Tao *et al. EURASIP Journal on Wireless Communications and Networking* (2018) 2018:294
https://doi.org/10.1186/s13638-018-1309-9

EURASIP Journal on Wireless Communications and Networking

RESEARCH Open Access

Graph database-based network security situation awareness data storage method

Xiaoling Tao[1,2,3], Yang Liu[1†], Feng Zhao[1*†], Changsong Yang[1,2] and Yong Wang[1]

Abstract

With the rapid development of the Internet, network security situation awareness has attracted tremendous attention. In large-scale complex networks, network security situation awareness data presents the characteristics of large-scale, multi-source, and heterogeneous. Recently, much research work have been done on network security situation awareness. However, most of the existing methods store different types of data in different ways, which makes data query and analysis inefficient. To solve this problem, we propose a graph database-based hierarchical multi-domain network security situation awareness data storage method. In our scheme, we build a hierarchical multi-domain network security situation awareness model to divide the network into different domains, which can collect and dispose the awareness data more efficiently. Meanwhile, to unify our storage mode, we also define network security situation awareness data storage rules and methods based on graph database. Finally, extensive experiments on real datasets show that our proposed method is efficient compared to state-of-the-art storage models.

Keywords: NSSA, Data storage, Hierarchical multi-domain, Graph database

1 Introduction

With the advancement of network technology and the expansion of network scale, the network security risks are increasingly prevalent, such as network attacks, network vulnerabilities, data and privacy security [1, 2], and so on. To assess network security threats and predict the future status, network security situation awareness (NSSA), as a technology of active large-scale network security monitoring, has attracted tremendous attention and has become a hot research topic.

NSSA has been extensively studied in the past decades. In 1995, Endsley [3] proposed a three-level theoretical situation awareness model. In 2011, Zhang et al. [4] presented a novel multi-heterogeneous sensor-based network security situation assessment model. In their model, they utilized D-S evidence theory to fuse security data, which is submitted from multi-sensors. To facilitate the information security risk management process, Webb et al. [5] adapted Endsleys situation awareness model and designed a novel situation aware ISRM (SA-ISRM) process model. They addressed the problem of the poor decision-making and inadequate or inappropriate security strategies by an enterprise-wide collection, analysis, and reporting of risk-related information. Although they can analyze the impact of security incidents on a network system and accurately evaluate system security, these schemes cannot uniformly describe and synthetically analyze the data.

Besides, most of the existing NSSA schemes use relational databases to store awareness data. Chen et al. [6] proposed a cloud computing-based network monitoring and threat detection system, which utilized Hadoop MapReduce and Spark to process the data, and the results will be restored in the MySQL database server. The detector detected the malicious behavior of the data in the MySQL database and returned the detection results to the MySQL database. Masduki et al. [7] designed an intrusion detection-based NSSA system, which utilized intrusion detection software Snort and Bro to collect and analyze malicious traffic, and the data should be stored in the PostgreSQL database. However, these awareness schemes used relational database which cannot satisfy the storage requirement for heterogeneously multi-source data.

*Correspondence: zhaofeng@guet.edu.cn
†Feng Zhao and Yang Liu contributed equally to this work.
[1]Guangxi Colleges and Universities Key Laboratory of Cloud Computing and Complex Systems, Guilin University of Electronic Technology, Guilin, China
Full list of author information is available at the end of the article

In NSSA, network topology has similar nodes and relationship types in a graph databases. Personnel management is similar to social network. The network attack graph uses the node and the directed edge to display the attack path [8]. These network security-related application scenarios are in compliance with the characteristics that the graph database uses the concept of nodes and relationships to describe the data. Therefore, a graph database can be used in the NSSA. However, to the best of our knowledge, it seems that there is no NSSA model based on graph database model. Therefore, we propose a graph database-based NSSA data storage method for this purpose.

1.1 Our contributions

In this paper, we propose a graph database-based hierarchical multi-domain NSSA data storage method. In our method, we can comprehensively obtain network security data and store them in a graph database. By using Neo4j graph database, our method can truly reflect the network security situation and improve the efficiency of data query and the visualization of query results. The main contributions of this paper are as follows:

- We propose a hierarchical multi-domain NSSA model. We divide the network into many fields to collect network security situation data. Compared with the traditional hierarchical models, our model takes into account dependency and user security, which can reflect the network security situation more comprehensively.
- We define storage rules and methods for graph database-based NSSA data. Our storage method can effectively solve the problem of differences for data storage paths by storing the NSSA data into a graph database Neo4j. By using the Neo4j, we can efficiently query and analyze the data, and the query results can be visualized directly.

1.2 Related work

Because of its high efficiency, flexibility, and scalability, graph databases have been widely used in various fields, such as social network, recommendation system, power system, and so on.

In 2016, Constantinov et al. [9] proposed a real-time recommender engine. In their scheme, they applied a graph database to store and process social network information, and they use Neo4j to implement social network real-time recommendation system. Recently, Gu et al. [10] proposed a large-scale social network-based parallel layout algorithm. In their algorithm, they introduced the Neo4j graph database to the proposed parallel computing framework, which based on the Spark. They stored the data as the nodes and the relationships between the nodes in Neo4j, which can adequately utilize the advantages of the social networks.

In 2015, Zarrinkalam et al. [11] presented a file recommender system. Their scheme used the Neo4j graph database to store the background data to improve the efficiency. Patel and Dharwa [12] proposed a graph database-based integrated hybrid recommendation model in 2016. They used the Neo4j graph database to store item data, user preferences, and knowledge graph. Besides, the graph database is gradually applied in power network. To store and dispose the power system data, Raikumar and Khaparde [13] designed a common information mode-oriented graph database (CIMGDB) in 2017. In the proposed CIMGDB system, they used Neo4j graph database, which can improve the efficiency for power system data of any scale. Besides, Kan et al. [14] used Neo4j graph database to build a network model for a power grid. The Neo4j in this model is implemented based on the shortest path search function. And they showed that using Neo4j for energy network analysis is better than using PostgreSQL by experiments.

At the same time, graph databases are gradually applied to the network security field. In 2014, Barik and Mazumdar [15] proposed a graph data model for input information storage. In their model, they utilized popular Neo4j graph database to store the graph information instead of relational database. Besides, they used graph queries to generate attack graph and performed the typical analysis tasks over the generated attack graph. In 2015, Noel et al. [16] proposed a method for modeling, analyzing, and visualizing attack graphs. Their method can associate attack paths with security events. The analyzed attack graphs are stored in Neo4j, then they queried and analyzed the attack relationships, and visualized queries results.

In 2016, Barik et al. [17] used a constrained graph model to analyze network vulnerability, and they proposed an extended attribute graph model-based graph constraint specification language, which is used to analyze the attack graph-based network vulnerability. By implementing the attribute graph model in Neo4j, they verified that using these constraints can guarantee the accuracy of the attack graph generation and analysis process. Ashwin et al. [18] proposed an efficient and secure information retrieval framework for content centric networks. Their scheme used the Neo4j graph database to replace the content storage in the current CCNx implementation. They use Neo4j to improve the efficiency of storing and processing large-scale data since Neo4j does not use connection operations.

1.3 Organization

We organize the rest of this paper as follows: In Section 2, we give some preliminaries. Then, the proposed method is introduced in Section 3 in detail, including the system model, data storage rules based on a graph database, and the hierarchical multi-domain NSSA data storage method. The experiment results are shown in Sections 4 and 5, including query analysis experiment and query efficiency comparison experiment. Finally, we give a brief conclusion of our method in Section 6.

2 Preliminaries

In this section, we introduced some preliminaries. Firstly, we described the graph database, then, we introduced the specific instantiation of graph database–Neo4j. Finally, we gave a short description of the traversal mode of Neo4j.

2.1 Graph database

Graph database is a type of NoSQL database, whose data storage structure and query method are based on the graph theory [19]. There are three basic elements in a graph database, i.e., nodes, relationships, and properties, respectively. Specifically, nodes are abstract representations of entities or objects, and they are connected by relationships. Nodes and relationships both can have one or more properties [20]. Graph database supports create, read, update, and delete (CRUD) operations, with transaction integrity and operation availability [21].

There are three common kinds of graph data models, which are property graphs, hypergraphs, and triples. Among them, the property graphs model can be understood intuitively and easily, which can describe most of the graph usage scenarios, and it is the most popular graph data model, for example, Neo4j uses this property graph model. The property graph model contains nodes and relationships. Each node can have properties and one or more labels, while relationships have names and directions, and it always contains a start node and an end node [19].

2.2 Neo4j graph database

Neo4j [22] is an open source and high-performance graph database, which utilizes graph-related concepts to describe the data model. Neo4j has four basic data elements: node, relationship, property, and label. Neo4j can be deployed in the enterprise with the advantages of high-availability, fault-tolerant, and scalable clusters [23], and it can store hundreds of trillion entities. Neo4j can support the operations of storage, query, backup, and redundancy for large-scale data, and it also has the properties of atomicity, consistency, isolation, and durability (ACID) [24]. At the same time, it also supports the query language Cypher, which is an expressive and efficient declarative graph database query language [25]. Besides, Cypher is so scalable that the users can customize their own query methods conveniently.

2.3 Neo4j traversal mode

Traversal is the operation of moving and accessing a set of nodes by following the relationship in a graph database [26]. In Neo4j, each node record contains two pointers, one points to the first attribute of the node, and the other one points to the first contact in the link chain. With fixed-size storage records and pointer IDs, traversal and high-speed execution can be easily performed by following the pointers. To traverse a specific relationship from a node to another, we only need to traverse several pointers in Neo4j and then perform some low-cost ID computation, which is considerably less time-consuming than that of the global index.

In our method, we define the query depth in Neo4j as follows: when we query from the starting node A to node B, we define the query depth as one; further, if we query from node B to another node C, that is, we query from A, through node B, to node C, we define the query depth as two, and so on. The query methods are divided into single query and traversal query, both of them do multi-query depth query. Although the middle query process of the single query may *contain* multiple results, it finally returns one query result only. Compared with the single query, *traversal query will return all results of every depth, and it will only return the number of the nodes as the result of the query since the amount of the data is large.* Taking Neo4j query as an example, single query and traversal query respectively perform five kinds of queries whose query depths are five. The final result of a single query is only one node; however, the middle result may have multiple nodes. Specially, count query is the query whose query result is the number of nodes, and the traversal query is the query which returns a specific number of nodes.

3 Hierarchical multi-domain NSSA data storage method

In this section, we presented a hierarchical multi-domain NSSA data storage method. In the following, we introduced a hierarchical multi-domain NSSA model firstly, then we defined the graph database-based data storage rules, and finally, we gave the details of the NSSA data storage method.

3.1 Hierarchical multi-domain NSSA model

Our proposed hierarchical multi-domain NSSA method contains three layers: basic security layer, security analysis layer, and threat intelligence layer. Each layer contains one or more corresponding domains. We analyze the adjacency among different domains and describe the intersection among adjacent domains as an adjacent layer to show the connectivity and their communication. The architecture of the system model is shown in Fig. 1.

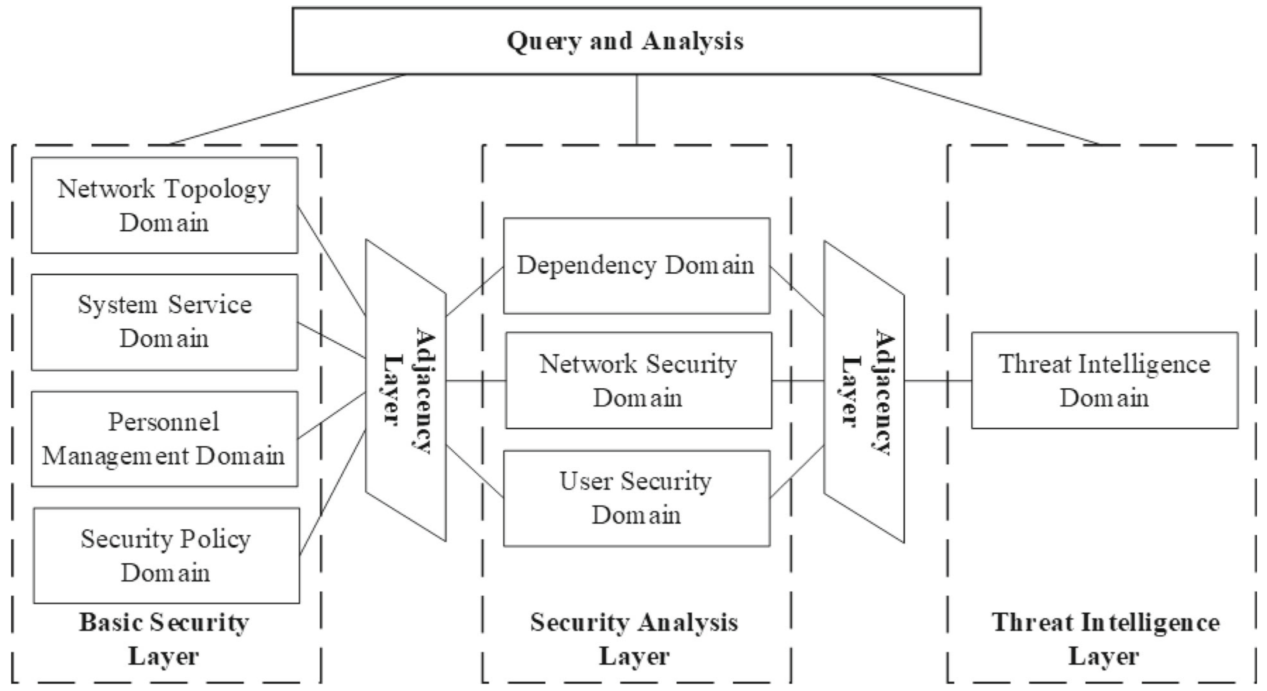


Fig. 1 Hierarchical multi-domain NSSA model. This figure shows the architecture of the system model, which includes three layers: the basic security layer, the security analysis layer, and the threat intelligence layer

- *The basic security layer.* The basic security layer contains a topological domain, a system service domain, a security policy domain, and a personnel information domain. This layer is used to describe the basic network environment and the basic safety information.
- *The security analysis layer.* The security analysis layer contains a dependency domain, a network security domain, and a user security domain. This security analysis layer is used to record and analyze all kinds of security problems and security incidents.
- *The threat intelligence layer.* The threat intelligence layer contains threat intelligence domains which based on the STIX. The threat intelligence layer is used to correlate the contents of the security analysis layer and restore the attack portrait.

3.2 Graph database-based data storage rules

Neo4j contains four basic data structures: node, label, relationship, and property. The node is usually used to store entity information. Each node can have multiple labels, which are used for indexing and some limited model constraints. The relationship connects the different nodes, and there can be multiple relationships between two nodes for different directions. Nodes and relationships can both have one or more properties, which are in the form of key-value pairs. Therefore, according to the hierarchical multi-domain NSSA model and the Neo4j data structure, we propose the following modeling rules:

- *Node.* The entity or object interacting with the outside is regarded as a node. For easy of processing and data analysis, all the objects that are related to other nodes are regarded as a single node, and we assign the node the name of the entity or object.
- *Label.* Each node can have one or more labels, so when a node belongs to a certain domain or some domains, we use the name of the domain or these domains as the node label. At the same time, we can also add the category labels that the node belongs to for managing query, and label names are all capital letters.
- *Property.* Property is the information of a node that does not interact with other nodes. We regard the name of the node and the metadata of some necessary entity or object as properties. Property is represented as one or more key-value pairs.
- *Relationship.* The relationship is the connection between different nodes, such as communication, subordination, and connection. There can be one or more different directed *relationships* between the

nodes, and the start node and the end node cannot be null. Although each relationship can contain one or more properties, it can only have one type, and the name of the relationship type can only be capital letters.

3.3 The details of the NSSA data storage method

In the hierarchical multi-domain NSSA model, we store the data in a graph database Neo4j, and the storage method is described as follows:

- *Network topology domain.* At the basic security layer, the network topology domain is constructed on the basis of the corresponding network topology in the organization. The devices in the network topology such as the communication device, the security device, the server, and the user terminal are taken as a node of the graph database. The properties of the device which do not communicate with other nodes are used as the property of the nodes in the graph database, such as device name, model, and other information. Similarly, the properties that communicate with other nodes in the device are treated as a node respectively, such as the open port of the device, IP address, operating system, and so on. The device port consists of the device abbreviation and the port number, for example, port 5 of the switch means SW_5. These abstracted nodes are connected to device nodes by the relationship in the graph database. Besides, the links between the various device nodes are transformed into "relationships" in the graph database, and we add in "topology" as a public label for these nodes to demarcate the area which the limited nodes are located.
- *System service domain.* Combining with the network topology domain, the system service domain takes the operating system, the open port, the running service, and the application as a node in the graph database, which is connected with each other by relationship in the graph database. Furthermore, they are added the label "system service" to divide the region in *where* the nodes are located.
- *Personnel management domain.* The departments and groups in the organization are abstracted to a node by the personnel management domain according to the personnel management information. Besides, each person is treated as a node, *their* name, gender, and age, and other information are regarded as the *properties* of the node. The person node is connected to the department or group node through the relationship in the graph database. At the same time, the computer or server used or managed by the person is regarded as a node, which is connected to personnel nodes through the relationship in the graph database. Finally, these nodes are assigned the label "personal management."
- *Security policy domain.* It formulates a series of rules and security policy on the basis of network security defense equipment in the network environment; regards the security device, domain, and interface as a node in the graph database; and connects each other with their behaviors and the strategy of relationship. Besides, the label of security policy is added to these nodes.
- *Dependency domain.* In the security analysis layer, the dependency domain captures corresponding security dependence on the basis of network topology and the system service information obtained from basic security layer. Dependencies consist of operating systems, network services, applications, services, and programming languages, and they are abstracted as node. Further, the specific content of these categories is abstracted as a node, for example, in the operation system, the Windows systems, Linux systems, Mac OS, Android, ios, etc, are abstracted as a node respectively. Then, we can abstract the CentOS and Ubuntu as a node in the Linux system. They can be abstracted in this way layer-by-layer, and they are connected by relationship. Besides, nodes that contain vulnerabilities are divided into CVE vulnerabilities, SVSS systems, levels, and solutions, then they are set the same label as "dependency."
- *Network security domain.* Network security domain is constructed on the basis of the security events and abnormal information based on the network and the host. It presents the network security events by the way of graph database representation and regards each attack source as a separate node. Besides, the destination source is regarded as the existing node, the attack means are connected as relationship, and the timestamp and the hazard level act as the node attributes.
- *User security domain.* The user security domain utilizes the graph database to describe the user's behaviors which exceed the security threshold. Then, it associates the user with the machine, and it also associates the abnormal event with the timestamp. Besides, it connects personnel nodes which belong to the personnel management domain with the device nodes in the network topology through operation behavior.
- *Threat intelligence domain.* We build the threat intelligence domain on the basis of the STIX threat intelligence standard and abstract the attacker and the victim as a node of the graph database respectively. The attack technique is regarded as the connection of the graph database relationship, and the attack behavior is described as the property of the node.

4 Query analysis of graph database-based NSSA data

4.1 Experiment environment

In order to verify the effectiveness of the proposed NSSA model and data storage method and minimize unnecessary overhead simultaneously, we build a sparrow network environment as shown in Fig. 2. The network topology contains four network segments: campus network, office network, DMZ, and internal application service area. In order to simulate the office environment, we deploy three PCs in the office network. In the DMZ, multiple virtual machines are deployed in two servers to simulate multiple types of network services, such as DNS service, Web service, FTP service, etc., which provide the permission for external network to access this area. The internal application service area deploys several types of application services in two servers, including database service, FTP service, and so on.

To obtain the NSSA data, we set up five data collection points A, B, C, D, and E in the network environment as shown in the Fig. 2 and simulate the real network attack methods. To analyze and display the experiment results, we simplify the data content and quantity. More precisely, only a proportion of the network topology information, personnel management information, vulnerability data, and the attack data are imported into the graph database Neo4j according to the modeling rules. When the graph database-based NSSA data is imported, it can query analysis according to different requirements. We mainly use Cypher query language to query in the Neo4j web management console, whose query language is simple and results can be visualized directly.

4.2 Network topology query analysis

As we have added the topology label to all the devices and interfaces in the network topology, then we only need to execute the following query:

MATCH (n: Topology) RETURN n

We can get the network topology shown in Fig. 3, and we can see the devices and the relationship among

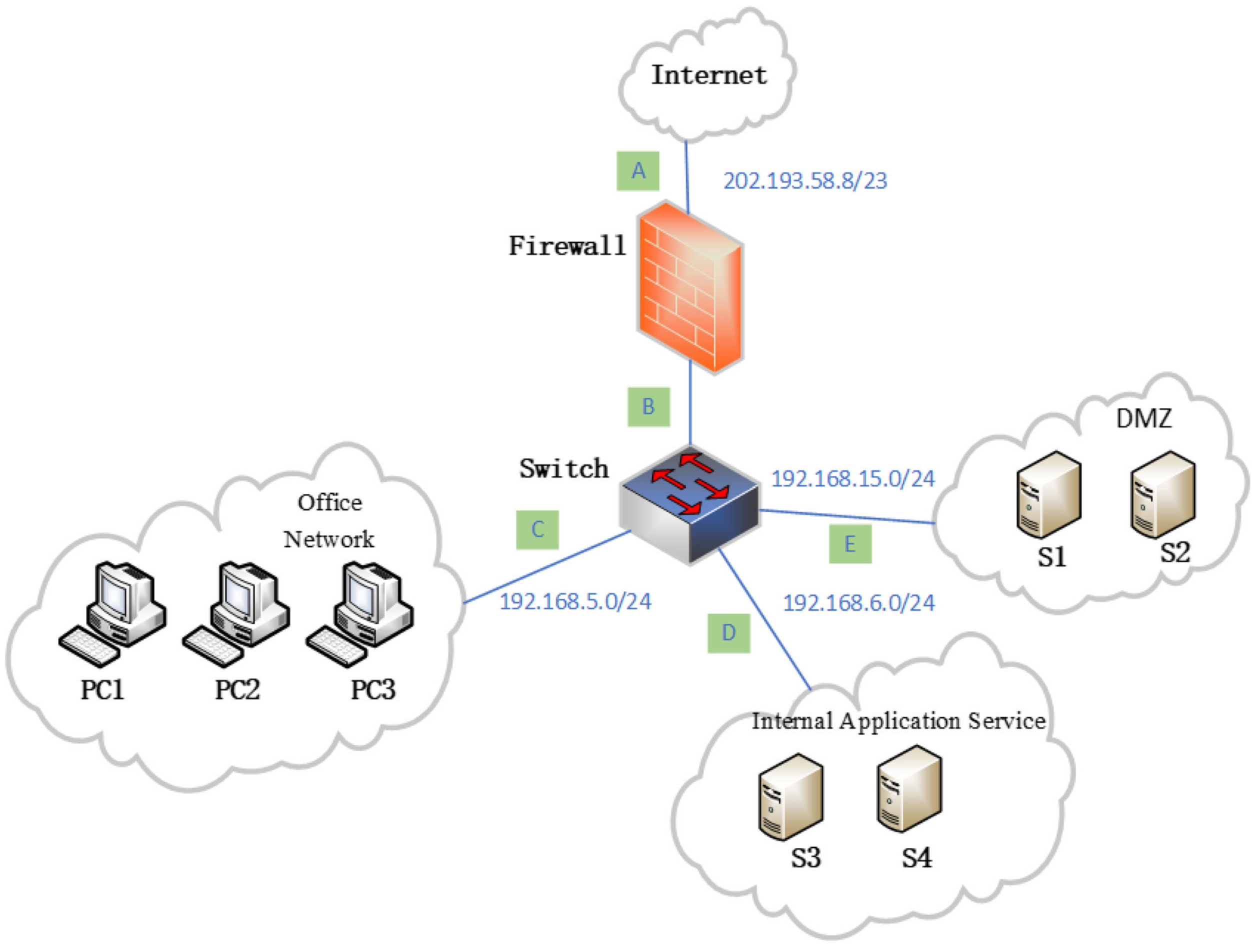


Fig. 2 The network topology of the experiment environment. This figure shows network topology of the experiment environment, which includes five data collection points A, B, C, D, and E in the network environment

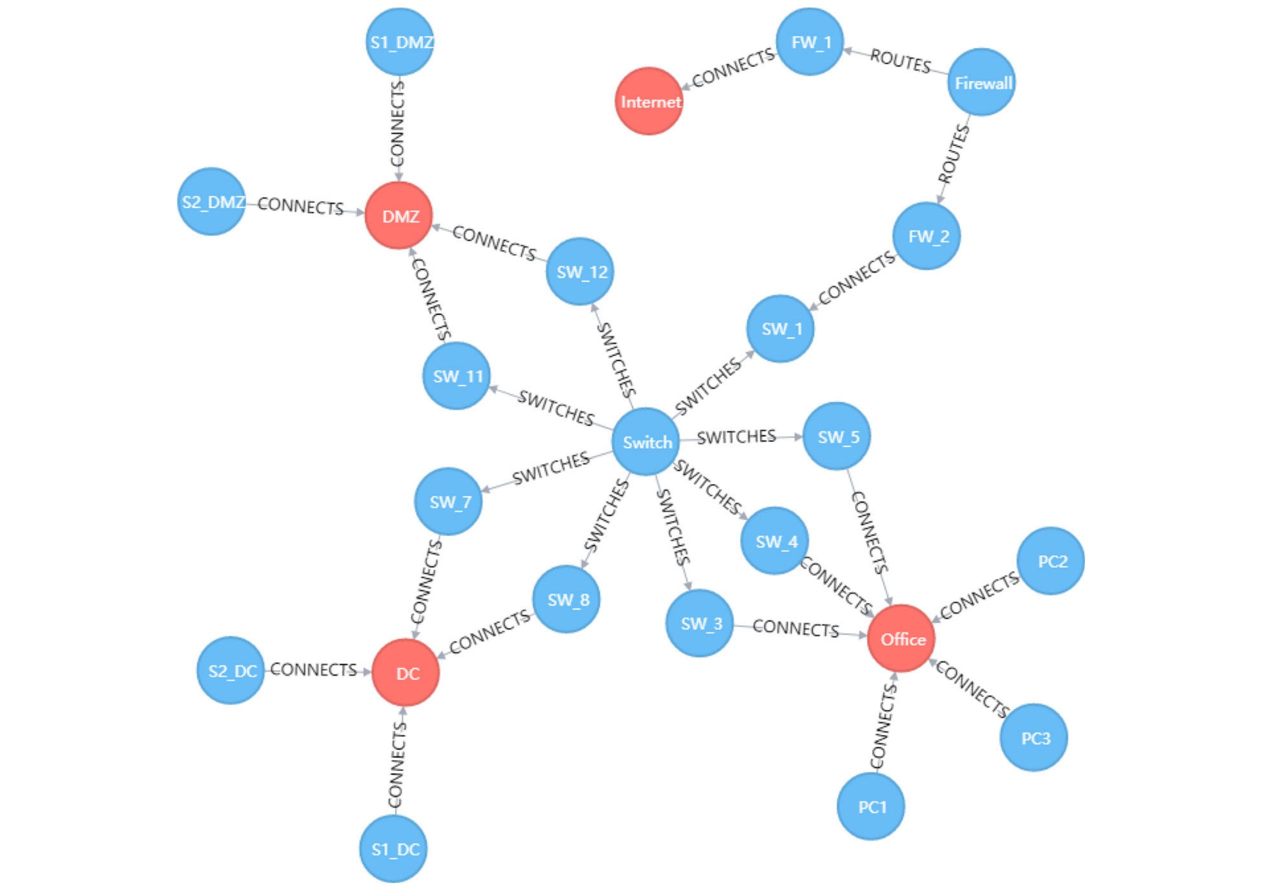


Fig. 3 The network topology query results. This figure shows the network topology after we execute the following query: MATCH (n: Topology) RETURN n. So, we can see the devices and the relationship among them directly

them directly. The network environment in the model is connected by the Internet, passes the firewalls, and reaches routers, consisting of three regions: DMZ, office, and DC, where each area contains several servers or PCs.

4.3 Network vulnerability query analysis

Network vulnerability data refers to the vulnerability information that exists in the implementation of the hardware, the security configuration strategy of the software, and the design of the protocol. Attackers usually use these vulnerabilities to achieve illegal invasion and destruction. It is very important to know the loopholes in the network environment and then to repair and prevent them from happening. To query the vulnerabilities and dependency information in the network, we need to execute the following query statement:

MATCH (n: Dependency) RETURN n

Subsequently, we can get the vulnerability information and web server dependencies in the Ubuntu system whose version is 14.04, as shown in Fig. 4.

4.4 Internal attack query analysis

In the experiment, we can find that most of the attacks are originated from a host whose IP address is "192.168.5.14." Therefore, we can query the equipment and personnel information related to this IP in the graph database. Firstly, we find the device and the area of this IP by using the following query statements:

MATCH (e: Employee) - [] - (n ip: "192.168.5.14") - [] - (z: Zone) RETURN e, n, z

Then, we obtain the result as shown in Fig. 5. From Fig. 5, we can see that the host device with IP "192168.5.14" is PC3, the user of the host is Eric, and the domains of the IP address is Office.

To get more detailed information about the user Eric, we then click the "Table" option, and the result is shown in Fig. 6. From Fig. 6, we can find that the user Eric is a programmer, and he is waiting a post.

Subsequently, we execute the following query command to check Eric's relationship network:

MATCH r = (n name: "Eric") - [* 1..3] - (e: Employee) RETURN r

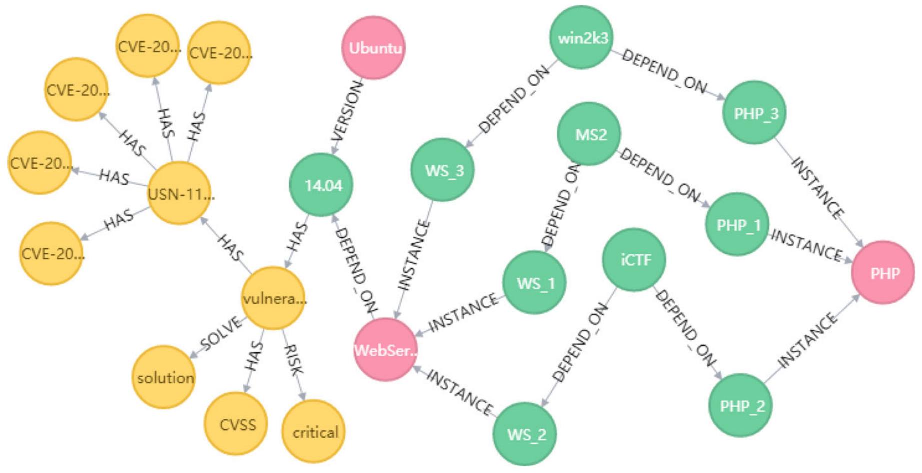


Fig. 4 The results of vulnerability and dependency query. This figure shows the vulnerability information and web server dependencies in the Ubuntu system whose version is 14.04 after we execute the following query: MATCH (n: Dependency) RETURN n

In this query command, we set Eric as the starting node, and the query depth is from one to three. Through this query, we can find out all the people who have connection with Eric. And the query result is shown in Fig. 7.

From the Fig. 7, we can see clearly that Eric was fired by Bob. Given that Eric is waiting for a post, we can speculate that Eric is likely to attack the network deliberately for revenge or stealing confidential data. Besides, Frank and Cindy are friends of Eric; therefore, they also should be closely monitored.

4.5 External attack query analysis

In the experiment, it is found that the outside interacted frequently with the device whose IP is “192.168.5.8” by the network traffic monitoring. We suspect that there are Botnet attacks. Therefore, we perform query command:

MATCH r = (n {name: “192.168.5.8”}) - [:LESTENING_ON] - (p1: Port) - [:CONNECTS_TO] - (p2: Port) - [:LESTENING_ON] - (m: IP) RETURN r LIMIT 100

And the query results are shown in Fig. 8. We can see that other IP addresses are connected to port 25 which is open for Simple Mail Transfer Protocol (SMTP) for sending email. Hackers usually use port 25 to find SMTP servers and conduct attacks.

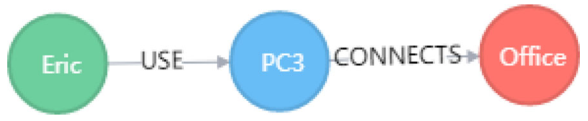


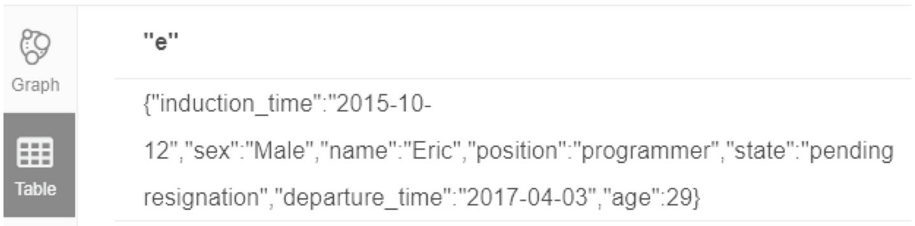
Fig. 5 The results of the IP query. This figure shows the result of the following IP query statements: MATCH (e: Employee) - [] - (n ip: “192.168.5.14”) - [] - (z: Zone) RETURN e, n, z. We can see that the host device with IP “192168.5.14” is PC3, the user of the host is Eric, and the domains of the IP address is office

5 Query analysis of graph database-based NSSA data

In the query efficiency experiment, we use the data of Neo4j official sandbox network and IT management as the experiment data. On the one hand, this data can be processed conveniently because it is well formatted. On the other hand, it can be easy for us to make the query mode and result in multiple databases to be the same since the data volume is moderate. Therefore, we use this data to conduct the query analysis comparison experiments. The database contains a total of 68,122 nodes, 98,610 labels, 150,732 relationships, and 121,098 properties. And the datacenter contains four zones, and each zone contains its independent network and ten racks. Every rack has different types of servers, which are connected to the switches by interfaces.

After importing the network and IT management data from the Neo4j sandbox of the GitHub into our local Neo4j graph database, the data in the local Neo4j graph database is exported to a CSV file by using neo4j-shell-tools. Each piece of data is exported in the form of “node-relationship-node”. Each node contains node id, label, and various attribute fields. The relationships are related to the relational type, which connect the two conjoint nodes. The CSV file is imported into the MySQL database by the LOAD DATA INFILE function of MySQL. Besides, it is imported into the MongoDB database by the mongoimport function of the MongoDB.

In the experiment, we use a DELL PowerEdge T130 server, which is configured with 4 core Intel(R) Xeon(R) CPU E3 processors, 32Gb memory, 4TB hard disk, and the



"e"

```
{"induction_time":"2015-10-12","sex":"Male","name":"Eric","position":"programmer","state":"pending resignation","departure_time":"2017-04-03","age":29}
```

Fig. 6 The detailed information of Eric. To get more detailed information about the user Eric, we then click the "Table" option, and the result is shown in Fig. 6. From Fig. 6, we can find that the user Eric is a programmer, and he is waiting a post

operation system is Ubuntu14.04. We compare the query efficiency of single query, count query, and traversal query on graph database Neo4j, relational database MySQL, and non-relational database MongoDB, respectively. During the experiment, the databases are restarted before each query. To make the results more accurate, we repeat each query five times, and then, we compute the average value as the final result.

5.1 Experiment results

Figure 9 shows the running time of single query on Neo4j, MySQL, and MongoDB respectively, with query depth varying from 1 to 5. From Fig. 9, we can see that for MySQL with query depth 1 to 3, the time spent basically remained the same. But when the query depth increases to 4, the time spent significantly increased, which is an order of magnitude higher than that of MongoDB and Neo4j. MongoDB takes more time than Noe4j when query depth increases from 2 to 5. The overall Neo4j query time is the smallest, and the query time remains basically the same when query depth increases from 3 to 5.

Figure 10 shows the results of traverse query on Neo4j, MySQL, and MongoDB, respectively. Here, the traverse query only returns the number of the result. As can be seen from Fig. 10, when the query depth is 2, the query time of MySQL is obviously higher than that of neo4j and MongoDB. The Neo4j is superior to MongoDB. More precisely, when the query depth is less than three, performance gain of Neo4j is not very significant compared with MongoDB. However, once the query depth exceeds three, Neo4j outperforms MongoDB tremendously when processing traverse query.

We also investigate performance of Neo4j with respect to single query, count query, and traversal query, and the experiment results are given in Fig. 11. The results show that single query, count query, and traversal query have similar running time when the query depth increases from 1 to 4. When query depth is 5 and the number of returned data items is fixed to 200, the running time of traversal query is obviously higher than that of single query and count query, which indicates that the Neo4j affects the performance of the traversal query with multi-layers and multi-result.

5.2 Complexity analysis

Our NSSA model is based on Neo4j graph database, which is a complex yet efficient database management system for graph data. Neo4j uses Lucene index for its internal data indexing, and the time complexity of Neo4j depends

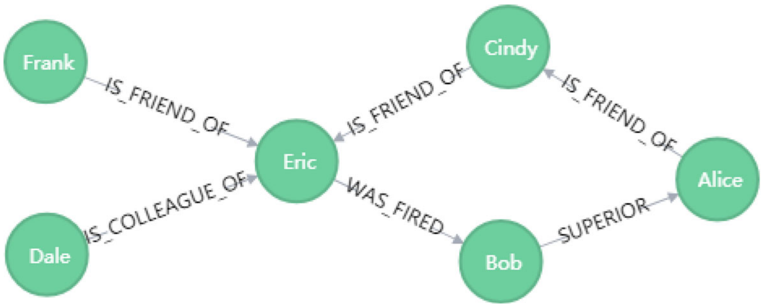


Fig. 7 Eric personnel relations query results. This figure shows the result of the following query statements: MATCH r = (n name: "Eric") - [* 1..3] - (e: Employee) RETURN r. In this query command, we set Eric as the starting node, and the query depth is from one to three. Through this query, we can find out all the people who have connection with Eric

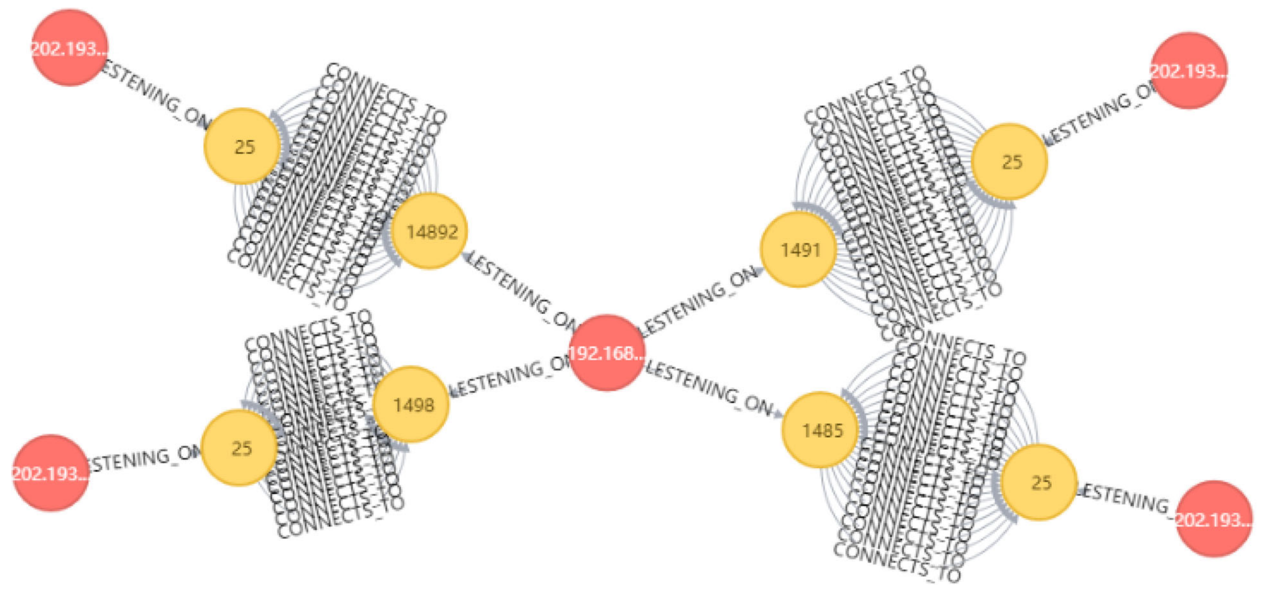


Fig. 8 The external attack query results. The query results after we perform query command: MATCH r = (n name: "192.168.5.8") - [:LESTENING ON] - (p1: Port) - [:CONNECTS TO] - (p2: Port) - [:LESTENING ON] - (m: IP) RETURN r LIMIT 100. We can see that other IP addresses are connected to port 25 which is open for Simple Mail Transfer Protocol (SMTP) for sending email

on the type of query and Lucene index. Generally, for looking up an entity in the Neo4j database, let us say the "match" query, the time complexity is $O(log(n))$, where n is the number of total entities in the database. For other type of queries, for example the shortest path query, since Neo4j employs standard Dijkstra's algorithm and A^* search algorithm, the time complexity is $O\left(V^2\right)$, where V is the number of nodes in the graph database. Hence, our NSSA model has the same time complexity as that of Neo4j.

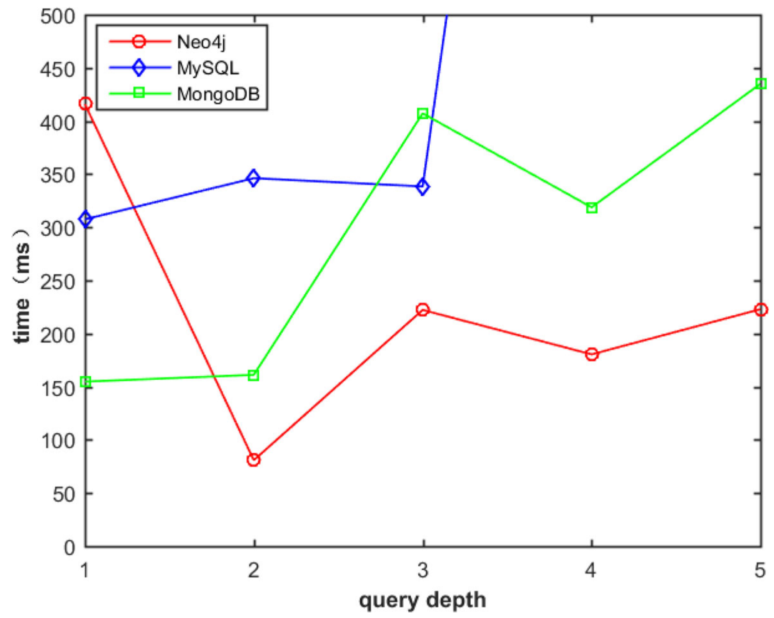


Fig. 9 The results of single query. Figure 9 shows the running time of single query on Neo4j, MySQL, and MongoDB respectively, with query depth varying from 1 to 5. From Fig. 9, we can see that for MySQL with query depth 1 to 3, the time spent basically remained the same. But when the query depth increases to 4, the time spent significantly increased, which is an order of magnitude higher than that of MongoDB and Neo4j. MongoDB takes more time than Noe4j when query depth increases from 2 to 5. The overall Neo4j query time is the smallest, and the query time remains basically the same when query depth increases from 3 to 5

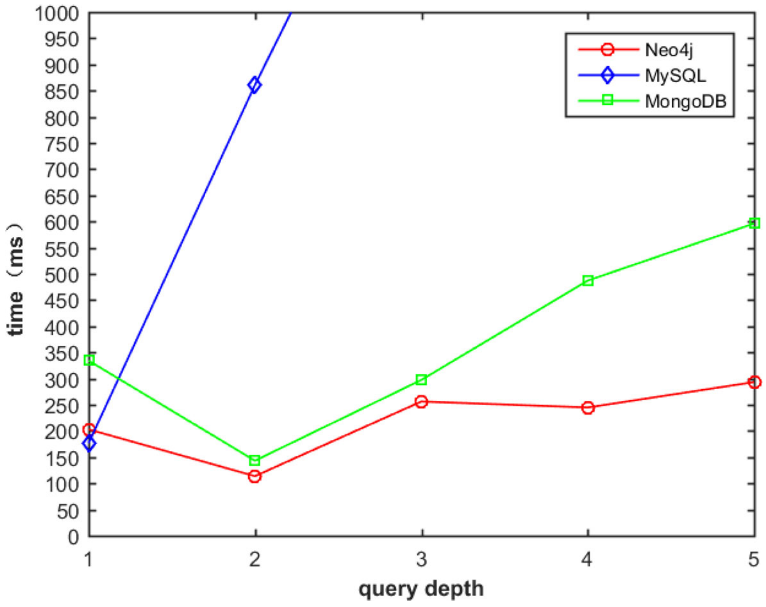


Fig. 10 The results of traversal query. Figure 10 shows the results of traverse query on Neo4j, MySQL, and MongoDB, respectively. Here, the traverse query only returns the number of the result. As can be seen from Fig. 10, when the query depth is 2, the query time of MySQL is obviously higher than that of neo4j and MongoDB. The Neo4j is superior to MongoDB

6 Conclusion

In this paper, we propose a graph database-based hierarchical multi-domain NSSA data storage method. In our method, we use graph database Neo4j to store NSSA data, which cannot only query the basic network information easily, but also can query the internal and external attacks conveniently. Finally, we evaluate the proposed method through extensive experiments. Compared with MySQL and MongoDB, the graph database Neo4j is more efficient when querying network vulnerability.

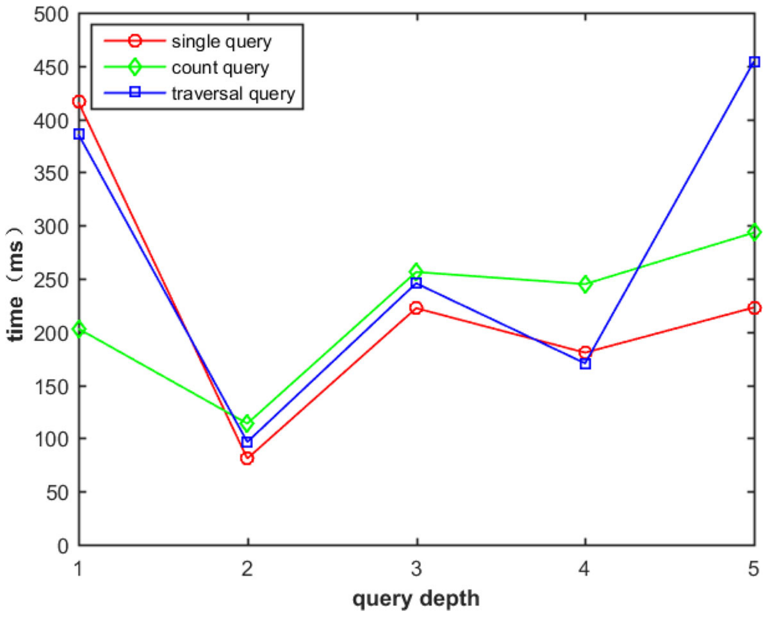


Fig. 11 The results of the Neo4j database query. The investigate performance of Neo4j with respect to single query, count query, traversal query, and the experiment results. The results show that single query, count query, and traversal query have similar running time when the query depth increases from 1 to 4. When query depth is 5 and the number of returned data items is fixed to 200, the running time of traversal query is obviously higher than that of single query and count query, which indicates that the Neo4j affects the performance of the traversal query with multi-layers and multi-result

Abbreviations
ACID: Atomicity, Consistency, Isolation and durability; CCNx: Content Centric Networking; CIMGDB: Common Information Mode Oriented Graph Database; CRUD: Create, read, update, and delete; CVE: Common vulnerabilities and exposures; D-S: Dempster-Shafer envidence theory; DMZ: Demilitarized cone; NSSA: Network Security Situation Awareness; SA-ISRM: Situation Aware ISRM; STIX: Structured Threat Information eXpression ; SVSS: Sacramento Valley Soaring Society

Funding
This work was supported by the National Natural Science Foundation of China (No. 61363006), the Open Projects of State Key Laboratory of Integrated Service Networks (ISN) of Xidian University (No. ISN19-13), the National Natural Science Foundation of Guangxi (No. 2016GXNSFAA380098), and the Science and Technology Program of Guangxi (No. AB17195045).

Authors' contributions
XT, YL, and FZ contributed to the conception and algorithm design of the study. XT, YL, and YW contributed to the design of experiment scheme. XT, YL, and CY contributed to the analysis of experimental data and approved the final manuscript. All authors read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Author details
[1]Guangxi Colleges and Universities Key Laboratory of Cloud Computing and Complex Systems, Guilin University of Electronic Technology, Guilin, China. [2]State Key Laboratory of Integrated Service Networks (ISN), Xidian University, Xi'an, China. [3]Guangxi Cooperative Innovation Center of cloud computing and Big Data, Guilin University of Electronic Technology, Guilin, China.

Received: 20 August 2018 Accepted: 22 November 2018
Published online: 22 December 2018

References
1. Z. Cai, X. Zheng, in *IEEE Transactions on Network Science and Engineering*. A private and efficient mechanism for data uploading in smart cyber-physical systems (IEEE, 2018)
2. Z. Cai, Z. He, X. Guan, Li Y., in *IEEE Transactions onDependable and Secure Computing*. Collective data-sanitization for preventing sensitive information inference attacks in social networks (IEEE, 2018), pp. 577–590
3. M.R. Endsley, Toward a theory of situation awareness in dynamic systems. Hum. Factors. **37**, 32–64 (1995)
4. Y. Zhang, S. Huang, S. Guo, J. Zhu, Multi-sensor data fusion for cyber security situation awareness. Procedia Environ Sci. **10**, 1029–1034 (2011)
5. J. Webb, A. Ahmad, S. B. Maynard, G. Shanks, A situation awareness model for information security risk management. Comput. Secur. **44**, 1–15 (2014)
6. Z. Chen, G. Xu, V. Mahalingam, L. Ge, J. Nguyen, W. Yu, C. Lu, A cloud computing based network monitoring and threat detection system for critical infrastructures. Big Data Res. **3**, 10–23 (2016)
7. B.W. Masduki, K. Ramli, M. Salman, in *International Conference on Smart Cities, Automation & Intelligent Computing Systems (ICON-SONICS)*. Leverage intrusion detection system framework for cyber situational awareness system (IEEE, Yogyakarta, 2017), pp. 64–69
8. F. Chen, Y. Zhang, S. U. Jin-Shu, W. B. Han, Two formal analyses of attack graphs. J. Softw. **21**, 49–63 (2010)
9. C. Constantinov, C.M. Poteras, M.L. Mocanu, in *17th International Carpathian Control Conference (ICCC)*. Performing real-time social recommendations on a highly-available graph database cluster (IEEE, Tatranska Lomnica, 2016), pp. 116–121
10. H. Gu, Z. Han, J. Xu, Framework of parallel layout algorithm based on large-scale social networks. Comput. Appl. Softw. **34**, 73–78 (2017)
11. F. Zarrinkalam, M. Kahani, Paydar S., in *International Symposium on Telecommunications*. Using graph database for file recommendation in pad social network (IEEE, Tehran, 2015), pp. 470–475
12. A.A. Patel, J.N. Dharwa, in *International Conference on ICT in Business Industry & Government (ICTBIG)*. An integrated hybrid recommendation model using graph database (IEEE, Indore, 2016), pp. 1–5
13. G. Ravikumar, S.A. Khaparde, A common information model oriented graph database framework for power systems. IEEE Trans. Power Syst. **32**, 2560–2569 (2017)
14. B. Kan, W. Zhu, G. Liu, X. Chen, D. Shi, W. Yu, Topology modeling and analysis of a power grid network using a graph database. Int. J. Comput. Intell. Syst. **10**, 1355–1363 (2017)
15. M.S. Barik, Mazumdar C., in *International Conference on Security in Computer Networks and Distributed Systems*. A graph data model for attack graph generation and analysis (Springer, Berlin, 2014), pp. 239–250
16. S. Noel, E. Harley, K.H. Tam, G. Gyor, *Big-data architecture for cyber attack graphs representing security relationships in nosql graph databases*. (Citeseer, 2015)
17. M.S. Barik, C. Mazumdar, A. Gupta, in *International Conference on Information Systems Security*. ICISS 2016. Lecture Notes in Computer Science, vol 10063., ed. by I. Ray, M. Gaur, M. Conti, D. Sanghi, and V. Kamakoti. Network vulnerability analysis using a constrained graph data model (Springer, Cham, 2016), pp. 263–282
18. A. K. TK, J.P. Thomas, S. Parepally, An efficient and secure information retrieval framework for content centric networks. J. Parallel Distrib. Comput. **104**, 223–233 (2017)
19. Z. Zhang, G. Pang, J. Hu, L. Su, *Neo4j authoritative guide*. (Tsinghua University Press, 2017)
20. L. Zheng, L. Zhou, X. Zhao, L. Liao, W. Liu, in *International Conference on Information Science and Control Engineering*. The spatio-temporal data modeling and application based on graph database (IEEE, Changsha, 2017), pp. 741–746
21. I. Robinson, J. Webber, E. Eifrem, *Graph databases*. (O'Reilly Media, Inc., 2013)
22. J. Webber, in *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*. A programmatic introduction to Neo4j (ACM, 2012), pp. 217–218
23. H. Huang, Z. Dong, in *Communications and Networks, International Conference on Consumer Electronics*. Research on architecture and query performance based on distributed graph database neo4j (IEEE, Xianning, 2013), pp. 533–536
24. A. Sharma, S. Batra, in *Fifth International Conference on Advances in Computing and Communications*. Enhancing the accuracy of movie recommendation system based on probabilistic data structure and graph database (IEEE, Kochi, 2016), pp. 41–45
25. C.I. Johnpaul, T. Mathew, in *International Conference on Advanced Computing and Communication Systems*. A Cypher query based NoSQL data mining on protein datasets using Neo4j graph database (IEEE, 2017), pp. 1–6
26. A. Vukotic, N. Watt, T. Abedrabbo, D. Fox, J. Partner, *Neo4j in action*. (Manning Publications Co., 2014)