IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 16, Issue 4, Ver. IV (Jul – Aug. 2014), PP 29-34
www.iosrjournals.org

Attack Graph to Graph Database

Saurav Saraff[1]
Department of Information Technology, Institute of Engineering & Management[1]

***Abstract*:** *Databases are an integral part of almost any computing system today, and users heavily rely on the services they provide. When we interact with a computing system, we expect that any data be stored for future use, that the data is able to be looked up quickly, and that we can perform complex queries against the data stored in the database. There are many different emerging database types available for use, such as relational databases, key-value databases, object databases, graph databases, and RDF databases. Each type of database provides a unique set of qualities that have applications in various domains. Our work aims to investigate and compare the performance of relational databases to graph databases in terms of handling Attack Graph data. In this following project work, we transform the Attack graph data (stored in the xml format) to a Graph Database format using Neo4j. This converted format can be viewed using the Neo4j service & local host in the web browser. The further work of the project has also been attached.*

I. Introduction

The relational database management system was first created in the 1970s. Since then, its popularity has skyrocketed, and it has become a primary data storage structure in both academic and commercial pursuits. Relational databases range from small, personal databases like Microsoft Access to large-scale database servers like Oracle, Microsoft SQL Server, MySQL and

PostgreSQL.

Research into graph databases was popular in the early 1990s, but died out for a series of reasons including the surge of hypertext and XML research. With the rise of the Internet as a tool for the general public, data began to increase both in volume and interconnectedness. The graphmodel was used to represent tremendous amounts of data more often than it had in the past.

Traditional data stores were often capable of handling graph data. Yet, they were often neither designed to do so nor efficient at it. There was a clear desire for a data store tailored to the needs of graph data.
Social Networks is a very popular topic not just in society,

Social networks, not only introduces a profound amount of data, but presents large graph data problems for the research community. For large social graphs we are most interested shortest path queries and clustering. These graph algorithms provide analysis of relations of two nodes and determination of communities or social networks. Currently social networking sites like Facebook do not use a graph databases. Instead they use key-value stores or column stores Cassandra. Chemical and Biological data are also modeled as a graph by assigning atoms as nodes and bonds the edges between them.

Attack Graph can also be represented in a graph database. It may be noted that the graph database code is more intuitive because it works with abstractions (nodes, relationships etc.) that fit the domain better. And the relational model is based on the assumption that database records are what primarily matters while relationships between records come second place. But the graph database model focused on the relationship as one of the most important feature and by using attack graph our main purpose is to find the attack path; we store the information in a graph database.

Some argue that most data is inherently a graph, and that all data can be stored as a graph. Using graphs to store data not only allows for a dynamic schema, but also provides representations of data not previously possible. The ability overlay different graphs (Ex. social, temporal, and special) on data extends the functionality of querying data.

II. Model For Attack Graph

In recent years there have been increased numbers of reported incidences of attacks against networks of critical sector enterprises. With the rapid advancement in the area of information and communication technology and its adoption in every aspect of human society, such incidences in future are more likely. Countermeasure activities require the ability to rapidly and correctly identify, rank and react to probes and attacks. This has been further complicated by the fact that organizational networks of many of the critical sector enterprises are now being interconnected for business reasons. Security analysis of such conglomeration of networks is challenging

and thus calls for efficient techniques.

Incidentally, many of these attacks when considered in isolation are very simple and easier to detect with available tools. But in most of the cases attackers combine those attacks to launch multistage attacks against critical assets which the security administrators might have thought of as secured enough. Conventional defense approaches has been mostly host centric, where attention is given to identifying vulnerabilities of the individual hosts and taking measures to mitigate them. But these methods are less effective in the face of multistage attacks.

Attack graph is a handy tool which helps in mapping isolated intrusion alerts to known multistage attack paths and thereby enabling prediction of future attack strategies of the attacker. Attack graphs once generated are used to perform wide array of analysis to discover information of high interests. Reachability analysis is one such technique

which helps in determining whether from a given vantage point on a host, an attacker can compromise a target host by executing a sequence of exploits against vulnerabilities on connected intermediate hosts. This query can also be used to find all possible starting positions, from where the attacker can compromise one or more critical resources or to find all possible paths from a particular starting location of the attacker.

Very often, enterprise networks live with vulnerabilities which cannot be patched or removed due to operational constraints. Tools commonly known as vulnerability scanners are available for discovering such vulnerabilities in individual hosts. But they cannot identify situations where combination of configurations on the same host or multiple hosts on the same network can contribute to the compromise of some critical hosts or the network as a whole. Attack graphs supplement these vulnerability scanners with the missing information about relationships among vulnerabilities.

There have been number of formalisms for attack graph representation, reported in literature. We have used the most popular one which is described briefly in the subsequent sections.

Consider the following example (taken from [35]) of very simple organizational network configuration as shown in Figure 5.1. The firewall protects hosts in the internal network i.e., Host 1 and Host 2 from any external host Host 0. Host 1 is a file server which runs ftp, rsh and ssh services and Host 2 is a database server which runs ftp and rsh services. The firewall allows ftp, rsh or ssh connections through it and blocks all other.

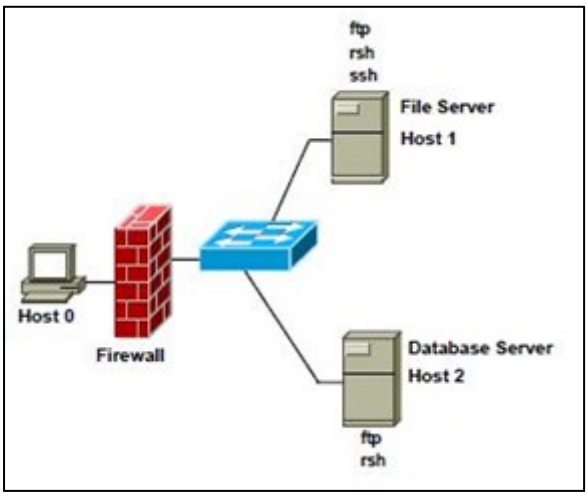


Figure.1: A Simple Network Configuration [36]

We assume that hosts have unique symbolic identifiers within a network.

Formally, an attack graph is represented as a directed graph G = (V;E), where each node and edge has labels. Nodes can be of two types. One type of nodes represents either some attacker capability or some network conditions. Another type of nodes represents exploits. Directed edges from first type of nodes to second type represent prerequisites or preconditions of exploits. Directed edges from second type of nodes to first type represent effect or post conditions of executing the exploit. This representation is based on the monotonicity assumption, which states that an attacker never relinquishes any obtained capabilities.Attacker's capabilities are described by the syntax capability name(host id) meaning that attacker has a capability name on a specific host host id. Name of some attacker's capabilities are user, superuser etc. Capability description user(1) means that the attacker can execute arbitrary program as a normal user on Host 1. Capability description superuser(1) means that the attacker can execute arbitrary program as superuser on Host 1.

Network conditions are described by the syntax service name(source host id; destination host id) or relation name(source host id; destination host id) meaning that any service service name running on destination host id can be accessed from source host id or an unidirectional relation relation name exists from source host id

to destination host id. Network condition ftp(1; 2) means that the ftp service running on Host 2 is accessible from Host 1. Network condition trust(1; 2) means that a trust relationship exists from Host 1 to Host 2 allowing any user on Host 2 to remotely execute shell commands on Host 1 without providing any password.

The other type of nodes represents exploits. For executing an exploit an attacker may require multiple capabilities or network conditions known as preconditions of the exploit. Successful execution of an exploit may create new attacker capabilities or new network conditions known as post conditions of the exploit. Exploits are described by the syntax exploit name (source host id; destination host id) meaning that an attacker having some capability on host source host id with the help of some existing network conditions, can perform the exploit exploit name, exploiting some vulnerability on host destination host id thereby gaining new capabilities or establishing new network conditions. Exploits can also be described by the syntax exploit name(host id) meaning that the source and destination hosts are the same, i.e. exploit is performed locally.

Figure 5.2: Attack Graph of Simple Network Configuration shows the attack graph representation corresponding to the simple network configuration of Figure 5.1. This is a partial attack graph with respect to the critical asset ”database server” on Host 2. It shows all possible attack paths leading to its compromise i.e. attacker gaining superuser privilege on Host 2. A full or complete attack graph on the other hand shows all possible attack paths between arbitrary host pairs. Only the partial attack graph is shown to have clarity in presentation and understanding [36].Initial network conditions are represented by nodes ftp(0; 1), ftp(0; 2), ftp(1; 2), ssh(0; 1), rsh(0; 1), rsh(0; 2), rsh(1; 2) etc. Attack can be launched from any host and we assume that the attacker must have at least a user capability on that host.

One of the attack path starts with sshd bof(0; 1). This describes a buffer overflow exploit executed from Host 0 against Host 1 (the file server), i.e., against its secure shell service. The preconditions of this exploit are user(0) and ssh(0; 1), meaning that in order to perform the exploit the attacker must be a normal user on Host 0 and be able to access ssh service running on Host 1. The result of the sshd bof(0; 1) exploit is that the attacker can execute arbitrary code on the file server indicated by the node user(1). After this, the attacker can execute the ftp rhosts(1; 2) exploit, where the attacker exploits a particular ftp vulnerability to anonymously upload a list of trusted hosts from Host 1 (the file server) to Host 2 (the database server). The attacker can use the newlyestablished trust to remotely execute shell commands on the database server on Host 2, without providing a password, i.e., the rsh(1; 2) exploit. This exploit gives the attacker control over the database server as a normal user. The attacker then executes a local buffer overflow exploit enabling her to execute code on the database server with superuser privileges

Figure 2: Attack Graph of simple Network Configuration [36]

III. Model Of Graph Database

In graph database every record is stored in an individual node, and it can store any no of properties, so there is no schema specified.

But for PostgreSQL we setup a database which has two tables “Nodes” which store the data of nodes from the Attack Graph and “Edges” which store the data of the edges from the attack Graph. The schema of these two tables is given below:

For ‘Nodes’ we have 3 attributes: ID, Integer, Primary Key

LABEL, Character(100)

TYPE, Character(10)

For ‘Edges’ we have four attributes: ID, Integer, Primary Key

SOURCE, Integer, References Nodes(ID)
DESTINATION, Integer, References Nodes(ID)

TYPE, Character(10)

A. Parsing Attack Graph using SAX Parser

To compare the performance of these two databases on graph queries, we have to parse the Attack Graph xml document and store the information about nodes and edges in these databases.

To parse the xml document we use the SAX parser. A sample code segment is given below to understand the mechanism of SAX parser.

The import statements for the classes the application will use are the following

import org.xml.sax.Attributes; import org.xml.sax.SAXException;

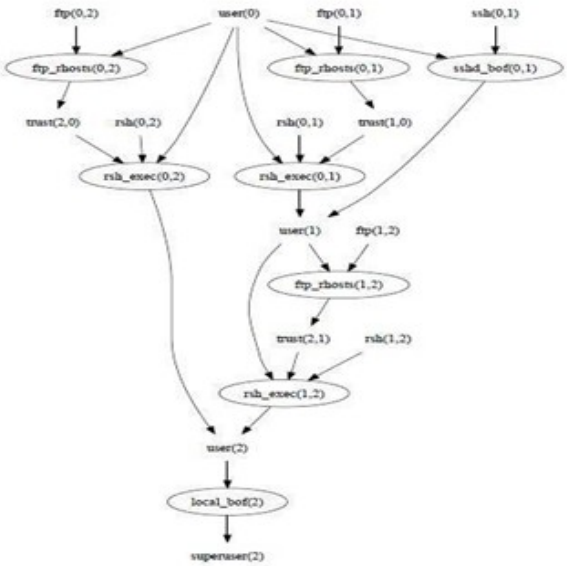
import org.xml.sax.helpers.DefaultHandler;

The most important interface in XmlParsing is ContentHandler. This interface requires a number of methods that the SAX parser invokes in response to various parsing events. The major event-handling methods are: startDocument, endDocument, startElement, and endElement. The easiest way to implement this interface is to extend the DefaultHandler class, defined in the org.xml.sax.helpers package. That class provides do-nothing methods for all the ContentHandlerevents.

B. Creation of nodes and edges in Neo4j

To perform a creation, update, query or any form of transaction we have to start, commit and finish the transaction properly as in the sample code given below.

To create a node with the given properties we use the Java Native API for Neo4j. To create a node and set the property of the node we use the code segment given below. We also created an index for the node prope



try 'Id' earlier. Here when we create a new node we have to add the property in index every time

To create a node with the given properties we use Cypher Query Language for Neo4j. To perform any execution we have to create an execution engine first. It is not necessary to create an execution engine every time. And most important part is that when we pass the parameters to the query we have pass them through a map.

IV. Observation

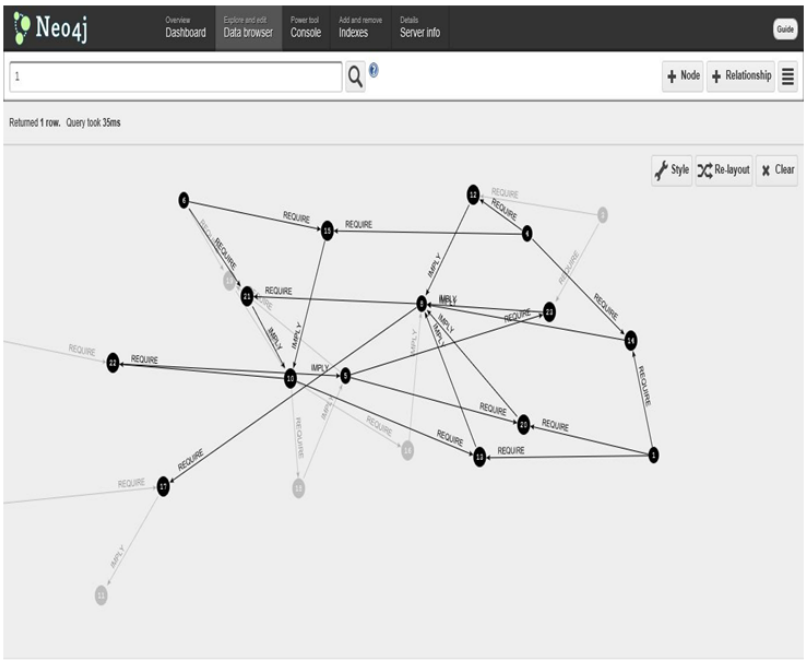
From the above model it's clear that graph database is much faster, as compare to other database tools. Schema-less and Efficient storage of Semi Structured Information. The graph data model allows storage of all the connections of the node along with the node, so that there is no additional step in computing connected data apart from reading the node into memory. The graph data model also prescribes that relationships have properties. Property rich relationships are absolutely critical when trying to explore connected data. The same constructs are achieved in relational databases using foreign keys, joins and join tables (Not going in depth into this since it is pretty standard info). These relational constructs end up as a part of the domain model, but not of the physical storage model (unless heavily optimized).

It is therefore quite obvious that the graph data model allows you to find interconnected data much faster and in a much more scalable manner as compared to the relational data model (think cascading reads of node(row) addresses vs. cascading joins on foreign keys). Most optimization techniques such as indexing can be applied equally well to both kinds of databases. This is the reason why the graph data model should perform much better than the relational data model in information retrieval - type recommendation problems.

Of course, when we talk about graph databases from different vendors, the requirements (CPU, RAM,

storage), capabilities, reliability and product maturity vary greatly. Given the maturity of the graph databases on the market currentlnot think it is a good idea to put production critical data in there, even if it is relevant for making recommendations (a relational data store is still the best for that). I would use a graph database as an efficient quasi-online data warehouse for highly interconnected data, a role in which it should surely The following list outlines the different types of graphics published in IEEE journals. They are categorized based on their construction, and use of color / shades of gray: outperform a relational database Of course, when we talk about graph databases from different vendors, the requirements (CPU, RAM, storage), capabilities, reliability and product maturity vary greatly. Given the maturity of the graph databases on the market currentlnot think it is a good idea to put production critical data in there

V. Conclusion



The very popular XML format has drawbacks of Signature issue,readability and other related and other related problems. The Attack Graphs forms a very important part of security. Thus the attack Graph data stored in ".xml" format need to be converted into a more useful form of database. Graph database, new technology, helps in removing the pitfalls of XML. In our given project, we convert the Attack Graph (in XML format) to the very useful format of graph database. The data stored in this new format can be used for further processing. Moreover the traversal and searching of nodes(data) in Graph database is much easier and requires much lesser time as compare to other database formats. This new Graph database technology becoming increasing popular today. Social networking sites like Facebook, the search engine giant Google is also implementing graph database for data maintenance.

References

Basic format for books:

[1]. R. A. a. C. Gutierrez., "Survey of graph database models.," 2008.
[2]. Wikipedia,"Relational database," July 2012.[Online].Available: http://en.wikipedia.org/wiki/Relational_database. [Accessed July 2012].
[3]. "PostgreSQL," [Online]. Available: http://en.wikipedia.org/wiki/PostgreSQL. [Accessed 2013].
[4]. Neo4j, "Neo4j," 2012. [Online]. Available: http://neo4j.org/learn. [Accessed 15 July 2012].
[5]. J. D. S. G. W. C. H. D. A. F. Chang, "Bigtable: A distributed storage system for Structured System," ACM Trans. Comput. Syst., p. 26(2):1–26, 2008.
[6]. G. DeCandia, D. Hastorun, M. Jampani,G. Kakulapati, A. Lakshman, A. Pilchin,S. Sivasubramanian, P. Vosshall, and W. Vogels., "Dynamo: amazon's highly
[7]. available key-value store.," SIGOPS Oper. Syst. Rev., 41(6):205–220, 2007. [7] A. Lakshman, "Cassandra - a structured storage system on a p2p network.,"
[8]. //www.facebook.com/note.php?note_id=24413138919,, 2008.
[9]. "Project Voldemort. Project voldemort: A distributed database. http://project- voldemort.com/,," 2009.
[10]. H. W. a. Y. X. Bin Shao, "Managing and mining large graphs: systems and implementations.," in Proceedings of the 2012 ACM SIGMOD International Conference and management, New York, NY, USA, 2012.
[11]. "Allegrograph.," [Online]. Available: http://www.franz.com/agraph/allegrograph.
[12]. "Dex.," [Online]. Available: http://www.sparsity-technologies.com/dex. [12] "Hypergraphdb.," [Online]. Available:

http://www.hypergraphdb.org. [13] "Sones.," [Online]. Available:
[13]. http://www.dekorte.com/projects/opensource/vertexdb/. [14] "Horton.," [Online]. Available: 5 http://research.microsoft.com/en-us/projects/ldg/.
[14]. InfiniteGraph," [Online]. Available: http://infinitegraph.com/.. [16] "Vertexdb.," [Online]. Available: http://www.dekorte.com/projects/opensource/vertexdb/.
[15]. "Cloudgraph," [Online]. Available: http://www.cloudgraph.com/.
[16]. "redis graph," [Online]. Available: https://github.com/tblobaum/redis-graph. [19] "Trinity.," [Online]. Available: http://research.microsoft.com/en-us/projects/trinity/..
[17]. "Orientdb.," [Online]. Available: http://www.orientdb.org/..
[18]. J. M. a. A. Deshpande., "Managing large dynamic graphs efficiently.," in ACM SIGMOD International Conference on Management of Data, SIGMOD'12, New York,USA, 2012.
[19]. "Filament," [Online]. Available: http://filament.sourceforge.net. [23] "G-store.," [Online]. Available: http://g-store.sourceforge.net/..
[20]. M. H. A. J. B. J. C. D. I. H. L. a. G. C. Grzegorz Malewicz, "Pregel:a system for large-scale graph processing.," in 2010 ACM SIGMOD International Conference on Management of data, SIGMOID'10, New York,USA.
[21]. "Phoebus.," [Online]. Available: https://github.com/xslogic/phoebus. [26] "Giraph," [Online]. Available: https://github.com/apache/giraph.
[22]. Wikipedia, "AllegroGraph," [Online]. Available: http://en.wikipedia.org/wiki/AllegroGraph.
[23]. "ArangoDB,"[Online].Available:http://www.arangodb.org/manuals/current/FirstStepsArangoDB.html.
[24]. "DEX (Graph database)," [Online]. Available: http://en.wikipedia.org/wiki/DEX_%
[25]. 28Graph_database%29. [30] "Oracle Spatial and Graph," [Online]. Available: http://www.oracle.com/technetwork/database-options/spatialandgraph/overview/index.html.
[26]. "Titan," [Online]. Available: https://github.com/thinkaurelius/titan/wiki.
[27]. M. D. Marzi, "Cypher Query Language," in Chicago Graph Database Meet-Up, Chicago, 2012.
[28]. "Intro to Neo4j Cypher Query Language," 2012. [Online]. Available: http://nosql.mypopescu.com/post/19950124685/intro-to-neo4j-cypher-query- language.
[29]. [34] "Linux Installation," [Online]. Available: http://www.neo4j.org/download/linux. [Accessed May 2013].
[30]. L. W. A. S. a. S. J. S. Noel, " Measuring security risk of networks using attack graphs," IJNGC, 2010.
[31]. C. M. Mridul Sankar Barik, "A Novel Approach to Collaborative Security using Attack Graph".
[32]. "Parsing an XML File Using SAX," Oracle, [Online]. Available: http://docs.oracle.com/javase/tutorial/jaxp/sax/parsing.html. Neo4j, "Screenshots," 2012. [Online]. Available:http://neo4j.org/scratchpad/doc/screenshots/. [Accessed 20 July 2012].
[33]. A. a. B. S. a. S. S. a. W. G. Gubichev, "Fast and accurate estimation of shortest paths in large graphs," Proceedings of the 19th ACM international conference on Information and knowledge management, pp. 499--508, 2010.
[34]. C. a. M. M. a. Z. Z. a. N. X. a. C. Y. a. W. D. Vicknair, "A comparison of a graph database and a relational database: a data provenance perspective," ACM, p. 42, 2010. 6