updating the database.

There are some existing techniques in literature such as gIndex[1], GraphGrep[2], Tree+ $\delta$  [3], GCoding[4] etc and we could start with these resources as listed in the following.

#### Research Papers:

1. X. Yan, P. S. Yu, and J. Han, *Graph Indexing: A Frequent Structure-based Approach*, SIGMOD 2004.  
[http://www.cs.uiuc.edu/~hanj/pdf/sigmod04\\_gindex.pdf](http://www.cs.uiuc.edu/~hanj/pdf/sigmod04_gindex.pdf)
2. D. Shasha, J. T. L. Wang, and R. Giugno, *Algorithmics and Applications of Tree and Graph Searching*, PODS 2002.  
<http://cs.nyu.edu/cs/faculty/shasha/papers/pods2002.pdf>
3. P. Zhao, J. X. Yu, and P. S. Yu. , *Graph Indexing: Tree + Delta  $\geq$  Graph*, VLDB 2007  
<http://www.vldb2007.org/program/papers/research/p938-zhao.pdf>
4. L. Zou, L. Chen, J. X. Yu, Y. Lu, *A Novel Spectral Coding in a Large Graph Database*, EDBT 2008.  
<http://www.cse.ust.hk/~leichen/papers/edbt08-Spectralcoding.pdf>
5. H. Shang and X. Jin, *Indexing and Mining of Graph Database Based on Interconnected Subgraph*, IDEAL 2006.  
<http://www.springerlink.com/content/967371t277208284/>
6. S. Zhang, M. Hu, and J. Yang. *TreePi: A novel graph indexing method*. ICDE, 2007.  
<http://vorlon.case.edu/~jiong/research/papers/67.pdf>
7. H. He and A. K. Singh. *Closure-tree: An index structure for graph queries*. ICDE, 2006.  
[http://www.cs.ucsb.edu/%7Edbl/papers/he\\_icde\\_2006.pdf](http://www.cs.ucsb.edu/%7Edbl/papers/he_icde_2006.pdf)
8. M. Christorescu and S. Jha, *Static Analysis of Executables to detect Malicious Patterns*, 12<sup>th</sup> USENIX Security Symposium.  
[http://pages.cs.wisc.edu/~jha/jhapapers/security/usenix\\_2003.html](http://pages.cs.wisc.edu/~jha/jhapapers/security/usenix_2003.html)
9. D. Bruschi, L. Martignoni, and M. Monga, *Detecting Self-mutating Malware Using Control-Flow Graph Matching*, DIMVA 2006.

<http://homes.dico.unimi.it/~monga/lib/dimva06.pdf>

10. J. R. Ullmann. *An Algorithm for Subgraph Isomorphism*. J. ACM, 23(1), 1976.  
<http://portal.acm.org/citation.cfm?doid=321921.321925>

**Database of graphs:**

<http://amalfi.dis.unina.it/graph/>

<http://www.msri.org/about/computing/docs/magma/html/text1477.htm>

A Database of Graphs for Isomorphism and Sub-Graph Isomorphism Benchmarking:  
(<http://amalfi.dis.unina.it/graph/db/papers/database.pdf> )

#### **4 Research Tasks and Deliverables**

- Task 1: Explore the design space of graph database indexing
  - Task 2: Empirically compare them using real-world graphs
  - Task 3: Investigate novel and practically meaningful distance measures on graphs, develop associated indexing techniques and design algorithms for exact/inexact graph matching
  - Task 4: Prototype the most promising approach
- 
- Deliverable 1: A technical report detailing approaches taken, experimental results and conclusions
  - Deliverable 2: Presentation to Symantec Research Labs and separately to IPAM RIPS at Project's day
  - Deliverable 3: The prototype source/software

#### **5 References**

Internet Security Threat Report: September 2007. Symantec Corporation